

Computer Vision: Assignment 3

Feature Extraction and Image Stitching

Pablo Mesejo and Víctor Vargas

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA



Index

- Submission rules
- Assignment description

Index

- **Submission rules**
- Assignment description

Submission rules

- Only *.ipynb* files should be submitted (including code, results, analysis and discussion).
 - Don't upload the images!
- Don't write anything to disc/Drive.
- The template structure must be respected.

Submission

- Deadline: 31 December
- Maximum score: 10 points
- Submission Site: <https://pradogrado2324.ugr.es/>
- **Explanation/discussion accompanying code and results is essential.**

Goals

- Learn how to **detect** relevant **regions** (in this case, using the Harris corner detection algorithm and multiscale blob detection using the LoG)
- Learn how to **find correspondences between images using descriptors** (in this case, SIFT and Haralick features).
- Learn **to compose a mosaic/panorama** from a set of images
- This is an assignment oriented towards **the more geometrical part of computer vision.**

What materials do you have at your disposal to carry out the assignment?

- This presentation of **introduction to P3**
- Support materials about **RANSAC** and **eigenvalues**.
- *A template* with the statement and some functions:
P3_template.ipynb

Index

- Submission rules
- **Assignment description**

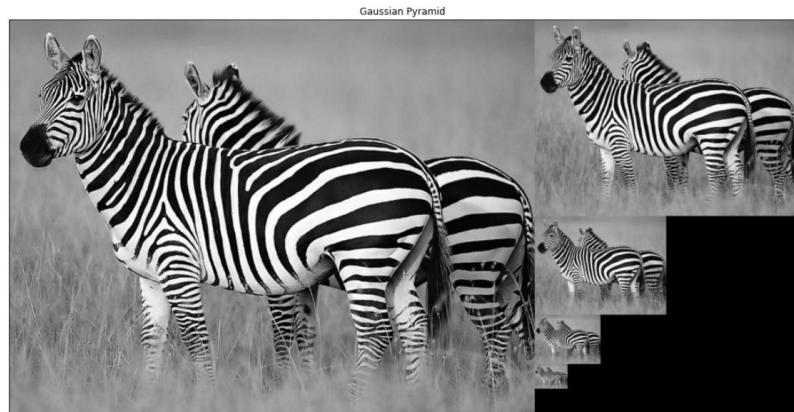
Readings

- ***Homographies and Image Stitching***
 - Forsyth and Ponce (2012), Chapter 12.1
 - Szeliski (2022), Chapter 8.1 and 8.2
 - Sonka, Hlavac and Boyle (2015), Chapter 11.2
 - Hartley and Zisserman (2011), Chapter 4 and 13

Previous assignments

Assignment 1

Image Processing



Assignment 2

Deep Learning

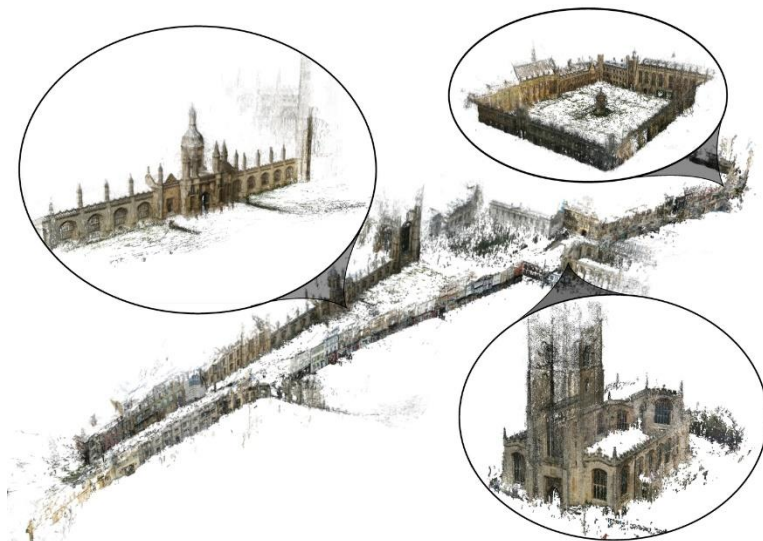


<http://cs231n.stanford.edu/>

Now, interest points detection and description, and image stitching

Assignment 3

Geometry



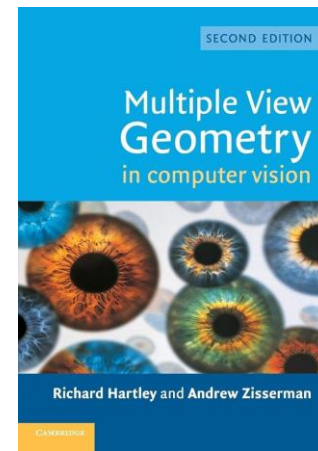
A structure from motion reconstruction of the geometry around central Cambridge, UK

