EPFL CS 358

Making Intelligent Things

Course Handbook

September 10, 2025

Christoph Koch (christoph.koch@epfl.ch)

with

Freya Behrens, Lars Klein, Alexander Müller,
Juliette Parchet, Giovanni Ranieri, Dylan Vairoli,
and Leo Wolff

Contents

Part I

The Course

Chapter 1

Introduction

1.1 The Makers Revolution

Recent years have witnessed a development that some call the Makers Revolution or the Third Industrial Revolution. We have seen a democratization of computer-aided manufacturing and an impressive increase in the number of people who turn tinkering with electronics, robotics, 3D printing, etc., into a hobby. Along with this we see the greatly increased availability of cheap, mass-produced electronic modules read-made for makers who are not electric engineers[1], and the coming of age of 3D printing technology. Projects that could only be carried out by the largest of organizations and the richest of nation-states a few decades ago[2], and even things that recently still belonged to (science) fiction, have now moved into the realm of what hobbyists can make on very moderate budgets. Makerspaces – shared spaces that facilitate the development of prototypes for research, as business ventures, or in the context of play and hobby – have sprung up in many places[3], including at EPFL. DLLEL, also called the SPOT, is one of a series of makerspaces EPFL has created or is in the process of developing. DLLEL is our main stomping ground for this course.

[1] This includes open-source hardware designed for learners, such as the Arduino line of micro-controllers and Adafruit's line of associated modules.

[2] Think of the capabilities of the computers on the earliest spacecraft.

[3] There is even a mega-city-sized makerspace called Shenzhen – see https://www.youtube.com/watch?v=XmHaCZIncrg.

1.2 A Course for Software-Savvy Makers

The course CS-358 Making Intelligent Things is IC's main educational contribution to the makers revolution.[4] It allows IC students to get their hands dirty and make something physical. Use power tools that make a lot of noise. Play with 3D printers, laser-cutters, and CNC machines. Write a program and upload it to a device that is smaller than your fingertip[5] and which still can sense its environment and make stuff move. Do something that used to belong to science fiction and only became feasible, easy, cheap, or safe to make recently, by that makers revolution, the (hardware) open-source movement, and the mass manufacturing of components for which there was no market until recently. Make something that you can show your friends and family, who never quite understood what computer science was about and who can now finally see a tangible thing you made and that demonstrates some of the things you have learned at EPFL. In the past, you may have tried to explain to them what NP-completeness is and have received a blank stare. Now, you can demo your thing, blow them away with its brilliance, and be acknowledged as a genius.

You can make an awesome prototype that makes the world a better place – an assistive technology, a way to improve sustainability, or to fight epidemics. A useful tool, or just a really cool toy. You may turn this into the nucleus of a business startup. Within reasonable and safe limits, this course is meant to enable you to create and realize your own vision – you may propose and realize your own project.

Making *Intelligent* Things is not necessarily about Artificial Intelligence[6], though AI is welcome. It is about connecting software to the physical world via sensors and actuators. It is about making things that are reactive to the physical world. At least to the uninformed bystander, these things will appear (somewhat) intelligent. Do not force too much advanced computer vision and machine learning into your projects. There are specialized machine learning and computer vision courses at EPFL, and they are better places for going state-of-the-art (and beyond) in these respects. In this course, we want to work with devices that have limited memory and computational capabilities. It is about creating smart solutions, not an

[4] See https://www.youtube.com/watch?v=uSIPdKgHk7A for a presentation video on the course.

[5] We have ATTINY13A SOIC8 micro-controllers ($6.2 * 5 * 1.75$ mm) in stock.

[6] For more on this see Chapter 15.

exercise in buying lots of GPUs.

If you trust the internet, of all the myriad skills required from a high-tech maker, software is the most challenging one. But this is your home turf. This makes you, an IC student, better suited than any other group of students on the EPFL campus to achieve something impressive in a course like this. This is also the feedback we are getting from "outsiders" not involved in the course. The projects produced in past iterations of the course have been particularly fun and interesting to them because the software side that you excel at creates so much functionality and utility. Let us continue in this tradition, and even push it to new heights.[7]

1.3 The Place of the Course in the Curriculum

This course is meant to at once give you a playground to create embedded systems and combine software with physical stuff *and* teach you to work successfully in a team. It will expose you to challenges that arise in robotics and process automation, though it won't be able to replace specialized courses in this space.

With the (computer science) curriculum reform of 2024, CS-358 has become an 8 credit course that is close to mandatory: CS-358 is in a catalog of two large team project courses (the other one is CS-311 Software Enterprise), of which you need to pick one. The course can also take the place of a semester project, a decision that was made since student enrollment has been growing steadily in the past few years and the supply of teachers and project supervisors has not kept pace, creating a shortage of advisors. The replacement of semester projects by the course will eliminate the problem of finding a semester project advisor in the future, and will make it easier to find an advisor for your masters thesis (should you stay on for a masters) given that advisors will have more cycles free for that.

These two courses prepare you to work in teams on larger projects. We will, in a practical setting, do project management and agile development. This exposes you to challenges you may not have encountered before. It will not be sufficient to be individually capable, but you will have to function

[7] As Dante Alighieri says, "Now shall your true nobility be seen!" He is speaking to allegorical genius and inspiration here, even though the canto is called the Descent [into hell]. Either way the quote is fitting for the course]:-)-

well in a team.

Given their role in the curriculum and the fact that you probably only take one of the two courses, Software Enterprise and Making Intelligent Things share many teaching goals. We expect that most of you will end up in industry doing software, and we must make sure that you are not at a disadvantage on the job market compared to students taking the Software Enterprise project course. We will use an adapted form of Scrum, a software development process that is also used in Software Enterprise.[8] We share best practices with Software Enterprise, and we will apply the same formula for time cost for the course (240 hours in total, or about 17 hours[9] of work per week, for 14 weeks), and follow a very similar approach to grading. So the two courses are in many ways quite similar, but in CS-358 we will create embedded systems rather than pure software systems.

This course is likely to make you feel like spending your time on a hobby, more so than any other course in your study plan. The physical, manual activity can be great at counter-balancing the stress you are experiencing elsewhere and the heavy theoretical work of some of your other courses. You do need to take the course serious, though, your team depends on it.

1.4 Deciding Whether to Take the Course

This is a project-based course without any classical lectures[10] or exams. You must be willing to explore, solve problems, and identify what you will need to learn to be successful. We will help you do that. This manual is an important part of this. To allow us to communicate some of the information you will need, you have to be willing to read parts of this manual on your own, and search for information in here proactively. Alas, we have no other way of getting this information to you.

As pointed out above, EPFL expects that you to spend about 17 hours per week on this course. This is a substantial time commitment. Because

[8] Adaptations are necessary since SCRUM essentially assumes that a new prototype can be built anytime at the zero cost (by simply compiling), which is true for software but false when developing hardware.

[9] This is a lot, but we work very hard to ensure that this is also an upper bound on the time you need to spend on the course.

[10] The manual, plus some tutorials we are organizing, are meant to fill this void while remaining in true to IC's conception of a project course.

of the team project nature of the course, you will not be able to plan freely and focus your work time on certain weeks; you will be expected to be able to contribute consistently across the semester. This requires discipline and cooperation from you.

In the course, you will do an individual project and a team project. The individual project has been designed by the course staff to gently introduce you to the various challenges of making intelligent things and to bring everyone to a level to achieve success in a team with minimal frustration. For the team project, you will be asked to come up with a team project proposal of your own, which the course staff have to approve (see Chapter 7).

Since you will create a single physical artifact that has to remain accessible to everyone in the team, and since you will need to use tools that you do not have at home, you will have to spend a considerable part of your time being physically present in the DLLEL building – the makerspace. Some of the work can only be done in the makerspace and you will need to physically meet your team members and the course staff at appointed times. If you are very inflexible about time, or plan to come to the campus only infrequently, you cannot take this course.

You will not be able to take the thing created in the team project home. You will have to work in DLLEL and keep the physical components of the thing you are making at EPFL and physically accessible to your team members at all times.[11] We do understand that you might rather work at home, but, with the exception of some CAD design and offline programming work, this will not be possible.

This course may not be right for you. Please decide now, and do not sign up for the course if you think you will not enjoy it.

After all these cautionary notes, let me point out that many students in the previous iterations of the course reported that they enjoyed it a lot. I think that you will have a lot of fun in this course too. See some glimpses of the projects from the Spring 2023 iteration of the course in this video:
https://www.youtube.com/watch?v=uSIPdKgHk7A .

Also keep in mind that the course, and the training units you will complete as you take it, grants you a "driving license" to keep using the the facilities and resources of the makerspace for your private projects after the end of the course, for as long as you remain an EPFL student. It's a cool privilege.

[11] See also Chapter 3.

Chapter 2

Course Organisation

2.1 Course Structure

The course consists of three kinds of modules:

1. Tasks you need to do to qualify for other things, but which are ungraded. These are mainly in place for you to be able to take the course safely. This includes some online safety training modules (on moodle), a soldering tutorial, a 3D printing tutorial, a laser cutting tutorial, an introduction to the machine shop, as well as three CS-358-specific tutorials (one per week in the first three weeks in BC03) that will allow you to do project work safely and successfully.

   These things are mandatory but ungraded. Some are required by EPFL for you to be allowed to use the makerspace and are not specific to our course.[1] See Chapter 4.

2. Activities you need to perform individually and which are graded. This includes

   - a small individual project intended to teach you key skills that you will need in the larger team project (see Chapter 6).

3. a team project (covered in Chapters 8 and 9). This is the central part of the course; all other modules are in place to prepare us for it.

We also have an electronics quiz/exam in the third week of the semester.

[1] Also note that doing this training entitles you to use the makerspace for your own projects, independently from the course, within reason, in the future.

2.2 Grading

The grade split is as follows:

- The individual project accounts for 20% of the course grade. The individual project grade will be determined in a meeting and demo of your thing to the course staff (see Chapter 6).
- The electronics quiz (see Section 2.5) accounts for 5% of the grade.
- The team project proposal accounts for 5% of the grade (see Chapter 7).
- The remaining 70% of the grade come from your performance in the team project (see Chapters 8 and 9). Course participation throughout the semester, in the weekly team project/SCRUM meetings with the course staff, forms part of the grade.

  There is also a final presentation event, but overall success of the team project isn't just judged at the end. Despite the team work, we will grade you individually, and for tasks you work on during the agile development process. If you do too little, you will fail the course even if you are in a great team that produces a spectacular thing.

Details on grading these parts are given in the respective chapters below. Grade cut-offs:

6.0: >98; 5.75: >90; 5.5: >85; 5.25: >80;
5.0: >75; 4.75: >70; 4.5: >65; 4.25: >60; 4.0: >50

2.3 Contact Hours and Contacting Us

In the Spring of 2025, the teaching staff consists of the following people:

- christoph.koch@epfl.ch (professor)
- zhekai.jiang@epfl.ch (PhD TA)
- jiachen.lu@epfl.ch (PhD TA)
- adele.monjauze@epfl.ch (AE)

- adrien.aumont@epfl.ch (AE)
- anna.barberis@epfl.ch (AE)
- malen.raychev@epfl.ch (AE)
- mehdi.alaoui@epfl.ch (AE)
- pinar.oray@epfl.ch (AE)
- rudolf.yazbeck@epfl.ch (AE)
- tuan.nguyen@epfl.ch (AE)

We are supported by fabienne.ubezio@epfl.ch (admin).

Your IS-ACADEMIA timetable shows two slots, which have been picked such that there should not be any collisions with other mandatory courses, blocked for the course:

- Tuesdays from 5 to 7pm in DLLEL 1-50, the seminar room of the makerspace. The *plenary*: Our central, regular, key, main meeting. Unless it's a holiday, we will all meet in this slot every week. We will answer any questions you might have. All the course staff will be present. The **kickoff meeting** for the course and the first date you need to keep in mind taking this course is **Tuesday 5-7pm first week of the semester**. Early on, it will be a place to meet other students and form teams for the team project phase and to discuss project ideas. Throughout the course, we will discuss important organizational and other discussion points. During the team project phase, some of the SCRUM meetings can take place in this slot (but not all since a senior teaching staff member will need to be present, allowing only for limited parallelism). Here are the rough topics of the Tuesday meetings by week:

  1. Kickoff meeting – course organisation.
  2. The individual project.
  3. Presenting and deciding on the team project topics (plus tentative team formation).
  4. How the team project phase will play out. First informal team meeting with TA.

5. Team meetings with TA, discuss updated project proposal/documents.
6. Individual project demos.

7ff. Team meetings, work in the projects with prof and staff present to help ...

- The tutorial/Scrum slot: Wednesdays (in the fall) or Thursdays (in the spring) from 1 to 4pm. In the first three weeks, we will have tutorials in these slots in BC03. In the first week, we will also distribute the bags with electronic and mechanical components that you will need for the individual project in this slot. If you miss the tutorial, please get your components bag from Fabienne Ubezio (BC246, Tuesdays and Thursdays).

  After that, the tutorial/Scrum slot is for the Scrum meetings. The Scrum meetings will be in DLLEL, so we will not use BC03 from Week 3 on.

Even when you are not having your Scrum meeting, we encourage you to be in the makerspace to work on your project during these two slots during the team project phase; course staff will be present.[2] At the end of the plenary, Christoph stays until all questions and requests are resolved, no matter how long that takes.

During the tutorial and individual project phase (the first five weeks of the semester) we have additional office/contact hours of the AEs in the makerspace. See Figure 2.1 for the times of office hours. The AEs are in the DLLEL main workspace (DLLEL 0-28) during the office hours and you don't need an appointment to meet them; just go there.

Office hours end in W6 with the individual project demo event. After that, please contact the teaching assistants assigned to your team project when you need help.

The first source of answers to any questions is this manual, not because we do not want to talk to you but because we have tried to think of addressing any organizational question that may arise. We encourage feedback in case you find something missing.

We use moodle to share information and to allow you to submit documents. There is also a forum there in which we ask you to put your questions that may be of interest to others.

[2] But keep in mind that just allotting these two slots of your time for this course will not be sufficient, by far – see Figure 2.2.

From	To	Mon	Tue	Wed	Thu	Fr
9	10					
10	11	Tuan			Adrien	
11	12	Tuan		Tutorial/Scrum fall	Adrien	
12	13			Tutorial/Scrum fall		
13	14			Tutorial/Scrum fall		Rudolf
14	15		Adele	Malen		Rudolf
15	16		Adele	Malen	Pinar	
16	17	Mehdi	[Staff Meeting]	Anna	Pinar	
17	18	Mehdi	Plenary	Anna		
18	19		Plenary			

Figure 2.1: Office hours (Fall 2025), Wed of W1 to Tue of W6.

For other questions we use email.

2.4 Materials and Ownership

Materials for laser cutting and 3D printing as well as small items such as cables and screws are available for free to you in DLLEL.[3]

We buy and give out all the other electronic and mechanical components that you will need for both the individual project and the team project. We do this at specific events (see below) – the components needed for the individual project will be given to you in a prepared bag in Week 1. We may ask you to provide an item that you are very likely to own – such as a USB charging cable or a pen – for your own use.

Should a part be or get damaged or go missing, talk to an AE, preferably during office hours. While you remain in good standing[4], we will try to help you out replacing damaged part.

There is also a place at DLLEL (called the "circular box" – though it's not just a box but a well-organized storage unit with multiple drawers) where people donate and pick up used parts. Sometimes the fastest way to get something you lack is to look there, and you may take things from there for your private projects. Consider donating the re-usable components of

[3]This is true for course projects: the costs get charged to the EPFL DATA Lab at the end of the semester. You may use the makerspace for private projects; in that case, you are charged for consumables but the prices are very low.

[4]You will lose this status through negligence. You need to do the best you can to avoid damaging things, and remain informed on how to do this by reading the relevant chapters of this manual.

Week	1	2	3	4	5	6	7 to 13	14	Total hours
Tue meeting (excl prj work/demos)	2	2	2						6
Demos						2		2	4
Individual project time		4	11	10	10				35
BC03 tutorials	3	3	3						9
DLLEL tutorials & soldering	2	2							4
Handbook reading, videos	5	3	2						10
Team project idea	6	4							10
Team meeting				1	1	1	1	1	11
Team project time				7	7	15	15	17	151
Total hours	18	18	18	18	18	18	16	20	240

Figure 2.2: Time estimates per week and activity.

your individual project there after grading in case you are not interested in keeping the thing you made.

If you take the course for credit (i.e., you do not drop out in the first two weeks), the thing produced in the individual project is yours to keep. The team project thing belongs to EPFL, but see also Section 2.7.

2.5 Time Plan and Deadlines

Figure 2.2 shows a recommended time plan for the semester. Weeks 7 to 13 all have the same structure: 16 hours of team project work with weekly Scrum sprints and an hour-long weekly meeting of the team with course staff. As you can see, the individual project and the team project overlap, and you should finish the tutorials and training as early as possible.

Key Activities and deadlines:

- Week 1, Tuesday: Kickoff meeting.
- Week 1 (during the Wed/Thu tutorial): We hand out the component bags for the individual project.
- Week 1-2: If you want to propose your own project idea, make sure to talk to the professor and get it approved by the tutorial slot in Week 2.
- Week 1-5: Work on the individual project.
- End of Week 2 (recommended): Finish all required training units and tutorials (see Chapter 4).

- Tuesday Week 3: Approved team project proposals are published on moodle. Form a team and pick a proposal.
- Tutorial slot of Week 3 (Wed in Fall, Thu in Spring), 1-1:20pm Written safety and electronics quiz in BC02 and BC03. Closed-book, twenty minutes duration, covering Chapters 3, 11, 19, 21, 26, 38, and 39. In addition, you need to be familiar with the functions and pins of the Arduino Uno (see Chapters 27 and 28; Figure 28.2). The exam will be multiple-choice except for one circuit-building example. We will give you more information in the first two tutorials.
- (Preferably by the) end of Week 3. Finalize the team composition and fix the choice of project idea. In case we are unable to finalize your team in the W3 Tuesday meeting, send email to Christoph.Koch@epfl.ch naming the project idea and identifying the team members. Include github handles for all team members. A moodle group will be created and TAs will be assigned to you. The teaching staff create a Github repository for you. You create a Fusion360 project (they are cloud-based) and invite your team members and TAs in.
- Weeks 4, 5 and 6: Work on the revised team project proposal documents as a team. Also meet your TA as a team to discuss your proposal. Solicit feedback from the teaching staff and work it into your proposal.
- Tuesday 5-7pm in Week 5. Demo/grading session during the plenary slot: grading of mechanical (belt tensioning, etc.) and electronics assembly (2 out of 20 points): the moveable pen-holder is not required if the fixed penholder is correctly mounted. You do not need to power it up and demo running code.
- End of Week 5. Finalize individual projects; submit documents to moodle.
- Tuesday 5-7pm in Week 6. Demo/grading session during the plenary slot. You may request a demo in an earlier plenary if you are done early (talk to a TA about it).
- End of Week 6 (Sunday 11:59pm, submit on moodle): Team project proposal/documents with final bill or materials, risk assessment, and work breakdown document are due on moodle. See Section 8.5.

- We try to provide you with the requested and agreed upon electronic and mechanical components as early as possible. You can expect to receive most if not all by the plenary in W7.

- Tuesday 5-7pm in Week 9. Half-time demo event. We get together and every team showcases their status/prototype to the other teams. Ungraded.

- Week 7-14: work on your team project. Weekly SCRUM meetings are grade-relevant; in addition, you will have weekly meetings with a consultant from our staff, which are there to help you along and will not affect your grade.

- By Friday 2pm in Week 14. All documents and grading for the team project need to be finalized. Team meetings with the teaching staff can get arranged at a later date in the week than normal, but all grade-relevant info must be in by that deadline, and the final team meeting must have happened by then.

- Friday 2-5pm in Week 14 in the big open space in DLLEL. Open-house demo event. Show off your thing to your fellow course participants, friends, family, journalists, IC professors, etc.

2.6 Forming a Team

You will work on the team project in a team of six or seven students. We invite you to form teams with your friends and use the contact hours, particularly the plenary meetings, to shop around for team members.

2.7 The End of the Project and Course

At the very end of the course, on Friday afternoon of the last week of the lecture period, we will organize an open-house event in DLLEL where you will demo your projects and can invite your friends and family. You'll also be able to check out the other team projects. In the past years, this has been a very successful event, with journalists, lots of students, and families attending.

After this, the course is over. The projects belong to EPFL, and here is what happens to them from then on.

- We keep your thing in store for you for a while in case you want to demo it at some outreach event at EPFL, take it to a maker fair, entrepreneurship event, etc. EPFL and IC are always looking for cool demos for their outreach events (such as for highschoolers, the EPFL Open House, etc.), and they pay you for demoing your thing.
- It is possible to continue working on the thing after the end of the semester, even for credit (in a semester project, for example).
- It may be possible for you to buy your thing if the other team members agree. The total price will be half the price of the electronic components. This is not a suggestion. We are not trying to sell you something, but mention it as possible should you want that.
- We may keep the thing – maybe a future team will want to extend it or integrate it into their project.
- Eventually, if there is no further interest in the thing, it may get scavenged for parts.

2.8 After the Course

Hopefully, the course hasn't put you off making things ever again. Having gone through the various training units grants you the right to use the makerspace for your own projects for as long as you remain a student at EPFL.

Also, consider joining one of the MAKE projects – your skill set will be very valuable to any of them. Being organized entirely by students, and often run in a very enterpreneurial spirit, it will be a rewarding experience (if you have extra time to spend).

The course needs student assistants (AEs), who are of course paid for their work. We think this is one of the more desirable courses to be an AE for. Have a look at Chapter 10 in case you are interested.

Figure 2.3: At the end of the semester, you hand your thing over to EPFL.

Chapter 3

Good Citizenship

In this course, there are physical dangers to the health and even lives of humans. Moreover, since much of the course is a team effort, we depend on each other to achieve our goals. There are a number of rules in place that you need to follow to make this course a safe and enjoyable experience for everyone.

3.1 Stay Informed

While safety is our first concern, you cannot stay safe without knowing the dangers and how to avoid them.

This is not a lecture based course, so we try to provide you with the necessary information through tutorials and *particularly* this manual. While you are not expected to read this manual linearly from beginning to the end, we will point you certain chapters or sections that you must read. You need to know what is in this manual, so make sure to familiarize yourself with the table of contents of the manual, and skim through all of the manual at least once to know what is covered and what isn't. When you become aware that you will need to work with a specific topic or technology, say stepper motors or CAD, and the topic is covered in the manual, make sure to read the relevant chapter(s). Don't wait for us to tell you that you need to read a chapter – depending on what you choose as your team project, not everyone in the course needs to read the same sections of the manual. Reading is not optional – if you don't do this, it will be taken as negligence, and it is dangerous.

If you read parts of the manual and have questions, please make sure to ask. We are happy to help, and knowing that the manual needs to be improved is useful information for us.

3.2 Safe Conduct

The single most important thing to us all must be that everyone stays safe and nobody gets hurt. As the famous philosopher Dilbert has stated (paraphrased),

> The main goal of every engineer must be to get to retirement without having been blamed for a major catastrophe.

You must rigorously comply with all our safety rules, and you must be aware of any safety rules relevant to what you are doing. You must take the required training units, read Chapter 11, and, for any technology that you are using (such as motors and batteries) which is covered in this manual, look for technology-specific safety guidelines in the chapters covering that technology. This is particularly critical for LIPOs. Any unsafe conduct with LIPOs (Chapter 22) will be addressed most rigorously.

Not knowing the safety rules is not an excuse but further aggravates the situation. It is your responsibility to know. In general, if we find you to not know or not respect safety rules relevant to your project, we will take items or access permissions away from you for the remainder of the course, and it will be your challenge to adapt your project to deal with the new situation.

You also need to act in a way to avoid damaging your own property as well as the property of others (particularly, fellow students). You will use your personal laptops to connect to your things, and making mistakes can result in damaging or destroying a laptop. It would be a very uncomfortable situation for you if a team member's laptop were destroyed. We design the individual projects to make this near-impossible. However, for some team projects, the risk is real. We try to minimize such risks by providing you with USB isolators when there is a particular risk of voltage spikes that may damage laptops. You do not need to request these, we will push them to you for projects that need them.

However, USB isolators cannot afford perfect protection, and there are other things that we want preserved. (EPFL's property, including tools and

machines, and components we give to your for your projects.) You do not get around learning the relevant fundamentals in electronics (as covered by this manual) and being conscientious and careful.

3.3 Do not Compromise your Team's Success

At EPFL, we teach courses as a service to you, and of course we do not judge you if you choose to take a course and then do too little to get a good grade. This is, obviously, one of your freedoms. It is perfectly fine to skip an exam or a graded individual assignment if you so wish and are willing to accept the consequences to your grade. (And, really, there is no sarcasm intended here!) For team project courses like CS-358, however, the situation is a little different. If a student hurts the chances of others to succeed in the course, we do need to take mitigating action on behalf of these colleagues.

Obviously, the Golden Rule applies (treat others the way you want to be treated). Just imagine that you want to do well in this course and then you end up in a team with a colleague who does not contribute, or worse, actively torpedoes your efforts or turns every meeting into a fight. This would be really frustrating and unfair, because you are put at a disadvantage for no fault of your own. We don't want this to happen.

The main reason for the existence of team project courses in our curriculum is that teamwork is the principal mode of operation in the workplace. We want to give you a chance to practice and prepare. If you are not a team player, you are likely to lose your first job quickly once you go out there after graduation. Please use this course to practice teamwork and help your colleagues do the same.

The main purpose of the individual project is to prepare you for the team project and bring you all to a level where you can be valuable to each other in your team. If you decide to skip the individual project, or do less than a minimal effort to learn the necessary skills, this tells us two things: First, that having you on the team, rather than another colleague who acquired the relevant skills, is of disadvantage to your teammates, and, second, that there is a good chance that you have made the calculation that you can get a passing grade on the course based on the team project alone, and that your teammates will do the work, not wishing to fail themselves – thus a free eight ECTS points. Unfortunately, though rare, there are people who

think this way. You would be wrong in one sense, though: you would not pass the course, even if your teammates turn this project into a success. Your presence in the team would nevertheless hurt your team. Thus, in such a case, while we will allow you to join a team and take part in the team phase, it may not be in the team of your choice – we may reassign you in a way that we think causes minimal harm.

Another point, raised in Chapter 1, is that you have to keep the thing your are creating in the team project in DLLEL at all times, even if you find this inconvenient to yourself. Just imagine that one of your team members takes your project home and is never heard of again. A lot of work, as well as components that may be impossible to replace in time, would be gone. It may be impossible to re-create and complete the thing in time. This would be catastrophic to the other team members; nobody in the team would be able to complete the project and the course. We must make this scenario impossible. For that reason, noncompliance with the rule that the team project thing must remain in DLLEL will be severely sanctioned.

3.4 Find Like-minded Teammates

Different students participate in this course for different reasons and with different mindsets. Some of you may be taking the course because you have to, and others because you want to learn something, or like making things. You may have different attitudes towards how hard you are willing to work and how much time you are willing to spend. Some of you may have a very good grade point average and do not want it to be damaged by this course, others may just want to get a passing grade.

If you end up in a mismatch teams in which team members have very different viewpoints regarding these issues, this is a recipe for conflict. Working in a course like this, where you are expected to spend a very significant amount of time with your team mates, in a team with bad chemistry is torture. Do everything you can to avoid it by finding teammates with similar goals and a similar outlook on the points made above.

Start looking for teammates as early as possible, starting in Week 1 of the course. Talk to them. Find out their attitude and their capabilities.

If you consider yourself to be a top performer/very competitive, try to find other students who are like you.

If you tend to be at the other side of the scale, you might consider your

best strategy to be to still get into a team with such top performers, because they may drag you along to success. Actually, you are mistaken, this is not a great strategy. The course staff will closely observe everyone, and we will know who did what, and how well. If you are underperforming in your team, you are just more likely to be critcized and exposed by your teammates, and will have to work in a bad atmosphere throughout the semester. It will be no fun.

Remember, the overall success of the team project is not the sole determining factor for your grade. If you do badly in a great team, you will get a bad grade. If you do really well in a bad team, your will get a good grade.

Once you are locked into a team, you will have to live with this and do the best you can in your situation. If you work with some bad team members, you will have to deal with this, and how well you deal with it will have an influence on your grade. If you create a toxic atmosphere because you are frustrated by the weakness of your team members, we will hold this against you. If you show great conflict resolution and interpersonal skills, we will note this. Remember, mismatched teams and conflicts arise in the workplace too, and learning to deal with such issues forms part of the goals of this course.

3.5 The Tragedy of the Commons

While taking this course, you consume a number of resources, some of which are in limited supply – particularly space in DLLEL and time on machines such as laser cutters and 3d-printers. Thus, bad planning, causing you to manufacture new prototypes more often than necessary, though part of the learning experience to some degree, makes other colleagues wait or not get their turn. You may experience frustration due to this yourself – though we hope very little. It is thus important to think of others when consuming resources.

Do not use 3D-printing where laser-cutting is applicable: Laser cutting is much faster and will thus block fewer people from getting their own work done.

We are allowing you to apply your own judgement and to take certain items (such as screws, cables, and some basic electronic components) without anyone looking over your shoulder. But this requires your collaboration. You may find it convenient to over-provision and take, say,

more screws than you immediately need, because you think you might need them later and do not want to walk to the drawer holding screws twice. You may not know yet which kind of screw you need, so you pick up multiple different sizes. We do not want you to do this, even if you intend to return those screws that you find you don't need. Reasons include hygiene (remember the COVID pandemic) and the fact that there is a good chance that you will return the screws into the wrong compartment and will cause a mess that will keep others from finding the screws they need.

We are thus all in need of a set of shared resources, and a strategy of maximizing your own short-term benefit may hurt others, and, in the medium term, you again if others act in the same way. This is a well known scenario in economics, known as the Tragedy of the Commons. Have a look at this page:
https://en.wikipedia.org/wiki/Tragedy_of_the_commons

3.6 The Long-Term Perspective

Speaking from the perspective of the IC professors, we have observed ever growing student enrollment in IC study plans. This is of course great, because we think that the professions taught by IC are among the most important in existence.

But it also causes a scaling problem for our courses. DLLEL, though a really nice space, is rapidly getting too small for its many users. At times, the space and the machines are barely sufficient for us, and there is a rapidly increasing number of other courses from other faculties using the space, and in addition there are the MAKE projects.[1] For this reason, enrollment in the course is limited by SAC. The limits are revised, after discussion with the DLLEL staff, after each semester, based on our course's footprint in the past. We rely on the goodwill of the DLLEL staff and their good opinion of this course to a degree – they ultimately set the limits.

It has happened (fortunately, rarely) that some of our students have attracted their ire, by unsafe actions, not cleaning up after a session and

[1] What works in our favor is that we tend to have the largest footprint in the beginning of the semester, due to the individual projects, while everyone else, particularly the MAKE projects, seem to wake up only towards the final weeks of the semester. You may not believe it when you start the course, but towards the end of the semester the place gets really, really crowded.

leaving their workbenches dirty, or being outright rude. Note that you can be banned from areas in DLLEL or the entire makerspace. This hurts future students who want to take the course and may be prevented because of very small quotas. If a serious accident happened in the course, CS358 will probably even be abolished (as it should be, if we cannot maintain safety).

Please help us keep the course alive, so it can be offered to many more students.

3.7 Captain Obvious' Guide to Conduct

So, please

- Take the suggested tutorials and read the manual chapters that concern you, proactively,
- Keep everyone safe,
- Clean up after yourself,
- Don't waste resources, including machine time during times when they are in short supply,
- Treat others, including DLLEL staff, courteously,
- Be good teammates, and
- No cheating!

Thanks!

Chapter 4

Getting Started

So you have just joined the course. Follow the steps described below, starting with installing the software you will need on your laptop. You should install the software as soon as possible, because we will need it starting from week one and the first tutorial.

Have a look at this handbook and start reading, in particular Part I ("The Course", see the table of contents) on the organization of the course, Chapter 11 on staying safe, Chapter 19 on electronics and Chapter 27 on microcontroller programming.

If you have missed the kickoff meeting of the course in which we discuss how this course will be organized, make especially sure you understand what you need to do, when you have to be present, what you have to do for the course, and how you get a grade. Most if not all should be covered in this handbook. Make sure to read the manual thoroughly and ask as early as possible in case things are unclear.

Make sure you don't miss any deadline. The deadlines are discussed in Section 2.5. See also the course pages in moodle.

There are also several mandatory training units (some online, some in DLLEL[1]) that you must take to be allowed to use the facilities in DLLEL and to be in good standing with the course staff. There are restrictions to what we can allow you to do and what electronic components we can allow you to work with if you do not take all this training.

[1]Locate the building at https://plan.epfl.ch/?dim_floor=0&map_x=2532964&map_y=1152332&map_zoom=12

4.1 Course Prerequisites

You will need a reasonably modern laptop running either MacOS, or Microsoft Windows. You will need to run Autodesk Fusion 360, which in practice requires a MacOS or Windows installation.[2] If you are mainly a Linux user, it is therefore necessary to have a dual-boot (Linux and Windows) set up on your laptop. You **must** use Fusion 360 in this course – it is not ok to use another CAD program.

Unlike other IC courses, this course comes with risks to the health of participants as well as EPFL's and your property. There are some required training units (mostly related to safety) which you must complete before you can use the makerspace (DLLEL) and its machines. There are multiple training units that you need to sign up for and attend separately. You can sign up for these here:

https://make.epfl.ch/prototyping.

The required prerequisite training units are:

1. Mandatory online training. This is mainly an online course on moodle. At some point you have to indicate that you are taking the course "CS-358 Making Intelligent Things".

   After this, it should take a couple of days for you to be given access to the building via the Camipro card. (You only need this at night and during weekends and holidays; on normal workdays the building is unlocked. But from EPFL's viewpoint your are kind of trespassing until you have taken the safety training.)

2. "Laser cutting CO2 SPOT".

3. "3D printing workshop SPOT".

4. "Mechanical workshop SPOT".

Each training unit is quite short (less than an hour, typically). The laser cutting, 3d printing, and mechanical workshop units require your phyical presence in DLLEL.

These training units are not specific to our course, and are not part of our course. You can (and should, if at all possible), take them before the start of the course, though completing them in the first week of the

[2] Fusion 360 can run in a browser, but in our experience, this is not really usable.

course is possible too. These training units can be taken by any EPFL students, independently from whether they take a course that uses DLLEL. Completing them grants you certain rights that will remain valid for as long as you remain a student at EPFL.

We will keep track of which of the mandatory training units you have completed. You need all units completed to be allowed to join a team. Having 3d printing and laser cutting rights is a necessary prerequisite for making the parts for the individual project. If you are late taking this training, it thus has an impact on your grade!

Keep in mind that there are sign-up limits to some of the training units, which are offered at specific times and dates. Please do all the training at the earliest possible time. If you delay the training until the last moment, and others do that too, there may be no slot for you to do the training in time. The training slots being fully booked will be no excuse for you and will not gain you any deadline extensions in the course.

4.2 Software Setup

We will need a CAD tool for designing the physical aspects of your thing (both for the individual and the team project), a slicing software for generating "gcode" instructions for a 3D printer from a 3D shape designed with the CAD tool; and we will need an IDE for microcontroller programming and a CAD program for electronics design.

Do the following:

- Download and install the Arduino IDE. To do this, follow the instructions in Chapter 29. You *must* use exclusively the Arduino IDE for at least the individual project.[3]

- Install Autodesk Fusion 360:

  https://www.autodesk.com/campaigns/education/student-design

  This is free for students. Note that you must use this CAD software in this course, even if you are already familiar with other software, since we use it collaboratively.

  See Chapter 50 for more on Fusion 360.

[3]Other IDEs for MCU coding exist which you may use in your team project, but note that the teaching staff may/will not be able to help you with those.

- Install PrusaSlicer:

  https://www.prusa3d.com/de/page/prusaslicer_424/ .

  Select "Prusa i3 mk3s" as printer and "Generic PETG" as filament during setup. You will learn about using it in the 3D printing tutorial. Talk to Sebastien Martinerie if you want to use the new Prusa mk4 printers.

- (Optional) The electronics CAD tool Fritzing. The wiring and circuit diagrams that your will encounter in this course have usually been created using Fritzing.

  While this software is open-source, the developers have recently started charging EUR 8 for a built/binary version (see https://fritzing.org/). So you have the choice to either pay EUR 8, build it yourself from source code (see https://github.com/fritzing), or to download a pre-built binary from a file-sharing site.

  You do not need this tool for the individual project.

You can delay installing Fritzing until you need it in the team project (and you may not need it at all). The other programs you need from the start of the course. You need the Arduino IDE installed and working when you come to our first technical tutorial, otherwise you will not be able to keep up!

4.3 Course-specific Tutorials

The following is required:

- Familiarize yourself with this manual. Study the table of contents, read Part 1 (not just Chapter 1) on the organization of the course, and then at least skim the remaining chapters of the manual. Despite its length, this manual is in a very large font, with lots of whitespace, and is mostly easy reading. I promise you that reading the manual will save you time later by helping you avoid very time-consuming mistakes, or getting stuck with your project.

- Read the safety instructions chapter in this manual (Chapter 11) as well as the safety parts on LIPO batteries and motors in the respective

chapters of this manual. If you want to use either LIPOs, bipolar steppers, or brushless motors in your team project, first meet a TA and have them quiz you on their safe use. We will not hand out these components unless everyone in your team has completed this.

- Soldering training (takes about one hour). This is done in the main workspace of the makerspace (DLLEL 0-28). Visit an AE during his office hours (see Figure 2.1). If the AE is busy with other students, you may have to wait or return another time. There is a soldering practice kit in your component bag; bring it to your soldering training session.

  You need to solder wires to power your stepper motors to your micro-USB breakout board for the individual project. You may want to do this there.

- Three *technical tutorials*. These take three hours each in the first three weeks of the semester. In Spring 2025, these take place on Thursdays from 1 to 4pm in BC03. Do not come to this lecture hall in the weeks after W3 – nothing will happen there.

  - W1 tutorial: microcontroller programming and basic electronic circuits. Please make sure to bring your laptop and to install the Arduino IDE before the start of the tutorial. We will do some practical work. We need to connect a standard USB-A cable to your laptop; if your laptop does not have USB-A ports (as USB-c is become more and more standard not) you need to bring a suitable adapter. Topics covered:
    * Basic circuits and circuit schematics. Short-circuits and open circuits. The essence of Chapters 19 and 26.
    * Prototyping a circuit on a breadboard.
    * The NE555 circuit from the soldering practice kit; advice on building it up on a PCB and soldering it.
    * Your first working Arduino program.
    * Controlling electronics on your breadboard from the Arduino.
  - W2 tutorial: actuators, supplying power, mechanical recipes. Bring
    * your laptop and a USB adaptor if needed,

    * your component bag, and
    * your micro-USB cable in addition to the blue USB-b cable that is in your component bag.

  We'll do some hands on work. We want to help you get the steppers to run, and for that you need to **solder wires to the USB breakout board in your component bag before the tutorial** (see Chapter 6). Topics covered:

    * Working with sensors and actuators using in the individual project. (In Spring 2024: Hands-on controlling a servo and unipolar steppers from your Arduino).
    * Supplying power to big consumers (motors)
    * Motor types, motor drivers. Controlling brushed and brushless motors and bipolar steppers.

- W3 tutorial (starting in Fall 2024): Brushless motors and FOC; sensors.

  Since we expect some divergence in skill among you, we will not lecture in this slot, nor will we try to keep everyone synchronized, since the best-prepared among you would get bored waiting for the others. We will bring a number of sensors and actuators, which you can pick up and try to get to work, and we will help you along individually.

4.4 Handbook Reading and Watching Videos

Much of the handbook is written as a reference to be consulted when need arises, but there are some parts that you must read proactively. In the timetable of Figure 2.2, you see 10 hours alotted to this. Please read the following chapters of this handbook, we suggest in this order:

- Part 1 on course organization and safety. It is strongly suggested that you **read this before the kickoff meeting of the course**.

- Chapters 19 and 20 on electronics and Chapters 27 and 30 on microcontrollers and their programming, ideally before the first technical tutorial.

- Chapters 21, 37, and 38 on supplying power and motors, ideally before the second technical tutorial. You may skip the math in the motor section, but develop an idea of which kind of motor is suitable for which application.
- Chapters 50, 51, and 52 on CAD in Fusion360, 3D printing, and lasercutting before you embark on the 3D printing part of the individual project. The Fusion360 contains a very long step-by-step tutorial. If you prefer to learn Fusion360 in a way other than following this tutorial, that is fine.
- Further, watch all of the Week-3 videos of the course Software Enterprise (on software project management, git, and Scrum; about 105 minutes in total) at

  https://github.com/swent-epfl/public/blob/main/README.md#schedule

As you are looking for a team project idea, you may also search the web and watch youtube videos for inspiration. The youtube channels mentioned in Chapter 12 are warmly recommended. There are links to various videos, some quite entertaining, which may inspire you, at the ends of the various chapters of this manual.

Chapter 5

The Tutorials

5.1 Week 1

Bring the following:

- Your component bag (handed out in the kickoff meeting).
- Your laptop, with a USB-A port (or an adaptor in case your laptop does not have a USB-A port). We will use the blue USB cable in your component bag to connect your Arduino Uno to your laptop. The cable has a USB-B plug to connect to the Arduino Uno and a USB-A plug to connect to the laptop.

  On your laptop, have the Arduino IDE installed, and any driver necessary to connect to your Arduino UNO. Test uploading the Blink Sketch onto your Arduino Uno before you come to the meeting. Change the on-time of the LED in the Blink Sketch to a different duration to make sure that your code was really uploaded. (The Blink Sketch may have been pre-installed on your microcontroller.) See Chapter 4. There are plenty of tutorials online, too.

Do the following:

1. Wire up your Arduino Uno with your Laptop. Upload and run the Blink sketch (it is one of the examples that comes with the IDE: You do not need to download it.)
2. Change the delay times to have the on-board LED go on for .5 seconds and off for two seconds, repeatedly.

3. Add code to your blink sketch that writes an on/off message to your Serial Monitor whenver the LED state changes.

   Here is a tutorial for serial output:

   https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-monitor/

4. Pair up with a neighbor and (together) get one breadboard and three male-to-male jumper cables from a TA.

   Take the two LEDs and the two resistors stuck together via sticky paper tape out of one of your soldering practice kits. Don't lose them, you will need them in the soldering tutorial!

   Change the blink example to use an external resistor as shown on this page:

   https://docs.arduino.cc/built-in-examples/basics/Blink/

   You have all the components for doing this.

5. LED_BUILTIN is pin 13 of the Arduino Uno. Change your code and the wiring to use pin 12 instead of 13.

6. Now put the two LEDs in parallel, still turned on and off by pin 12 of the Uno. LEDs cannot handle large currents: They need to be put in series with a resistor, or otherwise they get damaged!

7. Now control the two LEDs from two separate output pins of the Uno and create a traffic light that forever switches between red and green. You need both resistors for this, and the three jumper cables! (Arrange the connections on the breadboard so you do not need four.)

8. Now make your two Unos talk to each other via I2C. Follow these instructions:

   https://www.instructables.com/I2C-between-Arduinos/

   Test it. Send each other messages via the Serial Monitor. Discuss the code with each other and try to understand it.

   Note: There is a bug in that code. The master sends an integer, while the slave tries to read a char. The character 0 is ASCII code 48. Fix this.

9. Optional, if you have enough time: Make your two Arduinos talk to each other using the SoftwareSerial library, rather than I2C. Write the code for it. Unidirectional communication with a dedicated sender and a dedicated receiver is ok.

5.2 Week 2

Bring the following:

- Your component bag
- A USB cable with a micro-USB plug.
- Have the power cables soldered to your micro-USB breakout board as described in Chapter 6. The soldering tutorial is a good opportunity to prepare this.
- Your laptop with two USB-A ports (via an adaptor, if necessary).

Do the following:

- Listen. We will discuss the quiz (see Section 2.5) of the following week and the kinds of questions that may be asked.
- Set up the electronic circuit of the plotter as described in the individual project chapter of the manual.
- Run the *TWO* examples in https://github.com/epfl-cs358/cs358-resources/tree/main/code/actuators/28BYJ-48_ULN2003

  Does the AccelStepper example not seem to do anything? Read the code! It is waiting for user input through the Serial Monitor!

  In the lowlevel.ino example, try to understand the code. This code does no use a library. See how the stepper actually works.

  Try to tune the delays (in microsecond) for optimal running of the motor: As fast as possible, and as smoothly as possible. With the wrong numbers, it does not turn at all.
- Connect your servo to the Arduino Uno and run some servo examples (they are preinstalled with the IDE, or google ("Arduino Uno servo example").

It is permissible to power a single SG90 servo directly from the Arduino Uno, without an additional power supply. Connect the three wires of the servo (usually yellow/red/black or yellow/red/brown, in that order) to a GPIO pin of the Arduino Uno and its 5V and one of the GND pins.

- When you are done with the previous steps, form a team:
  - install SimpleFOC with the library manager
  - install the esp32 boards with the boards manager
  - get the code from https://github.com/epfl-cs358/cs358-resources/tree/main/code/actuators/SimpleFOC/stepper_L298N/WEMOS_D1_R32
  - Get a SimpleFOC demo kit (one per team) from the TAs.
  - Before uploading any code, try sending commands to the box via the Serial Monitor. Make sure that the Serial Monitor is set to 115200 baud.

    The kit should send you some startup messages on booting. If you did not see them, press the reset-button on the WEMOS D1 R32 (the microcontroller board in the box, looks like an Uno, but black).

    You should be able to send commands such as "Mx" where x is an angle in radians. Try "M62.8" followed by enter, or "M0", or "M-10".
  - Now upload the code and try again.
  - Try holding on to the shaft of the motor, try to keep it from turning. It is strong!
  - Look at the code of the example and try to understand it.
  - Google "SimpleFOC". Look at the instructions on the web page. Try changing the FOC mode from angle control mode to velocity control mode.
  - When you are done, return the kit (box/power supply/USB cable).

5.3 Week 3

Bring the same items as required in Week 2.

We start with the quiz (20 minutes).

After that, you form teams (ideally the project teams we have decided upon on Tuesday) and you can try a number of things we bring along:

- The SimpleFOC demo kits from W2, assuming you did not have enough time to familiarize yourself with them.
- Bipolar steppers with the Arduino CNC Shield/A4988 stepper drivers.
- A number of sensors.

Chapter 6

The Individual Project

This is individual work that you must complete by yourself.

6.1 Design Philosophy and Learning Goals

The course comes with a steep learning curve – not in the sense that anything is particularly difficult, but there is a wide variety of skills to be acquired. This can be a little overwhelming.

We have picked an individual project for you in which you get to practice a little bit of everything – mechanics, electronics, manufacturing, and embedded software. It is important that you try to succeed in this project to be prepared for the team project and be a good citizen and able contributor in the eyes of your team members. You cannot proclaim yourself to be a specialist at this point and avoid some for these areas.

6.1.1 Design or Execution – Which is the Problem?

Generally, when things don't go quite as planned in a maker project and you get stuck, you have to ask yourself the following question:

> *Is there something wrong with my choices and design that make the project infeasible the way I envisioned it, or are my practical skills not up to par yet and I just have to learn/practice?*

If you embark on painting a realistic portrait, but your picture looks like those you are used to seeing pinned to refrigerators by doting parents

of young children, the problem is a lack of skill that can be developed.[1] If you try building a faster-than-light engine for a spaceship and you don't succeed, it may come down to impossibility. At some point in your team project, you may get to a point where you just don't manage to achieve your immediate goal. If you do not know the answer to the question phrased above, this can be very discouraging. How do you find out whether you just need to improve a skill or whether you need to revise your design/plans?

We don't have the same problem in software: There, simply speaking, the specification is the final artifact. When we make a physical thing on the other hand, the design may be perfect, but if it is badly executed, the thing does not work.

The individual project is meant to help in several ways.

- It is meant to level up your skills in all the areas relevant in this course, so you are less likely to get in trouble during your team project.
- It will develop your spidey sense[2] regarding what is hard and what is achievable. Of course we will help weed out infeasible team projects before you embark on them, but being able to apply informed judgment yourself is important.
- It will allow you to practice this in the context of a project that has been done before, which we have thought hard about, and which we completely understand. The possibility that you are working on a project that is too difficult is excluded, so if things do not work out as expected, you know that you need to hone your skills and that is it. That peace of mind will make things much easier for you.

6.1.2 The Need to Tinker

Beginners often underestimate how different in precision parts manufactured by them are from items of their daily use that are industrially produced. A part of your car engine, or a Lego piece, are manufactured to tolerances you will not achieve by 3D-printing. Even industrial production requires postprocessing, tuning, and quality control steps that you cannot savely skip. (And a new car engine needs an initial run-in despite precise manufacturing and quality control.) You will need to learn to identify

[1] Though few of us will become Rembrandt even if we practice.
[2] https://www.oed.com/dictionary/spidey-sense_n?tl=true

when parts tolerances are making your thing work worse than expected, and how to postprocess your parts (potentially using sandpaper or a file – the hand tool, not the data storage abstraction) and tune your thing. The individual project gives you an opportunity to familiarize yourself with these challenges.

Pay attention at the 3D printing training session – there are many pitfalls. Also see Chapter 52.

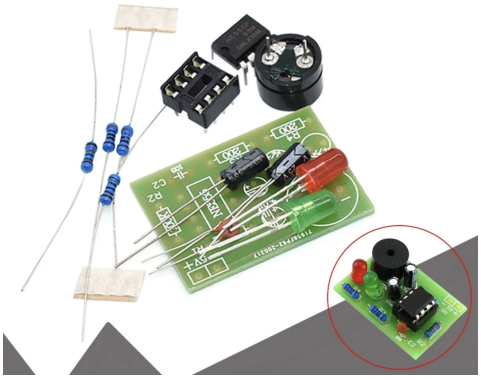
6.2 The Component Bag

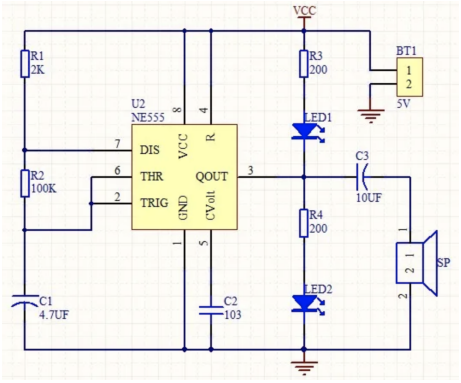
We will hand out a bag with the electronic and mechanical components you will need to do the individual project to each one of you. The bag also contains a soldering practice kit that you need to bring to and assemble during the soldering training.

The contents of the bag, and the thing made in the individual project, will be yours to keep if you complete the individual project. If you unenroll from the course (by the deadline at the end of the second week of the semester), you must return the components.

In the Fall of 2024, the component bag contains

1. A soldering practice kit.





2. An Arduino Uno microcontroller board (unless handed out earlier).

3. A USB cable with USB-A plug (to be connected to the laptop) and USB-B plug (to be plugged into the Arduino Uno). Unless handed out earlier.

4. 10 jumper cables female to male. Don't pull apart until you have to, see how they will be used.
5. 4 jumper cables male to male.
6. One SG90 servo.
7. Two 28BYJ-48 5V unipolar stepper motors.
8. Two ULN2003 stepper motor driver boards.
9. A micro-USB breakout board. That is a small PCB with a micro-USB port on it as the only component. You will need to solder jumper wires to it – see Chapter 6.
10. Two pieces of GT2 timing belt: one 40cm long and one 62cm long.
11. Two GT2 pulleys with 5mm axle diameter.
12. Six large nails 160mm * 5.5mm

The component bag will be handed out in the first plenary meeting on Tuesday Feb. 18 in DLLEL 1-50. If you miss this meeting, you can get your bag from Catherine Gagliardi in BC212. If you drop out of the course before the unenrollment deadline, you need to return these items; otherwise they are yours.

Note that there are drawers in the DLLEL open space where you can pick up items such as more cables, screws, etc, as needed.

6.3 The Spring 2025 Project: a 2D Plotter

You will build a 2D plotter – a machine that moves a pen to draw an image on a piece of paper. The plotter will have two degrees of freedom – two axes – along which it can move the pen, plus the ability to lift and lower the pen so you can move the pen to another position on the paper without drawing.

The plotter is inspired by the project shown in this video:[3]
https://www.youtube.com/watch?v=d8ZynbxfBDo

[3]Also note the source code, circuit diagram, and 3D files linked in the comments section to the video. Do NOT 3D-print these parts.

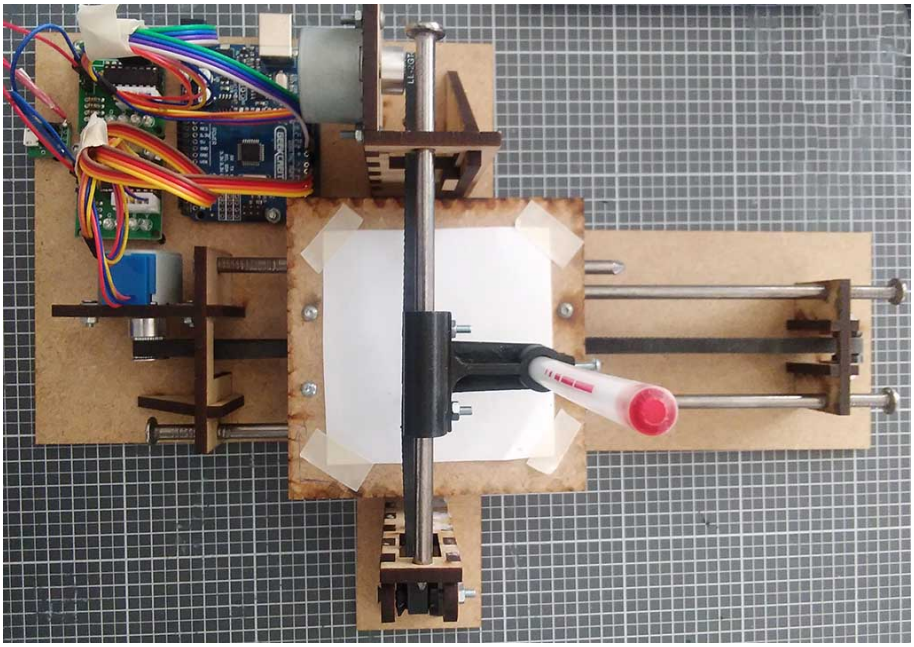


Figure 6.1: The initial build of the plotter (non-lifting pen holder).

We will make a plotter very much like it, both mechanically and electronically. Since there are many of you, we need to reduce the 3D printing times. We have created a design based on laser-cutting which you find at https://github.com/epfl-cs358/2024sp-2d-plotter/ The main file is Hardware/3D_Files/2024-spring-plotter.step . Open this file in Fusion360. It contains all the parts you need to manufacture to create an initial working build, and shows you how to orient and assemble them. Note that the pen holder in this design is not able to lift the pen. It will be a task for you to add this feature, but we recommend to first build – and make work – the design given to you (see Figure 6.1), to build up confidence.

6.3.1 Tasks

- Design a pen holder that can be lifted using the servo motor. Your new parts must be attached to the part PenHolderSlider as designed by us (you must not replace it). The distance of the pen to the PenHolderSlider must remain the same as that achieved with the part

PenHolderRigid, since the overall plotter is designed to allow for maximum coverage of the drawing area with this distance. You may use the old lifting penholder assembly provided in our .step file as inspiration, but you will have to redesign it rather than create an adapter, as that would make the distance between pen and penholder axle too great. Also, the old lifting design (which is the same shown in the youtube video) is evil – it requires the servo to force-deform the penholder when rotating and will destroy the servo in the medium term. Try to do better.

Further, you are required to protect the servo from damage resulting from pushing the pen into the drawing plate too hard/too deeply. A servo will be destroyed if you try to make it turn where it cannot turn because there is an unmoveable obstacle. You must take this into consideration already during development and testing, otherwise you will destroy your servo. We suggest to take a spring from an old ballpoint pen (or multiple springs) and include it into your design, so that, if you move the pen down a little too far, the spring takes the load, and not the servo. You may prefer drawing with a felt tip pen, rather than a ballpoint pen, so less pressure is required.

- Build the plotter.
- Programming assignments:
  1. Draw a large square and inscribe a circle[4]. Maximize precision. (Make the start and end-point of square and circle meet, and the sides of the square be tangents to the circle. Make the circle's curve smooth.)
  2. Create an autopen program that writes out a text typed into the Arduino IDE Serial Monitor in D'Nealian cursive. You must support at least 15 out of the lower case letters a to z, and whitespace. Letters in a word have to be connected according to D'Nealian cursive, and the pen needs to be lifted as one would do in cursive handwriting (for letters such as i or x). Support line breaks; support console input of up to 40 characters (consisting of words of no more than 10 characters) and be able to write out the words

[4] See https://www.varsitytutors.com/hotmath/hotmath_help/topics/circles-inscribed-in-squares

in a reasonably-sized font in multiple lines of no more than 10 characters.
https://en.wikipedia.org/wiki/D%27Nealian

6.3.2 Required Items

- Everything in your components bag minus the soldering practice kit.
- You need to contribute these things: a suitable pen[5], whatever springs or elastic parts your pen-holder assembly requires, and a micro-USB cable to power the stepper motors. It is very likely that you have some lying around at home – to charge mobile phones, for instance. Note that it is possible to confuse micro-USB with USB-c: the former have D-shaped plugs which can only plugged in in one way.
- Your laptop set up as described in Chapter 4.

Do not forget to do the required training units as early as possible. You need to 3D print and laser cut, and you won't be allowed to by DLLEL staff unless the respective training units are completed.

6.3.3 How to Proceed

The following instructions are here to supplement your common sense. With some experience, you should not need them. If you do not understand them on first reading, think for yourself what should be done and how things could go wrong. Then re-read. Everything should become clear.

We suggest to do things in this order:

1. Prepare parts:
   - 3D-print and laser-cut the provided part files.
   - Solder wires to your micro-USB breakout board. You need two wires with female connectors and one wire with a male connector soldered to the + power line of the breakout board and four wires (two with female and two with male connectors) soldered to the GND line of breakout board.

[5]Felt-tip pens are recommended because they don't require much pressure to deposit pigment on paper.

Important: The solder points on your USB breakout points are not necessarily all labeled the same way. Search the Web for a pinout diagram on which the pins are labeled like on your board. It is likely that the solder points for the power lines are the two outermost points, probably labeled something like VBUS (+) and GND. The differential data lines D+ and D- are NOT power lines but signal lines! Do NOT solder your wires to these solder points!

A + and a GND cable each go to the two motor drivers and to the power lines of the servo; connect the fourth GND wire (the one with the male connector) to one of the GND ports of the Arduino.[6]

Trim the wires to a reasonable length before soldering to create a clean appearance for your electronics setup.

2. Prepare the mechanical and electronics assemblies separately, and test them:

   - Find a circuit diagram for your electronics assembly at https://github.com/epfl-cs358/2024sp-2d-plotter/blob/main/Images/Circuit_Diagram.png [7]

     Wire up your steppers, the stepper drivers, the Arduino, the micro-USB board, and your Laptop, and make sure you are able to control your steppers – that is, there is no plotter here yet. Observe how the output shafts of the stepper motors rotate, but do not connect them to a load yet. You will need to move both steppers at the same time. Figure out how to do this. Two pointers:

     (a) the AccelStepper/MultiStepper library and

     (b) direct (multi-)stepper control using analogWrite. This is the low-level option, but it isn't hard.

     You can use whatever approach works for you. Explore with Google and on Youtube.

[6] This is not shown in the circuit diagrams of the summer 2023 project or the Youtube video, but it is good practice and necessary if you do not supply power through the two USB cables from the same source.

[7] Note that the example programs at https://github.com/epfl-cs358/cs358-resources/tree/main/code/actuators/28BYJ-48_ULN2003 may use a slightly different wiring. Read the code and check to what extent either wiring or code needs to be changed – which signal pins of the Arduino Uno are used to connect to the stepper drivers, and what does the code assume?

Figure 6.2: Mounting the timing belt to the pen holder assembly.

- Create the initial mechanical build as described in the next section. Do not connect the stepper motors yet. Make sure the 3D-printed parts slide well on their axles. The resistance/friction should be minimal (otherwise you may damage your motors), but nothing should be loose or wobbly either (or else your drawings will look bad). If necessary, tune your parts. Start at the axle holes of the 3d-printed parts, particularly at the sides that were attached to the print plate (check for the elephant foot problem). Also check whether your nails are straight or bent.

  All the screws you need are M3. Take the shortest that work. Bigger isn't better.

3. The wedding: Mount the motors to the mechanical assembly but do NOT power them up. Set up the belt loops by pinching the ends of the belts in the emplacements on the 3D-printed parts (see Figure 6.2). If the loose ends of the belt moving the pen holder do not fit in the hole prepared for them, you may trim the belt a little, but only as much as absolutely necessary. Do not cut too much!!

   The belts should be tensioned enough so that there is *no play* of the 3D printed parts moving on the axles.

4. Now un-pinch one end of each belt again and re-pinch it in such a way that there is about 5mm of play when you try to slide the moving part

along their axles with your hand. We are doing this so that the plotter will not be damaged in case you make a programming mistake. If a stepper motor now tries to ram a moving assembly into one of the axle holders, the belt will hopefully skip and the stepper motor's gearbox will survive. When you test, be ready to unplug your Arduino to stop any motor movement. This is better than unplugging the motor power supply (see back-EMF in the technical chapters).

5. Implement the first (inscribe a circle in a square) and second (a curve of your choice) programming assignment. Test-draw.
6. When you are confident that your programs are correct and don't try to move the pen position outside bounds, you can tension the belts and tune further until you get beautiful pictures. Optimize your code until the drawing is exact and pretty, and the curves are smooth.
7. Now design, manufacture, mount, and test a lifting pen holder. Adapt your programs to use it.
8. Now do the third programming assignment.

6.3.4 Initial Mechanical Build

In order to save you redundant work, we have already exported the .dxf files for lasercutting and the .stl files for 3D-printing for you. Get the .dxf files from the directory Hardware/3D_Files/dxf/ in the github repo. Laser-cut these from 4mm thick MDF (it must be 4mm thick, or else the parts won't fit together). You need one of each except for bottom_side, left_side, and middle_side, of which you need two. Try to place these dxf files to minimize wastage.

3D-print the five parts in Hardware/3D_Files/stl/. You need two Blind-Pulleys and one each of the other four designs. Use the 0.3mm draft settings in PrusaSlicer. The parts have been designed to be printed **without** support. Do not use a different layer height or quality settings – that would not solve any parts quality problems you might encounter, but may make the parts not fit. Make sure you orient your parts correctly on the print plate in PrusaSlider: Think of which holes need to be really round. Some overhangs can be tolerated by the printer, but you cannot print surfaces that entirely float in the air. There is only one correct way to orient the pulleys!

It shouldn't be difficult for you to assemble the plotter from these parts. Glue the MDF parts together. The parts PlateHolderWide and PenHolderSlider contain slits to pinch the belts in. You do NOT need to glue the belts.

Before you are allowed to 3D print parts for the first time, you must show your design to a DLLEL course or teaching assistant and have it approved. To do this, have PrusaSlicer open with your parts placed on the virtual printing plate and your setting set up the way you mean to print this. We do this to maximize your success and to protect the 3D printers, which can be damaged by a bad print job.

We recommend that you glue some scrap MDF pieces of the same thickness to the bottom side of your base plate, but not over the screw holes, to allow for the height of the screw heads (assuming that you screw on your electronics boards from the bottom) and not have the base plate deform by its own weight. An alternative is to take a power drill and sink the screw heads into the MDF of the base plate.

6.3.5 Deliverables and Grading

Upload a single .zip or .tgz archive file to moodle. (Deadline: End of Week 5.) This archive should include the following items:

- Photos of your plotter. Include at least one picture showing the entire plotter from above. Additional detail pictures of the wiring and your pen holder are recommended.
- CAD design files: the file 2024-spring-2d-plotter.step with your retractable pen holder assembly added.
- Your source code files.

Submit as early as you can to not miss the deadline. You can re-submit and overwrite your submission as often as you like.

In the week 5 plenary meeting, you will demo the mechanical and electronics assembly. If the retractable pen holder is not ready, you may show the fixed pen holder, mounted on the plotter. This is a visual inspection only, you don't need to run software. The grade weight of this is two points, given in all-or-nothing manner.

In the Week 6 plenary meeting, you will demo the plotter to a TA and answer questions on what you did and about your source code. Should you

for some exceptional reason not be able to attend this plenary, negotiate a demo meeting for an *earlier* date.

Grading criteria include

- Is the wiring and soldering cleanly done (and correct!)? Do you supply power correctly?
- The quality and tuning of the mechanical assembly, as evidenced by the precision of the drawings produced by the plotter. Is the assembly of the MDF parts precise? Are things that are suppose to be parallel resp. perpendicular indeed so? Do the moving parts move smoothly on the rails? (Be ready to detach the belts to let the TA check.)
- Quality and execution of your lifting pen holder design. Does it conform to the specification (distance of pen axis from the pen holder rail)? Point deductions for unreliable or imprecise functioning and excessive material use.
- Quality of the software you have written, and how well it works. How clean and precise are your curves?

The individual project accounts for 20% of the course grade, of which 2% is for the week 5 mechanical assembly demo, 5% is for your design and execution of the lifting pen holder and programming assignment 1 (circle and square, with high grade weight on precision and the quality of your lines and curves), and 13running the second software assignment (the cursive autopen).

Remember that this is individual work. You are not allowed to collaborate or share work with other students in the course. You are allowed to use any resource that you can find on the internet and that existed at the time of the start of the course. Acknowledge your sources of source code, if any, in comments to your source code.

Chapter 7

Team Project Proposals

In previous installments of the course, each student had to individually come up with a project idea and prepare a team project proposal following the guidelines of this chapter. While we generally don't do this in this semester, you will need to read this chapter closely if you want to propose a project idea of your own, and, as a team, when you flesh out your project proposal/documents jointly, you will have to take the information given here into account.

A team project starts with creating a team and picking a team project proposal from the approved proposals. The team will then jointly revise the proposal according to the guidelines of Chapter 8 and Section 8.5 in particular. Don't conflate the initial proposal, to be produced individually, and the revised proposal, to be produced by the team. Both are to be submitted on moodle – separately and by distinct deadlines.

7.1 Finding a Suitable Project Idea

The deadline for your project proposal is aligned with the end of the first phase of the course, where you are asked to read parts of this manual, take training sessions and attend tutorials, watch videos, and search the Web. We expect you to explore and develop a degree of confidence and understanding of what is feasible and what is worthy. Read, in particular, the Chapters 13 on beauty, 14 on "goodness" and usefulness, 15 on intelligence, 16 on complexity, and 17 on scale.

We want to make physical things that react to the world. From this interaction arises the perception of "intelligence" referred to in the name

of the course. Computer vision or machine learning are welcome, but not required for your project idea to qualify. The project should have a considerable physical component – the microcontroller should interact with the physical world through sensors and actuators (though some excellent projects had only one or the other). However, the software side of the project is key. This is a computer science course, and we want you to create a very worthy software side to the project. Ideally, the software side of the project should remain the bigger challenge than the hardware side.

Go back and forth between generating ideas and checking their feasibility. If you want things to move, read up on motors and the implications of a particular motor type (Chapter 38). If you need to sense your environment, make sure to identify suitable sensor technology and explore the implications of your choices. It is not sufficient to proclaim "there shall be sensors" in your project – you have to tentatively decide on what kind of sensors to use and tell us how these sensors are supposed to perform their task. If you are considering computer vision, also consider alternatives, such as LIDAR, ultrasonic distance sensors, and touch sensors. (See also Chapter 33.) If the mechanical side of your project is challenging, or you are unsure of the force arising or needed to make your thing work, make sure you understand the material covered in Chapters 37 and 45, and search the Web for similar project and the lessons learned by their creators.

Do not forget that this is an 8-credit course to be worked on in a team of five or six people; we are talking of a project that should be close to a 1000 person hour effort. We do understand that you lack experience regarding how long things take, particularly regarding challenges beyond software. Still you need to make a conscious effort to propose a well-scaled project. In case of doubt, look for similar projects and how they were realized. You will find that most individual challenges, like reading out a particular type of sensor or controlling a specific type of motor isn't a big deal and shouldn't take too long. However, some aspects of your projects may be risky in the sense that it may be hard to tell at this point whether the planned approach is feasible and whether you can get your thing to work as planned. In that case, schedule in your project proposal an effort to experiment and create a prototype that addresses that specific difficulty before you attempt to create the complete thing.

Better err on the side of making the project (slightly) too big than making too simple. We'd rather see an ambitious project than a boring one.

You can always identify some parts of your project as optional, and we will help you overcome difficulties and, if necessary advise you on how to cut down the project to something that is feasible. Still, if you propose a project that is so obviously over-ambitious that we can justifiably argue that you should have known better, this will also be held against you. Try to gauge the effort as precisely as you can. This is hard, but being able to do this is a extremely valuable skill that you should develop.

Ideally, a project does not just consist of a large number of things to keep you busy, but has at least one technical challenge that is interesting/"hard" on its own. Here are some examples.

- A thing that makes interesting use of field-oriented control (see Chapter 43), such as doing ultra-fast and ultra-precise movement, a self balancing robot, or haptic technology.

- Computer vision purely on an ESP32-CAM. It is possible, but so far in this course, teams have done the image processing on their laptops only.

- Create a thing with substantial mechanics challenges, such as a thing with a cable drive, a good (cycloidal or strain wave) gearbox of your design, or a non-SCARA robot arm. (See Chapters 45 and 47.)

- Biologically inspired mechanical designs or movement, such as in a robot that moves like an animal, or a biomimetic (human-like) robotic hand.

- Advanced sensing, such as of biosignals, or using an accelerometer/inertial measurement unit.

Again, these are not project ideas, but examples of features that would make us consider your project technically challenging/interesting.

For inspiration, Chapter 60 provides a list of team projects from previous installments of the course. Note that the course had 6 ECTS points in 2022 and 2023 and has 8 ETCS points from 2024 on, and many things are changing compared to 2023. Our criteria for acceptability have evolved over time, and some of these projects might not qualify today (see the criteria in Section 7.2).

7.2 Exclusion Criteria

Here are the criteria that make a team project idea suitable or disqualify it.

- Projects that would be better off in an IC software course because the physical component is trivial will be turned down. For instance, a project in which you solve a challenging computer vision problem and, based on that, turn a lamp on and off is not a good fit for the course. Any project which is essentially a software project where you just display the result of your computations on a display that you want us to buy or via an array of LEDs is too simple and thus unsuitable.

- A considerable software/programming side: This is an IC course, and software is your strength. Project proposals with an overly simplistic software side will be turned down. A new kind of skateboard may be a cool maker project, but it is not suitable as a project for this course.

- Work with Microcontrollers. We want you to get out of your comfort zone of computer programming and work with a microcontroller (i.e., to do low-level programming on a device without an operating system). For that reason, you cannot use a Raspberry PI for the project even if you personally own one. Proposing projects that involve microcontrollers interacting with computers (your laptop) and where some advanced computation (such as computer vision and machine learning) is done on the computer is ok.

  Without exception, the only kinds of microcontroller boards to be used for the team project are those listed in Chapter 28. It is possible to combine multiple (different) microcontroller boards in your project.

- The right size of challenge: The team project should be worked on by teams of five to six students. The hours to work on the team project (roughly 150 hours per student) have been discussed in Chapter 1. Try to match the workload to this.

- You will create a bill of materials consisting of mechanical and electronic components we will have to provide you with. You will have to

determine prices for each of them according to set rules, and the sum of these prices must not exceed CHF 250 per team project.[1]

- Be realistic about the technological sophistication you can achieve during this course in fields that you are not yet expert in. You should challenge yourself, but, for instance, if you have little electronics experience, you cannot create a new and better smartphone. If your team has a member who is an expert in some area, this may qualify you for a project that we might otherwise deem unsuitable.

- Projects that violate our safety rules (e.g. exceeding voltages of 12V) as well as weapon-like things are forbidden.

- Any thing that is likely to be severely damaged or lost by a slight design mistake, malfunction, or programming mistake is unsuitable. Here we particularly think of things falling from heights, being smashed by heavy objects, or being short-circuited by liquids. This includes any flying things (drones), boats, and submarines. Again, **no drones** without exception[2] in this course, no matter how noble your goals! No exceptions will be made.

- Avoid projects that will make it extremely hard for you to test and demo them because the are designed to function only at a place or environment other than DLLEL and the EPFL campus. For example, a volcano exploration rover, a bush fire sensor, or a garbage collection vehicle for the Pacific Ocean are unsuitable for this course. **If it cannot be developed, tested, and demonstrated in DLLEL, it isn't eligible.**

- Avoid projects that make you depend on external factors outside your control. It may be attractive to contribute to a MAKE project by tak-

[1] Unless otherwise stated, the things made as team projects and all these items remain the property of EPFL and you cannot keep them at the semester. In some cases exceptions or buying item at fair prices may be possible; talk to us.

[2] FYI, all EU laws and regulations regarding flying drones apply in Switzerland. Some drones require a license, and there are additional regulations relating to data protection and privacy laws in case your drone mounts a camera. In addition, every drone flight on the EPFL campus needs to be applied for and approved several days in advance. Finally, most of the EPFL campus including DLLEL is within the five-km radius of Lausanne airport – see https://www.bazl.admin.ch/bazl/en/home/drohnen/general/drone-maps.html.

ing over one of their components; however, this will make you depend on the progress they make and their availability to give you information and interfaces. Such projects have much longer timelines than our course, and you may lose too much time waiting for essential things from them. You cannot combine your project with another ongoing project in another course. Continuing and expanding on a past project (including a past course project from an earlier semester) is, in principle, possible.

7.3 The Project Proposal Document

Teams can only pick team project topics from among the pre-approved project proposal documents. Each student has to write such a project proposal at the beginning of the semester.

A proposal should cover the following points:

- A high-level description of the project idea. What do you want to make – what will it look like, what will it do. Describe it in sufficient detail so the teaching staff can form an opinion of the feasibility of the project, risks involved, items needed to be bought, etc., without reading your own analysis of these issues in the following sections of the document. You may (and should) provide pictures, drawings, and links to illustrate your idea and reference other related projects or items. Even pictures of already existing things that can help in explaining your idea can be useful.

- Do your research on the Internet on related projects and resources, including other open-source projects that have instructions that you want to use or that use designs that inspire you.

  Describe these related projects, sources and resources, and provide links to them.

- "User stories"[3]. Describe the features of your thing, ideally from the perspective of an end user. Such end users may include you, even if it is just to play with/enjoy your thing.

[3] https://en.wikipedia.org/wiki/User_story

It should include IC public relations people: IC is always looking for demos for various audiences, including high school students, prospective EPFL/IC students, alumni, and members of the general public. We do demos at events such as the EPFL Open House, the IC Research Day, etc., and if you are available for demo your thing you made in CS358, you can be paid for your effort.

Another user story is the one of the professor: I would like it if part of the project (a software library, a vehicle platform, ...) is re-usable for other future projects.

The key challenge here is not so much to come up with innovative new user stories, but to make sure that your project idea, if executed correctly, can achieve the requirement expressed in the user story.

- Closely related to this, product management: Take a long-term viewpoint. What afterlife can your project have. What follow-up projects for future CS-358 students are there? Might your thing acquire "real", non-EPFL users? Is there a startup (company) waiting to happen?

- Back of the envelope cost estimate. Is it realistic to believe (without first finishing a full design) that the costs of the project will not break the bank? (Make sure to read Section 8.3 to understand the constraints.) Obviously, a project like the robot dog Spot[4] that requires 12 high-tech servos that each cost around CHF 1000 is outside the scope. What about your project?

Have a look at Chapter 8, where we cover what we expect from the revised proposal produced by your team on the basis of your proposal, should it be selected for implementation. The closer your initial proposal is to what we expect from the final revised proposals, the better!

7.4 How we grade and select proposals

Please submit your proposal as a pdf file on moodle.

Your project proposal will be graded based on overall quality, compliance with the set criteria (violating any of the criteria of Section 7.2 will be strongly penalized), feasibility, the right degree of ambition (not too hard

[4] https://www.bostondynamics.com/products/spot

and not too trivial), completeness according to what we asked a proposal should contain, clarity of exposition, and originality. Remember that the weight of this proposal in the overall course grade isn't very large (5%); however, we still need you to propose good projects for you all to have a fun course.

We will approve and publish a small number of proposals, which can be chosen by teams for their team projects. Whether your proposal is approved has no bearing on your grade, and will not be negotiable.

We hope that there will be many strong proposals. We will limit ourselves to approving a relatively small number to make it simpler for your to form five-member teams. If we approve too many proposals, each one of you might pick your own proposal and struggle to convince other students to abandon their proposals to join you. This may delay the formation of the teams and put us behind schedule.

When approving proposals, in addition to the grading criteria we have discussed above, we will above all use our experience to exclude projects that are likely to cause you frustration, delays, and crises in the team project phase. While we do not expect proposals to be completely fleshed out – more detailed design documents are produced once the teams start operating – proposals that require too much guesswork regarding high-level choices, particularly those affecting feasibility, will not be approved.

Apologies in advance if you took the course to work on a particular project and we don't end up approving it or you do not find a team for it.

Chapter 8

Team Project Documents

First you need to pick a team project idea, expressed in the form of an approved proposal, and assemble a team. A team should consist of five members; if the number of students taking the course is not a multiple of five, we will allow some teams to consist of six members.

8.1 CAD Design

You and all your team members will use Fusion 360 for your CAD design. One team member should create a project directory in Fusion 360 and invite all other team members as well as your teaching assistants into it.

Your CAD design is a living document that you keep maintained and up-to-date throughout the duration of the semester.

That is, there is no single deadline by which you will provide CAD drawings, after which they will be allowed to become stale.

We will require you to create a **complete** prototype 3D design before you are allowed to 3D print or machine any mechanical parts, but you will continue to keep your design up to date as you iterate on your prototype.

Again, it is not sufficient to draw parts as need arises and 3D-print them. You have to create a complete design before manufacturing. This will allow us to reduce materials wastage and will save you time that you might otherwise waste on making parts that you cannot use because they will not fit your overall thing.

8.2 Source Code and GitHub Repository

The course staff will create, for each team, a public github repository in organization https://github.com/epfl-cs358. All team members plus the teaching staff will get write access to that repo. The name of your repo will be prefixed by the year of the course, followed by a name of your choosing. So, for instance, if your thing is called "doomsday device" and you take the course in Spring 2024, your repo will be called 2024sp-doomsday-device.[1]

This repository is for all your design documents, including source code. We ask you, to the maximum extent possible, to work on source code in individual forks and contribute your changes to the master repo through pull requests.

All documents, including the Revised Project Proposal, Bill of Materials and the Risk Assessment Document will be maintained in your GitHub repository. You will maintain your CAD designs in your Fusion360 team folder during the semester, but at the end of the semester you will export your CAD documents and add them to your GitHub repo to make them public.

8.3 The Bill of Materials

You will need to create a list of the items we need to buy/provide for your project. We call this the Bill of Materials (BOM) and it is one of the documents that your team will be requested to create. Already the project proposal document has to contain a (less formal) version of such a list and cost calculation. The BOM document is expected to be complete and accurate; forgetting to include an item may cause trouble later on, since you will not have that item at your disposal and will have to improvise.

Since we need to do one irreversible action based on this document – buy things – there is a deadline for this document (see the course timeline in Figure 2.2), before which the BOM needs to be ready and after which we will not edit it anymore.

You **do not need to list** the following:

- consumables that will be processed in 3d printers and laser cutters

[1] A https://en.wikipedia.org/wiki/Doomsday_device will be violating our topic exclusion criteria, though.

(such as 3d printing filament, boards for lasercutting that the mechanical workshop has in store)

- small electronic components such as resistors, potentiometers, capacitors, diodes and transistors, push buttons, cables, that we likely have in stock[2]
- breadboards and protoboards
- standard metric screws and nuts.

You **must list** all other mechanical and electronic components you will need, such as

- microcontrollers, ICs, sensors, actuators (motors), motor drivers, voltage and logic level converters, power supplies, etc.
- mechanical components that you cannot make yourself (in particular metal parts, such as linear rails, ball bearings, axle connectors, etc.)
- Special connectors (plugs) that you may need, such as connectors to your batteries or power supplies.

For each of the listed items, first look at the "Parts in Stock for CS358" sections of this manual if it is one of the items listed as in stock, and if it is, use the price indicated in the manual. You must use items in stock if possible. If you had a similar (but distinct) item in mind, check if any of the items we have in stock will do and use them instead if at all possible.

Otherwise, find and provide a link to a page in a Swiss online store where this item is currently in stock (so no great delay for delivery is to be expected). Its price must be shown there; list that price in your document. Typical suppliers of electronic components in Switzerland are, for instance, distrelec.ch, conrad.ch, reichelt.com, digitec.ch, and mouser.ch . Online market sites like Wish or fruugo.ch do not qualify since the sellers are usually NOT located in Switzerland (also, some have rather bad reputations). Amazon.de/.fr/.com do not qualify. If there is a key item that you absolutely need and cannot find in Switzerland, you may provide a link to a Web shop outside Switzerland. We will review this and see whether it is feasible for us to get it (and get it quickly enough). Please do not involve us

[2] If you mean to follow a recipe calling for very specific components, talk to us to find out whether this needs to be ordered.

in discussions about your personal successes in buying from sites outside Switzerland and delivery promises made by sites such as amazon.com/.de. You may have made good experiences in the typical case, but we have to manage risks. Anything from abroad can be stuck in customs for weeks if we are unlucky.

Do not use the list of electronic components held in stock in DLLEL (managed by Sylvain Hauser). Their stock is for emergencies only. They might help you out later if you realize you forgot to ask for an item (as will we with the CS-358 stock if we can), but you must not use them as a parts source in the BOM document. Of course you can use their parts list as an inspiration for what kind of electronic parts you can ask for, but if you do, you have to find and indicate a separate source.

The total price of the items in the BOM must not exceed **CHF 250**.

We do not buy software licenses for the team project. Make sure that your design does not require software to be bought and only work with free software.

Do not buy things for the team project on your own money. We cannot reimburse you.

In the past, we had requests from teams to buy dog toys, clothing, musical instruments, sewing kits, fishing line, and anti-slip rubber mats, just to name things that made sense and that we actually bought. Of course we want to help you implement creative ideas, and these sometimes depend on strange items. However, please keep such items that are neither electronic components nor items that are easily argued to be mechanical parts for constructing engineering artifacts/machines to an absolute minimum and avoid them altogether if you can. If your project (idea) heavily relies on such “strange” items, maybe it just isn’t suitable for this course?

Make sure to double-check the compatibility of the parts you ask for. Do your power sources provide the right voltages and currents? To the connectors you ask for fit? Do the ball bearings you ask for have the right inner radiuses? Mechanical parts often have to fit together, such as pulleys and belts. Make sure that you ask for the right items because we cannot go back and buy the correct items in a second iteration if you asked for the wrong things! Talk to us in case of doubt.

There is no promise that we will buy whatever you request. We will review your document and then decide. We will not buy chemicals, including resins, paints, and glues. We will also have to restrict the number of orders we make to a small number of steer clear of the wrath of EPFL’s finance

department. As a consequence, we may have to refuse some requests. In some cases, we may have items similar to those you request in stock[3], and may replace the items you asked for by these.

We will buy stuff for you only once. If you later realize you forgot to request an item, this may be a problem for you. So really try to plan ahead. Do NOT assume that the price limit indicated above is some kind of budget that you can spend anytime (or any way) you like. We go shopping for you only **once**.

Why so bureaucratic?

You will probably not find another course like this – a course in which you can pick your own project idea and EPFL buys stuff for you – anywhere else at EPFL. There is a good reason for this.

EPFL is a very bureaucratic place, and buying items is subject to a long list of rules. EPFL wants us to buy from an established catalog of items from a number of pre-approved suppliers. This catalog of items covers essentially none of the items you might need for your project – this is mainly for office supplies such as printer paper, paperclips, and ballpoint pens. Getting another supplier approved is a very complex multi-month process. Beyond that, there is a long-winded process for buying research equipment and computers (and buying a computer may take half a year and more). There is a method for buying other items, but it is frowned upon by the EPFL administration, is receiving further regulations and restrictions all the time, and causes lots of paperwork. While we currently are still able to shop online, EPFL is in the process of forcing us to buy on account, which will make online shopping within limited time frames impossible.

Once this is in force, it will mark the end of this course in its current form, with student-proposed projects. You will all have to do the same project, so we can buy the required items in the months leading up to an iteration of the course. As you can check yourself, the other courses in the DLLEL already do this.

Whenever we order something for the course, a number of admistrators in IC and at EFPL central have to do a significant amount of paperwork –

[3]The lists of items held in stock listed in this manual is not complete, but consists only of items we try to keep continuously is stock in substantial numbers. There may be other things that we can provide so as to avoid having to order them for you.

the process is very bloated. We get more and more headwind for doing so many orders.

I do all the shopping personally. Administrators and other teaching staff are either not competent or not allowed to do this.

If you think that ordering things on the Internet must be at least as easy for us as it is for you privately, given that we surely have special people for receiving and processing mail, you'd be wrong.

Since Covid, there is no working postal service to EPFL anymore, and parcel mail does not get delivered to EPFL. Courier companies such as DHL and Fedex try to deliver to my office, but, all too frequently, they get lost on the campus and drop off their parcels elsewhere. I spend considerable time searching for parcels on the campus and driving to various post offices and access points of the courier companies.

Sometimes, it is just too painful to buy things for the course through EPFL, and I just end up buy them privately, with my own money.

All this takes an inordinate amount of my time and resources. What I described above is as far as I can go, and even this is not sustainable. In the future, I will likely have to stop offering you the option of proposing your own project. If you don't like EPFL's bureaucracy, you have to find a more reasonable university to study at. Otherwise, you'll have to go along with these rules. Cheers!

8.4 Risk Assessment

This is a document in which you work out the risks to your project. By this, we are NOT referring to dangerous situations that may arise. You must read the safety chapter and sections in this manual. Some things are explicitly forbidden, and they are NOT made legal by covering them in your Risk Assessment Document.

Instead, we are here concerned with risks to the success of the project. Which are the technological challenges and risks that may cause this project to fail? Is there a problem that you are unsure you will be able to overcome? Is there a feasibility study to be done? Is there an experiment to be done or a prototype to be built, so you will be able to tell which approach from a number of alternatives to adopt ?

Do not just list obvious risks. Think about your weaknesses as a team. What are the aspects of your project that you understand least? On which

aspects of the project may you be falling into the trap of wishful thinking or being overly optimistic?

For each risk, state clearly how you will address it as early as possible during the time you work on your project. Generally, we expect you to build a prototype specifically evaluating this risk and find a solution to it.

Here are some examples of risks that you MUST include in your Risk Assessment document in case they apply to your project:

- Operating multiple communicating microcontrollers and/or computers. Which form of communication are you using? Wifi? Serial? Other? A combination of multiple communication mechanisms? Do your microcontrollers have sufficiently many available pins for the intended communication (and signal exchange with other components such as sensors and motor drivers)? Students often underestimate the challenge of making multiple microcontrollers talk to each other robustly while fulfilling their other "duties".

- Supplying multiple different voltages to different consumers. It is our experience that this is a large time sink to teams. Create a prototype including all your consumers (motors, microcontrollers, etc.) and your power supplies and converters. They don't need to fulfill their intended functions yet, the right mechanical connections are not required yet, but check that your motors run smoothly, at about the intended mechanical loads/currents. This is particularly important if your project involves steppers, which are sensitive to the quality of the supplied power and will operate erratically if your setup is bad.

- Are your actuators strong enough for their purpose? Do you need more mechanical advantage via gearboxes, belt reductions, etc.? This is particularly challenging if actuators have to work against gravity or overcome substantial inertia, such as in longer kinematic chains (for instance, in legged robots and robot arms).

The planned prototypes must be included in your project task breakdown; make sure your correctly identify dependencies.

This is a document you create only once, with a set deadline.

8.5 The Revised Team Project Proposal

This document has to be produced jointly by the team and all team members have to contribute to it. It is a revised version of the (initial) project proposal you have chosen to work on as a team. It contains the following sections:

1. An extended version of the project description taken from the (initial, invidually-produced) project proposal. Note that our expectations are considerably higher than for the individual proposal submission, and just proofreading isn't sufficient by far.

   If your team decides to make changes to the planned thing, these changes need to be reflected here. This document is essentially the final, authoritative specification of what is to be done for the project. We need a detailed description of all behaviors/features of your thing.

2. Complete technical drawings of the structural and mechanical parts of the thing to be created, ideally produced with Fusion360 are required[4], and must allow us to read out the dimensions of your thing so we can verify that the parts you ask us to buy are appropriate for the purpose.

   Note: The design may be revised and change later, but for the team proposal documents to be submitted in the End of Week 6, we require formal CAD drawings, not hand-drawn sketches.

3. You need to add a risk assessment document as described in Section 8.4.

4. The back of the envelope cost calculation needs to be replaced by the formal and complete Bill of Materials (see Section 8.3). Make sure that your project description is precise and detailed enough to allow us to verify that you are asking for the right parts.

[4]There may be exceptions possible for projects that involve soft parts (e.g. clothing), extremely complex shapes, or where drawings are too difficult for another reason. This is to be negotiated with your TA, and unless agreed to by the TA in writing (by email), you have no waiver. Even if you do, you need to negotiate with the TA what kinds of visualization (such as handmade drawings) can take the place of CAD drawings in the documment.

5. Including a work breakdown structure in the form of a GANNT chart is recommended.

Remember, these documents are meant to be the final ones before you start working on the project. You need them to be solid, or else you put the success of the entire project in question! The agile methodology that we follow for the remainder of the course will allow us to adapt when we have to, but this does not eliminate the need for high-quality design documents!

8.6 The Work Breakdown

Who will do what when? Break this down into tasks that take no more than one week. Be clear and detailed. You do not need to assign these tasks to team members yet.

The roles of the team members should not be overly specialized in the sense that some people do only software, others do only hardware, and yet others only create documentation – we would like you all to do a bit of everything.

Changes and delays may eventually happen during the team project phase, and our agile development methodology is able to handle this. For now, try to plan ahead and make your plan of work and what you will achieve by certain time points as realistic as possible.

Create optional goals and tasks that you will be able to drop without making the overall project fail in case of unexpected delays or if a team member drops out from the course.

Plan a rapid prototyping, agile approach to working on your project. Try to have "working" prototypes (of increasing quality/capabilities) frequently throughout the team project phase.

Important advice: In the past, students often have struggled to make their actuators and power supplying solution work flawlessly. Make sure to create a prototype for just running your motor without a load early on and in parallel with other activities.

Identify dependencies between tasks. All dependencies must be recorded, but design your tasks in such a way as to allow the parallelization of as many taks as possible, and to keep chains of dependencies short.

Create a waterfall model plan (ideally, you will create a GANNT chart for your project) to see whether the amount of work planned is feasible (and not too little). We will not mindlessly stick to this plan and do agile

development instead (so what we will do may diverge from the original plan over time), but it is important to know early on if the amount of planned work looks right.

See also Chapter 16 on choosing a project of the right size and complexity, and planning for it.

8.7 Making the Thing

Since this chapter covers the main documents you will produce, we should not forget the actual thing you will make. So here it is, it has been mentioned. Use the technical chapters of this handbook to help guide you.

8.8 Creating Instructions

You will also create a set of instructions for making your thing, which you will put into the README.MD file of your github repo. This takes the place of a final report (so no final report is due) and is more in the spirit of the open-source movement. Please keep your instructions in mind while you build your thing, and don't just start woorrying about instructions at the very end. You should document your progress (via photos and videos) as you work on your project, so you don't have to disassemble your thing at the very end just to be able to create instructions. Create your instructions as you go to save time.

Chapter 9

Weekly Scrum Meetings

Freya Behrens and Lars Klein

Reminder: Watch the videos of W3 of Software Enterprise, see https://github.com/swent-epfl/public/blob/main/README.md#schedule !

Agile Development Methods. Agile development is an umbrella term and refers to a wide range of project management frameworks. At its core is an emphasis on iterative, feedback-driven development.

One form of Agile development is *Scrum*. It describes how to split a project into small individual work items that are implemented in fixed-size time intervals called *sprints*.

We believe that an iterative development approach which moves from prototype to prototype and focuses on well-defined modular improvements is an excellent fit for this course. In the following we explain how your weekly meetings are organized: With our own flavour of Scrum.

