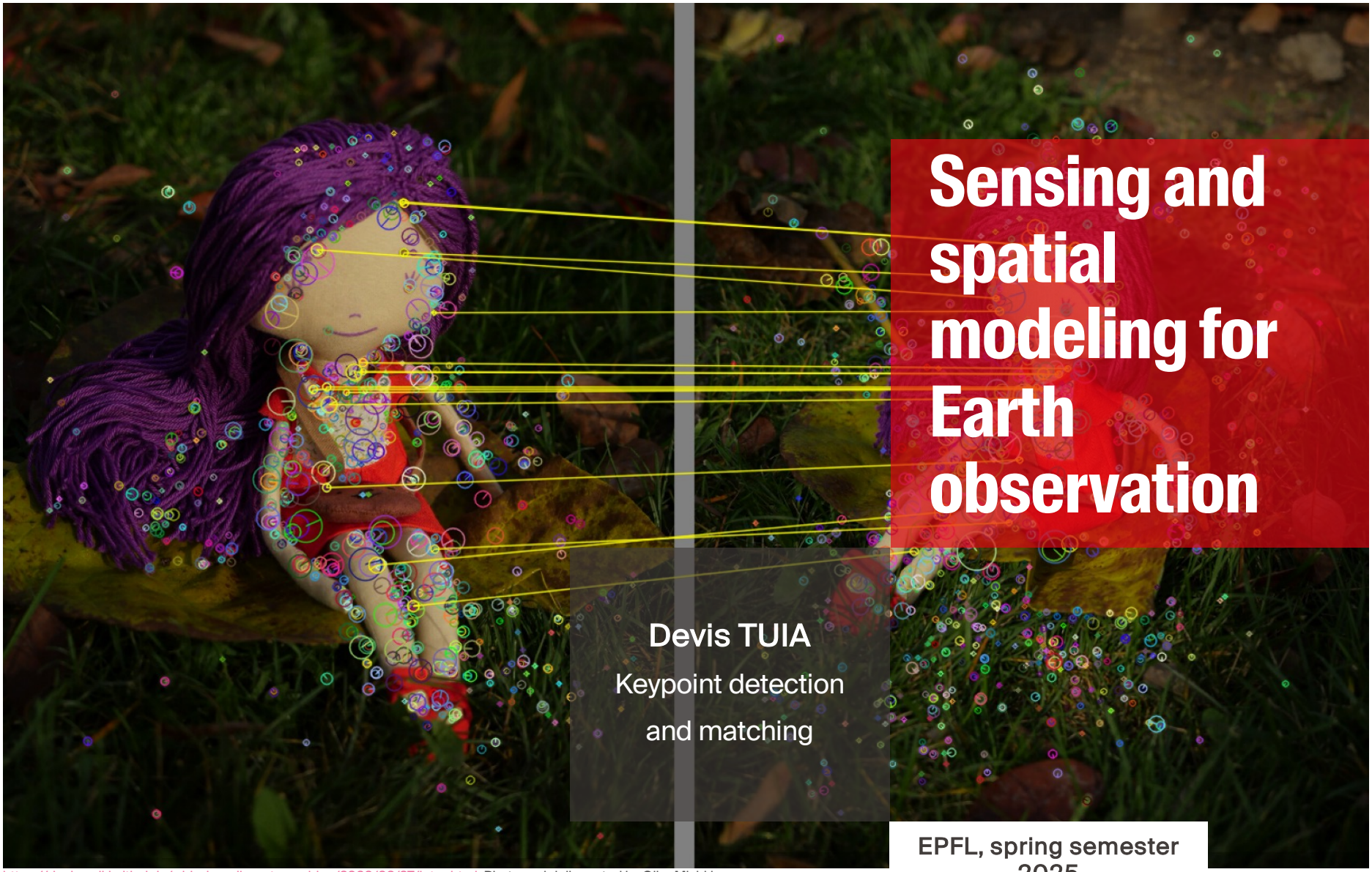
EPFL



Sensing and spatial modeling for Earth observation

Devis TUIA

Keypoint detection and matching

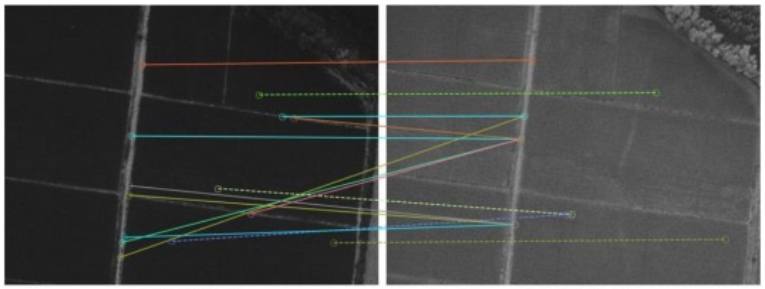
EPFL, spring semester 2025

https://ducha-aiki.github.io/wide-baseline-stereo-blog/2020/03/27/intro.html. Photo and doll created by Olha Mishkina

École polytechnique fédérale de Lausanne

Why image matching?

Paperswithcode.com



(a) Red Band (b) REG Band

Meng et al., JAG, 2021

When we acquire series of images (e.g. with a drone) and want to

- Create a mosaic
- Reconstruct a 3D surface
- Make 3D point measurements
- Identify and track moving objects
- Align different bands/ sensors acquired at different times

How does image matching work?

- AIM: **Identify** and uniquely **match** identical object features in two or more images of the object

- Here
  - Green points are correct matches
  - Red points are misses

Sarlin et al., CVPR, 2019

How? (in non-technical terms)

Let's say you have found the picture on the left and would like to take the same image in summer.

How would you do it?

What if you cannot take the image with you?

https://ducha-aiki.github.io/wide-baseline-stereo-blog/2021/01/09/wxbs-in-simple-terms.html

How? (in non-technical terms)

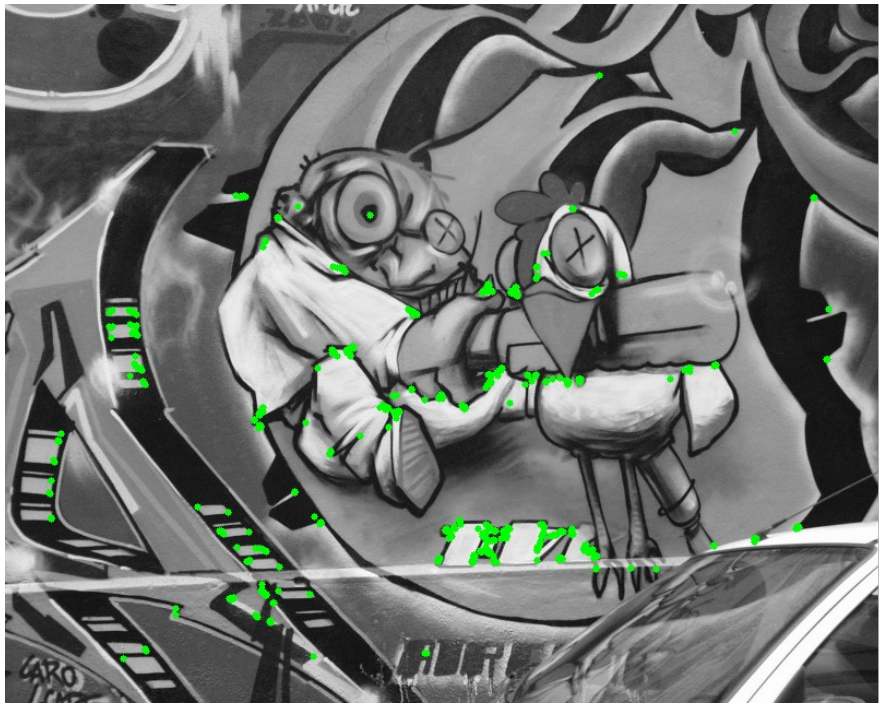
1. Identify salient objects and features in the images ("trees", "statues", “tip of the tower”, etc).
2. Describe the objects and features, taking into account their neighbourhood: “statue with a blue left ear".
3. Match: establish potential correspondences between features in the different images, based on their descriptors.
4. Estimate in which direction one should move the camera to align the objects and features.

https://ducha-aiki.github.io/wide-baseline-stereo-blog/2021/01/09/wxbs-in-simple-terms.html

Given two images to be matched

https://towardsdatascience.com/improving-your-image-matching-results-by-14-with-one-line-of-code-b72ae9ca2b73

Detect some keypoints

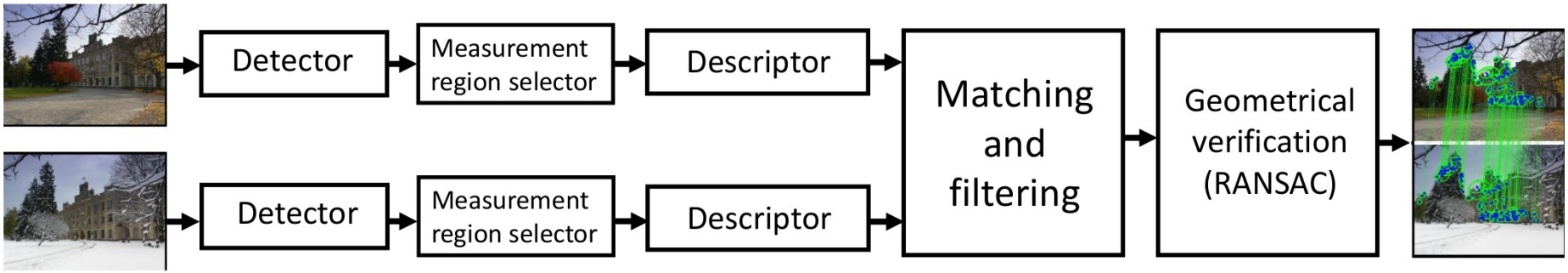


https://towardsdatascience.com/improving-your-image-matching-results-by-14-with-one-line-of-code-b72ae9ca2b73

And find a way to match them!

https://towardsdatascience.com/improving-your-image-matching-results-by-14-with-one-line-of-code-b72ae9ca2b73

The recipe is simple



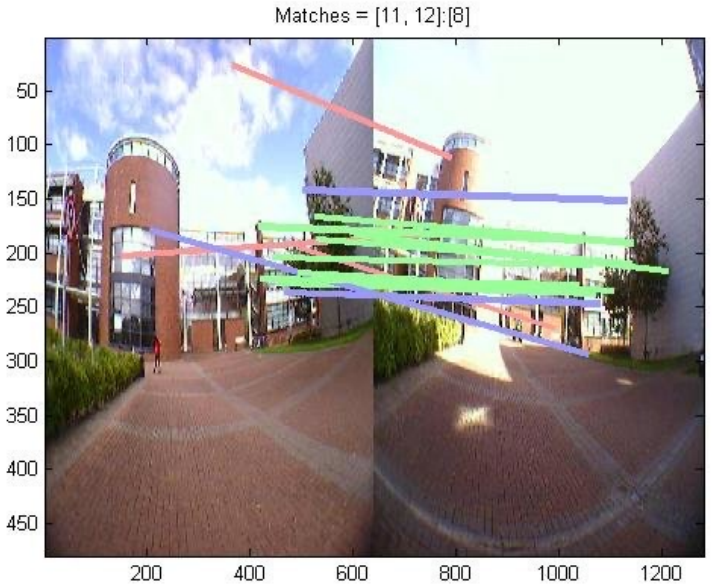
https://ducha-aiki.github.io/wide-baseline-stereo-blog/2021/02/11/WxBS-step-by-step.html

1. Compute interest points/regions in all images independently
2. For each interest point/region compute a descriptor of their neigborhood (local patch).
3. Establish tentative correspondences between interest points based on their descriptors.
4. Robustly estimate geometric relation between two images based on tentative correspondences with RANSAC.

Why so difficult?