Recommended Reading:

<https://alexgkendall.com/computer-vision/have-we-forgotten-about-geometry-in-computer-vision/>

Computer Vision and Geometry

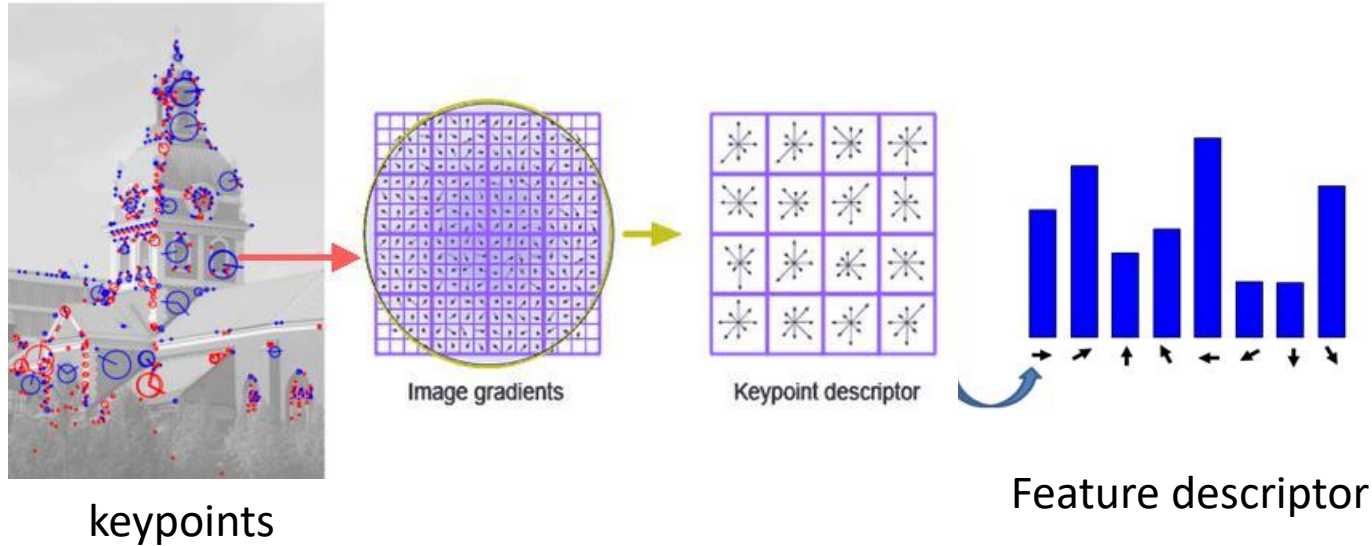
- In 1987, [Joseph Mundy](#) said:
 - "the necessary courses a student should take to really be prepared to carry out research in **model-based vision**. As we can see the geometry of image projection and the mathematics of transformation is a very key element in studying this field [...]. If we are going to talk about segmenting images and getting good geometric clues, we have to understand the relationship between the intensity of image data and its underlying geometry. And this would lead the student into such areas as optics, illumination theory [...]. And also the mathematics underlying this kind of computations would of course require signal processing theory, fourier transform theory [...] courses in algebraic geometry and higher pure forms of algebra will prove to be necessary in order to make any kind of progress in research to handle **curved surfaces**. "