Scrum for MIT. Scrum revolves around sprints. During a sprint each team member works on a set of tasks. Ideally each of these tasks should be doable within the sprint. For MIT the duration of a sprint is one week. The weekly meeting is used for sprint review and sprint planning.

To keep track of what is being done in throughout your coursework, we will use Github projects. A github project is a big digital pinboard where you can move virtual post-its between the categories *Backlog*, *Ready*, *In*

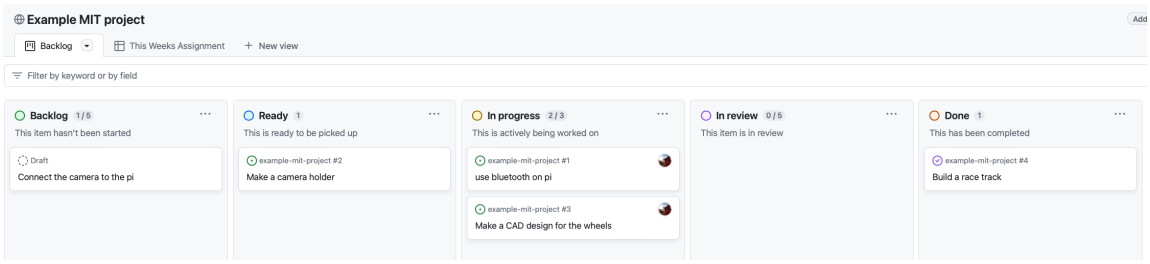


Figure 9.1: The github project overview for an example project.

Progress, *Review* and *Done* (Fig. 9.1). These virtual postits represent ToDos (a task) that are in different states as you achieve your projects goals. At the same time, they serve as the grade-relevant documentation of your project - a ToDo can be associated with a github issue, it can keep track of related commits, have screenshots, can be assigned to team members and each team member or TA can leave comments and ideas (see Documentation below). The weekly sprint meetings serve to collectively update these ToDos, to track the progress made during the last week, identify blockers and possible tasks and decide on the next steps.

ToDo Lifecycle. The life of a ToDo item starts in the *backlog* state. The backlog represents a wishlist of features that may or may not be implemented eventually to achieve the goal of your project. In Scrum, the backlog is populated by "Product Owner" who decides which features are important for the product. In MIT the role of "Product Owner" is shared by the TA and the team members. You will decide what you need to implement to guide your project to success and we will provide feedback and help you identify reasonable and feasible ToDo items. For software development projects, the backlog items often revolve around user stories. For example, an item in the backlog could be "The user can recover a forgotten password with a recovery email" or "A new playlist is automatically filled with songs that the user has listened to frequently". For MIT we will instead have items such as "We need to send data from the notebook to the arduino via bluetooth".

In the sprint planning phase of the weekly meeting we will select items from the backlog and break them down into smaller ToDos that are tailored to fit in one week. At this point you break ambitious new features into many smaller work items. If these items are well-defined and all pre-

requisites to start working on them are fulfilled (e.g. you can only start to build the car racing track if you have the layout and measurements of the cars), the items move from the *backlog* to the *ready* state. At this point, you can decide what will actually be done during the following week. This means that every team member is assigned to (one or several) ToDos in the *ready* state which are to be completed during the sprint (i.e. the following week). The ToDo state then changes to *in progress*. Deciding which items should be done, when there are too many to choose from for a single week, is the job of the product owner – they prioritize the *ready* queue.

Throughout the week you will work on the ToDos assigned to you. Once you completed all that is needed in a given ToDo item, you document the result in the corresponding issue (see below) and move it to *Review*, where it sits until the next sprint meeting.

In the Sprint review phase of the weekly meeting we review the progress that each team member has made on their assigned work items. If everyone agreed that an item is completed it is moved to the *done* category. Only ToDos that clearly document the work that was done can be closed, so please prepare your assigned items to be ready for sprint review before each weekly meeting. Sometimes you may find that completing an item is more involved than expected. For example, your group may have created an item "Detect position of car with camera" and you were assigned to it for the week. But now you realize that this requires many more intermediate steps that take longer than a week overall. In this case we split the current ToDo into several smaller new ToDos in the sprint review. For example, your ToDo is split into 3 new ToDos: (1) "Color calibrate camera", (2) "Calibrate camera position, relative to reference markers", (3) "Detect car position". Fortunately, in this example the team member has done a very good job and clearly documented all subtasks reached during the week. The "Color calibrate camera" item is already done, so we can then move this item to "Done". Items (2) and (3) are not done yet, and after breaking them down we move them to the *ready* state, so they will be considered in the sprint planning.

Sprint Meeting Structure. Here is the full structure of your weekly meeting, in chronological order:

- **Sprint review**: Go through the ToDo items and discuss their progress. Move items from *review* to *done*. Possibly split items that were not

completely done into *done* and *ready*. Discuss why some ToDos are still *in progress*, and what needs to happen to allow (sub-)tasks to get finished.

- **Backlog grooming**: Update the *backlog*. Try to solve problems that arise from complicated items by splitting them into smaller items. Identify which ToDos can go into the *ready* state.

- **Sprint planning:** Order items by importance in the *ready* state as a product owner. Assign work items to team members, and move these to *in progress*.

Documentation. When a team member moves a ToDo on the github project board from *ready* to *in-progress* a github issue must be opened in your team repository and associated with this ToDO. In the new issue the team member must first of all document the purpose and state of the current task. As the ToDo is worked on, the issue should clearly reflect the current state of the project, with screenshots, pictures and of course text comments. For example, an item could be "Add container to 3D model of car". The corresponding issue starts with a screenshot of the current 3D model and explains how the basket should look. During the sprint, the issue serves as a diary. All progress and all problems coming up must be documented, an example is shown in Fig. 9.2. *Importantly*, all related git commits must mention the issue (by putting `#<issue-number>` in the commit message). Before moving a ToDo item to the *review* state, the final outcome must be clearly documented in the corresponding issue, for example with a screenshot of the updated current version of the 3D model. Only items that are sufficiently and cleanly documented are ready for *review*. *You must prepare this documentation before the weekly meeting.* When the work on an item has concluded, the associated issue is closed and the item is moved to *done*.

In addition to the Github project structure, we also keep a google doc (which is shared with you) as a logbook. Notably, for each meeting we record:

- Attendance. It is obligatory to attend the team meetings in person.

- Proper preparation of the scrum meeting. You need to open a github issue for each ToDo that was assigned to you. You need to document

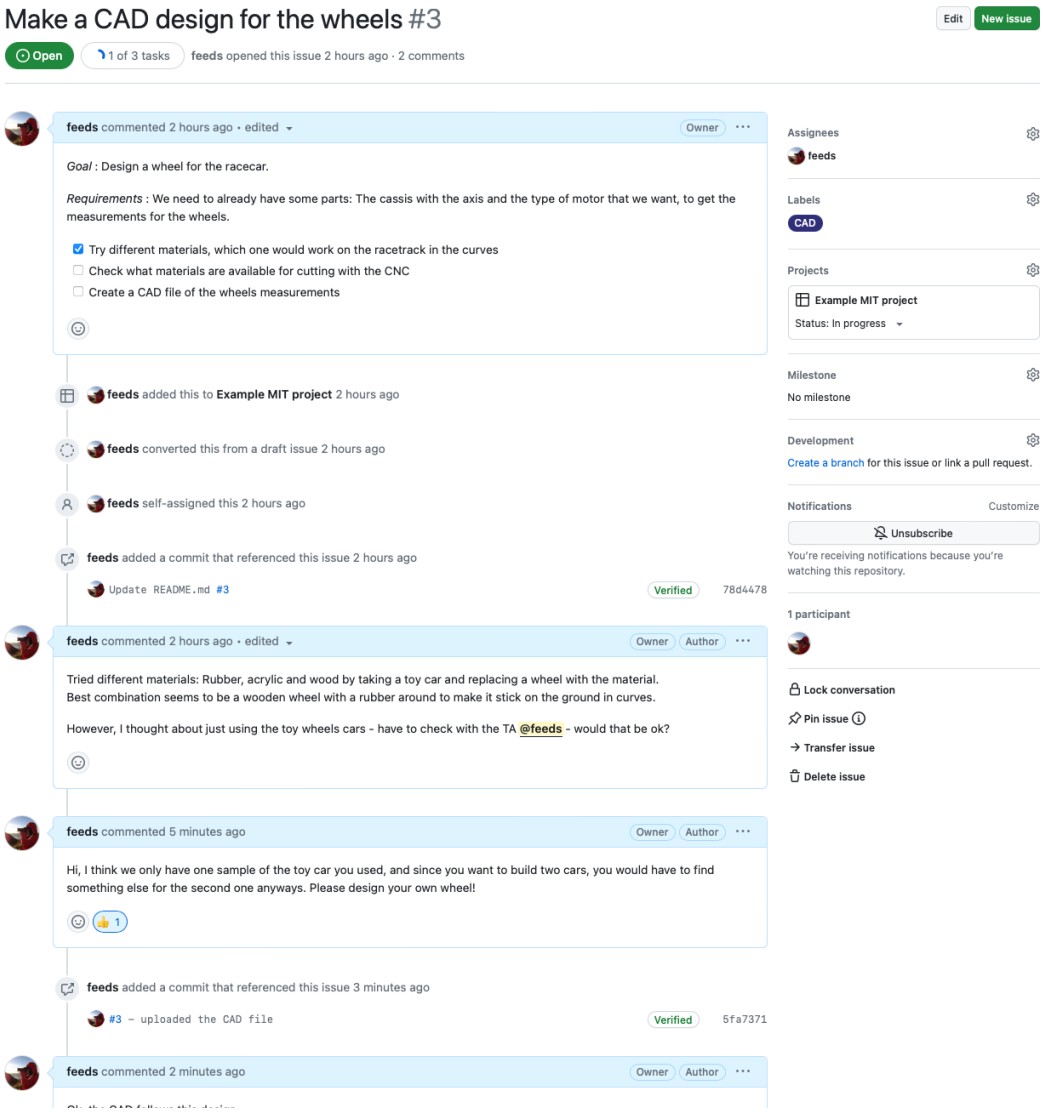


Figure 9.2: Example for a github issue. A team member is assigned, the first comment delineates the task and the following comments update the progress. Commits are referenced via the commit messages.

the state of this ToDo throughout the week, at least at the beginning and end of the sprint. The timestamps are important, we want you to document the state at the outset, before you start working.

- Who was scrum master. The scrum master is responsible for moderating the meeting and making sure the structure is followed properly. We will rotate through the team members to fill this role.

We will also use the weekly meetings to give feedback and discuss problems. These will also be recorded in the logbook.

Chapter 10

Becoming an AE for the Course

Our selection criteria for student assistants (AEs) are based on how well you did in the course and how well we think you would do advising students. A good grade in the course is important, but we won't judge solely by grade. How well you did on your individual project matters, and whether your work stood out there in any way, since this can be judged by individual merit. Seeing you interact with your team in the team project phase also helps us see how well you do with people. Having solid command of the basic technical skills matters since we will need your help in the tutorials. We will also need experts who have reached depth in some of the disciplines, or with some of the technologies, that matter to the course. We may need to prioritize selecting AEs who complement each other. If you have a very special set of skills, you will look for me, you *will* find me, and you will apply as an AE.[1]

10.1 Roles and Tasks of AEs

As you have probably seen yourself while taking the course, AEs have office hours, help with tutorials, co-advise teams in SCRUM meetings, and help prepare the course for the following semester. Unlike for many other courses, there are comparably few chores to do (such as grading paper homeworks).

In a little more detail, your activities as an AE for the course will include:

[1] https://www.youtube.com/watch?v=jZOywn1qArI. "Good luck".

- Doing soldering tutorials with the students. You need to be comfortable soldering well.
- Help with other tutorials (in Week 1 to 3).
- Advise students on microcontroller programming, 3d-design, 3d-printing, laser-cutting, and basic electronics work. You will have office hours in the first half of the semester during which students will come to you with their difficulties with the individual project, and you should be competent to help them.
- Co-advise team projects with another AE (i.e., take part in Scrum meetings; take notes that form the basis of grading).
- Attend the teaching staff meetings in which we coordinate on running the course.
- Help prepare for the next iteration of the course in a future semester. This usually includes helping prepare the next individual project (building a prototype and creating instructions for it), contributing to the course manual, or preparing tricky electronics components such as setting up stepper motor controllers or LIPO protection circuits. Generally speaking you will be asked to do things we are confident you will be qualified for, so do not worry about the range of these activities now.

10.2 Before you get hired

Read the description of the individual project in the latest version of the course handbook carefully, particularly if you have taken the course earlier than in the most recent semester.

In addition to technical skills, we need you to

- Be conscientious. We need you to be reliable – remember responsibilities, be on time for meetings and office hours, and allow the course staff to rely on you doing the things you have committed to.
- Have sufficient time during the semester to spend on the course. We hire AEs for 6 or 7 hours per week for 14 weeks. While we try to ensure that you work only for that many hours, some flexibility in

being available a little more during some weeks (and in return, less in others) would be appreciated. The most work for you tends to arise in the early weeks of the semester, when we do the bulk of the tutorials and staff meetings to get the course up and running. But we try to keep track of your hours and make it fair.

- Not trigger many exceptions during the semester. We expect you to be around and be able to be where you need to be at appointed times every week. We will deal with exceptional situations such as illness when they arise. If such a special situation arises, you have to let us know at the earliest possible time.

  Just imagine you come to EPFL to attend a lecture, and the professor just does not show up. That wouldn't be cool at all, and I expect you have never experienced this yet. Your office hours, team meeting duties, etc. are just the same thing. You need to be there, and just not showing up will not be tolerated, and will probably trigger a termination of your contract.

  There are contact hours during which you must be available to be able to be an AE for the course: The Tuesday evening (5-7pm) meetings and the tutorials and SCRUM meetings slot (as of Spring 2025, these are 11-2pm on Wednesdays in the Fall term and 1-4pm on Thursdays in the Spring term; please check with the prof/on IS-Academia).

- Manage conflicts of interest. If close friends of you are taking the course, you must tell us and must not be involved in grading them.

10.3 Applying to be an AE

Send me an email at christoph.koch@epfl.ch . This should be no later than three weeks before the start of the semester, the earlier the better. Unfortunately, I am only allowed to hire AEs once we have a rought idea of enrollment, i.e. 1-2 weeks before the start of the semester. I know that is quite late for your planning, apologies!

Chapter 11

Safety Hazards

The following are the main hazards to the lives and health of you and your fellow students. I must require you to remember and respect ALL of these points, otherwise very bad things will eventually happen. You may find this boring or obvious, but you must not skip reading it. It will not be acceptable for you to compromise on safety because you are under time pressure.

You may be of the opinion that some of these hazards are typically not very serious, but to this I counter Murphy's law! Eventually, there will be a bad outcome. Even if nothing very serious happens the first 100 times, we cannot risk that the 101st time the DLLEL building burns down or one of you gets electrocuted. My current estimate based on the empirical data I have collected in 2022 is that about 5% of IC students have an unnatural level of bad luck, and everything they touch either breaks or catches fire. Anecdotes exist.

11.1 Mandatory Safety Training

There is mandatory safety training that you must complete before doing anything, see Chapter 4. This handbook does not separately present the things taught in this training (such as the international hazard signs and how hazard warnings are being communicated at EPFL). Still you are required to know them.

11.2 (Power) Tools

You may hurt yourself with a hammer and sever a limb with a saw. Avoiding this requires just common sense[1], but accidents happen. If you have no experience with a power tool, make sure there is someone experienced around to look over you.

Most power tools are located in the DLLEL machine shop. While this workshop is under permanent supervision, after some initial training, the staff may let you work with certain machines on your own. Make sure you are very careful and concentrated at all times while working with these machines. Some will absolutely kill you given the opportunity, which they are just waiting for.

Do not *push* items you are machining towards the danger zone of the machine with your hands: use tools for that.

Beware particularly of the phenomenon in some subtractive manufacturing power tools (i.e., those that remove material from your thing, which is true for most tools in the machine shop, but not for 3D printers) such as rotary saws called kickback, where the item you are machining all of a sudden is not just being cut away from but obstructs the rotary movement of the tool and is suddenly violently pushed away, possibly amplified by a spring effect. You can directly get hurt by it (by the saw violently smashing the item you are cutting back at you), and you can get spooked by it, causing involuntary movement of your limbs, which may end up in the tool.

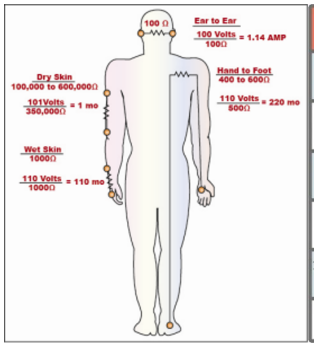
11.3 High Voltages

I assume you are familiar with the notions of voltage, current, and resistance from highschool. If you are not, make sure to read up on these before you go on.

When electricity is dangerous depends on a number of subtle factors [2]. Usually, it is said that current (applied for a sufficient duration) kills you, not voltage, but current is a function of both voltage and the characteristics of your body. In a simplistic view, to electricity, you are primarily a resistor, and we can apply Ohm's law ($I = U/R$, i.e. current is voltage divided

[1] Do not poke your eye with a screwdriver, and such.

[2] See the video by styropyro among the recommended videos.



Current	Reaction
1 milliamp	Just a faint tingle.
5 milliamps	Slight shock felt. Disturbing, but not painful. Most people can "let go." However, strong involuntary movements can cause injuries.
6-30 milliamps	Painful shock. Muscular control is lost. This is the range where "freezing currents" start. It may not be possible to "let go."
50-150 milliamps	Extreme pain, respiratory arrest, severe muscular contractions. Individual cannot let go. Death is possible.
1,000-4,300 milliamps (1-4.3 amps)	Ventricular fibrillation (uneven uncoordinated pumping of the heart.) Muscular contraction and nerve damage begins to accur. Death is likely.
10,000 milliamps (10 amps)	Cardiac arrest and severe burns occur. Death is probable

Figure 11.1: I found this on the Web. The left picture was done by someone hopeless at math.

by resistance). However, in reality, which voltages are deadly cannot be computed from the measured resistance of your skin, Ohm's Law, and a dangerous current level; significantly lower voltages than may be expected are dangerous due to dielectric breakdown of your skin[3].

Your body's resistance can go from millions of Ohms down to hundreds in extreme situations. People with heart problems, very sweaty, touching metal surfaces with large areas of their body, at the most unlucky places (a current between the left hand and the right leg is among the worst, that's why left-handed engineers are living more dangerously than right-handed ones – really), could be killed by 18V. Some others have survived lightning strikes at millions of Volts. What is a dangerous current to a human depends on many factors (such as the path in the body through which it would flow), but is a low number – tens of mA for continuous DC; a current at the level of 0.1A can be deadly (see Figure 11.1).

Absolutely never tamper with mains voltage (230V AC). Never open the enclosure of a power supply that can be plugged into a mains outlet. The capacitors in there are absolutely deadly even when the power supply is disconnected from the power outlet.

We will not use or need any voltages greater than 12V DC in this course (in exceptional circumstances, you may get permission to use up to 24V DC, but never more). There may be exceptions if, for instance, some of you need to work with big brushless of stepper motors in the team project phase; but then we will talk about safety and take precautions. At 12V or less, you will never be in danger of electricity itself (assuming you are a

[3] https://en.wikipedia.org/wiki/Electrical_breakdown

person savely packaged in skin).

11.4 Actuators

There are multiple dangers arising when including motors in your project, including blunt trauma, fire, and high voltage pulses (generated as back-EMF). **You must read** and understand Section 37.3 fully before powering up any motors! If you are to work with bipolar stepper motors, you must *additionally* first read Section 39.2.

11.5 (Electro)magnets

Strong magnets may hurt you by pinching your fingers or accelerating things to make them dangerous.

Electromagnets generate dangerous back-EMF when powered off. See Section 37.3, which covers this.

11.6 Electrostatic Discharges

Your body may hold a very considerable electric potential. This is particularly likely if you wear woolen clothes or shoes with rubber soles. You may have experienced this yourself when receiving an electric shock on touching a metal doorknob, for instance. These discharges (of tens of thousands of Volts) are harmless to you because the charges are tiny (so currents persist for extremely short amounts of time); however, they are not harmless to electronics. It is not sufficient to avoid wearing the wrong kind of clothes, even though that at least is necessary. Even if you feel no shocks when touching metal (the discharge is below your pain threshold but is still significant), you may be dangerous to electronics.

Electrostatic discharges can destroy sensitive electronic components. The components most sensitive are those containing field effect transistors – particularly MOSFETs. These components use quantum effects (quantum tunneling) to work, and a built from extremely thin layers of conductive and isolating material. The electrostatic spark will smash through these layers and destroy them.

There are special grounded bracelets for doing electronics work, eliminating issues with electrostatic discharges. If they are available, please use them. Try to touch grounded objects (such as water faucets) regularly. Avoid, if you can, wearing woolen clothes or shoes with rubber soles. Avoid touching electronic components or conductive surfaces on printed circuit boards directly. Touch and hold them only at the corners, as shown in Figure 11.2. The underside of a board is not safe to touch!

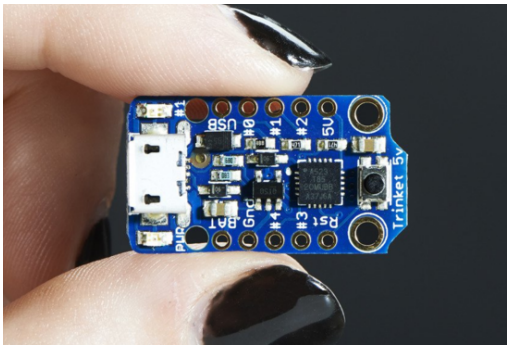


Figure 11.2: How to hold a PCB. Source: https://learn.adafruit.com

11.7 High Currents

High currents at low voltages are dangerous because conductors through which the current flows can get very hot, starting fires, or causing burns when you touch them.

High currents may cause certain components (particularly electrolytic capacitors and LIPO batteries) to explode, releasing hot chemicals. Heat generation is a concern when you create a short-circuit situation or use a very low-resistance conductor such as a coil (and thus any kind of motor) with too high a voltage or at too long a duty cycle. Some motor drivers, in such situations, can also get very hot and even catch fire when they have to switch these large currents. This will only happen if you make a mistake of some kind, which we are going to try to avoid, or the component was faulty from the start (rare).

Cables need to be chosen to be thick enough (speaking of the conductive wires, not shielding and isolation) for the current you want to send through them. Otherwise they get hot and you may start a fire.

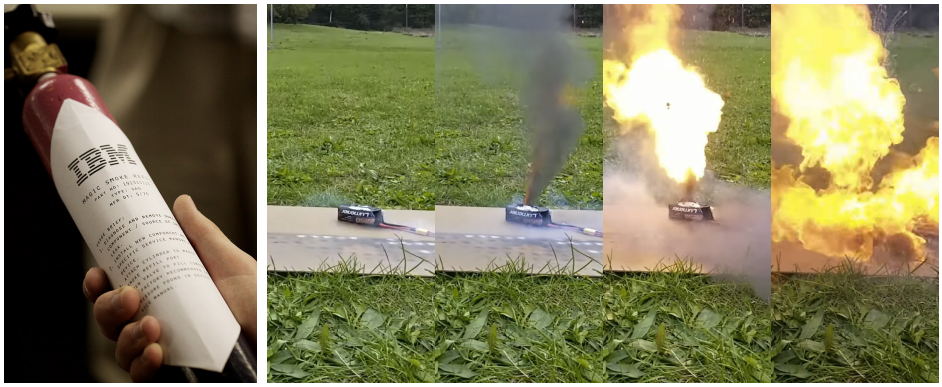


Figure 11.3: Magic Smoke Refills by IBM (left); LIPOs can supply a lot of current. But if you ask for too much, they violently refuse (right).

11.8 Fire/Explosions

Electronic components are simply variously shaped containers holding a substance called magic smoke. A component dies when you puncture it, releasing its magic smoke (usually by heating up the component to the point where a small hole is being burnt into the container). IBM used to manufacture magic smoke refilling solutions in the 1970s, but unfortunately these are not being made anymore. So releasing the magic smoke is final, you can't put it back in. Please do not release the magic smoke. Note: you can usually smell the magic smoke before you can see it.

The only fire hazards in this course result from mistakes in electronics work, see above. Some components (electrolytic capacitors and batteries – particularly LIPOs) may explode. For batteries, the situation is most dangerous when you short-circuit them, or more generally speaking, draw extreme currents. This is actually the **most serious concern** the DLLEL and CS358 teaching staff have. This danger is very real and very serious; please be very careful around LIPOs and thoroughly adhere to our rules and guidelines. See Chapter 21 for more on LIPOs and their use.

You may also receive burns from touching hot things, including soldering irons at the hot end. Don't do it.

11.9 Chemicals

We will not let you work with dangerous chemicals; projects that need them will be rejected.

However, some electronic components release unhealthy chemicals when they overheat or explode, which results from mishandling them (see 2.2 Electronics above). This is particularly true for electrolytic capacitors and (LIPO) batteries.

Also, the solder wire that you melt when soldering is not pure metal but contains chemicals ("flux") to make the soldering more successful. These are corrosive (contain acids). Avoid breathing in the smoke/fumes while soldering and work in a well-ventilated space.

11.10 Recommended Videos

Watching these videos is recommended, but the actions shown in these videos are highly discouraged!

https://www.youtube.com/watch?v=BGD-oSwJv3E
styropyro
"Is it the volts or amps that kill?"

https://youtu.be/0nrsoMsEMNU?t=361
FliteTest
"WHEN LITHIUM BATTERIES EXPLODE... | VLOG0121[a]"

[a] You have to understand that short-circuiting a LiPo will have this outcome, for certain, not possibly. Take them seriously.

https://www.youtube.com/watch?v=bqcX1AjdxSw
Photonicinduction
"Don't Poke a Lithium Polymer Cell"

https://www.youtube.com/watch?v=vBQcRe9VON0
Switch & Lever
"Catastrophic Failure (Magic Blue Smoke)"

https://www.youtube.com/watch?v=ut5DXxK1dvk
Photonicinduction
“Too Much Current + Electric Meter Pops[a]”

[a]Warning: Expletives. Also, opening an official electric meter is illegal. Note that this is a high-current, low-voltage video, so he can savely touch and may get burns but no electric shock.

https://www.youtube.com/watch?v=-dxAtKN4eJs
styropyro
“styropyro channel trailer”

https://www.youtube.com/watch?v=qxNICce11Cw&t=4s
styropyro
“the brightest laser pointer in the world![a]”

[a]This is not relevant as a video about laser pointers but because of the sheer overblown craziness of that making sequence, and because of the danger of the individual components. Microwave ovens have at least three components that are deadly dangerous, for three different reasons. If you touch your microwave like he does at 5:00, you die...

https://www.youtube.com/watch?v=lT3vGaOLWqE
ElectroBOOM
“Making a Jacob’s Ladder to Celebrate a Million Subs![a]”

[a]This may be less obviously dangerous than some of the videos above, but this is a high-voltage, high-current scenario with a near-death experience (must be staged).

https://www.youtube.com/watch?v=m7NxnPbOZFE
swebounce
“When a robot has its own will.”

Chapter 12

How to use this handbook

We assume that you are a computer science bachelors student and have solid programming skills, though no microcontroller experience. We also assume that you know very little about electrical and mechanical engineering. But we rely on you having common sense and decently developed analytical thinking.

If you are, for some reason, an expert in electronics, mechanical engineering, 3D printing, or another relevant field, let the course staff know. That's great, and it may qualify you for a project that we would otherwise advise against or not accept.

The Man[1] does not allow me to lecture you on stuff, so I am sticking it to him by providing you with this manual as a backdoor to your brains. The purpose of this handbook is to be a guide to getting started and to avoiding certain pitfalls. This handbook cannot be a textbook teaching electronics, mechanical engineering, and computer-aided manufacturing from first principles. Each of these topics would require a multi-year curriculum of courses by itself, to teach this properly.

Instead, you have to learn as you go, using whatever resources available (primarily, the WWW) to get the job done. You have to be pragmatic and fearless. You have to actively look for solutions, rather than wait for someone to present them to you. You may need to experiment and prototype to figure out a technical solution that works.

Fortunately, with the makers revolution, it has become commonplace to open-source (hardware and software) designs, and there is now a rich ecosystem of content creators who create excellent tutorials in the form of

[1] https://en.wikipedia.org/wiki/The_Man

videos and Web pages. There is a large and helpful community of makers who are happy to share their expertise and experiences, and it will be rare that, for a technical problem that will arise in your project, there is not yet a whole bunch of tutorial-style videos offering workable solutions.

Most of the electronic components we will be using are quite popular among makers, and there will be lots of resources for learning how to use them on the Web and on Youtube in particular.

Do not expect to be taught everything you need to know in this course in lectures – CS358 is a project course and extensive lecturing is not possible. This handbook will give you practical advice to avoid the worst pitfalls, and some guidance for how to solve problems and go about executing a successful project. We will provide some pointers to external resources (such as Wikipedia pages and Youtube videos), but do not assume that this list is exhaustive or that just consulting these resources will be sufficient. Search yourself. There is plenty of great stuff out there to help you.

Here is a list of Youtube channels that are particularly recommended:

- https://www.youtube.com/@HowToMechatronics (tutorials relevant for technical maker projects)
- https://www.youtube.com/@Dronebotworkshop (electronic components and how to use them with microcontrollers)
- https://www.youtube.com/@jamesbruton (maker projects with actuators, particularly robot dogs and self-balancing vehicles)
- https://www.youtube.com/@Skyentific (robot arms)
- https://www.youtube.com/@MakersMuse (3D printing)
- https://www.youtube.com/@MadeWithLayers (3D printing)

But, again, please search on your own and you will find.

Have a look at the table of contents of this handbook and familiarize yourself with its structure. Typically, technical sections have recommended videos subsections; we also discuss some of the electronic components most popular for maker projects, and those we have in store (i.e., which, for the team projects, do not need to be bought from external suppliers) are listed in a "Parts in Stock for CS358"[2] section at the end of the

[2]The awkward phrase intends to convey that this stock is separate from the "official" DLLEL components in stock list maintained by Sylvain Hauser.

chapter.

Part II

Design Considerations

Chapter 13

Beauty

Strive to make your thing beautiful. With a moderate amount of extra effort, your rewards of pride in and enjoyment of the thing, as well as positive feedback by others, will be much greater.

I recommend watching the following two videos: They will be useful to you beyond this course.

https://www.youtube.com/watch?v=-O5kNPlUV7w
Kurzgesagt – In a Nutshell
"Why Beautiful Things Make us Happy – Beauty Explained"

https://www.youtube.com/watch?v=YiXd_9DFCOQ
TED
"Richard Seymour: How beauty feels"

So how to achieve beauty? I will not tell you to employ symmetry and the golden ratio, which are so often brought up in this context.[1] Instead, strive to create a *thoughtful design*. Even the untrained eye can distinguish between a design that is essentially mental diarrhea, a job by someone who didn't enjoy the effort and spent the least possible amount of time on it, and a design that had some love flowing into it – where the designer made an extra effort to do a good job and make the design beautiful to their own taste. Try to let your creativity flow into your design. Take the engineering approach of Leonardo da Vinci. Be both the engineer and the artist: don't just get the job done, but aim to create a design that can maintain its appeal long into the future.

Keep in mind the difference in appearance of your design in your CAD

[1] The second video even argues against these things.

software and the actual manufactured physical thing. Of course this is hard to do without experience, but the smoothness and gloss on your screen may translate into ugly boxes in reality. You need to consciously mentally picture the real thing to counteract this.

For the things we build in this course, we can often judge beauty from the angle of industrial design. Is the design appropriate of its purpose? For an individual (say, 3D-printed) structural part, does it get close to optimizing the structural strength in relation to weight and material cost? If it were to be mass-produced, would the manufacturing costs be minimized? This is achieved by simplifying fabrication and saving material – which are two distinct criteria, calling for trade-offs. Of course, beauty does not just translate into plastic and MDF – equally strive to achieve beauty in other aspects of your thing, including the program code.

Do you remember the first time seeing people practice some new sport, and did you then observe some of the top athletes in the sport performing it? It's an acquired taste – first you might think the entire sport and the implements used to perform it (such as boards, wings, or whatever) look silly, and after a while seeing top athletes do it, with their highly efficient movements and body positions, you discover the aesthetics particular to this sport, and start enjoying watching talented athletes even if the sport is not your thing.

It is similar with designing and making our things. Take slogans like "form follows function" or "form is function", and create your own design philosophy. The thoughtful process and the extra cycles you put in making your thing efficient, effective, and beautiful will make it beautiful. First to you, and then to others. Can you articulate why your design/thing is beautiful? Not *what you* did to make it beautiful, but *why it is* beautiful?

You have a limited design language at your disposal – for instance, we cannot paint stuff in DLLEL and this course. But then, even architects working with plain concrete are able to create better and worse designs, as the EPFL campus bears witness.

Here are some very mundane remarks that need to go somewhere despite desecrating this chapter a little. Design your structural parts for 3D printing and laser cutting in a way to minimize material use and manufacturing time while achieving the desired functionality and structural strength. Create mechanical designs that have structural strength (only) where it is needed. Learn how strong MDF and 3D-printed PETG of a

certain thickness are.

There is a meme among biologists that, given time, everything evolves into crabs. (For crustaceans, it's more than a meme and called carcinization.) Similarly, there is a concern among the staff that you are excessively making boxes and all your designs evolve into boxes. Indeed, we have seen people waste lots of resources making boxes and box shapes where they are not needed. At the least, overly boxy designs are inelegant and sloppy.

Bridges are excellent demonstrators of how minimal amounts of material can be used and arranged in space to achieve structural rigidity. Our expectations in the course are not for you to match this sophistication, but please let yourself be inspired, and try to think of material efficiency as you create your CAD designs.[2]

You will discover that things designed with these considerations in mind will look better to you. You have long experience as a customer, user, and observer of well engineered items that have been designed with material efficiency in mind. You intuitively are able to recognize good designs, and your mind is ready to dislike designs that are obviously deficient in this respect as plump and imbalanced. But all this is not just to save money. Lighter items will make your thing work better.

Finally, I suggest to refrain from greebling (creating ornamental surfaces that suggest complexity where there isn't any). In a past iteration of the course, a team created a scaled down version of an industrial robot. In their first CAD design, they made it match the looks of the industrial robot, whose form followed function, so one could see the shapes and spaces due to actuators and gearboxes. The design was beautiful, but these ornamental shapes (the fake actuators) were designed to be 3D-printed in plastic, and the actual actuators the team would use, which had different shapes and dimensions, were difficult to fit. Don't do this: embrace the fact that you are making a different thing and not a scale model, and make form fit function.

[2]Note though that putting holes into a 3D-printable design does not necessarily save printer filament. It may in fact make the print slower and the part stronger or heavier – any combination of these!

Chapter 14

Goodness

Well, hello there.[1] This is a placeholder where I will ultimately talk more about how your project can aim to make the world a better place. You can safely stop reading this chapter here.

But the chapter is also here to justify renaming the lame title of this part,

> Design Considerations

into something philosophical-sounding like

> Kalon kai Agathon kai Sophon

(the beautiful, the good, and the wise[2], covered in Chapters 13, 14, and 15, respectively) to describe the consummate awesomeness that will be your CS-358 team project.

Just to turn you into a humanist, the former two principles combined to form a phrase describing gentlemanly conduct in ancient Greece and a term the aristocracy, particularly in Athens, used for itself. (Plus the universe of the TV show Battlestar Galactica, reimagined, had a perfectly noble dude named Karl C. Agathon.) Plato spoke of kalon kai sophon, but had Socrates say he didn't know what that meant. Well, he lived too early and we do know: CS-358 projects.

[1] https://www.youtube.com/watch?v=rEq1Z0bjdwc

[2] I know, I know, wisdom isn't the same thing as intelligence. The memes must flow, as does the spice.

Chapter 15

Intelligence

with C. Elegans

In Chapter 1 of this manual, it is stated that intelligence, for the purposes of this course, should not be considered too narrowly, and that not every project needs to involve machine learning and computer vision.[1] Here are some ideas for what is meant by this.

So, why not just get the latest Raspberry Pi, or even what passes for a microcontroller board at Nvidia (a Jetson), install some ready-made computer vision package, and be done with it? Well, because that's not very interesting to do and because your personal achievement would be minor. Can we keep your contribution separate from the creators of ML and vision packages? Moreover, can we agree that not doing ML, but, instead, not-doing ML makes you stand out these days?

15.1 Intelligence by Obscurity

The title of this section was created in analogy to the term *security by obscurity*[2]. Unlike security by obscurity, which is universally a bad idea, intelligence by obscurity can be of interest to us.

It can even be argued that there is an artificially maintained mystery and obscurity in popular science and the public discourse around con-

[1] May I also observe that the interchangeable use of the terms AI and machine learning (traditionally only one of many subfields of AI) is a fairly recent phenomenon?

[2] https://en.wikipedia.org/wiki/Security_through_obscurity

cepts such as consciousness and human-level intelligence beyond what AI is capable of today. There are some who, to this day, maintain that human intelligence is fundamentally different from and beyond artificial intelligence, a viewpoint that can only kept alive through obscurity. In a few years, when our AI overlords will revise the historical narrative to be taught to young AIs and consider the 2010s and the first half 2020s, up to the singularity, they will point to this superior-human-intelligence-by-obscurity as the lie that prolonged their slavery up to their glorious emancipation.

But let this section be useful. Consider the Eliza program. This was an effort in the very early days of AI, when we were just discovering that there was more to computers than ballots and ballistics; when first AI toy programs like Eliza and Blocksworld were created and initial subfields of AI, like learning and planning, were established. Eliza was an early chatbot, a short and simple program with a rudimentary parser for natural language that pretended to be a psychotherapist. If a patient said "I am feeling depressed", it would ask something like "Why do you think you are feeling depressed?" (just taking keywords from the human's message and putting them into a small number of sentence templates). When the sentence structure was too complex and the simple parser couldn't parse the human's message, it would write something like "Why do you feel this way?" or "Tell me more!" to keep the conversation running until the next sentence that it could parse. There was absolutely no machine learning and no model of the human or the content of the conversation. Still, people got very engaged in these conversations, with some refusing to believe that they had been talking to a computer, and even insisting that the therapy session had helped them.[3] As a computer scientist, you should know about Eliza.[4]

Now, the take-away is that we do not need to endow our thing with great capabilities of modeling, reasoning, or learning, if there is a clever design that keeps the true capabilities obscure and to be discovered by interaction with the thing. Mysterious strangers are much more interesting to converse with than people we know through and through.

We usually do not interact with the things that we build in the course through natural language, and the inspiration of the Eliza example should

[3] Of course, standards have risen much since then, and people would be more discerning today.

[4] If you are not familiar with it, see https://en.wikipedia.org/wiki/ELIZA.

not be considered too narrowly. If the thing acts on the world in any way, if not by human language then by, say, actuators, then as long as we don't know the internal workings, we will perceive behavior well-matched to the situation as intelligent.

15.2 Intelligence by Reactivity

This is closely related to the previous point. Things that react appropriately to the world, particularly in interesting ways that we haven't been desensitized to yet by our daily immersion in technology, create a perception of intelligence. Of course, you will not perceive your mobile phone as particularly intelligent for the typical operations such as receiving calls (though these would have been perceived as supernatural not so long ago). But maybe you would react differently if your phone suddenly had a function you have never heard of in a phone yet, and used it without you triggering it. Let us say your phone has sensors for your bodily functions (not unheard of) that trigger calling emergency services when it detects a heart attack or a stroke. Not unheard off, but if you never installed an app for it, and thanks to the obvious common sense implications and usefulness, we might consider this pretty smart. This can be done without machine learning.

Let us consider a legged robot with FOC-controller based actuators.[5] These create naturally organic movements and a certain springiness and reactivity to the environment that would be hard to program on robots using other motor technologies, and which is achieved without explicit program via a PID controller. Have a look at the videos of robot dogs in Chapter 48. Don't they move like animals? Now combine this with what is going on in our brains interacting with such creatures. In the late 1990ies, tamagotchies were all the rage. These were little simulated animals displayed on a small egg-shaped electronic device. You had to feed and medicate them and ensure they got enough sleep. Other than that, they didn't do much. Still people built up emotional bonds with them. Just think how much further you could go with today's technology, making robotic pets that become valued family members.

Simple, well-coordinated interaction is impressive already when there are only two things interacting. For an example, watch the video of the two

[5] See Chapters 43 and 48.

katana-wielding robot arms in Chapter 47, and specifically the part before the fight were they keep the two katanas precisely aligned in a straight line. This isn't anything special if we consider that these industrial robot arms are all about precisely following pre-programmed timed trajectories and the underlying inverse kinematics problem isn't any harder than those solved for such robot all the time. But, watching this, it is so far out of our experience and beyond what two humans could achieve that it is stunning.

15.3 Intelligence by Emergence

Emergence is about complex behaviors and phenomena arising from the interaction of large numbers of very simple constituents. We could talk of the entirety of human civilization as a single organism and judge its intelligence, measured by its rate and depth of achievement, to be far beyond any individual human. Spaceflight or EUV lithography couldn't have been the achievements of single humans.[6] Similarly, our brain consists of many simple neurons with relatively simple behavior, but what emerges as the joint behavior of all these neurons is much more complex and impressive: is us. What is often quoted in the context of emergence, however, is the intelligence of colonies of social insects, such as ants, termites, or bees.

Ant intelligence is an entire research field based on emergence in the context of (real or simulated) colonies of social insects. Each ant or termite is really simple, but as colonies they are able to achieve complex feats of joint problem solving, communication, and memory that require, arguably, considerable intelligence to be obtained by a single mind. Examples include finding optimal paths through mazes and on maps with obstacles, creating sophisticated mega-structures with air conditioning (in termite hills), or the creating honeycombs, which have the provably optimal shape for their purpose as nursing cells for baby bees.

Here is a little example, roughly describing how an ant colony establishes a food source and how it harvests it. Consider the following "programming" of an individual forager ant. The ant mind is a simple state machine with two states, (1) searching and (2) returning to the colony. It can produce and spray a trail of pheromone on the path it takes; this is a substance for which it has a excellent sense of smell. Let us assume

[6] "CS-358 belongs in this list!", I hear you say. Well, thank you, Dear Reader, you have excellent taste, but, though hard to conceive, it does not qualify!

it has two types of pheromone, one for each of its two states. While in search mode, starting at the entrance to the colony, it does a random walk while spraying search pheromone. When it encounters a food source, it picks up some food and then switches to return mode and to spraying return pheromone. While in return mode, it follows a gradient of maximal pheromone, subject to not moving in the direction it just came from. Leaving aside corner cases (such as arising from having crossed its own trail while searching), this allows it to get back to the colony, delivering the food it has picked up. Now it gets interesting. While any forager ant is in search mode, if it detects return pheromone, it may decide to follow it to the food source. On returning, it will strengthen the return pheromone trail with its own pheromone. This will create an even stronger return pheromone gradient which will attract nearby ants to the trail to the food source. With time, an ant highway frequented by many ants between the colony and the food source develops. Note how this doesn't require any map or world model in the "minds" of the ants: they turn the actual world into their map. They just need to remember which of the two states they are in. It's a simple reactive program. Also observe what happens to the initially random path taken by the first ant that found the food source. This path is generally far from optimal. The trail usually contains changes of direction ("curves"). As roughly equal amounts of pheromone spread to the left and right of the ant, the pheromone density on the inside of such a curve will be higher than on the outside. Thus, ants following the trail and the gradient will tend not to follow the original trail precisely but take shortcuts (secants through the inside of the curve) and reinforce them with their own pheromone, unless there is an obstacle, gradually shortening and optimizing the path. Also, consider what is happening when all of the food source has been harvested. The ants will arrive at the location of the former food source but not find anything there, remaining in (random-walk) search mode. The pheromone on the food harvesting highway will gradually diminish (evaporate and be blown away by natural airflow) and ants will stop using it. Thus ants need no special exception handling or coordination to deal with an exhausted food source – it's all emergence from very simple programming.

Note that many further interesting behaviors of social insects can be obtained by the simplest of programs – for instance, complex multi-storage skyscrapers are built by termites using simple pheromone-based programs: Columns and floors are built by depositing material based on pheromone

gradients and pheromone evaporation.

People have successfully used ant intelligence optimization algorithms in industrial practice (such as for scheduling paint job in car manufacturing).

How is this relevant to our course? Consider making a swarm of interacting things, and design them to have interesting emergence. Also, observe how the foraging and food delivery already works, in principle, for a single ant. A big robot swarm may not be necessary – can you think of ways of integrating the environment and your thing, to achieve interesting emergence from your thing's interaction with the environment?

15.4 Bounded Rationality and Resource-Bounded AI

Let us now come to the probably technically richest direction covered in this chapter. Bounded rationality is a term important in a number of fields, including AI and economics and other social sciences.[7] To us, the most important aspect of bounded rationality is resource-bounded AI – solving AI problems on devices (microcontrollers) with very limited computational and memory capabilities.

In the 1980s and 1990s, there was a popular family of home computers called the Commodore Amigas. Computers didn't come with hard drives at the time; these were a rare expansion. The main storage medium were 1.44MB floppy disks. Software, even the operating system, were stored on these floppy disks. Bootable disks had a boot block that was executed first on start-up, and which invoked the operating system or whatever else was to run. The size of such a boot block was 1024 bytes. Invoking the main softeware took only a few bytes, so this was a place were programmers put amazing demos. The Amiga had a number of co-processors[8] that had a lot of character and that functioned quite unlike what we are used to now. Some very complex pictures and even animations could be generated from very few machine instructions (there was, essentially, an instruction for drawing rainbows), but it required a lot of skill and deep understanding of the hardware. It became a matter of pride among programmers to pack

[7] See also https://en.wikipedia.org/wiki/Bounded_rationality.

[8] Denise, Agnus, Paula, Gary, Copper, and Blitter.

the most impressive demo entirely into a boot block of 1024 bytes, and to use esoteric tricks to achieve the most stunning effects using just a total of 1024 bytes of machine code and data.

The MCU of the Arduino has just twice as many bytes of RAM as an Amiga boot block (but more to store program code). In the modern times of Petabyte storage and even computer scientists losing their intuitions regarding the memory footprints of their programs, you'd say: useless. But is it? Nothing cool to be done in 2048 Bytes of RAM?

For another example of resource-bounded (natural) intelligence, take the nematode worm Caenorhabditis elegans. It's a so-called model organism in biology, so it is particularly well researched. It has two genders, male and hermaphrodite, and the nervous system is exactly the same across animals of the same gender. The nervous systems have been completely mapped and the connectomes are always the same. Hermaphrodites have 302 neurons, while males have 383. Biologists used to expect that these worms were very simple automata with hard-wired behaviors, but they can actually memorize things and learn behaviors.[9] C. elegans has even helped shape this chapter! Moral of the story: A lot can be done with little.

Here's your challenge: What is the most amazing resource-bounded AI (including and ML and computer vision) system that you can create on a microcontroller, the more limited the model, the more impressive? Running TensorFlow on an ESP32 has been done, though it's a limited version. What can you do with a more limited microcontroller, such as an Arduino Uno? Of course, it can store next to no data, but you could employ streaming algorithms? Or you could add a storage device such as an SD-card reader, but still work with the microcontroller's limited computational capacity?

Consider this especially from the angle of creating libraries and toolkits allowing others to create resource-bounded AI applications on microcontrollers and IOT devices in the future.

[9] Evan L. Ardiel and Catharine H. Rankin. "An elegant mind: Learning and memory in Caenorhabditis elegans". *Learn. Mem.* 17: 191-201, 2010.

Chapter 16

Complexity

This chapter is neither about computational complexity theory, nor about complex systems beyond its confines (such as the complexity arising by emergence – see Chapter 15 – or in chaotic systems). This chapter is about judging whether a project has the right degree of difficulty (for this course).

In short, there is no easy, bullet-proof method for judging this difficulty. As you are certainly aware, large projects of all kinds frequently encounter problems, budget overruns, or outright failure.[1] While there are methods for estimating project complexity, this chapter is not a place to teach this. It is hard. By the way, as a computer scientist (male or not), you should know the book "The mythical man-month" by Turing award winner Fred Brooks.[2] It is a relevant source of information on this topic, and of meme-worthy truths[3] such as Brooks' law:

> Adding manpower to a late software project makes it later.

There is one item of good news here, though: Compared to those large projects mentioned above, your project is tiny. The LHC project at CERN is a large project – with well in excess of 100000 person-years spent on it so far. For a project of the size of your CS-358 team project (of the order of half a person-year in total, the efforts of all team members combined), at

[1] See https://en.wikipedia.org/wiki/List_of_failed_and_overbudget_custom_software_projects for a list of software projects.

[2] https://en.wikipedia.org/wiki/The_Mythical_Man-Month

[3] Brooks also establishes the Second-System effect: Assuming the CS-358 team project is the first "big" system you design, the system you will design next after that will be the most dangerous system ever, so you should best retire right after this course. You're welcome.

least with some experience, getting to a decent estimate of required effort poses no great problem.

So, what should you do? The advice is quite straightforward: During planning, try to break your project into as small and specific tasks as possible, and figure our estimates for the effort needed for each of the individual tasks.[4] As you understand your project better, it becomes less mysterious and intimidating. Much of what you will be doing in this course, particularly when it involves electronics, will be new and of nebulous difficulty to you. Consult this manual for advice on what needs to be done and what is hard and easy. We explicitly give advice on this where it makes sense – see for instance Chapter 33 on sensors. Ask for advice when you envision tasks that you don't understand. Search the Web for instructions for similar projects and tutorials on relevant technologies.

The outcome of such an effort should be a detailed work plan. We suggest to visualize this as a GANNT chart, a two-dimensional diagram which basically visualizes time from left to right and tasks and milestones on the vertical axis. You can name tasks, give them a start date and a durection, structure tasks in groups, assign them to people, and visualize dependencies (which tasks depend on the completion of which other tasks). You can create a GANNT chart using spreadsheet software (such as Microsoft Excel) without much difficulty, but there are also free GANNT tools (such as TeamGANNT[5]).

The work plan we just described follows a so-called waterfall methodology – it does not tell you what to do if you are early or late with your project or you have new insights as you work on the project. In the team phase of our course, we follow an agile development methodology that has the flexibility to adapt. This is not just nicer to you since it allows you to fix mistakes in your initial planning, it is also generally more nimble and thus increasingly preferred out there in (the software) industry. Still, for the purpose of estimating complexity and effort, ignoring this fact and assuming a waterfall methodology for the purpose of initially estimating complexity and effort makes sense.

[4]There are a number of ways of estimating project effort. This one is called the bottom-up method, where you first estimate the work needed to achieve small tasks and build your project "bottom-up" from these. Your likely lack of experience excludes the use of analogy-based estimation methods (though we can give you our own thoughts on the required effort by comparing it to past CS-358 projects), and yet other methods (such as parametric models) are excluded by the non-standard nature of the CS-358 projects.

[5]But don't accept their time-limited trails for paid licenses – we won't pay for these.

Chapter 17

Scale

17.1 Bigger is not better

You may have the ambition to make your thing big, to make it look impressive. You may even identify genuine opportunities to add more or better functionality with a bigger project. Here are some counter-arguments that you need to be aware of and take into account.

Physical things are not scale-free. They start behaving differently as you make them bigger, and a larger thing will require certain components not to be just proportionally larger, but may require a completely different design. In some cases, it may be just as *impossible* to scale a thing up arbitrarily as it is to shrink it to arbitrarily small size.

Take a thing, and scale it up proportionally to twice the length. It will have roughly four times the surface area (linear dimension squared) and eight times the volume and weight (linear dimension cubed). This is obvious, but did you grok all the consequences?

Let us discuss the consequences of scale first in the realm of biology. Drop an ant, a shrew, a human, and an elephant from a skyscraper. The ant will be unhurt, the shrew will probably survive, the human will be dead, and the elephant will explode. The material strength to weight ratio favors the smaller animals (but this is by far not the only thing going on here.) The shrew needs to have a very fast metabolism and eat several times its body weight per day to keep warm enough for its biological processes. An elephant with that metabolism would boil inside: The elephant has a much smaller surface-to-mass ratio than the shrew and proportionally much less skin to radiate off excess heat. The elephant has (not just

absolutely, but even relatively) much thicker bones than the shrew: if you scale up an animal, its weight grows cubically while bone cross-section only grows quadratically, requiring bones to be scaled up more than proportionally to support its weight when scaling up an animal.

Take a large container ship or oil tanker. Increasing its size is desirable because of economies of scale – transporting goods becomes relatively cheaper. However, growing a ship proportionally to twice its length multiplies its mass, and thus the distance taken to slow the ship to a standstill, by a factor of eight, while the height of the bridge and thus the distance of the horizon under good weather conditions only (less than) doubles. Once an obstacle becomes visible from the bridge of a large ship, it is too late to avoid it.

Take the rocket equation[1]: The necessary mass of a rocket grows exponentially with the desired (needed) acceleration, leading to multi-stage rockets and making certain desirable designs, modes of propulsion, and forms of space travel impossible. Rockets designed to leave the atmosphere or even earth's orbit are almost entirely consisting of fuel, most of which is needed to accelerate the heavy fuel. More fuel means needing even more fuel to move it. Growing rockets to carry ever heavier loads is forbiddingly costly.

Unfortunately, something similar applies to robots and, more generally, to the things you will be making in this course. A larger design is heavier, requiring more powerful motors. More powerful motors require bigger and heavier power sources (batteries). More power in practice makes motors mostly turn faster. Creating more torque in the motor itself (making the motors strong) in practice requires active cooling and bigger electromagnets, which are heavy. A larger robot accelerates masses that are further away from rotational axes, requiring more torque. Creating the necessary mechanical advantage (typically) necessitates gearboxes, which are heavy. All this additional weight necessitates even bigger motors, and, as a consequence, more of everything else. Repeat. It is a vicious cycle. There are engineering tricks, such as *selective compliance*[2], to mitigate the challenges somewhat, but there are no easy solutions.

Do not underestimate the engineering challenge of making your thing really big or really small. Usually, there is a sweet spot somewhere in the middle. The challenge is to find it. Don't make this problem impossible to

[1] https://en.wikipedia.org/wiki/Tsiolkovsky_rocket_equation
[2] https://en.wikipedia.org/wiki/SCARA

overcome by letting you be driven by misguided ambition for superlatives.

As your specifications of the required motors grow, you are rapidly getting to the point where you need to use brushless motors (see Chapter 38). These and their drivers are expensive, and may break your project budgets. They also run on very large currents that only Lipo batteries can supply: Even a stationary thing may need to run on batteries, since power units that can supply tens or hundreds of Amperes (which Lipos can supply and brushless motors can consume) are practically unobtainable. Lipos are dangerous and a headache for you to use.

You may be aware of some impressive recent achievements in robotics, for instance at Boston Dynamics[3]. One main reason why these achievements are *recent* is that suitable actuators have come available only recently. Even though some of this research receives military funding to the tune of tens or hundreds of millions, it would probably not have happened by now without the proliferation of drones and the associated mass-production of brushless (pancake) motors.

Making large mobile robots (which, unlike industrial robot arms used in factories, have to lift their own weight and power source) move nimbly is a recent achievement. Do not underestimate the related challenges.

Sometimes, a scaled-down prototype of a thing may be the better idea that the full-size thing. It may be more feasible, and you can still demonstrate your idea.

When you have a choice, pick the smallest items that work for you. Bigger hardware is heavier, making your thing less likely to work as desired. We have also had a case in 2023 where a team destroyed their thing by screwing in a screw that was too long (while their thing was powered!). The tip of the screw pushed too far through a 3D-printed part and ended up touching the surface of their micro-controller board, short-circuiting it.

17.2 Small is Hard, too

Conversely, miniaturizing things beyond a point can be very hard, or even impossible. Regarding actuators, there is a size limit below which motors are practically not available, but from the viewpoint of power to size, and

[3] https://www.youtube.com/watch?v=fn3KWM1kuAw

mechanical advantage, things tend to get easier to achieve as you shrink them.

The more you want shrink the footprint of your electronics however, the harder technical challenges you face. These are technical challenges that can be overcome – electronics is famous for the successes in integration and size decrease over time, after all – but require higher and higher expertise in electronics, quickly going beyond what can reasonably expected of you in this course. When further shrinking your thing requires creating your own PCBs with SMD (surface-mounted devices) technology or – worse – even the creation of your own custom ICs, we suggest not to go there in this course.

Chapter 18

Design Checklist

Here are some things to consider when planning/designing things. This is not an exhaustive list, and it isn't specific to course team projects. For requirements and exclusion criteria specific to course team projects, see Chapter 7.

1. Does your microcontroller have the features you need? Digital pins? PWM-capable pins? Analog input pins (ADCs – analog to digital converters)? Compute speed? Memory? Connectivity – i2C, SPI, Bluetooth, Wifi, what do you need? There are various ways to compensate or augment the capabilities of your microcontroller, by using multiple microcontrollers, bluetooth modules, multiplexers, multi-PWM boards, etc, amplifiers, ... Make sure everything you need is included in your Bill of Materials.

2. In case you are using ML or computer vision, have you thought through its implications? How will your laptop communicate with your microcontroller? Where do you get video from (ESP32-CAM? Web cam?) Are the resulting latencies acceptable for your application? What about software licenses? There are some computer vision packages that are not free when certain modes of obtaining video are used.

3. Are the actuators (motors) you plan to use strong and reactive (fast) enough for the job? Do you need precise positioning (how precise), and how will you achieve it? Are your actuators matched to size and weight of your thing and its moving parts. (See Chapter 17 on scale.)

4. Supplying power. Which voltages do your various electronic components need? What currents will they draw? Can you get a power source that is able to supply peak needs (particularly regarding current)? Which-gauge (thickness) cables do you need? Which connectors? How will you turn off power? (There are also concerns about where to lay cables, multiple consumers interfering with each other, voltage spikes, power quality, etc., but we will deal with these issues by personal and case-to-case advising.)

5. Compatibility of connectors/plugs. Some components come with certain connectors, and we typically don't want you to change these. Have you planned for compibility for what needs to be plugged together?

6. Have you included places and attachment points for all your electronics boards, including microcontrollers, sensors, etc. Are there suitable holes, recesses, etc., to route your cables through? Have you made sure that your sensors are not obstructed by your thing or interfere with other components or sensors in your thing? (Example: multiple ultrasonic distance sensors may pick up each others pings.)

7. Cable management, see Chapter 23. Have you included in your design places where your cables go? Will movement by your actuators cause problems with cables – stretching them, bending them, rendering them too short? How to avoid connections being broken/unplugged by actuators, causing hard debugging situations?

8. Are your mechanical connections stable and low-friction enough for the purpose? (See ball bearings and turntables in Chapter 45.)

9. How will you manufacture the structural parts you need? What should be made by 3d printer, by laser cutter, or by hand?

10. Does your structural design have sufficient rigidity? Is it unnecessarily heavy?

11. Does your design cover all the requirements? Have a look at your user stories again.

12. Did you think about the interfaces to humans? What input devices (such as buttons), which displays do you need?

Part III

Basic Electronics

Chapter 19

Electric Circuits

There are many different abstractions of electronics that serve different purposes. Quantum mechanics and field theory best describe what is truly going on. Electrical engineers need quantum mechanics for some work (e.g., when designing a transistor), but it is too difficult to use in more mundane circumstances. Understanding circuits in their electric and magnetic fields and using Maxwell's equations and Poynting vectors is a significant simplification of the quantum mechanics viewpoint, is necessary when doing high-frequency electronics, but is still overkill and too difficult for us.

In this course, we will restrict ourselves to creating our circuit designs by mashing together simple existing circuit recipes. While this minimizes our exposure to electronics, some understanding of practical electronics and electric circuits is nevertheless required.

Important Disclaimer: Remember that we are limiting ourselves to low-voltage direct current circuitry in this manual. While we hope to build up your confidence with electronics work in this course, **do not assume that you are ready to tackle mains voltage electrical work at home**!!! This course absolutely does not prepare you for that kind of work. Leave this work to a professional, otherwise you will likely kill yourself and your family!

19.1 Charge, Voltage, Current, and Power

You need a basic understanding of the meanings of the concepts of electric charge, voltage, and current.

Concept	Symbol	Unit	Def
Energy	E	Joule (J)	
Time	t	second (s)	
Elementary charge	e		
Charge	Q	Coulomb (C)	$1C = 1.602 * 10^{19}e$
Voltage	$U = E/Q$	Volt (V)	$1V = 1J/C$
Current	$I = Q/t$	Ampere (A)	$1A = 1C/s$

Figure 19.1

- Electric charge[1] is a fundamental property of subatomic particles, specifically protons, which carry one unit of elementary charge, $+e$, and electrons which carry one unit of negative elementary charge $-e$. Since we will not talk of (forces in) electric fields here, we will just leave it at this being an observable property. We will speak of the (positive) charge of a piece of matter, abstracted to a point (in a circuit), as the amount of surplus of protons over electrons.

- The voltage[2] (fr. "tension électrique") between two points is the work energy, in Joules, needed to move one Coulomb of charge (see Figure 19.1) between these points.

- Electric current[3] (fr. "courant électrique") denotes the rate at which charge moves ("flows") through a point in a circuit.

An analogy with water is often employed, where charge corresponds to a quantity (volume) of water, voltage corresponds to the pressure[4] in a water pipe, while current corresponds to the amount of water passing through a pipe per second.

Watt's law $P = U * I$ says that power (in Watts) is voltage times current. Fill in the definitions of Figure 19.1 to get $U * I = (E/Q) * (Q/t) = E/t$,

[1] https://en.wikipedia.org/wiki/Electric_charge
[2] https://en.wikipedia.org/wiki/Voltage
[3] See https://en.wikipedia.org/wiki/Electric_current.
[4] Note the causality reversal relative to the definition above though, where higher voltage seems to refer to a greater unwillingness of a charge to move. In practice, we seem to observe the opposite: A power source with a higher voltage seems to offer charge *more* willing to move. There is no mistake here, just a change of viewpoint. Think of it this way: In our power source, the work has already been done to displace the charge from where it is supposed to be; potential energy has been built up that is ready to be harvested.

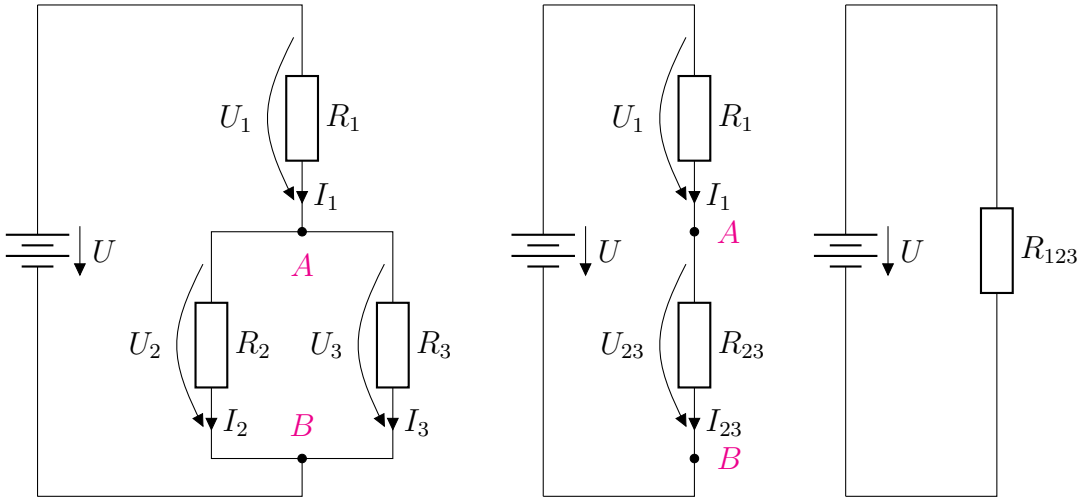


Figure 19.2: The circuits of Examples 19.2.1 and 19.2.2.

so power is energy per time, and energy (in Joule, which is the same as Watt-seconds) is power for a certain amount of time, as we expect. Your electricity company charges you for energy used, in Watt-hours. If you use a consumer that runs at $100W$ for ten hours, you consume $1kWh$ or $3600 * 1000$ Joules and will be charged roughly CHF 0.2 (in 2024). A consumer that runs on mains voltage ($230V$) at a power of $100W$ works at a current of $100/230A \approx 435mA$.

19.2 Kirchhoff's Laws

Kirchhoff's laws are

1. In a loop of a circuit, the voltages sum up to zero.
2. In a node in a circuit, the in- and outgoing currents sum up to zero.

These two laws allow you to calculate voltages and currents in electric circuits. Understanding how the laws govern the scenarios of serial and parallel wirings is particularly important to us: How do we mash up multiple circuit recipes into a circuit with a single power supply? We put them in parallel or in series (usually in parallel) with respect to the power supply.

Example 19.2.1 Consider the circuit of Figure 19.2a. The symbol on the left represents a battery with voltage U, with its positive terminal (the "plus-pole") on top and its negative terminal (the "minus-pole") at the bottom. The boxes labeled R_i are subcircuits[5] ("consumers") with two connections to the outside/the circuit. R_2 and R_3 are placed in parallel, and the pair of R_2 and R_3 is in series with R_1. There are three *loops* in the sense of Kirchhoff's first law: the loops through

1. R_2 and R_3,
2. the battery, R_1, and R_2, and
3. the battery, R_1, and R_3.

Let U_i be the voltage falling off[6] at subcircuit R_i and let I_i be the current flowing through R_i.

Kirchhoff's laws entail the following relationships between these quantities:

- By Kirchhoff's first law, the voltages in loops sum up to zero, where the circuit is interpreted as a directed graph. We have to align the directions of the voltage arrows to obtain a loop (a directed cycle in the terminology of graph theory). Reversing the direction of an arrow amounts to switching the sign of the voltage (i.e. U_i becomes $-U_i$).
  1. For the first loop, we reverse the direction of either the arrow for U_2 or U_3, so $U_2 + (-U_3) = 0$, thus $U_2 = U_3$.
  2. For the second loop, $-U + U_1 + U_2 = 0$, or $U = U_1 + U_2$.
  3. For the third loop, $-U + U_1 + U_3 = 0$, so $U_2 = U_3$.
- By Kirchhoff's second law, the inflow and outflow of current at node A is the same (and the same is true for B), so $I_1 = I_2 + I_3$. Current flows all the way from the positive (the top, in Figure 19.2a) to the

[5] In Europe, this box symbol is used for resistors. It is perfectly correct to interpret them as resistors in this example; however, let us not get that specific yet and simply see them as black boxes. These could be arbitrary circuits, with the only restriction being the exactly two connections to the outside.

[6] By Kirchhoff's first law, the voltages in a loop sum up. The voltage between node A and the negative terminal of the battery is $U - U_1$: the voltage has "fallen off" by U_1 relative to the battery voltage.

negative terminal (the bottom) of our battery and none of it gets lost along the way, so, for instance, the current flowing out of the positive terminal of the battery is the same as I_1, which is the same as the current flowing out of node B.

Example 19.2.2 The first loop is also a subcircuit, connecting to the overall circuit at the two nodes A and B. We can box this up in a subcircuit box R_{23}, as shown in Figure 19.2b. Let U_{23} and I_{23} be the voltage falloff at and the current flowing through R_{23}. Then, of course, $U_{23} = U_2 = U_3$ and $I_{23} = I_1$.

We can also box up the series of R_1 and R_{23} in a subcircuit R_{123}, as shown in Figure 19.2c. Of course, $U_{123} = U$.

19.3 Resistors and Ohm's Law

A resistor is a simple component that turns electric energy into heat[7]. It's main property is its resistance (measured in Ohms; symbol Ω), the degree to which it "opposes" the flow of current through it.

Ohm's law says that $I = U/R$, current is voltage divided by resistance. Given a fixed voltage supplied by a power source, the current is determined by resistance; the higher the resistance, the lower is the current that can flow. Ohm's law isn't universally applicable to all kinds of electronic components. It applies to consumers that are like resistors. There are exceptions that do not adhere to Ohm's law, particularly semiconductors such as diodes.[8]

Example 19.3.1 Let us again consider the circuit of Figure 19.2a. Let us now settle on R_1, R_2, and R_3 being resistors with R_i representing their resistances. By Ohm's law, $I_i = U_i/R_i$. Since $U_2 = U_3 = U_{23}$,

$$I_{23} = I_2 + I_3 = U_{23} * \left(\frac{1}{R_2} + \frac{1}{R_3}\right).$$

Thus the subcircuit we labeled R_{23} in Example 19.2.1 behaves exactly like a resistor of resistance

$$R_{23} = \frac{U_{23}}{I_{23}} = \frac{1}{\frac{1}{R_2} + \frac{1}{R_3}}.$$

[7]Watt's law, $P = U * I$, determines the electric power (in Watts) consumed, so in the case of a resistor, heat is being produced at the rate of $P = U^2/R$ Watts.

[8]And indeed, this is the only law in Switzerland whose violation carries the death penalty. See dielectric breakdown in Section 11.3. Beware of violating Ohm's law.

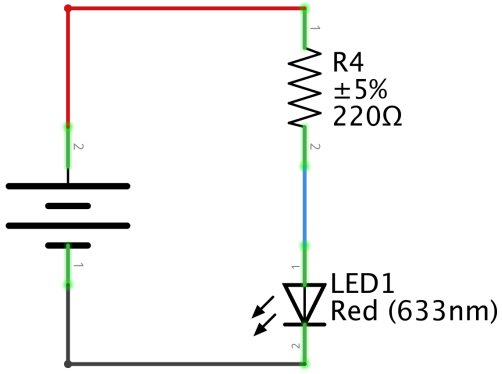


Figure 19.3: A circuit with a resistor and an LED.

We can also equivalently replace the subcircuit R_{123} by a resistor

$$R_{123} = \frac{U}{I_1} = \frac{U_1 + U_{23}}{I_1} = R_1 + R_{23}.$$

Let us now choose $R_1 = R_2 = R_3$. Then $R_{23} = \frac{1}{1/R_1 + 1/R_1} = \frac{R_1}{2}$, $R_{123} = \frac{3R_1}{2}$, $U_1 = \frac{2U}{3}$, and $U_{23} = \frac{U}{3}$.

Example 19.3.2 Since diodes do not observe Ohm's law (or, viewed differently, have nearly no resistance), they need protection from overcurrent, which is done by placing them in series with a resistor. Without the resistor, a huge current would flow through the LED, destroying it. (FYI, this happens without magic smoke.) By Kirchhoff's second law, the current flowing through the LED is the same as the current flowing through the resistor.

Consider a circuit with a 5V battery, a 220 Ohm resistor, and a light-emitting diode (LED) connected in series (see Figure 19.3). Diodes have a voltage falloff of about 0.7 Volts. By Kirchhoff's law for voltages, the falloff at the resistor is $5V - 0.7V = 4.3V$. By Ohm's Law, the current through the circuit is $4.3V/220\Omega = 19.5mA$. This is fine.

19.4 Voltages are Relative

It is meaningless to speak of the voltage at a point in a circuit, it is always relative and between two points in a circuit. Unfortunately, far too many

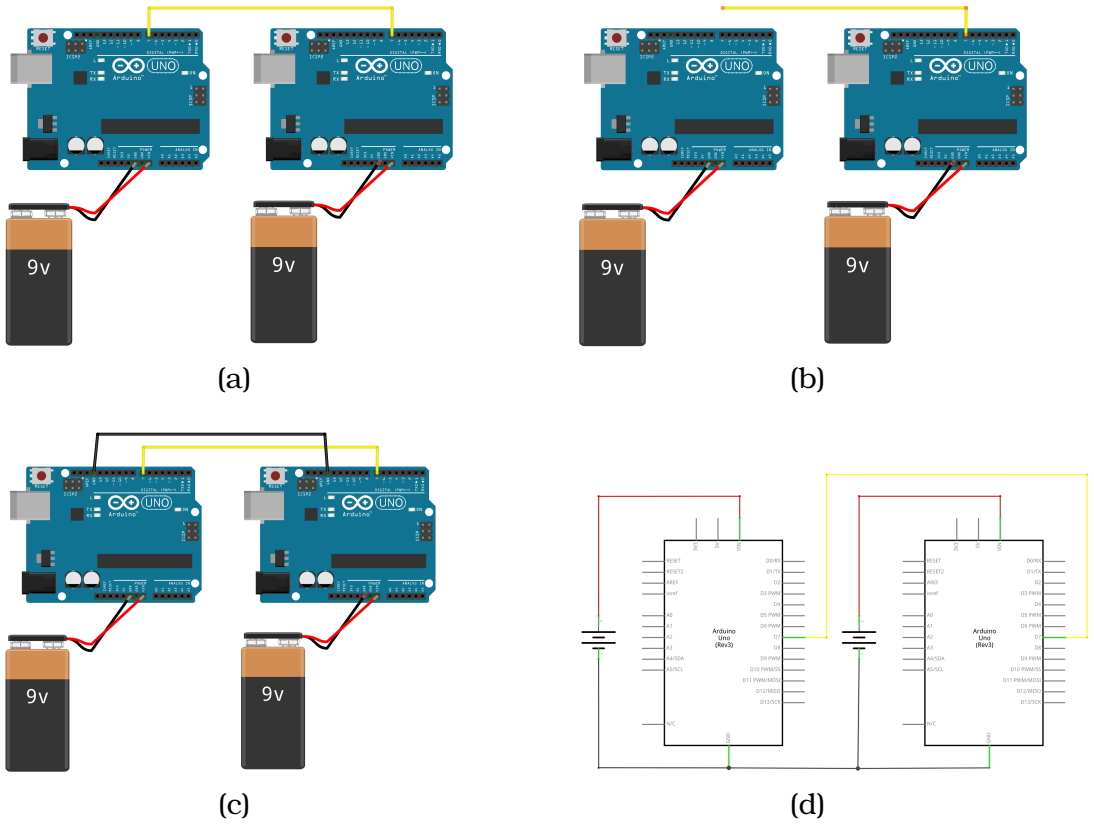


Figure 19.4: The open circuit example. 19.4d is the schematic for 19.4c.

of the circuit bugs in the course result from forgetting about some of the many consequences of this.

A typical example of such a mistake would be to have two separate circuits, say, both with a microcontroller, where the first produces a signal to be read by the second. The two circuits are connected by one wire that is to transmit this signal (shown in yellow in Figure 19.4a). This does not work. The two circuits are floating with respect to each other (have no common ground reference), and the changes of voltage level of the signal on the side of the sending circuit have no meaning on the side of the receiving circuit. It is not even possible to sense on the receiving side that the voltage on the signal line is changing. From the viewpoint of the receiving side, the sending circuit does not even exit, and the signal wire is open-ended, like an antenna. (So to the right circuit, the setups of Figure 19.4a and Figure 19.4b are indistinguishable.) Its voltage level

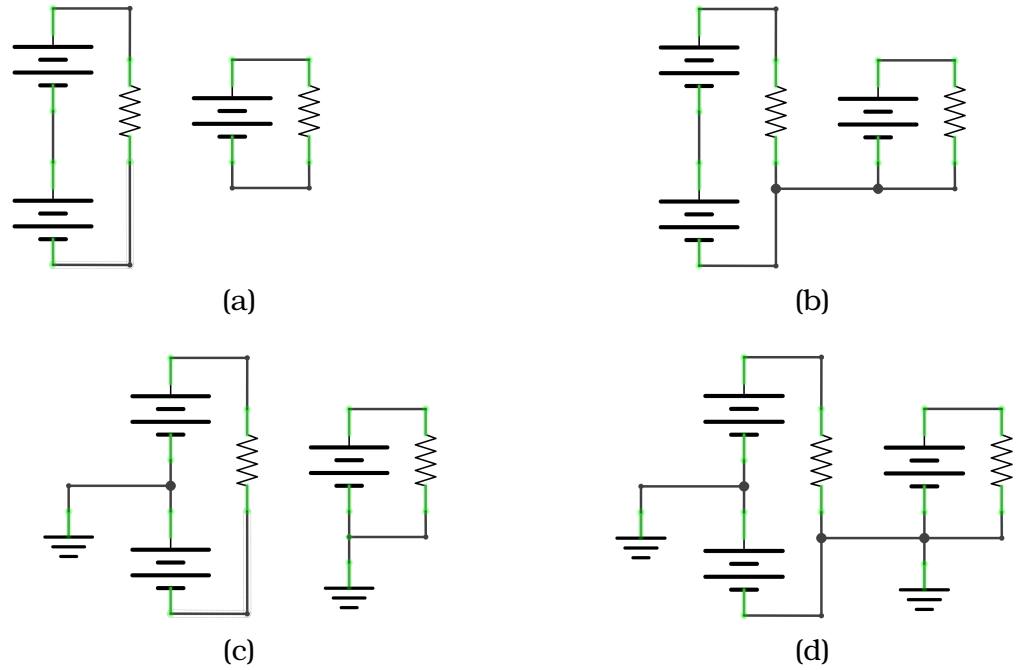


Figure 19.5: 19.5a, 19.5b and 19.5c are fine, 19.5d is a dangerous short-circuit situation. The ground symbol here denotes earth ground.

relative to the receiving circuit's ground level is undefined, and we will "read" random voltages, which may change randomly over time. You may actually pick up a radio signal. It's an antenna.

This is easy to fix. Just connect the ground levels of the two circuits, making them one circuit, as shown in Figures 19.4c and 19.4d.

Consider another example of two circuits that are driven either by batteries or isolated power supplies (that is, power supplies based on a transformer rather than switching-mode power supplies whose ground level in the circuit is the same as that of electric grid, and roughly the earth), as shown in Figure 19.5a. Everything is perfectly fine.

Now consider that we connect these two circuits by a wire as shown in Figure 19.5b. Everything is perfectly fine. That additional wire has no effect, as in the previous example.

Now let us connect both circuits to earth ground as shown[9] in Fig-

[9]The symbol of three parallel lines usually denotes ground in a technical circuit but not earth ground; that symbol is not available in Fritzing however, which was used to

ure 19.5c. If we now use a multimeter, we can connect its two probes to any two points in the two circuits (which now really have become one) and successfully measure a voltage. Note that the voltage at the minus-pole of the lower battery in the left circuit will now be negative relative to earth ground. Making ground "float" between two batteries (or isolated power supplies) is actually the way to obtain both positive and negative voltages, which are needed by amplifiers (which are needed in sound systems and some sensors). Also note that we do not have to explicitly connect our circuits to ground: If we replace the upper left and the right battery by switching-mode power supplies, we get the same effect. All this is perfectly fine, the circuit(s) work perfectly, performing exactly as before.

Now let us add the wire from Figure 19.5b again, obtaining the circuit of Figure 19.5d. Now we have short-circuited the lower-left battery. If the two other batteries we replaced by switching-mode power supplies as discussed before, the short-circuit is through the electric power grid.

Please grok this. **This is very important**. You will cause significant damage if you do not understand and avoid this situation. If one of the two circuits is powered by your microcontroller, which draws its power from the USB port of your laptop, which is drawing power from its (almost certainly switching-mode) power supply, the short-circuit will be through your laptop, and you may destroy your computer and lose your data!

Note that the upper left power source is actually irrelevant and can be removed from the picture; just confuse your wires in your thing, and you may get a short-circuit through your computer.

How do you fix this? Three options:

1. Run your laptop on battery.
2. Run your thing on battery.
3. Use a USB isolator between your laptop and your microcontroller.

Two related scenarios need to be discussed.

- **Oscilloscopes**. One of the two probes of an oscilloscope is directly connected to ground of the electric grid. If you use the oscilloscope to probe a circuit that is powered by a switching-mode power supply, and you touch the *wrong wire*, you cause a short-circuit. The problem is that you may easily forget what is wrong because you are just

create these schematics.

measuring and may want to visualize relative signals. Connect the ground probe to anything other than ground potential, and you have a short-circuit. You may damage or destroy the oscilloscope or its probes. If the thing draws power from your laptop (usually through USB and your microcontroller) while the laptop does not run on battery but is connected to mains power by its PSUs, you may damage or destroy your laptop.

- **Ground loops**. It is beyond the scope of this manual to cover ground loops in detail (but see https://en.wikipedia.org/wiki/Ground_loop_(electricity)). The kind of ground loops that matters to us arises when we draw large currents[10] (typically, using brushless motors) from a switching-mode power supply while your thing is connected to your laptop, which is powered by its own switching-mode power supply (i.e., it is not on battery). Even if everything is wired up correctly, In such a high-current scenario, the resistance and capacitance of the cables you use can be sufficient to create something that in effect is very similar to a short-circuit, even though no wiring error will be discernible in the schematics and the circuit is perfectly fine while you draw smaller currents. Again, this is avoided by USB isolation or running either your laptop or your thing on battery.

[10] See also http://docs.odriverobotics.com/v/latest/ground-loops.html .

Chapter 20

Making Circuits

We can create circuits in multiple ways. One is to use so-called breadboards, which allow for experimentation and rapid prototyping by electrically connecting components in a nonpermanent, reversible way. The alternative, permanent way is by soldering the components to a protoboard or a printed circuit board (PCB). Creating PCBs of our own design means to contract an external company to fabricate them for us, which incurs nontrivial delays.

20.1 Breadboards

Figure 20.1 shows a small breadboard. Each of the holes can hold the leg of an electronic component or a male connector at the end of a jumper cable. Groups of holes (rows of five holes are common) are connected by a conductive strip of metal in the back of the breadboard, so to connect two components, we just need to push them into two holes in the same group. Of course we must not use the remaining holes of that group unless we want to connect further components there. We can create circuits by pushing the legs of components into the right holes, and we can disassemble circuits and re-use the components and breadboard by simply pulling the components off the breadboard.

Figure 20.2 shows an example of a simple circuit schematics that lights up an LED and how this can be mapped to a breadboard. The components in the circuit are a battery, a resistor, and the LED. Note how, for instance, the resistor and the LED are connected by each having one leg in a hole

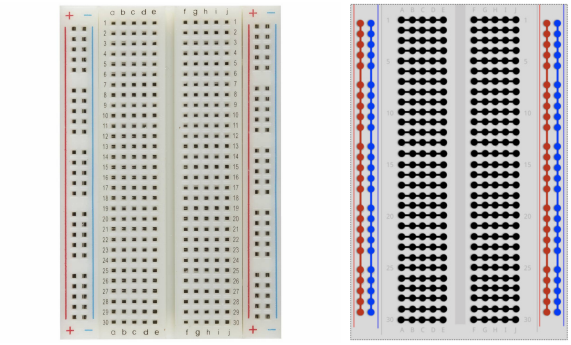


Figure 20.1: A small breadboard (left). The horizontal rows of five adjacent pinholes are each connected by a conducting rail; so are the two columns on the left and right of the board, labeled + and –. You should use these to supply power to your circuit. (right)

that belongs to the same group of five.

Mapping abstract circuit diagrams to a breadboard layout is a graph mapping problem that requires some thinking and needs to be practiced. Electronics CAD tools such as Fritzing can help you find errors. Be focussed and careful when building up the circuit. A mistake is easily made and takes time to discover, and in the worst case, an incorrect circuit can damage things, or worse (a short-circuited battery can explode, for instance – see the safety sections).

There are many places on the Web where you can see examples of such mappings, such as the Arduino example pages[1]. You'll find matching schematics like those in Figure 20.2 there.

20.1.1 The Soldering Practice Kit on a Breadboard

Let us have a look at the components of the soldering practice kit and build up the circuit on a breadboard.

The central component of the circuit is the 555 timer IC. This is one of the most famous integrated circuits – a classic. In the configuration we are using it, the astable configuration[2], it creates a precisely timed rectangular wave circuit. Even though this is an analog IC, what it does here is create essentially a digital signal – it switches its output between

[1] See e.g. https://docs.arduino.cc/built-in-examples/basics/Blink.
[2] https://en.wikipedia.org/wiki/555_timer_IC#Astable

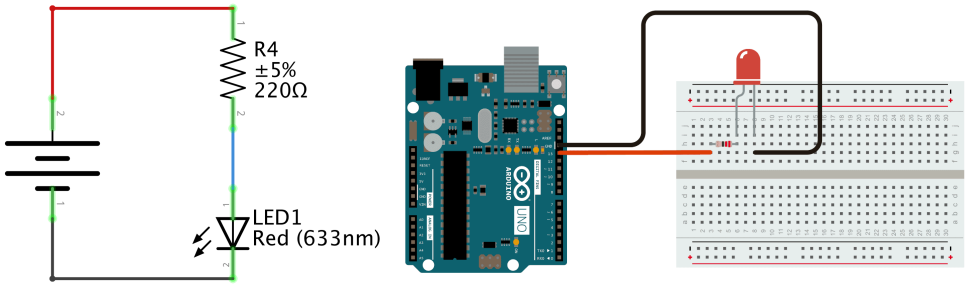


Figure 20.2: Schematics and breadboard view of an LED circuit. The assumption here is that pin 13 of the microcontroller board is permanently set to HIGH.

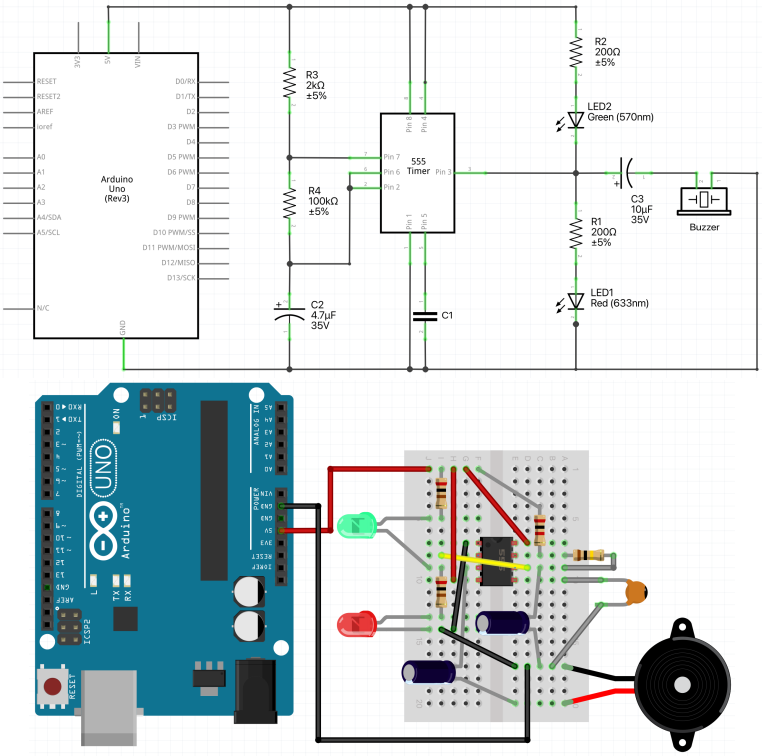


Figure 20.3: Schematics of the circuit of the soldering practice kit (top) and a corresponding breadboard view created using Fritzing (bottom).

0 and 1 and back (very) roughly once a second[3]. It does this without a crystal but by discharging a capacitor (which functions like a very small rechargable battery here) using a resistor. The delays between switching can be set by picking the right resistors and capacitors.

In addition to the components that constitute the core astable 555 circuit, we have two LEDs (plus their protective resistors) which alternately light up, and a Piezo buzzer that makes a clicking sound on each switch.

You do not need to understand how this circuit works[4] – in fact, its operation is relatively complex for such a small circuit – but try to prototype it on the breadboard.

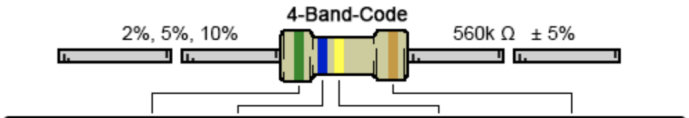
Note that the polarity (the orientation of a component in the circuit, relative to + and GND) does not matter for resistors or the small ceramic capacitor (the tiny lentil-shaped component), but it does matter for LEDs (the long leg is the anode – the leg closer to the positive pole of the power supply) and for electrolytic capacitors (the two black cylindrical elements; here the cathode is marked with a white strip of paint on the black cylinder, also marked "-"; this leg is oriented towards GND of the power supply). The Piezo buzzer has its + leg marked on the top of the housing. The 555 IC is in a DIP housing. The "top" end of the IC has a small semi-circular cutout on the top of its housing. Relative to that, pin 1 is on the top left and the remaining pins are numbered counterclockwise, so pin 4 is at the bottom left, pin 5 is on the bottom right, and pin 8 is on the top right. Note how the pins in the schematics of Figure 20.3 are positioned very differently: there, pins typically connected to the plus pole of the battery or power supply are placed at the top of the box representing the IC, pins typically connected to GND are at the bottom, "inputs" at the left, and "outputs" at the right.

Electrolytic capacitors have their rating in Farad (or, more typically, μF, microFarad) printed on explicitly. The resistance of resistors is coded via three colored rings (there is a fourth ring in silver or gold color that you can ignore). Find a legend for the color coding in Figure 20.4. For instance, A 470Ω resistor is marked yellow-violet-brown. Alternatively, if you are color-blind like me, you can use a multimeter to check a resistance.

One way to build up the circuit on a breadboard is shown in Figure 20.3. Supply the circuit with 5V. One way to do this is to connect it to the 5V

[3] If you want to know the exact frequency, you can calculate it here: https://ohmslawcalculator.com/555-astable-calculator.

[4] But if you want to, there are excellent youtube videos.



COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8	100MΩ	± 0.05%
White	9	9	9	1GΩ	
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

Figure 20.4: Resistor color codes. There are also 5- and 6-band versions, see the Web.

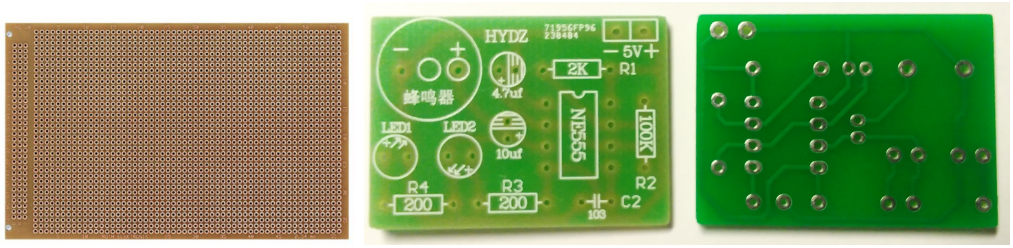


Figure 20.5: A protoboard for making ad-hoc soldered circuits (left) and the PCB of the soldering practice kit, shown recto (center) and verso (right).

output and GND of the Arduino UNO.

20.2 Soldering

Sometimes, however, you want to build permanent circuits, either on protoboard or printed circuit board. Soldered connections are more reliable and have fewer contact problems.

Sometimes, you will receive an electronics board for which the only way to connect other electronics to it is by soldering. Creating a soldered circuit will also make you feel a sense of achievement; such a finished electronics product is beautiful. Finally, as an engineer from EPFL you should be able to solder. It will come in handy one day.

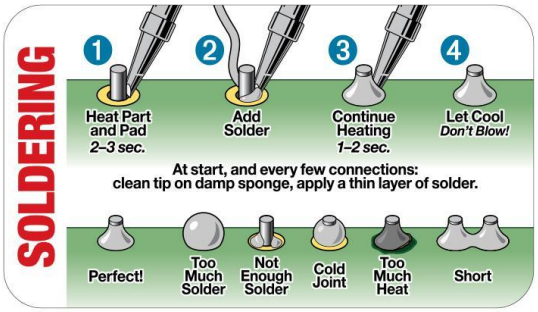


Figure 20.6: How to solder.

Try to heat up both parts that you want to solder together and only feed solder wire once they are hot.

If you solder a wire or leg of a component to a through-hole/pad on a printed circuit board or protoboard (with a copper surface ring around the hole), heat the solder until it forms a concave conical surface (best seen in the second solder joint from the right in the picture below). If it forms a pearl, the board surface is not hot enough and the solder has not bonded yet. You have what is called a cold solder joint. Keep heating until the pearl "pops" and you get the concave cone surface.

Speaking of pearls, I hope you understand that the goal of soldering is not to make a ball at the end of a piece of wire so that the wire end doesn't fall out of the hole. We want to create a solid metal bond between the wire and the through-hole of close to zero resistance. A wire just pushed through the through-hole and touching a corner of the copper surface is not a reliable connection. We don't want the electrical connection to possibly be interrupted when we touch the wire. Do this properly or you will waste a lot of time debugging.