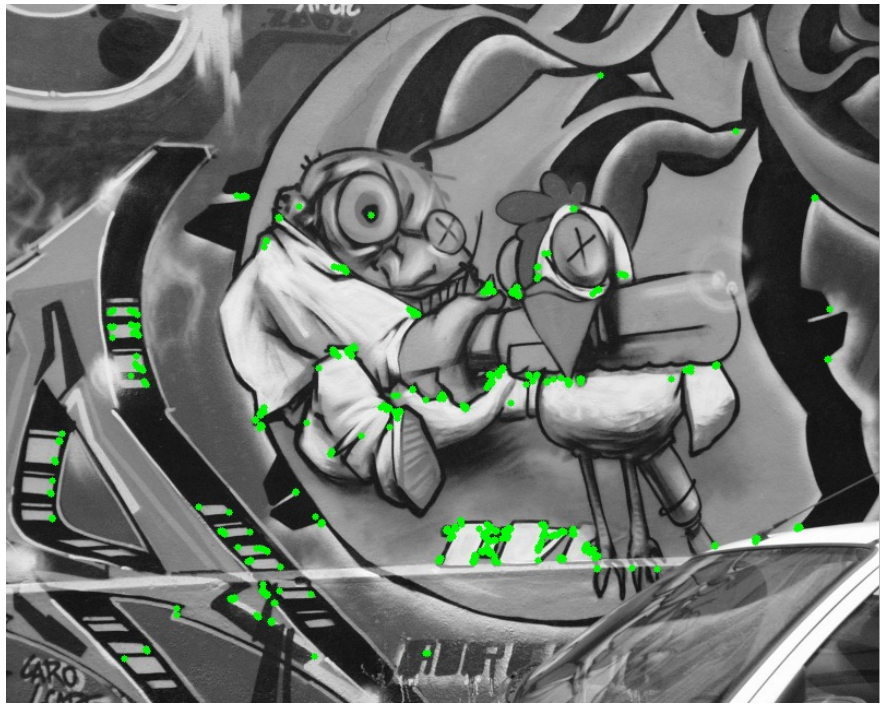
Matching is inherently ill posed, a unique and univoque solution might not exist, due to:

- Occlusions
- Several candidates for a match
- Noise in the image
- Changes in acquisition conditions (day/night, intensity)
- Changes in the object in between acquisition



Redzic et al., DEXA 2010

1. Detect keypoints: interest points / regions



https://towardsdatascience.com/improving-your-image-matching-results-by-14-with-one-line-of-code-b72ae9ca2b73

Interest operators extracting keypoints

- Algorithms that extract distinctive image points (keypoints)
- These points are potential suitable candidates for matching

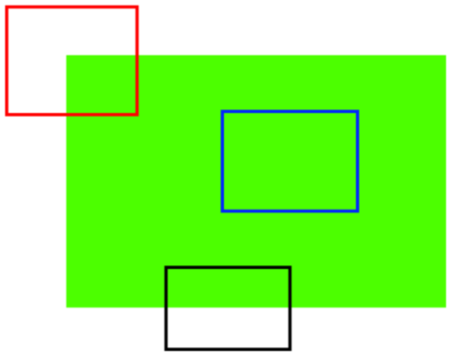
- Good characteristics:
  - Individuality = locally unique, distinct from background
  - Invariant to geometric and radiometric distortions
  - Robust to noise
  - Rare = globally unique

- These keypoints are extracted on each image separately

Interest operators extracting keypoints

- We want keypoints that are unique and invariant
  - **Unique**: they happen rarely in the image
  - **Invariant**: if you shift or rotate the image, they are still the same

- Blue patch: you move it around, it looks the same
- Black patch: on an edge, if you move it horizontally it looks the same
- Red **patch: corner, anywhere you move it, it looks different → Good keypoint!**



image

https://docs.opencv.org/4.x/df/d54/tutorial_py_features_meaning.html

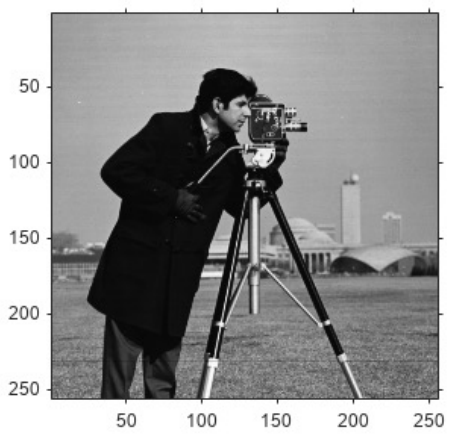
Simple interest operators

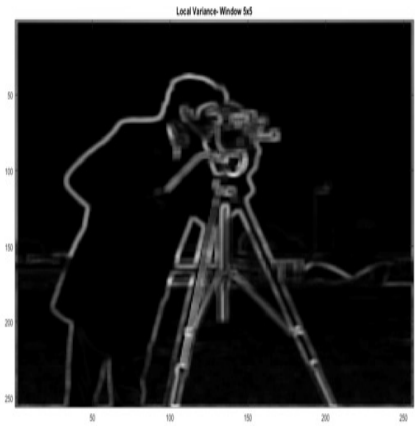
- How to find them?
- We use criteria within an interest window to see how “interesting” the center pixel is

- A first try could be the local variance (in a M x M window)

Not good,

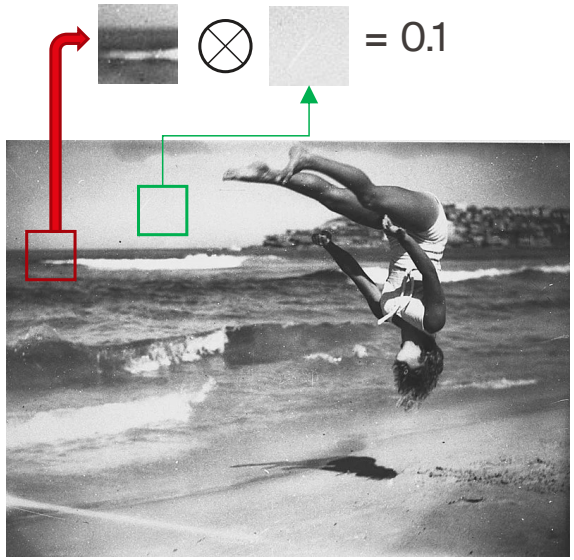
- No geometric meaning
- All edges would seem interesting





Self-similarity and local values comparison criteria

- We could look at patch similarity across the image, i.e. the cross-correlation function



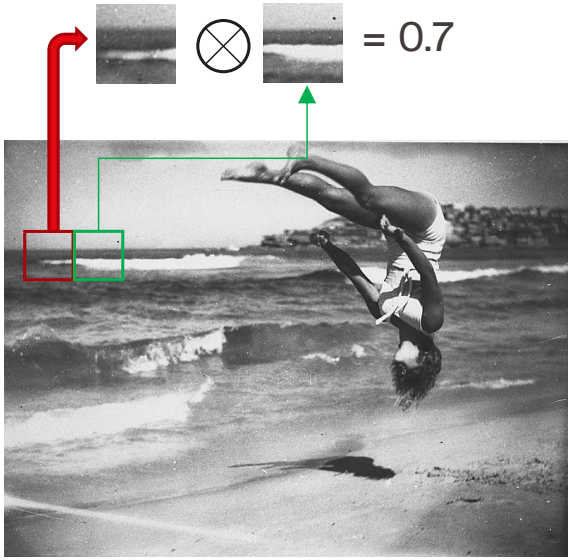
[What is the cross-correlation function?]

take each patch and slide it over the image. Every time, you calculate the similarity with the area it is superimposed to:

- if the patch is unique, you will have only one maximum (patch with itself, max similarity),
- if it repeats across image, there will be multiple areas with high similarity, so not interesting.

Self-similarity and local values comparison criteria

- We could look at patch similarity across the image, i.e. the cross-correlation function



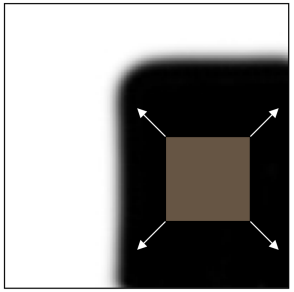
[What is the cross-correlation function?]

take each patch and slide it over the image. Every time, you calculate the similarity with the area it is superimposed to:

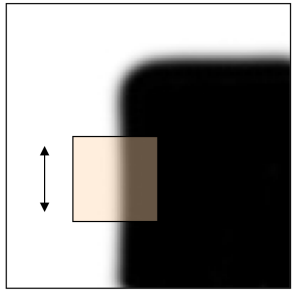
- if the patch is unique, you will have only one maximum (patch with itself, max similarity),
- if it repeats across image, there will be multiple areas with high similarity, so not interesting.

The Harris detector (1)

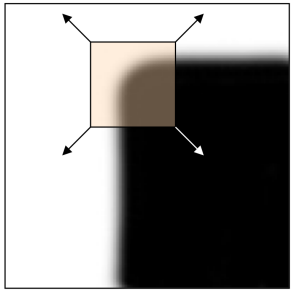
- Now let's take this idea of comparing patches, but locally.
- Moving a patch around could give us a good impression of whether we are in a corner region



"flat" region: no change in all directions



"edge": no change along the edge direction



"corner": significant change in all directions

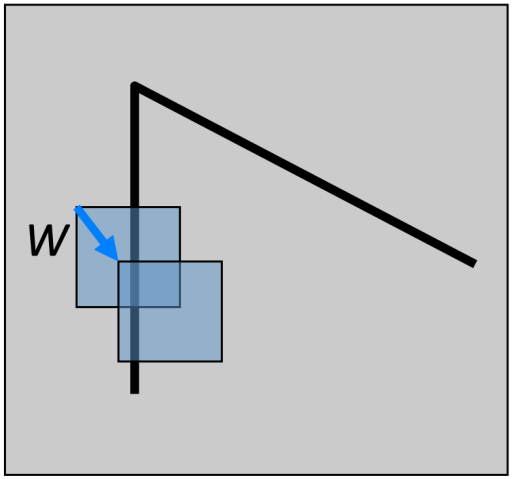
rce: A. Efros

Strongly inspired by: https://www.cs.cornell.edu/courses/cs4670/2016sp/lectures/lec10_features2_web.pdf

The Harris detector (2)

- Now let's take this idea of comparing patches, but locally.
- Moving a patch around could give us a good impression of whether we are in a corner region
- We can calculate the sum of squared differences for a location (x,y) after shifting by w = (u,v):

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$



The math behind (for your curiosity)

- We want to assess E(u,v) for small motions

Corners and edges are 1st order gradients

Taylor Series expansion of *I*:

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approximation is good

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

$$\approx I(x,y) + \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

shorthand: $I_x = \frac{\partial I}{\partial x}$

This is a Sobel filter, for example

The math behind (for your curiosity)

- Coming back to the original equation:

$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2 \\
&\approx \sum_{(x,y)\in W} [\cancel{I(x,y)} + I_x u + I_y v - \cancel{I(x,y)}]^2 \\
&\approx \sum_{(x,y)\in W} [I_x u + I_y v]^2
\end{aligned}
$$

The math behind (for your curiosity)

$$E(u,v) \approx \sum_{(x,y)\in W} [I_x u + I_y v]^2$$

$$\approx Au^2 + 2Buv + Cv^2$$

$$A = \sum_{(x,y)\in W} I_x^2 \qquad B = \sum_{(x,y)\in W} I_x I_y \qquad C = \sum_{(x,y)\in W} I_y^2$$

- Thus, *E*(*u*,*v*) is locally approximated as a quadratic error function

The Harris detector (3)

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$

- Using Taylor expansion, this is equivalent to

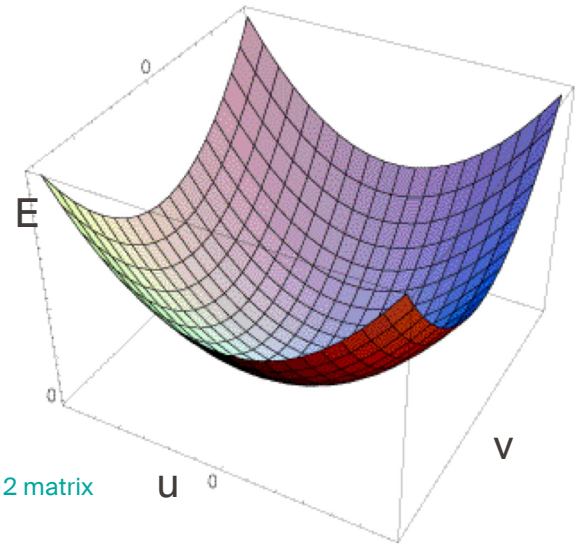
$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

For a single pixel, 2 x 2 matrix

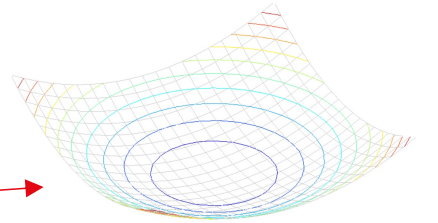
$w_{x,y}$

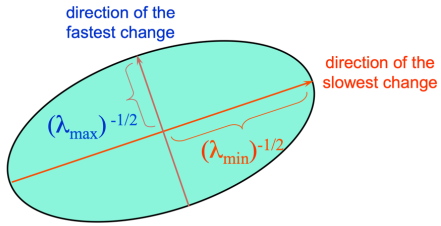
Often a Gaussian weight
= distance from central pixel



The Harris detector (4)

- E(u,v) = *const* is locally quadratic
- It can be approximated by an ellipse, a.k.a each color ring in
- An ellipse can be decomposed into its
  - eigenvectors (x) : orientation vectors
  - eigenvalues (λ) : length of the axis



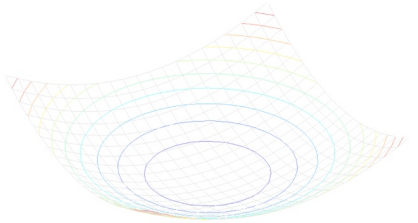


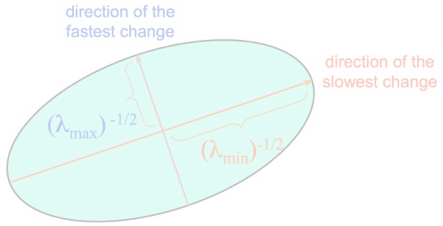
The Harris detector (4)

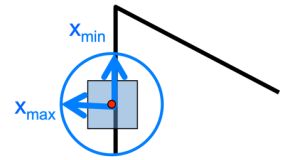
- E(u,v) = *const* is locally quadratic
- It can be approximated by an ellypse, a.k.a each color ring in
- An ellipse can be decomposed into its eigenvectors (x) and eigenvalues (λ)
- Since [u,v] are basis vectors of the space (=the *xy axis*), M is the ellipse.

