Legged Robots: Practice Sessions

Week 2

Updates

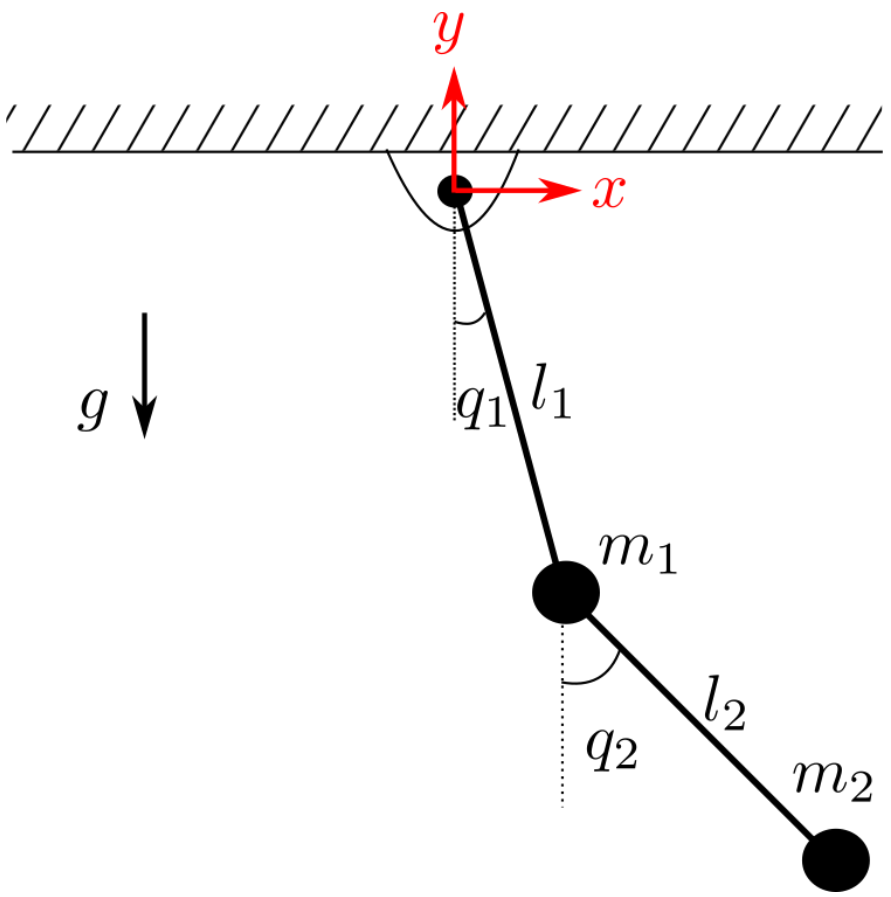
- Instructions for virtual environment installation and tutorials have been added on Gitlab: https://gitlab.epfl.ch/lgevers/lr-practicals

- Solutions for the practicals will be released by the end of Week 3

W1-3 Fundamentals

- Part 1: Double pendulum kinematics and dynamics
- **Part 2: Jacobian (Cartesian PD + Force Control)**
- Part 3: Inverse Kinematics (compare with force control)
- Part 4: Single-leg hopping

Recall: Double Pendulum Forward Kinematics



Position

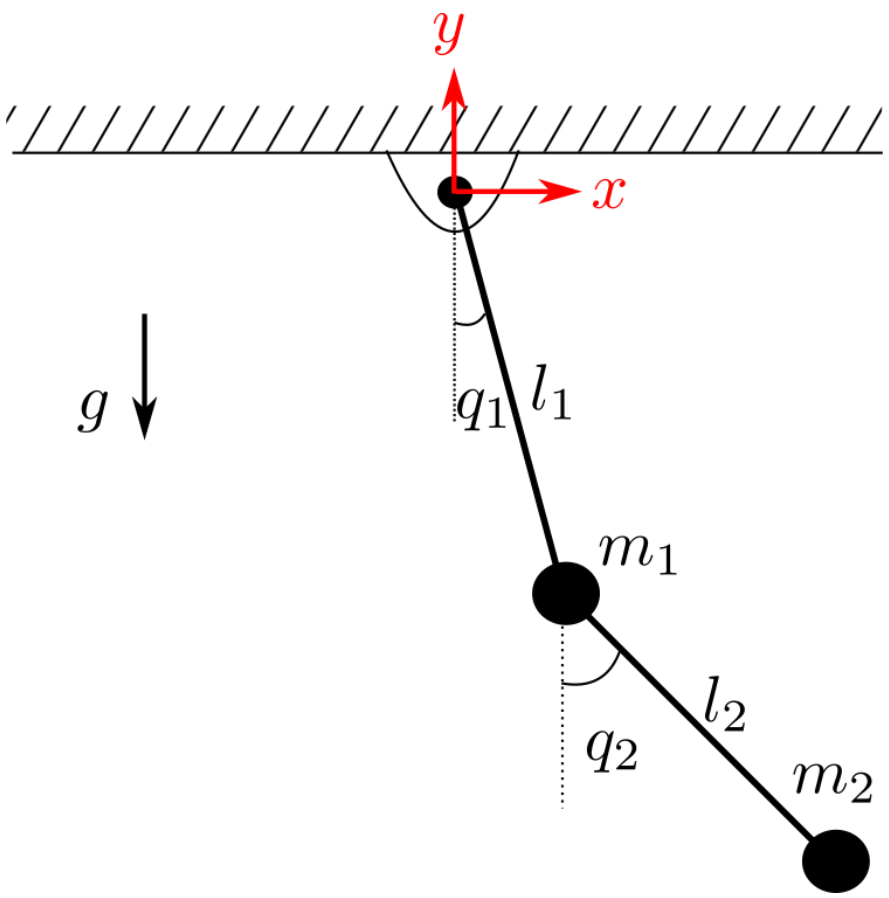
$$x_1(q_1) = l_1 sin(q_1(t))$$
$$y_1(q_1) = -l_1 cos(q_1(t))$$
$$x_2(q_1, q_2) = l_1 sin(q_1(t)) + l_2 sin(q_2(t))$$
$$y_2(q_1, q_2) = -l_1 cos(q_1(t)) - l_2 cos(q_2(t))$$