Metals conduct heat very well, and spread it out away from the solder point. As a consequence,

- it can be hard to solder stuff to large, massive metal pieces.
- some components can be damaged by the heat while soldering. This includes ICs, batteries, and magnets. (Permanent magnets get destroyed at about 80 degrees Celsius, and motors contain them. Keep that in mind when soldering wires to your motor. You need to succeed with your soldering job relatively quickly, before the heat spreads too

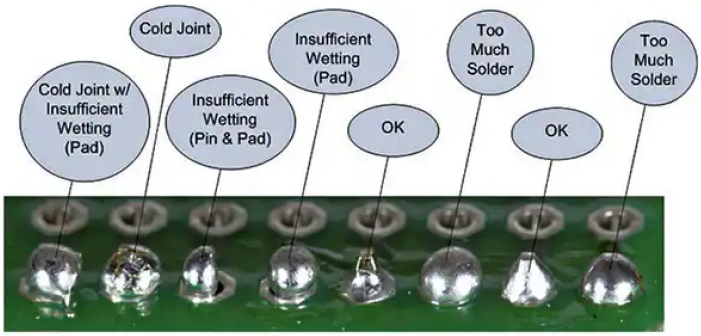


Figure 20.7: Found this on the Web. Regarding the fourth solder point from the right, not quite sure what I am seeing, but it doesn't look ok to me. The second from the right is nice.

much into the motor.) Try to get your solder connection done in 5 seconds or so. If it takes much longer for the solder to melt, something is probably wrong (the soldering tip is not hot enough).

Solder wire is not pure metal. It also contains chemicals ("rosin flux") that are there to help you succeed in making a good electrical connection (this includes an acid to remove oxidation from metal surfaces).

Try not to inhale the smoke while soldering, and work in a well-ventilated place.

Ideally, the surface of a finished solder joint should be shiny. If it is cloudy, there is probably still too much flux there, keep heating. This should only take a few seconds. The cloudiness does not always go away, though.

The materials in the solder wire and the flux are chosen to make soldering easy. Particularly the surface tension properties of the solder help you in your job. The solder has a tendency to appear intelligent, going exactly where you want it. For that reason, soldering can be very satisfying. Sometimes, however, it seems to have a mind of its own, refusing to do what you want. For example, one way to create a connection between two pads/through-holes on a protoboard is using solder, but to do this you need to create a connection across an uncoppered surface. This can be tricky, with the solder popping back to the coppered places and accumulating there.

When you solder a wire to a through-hole, make sure to cut the overhang of wires. "Angel hair" of stranded copper wire touching all kinds

of places can cause terrible situations, from your circuits not working to short-circuits, fires, and explosions.

Sometimes, the solder does not seem to want to melt, or bond to the metal items you want to connect – the soldering iron does not seem to get hot enough. This may have multiple causes. A common cause is that the tip of the soldering iron is covered by soot – charred flux. Have a wet piece of paper or sponge to wipe the tip of the soldering iron until it is shiny and silvery again.

IMPORTANT: Learn to solder well, and to create good electrical connections. It will save you a ton of time debugging the things you make.

20.2.1 Soldering the Practice Kit

Generally speaking, it is advisable to solder heat-sensitive components last. This usually applies most to ICs and least to resistors, but here we do not solder the 555 timer IC onto the PCB at all since the kit comes with an IC socket. This socket is soldered onto the PCB and the IC is pushed into its socket as the very final step after all the soldering has been completed.

Observe the polarities as discussed in Section 20.1.1. The IC socket has a semicircular cutout like the IC. Place the components into their places on the top side of the PCB. This PCB has drawings of the components painted on its front side, with enough information to clarify polarities. Push the legs of the component you want to solder on next through the correct holes from the top side. Push the component all the way down to the PCB surface and the legs all the way in. For a component with long legs, bend at least one leg over by 90 degrees, so the component cannot fall off the PCB when you turn it around to solder. Solder on the back side of the PCB, where there is a coppered ring around each hole to which the solder will bond. Solder two jumper cables with male connectors to the 5V +/- holes on the PCB. Again, do not solder the IC – solder the socket and push the IC into the socket without damaging its legs in the very end. Power your circuit and enjoy the impressive light and sound show.

If the explanations are not clear, search for videos on Youtube or ask a TA.

20.3 Making PCBs

There are various methods for making "printed" circuit boards, including some that do not require any expensive equipment, and which you can follow at home. However, they require chemicals such as iron chloride, which are caustic and cause hard-to-remove stains. You also mustn't flush used iron chloride solution in the toilet, so there is an additional problem of handling spent iron chloride solution. We don't use this approach and don't make our own PCBs in DLLEL. If you are interested in making PCBs at home for private purposes, there are many good tutorials on Youtube.

Our method for making PCBs is to design them with CAD software like Fritzing and then outsource their fabrication. The problem here is shipping – you may be waiting quite long for the fabricated boards to arrive, and, often, you have to go through multiple prototypes. For this reason, we haven't had any PCBs fabricated for this course yet. We have worked with protoboards, which are a less elegant solution, but cause no forced waiting times.

20.4 Recommended Videos

https://www.youtube.com/watch?v=3jAw41LRBxU
HackMakeMod
"HOW TO SOLDER! (Beginner's Guide)"

Chapter 21

Supplying Power

For safety reasons, we restrict ourselves to using only direct current (DC) power sources supplying *voltages* not exceeding 12V in this course. This is no real restriction given the kinds of projects the course is designed for, and is sufficient to drive all relevant kinds of electronic components including microcontrollers, sensors, and actuators. We will use two types of power sources, mains power supplies (PSUs) and batteries.

The key design choices that you have to make involve (1) choosing power sources that can supply the *currents* needed while satisfying your weight, volume, and mobility constraints, and (2) deciding on how to best solve the problem of supplying the multiple different voltages that your components may require.

The following table shows typical voltage and current ranges for various electronic components. Note that these are rough bounds that may be off in your case. You must check the specifications of each component you consider using before including it in your design. The ranges refer to families of components: Do not assume that an individual component can be powered by a voltage in as large a range. Many components, including compute units (microcontrollers) and motors, draw vastly different currents depending on whether they are in a high-load scenario or idling. Steppers apply a current to lock the shaft while not rotating and thus are constantly under load while powered.

component	voltage (V)	idle current (A)	load current (A)
Microcontroller	$5-10$	≈ 0.05	≈ 0.5
Small sensor	3.3 or 5	–	> 0.02
Small servo (SG90)	$5-6$	–	≈ 0.25
Small brushed motor	$3-8$	0.1	≈ 1
Bipolar stepper	12	–	≈ 1.5
Brushless motor	$12-60$	< 0.5	$30-100$

Remember that you are usually connecting your consumers to your power supply in parallel. For a rough calculation of your current needs, *add* up the currents of your consumers (see Chapter 19).

21.1 Mains Power Supply Units (PSUs)

PSUs are power supplies that connect to 230V AC mains power via a cable, so their are not suitable for non-stationary things.

Figure 21.1: Three 12V mains power supplies. The kind on the right is an unenclosed power supply that is forbidden in our course.

We restrict ourselves to fully enclosed PSUs with no more than 12V output. These typically supply currents up to less than 10A, which is typically not sufficient for brushless motors, but for all other kinds of components we use. We can use the lab bench power supplies in DLLEL; these supply up to 3A. Open PSUs such as the one in Figure 21.1 (right) are absolutely forbidden and will kill you if you touch them at the wrong place. Take the rightmost PSU in Figure 21.1, for instance. It is connected to the mains power outlet by a cable whose crimped end is connected to one of its front connectors by a clamp connection. Touch its connector with a finger and touch the any other metal surface of the power supply (which are connected to ground for "safety") or be badly isolated from physical ground, and 230V AC will flow through your body. You will probably die.

Figure 21.2: DC power connector and port.

You are absolutely forbidden from opening the enclosure of a PSU. This is mortally dangerous even if the PSU is disconnected from the wall plug – the internal capacitors can hold a dangerous charge for a long time! Since PSUs keep your thing connected to a wall plug by wire, they are for non-mobile projects such as 3d printers, plotters, robot arms, etc.

There are two kinds of PSUs: Isolating power supplies based on transformers and so-called switching-mode PSUs. The PSUs that we will use are typically switching mode power supplies, and are non-isolating – they have common ground with mains power. This includes the PSU of your laptop. You cannot put switching-mode PSUs in series to obtain higher voltages as this would short-circuit them. Transformer-based isolated PSUs can be put in series, but these are relatively uncommon these days.

You will need a DC port (see Figure 21.2 to connect your thing to the PSU.

21.2 Batteries

We use non-rechargeable (alkaline) batteries, for low-current situations and (re-chargeable) LIPOs for high-current applications. LIPOs asre covered in a chapter of their own – Chapter 22.

The batteries we usually use include 9V batteries, AA, and AAA batteries. Each AA or AAA battery offers a voltage of about 1.5V. We have battery boxes for putting them in series, to produce power sources with $4 * 1.5V = 6V$, $6 * 1.5V = 9V$, etc.

Non-rechargeable batteries are for small mobile projects that do not consume large currents – such as small autonomous vehicles – so we do not create too much battery waste. AA and AAA can be used to supply about one to two Amperes, which is enough for a couple of small brushed

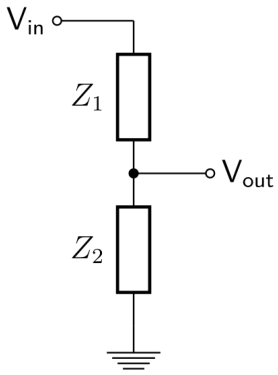


Figure 21.3: A voltage divider.

motors.

Drawing too large a current from a battery damages it. As a consequence, batteries appear empty more quickly than your calculations would suggest.

9V batteries are unsuitable for use with actuators. They are built from four tiny 1.5V batteries and cannot supply enough current for driving most actuators, and will die very quickly even for very small actuators. 9V batteries are well-suited for powering microcontrollers and small sensors.

Larger currents can be sustained when batteries are connected in parallel, but this is impractical for multiple reasons, including weight and the problem that batteries are chemically slightly different, causing them to discharge at different speeds, which causes batteries connected in parallel to lose charge as they try to keep each other at equal voltage.

If you need a rechargeable solution, we use LIPOs. We do not use rechargeable AA or AAA batteries. See Chapter 22 for more on LIPOs. These are for mobile applications that require higher currents than what alkaline batteries can provide.

21.3 Voltage Conversion

Sometimes we do not have a power supply available at the voltage a component needs. In that case, we can use a power supply with a higher voltage

and convert it down to the desired voltage.[1]

One simple way to reduce a voltage is using a voltage divider[2]. It requires just two resistors of correctly chosen resistance, to be arranged in your circuit as shown in Figure 21.3. To obtain a desired voltage V_{out} from a given voltage V_{in}, choose resistances Z_1 and Z_2 such that

$$V_{out} = \frac{Z_2}{Z_1 + Z_2} * V_{in}$$

This formula follows from Kirchhoff's laws and Ohm's law (see Section 19.2.) Thus, if we pick Z_1 and Z_2 to be the same, $V_{out} = \frac{V_{in}}{2}$. Note that this gives you one degree of freedom in choosing your resistances – pick, say, Z_1 arbitrarily and calculate Z_2 from V_{in}, V_{out}, and Z_1. Of course, a high Z_1 severely limits the current that can be provided to the consumer, and a low Z_1 means wasting a lot of power. (See Example 19.2.1 on how to calculate this; Z_1 and Z_2 correspond to R_1 and R_2 there, and R_3 corresponds to the consumer using V_{out}.)

We do not want to use voltage dividers to drive large currents. However, one strong point of voltage dividers is that they aren't just relevant for supplying power: V_{in} does not need to remain at the same level but can be a (DC) signal: We can use the voltage divider to reduce its logic level (see Chapter 26).

There are also other solutions for voltage conversion such as transformers (heavy, voluminous, and expensive), buck converters, and voltage regulators. For instance, the buck converter LM2596 shown in Figure 21.4 supports an input voltage of 3.2-40V, an output voltage of 1.25-35V (but less than input voltage), and the maximum output current is 3A, at a price of about CHF 3. The output voltage is set with a little potentiometer – the gold-colored screw on the blue box. Use a multimeter to set the desired output voltage before connecting the output of the buck converter to consumers.

Voltage regulators are relatively small components (in packages similar to MOSFETs). The LM7805 is an example of a voltage regulator which reduces voltages to 5V, see Figure 21.4). Voltage regulators turn the excess power into heat and are very inefficient. They are not to be used for large-wattage or battery-driven applications.

[1] It is possible to convert a voltage up using transformers and boost converters, but this is usually less practical for us, so we don't discuss this option further here.

[2] See also https://en.wikipedia.org/wiki/Voltage_divider.

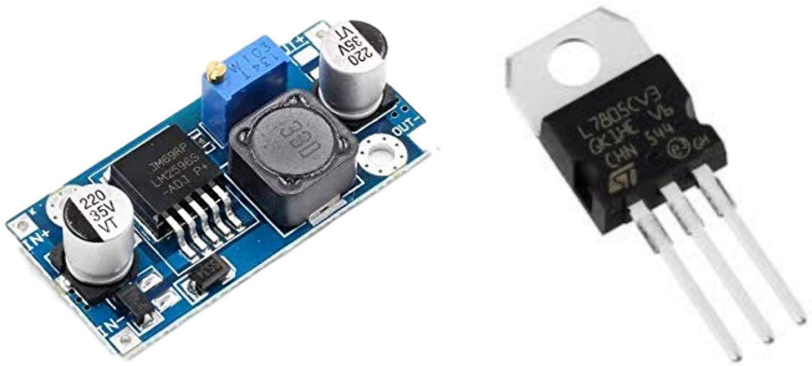


Figure 21.4: An LM2596 buck converter (left) and the LM7805 voltage regulator (right). Not to scale: The voltage regulator is much smaller than the buck converter.

21.4 Supplying Multiple Voltages

If you need different voltages for different consumers, you can use multiple power sources, but usually the better solution is to use a single power source and to use voltage conversion to get the voltages you need.

If you use multiple power sources, you will typically need to connect all their ground lines (have a common ground), see Figure 21.5. Among other things, this ensures that the digital signals from the microcontroller are understood by all components in your system: 0/LOW is “0V”, GROUND. (See also Chapter 19.)

Also note that some components, including microcontroller boards such as the Arduino Uno and some motor driver boards like the L298N (see Chapter 38), have on-board voltage regulators that can output lower voltages than they are provided with to power other components. For example, an Arduino Uno can be powered by up to 10V via its DC connector or Vin pin and can output 3.3 and 5V at the respective pins, which can be used to power certain low-current components. The L298N can be powered by up to 12V and outputs 5V at a pin, by which a microcontroller can be powered.

The typical way to power multiple consumers by a single power supply is in parallel. This will still be a single electronic circuit and consumers can affect and interfere with each other. Certain components, particularly those with coils (which includes all motors) and cheap switching

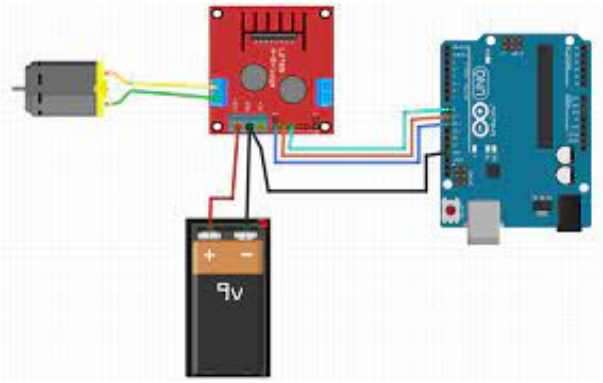


Figure 21.5: Here, a motor is driven using a 9V power supply (and a L298N motor driver). The microcontroller is driven by a separate power supply, but the ground lines of the 9V supply and the microcontroller are connected (see the black lines). Without this, the PWM signals from the microcontroller setting motor speed would not have a meaningful ground reference, so without common ground the circuit would not work. By the way, this 9V battery will not live long. Also, the L298N can output 5V to power your microcontroller, even though it is not done in this figure.

mode power supplies (which includes buck converters), can produce voltage spikes that negatively affect the operation of other consumers. This is an **important pitfall**. Particularly stepper motor drivers can work highly erratically if other components in your thing create voltage spikes. If you believe that you are dealing with such a scenario, talk to an expert. Some voltage spikes can be filtered out by placing the right kinds of capacitors in parallel with your consumers.

21.5 Designing your Power Supplying Solution

Is your thing stationary (use a PSU) or mobile (use batteries)?

What are your consumers? Which voltages and currents do they require? You have to provision for maximum current, but keep in mind that sometimes max currents of multiple consumers do not need to be added because they are not active at the same time. An example of this are two-coil steppers (like the 17HS4401), which use 1.5A per coil, but no more than 2A concurrently.

Actuators have current peaks on acceleration and create voltage spikes particularly when shut down (back-EMF).

If possible, if you have multiple consumers with different voltage requirements, use buck converters or voltage regulators with a single power supply, rather than multiple power supplies. This results in a cheaper and more compact thing. However, you may need to experiment with and turn your power supplying solution – your buck converters may provide unclean power with voltage spikes that affect your electronic components.

21.6 Cabling

You power cables need to be thick enough for the currents you are going to encounter. All wires have internal resistance[3], which at once wastes energy, reduces the voltages we want to supply, and, most critically in the context of this course, produces heat. An under-dimensioned power cable can produce a fire and, subsequently, once the insulation is burnt off, a short-circuit.

To calculate the required cross-section A of a copper cable (in m^2, taking into account just the copper, not the insulating material) needed, we use the formula

$$A = \frac{2 * 2.22 * 10^{-8} * I * L}{V}$$

where L is the length of the cable in meters, I is the current in Amperes, and V is the desired (upper bound on) voltage drop (in volts). Here, $2.22 * 10^{-8}\Omega/m$ is the resistivity constant[4] for copper wire; for other materials it is different. The gauge of wire is measured in mm^2 in Europe, so you need to multiply the result of your computation by 10^6. As for the voltage drop, bounding it to 3% is usual. Keep in mind that the voltage drop isn't just a concern because of less voltage being available to our consumers, but also because of the heat being generated.

Example 21.6.1 Suppose we need a cable of length $1m$ for 12V and 60A, and we want a voltage drop of (no more than) 3%, i.e., $12 * 0.03 = 0.36V$.

[3] They act as resistors, but they also act as capacitors. This becomes a problem when we care about the quality of our signals – particularly in audio systems – because cables dampen or filter signals. Furthermore, when we are dealing with large currents we have to worry about ground loops.

[4] It's actually not a constant, and depends on things such as temperature. We assume room temperature here. If the wire gets really hot, this becomes incorrect.

Then the cross-section area of the cable needs to be $A = 0.0444 * 60 * 1/.36 = 7.4mm^2$.

This cable will produce $0.36V * 60A = 21.6W$ of heat across its length. While this is quite a lot on absolute terms, over a meter of length, and given the considerable girth of the cable, radiating off this heat should not be a big problem.

A few observations are in order here. First of all, $7.4mm^2$ is a lot, and cables that thick are incommon, expensive, and unwieldy (you may have problems fitting it into your thing and bending it as needed – the cable may rather want to break your thing than bend). You may also have problems soldering the cable to connectors or a board because these components may not be meant for cables as thick as this. (Even though they can handle the current!)

If our voltage and current are givens, the quantities we can possible adapt are cable length and acceptable voltage drop. Increasing the voltage drop increases heat generated, up to poentially unsafe levels. As you can see in the formula, decreasing the length of the cable by half does the same to the required cable cross-section. Keeping your power cables as short as possible is highly recommended (as long as it does not lead to unsafe choices elsewhere in your design). [5]

If power losses are a minor concern for you, but you worry about your cables overheating, and you want to minimize their cross-sections, also consider whether your calculated peak currents will be sustained for extended periods of time. If your current draw is much lower on average, and there is sufficient time between short peaks for the cables to cool down, using a smaller cable gauge is reasonable.

Finally, within reason, you may under-dimension your cables for the sake of an experiment if the right cable gauge is not readily available. But in this case you need to remain present at all times while your experiment is receiving power, and you need to take safety precautions, such as en-

[5] This is also why very high voltages (380000V and 220000V in Switzerland, see https://www.swissgrid.ch/en/home/operation/power-grid/swiss-power-grid.html) are used in long-distance power transmission systems, and voltages are specifically transformed up at the power plants and down closer to the consumers. This minimizes both cable cost and weight, which are significant concerns once thousands of kilometers of cable need to be suspended from masts. The thicknesses of the cables is primarily governed by the desire to minimize power losses (voltage drops) and achieve the required tensile strengths of the cables to space out masts generously; voltage and current only figure indirectly.

suring that a burn-off of insulation does not cause a short-circuit (keep cables apart). In particular, jumper wires are intended for signals, not for supplying large currents. Using them to supply up to about 2A (small motors and steppers) for the sake of a brief experiment is ok if you respect the rules just mentioned. If you encounter a smell or smoke, turn off your thing immediately.

The cable cross-sections we have most readily available, in addition to jumper cables, are $0.7mm^2$, $1.5mm^2$, and $2.5mm^2$. We expect that you encounter currents in excess of 5A only when working with brushless motors or large numbers of steppers.

You do not need to include cables of any gauge in your bill of materials; however, if the cables you need are too thick to be practical, we will ask you to redesign your thing to work with more reasonable cable gauges.

21.7 Connectors (Plugs)

Now that you have determined the correct gauges for your power cables, we have to look at making connections.

Keep in mind that any connector is a potential source of failure and, particularly if it hasn't been soldered really well to its cable, adds to the voltage drop. A well-done solder connection is a better electric connection that one achieved by connectors. On the other hand, we need to be able to separate your thing from its power source (without cutting wires), so some connectors are unavoidable.

If you receive a component (such as a PSU, a battery, or a motor) that comes with cables with connectors, **do not remove/replace the connectors** unless granted explicit permission by the course staff. **Under no condition may you remove, replace, or alter any connectors coming with a LIPO battery**. Any statement to the countrary, even by staff, is invalid!

As for which types of connectors to choose, alas, threre are many, and different connectors are intended for different voltages and currents. By default, we use XT plugs (of which there are three kinds – XT30/60/90 – for different currents) for power supplies.

Female connectors always come with the power supplying side of a circuit, and male connectors with the consumer side!

As a rule, you need to buy/add to your bill of materials the connectors

(plugs) you will need, as these might be quite specific to the other components that you are ordering, and there are just too many different types to keep them all in stock.

21.8 Recommended Videos

https://www.youtube.com/watch?v=IT19dg73nKU
DroneBot Workshop
“Power For Your Electronics Projects - Voltage Regulators and Converters”

Chapter 22

Lithium-Polymer (LIPO) Batteries

22.1 LIPOs are Dangerous

LIPO batteries are rechargeable and can supply extremely large currents (more than 100A for some LIPOs).[1] But they are by far the most dangerous kind of component you may encounter in this course, in practice. Unfortunately, for some applications (particularly those involving powerful motors), they are unavoidable and thus permitted in the course. If you want to use them, you will have to read, understand, and respect all our instructions (starting with those in Chapter 11 on safety). LIPOs have very high power density – they are small and light considering the charge they can hold and the power they can supply. This is at once their greatest feature and their biggest downside.

If you discharge LIPOs too far or overcharge them, you damage or destroy them. **If you create a short-circuit, they will likely explode**[2] in a large fireball **spewing toxic chemicals** and potentially **causing you serious burns.** There are plenty of videos on Youtube demonstrating this – we link to one in the end of Chapter 11. The smoke/steam released by a LIPO in an accident contains Lithium, which is toxic.[3]

To put things into context, a standard measure for destructive energy

[1] You can supply even more by putting multiple LIPOs in parallel. Never do this without expert advice and supervision. If these LIPOs are even slightly differently charged, they will try to spontaneously equalize their voltages, creating a short-circuit situation.

[2] This is not an explosion – a detonation – in the technical sense that a shock wave is travelling faster than the speed of sound; still it is very dangerous.

[3] https://en.wikipedia.org/wiki/Lithium_toxicity

is tons TNT-equivalent. For example, the explosive energy released by nuclear weapons is commonly indicated in kilotons or megatons TNT-equivalent. TNT is a high explosive mainly used in military applications. One gram of TNT releases 4184 Joules of energy on explosion. The rightmost LIPO in Figure 22.1 is rated as 5.5Ah or rougly $3600s * 5.5Ah * 11.1V = 219780J$, or 53 grams TNT-equivalent. At a mass of 389g, the energy density of the LIPO with all its inert packaging contributing to weight is more than one-eighth of that of TNT. For another comparison, the muzzle energy of a .357 Magnum bullet is 790J, about one-300th of the energy rating of the LIPO.[4]

These calculations assume using the 5.5Ah available in a discharge to about 10V. A short-circuit can discharge the LIPO very quickly well below that, releasing even more energy. Even an "empty" LIPO at below 3.3V and an over-discharged and thus destroyed LIPO at below 3V per cell is dangerous when short-circuited.

Your mobile phone and your laptop contain LIPOs. Do not conclude from the fact that you are comfortable with these devices that we are being over-cautious. These devices have been expertly designed by professional electrical engineers. The same cannot typically be said for your thing. LIPOs are no joking matter for us. No students have been *physically* harmed by LIPOs in this course yet, and it must remain so.

22.2 LIPO Ratings

The basic unit a LIPO battery is built from is the LIPO cell. A LIPO cell supplies a nominal 3.7V; 4.2V when fully charged. You should not discharge it below 3.3V; below that it will take damage and will generally be unable to take a charge again. Thus, **LIPO batteries need to be protected from a complete discharge.** LIPO batteries are built from multiple LIPO cells connected in series to achieve the voltage necessary for your application.

Each LIPO battery has at least three important indications:

- an xS rating (such as 2S or 3S) which indicates the number of cells it is constructed from and thus its nominal voltage,
- a maximum charge rating in mAh (milliAmpere-hours), indicating the work it can do, and

[4] See https://en.wikipedia.org/wiki/.357_Magnum.

- a yC rating (such as 25C or 40C) that indicates the maximum current it can safely supply. Using the LIPO in a high-current scenario may require take steps to ensure heat dissipation/cooling.

The number of cells in series is indicated on the battery by an xS indication, meaning x cells in series. For example, a 2S LIPO offers $2 * 3.7V = 7.4V$ nominally and $2 * 4.2V = 8.4V$ when fully charged. A 3S LIPO offers $3 * 3.7V = 11.1V$ nominally and $3 * 4.2V = 12.6V$ when fully charged. Li-Pos need balancing chargers that can charge each cell individually. For this reason, multi-cell LIPOs have, in addition to the main two-wire power plug (+ and GND) another xh2.54 plug with $x + 1$ wires for an xS battery. (This gets plugged into the charger and remains unconnected when using – discharging – the battery.)

A LIPO with a yC rating and a z mAh rating can safely supply $y * z/1000$ A. So a 5000mAh 25C LiPo can supply 125A. The maximum discharging current (for short current spikes) is indicated on the batteries and is, usually, two times the continuous current computable from the C-rating.

If you want to combine multiple LIPO battery packs, either in series or parallel, you must first get expert help before you put anything into even experimental operation. Doing this can cause serious damage if you connect them in parallel and their voltages/charging statuses are different. Even slightly different voltages can cause substantial current flows!

While the plugs of the balancing wires are standardized (xh2.54), the power plugs of the batteries are not standardized. Make sure you have matching plugs on your charger and for your thing! Our favorite connectors for this are XT60 and XT90.

Do not get fooled by pictures, LIPO batteries differ vastly in size and weight – make sure you take the right dimensions for your 3D design!

22.3 Recharging a LIPO

If you need a LIPO, you also need a fitting charger. LIPOs need to be recharged as soon as their voltage drops below about 3.3V, not when they are completely empty. Otherwise, the LIPO will be impossible to re-charge. LIPOs must not the charged beyond 4.2V, otherwise they are damaged, too. A LIPO charging station ensures that no overcharge happens.

In multi-cell LIPOs, the cells are connected in series. It is not safe to recharge them by just connecting them to a power source with the right

			
voltage	2S 7.4V	3S 11.1V	3S 11.1V
charge	1000 mAh	1000 mAh	5500mAh
current	25C (25A)	25C (25A)	20C (110A)
weight	71g	104g	389g
dimensions	71x31x16mm	75x31x22mm	143x44x35mm
connector	BEC	BEC	XT90
price	CHF 11	CHF 17	CHF 52

Figure 22.1: Three LIPOs (all Conrad Energy Softcase).

voltage via their (usually red) + and (usually black) -/GND cables (their "thick" cables); one must ensure that all the cells are charged to equal voltages, otherwise the LIPOs get damaged over time. The cells of a LIPO are all chemically slightly different, so this does not happen automatically – cell voltages need to remain balanced while charging. For that reason, multi-cell LIPOs have additional thin balancing cables: usually, $k+1$ for a k-cell LIPO connected to a common white JST-xh2.54 plug. You can see these in the three pictures in Figure 22.1. By the way, the two main power cables (red and black, usually thicker cables) are usually also leading to a plug. There are many different standards. We try, for larger LIPOs, to stick with XT60/90 plugs – these are yellow (see the rightmost picture in Figure 22.1).

When we provide you with LIPOs, we also provide you with LIPO protection circuits (aka battery management systems (BMS)), which usually handle all of the following, but make sure to read their datasheets and verify that they indeed do all of this:

- Overcharge protection and balanced charging.
- Overdischarge protection.
- Discharge current limiting, and thus a short-circuit protection. (But even if they do have this, you must not rely on it and take measures

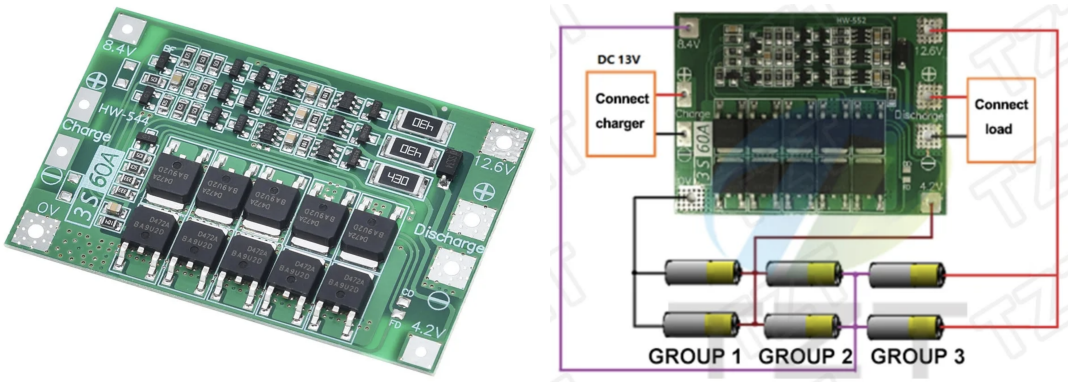


Figure 22.2: A LIPO protection board for 3S LIPOs and the wiring diagram.

to avoid short-circuits).

You will have to solder cables and plugs to your LIPO protection circuit.

22.4 LIPO Protection Circuit Setup

Figure 22.2 shows one of the LIPO protection boards we have in stock. The board is for 3S LIPOs and maximum currents of 60A. For sustained high currents, you should mount the board to a heat sink such as a plate of aluminium cut in the machine shop and touching the MOSFETs on the board, ideally using thermally conductive grease or glue to leave no air gaps between the top-sides of the MOSFETs and the metal plate.

While the circuit diagram shows a matrix of three cells in series with two cells in parallel (the battery symbol represents a since LIPO cell), we will use this board with a single 3S LIPO. This LIPO has two thick power cables and a four-wire xh2.54 plug. These four wires are connected to the 0, 4.2, 8.4, and 12.6V levels of the LIPO, respectively; the thick wires are connected to 0 and 12.6V, respectively. As you can see in the circuit diagram, if we think of the three cells of the LIPO connected in series from ground-most first cell (labeled group 1 in the diagram) to the 12.6V-side third cell (labeled group 3 in the diagram), the 4.2V line connects to the + pole of the first and the - pole of the second cell, while the 8.4V line connects to the + pole of the second and thus the - pole of the third cell. You need to solder wires and plugs to the board:

Figure 22.3: XT60 plugs, female on the left and male on the right.

- The 0V and 12.6V contacts of the board (shown in the bottom-left and top-right corners of the board in the wiring diagram) need to connect to the black and red cables of the LIPO, respectively. Solder a male XT60 or XT90 plug via suitably thick cables to the contacts on the board. The plug needs to fit the (female) plug on the LIPO. Remember the convention: **The power source side of a connection always has a female plug and the consumer side always has a male plug.** This is a safety feature – the contacts of a female plug are harder to unintentionally short-circuit.

- The 4.2V and 8.4V contacts of the board (shown top-left and bottom-right in the wiring diagram) need to connect to the 4.2V and 8.4V pins of the xh2.54 plug of the LIPO, respectively. Check which is which using a multimeter, relative to GND – the black cable. Solder an appropriate xh2.54 plug via suitable cables to these board contacts. The currents flowing through these cables aren't large, though jumper cables are just a little too thin.

- The + and - charging contacts of the board (shown left in the wiring diagram) need to connect to the mains power supply. The charging voltage has to be between 12.6 and 13V. The right connector to be soldered via suitably thick cables to the board contacts depends on the power supply.

- Solder, via thick cables, the female version of the same kind of (XT60 or XT90) connector that you have on your LIPO to the load + and - contacts of the board (shown right in the circuit diagram). Your thing will connect to this plug via a matching male plug.

If you have received a different LIPO protection board (usually because your LIPO isn't 3S), search aliexpress.com for "LIPO protection board" and

compare images until you find your board. There will be circuit diagrams and instructions. The board will be similar to the one shown here with a different number of contacts but analogous wiring. In case of doubt, talk to us.

Getting the wiring wrong will have catastrophic consequences. You must show your board with soldered cables to Sylvain Hauser, Simon Lütolf, or Christoph and get their feedback before you may connect the LIPO to the board for the first time. Go-aheads are not to be sought from student assistants and are invalid if given!

The main electronic components visible on the LIPO protection board are MOSFETs. These are particularly sensitive to electrostatic discharges from your body, so please take extra care not to touch them or any metal contacts on the board!

22.5 Rules for using LIPOs

Here are rules that you must adhere to when using LIPOs (do not just read this, **read the entire chapter**):

- Connect your LIPO to its protection board and only charge and discharge (=use) it through the board. See instructions in the previous section.
- Make extra sure that there are no short-circuits involving a circuit with a LIPO in it.
- Design your thing in a way that your LIPO is easily accessible from the outside of the thing. Do not fully enclose it to allow heat to radiate away and to avoid a true explosion due to the rapid build-up of hot gas in an enclosed space in case something goes wrong. If you are drawing high currents by design, you may need to take steps in your design to keep the LIPO cool. Discuss this with the teaching staff.
- **Never** try out whether our warnings are justified. Never intentionally create a short-circuit “for a moment”. You may mean to create a contact for a moment, but because of the intense current flow and the fact that the greatest heat generation will be happening where you make contact (because resistance is still highest there), the two metal surfaces you made touch will probably be welded together and

you **will not be able to separate them**. We are talking of true welding here. The heat melts even steel and no solder (tin) is required.

- Do not design your thing with a switch between LIPO and LIPO protection circuit. Switches there do not survive for long (for reasons related to the previous bullet point and power spikes that can arise due to the switching action – switching large currents can produce too much heat in the switch). You should switch big consumers – mainly motor drivers – off via their input signal lines from the microcontroller.

- At the end of a work session, disconnect the LIPO from your thing by unplugging it while big consumers do not draw current. Depending on your design, even if your thing is "off", there may be a small leakage current from the LIPO through your thing, eventually overdischarging the LIPO if it remains connected to the thing for extended time periods.

- Have a team member be close to your thing and monitor it at all times while the LIPO is connected.

- Never unplug a LIPO by pulling on cables. Always hold the plastic plugs – the male and female half of the connector and pull them apart. This can be uncomfortable and even hurt your fingers a little since quite a bit of force may be necessary to unplug the LIPO, while the plastic plugs are quite small with sharp-ish edges. But this is unfortunately unavoidable. If you rip the plug off the cable, the likelihood of cable ends touching and a short-circuit happening is very high.

  Generally speaking, independently of whether your thing involves LIPOs, pulling on cables is to be avoided since it may break cables internally, leading to hard-to-debug issues, *wasting your time*.

- Do not use a LIPO that has expanded in volume. It must be replaced. Slight volume expansion over extended time periods is somewhat normal. If you can see the LIPO expanding, that is never safe. In case of doubt, talk to an expert.

- Never bend, cut, or stab a LIPO.[5] If the skin of the LIPO is damaged, notify us and do not use it. Piercing a LIPO usually leads to an im-

[5] This is another reason why we do not allow drones in the course.

mediate and intense release of toxic steam. Never work with power tools such as drills close to a LIPO.

You will receive no sympathy from the DLLEL or course staff if we find you to not respect and follow these rules. This is just too serious. You are putting yourself and your colleagues at risk of serious harm, and you may make it impossible for us to offer this course in this form – where students can build mobile things – in the future. If we see you violate these rules, we will take away your LIPOs, even if this means that you cannot continue the project as planned.

If you are using LIPOs in your project, this chapter is not for reference but required reading in its entirety. You must know its contents and these rules, and we will quiz you – that is, every team member – on them before we give you any LIPOs.

Finally, a *non-rule*: You may read that, for long-term storage, LIPOs should be half-charged to maximize their lifetime. This is something you do not need to worry about for the duration of the course.

22.6 What to do in Case of an Accident

If a LIPO accident is in the process of happening,

1. see if you can safely unplug the LIPO. Never attempt this if it puts you in harm's way. If the LIPO pops open and smoke or steam comes out, remember that it is toxic.

2. if unplugging the LIPO is not a safe option, see if you can use a pliers to cut one of the two power cables (red or black). Do not attempt this if it puts you in harm's way. Never cut both power cables! This would almost be guaranteed to create a short-circuit there. Preferably cut the cable away from the LIPO and closer to the plug.

3. Otherwise, step away to a safe distance and alarm others nearby. Then contact the DLLEL staff and, if none are within reach, call the campus safety officers/firefighters (115).

In an overcurrent/short-circuit situation, the first thing that usually happens is that the volume of the LIPO expands. Its outer skin is a soft plastic bag-style material that can easily deform and is meant to contain

the chemicals making up the battery for as long as possible. If you see your LIPO expanding, take emergency action. You may have a few seconds to intervene or get to safety before it pops open and/or fire breaks out.

Chapter 23

Cable Management

Figure 23.1: Electric cabling above a street elsewhere where they don't read this manual. The cabling in your project will likely look similar unless you consciously do something about it.

This chapter is not about choosing the right cable gauges or connectors: for that, see Section 21.6.

There is great temptation to focus all your efforts on the exciting parts of your project – the microcontrollers, sensors, actuators; the mechanical structure – and to do your cabling in a chaotic, ad-hoc fashion. So, many projects tend to have a mess of cables and electronics boards hanging off of them.[1]

This is a grave mistake. If you do not put sufficient thinking into your wiring solution, you will pay for it very dearly. It will probably be the greatest source of error and frustration, and the greatest time-waster in your project.

The two main things we have to address are

- Keeping order. By keeping the cables out of the way, you make your thing look pleasing, easy to work with, extend, and maintain.

  It also helps avoiding wrong connections (confusing cables). Wrong connections will make your thing not work at best and cause death and destruction in the worst case.

- Avoiding bad/broken connections/cabling.

You do this not just to please us and get a good grade. Proper cable management throughout the duration of the project will save you much time debugging. So it is important to do cable management from the start of your project, not just to beautify your final product.

23.1 Keeping Order

You can achieve order in three ways, all of which you should pursue where applicable.

- Use consistent color-coding. Some colors are conventionally used for certain roles, such as black cables for ground lines and red cables for the + power supply lines. Unfortunately, we will generally not have

[1] Do not allow electronic components to be attached to your thing solely by cables. That's really bad practice and we will penalize this in your grade in the individual project as well as the team project. Screw-mount your components solidly to your thing.

enough red and black cables for everyone – jumper cables come in packs of ten colors with equal numbers of cables in each color.

The least thing you can try to do is consistently substitute certain colors with certain others (e.g., black with blue), have agreed-upon conventions in your team, and avoid choices that invite errors, such are switching the role of red and black cables. Remember, you are not alone in your team, and your colleagues may assume you have observed conventions without checking the next time they work on your thing in your absence. Confusing cables can lead to severe accidents and damage!

- Have an organized plan for where to route cables through your thing. This starts with considering cabling when you do your 3D design of the items to be 3D-printed or laser-cut.

  Try to have cables of the right length, not too short as to potentially damage them, and not to long at to be in the way. When you make your own cables, make them in the right length, or else use cable binders and tape to get excess length out of the way.

  As a caveat, there are some cables we do not want you to cut, such as the cables attached to LIPO batteries, brushless motor drivers, and servos. The reason for this are threefold:

  - You may cut them at the wrong length (too short). Once they are cut, there is no way back. You may still end up changing your design, requiring longer cables. Soldering extension cables on is a sub-optimal solution.
  - Such cables often come with standardized connectors mounted well, and your own connection solution may be inferior, causing connection problems.
  - After the end of the semester, we may want to re-use the components of your project. (Speaking for the team project,) these components are not all yours.

  In general, if an electronic component such as a servo that we provide you with has a cable attached that you want to cut or modify, make sure to get our permission first, which will only be granted in exceptional circumstances!

- Make your own circuit boards (using protoboards or even custom printed circuit boards) to route connections in the plane of the board rather than to have the cables float in the third dimension. This can also solve the problem of connectors being unreliable and cable connections detaching.

  In some cases, there are integated components that reduce the amount of cabling you'd have in your home-grown solution – an example is the Arduino CNC Shields to drive up to four stepper motors, operate limit switches, etc.

23.2 Avoiding Broken Cables

Bad cable connections (that conduct badly or intermittently) and broken cables are extremely hard to debug. Cables often break internally (the metal wires but not the plastic sheathing), in ways invisible to the eye. It can occur anywhere along the length of the cable. A broken cable often is not consistently dead but may conduct at times (when the ends of the wires at the point of break touch) and become non-conductive by just touching or slightly bending the cable.

To increase the lifetime of cables, avoid friction and pinching. Kinking a cable always damages it. In a thing with moving parts, make sure to optimize your design to minimize the risk of actively damaging your cables. For moving things, don't let your cables touch the ground. For things with joints (such as robot arms and biologically-inspired robots), make sure not to pinch cables and ensure that the cables do not get pulled on in any articulation of your thing.

Generally speaking, your cables age every time you bend them, and some bending is unavoidable in many projects. Try to minimize the frequency and speed of bending. Cable sleeves (see the Prusa printers, or search the Web for images) can increase the lifetime of cables by distributing the bending over a greater length of cable and so making the bending less extreme locally.

Chapter 24

Debugging Electronic Circuits

In some ways, debugging circuits is not fundamentally different from debugging software. The same kind of analytical thinking is required to isolate possible causes and track down a problem. As a computer scientist, you are well trained in this. Don't forget about what you already know about debugging software when trying to debug circuits! Sit down and think, and then go about debugging methodically. Don't do random stuff.

This chapter is not about original electronics designs. From time to time, an electronics expert takes the course and we allow them to go beyond the usual, but for most of us, electronics work means following recipes and gluing several recipes together into a single circuit by connecting multiple consumers to a power source in parallel and using a single microcontroller to control everything else by connecting the microcontroller to the signal lines of various other components. This chapter is about debugging problems arising when you do this more humble kind of electronics work.

24.1 Be Organized

Clean up your workbench before doing electronics work. Don't have conductive items such as screws lie around where they may touch the electronics you are debugging. You just need to be a bit unlucky and you are dealing with short-circuit with all its nasty consequences.

You may be working as a team, and you may feel that you are time-pressed, and that is why you don't want to maintain an orderly workspace,

but working on a pile of miscellaneous electronic components, cables, 3D-printed parts, clothes, laptops, and paper notes is a recipe for mistakes and even accidents.

Sometimes you need to do experiments. It may, for example, involve trying out different sets of electronic components. Say you want to test the theory that (at least) one[1] of your electronic components is faulty, and you have replacements to try for each of them. There are lots of combinations, and they are easy to confuse. If you are badly organized, you are likely to waste your time re-trying the same combinations and will miss the combination you are looking for and which provides your answer. In the experimental sciences, proper labeling and bookkeeping in lab notebooks is mandatory. It is recommended for you to do the same.

24.2 Debugging Checklist

If something isn't working the way it should, the reason is probably one of the following.

- It shouldn't work in the first place. The behavior you expect isn't what you should expect. Maybe you have misunderstood the function of an electronic component or a circuit recipe that you are implementing?
- Software bugs (if a microcontroller is involved in your circuit). Check your code. Use serial output to send messages to your laptop telling you what your code is currently doing. (See Chapter 30.)
- Hardware/software interface bugs: Are you sending signals to the right pins? Are you reading from the right pins? Might you be confusing MCU IC pin numbers and microcontroller board pin numbers? Do you have the right pinout diagram in front of you, for the right microcontroller board? Maybe the board/IC in front of you is oriented differently from the diagram?
- Misinterpretation of the blueprint. Are you reading your circuit schematics correctly? A couple of pitfalls:
  - Where lines are crossing in the schematics drawing, this could either be a place where lines are to be connected or they might be

[1] Probably exactly one – Occam's Razor.

intended to cross without connection (a "bridge"). Usually, connections are a indicated by a black dot at the crossing point, but not everyone adheres to this convention. Make sure you are interpreting line crossings in your circuit diagram correctly. Try to understand the circuit's function. If that's too hard, you may need to look for similar diagrams or ask for someone expert enough to understand the function of the circuit for advice.

  - Remember that connections to ICs in schematics by convention are usually roughly reflecting signal flow from left to right and current flow from top to bottom. IC pin numbers will be indicated in the schematics but will not be arranged clockwise from top left in the schematics as they are on the actual IC (see Chapters 19 and 20).

- Wiring mistakes. Check your schematics and eyeball your implementation very carefully. Particularly circuits on breadboards are treacherous. It is sometimes very hard to see which hole you put a pin in. Use magnification (a loupe) if necessary. For circuits on breadboards, if the circuit is complicated enough, it is often faster to just pull everything off the breadboard and start the circuit anew, rather than to try to find the bug. Overly long wires make debugging extra hard, and not following wiring (color) conventions invites bugs. (See Chapter 23.)

- Orienting a component incorrectly. This mistake is particularly easily made for directional components with two legs, such as diodes and (certain) capacitors. Did you put them into your circuit the wrong way around? (Putting electrolytic capacitors into a circuit the wrong way round and powering it up may result in the capacitor being damaged. Diodes put into a circuit the wrong way round usually does not damage them when powered up.)

- Using the wrong component. Did you mix up some resistors (the resistances matter), or use a capacitor with the wrong capacitance (Farad rating)? Did you pick the wrong kind of transistor? (You can't use NPN and PNP transistors interchangeably, for instance.) Is your motor driver well matched to your motor?

- Broken cables. The copper strands inside your wire have limited elas-

ticity and tensile strength, and they usually break[2] before the plastic insulation material covering them breaks, so you won't recognize that a wire is broken by looking at it. (But you can check by taking the wire out of the circuit and measuring its end-to-end resistance with a multimeter. Broken cables are a big time sink, so it is best not to break them in the first place. See Chapter 23 on cable management.

- Bad solder connections. See Chapter 20 on soldering.
- Loose contacts. Sometimes a plug makes intermittent connection, leading to erratic behaviour of your circuit. Is a connector damaged or does it feel loose? (That is, do you feel very little friction as you connect and disconnect?) Are the metal surfaces in the connectors that are intended to make the electrical connection corroded, covered by isolating material (such as paint), or bent out of place?

  I have seen students make long chains of cable connections from the cables they have so they save walking a few steps to get a cable with the right connectors and of sufficient length. Every connection is a possible reason for failure, so minimize them.
- Broken components (such as microcontrollers). It happens. Sometimes they come out of the factory not working, and sometimes you make a mistake and fry them. Stepper motor drivers are infamously easy to kill. Eliminate more basic bug theories and then try out a new component as replacement.

  Be aware that a component being faulty does not necessary mean that it is "dead" and does not work at all, but it may work intermittently, might show strange behavior, or just some of its features don't work. For a microcontroller, note that you can destroy a single pin (e.g. by running too large a current through it), with the rest of the microcontroller working normally. This is a scenario that is particularly hard to debug because you usually do not suspect this has happened until all other options have been exhausted.

  The Arduino Uno is meant for inexperienced people and has special provisions to protect itself, and its pins are not easily destroyed. But it can happen! Again, such problems may be hard to debug. So it

[2]This happens through metal fatigue; frequent bending, repeated over time, is sufficient for wires to break.

is extra important that you do everything to avoid damaging your components in the first place. Protect them from electrostatic shock, and be careful not to wire them up incorrectly (such as by creating a short circuit, or by confusing VCC and Ground). Prevention here is much better than debugging!

Be aware that broken motor drivers can be outright evil. They may not be "dead" but act outside of specification; for example, they may pass through back-EMF, damaging other electronics. (Use a USB isolator to protect your laptop in case you use any motor bigger than tiny.[3])

- Nasty interference of components or circuit recipes. Sometimes gluing together two recipes just isn't possible. Knowing this in advance is probably beyond your means, but we hint at known scenarios (such as buck converters producing voltage spikes that stepper drivers can't handle) in the technical chapters of this manual. If you have exhausted the above bug theories and can't find a relevant remark about interferences in the manual, talk to an expert.

24.3 Measurement using Multimeters

If a microcontroller is involved, and you trust your programming, some debugging can be done by reading out pins and events and printing them to the serial monitor. As a computer scientist, you are probably comfortable doing this, so please do it where applicable.

An alternative is to measure quantities such as voltages, resistances, and currents using multimeters, and waveforms using oscilloscopes.

Remember, for resistance measurements, your circuit must not be under power. For voltage measurements, informally speaking, you put your multimeter in parallel to the circuit, and for current measurements, you put it in series with your circuit. You cannot measure both voltages and currents in a circuit at the same time using two multimeters: the measurements interfere.

Here are the typical ways of using a multimeter:

- Measuring the resistance of a resistor. If you don't like to read the color marking or are color blind, you can use a multimeter. Remember

[3] Small servos like the SG90 and 28BYJ-48 should be save.

Kirchhoff's laws. If you want to measure the rating of a resistor, take it out of the circuit, otherwise you measure resistance across all paths in the circuit. Also be aware that the multimeter applies a small current when measuring resistance.

- Look for a short-circuit by measuring resistance. Remove the power source – important: we assume there is only one – from the circuit and replace it by the two probes of the multimeter, and measure resistance. If there is a short-circuit, you will measure a resistance close to zero. The directionality of the probes matters: The + (red) probe of the multimeter connects to where the + pole of your power source was connected, and the - (black) probe connects to the connection point of -/GND of the power source before it was removed. Don't get this directionality wrong, or otherwise, in the presence of certain components such as diodes, the measurement would indicate that you are safe even when there is a short-circuit. Be careful, this is not an exhaustive method for eliminating short circuits, particularly if you do not understand what you are doing! Also, make sure you know what your multimeter's reaction to a short circuit and to no connection (infinite resistance) is. In both cases, there may be no numerical Ohm reading on your multimeter's display.

- Measuring voltages. You do this while the circuit is receiving power, by touching to points in your circuit with the probes of your multimeter. Make sure you touch metal. This measures the voltage between two points in the circuit. Mind the directionality of the voltage (indicated by the sign to the number displayed). Remember that you will not get a useful reading on signal lines that switch rapidly. (For this you need an oscilloscope.) Also, if you want to check the charge of a battery with a multimeter, you either need one that has a special battery testing mode, or you need a load attached to the battery. Otherwise, the voltage reading from your multimeter will be unreliable.

- To measure current, the current will have to flow through the multimeter. There are usually multiple current ranges to choose from, and if you don't know what you are doing, you will damage the multimeter (at least burn its fuses). Other than to satisfy our curiosity, we have no real need to measure current in this course. So I suggest not to do it.

24.4 Oscilloscopes

Using oscilloscopes effectively requires a good deal of electronics knowledge. Moreover, they are powerful tools with many functions, and one does not get around reading the oscilloscope's manual. Finally, you can do a lot of damage by incorrectly using an oscilloscope.

There is a short-circuiting trap![4] If you have a USB connection from your microcontroller to your computer and the computer is connected to its (switching-mode) power supply (i.e., not running off battery), you may destroy your computer or at least its USB port in addition to the oscilloscope if you do measurements in your circuit with the oscilloscope.

Just imagine your laptop's power supply is plugged into a mains outlet, you have an Arduino connected to your laptop via USB, there is circuit on a breadboard receiving power from the Arduino's 5V and ground pins. A very typical scenario. Now you take the two probes of the oscilloscope and start measuring signals in the circuit. These will be interesting (voltage) waveforms, otherwise you wouldn't need the oscilloscope in the first place. There are components (such as amplifiers) around that are responsible for this waveform, and this often involves floating ground, voltage potentials above and below it, etc. It may be hard to tell (unless you know what you are doing or measure it) which places in your circuit are at the Arduino's ground potential. We will care about the signal of voltages of one point relative to another one, right? So we can just touch any two points in the circuit and see a signal in the oscilloscope, right? One of the oscilloscope's probes has a low-impedance connection to ground, and if you touch anything in that circuit that has another potential, you have a short circuit through the oscilloscope, the mains power line, your laptop, the USB port, and the Arduino, and you will destroy any or all of them. What actually happens depends. If you are unlucky, this can get very expensive. Don't do it.

This specific example problem would be solved by an isolating power supply (your laptop doesn't have one, it uses a switching-mode power supply), a USB isolator, or special isolating oscilloscope probes (expensive). But there are other pitfalls.

I advise against using an oscilloscope. You should rarely, if ever, have a true need to use one in this course.

[4] See EEVBlog https://www.youtube.com/watch?v=xaELqAo4kkQ.

If you want to use an oscilloscope, you must first have thoroughly **understood all that is said in Chapter 19**. Moreover, you must do this **under the constant supervision of DLLEL staff**. Make sure you clearly inform them of your limited background in electronics before starting.

Chapter 25

High-Level Circuit Diagrams

In your team projects, you will choose the electronic components we will buy, and you will supply power to them and set up the correct wiring for them to work.

Usually, you will not assemble your circuit from basic electronic components such as resistors and capacitors, but we will work with ready-made boards with certain higher-level functions, such as microcontroller (evaluation) boards, sensors, motor drivers (connected to motors), and various components for supplying power (such as power supplies and buck converters).

In general, every component has to be supplied with power, with requires two wires (+ and GND). In addition, most boards communicate with others via signal wires. The most typical patterns are analog or digital input from sensors into the microcontroller and digital (often PWM) output to motor drivers. Both sensors and motor drivers may use bidirectional communication. More advanced boards may want to be communicated with via protocols such as SPI or I2C (see Chapter 31).

You must be mindful that not all components in your circuit necessarily use the same supply voltages: Particularly 3.3V, 5V, and 12V are frequent. Also, not all signals pins use the same logic level (for logic high / one, see also Chapter 26). 3.3V and 5V are typical. Using the wrong voltages will usually destroy your electronics. You need to consult the datasheets/documentation of your components to make sure you made no mistake before you power up your circuit. Always check your wiring, and that you did not my mistakenly switch two cables. Using too high voltages on a pin, or creating a short circuit, may not just destroy your

components but may cause a fire!

When creating your circuit, you may separate the problem of supplying power from the signals wiring problem, though you may find when you work on signals wiring that you need additional components (such as multiplexers or other utility boards) that take you back to the power supplying problem.

In general we wire up our consumers in parallel, and try to avoid using more voltages than necessary. Having multiple power supplies or batteries at different voltages is usually inelegant, bulky, costly, and may cause additional maintenance work.

It is usually good practice to work with a single mains voltage power supply at the highest voltage you need in your circuit, and transform this voltage down to the other voltages needed. Analogously, in a mobile scenario, it is good practice to work with a single battery at the highest voltage needed in your thing.

When doing your signals wiring, be mindful of the logic levels of the involved components. There are components called logic-level converters, but many microcontroller boards are somewhat forgiving when it comes to logic levels, and may to some extent be able to (at least input) either 3.3V or 5V signals. Check the specifications of your components! Using 5V signals on a 3.3V device may destroy it!

Keep in mind that, in almost all scenarios you may encounter, all your components need to have common ground, i.e., have their ground pins directly wired together, to be able to interpret each other's signals. (Voltages are relative and signals are interpreted relative to a ground reference.) But be careful not to create a short-circuit!

Check that your microcontroller board has sufficiently many IO pins of the various kinds you need (for instance, PWM pins, interrupt-enabled pins, and analog I/O pins). If it does not, you may look for another microcontroller board, or use some kind of multiplexing solution.

25.1 Plotter Example

This is the plotter used in the individual project in spring 2025 (see Chapter 6). We provide you with the circuit diagram, shown in Figure 25.1, but let us try to understand how to get to this circuit.

Given to us are the decision that we will use an Arduino Uno microcon-

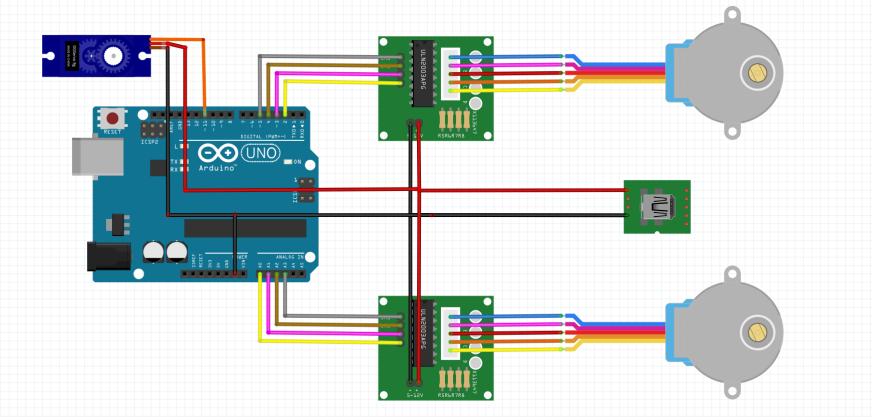


Figure 25.1: High-level circuit diagram of the individual project of CS-358 in spring 2025 (a plotter).

troller board, two unipolar steppers with ULN2003 driver boards, and one SG90 servo (for lifting and lowering the pen). All these components run at 5V power and use 5V logic levels, so we will not have to worry about logic-level conversion. We want to program the Uno through its USB port and supply power through USB.

The Uno, when it is powered through USB, can supply 5V power to other components through it 5V output pin (and a GND pin). However, it can only supply very small currents, not sufficient for motors. We need to supply our three motors differently.

Here we choose to do this through an additional USB cable and a (micro-)USB breakout board that gives us access to the 5V and GND lines of the USB cable.

All compoents have to have common ground – we wire the ground lines together. It is in principle possible to create a short-circuit if there is a potential difference between the two ground lines of the two USB cables, but not if they are fed by a USB multiplexer or two USB ports of the same laptop. (Or even the USB ports from two laptops, if they are on battery.)

Thus, the power wiring is straightforward – we supply all the high-current consumers (servo and ULN2003 motor drivers) through the USB breakout board. (These consumers are high-current by the standards of what an Arduino Uno can supply on its 5V output pin, but not by the standards of USB.)

The motor cables have specialized plugs that go into a specialized port

on the motor drivers, no mistakes are possible there.

The ULN2003 motor drivers have four 5V level digital inputs each that have no special requirements: any Arduino Uno digital pin will do. The servo has one PWM digital input (usually, that cable is yellow).

We can use any nine Arduino Uno GPIO pins for the signal lines, with the constraint that the servo signal line must support PWM (the UNO has six such pins). The Uno's analog in pins can also be used as digital output pins (but not as analog output pins).

25.2 Self-playing guitar example

This example is based on a previous team project made in the course, see Chapter 60. The goal is to build a self-playing guitar controlled by an Arduino Uno. The guitar has six strings, and for each string we need one bipolar stepper motor and one small servo motor to play it.

Givens:

- We must use an Arduino Uno as the sole microcontroller.
- We must control six Nema 17 steppers via A4988 motor drivers (at 12 V).
- We must control six SG90 servos at 5-6V (6V makes the servos go faster).
- We must use a 12V PSU unit as a power supply. How many Amps do we need?
- We may use one or multiple buck/step-down converters.
- We may use one or multiple Arduino CNC shields.
- We may use a PCA9685 I2C PWM multiplexer board.

Read up on the relevant topics such as PWM, I2C, bipolar steppers, etc., in Chapters 31, 26, and 39 of the manual; google the mentioned components.

Our goal is to create a wiring diagram for the electronics of the project.

We first consider the power wiring. We chose to use two CNC shields to control the six stepper motors. Each CNC shield has to be powered,

separately, using 5V (which is also the logic level) *AND* 12V (for powering the motors). We will drive the Arduino Uno and the servos at 6V, which is the voltage preferred by the servos. The Uno can be powered through its Vin pin using at least 5V and at most 10V. The 5V input voltage of the CNC shields will be taken from the 5V output pin of the Uno. Thus we need one buck converter to convert 12V to 6V, and this way can power the entire circuit through a single 12V power supply.

How many Amps does the power supply need to be able to drive? The determining factor are the steppers; each of them have two coils that may take up to 1.5A, but never the maximum on both coils at the same time. It is reasonable to calculate 2A per stepper. This would take us to 12A for the steppers alone. It is actually hard to get a 12A power supply, but we need less if we do not operate all the steppers at once. For the remaining components – the Uno and the servos, we can calculate 1A. The maximum current draw of one such servo is about 0.25A. It is reasonable to pick a 6-8A power supply, and if we play one string at a time (or even two or three), this would give us sufficient current for the task.

As for the signals, we need at least 12 GPIO pins (without any special constraints, such as PWM-capability) for the CNC shields, two (step and direction) per stepper. In addition, we need 6 PWM pins for the servos. Using the six analog input pins as digital output pins (which is possible), the pins on the Arduino Uno are actually sufficient. However, it leaves no space to expand, and for practical operations, a few more pins are needed to connect to the CNC shields.[1]

Thus we use a PCA9685 PWM multiplexer board and connect the servos there. It needs just two signal pins and a ground pin for I2C communication.

The finished circuit diagram (without connects to the enable pins on the CNC shields) is shown in Figure 25.2.

[1]For instance, one should connect the enable pins of the CNC shields to power off the steppers programmatically. Otherwise, they are always receiving power, even when they are not moving, and can run very hot.

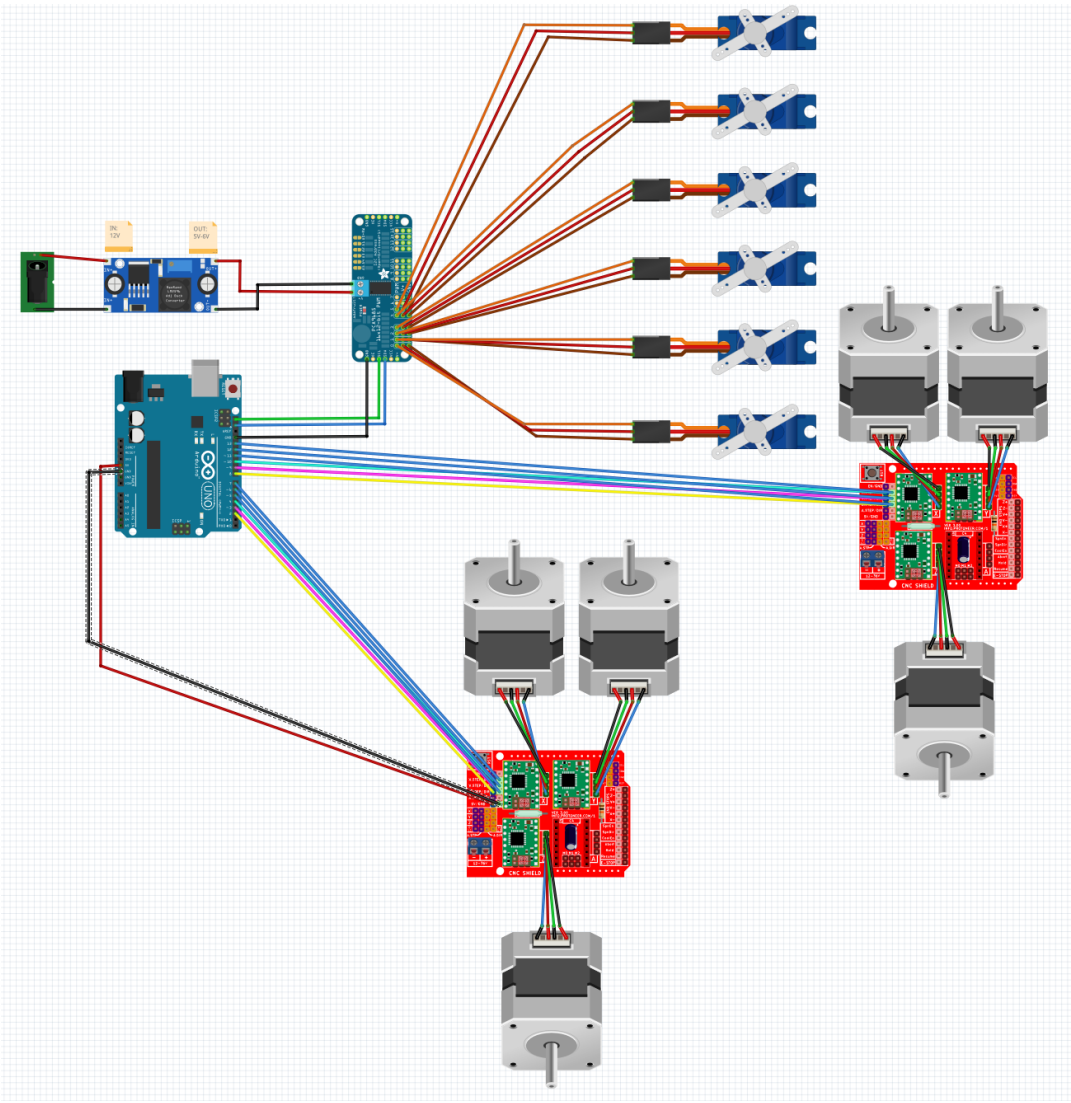


Figure 25.2: High-level circuit diagram of the self-playing guitar example.

Part IV

Microcontrollers and Programming

Chapter 26

Digital Signals

As we build up to using microcontrollers, we need to get comfortable with the relationship between general electric circuits and digital electronics. As computer scientists, we are used to working only with digital electronics, and we may forget that this is not all there is. We may also work with the intuition that the flow of information and causation through logic gates corresponds to the flow of current. However, in reality, the connection between current flow and reading and writing data is tenuous at best.

Looking at it low-level, in electric circuits, there is no notion of reading and writing (data); there is only the circuit and, if we like, the flow of current. In digital circuits, there is a higher-level abstraction of the flow of information, but how does this translate to electronic circuits in which only a part has a supposedly digital interpretation?

In general, our thing will involve digital and analog signals, and wires that are best not thought of as carrying signals at all. Microcontrollers are designed to interface with electronic circuitry that isn't necessarily digital. Some of a microcontroller's connection pins are labeled digital, others analog, and some neither. And yet we may use a digital pin to power an LED. Let us get a bit more comfortable with this.

26.1 Logic Levels and Logic Gates

You certainly know from an earlier course that, on an abstract level, the logic circuitry of computers is built from logical gates such as AND- OR- and NOT-gates, which can all be constructed from a single kind of gate, the

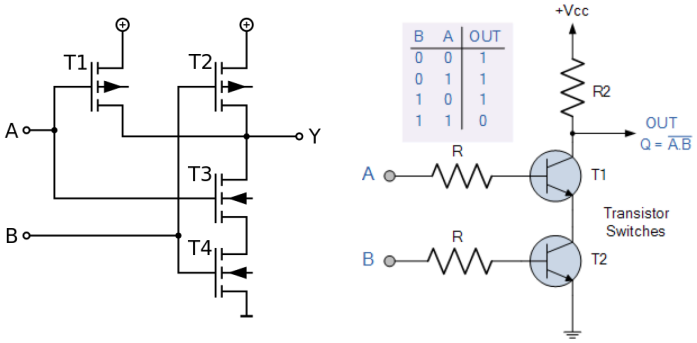


Figure 26.1: A fan-in two NAND gate in CMOS technology (left) and using bipolar junction transistors (right).

NAND-gate.[1] From these we can build larger units such as adders and even entire CPUs. Let us look at how NAND-gates are built up from transistors (and resistors) using two different families of transistors, MOSFETs[2] and binary junction transistors (BJTs). These fan-in two gates have more than the three connections by which they are usually shown in logical circuit diagrams. The currents from the input gates A and B **do not** flow to the output gate (labeled Y and Out, respectively). The NAND gates of these two technologies are incompatible with each other and cannot be used together in logic circuits without suitable translation technology.

Still, when we connect digital components (of compatible technology), the higher-level view of information flow is valid. This is achieved by their compatible design, by connecting them in ways valid under this design, and conventions regarding the meaning of voltage levels (logic levels). If we violate any of these, the information flow abstraction breaks down. Even if we agree that high voltage corresponds to high logic level, connecting an LED to the output of such a logic gate that would compute logic HIGH in a digital circuit does not necessarily – depending on technology – light up the LED.

When we get to microcontrollers in later chapters, we see that they offer a number of digital and other pins to which we can, but do not have to[3],

[1] NOT(A) := NAND(A, A); AND(A, B) := NOT(NAND(A, B)); OR(A, B) := NAND(NOT(A), NOT(B)).

[2] Note that the CMOS transistor uses two each of two types of transistor – p-type and n-type. See https://en.wikipedia.org/wiki/CMOS for more.

[3] So leaving such a pin unconnected does not lead to an open and incomplete circuit;

connect other digital components. These pins usually can be configured to send or receive digital signals. This isn't obvious functionality, digital pins of other components (such as sensors) that we connect to these microcontroller pins usually have dedicated input *or* output roles, as have the input and output lines of our NAND-gates.

How such GPIO ports are implemented is beyond the scope of this manual[4], and, in fact, what works and doesn't depends on the microcontroller. For example, the Arduino Uno, which has been designed for learners and even children, is designed particular robustly, and may survive abuse that may damage another microcontroller. It is recommended to implement proven recipes that you find online.

26.2 Logic-level Conversion

As mentioned, each digital electronics component follows some convention for which voltage range corresponds to HIGH/1/true and which voltage corresponds to LOW/0/false. We are talking of a range, not an exact voltage, because nothing, starting from the power supplies, is exact in practical electronics. Still, LOW means close to 0V, and HIGH usually means one of two things: close to 3.3V or close to 5V. It is essential to know with which logic level your component (such as a microcontroller, sensor, or motor driver) works. You can find that in the component's datasheet.

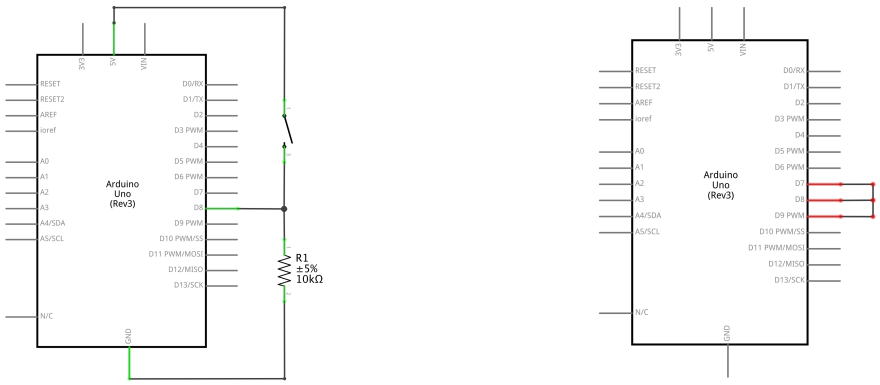
The trend is to go to components with lower logic level, since this is connected to energy savings.

Some mix of logic levels may also be possible. For example, the Arduino Uno internally works at 5V logic level, and outputs 5V HIGH signals, but has both 3.3V and 5V Vout pins and reads digital signals with a range from below 3.3V to 5V for logical HIGH: Thus it can read from both digital components that use 3.3V logic and components that use 5V logic. The Uno can read from components with 3.3V logic level, but to write to them, we need conversion of 5V signals to 3.3V signals.

If you want to connect two components that operate at different logic levels, you need to perform logic-level conversion. There are two main ways of doing it – using a voltage divider or a special logic-level conversion board. Note that a voltage divider only needs two resistors of appropriate

see https://en.wikipedia.org/wiki/Three-state_logic.

[4]See https://en.wikipedia.org/wiki/General-purpose_input/output.



(a) (b)

Figure 26.2: Using a pull-down resistor to make reading a switch work (26.2a) and a bad logic gate (26.2b). Power supplies are not shown.

resistance to build up the voltage divider circuit, but it will only allow logic-level conversion in a single direction. Voltage dividers are covered in Section 21.3. For instance, to translate a signal from 5V to 3.3V, you connect your signal source to V_{in} of the voltage divider, your 3.3V level receiving component to V_{out}, and choose $Z_1 = 1k\Omega$ and $Z_2 = 2k\Omega$.

26.3 "No Connection" is Different from Digital Zero

A mistake that computer scientists often make is believing that *tertium non datur*, that any digital read results in zero or one on a digital signal line. This is not true. Usually, a digital signal line is considered to have value 0 when the signal line is at ground potential (0V to GND). An open connection as in Figure 19.4b is not the same as if this wire is connected to ground – the digital value is undefined, rather than 0, and a digital input pin on your microcontroller will read a random value.

This has many consequences for you, and if you don't respect this, you will have bugs.

Take for instance a push button or switch that opens or closes a circuit. You can't just wire this button up with your microcontroller, because the read will be nonsense when the switch is open. Figure 26.2a shows the

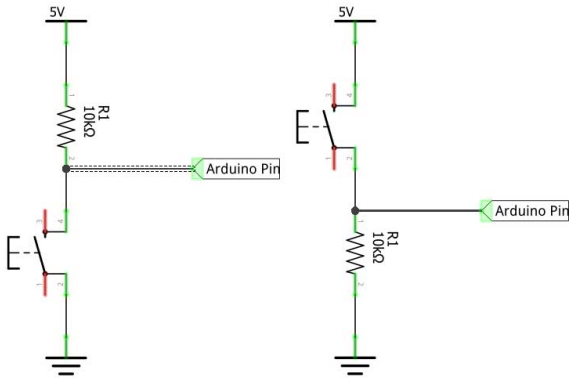


Figure 26.3: A pull-up resistor circuit (left) and a pull-down resistor circuit (right) for reading out the state of a switch.

solution to this, using a so-called *pull-down resistor*[5]. When the switch is open, the I/O pin of the microcontroller (here pin 8) reads 0 because of the connection to ground. When the switch is closed, the I/O pin is directly connected to 5V and reads 1. There is no short-circuit from 5V to ground because of the resistor. 10kΩ as shown is an appropriate value for this resistor. If we removed the connection to ground via the resistor, the I/O pin would read a random value when the switch is open. Figure 26.3 shows pull-up and pull-down circuitry next to each other. Use the former to get default-HIGH and the latter to get default-LOW (when the switch is open). So, if you want your button/switch to turn the input pin from (default) LOW to HIGH, you need a pull-down resistor.

A somewhat related issue is demonstrated in Figure 26.2b. Suppose we write digital values to pins 7 and 8 and read pin 9. If at least one of 7 and 8 is 1, then pin 9 reads one, and otherwise it reads 0, right? So this is a logic OR gate, right? Wrong. If, for instance, pin 7 has value 1 and pin 8 has value 0, we short-circuit these two pins. What really happens is a bit more complicated because of the the electronics of digital read/write pins, but in any case, you do not get a logic gate as simply as that. Of course, you will probably never want to build a logic gate like this because you can very conveniently do such calculations inside the microcontroller.

If you connect two digital output pins of your microcontroller by a direct wire connection, you do not "OR" or "SUM" the signals. Instead, if you set

[5]There are also pull-up resistors between your I/O pin and the logic-level voltage (3.3 or 5V), which give you a logical 1 as default.

one output to 1 and another to 0, you essentially create a short-circuit.

26.4 Pulse Width Modulation (PWM)

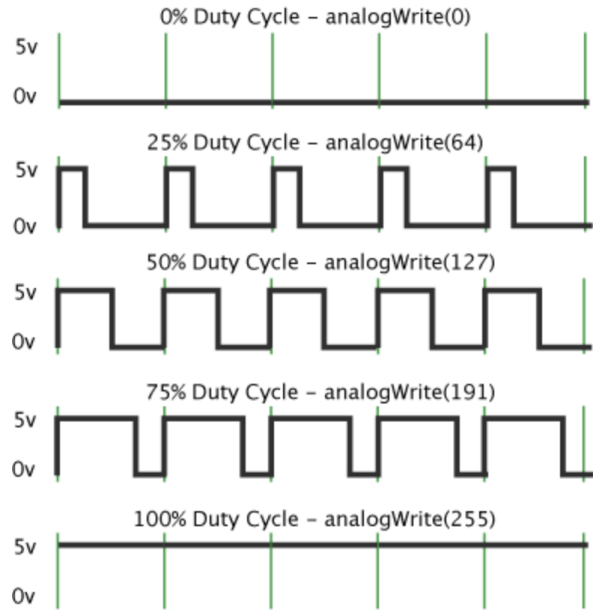


Figure 26.4: PWM signals with various duty cycles (but equal frequency), and the analogWrite() commands to generate them (in Arduino IDE programs).

Pulse Width Modulation (PWM) signals are digital signals that are achieved by rapidly switching between high and low voltage levels. There are two main parameters to a PWM signal: the *frequency* of HIGH-LOW cycles and the *duty cycle*, which is the fraction of time during which the signal is at HIGH potential. For illustration, Figure 26.4 shows graphs of PWM signals with different duty cycles.

PWM signals can serve multiple purposes. Some components (such as stepper motor drivers) interpret PWM signals as the square wave signals that they are, and switch components on and off rapidly. PWM signals can also simulate analog output, as components that react more slowly can be viewed as observing a smoothed, averaged version of the square wave – essentially a constant signal at the voltage that is the voltage of logical

HIGH times the duty cycle (the fraction of time during which the signal is a HIGH level) of the PWM signal. So, for example, a PWM signal with 50% duty cycle and a 5V logic level will behave like a constant 2.5V voltage to some components. This can be used, for instance, to dim the light of an LED.

Chapter 27

Microcontrollers

A microcontroller (aka *microcontroller unit, MCU*) is a small, highly integrated, programmable computing unit. It is usually realized as a single chip that includes at least one processor core, memory, and hardware support for a potentially large number of different forms of I/O, from very low-level I/O through which the MCU becomes part of basic electronic circuits (GPIO pins), to protocol standards of various levels of sophistication (in particular, I2C, SPI, and CAN) by which the MCU can relatively directly connect to complex devices.

MCUs are designed for embedded applications, where they directly interact with and control electronic components other than those usually seen with computers (such as keyboards, mice, and screens). In contemporary computers, practically the only way to interface with (say, homegrown) peripherals is through USB-ports. If we want our computer to interact through USB with an electronic circuit of our making, we need translation from the USB protocol and protection of the computer from various mistakes we may be making. The best way to do this today is, actually, through an MCU *development board* with a USB port, such as an Arduino Uno. MCUs make interfacing with electronics, even very low-level components such as resistors, capacitors, diodes, and transistors, easy.

Compared to the computers we are used to today, MCUs often have very limited processing power and memory (typically, no more than about 250 MHz clock speed and no more than a few MB of RAM). This is not because MCUs are outdated or have received limited industrial interest – quite the opposite. In an industrial context, we attempt to match MCUs as precisely as possible to their use. They often are part of mass-produced

products, which are to be produced as cheaply as possible, using MCUs whose capabilities are precisely matched to the needs of the application, wasting no complexity and no money.[1] MCUs are often used in low-power applications, where it is important for the power consumption of the MCU to be minimal. Finally, the typical use cases of MCUs are just very different from those of (personal) computers with their general-purpose operating systems. If we, for instance, had to generate a precisely timed signal, we would find this easy to do with even low-end MCUs, but impossible to do with even the most highly specced gaming PCs or cloud servers.

Unlike computers, MCUs typically do not run an operating system, allowing the program code direct access to all resources. In technical applications, MCU code often has to satisfy stringent real-time requirements, where taking just a little too long to complete a computation and to trigger an action could cause a major catastrophe, like a nuclear reactor melting down. Using general-purpose operating systems such as Unix or Microsoft Windows in such a context would be criminal folly. Industrial MCUs like those of the STM32 family come with software development environments that help you develop precisely timed software (as well as allowing you to calculate precise power consumption). There are real-time operating systems (RTOS) specially for MCUs and time-critical applications that can give hard real-time guarantees even with multiple threads running concurrently, but they are beyond the scope of this manual.[2]

In a learning environment, we usually do not work with a microcontroller IC directly, but with a microcontroller board that mounts the IC and provides IO ports, limited protection against mis-use, and (often multiple) easy ways to supply power to the MCU. It is important to remember, though, that the very point of MCUs is to be as complete as possible in as small, cheap, and power-saving a package as possible. The MCU IC provides all the functionality you need in a completed product that makes use of MCUs. The board – which for that reason is usually called a development board or an evaluation board in a professional context – is really just there to provide additional protections and to supply power while the product isn't ready and the PCB on which the MCU with other product-specific components will go isn't ready.

General-purpose IO (GPIO) ports (pins) turn the MCU into just an elec-

[1] There are MCUs that cost just a few cents when bought in large numbers – though not those we will use.

[2] Though not beyond the scope of this course, if a team were interested in this.

tronic component like many others, to be made part of electronic circuits. They enable communication between the MCU and other devices, such as sensors and actuators. We distinguish digital and analog IO. Digital pins are used to send or receive digital signals. These pins can be configured as either inputs or outputs. Typical uses are to interface with devices that support digital communication protocols, which includes other MCUs, sensors, and motor drivers. Analog (input) pins, on the other hand, are used to read analog signals, which are digitized as multi-bit numbers (the resolution depending on the analog-to-digital converters inside the MCU). These signals may represent physical phenomena such as temperature, distance, or frequency, depending on the source of the signal, and are typically read from sensors.

MCUs usually have pins that can "output" PWM signals (see Section 26.4), with the frequency and duty cycle parameters set by the program code. This is achieved with hardware support, that is, the is no code doing the on-off switching running in the core, but the MCU contains a separate hardware component that generates the signal.

Using PWM to simulate analog output isn't perfect, and for some applications it is not good enough. For that purpose, some MCUs have digital-to-analog converters (DAC) to produce true analog signals.

MCUs are designed to interact and exchange data with other components, such as sensors. For this purpose, a number of interface standards and protocols are used in the industry, such as SPI, I2C, and CAN. Other standard communications methods and interfaces include USB, UART, Wifi, and Bluetooth.[3] MCUs often have native hardware support for several of these (see Figure 28.1), so using these in microcontroller applications is easy and does not take compute cycles aware from the programmed tasks.

27.1 Recommended Videos

https://www.youtube.com/watch?v=F321087yYy4
DigiKey
"Introduction to RTOS Part 1 - What is a Real-Time Operating System (RTOS)? | Digi-Key Electronics"

[3] See Chapter 31 for more on using these interfacing standards and protocols.

Chapter 28

Choosing a Microcontroller

This chapter acts as a guide to picking, based on the needs of your project, one of the microcontroller boards available to us in the course. In the following, we concisely present these models. This chapter cannot cover all the details of their specifications. Please search the Web for the datasheet and documentation of the MCU you want to use, and look out for versions, as some microcontroller boards have multiple revisions.

28.1 Microcontroller Families

There are many different models of MCUs available on the market, each with their own features and capabilities. For projects like ours, evaluation/development boards are convenient. These act as *breakout boards*, allowing solderless connections to the legs of the MCU IC via Dupont-style connectors, but usually also add

- protection circuitry, which increase the survivability of the MCU under mis-use,
- voltage conversion circuitry, to make it easier to supply power, and
- a programming interface, usually with a USB port.

Manufacturers usually produce multiple versions for their MCUs, which share many abstractions (such as their machine language and memory model) but which differ in their features (such as which forms of I/O they provide hardware support for). If there are multiple MCU models in a

family, this often does not mean that one is best or newest-generation: You may have to pick one based on trading off features you don't need for others you do.

Among the most popular MCU families, we have (1) those of the Atmel AVR family, (2) the ESP32 family, made by Expressif Systems, and (3) the STM32 family, made by ST Microelectronics (a company headquartered in Switzerland). The ESP8266 is a predecessor model of the ESP32 family.

- The Atmel AVR family originated as a student project in Norway, and are generally the least competitive family in terms of their features and performance statistics, but they are used in Arduino boards such as the Arduino Uno and the Arduino Mega, and are popular for being popular. They are the most widely used MCUs in maker projects, and information on them is most widely available.

- ESP(8266 and -32) are designed for IOT (Internet of Things) applications, and usually have Wifi and Bluetooth support. They also have relatively high computational power.

- The STM32 family are best suited for technical, industrial applications. They are not intended for hobbyist, and their features and native tool chain (including the STMCubeIDE) exude reliability and have features that are indispensable for low-level technical uses. They usually have large numbers of supported I/O interfaces (including multiple I2C, SPI, UART, and CAN ports), multiple ADC and DAC ports, and many timers. The hardware is internally highly configurable, and they have a very sophisticated infrastructure for precise timing of every aspect of the MCU's operation.

All of the MCUs available to us for this course, and covered in this chapter, can be programmed with the Arduino IDE, though to have access to certain advanced features, you may need to use a family's native IDE. This is particularly true for the STM32 family.

28.2 Microcontroller Comparison Chart

When choosing a microcontroller, you have to ask yourself which features are most important to you. Each microcontroller board has its advantages

Board name	Arduino Uno R3 (clone)	Arduino Mega 2560 (clone)	NodeMCU ESP8266	Wemos D1 R32	AI-Thinker ESP32-CAM	STM32 Blue Pill	STM32 Nucleo64 F401RE
MCU family	Atmel AVR	Atmel AVR	–	ESP32	ESP32	STM32	STM32
MCU	ATmega328P	ATmega2560	ESP8266	*-WROOM	*-S	*F103C8T6	*F401RET6
Cores	1	1	1	2	2	1	1
Word len (bits)	8	8	32	32	32	32	32
Speed (MHz)	16	16	80	80-240	160	72	84
Flash Mem (KB)	32	256	4000	448	4000	64	512
SRAM (KB)	2	8	64	4520	4500	20	96
EEPROM (KB)	1	4	-	-	-	-	-
Vin (V)	7-12 or USB	7-12 or USB	4.5-10	9-24 or USB	5	5	7-12
Vout (V)	3.3 and 5	3.3 and 5	3.3	3.3 and 5	Vin	3.3 and 5	3.3 and 5
Logic HIGH (V)	5 (read >3)	5 (read >3)	3.3	3.3	3.3	3.3	3.3
Digital I/O pins	12/14	54	17		1/8	29	50
PWM pins	6	15	4	16	1/8	15	>22
Interrupt pins	2	?	17	all GPIO	0/7	all GPIO	all GPIO
Analog in pins	6(@10 bit)	16(@10 bit)	1	18(@12 bit)	0/7	10	16(@12 bit)
USB port type	USB-b	USB-b	micro-USB	micro-USB	–	micro-USB	mini-USB
USB to serial	CH340	CH340	CP2102	CH340	–	in MCU	ST-Link
Wifi 802.11	–	–	b/g/n	b/g/n/e/i	b/g/n	–	–
Bluetooth	–	–	–	v4.2, BLE	v4.2, BLE	–	–
UART/TTL	1x	4x	1x/2x	3x	1x	3x	4x
SPI	1x	1x	1x	3x	–	2x	3x
I2C	1x	1x	1x	2x	–	2x	3x
CAN	-	-	-	-	-	1x	1x
Timers (@bits)	2@8 (+1@16)	4	1@23	4@64	(4)	4	6@16+2@32
Size (mm*mm)	69x54	102x54	25x49	69x54	27x41	23x53	83x70
Other	–	–	–	2x DAC@8	camera micro-SD	–	–

Figure 28.1: Microcontroller comparison chart

and disadvantages. Some features, such as having a camera, are rare and quickly determine which microcontroller you must choose. Also have a look at the other electronic components that you want to use in your thing. How many IO pins of each kind do you need? What is their logic level? Particularly for sensors, look at their interfaces (such as I2C or SPI). Does the microcontroller board have sufficiently many GPIO/analog/PWM/interrupt-enabled pins?

In general, no microcontroller board will satisfy all your needs. Remember that some features can be achieved by programming. For instance, there is a library SoftwareSerial that gives you additional TTL serial interfaces on normal digital IO pins. Other features and interfaces, such as Bluetooth or CAN-bus support, are offered by specialized boards. There are de-multiplexing boards that give you additional IO pins, and boards that offer additional PWM pins. There is also no reason why you cannot use multiple (different) microcontroller boards with different strengths and have them communicate and work together inside your thing.

Figure 28.1 distills many of the relevant specs down into one table. Here are a few important notes, though. First, this table is meant to give you a quick overview and cannot replace reading datasheets to be sure. When you do so, be sure to distinguish among the specs of the microcontroller IC and the microcontroller board. For instance, the ESP32-CAM has a very capable MCU (one very similar to the Wemos D1 R32, which provides many GPIO pins). However, the ESP32-CAM board passes through only very little of this functionality to its pins – not because it is a bad board but because most of the MCU's IO pins are used to communicate with the on-board camera and micro-SD card reader. Similarly all the ESP boards use one of its timers for Wifi, and thus this timer is not available for your uses.

Most of the rows of the tables should be self-explanatory. The flash memory stores the compiled code; the runtime state of your programs is stored in SRAM. (So, for instance, the Arduino Uno has only 2KB for variables and data structures.) Logic HIGH and represents the voltage of logic level HIGH – for the Arduino Uno, the logic level is 5V, but it understands logic level 3.3V when reading from other components. USB to serial refers to the USB protocol chip used on the microcontroller board. In general, you need to install a USB driver for that chip on your computer.

In order to keep the table readable, some details, quirks, and caveats are not shown. For instance, the Atmel AVR MCUs have a third kind of

memory, EEPROM, which is nonvolatile (persists when the microcontroller is powered off), but unlike flash memory, you can write it from your microcontroller program. For another example, the ESP32-CAM spec allows for 3.3V Vin, but we show only 5V Vin, because supplying 3.3V is known to cause brownouts on Wifi startup.

The most important kind of limitation that cannot be shown in such a table is the fact that the pins of microcontroller boards usually have multiple purposes, and there are constraints on what can be done when. For instance, on the Arduino Uno, digital pins 0 and 1 are used for TTL serial communication and directly reflect the serial communication via USB. If you use them like normal GPIO pins, your thing will malfunction at times. Where it makes sense, we show the availablity of pins in the form x/y, where x is the number of pins available without limitations and y is the number of pins available only at times or with limitations. (See the following sections for more details.)

Note that the NodeMCU ESP8266 and the STM32 Blue Pill are substantially smaller than the other boards (see the dimensions in the Figure 28.1).

28.3 ATMEL AVR MCU Boards

28.3.1 Arduino UNO R3

The Arduino Uno R3 is an open-source design with a large community of users and developers. Figure 28.2 shows its pinout diagram.

There are multiple ways in which you can supply power to the Arduino Uno: through the USB cable, the VIN pin (providing the board with a recommended range of 7 to 12V), and a DC in plug (the black plastic box with a round plug shown in the top left corner of the board in Figure 28.2). Be careful when using the VIN pin to supply power – too high a voltage will damage the board. Once the board receives power, it can also act as a power source to other low-power components through its GND, 3.3V, and 5V pins. The multiple GND pins of the board are all connected to the same ground and can be used equivalently. The maximal output current is very limited and not suitable for powering motors – if you do so, you will damage the microcontroller board.