$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

- The axes of the ellipse are thus the eigenvectors of M and the scale of the axes the eigenvalues







$$M\, x_{max} = \lambda_{max} x_{max}$$

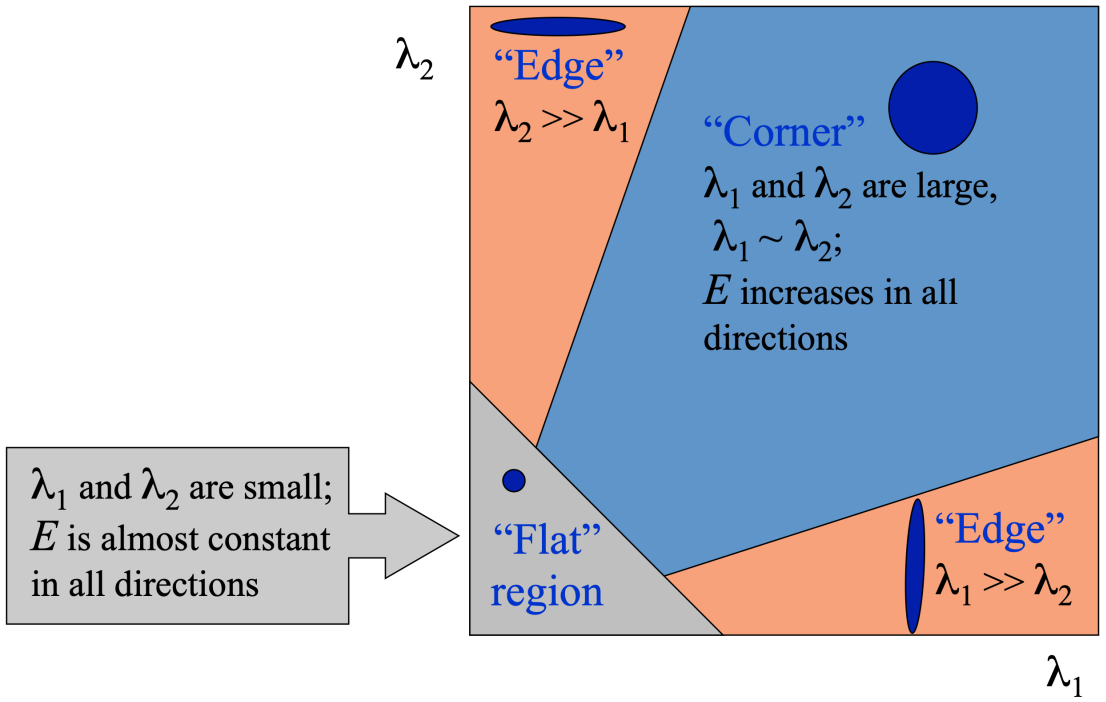
$$M\, x_{min} = \lambda_{min} x_{min}$$

Eigenvalues and eigenvectors of M

- Define shift directions with the smallest and largest change in error
- x_{max} = direction of largest increase in *E*
- λ_{max} = amount of increase in direction x_{max}
- x_{min} = direction of smallest increase in *E*
- λ_{min} = amount of increase in direction x_{min}

The Harris detector (4)

- Using the two eigenvalues, we can characterise the "edge-iness"



The Harris detector (5)

Final touch,

In 1988, Harris and Stephens merged all this into a single index, called the Harris detector

It uses the ingredients above to create a unified, simple to compute index

Given

- $\det(M) = \lambda_1 \lambda_2$
- $\text{trace}(M) = \lambda_1 + \lambda_2$
- λ_1 and λ_2 are the eigenvalues of M

The Harris detector is:

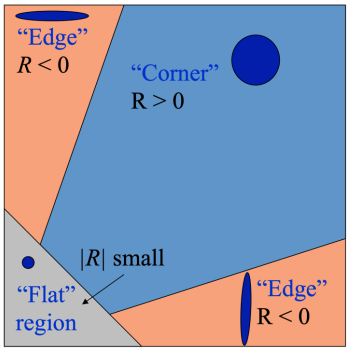
$$R = \det M - \alpha trace(M)^2 = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

The Harris detector (5)

So the magnitudes of these eigenvalues decide whether a region is a corner, an edge, or flat.

- When $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
- When $R < 0$, which happens when $\lambda_1 >> \lambda_2$ or vice versa, the region is edge.
- When R is large, which happens when λ_1 and λ_2 are large and $\lambda_1 \sim \lambda_2$, the region is a corner.

α: constant (0.04 to 0.15)



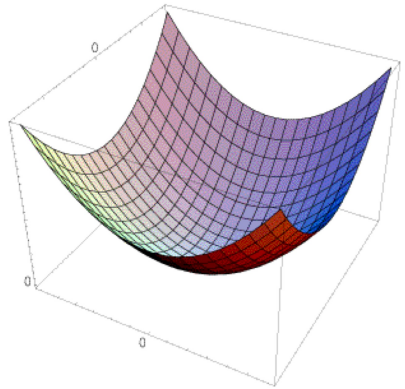
$$R = \det M - \alpha trace(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

You can compute the Harris detector fast!

- So far, we saw the math pixel by pixel
- Each pixel gave us a M matrix of size (2 x 2) that you can decompose in two eigenvalues

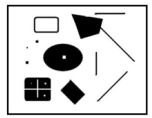
$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



You can compute the Harris detector fast!

- So far, we saw the math pixel by pixel
- Each pixel gave us a M matrix of size (2 x 2) that you can decompose in two eigenvalues
- If you consider the whole image
-

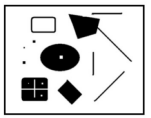


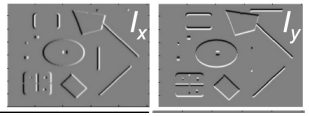
You can compute the Harris detector fast!

- So far, we saw the math pixel by pixel
- Each pixel gave us a M matrix of size (2 x 2) that you can decompose in two eigenvalues
- If you consider the whole image, M is of size (2 x height_image, 2 x width_image)
- This matrix is made of the gradient images,

$$\mu(\sigma_I,\sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

1. Image derivatives (optionally, blur first)

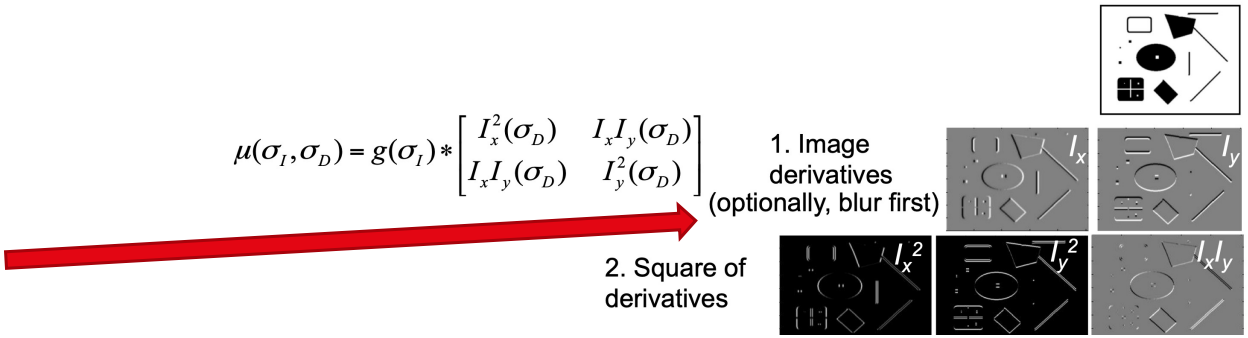




You can compute the Harris detector fast!

- So far, we saw the math pixel by pixel
- Each pixel gave us a M matrix of size (2 x 2) that you can decompose in two eigenvalues
- If you consider the whole image, M is of size (2 x height_image, 2 x width_image)
- This matrix is made of the gradient images,

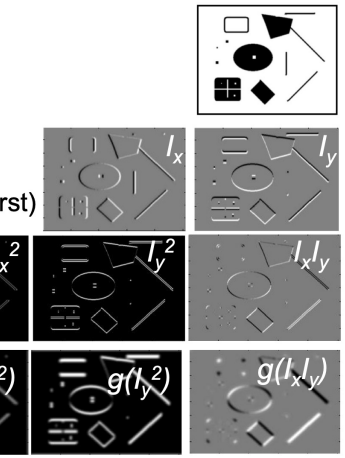
$$\mu(\sigma_I, \sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$



You can compute the Harris detector fast!

- So far, we saw the math pixel by pixel
- Each pixel gave us a M matrix of size (2 x 2) that you can decompose in two eigenvalues
- If you consider the whole image, M is of size (2 x height_image, 2 x width_image)
- This matrix is made of the gradient images, multiplied by blur kernels g()

$$\mu(\sigma_I, \sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$



You can compute the Harris detector fast!

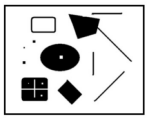
- So far, we saw the math pixel by pixel
- Each pixel gave us a M matrix of size (2 x 2) that you can decompose in two eigenvalues
- If you consider the whole image, M is (2*height_image, 2*width_image)
- This matrix is made of the gradient images, multiplied by blur kernels g()
- The Harris detector can be implemented without the need for eigen decomposition, only the determinant and trace of M!

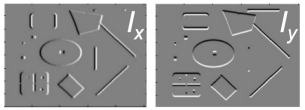
$$trace\begin{bmatrix} A & B \\ C & D \end{bmatrix} = A + D$$

$$\det\begin{bmatrix} A & B \\ C & D \end{bmatrix} = AD - BC$$

$$\mu(\sigma_I,\sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_xI_y(\sigma_D) \\ I_xI_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

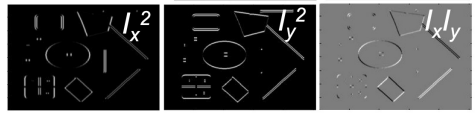
1. Image derivatives (optionally, blur first)





$$\det M = \lambda_1\lambda_2$$
$$\text{trace}\, M = \lambda_1 + \lambda_2$$

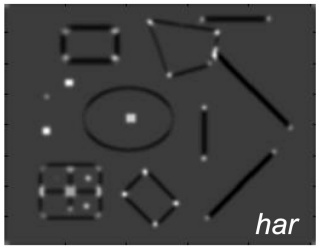
2. Square of derivatives



3. Gaussian filter $g(\sigma_I)$

4. Cornerness function – both eigenvalues are strong

$$har = \det[\mu(\sigma_I,\sigma_D)] - \alpha[\text{trace}(\mu(\sigma_I,\sigma_D))^2] =$$
$$g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$



5. Non-maxima suppression

You can compute the Harris detector fast!

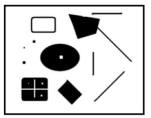
- So far, we saw the math pixel by pixel
- Each pixel gave us a M matrix of size (2 x 2) that you can decompose in two eigenvalues
- If you consider the whole image, M is (2 x height_image, 2 x width_image)
- This matrix is made of the gradient images, multiplied by blur kernels g()
- The Harris detectorcan be implemented without the need for eigen decomposition, only the determinant and trace of M!

$$trace\begin{bmatrix} A & B \\ C & D \end{bmatrix} = A + D$$

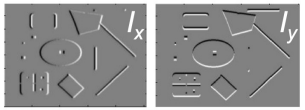
$$\det\begin{bmatrix} A & B \\ C & D \end{bmatrix} = AD - BC$$

- So you can compute the Harris detector for the whole image in one go by summing/substracting the gradient images

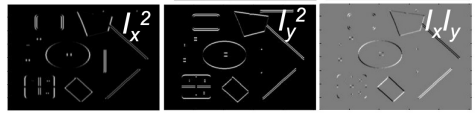
$$\mu(\sigma_I,\sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_xI_y(\sigma_D) \\ I_xI_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$



1. Image derivatives (optionally, blur first)



2. Square of derivatives

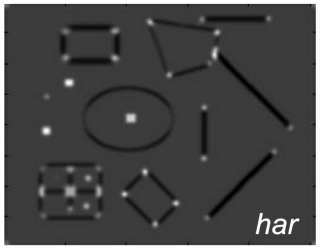


$$\det M = \lambda_1\lambda_2$$
$$\text{trace}\, M = \lambda_1 + \lambda_2$$

3. Gaussian filter $g(\sigma_I)$

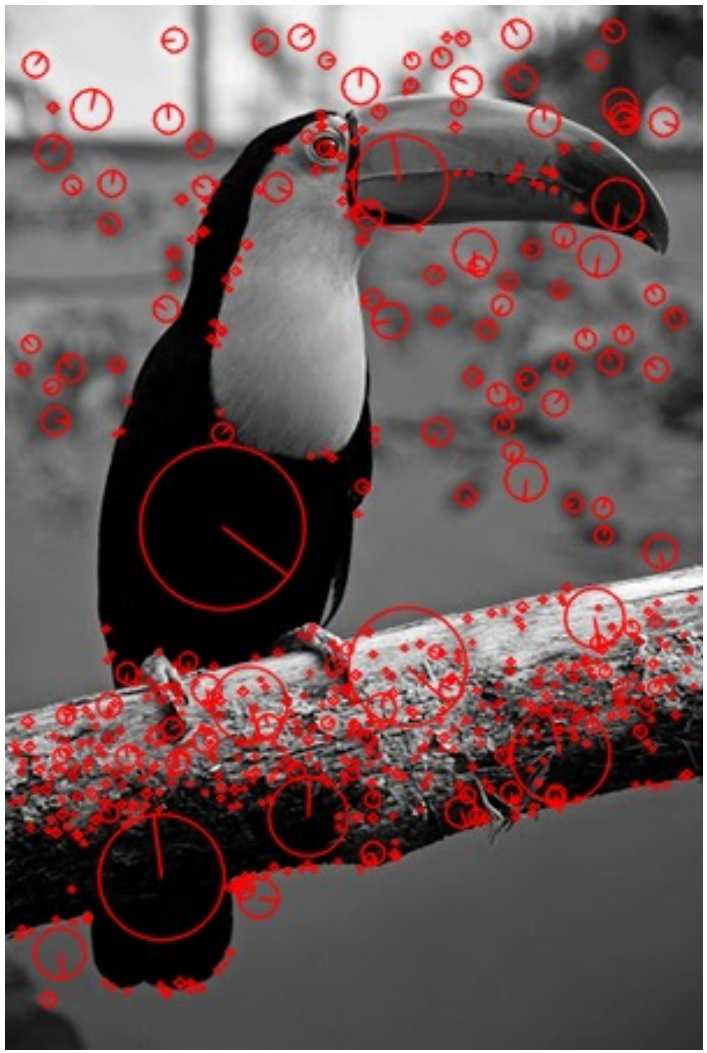
4. Cornerness function – both eigenvalues are strong

$$har = \det[\mu(\sigma_I,\sigma_D)] - \alpha[\text{trace}(\mu(\sigma_I,\sigma_D))^2] =$$
$$g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$