$$\dot{x} = \frac{dx}{dt}$$

Velocity

$$\dot{x}_1(q_1, \dot{q}_1) = l_1 cos(q_1(t))\dot{q}_1(t)$$
$$\dot{y}_1(q_1, \dot{q}_1) = l_1 sin(q_1(t))\dot{q}_1(t)$$
$$\dot{x}_2(\dots) = l_1 cos(q_1(t))\dot{q}_1(t) + l_2 cos(q_2(t))\dot{q}_2(t)$$
$$\dot{y}_2(\dots) = l_1 sin(q_1(t))\dot{q}_1(t) + l_2 sin(q_2(t))\dot{q}_2(t)$$

Rewrite



Position

$$x_1(q_1) = l_1 sin(q_1(t))$$
$$y_1(q_1) = -l_1 cos(q_1(t))$$
$$x_2(q_1, q_2) = l_1 sin(q_1(t)) + l_2 sin(q_2(t))$$
$$y_2(q_1, q_2) = -l_1 cos(q_1(t)) - l_2 cos(q_2(t))$$

$$\dot{x} = \frac{dx}{dt}$$

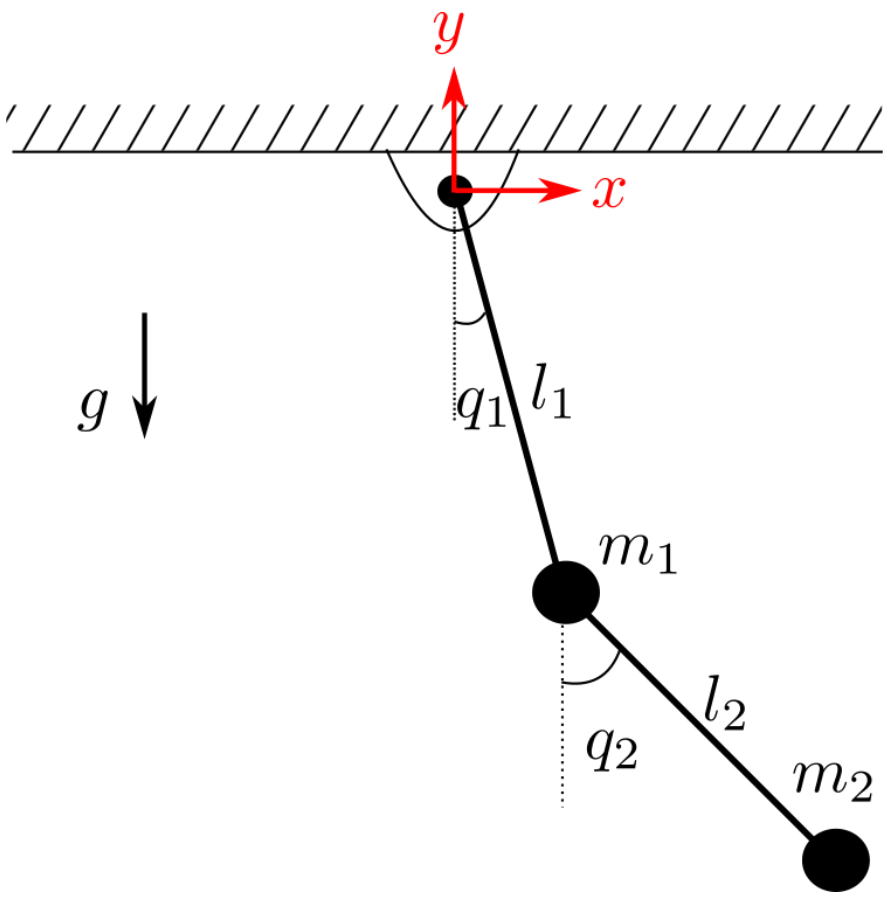
Velocity

$$\dot{x}_1(q_1, \dot{q}_1) = l_1 cos(q_1(t))\dot{q}_1(t)$$
$$\dot{y}_1(q_1, \dot{q}_1) = l_1 sin(q_1(t))\dot{q}_1(t)$$
$$\dot{x}_2(\dots) = l_1 cos(q_1(t))\dot{q}_1(t) + l_2 cos(q_2(t))\dot{q}_2(t)$$
$$\dot{y}_2(\dots) = l_1 sin(q_1(t))\dot{q}_1(t) + l_2 sin(q_2(t))\dot{q}_2(t)$$