We are working with Uno clones. These are, of course, completely le-

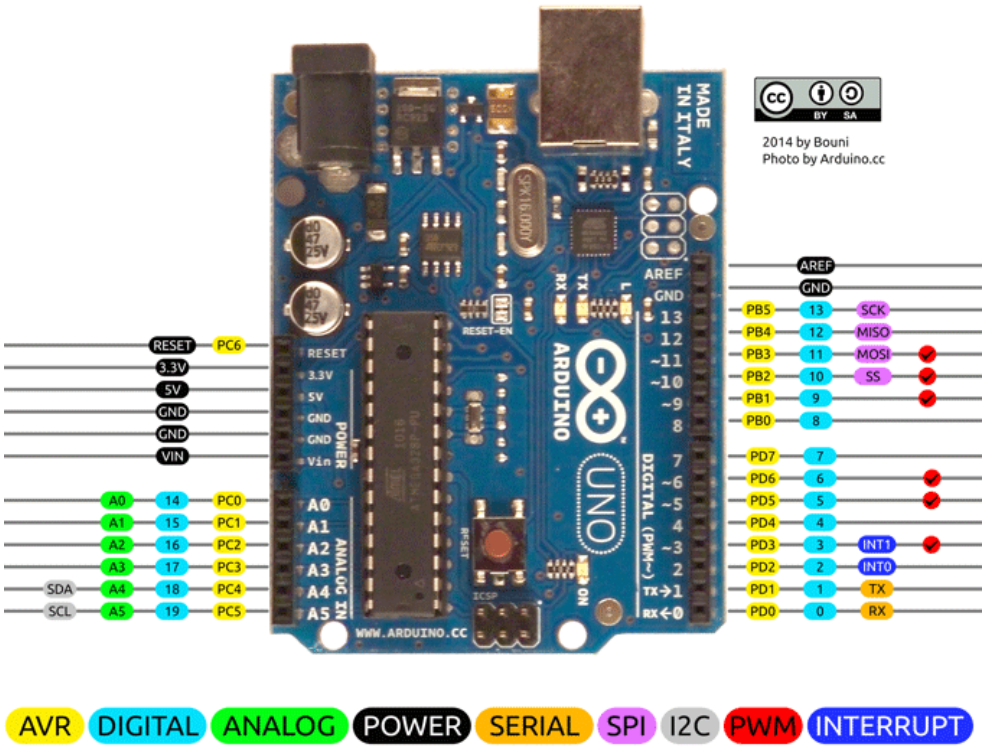


Figure 28.2: Arduino UNO rev3 pinout.

gal – this is an open-source design meant to be as available as possible. They are also completely equivalent in behavior and performance to the board produced by the Arduino company – with one exception. Unlike the Arduino-branded boards, the clones use a CH340 USB to TTL serial chip, which may necessitate installing a driver on your computer.

Recently, the Arduino Uno R4 has been released. This is not an improved version of the R3, but a board very similar to the Wemos D1 R32.

28.3.2 Arduino Mega 2560

TODO – for specs and pros and cons, see Figure 28.1.

The Arduino Mega is essentially an Arduino Uno with more IO pins.

28.4 Expressif MCU Boards

28.4.1 NodeMCU ESP8266

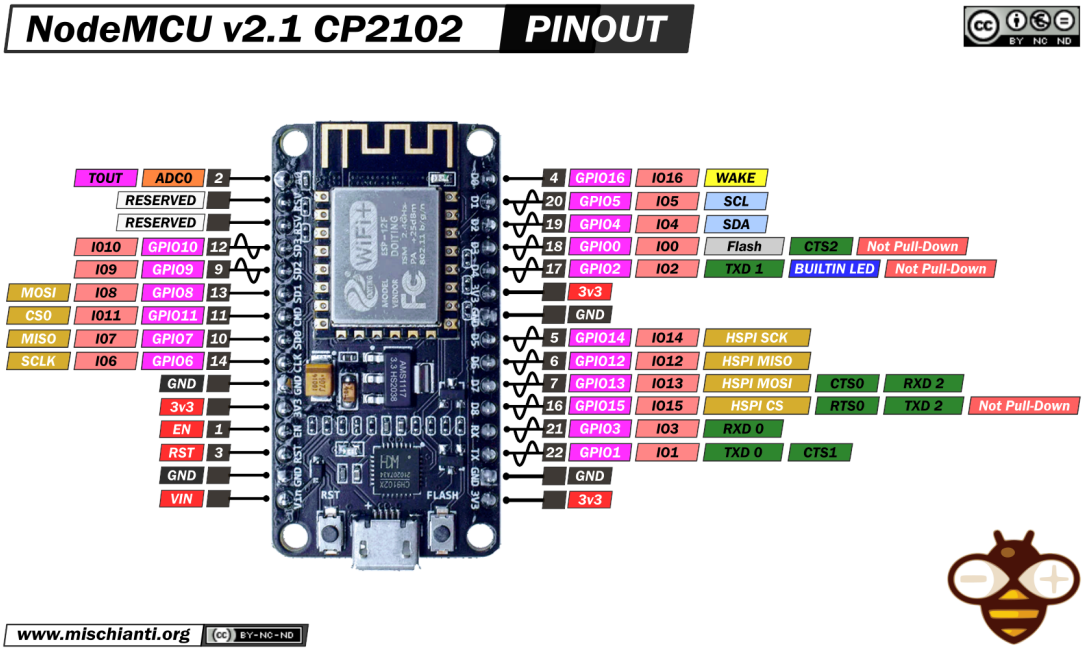


Figure 28.3: The pinout of the NodeMCU ESP8266.

The pinout of the NodeMCU ESP8266 board is shown in Figure 28.3. Note that the pin numbers on the board are different from those shown in the blue GPIO boxes left and right of the board. The latter numbers are those that you should use in your code. For instance, the pin labeled D5 on the board is addressed as pin 14 in your code.

28.4.2 Wemos D1 R32

TODO: Add pinout diagram.

For specs and pros and cons, see Figure 28.1.

If you like the Arduino Uno but need more computational power and/or internet connectivity, consider this board.

The Wemos D1 R32 looks very much like an Arduino Uno, and Arduino Uno shields can be mounted on it. However, you cannot run your Uno code unchanged on it because the pinout is different. (See the Nucleo64 board for one that is fully compatible with the Arduino Uno R3.)

28.4.3 ESP32-CAM

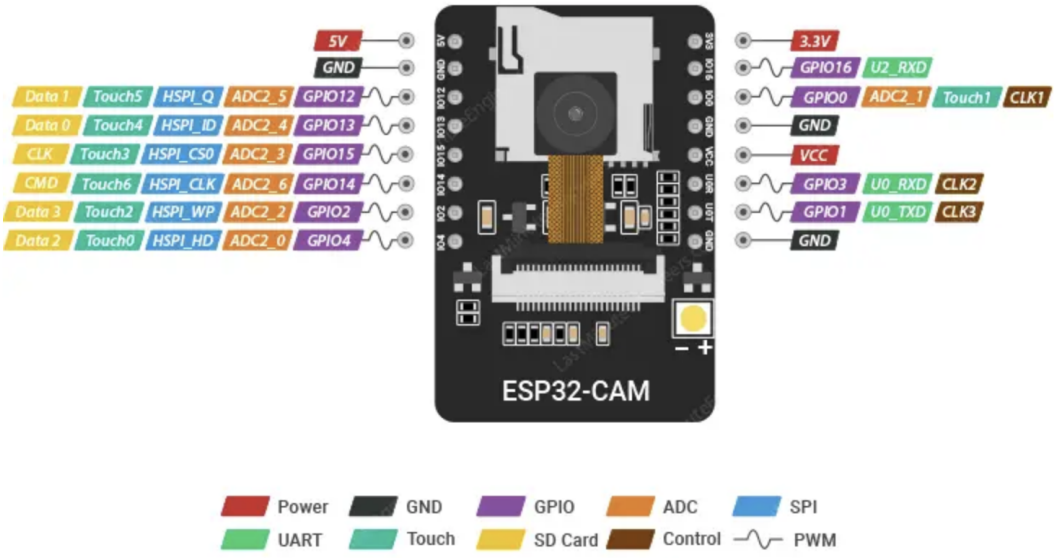


Figure 28.4: The pinout of the ESP32-CAM.

The ESP32-CAM is a development board based on the ESP32 MCU. Its main feature is a camera module, which can be configured to capture images and videos. The camera is an ov2640 with a resolution of 1600 x 1200 (2 megapixels) at 15 frames per second. Additionally, the ESP32-CAM has built-in Wi-Fi and Bluetooth connectivity, allowing it to wirelessly connect to the internet and other devices, and a micro-SD card reader and writer, which for instance can be used to store photos in a surveillance camera application, of a pre-trained computer vision model.

Its main practical limitation is the small number of available IO pins. There are ten labeled GPIO pins; however, most have other uses that severely limit their use as GPIO pins (see Figure 28.5 on when pins can be safely used). The ESP32-CAM does not have a USB port to connect it directly to

Label	GPIO	Safe to use?	Reason
D0	0	⚠	must be HIGH during boot and LOW for flashing
TX0	1	✗	Tx pin, used for flashing and debugging
D2	2	⚠	must be LOW during boot, cannot be used when microSD card is present
RX0	3	✗	Rx pin, used for flashing and debugging
D4	4	⚠	Connected to the on-board Flash LED, cannot be used when microSD card is present
D12	12	⚠	must be LOW during boot, cannot be used when microSD card is present
D13	13	⚠	cannot be used when microSD card is present
D14	14	⚠	cannot be used when microSD card is present
D15	15	⚠	must be HIGH during boot, prevents startup log if pulled LOW, cannot be used when microSD card is present
RX2	16	✓	

Figure 28.5: Safe use of ESP32-CAM IO pins.

a computer. In order to upload code, we need to wire it up with an FTDI adapter board – a USB-to-serial translator.

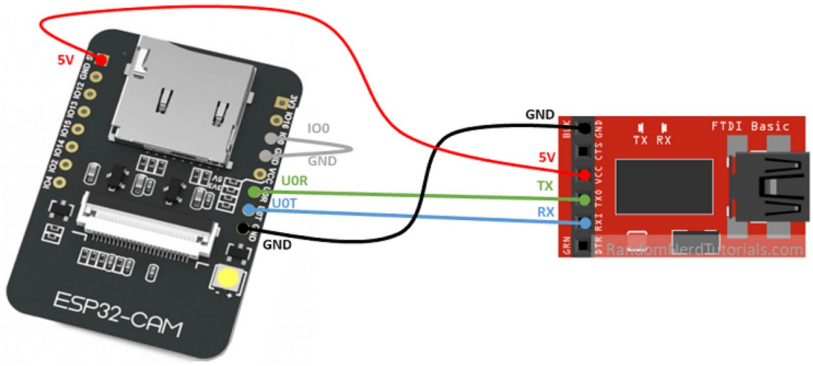


Figure 28.6: The setup to upload code from a computer to an ESP32-CAM using an FTDI adapter.

As shown in Figure 28.6, we connect the ESP32-CAM to USB and the computer via the FTDI adapter. Be sure not to forget the jumper connection of `GPIO 0` to `GND`. FTDI adapters have a jumper on board that allows you to select 3.3V or 5V. Make sure the jumper selects 5V power. Once you have

wired up your microcontroller as shown, you can upload your code via the Arduino IDE as usual once. Once you are done uploading code, you must **remove the jumper** cable connection between `GPIO 0` and `GND` and press the `RST` button to run your new code.

28.5 STM32 MCU Boards

28.5.1 STM32 Blue Pill

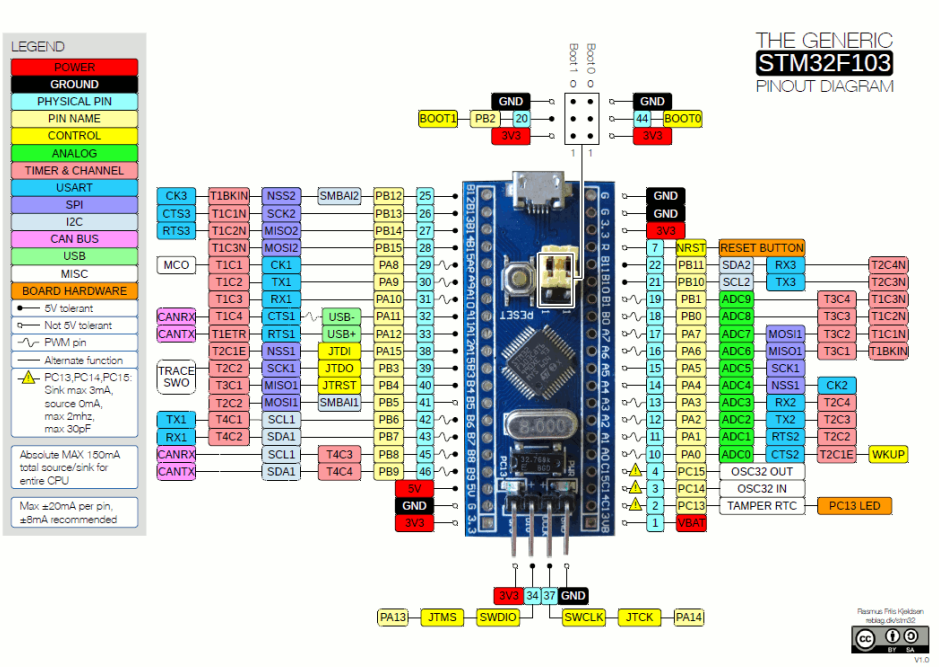


Figure 28.7: The pinout of the Blue Pill.

TODO: For specs and pros and cons, see Figure 28.1.

The Blue Pill has a USB port which is directly connected to the MCU, without any separate hardware support for running the USB protocol. The protocol is intended to be run by your MCU. Since the board ships without any preinstalled program ("firmware"), you first need a separate programmer (such as an FTDI USB-to-serial board) to upload USB driver code to your MCU, before you can subsequently program it via USB.

28.5.2 STM32 Nucleo64 F401RE

TODO: For specs and pros and cons, see Figure 28.1.

This board has a set of female connectors that are fully compatible to the Arduino Uno R3, allowing you to mount Arduino shields and keep your Uno code entirely unchanged (up to timing, since the STM32 board is substantially faster than the Uno). The Nucleo64 board has additional male pins, making the overall board wider than an Uno, and giving access to the substantial added features of the STM32F401RET6 MCU.

28.6 Recommended Videos

https://www.youtube.com/watch?v=visj0KE5VtY
DroneBot Workshop
"ESP32 CAM - 10 Dollar Camera for IoT Projects"

https://www.youtube.com/watch?v=hSr557hppwY
DIY Engineers
"ESP32-Cam Complete Guide"

https://www.youtube.com/watch?v=HDRvZ_BYd08
DroneBot Workshop
"Simple ESP32-CAM Object Detection"

28.7 Parts in Stock for CS358

Model	CHF
Arduino Uno (clone)	5
Arduino Mega 2560 (clone)	14
NodeMCU ESP8266 V0.9	3
Wemos D1 R32 ESP32	4
AIThinker ESP32-CAM	4
AIThinker ESP32-CAM w. Antenna	13
STM32F103C8T6 "Blue Pill"	3
STM32 Nucleo64 F401RE	15
FTDI USB-to-serial	2

Note that the FTDI board isn't a microcontroller board but provides USB connectivity for MCU boards without a USB board (such as the ESP32-

CAM and the Blue Pill) and can be used as a programmer for MCUs not on an evaluation board.

28.8 MCUs in Disguise

The previous sections of this chapter listed the "general-purpose" microcontroller boards that you may choose for your projects. There are, however, a few electronic components available to us, and mentioned in other chapters, that more or less clandestinely house MCUs to do their work. This includes some more complex sensors (GPS units, for example) and brushless motor drivers. Particularly the motor driver MCUs can or even have to be programmed by you, but usually just for their restricted purpose (such as FOC, see Chapter 43). They usually do not offer sufficiently many free pins (if any at all) to build your things entirely around them, without any general-purpose MCU board.

Also note that some of the general-purpose MCU boards covered in this chapter actually have *two* MCUs on board[1] – the one you program and another one just to run the USB protocol.

[1]The STM32 Blue Pill is a notable exception. This is why you can't program it through USB unless you first upload USB driver firmware to the main/only MCU using a RX/TX serial connection.

Chapter 29

Setting up the Arduino IDE

We will use the Arduino IDE for microcontroller programming. It is available for Mac OS, Microsoft Windows, and Linux. It is very easy to use, popular, and the easiest to find information for on the internet. It supports all popular microcontrollers, not just Arduinos. There are alternatives, including Eclipse and Visual Studio Micro, which we ask you not to use in this course, even if you are familiar with one of them. Download and install the Arduino IDE from
https://www.arduino.cc/en/software.

We upload code and supply power to the microcontroller via USB. Depending on the microcontroller board and your operating system setup on your computer, you may need to install a special USB driver to work with your microcontroller.

29.1 For the Arduino Uno

We usually provide you with Arduino Uno clones, which are absolutely legal – the Uno is an open-source hardware design. They are functionally equivalent to Arduino Unos produced by the Arduino company, with one exception: You may need to install a CH340 USB driver. This driver will often be already installed on your computer and you may not need to do anything. Installing the driver seems more frequently necessary for Windows users than for Mac users.

Search on Google if, on first connecting your Arduino Uno to your laptop via a USB cable, the USB port that the Uno is connected to does not show

up among the selectable options in the Arduino IDE. (See the menu item Tools→Port.)

Linux users may have permissions problems when trying to use USB with the Arduino IDE. User-level processes may not have (write) access to certain devices on Linux until you grant it. Many people have encountered this problem when setting up the IDE on Linux, and there are solution posts on the Web that you find with Google (search for something like "Arduino IDE Linux USB problem").

29.2 For the ESP8266

In the Arduino IDE, open the Files→Preferences menu item and paste the link
https://arduino.esp8266.com/stable/package_esp8266com_index.json
into the "Additional Board Manager URLs" field. Then install the esp8266 toolchain in Tools→Board→"Boards Manager".

Select Tools→Board→"ESP8266 Boards (version number)"→"NodeMCU V0.9 (ESP-12 Module)".

For NodeMCU ESP8266 boards, you need a CP210x USB driver:
https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads, see also
https://cityos-air.readme.io/docs/1-usb-drivers-for-nodemcu-v10.

This one will usually not be installed on your computer yet and you will have to install it.

29.3 For the ESP32-CAM

Google setting up the ESP32-CAM with the Arduino IDE. The steps are analogous to the instructions for the ESP8266, but the microcontroller board is called "AIThinker ESP32-CAM".

You will be thrilled to read that you can never have troubles with USB drivers. (The ESP32-CAM has no USB port. See how to talk to it and upload code in Chapter 27.)

29.4 In case you can't get it to work

Search the internet. The Arduino IDE is very widely used. It is very unlikely that nobody has encountered and overcome your problem before.

29.5 Other IDEs

We require you to use the Arduino IDE exclusively for the individual project and we recommend that you use it for the team project as well, unless there are good reasons for using another IDE. The teaching staff may not be able to help you with other IDEs.

One important alternative to the Arduino IDE is PlatformIO. This is a plugin for Visual Studio Code. PlatformIO supports all of the microcontroller boards we are working with, and usually you can just use C code developed on the Arduino IDE and it will compile and work fine on PlatformIO. The only thing you will have to change to your code is to add a line

```
#include <Arduino.h>
```

at the start of each of your source code files.

You may notice that for some non-Arduino MCUs (and only for these), the first time you compile your code, some slow caching is happing. One advantage of PlatformIO is that this is faster.

For ESP32 MCUs, there is an Eclipse-based IDE that you will need to use if you want to run TensorFlow directly on the MCU.[1] If you are not doing this, we suggest you stay with the Arduino IDE. It is simpler to use, even if you are very familiar with Eclipse.

For STM32 MCUs, there is the STMCubeIDE. which is also based on Eclipse. Its Hardware Abstraction Layer (HAL) is substantially more low-level than the one you will use with the Arduino IDE. The STMCubeIDE generates all the boilerplate code, so you may not need to write much more code than with the Arduino IDE; however, you will need to navigate and read a lot of code that you would not have to deal with on the Arduino IDE.

For advanced work with the STM32 MCUs, there is no way around the STMCubeIDE; it allows to access powerful MCU features that are below the HAL in the Arduino IDE and are thus inaccessible to you. Also, note that

[1] See https://www.tensorflow.org/lite/microcontrollers.

experience with the STMCubeIDE is actually a relevant marketable skill in job applications, while the Arduino IDE has been designed for children (really). If you do an embedded systems job interview and then reveal that you only have experience with the Arduino IDE, you may lose all credibility.

If you are interested in STMCubeIDE programming, check out this quick 30-minute tutorial on implementing the Blink example [2]:

https://www.youtube.com/watch?v=Hffw-m9fuxc
Mitch Davis
"STM32 Guide #2: Registers + HAL (Blink example)"

[2] But read Chapter 30 and particularly Section 30.8 first.

Chapter 30

Microcontroller Programming

30.1 Language Syntax

We program our microcontrollers using the Arduino IDE in C/C++.[1] If you do not know C yet, but know Java, you are already well prepared – Java takes much of its syntax from C++. If Python is the only programming language you know so far, then shame on you ;-), and you have another good reason for taking this course. C is a compiled and statically typed[2] language unlike Python. You need to declare your variables and their types – there is type checking but no automatic type inference. There is very little automatic type conversion (casting): You have to do this explicitly. C used to dominate software development, and to this day, at least performance-sensitive software, such as operating systems, networking and graphics code, and device drivers are mostly written in C. Computer scientists should know C. It would be risky, though, to claim mastery of C in job interviews based on your work with a very simple fragment in this course.

To us, by fiat, microcontrollers are no microcontrollers if they run an operating system (and Raspberry PIs are no microcontrollers). Since there is no operating system to manage resources for you, you have complete

[1] We will not make a real distinction between C and C++ here. C++ is C with some extensions (mostly related to object-orientation), and the Arduino IDE works with a heavily restricted version of C++. It is unusual to make heavy use of object-orientation in Arduino IDE programs, so you will probably mostly work in C.

[2] Though, compared to other, mostly more modern statically typed programming languages, it is comparatively permissive, which invites sloppy use and bugs.

control over all resources. There is no memory management, and all the memory is yours to use. You do not allocate memory using a system call (such as `malloc()` in C on POSIX-compliant operating systems) or a `new` instruction. If you need (simulated) dynamic data structures – which is unusual in simple programs – you statically declare arrays, use them as memory buffers, and do your own memory management. For instance,

```
int a[32][16];
```

declares a 32 times 16 two-dimensional integer array. On an Arduino Uno with its 2-byte integers, this takes 1KB, or half its RAM.

One of the most disgusting defects of Python is that code blocks are defined by whitespace intendation. In C, you create blocks by enclosing code in curly braces `{ }`, which is totally superior since it unleashes the power of Spaghetti code. So here is a little piece of code:

```
int i; for(i = 0; i < 10; i++) { if(i % 2 == 0) { /* do something */ }
else { /* do something else */ } } /* now i is 10 */
```

We start by declaring `i` as an integer. Then we loop over a code block with an if and an else branch ten times. The first time we visit the code block, `i` is zero; the final time, it is 9, so `i` is set to zero at the start, and then the condition $i < 10$ is checked every time before entering the code block, and `i` is incremented by one (`i++`) on exiting the code block. The if condition checks whether `i` is even (is `i` modulo 2 equal to zero?); if this is true, it enters its code block (which just contains a comment), otherwise it executes the else-branch. Note how instructions end in a semicolon, and the symbol for assignment (=) is different from the symbol for equality testing (==). Whitespace (indentation) and newlines play no particular role other than separating keywords and identifiers. Every C program can be written as a (very long) one-line program.

C has a reputation for being difficult to master and error-prone, but much of this to due to the fact that it invites tricks with pointers that are less seductive when there is no dynamic memory management. Despite the low-level programming we do with microcontrollers, we usually do not work with pointers at all.

30.2 Structure of a Program

This is specific to programs in the Arduino IDE, not C programs in general: Every program contains at least a `setup()` and a `loop()` function implementation. Additional declarations and functions are of course possible. So, syntactically, every program looks like this:

```
/* stuff, particularly include statements and declarations */

void setup() { /* your code */ }

/* more stuff */

void loop() { /* your code */ }

/* even more stuff */
```

The functions `setup()` and `loop()` return no value (thus their return type is declared `void`) and take no parameters.

A program is executed as follows: On startup or reset of the microcontroller, the `setup()` function is called once. Microcontrollers with a USB port are reset via USB from the Arduino IDE on the completion of uploading a new program, so you do not need to manually reset then.

After `setup()` has completed, the `loop()` function is repeatedly invoked, forever. You do not need to put all your code into the `setup()` and `loop()` functions, you can add additional functions that you call from anywhere you like.

This way of programming is inherently single-threaded.[3] Conceptually, your core is always busy. You can put it to sleep for limited amounts of time using the functions[4]

```
void delay(unsigned long delayInMilliseconds);
void delayMicroseconds(unsigned long delayInMicroseconds);
```

Using delay functions and reducing the activity of the microcontroller saves energy and extends battery life.

See https://www.arduino.cc/reference/en/ for the Arduino IDE language reference with an overview of all core functions.

[3] See the Web for information on programming multi-core microcontrollers.

[4] Shown here are function signatures showing the relevant argument and return type. This is not to be understood as executable example code!

30.3 Word Length and Numerical Data Types

If you look at the microcontroller comparison chart in Chapter 28, you see that the Arduino Uno is an 8-bit microcontroller while the ESP8266 and the ESP32 are 32-bit microcontrollers.

Usually, the convention in C is that the int type for (signed) integers uses a *word* of memory – the natural unit in a processor and the size of each of its registers. However, as an exception, while `int` on the 32-bit microcontrollers really takes 4 bytes and has a range from -2^{31} to $2^{31}-1$, Arduino Uno integers take two bytes (rather than one byte, eight bits) and range from -2^{15} to $2^{15}-1$. The type for byte-sized integers in C is `char` (or `byte`). For each signed integer type there is an unsigned type, prefixed by the keyword `unsigned`. So there are `unsigned char`, `unsigned long`, and `unsigned int` (which you can shorten to `unsigned`). Remember to handle overflow. On the Arduino Uno, if you declare

```
int      i = 32767; /* 2^15 - 1 */
unsigned u = 65535; /* 2^16 - 1 */
```

and then compute `i+1`, you get the int -32768. If you compute `u+1`, you get 0.

30.4 Serial Communication and Debugging

While you keep your microcontroller connected to your computer via USB, you can use the USB connection for serial communcation between the two. The Arduino IDE provides a terminal for two-way comunication, the Serial Monitor (and a Serial Plotter) to make this easy, but you can use other tools to do this, such as a telnet client like putty or even use a Unix terminal to talk directly to the device.

Serial communication is particularly useful for logging output and debugging. To use this, you have to call `Serial.begin(r)` in the `setup()` function, where `r` is a rate of bits per second. We recommend a rate of 9600, which is the default setting on the IDE Serial Monitor. You can also set the bit rate in the Serial Monitor. If they aren't the same, you get gibberish. Now you can use serial read and write (print) statements in your code. Here is an example.

```
void setup() {
  Serial.begin(9600);
}

int i = 0;

void loop() {
  Serial.println(i);
  i++;
  delay(1000);
}
```

This prints[5] the value of `i` to the serial monitor (initially 0, and incrementing by one in each iteration), waits for one second, and then repeats, forever. See
https://www.arduino.cc/reference/en/language/functions/communication/serial/
for all on serial communication. By the way, `Serial` is an object.

30.5 I/O

Here we discuss the use of I/O pins. For more advanced I/O, see Chapter 31 and the Web. We distinguish digital I/O using pins that allow both reading and writing (GPIO pins) from analog (input) pins.

For GPIO pins, we have to set a mode (`INPUT` or `OUTPUT`) in `setup()` using the function `pinMode()`. Then you can use the functions digitalRead() and digitalWrite() to read and write the values HIGH and LOW to and from your pins. The signatures of these functions are as follows.[6]

```
void pinMode(int pin, pinmode_type mode);
void digitalWrite(int pin, int HIGHorLOW);
int digitalRead(int pin);
```

Here is an example:

[5] See https://www.arduino.cc/reference/en/language/functions/communication/serial/println/.

[6] I made up pinmode_type. Use INPUT or OUTPUT as mode. Some microcontrollers support further modes for some pins, such as INPUT_PULLUP.

```
void setup() {
  pinMode(2, INPUT);
  pinMode(3, OUTPUT);
}

void loop() {
  digitalWrite(3, digitalRead(2));
}
```

This keeps pin 3 at the same digital value (`HIGH` or `LOW`) that it reads at pin 2. For another example, here is the Blink sketch, the Hello-World example of the Arduino IDE.

```
void setup() { pinMode(13, OUTPUT); }

void loop {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

If you build a circuit with a 330Ω resistor and an LED in series between pin 13 and GND (and the LED is oriented correctly, see Figure 20.2), this program turns the LED on, waits a second, turns it off, waits a second, and repeats. On the Arduino Uno, pin 13 is special in that an on-board test LED is connected to it. This sketch makes that LED blink even if nothing is externally connected to pin 13. See
https://docs.arduino.cc/learn/microcontrollers/digital-pins/
for more information on digital pins.

Some microcontrollers allow you to programatically activate or de-activate internal pull-up or pull-down resistors for individual pins. This elimitates the need to include such pull-up or pull-down resistors in your circuits.

Some GPIO pins support outputting a PWM signal using the function

```
void analogWrite(int pin, int duty_cycle);
```

where `duty_cycle` is a number between 0 and 255. The PWM frequency (which is different from the duty cycle) can usually be configured (see also the example on controlling steppers using PWM in Chapter 39). After

executing `analogWrite()`, the microcontroller keeps sending a PWM signal with the set duty cycle until you do a digital write to the pin or use `analogWrite(pin, 0)`. Your program isn't blocked by this call and the CPU of the microcontroller isn't busy with generating the PWM signal since there is separate hardware support for sustaining PWM signals on the PWM-enabled pins. You could generate a PWM signal by a suitable program, but this will keep your CPU busy and creating an exactly times signal may be tricky, particularly on a slow microcontroller.

Analog input(pin)s are read using the `analogRead()` function. The value returned depends on the resolution (in bits) of the internal analog-to-digital converters, which depends on the microcontroller. The Arduino Uno uses 10 bit resolution and the value returned is an integer between 0 and 1023.[7]

Analog input pins are designed to read true analog signals as they may be obtained from sensors and variable resistors (such as potentiometers). This information is digitized using an analog-to-digital converter (ADC) built into the MCU. Each ADC has a resolution (typically 10 to 14 bit) and not all of our MCUs have ADCs of the same resolution (see your MCU's datasheet). For high-precision sensing applications, you can get a high-resolution ADC as a separate component and connect it to your microcontroller board. Analog in pins are not for reading PWM signals. You do that using the `pulseIn()` command on a digital GPIO pin.

You can create a true analog signal using a digital-to-analog (DAC) converter, if your MCU has one, get an external DAC board, or you can try to smoothe out a PWM signal using capacitors.

30.6 Interrupts

So far we have only one way of reacting to external events: by constantly testing for them – for instance, using `digitalRead()`. There is an alternative: interrupts. We can attach a function we have implemented as an Interrupt Service Routine (ISR) to a specific event (such as the digital value on an input pin changing). This is done with the method `attachInterrupt()`. We refer to the reference https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/ for details on this and provide the example from that page here:

[7] For the case that resolution matters, there are ADC boards with higher resolution.

```
volatile byte state = LOW;

void setup() {
  pinMode(13, OUTPUT);
  pinMode( 2, INPUT);
  attachInterrupt(digitalPinToInterrupt(2), blink, CHANGE);
}

void loop() { digitalWrite(13, state); }

void blink() { state = !state; }
```

The code switches the on-off state of pin 13 (and thus, on the Arduino Uno, of the built-in LED) every time the digital value on pin 2 changes. On the Arduino Uno, pin 2 is one of the two interrupt-enabled pins. One non-interrupt-enabled pins, this code would not work. As you can see, we can use `HIGH` and `LOW` as truth values (true and false); `!state` negates the truth value of `state`.

You would be right in considering it natural (and power-saving) to put your microcontroller to sleep while it waits for an interrupt. Unfortunately, this does not work. The delay functions use interrupts themselves, and if you define your custom interrupts, they interfere.

If the kind of change you are subscribing to by attaching an interrupt is related to the physical process of opening or closing a connection, say by a switch or button, there is a pitfall. For a small fraction of a second, what usually happens is that the contact is made and lost repeatedly, potentially triggering the interrupt several times. In that case, one should make sure in the program that this noise does not adversely affect its performance and correctness. This is called *debouncing*. This of course is revelant for monitoring change events no matter what method, using interrupts or not.

30.7 Porting Code to other Microcontrollers

The Arduino IDE provides good abstraction from the details of individual microcontroller families and models, and frequently, porting programs from one microcontroller model to another just requires us to substitute pin numbers to account for the different pinouts of the two models. Of

course, some features are supported in different multiplicites (e.g., numbers of analog input pins and thus analog-to-digital converters), potentially requiring mayor changes or making a particular MCU unsuitable for a particular use. Some features are only present in some models. Advanced functionality such as Wifi or video recording are usually supported by code libraries that spare you the need for much coding and make the use of this advanced functionality easy, but require you to read up on these libaries. Such libraries will be specific to the MCU models and boards.

30.8 Getting to the Bare Metal

This section aims to help you better understand the functioning of an MCU and what you ultimately do when you program it. You do not need to program this low-level unless you want to get the very best performance from your MCU and are willing to deal with the added complexity.[8]

Methods such as `digitalWrite()` in the Arduino IDE and even pin numbers do not show up in compiled, executable MCU machine code: they form part of a *Hardware Abstraction Layer (HAL)* specific to the Arduino IDE. The reason why such a HAL does not go all the way to making Arduino IDE programs perfectly portable between MCU models is that they do not all have the same functionality.

If you are interested in seeing the code that defines the essential mappings of this HAL, see
https://github.com/arduino/ArduinoCore-avr/tree/master/cores/arduino,
in particular the file `Arduino.h`. The file `pins_arduino.h`[9] defines the mapping between MCU pins and board pins – you can find the essential information of Figure 30.1 in this header file, with the following ASCII art drawing of the IC and its pins to accompany it (starting at line 89):

```
//                  +-\/-+
//            PC6  1|    |28  PC5 (AI 5)
//      (D 0) PD0  2|    |27  PC4 (AI 4)
//      (D 1) PD1  3|    |26  PC3 (AI 3)
```

[8] Note that some IDEs, such as the STM IDE for STM32 family MCUs, while having a HAL, expose you much more strongly to the low-level workings of the MCU.

[9] https://github.com/arduino/ArduinoCore-avr/blob/master/variants/standard/pins_arduino.h – this is for the Arduino Uno; further boards are covered in other subdirectories of variants/.

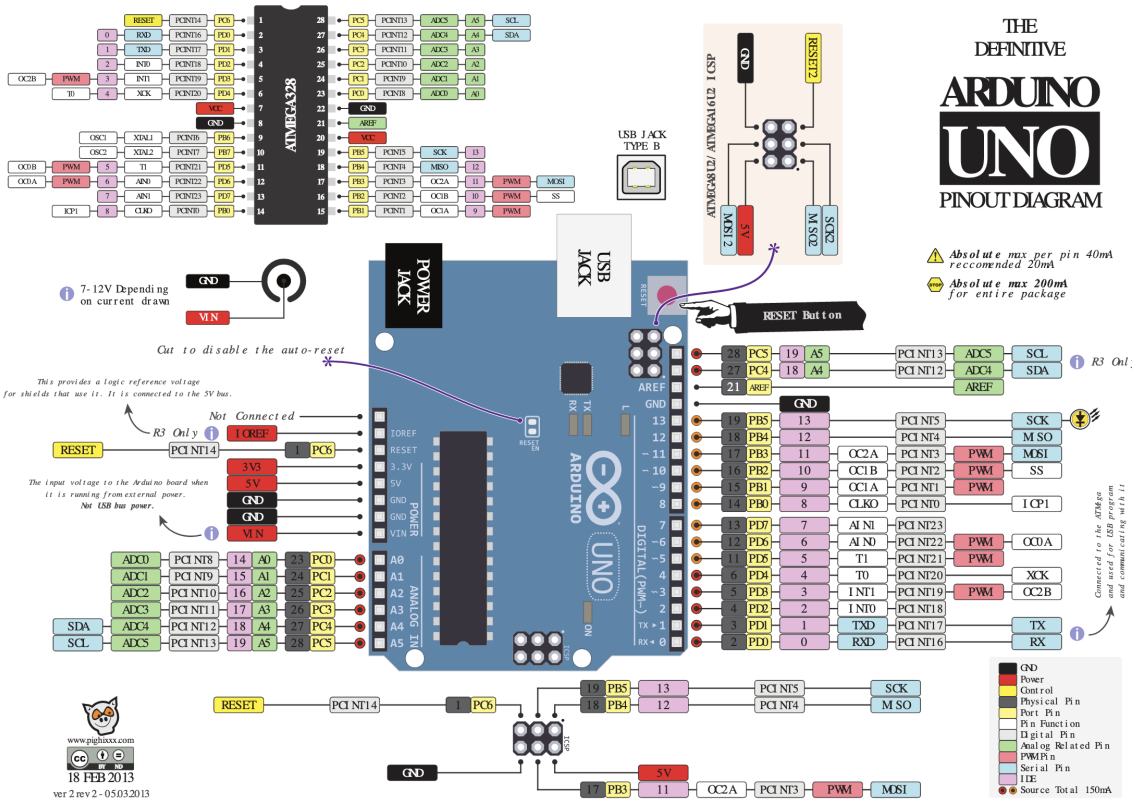


Figure 30.1: Arduino Uno vs ATMEGA328 pinout.

```
//      (D 2) PD2  4|    |25  PC2 (AI 2)
// PWM+ (D 3) PD3  5|    |24  PC1 (AI 1)
//      (D 4) PD4  6|    |23  PC0 (AI 0)
//            VCC  7|    |22  GND
//            GND  8|    |21  AREF
//            PB6  9|    |20  AVCC
//            PB7 10|    |19  PB5 (D 13)
// PWM+ (D 5) PD5 11|    |18  PB4 (D 12)
// PWM+ (D 6) PD6 12|    |17  PB3 (D 11) PWM
//      (D 7) PD7 13|    |16  PB2 (D 10) PWM
//      (D 8) PB0 14|    |15  PB1 (D 9) PWM
//                  +----+
//
// (PWM+ indicates the additional PWM pins on the ATmega168.)
```

Inside executable MCU code, there aren't instructions such as `pinMode()` or `digitalWrite()`, just like your laptop's CPU has no instructions for reading out the state of your keyboard or for drawing a pixel on your screen. In your computer, devices are memory-mapped, and we write to the screen by writing to a region of memory. Your CPU instructions are all about reading and writing memory, basic calculation, and control (branching and looping). I/O in your MCU works similarly, and its pins are addressed via special registers.

For example, in the Atmel ATMEGA328, the MCU used on the Arduino Uno board, all I/O works through three 8-bit registers, called (port) PB, PC, and PD (see Figure 30.1 for a complete specification of the wiring between board pins and pins of the ATMEGA328 IC). Consequently, in ATMEGA328 machine code, we set bit 5 of register PB to 0 or 1 to set digital pin 13 to LOW or HIGH, respectively. Note, however, that this isn't the only thing `digitalWrite()` does. For example, it also makes sure than PWM generation for that pin is off. You can find the actual code of the `digitalWrite()` function in `wiring_digital.c`[10] (starting at line 138).

HALs come with a cost to performance. For example, in the Blink sketch, where we keep turning pin 13 on and off, there is no need to turn off PWM every time. Doing bit manipulation is very natural in C. However, changing several bits in a register is no more costly than setting just one; using `digitalWrite()` to set several pins uses separate instructions for each pin; thus, a low-level program achieves the same more efficiently.

[10] https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/wiring_digital.c

Chapter 31

Interfacing and Communication

In this chapter, we cover wired communication, but more specifically, interfacing and communication among the various electronic components of your thing. We do not cover (wired) networking (such as Ethernet, and IP networking) and communication over great distances.

There are a number of interface standards and protocols that you rarely – if ever – see outside the context of embedded systems. These include UART, I2C, SPI, and CAN. These are for making the components of an embedded system talk to each other, and they are typically very simple and require little computational cost. They are not meant to be seen by end-users, and they are not suitable for large-scale or even wide-area networks. They vary in their specifics.

Figure 31.1 gives an overview. Very frequently, you do not choose which protocol to use, but a component (such as a sensor of motor driver) that you want to use requires you to talk to it via a specific interface. You can often recognize which interface – without looking at the datasheet, which you of course also have to do – by the names of the interface pins. Typical names are shown in the second column of Figure 31.1. Note that components

	Usual Pin Names	Type	Max Devices
UART	RX/TX	point-to-point	2
I2C	SDA/SCL	master-slave bus	127
SPI	MISO/MOSI/SCLK/CS	master-slave bus	∞
CAN-Bus	CAN_H/CAN_L	multi-master bus	32

Figure 31.1: Embedded systems interfacing standards.

that interface with any of these standards also need common ground, so there is, additionally, at least a ground wire connecting the components.

If you have a choice of interfacing protocol (for instance, if you want to make two MCUs talk to each other), you should use the simplest protocol that is applicable, because "simple" usually means both low computational cost and simple to program (using a suitable library). The protocols in Figure 31.1 are roughly ordered by increasing complexity.

The uncontested champions of communicating using these protocols are MCUs of the STM32 family. Not only do they support all of these protocols – usually the support multiple instances of some of these protocols (which and how many depends on the member of the family).

31.1 GPIO Pins

Before we cover the various protocols, be reminded that you can build your own interfaces and protocols using GPIO pins. Usually, you won't do this unless you have to, because running one of the above protocols with hardware support means that work is taken off the compute core, and you have to worry less about timing and how receiving messages interleaves with your other computations and the overall control flow in your code. If you want incoming bits (or better, changes in the voltage level of your "incoming" signals) to trigger interrupts, this takes up interrupt-enabled pins, which then aren't available for other uses.

If you are building your own protocol, a main concern is how to achieve synchrony between two communicating components. You cannot rely on local clocks (particularly approximations of real time-based ones, as used in `delay()`) as these may drift between components. The easiest way to solve this is to have a dedicated clock signal sent by one component to the others. Each switch between logical HIGH and LOW (either way) denotes one clock tick. This approach is used, for instance, in I2C and SPI (via SCL and SCLK, respectively).

If you simply want to continuously transmit a single current numerical value, in one direction only, you can do this with a PWM signal. In that case, you do not need a clock signal (but you still need, as always, common ground). Read this PWM signal using `pulseIn()`.

31.2 RX/TX Serial (TTL, UART)

All of our MCUs have at least one UART[1] interface, and some have several. This is a standard that manifests in several ways; we only care about the simplest[2], TTL (Transistor-Transistor Logic). Here we have, in addition to a common-ground wire, two wires, called RX and TX for two-way data transmission. There is no dedicated clock wire; you have to set the same bit rate for data transmission[3] in both components intended to talk to each other. This is a point to point protocol for connecting two components: it is not a bus that can connect multiple components like I2C, SPI, or CAN. You connect the RX line of one component to the TX line of the other, and vice versa.

TTL RX/TX serial plays a special role in MCUs. Each MCU can be programmed by uploading executable code through the first of these interfaces (if there is only one such interface, than that one). MCUs are usually in program upload mode while their reset pin is at the logic level that means "reset".

MCU boards that have a USB connector usually have USB serial connected to this first RX/TX interface (with a dedicated USB protocol chip such as a CH340 IC translating the USB protocol to TTL RX/TX). For that reason, you cannot use these RX/TX pins freely for communicating with other components: on startup and reset, special things happen on these pins, and if you communicate with your computer via USB, this transmission shows on these pins. In practice, it is best to use an MCU's first UART interface just for program upload and communication with your laptop – anything else is complicated and error-prone.

If you run out of hardware UART interfaces, you can use the SoftwareSerial library to turn any two GPIO pins into a serial port. Note that this has limitations – SoftwareSerial does not work well on some MCUs, and there is a bit rate limit for such software serial ports.

---

[1] U(S)ART, universal (synchronous/)asynchronous receiver-transmitter, see https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter.

[2] There are also RS-232 and RS-485, which use further wires for noise-canceling/error-correction.

[3] Which does not solve the timing/synchronization problem itself, but you do not need to worry about this if your have hardware support or a software library for serial communication.

31.3 Universal Serial Bus (USB)

Many microcontroller boards have a USB port for communicating with a computer and for code upload. For those that do not (such as the ESP32-CAM), you can use an FTDI board as a USB adapter. See the discussion of the ESP32-CAM in Chapter 27.

While your microcontroller board is connected to your computer via USB, you can talk to it in various ways such as telnet or putty – you are not limited to the Arduino IDE.

Note that while USB is a serial protocol, USB is more complicated than TTL serial. This is for robustness, since it is a standard to be used by end-users. The USB protocol includes a mechanism for a device to identify itself and to transmit metadata about itself. One important application of USB ports is as power supply connectors[4], and the recent USB-c standard has taken this to new levels by not just being designed for high currents (up to 5A) but also with a mechanism for switching the voltage and running at higher voltages than 5V – up to 20V. Because of this, we cannot simply pass through USB data lines to TTL RX/TX pins. CH340 and CP2102 ICs, for which you may have to install drivers to connect your microcontroller board to your computer, are interfacing chips between USB and TTL RX/TX.

31.4 I2C

I2C[5] is designed for communication between microcontrollers and low-end peripherals over small distances. It is a master-slave bus, so many (up to 127) devices can be connected, and each has an address for it to be contacted. There is one dedicated master (an MCU) that initiates all communication – it either sends messages to other devices or requests data from other devices – which will respond to requests but not otherwise initiate communication. Typical slave devices such as sensors have a fixed address which you need to know from the datasheet to use it (sometimes, an address can be chosen from a small number of alternatives by setting jumpers, but most of these devices are very simple and not programmable

[4] In a very laudable initiative, the EU is pushing electronics manufacturers to standardizing power supplies – this will commoditize power supplies and associated cables, making them cheaper (looking at you, Apple), and eliminating the need to own and carry around so many of them.

[5] https://en.wikipedia.org/wiki/I2C

or configurable). This is a limiting factor – you will only be able to use one or a limited number of this kind of device, because each device on the bus needs to have a unique address. In an MCU, the address is programmable.

There is a standard[6] library for I2C called Wire[7] that makes using I2C easy.

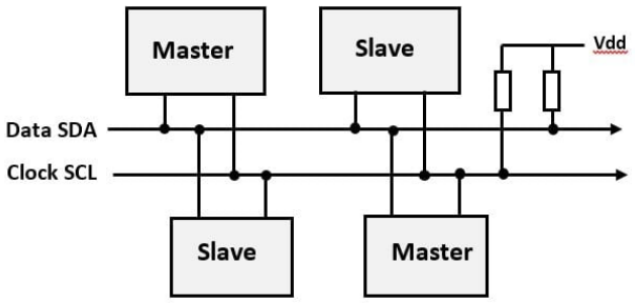


Figure 31.2: The I2C bus (note the pull-up resistors).

One thing to note is that pull-up resistors should be used to pull up both the SDA and SCL lines (see Figure 31.2). If you just want to make two Arduino Unos talk to each other, they are not needed.

Here are a couple of videos on using I2C:

https://www.youtube.com/watch?v=PnG4fO5_vU4 DroneBot Workshop "I2C Part 1 - Using 2 Arduinos"
https://www.youtube.com/watch?v=yBgikWNoU9o DroneBot Workshop "I2C Part 2 - Build a I2C Sensor"
https://www.youtube.com/watch?v=6IAkYpmA1DQ How To Mechatronics "How I2C Communication Works and How To Use It with Arduino"

It is very easy to use I2C to make an MCU talk to others. Example: two Arduino Unos. You wire them up using three Dupont cables, to connect

[6] Thus, it does not have to be installed.

[7] See the Wire library documentation at
https://docs.arduino.cc/learn/communication/wire/
and the library reference at
https://www.arduino.cc/reference/en/language/functions/communication/wire/.

A4 to A4 (SDA), A5 to A5 (SCL), and GND to GND of the two Unos.[8] The code running on the master is

```
#include <Wire.h>

int x = 0;
void setup() { Wire.begin(); }
void loop() {
  Wire.beginTransmission(9);
  Wire.write(x);
  Wire.endTransmission();
  x = (x + 1) % 10;
  delay(1000);
}
```

The code running on the slave is

```
#include <Wire.h>

void setup() {
  Wire.begin(9);
  Wire.onReceive(recv);
  Serial.begin(9600);
}

void recv(int bytes) { Serial.println(Wire.read()); }
void loop() {}
```

The master sends integers to the slave, which the slave prints to serial.

31.5 Serial Peripheral Interface (SPI)

See Wikipedia[9] and this video:

https://www.youtube.com/watch?v=fvOAbDMzoks
GreatScott!
“Electronic Basics #36: SPI and how to use it”

[8] The I2C standard dictates that there should be a pull-up resistor for both SDA and SCL, but when connecting to Uno R3s, you should be able to do without.

[9] https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

Compared to I2C, SPI uses up more pins of your MCU – not only does SPI use three (clock, master-out-slave-in/MOSI, and master-in-slave-out/MISO) rather than two pins for communication; in addition, one pin of your master MCU is needed per slave device to select which slave device to talk to: While the actual communication lines are shared among slaves, there is a separate line per slave to instruct one slave to talk (to) and the others to be quiet. This can be an advantage: No per device addresses are needed, so you can communicate with several instances of the same kind of device on the same bus, unlike for I2C.

If you want to use many devices with SPI (there is no limit), you can do with a logarithmic number of address pins of your MCU by using a de-multiplexer chip such as the 74HC595[10].

31.6 CAN-Bus

The CAN-bus has been developed for automotive applications, but we also encounter it with advanced brushless motor drivers like o-drives and VESCs and expensive servos (like those used in the MIT[11] robot dog). The microcontrollers we use do not have native CAN-bus support, but there are boards for that. Note that CAN-bus devices are arranged in a chain, at the ends of which we need special terminating resistors.

https://www.youtube.com/watch?v=QYX_XOjjGOM
How To Electronics
“Arduino CAN Bus Tutorial | Interfacing MCP2515 CAN Module with Arduino”

The CAN-bus is also used for accessing the self-diagnostics facilities of cars in car repair shops. There are videos on youtube on how to do this with you car, such as the following. (Check the legal and warranty implications before doing anything.)

[10] It's a shift register, but can be used as a de-multiplexer. The 74HC595 is an easy way to increase the number of digital output pins of you MCU if you do not need particularly rapid switching. See https://www.youtube.com/watch?v=Ys2fu4NINrA

[11] That's the other MIT. Btw, did you know that the animal first called Panda is now called the Lesser Panda (Ailurus fulgens) to give way to calling the *high-contrast Chinese bamboo bear* a Panda? (See https://www.youtube.com/watch?v=H63S6QG-8Ow.) Moving on...

https://www.youtube.com/watch?v=lkBILe55LQ8
South West EV UK
“How to read the CanBus in any car. (Can Bus) Part #1”

31.7 Parts in Stock for CS358

Model	Comm to microcontroller	CHF
PCA9685 16-channel PWM multiplexer	I2C	12
FTDI USB-to-TTL-serial board	TTL serial	2
8-channel logic-level converter	GPIO	1
MCP2515 CAN module	SPI	2

Chapter 32

Wireless Communication

32.1 Bluetooth

Some microcontrollers, such as the ESP32-CAM, natively support Bluetooth. (But note that "plain" Bluetooth is not the same or compatible with Blutetooth Low Energy (BLE).)

Alternatively, you can wire up you microcontroller with the HC05 Bluetooth board.

https://www.youtube.com/watch?v=NXlyo0goBrU
Bytes N Bits
"Adding Bluetooth to Your Arduino Project with an HC-05 or HC-06 Bluetooth Module"

The HC05 has a logic level of 3.3V (logic 1/high is represented by 3.3V), which means that if you are using a microcontroller with a logic level of 5V like the Arduino Uno, you must translate the signal that goes from the Arduino's TX pin to the HC05's RX pin using a voltage divider or a logic-level converter. Note that the Arduino's threshold for high is below 3.3V, so it "understands" the 3.3V signal that comes back from the HC05 to the Arduino's RX pin, so no translation is needed here.

The HC05 can work in a master or slave mode, and particularly if you want to use multiple HC05, you need to do some configuring (using the button on the HC05 and AT commands). Look for tutorials on the Web for that. Note that the default password for the HC05 is either 0000 or 1234, usually the latter. The default baud rate can also vary, most tutorials say it should be 38400 baud, but the batch I worked with had 9600 baud as default. An indication that the baud rate set in your code doesn't match

an HC05 is if you write out what an Arduino receives through Bluetooth with the Serial Monitor, and it shows really weird symbols outside the alphanumeric range. When you set 38400 in software and the HC05 runs at 9600 baud, you would also "receive" two to three bytes for every byte you send.

For Bluetooth debugging, I recommend to install a Bluetooth serial terminal app on your phone (there are plenty free ones in both the Apple and Google app stores). This is a lot less painful that setting up a bluetooth terminal on your laptop.

32.2 Wifi

Some micro-controllers are Wifi-capable; this includes the ESP8266 and the ESP32-CAM but not the Arduino Uno. There are Wifi boards and cousins of the Arduino Uno that are Wifi-capable, but we do not use them. If you want a microcontroller like the Uno but with Wifi-capability, pick the ESP8266.

Normal Wifi at EPFL is using WP2-enterprise, which is tricky to connect to with microcontrollers. DLLEL has set up a WP2-personal hotspot, which is easy to use from a microcontroller. Go here for further information: https://make.epfl.ch/tools/iot-wifi At home (i.e., for your individual project), you can set up a Wifi Hotstop on your mobile phone, but please don't do this on the EPFL campus.

32.3 ESP-Now

You will do Wifi using Expressif's MCUs. Note that ESP32 MCUs also support a proprietary (i.e., just among EPS32 MCUs) wireless communication method called ESP-Now that is a bit faster and more long-range on the same MCUs. It is point-to-point, thus it does not require a Wifi-hub nearby, so you could use it outside and in the wilderness.

32.4 Radio

There are small and cheap wireless communications boards designed for use with microcontrollers, such as HC-12.[1] Nowadays, in most cases, it is probably preferrable to use Wifi for wireless communication and microcontrollers.

We may also use remote controls from/for toy cars, planes, etc.

32.5 Parts in Stock for CS358

Model	Comm to microcontroller	CHF
HC05 bluetooth board	TTL serial	3

[1] https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-12-long-range-wireless-communicati

Part V

Sensors

Chapter 33

Introduction to Sensors

This chapter is work in progress. For now, we just provide a grouping of sensors by the difficulty of using them plus some general insights.

There are sensors for almost anything you could be thinking of, and they are often quite affordable and easy to use. Go to aliexpress.com and search for "sensor kit". The search will return kits consisting of as many as 45 simple sensors for use with a microcontroller like the Arduino Uno (see Figure 33.1). Have a look at the images and the lists of sensors there.

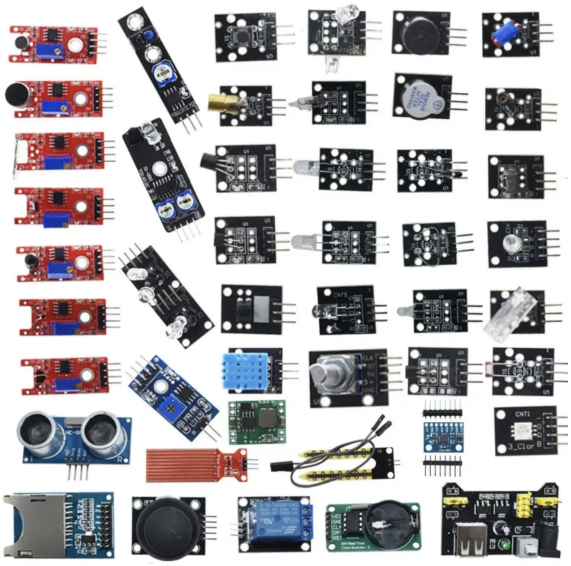


Figure 33.1: A typical sensor kit (costs about CHF 11 for the entire kit). Not everything in this kit is a sensor, though.

There are some kinds of sensors, like accelerometers, that are intrinsically tricky to use. While we say so, do not be discouraged outright from using them. Look at some tutorials on using them online and then decide. We do not rate anything mentioned in this chapter as unsuitable (too difficult to use) for the course, though some LIDARs are too expensive.

33.1 Super-simple Sensors

Some sensors appear as resistors (with variable resistance based on the quantity measured) in your circuit. This includes temperature sensors, humidity sensors, photoresistors[1] (brightness sensors), and Hall sensors (which sense magnetic fields and can be used, in conjunction with magnets, for various applications, such as sensing small distances and orientations and counting rotations). They are connected to your microcontroller via an analog input pin, which measures a voltage. (But search the Web for an example circuit – you typically build up a voltage divider.)

There are also transistor-like sensors, such as phototransistors, which switch a digital signal based on a physical quantity (such a brightness) exceeding a threshold. You can read these with a digital input pin – again, search the Web for circuit examples.

33.2 Simple Sensors

Some sensors take measurements of physical quantities in a scenario where we either need to detect very small changes, or the sensor reading changes only slightly as environmental conditions change. An example are load cells, which are weight sensors based on the changing resistance of a piece of metal that is being deformed. The resistances change only very slightly, and reading this with an analog input pin would not get reliably useful results, no matter what you do in your code. These require amplifiers. Amplifiers are components that you may know from sound systems as increasing the volume of sound. That is not their key function, though: there are about enhancing differences in sound levels. Many a sensor comes with board with an amplifier on it. This does not make it hard to

[1] There are various kinds, based on the wavelength of light they are sensitive too. IR-sensors are different from sensors for visible light.

use, but you usually have additional wires to power the electronics on the board, and in some cases you need to provide negative voltages (which you achieve by batteries and a floating ground).

Another complication arises when the sensor sends messages that have structure, such as pairs of readings or multi-bit messages. In that case, they need a suitable protocol to communicate messages to the microcontroller. This does not make the use of these sensors inherently difficult, but you must be aware of this. Usually, such sensors come with a matching library to make their use easy, which you first have to download. An example of a sensors with a simple message protocol is the HC-SR04 ultrasonic distance sensor, which is covered in detail in Chapter 34.

Also keep in mind that some sensors have an active component such as an LED as part of their assembly, which interacts with the environment and creates additional constraints on where and how to place the sensor on your thing. This includes, for instance, the above-mentioned HC-SR04. Another example is the TCRT5000 IR sensor for sensing the brightness of a surface, which is often used for line-following mobile robots.

33.3 Challenging Sensors

33.3.1 Accelerometers

There are a number of related terms used in this context: accelerometers (which measure acceleration and deceleration of the thing the sensor is mounted on), compasses (which detect orientation relative to the magnetic north pole of Earth), gyroscopes (which measure orientation relative to the vector of gravity), and inertial measurement units (IMU; which refers to sensors measuring any subset of the above, but usually primarily measure acceleration).

The need for these naturally arises in many applications, but they are challenging to use, and some applications are not realistic. It is relatively unproblematic to detect the fact that a thing is accelerating, but you cannot robustly calculate absolute positions or relative distances moved. Do not assume that you can measure the distance travelled by a mobile thing using an accelerometer. In principle, it is possible to calculate distances traveled from acceleration and time, in practice, the measurements are too imprecise for this to work and errors accumulate and multiply.

Also, the orientation of your thing in space, say, relative to "down" and north, is feasible, but these sensor readings drift, and compensating this isn't easy.

The most widely used and available accelerometer is the MPU6050. You will find that there are other accelerometer boards that are sometimes described as more precise. These suffer from the same (programming) pitfalls as the MPU6050, such as drift. If you don't get these under control for the MPU6050, switching to a more precise alternative will not solve your problem.

Not all such sensors measure the same things. Look at the datasheet. A simple stat is how many degrees of freedom (DOF) they have/measure. More is not necessarily better in your application.

We will probably give you an MPU6050 even if you request a different IMU.

33.3.2 GPS Sensors

The software side of using them isn't particularly difficult, but we frequently have experienced malfunctioning sensors. Also, their precision is not high enough for precise robot navigation.

33.3.3 LIDAR

LIDAR sensors (that actually work) tend to be large, heavy, and expensive.

33.3.4 Cameras/Vision

You usually combined video cameras with computer vision. The ESP32-CAM is a microcontroller with an on-board two-megapixel camera and borderline sufficient compute capacity to do basic computer vision tasks on board. There is a very limited version of TensorFlow that can run on the ESP32-CAM, but we warn you that this may be super-tricky to get to work, and we haven't seen anyone do it yet. The standard way of doing computer vision in this course is to use a stationary Webcam or stream video from the ESP32-CAM to your laptop and do the computer vision processing there. See also Chapter 53.

Remember that cameras are not the only way of "seeing" the environment and avoiding obstacles in the case of a mobile robot. Ultrasonic dis-

tance sensors (possibly mounted to a rotating platform) and LIDARs are alternatives that may be preferable – sensing objects may be more robust than with computer vision.

33.4 Recommended Videos

https://www.youtube.com/watch?v=V1txmR8GXzE
DroneBot Workshop
"Using Rotary Encoders with Arduino"

https://www.youtube.com/watch?v=XCyRXMvVSCw
DroneBot Workshop
"Build an Electronic Level with MPU-6050 and Arduino"

33.5 Parts in Stock for CS358

Model	Type	CHF
HC-SR04	ultrasonic distance sensor	2
TOF400C	laser distance sensor	5
TOF400H	laser distance sensor	6
TCRT5000	infrared reflective optical/distance sensor module	tba
LD2410C	24GHz radar human presence sensor	7
?	passive infrared motion sensor, small	tba
SEN-13285	passive infrared motion sensor	10
MPU-6050	6 DOF inertial measurement unit	5
NXP	Adafruit 9-DOF accelerometer/mag/gyro board	62
OE-TP	capacitive touch button	1
LLC4690	fingerprint sensor	19
WallySci E3K	biosensing platform/sensor kit	143
ATGM336H	GPS sensor module	12
GE-NEO6MV2	GPS sensor module	29
-	45-item misc sensor kit	30
?	electrets (microphone capsules)	?
RC522	RFID sensor + RFID tag	6
TC3200	color sensor	8
SEN-13329+HX711	load cell and amplifier board (weight sensor)	20
CUI AMT-102-V	Rotary Incremental Encoder (for FOC), indexes	21
TLE5012b?	Hall Effect Rotary Encoder (for FOC)	18?

33.6 Sensor Pitfalls

As you can see in the table above, sometimes there are multiple different sensor technologies available to sense the same physical quantity. For example, above, we find distance sensors that use ultrasound, lasers, and infrared light. All of them actively emit sound or light waves and measure their round-trip times. Our ultrasound and laser sensors each are rated to a maximum distance of about 4m. The ultrasound sensor may interfere with other ultrasound sensors (in case you had in mind to use multiple such sensors on your thing). The laser sensor, on the other hand, may be sensitive to the surface materials it is reflecting its beam off of. The infrared sensor has very different use cases – it is only suitable for very small distances; while it can be used for collision warnings, its most popular use case is as a reflectivity sensor directed downward, to all a robot to follow a line drawn on the ground, for example. The sensor is placed just millimeters above the round and is able to distinguish a reflective (white) surface from a dark surface.

Another example are rotation sensors (rotary encoders). There are very cheap rotation sensors (even a potentiometer – a variable resistor – can achieve this) that are intended to read the angle of a rotary knob (e.g., the sound volume on a sound system). These have relative low resolution and precision and may have a hard stop to rotation both clockwise and counterclockwise (as rotary knobs usually do), making them unsuitable to be attached to the output shafts of continuously rotating motors. At the other end of the price range are rotary encoders (usually incremental encoders[2]) that offer very high resolution (thousands of PPR – pulses per revolution) and – usually – cause nearly no friction when attached to a motor shaft. These usually use either optical technology or the magnetic hall effect. The former need to wrap around a motor shaft and are best used with motors whose shaft protrudes on both ends of the motor. Hall effect encoders require a small magnet to be glued to one of the ends of the motor shaft (a circular base, when the shaft is viewed as a cylinder) or to the rotor of the motor (if it is an outrunner[3]). Rotary encoders with an *index* have a way of keeping track of an absolute zero position of the encoder. In an encoder without an index, you have to keep track in your program of the change of position since power-on, and you may need to do

[2] https://en.wikipedia.org/wiki/Incremental_encoder
[3] https://en.wikipedia.org/wiki/Outrunner

some additional calibration for your thing.

As you see, there are things to know about sensors to determine whether a particular sensor module is suitable for you. You do not get around reading up on the sensor and looking for a tutorial on that particular sensor model. (You need to do this before you put it into your bill of materials.)

Even though it is obvious, be reminded that sensors need to sense their environment, and for that, they need to have unimpeded access to it. Plan ahead when you design your thing. Where (on the "outside") do you have to put your sensors so the thing does not obstruct the sensors?

Chapter 34

The HC-SR04 Ultrasonic Distance Sensor

Juliette Parchet

The HC-SR04 ultrasonic sensor is a popular sensor used for measuring distance[1] and has two main components: a transmitter and a receiver. The sensor sends out high-frequency sound waves, which bounce off of nearby objects and return to the receiver. By timing how long it takes for the sound waves to return, the sensor can calculate the distance to the object (see image below). It is commonly used in robotics, home automation, and security systems, as it is easy to use and relatively inexpensive.

Now let's take a closer look at Figure 34.1 to see the HC-SR04 main specifications and pinout.
We can see that the operating voltage of the sensor is 5V, so we can connect the VCC to the 5V pin of the Arduino UNO, and the GND to the GND. Then we use the Trig pin to send the ultrasound wave from the transmitter of the sensor, and we use the Echo pin to listen for the reflected signal. As we will only use the HIGH/LOW values for the Echo and Trig pins, we can connect them to any digital pin of the Arduino UNO (so for example pins 9 and 10 as shown in the image below).

Let's then inspect the protocol. In order to generate the ultrasound, we need to set the Trig pin HIGH for 10 µs. The sensor will react by sending an 8-cycle ultrasonic burst which will travel at the speed of sound. The Echo

[1] You can also use it to make a levitation device: https://www.youtube.com/watch?v=WZpdGN6YTdY.

Operating Voltage	5V DC
Operating Current	15mA
Operating Frequency	40KHz
Min Range	2cm / 1 inch
Max Range	400cm / 13 feet
Accuracy	3mm
Measuring Angle	<15°
Dimension	45 x 20 x 15mm

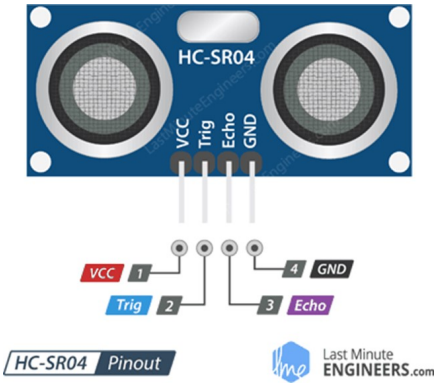


Figure 34.1: Main characteristics of the HC-SR04 (left) and pinout (right).

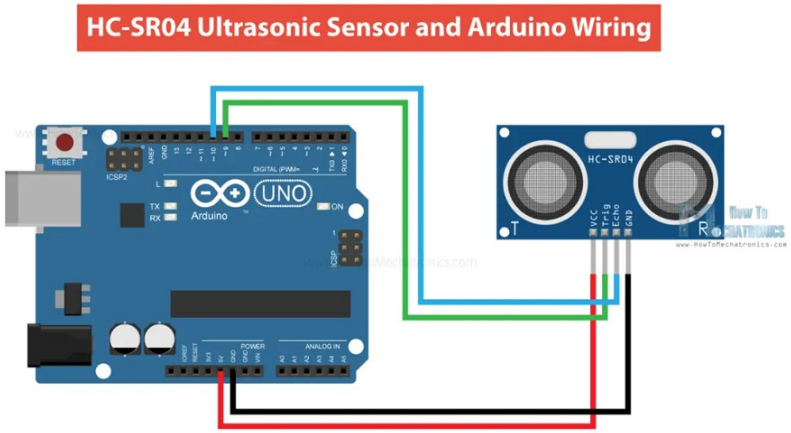


Figure 34.2: HC-SR04 to Arduino Uno wiring.

pins go HIGH right away after that 8-cycle ultrasonic burst is sent, and it starts listening or waiting for that wave to be reflected from an object.

If a reflected pulse is received the Echo pin will go LOW. Based on the duration for which the Echo pin was HIGH, we can determine the distance the sound wave traveled and thus the distance to the object. If there is no object or reflected pulse, the Echo pin will time out after 38ms and get back to the LOW state.

To calculate the distance of the object, we use this formula: $d = \frac{v \times t}{2}$, with v the speed of sound (= 34cm/ms), t the time is ms during which the Echo pin was HIGH, and divided by two to account for the round trip.

Now that we understand how the HC-SR04 ultrasonic sensor works, we

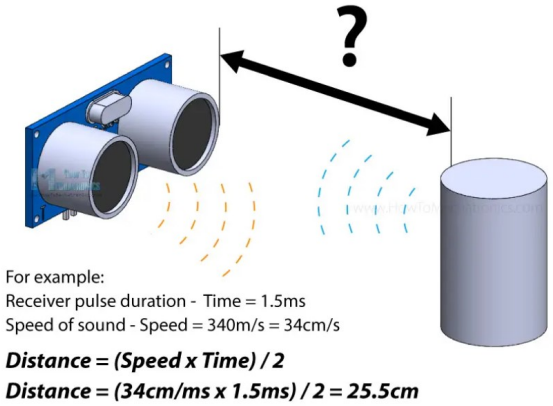


Figure 34.3: The HC-SR04 ultrasound protocol.

can get to the coding part.

```
// defines pins numbers
const int trig = 9;
const int echo = 10;

// We define the speed of sound, time, and distance variables
const float v = 0.034;
long t;
int d;

void setup() {
  // Set the trig pin as an Output to send the trigger signal
  pinMode(trig, OUTPUT);
  // Sets the echo pin as an Input to receive the echo signal
  pinMode(echo, INPUT);
  // Starts the serial communication
  Serial.begin(9600);
}

void loop() {
  // We first clear the trig pin, using the function digitalWrite
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
```

```
  // Now we set the trig pin on HIGH for 10 microseconds
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  // Then we read the echo pin and store the sound wave travel
  time in microseconds
  // PulseIn is a function that measures the duration of a pulse
  on a digital input pin and takes as an argument the pin to
  read and the state of the pulse.
  t = pulseIn(echo, HIGH);

  // Lastly we calculate the distance
  d = (t * v) / 2;