5. Non-maxima suppression

2. Compute a descriptor of the neighborhood of the keypoint



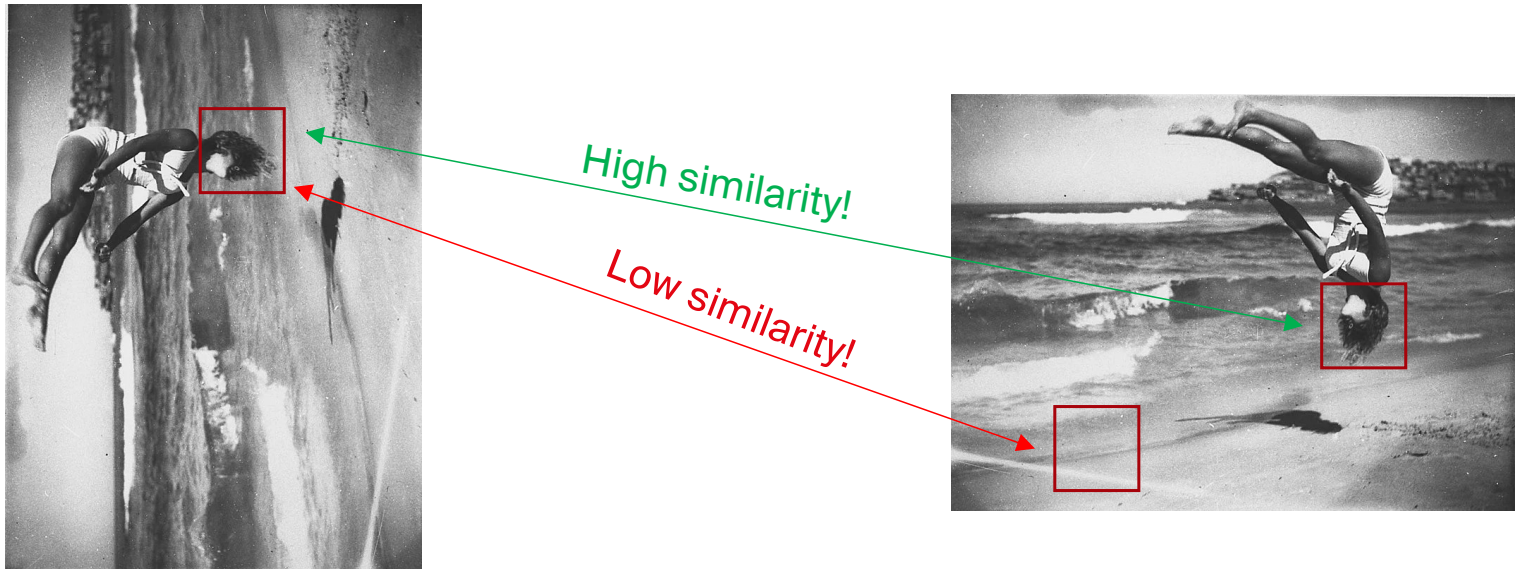
https://www.pexels.com/photo/black-and-green-toucan-on-tree-branch-17811/
https://medium.com/@vad710/cv-for-busy-developers-describing-features-49530f372fbb

Now that we have corners, what?

- We want to describe the content of the image at that location
- Taking the example of the beginning they could be textual descriptions (“a statue”, “a red flag”) that could help us matching corresponding objects in different images
- In image matching, we use image features, e.g. grayscale gradients, particular patterns, etc.
- We need to find features that are
  - Unique (so we can find a clear match in the other image)
  - Invariant to scaling and rotation
  - Invariant to color deformations (atmospheric conditions, illumination, etc)

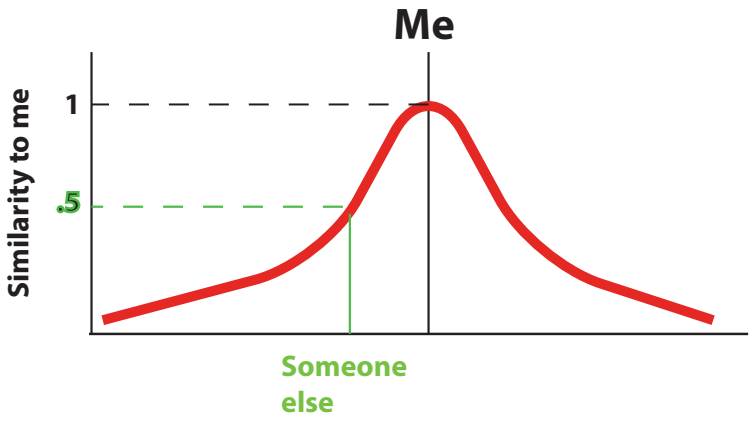
Why invariance?

- We want patches centered on the same keypoint of different images to have high similarity between each other
- We need to find the right function of the pixel values, a descriptor
- What we want:

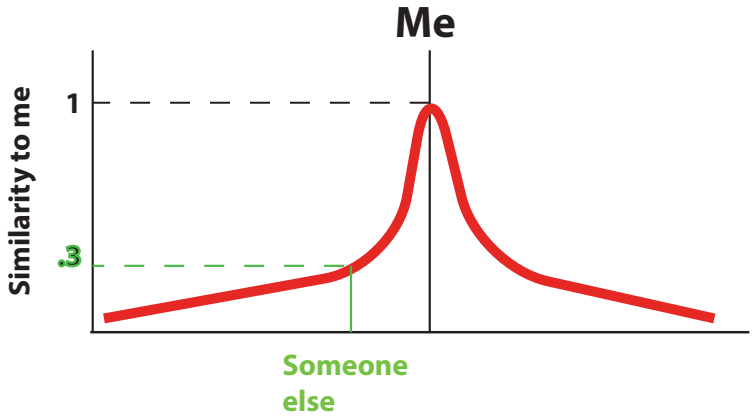


Why invariance?

- We want patches centered on the same keypoint of different images to have high similarity between each other
- Similarity is a measure of feature distance between the patches being considered. For instance a Gaussian distance, an Euclidean, ...



Large bandwidth (large gamma)



Small bandwidth (small gamma)

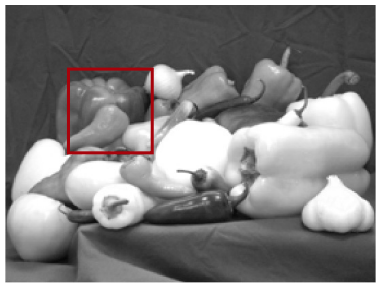
me = me

se = someone else

Why invariance?

- We want patches centered on the same keypoint of different images to have high similarity between each other
- Similarity is a measure of feature distance between the patches being considered. For instance a Gaussian distance, an Euclidean, ...
- But we have images that are distorted, at different scales, etc! So the features we use to describe the patch are important!

- Example:
  - If we take the grayvalues directly as features and we rotate the image, the same patch will have very low similarity
  - If we sort the values, the feature is rotation invariant

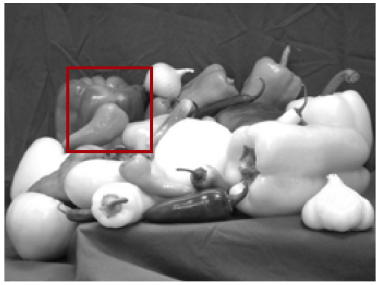


Original

-90°

Pixel by pixel Euclidean distance: 507.28!!
Sorted values pixel by pixel distance: 0

But a description should be invariant AND discriminative

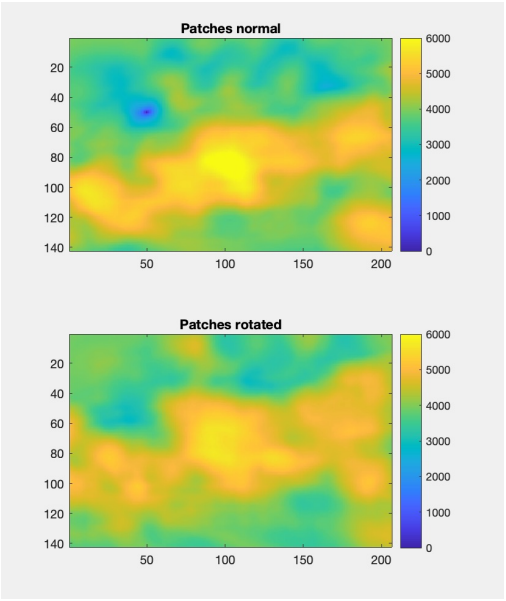


- If we sort, our distance becomes rotation invariant (the two maps in the right column are the same)

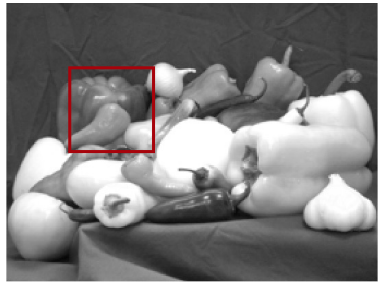
- But we have less discrimination, since we destroyed all image structure
  - many more patches look similar (the right column is more blue)
  - The original distance map (top left) is better (it just doesn't work as soon as we have differences, bottom left)

Comparing original pixel values, no rotation invariance

Comparing sorted pixel values, rotation invariance, but poor discrimination



But a description should be invariant AND discriminative

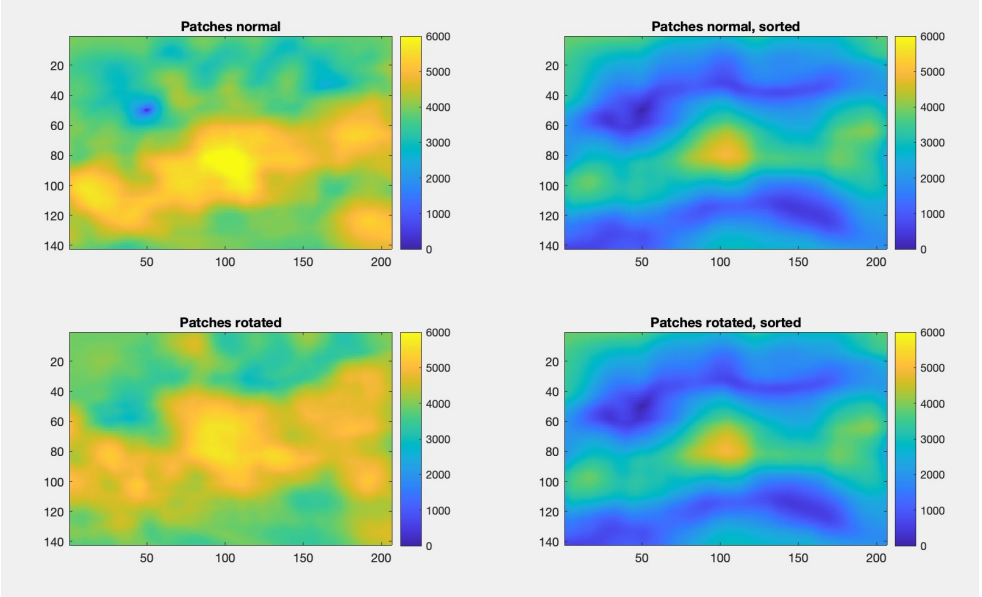


- If we sort, our distance becomes rotation invariant (the two maps in the right column are the same)

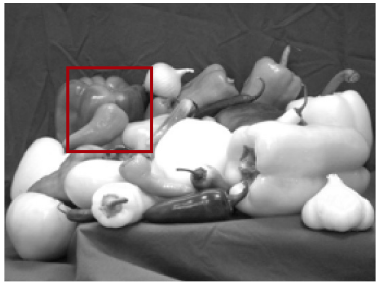
- But we have less discrimination, since we destroyed all image structure
  - many more patches look similar (the right column is more blue)
  - The original distance map (top left) is better (it just doesn't work as soon as we have differences, bottom left)