Velocity matrix form

$$\begin{bmatrix} \dot{x}_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} l_1 cos(q_1) & l_2 cos(q_2) \\ l_1 sin(q_1) & l_2 sin(q_2) \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

Jacobian



Velocity matrix form

$$\begin{bmatrix}\dot{x}_2\\ \dot{y}_2\end{bmatrix} = \begin{bmatrix}l_1cos(q_1) & l_2cos(q_2)\\ l_1sin(q_1) & l_2sin(q_2)\end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix}$$

Jacobian

$$J(q) = \begin{bmatrix}l_1cos(q_1) & l_2cos(q_2)\\ l_1sin(q_1) & l_2sin(q_2)\end{bmatrix}$$

Velocity matrix form

$$\begin{bmatrix}\dot{x}_2\\ \dot{y}_2\end{bmatrix} = \begin{bmatrix}l_1cos(q_1) & l_2cos(q_2)\\ l_1sin(q_1) & l_2sin(q_2)\end{bmatrix}\begin{bmatrix}\dot{q}_1\\ \dot{q}_2\end{bmatrix}$$

$$v = J(q)\dot{q}$$

Jacobian

- Helps to connect Joint space (joints) and Task space (end-effector).
- Matrix equivalent of the derivative
- The derivative of a vector-valued function with respect to a vector

$$\boldsymbol{y} = F(\boldsymbol{x}) \qquad \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m$$

- Linear map between differentials
- The Jacobian is an *m* x *n* matrix

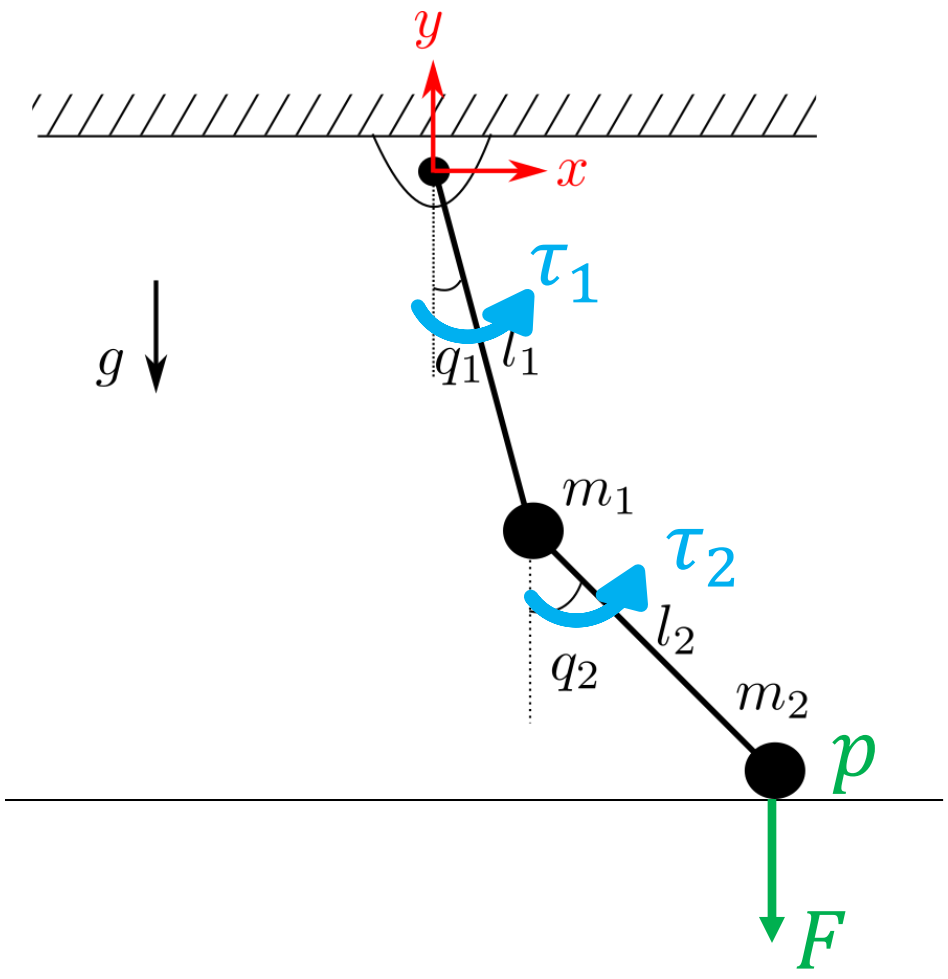
n

m

$$\boldsymbol{J} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

How can we use the Jacobian to control our system?

$$\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$$



- If we want to apply a force ***F*** at the end effector, we can compute the corresponding desired joint torques $\boldsymbol{\tau}$ with:

$$\boldsymbol{\tau} = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{F}$$

- Why? Principle of Virtual Work

Virtual = Small/imagined

  - For a virtual displacement $\delta\boldsymbol{q}$ at the joints, the virtual displacement $\delta\boldsymbol{p}$ at the end effector is:

$$\delta\boldsymbol{p} = \boldsymbol{J}(\boldsymbol{q})\delta\boldsymbol{q}$$

  - Work done by ***F***:

$$\boldsymbol{F}^T\delta\boldsymbol{p} = \boldsymbol{F}^T\boldsymbol{J}(\boldsymbol{q})\delta\boldsymbol{q}$$