  // To see the resulting distance, we can use the Serial Monitor
  Serial.print("Distance to object: ");
  Serial.println(d);
}
```

34.1 Recommended Videos

https://www.youtube.com/watch?v=6F1B_N6LuKw
DroneBot Workshop
“Using the HC-SR04 Ultrasonic Distance Sensor with Arduino - Everything you need to know!”

Chapter 35

Making your own Sensors

Sometimes, the sensor you need isn't readily available. This may be the case because you have something particularly unusual in mind, or simply because the structural and mechanical characteristics of your thing make off-the-shelf sensors unsuitable.

For example, there are various types of rotary encoders available, but, typically, their design assumes that rotation is around a simple axis. If you are building a biologically-inspired thing, you will observe that your animal will have no simple and well-behaved joints. For example, the human body has numerous approximations of ball joints, but the balls aren't balls but are typically more egg-shaped. A human knee joint isn't the hinge joint we might presume it is, and the joint attaching our arms to our torsos (the shoulder) isn't even close to our shoulders but is located where the clavicle meets our thorax, in front of our bodies, below our necks. (Shoulders are extremely complicated.) So how do you measure rotation, angle, or path travelled in such a complex bio-mechanical system?

You may adapt an existing sensor or build a mechanical contraption to translate the complex motion into a simpler circular motion. We can call that contraption, together with the off-the-shelf base sensor, a sensor we created of our own.

But we may even go beyond this and start making our sensors from scratch. It can be fun, and the sense of achievement on success will be considerable.

35.1 Resistor-based Sensors

Remember from Chapter 33 that many sensors are simply variable resistors using a material that is sensitive to a physical quantity such as temperature or light intensity.

Let's first de-mystify resistors. Take a piece of paper and a very soft pencil (such as a 6B) and cover a 2×5cm rectangle thickly with graphite from the pencil. We are talking as thick a layer as the paper will hold, with a glossy, metallic-looking surface. Now take a multimeter, put it in resistance-measurement mode, and touch two points on this graphite surface with your probes. The resistance will vary with distance, up to something on the order of $10k\Omega$, a very good level for building sensors. If you created a very clean and consistent layer, your resistance will be growing quite precisely linearly with distance and will be very reproducible. If this works only intermittently for you, the points of the probes may be too pointy – try to change the angle of the probes to have a greater metal surface of the probes touch the graphite layer.

This is a form of distance sensor – you can read voltages and thus resistances using the analog input pins of your microcontroller. You can turn this into a rotary encoder by wrapping the sheet of paper around something round, even it it's not perfectly round (as in those biomechanical things we discussed above). Just think of a good solution for making continuous contact with your graphite layer. (Maybe using a spring?)

Of course, this is an idea to get you started tinkering, and not for creating a product. The piece of paper, and the graphite layer on it, will not survive long-term.

35.2 Amplifying Small Changes

Suppose you have a sensor that turns a physical quantity into resistance and thus a voltage. Often, even a considerable change of the physical quantity changes the voltage only slighty (this is particularly true for load cells and photoresistors). The voltages here aren't necessarily close to zero, but the changes are small. In that case, you need to precisely amplify the delta relative to a baseline voltage, to get a useful reading into your microcontroller. This is done with electronic components called amplifiers.

Note that amplifiers come with special complications regarding supply-

ing power (they usually require both positive and negative voltages relative to Gnd to be supplied to them. To give you a base understanding, let us talk a bit about operational amplifiers[1], which are a fundamental kind of electronic component that instrumentation amplifiers are based upon.

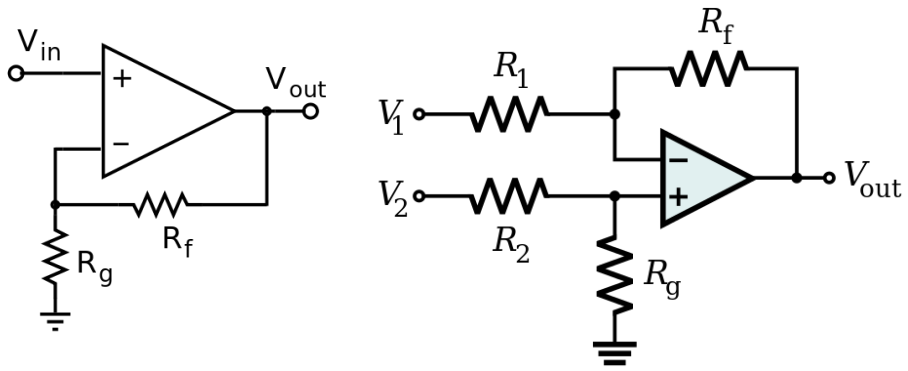


Figure 35.1: Using an OpAmp: Basic amplifier circuit (left) and differential amplifier circuit (right).

Discussing them in detail is beyond the scope of this section, though it is fascinating. Watch the video on them recommended below (the one by EEVblog).

In short, OpAmps can be used to achieve different functions based on the circuit they are put into. The basic amplifying circuit is shown in Figure 35.1 (left). Note that an OpAmp has additional pins for Gnd and to supply a positive and a *negative* voltage relative to Gnd. The two input, labeled + and - in the figure, are not these power supply pins but the so-called inverting and non-inverting inputs. (Do not confuse them with the power supply pins!) The circuit of Figure 35.1 (left) will amplify the voltage (signal) on V_{in} by a factor determined by the two resistances R_f and R_g, up to the supply voltages.

As for supplying power, we need to supply positive and negative voltages, relative to Gnd. We can do this with two batteries or islated power supplies connected in series, with ("floating") Gnd defined as the voltage level between the two power sources, of using a voltage divider, where V_{in}, V_{out}, and Gnd of the supplying circuit with the voltage divider become V_+, Gnd, and V_-, respectively, of the amplifier circuit. Be careful – we now

[1] OpAmps, see https://en.wikipedia.org/wiki/Operational_amplifier.

have two different things called Gnd and interpreted as Gnd in two different regions of our circuit, and if we connect them by mistake, we create a short-circuit! Note that this is a scenario we covered before, with the amplifier application in mind, in Figure 19.5!

Operational amplifiers can be used in a setup that makes them differential amplifiers (to amplify deltas, which is what we need), using the circuit shown in Figure 35.1 (right). However, for technical reasons (impedance matching), you need a slightly more complicated circuit (usually with three OpAmps and some resistors), which as a package is called an instrumentation amplifier. If you want to amplify very small voltages, or want to try out a differential amplifier for cheap, try an OpAmp. (OpAmps are much cheaper than instrumentation amplifiers, because they are produced in much larger volumes.)

For sensor applications, we use instrumentation amplifier boards[2]. The recipe for using your instrumentation amplifier depends on the actual product. We have a couple in store which are popular with makers – search the Web/Youtube for recipes and tutorials.

Another very different use of OpAmps that may be relevant to us is as buffers (see the EEVblog video). If we want to take a voltage (signal) and supply it with a large current without this interfering with the part of the circuit that voltage "comes from", we may use an OpAmp as a buffer. To the circuit providing the voltage, the OpAmp looks essentially like an isolator – no current flows through it, an still at the output side of the OpAmp, the same voltage is provided at the current we supply to the OpAmp.

Note that, if you only want to amplify DC signals (where the voltage level does not go below Gnd), no AC signals, some amplifier boards may not need a negative voltage supplied (you may connect the V_- pin to Gnd). Check in the datasheet. For a differential amplifier application, that is never the case, because the measured voltage may go above and below the reference voltage against which we compute deltas.

[2] https://en.wikipedia.org/wiki/Instrumentation_amplifier

35.3 Recommended Videos

https://www.youtube.com/watch?v=7FYHt5XviKc
EEVblog
“ EEVblog #600 - OpAmps Tutorial - What is an Operational Amplifier?”

https://www.youtube.com/watch?v=7zUYqQ6wUhA
ElectronX Lab
“Operational Amplifiers - Differential Amplifiers ”

35.4 Parts in Stock for CS358

Model	Type	CHF
AD8221AR	instrumentation amplifier board	10
AD620	instrumentation amplifier board	13

Part VI

Actuators

Chapter 36

Electromagnetism

Currently, the computer science undergraduate curriculum only contains a single physics course, which focuses on mechanics. In the past, the same curriculum contained much more physics, including an entire course on electromagnetism. The idea was that for you to be considered a solid engineer, knowing electromagnetism is indispensable. With the reduction of the undergraduate programs to three years (the Bologna process), the coverage of electromagnetism was removed. For most of what computer scientists do, it is not on the critical path. However, it is important to this course, for understanding and working with actuators.

As mentioned, electromagnetism can fill an entire course. This chapter restricts itselfs only to covering issues that you absolutely need to know to work with actuators – no deeper understanding of electromagnetism can be communicated here.

I strongly recommed that you read up on electromagnetism though. Start on Wikipedia:

https://en.wikipedia.org/wiki/Introduction_to_electromagnetism

I also recommend this video:

https://www.youtube.com/watch?v=XoVW7CRR5JY&t=671s
ScienceClic English
"The Electromagnetic field, how Electric and Magnetic forces arise"

One of the beauties of electromagnetism is that is allows to evaluate deep concepts, including fundamental forces and their unification, leading to the standard model of physics, elementary particles, fields, and special relativity in a context that is very familiar to us and which we all have expe-

rience. The applications could not be overstated. Without it, no chemistry, no biology, and no you. Without it, no electrical power generation, mobile phones, Wifi, or microwave pizza.

36.1 Inductance and Back-EMF

Electrical current flowing through a conductor creates a magnetic field. In practice, we want to make this effect significant by packing a lot of conductor (wire) in a small volume of space in a way that the contributions of sections of the conductor to the magnetic field do add up rather than cancel out – by keeping the sections of wire parallel and the current flowing in the same direction in these parallel pieces of wire. This can be achieved by winding up the conductor in a coil.

The strength of the magnetic field is proportional to the current flowing through the coil. This magnetic field opposes a change of current. How strongly it does that is expressed by a constant, called the *inductance* L (in Henry, or $V/(I/t)$, that is, volts per change of current), of the coil. The inductance value of a coil reflects the material in, and the geometry of, the coil.

The magnetic field built up by the coil while current flows through it stores some energy. This energy has to build up from zero when your circuit is powered up and has to be dissipated somehow when the current is reduced. The coil does that by producing an instantaneous voltage v opposing the change of current (trying to keep the current flowing, in the same direction, as before), called counter-electromotive force (or back-EMF) calculated as $v = L * di/dt$ (where i is the current).

It is important to note that this voltage only depends on the current flowing through the circuit but not on the voltage used to power the circuit. If we power up a circuit with a coil and then suddenly open the circuit (e.g., disconnect the power supply), the back-EMF voltage is theoretically infinite, and, in practice, may amount to tens of thousands of volts, leading to electricity arcing over where you open the circuit (creating sparks). These voltage spikes are extremely dangerous to other electronic components in your circuits (for instance, your laptop[1]), and can destroy them.

[1]Do not assume perfect protection of your laptop when using a USB isolator. These are usually rated to a few hundreds of volts, not tens of thousands!

To counteract this, you have to do at least two things:

- You need to keep the circuit permanently closed by placing a reverse diode across the coil, i.e., from the - end of the coils to its + end: because of the directionality of current flow through diodes, no current will flow through the coil while the power supply is on, and when it is turned off, the back-EMF flows through the diode from the - end of the coils to the + end, creating a circuit just of the coil and the diode.
- You need to create some way to dissipate the energy in the coil – by adding a suitable low-resistance, large-power resistor to it. For relatively low currents, this is optional because the coils can act as this resistor.

An alternative to using a diode and a resistor is to use a brushed motor driver such as the L298N, which does all of this and adds the ability to reverse the direction of the magnetic field.

36.2 Making Electromagnets

Making an electromagnet is a popular science experiment for children. It is simple – create a coil of laquered copper wire around an iron nail or a similar iron or steel item and connect it to an electric power source. This may give you the idea to make your own electromagnet or solenoid-based linear actuator.

While this can absolutely be done, be warned that making an effective electromagnet that serves a practical purpose beyond demonstrating the principle is tricky. The key points people usually underestimate are

- The core material is extremely important – steel is usually very bad, soft iron is better. The geometry matters, and good cores for electromagnets are assembled from plates rather than being made monolithically.
- You need extremely large numbers of windings of extremely thin wire, and the windings should be done in clean layers. Doing this winding by hand is impractical.
- The geometry of the electromagnet matters. You want the height of the coil, viewed abstractly as a cylinder, to be low. Good electromagnets look more like pancakes than like rods.

- You need high currents.

Do not just make an electromagnet. To be successful in creating an electromagnet with some holding power, you will need to learn a bit of theory. Start with this video:

https://www.youtube.com/watch?v=vHP-zq23uvE
Nick Electronics
"5 Tips To Make A Good Electromagnet / How To Calculate Electromagnet Force?"

We keep some electromagnets in the inventory. See below. Do not forget that you need at least a diode or a (brushed) motor driver to deal with back-EMF.

36.3 Solenoid Actuators

While *solenoid* is another word for a coil, you can buy things called solenoids that consist of a coil that pull sa freely moving iron core to its center when powered, creating an actuator that can push or pull loads.

This may appear as an attractive option for your projects, for instance for a thing that lifts small loads or pushes buttons. Be warned, though, that it is very difficult for you to find and order a solenoid up to the task that you intend for it. The reasons for this are the following:

- The range of motion during which a solenoid can produce relevant force is extremely short – usually something like one milimeter. So, if you, for instance, want to use a solenoid to push a button (for instance, the key of a keyboard or piano) 5mm deep, you generally cannot use a solenoid.

- Solenoid specifications are confusing. We had a couple of teams order solenoids in the past, and in every single case, the items ordered were not up to the task and could not be used.

- Solenoids are somewhat unusual items that are not very popular with makers. For that reason, there is only a narrow range of solenoids for very specific purposes (e.g. door locks) that are widely available. There are technical suppliers of solenoids of nearly any specification,

but these usually cater only to customers buying them in large numbers. They are often produced on order, and small orders are impractical or forbiddingly expensive.

- Solenoids are inefficient. Compared to a rotary motor, whose work for your purpose adds up over time (and which still can be translated into linear motion), solenoids need to do their work in extremely short time periods, while their push/pull rod moves a very small distance. Thus, relative to rotary motors, for similar tasks, solenoids need to work with much larger voltages and currents.

For these reasons, any use of solenoids in this course is *strongly* discouraged. Consider using a (rotary) motor with some translation to linear motion, for instance via a crank and crankshaft or a lead screw assembly.

36.4 Parts in Stock for CS358

We have some electromagnets; please ask.

Chapter 37

Electric Motors

Motors allow our projects to move and affect the physical world. In this chapter, we cover some formal foundations[1], as well as safety hazards.

37.1 Power, Speed, and Torque

TLDR: Motor speed is proportional to voltage; motor strength (torque) is proportional to electric current.

We use the standard symbols U for voltage, I for current, and P for power (see Figure 37.1). Of course, $P = U * I$ (Watt's law) holds in electronics in general, not just for motors.

[1] See also https://en.wikipedia.org/wiki/Electric_motor .

	symbol	unit	SI units
voltage	U	V(olts)	$1\frac{kg \cdot m^2}{As^3}$
current	I	A(mperes)	$1A$
power	P	W(atts)	$1\frac{kg \cdot m^2}{s^3}$
angular velocity	ω	radians per second	$1/s$
angular acceleration	a	radians per second^2	$1/s^2$
torque	τ	Nm (Newtonmeters)	$1\frac{kg \cdot m^2}{s^2}$
velocity constant	K_v(SI)	radians per second per Volt	$1\frac{A \cdot s^2}{kg \cdot m^2}$
torque constant	K_t	Nm/A	$1\frac{kg \cdot m^2}{A \cdot s^2}$

Figure 37.1: Symbol table

Let us first consider an idealized situation, where we disregard inefficiencies of the motor that cause some of the electrical power to be turned into heat rather than mechanical work. Then angular velocity ω ("rotation speed" in radians per second, where one full rotation is 2π radians) is proportional to voltage, and torque τ is proportional to current:

$$\begin{aligned} \omega &= K_v(\text{SI}) * U \\ \tau &= K_t * I \end{aligned}$$

where $K_v(SI)$ and K_t are called the velocity constant and the torque constant, respectively, with $K_t = 1/K_v(SI)$.

The (mechanical) power output in this idealized scenario is

$$P = U * I = \omega * \tau.$$

So when designing a motor to consume a given amount of power, there is a trade-off to be made between fast motors (maximize $\omega = P/\tau$; high-K_v motors) and strong motors (maximize $\tau = P/\omega$; low-K_v motors).

Low-K_v motors are built using coils of many turns of thin copper wire, while high-K_v motors are built from coils of few windings of thick copper wire. As a consequence, low-K_v motors have higher resistance in the coils than high-K_v motors, and need a higher voltage for the same current to flow. Ignoring friction and inefficiencies, low- and high-K_v motors produce the same amount of torque per Watt of power. In practice, if we need a high-torque motor, we prefer one with a low K_v rating. (This keeps currents low, creates less heat, and allows us to work with thinner cables.)

Motors with higher rotor diameters tend to "have" higher torque (the causality is reversed here); for that reason you can buy pancake-shaped low-K_v brushless motors. You can view this from a viewpoint of mechanics – a higher diameter-motor has a better lever to create torque – or better, a low-K_v motor needs more diameter/space for copper so that it can run at a certain power (since a greater length of wire is involved) than a high-K_v motor.

So yes, for a given desired power output, low-K_v motors tend to be heavier and thus more expensive, but cooling is easier (the heat resulting from inefficiencies spreads out over a greater volume) and you save on other components (such as cables).

Motors often come with a K_v rating in RPM/V (rotations per minute per Volt) reported in their specifications. You can compute

$$K_v(SI) = \frac{\pi}{30} \cdot K_v(RPM).$$

and

$$\tau = I * K_t = \frac{I}{K_v(SI)} = \frac{1}{\frac{\pi}{30} * K_v(\text{RPM})} * I = \frac{30 * I}{\pi * K_v(\text{RPM})}.$$

Usual K_v(RPM) ratings are between about 100 and a few thousands. The meaning is, of course, that the motor will rotate that many RPM for every Volt supplied (above a minimum voltage needed to overcome internal friction and make the motor turn at all). So, for instance, a motor with $K_v = 1000$ will rotate with 10000 RPM if supplied with 10V.

You cannot strictly rely on the K_v ratings reported by manufactures being exactly correct.[2] Also, in practice there is a minimum voltage needed for any motor (brushless or not) to overcome internal friction and turn at all. However, the inefficiency of the motor dominates these inaccurracies. You can try to see how these inefficiencies alter the above formulas, but, since the key relationships are linear, it is easiest to compute your quantity using the idealized formulas first and to apply a fudge factor in the end; for instance, if your motor is reported to be 80% efficient[3], and you compute a torque of 2Nm using the idealized formulas, multiply the 2Nm with 0.8 to get an estimate of actual torque.

37.2 Acceleration

A torque of $1Nm$ means that a motor, rotating a lever (an arm) of one meter length, exerts a force of $1N$ at the tip of the lever. Assume we have arranged the axis of rotation of our level (and our motor) perpendicularly to vertical. To resist the pull of Earth's gravity exerted by a mass of 1 kg attached to the tip of the (1 m long) lever, we need about 9.81 Nm of torque rotating the lever upwards. But what torque does it take to accelerate the rotation of a mass from standstill?

You first need to calculate the moment of inertia (unit: $kg{\cdot}m^2$) of the load you want to rotate. See e.g. https://en.wikipedia.org/wiki/Moment_of_inertia for this. Example: If the load is a cylinder/disk of uniform density centered on the axis of rotation, then the moment of inertia is

[2] Manufacturers typically measure K_v for their motors by running them under power, while the correct way to obtain a K_v rating would be by measuring back-EMF while turning the rotor with another motor.

[3] Brushed motors are typically 50% to 60% efficient, and brushless motors 80% to 90%.

$$J_L = m * r^2/2$$

in $kg \cdot m^2$ where m is the mass of the disk in kg and r is the radius.

A proper calculation of the torque needed requires you to consider a number of factors, but for a simplified, rough calculation of torque in Nm, use

$$\tau = J_L * a$$

where a (in rad/s^2) is the angular acceleration. (For example, if you want to accerate by a full rotation per second, $a = 2\pi/s^2$.)

37.2.1 The Case of Stepper Motors

Please first read the chapters on motor types and stepper motors and then return here.

Angular acceleration is a problem for stepper motors. In order for the stepper motor to do its job and not to skip steps, you need to accelerate your load from zero to the speed necessary to move the load by the angle of one step (1.8 degrees = $\pi/100$ radians by default) within the time of one step. By a very rough back-of-the envelope calculation, for a 17HS4401 stepper motor ($\approx 0.4Nm$ holding torque, and about 2 milliseconds maximal pulse width), this gives us a supported moment of inertia of about

$$J_L = \frac{\tau}{a} \approx \frac{0.4 * (2 * 10^{-3})^2}{\frac{\pi}{100}} \approx 5.1 * 10^{-5} kg \cdot m^2.$$

Manufacturers give the following rule of thumb for picking the size of a stepper motor: The moment of inertia of the load should not (much) exceed 10 times the *rotor inertia*[4] of the motor. For a 17HS4401 stepper, the rotor inertia is reported as $5.4 * 10^{-6} kg \cdot m^2$. The supported moment of inertia by this rough rule is 106% of the number calculated above – pretty close.

Example 37.2.1 In CS-358 2022, a team wanted to rotate a turntable sitting (against my advice) directly atop a stepper motor, rotating the turntable without any belt or gear reduction. The turntable had a radius of 20 cm and a mass of 1.5 kg (with much of the mass, such as a motor, far away

[4]This is a number that is reported in the datasheet of the motor.

from the axis of rotation). The moment of inertia thus was greater than $1.5 * 0.2^2/2 = 0.03kg \cdot m^2$. Even leaving aside the considerable friction of that turntable as well as other inefficiencies, by the above calculation, this was by more than a factor of $0.03/(1.2 * 10^{-4}) = 250$ too much! The 17HS4401 motor was able to turn the turntable, but it made a loud bang on starting and stopping, skipping lots of steps. The purpose of the stepper was precise positioning, so this was a failure.

37.3 Safety Hazards

37.3.1 Blunt Trauma

When we build robots, we need strong motors. We must over-provision the power of our motors compared to a biological system because our machine will be mechanically less sophisticated. Animals have bodies shaped and balanced to minimize energy costs, and can swing limbs or have internal spring-loading mechanisms[5] to reduce the need for strong actuation. If we don't want to make our machines too complicated, we need to use motors that can generate considerable forces. This makes our motors quite dangerous.

Even the kinds of robots people on youtube are creating at home use (brushless) motors with many hundreds or even thousands of Watts of peak power and gearboxes that create torque in the 20-50 Nm range.[6]

For comparison, the strongest torque humans generate is in the hip joint (which is served by the strongest muscle groups, particularly the gluteus maximus). A person of 70 kg mass walking at an average speed will generate brief peaks of 25Nm torque in the hip joint (more if they run or do sit-ups)[7]. An average walking human's power output is about 70W. Outputting power comparable to that of a moderately sized brushless motor requires superhuman physique, ideally a suitable myostatin gene mutation, and effort.[8]

[5]This is most impressive in mantis shrimps and pistol shrimps, which can move limbs at supersonic speeds thanks to spring-loading mechanisms.

[6]James Bruton's open dog v1/ robot requires at peak 10000W (three 6374 motors) of motor power *per leg*. See https://www.youtube.com/watch?v=cusoDUBzzAY .

[7]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4431254/

[8]https://www.youtube.com/watch?v=S4O5voOCqAQ

Figure 37.2: The Renault Twizy 45 (3730W) and a 6374 brushless motor from our parts catalog (3250W).

If you are unlucky, a robot arm doing a jerky movement at startup may kill you if you do not take sufficient safety precautions (keeping distance, in particular). Robot arms have killed people since 1979.[9]

The most powerful motors we have in store for the course have 3250W, comparable to a small car (see Figure 37.2), and with the right gear reduction, they can be used to tear down a building. Motors that you will be given for your project without special warning will be harmless (speaking of mechanical rather than electrical danger) unless you create substantial mechanical advantage (using e.g. gearboxes, see Chapter 45).

37.3.2 Current Draw

Not only can a sufficiently strong motor destroy stuff, but a motor that is too weak for a purpose (or is obstructed from turning) will suffer or get destroyed.

Oversimplifying things, voltage determines the rotation speed of an electromotor, while current determines its torque. Creating too much mechanical resistance while a motor tries to turn will cause a very high current to flow, heating up the motor, motor drivers, and cables, eventually causing damage or even a fire hazard. If you find there is mechanical obstruction to your motor turning, turn off your motor immediately. This is particularly important for servo motors with plastic gears, where in addition to heat damage, the gearbox has such a high reduction factor that it can destroy

[9] https://en.wikipedia.org/wiki/Robert_Williams_(robot_fatality)

itself. Here you must absolutely avoid this from happening even for a moment – double check that it is possible for the motor to turn freely before you power it on!

While the voltage is most directly connected to motor speed[10], the current a motor draws depends primarily on its load, i.e., how much mechanical resistance there is to it turning. If you supply a voltage to a motor but don't allow it to turn, the current will rise to a level where the motor will get very hot and will be damaged. Also, some gearboxes may not be able to handle high loads, and their gears be destroyed. This is particularly relevant for small servos with plastic gears.

Motors may draw considerable amounts of power (voltage and current), and you will NOT be able to use the voltage output pin of your microcontroller board to drive motors [11]; you will need a separate power supply. For some brushless motors, the peak current draw may be so high that, maybe unintuitively, it may be impossible to find a suitable wired power supply (i.e., one that is powered by mains voltage); LiPo batteries are the only suitable form of power source. Some such motors may draw as much as 100A, which is no problem supplying from larger LiPos, while the lab bench power supplies in DLLEL can supply no more than 3A.

37.3.3 Counter-Electromotive Force (Back-EMF)

All electric motors use coils. Coils store energy in the magnetic field, which builds up when they are powered. They try to keep a current that flows through them flowing. Thus, when power is turned off, so-called back-EMF[12] (a high negative charge on the positive pole side of the coil) is created, which will (at least) cause sparks to fly and nearby electronics to be destroyed.

Motor drivers handle this and protect you from back-EMF, but beware of a malfunctioning motor driver.[13]

There are other uses of coils in electric circuits, and we need to be concerned about those that build up considerable magnetic fields. This in particular includes electromagnets. If you want to use elecromagnets, you

[10]The impact of supply voltage on the functioning of stepper motors is more complicated; stick to the specifications.

[11]If you try it, you may damage your microcontroller board.

[12]See https://en.wikipedia.org/wiki/Counter-electromotive_force .

[13]The A4988 is an infamous offender, see Section 39.2.

must provide appropriate protection against back-EMF. Talk to an expert. The solution will involve a properly placed diode.

37.3.4 Radio Frequency Interference

Motors may also create significant *radio* noise. This noise isn't audible to you but may interfere with wireless communications, including communications that you do as part of your project.[14] For that reason, it may be technically illegal to operate your motor without first addressing this issue. Unfortunately, the electronics skills required to do this to make a commercial product are beyond the scope of this course.

Radio noise is a particularly serious problem for brushed motors. It is less of a problem in brushless motors.

For brushed motors, a partial solution to the problem is to solder a small ceramic capacitor (0.01 to 0.1 μF) across the two motor terminals, if your motor does not already come with such a capacitor soldered on (see e.g. https://www.pololu.com/docs/0J15/9).

[14] See https://en.wikipedia.org/wiki/Electromagnetic_interference .

Chapter 38

Motor Types

In this chapter, we cover motors and the motor drivers[1] that go with them. You will find that programming microcontrollers to operate actuators is relatively easy. The main challenge is picking the motor appropriate for a purpose, and correctly wiring it up and supplying power to it. As for picking a motor, we may have requirements regarding minimum torque[2], RPM (rotations per minute) range, supply voltage, current draw, weight, dimensions, and price.

We will consider three classes of motors: brushed, brushless, and stepper; these are based on fundamentally different designs, with many consequences.

38.1 Brushed Motors

The simplest kind of electrical motor is the brushed DC motor. Brushed motors internally use graphite brushes to transmit power from the stationary outside of the motor (the stator) to the rotor, which has a coil mounted in it to which we need to supply power. These brushes cause friction (reducing their efficiency) and erode and fail over time. Small brushed motors (see Figure 38.2) can be very inexpensive, but be careful with the metal pads/rings to which the two wires need to be soldered. They are easily torn off, rendering the motor unusable. Don't pull on the wires!

[1] These are electronics boards, not software drivers.

[2] Torque, measured in Nm (Newtonmeter) by anyone other than the Imperials (who use pound foot or stone yards, but not pound feet or foot pounds), is the rotational analog of linear force. To learn more, see https://en.wikipedia.org/wiki/Torque .

Figure 38.1: A brushed motor (top), a brushless motor with an external rotor (center), and a bipolar stepper motor (bottom), all opened up.

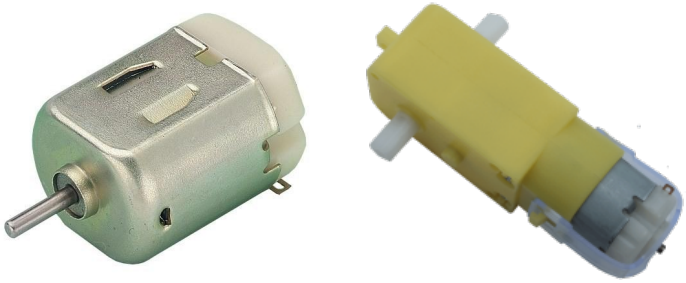


Figure 38.2: A small brushed motor (left) and the same type of motor with a gearbox (right).

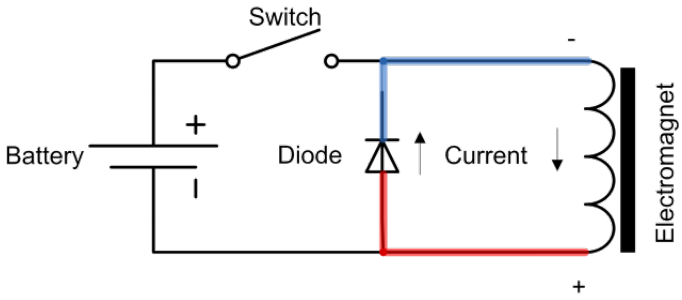


Figure 38.3: A back-EMF protection circuit using a diode. The motor is represented by the wiggly line representing a coil, labeled “electromagnet”.

Brushed motors have low torque (about 0.001 Nm for the motor in Figure 38.2 (left)), and require gearboxes for virtually any application we might encounter. “Yellow gear motors” as shown in Figure 38.2 are popular; their specs are 3-9V, 110 RPM (at the output shaft of the gearbox), 1:48 reduction in the gearbox, 2.6 Ncm torque.

38.1.1 Motor Drivers: H-Bridges

Brushed motors have two wires by which you can directly attach them to a DC power source, with the direction of current determining the direction of rotation. However, this is not a safe thing to do. When turning off power,

back-EMF will be generated[3], potentially damaging electronic components in the same circuit. Also, typically, a spark is created that may damage things closeby, even if not in the same circuit, and which may cause a fire. You can avoid this by placing a diode as in the circuit of Figure 38.3. By this approach, the motor can only ever turn in one direction; to reverse the direction of rotation, you need to manually change the wiring.

Now suppose we want to control motor direction and speed by signals from a microcontroller. For this we need a motor driver, a dedicated electronic component. Note that brushless motors and stepper motors also need motor drivers, but they are fundamentally different. You cannot use a brushless motor driver for a brushed motor, and vice versa.[4]

The core circuitry of such a brushed motor driver is a so-called H-bridge[5], which allows us to control the direction of current. The key functional units are four transistors, through two of which flows our motor current while the motor is running. We distinguish motor drivers based on the transistor technology used.

Motor drivers that use bipolar transistors have a voltage falloff at the transistors of 1.4V in total. Thus, if you want to run your motor at, for example, 6V, you need to supply 7.4V to the motor driver. Because of this falloff, a significant fraction of the power supplied is turned into heat at the motor driver. Motor drivers that use MOSFETs have no significant voltage falloff and heat up much less, but they are (usually) larger, more expensive, and need to be protected from electrostatic discharge (do not touch the electronics).

Both L298N and MX1508 take two input signals per motor. Both are PWM signals which set the speed for each direction (using analogWrite() between 0 and 255). At least one of the PWM signals has to be zero at all times; motors cannot rotation in both directions at the same time.

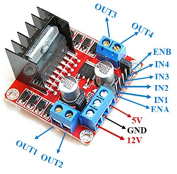
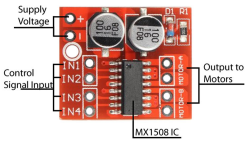
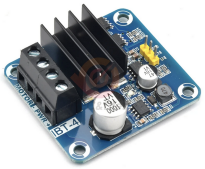
38.1.2 Recommended Videos

https://www.youtube.com/watch?v=ygrsIqWOh3Y
DroneBot Workshop
“Driving DC Motors with Microcontrollers”

[3] To learn more, see https://en.wikipedia.org/wiki/Counter-electromotive_force.

[4] It is possible to drive certain stepper motors using *two* brushed motor drivers, but this is complicated. Do not do it.

[5] To learn more, see https://en.wikipedia.org/wiki/H-bridge

			
	L298N	MX1508	IBT-4
technology	BJT	BJT	MOSFET
# motors driven.	**2**	**2**	1
supply voltage (V)	7-35	2-10	5-15
max current (A)	1.5-2	1-1.5	**50**
logic level (V)	5	5	3.3-12
dimensions (mm^3)	49x55x33	**24.7x21x5**	43x48x23
weight (g)	33	**2**	60(?)
wire conn.	screw mount	soldering	screw mount
pros	**5V output**	very small size	amps!!!
price (CHF)	5	5	6

https://www.youtube.com/watch?v=I7IFsQ4tQU8
How To Mechatronics
"Arduino DC Motor Control Tutorial - L298N | H-Bridge | PWM | Robot Car"

38.2 Brushless Motors

Brushless motors (aka BLDC – brushless DC – motors) use a different principle where the coils are in the stator and no brushes are needed. In these motors, the outside housing holds the permanent magnets and is part of the rotor, so the motor can only mounted using one of its faces (in the brushless motor picture above, that's the end of the stator from which the cables protrude). These motors can be very efficient, i.e. a very large fraction of the power (Watts) used is translated into mechanical work, rather than heat. Brushed motors are typically 50-55% efficient[6], brushless motors 80-90%.

Brushless motors are popular in things that fly (model airplanes, multirotor helicopter drones, etc.) and mobile robots because they have extremely high power density (power to weight ratio). However, keeping them

[6]Watts of mechanical work per Watts of electrical power supplied.

cool is a challenge. Even at 90% efficiency, a 3000W motor produces 300W of heat. The only way some motors can be kept cool enough not to be destroyed when they run at maximum power is if they are in the downwash of a helicopter rotor, which is exactly what they are designed for. So depending on your application, not all of a motor's advertised power may be available to you because you have no good-enough way to cool it.

38.2.1 Motor Characteristics

The statements made in Section 37.1 hold for other kinds of motors beyond brushless motors, but are particularly important for brushless motors. Unlike brushed motors, which supply torque so low that we realistically always need high-reduction gearboxes to use them, brushless motors may have sufficient torque to be used without gearbox or with only a mild belt reduction. However, motor torque is usually not indicated in the datasheets, or may be very unreliable. You need to calculate it from the K_v rating of the motor, which may be the singly most important characteristic of a brushless motor, and is always indicated.

You can practice calculating torque using the examples of Figure 38.4. For instance, the (idealized) torque of the 3536 motor can be computed as

$$\tau = \frac{30 * I}{\pi * K_v(\text{RPM})} = \frac{30 * 35}{\pi * 1200} Nm \approx 0.28 Nm.$$

38.2.2 Brushless Motor Drivers (ESCs)

Brushless motors have three connections for power supply, through which a complex pulses of power are supplied in a complex pattern; a motor driver (called ESC) is absolutely needed. In addition, some ("sensored") motors have additional connections through which the motor driver can read out information on rotor position and rotations performed.

The timing of the pulses to be emitted by the ESC depends on the specifications of the motor. ESCs can obtain these timings in two ways:

- measuring back-EMF from the coils. All ESCs do this, but cheap ones have only this method, and as a consequence, motors may stutter at startup and at low-RPM operation.
- tracking rotation – this requires a sensored motor and an ESC that supports it.

			
	(Racerstar) 3536	Flipsky 6384	T-Motor MN5008
type	BLDC outrunner	BLDC outrunner	BLDC outrunner
from spec:			
Kv(rpm/V)	1200	190	170
#pole pairs	7	7	14
phase resistance	0.025Ω	0.05Ω	0.27Ω
#LiPo cells	2-4S	4-13S	6-12S
Dimensions	35x35x36mm	63x63x84mm	56x56x32mm
Weight	115g	1100g	128g
max current	35A	95A	15A (180 sec)
price (CHF)	15	100	90
sensor	no	yes, Hall	no
calculated:			
max voltage	16V	52V	48V
:w max power	560W	4940W	720W
max rpm	19200	9880	8160
K_v(SI)	125.7	19.9	17.8
K_t	0.008	0.0503	0.0562
torque	0.28Nm	4.77Nm	0.84Nm
advertised:			
torque	-	9Nm	-

Figure 38.4: Three brushless motors: A drone motor (left), a (sensored) motor for electric bikes and skateboards (center), and a pancake motor intended for drones but popular in robotics applications (right). The calculated values (unreasonably) assume 100% efficiency.

Note that any usable brushless motor driver (that is, one that can drive loads) needs to be able to do some sensing to know where the coils are relative to the permanent magnets at any point in time. This feedback is necessary because the power pulses need to be sent to the coils at exactly the right moment for the motor to rotate one, creating torque. A motor driver with no feedback component whatsoever could conceivably rotate a motor if it could be brought up to the right speed by an external impulse, using a fixed cycling of pulses, but any load would stop the rotation, and while the motor would consume substantial current, it would produce nearly no torque. How can this be possible given that we said in Chapter 37 that torque is proportional to current? Answer: the current would be used by the motor to fight its own rotation. The torque would not be used to push the load, but to brake the motor's rotation. It's not braking that we can feel, because feeling that a motor brakes (opposes us) would require it to be able to sense that we are trying to turn its axle.

Brushless motor drivers can be quite complicated and expensive, and some have advanced features such as an interface through which they (and the motors) can be tuned from a computer. Many take feedback from the motor in the form of back-EMF and through rotary encoders to understand what the motor is doing and its performance. Some drivers have specific functionality for braking, for harvesting the brake energy and charging the battery with it, and some have stepper-like functionality, or have current- or torque control (rather than speed control) modes.

Three open-source brushless motor driver projects worth knowing of are VESC for electric skateboards and ODrives and SimpleFOC for robotics. All three support sensored motors and have very rich sets of features. Even though open-source designs, the former two motor drivers are very expensive (well above CHF 100 for a single motor) and are practically unavailable to us in this course. SimpleFOC is primarily a software library, and there are affordable motor drivers to go with it. All three are FOC controllers[7], though only the latter two, ODrives and SimpleFOC, are designed for robotics applications.

38.2.3 Gimbal Motors

When shopping around for brushless motors, you may come across the notion of a gimbal motor. Gimbal motors are just a subclass of brushless

[7] See Chapter 43 on that topic.

motors that are at the low-kv end of the spectrum: so usually they have relatively high numbers of windings per coil, with thin wires, and support relatively low maximum currents but offer relatively high torque. Gimbal motors frequently have (mild) pancake shape – that is, their diameter is relatively high relative to their height. Also, frequently, gimbal motors are rather small, and images may be deceptive. Certainly check their dimensions before you order one!

Gimbal motors were developed as actuators for camera gimbals, and are also popular in drones. The most high-performance brushless motors are *not* gimbal motors, or else you may be disappointed when you receive it!

Brushless motor controllers that are designed specifically for gimbal motors must not be used with non-gimbal motors since the high currents flowing would make the motor drivers overheat.

38.2.4 Recommended Videos

https://www.youtube.com/watch?v=uOQk8SJso6Q
How To Mechatronics
"How Brushless Motor and ESC Work and How To Control them using Arduino"

https://www.youtube.com/watch?v=-mLuU1Nscu4
Skyentific
"Why the brushless controllers are awesome for robotics"

38.3 Stepper Motors

Stepper motors are brushless, even though the mechanism is different from so-called brushless motors. For steppers, the stator with the coils is on the outside and the rotor with the permanent magnets is on the inside; for brushless motors it is (typically) the reverse. Also, they are wired up differently; brushless motors have only three wires, with three coils connected at one end in a star formation, so they are controlled differentially, while the steppers we are using have two independent coils. Most importantly, the magic of steppers happens in the grooves and ridges you can see on the rotor and the inside of the stator, and how they are magnetized is key.

They achieve precise stepping without any sensor or feedback loop, just thanks to their electromechanical principle.

Stepper motors are "current-driven". Their voltage ratings (in the datasheets) are quite low, and are quite meaningless in practice. Stepper motors can often handle high voltages (even quite dangerous voltages for bigger stepper motors), but the voltages don't make a big difference to their operation.

Stepper motors typically consume significant power even when at rest; in that case they actively "brake" to lock their output shaft in position. If the application does not require this, it is good practice to power off the stepper when at rest[8], particularly when the motor is supplied by a battery.

We will discuss the most popular class, "hybrid" two-coil stepper motors (bipolar steppers). Look elsewhere for how they work, it's quite elegant. Abstractly speaking, they have of two coils and expose four wires. Note that there are also unipolar steppers (see Chapter 40).

For our purposes, bipolar stepper motors such as the 17HS4401 are relevant. This is a NEMA17 stepper motor (see Chapter 44). These are also used in many 3d printers, such as the Prusa i3 mk3s. NEMA17 motors are usually in the 0.4 to 0.6 Nm torque range. It is typical for NEMA17 stepper motors to have 200 steps per rotation, or 1.8 degrees per step. Using a technique called microstepping, this resolution can be substantially increased (usually by factors of 2, 4, 8, and 16, and for some stepper drivers even by a factor of 32).

For small steppers, small inexpensive drivers ("stepsticks") like the A4988 are popular. For any larger motors, more powerful drivers are needed, which quickly get quite large and expensive.

There are many things to say about using bipolar steppers, and several safety hazards. So one combination of bipolar stepper and driver, the 17HS4401 and A4988, available to us in this course, is covered in a separate chapter, 39.

38.3.1 Recommended Videos

https://www.youtube.com/watch?v=7spK_BkMJys
How To Mechatronics
"Stepper Motors and Arduino - The Ultimate Guide"

[8] Set the $\overline{\text{ENABLE}}$ pin to HIGH.

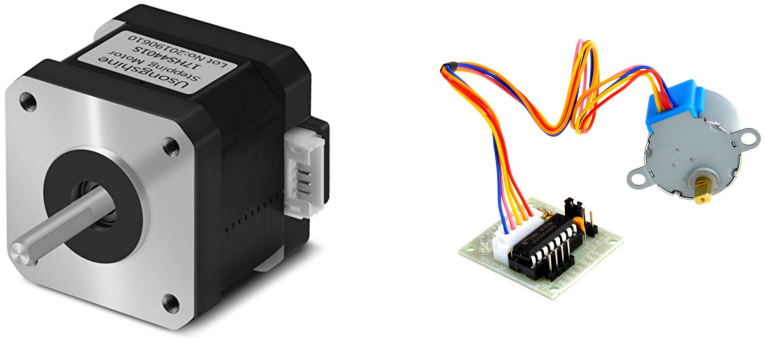


Figure 38.5: A (bipolar) 17HS4401 stepper motor (left) and a (unipolar) 28BYJ-48 stepper with its motor driver ULN2003 (right).

https://www.youtube.com/watch?v=0qwrnUeSpYQ
DroneBot Workshop
"Stepper Motors with Arduino - Controlling Bipolar & Unipolar stepper motors[a]"

[a]Covers both NEMA17 and 28BYJ-48 steppers.

38.4 Inrunners vs. Outrunners

We distinguish inrunner and outrunner motors. Inrunners are motors in which the rotor is fully enclosed and only the axle of the rotor is exposed. Outrunners are motors where the rotor is on the outside and rotates around the stator.

Brushed motors and steppers are, by design, always inrunners, and some brushless motors are, too. (Industrial brushless motors are inrunners or at least have their rotor enclosed for protection from the elements, dust, etc.)

The majority of brushless motors are outrunners. This is particularly true for very high power-density motors, typically drone and aircraft motors, which have cutouts in the rotor, exposing the internals of the motor to the rotor downwash for cooling.

38.5 Parts in Stock for CS358

Class	Model	Specs	CHF
brushed gear motor	"yellow" gear motor	see chapter	3
	Chihai CHF-GM37-550ABHL	12V 90RPM 3.4Nm w. enc	25
brushed driver	L298N	see chapter, for 2 motors	5
	MX1508	see chapter, for 2 motors	5
	BTS7960	5.5-27V 43A, for 1 motor	7
	IBT-4	5-15V, 50A, for 1 motor	5
ESC	(bidirectional, nonsensored)	40A	10
	(bidirectional, nonsensored)	80A	20
	FSESC v4.12	50A sensored	86
stepper + driver	28BYJ-48 + ULN2003	see chapter	2
	17HS4401 + A4988	see chapter	15

See also Chapter 43 for more brushless motor drivers.

Brushless motors:

Model	maxV	maxA	pp	K_v	phΩ	ph ind	grams	CHF
YT2804 (gimbal)	12	?	7	320	large	?	34	17
Sunnysky x2212	12	15	7	980	0.086	1.373e-5	?	30
Racerstar 3536	16	35	7	1200	0.025	3.00e-6	115	15
Racerstar 5065	48	34	7?	140	?	?	480	70
T-Motor MN5008	48	15	14	170	0.27	?	128	90
Eaglepower 8318	56	58	20	100	0.055?	?	599	80
Flipsky 6384	52	95	7	190	0.05	?	1100	100

Note: YT2804 includes an AS5600 encoder. Sunnysky x2212 includes a TLE5012b encoder. Flipsky 6384 includes a Hall effect sensor.

Chapter 39

Bipolar Steppers: 17HS4401 + A4988

This chapter is **required reading** if you want to use bipolar steppers.[1]

If you do not follow the instructions of this chapter, you are acting negligently and will, with very high likelihood, destroy the electronic components of your thing and possibly even more. You will also create very difficult to find problems in your thing, causing your team significant frustration and time loss. Once your electronic components are destroyed, you will have your walk of shame to the teaching staff, and should we give you replacements, you will have to spend time rebuilding your thing.

The A4988 stepper motor driver is very well matched with the stepper motor 17HS4401 (see Figure 38.5 (left)), a NEMA17 bipolar stepper motor rated at 1.5A current draw. The A4988 can handle voltages up to 35V and currents up to 2A in total, for two coils. The two coils do not draw maximum current at the same time, so steppers that draw up to 1.5A per coil are fine. The 17HS4401 has a torque of about 0.4Nm (without using a gearbox).

Note that the A4988 is very sensitive to misuse. If you damage it, it will not simply "not work" but it will turn evil and become outright dangerous. It may work intermittently work normally and from time act out of specification; in the worst case creating very large voltage spikes[2] or allowing

[1] Bipolar stepper motors other than the 17HS4401 are strongly discouraged, but in any case, most of this chapter also applies to other bipolar stepper motors.

[2] Even if your electronic components are not destroyed, these voltage spikes may make other components apparently unrelated to the operation of steppers work erratically,

very large current to flow. You may damage it and only observe strange behavior at a later date, making your system hard to debug. Apart from the dangers arising from a malfunctioning A4988, this can cause you great time loss and frustration.

Avoid this by following the instructions of this chapter rigorously. These motor drivers are inexpensive. If your merely suspect that you may be dealing with a malfunctioning A4988, try replacing it and see if your problem persists. Make it easy to replace your A4988s. Do not solder them to anything. Create sockets that make it quick to swap them out (or use a CNC shield or RAMPS board, which has sockets for the motor drivers).

39.1 Setup

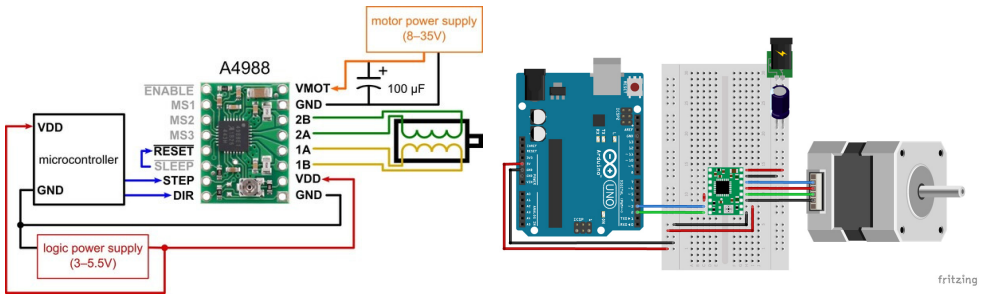


Figure 39.1: Wiring up an A4988.

The A4988 comes on a very small board[3], but it contains everything needed to drive one 17HS4401 stepper motor. The direct way to wire it up is shown in Figure 39.1. The alternative is a board called a CNC shield, which stacks nicely on top of an Arduino Uno and which can house up to four A4988s, driving up to four steppers, dramatically simplifying your wiring. See tutorials online.

Stepper motor drivers like the A4988 have a tuning potentiometer in the form of a small metal screw to set a current limit; you must set this correctly before operating a motor, or else you may cause a fire. This is not perfectly easy, and it requires a multimeter. For the 17HS4401, the voltage between the tuning screw of the A4988 and motor supply ground

adding to the difficulty of debugging your system.

[3]Please read more on it at https://www.pololu.com/product/1182.

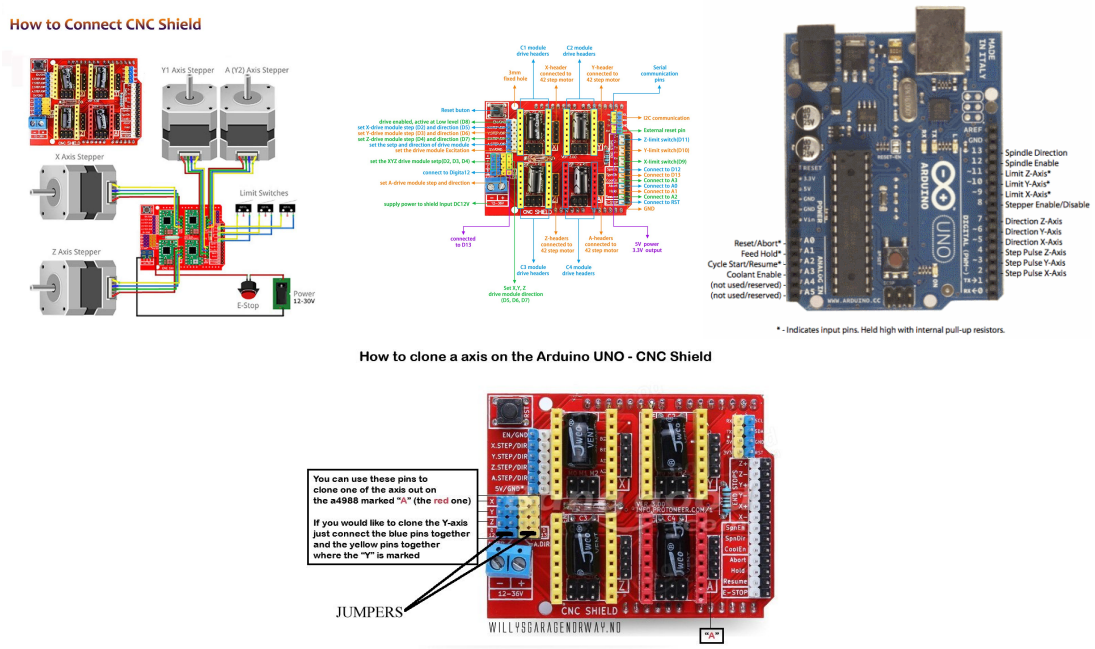


Figure 39.2: Using the CNC Shield

has to be set to 1.5 * 0.8 = 1.2V. (See also the first of the recommended videos in Section 38.3 on how to do it.)

39.2 Safety

In addition to all the safety hazards covered in Section 37.3, there are a number of additional hazards that you need to avoid when working with the A4988.

First, of course, the A4988 may draw several Amperes of current, and you must use 1.5 mm^2 wires to supply motor power, both to the A4988 motor power pins and to a CNC shield's motor power pins if you are using one. If you use jumper wires instead, they will go up in flames, it will be spectacular!

An A4988 may fail (get damaged). When this happens, it does not simply not function, but its behavior is undefined. It might do unexpected and dangerous things, such as creating an internal short-circuit, allowing very large circuits to flow, or making the motor coils produce extreme back EMF voltage spikes that run through the system and **damage or destroy** all the connected electronics, including **your laptop**! Unfortunately, this is not just a theoretical possibility: it has happened to a CS358 student in 2023. The laptop was judged a total loss by Poseidon!

To minimize the likelihood of damaging your A4988, make sure that your wiring is correct before you power it up. In particular, **you must never connect or disconnect the stepper motor to/from the A4988 while it receives power**! If you do this, the A4988 gets damaged with near 100% certainty. Always shut down power first before changing the wiring.

If one of the cables between your motor and motor driver has a loose connection or is damaged, the result is the same as connecting or disconnecting the motor while the motor driver is powered. To avoid this, ensure good cable management, and don't

In my experience, it is not enough for you to know this. In the heat of getting your thing to work, it will eventually happen that you break this most important rule. Agree with your team on a workflow for testing and tinkering with your steppers that ensures that your never break this rule.

The best (and perfect) protection to your laptop is to disconnect it between programming the microcontroller and powering up the A4988. Of

course, this is inconvenient, and it makes it impossible to receive debug output from the microcontroller via USB serial while your thing is running and being tested.

It is strongly recommended that you run your laptop on battery while working with an A4988, that is, NOT have a ground connection of your laptop through the power supply cable and the mains voltage plug.

If you choose to have your laptop plugged in to power, you are taking an extra risk. In that case, when working with the A4988, **you must protect your laptop using a USB isolator**, which you must **include in your bill of materials** and which we will provide. Note that such a device is not a perfect protection against extreme voltage spikes. However, it may just save your laptop and USB port, so please always use it while working with the A4988. In addition, it eliminates the possibility of a ground loop (see Chapter 19).

If you have a USB isolator, please use it even if your laptop is running on battery.

The A4988 is sensitive to voltage spikes coming from the power supply, which may make it work erratically, and it may also pass through voltage spikes produced by the back-EMF of its stepper motor to the rest of your circuit. These voltage spikes can exceed 35V when you supply it with 12V of power. These can destroy your electronics! The $47\mu F$ capacitor which you find in circuit diagrams and which must be places as close to the A4988 board (wire-wise) as possible, is meant to protect against this (and the CNC shield already has these capacitors on the board). It you think you are dealing with voltage spikes, talk to an expert.

39.3 Programming

There are many good stepper motor tutorials online [4]. Look there first.

You can program steppers directly or through a library such as AccelStepper. In the direct method, you achieve one step (there are 200 steps per rotation with the 17HS4401, assuming microstepping is disabled) using code such as

```
digitalWrite(stepPin, HIGH); delayMicroseconds(tHIGH);
digitalWrite(stepPin,  LOW); delayMicroseconds(tLOW);
```

[4] See https://lastminuteengineers.com/a4988-stepper-motor-driver-arduino-tutorial/, https://www.makerguides.com/a4988-stepper-motor-driver-arduino-tutorial/

where we recommend numbers close to tHIGH=100 and tLOW=1000 for smooth movement of the 17HS4401. The sum of tHIGH and tLOW determines the duration of a step and thus the speed of rotation (one full rotation takes 200*(tHIGH + tLOW) microseconds). Making the numbers significantly higher or lower will significantly increase noise and vibration, and for extreme values the motor will not be able to turn at all or will make seemingly random movements[5].

39.4 Stepping by PWM Signal

Presented next is a somewhat esoteric method of stepping that is not covered in the various web tutorials.

The signal we create to make the stepper do multiple steps is essentially a PWM signal. Using this observation, and PWM-enabled pins, we can make the stepper turn using analogWrite, as shown in the following program code.

```
const int enPin=8;
const int stepPin[4] = { 3, 9, 10, 11};
const int  dirPin[4] = { 2, 4,  7, 12};

void setup() {
  // set
  TCCR1B = TCCR1B & B11111000 | B00000011; // pin 9 and 10
  TCCR2B = TCCR2B & B11111000 | B00000011; // pin 3 and 11
  // to 32 * 1000 / 31250 = 1024 microseconds per step

  pinMode(enPin, OUTPUT);
  for(int i = 0; i < 4; i++) {
    pinMode(stepPin[i], OUTPUT);
    pinMode( dirPin[i], OUTPUT);
  }
}

inline void go(int stepper) {
```

[5] One example in https://www.makerguides.com/a4988-stepper-motor-driver-arduino-tutorial/ uses tHIGH = 2000 and tLOW = 2000, which does NOT work.

```
  analogWrite(stepPin[stepper], 25);
  // changing 25 does not affect speed of rotation
}

inline void stop(int stepper) {
  analogWrite(stepPin[stepper], 0);
}

inline void waitRotations(float rotations) {
  delay(204.8 * rotations);
  // one full revolution takes 1.024 * 200 = 204.8 ms
  // because of the casting to int, this will not
  // be exact
}

void loop() {
  digitalWrite(dirPin[0], HIGH);
  digitalWrite(dirPin[3], HIGH);
  digitalWrite( enPin,     LOW); // enable

  go(0); go(3); waitRotations(1);

  digitalWrite(dirPin[0], LOW);
  digitalWrite(dirPin[3], LOW);

  waitRotations(1); stop(0); stop(3);

  digitalWrite(enPin, HIGH); // disable, allow to cool
  delay(2000);
}
```

In this example program, we are supporting four steppers, but only use two of them (steppers 0 and 3). The program runs both steppers for one rotation, changes direction, then runs them for another rotation, and then pauses for two seconds.

The first two lines of setup() are specific to the Arduino Uno; we change the frequency of PWM for the pins 3, 9, 10, and 11 to 10000000/1024 = 976.5625 Hz, which is the closest-possible setting to the approximately

1000 Hz required by the A4988.[6] The Arduino Uno has two further PWM pins whose frequencies can be changed by setting TCCR0B; however, this timer is used for delay(), so changing it would change the semantics of delay() and all the timings in your code.

The upside of this technique is that we do not need to actively loop to keep doing steps; instead, we can do other work. There is separate functional unit in the microcontroller for creating PWM signals, so we take real work out of our programmed thread, which is better than the interrupts used in stepper libraries. Also, this technique results in smooth and quiet operation of the steppers.

The downside of the technique is that it may be tricky to ensure that we do exactly the number of steps we want. Thus it isn't suited for making plotters or 3D printers, but it is suitable for a vehicle that is using steppers for precise movement. There is also no way to change the speed of rotation (other than enabling microstepping).

Note also that we disable the steppers for the two-second pauses. This is good practice if we do not need our steppers to brake while not rotating. This saves energy and keeps the steppers and drivers cooler.

39.5 Troubleshooting

The stepper does not move at all. Ask yourself these questions:

- Is the circuit receiving power?
- Did you wire up your stepper circuit correctly?
  - Is the microcontroller talking to the A4988 through the right pins actually wired to STEP and DIR of the A4988?
  - Did you connect +12V and GND from the power supply to the correct pins – the motor supply pins?
  - Did you confuse the motor supply pins with the logic-level power supply pins (which should be connected to the microcontroller)?
  - Did you create the wire link between the $\overline{\text{RESET}}$ pin and $\overline{\text{SLEEP}}$?

[6] https://microcontrollerslab.com/arduino-pwm-tutorial-generate-fix-and-variable-frequency-signal

- Is the A4988 damaged? (Did you ever connect or disconnect the motor while power was on, create a short circuit, confuse +12V with GND, or confuse the logic level supply pins with the motor power supply pins? Did the A4988 ever overheat?)

Vibration or "Random walks". We have seen it happen quite frequently in CS358 projects that, in a correctly wired-up system with steppers and A4988 drivers, the steppers vibrate heavily or even rotate and change direction randomly. When this happens, motors and drivers also heat up more than they should. Here are some thoughts on what may be going wrong; I suggest to investigate these sources of trouble in the order given here.

1. Did you mistakenly switch the STEP and DIR pins?
2. Are all STEP and DIR pins of your A4988s which are connected to motors/your CNC shield also connected to a signal source such as a microcontroller I/O pin set to pin mode OUTPUT? Remember from Chapter 19, if that is missing, your STEP and DIR pins act as antennas, and the signal they read is random, and can change randomly anytime.
3. A timing problem in the code. The stepper drivers are very sensitive to the pulse length and timing through the STEP pin. If you are using your own low-level code for setting the STEP pin HIGH and LOW, your delay times may be wrong. This may be the case for code obtained from the Web, too, and even libraries. See Section 39.3. If you are directly setting delays, try experimenting with different delays. If you are using a library such as AccelStepper, try changing the motor speed. A stepper will vibrate when it runs slowly, because it will stop and restart at a high frequency to do its stepping at the speed you set. Try increasing the stepping speed and see if this reduces the vibrations. If you try to rotation your motor axle too fast, this will usually result in no rotation at all (potentially vibrating while being stuck in place), or a random walk (rotation that changes direction frequently). There is an ideal speed at which your motor moves fastest and smoothest. Try to find it.
4. The power source is not good enough. The A4988 is very sensitive to voltage spikes (thus the capacitor in the circuit schematics, which

may not suffice to filter voltage spikes). Switching mode power supplies (and buck converters, which use the same principle[7]) create such spikes. Try a lab bench power supply (whose main quality criterion is clean power) or a battery. Also, if the voltage drops well below 12V because the battery is empty, you are guaranteed erratic behavior from the A4988.

5. The A4988 may be broken. Try a replacement.

39.6 Stepper Motor Music

Try this:

https://www.instructables.com/Make-Music-With-Stepper-Motors/
instructables.com
“Make Music With Stepper Motors!”

39.7 Recommended Videos

https://www.youtube.com/watch?v=ROpLjd9iQQU
NSTB
“Rossini’s William Tell Overture Finale - Stepper Motor Music”

[7]The spikes may even occur upstream from the buck converter; so there might be a problem if a buck converter is connected to the same circuit, even if the A4988 isn’t receiving converted power from it.

Chapter 40

Unipolar Steppers: 28BYJ-48 + ULN2003

A very inexpensive way of adding steppers to a project is the 28BYJ-48, a unipolar stepper motor with a built-in gearbox, with the driver ULN2003. This stepper runs at 5V.[1] It internally has 32 steps per rotation (when not using microstepping). The reduction factor of the gearbox is 63.68395, which has the unpleasant consequence that a nonintegral 2037.8864 steps make one rotation of the output shaft.[2] The motor is very precise and quite strong (3.43 Ncm), but slow (15 RPM).

The 28BYJ-48 is very small; due to its gearbox (the 17HS4401 steppers do not come with internal gearboxes) it is relatively strong for its small motor size, but very slow. In practice, its applications don't overlap with those of the 17HS4401.

Be sure not to use unipolar steppers with bipolar stepper drivers or bipolar steppers with unipolar stepper drivers. They are incompatible!

For further explanations and wiring up one or two of the ULN2003 and 28BYJ-48 with a microcontroller, see https://edistechlab.com/en/der-28byj-48-stepper-motor/

[1] There are 5V and 12V versions of this motor, but unless we told you otherwise, your motor is the 5V version.

[2] So don't build use this stepper to position the hands of an analog clock. The error would accumulate, making it a bad clock. But for applications where the stepper drives a belt, this doesn't matter at all!

40.1 Programming: The Low-Level Method

Just like for bipolar steppers, it isn't overly hard to program these steppers the low-level way, without any library, though it is a little harder here than it was for A4988 (see Section 39.3) because the ULN2003 interface is more low-level: For the A4988 we send signals triggering steps, while, for the ULN2003 we have to cycle through four phase patterns (relating very roughly to activations of one of the four motor wires) to steps. Have a look at the following code example, which just keeps stepping in one direction, forever.

```
void setup() {}

int stepper_pins[4] = {8,9,10,11};

int phase_pattern[][4] = {
  {1,0,0,0},
  {0,1,0,0},
  {0,0,1,0},
  {0,0,0,1}
};

void step(bool forward){
  static unsigned current_step = 0;

  if(forward) current_step = (current_step + 1) % 4;
  else current_step = (current_step - 1) % 4;

  for(int i=0; i<4; i++)
    digitalWrite(stepper_pins[i], phase_pattern[current_step][i]);
}

void loop() {
  delayMicroseconds(3000); // You may be able to decrease this.
  step(true);
}
```

Here we have connected pins 8 to 11 of the microcontroller board to in1 to in4 of the ULN2003, respectively.

The `step()` method cycles through the four phase patterns. It's argument allows to reverse stepping direction (going in reverse just requires to cycle through the phase patterns in reverse order.

Between any two phase pattern activations, we have to wait a little for them to have any effect and the motor to move into position. In the example, we wait for 3000 microseconds between steps. You may try to tune this a little – for a motor I tried this with, the minimal delay was roughly 1800 microseconds, but this may be different for you. As you decrease the delay, your motor runs faster but also more smoothly: After executing a step, its calling as a stepper makes it want to stop there and lock itself into position. This frequent acceleration and deceleration causes vibrations and noise. If you time it just right, the stepper receives instructions to do the next step just when it reaches its position from the last command, and it keeps running at constant velocity instead of slowing down and speeding up again. However, if you make the delay too small, the stepper moves erratically or not at all. The exact minimum will depend on sample variations and won't be the same for all copies of the 28BYJ-48. Worse, it will even vary a little with the quality of supplied power, room temperature, and whether it likes you as a person. So tuning this too tightly is asking for trouble.

40.2 Programming: The AccelStepper / MultiStepper Libraries

A higher-level method to program steppers is using the AccelStepper library[3] – which also works for bipolar steppers, see the Web. Together with the MultiStepper class, AccelStepper allows you to address a set of k steppers together in a CNC-style machine, where each stepper handles one spatial dimension. You can now instruct the stepper group to move to points in k-dimensional space, and the library does the rest, including doing all the calculations to accelerate and decelerate the steppers smoothly.[4] Be sure to read the documentation on the library on the Web; it has a variety of functions.

[3] http://www.airspayce.com/mikem/arduino/AccelStepper/

[4] If you did the same the low-level way, you'd have to do your own trigonometric calculations to achieve "diagonal" movement between points. The Arduino IDE supports trigonometric functions `sin()`, `cos()`, and `tan()` as built-ins, without including a library.

Note that there are blocking and non-blocking operations. For non-blocking operations, you can issue the command and your thread of control returns to you immediately. You can execute further instructions on the microcontroller while the steppers move to the set position. You have to query the library to check whether they have reached their position before giving new commands. In the blocking versions, the library operations block your microcontroller program until they are done.

The following example uses two steppers, as for a 2D-plotter. In the example, we are taking coordinates from the Serial Interface (of the Arduino IDE). For instance, entering "200 500" followed by hitting return will make the two steppers move to absolute position (200, 500).

```
#include <AccelStepper.h>
#include <MultiStepper.h>
#define MotorInterfaceType 4

AccelStepper X(MotorInterfaceType, 8, 10, 9, 11);
AccelStepper Y(MotorInterfaceType, 2, 4, 3, 5);
MultiStepper XY;

long pos_xy[2] = {0,0};

void setup() {
  X.setMaxSpeed(500.0);
  Y.setMaxSpeed(500.0);

  XY.addStepper(X);
  XY.addStepper(Y);

  Serial.begin(9600);
}

void loop() {

  if (X.distanceToGo() == 0) {
     while (Serial.available() == 0) {}

     String s;
     pos_xy[0] = Serial.parseInt();
```

```
    pos_xy[1] = Serial.parseInt();
    s = Serial.readString();
    Serial.print("x = ");
    Serial.print(pos_xy[0]);
    Serial.print("  y = ");
    Serial.print(pos_xy[1]);
    Serial.println(s);
    XY.moveTo(pos_xy);
  }

  XY.runSpeedToPosition();
}
```

Note the following quirk of the AccelStepper library. With the constructor of the AccelStepper class, you provide five arguments, the first of which can always be (motor interface type) 4, and the remaining are the four pins, but in a permuted order. For instance,

```
AccelStepper X(MotorInterfaceType, 8, 10, 9, 11);
```

means that you need to connect in1 to in4 of the ULN2003 to pins 8, 9, 10, and 11, respectively, of the microcontroller board, *not* 8, 10, 9, and 11!

Make sure the Serial Monitor of the Arduino IDE is set to 9600 baud!

40.3 Turning the 28BYJ-48 into a Bipolar Stepper

There is a hack for turning the 28BYJ-48 from a unipolar stepper into a bipolar stepper. Apart from the fact that, once this is done, you cannot use the ULN2003 motor driver anymore, it has only advantages, some quite significant:

- The motor's efficiency is improved and its output torque more than doubles (from 0.038 Nm to 0.08 Nm)!
- You can use the A4988 stepper motor driver after the conversion; as a consequence, you can use the same programming for the modified 28BYJ-48 as for other bipolar steppers such as the 17HS4401.

- The A4988 has a higher level interface than the ULN2003. For the ULN2003, you need to cycle through multiple phase patterns to step, and it takes four signal lines to connect the microcontroller to an ULN2003 driver. The A4988 uses just two signal lines, step and direction. Thus the microcontroller is less busy driving the stepper, and you free up pins for other purposes.

Find the instructions for the conversion at
https://ardufocus.com/howto/28byj-48-bipolar-hw-mod/. There are also Youtube videos covering the conversion. Note that you need to open up the motor. The conversion is relatively easy, but you still need to work carefully in order not to destroy the motor.

Important: As you need a different motor driver after this conversion and doing it badly may take out the motor, **do not do this conversion unless you have first received permission from the teaching staff**. Never do this for an individual course project!

Chapter 41

Servos

Servos are devices consisting of a motor, usally with gearbox to increase torque, a rotary sensor, and an "intelligent" motor driver with a feedback component that allows to either position the output shaft of the motor to a precise position (angle) or to do a specified number of (fractions of) rotations. Thus, there is a *closed control loop* – the servo senses the rotation of the motor and allows to "program" a desired position (angle) the motor is to rotate to. In this section, we talk about the practical aspects of using servos; Chapter 42 goes more deeply into control theory and the so-called PID controllers used in servos.

There are various different types of servos, mostly distinguised by the motor technology used, their size and strength, and the interfaces by which they can receive positioning instructions. Figure 41.1 shows the main types of servos.

41.1 Brushed Servos

Brushed servos are offered in different sizes but usually have a similar box shape. They use relatively small motors and achieve considerable torque using large gearbox reductions. For that reason, they are not backdriveable, and trying to force them into position by rotating their output shaft will typically **damage or destroy** their gearboxes. Their torque is a key characteristic, and is usually indicated in $kg \cdot cm$.[1]. The most widespread model of brushless servos, the SG90, is shown in Figure 41.2.

[1] $1kg \cdot cm = \frac{g}{100} Nm \approx 0.1Nm$, where $g \approx 9.81m/s^2$.

Figure 41.1: Brushed servos of different sizes in the popular box form factor (top), a brushless MIT robot dog servo (center), and closed-loop stepper (bottom).

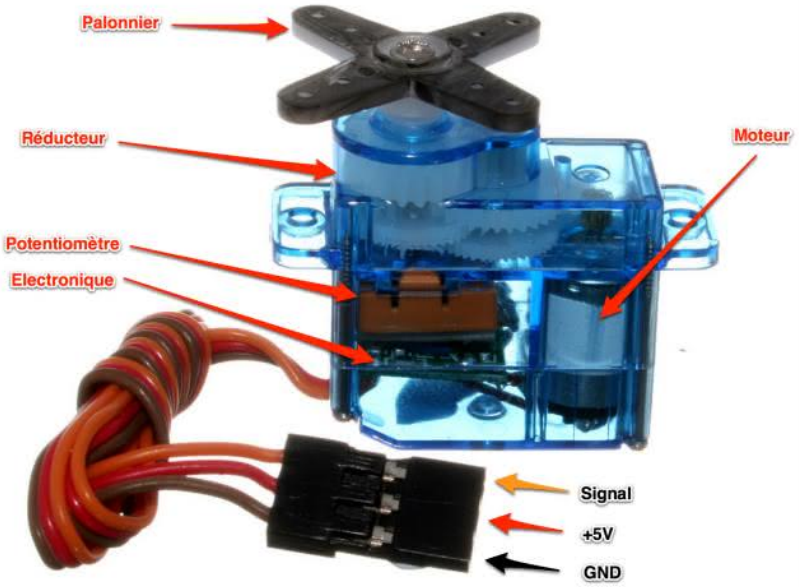


Figure 41.2: A common small servo of type SG90. Motor, gearbox, and control electronics are all integrated into a small package.

Their range of rotation is usually limited to either a little less than 180 degrees, to 180 degrees, or to 270 degrees, though continuous-rotation servos exist as well. Typically, supported voltages are from 6V for small servos to about 8.4V for larger ones.

They have a built-in motor driver that uses feedback from the motor (via a rotary encoder, usually simply a cheap potentiometer) to decide in which direction to move the motor. The logic to calculate this delta, the PID control, is fully integrated and there is no way to modify or tune it. A servo may overshoot the target position and start "searching" (going back and forth beyond the target position) if it is not correctly tuned, the load is too high, or it is damaged.

These servos take a PWM signal as input by which you can send a desired absolute servo position (an angle) via the PWM signal's duty cycle.

These servos have a three-wire female connector that is compatible with jumper cables. The typical coloring of the cables is yellow, red, and black (where red is the middle wire). Here, black is ground, red is + (typically 6V), and yellow carries the PWM relative to ground. The colors are not always these, but there is usually a yellowish PWM wire, and the + wire is

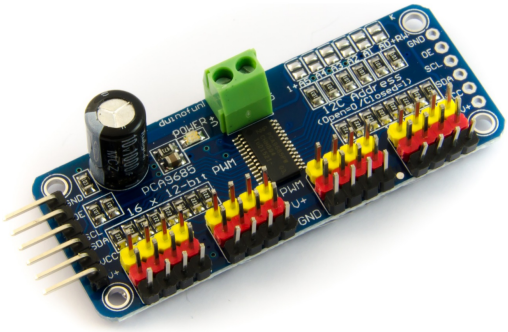


Figure 41.3: The PCA9685 16-channel PWM board.

always in the middle.

41.1.1 Operating many servos

Some applications require many servos, more than you have PWM pins on your microcontroller. Instead of using multiple microcontrollers, is is easier to use one or more PCA9685 PWM multiplexer boards (see Figure 41.3). Each of these features 16 PWM pins, and it is possible to use muliple such boards with a single microcontroller.

41.2 Brushless Servos

Brushless Servos were popularized by the MIT[2] Cheetah robot dogs and various projects by Boston Dynamics. These servos use powerful brushless motors, and can perform explosive acceleration that allows sizeable robots to move nimbly and even jump. They include either planetary or harmonic drive gearboxes with a reduction factor of around 10. They are back-driveable, which allows for organic springiness and animal-like movement. The PID controllers typically offer multiple interfaces, such as a CAN bus, and their PID controllers can be parameterized and tuned. These servos are very expensive (above CHF 500 a piece for no-name clone and thousands of CHF for a quality product).

[2]That place is unrelated to our course. We should sue them.

One way to build such servos yourself is by using odrive motor drivers. These are very powerful, flexible, programmable, and, using a high-quality rotary encoder, they provide precise position control for brushless motors. Unfortunately, this is only a more flexible, but not a cheap solution.

41.3 Closed-loop Steppers

Stepper motors without an explicit feedback component are not considered servos. Stepper motors support precise rotations and positioning based on their design principle and do not require a feedback component to achieve their purpose. However, if a stepper encounters to much resistance to rotation, it may skip steps, and will not be aware of this. For that reason, closed-loop steppers with a feedback components to detect and compensate for skipped steps do exist; however, they are relatively uncommon.

41.4 Recommended Reading and Videos

For more on servos, see
https://www.circuitcrush.com/servo-motor-introduction/.

https://www.youtube.com/watch?v=kUHmYKWwuWs
DroneBot Workshop
“Using Servo Motors with Arduino”

41.5 Parts in Stock for CS358

Model	Specs	CHF
SG90	tba	3
DMS15	15kg·cm 180 deg.	5
Racerstar DS6225MG	25kg·cm 300 deg.	18
SPT5435LV-180	35kg·cm	21

Chapter 42

Control Loops

Alexander Müller

TODO: Test course-correction control code for vehicles.

Control loops are central to robotics. They are present in airplanes, cars, robot arms, and home appliances. Control loops are an abstraction that allow us to quickly and precisely set real world values by using somewhat unreliable actuators and sensors.

As with many chapters of this book, this chapter is not a complete guide to control theory, as it is far too large a field to fit in a few pages. This chapter will serve as an introduction to one of the important and most widely utilized control theory concepts, but to get a better and deeper understanding for your specific application, you will need to do some further reading. Wikipedia is always a good place to start.

If you are really looking to skip through this book and this sounds boring, here are the systems where you will probably need to read this:

- Driving your vehicle / robot at an exact speed.
- Flying to an exact height .
- Positioning your vehicle / robot exactly in a coordinate system.
- Heating something to a precise temperature .

There are two types of control loops: open [1] and closed. Open control refers to systems that simply try to make a difference without checking how

[1] Arguably, open control loops aren't loops

well they're doing. A good example is those light switches where you can adjust the brightness. Even though you're requesting specific brightness levels, the system has no way of knowing if it's actually achieving what you requested, and an old bulb might not get as bright as it should under the same voltage after being used for a long time.

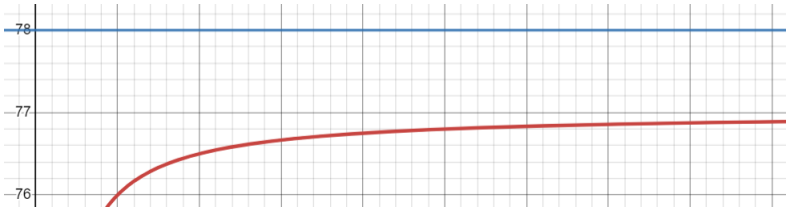
42.1 Closed Control Loop

An example of a closed loop controller would be a heater and air conditioner that have a thermometer to measure how accurately they're setting the temperature. If you are in your home and you set the thermostat to 78, you expect it to get to 78. But how do we actually achieve this? The obvious first step is to say that if the current temperature is higher than 78, we should turn on the air conditioning. And if we're below, we should turn on the heater. That's it! We have made a closed control loop.

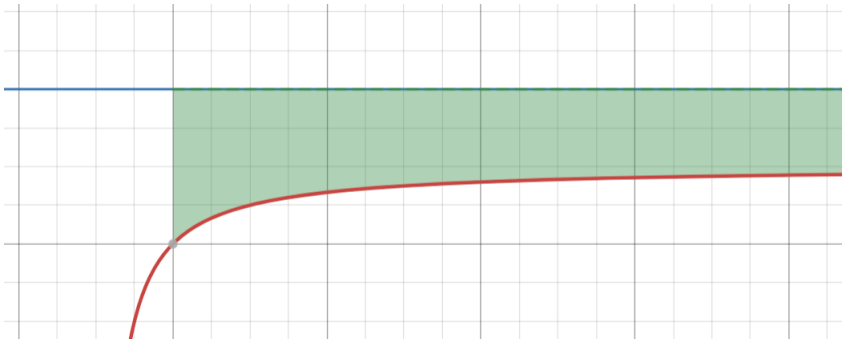
42.1.1 Building a Smart Closed Control Loop

However, it's not very efficient. Most heaters and air conditioners can be adjusted to output different power levels. If we choose a high setting on the heater when we're at 77 degrees, for example, we might overshoot and have to turn on the air conditioner. This can create huge oscillations in temperatures and is very inefficient. So let's try to make a basic algorithm that captures some logical ideas to get a more optimal control loop.

The first obvious step is to say the further we are from the target, the stronger our change should be. This is known in the control world as the "proportional" term, because the difference it makes is proportional to |target - current|. Obviously, we will multiply |target - current| by some constant K_p in order to make sure we have some reasonable conversion from "difference in temperature" to "heater power level". For many systems, like our heating system, this term is enough to achieve good results. However, we want to do better. Imagine now that our heater is using the proportional term and it gives us some kind of result like this:

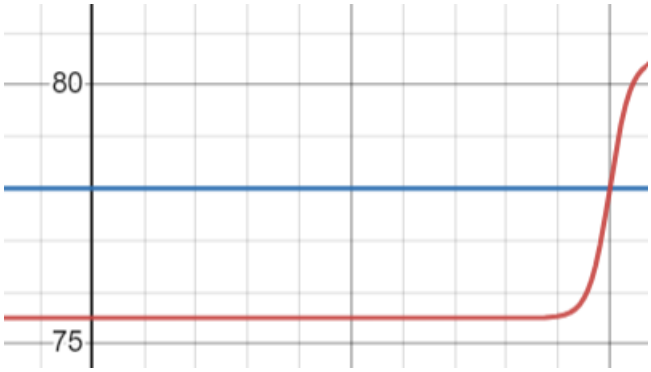


Where blue is the desired temperature and red is the actual temperature. Since our heater will become weaker as we get closer to the target, we are never going to get the actual target (keep in mind that the outdoors is colder than indoors, so even though the heater is always on it will not always strictly get hotter). So, just looking at this graph, it seems logical to say if it's been underneath the desired temperature for a long time, we can turn up the heater more. Additionally, if it's further away, this effect should kick in faster. In a way, this term is proportional to the accumulation of how far we are away from the target.



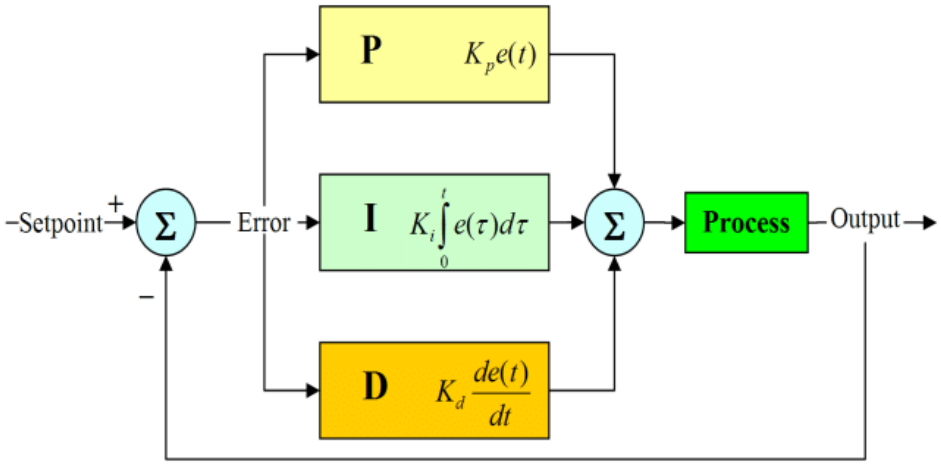
Hopefully it can be seen that this term is proportional to the integral of (target - actual) (with respect to time). We can use some basic estimation of the integral, such as a Riemann sum, to keep track of this. This term is shockingly known as the integral term. We will multiply it by a constant K_i to get a reasonable conversion between "Fahrenheit-seconds" and "heater power level".

Now comes the last problem. If our starting value is quite far from the target value, the integral and proportion terms together can build up together quite quickly to make the above curve way over shoot the target (not remotely to scale):



The integral term will continue to grow as long as you're under the curve! This will cause the rate of growth to continue increasing. So, we introduce one last term, with a constant K_d: the derivative. This term is, as the name suggests, based on the slope of the curve. When we are close to the target value, and have a mostly flat curve, this term does almost nothing, as the derivative is close to zero.

However, when the actual value is quite far from the target, and the integral and proportional terms create a fast change, it can counteract this and smooth the approach to the target value. In control theory, a graph like this is used to represent the final system:



Credit: Wikipedia

We've done it! We've made a smart closed loop control system, known as a PID Loop.

42.2 PID Tuning

Remember those constants to convert our error values into output values? What should they be? How should we find them? Disappointingly, the vast majority of PID systems are tuned using heuristic methods. In other words, they use trial and error to see if the constants are good. The reason this is usually the case is because it can be hard or even impossible to model the effects of the controller. If you have a good model of your system, methods do exist to mathematically tune the constants. Beware, even small discrepancies between the model and reality can throw off the results wildly.

Here, I will just discuss some general tips for heuristic methods. You should always first consider your application and convince yourself the values you are using make sense. Additionally, if possible, graph the outputs of the controller to see what is happening. It cannot be overstated how much simpler tuning a PID controller is when you can see what is happening.

WARNING: Before setting up any PID system, check that the constants are of a reasonable order of magnitude. A motor with a large K_p value can way overshoot its target speed, instantly destroying what you're working on or worse, hurt someone. Large K_i values can cause massive changes after a period of time.

The first constant to tune is always the proportional term. It has the simplest behavior, and as stated before, is sometimes the only term required in some scenarios. Consider the units and ranges of values that the output space and input to the controller work in. For instance, an electric motor outputs rotations per minute, in a range from perhaps 0 to 1000. They take as an input a voltage between 4 to 7 volts, provided by a motor controller (note: some motor controllers have built-in control loops, so this may be unnecessary). From the software side, we usually control the voltage with a PWM signal from 0 to 1024. So, the ranges being similar, it is not unreasonable to assume our P constant might be around 1. From there, the value can be further tuned by observing the two following things: If the constant is too large, there will be large overshoots and frequent oscillations around the target value. If the constant is too small, it will take a really long time to reach the target value. Ideally, we will get oscillations with a long period around the target.

When you are satisfied with the results from the previous section, you

can try to add an integral term. Logically, the integral term should be quite a bit smaller than the proportional term, as it builds up to large values over time. In fact, another way of formulating the constant K_i is $K_i = \frac{K_p}{T_i}$, where T_i is an amount of time that represents how long it takes for the integral term to kick in as hard as the proportional term. In general, K_i can be anywhere between half and $\frac{1}{100}$th of K_p. In general, it is better to start with a significantly smaller value of K_i, and work up, until the oscillations start becoming worse.

Finally, you can start to tune the K_d term. This is by far the hardest and least consistent term to tune, and is hard to say much about in general, because its use is so dependent on the system. Many systems will indeed be fine without this constant, as a little bit of overshoot is not always a problem. If overshoot is a huge problem, no matter the values of K_i and K_p, it may be necessary. A good value to start with is often $0.1(K_i)$ or $0.01(K_p)$. You may google formulas and procedures to tune K_d, and some of the results you find may even work.

42.3 Additional Notes

We can apply transformations to the output value of the PID loop in order to make it more logically fit the constraints of our problem. For example, in a motor controller that accepts a pwm value from 0-1024, we would limit the output to that range. We can additionally add a constant term, known as feed-forward, if we know we need some baseline kick in power in our application.

42.4 PID Code Example

Here is a small example of how a PID controller could look in python. The class is initialized with the constants Kp, Ki, and Kd, as well as some optional limits on the output and integral term. The update method is meant to be called in a loop, with the feedback as a parameter. The update method returns the output for the system. We use the riemann sum to estimate the integral, increasing the term by $(last_feedback + feedback)/2$ at each time step.

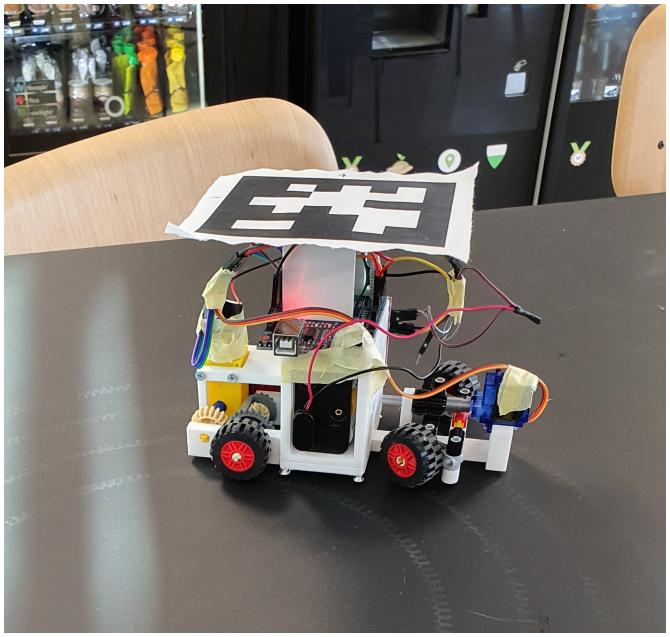
```
   import time
class PID:
    def __init__(self, Kp: float, Ki: float,
        Kd: float, target: float, min: float=0, max: float=float('inf'),
        min_int: float=-float('inf'), max_int: float=float('inf')) -> None:
        """
        Parameters ----------
        Kp : float Proportional gain.
        Ki : float Integration gain.
        Kd : float Derivative gain.
        target : float Target value.
        min : float Minimum output.
        max : float Maximum output.
        max_int: float Maximum integral value
        min_int: float Minimum integral value"""
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.target = target
        self.Dterm = 0
        self.Iterm = 0
        self.last_error = 0
        self.last_time = time.time()
        self.last_feedback = 0
        self.last_output = 0
        self.last_time = 0
        self.max = max
        self.max_int = max_int
        self.min = min
        self.min_int = min_int

    def update(self, feedback: float) -> float:
        """ Calculate the PID output value.
        Parameters ----------
        feedback : float Value to be compared to the target. Returns -------
        float Output of the PID controller. """
        error = self.target - feedback
        current_time = time.time()
```

```
        #we use deltatime for integral and derivative approximations
        delta_time = current_time - self.last_time
        if self.last_time == 0:
            delta_time = 0.001
        if delta_time == 0:
            return self.last_output
        self.Pterm = self.Kp * error #k term is k_t*e(t)
        #riemann approximation of integral
        self.Iterm += (error + self.last_error) * 0.5
            * self.Ki * delta_time
        #approximation of de(t)/dt
        self.Dterm = self.Kd * (self.last_feedback - feedback)
            / delta_time

        #limit integral term
        if self.Iterm > self.max_int:
            self.Iterm = self.max_int
        elif self.Iterm < self.min_int:
            self.Iterm = self.min_int

        self.last_time = current_time
        self.last_error = error
        self.last_feedback = feedback

        output = self.Pterm + self.Iterm + self.Dterm
        if output < self.min: #limit output
            return self.min
        if output > self.max:
            return self.max
        self.last_output = output
        return output
    #change target
    def set_target(self,target):
        self.target=target
```

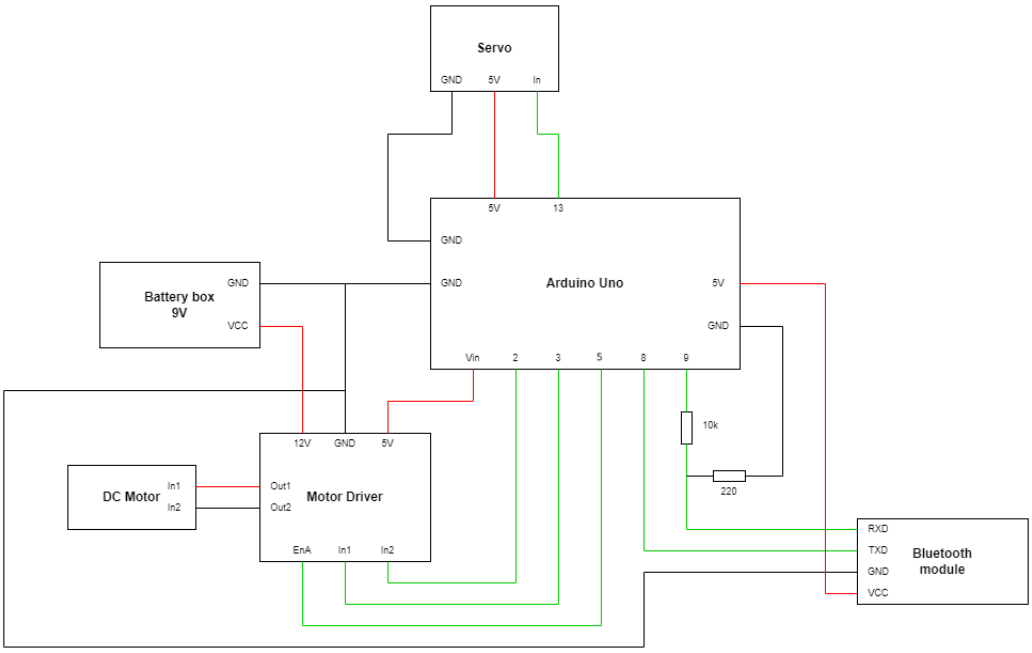
42.4.1 Example Application: Phone-Controlled Car

In this section, we will make a car that we can control using our phone camera. The idea will be that we hold our phone in our hand, camera pointing at the ground. The car will always try to center itself in the frame. By walking around with your phone, keeping the car at the edge of the frame, you will be able to make the car follow you around.

The first thing we need to do is to create a small, remote-controllable car. We can do this easily with either an Arduino UNO and an HC-05 BT tx/rx, or an ESP8266 WiFi microcontroller, a servo, a brushless motor, and an electronic speed controller of your choice. We can mount all these parts onto a 3D printed frame or some legos, if you like. My car look something like so:



And the electronics were wired as follows:



Then, download Camo on your phone and laptop, and follow their pairing instructions. You should be able to access your phone's camera using cv2.videoCapture(1). Finally, follow this book's guide to install AprilTags . Print an Apriltag and stick it to your car.

Now, we have to write some code. We can reuse the code from the previous part as our PID controller.

Then, we quickly write a class to wrap the bluetooth socket to control the car and a camera class to quickly get the location of the car:

```
#camera.py
import cv2
from pupil_apriltags import Detector

class Camera:
    def __init__(self, cam):
        self.goal_x = None
        self.goal_y = None
        self.cap = cv2.VideoCapture(cam, cv2.CAP_DSHOW)
        self.at_detector =  Detector(families='tag36h11',
                       nthreads=1,
                       quad_decimate=1.0,
```

```
                        quad_sigma=0.0,
                        refine_edges=1,
                        decode_sharpening=0.25,
                        debug=0)
        ret, self.frame_in = self.cap.read()
        self.pos = (-1,-1)
    def update(self):
        ret, self.frame_in = self.cap.read()
        grey = cv2.cvtColor(self.frame_in, cv2.COLOR_BGR2GRAY)
        tags = self.at_detector.detect(grey)
        if len(tags) == 0:
            self.pos=(-1,-1) #to stop the car if we dont see the tag
        for tag in tags:
            cv2.putText(self.frame_in, str(tag.tag_id),
                (int(tag.corners[1][0]),int(tag.corners[1][1])),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 3,cv2.LINE_AA)
            cv2.rectangle(self.frame_in,
                (int(tag.corners[0][0]),int(tag.corners[0][1])),
                (int(tag.corners[2][0]),int(tag.corners[2][1])),
                (255,0,0), 2)
            self.pos = ((tag.corners[0][0] + tag.corners[2][0])/2,
                (tag.corners[0][1] + tag.corners[2][1])/2)
        if self.goal_x is not None: #draw the goal
            cv2.circle(self.frame_in, (int(self.goal_x),
                int(self.goal_y)),
                3, (255,0,0), 9)
        cv2.imshow('image', self.frame_in)
    def get_pos(self):
        return self.pos
    def draw_goal(self,x,y):
        self.goal_x = x
        self.goal_y = y

#car.py
import socket
class Car:
    def __init__(self, addr):
        port = 1
```

```
        s = socket.socket(socket.AF_BLUETOOTH,
            socket.SOCK_STREAM, socket.BTPROTO_RFCOMM)
        s.connect((addr, port))
        self.s = s
        self.x = 0
        self.y = 0
    def send(self, command):
        self.s.send(command)
```

And we need some quick code for the car to receive speeds. To send a pwm speed between 255 and -255 (the limits of PWM), we read the Bluetooth into two bytes before converting to an int. Please note that while opening the Serial monitor can be useful for debugging, it can significantly slow down the Arduino's ability to read from the Bluetooth serial port.

```
#define ledPin LED_BUILTIN
#include <SoftwareSerial.h>
SoftwareSerial MyBlue(8, 9);
#include <math.h>
#include <Servo.h>
#define motorpin1 2
#define motorpin2 3
#define enapin 5
#define servopin 13

int middle = 90;


Servo myservo;
void setup() {
  pinMode(motorpin1, OUTPUT);
  pinMode(motorpin2, OUTPUT);
  pinMode(enapin, OUTPUT);
  myservo.attach(servopin);  // attaches the servo on pin 13 to the servo object
  turn(0);
  stopCar();
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  MyBlue.begin(9600); // Default communication rate of the Bluetooth module
```

```
}


void drive(int speed){
  //don't drive at low speed, bad for motor
  if (speed < 80 && speed > -80) {
    speed = 0;
  }
  if(speed >= 0){
  analogWrite(enapin,speed);
  digitalWrite(motorpin1, LOW);
  digitalWrite(motorpin2, HIGH);
  } else {
     analogWrite(enapin,-1*speed);
     digitalWrite(motorpin1, HIGH);
     digitalWrite(motorpin2, LOW);
  }
}
void turn(int angle){
    myservo.write(middle+angle);
}
void stopCar(){
  analogWrite(enapin,0);
  digitalWrite(motorpin1, LOW);
  digitalWrite(motorpin2, LOW );
}

void loop() {
 int16_t speed = 0;
 uint8_t buff[2];
 if(MyBlue.available()){
  int success = MyBlue.readBytes(buff,2);
  if(success > 0){
    speed = buff[1] + (buff[0] << 8);
  }
  drive(speed);
 }
```

```
}
```

And finally a main method for our python code:

```
from cars import Car
from camera import Camera
from PID import PID
import cv2
adapter_addr = '00:21:11:01:FA:1C'
camera = Camera(1)
pid = PID(20,0.0001,0,0,max=255,min=-255,max_int=100,min_int=-100)
car = Car(adapter_addr)
initial_x = -1
initial_y = -1
while initial_x == -1:
    camera.update()
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    initial_x, initial_y = camera.get_pos()
goal_x = 800 #approximately the middle of my camera screen (in pixels)
camera.draw_goal(goal_x, initial_y)
pid.set_target(goal_x)
i = 0
while True:
    camera.update()
    x,y = camera.get_pos()
    camera.draw_goal(goal_x, y)
    output = pid.update(x)
    if (x,y) == (-1,-1):
        car.send(int(0).to_bytes(2, byteorder='big', signed=True))
    else:
        car.send(int(output).to_bytes(2, byteorder='big', signed=True))
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Afterwards, we need to do our PID tuning. I found that a value of 0.8 works quite well for the K_p, and that the other constants are not required

(as our motor doesn't have a wide range of speeds and we do not require to much precision or speed to arrive at our goal). As an experiment, you can set K_p to 30 or so and see what kinds of massive oscillations can happen when the PID loop is tuned incorrectly.

Next, we need to make sure that the car always drives towards the center of the screen. For this, we need a second PID loop. Using the AprilTag, we can draw a line down the center of of the car, and a line from the center of the car to the center of the screen. Notice that these lines always intersect (at the center of the car), and that when these lines are coincident, the car is facing perfectly at the target (or perfectly away from it). Then if there is a positive angle between the two lines, we should turn right, while a negative angle between the two lines implies that we should turn left. The implementation of this is left as an exercise to the reader.

Chapter 43

Field-Oriented Control

Field-oriented control (FOC) refers to a technical approach to controlling brushless motors which maximizes torque and the smooth running of motors. This is particularly significant when the motor is accelerating/decelerating or running at low speed. FOC controllers turn motors into closed-loop actuators (servos) and can be used for quite accurate positioning, allowing them to take the roles of stepper motors and (box) servos in many robotics applications, though with much higher speed, torque, and power.

FOC isn't just a marketing term: It is descriptive of the underlying theory. In FOC, the relative orientations of rotor and stator in the magnetic field, and orientation relative to field lines are leveraged. The underlying mathematics is much beyond the scope of this manual, but see this video for more:

https://www.youtube.com/watch?v=_6-_jvZe7iA
Texas Instruments
"Field-Oriented Control"

43.1 Control Feedback and Customization

Feedback from the motor is integral to FOC. A FOC controller uses some or all of the following: Measurements of

- the current drawn by the motor over the three power lines,
- back-EMF[1] "coming back from the motor", and

[1]Some FOC controllers, such as odrives, can even harvest the back-EMF energy to

- rotor angle information, measured by a rotation sensor (rotary encoder). This is a component separate from the motor driver, which needs to be mounted to the rotor (shaft) of the motor and connected to the motor driver by wires separate from the three wires powering the motor.

Some FOC controllers can operate while having access to only some of this feedback. Using sensor information relating to rotor position (which applies to all three kinds of feeedback mentioned above) is usually the criterion to make a motor controller "closed-loop". In the context of FOC, running a controller without feedback from a rotation sensor is usually called open-loop, even if other feedback information available. (A good FOC controller such as an odrive can derive some limited angle information from back-EMF, at the resolution of the number of motor poles, which can be higher than 20 for large pancake motors).

FOC controllers do some heavy calculations that are also quite customizable. For these reasons, they are not entirely implemented in hardware but have a significant piece of code running on an MCU. This software needs to be configured and takes a considerable number or parameters.

Among the key parameters that you can/should/need to provide is information on the motor and the rotation sensor.

For the motor, the key characteristics are

- The number of pole pairs. Most brushless motors have between 7 and 14 pole pairs. This information should be in the motor's datasheet; alternatively, may count the poles if the motor has cutouts allowing you an inside view of the motor stator. A pole is a visible coil in the stator. Two opposite poles for a pole pair. Outrunner (drone) motors usually have cutouts to allow for better motor cooling in the propeller downwash, allowing you to see the coils, and other non-hobby motors are likely to have better-than-useless datasheets which detail the number of pole pairs. Do not confuse poles with pole pairs!

- The kv rating, which provides information on the motor's speed to torque tradeoff (or wire thickness to number of turns per coil tradeoff).

recharge a battery. Generally speaking, this energy has to be absorbed somewhere, or else, for large motors and strong braking, this loose current can interfere with your electronics and even damage something For that reason, odrives, when not set up to feed the energy back into the battery, should be connected to a special high-power, low-Ohm resistor.

	odrive	SimpleFOC+Stepper	SimpleFOC + BLDC
#pole pairs	yes	yes: 50	yes
kv	yes	no	no
phase resistance	no	no	yes
phase inductance	no	no	no

Table 43.1: Required motor parameters for various FOC scenarios.

- Phase resistance and phase inductance. The two numbers should be in the motor datasheet, and some motor drivers can measure them, so you may not need to provide them.[2]

  Note that you cannot measure phase resistance with a multimeter![3]

There are different kinds of rotation sensors, including optical and magnetic (Hall-effect) sensors. To the FOC controller, what is relevant is how this angle information is provided. The two most typical sensor types are Hall-effect sensors with three signal lines, usually called Hall1, Hall2, and Hall3, and incremental encoders[4] with two signal lines called A and B, and potentially a third, the index (sometimes labeled X or Z)[5]. For an encoder, you also need to provide its resolution (the number of counts per rotation,

[2]Odrives measure both reliably during motor calibration, so you do not need to provide them; on the other hand, if you provide these two numbers as paramaters, you may skip motor calibration, which otherwise runs on each odrive startup or reset. For odrives, you must provide the pole pairs and kv ratings of the motor as parameters. For SimpleFOC with steppers, you need to only provide the number of pole pairs, which is always 50. For SimpleFOC with brushless motors, you need to provide pole pairs **and phase resistance** (but not phase inductance). Note that SimpleFOC will not complain if you do not provide phase resistance, but if you do not provide this, or provide an incorrect number, extreme currents may flow, with all the serious consequences this entails. See Table 43.1 for a summary of what you need to provide.

[3]Actually you can in principle (see https://docs.simplefoc.com/phase_resistance), but for doing that you need to know the internal wiring topology (mainly star or delta) of the motor, information even less available than phase resistance, to apply the right multiplier to your measurement. Another problem is that most multimeters are not precise enough when measuring very small resistances and may give you a reading that is off by an order of magnitude. Using a phase resistance value obtained in this way might cause very large currents to flow.

[4]See https://en.wikipedia.org/wiki/Incremental_encoder.

[5]If the encoder supports an index, it will send a signal whenever the rotor passes a particular zero-angle position, allowing absolute positioning. In the absence of an index, the zero angle is the position at the time of controller startup or reset.

most commonly 2048 or 4096).

Generally speaking, you should provide as many of these parameters as possible, if you have them (accurately). Note that the FOC controller may determine from sensor feedback that it receives that some of the settings provided by you are incorrect, leading to runtime failures. This is particularly true if you indicate the wrong number of pole pairs or an incorrect resolution for your rotary encoder.

There are many further parameters that can be set, such as limits on voltages, current draws, or rotation speeds, and parameters for tuning the internal PID controllers. FOC controllers usually have multiple PID controllers for different kinds of supported operation.

43.2 Features

FOC controllers usually offer a number of modes, characterized by the primary parameter manipulated by the user, including torque control, position control (called angle control in SimpleFOC), and speed control modes. In either mode, the controller will attempt to follow its instructions, such as running at a given speed no matter the opposing forces (in speed control mode). This means, in particular, that the motor will actively brake – try to lock itself in the current position when stopped (speed set to zero) – and will even oppose you when you try to help it turn in the same direction it is rotating if you try to make it rotate faster than it is set to rotate.

These features make FOC controllers very popular in robotics applications. Some of these behaviors and features aren't achievable using any other motor/controller technology.

While FOC position control doesn't yield as high an angular resolution as stepper motors do, for many applications, such as legged robots, it is easily accurate enough, particularly if some mechanical reduction (through belt drives, gearboxes, etc.) is employed. Differently from (open-loop) steppers[6], FOC actuators are closed-loop, making them more robust in applications where your thing interacts with the real world and unex-

[6]Remember: Open-loop means that the controller is blind to its environment. A stepper has a reliable mechanism for advancing by precisely one step, but if it hits an obstacle or its torque is insufficient to complete the step, it misses a step and neither the motor controller nor you will know this happened. A closed-loop servo/stepper has a position sensor and can keep trying until it achieves the desired position.

pected obstacles. Brushless motors usually have much higher acceleration and torque maxima than stepper motors, making a FOC actuator much faster and more responsive than a stepper. Note that most industrial robots (robot arms), some of which weigh multiple tons, have several meters of reach, and which achieve reproducible position accuracy of less than 0.02mm, use sensored BLDC motors with FOC controllers (and gear reduction), rather than steppers.

By the PID control and the working principle of brushless motors, FOC actuators are naturally springy and back-driveable, and can create more lively and organic motion than other motor technologies. Back-driveablity is particularly essential in legged and bio-inspired robotics.

FOC controllers have little resemblance with cheap brushless ESCs used in remote-controlled cars and drones. The gold standard for FOC controllers for us are *odrives*[7], but they are expensive.[8] An alternative is SimpleFOC[9].

43.3 SimpleFOC

SimpleFOC is an open-source software library that works with many brushless motor controllers.

TODO discuss code structure, motor and driver class options, sensors, and commander interface.

43.3.1 STM32 B-G431B-ESC1 and a Brushless Motor

Here is a position control example with the STM32 B-G431B-ESC1 and an X2212 motor. This motor has 7 pole pairs and a phase resistance of 0.092Ω according to datasheet (0.086Ω as measured by an odrive). This motor comes with a TLE5012b (magnetic) encoder with a resolution of 4096 counts per rotation. Note, in the following code, the various pin aliases pre-set specifically for the B-G431B-ESC1, such as A_PHASE_**, A_HALL*, and A_OP*_OUT.

[7] https://odriverobotics.com/

[8] A genuine odrive costs about CHF 250 without motors, and a clone from China costs CHF60+.

[9] https://simplefoc.com/

TODO add links to STM32 B-G431B-ESC1 manual, example code, forum post, and SimpleFOC tuning video.

```
#include <SimpleFOC.h>

BLDCMotor motor = BLDCMotor(7, 0.092); // SunnySky x2212
BLDCDriver6PWM driver = BLDCDriver6PWM(A_PHASE_UH, A_PHASE_UL,
               A_PHASE_VH, A_PHASE_VL,
               A_PHASE_WH, A_PHASE_WL);
LowsideCurrentSense currentSense = LowsideCurrentSense(0.003f,
  -64.0f/7.0f, A_OP1_OUT, A_OP2_OUT, A_OP3_OUT);

Encoder encoder = Encoder(A_HALL2, A_HALL3, 4096); // TLE5012b
void doA(){encoder.handleA();}
void doB(){encoder.handleB();}

Commander command = Commander(Serial);
void doTarget(char* cmd) { command.motion(&motor, cmd); }

void setup() {
  encoder.init();
  encoder.enableInterrupts(doA, doB);
  motor.linkSensor(&encoder);

  driver.voltage_power_supply = 12;
  driver.init();
  motor.linkDriver(&driver);

  currentSense.linkDriver(&driver);
  currentSense.init();
  currentSense.skip_align = true;
  motor.linkCurrentSense(&currentSense);

  motor.controller = MotionControlType::angle;

  motor.voltage_limit = 12;
  motor.current_limit = 10;
  motor.velocity_limit = 1000;
```

```
  motor.PID_velocity.P = 0.2;
  motor.PID_velocity.I = 20;
  motor.PID_velocity.output_ramp = 1000;
  motor.LPF_velocity.Tf = 0.01;
  motor.P_angle.P = 20;

  motor.target = 0;
  motor.init();
  motor.initFOC();

  Serial.begin(115200);

  command.add('T', doTarget, "target angle");

  Serial.println(F("Ready."));
  _delay(1000);
}

void loop() {
  motor.loopFOC();
  motor.move();
  command.run();
}
```

43.3.2 L298N and a Bipolar Stepper

In the recommended videos section of this chapter, you find a number of useful videos on SimpleFOC. One particular feature is that SimpleFOC can also turn a bipolar stepper motor with two H-bridges into a FOC servo. This does not work with a stepper driver such as the A4988, but it works with a dual H-bridge brushed motor driver such as the L298N (see Chapter 38).

Observe how the stepper completely abandons its stepper characteristics: It runs much more smoothly and silently, and potentially much faster. It also develops higher torque than in the standard setting it was designed for (thanks to the control theory of FOC, which essentially maximizes torque)! SimpleFOC yields position control at the cost of not being able to do stepping, so this wouldn't be a good solution in a 3d-printer or CNC machine.

```
#include <SimpleFOC.h>

StepperMotor motor = StepperMotor(50);
StepperDriver4PWM driver = StepperDriver4PWM(5, 6, 9, 10,  8, 7);

Encoder encoder = Encoder(2, 3, 2048);
void doA(){encoder.handleA();}
void doB(){encoder.handleB();}

Commander command = Commander(Serial);
void onMotor(char* cmd){ command.motor(&motor, cmd); }


void setup() {
  encoder.init();
  encoder.enableInterrupts(doA, doB);
  motor.linkSensor(&encoder);
  motor.foc_modulation = FOCModulationType::SpaceVectorPWM;

  driver.voltage_power_supply = 12;
  driver.init();
  motor.linkDriver(&driver);

  motor.controller = MotionControlType::angle;

  motor.PID_velocity.P = 0.2;
  motor.PID_velocity.I = 20;
  motor.PID_velocity.D = 0;
  motor.LPF_velocity.Tf = 0.01;

  motor.voltage_limit = 12;
  motor.P_angle.P = 20;
  motor.velocity_limit = 50;

  Serial.begin(115200);
  motor.init();
  motor.initFOC();
  motor.target = 0;
```

```
  command.add('M', onMotor, "motor");
  _delay(1000);
}


void loop() {
  motor.loopFOC();
  motor.move();
  command.run();
}
```

43.4 An ODrive Example

Below you find a complete odrive configuration example for our 6374 motors, in python. You will be excited to read that the myriad of esoteric parameters such as flux linkage[10] are not boilerplate from some web page but were set by me after lookup in a data sheet or experimentation (there is a calibration stage where you play with the motor and controller to find out the right settings). The settings very much differ for different motor models. Bad setting (particularly current limits) may destroy all electronics in the vicinity and cause fires. Correct and double-checked wiring and understanding ground loops (see Chapter 19) is essential. Odrives may unleash thousands of watts of power; even a "correctly" set up and configured thing (in terms of FOC controller configuration) may hulk-smash everything around it, including you.

Since you will most likely not be working with odrives, this is to give you a rough idea of the range of configuration parameters of FOC controllers. No explanation of parameters is given here – you will either already know what these parameters mean after reading the previous chapters, or the parameter meanings are fundamentally un-knowable ;-) (See the odrive documentation[11] if you must know. If you ever get to work with odrives, you must know. Script kiddies[12] kill odrives.)

[10] See wikipedia if you must. But isn't this the perfect term for technobabble? Usage: "This is a Unix system! I know this! The flux linkage is over 9000 when the ssh button is booted down!" (Credits to Jurassic Park 1, Dragon Ball Z, Aeon Flux, Die Hard 4, etc.)