Comparing original pixel values, no rotation invariance

Comparing sorted pixel values, rotation invariance, but poor discrimination



But a description should be invariant AND discriminative

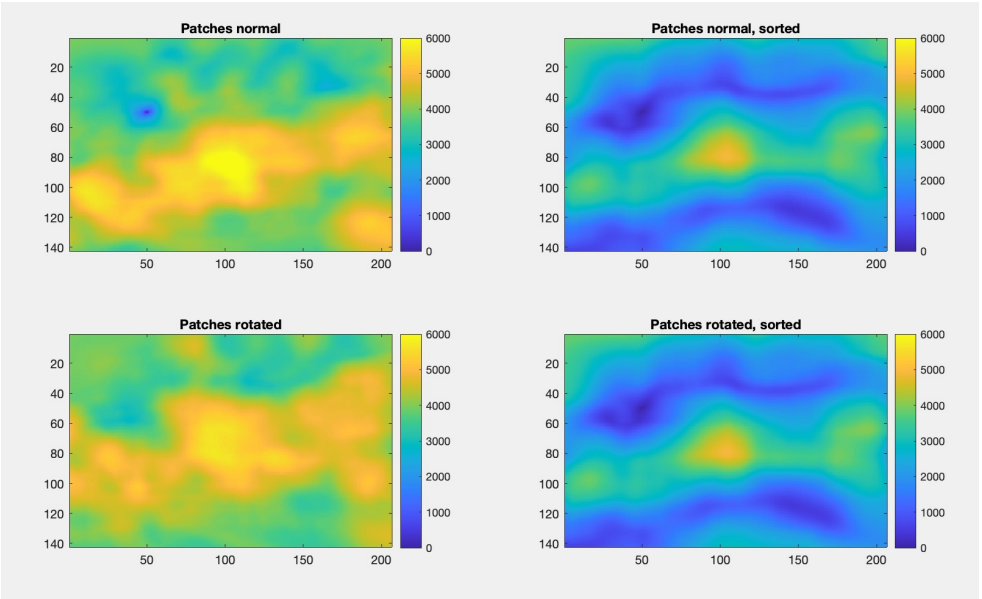


- If we sort, our distance becomes rotation invariant (the two maps in the right column are the same)

- But we have less discrimination, since we destroyed all image structure
  - many more patches look similar (the right column is more blue)
  - The original distance map (top left) is better (it just doesn't work as soon as we have differences, bottom left)

Comparing original pixel values, no rotation invariance

Comparing sorted pixel values, rotation invariance, but poor discrimination

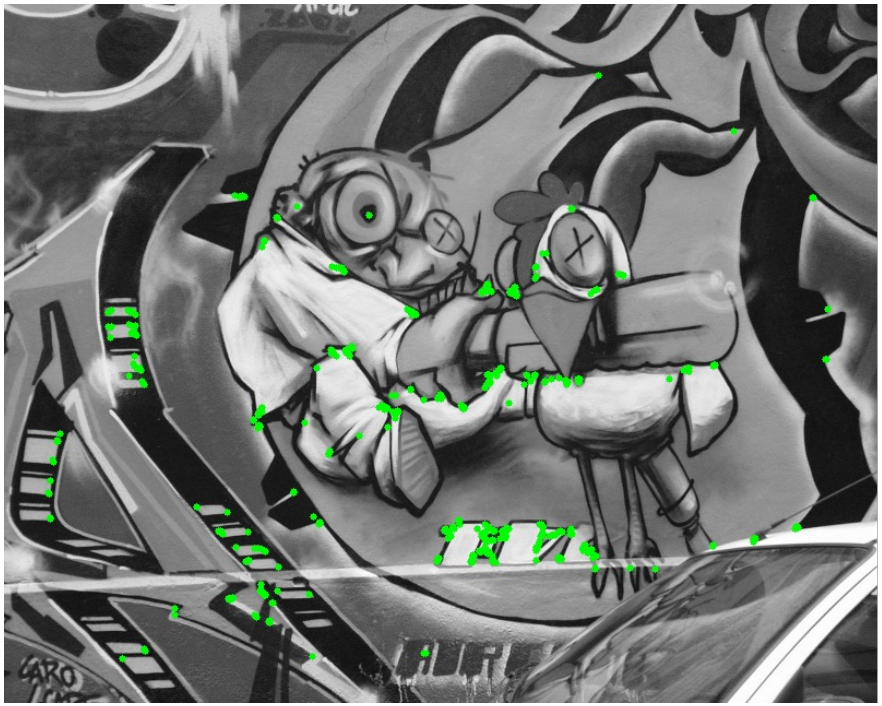


A good feature descriptor works across scales and orientations: SIFT

- A very popular one is the Scale-Invariant Feature Transform (SIFT)
- Invented by David Lowe in 1999
- Still probably the most used one around
- Was patented until 2020.
- Emulated implementations are around.

- SIFT includes two steps:
  - keypoint detection (you can use your own, e.g. the Harris detector)
  - keypoint description = extract meaningful features
- We will go through both, since the detection is crucial for scale invariance

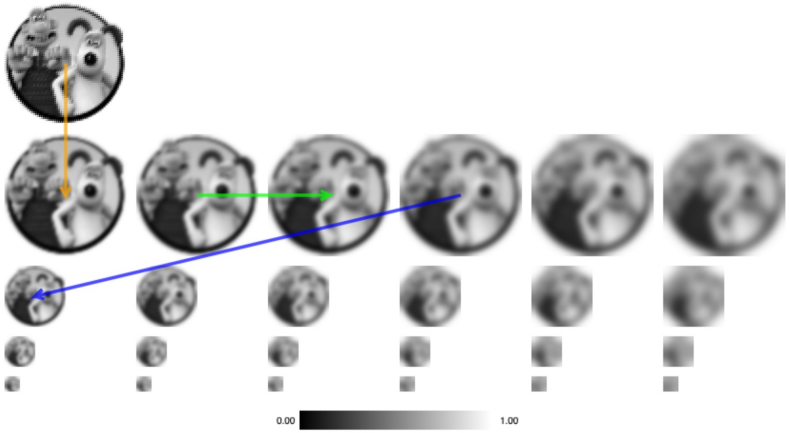
1.bis
Back to detecting keypoints: SIFT



https://towardsdatascience.com/improving-your-image-matching-results-by-14-with-one-line-of-code-b72ae9ca2b73

SIFT, phase 1 Detection

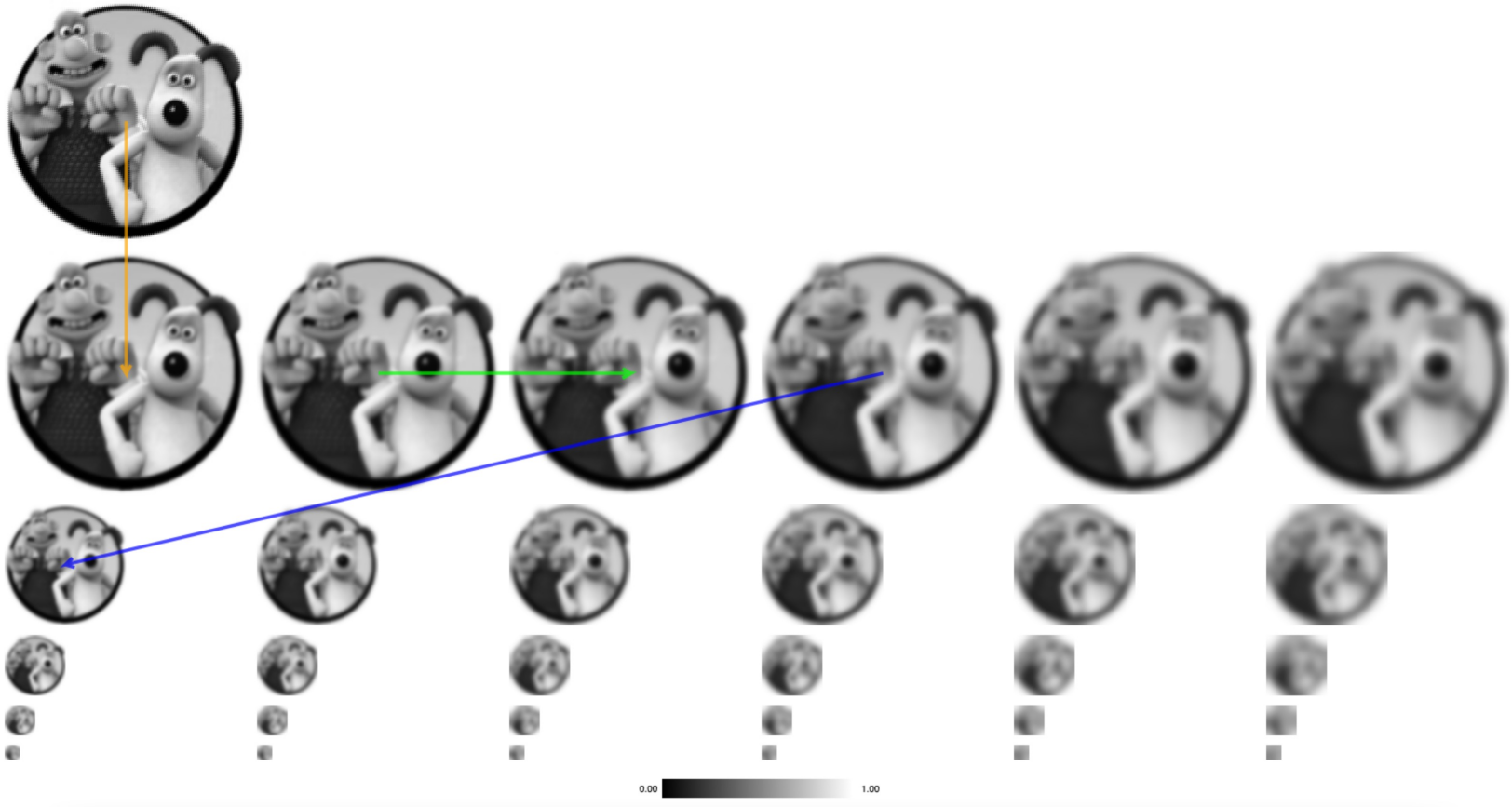
- Let's say we want to find good keypoints for this image
- SIFT first creates a pyramid of images at different levels of blurring
- We use Gaussian convolutions (green arrow)
- After some convolutions (here 3) we downsample and start over (blue arrow)
- Each row of 6 is called an *octave (it doubles the blur in the image)*
- All images are [0-1] normalised.