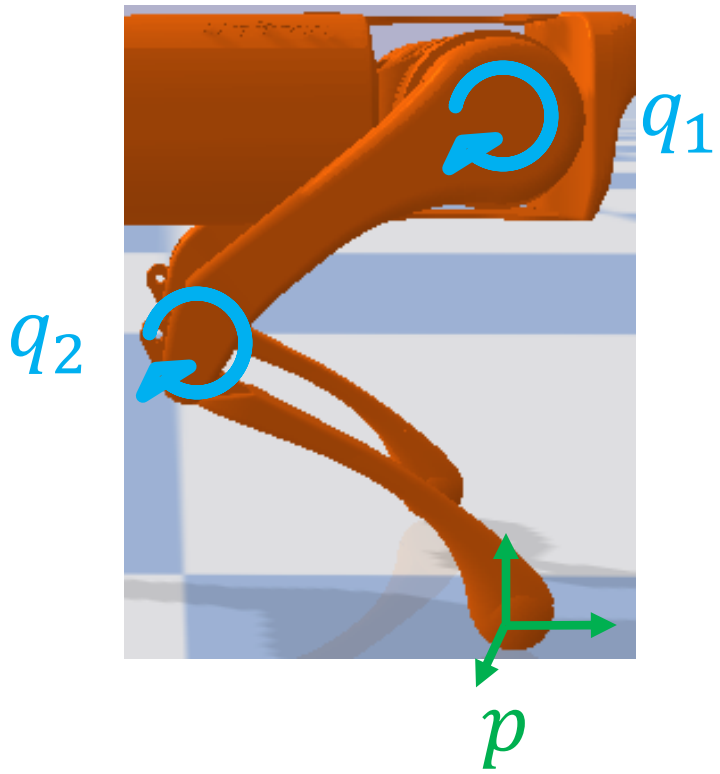
  - Work done by joint torques $\boldsymbol{\tau}$:

$$\boldsymbol{\tau}^T\delta\boldsymbol{q}$$

  - Work done must be equal

$$\boldsymbol{\tau}^T\delta\boldsymbol{q} = \boldsymbol{F}^T\boldsymbol{J}(\boldsymbol{q})\delta\boldsymbol{q} = \left(\boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{F}\right)^T\delta\boldsymbol{q}$$

Summary: Joint angles ←→ Cartesian space



$$p = f(q)$$ Forward kinematics

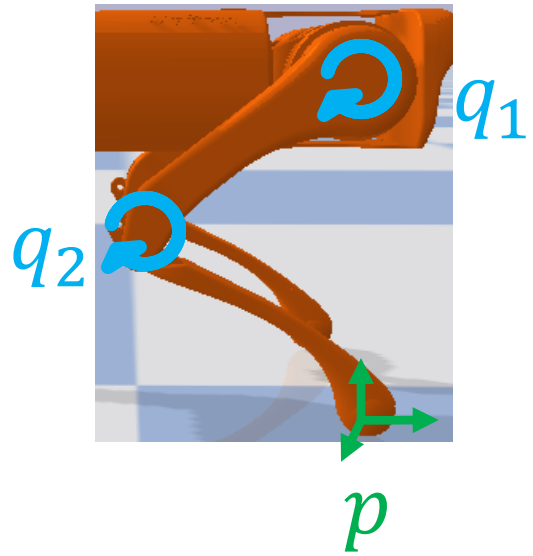
$$q = f^{-1}(p)$$ Inverse kinematics

$$\dot{p} = v = J(q)\dot{q}$$ Foot linear velocity

$$\tau = J^T(q)F$$ Map desired end effector force to torques

Leg Control

Goal: How much torque do I apply?



$$p = f(q)$$ Forward kinematics

$$q = f^{-1}(p)$$ Inverse kinematics

$$\dot{p} = v = J(q)\dot{q}$$ Foot linear velocity

$$\tau = J^T(q)F$$ Map desired end effector force to torques

"I want to move my joint to a certain angle" **Joint PD**

"I want my end effector to go to a certain position" **Cartesian PD**

"I want to apply an extra force at the foot" **Extra applied torque**

Leg Control

Goal: How much torque do I apply?

Joint PD

$$\tau_{joint} = K_{p,joint}(q_d - q) + K_{d,joint}(\dot{q}_d - \dot{q})$$

Cartesian PD

$$\tau_{Cartesian} = J^T(q)\left[K_{p,Cartesian}(p_d - p) + K_{d,Cartesian}(v_d - v)\right]$$