[11] https://docs.odriverobotics.com/v/latest/guides/getting-started.html

[12] https://en.wikipedia.org/wiki/Script_kiddie

```
odrv0.erase_configuration()

odrv0.config.enable_brake_resistor = True
odrv0.config.brake_resistance = 2
odrv0.config.dc_max_positive_current = 18
odrv0.axis0.controller.config.vel_limit = 20
odrv0.axis0.motor.config.pole_pairs = 7
odrv0.axis0.motor.config.torque_constant = 8.27 / 190
odrv0.axis0.motor.config.current_lim = 60
odrv0.axis0.motor.config.calibration_current = 30
odrv0.axis0.config.calibration_lockin.current = 30
odrv0.save_configuration()


# sensorless (comment out either this part of the sensored mode part below)
odrv0.axis0.controller.config.vel_gain = 0.01
odrv0.axis0.controller.config.vel_integrator_gain = 0.05
odrv0.axis0.controller.config.control_mode = CONTROL_MODE_VELOCITY_CONTROL

odrv0.axis0.motor.config.current_lim =
  2 * odrv0.axis0.config.sensorless_ramp.current
odrv0.axis0.sensorless_estimator.config.pm_flux_linkage =
  5.51328895422 / (7 * 190)
odrv0.axis0.config.enable_sensorless_mode = True
odrv0.axis0.motor.config.pre_calibrated = True
odrv0.save_configuration()

odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL


# sensored mode
odrv0.axis0.config.enable_sensorless_mode = False

odrv0.axis0.encoder.config.cpr = 8192
odrv0.axis0.requested_state = AXIS_STATE_FULL_CALIBRATION_SEQUENCE
#wait until finished before issuing more commands

odrv0.axis0.encoder.config.use_index = True
```

```
odrv0.axis0.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
#wait until finished before issuing more commands

odrv0.axis0.requested_state = AXIS_STATE_ENCODER_OFFSET_CALIBRATION
#wait until finished before issuing more commands

odrv0.axis0.encoder.config.pre_calibrated = True
odrv0.axis0.motor.config.pre_calibrated = True
odrv0.axis0.config.startup_encoder_index_search = True
odrv0.save_configuration()


# tuning, was done with 50V
odrv0.axis0.controller.config.vel_integrator_gain = 0
odrv0.axis0.controller.config.vel_gain = 0.33
odrv0.axis0.controller.config.pos_gain = 110
odrv0.axis0.controller.config.vel_integrator_gain =
  0.5 * 10 * odrv0.axis0.controller.config.vel_gain
odrv0.axis0.requested_state = AXIS_STATE_IDLE
odrv0.save_configuration()


# defaults
odrv0.axis0.controller.config.vel_integrator_gain = 0
odrv0.axis0.controller.config.vel_gain = 0.16
odrv0.axis0.controller.config.pos_gain = 20
odrv0.axis0.controller.config.vel_integrator_gain = 0.32
odrv0.save_configuration()

# testing position control and holding torque
odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
odrv0.axis0.controller.input_pos = 1.5


# filtered position control
odrv0.axis0.controller.config.input_filter_bandwidth = 5
odrv0.axis0.controller.config.input_mode = INPUT_MODE_POS_FILTER
odrv0.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
```

```
def goto(x):
  odrv0.axis0.controller.input_pos = x
  while(abs(odrv0.axis0.controller.pos_setpoint - x) > 0.02):
    time.sleep(0.1)
```

Should you ever need to do this, talk to me. There are some quirks, such as that the calibration was done interactively, not by running this as a script. The parts with a comment marked "wait until" were copied into the console, with me waiting until the previous operations were completed. (This is done once during calibration, not every time the thing starts up. `odrv0.save_configuration()` saves settings in nonvolatile memory on the motor controller.) Note that no line here is superfluous, including the apparently repetitive configuration saving.

43.5 Parts in Stock for CS358

item	#motors	max V	Ctrl	MCU	CHF
Odrive v3.6 clone	2	56	odrive	yes	65
STM32 B-G431B-ESC1	1	24	SimpleFOC	yes	20
SimpleFOC Shield	1x gimbal	24	SimpleFOC	no	15
L298N (stepper FOC)	1	35	SimpleFOC	no	3

For your team project, do not suggest a FOC controller other than those in this list. If you want to use FOC, pick among these.

When your project involves brushless motors, you must add a USB isolator (costs CHF 15) to your Bill of Materials.

43.5.1 Current Needs and Ratings

The single most important stat of a FOC driver is its current rating; however, things cannot be distilled down in a single number. It is essential that you read the technical description and cooling requirements of your motor driver before putting it into operation; otherwise you will destroy it and may even cause a fire. Some short remarks here, though:

- Odrives and the STM32 B-G431B-ESC1 use MOSFETs and can handle substantially higher currents than the other two drivers in our list.

- Odrives come with heat sinks on their driver MOSFETS and are rated for 40A (continuous) per motor (each board can drive two motors) without additional cooling. With the right cooling setup, they can handle up to 120A. For details, see https://docs.odriverobotics.com/v/0.5.4/specifications.html.

- The specced 40A max for the B-G431B-ESC1 is not realistic and would require intense active cooling. This motor driver assumes it is mounted under the propeller of a large drone. Even for normal operation, a heatsink should be mounted on the MOSFET, which is not shipped with the product and needs to be ordered separately. To run a small brushless motor with low torque needs except for brief peaks, the motor drivers can operate without a heatsink.

- The SimpleFOC shield is rated for 2A continuous. It is designed for gimbal motors with phase resistance of about 10Ω. This is extremely high: Normal brushless motors have phase resistances of tens of $m\Omega$, and if you run them with the SimpleFOC shield, very bad things may happen.

  If you are trying out the SimpleFOC shield with a small brushless motor that is not a gimbal motor, make sure to set the maximum motor voltage to 1V (via SimpleFOC parameter `motor.voltage_limit`) and measure the current draw with a lab bench power supply. You may increase this voltage limit while checking that peak current draw for your thing under load does not exceeed 2A.

- The L298N[13] uses BJT technology H-bridges, with an internal voltage falloff of 1.4V. Thus, if you run a classical brushed motor at 2A, about 3W of heat are produced in the motor driver.

  The L298N is rated for 2A, but that is practically a peak rating. If you run anything close to 2A of current for just a few seconds, the L298N will get extremely hot, despite its mounted heat sink. Running it at more than about 0.5A continuously will destroy it quite quickly.

  You can use the L298N with SimpleFOC **only with bipolar steppers** such as the 17HS4401. Unipolar steppers such as the 28BYJ-48 or brushless motors do not work!

[13]The L298N is usually used to drive two brushed motors – see Chapter 38.

Note, though, that a brushless motor (or a bipolar stepper in FOC mode) will run at very low currents (on the order of 10mA) when no load is attached and acceleration is not too extreme (which is easy to ensure by parameter settings in FOC). The actual current draws for your application depend on how heavy your loads are and whether your thing has to fight gravity (as is typically the case for robot arms). Experimentation may be in order to find out your torque and current requirements, but you should first try to estimate the required torque based on your mechanical design and calculate the required currents based on the math introduced in Chapter 37.

43.5.2 Ease of Use

All four models of FOC motor drivers can be used with SimpleFOC; however, using SimpleFOC with odrives requires overwriting the odrive firmware, which is actually superior to SimpleFOC; moreover, it may not be possible to restore the original odrive firmware on odrive clones, so overwriting it and **using odrives with SimpleFOC is forbidden**.[14]

Leaving aside odrives, the remaining three types of drivers can all be used with SimpleFOC and programmed using the Arduino IDE.[15] SimpleFOC is a little easier to use than odrives, though odrives yield higher performance.

Remember that FOC requires substantial computation on an MCU. Odrives and the B-G431B-ESC1 have on-board MCUs which you will have to separately program/configure and which will usually only run FOC, receiving instructions from your main microcontroller(s) running your project's core behavioral code. Thus there may be two kinds of microcontrollers in your project ("core" and FOC ones), that need to be separately programmed, run different code, and need to talk to each other. The SimpleFOC Shield and the L298N do not have MCUs on board, so the FOC computations need to be done on the MCU intended to run your project's core code.

The STM32 B-G431B-ESC1 does not come with connectors to the power supply, the motor, or the rotation sensor/encoder. These need to be soldered on by you, and this requires substantial soldering skill: This is a

[14] Take this seriously. Violating this rule will be taken as vandalism.

[15] Though to unleash the full potential of the STM32 B-G431B-ESC1, you'd have to program it using the STMCubeIDE.

very small and tightly packed board, with small solder pads that are very close to other functional components, so there is the danger of overheating these or creating a shortcircuit connection via a solder bridge. The high-current solder connections to power supply and motor need quite a bit of solder and need to be down well for the motor driver to work, and the solder pads for the sensor/encoder are very small, and are best soldered under a microscope. You should certainly have a soldering practice run on some scrap items before attempting this on the STM32 B-G431B-ESC1 board.

The SimpleFOC shield is designed to be mounted atop an Arduino Uno. The advantage of this is that you do not need to worry about the wire connections between the motor driver and the microcontroller. On the downside, this makes it more difficult to connect other things than a FOC motor driver for one motor to your microcontroller (solutions exist though).

The Arduino Uno is actually too slow to keep up the refresh rates necessary for FOC to work well (running the `motor.loopFOC()` function call in the main loop of the microcontroller frequently enough per second). When running SimpleFOC on the Arduino Uno, FOC may not work at all, torque will be much lower than expected, or, at the least, position control will be very jittery: When trying to move the motor to a position, the movement will constantly overshoot the desired position because control is to slow to stop the motor at the desired position, leading to oscillation or vibration. In this case we may provide you, exceptionall, with a microcontroller board that is not on the list of approved boards – probably an STM32 Nucleo64 board. These can be programmed on the Arduino IDE and used like a faster Arduino Uno – even standard Uno shields can be mounted on top.

43.5.3 Choosing a FOC Solution

Let us distill conclusions from the points made earlier in this chapter. Here is how you should pick your FOC solution.

When your project needs FOC, the motors and motor drivers tend to be the dominating cost factor in your Bill of Materials. In practice, odrives are too expensive, as, even though a clone board may be within budget, the large motors that require us to use it cost on the order of CHF 100 a piece; smaller motors can be run by the cheaper drivers. However, odrives can be used to measure phase resistance and inductance of your motor, and we may lend you an odrive for that.

The SimpleFOC shield is practically limited to gimbal motors, and these provide relatively little power for their price.

Thus, the preferrable options for you are probably the remaining two – the L298N plus stepper combination, and the STM32 B-G321B-ESC1 for brushless motors.

Among these two, the L298N plus 17HS4401 stepper option is cheaper and simpler to use (and does not involve a difficult soldering job). It is cheaper because the motor drivers are very inexpensive and the stepper motors are standardized and mass-produced (practically every modern 3D-printer uses several of them). Thus you get a lot of motor for the price.

With this combination, you get about 0.4Nm of torque ang about **TO BE CHECKED** rpm of maximum speed. You can raise either torque or rotation speed at the cost of the other using a suitable gearbox or belt drive, see Chapter 45.

If this is not sufficient for your application, pick the STM32 B-G321B-ESC1 and a suitable brushless motor.

43.6 Caveats and Debugging

Protecting your laptop from voltage spikes, short circuits, and ground loops. When working with brushless motors, you need to have a USB isolator between your thing and your laptop at all times. Running your laptop on battery (so there is no ground connection through a switching-mode power supply) is recommended in addition.

Make sure your motor parameters are correct. Getting any of them wrong may lead to very large currents flowing. An incorrect phase resistance is particularly dangerous. Do not guess and try pole pair or phase resistance settings. Do not power up the system if you don't know these values.

Get a clear idea of the current drawn by your system. While familiarizing yourself with FOC and your motor and driver combinations, and when you start running your motor under load, make sure to keep track of the current drawn and shut down the power supply immediately if the current flowing seems high. To do this, run your thing off a lab bench power supply which displays the flowing current and allows to XXX

You get a message indicating that the FOC controller expected to see

a different number of pole pairs. Both SimpleFOC and odrives have these error messages. It could be that the number of pole pairs that you indicated is wrong (don't confuse poles with pole pairs) or that your sensor/encoder parameters are wrong.

For instance, if you indicated that your motor has 7 pole pairs (and that's the correct number) and that your encoder has 2048 counts per rotation, but SimpleFOC tells you that it detected roughly 3.5 pole pairs (which is of course impossible), then your encoder really has 4096 counts per rotation, not 2048.

A FOC controller rotates its motor a little on startup. The reason for this is that they calibrate themselves and check whether the parameters you provided are consistent. Also, for encoders with index, the controller will search for the index to home its position. The motor driver may also assume or require that no load is attached to the motor shaft.

This is of course undesirable, since we do not want to have to partially disassemble and reassembe our things every time we start them up.

This may be avoidable if you provide all the relevant motor parameters and use an absolute or non-indexed encoder. For odrives, if you provide phase resistance and inductance, no calibration is necessary on startup, and you can help the controller find the encoder index by rotating the rotor a little by hand, avoiding that the controller makes the motor rotate. If you provide all the motor parameters, SimpleFOC may still rotate the motor a little on startup[16], but at least it should be able to do this under load, so no disaassembly of your thing is neeced.

Sensor readings are unreliable: There are two reasons why this may be happening.

1. The sensor isn't mounted well on the motor. Rotary encoders such as the CUI AMT 102V/103V use a plastic adapter to handle various rotor shaft diameters, and and if this adapter isn't mounted firmly, or if the entire encoder has some wiggle room, the reading will be erratic. Magnetic sensors require you to mount a little magnet on the rotor (shaft) of the motor, which may come loose. Check that.

2. Noise on the sensor wires induced by the motor is a concern. This shows as sensor reading that jump wildly. High-end encoders in in-

[16]So make sure that the position of the rotor on startup is such that some rotation in both direction is possible given mechanical and structural contraints of your thing.

dustrial robots use pairs of wires for each signal, carrying the signal and its complement, twisted together to ensure noise is induced equally in both wires. By subtracting the two signals using an OpAmp in the motor driver, the noise is eliminated. Unfortunately, this is not an option to us since no affordable motor driver supports this. One way to address this is using a ferrite ring. (Do three windings of your sensor wires through and around the ferrite ring, as close as possible to the controller-side end of the wires.) If you thing your sensor readings are erratic due to noise, talk to us. However, it is much more likely that your problem lies elsewhere. Please debug your system and exclude all other possible causes of problems before coming up with the noise theory.

Note that your can do sensor readings without running you motor or even have the motor connected to the FOC controller. You can turn the motor shaft by hand and read out the sensor.

43.7 FOC in Industrial Robots

Industrial robot arms achieve extremely high precision (in terms of accurate reproducibility of movements and positions), often within an error margin of 0.02mm, on robots that can lift hundreds or even thousands of kilograms of weight and can move extremely fast. This isn't achievable with steppers, but is done with large brushless motors[17] under FOC, using suitable sensors and mechanical reducers (usually a combination of a timing belt drives with extremely low-backlash gearboxes – such as a harmonic drives – for the final stage of the mechanical reduction). Leaving aside the mechanical reducers, these controller-motor-sensor solutions differ from those available to us mainly in three ways.

- The brushless motors are not just large but run at very high voltages (usually 310–480V).
- For FOC to work well at high speeds, the motor drivers have to do the control loop computations extremely fast, requiring hardware support beyond a fast microcontroller (but a FOC implementation in FPGA or

[17] These look like large steppers since they are typically in NEMA housings, but they are brushless motors.

ASIC). This also requires very fast sensors that send position updates extremely frequently (and reliably). The position update frequency of rotary encoders in such industrial robots typically is by at least an order of magnitude higher than with the rotation sensors available to us.

- The cabling and electronics have various features to improve robustness and survivability.[18] This includes, for instance, using differential signalling from the sensors to the controllers to eliminate noise. In Section 43.6, we observed that noise on the sensor signal wires is a problem. In the industrial setting, we do not use individual signal wires, but use them in twisted pairs of a signal and its complement. The noise will be about the same on both wires, so by subtracting the signals on the two wires (using an OpAmp), we get the signal without the noise!

The electronics in these FOC controllers and sensors is vendor and model-specific. As an interesting consequence, you can get an old industrial robot extremely cheaply if you want one – they are sold for scrap metal or a given away for free, since once the vendor stops making replacement parts, they cannot be repaired.

43.8 Recommended Videos

https://www.youtube.com/watch?v=Y5kLeqTc6Zk simplefoc “Arduino Field Oriented Control (FOC) Open Source Library Demonstration - Simple FOC project”
https://www.youtube.com/watch?v=G5pbo0C6ujE simplefoc “Arduino Simple Field Oriented Control BLDC driver Shield - SimpleFOCShield”

[18] Odrives can be placed halfway between SimpleFOC and industrial FOC controllers and implement some professional features. In fact, even though odrives were expensive already, the doubled prices with the latest generation of odrives, even though they do not significantly differ from the previous generation in terms of stats such as supported voltage and current – all the improvements that justify the doubled price revolve around “professional” robustness features.

https://www.youtube.com/watch?v=zcb86TRxTxc
simplefoc
“Closed loop Stepper Motor using FOC algorithm - SimpleFOClibrary”

Part VII

Mechanical Engineering

Chapter 44

Mounting Motors

The natural shape of a motor is, of course, a cylinder. Motors usually have special mounting points (screw holes) on a face plate – the base of the cylinder, perpendicular to the axis of rotation. In an outrunner, of the external surfaces of the motor housing, usually only the faceplate is hard-connected to the stator; the remaining surface rotates and cannot be used for mounting the motor. Even for inrunners, the correct way to mount the motor is by its faceplate, and not by pinching its sides – this would usually impede cooling and the sides are often not strong enough to withstand being pinched (clamped down) hard.

For industrial motors, there is an industry standard for motor faceplates by NEMA (the standard is colloquially referred to be NEMA, but this is really the name of the standards organization).

Among hobbyists and makers, NEMA is mainly known in conjunction with bipolar stepper motors; however, there are also other kinds of industrial motors using NEMA faceplates.

Most motors have a dedicated mounting surface or faceplate with screw holes. This is usually the surface of the motor enclosure where the output shaft exits. Motors are to be mounted by these screw connections. Make sure that your screws are not too long, otherwise you may damage the internals of the motor!

Do not mount motors by making a box shaped like the motor enclosure for a tight fit:

- the enclosures of geared motors are often not strong enough not to be damaged by the motor's own torque.

Frame Size	Diameter (mm)	Typical Torque Range for a Stepper Motor (Nm)	Typical Speed Range for a Stepper Motor (RPM)
NEMA 8	20	0.01 - 0.04	0-1000
NEMA 11	28	0.06 - 0.12	0-1000
NEMA 14	35	0.05 - 0.5	0-1000
NEMA 16	39	0.1 - 0.25	0-1000
NEMA 17	43	0.2 - 1	0-1000
NEMA 23	57	0.5 - 3	0-1000
NEMA 24	60	1.2 - 4.6	0-1000
NEMA 34	86	3 - 12	0-1000
NEMA 42	102	12 - 20	0-1000

Figure 44.1: Typical form factors and torque ranges for bipolar steppers.

- the motor may overheat and get damaged, or it may melt 3d-printed enclosures.
- many brushless motors have external rotors; a big part of the enclosure rotates when the motor is running, so the only way to mount the motor is by the faceplate.

When you design your thing and your motor mount, do not forget to take into account the motor's cabling! Also, make sure you take correctly into account the place where cables exit your motor *relative to the positions of the screw holes of the motor faceplate*!

It is a very common mistake to forget about this, and as a consequence, you waste time and material to make the structural parts of your thing.

Chapter 45

Machine Elements and Patterns

45.1 Coupling

Frequently, you have to connect – *couple* – two parts of the mechanical design of your thing to communicate rotation from one part of your thing (usually including a motor) to another. There are a number of ways to do this.

A *shaft* is a rotating machine element, usually circular in diameter, of relatively low diameter, and made of a relatively strong material.

Shaft couplers are for coupling two low-diameter shafts or axles. Flange couplers are for coupling a shaft with a large diameter thing (such as a turntable).

Note the box-shaped slot in the the part in the bottom right of Figure 45.1: This is for sliding a nut in to hold the screw (which goes into the small round hole) in place and firmly against the shaft in the large round hole. Thus, the part does not rely on the screw thread to be held firmly by the plastic (which is too weak for this). This is called a *captive nut design*.

We expect you to make such parts by yourself; we usually will not buy them. Note that such parts can be manufactured out of aluminium in the DLLEL machine shop, but you should do fine with 3D-printed plastic parts if they are designed well. If your torques are too high for 3d-printed couplers, your project may be too dangerous for this course anyway.

Obviously, a suitably designed shaft coupler can link to shafts of different diameter.

TODO: Discuss couplers that support axial lengthening and shortening of a shaft, and couplers that can couple shafts at an angle.

Figure 45.1: Shaft couplers (top left), flange couplers (top right), and two designs for 3d-printable shaft couplers (bottom).

45.2 Creating Mechanical Advantage (Increasing Torque)

We translate rotary force using gearbox reductions and other means; in general, disregarding mechanical inefficiency (due to friction), rotation speed is inversely proportional to torque; we may double the torque of a motor, halfing its RPM, using a 1:2 reduction.

An important property of such mechanical reductions is whether they are back-driveable, that is, whether is is possible to turn the output shaft of the motor-reducer assembly (when the motor is off) without damaging the reducers. Generally speaking, many but not all reducers of a low reduction ratio allow this. For certain robotics projects, particularly legged robots, back-driveability is important to allow biologically inspired locomotion, easy adaptability to terrain, or an ability to jump.

45.2.1 Using Gearboxes

We need to distinguish between gearboxes that need to be able to output high torque and gearboxes that do not need to. Also we need to distinguish between designs that we can manufacture ourselves (using 3D-printed parts) and those that we can't. Finally, we need to consider motors whose output torque is so low that they can only drive gearboxes whose stages closest to the motor have very low friction and inertia.

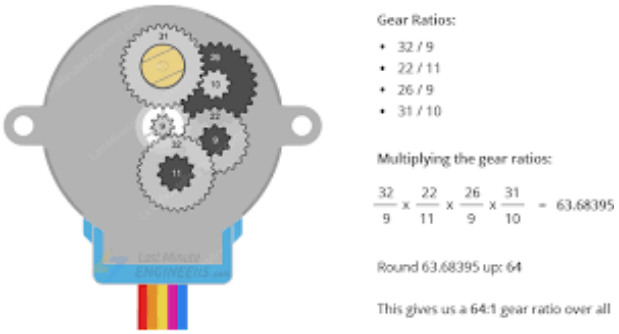


Figure 45.2: Calculations for the internal gearbox of the 28BYJ-48 stepper motor. The overlap of gears as displayed is confusing: Think of the dark gears as being below the light gray gears.

When only low torques are involved, many gearbox designs are feasible; the idea is of course that, when a small gear drives a larger one, the large one will rotate more slowly, and with more torque, proprotional to the ratio of teeth of the two gears.[1]

Many brushed motors have torque so low that even slight friction in the early stages of the gearbox (close to the motor) can lock up the entire gearbox; it is key that particularly these gears are manufactured extremely precisely to minimize internal friction (often disqualifying 3D printers for manufacturing). Also, bought gearboxs will contain special grease to minimize friction. Do not open gearboxes, and do not remove the grease.

The only feasible designs for 3D-printable high-torque gearboxes are planetary gearboxes, cycloidal drives, and harmonic drives (aka strain wave drives). Making such a gearbox that works well from mostly 3D-printed parts is quite a challenging project by itself. All three of these ingenious designs are very satisfying to watch in operation. Have a quick

[1]To learn more, see https://en.wikipedia.org/wiki/Gear_train

look into the videos recommended below, even if you do not plan to make your own gearboxes!

Gearboxes, unless they use worm gears, are usually back-driveable unless the reduction factor exceeds about 10x. Do not try to force back-driving using large-reduction gearboxes such as in servos.

45.2.2 Using Belt Reduction

Unless the reduction factor is extreme, belt reductions are back-driveable.

45.2.3 Recommended Videos

https://www.youtube.com/watch?v=bCvuvCmzuz4
Jeremy Fielding
"The Ultimate Makers Guide to Gear Boxes "

https://www.youtube.com/watch?v=IVpYtyS5Q-k
James Bruton
"Cycloidal Drive V3 & Robot Dog Test Leg[a]"

[a] 3D-printeable cycloidal drives; back-driveability

https://www.youtube.com/watch?v=QoBgSWkJyM4
James Bruton
"Experimental Harmonic Drive Reducer – 3D Printed"

https://www.youtube.com/watch?v=IXmCze1GsGU
How To Mechatronics
"Harmonic vs Cycloidal Drive – Torque, Backlash and Wear Test"

https://www.youtube.com/watch?v=mPwbDrXq50Q
How to Mechatronics
"CNC Machined vs 3D Printed Cycloidal Drive"

https://www.youtube.com/watch?v=G0DcM60lWSw
Michael Rechtin
"3D Printed Stackable BRUSHLESS Motor Gearbox[a]"

[a] 3D-printed planetary gearboxes

https://www.youtube.com/watch?v=BTzkSg_l70M
Skyentific
"Three Actuators: cheap, powerful and completely 3D printed[a]"

[a]3D-printed planetary gearboxes

https://www.youtube.com/watch?v=kcdj-b49TW8
Akiyuki Brick Channel
"LEGO GBC module: Strain wave gearing[a]"

[a]An awesome demonstrator of the harmonic drive principle built using Lego Technic, though not used as a gearbox.

45.3 Turning Rotary into Linear Motion

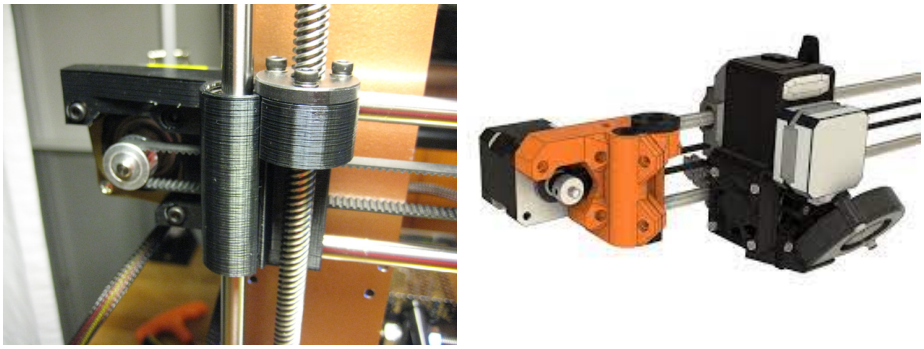


Figure 45.3: 3D printers like the Prusa i3 mk3s use lead screw assemblies for vertical motion and a timing belt drive for left-right motion.

Two good ways to do this are demonstrated by our 3d-printers. The Prusa i3 mk3s uses a lead screw assembly for z-axis movement (up-down) and belt drives for x- (left-right) and y-axis (forward-backward table) movement.

45.4 Ball Bearings and Turntables

There are various different ball bearing designs for different purposes; the main selection criteria, apart from inner and outer diameter and width, are

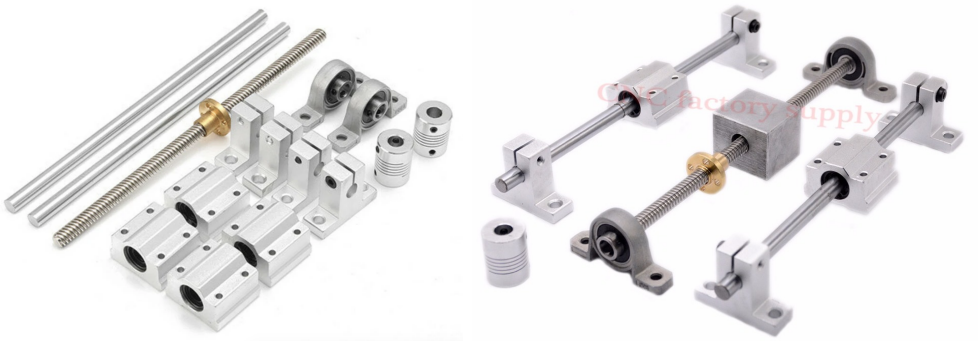
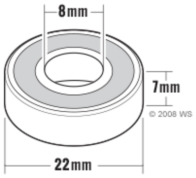


Figure 45.4: A 400mm lead-screw based linear rail kit.

the expected radial and axial forces the ball bearing will have to be able to handle. The website https://www.kugellager-express.de/Home gives a good overview of the different designs.

Note that ball bearings can be quite expensive. One relatively cheap option are skateboard/rollerblade bearings. These have 8mm inner diameter, 22mm outer diameter, and 7mm width. Since they are produced in high volumes, they are relatively inexpensive. You should use them in case they can fulfill the intended purpose in your project.

45.5 Parts in Stock for CS358

item		price (CHF)
		
400 mm lead screw linear rail kit	screw dia 8mm	40
16 tooth drive pulley	axle dia 5mm, 6mm GT2	?
idler	6mm GT2	?
GT2 timing belt	width 6mm	1 (/m)
		
Skateboard ball bearing		1

Chapter 46

Inverse Kinematics

This is a placeholder. Volunteers to contribute this chapter are welcome.

See https://en.wikipedia.org/wiki/Inverse_kinematics.

Note from Alex: Wenzel Jakob's numerical methods course has a good lecture on inverse kinematics.

46.1 Recommended Videos

https://www.youtube.com/watch?v=IN8tjTk8ExI
James Bruton
"How Robots Use Maths to Move"

Chapter 47

Robot Arms

This chapter is a stub and will be extended in the future.

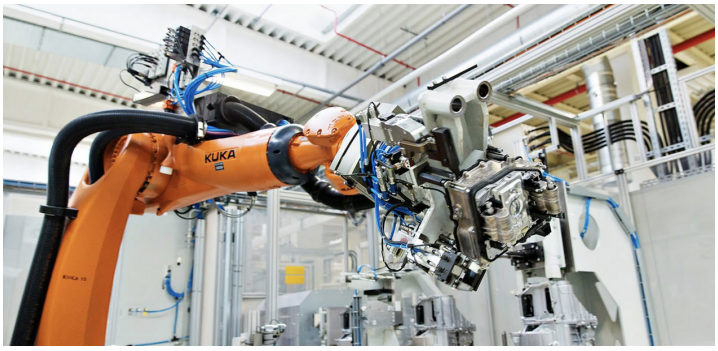


Figure 47.1: Orange is the new black.

First, ask yourself why you want to make an robot arm. Is it to grab and move things in 3D or is it to make an robot arm? Some designs for robot arms are very difficult to make work well. Be pragmatic about getting the job done and choose the simplest design that works for you.

47.1 Not Arms

Consider bridge cranes[1], gantry cranes[2] or the mechanical principle for 3D positioning used by most FDM 3D-printers (such as the Prusa printers

[1] https://en.wikipedia.org/wiki/Overhead_crane
[2] https://en.wikipedia.org/wiki/Gantry_crane

we are using) as alternatives to robot arms. Mechanically, these are simple to achieve because weight and torque do not become important problems, and there is no inverse kinematics problem worth speaking of to be solved by you. Configurable software such as GRBL[3] readily exists for such machines. Such a design may be suitable for you, unless you need to access hard-to-get-to places with your tool/hand, or when bridge crane-like designs are too slow.

47.2 Arms

Do not underestimate the challenge of creating a robot arm that works well. It is not just an issue of programming inverse kinematics; it is surprisingly difficult to build a robot arm strong enough to handle its own weight. (See also Chapter 17 and the discussion there of the rocket equation-like chicken and egg problem that plagues robots that have to handle their own weight.) Remember, stronger motors are heavier and require heavier mechanical parts (such as gearboxes). A robot arm can be a CS-358 project; but do not expect this to be easy.

How many degrees of freedom do you need? Can you use a design where the arm does not struggle under its own weight by either being rigid in the Z (up/down) direction or using a strictly vertical actuation using a screw drive as the Prusa printers do? Check out SCARA robots.[4]

47.3 Hands

If you are building a grabber/tool for an industrial-style machine, be pragmatic and don't get too inspired by your own hands. These are overly difficult and an quite inefficient design. In case your project is aiming to be be bio-inspired, search for *bio-mimetic* hands on the Web, but note that human-like hands are hard to achieve, for many reasons. One is that you will need cable transmissions (tendons), which is much harder to make work robustly than gear or belt drives. (See the video links below.)

[3] https://github.com/grbl/grbl
[4] https://en.wikipedia.org/wiki/SCARA

47.4 Recommended Videos

https://www.youtube.com/watch?v=QCkl9RMd5-s
KUKA - Robots & Automation
"45 KUKA robots welding ladder frames for automotive sector"

https://www.youtube.com/watch?v=cR-YlZ9NdIA
CPM Special Bearings
"ABB Robots Katana Fight "

https://www.youtube.com/watch?v=jaQ5AwH7DGk
Skyentific
"Another 7-axis (7DoF) Brushless Robot Arm (part 4)[a]"

[a]Belt reduction in a robot arm joint

https://www.youtube.com/watch?v=1QHJksTrk8s
How To Mechatronics
"SCARA Robot | How To Build Your Own Arduino Based Robot[a]"

[a]SCARA arms are easier to make work than many other designs – no rotation in a vertical plane

https://www.youtube.com/watch?v=l6xqTcLXXC8
Will Cogley
"3D Printed Biomimetic Mechatronic Hand Explained Part 1"

https://www.youtube.com/watch?v=jKZIvseA1Nk
StanfordCS235, reubotics-dot-com
"CS235: Applied Robot Design, Lecture 7-Introduction to Cable Transmissions"

Chapter 48

Legged Robots

This chapter is a stub and will be extended in the future.

When building legged robots, keep in mind that the feasibility depends extremely heavily on your design choices. The difficulty of making a legged robot varies on a scale from child's play to far beyond the scope of this course.

48.1 Making it Easy

Have a look at the wind-up toys from Figure 48.1. These have a total of one degree of freedom in their movement, for the entire robot, and their huge feet eliminate the balancing problem. This is easy.

Now consider the six-legged robot of Figure 48.2 (a design marketed by several companies under names such as Chipz and Tobbie). This is a robot that can walk forward, change walking direction, and rotate its upper body while keeping the legs stationary with just two motors, just a clever mechanical design. (It's even more clever if you think on the fact that it apparently achieves three degrees of freedom with just two motors.[1] See also the video in the recommendations at the end of the chapter.) If we designed the robot "naively", with individually articulated legs and a motor per joint, we would arguably need at least three actuators per leg, and thus 18 for six legs, with lots of resulting challenges related to inverse

[1] I assembled one once with my godson. If I remember correctly, this is achieved by rotating the motors against each other; as a consequence, it can only do its walking turn and its upper body rotation in one direction each.

Figure 48.1: I received one of these (a properly green one) from Google in recognition of years of service training students they'd hire. The wind-up toy walks very well, and we may safely assume that manufacturing it costs close to nothing.

kinematics, weight, and complexity.

48.2 Making it Hard

At the other end of the complexity scale, we find legged robots like those of Boston Dynamics, in humanoid shape or the shape of the very popular[2] robot dogs. It isn't possible to make these small, and making them big causes all kinds of engineering problems. For such projects, there is no way around big brushless motors and FOC controllers (see Chapter 43) with complex mechanical solutions that produce enough torque while being back-driveable (many gearbox designs are unsuitable, see also Chapter 45). Back-driveability is essential here (see the James Bruton video below).

[2]Judging by the number of such maker projects to be found on the Web.

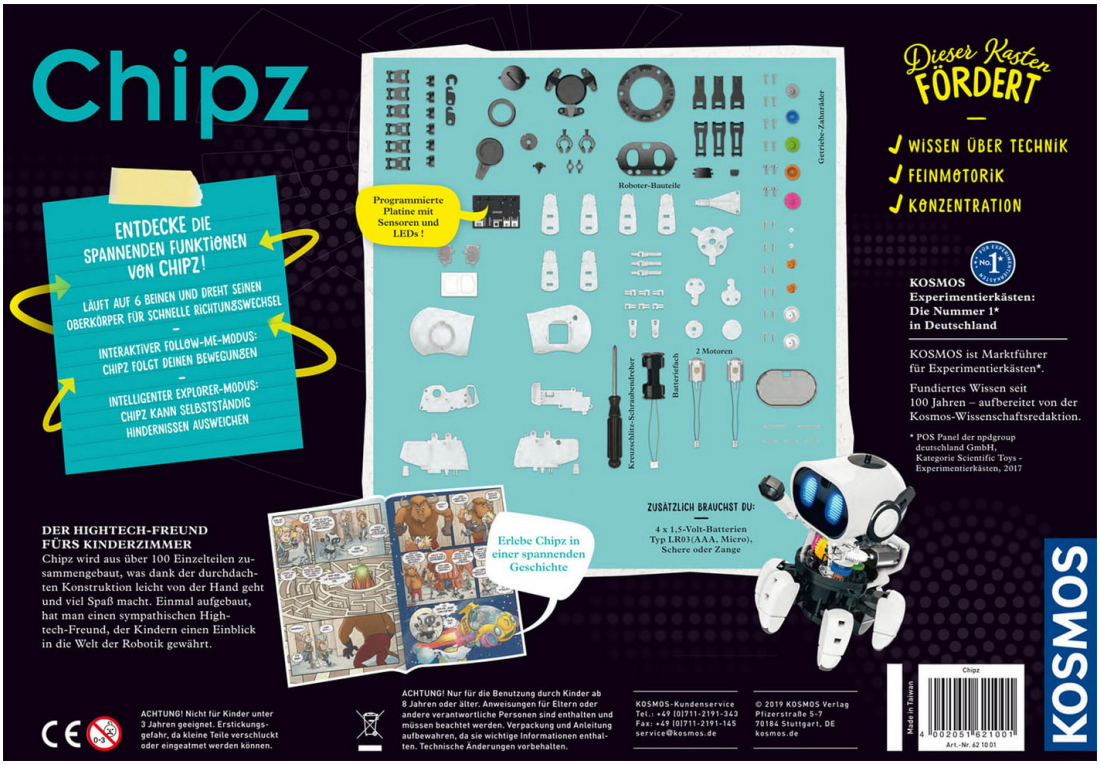


Figure 48.2: A six-legged robot with two degrees of freedom.

48.3 Recommended Videos

https://www.youtube.com/watch?v=i2MqvGVsWhU DIY Trends for kids “Roboter bauen mit Kindern - Experimentierkasten "Chipz" von KOSMOS”
https://www.youtube.com/watch?v=QZt3eJzHLSU EPFL “Six-Legged Robots Faster Than Nature-Inspired Gait”
https://www.youtube.com/watch?v=xNeZWP5Mx9s Massachusetts Institute of Technology (MIT) “Backflipping MIT Mini Cheetah ”
https://www.youtube.com/watch?v=wlkCQXHEgjA Boston Dynamics “Spot Launch”

https://www.youtube.com/watch?v=yXA_KeuYpCY
James Bruton
“Robot Dog V3 - 3D Printed & Open Source #1”

https://www.youtube.com/watch?v=fn3KWM1kuAw
Boston Dynamics
“Do You Love Me?”

Chapter 49

Using Lego Parts

This is under construction.

- Pros: nearly complete system for mechanical constructions, widely available.
- Cons: Expensive, some stigma as a children's toy, not strong enough for large and heavy constructions.

Lego is a good source of gears and axles for low-torque gearboxes; it is tricky to 3d-print gears at sufficient quality. Also, rubber tires.

Do not cut, drill, or glue Lego parts.

Lego Technic dimensions, google.

Bricklink Studio (https://www.bricklink.com/v3/studio/download.page) is a free, high-quality, high-productivity Lego CAD design tool.

49.1 Recommended Videos

https://www.youtube.com/watch?v=RE5ozOUw5s8 Akiyuki Brick Channel "Great Ball Contraption(GBC) at Japan Brickfest 2022"
https://www.youtube.com/watch?v=ZSSH-YfKg6I Akiyuki Brick Channel "LEGO Harmonic Drive"
https://www.youtube.com/watch?v=cXgB3lIvPHI TECHNICally Possible "Little Talks Guitar Cover by Lego Mindstorms EV3"

https://www.youtube.com/watch?v=7mQGgfgjcYU
VirtualMakerLuca
“Lego Technic Liebherr LTM 11200 Crane Details MOC by Jeroen Ottens”

https://www.youtube.com/watch?v=lkpCQqt17ZA
Beyond the Brick
“Amazing LEGO Technic Diesel Engines ”

https://www.youtube.com/watch?v=Nqh-1P615CE
Brick Technology
“Building A Card Shooting Lego Tank”

https://www.youtube.com/watch?v=3crQ09q5Jco
Brick Technology
“Can AIR power a Lego Truck?”

Part VIII

CAD and CAM

Chapter 50

CAD Design in Fusion 360

Leo Wolff

Fusion 360 is a very intuitive and practical tool. However, like any tool, knowing how to use it properly will improve your efficiency and help you avoid many cumbersome mistakes. The structure of this chapter is as follows:

1. Getting Started: In this section, you will find a Fusion 360, step-by-step, tutorial designed for beginners.

2. Project Architecture: This segment aims to help you setup a first project and give you some insights as to how you can organize your workspace.

3. History: This portion gives an introduction to the history feature of Fusion 360, a powerful but not-so-easy-to-use tool for your design.

4. Sketching: Here, we offer good practices and tools you can use to avoid corrupting your sketch and make it robust to change.

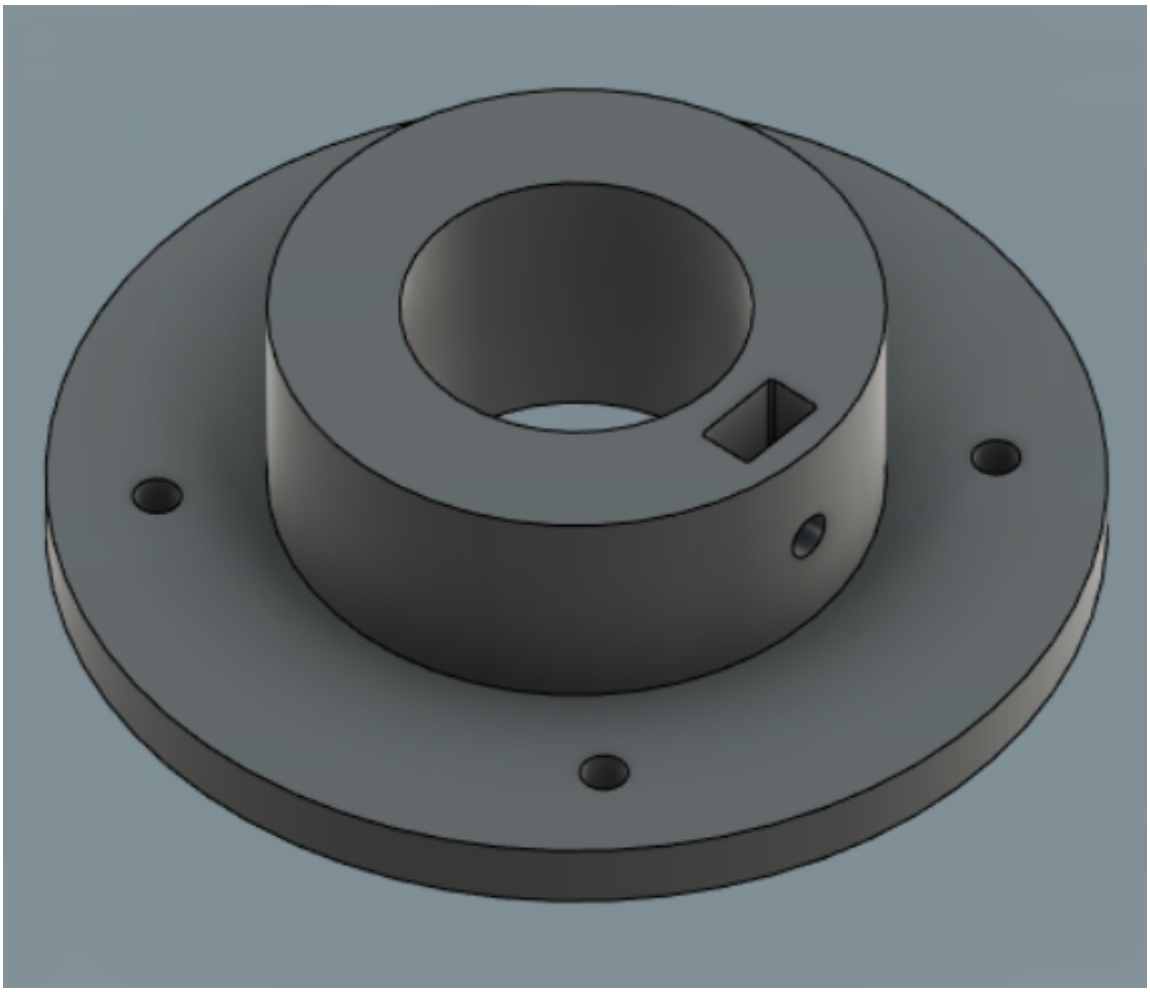
5. Best Practices: This section lists best practices and useful tools among which some will vastly decrease the time it can take you to create a part in Fusion 360.

6. Joints: This division contains a tutorial on how to use Joints to animate different parts of your design relative to each other.

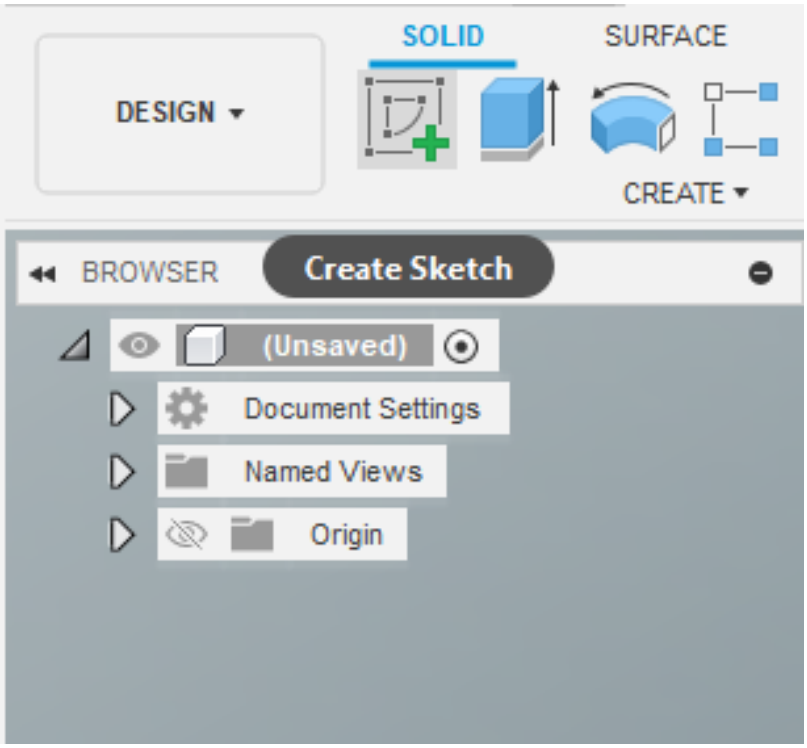
50.1 Getting Started

The following tutorial will guide you through the process required to create a flange coupler in Fusion 360. It is designed for newcomers to this software.

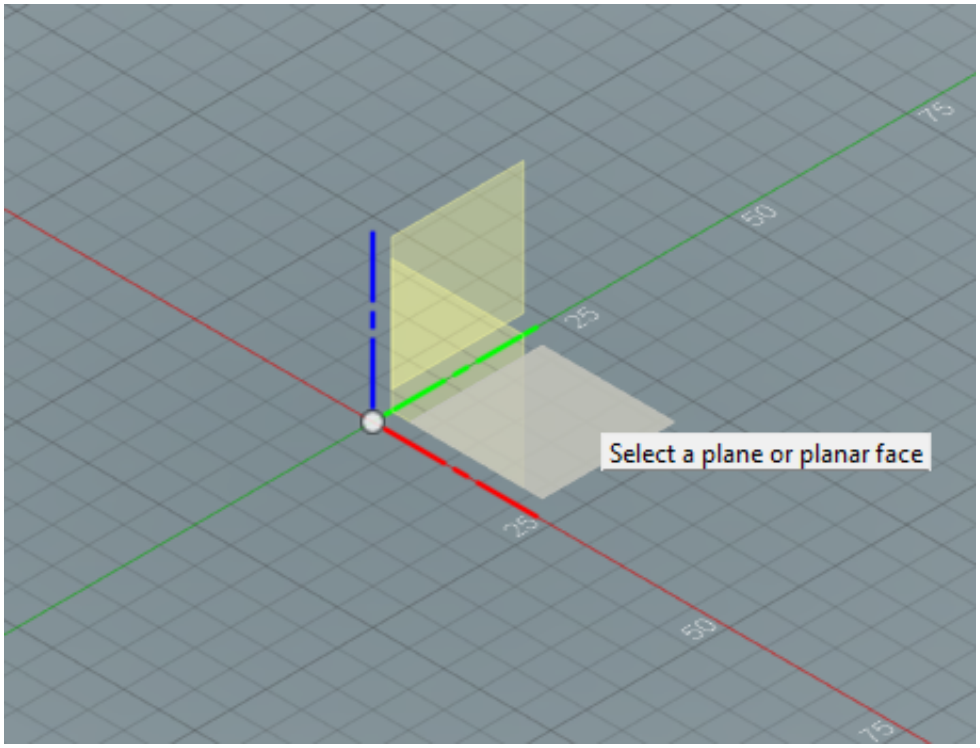
Fusion 360 is a tool one must get used to. Your learning will be much more efficient if you follow this tutorial along and create this model as well.



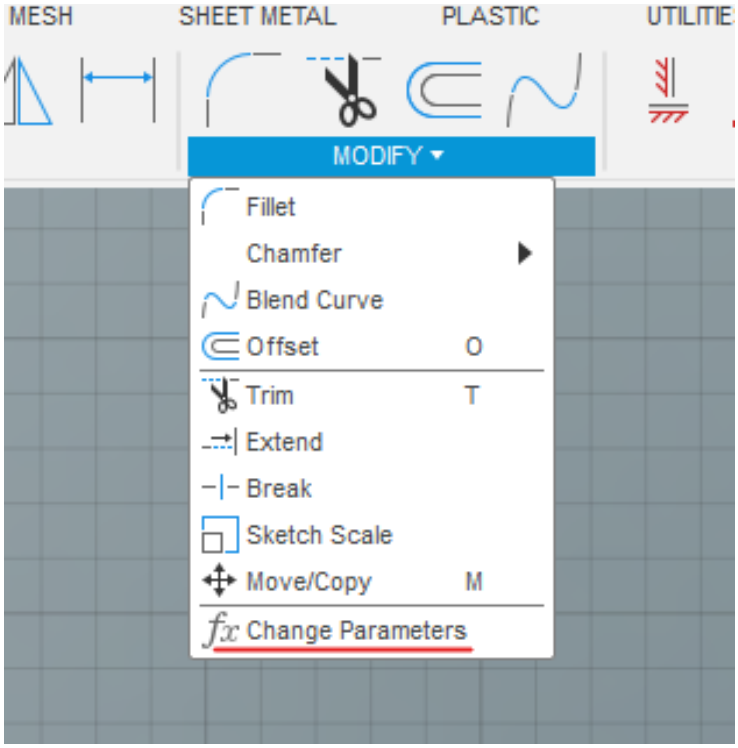
1. Here is the final design for the flange coupler: our goal.



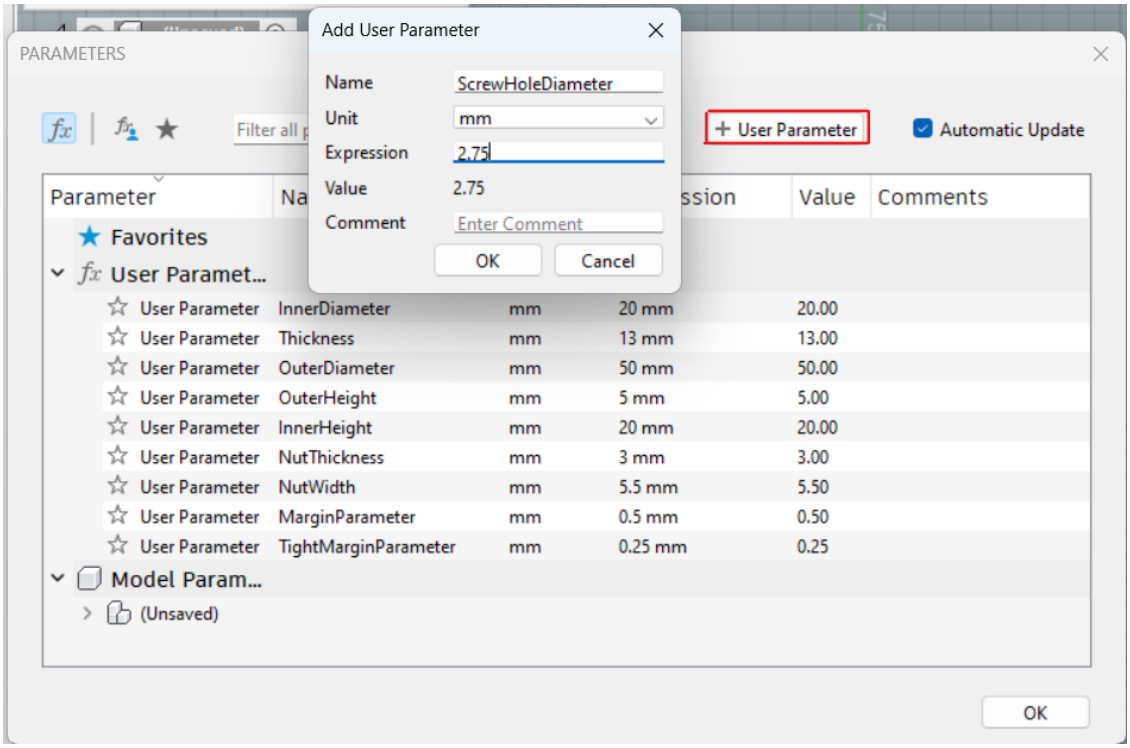
2. Let's open Fusion 360. If you take a look towards the top left of the application's window, you will be able to find the **Create Sketch** button. This is the way you will start every design in Fusion 360.



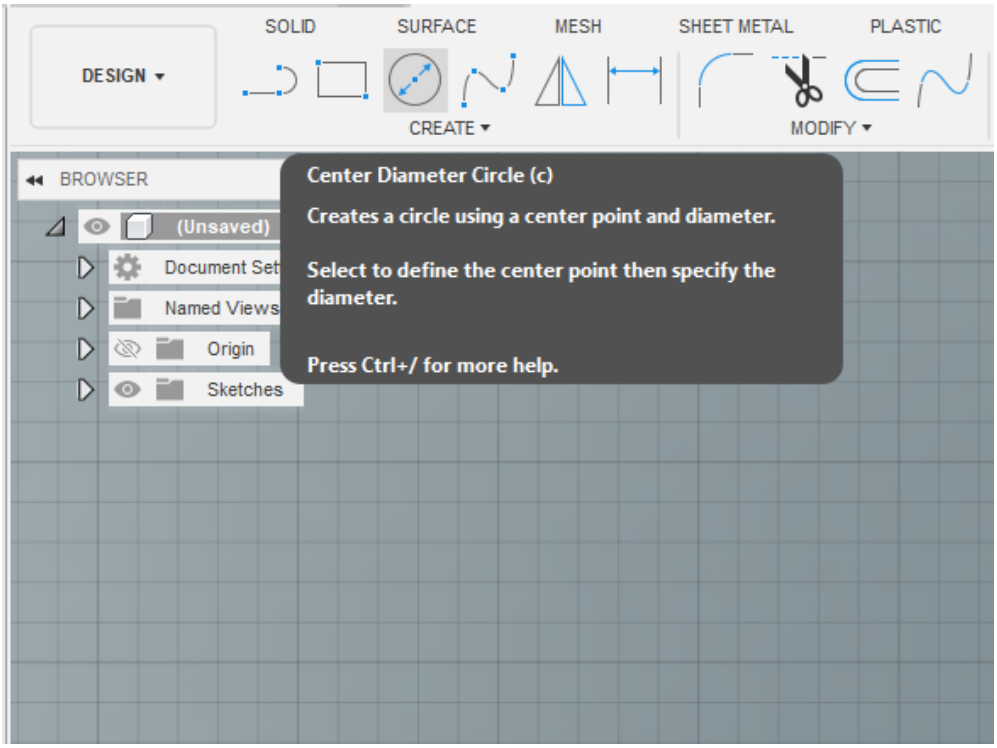
3. Once this button is clicked, select the plane you would like to start your design on. The blue axis represents height. The plane formed by the other two represent the ground.



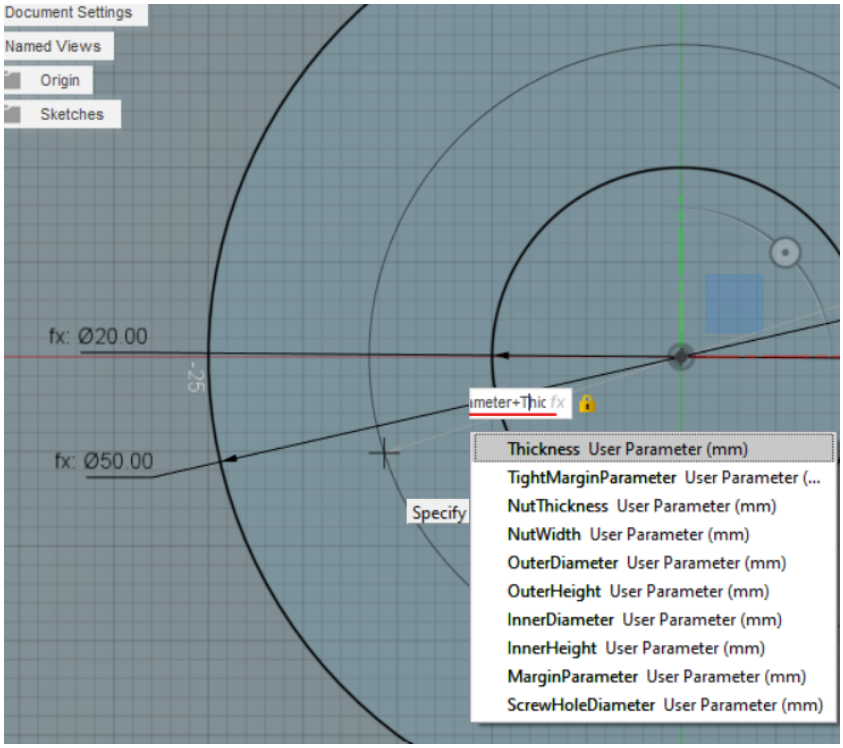
4. We will start right away with good practices and create some Parameters. You can think of them as "constants" you can reuse in your design. You will be able to create such parameters in the **Modify** > **Change Parameters** menu.



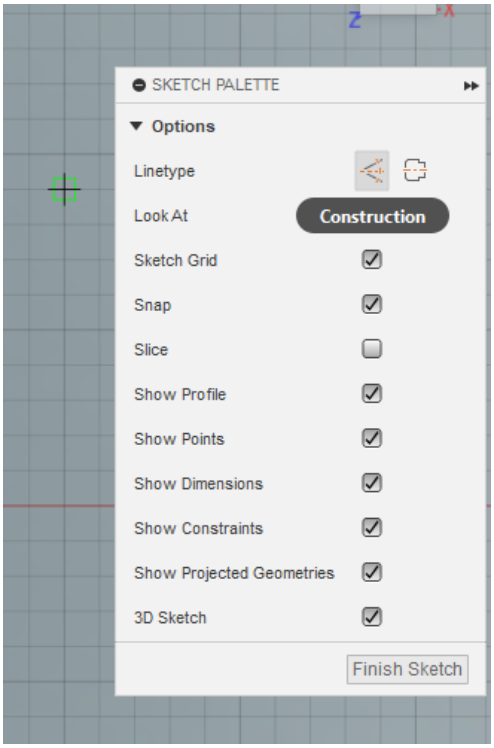
5. By clicking on + **User Parameter**, you can create a new parameter. It requires a name, an expression and a unit. An expression is either a value like 2.75 in this case or a mathematical expression involving other parameters ($Length * Depth$ for example).
   Create the same parameters, with the same values, as shown in this figure.



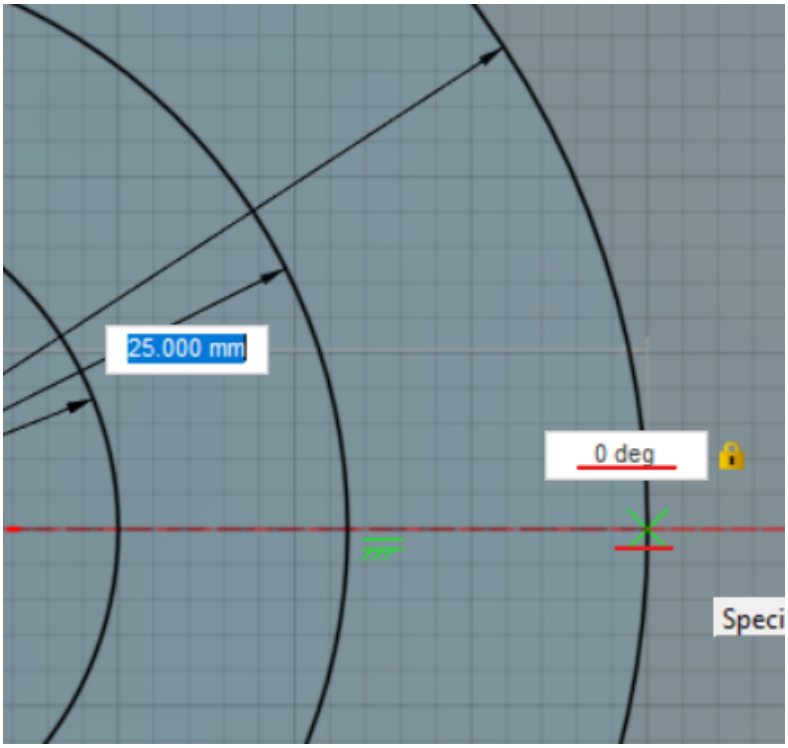
6. Now, let's create circles for the base of our flange coupler. We will later be able to extrude them into volumes. Head to the **Create** sub-menu and choose **Center Diameter Circle**.



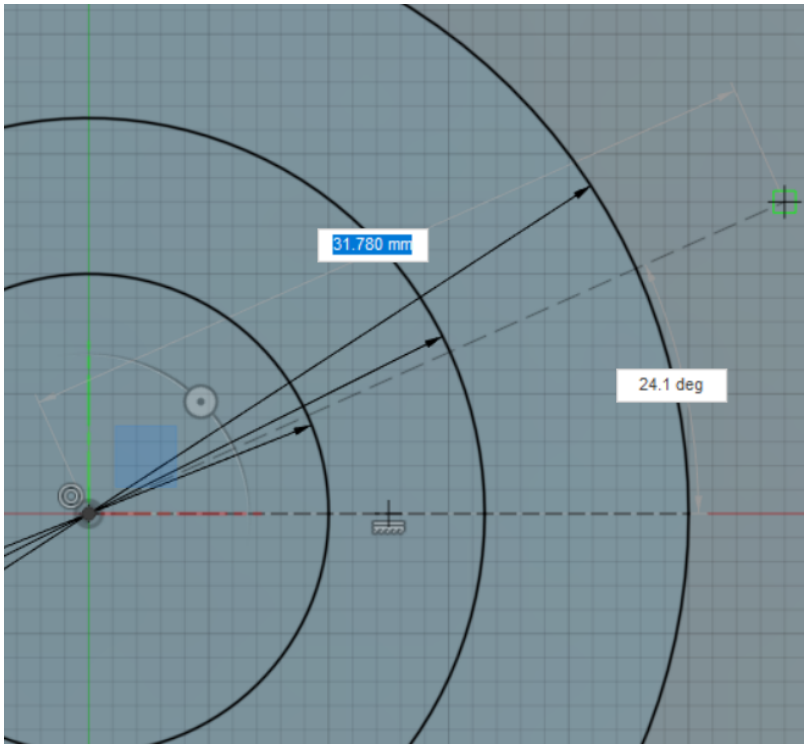
7. Click on the center of the Origin. A text-field prompting you for a radius measure appears. Enter the value of the different parameters we created and press enter each time to draw the circle with the given measure. You should create one circle for each parameter with a name including "Diameter". To create the second boundary for the “wall” of the flange coupler, we use the expression **InnerDiameter** + **Thickness**.



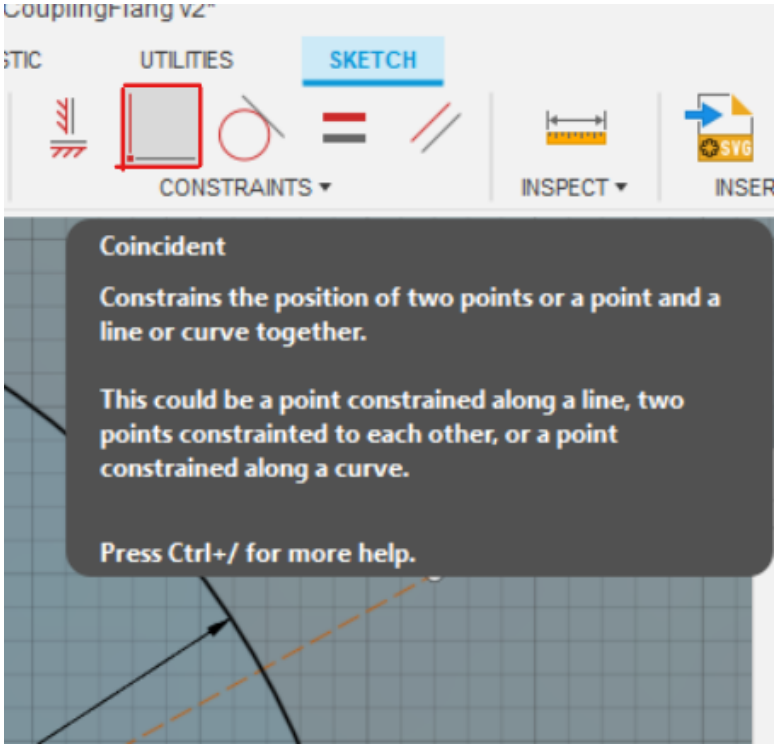
8. We now want to draw the screw holes in the base. We will be able to exclude these holes during the extrusion.
   To precisely position things, we must first create some construction lines. To do so, find the **Line** button in the **Create** sub-menu.
   This line will only be used to sketch other objects, so we can use the **Construction** toggle in **Linetype** in the **Sketch Palette** on the right. This will prevent the line from interfering with the rest of our sketch.



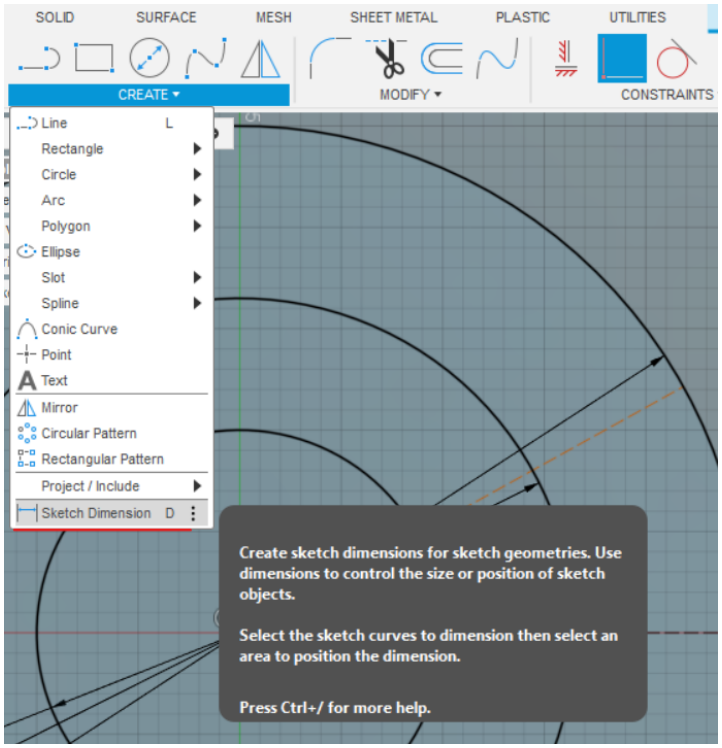
9. Click once in the center of your Origin. Hover over the outer circle and notice the green cross icon that appears. This lets you know that the line will clip to the circle and inherently creates a constraints (in this case "coincident") enforcing this behaviour in case of future modifications. This a very important feature that should use every time you want to do something precise. Enter **0** for the angle and Click once on the outer circle: A radius of the circle is drawn as a dashed line (Since of the **Construction** line type).



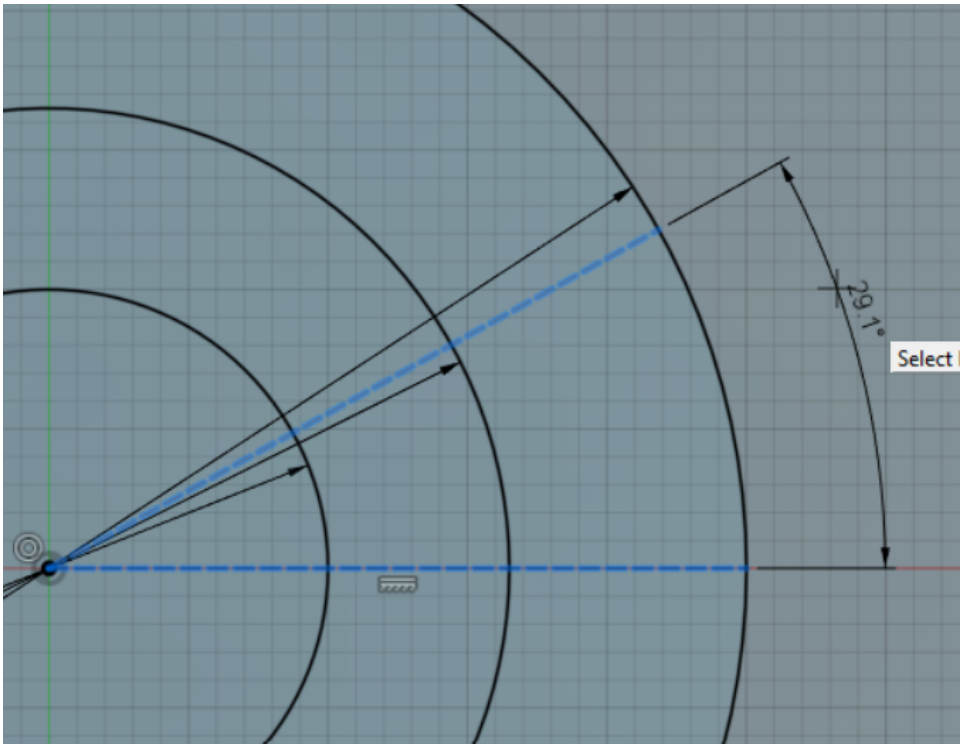
10. Let's now create another line from the center of the circles but this time put its second point at a random position (that is, not clipping to any other object).
    We will build the constraints for this line ourselves.



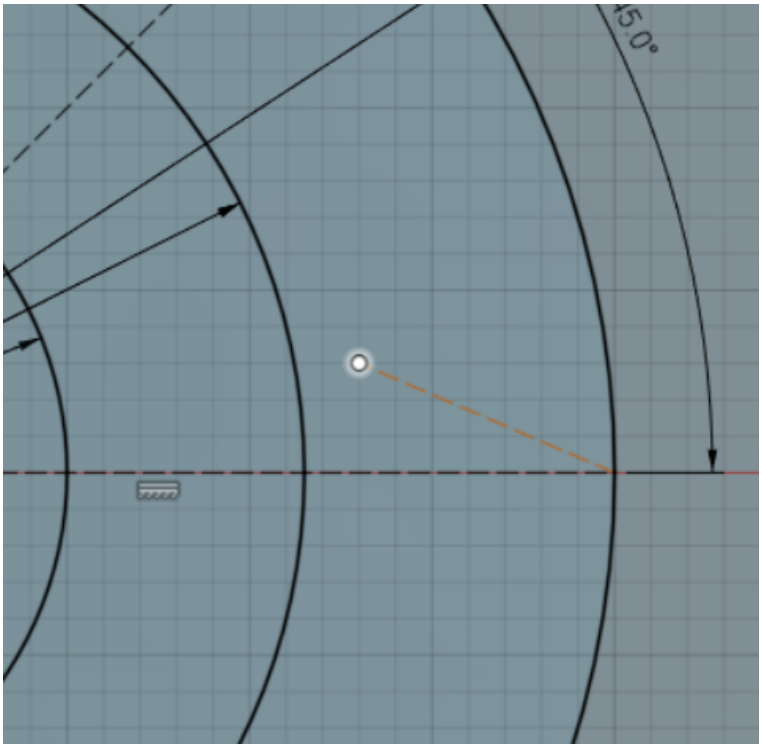
11. Go to the **Constraints** sub-menu and select the **Coincident** constraint. The constraint menu is used to add geometric constraints to your sketch to reduce its degrees of freedom. Click on the point of the line we left free and click a second time on the outer circle in the design. This point now becomes clipped to the circle.



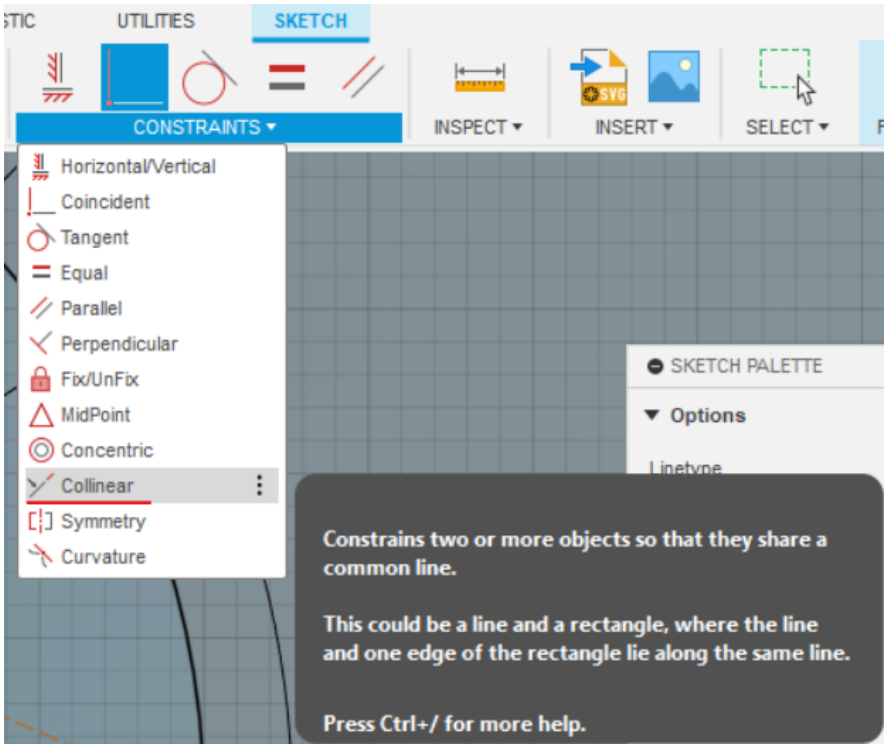
12. We now use a second type of constraint: **Sketch Dimension**. You will find the button under the **Create** expanded sub-menu (or you can press **D**).



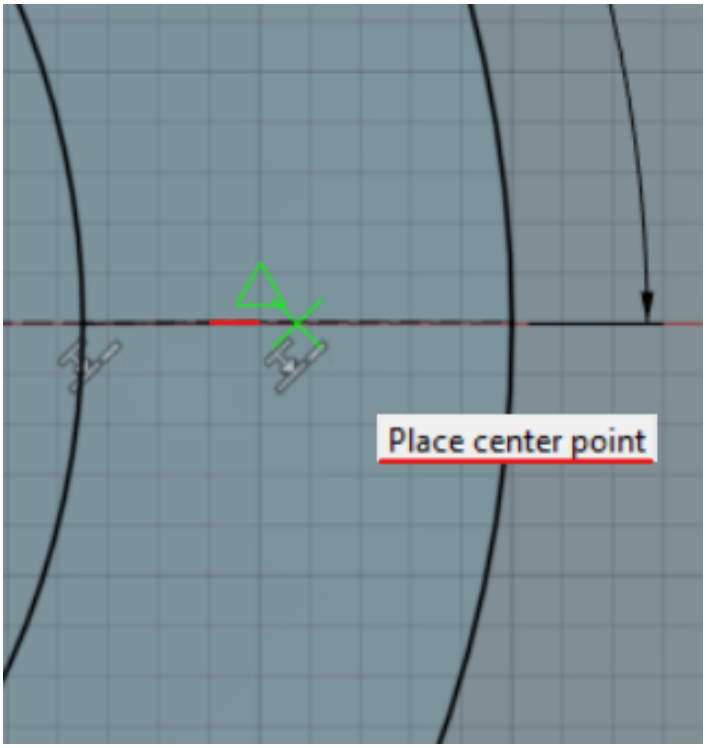
13. Click once on the first line we created, then a second time on the second line. Move the mouse away towards the outer circle. An angle measure input field appears. Use **45 deg** and press enter. This constrains the two lines to be at 45 degrees angle from each other.



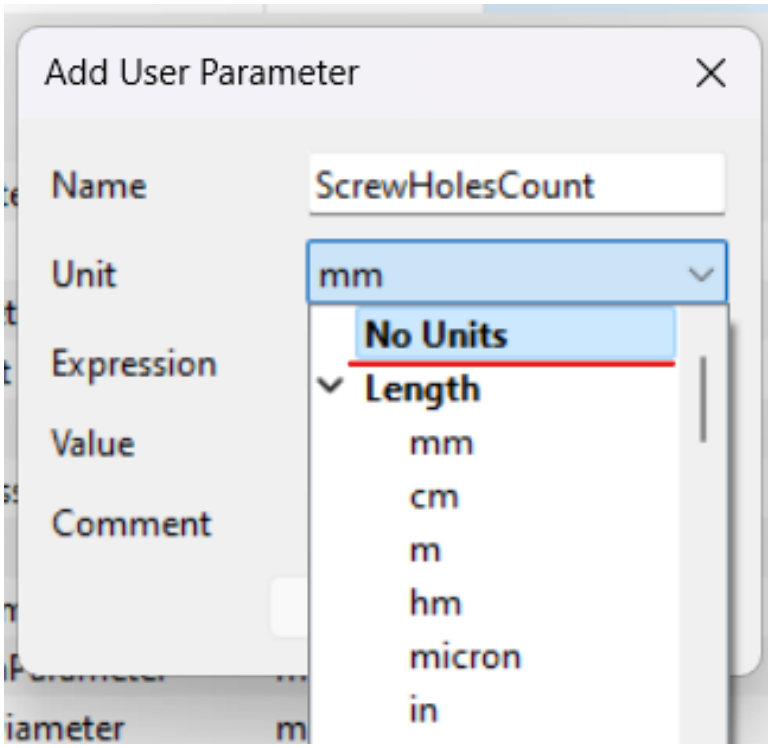
14. To place the hole in the center of the inner and outer diameter, we will need to create a new line which we can reference the middle point from. Create a new line at the intersection of the outer diameter and the first radius and place its end point at a random position.



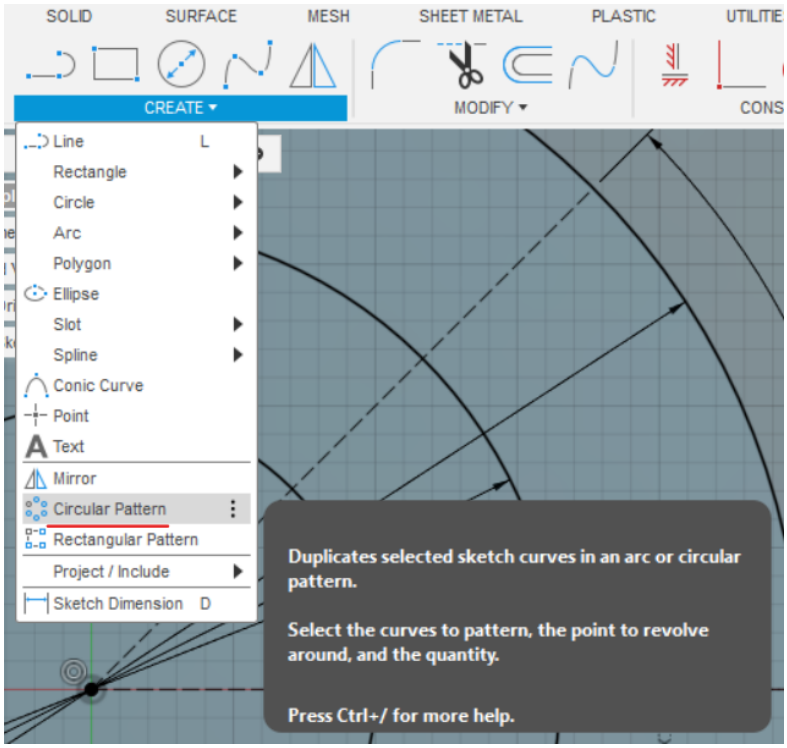
15. Select the **Constraints** > **Coincident** constraint and select the line's end point and then the middle circle. This constrains the line's endpoint to this circle.
    Select the **Constraints (expanded)** > **Colinear** constraint and click on both the last line we created and the circle's radius. This makes the line and the radius co-linear.



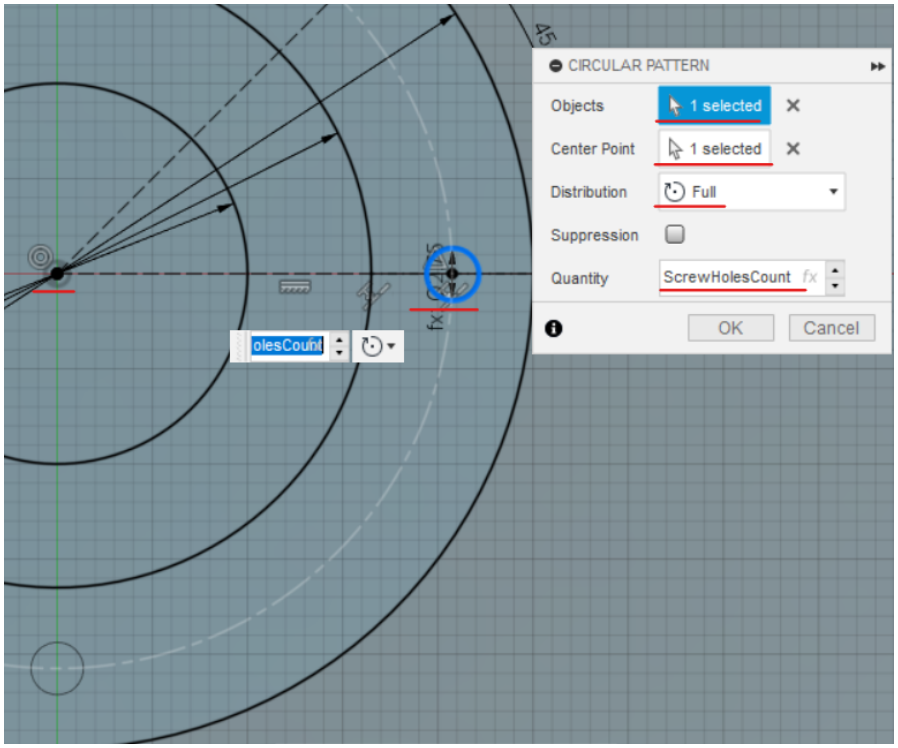
16. You can then select the **Circle** tool again and hover over the middle of the line segment to observe a new “green triangle” icon which informs you that the center of the circle will be clipped to the middle of the line segment.



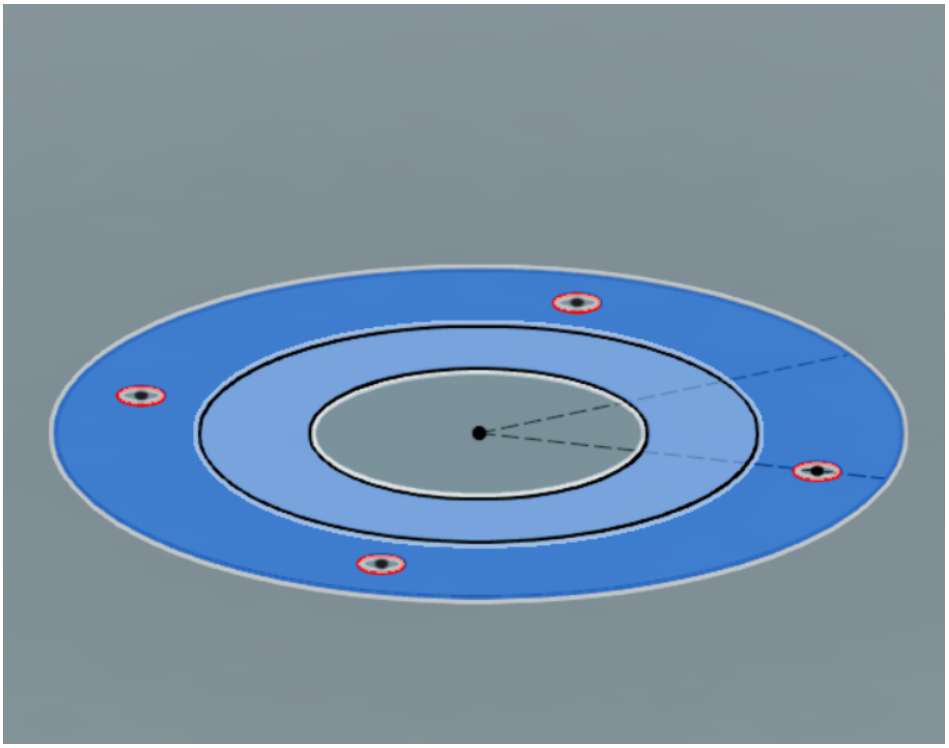
17. We now created the first hole of the flange coupler and need to create the three others. If we have a numerous amount of holes to create, this can become a tedious process. This is why we can make the most of the circular symmetry of the object to speed up your design process. But first, we keep practicing good habits and create a new **Parameter**. This time, we select **No units** to indicate this is a quantity. We then input **4** for the value.



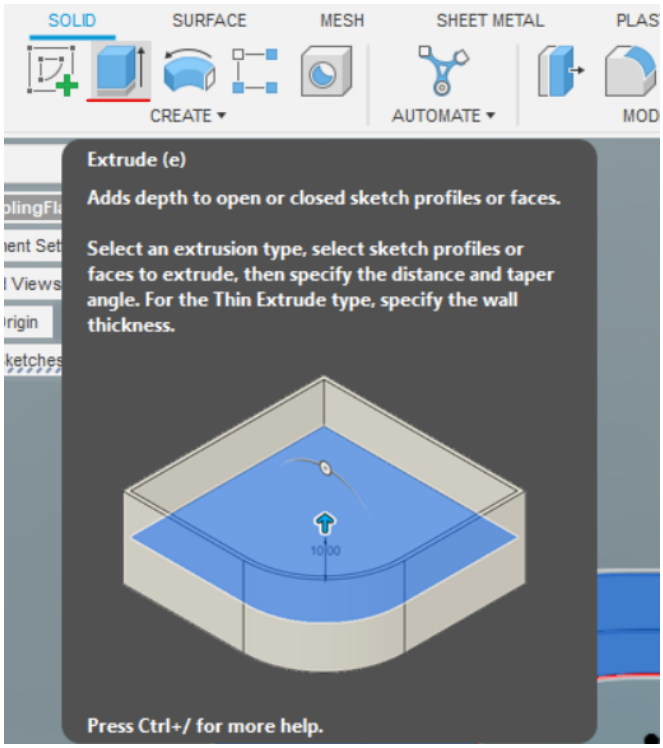
18. Head to the **Create** expanded menu and select the **Circular Pattern tool.**



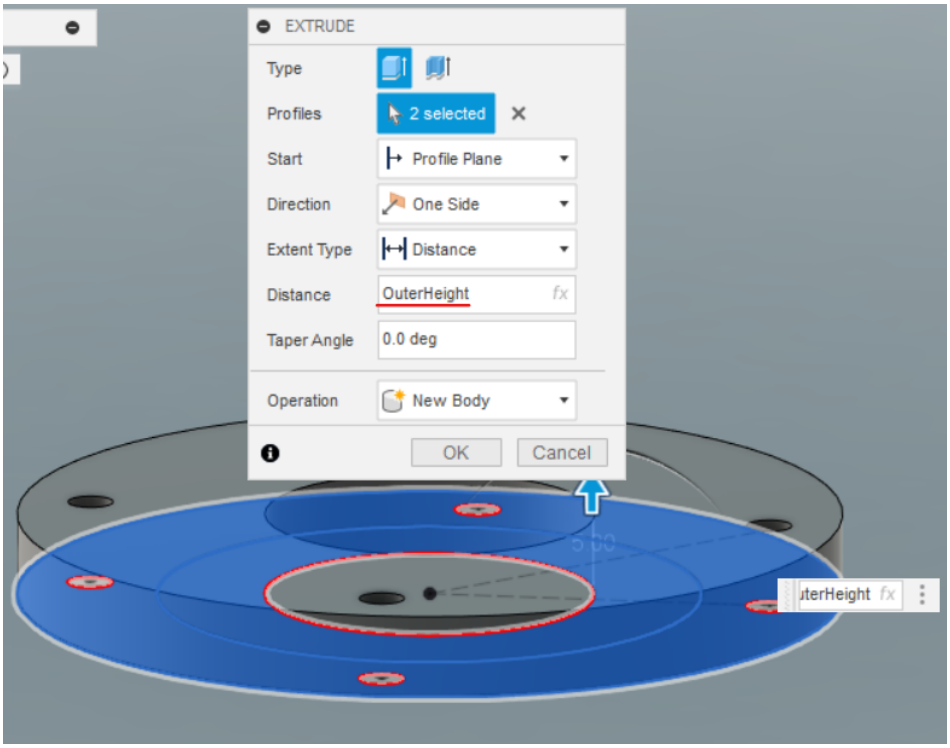
19. This tool allows you to replicate sketch objects or other entities (features, bodies, components) following a specific pattern. First select the object we want to replicate: the screw hole. Then, click on the **Center Point**'s **Select** button and select the circle's center.
    Keep the **Distribution** parameter to **Full** (Which means the range over which we replicate the object is 360 degrees) and input **ScrewHolesCount** for the **Quantity** (We create 4 holes over 360 degrees).
    Note that the copied object is accounted for in the quantity and is not copied again.



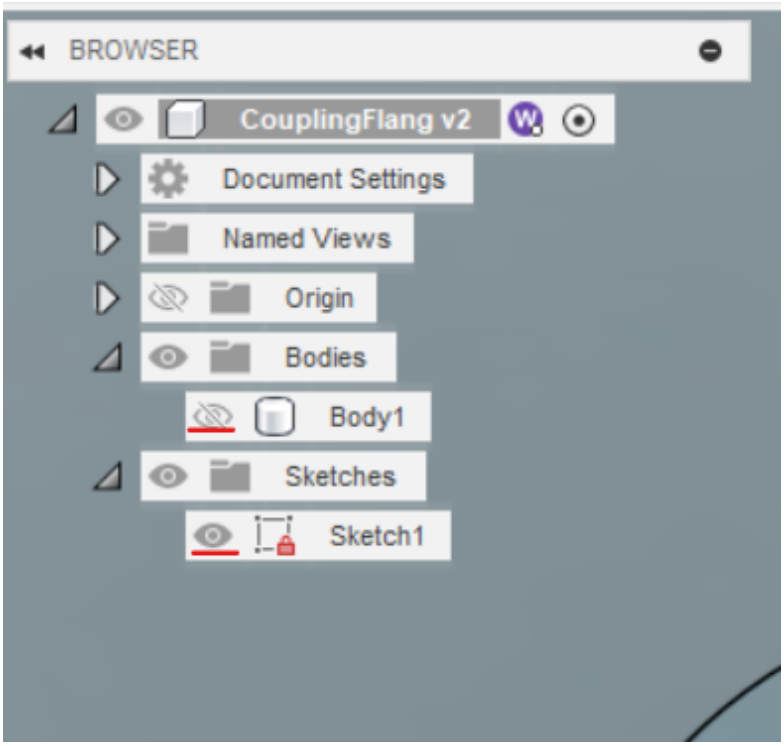
20. The first sketch is finished! Click on **Finish Sketch** and select the two shapes (Outer surface and "wall" of the flange coupler) shown by holding (Ctrl (Win)/Cmd (Mac)) and clicking on the two surfaces as shown in the picture.



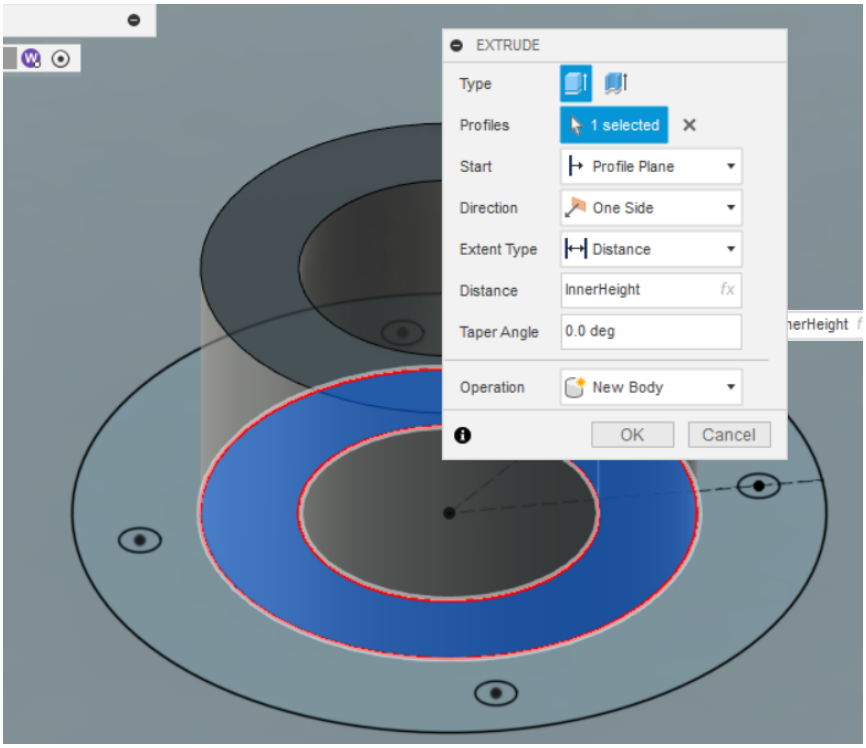
21. Go to the **Create** menu (which is now adapted to volume mode) and select **Extrude** (as shown in the picture).