http://weitz.de/sift/index.html

SIFT, phase 1 Detection

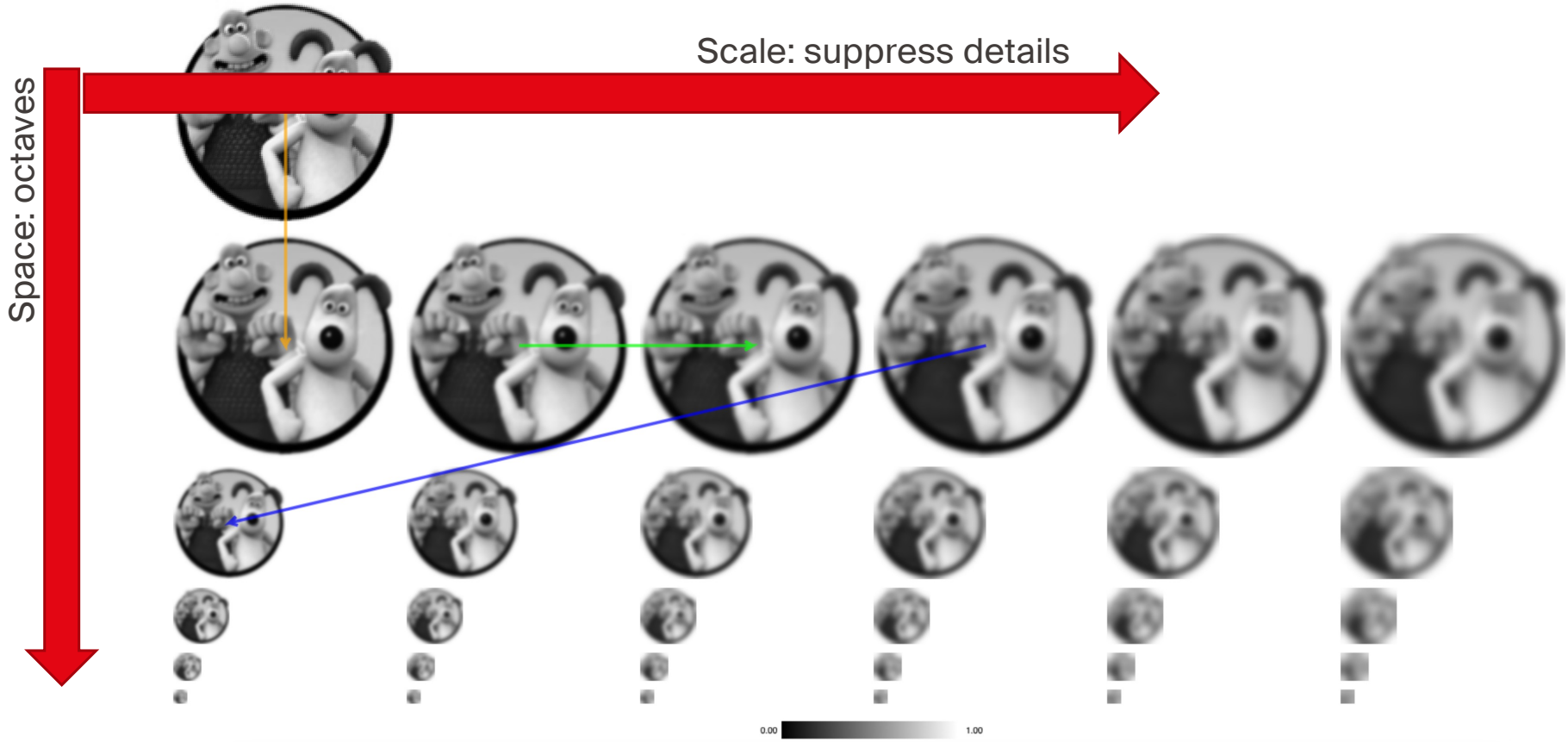
- By doing so, we built a scale-space



D. Tuia. ECEO

SIFT, phase 1 Detection

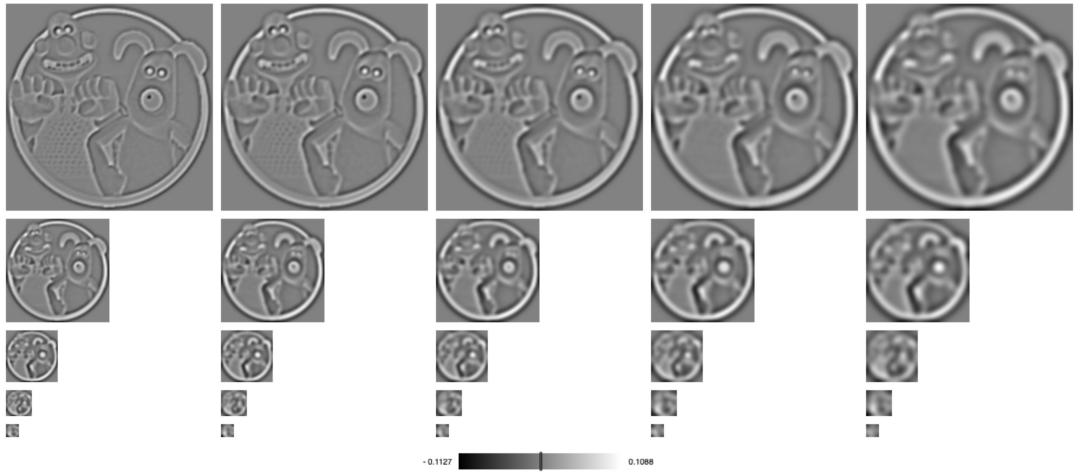
- By doing so, we built a scale-space



SIFT, phase 1 Detection

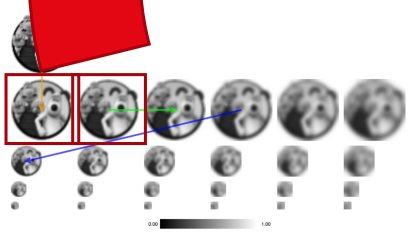
- To find the keypoints, we look for points that change a lot across scales and/or levels of detail
- To do so, we calculate the difference of Gaussians, the difference between the pixel values of the left and right images of the previous slide

(they were Gaussian blurred version of each octave, remember?)

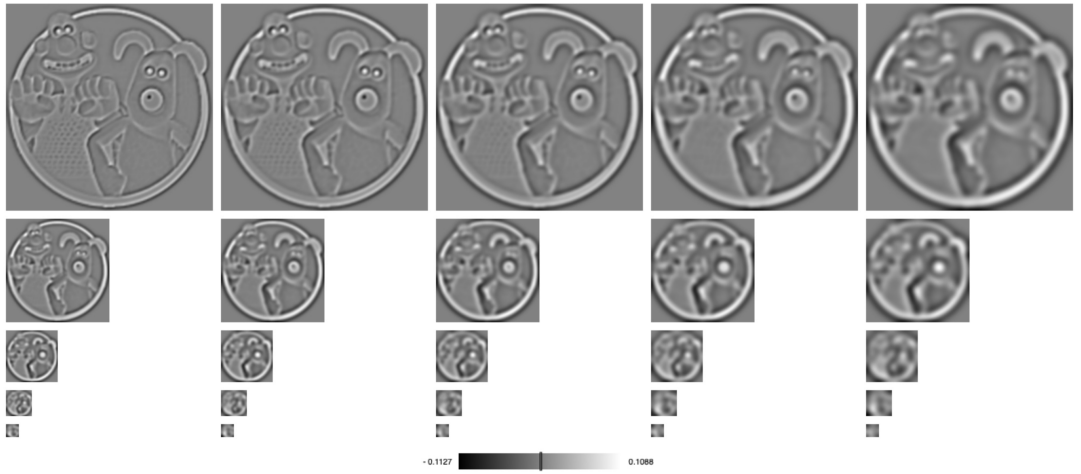


SIFT, phase 1 Detection

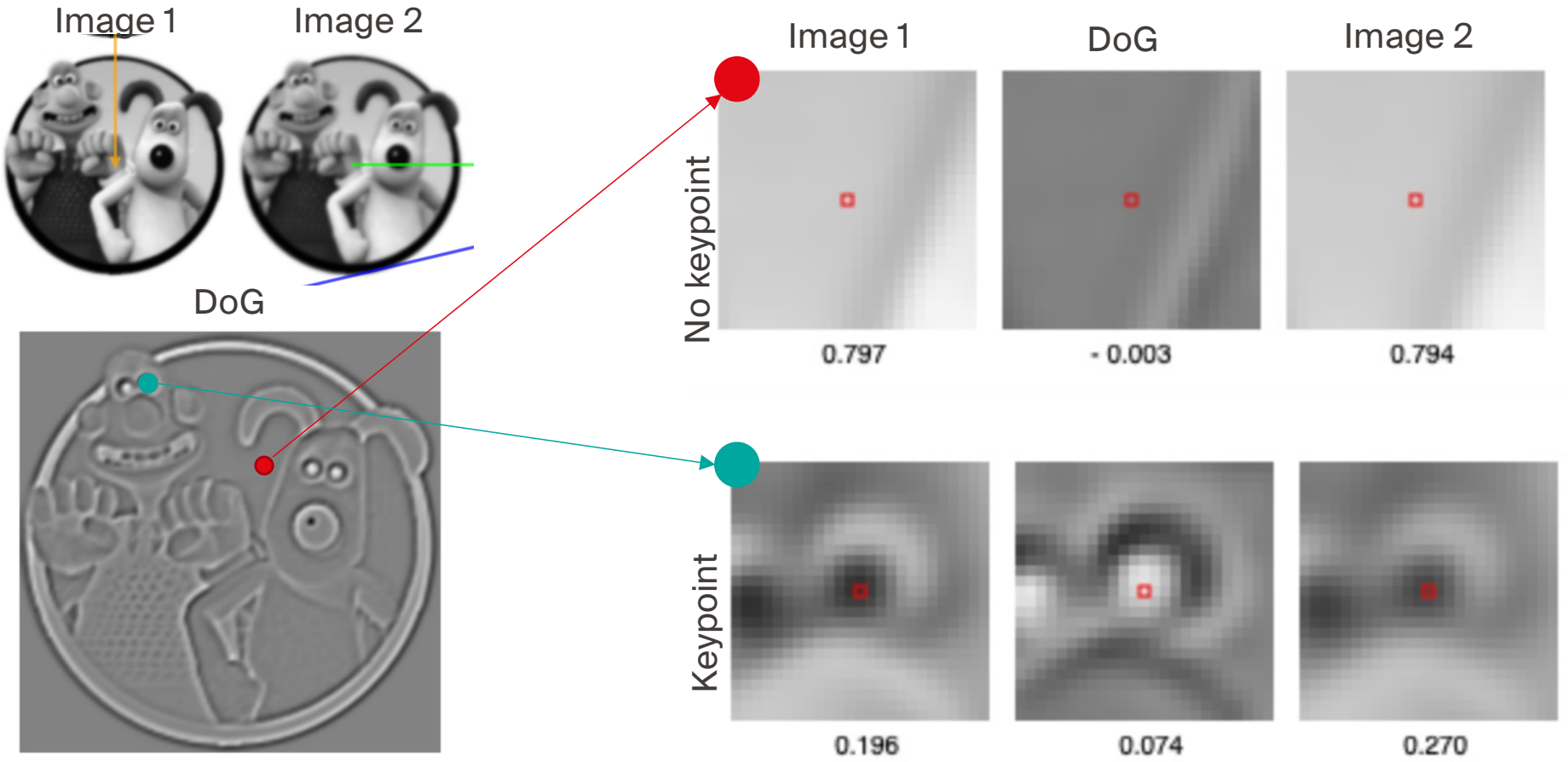
- To f… …e look for points that change a lot across … and/or levels … tail
- … do so, we calculate … ifference of Gaussians, the difference … etween the pixel value … e left and right images of the previous … ide



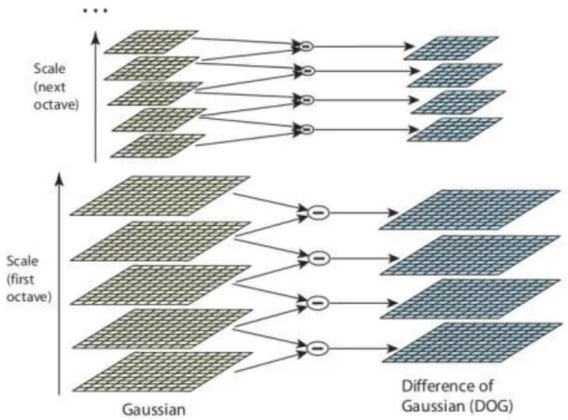
substraction



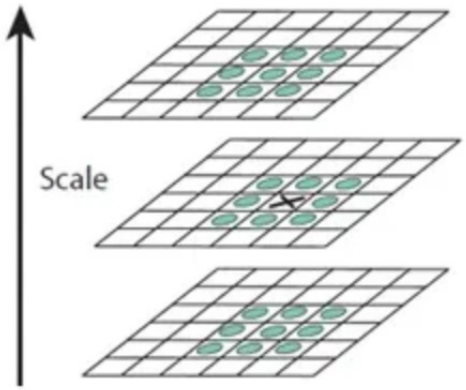
We detect interesting keypoints by looking at the DoG



We detect interesting keypoints by looking at the DoG

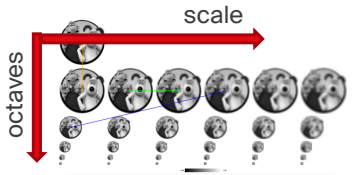


DoG for each octave / level of blur

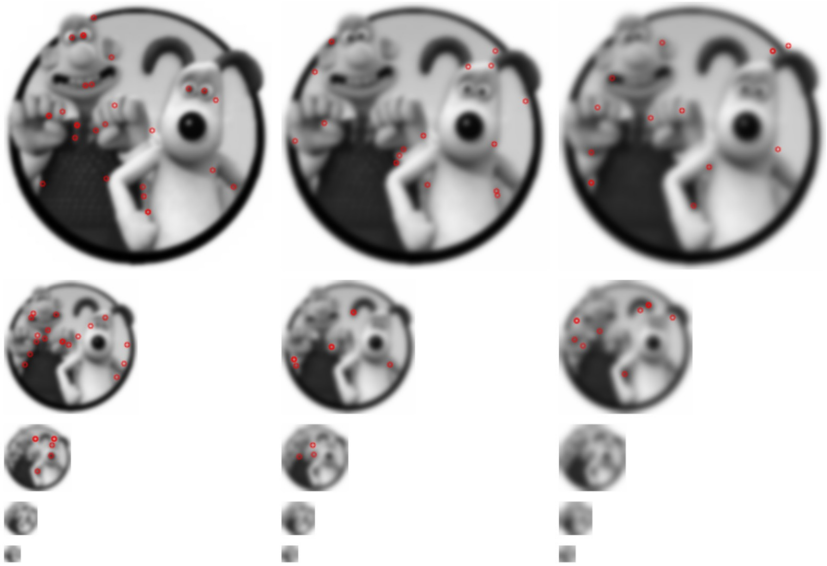


Each pixel in a DoG map gets compared to neighbors in the same map and the one at the smaller and larger level of blur.

- Keypoints are found by comparing the DoG values across nearby pixels, in scale space (we don't compare across octaves).
- We have 26 neighbors (green dots).



We detect interesting keypoints by looking at the DoG



PS: it's not so simple. there are several extra steps to remove spurious maxima (flat points, edges) looking at local curvature

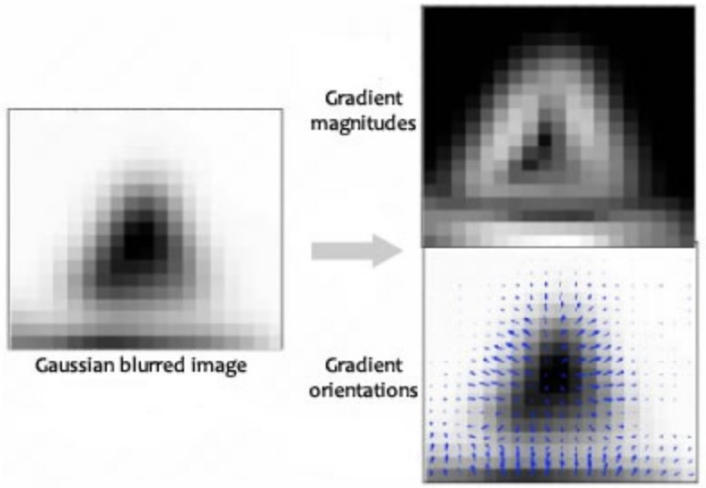
- Keypoints are found by comparing the DoG values across nearby pixels, in detail space (we don't compare across octaves).
- We have 26 neighbors (green dots on previous slide).
- A maximum (resp. minimum) is the point with DoG value larger (resp. smaller) than all its neighbors

Last thing: assign the main orientation

- For each detected point, we compute the main orientation of the gradients in the image.
- This will impact the features detected later on.
- Note: one keypoint can have more than one main direction (see next slides). In this case, it is used as multiple keypoints with different orientations (with same location and scale)

Last thing: assign the main orientation

- First we compute for every keypoint the magnitude (m) and orientation (θ) of the gradients of every pixel in the surroundings.