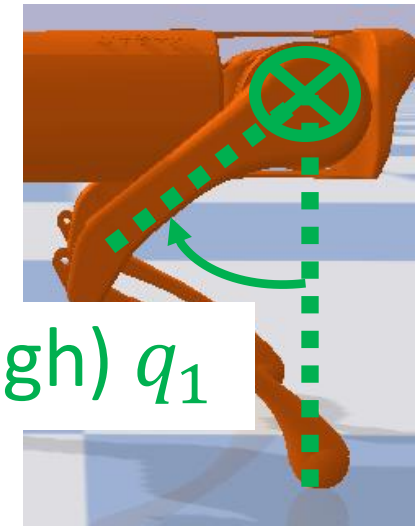
Extra applied torque

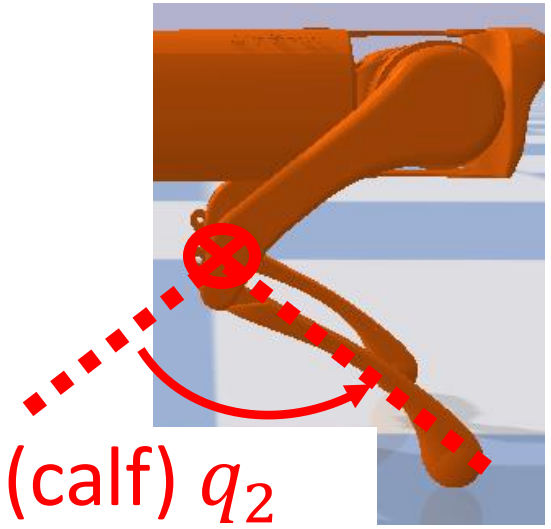
$$\tau = J^T(q)F$$

Total required torque

$$\tau_{final} = \tau_{joint} + \tau_{Cartesian} + J^T(q)F$$

Pybullet Quadruped Leg Joint References





- Careful: what is the axis?
- In figure, q_1 is $\pi/4$ (axis into page)
- In figure, q_2 is $-\pi/2$ (axis into page)

$$x_2(q_1, q_2) = l_1 sin(q_1(t)) + l_2 sin(q_2(t))$$
$$y_2(q_1, q_2) = -l_1 cos(q_1(t)) - l_2 cos(q_2(t))$$

Your turn: derive the Jacobian!

Hint

$$J = \frac{\partial F}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

W1-3 Fundamentals

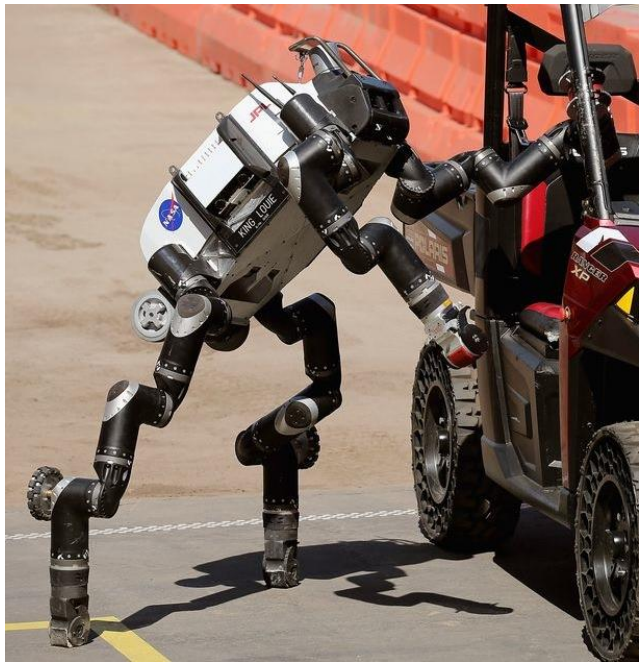
- Part 1: Double pendulum kinematics and dynamics
- Part 2: Jacobian (Cartesian PD + Force Control)
- **Part 3: Inverse Kinematics (compare with force control)**
- Part 4: Single-leg hopping

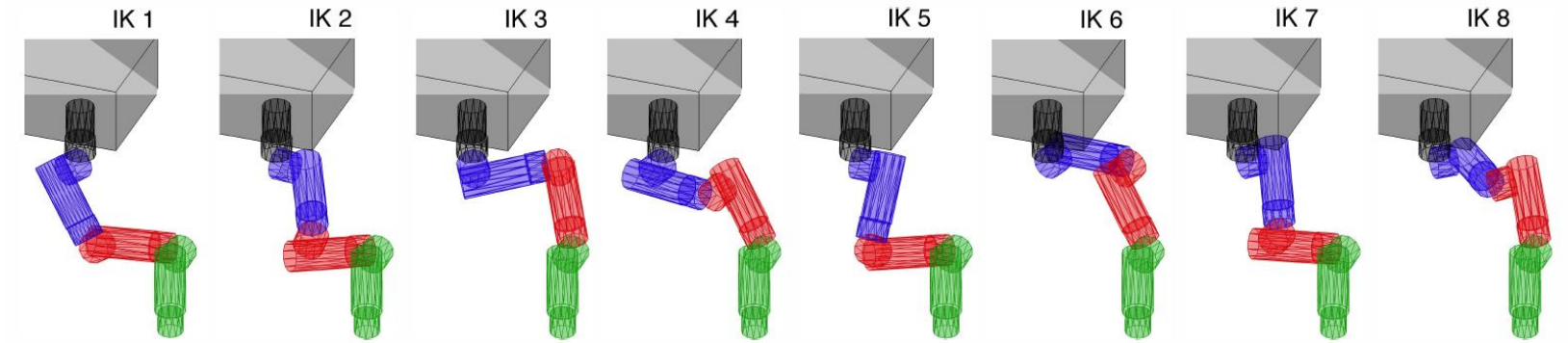
Inverse Kinematics Today

1. **Analytical**
   - Very fast
   - Exact solutions
   - Impossible for complex robots
   - Complicated equations

2. **Iterative (numerical)**
   - Can handle complex tasks
   - Can include constraints
   - Works for almost any robot geometry
   - Slow
   - Convergence depends on good initial guess