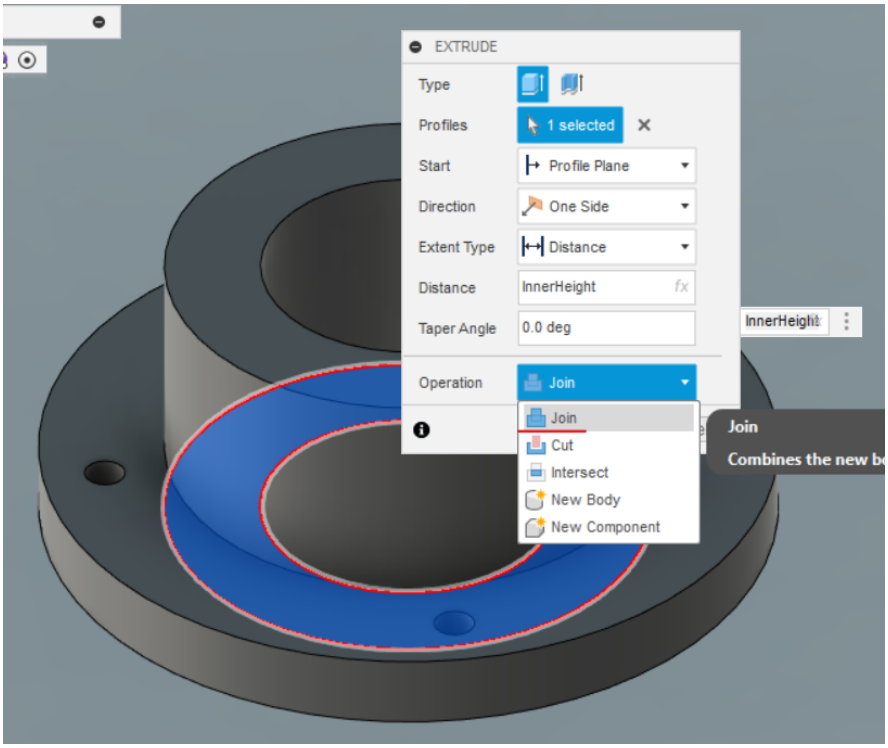
22. For **Distance**, use a previously created parameter (here, called **OuterHeight** with a value of 5mm) and press enter to complete the extrusion.



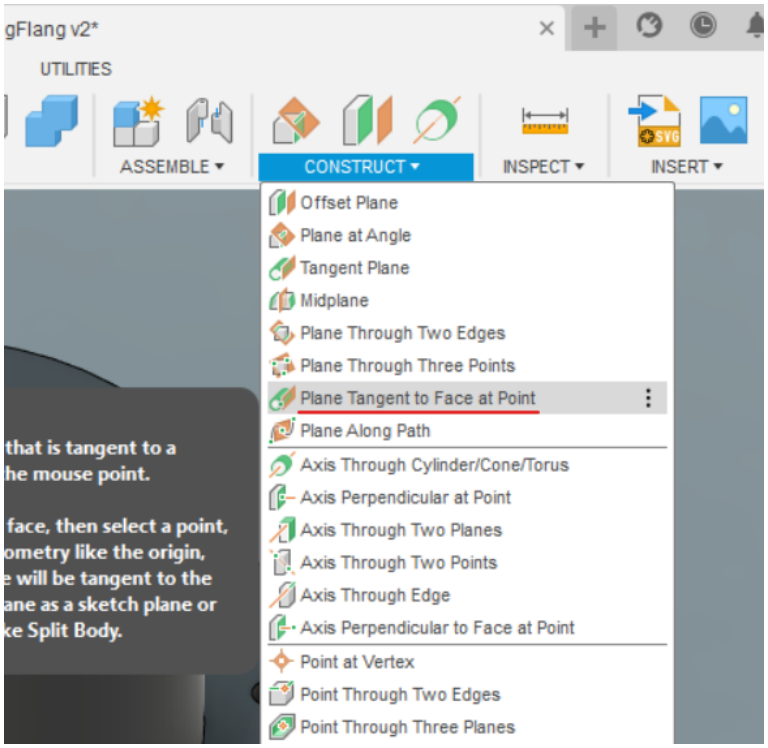
23. To create the second extrusion, we must hide the “body” (the volume we created) and show the sketch so that we can use it again. This can be done in the hierarchy tab on the left of the screen, by clicking on the “eye” icon.



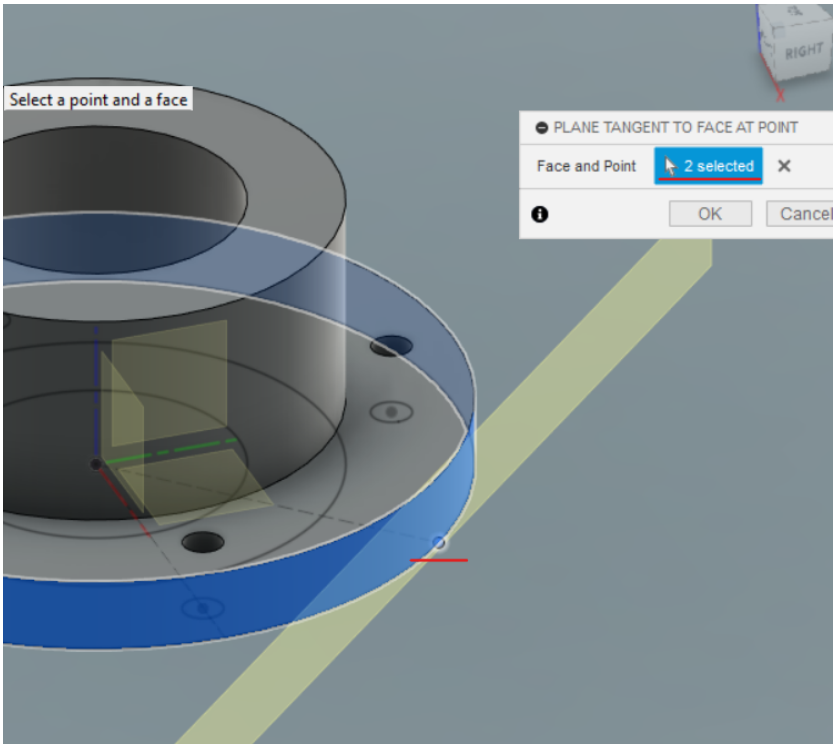
24. First select the inner ring and click again on **Extrude**. This time, we need to select **Join** under the **Operation** menu as the extrusion will overlap with the volume we previously created and the default behaviour is to cut (use the current extrusion to remove a part of the previous one) through it. Join will result in the union of the two volumes.



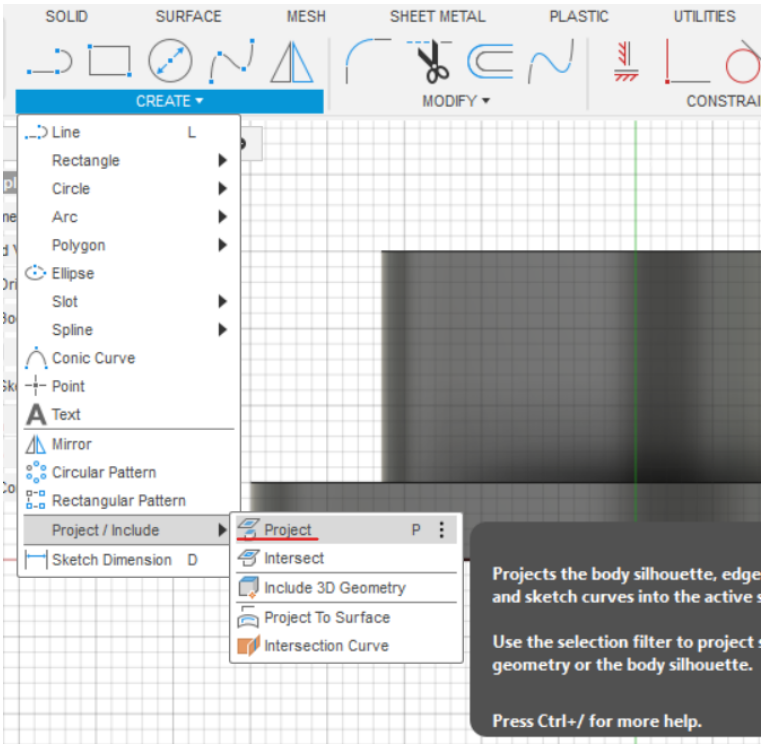
25. Since the volumes which are not visible at the time of the extrusion are not taken into account, we first need to make the volume “Body1” (the outer ring in our design) visible again before completing the extrusion so that it is joined with our second volume. Now complete the extrusion by clicking **Ok**.



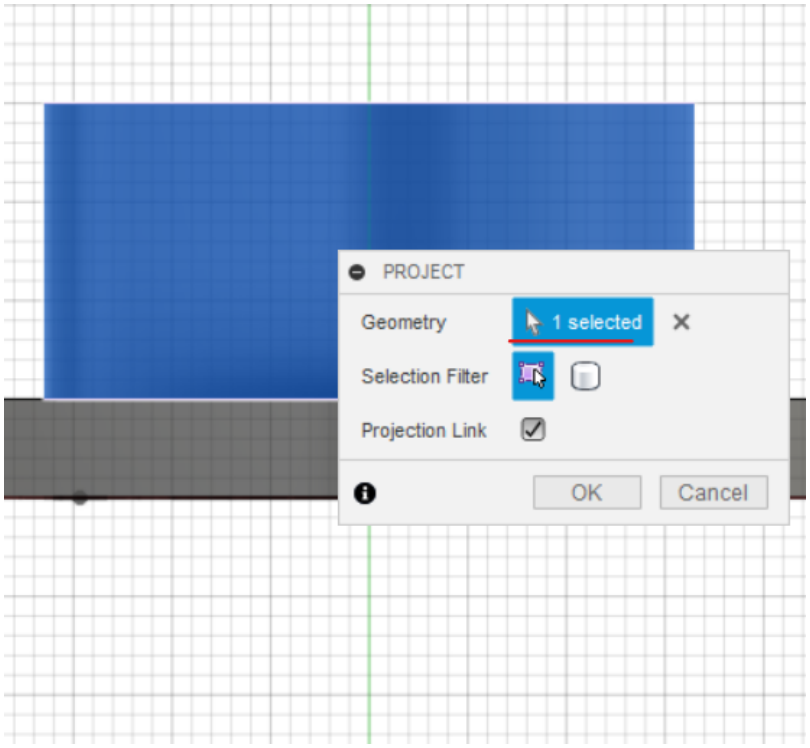
26. We now want to create the hole to insert the tightening screw (the horizontal screw which makes contact with the axle which we will mount this part on). To this end, we can select the **Construct (expanded)** > **Plane Tangent to Face at Point** tool.
Take a moment to note all the possible ways of creating a plane/line from a reference: These are very handy depending on the situation you find yourself in and the references you have available.



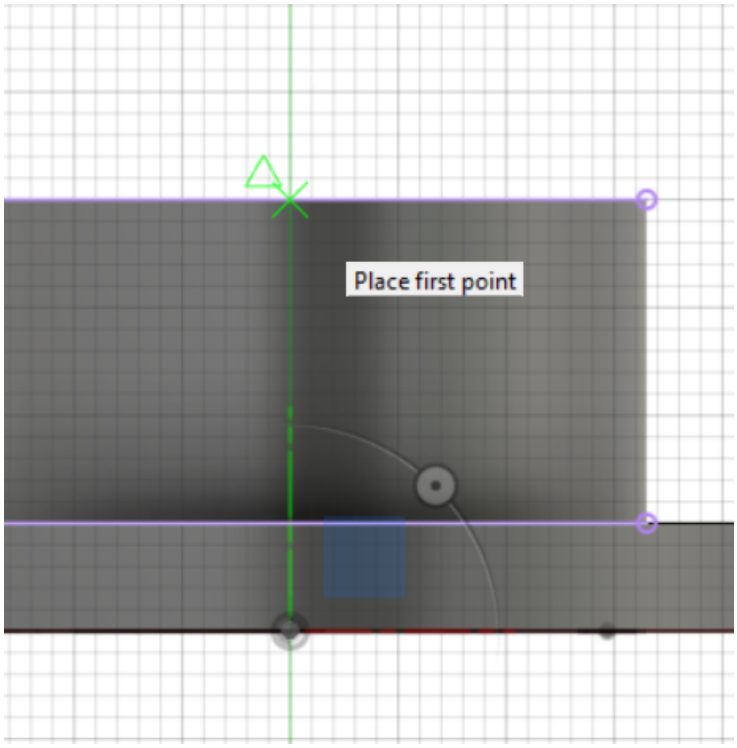
27. At this point, the initial sketch we created should still be visible and we should be able to highlight the point underlined in the figure. This will serve as a reference to create the plane we need to sketch the screw hole in.
    Complete the plane creation (the point should still be referenced) by selecting the face of the outer ring and clicking **Ok**.
    We can now **Create Sketch** on the plane we just created.



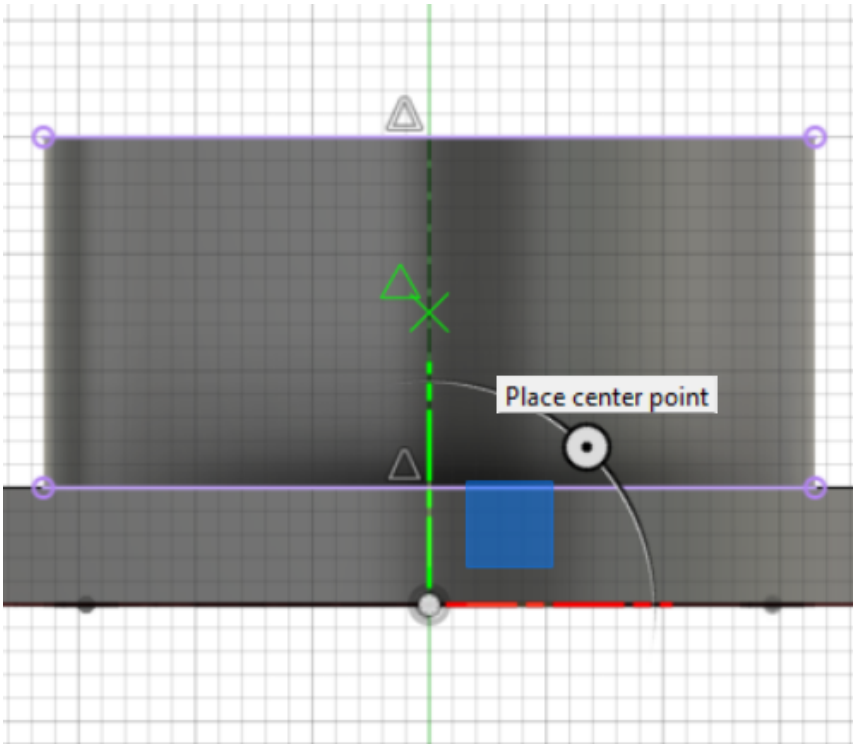
28. To perfectly place our screw hole in the middle of the upper face, we need to create more references: by default, only the face directly on the plane is accessible for references. Go to **Create** > **Project/Include** > **Project** (or press "**P**"). This tool enables you to project faces/geometry onto the sketch plane to use them as references.



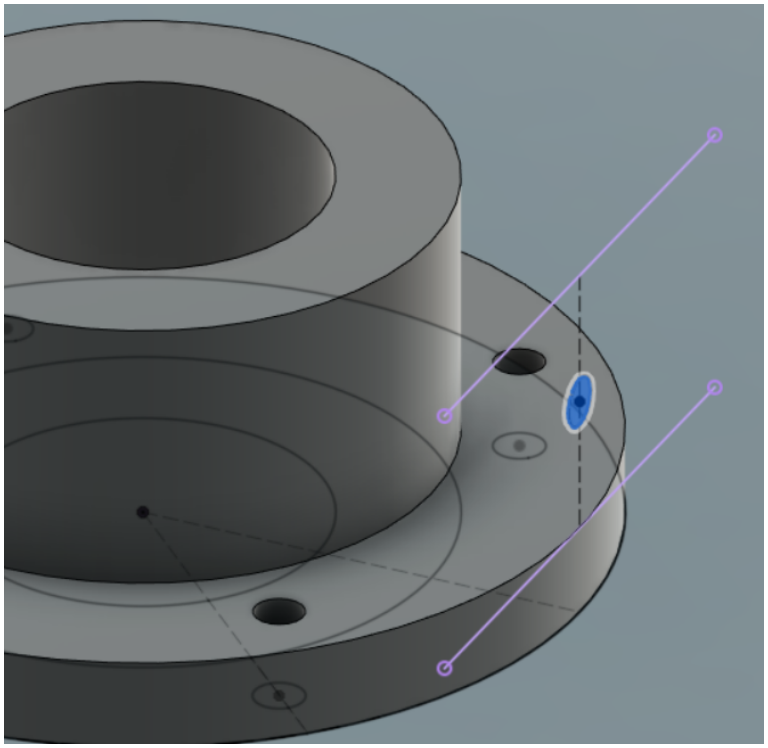
29. Click on the upper face of the design, as shown on the screenshot above and click **Ok** to project it.



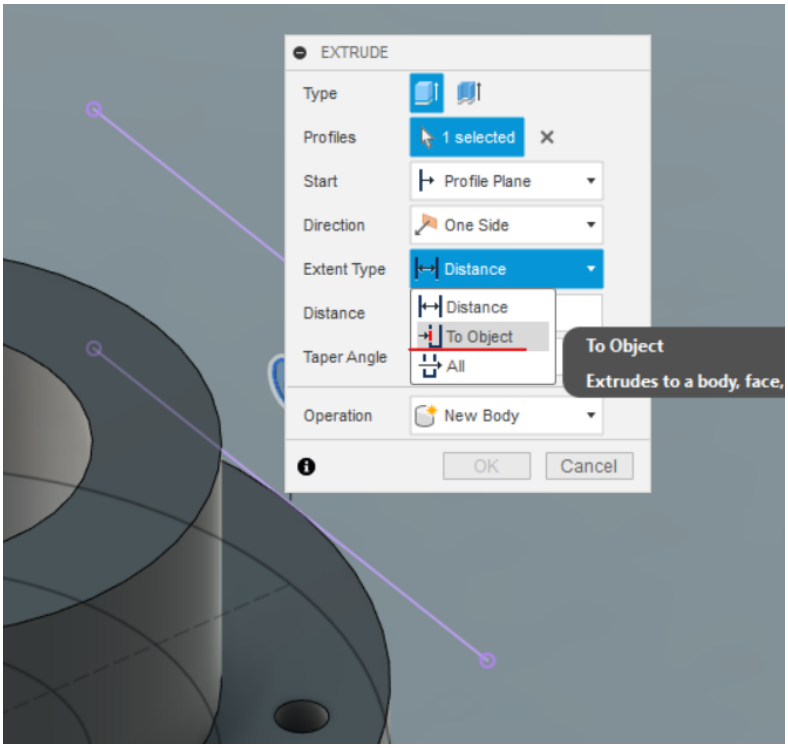
30. Now that we have our reference, let's create a new line (in **Construction** mode, accessible from the Sketch Palette on the right of the screen). Hover over the upper segment as shown on the image above to see the **Middle point** indicator (green triangle). Click once.
Hover now over the lower line and do the same thing, click a second time to create the line.



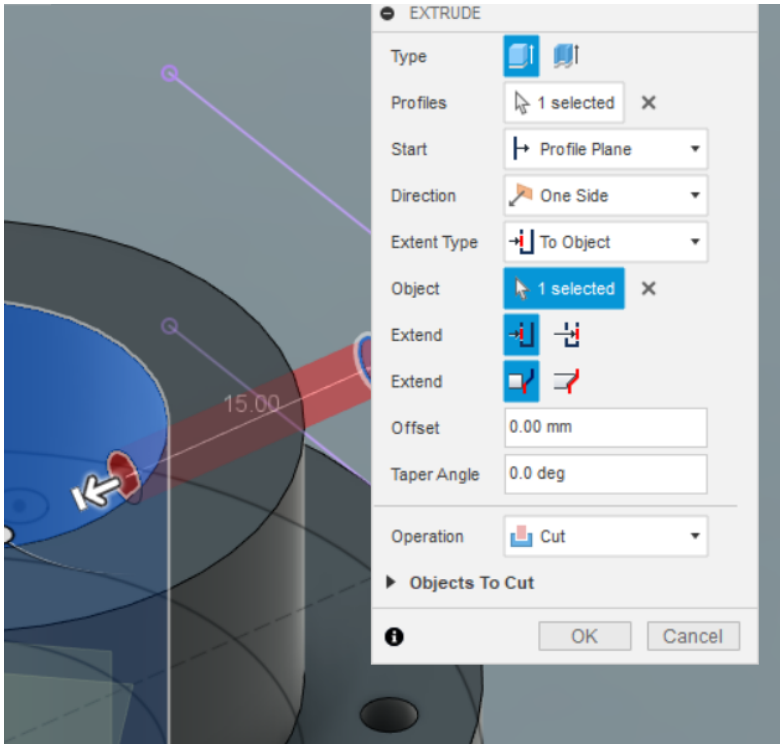
31. Now that we have everything we need, we can create the screw hole. Create (with **Construction** mode disabled) a new circle and click once on the midpoint of the line segment we just created to place its center. Input the value of the **ScrewHoleDiameter** for its diameter and press enter. Click on **Finish Sketch**.



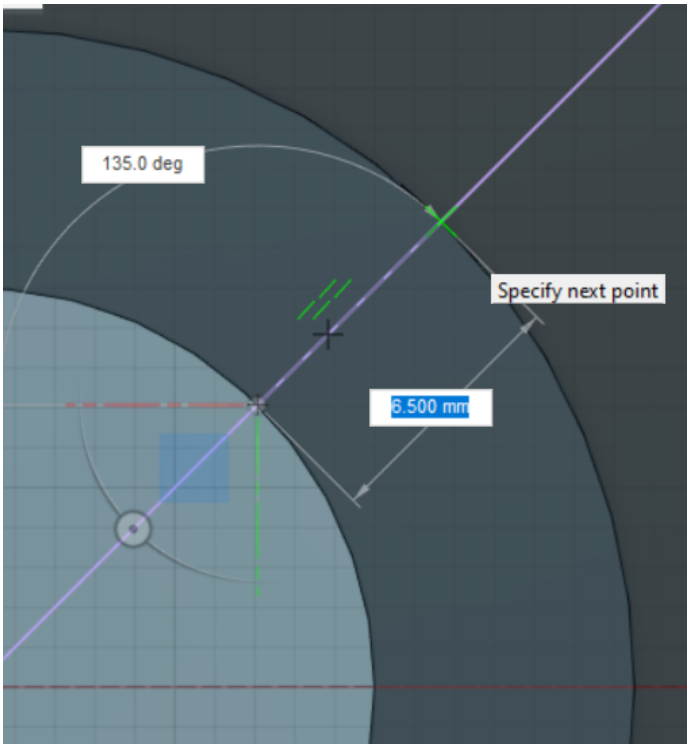
32. We can now select our newly created circle and start a new extrusion to remove the material where we want to create the screw hole.



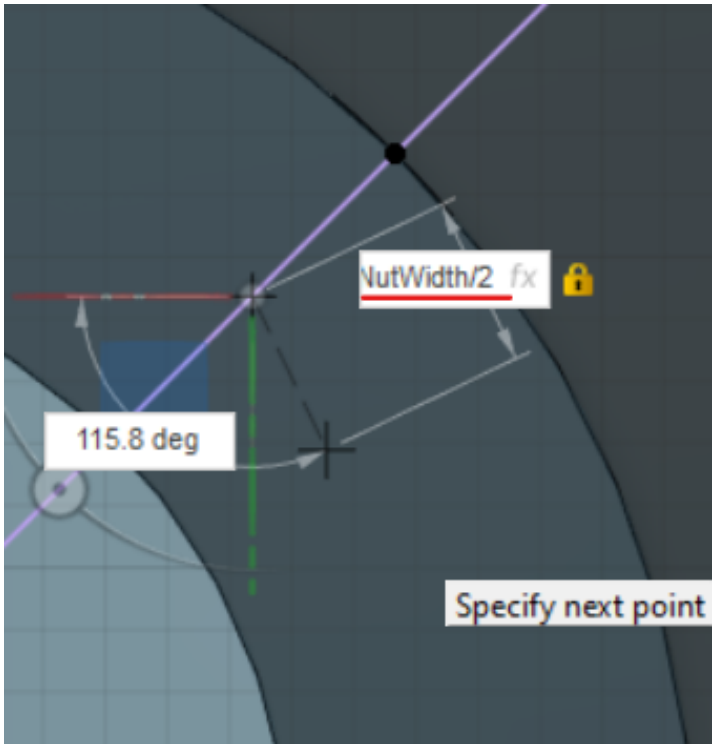
33. Since we want it to precisely stop at the inner diameter, we select **To Object** under the **Extent Type** menu. This will tell Fusion 360 that we want the extrusion to stop at the profile of an object we select.



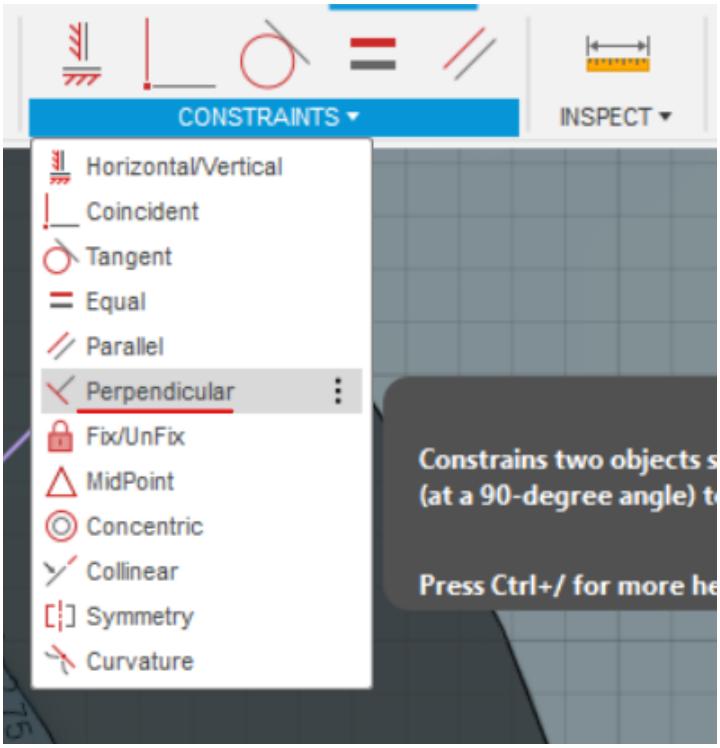
34. Under **Object**, select the inner diameter as shown in the picture above. You'll see a preview of what Fusion 360 wants to do. Note that because the extrusion adapts to the profile of the object, its respects the curvature of the part at the end. (The end of the extrusion is not flat.)
Complete the extrusion by clicking **Ok**.



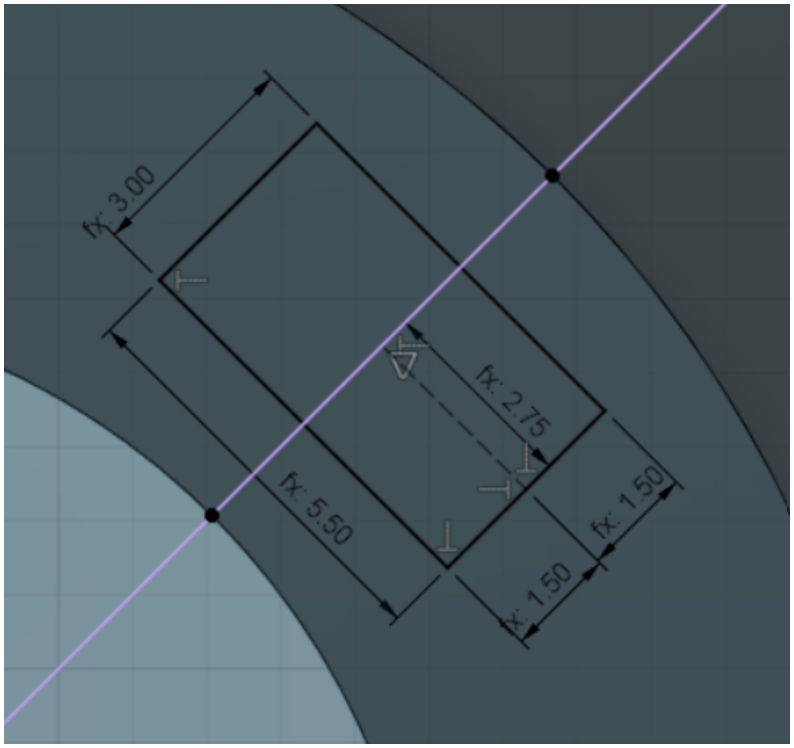
35. We can now start the last part of our design: Creating a nut slot.
    Start a new sketch on the upper face of the design.
    Project the line from the previous sketch we had put at a 45 degrees angle (the sketch has to be set visible if it's not already the case, which can be accomplished by clicking on the "eye" icon on the left-hand side of the sketch in the hierarchy).
    Create a new line from the two intersection points of the face's bounding circles and the projected line as shown in the above picture. Be sure the points clip to the intersections.



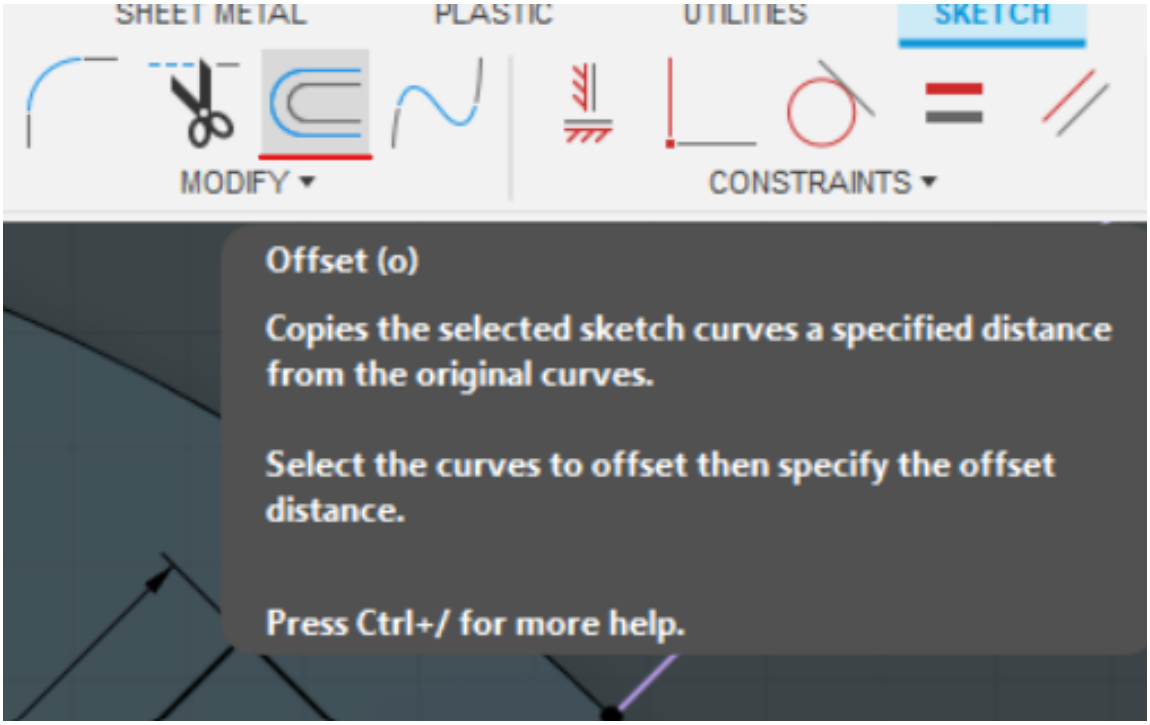
36. Start a line at the middle of the previous segment and position it at a random position after inputting $NutWidth/2$ as a length expression.



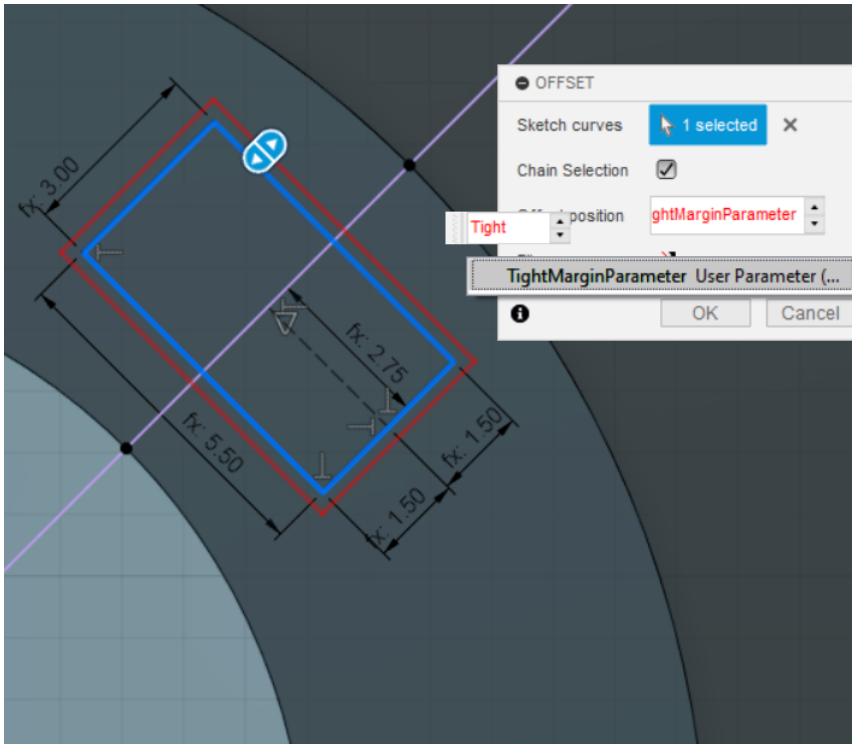
37. Select the **Constraints (expanded)** > **Perpendicular** constraint and apply it to the two last segments we created, by clicking on both of them once the constraint is selected.



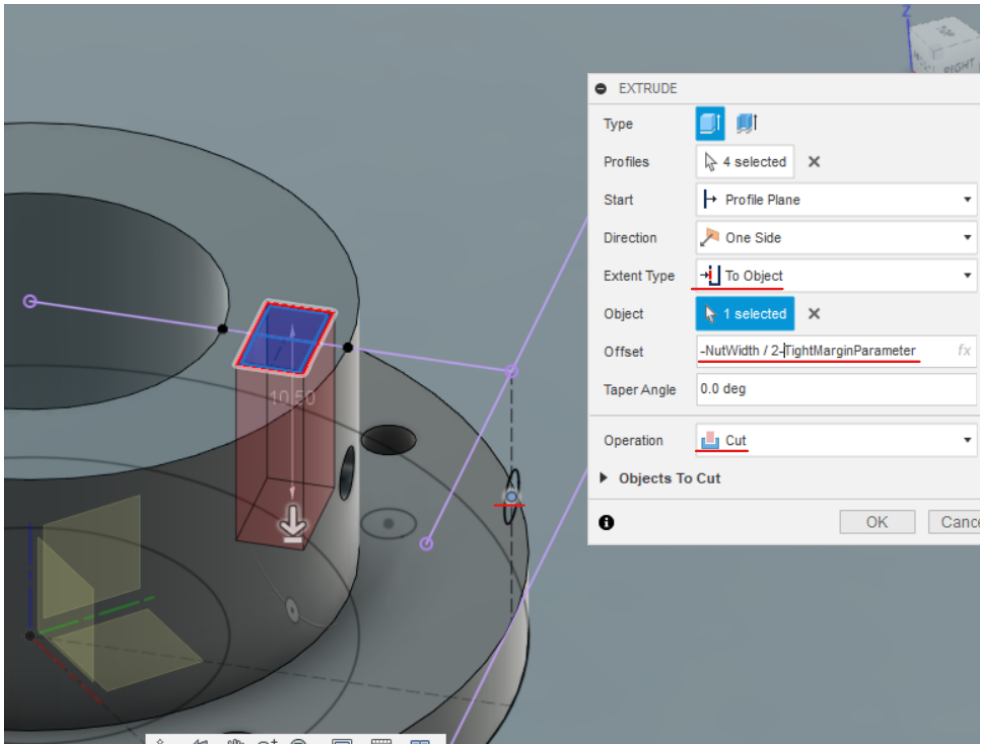
38. Use these techniques to create a full rectangle of height **NutThickness** and width **NutWidth** as shown in the picture above.
We don't use the **Rectangle** tool here because the design is positioned at an angle relative to the origin of the sketch.



39. Since we have to fit an object into the slot we will create, we will use some margin to be sure it fits correctly. Select the **Modify** > **Offset** tool and select the rectangle your created.



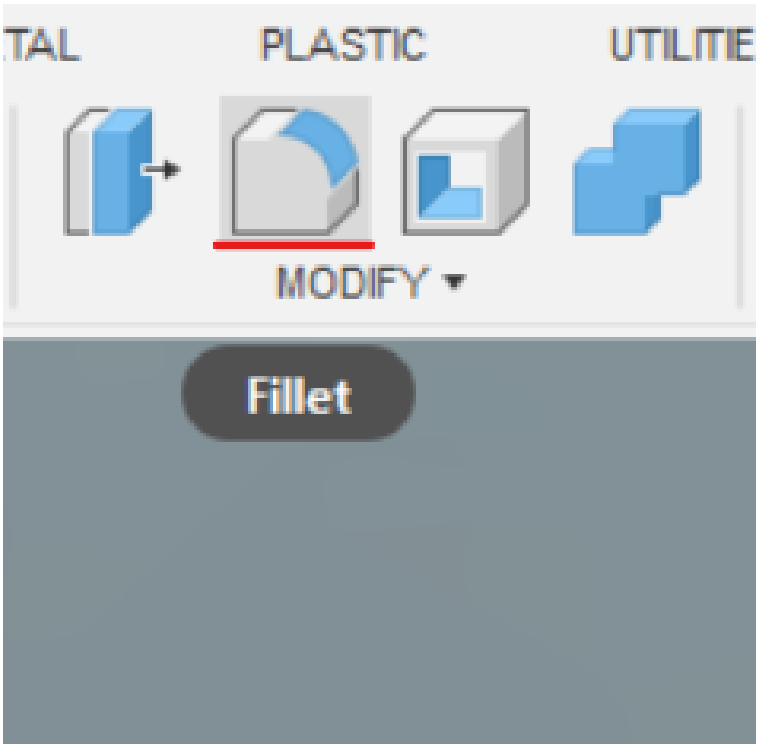
40. As a margin value, input **TightMarginParameter** (Used when an object should fight tightly, we would have used simple MarginParameter if it could be more loose).
Click on **Finish Sketch**



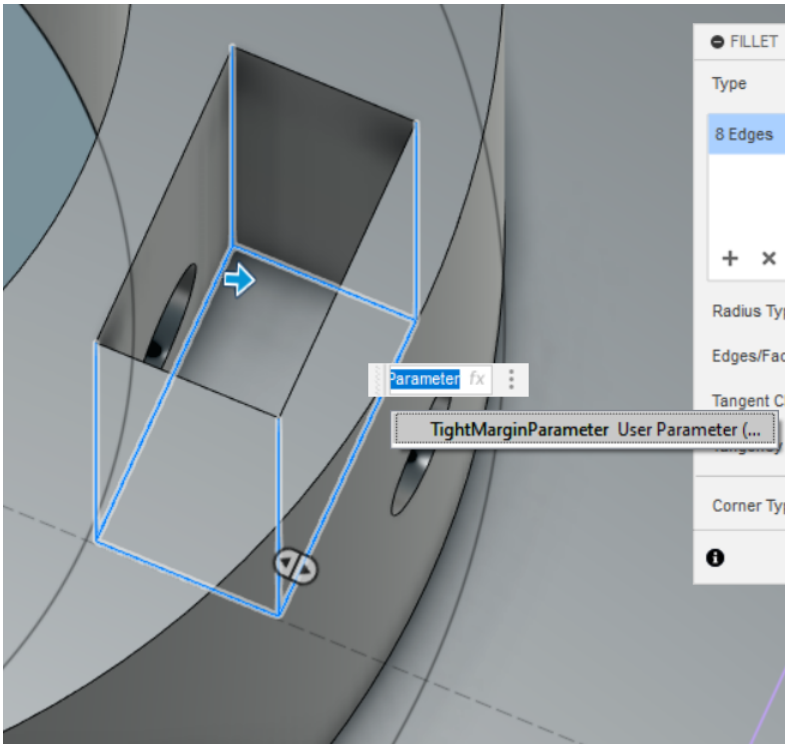
41. Select all parts of your sketch (Both inner rectangle and its margin) and create a new extrusion.
We use **To Object** as **Extent Type** with the center of the screw hole selected (available if the sketch is set to visible) to make the extrusion stop at this point.
Since we want the center of the nut to be aligned with the center of the screw hole, we still need to add the following expression as an **Offset** value:

$$-\text{NutWidth}/2 - \text{TightMarginParameter}$$

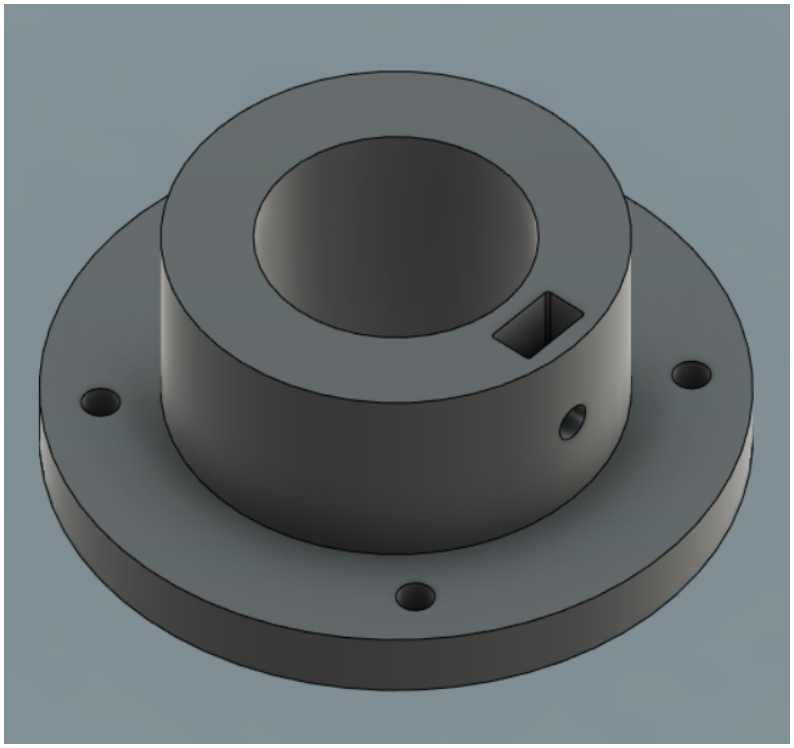
The **Operation** has to be set to **Cut** (which should be set by default by Fusion 360).
Complete the extrusion by clicking **Ok**.



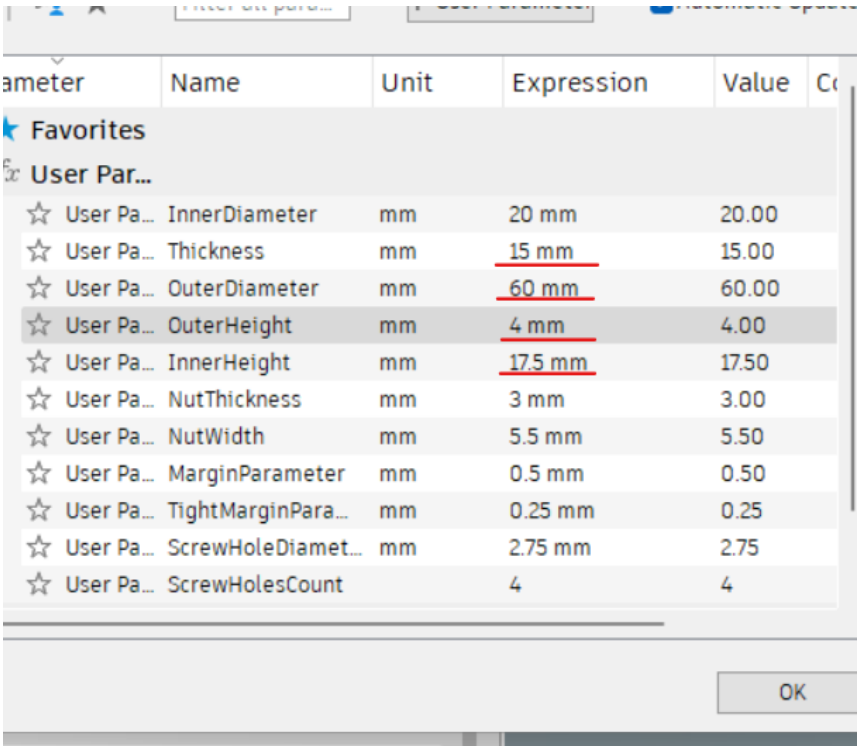
42. To strengthen the weak points of the design (the right angles) and make the part more elegant, we use the **Modify** > **Fillet** tool.



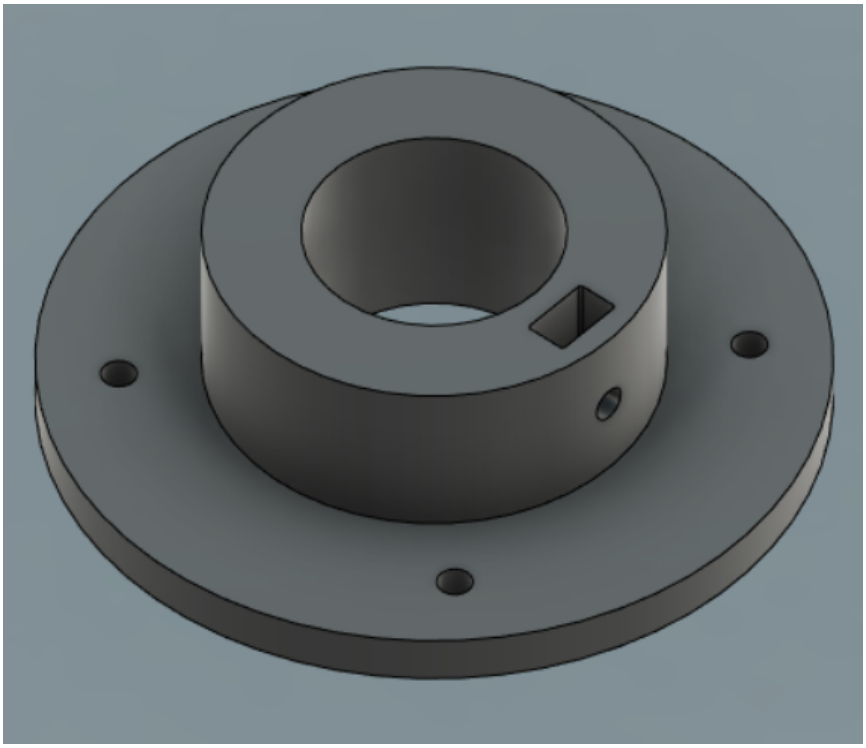
43. Select all the edges created by the extrusion as shown in the picture above and input **TightMarginParameter** as a fillet value.
Complete the operation by clicking **Ok**.



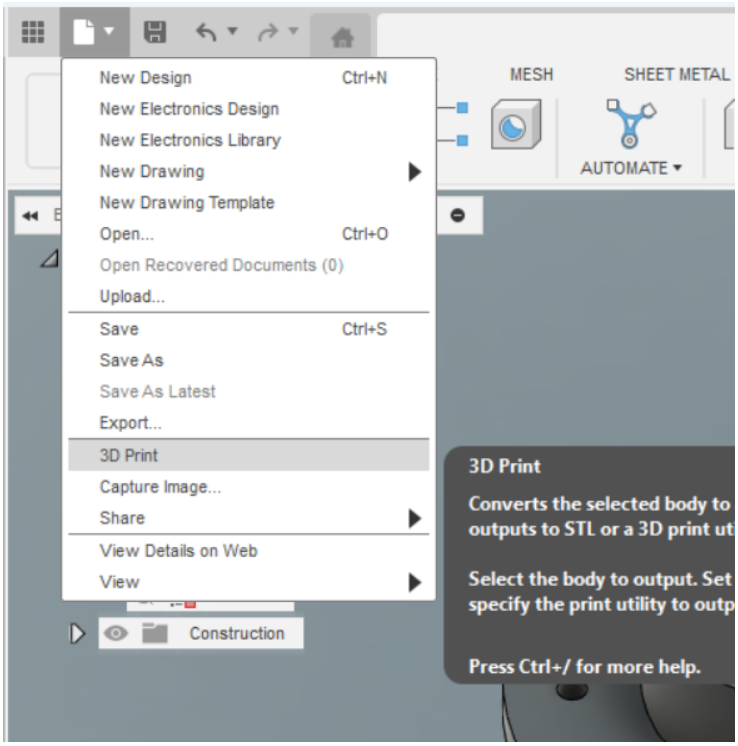
44. Now that our design is complete, we realize after all the efforts and time we put in it that there are things that could be improved.
    Considering the torque this part will be subject to, the screw holes should be further away from the rotation point. The nut slot also creates some thin walls.
    This is the perfect occasion to witness how powerful a tool Fusion 360 is and to test if our design is robust.



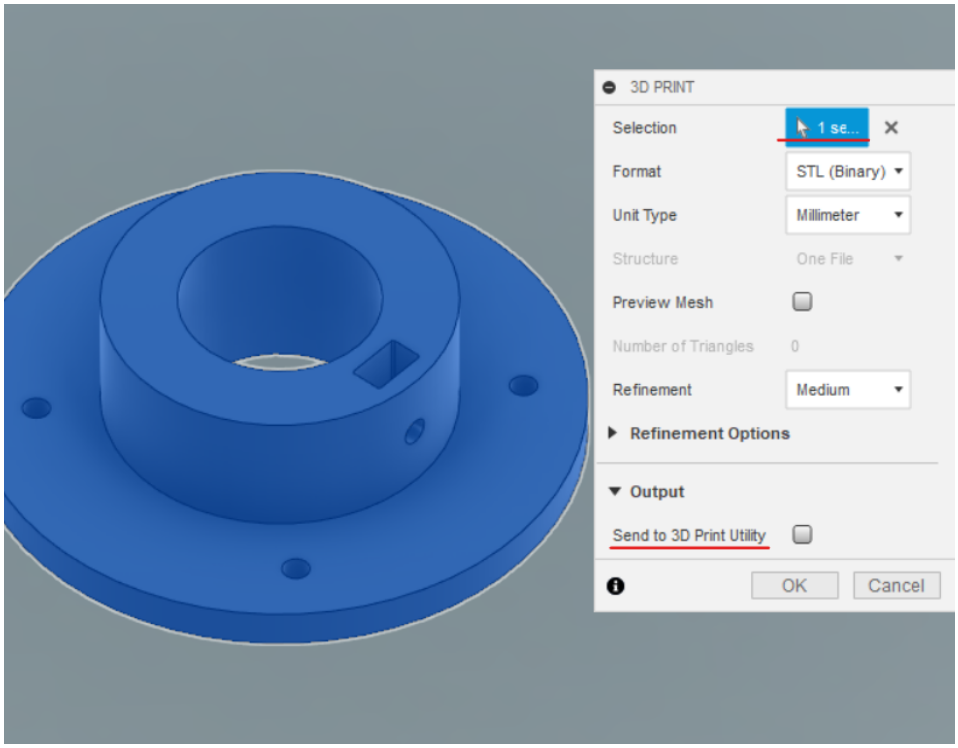
45. To resolve the issues we previously mentioned, we edit the **Parameters** we created as shown in the picture. (Only those underlined were changed to the new values you can see.)



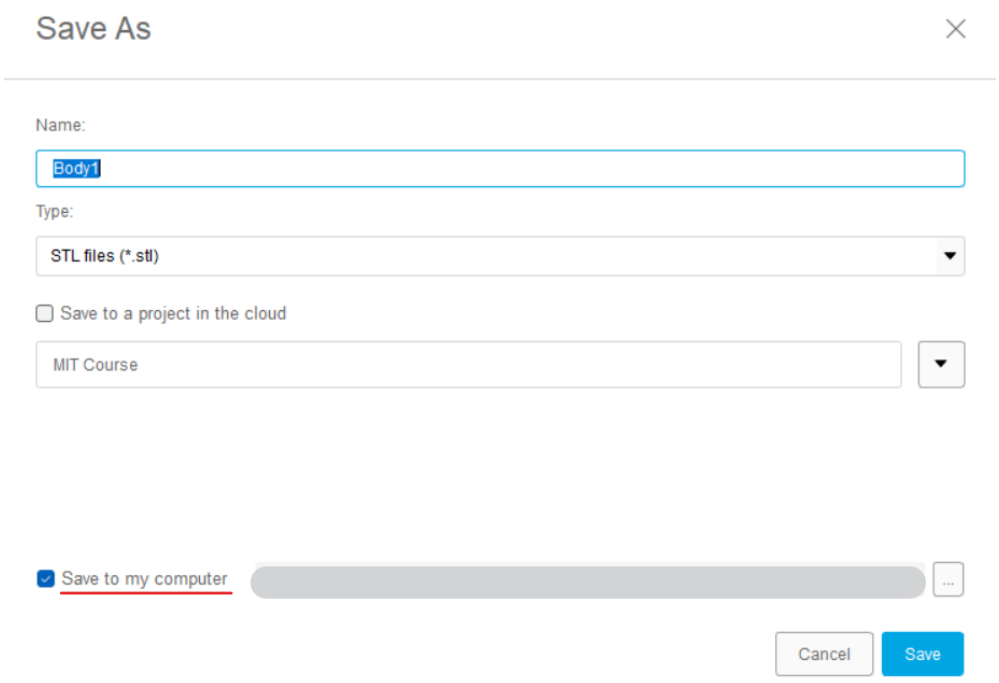
46. When we edit the parameters, we can see all our design changing and adapting perfectly to our specifications.
    Hence, something that could be tedious such as changing the inner diameter because we changed the axle size in our project, or changing the screw holes because the screws we planned to use at first are no longer available, is as simple as changing one value in a field.



47. Finally, in case you want to 3D print this piece or any other, you can do so by going to **File (icon)** > **3D Print**.



48. Select your design as the **Selection** input and un-check **Send to 3D Print Utility** (As we want to save our STL file). Click **Ok**.



49. Finally, check the **Save to my computer** option and save your STL file. You will be able to import this file into PrusaSlicer or any other Slicer to 3D print it.

50.2 Project Architecture: Alone or for a Team

Fusion 360 offers a versioning tool which allows you to work together with your team on different parts of a design. While you do not have the possibility to edit the same file at the same time, you can however import all parts of your design (each in a different file) in an additional "assembly" file. Each time one of the part is modified by a team member and saved, you will be able to update your assembly with its last version.
A Possible architecture for your team Fusion 360 project directory could be:

```
Root:
|-- Components/..  (Directory)
|-- Parts/..       (Directory)
|-- Assembly       (Fusion 360 File)
```

- Components: The imported models of pieces of hardware you use in your design.
- Parts: The parts of your design you will import in your assembly file. This enables each team member to work on one of the "part" file at the same time.
- Assembly: A Fusion 360 which **only** contains imported parts, used to visualize assembly and possibly joint between parts.

Some caveats

One of the trouble you will have if you choose to work with multiple files is keeping parameters synchronized.Parameters will not be imported along with the part your bring into your assembly by default. This makes it difficult to keep an harmonized set of parameters between the different parts of your design.

There is however a way to deal around this issue that you are invited to discover in the following video:

https://www.youtube.com/watch?v=VsqRV7JvBKc
Product Design Online
"Global Parameters in Fusion 360 | Explained in 5 minutes"

Another drawback of this architecture is the inability to directly edit your assembly file. Indeed, the final assembly file in which you import all the parts of your design your team and you created should solely be used for joining your design together. No additional sketches/volumes should be created in this file. Doing the latter will most likely mean you will have to do a ton of work fixing your assembly if you update one of the part's version, because most references to the modified object will have been destroyed/corrupted.

Organizing Objects in the Hierarchy

Another element to organize in Fusion 360 is the hierarchy in Fusion 360 files. Parts of your design that move relative to one another or that are assembled and fixed into place should be **Bodies** grouped under units called **Components**. You can later assemble components with the **Assemble** > **As-Built Joint** (see the section on joints for more details) tool which enables the components to move in specified ways or be fixed (using the **Rigid** joint) relative to another component.

Components in Fusion 360

In Fusion 360, bodies and components represent two different granularity at which you can work.

Bodies are the smallest granularity, it enables you to create basic to more advanced volumes.

Components offer a way to aggregate and organize these bodies, as well as modify them all at once (resizing, moving, ...). They have their own origin and history, which you can isolate by clicking on the radio button appearing on the right of the component while hovering its name in the hierarchy.

Here are some basic guidelines on the creation of components and bodies and when one should be preferred over the other:

- **Separate components for different parts:** If you have different parts in your design, like a wheel on a car, each part should be represented as a different component. This enables you to isolate each part to work on them separately.

- **Single component for simple shapes:** Simple shapes should be kept in a single body or under the same component.

- **Separate components for reusable objects:** If a part of your design should be reused, like wheels in a car, then this part should be kept in a separate component, which modification you can propagate to all its duplicate easily.

- **Single component for small details:** Sometimes, small details can be kept under the same component if having no movement relative to each other.

50.3 History feature in Fusion 360

Why should you care ?

The History feature of Fusion 360 is a list of all the steps which resulted in your current design. Each time an object is created, a feature added or a component moved, a new element will be added to this list.

Note that "sketches" are added as one aggregated element once completed, and not as different ones for each existing shape, line or constraints.

It is useful to think of the history as a **stack**: It is very easy to remove (pop) the last element from the history because it has no other features depending on it. However, when you pile up multiple features and want to remove one in the middle, every references to it (located in the elements above it in the stack) will be destroyed.

This leads to two different scenarios:

1. The design looks fine after the removal. However, because of the destroyed reference, the relations between features will not be preserved in case of a change. Moving the design/Re-scaling it/Changing the value of a parameter will most likely break your design.

2. The design is broken, some components are not where they should be, or strange artifacts are left floating in the air.

How to use the History

On a double-click on one of the elements in the timeline, Fusion 360 will rollback your design in time. You can think of it as hiding all steps above this elements in the stack.

This should be used before removing an element of the timeline to understand the context in which it has been created and the way its reference could be replaced once it is removed.

Before removing this element, you will want to create, in most cases, a "placeholder" object which will resolve all dependency issues once the object has been removed.

Once this placeholder object is created and the initial element removed, all features with lost references will be highlighted in yellow with a timeline and a right-click on them will prompt you to resolve the conflict, which you will then be able to do using your placeholder object.

Note that the history can be isolated for each component in Fusion 360. For this reason, you should try to plan out the components you will have to create in your design and start each first sketch in the correct component to isolate the entire history of the component from the beginning. This will greatly de-complexify the relations between objects in the history, making it easier to modify it later.

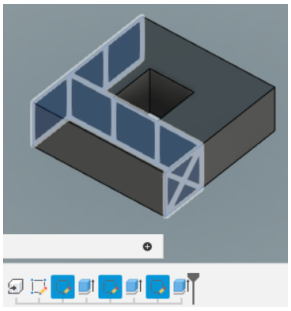
While doing so, don't forget to select the component you are working and verify you are not directly modifying your design through the top level component of the hierarchy (this can be seen with the "checked" on the right of the component's name in the hierarchy).

Keeping the History clean

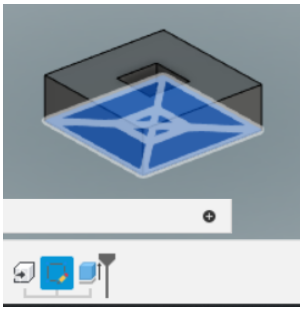
A good rule of thumb to avoid headaches around dependencies in the history is to try to make it into the most simple, minimalist recipe possible on how to create your component:

1. All unnecessary steps should be removed from the history
2. All redundant steps should be merged into one use of a feature when possible.

The example provided in Figure 50.1 makes the problem obvious and yet it often occurs as we progress through our design, since it evolves as



(a) This design includes a lot of redundant steps in the history



(b) Cleaning up the history results in the same design with only one extrusion and sketch.

Figure 50.1: Cleaning up the history

we go and wasn't entirely planned from the beginning. It can often happen that unnecessary or redundant steps are created as we don't find the right design on the first try and start to reiterate on top of what we built.

50.4 Sketching

Locked vs Unlocked Sketches

Sketches in Fusion 360 can be either locked (Displayed as black with a small "Lock" icon next to the sketch's name) or unlocked (Some part of the sketch is displayed blue). A locked sketch means it is fully constrained and no dimension is left unspecified.

It is important to try your best to have only locked sketches as unlocked sketches are very fragile and can be easily broken by any change. This is even more important when modifying sketches far away in the timeline: A broken sketch would result in a chain reaction corrupting your design.

Using existing components in your design

Often times, you will have to design a part which interacts with either other parts or existing components you already have or plan on having.

In the latter case, you can do several things to make your life easier:

- Find the 3D design of the component online and import it into your Fusion 360 file. This way, you can visualize in advance the position of your component and the way it fits into your design. You can also extract dimensions from the component's design (A useful feature to do that is to press "P" (for project), once in sketch mode, on an object/shape you want to project onto your sketch).
- Measure the component's dimension with a caliper as precisely as possible. When measuring an inner diameter, use the maximal measure you read on the caliper. When measuring an outer diameter, use the minimal measure you read.
- Think of the way the component will be inserted into your design: it must have a way to be placed in its place and to be fixed (A screwdriver must finds its way to screws).

50.4.1 Parametric Design

General Rule: Fusion 360 is a tool which rewards precise and clean design and severely punishes approximations.

Having all measurements defined as parameters and good organization of both bodies, components and feature references will result in very robust and dynamic designs.

On the contrary, forgetting constraints where they should exist, inputting "magic numbers" as distance measures where a reference to an object would be better (In an extrusion for example), will result in a fragile design.

Parametric design in Fusion 360 enables you to edit specific dimension in your file and have the rest of your design gracefully adapt to the new dimensions. It is useful when you realize the scale isn't right, some part is too small, etc...

To create and use parameters, you can use - either in sketch or volume mode - the "Modify > Parameters > Create New" menu. Once you created some parameter for your file, you can use it in any input field of the corresponding unit along with a mathematical expression involving it: From measurements in sketch mode to extrusion length in volume mode or even number of repetitions in patterns.

Since each parameter has a predefined unit, some operations may not be allowed: Dividing a length (mm) by another length (mm) will result in

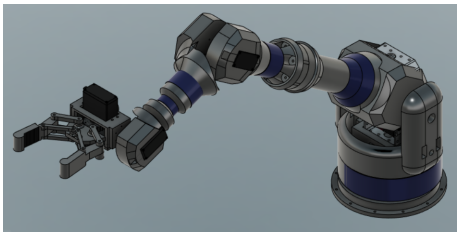
a unit-less expression which may not be allowed in an input field which expects a length.

In case you use Parametric design, it is even more important to have Locked sketches so that your design doesn't get destroyed when you change the parameter's value.

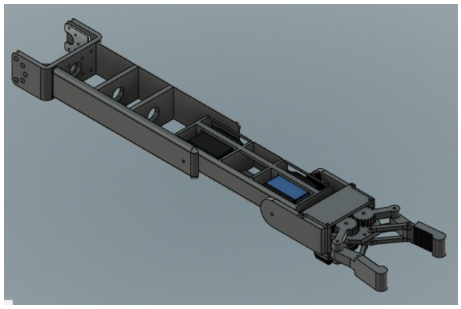
50.5 Best Practices

Minimalist Design

Plastic is way more robust than one can think at first, and creating box-shaped designs with an extensive amount of filling only wastes materials and makes the design heavier. As a general rule, designing minimalist-looking parts, like a skeleton of the shape you first envisioned, will lead to a smaller printing time, minimal material use and lighter design. Think first about the functions you want your design to accomplish. The design will come afterwards.



(a) A design for a robotic arm which took time to create and ended up being useless because of its weight.



(b) A design for a robotic arm which was created in a day and worked flawlessly.

Figure 50.2: Example illustrating "minimalist designs"

For the same reason, when creating big parts, it is best to design them in such a way they can be assembled (screwed into place, glued, ...) after they are printed, instead of using an extensive amount of support material to print parts with complex geometry.

50.5.1 Mesh vs. Solid-based modeling

If you ever used Blender, ZBrush or similar software before, you are already familiar with mesh-based modeling. Points in space called "vertices" form faces which aggregate into a surface or volume.

Fusion 360 uses a different type of modeling based on solids. By creating only primitive shapes and assembling them into something more complex, we get a model which is more precise, and easier to modify in a controlled manner.

For this reason, there are operations in Fusion 360 which are more efficient and useful than they would be in a mesh-based modeling software.

- **Boolean operations:** Boolean operations between bodies are very efficient in solid-based modeling. They produce a more accurate representation than they would in a mesh-based modeling software.
- **Parametric modeling:** This feature enables the design to be easily and accurately modified, and ensures a consistent result. This is crucial in CAD applications where precision and an ability to quickly iterate on your design matter.
- **Analysis:** Unlike mesh-based modeling, solid-based modeling offers powerful analysis capabilities. The section analysis tool is one of them.
- **Simulation:** A more advanced type of analysis are simulations. Simulations in solid-based modeling are much more effective since they don't have as much computation to do as in a configuration with numerous vertices.

50.5.2 Useful Fusion 360 tools

- **Solid** > **Inspect** > **Section Analysis:** Once a face is selected, displays a section of the object along the face. This enables your to see hidden parts of your design or remove ones which are in the way. Example here
- **Solid** > **Inspect** > **Measure:** Displays the measurements of an object (radius if it is a circle, length if a line segment, etc...) or the distance between two objects. If two points are selected, it can display both the

distance between them in a straight line and the distances projected on the x,y,z axes.

- **Solid** > **Construct** > **Plane:** Builds a plane from different geometry references. Useful if there is no clear way of selecting your sketching plan on existing geometry.
- **Solid/Sketch** > **Create** > **Pattern** > **Circular:** Reproduces a sketched object/geometric feature/body/component following a circular pattern. It avoids circumvents a considerable amount of work when you are working on a cylindrical geometry.
- **Solid** > **Modify** > **Split Body:** Cuts bodies along a specified plan. Can be used to split parts which move relative to one another or for other purposes.
- **Solid** > **Modify** > **Combine:** Creates a new body from a union of multiple bodies. Especially useful when trying to recombine bodies you cut with the split tool

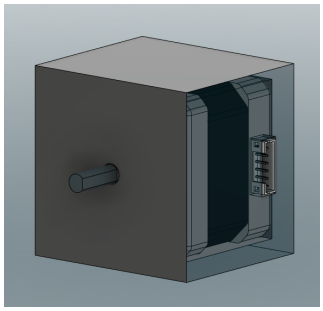
50.5.3 Thinking about Assembly

When designing in Fusion 360, it is important to bear numerous considerations in mind in order to have a successful first assembly.

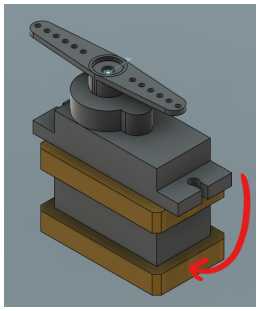
Always keep in mind which parts of your design must be removable or fixed in place. For parts which must be screwed in place, a screwdriver must be able to reach the screws. This can seem obvious when said out loud but it can also be easy to forget while working on your design.

Some electronic parts must also have their wires routed through your design and find their way to the other components they are connected to. This is especially important when importing electronic components' models as the wires are often not visible.

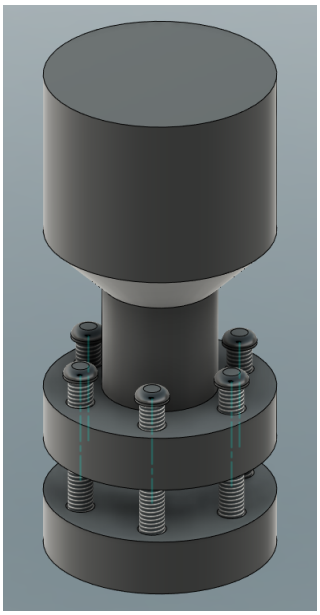
Screw holes must be created before assembling your design. Leaving them to be drilled in a laser-cut part is not a good practice and should only be done as a quick and temporary fix. Drilling holes for screws in 3D printed parts will not work at all because of the infill layers having a much lower filling density.



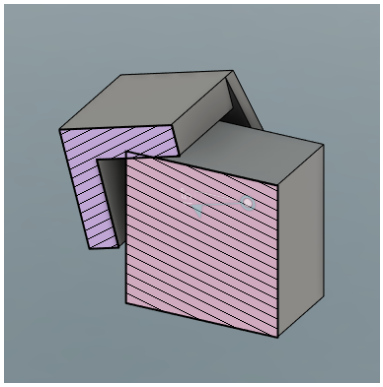
(a) The stepper is completely surrounded by plastic and has big chances of overheating.



(b) Even though it does not appear in the imported model, this servo has cables here, making the design wrong.



(c) If you want to screw these two parts in place, you will not be able to insert a screwdriver here because of the upper part's increasing radius.

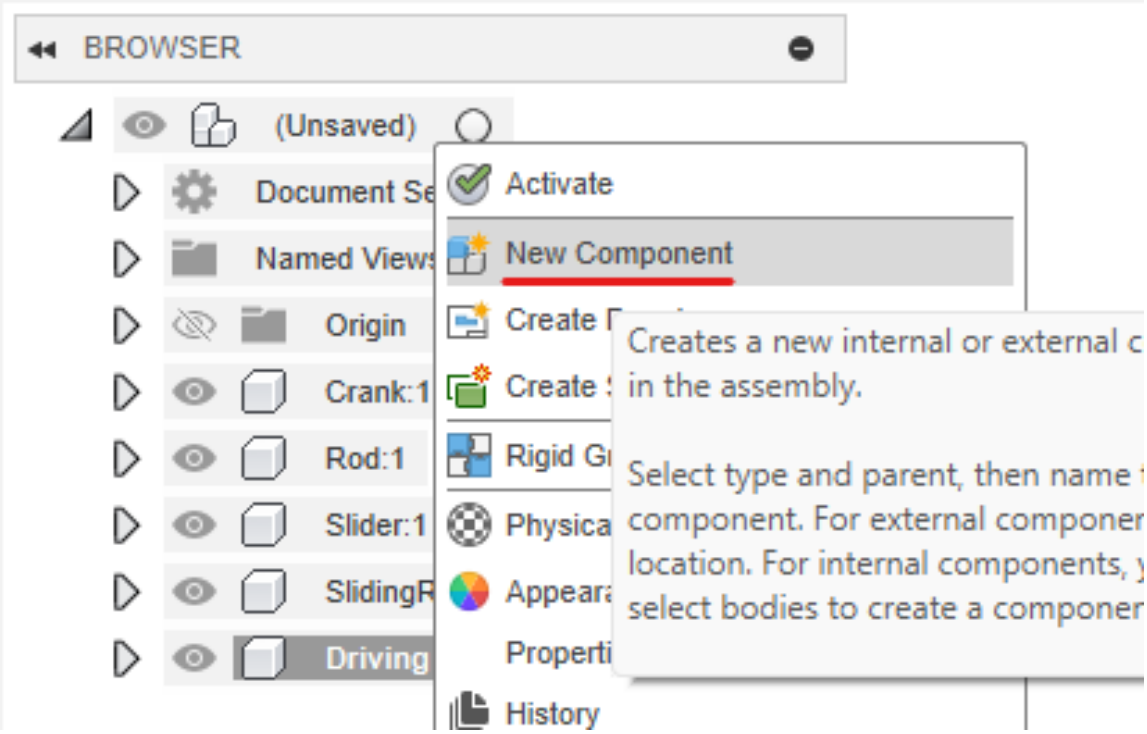


(d) Only a section analysis enables us to see that the joint is wrong and the two components will be stuck at an angle.

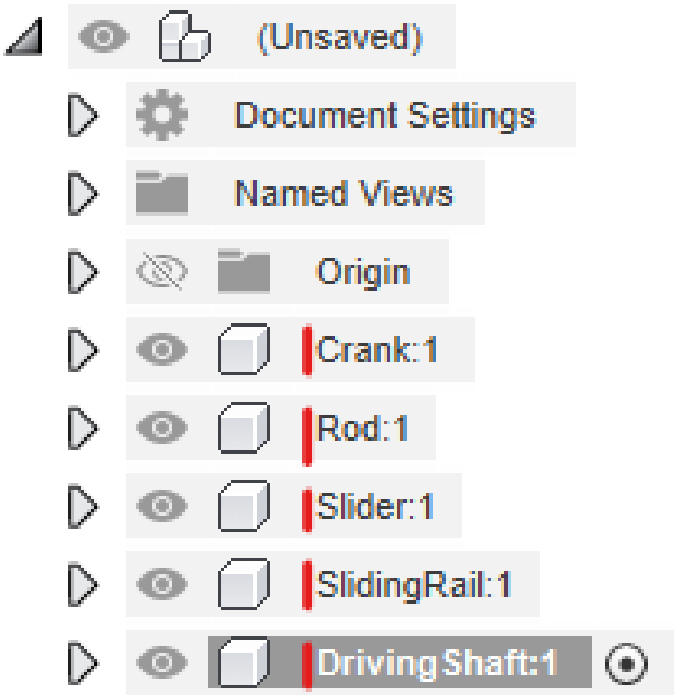
Figure 50.3: Possible design mistakes

50.6 Joints in Fusion 360

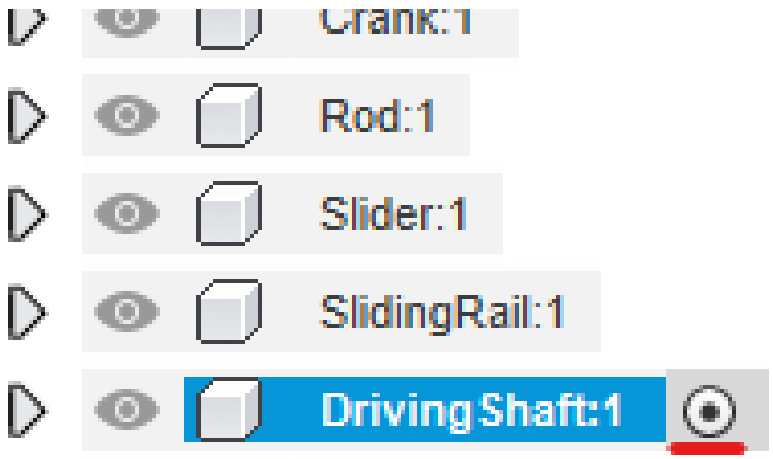
To create interaction between Fusion 360 components, we can use the **Joint** functionality. Here is a tutorial on how to use this functionality to create a mechanical piece which converts rotary movement into a translation. Note that this tutorial builds on top of the knowledge acquired with the getting-started tutorial and it is recommended to complete it first before starting the joints tutorial.



1. Once a new design has been created, head over to the hierarchy part of the window and create all the components we will need (via the **Right-click on root component > New Component** menu). In our case, every components are a part which will move relative to the others.



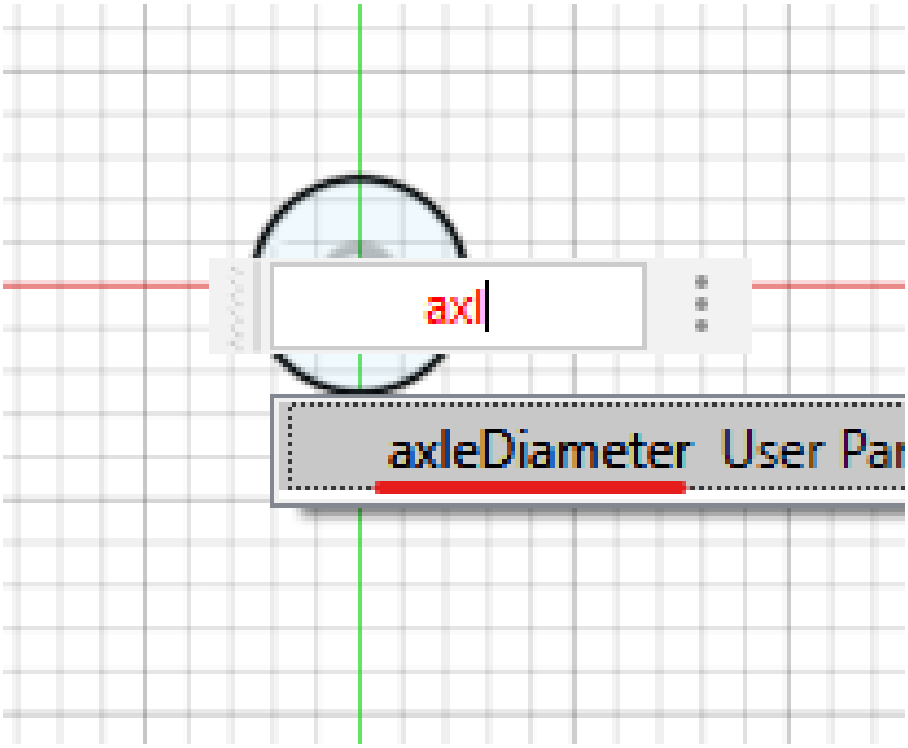
2. Our five components are:
   (a) Driving shaft: The axle which transmits the rotational motion.
   (b) Crank: The first part driven by the shaft.
   (c) Rod: The second part, driven by the crank.
   (d) Slider: A third part which, while linked to the rod by its axle, will be constrained to slide along one of the sliding rail's axes.
   (e) Sliding rail: The rail which constrains the movement of the slider.



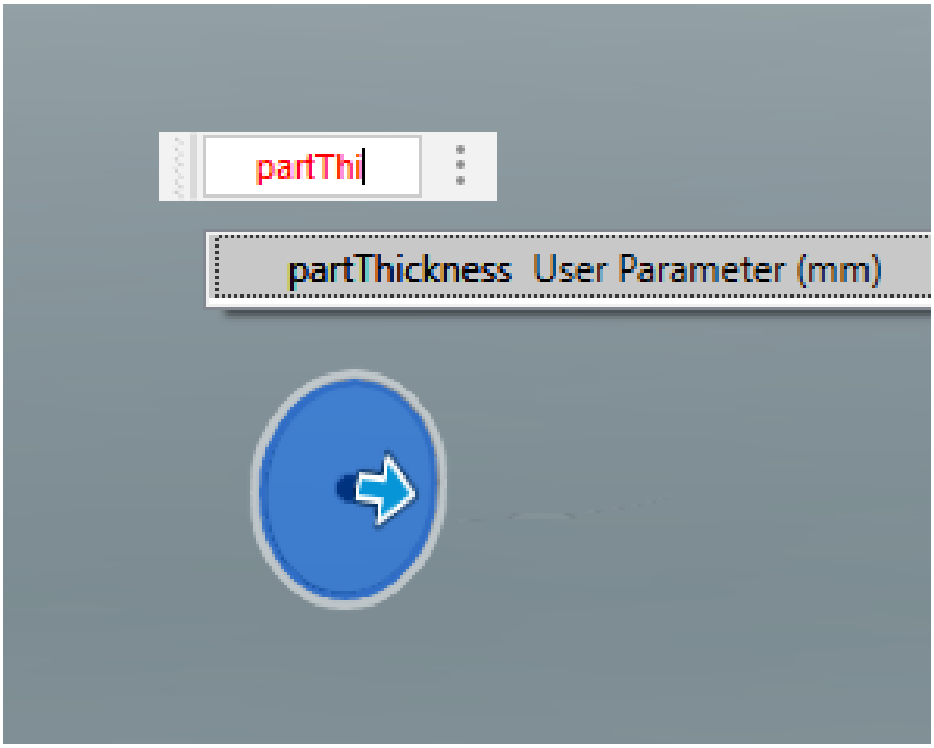
3. To start your design, activate the **DrivingShaft** component by clicking on the radio buttons which appears on hover (You can also use **Right-click on Driving Shaft > Activate Component**). This will isolate the component from the others so that you can work on it exclusively.

Name	Unit	Expression
axleDiameter	mm	5 mm
partDiameter	mm	10 mm
sliderDimension	mm	16 mm
partThickness	mm	20 mm
railLength	mm	160 mm
partLength	mm	50 mm

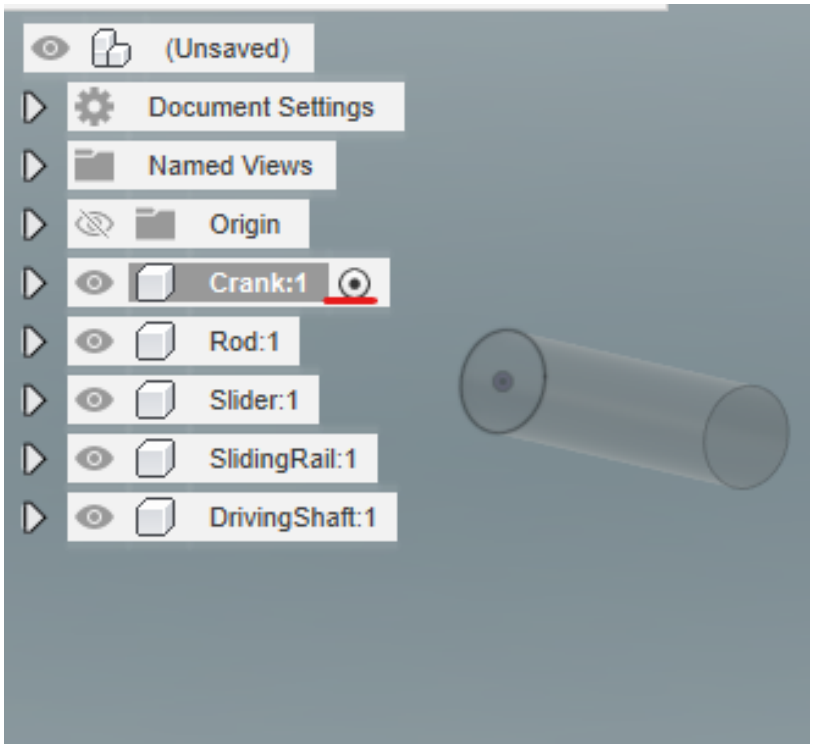
4. Let's create the parameters we will use in our design as shown above.



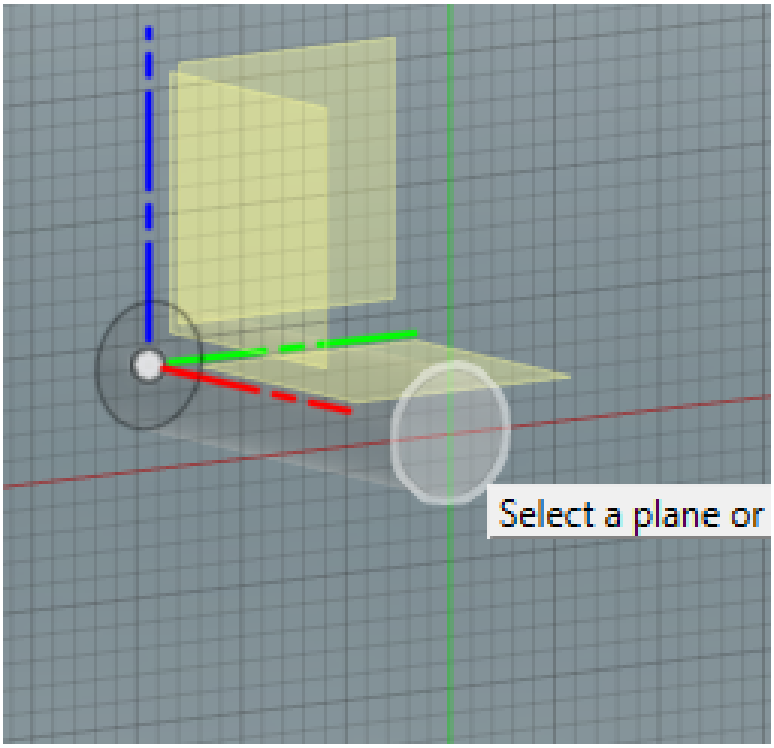
5. Create a new sketch (note that you should still have the **DrivingShaft** activated) and draw a circle of diameter **axleDiameter**.



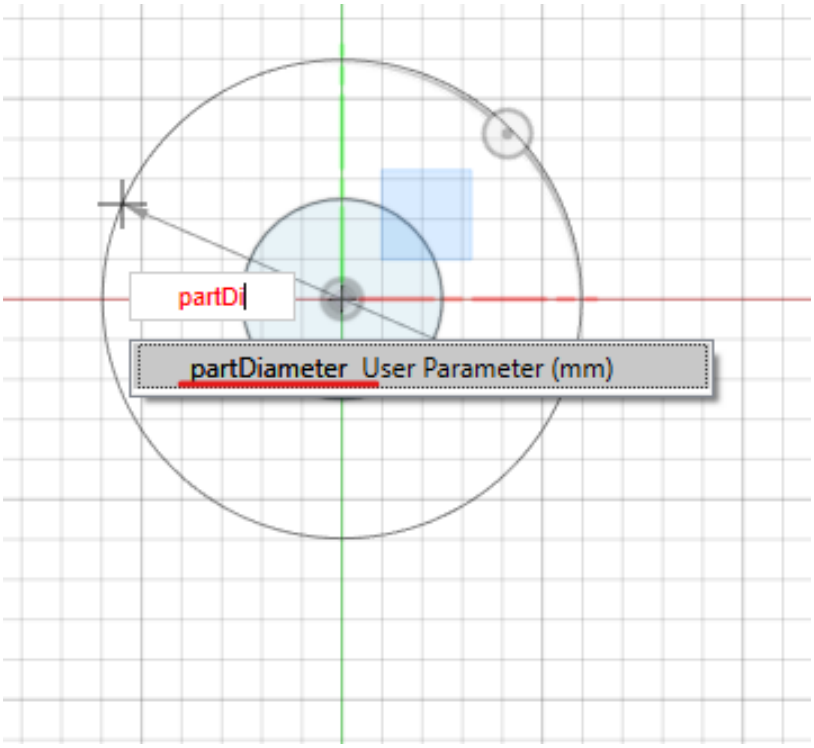
6. Extrude the sketch with a depth value of **partThickness**. This rudementary representation of the driving shaft is complete.



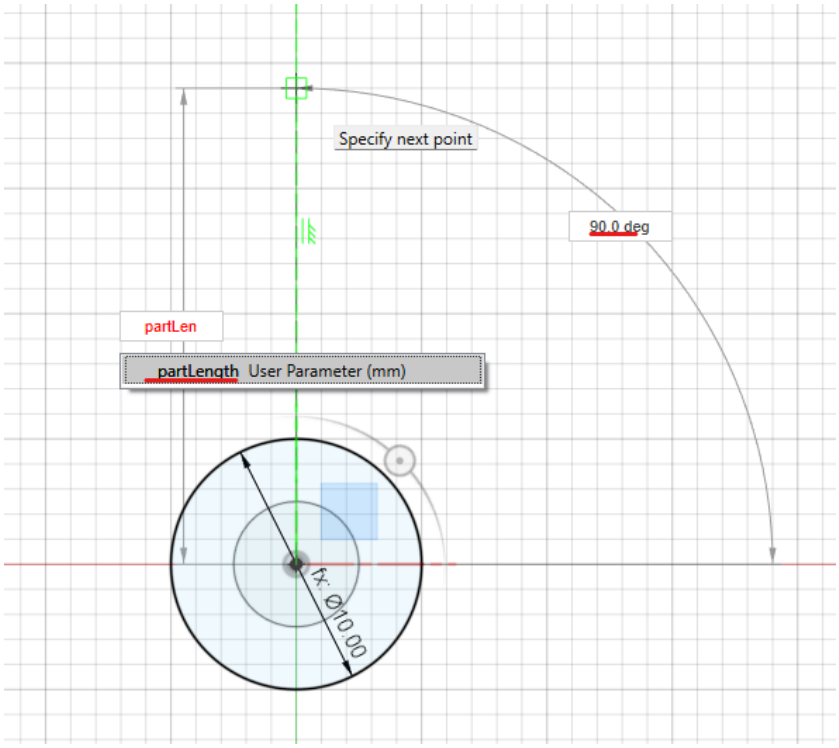
7. Select the crank component and activate it.



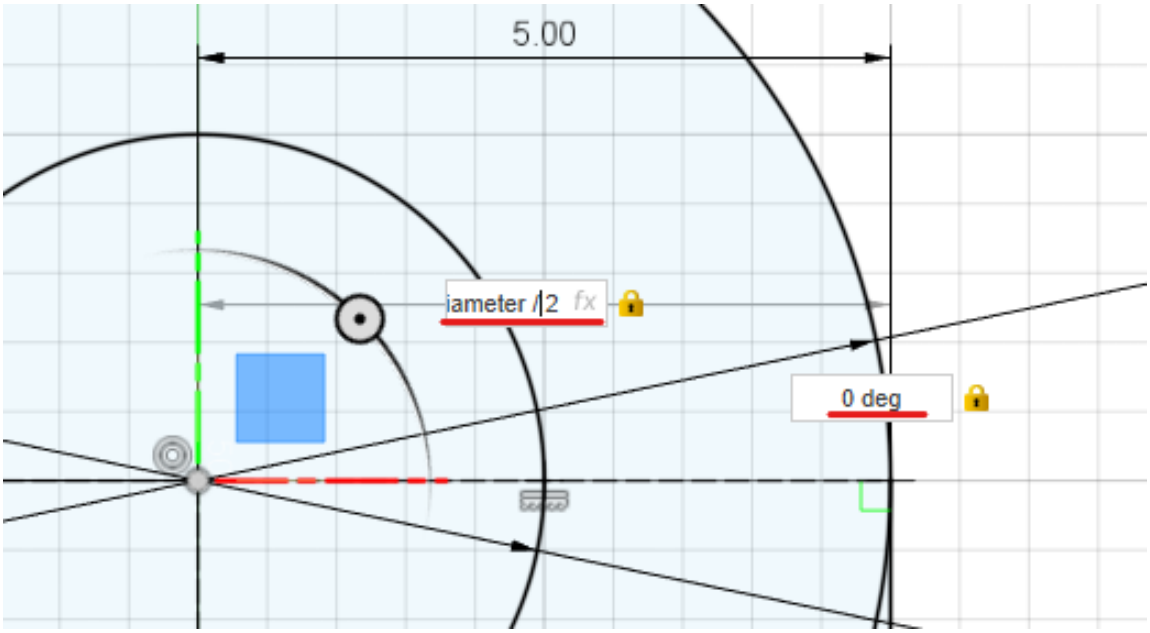
8. Create a sketch on top of the extruded face of the first component as shown above.