- We use the formulas below, where L is the Gaussian blurred image:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

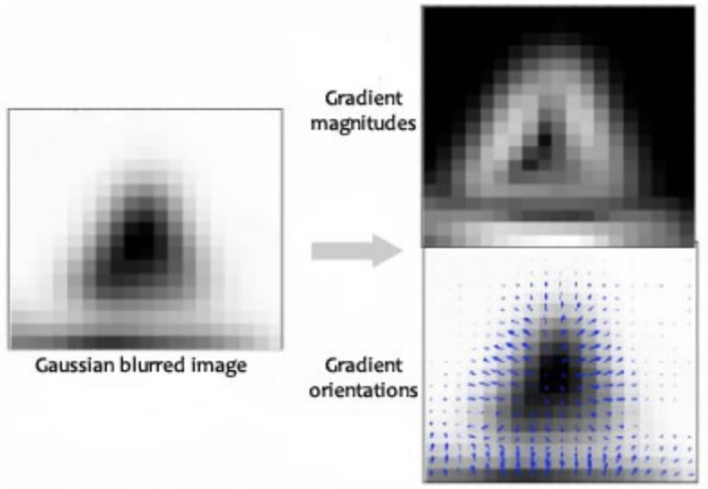
Last thing: assign the main orientation

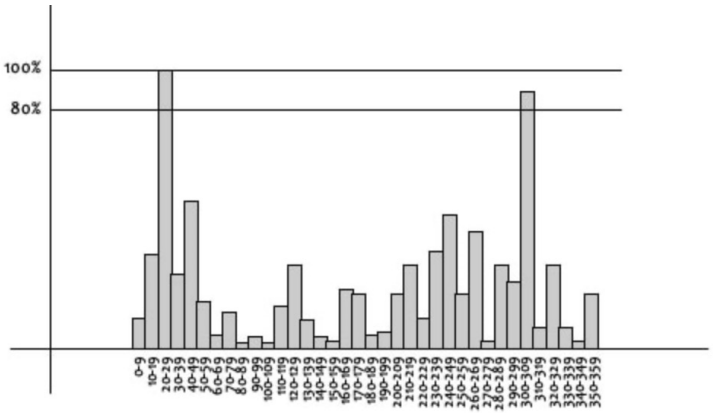
- First we compute for every keypoint the magnitude (m) and orientation (θ) of the gradients of every pixel in the surroundings.

- We make an histogram out of it.
  - We make 36 bins, each one accounting for 10°
  - We add in the specific bin a value proportional to the magnitude for each pixel.

  E.g. if a pixel has a 18.79° orientation and magnitude “10”, we add 10 in the 10°-20° bin.

- All orientations above 80% are considered a main direction.





2.bis
the descriptor of SIFT

https://www.pexels.com/photo/black-and-green-toucan-on-tree-branch-17811/
https://medium.com/@vad710/cv-for-busy-developers-describing-features-49530f372fbb

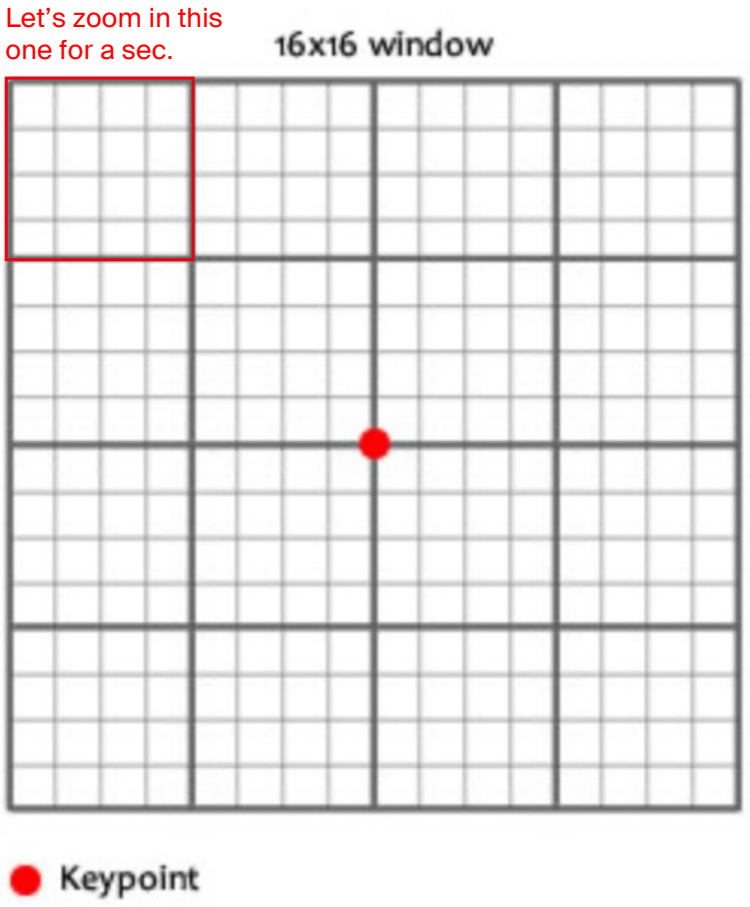
Ready to compute descriptors!

- Recall: the descriptors describe what is giong on in the direct surroundings of the keypoint.
- It's a kind of a fingerprint for each keypoint that we will use later to match.
- We need : the location, scale and orientation of each keypoint.

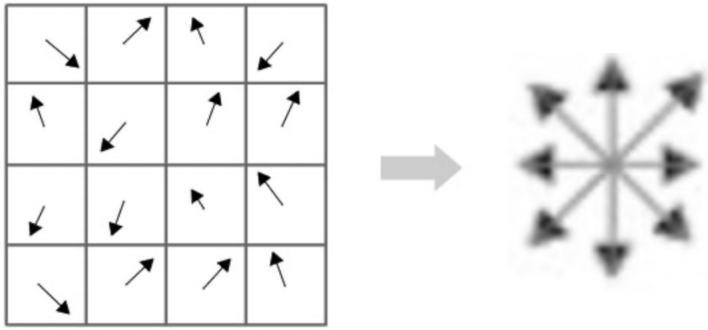
- It needs to be unique and easy to calculate.

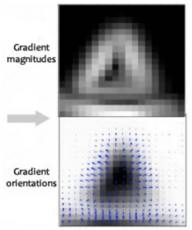
We take each keypoint

- We first break the neighborhood around the keypoint into a 16 x 16 window

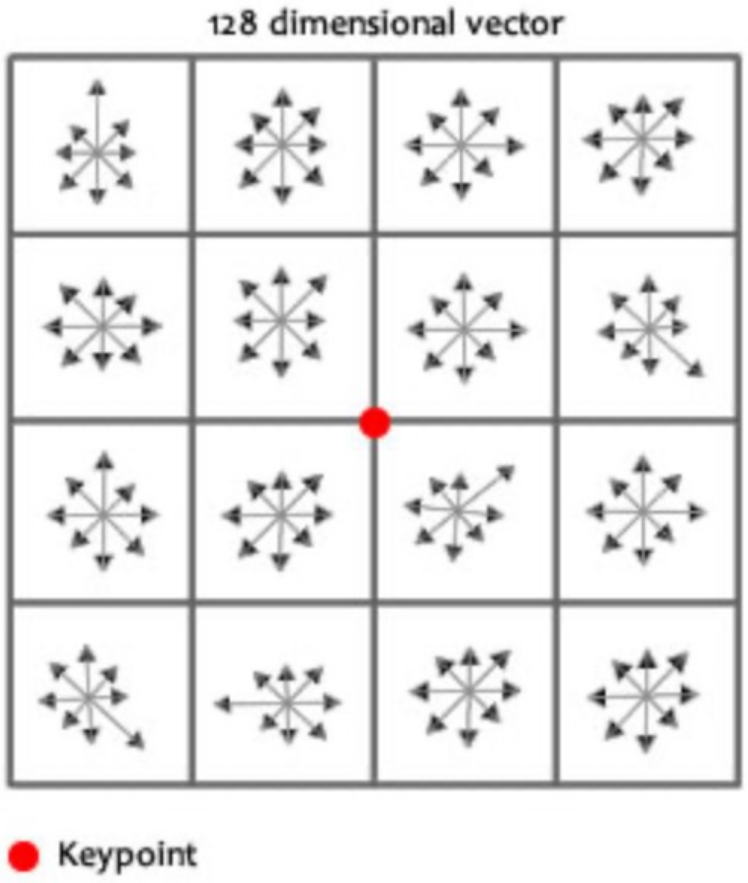


- We first break the neighborhood around the keypoint into a 16 x 16 window
- Within each 4 x 4 sub-window, a histogram of 8 bins (44° each) orientations and magnitudes recorded (like in the previous step!)

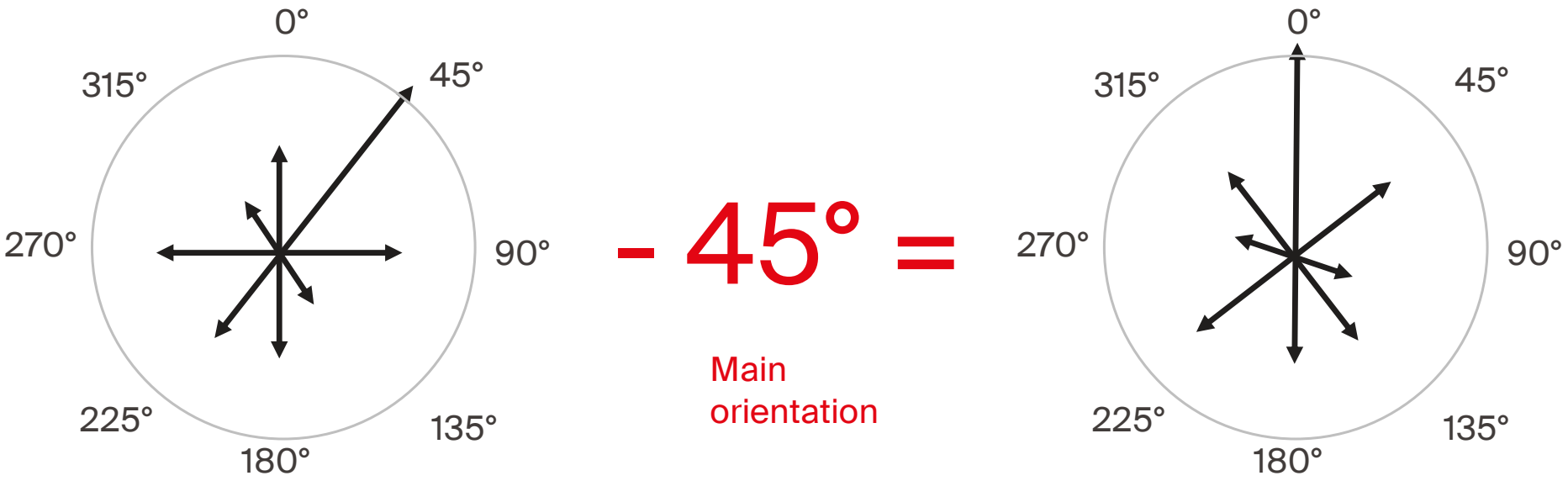




- We first break the neighborhood around the keypoint into a 16 x 16 window
- Within each 4 x 4 sub-window, a histogram of 8 bins (44° each) orientations and magnitudes recorded (like in the previous step!)
- To enforce rotation invariance, the keypoint's main orientation is substracted from the orientations

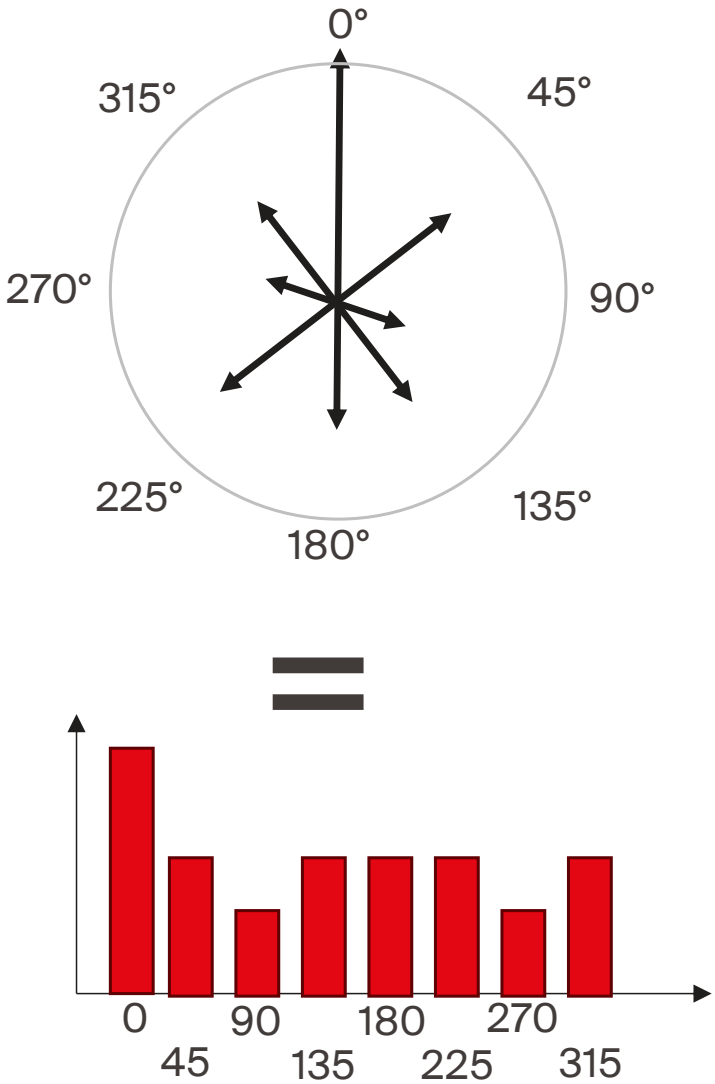


What is the effect of substracting the main orientation?