JPL's RoboSimian: Inverse Kinematics

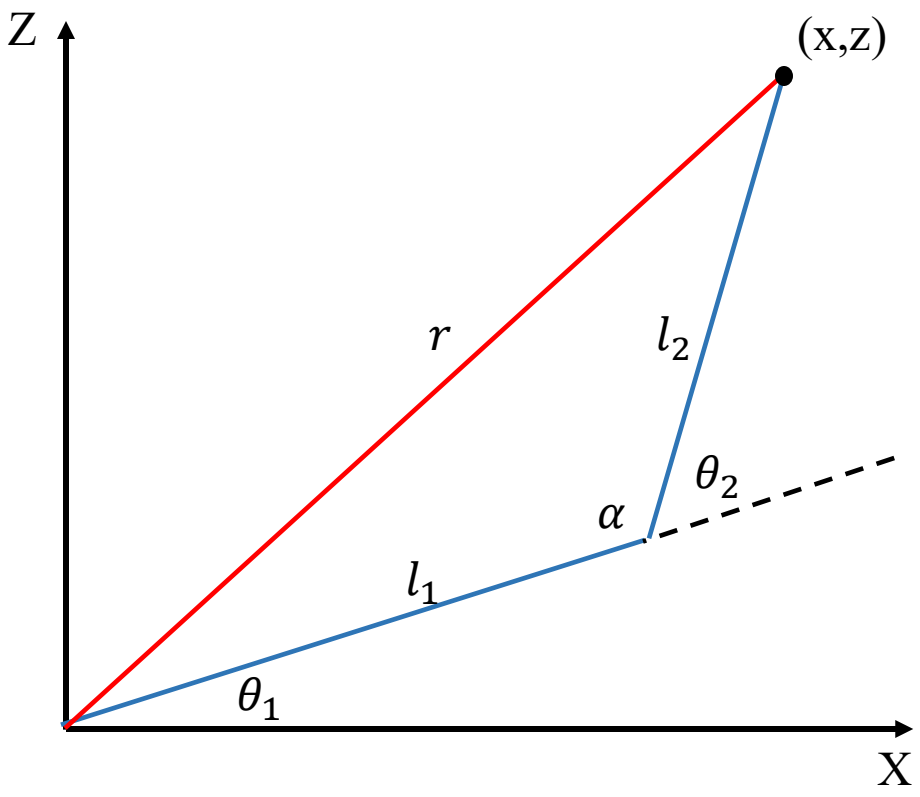




- Four identical limbs, each with 7 degrees of freedom (DOFs)
- 8 Inverse Kinematics "Families"
  - 3 "elbows" → 2^3 choices
- How should we choose between these?

Analytical Inverse Kinematic

Goal: Make θ_2 as primary variable



- Recall the cosine rule:

$$r^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos\alpha$$

- Therefore:

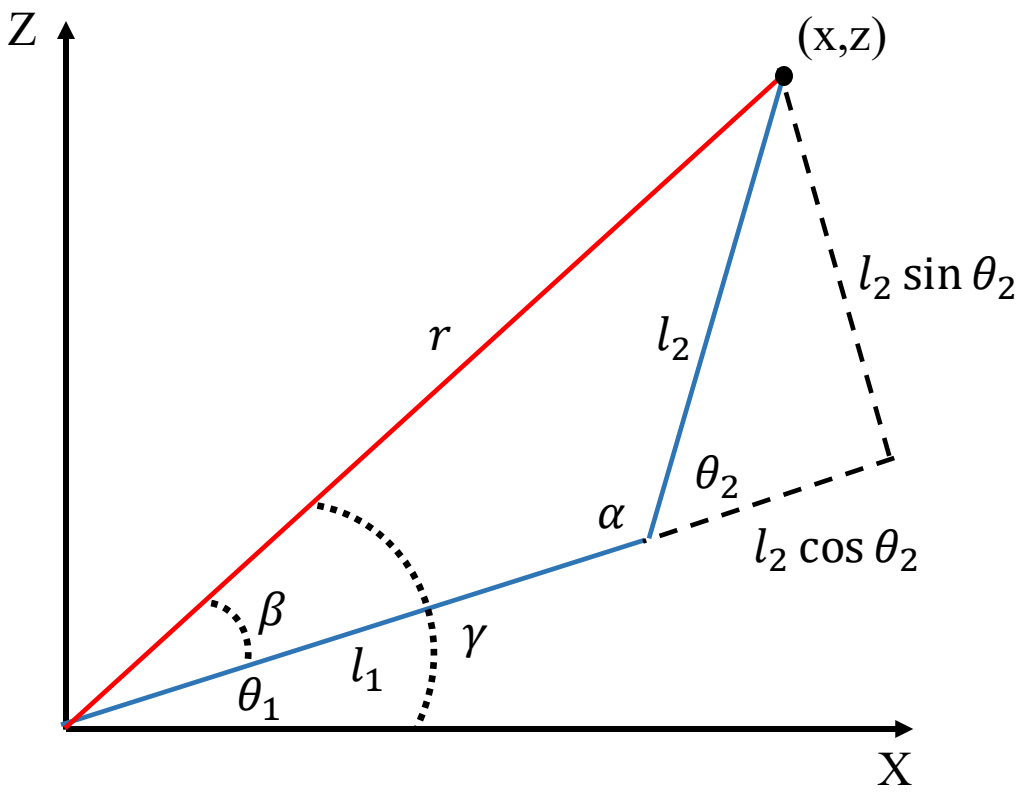
$$\cos\alpha = \frac{l_1^2 + l_2^2 - r^2}{2l_1 l_2} = \frac{l_1^2 + l_2^2 - x^2 - z^2}{2l_1 l_2}$$