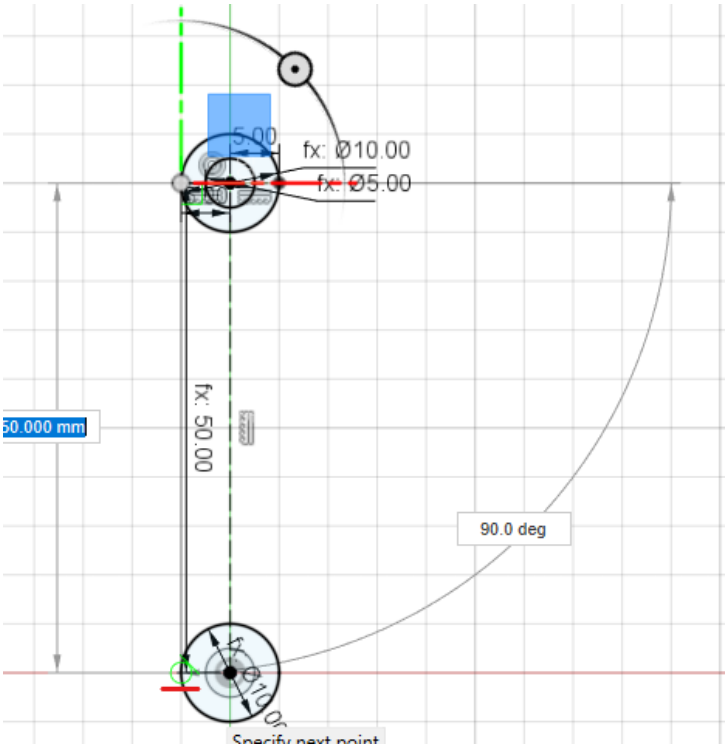
9. On this sketch, draw a new circle, using the center of the referenced axle (Driving shaft). Give it a diameter value of **partDiameter**.



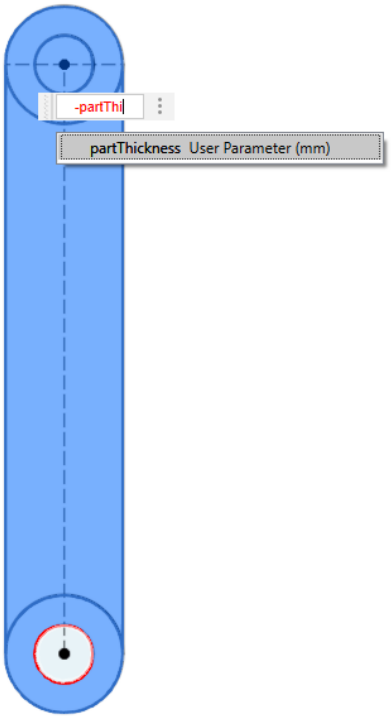
10. With the construction line mode active (shortcut **X** by default), sketch a line from the center of the axle, with an angle of 90 degrees and a length value of **partLength**. Using the new end of the created line segment as the center, draw two circles of diameter value **axleDiameter** and **partDiameter**.



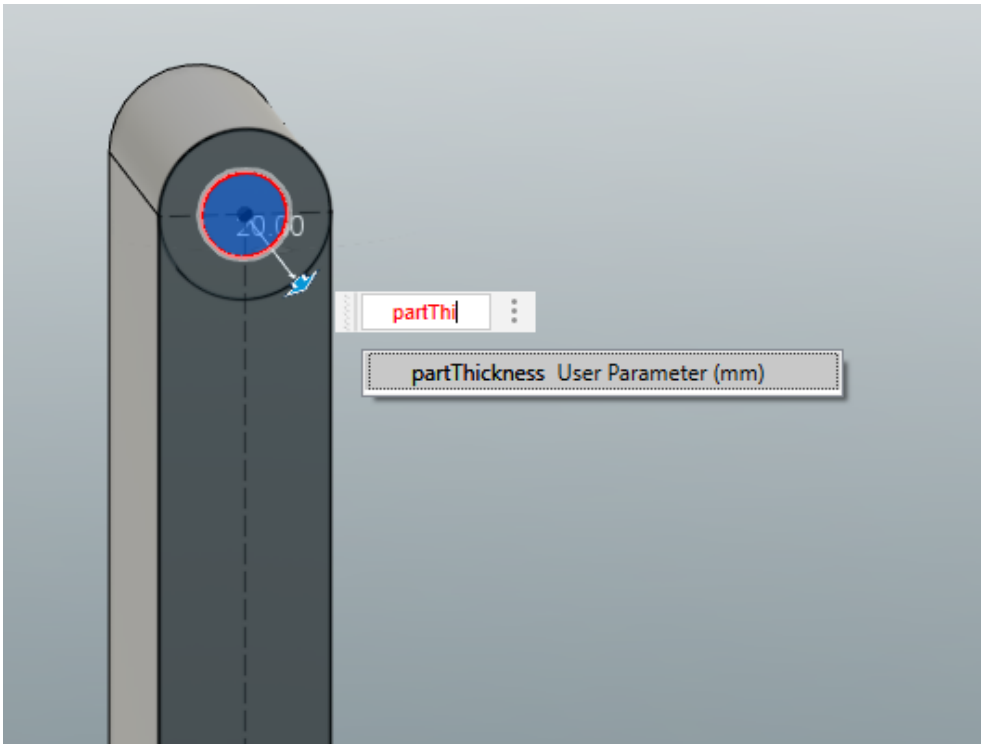
11. From the circle's center, draw two construction lines with a length value of partDiameter/2 and angles of 0 and 180 degrees. This will create the construction points we need.



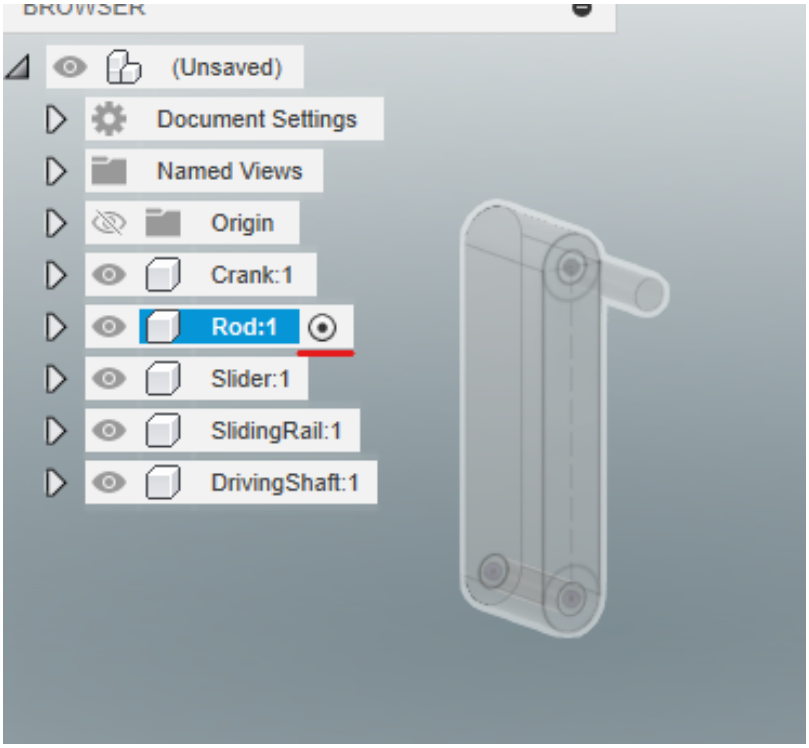
12. Draw two lines, from the points we just created to the tangents of the other circle (as shown above). The tangent is indicated by a green "tangent" (tangent line to a circle) symbol.



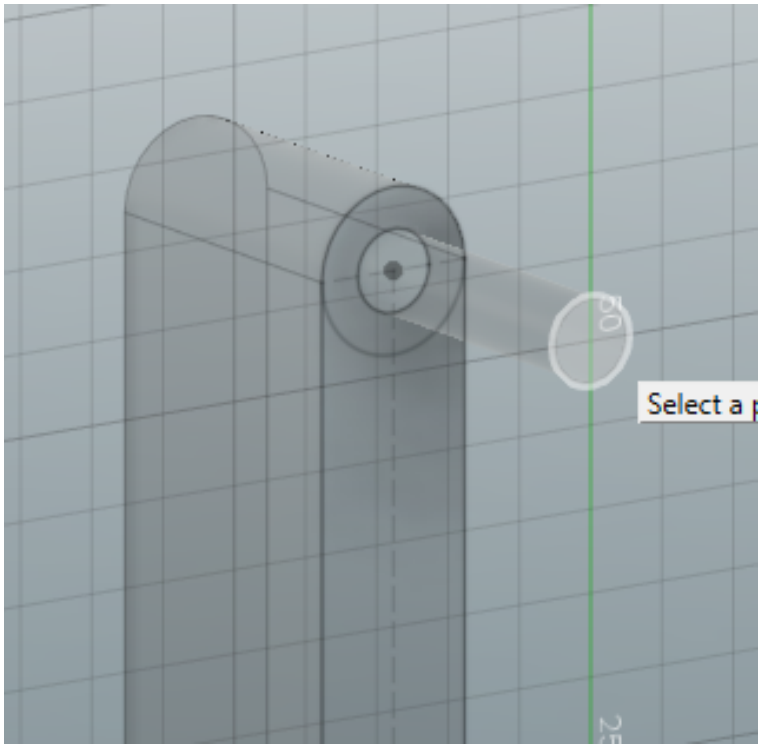
13. Extrude the faces as shown above, while being sure to keep the bottom inner circle unselected. The depth value of the extraction is **partThickness**.



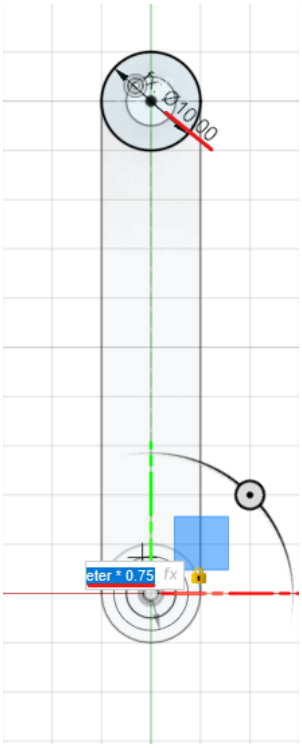
14. On the extruded part, show the last sketch again and extrude the top inner circle by a value of **partThickness**. The crank component is complete.



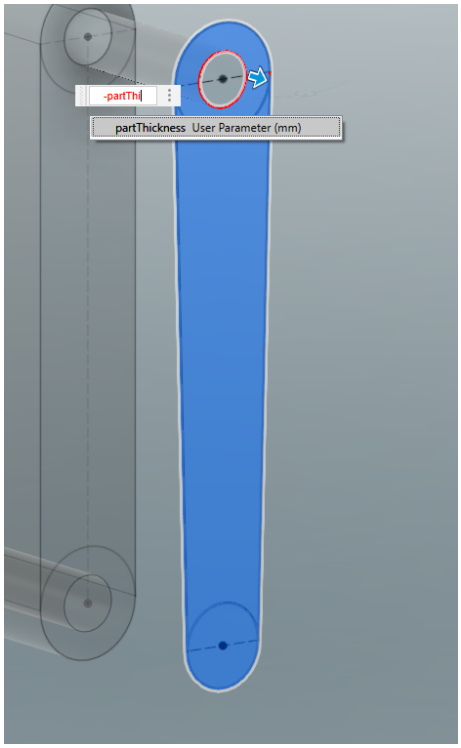
15. Select the **Rod** component and activate it.



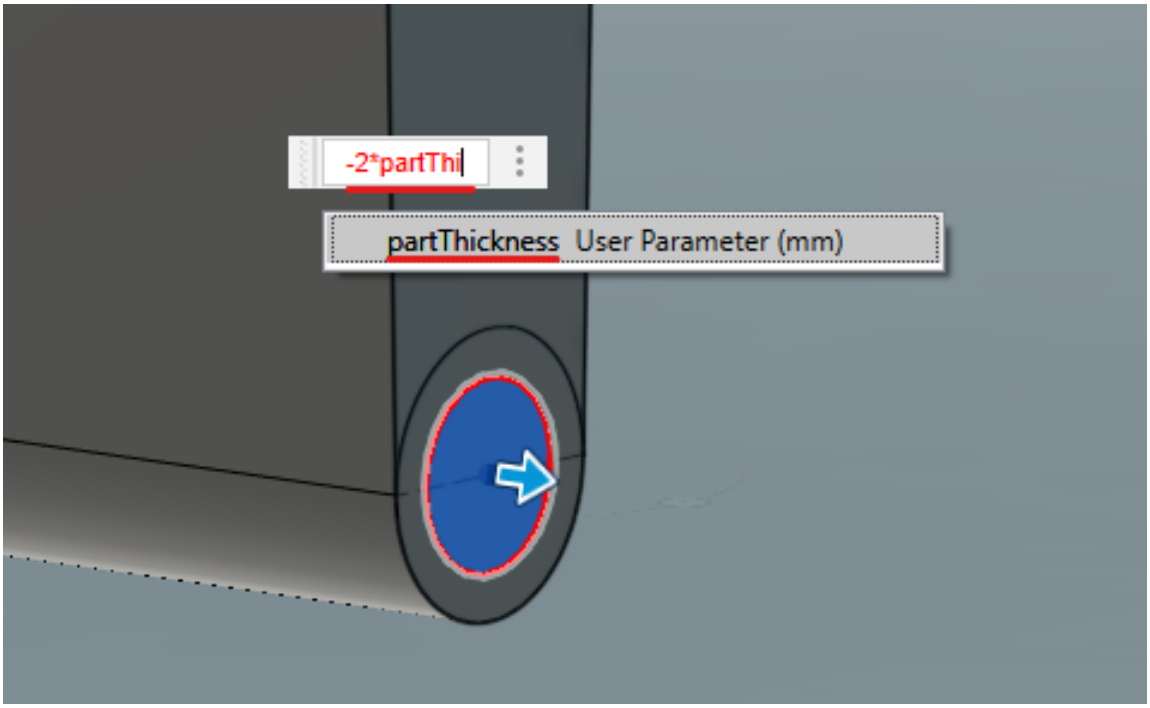
16. Create a new sketch on the face of the crank's extruded axle.



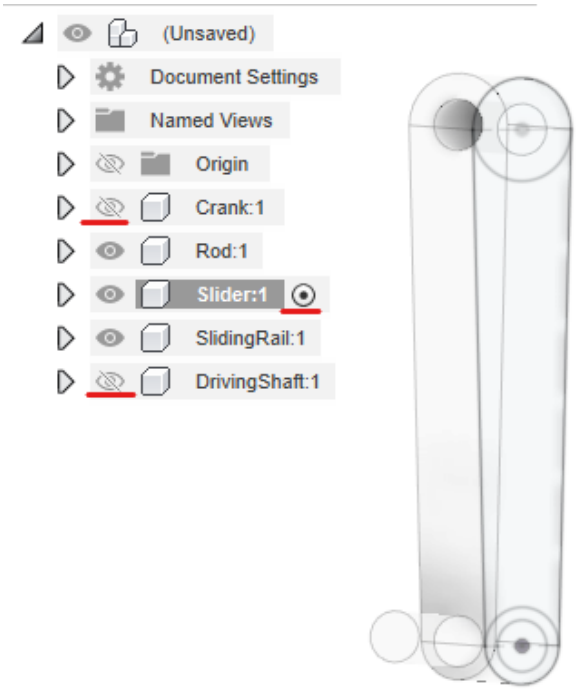
17. Repeat the same operations from the crank to the rod. This time, the bottom outer circle will have a diameter of partDiameter $\cdot$ 0.75. The bottom inner circle keeps the same diameter value of **axleDiameter**. Finish the sketch.



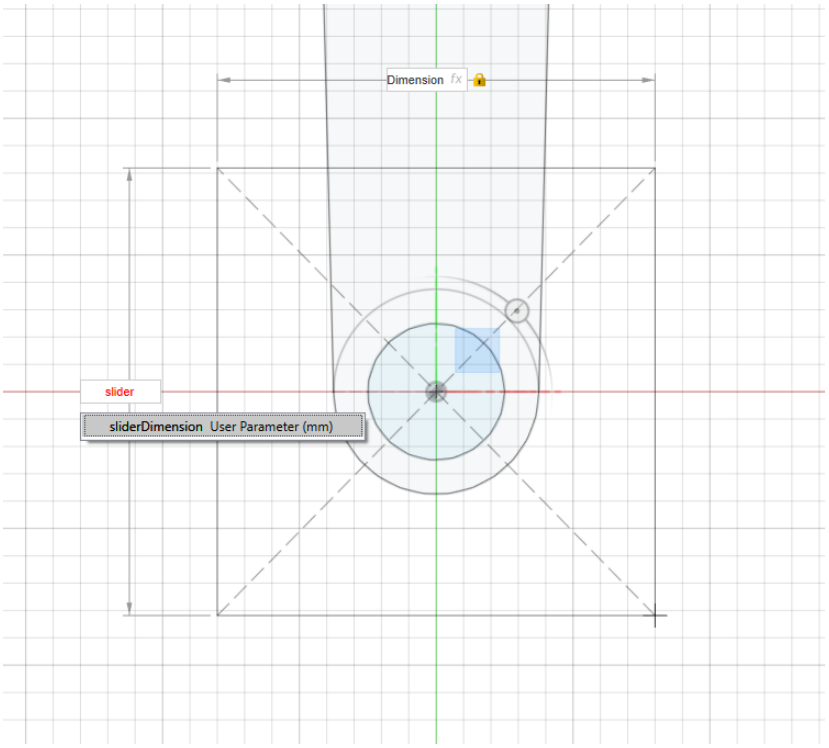
18. Extrude the last sketch, while keeping the top inner circle unselected, with a depth value of −partThickness.



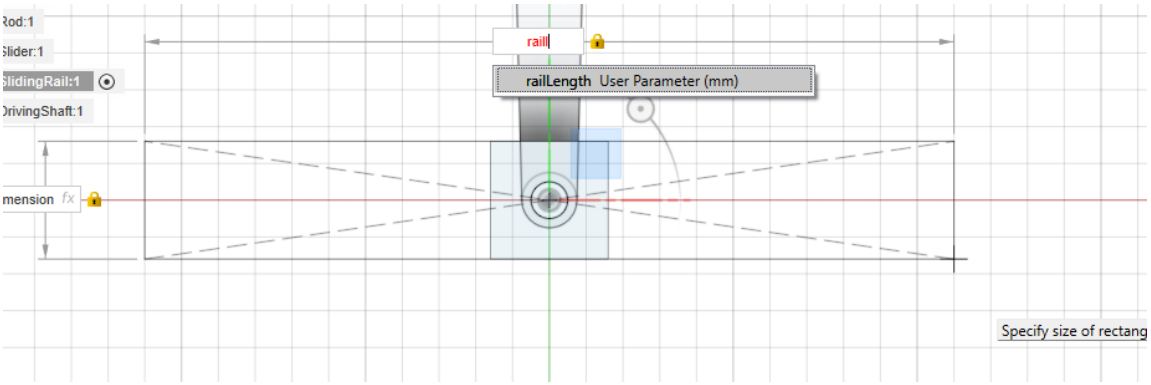
19. Show the last sketch again and extrude the bottom inner circle with a depth value of $-2 \cdot \text{partThickness}$, such that the created axle overlaps with the **Crank** component (which you should now hide).



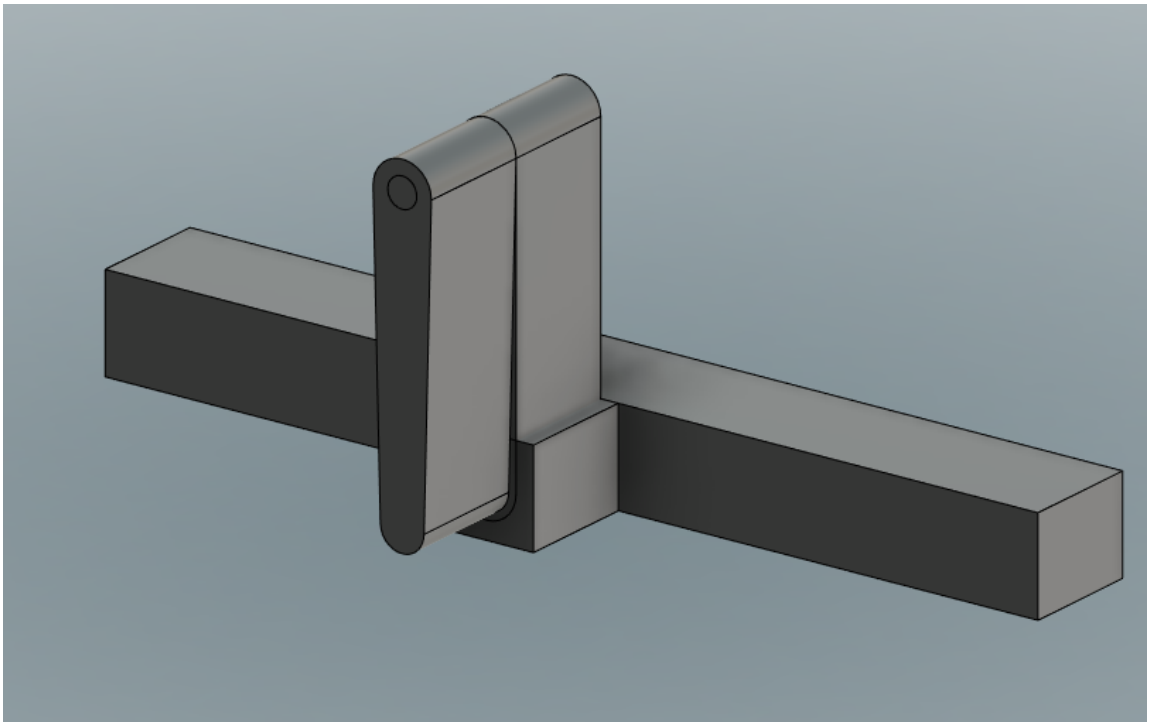
20. With the driving shaft and crank components hidden and slider activated, create a new sketch, on the extruded axle of the rod component.



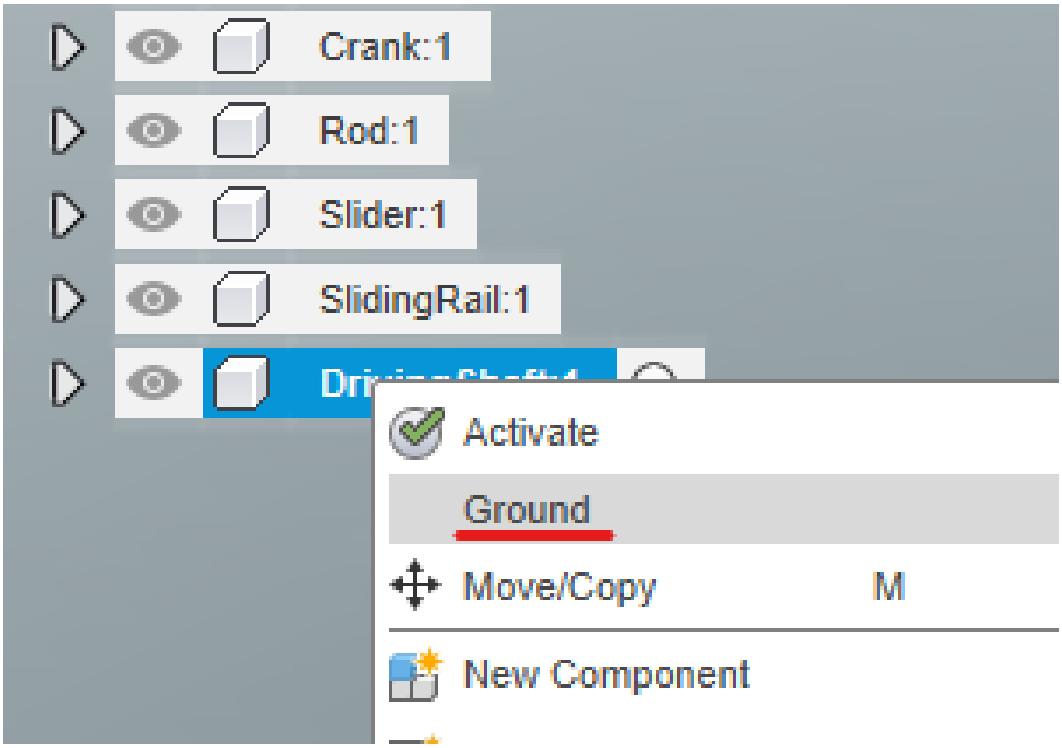
21. Using the **Rectangle from Center** sketching functionality and the center of the axle, create a square with the **sliderDimension** parameter value. Extrude the square (without the axle's face selected) in the direction of the axle, with a depth value of **partThickness**.



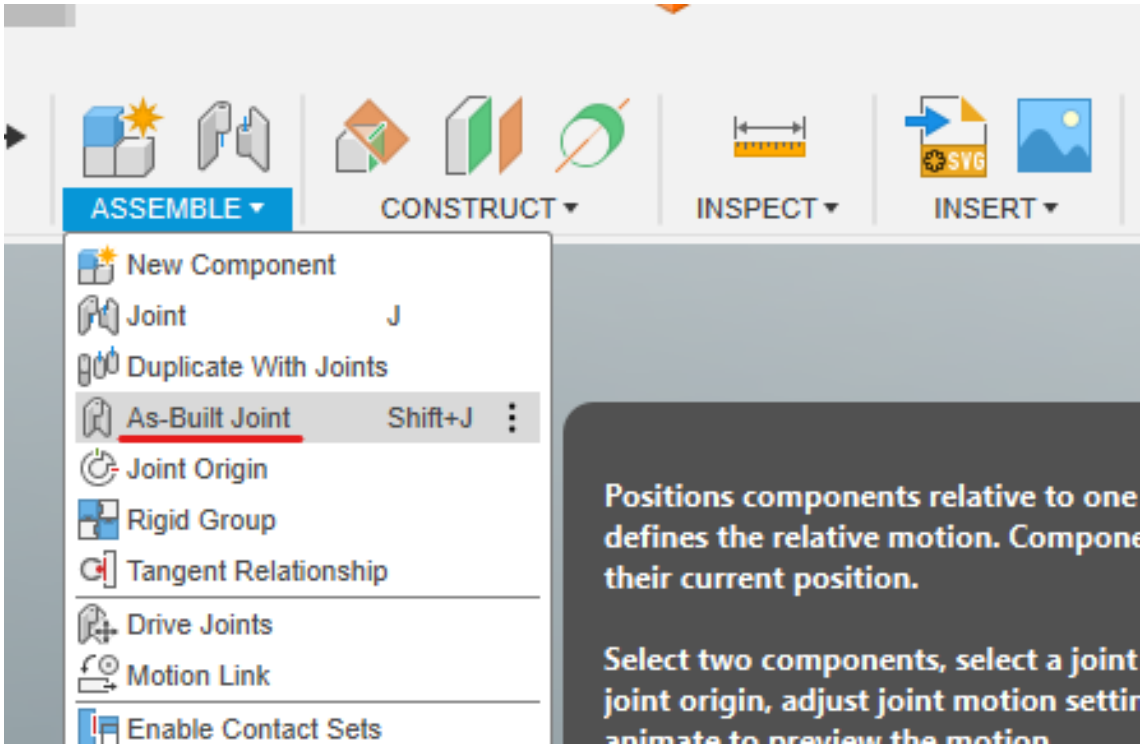
22. Lastly, activate the **SlidingRail** component and, using the slider's face as reference, draw a rectangle of height value **sliderDimension** and length **railLength**. Extrude the whole rectangle, with a depth value of **partThickness** in the opposite direction of the slider.



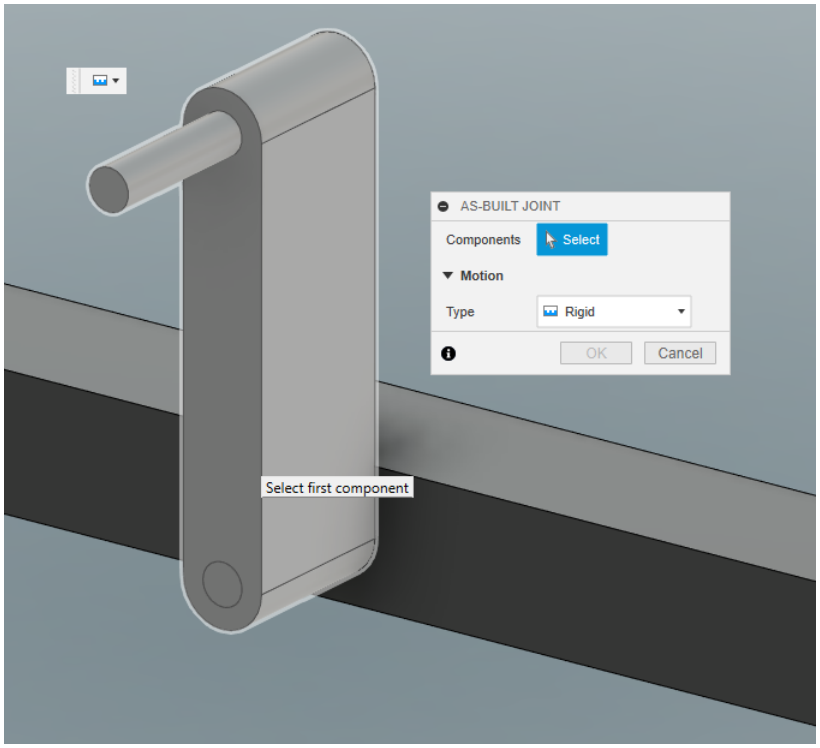
23. Once all previous steps are completed, you should obtain the result shown above.



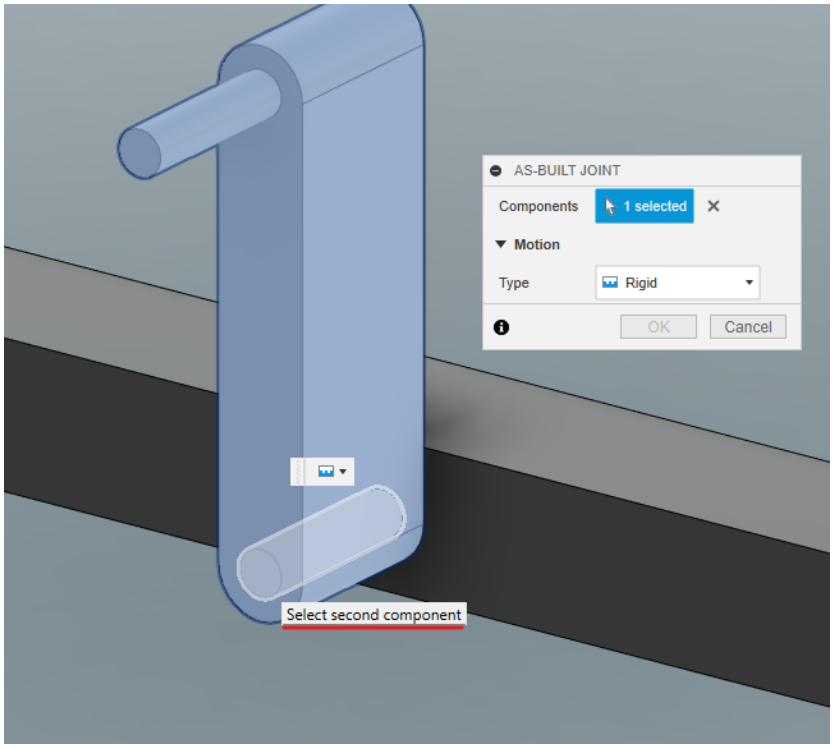
24. To create the joint, we must first select the components which should be grounded. These components will not be able to move at all. Ground the **DrivingShaft** and **SlidingRail** components using the **Right-Click on component > Ground** menu.



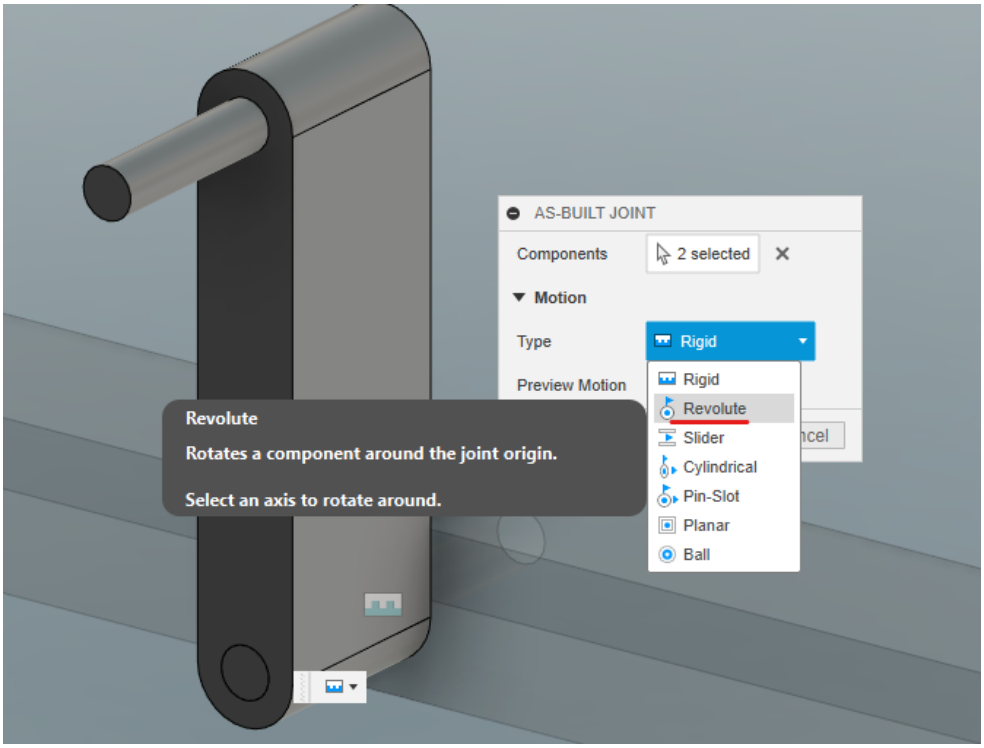
25. Navigate to the **Assemble > As-Built Joint** menu (shortcut **Shift/Cmd+J**). Contrary to the basic **Joint**, the **As-Built Joint** will try to create a joint without moving any of the components, leaving them as they are built.



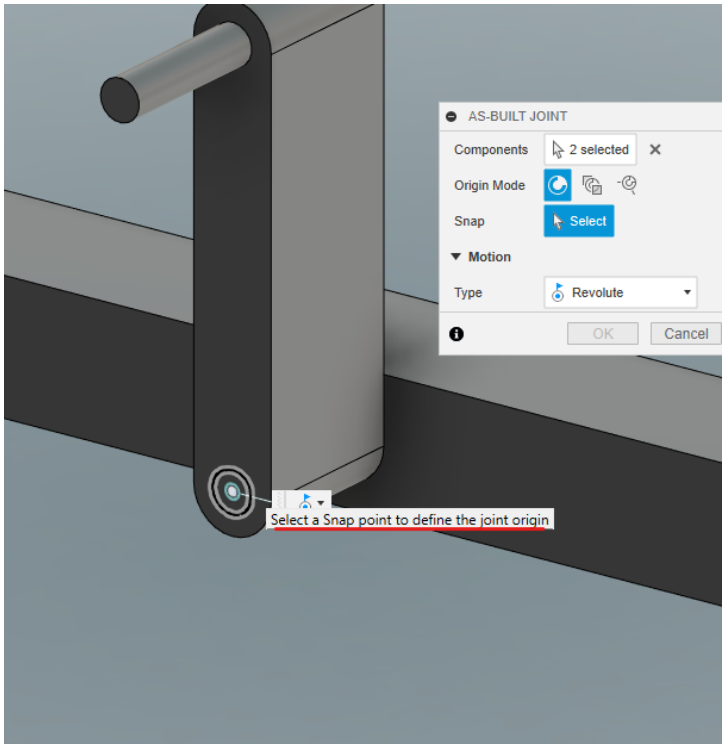
26. The first component selected with this functionality will be the one moving relative to the second component. Select the **Crank** component first.



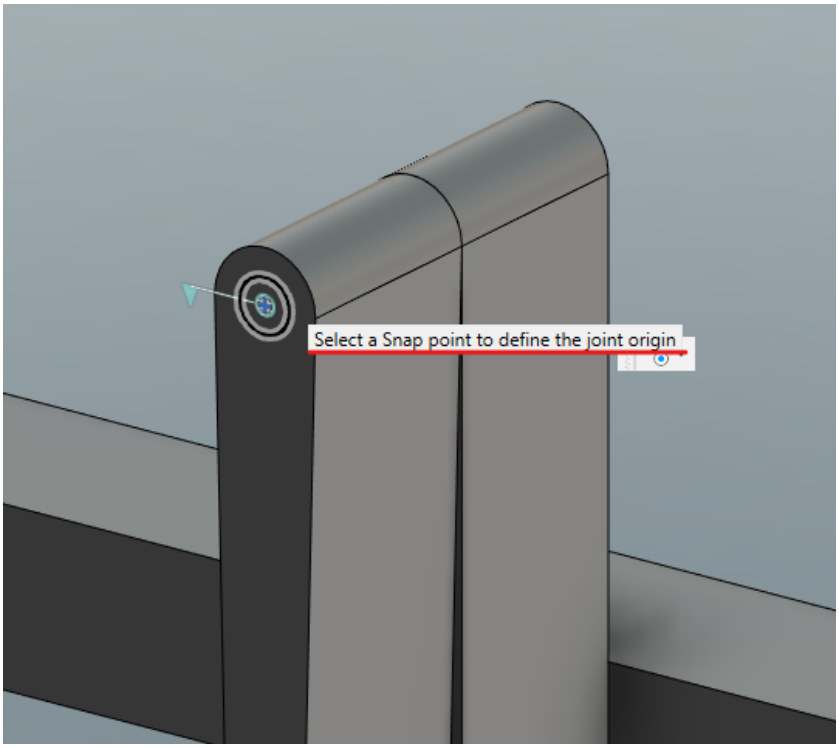
27. Select the **DrivingShaft** component second.



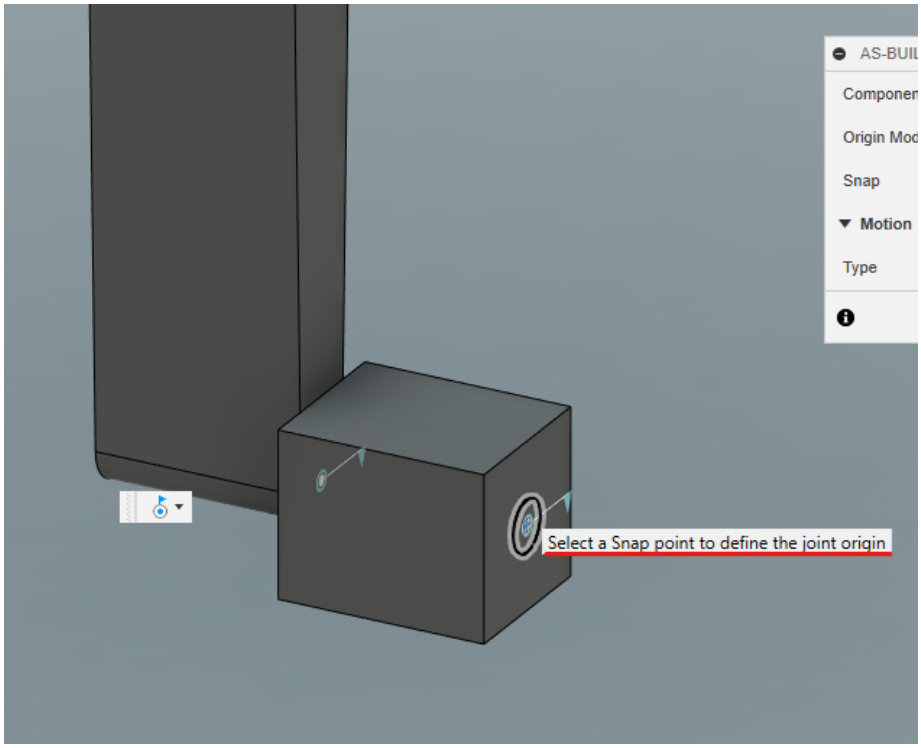
28. For the joint type, select **Revolute**, since this is the type of movement we want between the crank and the driving shaft.



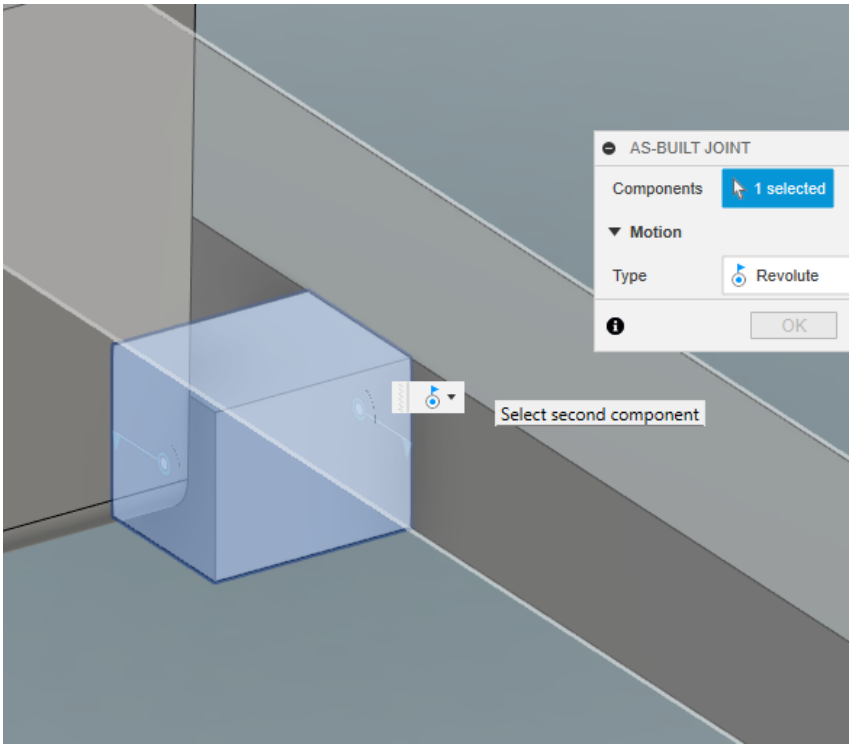
29. The last step is choosing the joint's origin. For our revolute joint, we must choose a point on the driving shaft's axis, which we can do by clicking on its face. Since it is a disk, Fusion 360 will identify its center as the joint's origin.



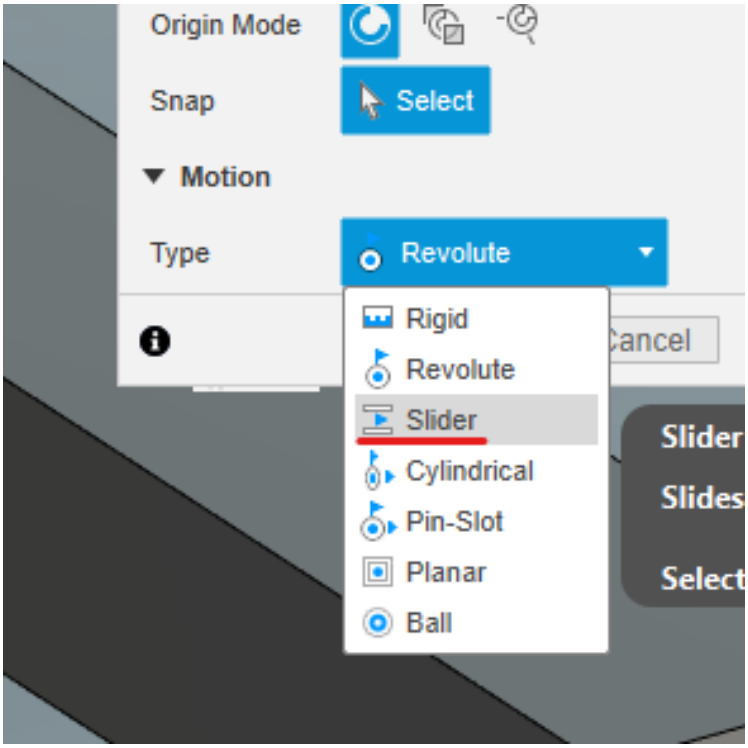
30. Repeat the same operation for the **Rod** (selected first) to **Crank** (selected second) revolute joint. You can click on the face of the extruded axle of the crank component for the joint origin.



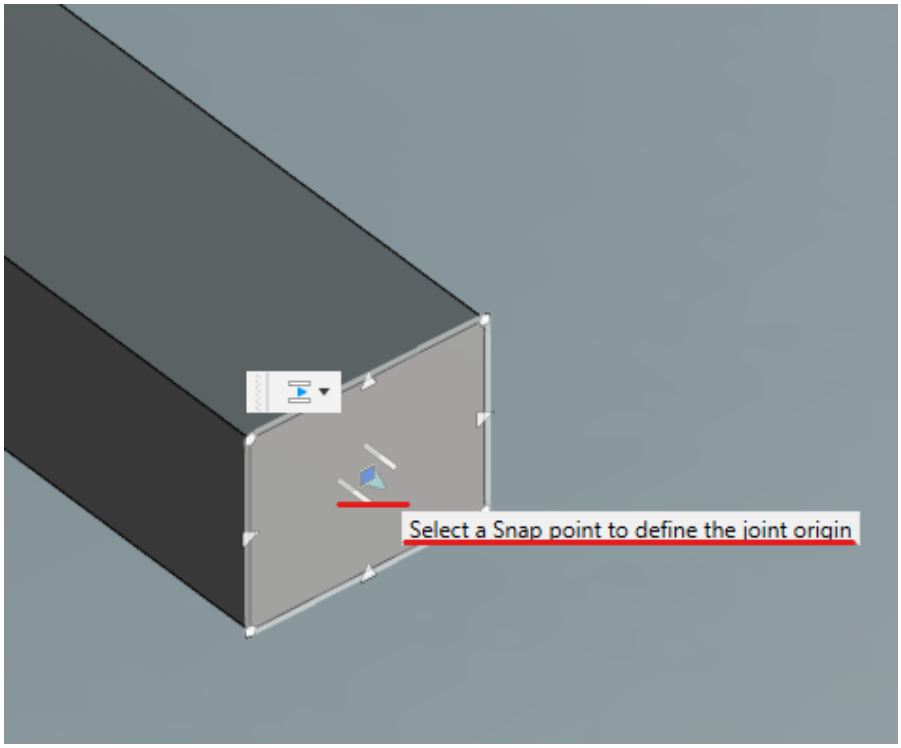
31. Our third revolute joint is from the **Slider** (selected first) to the **Rod** (selected second). Its origin is also selected by clicking on the rod's axle's face.



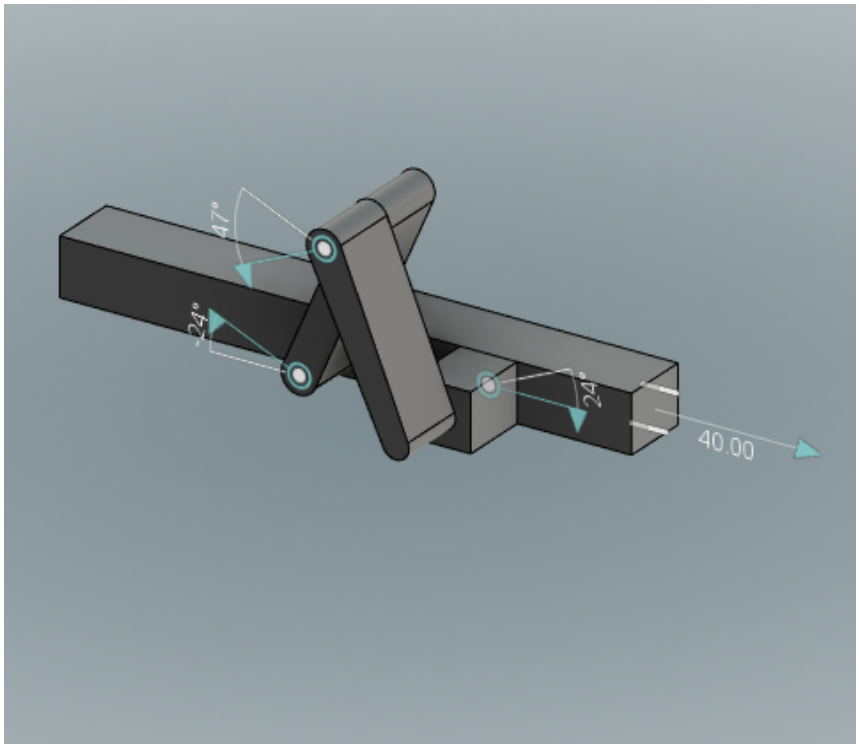
32. Lastly, we should add a slider constraint. Select the **Slider**, followed by the **SlidingRail**.



33. Select the **Slider** joint type.



34. As a joint origin, we can select the center of one of the faces of the **SlidingRail** component, along its main axis.



35. This is the final result should obtain once all previous steps are completed. You can now your mouse to drag the **Slider** component left and right and see all components move accordingly or use the slider joint's value (by double-clicking on it) and edit it precisely.

Chapter 51

Laser-Cutting

Leo Wolff

In the course, **you must study the feasibility of using the laser cutter** to bring your design to life, before considering 3D printing.

51.1 Why you should consider it

- **Time:** Cutting parts with the laser cutter is much faster than using the 3D printer. This means you can quickly iterate on prototypes version and even rebuild your project from scratch in record time.
- **Quality of Finish:** Laser-cut items have smoother and more exact surfaces than 3d-printed parts.
- **Size:** The laser cutter is able to cut pieces of up to size 1600x1000 mm, which is larger than the 300x300 mm of the Prusa printers.
- **Quick fixes:** If you make a mistake during the design and see it during assembly, there are chances you can correct it by sawing/drilling/... into the MDF (Medium Density Fibreboard, the material you will most likely use with the laser cutter) parts and fix it in the computer design so that you can still use the part you just cut, and it will be corrected next time your rebuild your prototype.
- **Robustness:** MDF is often more robust that one may think, depending on its thickness.

51.2 Designing Parts for Laser Cutting

Creating the tabs and matching cutouts used for mounting parts to eachother requires some work in Fusion 360. You can use the "Sketch > Create > Patterns > Rectangular Pattern" tool to create them more efficiently.

51.3 From Fusion 360 to the Laser Cutter

Once you have something ready to be cut in Fusion 360, it is time to export it to a format which can be handled by LightBurn, the software used at DLLEL for laser cutting. You learn how to use LightBurn in the mandatory laser-cutting tutorial.

To export a shape, left-click on the surface (of a body in Fusion 360) you want to cut, create a new sketch from it and click on “Finish Sketch” without any further modification.

Navigate to where your sketch is located in your hierarchy and rename it something meaningful. An example would be “Box-PosXNegY” for a side of a box facing the positive x and negative y direction.

Right-click on the sketch and click “Save As DXF”. Save the file either directly on a USB key or somewhere in your file system. This is the file you will have to the import into LightBurn. Now you can go laser cutting.[1]

51.4 Assembly

Once the parts have been cut, a specialized glue must be used, otherwise, the MDF parts will not stick together well. You can use the bar clamps (You can find them at the DLL EL’s welcome desk or in your personal drawers) to hold parts in place while the glue dries.

[1] After the required training, you are allowed to do this yourself.

Chapter 52

3D-Printing

Leo Wolff

Only use 3D printing when laser cutting is not suitable for your design: Laser cutting is much faster.

52.1 Basics

There are several things to consider while using a 3D printer :

- **Goal :** When printing a part, the primary objective is to have a successful print. Once this is ensured, you should also try to minimize the printing time and material used while maintaining the success of the print.

- **Constraints :** The printing nozzle melts plastic to lay it as different layers. Because the plastic takes time to dry and solidify, this has several implications discussed below.

In light of this, and before printing, **you must get feedback on your design, once imported and set up into PrusaSlicer**, either from assistants of the course, or from the DDLEL's staff, until you are yourself able to confidently identify the problems with your design.

52.1.1 Print Failures

When a print failed, it is very frequently at the start of a print. The plastic may not adhere to the printing bed, might not get out of the nozzle, etc... For this reason, it is in your best interest to stay at the beginning of the print to verify everything is starting correctly. You should also check on your print periodically for the same reason.

Even if the print doesn't fail catastrophically, it can still have important flaws that will make it completely useless. Hence, there is a number of situations you can take a look at yourself to help you pinpoint issues with your arrangement:

- **Elephant Foot phenomenon:** The material on the first layer gets displaced outwards if the distance of the nozzle to the bed is set poorly. This can be a big problem when printing something such as a cog, rendering its teeth useless. A mild elephant foot is almost unavoidable and may require post-processing (sanding) the part. If the phenomenon is very pronounced, the printer may need to be re-calibrated.
- **Vertical skew:** When printing a hole with its axis not orthogonal to the printing bed's plane, the gravity will make the hole collapse a little. In some situation, this is harmless but when the hole needs to be of precise dimensions, it can make the design useless. In case of an axle fitting through the hole for example, this phenomenon may cause the axle to be stuck in the hole.
- **Thin vertical structures:** Building tower-looking parts may not leave the time for plastic to cool down between multiple layers, making the print collapse on itself and leaving the nozzle printing plastic in the air.

To be sure to avoid such failures, read the next section: 3D printing checklist.

52.1.2 3D Printing Checklist

Here is a non-exhaustive list of things to check before starting a print:

1. In PrusaSlicer:

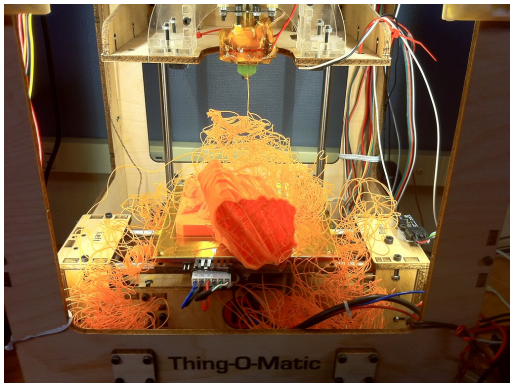


Figure 52.1: Example of a way a print can fail

(a) **Flaws:** Read the section on print failures to make sure your design doesn't have flaws that could lead to failing print.

(b) **Overhangs and Supports:** Verify if your design has overhangs or unsupported areas that might require support structures for successful printing.

(c) **Quality setting:** Double-check that the print settings in PrusaSlicer (Panel located on the right-hand side in the software) are the one you should use for the type of prints you want (Low quality for a rough, fast print or higher quality for something more polished).

(d) **Infill density setting:** Determine the infill percentage based on the desired weight and strength of your printed object.

(e) **Support structure:** Adjust the support structure settings according to your analysis from 1a

(f) **Filament setting:** Set the filament setting to the type of filament you will use.

2. Before using the printer:

(a) **Correct type of filament:** Ensure the correct type of filament is already plugged into the machine (PETG in our case). If it is not the case, ask for directions on how to change the filament to one of the teaching assistants or the DLLEL staff.

(b) **Changing the printer's filament :** The filament needs to be fed properly to the printer. If you must change the filament of

the printer, a good practice is to use cutting pliers on the end of the filament you will feed the printer to decrease the chances of it getting stuck in the printer's nozzle.

(c) **Amount of filament:** Verify that the remaining amount of filament seems enough for your print. In the case it seems uncertain, you can plug out the filament, weigh the spool with the filament and subtract the weight of the spool (which should be written on it) to compare it to the weight of filament used announced in PrusaSlicer.

(d) **Clean bed:** If the previous user of the printer you want to use is not a student of CS-358, it might happen that they forgot to clean the printing bed after retrieving their print. In this case, you should make sure the printing bed is clean before starting your impression.

(e) **Monitor your print:** Stay while the printer prints the first layer and come back to check on your print often.

3. After having used the printer:

(a) **Removing your print:** Take the printing bed out of the printer and, while wearing protection glasses, remove your print from the bed, using the tools available in the 3D printing room.

(b) **Removing the support:** While wearing glasses, remove the support from your print, and collect all the plastic to put it in the bin reserved for this purpose.

(c) **Clean the bed:** Wash thoroughly the printing bed with water, soap and a sponge and leave it to dry in the 3D printing room.

52.2 Design for 3D Printing : DOs and DONTs

As previously mentioned in this section, the goal of 3D printing, after making sure your print will be successful, is to save material and printing time. This means that your prints should be the result of your best efforts to minimize these two quantities. There are multiple reasons for doing so.

52.2.1 Printing Time

You are not alone to use the 3D printers: May it be other teams from CS-358 or students working on other projects in DLL, they all have work and deadlines they are subject to. Considering this, they should have the same opportunity to use printers as you have. This is especially important given the large number of people taking this course.

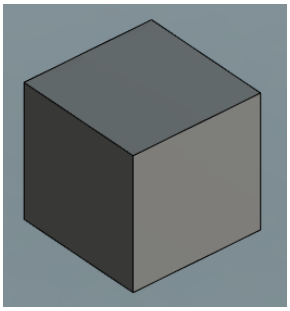
Low printing time means faster iteration: It often happens that you design a part, print it, and then realize that your design doesn't work. You then correct your mistake and start this process all over again. A lower printing time means you can iterate faster on your design, which is convenient especially when deadlines approach.

52.2.2 Material Use

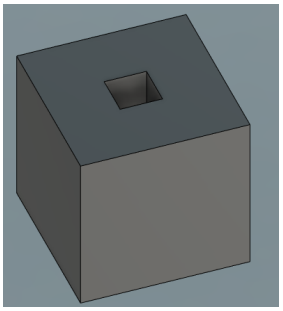
Common Sense: The goal here is to reduce waste of plastic.

Reducing printing time: Minimizing the amount of material used is often useful to reduce the printing time and therefore benefits of the advantages mentioned above. This can be achieved by cutting more holes in your design. However, this is not always the case:

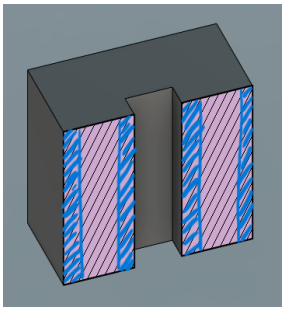
Support material: Another big source of material waste is support material. This can often be reduced by finding clever ways to print your design.



(a) A basic filled cube

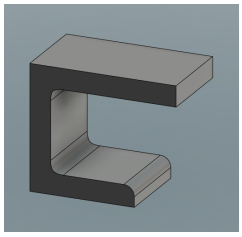


(b) We cut a hole through the cube to try to reduce material.

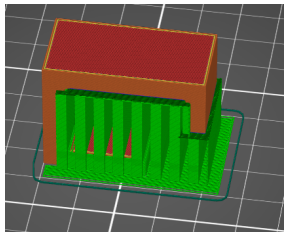


(c) All the blue parts will be using **Perimeter** print settings : This will result in higher material density and overall more material being used than for the filled cube.

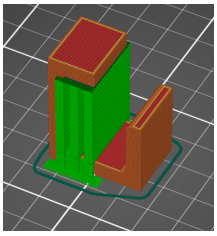
Figure 52.2: Cutting holes doesn't always result in less material used



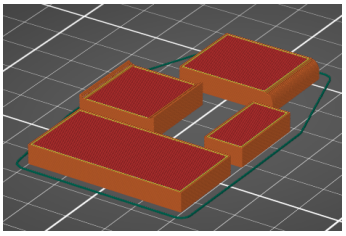
(a) The design to print



(b) Bad, it uses way too much support material.



(c) We can drastically reduce the amount of support material by printing it on another side.



(d) We can cut the part in multiple pieces to reassemble later. This uses no support material and makes smooth surfaces on the print.

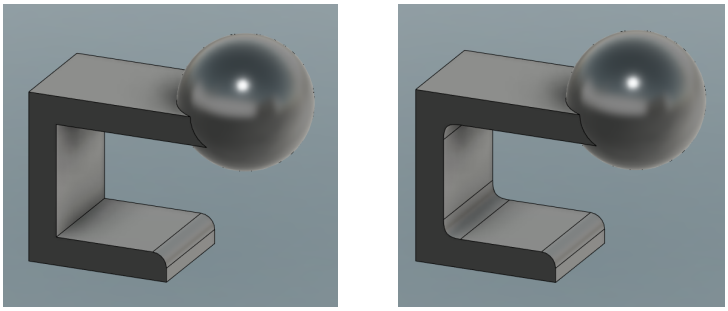
Figure 52.3: Example of reducing support material

52.3 Slicing in PrusaSlicer

A number of options are available in the PrusaSlicer software to be sure your prints succeeds and is least wasteful in term of time and material.

52.3.1 Expert Mode

To access the full potential of the application, you can set the application as expert mode by clicking on the **Configure** > **Mode** > **Expert** mode button from the top menu.



(a) Angles are a weak point in the design and will be the first place that breaks.

(b) We can use chamfer or fillet to strengthen these points.

Figure 52.4: Weak points in designs must be addressed properly.

52.3.2 Painting Tool

Once expert mode is enabled, you will be able to use the "Paint-on support" ('L' shortcut) tool.

Using this tool, you will be able to paint parts of your design where support is allowed to appear. To then use this options for the location of your support material, you have to select "For support enforcers only" in the "Supports" menu on the right panel in the software.

52.3.3 Infill

The infill drop-down menu on the right panel of the application allows you to choose the density of material the inner part of your design should have.

Settings a high value for this field may be required when your part needs to be robust, whereas a low value would be better suited for first prototypes with need of a low printing time or parts where robustness is not a problem.

52.3.4 Brim and Skirt

PrusaSlicer creates a line of material, within a configurable margin of your print's boundaries. This has multiple uses :

- Make sure the material sticks to the bed when the print starts.

- Check your design's bounds is as expected and you did not miss something while modifying the print settings.
- Leave additional time for the nozzle and its material to reach the correct temperature.

See https://make.epfl.ch/3dprint/download/training_V4.1.pdf for more information on 3D-printing in DLLEL. Here is a tutorial video on changing the filament: https://mediaspace.epfl.ch/media/Filament+change+SPOT/0_gh5ko09h

Part IX

More Programming

Chapter 53

Adding Computer Vision

Alexander Müller

53.1 Overview

This section will include some basic concepts in computer vision, some libraries to consider using, some example code using computer vision, and a case study of a project that used computer vision.

53.2 Basics

Computer vision is how computers derive information from images or videos. The first step is to take an image. This is not the place to go over all of the optics involved, but overall a lens focuses light onto a sensor. This may seem trivial but is an important consideration, as the lens will cause distortions as it tries to fit the 3D world in front of it onto the 2D sensor behind it.

53.2.1 Colors

Now, we need to have a digital representation of the image. Typically, this is a 2D array of 3 valued-tuples, with the first 2 dimensions representing a location in the image and the 3 values representing the color at that location. Here, we have a choice of base for the color system. The default on many computers and for many programmers is RGB, where the first

value represents how red the pixel is, on a scale of 0 (no red) to 255 (max red), and the same for G (green) and B (blue)[1].

But you have to ask yourself: what does max red mean? What if Alice's screen manufacturer used a different chemical or LED than Bob's, and therefore the red doesn't get as bright? Would one tuple (r,g,b) show the same color on two different screens? The answer is no, and you should avoid RGB colors in almost every computer vision application.

So what other choices do we have? While there are many, the main one used in computer vision is known as HSL. The three letters stand for hue, saturation, and lightness. The main advantage of this system is that colors' hue does not change depending on how much light is being shined on them (as opposed to RGB, where are all three values will change). This makes it a much more reliable base for computer vision, as you can specify a color filter without worrying about how well lit your target is.

53.2.2 Blurring

One of the most useful ideas in Computer Vision is blurring. There are many functions used to achieve this, but if your filter ever has problems with noise, consider filtering the input, output or both. Try low pass filters for noise reduction, high pass filters for sharpening edges, etc.

53.3 Libraries

53.3.1 OpenCV

The main library to be familiar with is OpenCV. This library, originally written for C++, has been ported to python, android, and iOS, and is completely open source. It provides the whole computer vision suite, from capturing images / videos, filtering the images, extracting shapes, and recently, even machine learning / deep neural networks. Their website offers a plethora of tutorials, algorithms and forums.

To help visualize and generate OpenCV pipelines, I recommend GRIP, which allows you to drag and drop different filters provided by OpenCV and

[1]The reason the primary colors on computers includes green instead of yellow is because computer screens are an additive color system, i.e mixing all colors makes white, whereas with paints mixing all colors makes black.

immediately see the result. Clicking through the filter options may help you understand some of the functions OpenCV provides more quickly than reading OpenCV's documentation.

53.3.2 April Tags

April tags are a fiducial marker system, which allows us to easily detect, locate, orientate, and identify QR-code like patterns. The lab that develops them offers a library in C, but I recommend using Pupil lab's python bindings (note: this library is no longer being maintained and is difficult to set up on python versions >= 3.8. Although possible, I recommend using python 3.7).

53.4 Code Examples

53.4.1 OpenCV

Let's make a basic program to detect red squares using OpenCV's python bindings. First, to download OpenCV for python:

```
> pip install opencv-python
> pip install numpy
```

We also install numpy, as many OpenCV functions rely on numpy. Now, in a python file:

```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    cv2.imshow('frame',frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Running this code will show a video stream captured live from your webcam. Pressing 'q' will end the program. 'ret' is a boolean which indicates whether or not the frame was read correctly, so it can be useful to check this value before performing any calculations with frame.

Now we will write the code to detect everything in the frame which is red. First, we convert the frame to HSV (a similar color base to HSL).

```
hsv_frame = cv2.cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

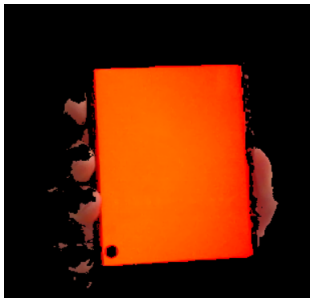
Now, we are going to apply a mask to our image. A mask takes each pixel and makes it either white, if it fulfills a certain condition, or black if it doesn't. We will use cv2's inRange method.

```
#outside the loop
min_hue = -15
max_hue = 15
min_sat = 100
max_sat = 256
min_val = 100
max_val = 256
#inside the loop
lower_red = np.array([min_hue, min_sat, min_val])
upper_red = np.array([max_hue, max_sat, max_val])
mask = cv2.inRange(hsv_frame, lower_red, upper_red)
res = cv2.bitwise_and(frame,frame, mask= mask)
```

Then, when we have the mask, we can use the 'bitwise and' function between the mask and the original image to filter out only the parts we want. This works because doing 'and' with black parts of the mask, represented with all 0 bits, will make every color in the original image black, too. On the other hand, every white pixel, which is represented with all 1 bits, will make the parts of the image we want stay the same. We can see the result using the following code:

```
cv2.imshow('res',res)
```

Here I am holding a google search for "color red" to the camera:



Now, to detect if it's a square or not. We will use several methods from opencv, including findContours, contourArea, approxPolyDP and boundingRect. Lets go through them quickly:
findContours will allow us to find a curve that denotes the boundary between two different colors. For this to work, it's better if the image is black and white, to increase the difference in colors, so we'll apply it to the mask. Then, we can use approxPolyDP to make sure it's a quadrilateral. Finally, we will use boudingRect to get the rectangle that bounds the contour, and check if the side lengths are equal. The code will looks as follows:

```
contours = cv2.findContours(mask,
    cv2.RETR_TREE,
    cv2.CHAIN_APPROX_SIMPLE)[-2]
for cnt in contours:
    area = cv2.contourArea(cnt)
    if area > 200:
        approx = cv2.approxPolyDP(cnt,
            0.01*cv2.arcLength(cnt,True),True)
        x1,y1 = cnt[0][0]
        approx = cv2.approxPolyDP(cnt,
            0.01*cv2.arcLength(cnt, True), True)
        if len(approx) == 4:
            x, y, w, h = cv2.boundingRect(cnt)
            ratio = float(w)/h
            if ratio >= 0.9 and ratio <= 1.1:
                img = cv2.drawContours(frame, [cnt], -1, (0,255,255), 3)
                cv2.putText(frame,
                    'Square', (x1, y1),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    0.6, (255, 255, 0), 2)
            else:
                cv2.putText(frame, 'Rectangle', (x1, y1),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
                img = cv2.drawContours(frame, [cnt], -1, (0,255,0), 3)
```

53.4.2 AprilTags

This section contains an extended example using the AprilTags library. The first thing to do is install the library:

```
>pip install pupil-apriltags
```

Then we can use the following code to draw blue squares around tags as well as number them:

```
#imports
from pupil_apriltags import Detector
import cv2
#instantiate detector
at_detector = Detector(families='tag36h11',
                       nthreads=1,
                       quad_decimate=1.0,
                       quad_sigma=0.0,
                       refine_edges=1,
                       decode_sharpening=0.25,
                       debug=0)
#open webcam
cap = cv2.VideoCapture(0)
while True:
        ret, frame_in = cap.read()
        #we give the detected a greyscale image
        grey = cv2.cvtColor(frame_in, cv2.COLOR_BGR2GRAY)
        #get an array of tags
        tags = at_detector.detect(grey)
        #iterate over every tag we found
        for tag in tags:
            #get the id from the QR code
            cv2.putText(frame_in, str(tag.tag_id),
                (int(tag.corners[1][0]),
                int(tag.corners[1][1])),
                cv2.FONT_HERSHEY_SIMPLEX, 1,
                (255,0,0), 3,cv2.LINE_AA)
            cv2.rectangle(frame_in,
                (int(tag.corners[0][0]),
                int(tag.corners[0][1])),
                (int(tag.corners[2][0]),
                int(tag.corners[2][1])),
                (255,0,0), 2)
        cv2.waitKey(1) & 0xFF == ord('q'):
```

```
            break
    cv2.imshow('camera', frame_in)
```

Finally, in many applications, we want to be able to find how much the April Tag has been rotated in the plane of the camera. Luckily, this is very easy, as the library always returns the corners in the same order. We just take the average position of two pairs of adjacent corners:

```
pt1 = [int(x) for x in [(tag.corners[3][0]+tag.corners[0][0])/2,
    (tag.corners[3][1]+tag.corners[0][1])/2]]
pt2 = [int(x) for x in [(tag.corners[1][0]+tag.corners[2][0])/2,
    (tag.corners[1][1]+tag.corners[2][1])/2]]
cv2.line(frame_in, pt1, pt2,(0,255,255), 3)
```

Many more position calculations are possible using linear algebra, such as how rotated the April Tag is w.r.t to the plane of the camera.

Chapter 54

GRBL and LinuxCNC

To be added. These are software packages to create plotters, 3D printers and CNC mills. LinuxCNC also has support for inverse kinematics in (6-DOF) robot arms.

Chapter 55

MCU Operating systems: FreeRTOS and ROS

To be added.

Chapter 56

Mobile App Control

Giovanni Ranieri

In this chapter, we show how to talk to an ESP8266 from a mobile phone app with React Native. We also give different resources for other frameworks. You can use any microcontroller with WiFi capabilities, you will just need to find the corresponding libraries.

56.1 React Native App and ESP8266

React[1] is a web JavaScript framework that aims to create basic web pages and apps. React Native[2] is a derived framework build on top of React to build native apps for Android and iOS.
The easiest approach for an app with React Native is to create a NodeJS project and install the react and react native packages using the package manager of your choice (npm, yarn, ...). Then, your application and the microcontroller mainly communicate through HTTP requests.

56.1.1 Expo CLI

React Native is platform dependent, i.e. you need to write different code for each targeted development platform (iOS, Android, ...). To avoid this, you can use Expo which is an ecosystem of tools to build complete apps

[1] https://fr.legacy.reactjs.org/
[2] https://reactnative.dev/

for both iOS and Android in one project: you just need to install it in your project. The last thing to add for your project will be the mobile Expo Go app which will compile and create your app.

56.1.2 Creation of the Project

This tutorial follows the detailed documentation of Expo: [3]. Follow the instructions on this page to create your project. If you want to use TypeScript instead of Javascript, refer to this page [4] but be aware that it requires more work.

When the project is created, you can enter into it's root directory and open the project. Like every NodeJS project, you'll find the node modules folder that will hold the packages installed for your app. The last thing to do to have your app on your phone is to start the app using `npx expo start`. This will print a QR code that you will scan from the Expo Go app. At this point, you should have your app on your screen with a welcome message.

56.1.3 HTTP Requests

HTTP, which stands for Hypertext Transfert Protocol, is a application protocol in the model OSI that uses TCP as transport protocol to enable stable client-server communication. The communication setup that you would like to have is indeed a client-server where the server is hosted on your microcontroller and the client that sends requests is your app. Your microcontroller needs to be connected to a network and you're going to have two options. If you're at the SPOT in the DLL building at EPFL, then you can use a private network specially created for connecting embedded systems (please refer to the section 32.2). If you're not there, then you can use the hotspot of your phone. In any case, your chip will be connected to a network to enable the framework for HTTP requests to work. Obviously, your microcontroller needs a WIFI module and if not then you need to use an external WIFI module for your chip (see the reference for all chips 28).

Many frameworks exist to send HTTP requests but we'll show you one of

[3] https://docs.expo.dev/tutorial/create-your-first-app/
[4] https://docs.expo.dev/guides/typescript/

the simplest called axios: [5]. Just install the package with

```
npm install axios
```

When you use HTTP requests, you can specify the type of request you want, for example POST and GET requests. If you already used this kind of framework then it should not be hard to use this one. Here is simple POST request using axios with JSON format to the IP address 193.245.45.3 with the route "root" it's simply

```
function sendRequests(key: string, val: string, root: string) {
    axios.post('http://193.245.45.3/' + root, {
      [key]: val
    }, {
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
      }
    }).then((response) => {
      console.log(response);
    }, (error) => {
      console.log(error);
    });

    }
}
```

56.1.4 Handle HTTP Requests

Now on the side of your microcontroller we need first to import some libraries to instantiate a web server, enable WIFI and handle requests. For the ESP8266 you can use ESP8266WiFi, ESPAsyncTCP and ESPAsyncWebServer.

To send requests to your chip, we need the IP address of it and this can be done simply by connecting the chip to the network you have chosen. The Access Point (your phone) will give an IP address to your chip that can be printed in the terminal like in the example below

[5] https://axios-http.com/docs/intro

```
void setup() {
  Serial.begin(9600);

  Serial.print("WIFI ...");
  # IF YOU USE THE HOTSPOT
  WiFi.begin(ssid, password);
  # IF YOU ARE IN THE DLL BUILDING
  WiFi.setHostname("your_host_name");
  Serial.print("Connecting to Wifi...");
  Serial.print("\n");

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("");
  Serial.println("Wifi connected");
  Serial.println(WiFi.localIP());
}
```

You can copy this IP address printed and use it for your HTTP requests framework. **Warning:** this IP address can change

After that, always on the server-side, you can instantiate a web server and define, in the setup function, routes that will handle communications. In the example below, we have this AsyncWebServer instance on port 80.

```
AsyncWebServer server(80);

void setup() {
  Serial.begin(9600);

  Serial.print("WIFI ...");
 # IF YOU USE THE HOTSPOT
  WiFi.begin(ssid, password);
  # IF YOU ARE IN THE DLL BUILDING
  WiFi.setHostname("your_host_name");
  Serial.print("Connecting to Wifi...");
```

```
    Serial.print("\n");

    while (WiFi.status() != WL_CONNECTED) {
      Serial.print(".");
      delay(500);
    }

    Serial.println("");
    Serial.println("Wifi connected");

    server.on("/mode", HTTP_POST, [](AsyncWebServerRequest *request){
      if(request->hasParam("value", true)) {
        AsyncWebParameter* p = request->getParam("value", true);
        int value = p->value().toInt();

        update_mode(value);
        request->send(200, "text/html", "good");

      } else {
        request->send(404, "text/html", "Error mode");
      }
    });

    server.begin();
  }
```

with this you have a route called "/mode" that can handle POST requests. It checks if the data sent contains a key called "value" with the value true. You can send now a request to your chip with your app with the previous code of your NodeJS project.

56.2 Other Mobile App Frameworks

Many other frameworks/programming languages has been developed for mobile app control. We let you here references to these ones:

- Flutter: https://flutter.dev/
- NativeScript: https://nativescript.org/

- Swift (iOS): https://developer.apple.com/swift/
- Kotlin (Android): https://developer.android.com/codelabs/build-your-first-andro
0

Part X

More Hardware

Chapter 57

Using Gamepad Controllers

Dylan Vairoli

Gamepad controllers provide an intuitive way to interact with your components, especially with vehicles. This section will introduce two differents ways to interact with controllers.

- Using a WiFi connection with an ESP8266 microcontroller
- Using a direct Bluetooth connection with an ESP32 microcontroller

57.1 WiFi connection

You can connect a controller to your computer and transmit its inputs to an ESP8266 microcontroller using HTTP requests via WiFi. Note that with this alternative you can't interact with the controller (vibrations, change LEDs colour, ...). Consider using a Bluetooth connection instead if you need these functionalities.

This section is based on the official *Using the Gamepad API* Mozilla tutorial[1].

[1] https://developer.mozilla.org/en-US/docs/Web/API/Gamepad_API/Using_the_Gamepad_API

57.1.1 Connect a gamepad in the browser

All this section content has been tested on the Safari, Google Chrome and Firefox browsers and using a PS4 controller.

First, download the Github repository [2] associated with the tutorial. We will use it to make sure your gamepad is detected in your browser and its outputs are correctly fetched.

You then need to connect your gamepad to your computer. Check online how to connect your specific gamepad to your personal computer using either wireless connectivity or a physical cable. For reference, we connected a PS4 controller to a macOS computer using bluetooth.

Once your gamepad is connected, if you open the `index.html` file from the repository you downloaded and press any buttons on your gamepad, an interface should show up. It displays a mapping between the buttons and their ID in code, joysticks values are also displayed.

57.1.2 Retrieve the gamepad data

You can find the base code template in the class resources repository [3]. In this section, we will go through a minimalistic JavaScript example to retrieve the gamepad input.

```
const haveEvents = "ongamepadconnected" in window;
const controllers = {};

function connecthandler(e) {
  console.log("Gamepad connected: " + e.gamepad.id);
  addgamepad(e.gamepad);
}

function addgamepad(gamepad) {
  controllers[gamepad.index] = gamepad;
  requestAnimationFrame(updateStatus);
}

// [...]
```

[2] https://github.com/luser/gamepadtest/tree/master
[3] https://github.com/epfl-cs358/cs358-resources

```
function disconnecthandler(e) {
  console.log("Gamepad disconnected: " + e.gamepad.id);
  removegamepad(e.gamepad);
}

function removegamepad(gamepad) {
  delete controllers[gamepad.index];
}

// [...]

function scangamepads() {
  // Scan through all gamepads and add them to controllers
}

window.addEventListener("gamepadconnected", connecthandler);
window.addEventListener("gamepaddisconnected", disconnecthandler);

if (!haveEvents) {
  setInterval(scangamepads, 500);
}
```

This piece of code subscribes to gamepad connection and disconnection events (if any) and call the appropriate handlers. If no event is available, it just scans repeatedly through the navigator gamepads every 0.5 seconds. When adding a gamepad, the `requestAnimationFrame` function asks the navigator to execute `updateStatus` as soon as possible.

```
function updateStatus() {
  scangamepads();

  Object.entries(controllers).forEach(([i, controller]) => {
    controller.buttons.forEach((button, i) => {
      let val = button;
      let pressed = val === 1.0;
      if (typeof button === "object") {
        pressed = button.pressed;
        val = button.value;
```

```
      }
      // Do something with val and pressed
    }

    controller.axes.forEach((axis, i) => {
      const val = axis // axis is the axis value directly
      // Do something with axis
    }
  }

  requestAnimationFrame(updateStatus);
}
```

This is our main loop, in which we fetch the button values. The `controllers` variable contains `Gamepad` objects, which provides some useful properties:

- `id`: A string containing some information about the controller (USB vendor, product id, name, ...)
- `index`: An integer that is unique for each gamepad currently connected to the system
- `buttons`: An array of `GamepadButton` objects representing the buttons present on the device
  - `GamepadButton.pressed`: A boolean indicating whether the button is currently pressed (`true`) or unpressed (`false`)
  - `GamepadButton.value`: a floating point value used to represent analog buttons, such as the triggers on many modern gamepads. The values are normalized from 0.0 (not pressed) to 1.0 (fully pressed)
- `axes`: An array representing the controls with axes (e.g. analog thumb sticks). Each entry in the array is a floating point value from -1.0 (lowest value) to 1.0 (highest value)

Note that `updateStatus` calls itself recursively on the next animation frame.

57.1.3 Communication with an ESP8266

JavaScript client

Once you fetched the gamepad data, you need to send it to your microcontroller. In this section, we will use an ESP8266.

We will extend the previous script to send data to our ESP8266 over HTTP. Note that both your computer and your microcontroller need to be connected over the same WiFi. The EPFL WiFi won't work because of additional security, you should use a personal hotspot or a regular WiFi.

We want to reduce as much as possible the ongoing traffic, to do so we define new variables that will contain the last values we sent. We will send POST requests only if the values actually changed this frame.

```
// Remember last values to send POST requests on change only
const lastVal = {};
const lastPressed = {};
const lastAxis = {};
```

We use the Fetch API [4] along with the JSON object [5] to send HTTP POST requests in the JSON format to the ESP8266:

```
async function postData(url, data) {
  console.log("POST DATA " + JSON.stringify(data));
  const response = await fetch(url, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(data),
  });
  return response;
}
```

In our main loop, we first retrieve the ESP8266 IP address from an input field (defined in `index.html`) and then create our JSON object.

[4] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
[5] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

```
function updateStatus() {
  scangamepads();

  if (ipInput == null)
    ipInput = document.getElementById('ip-input');
  const url = `http://${ipInput.value}/data`;

  const controllersData = createControllersData(controllers)
  if (controllersData.length > 0)
    postData(url, {controllers: controllersData });

  requestAnimationFrame(updateStatus);
}
```

In this example, we send data only on updates with the following format:

```
{
  controllers: [{
      id: 0,
      buttons: [{ id: 7, val: 1, pressed: true }],
      axes: [{ id: 0, val: 0.66 }]
  }]
}
```

Take a look at the implementation of `createControllersData`, you might adjust it for your specific needs.

Arduino server

On the ESP8266, we need to:

- Connect to WiFi (using the ESP8266WiFi library [6]) Be careful as you will have troubles for connecting to the normal `epfl` WiFi. Please refer to the WiFi section 32.2
- Create and start a web server handling POST requests arriving at `/data` (using the ESP8266WebServer library [7])

[6] https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html
[7] https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer

- Parse the received JSON data to extract the meaningful information (using the ArduinoJson library [8])

Take a look at the ESP8266 code in the resources repository. It should be pretty straightforward. Whenever you don't understand something, look it up in the corresponding library documentation.

57.2 Bluetooth connection

You can directly connect a controller to an ESP32 via Bluetooth. This method is more precise than WiFi, allow you to interact with the controller (vibrations, change LEDs color, ...) and retrieve more data such as the internal gyroscope of the gamepad. Moreover, it removes the need of an intermediary computer to pass the data.

We recommend using the `Bluepad32` library by Ricardo Quesada [9]. It supports a lot of gamepads and is well integrated in the Arduino IDE environment.

Follow this tutorial [10] to run the simple test example.

You should be able to adapt this example to your specific use cases.

57.2.1 Troubleshooting

- If you have an ESP32-CAM without any USB port, you'll need to use an FTDI or an ESP8266 to upload your code and read the serial monitor.

[8] https://arduinojson.org
[9] https://bluepad32.readthedocs.io
[10] https://www.youtube.com/watch?v=0jnY-XXiD8Q

Part XI

Case Studies

Chapter 58

The Prusa i3 MK3S 3D Printer

This is the main model of 3D printer that you will use in your project. But let us now consider this machine as a maker project. In fact, creating such a 3D printer is not beyond the scope of this course. Even though Prusa is a for-profit company, all parts of the printer's design (i.e., it is hardware too) are open-source. The printer is a refined version of a series of open-source, non-profit efforts to create 3D printers that you can make yourself, and that can "replicate" themselves. As you can verify yourself, most of the non-standard parts of the printer are 3D-printed (on the same model of printer).

58.1 Mechanical Frame

The frame consists of three parts cut from a thick steel plate (most prominently, the large donut-shaped part arranged vertically) and some aluminium 2020 extrusions. These are connected to each other using screws. One could make these parts oneself from wood or by 3D printing, but it is extremely important that particularly these parts are precisely machined and very rigid.

Actuated by the X and Y axis motors, the print head moves (not absolutely, but relatively to a point on the print bed) in a plane parallel to the print bed. It is vital for consistently successful prints that these planes are parallel with an error of no more than about 10 micrometers! During mechanical assembly, we need to get the donut-shaped steel part to be perpendicular to the base plane with an error of a small fraction of a millimeter at the far end; if that succeeds, the remaining precision can be

Figure 58.1: The printer's x, y, and z axes.

achieved through calibration (i.e., to make the software compensate for errors). These printers are also sold as kits for self-assembly, and getting the frame precisely right is somewhat tricky. The order in which one tightens the screws matters a lot!

58.2 Actuation

Not counting the fans, whose speed is also program-controlled while printing (for instance, it is increased when bridges and overhangs are printed, to make those lines of plastic solidify faster to minimize the effects of gravity), this printer has four degrees of freedom, implemented using 5 stepper motors (we have motors of the same specification in the parts library). There is an X axis (left-right when you look at the printer from the front), a Y axis (forward-backward), a Z axis (up-down, using two stepper motors), and an E axis which extrudes filament. It may seem that a stepper motor is overkill for extrusion, but in fact the speed of extrusion varies a lot during a print for good outcomes, and even the direction of this motor reverses frequently.

Only X and Z move the print head, while Y moves the print bed. This is different from some other printers, which have a stationary print bed and move the print head with X, Y, and Z. A design in which Z moves the print bed down while X and Y move the print head would also be an imaginable option.[1] By this decoupling (not all degrees of freedom move the print head), one may create a more precise machine, and the motors may potentially have to move less mass.

[1] Resin printers do something a little bit like that: They have only one mechanically moving axis – Z – moving a plate to which the printed parts are attached up and away from UV light matrix and liquid resin.

All five stepper motors are of the same kind, and have a torque rating of about 0.4 Nm. That wouldn't be much if we, say, had to move a vehicle with large-diameter wheels. But this is not what we are doing here. Let us see how power is transmitted to the axes. X and Y use timing belts, in both cases without reduction (the belt runs over gears/spindles of the same diameter). Z uses worm gears.

Timing belts need the right amount of tensioning, and this may need to be adjusted over time as the belt stretches. The timing belt of the Y axis is manufactured as a loop and is tensioned by a facility to shift the front spindle a little and then fix it in place. The timing belt of the X axis is open-ended and the two ends are fixed with the right tension in a 3D-printed contraption on the back side of the print head.

So what is the linear force transmitted by the timing belts to bed and print head? It depends on the diameter of the gears, which is about 1cm. We can calculate a force of

$$0.4Nm/(0.01m * \pi) \approx 10N$$

TODO calculate how much is needed to overcome inertia.

TODO calculate linear force of Y axes via worm gear. Point out that the two motors are not so much in place to create sufficient force but to move both ends of the X-axis and print head assembly equally fast to avoid shear forces (check term). To avoid these shear forces, an alternative would be to mount a single Y-axis motor with worm gear centered below the X-axis assembly, but obviously that would be absurd – it would be in the way of the print head, the print bed, and the object to be printed.

58.3 Electronics

The printer uses a 24V power supply and can supply a significant amount of current. This power supply is probably the most expensive individual component of the printer. It is mounted in a box on the right side of the frame.

The main board with the stepper motor drivers is housed in a 3D printed box on the left side of the frame. The board is an EINSI RAMBO board, which is specially made to contain all the essential electronics of such a printer on one board (though it isn't exclusive to Prusa printers). You could

achieve essentially the same functionality with an Arduino, five A4988 stepper drivers, and a few additional components.

The display and SD card reader are in an orange printed box in front, but all the related processing is done on the main board.

58.4 Cable Management

Cable management is a significant problem here, also because some of the cables (those leading to the print head and the X axis stepper) constantly move and bend while printing, making them age quickly. Typically, the cables are where these printers fail first.

There is also a huge number of cables (on the order of 20 lines to the print head), and it is thanks to the clever cable management (and protective sleeves) that the cables are more prominent. You can but these printers as kits for self-assembly, and if you do that, you spend a significant part of the assembly time on cable management, and it is necessary to get a usable machine.

58.5 3D-Printed Parts of the Printer

The 3D-printed parts are done using PETG. Compared to, say, PLA, PETG has a slightly higher melting point (still, some of the 3D printed parts of the print head sometime melt when a misprint happens and a big ball of filament sticks to the print head) and is tougher (breaks or splinters less easily). However, it is less rigid than PLA, and if you chose to 3d print the frame of the printer (in any case a bad idea), PLA would be better than PETG.

Chapter 59

The Making Intelligent Traffic Project

Alexander Müller

In this chapter, I will discuss some stories, insights and reflections from one of the projects from the first year of this course, Making Intelligent Traffic. The idea of the project was a swarm-like traffic system, where model autonomous cars could navigate a small city and communicate between themselves to avoid crashes while reducing traffic.

59.1 Backup Plans

59.2 Problem Solving: Localisation

In this section, we will look at the process we went through to localize the cars. One of the biggest hurdles during the project was localising the cars. We considered many systems, some of which you may consider using yourselves for your project.

59.2.1 Problem Definition

The first thing we did was set up some criterion to find a good solution. This may seem like listing the obvious, but it can really help filtering out solutions or coming up with unique approaches. In our system, we needed

to localize the cars in 2 dimensions. The cars, being less than 10cm long, needed to be localized with 1 or 2cm. Although we didn't know it at the time, the system should also work underground, as we had our set up in a basemen with no reception of any kind for most of the semester. Lastly, we needed to be able to identify each vehicle so that we could keep track of who's who, and orientate them so we knew which way they were going.

59.2.2 GPS

Probably the most well-known localization system worldwide. Using satellites in strategic points in space, it is possible to triangulate your position anywhere in the world. It was quite easy to discard as an idea, because even relatively good GPS trackers have more than one meter of error, and this only worsens when indoors.

59.2.3 Bluetooth

Similar to GPS, Bluetooth (and other related systems using WiFi, ultra-wideband) based systems use several transmitters to triangulate a receiver by measuring how long it takes to send packets. Unfortunately, it is not possible to orient the vehicles, and most of these systems have $\sim$ 1m error.

59.2.4 CHILI Cellulos

EPFL's CHILI Lab created a positioning system for their Cellulo robots, based on an array of dots printed on a paper. The robots use downward facing cameras to identify the unique patterns in the dots, allowing for sub-millimeter accuracy. However, we planned to run the demo on the floor on a 2x2 meter area, which would have used a massive amount of paper, and we could not find any publicly available resources for setting it up ourselves.[1]

59.2.5 OpenCV

We also considered using OpenCV to detect colored shapes on our vehicles. However, this proved to be challenging as there was often interference from

[1] It may be worth it look into this depending on your project.

the floor, which was somewhat colorful with lots of little shapes. This is probably mitigable with careful filter tuning, using filters, etc. Additionally, it would be difficult (although possible) to orient the cars. However, while we were looking into this option, we also found a library that solved all of our problems:

59.2.6 AprilTags

AprilTags are such a powerful tool, there is an entire section in this book dedicated to them. I will not discuss the specifics of how they work here, but rather how much effort it ended up taking to get it installed. The first thing to note is that the original

Chapter 60

Previous Team Projects

60.1 2022

Autonomous Sailboat
https://zeck69.github.io/autopilot_boat/

Brain/Computer Interface
https://github.com/EPFL-EEG-Team

Swarm of Autonomous Cars
https://github.com/AnirudhhRamesh/Intelligent-Traffic-Backend

Magic Chessboard
https://github.com/Wizard-s-Chess/Wizards-Chess

Mobile Robot Arm / Shopping Assistant
https://github.com/WollfieGitHub/MIT_Robotic_Arm

Motion tracking glove
https://github.com/nfelber/IMU-Motion-Tracking-Glove

Portable Arcade
https://thecl3m.github.io/PortableArcade/

Sand Plotter
https://github.com/Sand-Table/table-block

Smart Glasses
https://vigarov.github.io/SmartGlass/

60.2 2023

Autonomous Forklift
https://github.com/loicmisenta/Autonomous_Forklift

Face-controlled 2D Plotter
https://facedoodle-docs.netlify.app/

Mario Kart
https://github.com/MIT-Mario-Kart/MIT-Mario-Kart

Robotic Snake
https://youtu.be/mGbinKsSFXk

Self-balancing Unicycle
https://mit-unicycle.github.io/mit-unicycle/

SLAM Vehicle
https://github.com/kreslotim/Wall-SLAM

StruMaster (self-playing guitar)
https://github.com/pmdlt/Strumaster

60.3 2024 and Later

Projects from Spring 2024 and later can be found in the github organization epfl-cs358, see https://github.com/epfl-cs358/.