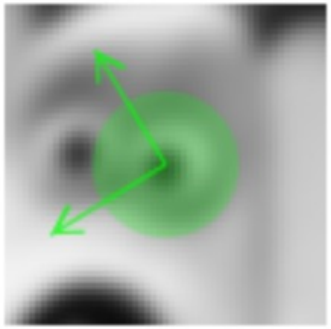
Visualising like histograms

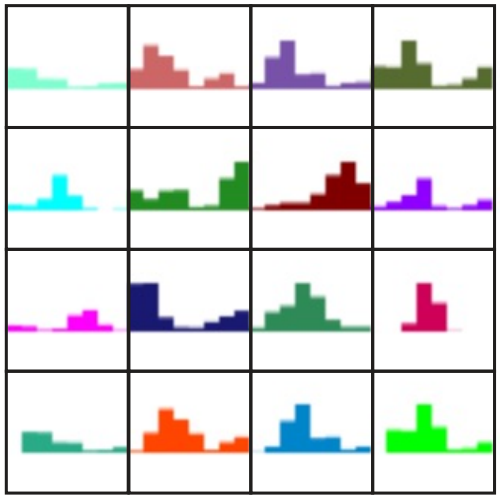
- We first break the neighborhood around the keypoint into a 16 x 16 window
- Within each 4 x 4 sub-window, a histogram of 8 bins (44° each) orientations and magnitudes recorded (like in the previous step!)
- To enforce rotation invariance, the keypoint's main orientation is substracted from the orientations
- Each radial plot corresponds to a histogram...



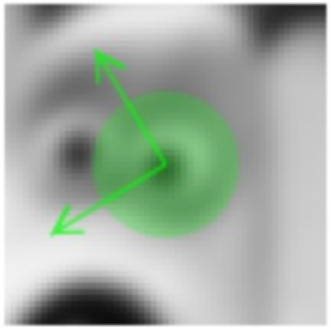
The SIFT descriptor

- We first break the neighborhood around the keypoint into a 16 x 16 window
- Within each 4 x 4 sub-window, a histogram of 8 bins (44° each) orientations and magnitudes recorded (like in the previous step!)
- Each bin of each histogram becomes a feature, so 8 x 16 =128 features!
- To achieve illumination invariance, all features with big numbers are clipped. E.g. everything larger than 20% is clipped to 20%.



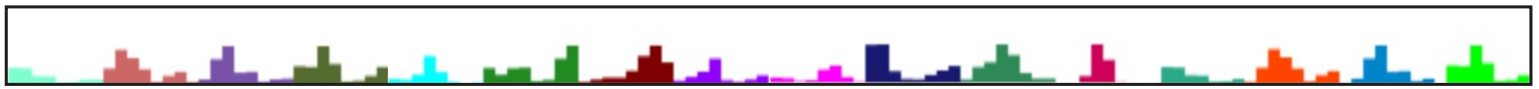


The SIFT descriptor



- There are more tweaks on the descriptor, which I omit here.
- In the end, the patch is described by the 128-dimensional vector, which has all the information about
  - location,
  - scale,
  - orientation and
  - magnitude gradients

  in the patch.

$SIFT_i =$ 

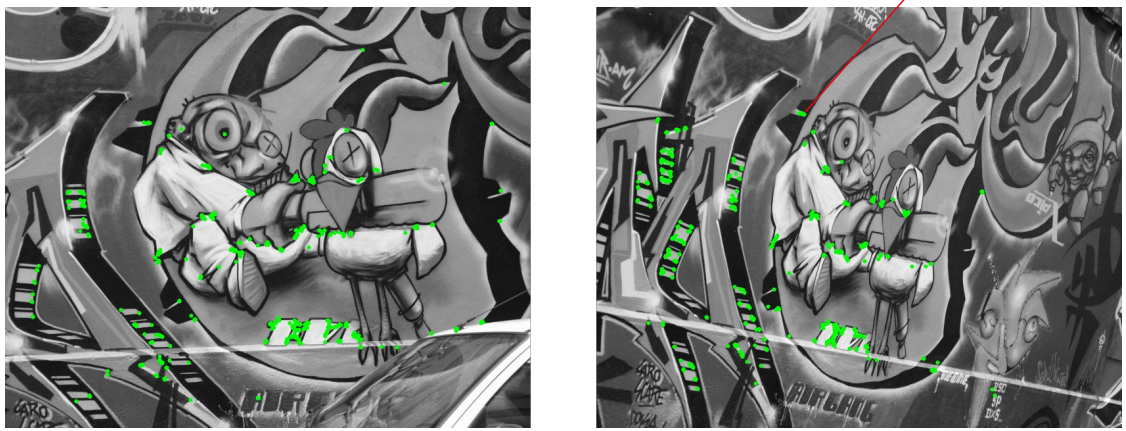
128

3. 4. Establish correspondences and make them robust with RANSAC

https://towardsdatascience.com/improving-your-image-matching-results-by-14-with-one-line-of-code-b72ae9ca2b73

Time to match!

- Now we have, for each image,
  - A set of keypoints
  - Each one with a 128-dimensional vector describing it.



https://towardsdatascience.com/improving-your-image-matching-results-by-14-with-one-line-of-code-b72ae9ca2b73

The simplest strategy to match is a "best match"

- Calculate distance between the SIFT features.
- Each keypoint from image "*I*" is matched with the one of image "*J*" showing the smallest distance in the SIFT features
- A keypoint *i* from the set k_I will be matched with the keypoint *j* from the set k_J as follows:

$$\arg\min_{j \in k_J} d(x_i^{SIFT}, x_j^{SIFT})$$

- $d(\cdot,\cdot)$ is a distance, e.g. Euclidean.

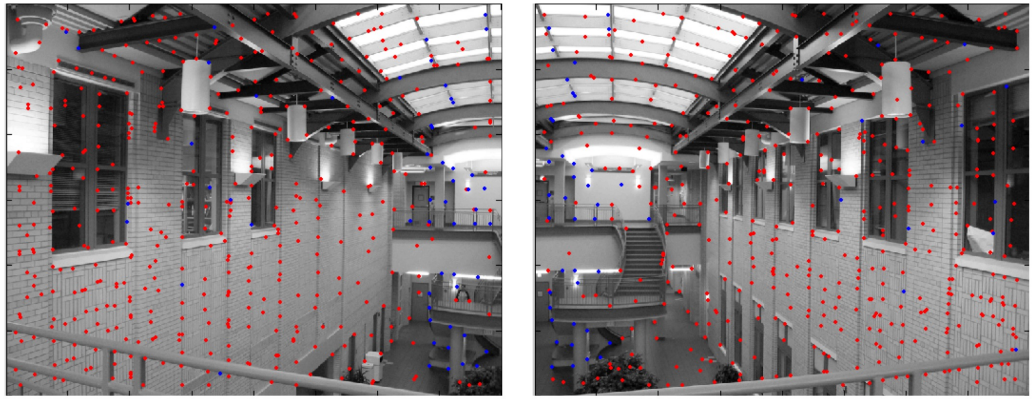
Sorting out for ex-aequos

- This strategy can work poorly if there are repetitive structure in images
- All these keypoints in J will score similarly
- A solution is to compare the best and 2nd match and keep the match only if the first leads to a much smaller distance than the second (= unique match).
- A division is what you need: if the result is close to 1, you discard the match. If it's small, you keep.

$$\frac{d_{1st}}{d_{2nd}}$$

Sorting out for ex-aequos

- Red: keypoints without a good match (comparing 1st and 2nd result)
- Blue: “good” matches, but including points matches to the wrong keypoint in the other image



https://people.cs.umass.edu/~elm/Teaching/ppt/370/370_10_RANSAC.pptx.pdf

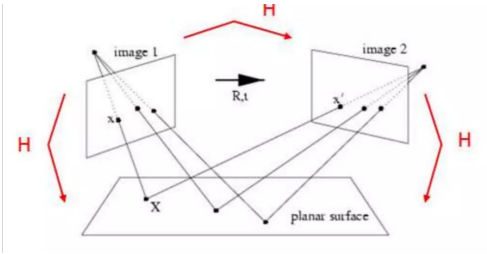
Sorting out for ex-aequos

- We can use an algorithm so sort out outliers, a favourites' choice is RANSAC

1. For
2. Select four feature pairs
3. Compute the homography H (the transform from one image plane to another)
4. Compute *inliers* (matches that lie within the distance of the homography)

$$d(x_j, Hx_i) < \epsilon$$

6. Keep the largest set of inliers
7. Re-compute the homography H with all inliers

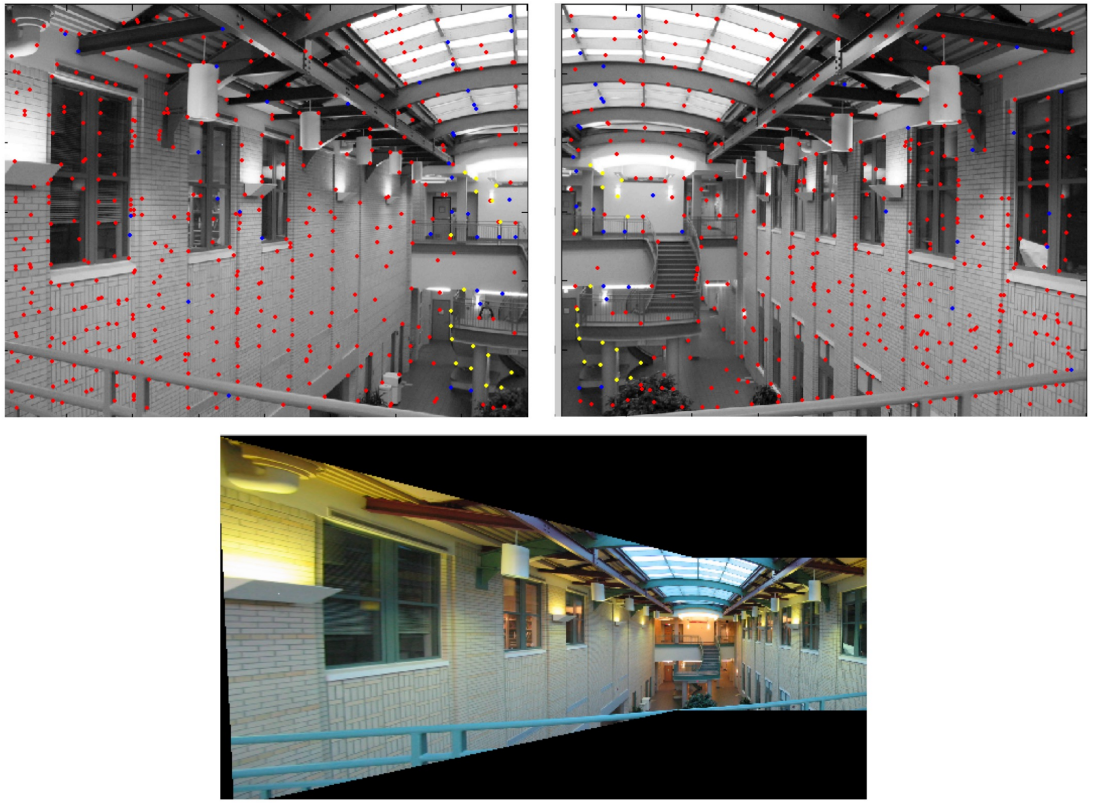


$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

https://people.cs.umass.edu/~elm/Teaching/ppt/370/370_10_RANSAC.pptx.pdf

Sorting out for ex-aequos

- Red: keypoints without a good match (comparing 1st and 2nd result)
- Yellow: correct good matches
- Blue: wrong “good” matches



Summing up

- Today we have learned about:
  - Why we need image matching
  - The steps of image matching
  - One simple corner detector: Harris
  - A complex keypoint detector + feature descriptor: SIFT
  - How to match the two images.

https://ducha-aiki.github.io/wide-baseline-stereo-blog/2020/03/27/intro.html. Photo and doll created by Olha Mishkina