- Since $\theta_2 = \pi - \alpha$,

$$\cos\theta_2 = -\cos\alpha = \frac{r^2 - l_1^2 - l_2^2}{2l_1 l_2} = \frac{x^2 + z^2 - l_1^2 - l_2^2}{2l_1 l_2}$$

Analytical Inverse Kinematics

Goal: Make θ_1 as primary variable



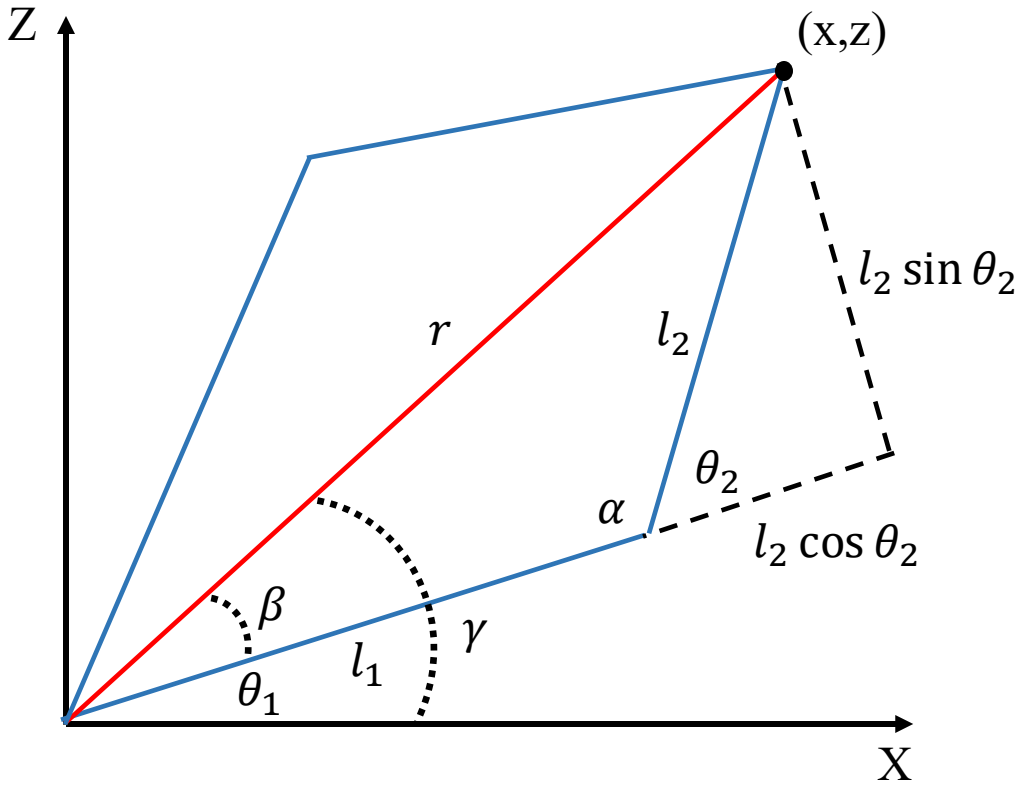
- New variables: γ, β

$$\gamma = \tan^{-1}\left(\frac{z}{x}\right), \qquad \theta_1 = \gamma - \beta$$

- Therefore:

$$\theta_1 = \tan^{-1}\left(\frac{z}{x}\right) - \tan^{-1}\left(\frac{l_2 \sin\theta_2}{l_1 + l_2 \cos\theta_2}\right)$$

Analytical Inverse Kinematics

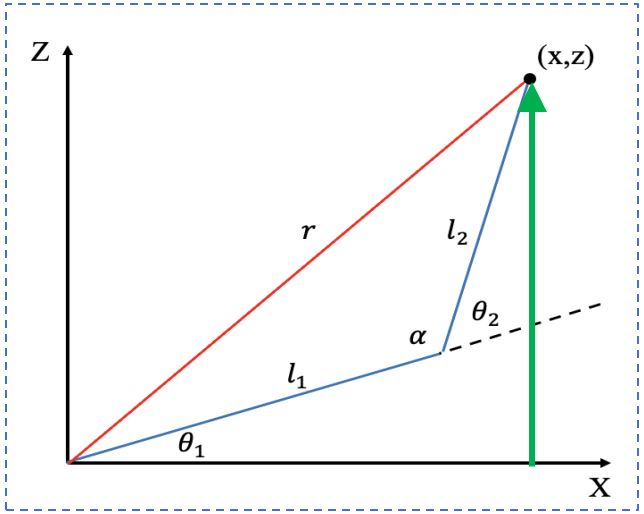


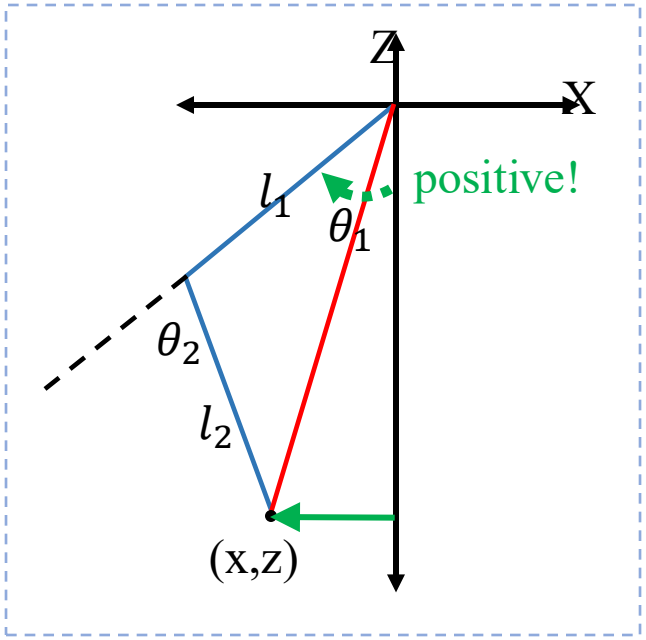
- Summary (both solutions):

$$\theta_2 = \pm \cos^{-1}\left(\frac{x^2 + z^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{z}{x}\right) \mp \tan^{-1}\left(\frac{l_2 \sin\theta_2}{l_1 + l_2 \cos\theta_2}\right)$$

Careful: Pybullet Reference Frame





$$\theta_2 = \pm \cos^{-1}\left(\frac{x^2 + z^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{z}{x}\right) \mp \tan^{-1}\left(\frac{l_2 \sin\theta_2}{l_1 + l_2 \cos\theta_2}\right)$$

$$\theta_2 = \mp \cos^{-1}\left(\frac{x^2 + z^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)$$

$$\theta_1 = np.arctan2(-x, -z) - \tan^{-1}\left(\frac{l_2 \sin\theta_2}{l_1 + l_2 \cos\theta_2}\right)$$

How is this useful?

$$\theta_2 = \mp \cos^{-1}\left(\frac{x^2 + z^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)$$

$$\theta_1 = np.arctan2(-x, -z) - \tan^{-1}\left(\frac{l_2 \sin\theta_2}{l_1 + l_2 \cos\theta_2}\right)$$

Joint PD $\tau_{joint} = K_{p,joint}(q_d - q) + K_{d,joint}(\dot{q}_d - \dot{q})$

Iterative Inverse Kinematics

Degrees of Freedom (DOF) = How many independent ways the mechanism can move

Actuation = How many joints you can control

- What if our system is more complex?
  - **Jacobian is not square**
    - Underactuated (Fewer actuations than DOF of task)
      - 3 DOF leg (3 joints (hip, thigh, calf) but only 1 is actuated/controlled, 2 are passive)
    - Overactuated/Redundant (More actuations than DOF of task)
      - 7 DOF arm controlling 3D end-effector position and orientation (6)
  - **Jacobian near singularity (det(J) ~ 0)**
    - Robot "loses ability to exert force in some directions"
    - Instability, leads to extremely large joint velocities

Iterative Inverse Kinematics

$$v = J(q)\dot{q}$$

$$\dot{q} = J^{-1}(q)v$$

Exists?

Goal: Is there any way to get a 'safe inverse'?

Moore-Penrose Pseudo-Inverse

Note: A = J

- Generalization of the matrix inversion operation for non-square matrices
- Consider a general matrix $A \in \mathbb{R}^{m \times n}$. When $m \geq n$ and $rank(A) = n$, define the *left pseudo-inverse* A_l^+, with $A_l^+ A = \mathbb{I}^{n \times n}$

$$A_l^+ \coloneqq (A^T A)^{-1} A^T$$

(Fewer joints than task dimensions)

- If $m \leq n$ and $rank(A) = m$, define the *right pseudo-inverse* A_r^+, with $A A_r^+ = \mathbb{I}^{m \times m}$

$$A_r^+ \coloneqq A^T (A A^T)^{-1}$$

(More joints than task dimensions)

- To handle singularities, define the *damped pseudo-inverses* with damping factor λ as:

Higher λ = Stable
Lower λ = Closer to true Pseudo-Inverse but less stable near singularities

$$A_l^+ \coloneqq (A^T A + \lambda^2 \mathbf{I}_{n \times n})^{-1} A^T$$

$$A_r^+ \coloneqq A^T (A A^T + \lambda^2 \mathbf{I}_{m \times m})^{-1}$$

Near singularity

Iterative Inverse Kinematics

Algorithm 1: Iterative inverse kinematics

Data: p_{des}, q^0 ; /* Desired foot position, initial joint angle guess */
Result: q ; /* Joint angles satisfying p=f(q) */
$q \leftarrow q^0$;
$p \leftarrow f(q^0)$;
4 **while** $\|p - p_{des}\| > tol$ **do**
  $J \leftarrow J(q)$; /* Evaluate Jacobian for current q */
1 $J^+ \leftarrow (J)^+$; /* Update the pseudoinverse */
2 $\Delta p \leftarrow p_{\text{des}} - p$; /* find the end-effector error */
3 $q \leftarrow q + \alpha J^+ \Delta p$; /* update the joint angles (step size α) */
end

1. Check what Pseudo-Inverse should be used
2. Compute current error in end effector
3. Compute joint angle — α = Scalar gain (Control step size)
4. Repeat iteratively until convergence

Assignment

- Gitlab link: https://gitlab.epfl.ch/lgevers/lr-practicals
- Instructions are on Moodle
- This week: Part 2 only
- The assignment for weeks 1-3 are not graded