The reference book in the field is dedicated to Joe Mundy

Interest Point Detection and Description

Feature = keypoint + descriptor

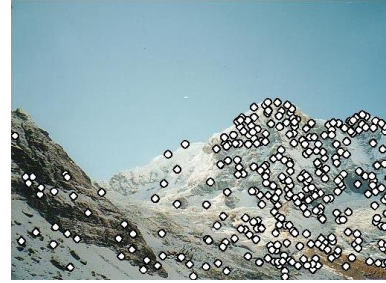


<https://gilscvblog.com/2013/08/18/a-short-introduction-to-descriptors/>

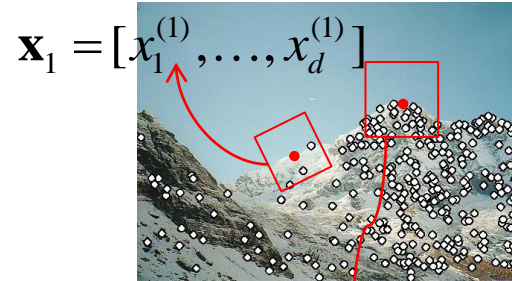
<https://www.codeproject.com/Articles/619039/Bag-of-Features-Descriptor-on-SIFT-Features-with-O>

Interest Point Detection

1) **Detection:** Identify the interest points



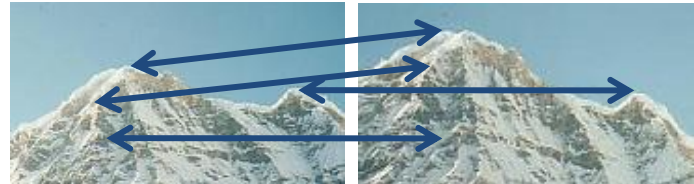
2) **Description:** Extract vector feature descriptor surrounding each interest point



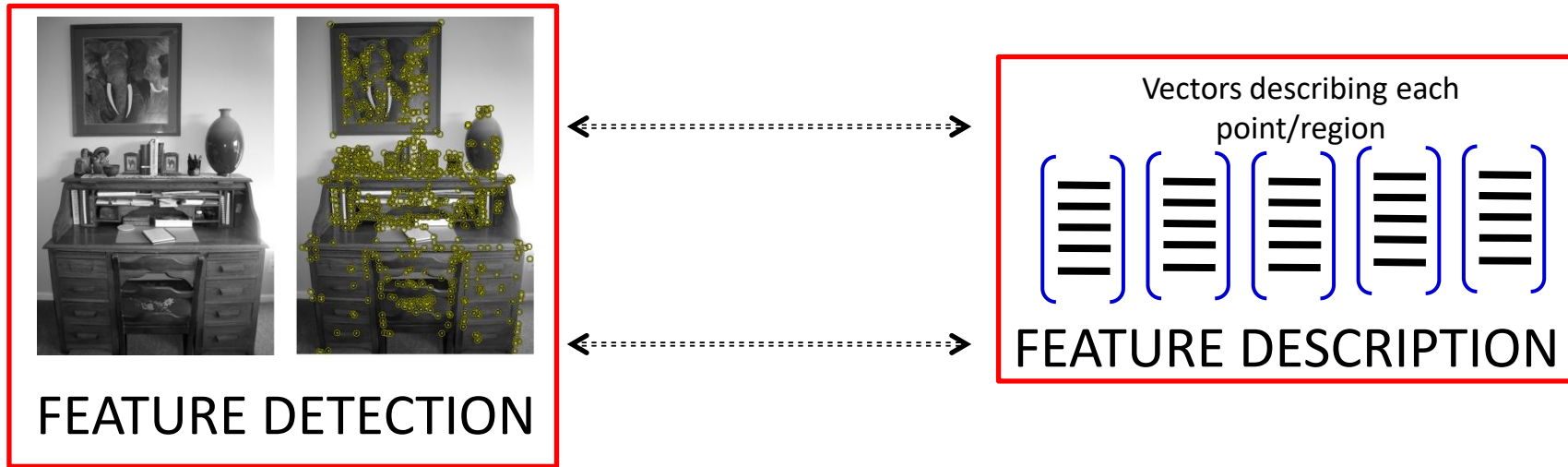
$$\mathbf{x}_1 = [x_1^{(1)}, \dots, x_d^{(1)}]$$

$$\mathbf{x}_2 = [x_1^{(2)}, \dots, x_d^{(2)}]$$

3) **Matching:** Determine correspondence between descriptors in two views



Interest Point Detection



For example, SIFT is both a detector and a descriptor and, for the keypoint detection part, it uses the Difference of Gaussians on rescaled images

Edge detection (Canny, Sobel,...)

Corner detection (**Harris**, FAST,...)

Blob detection (**LoG**, **DoG**,...)

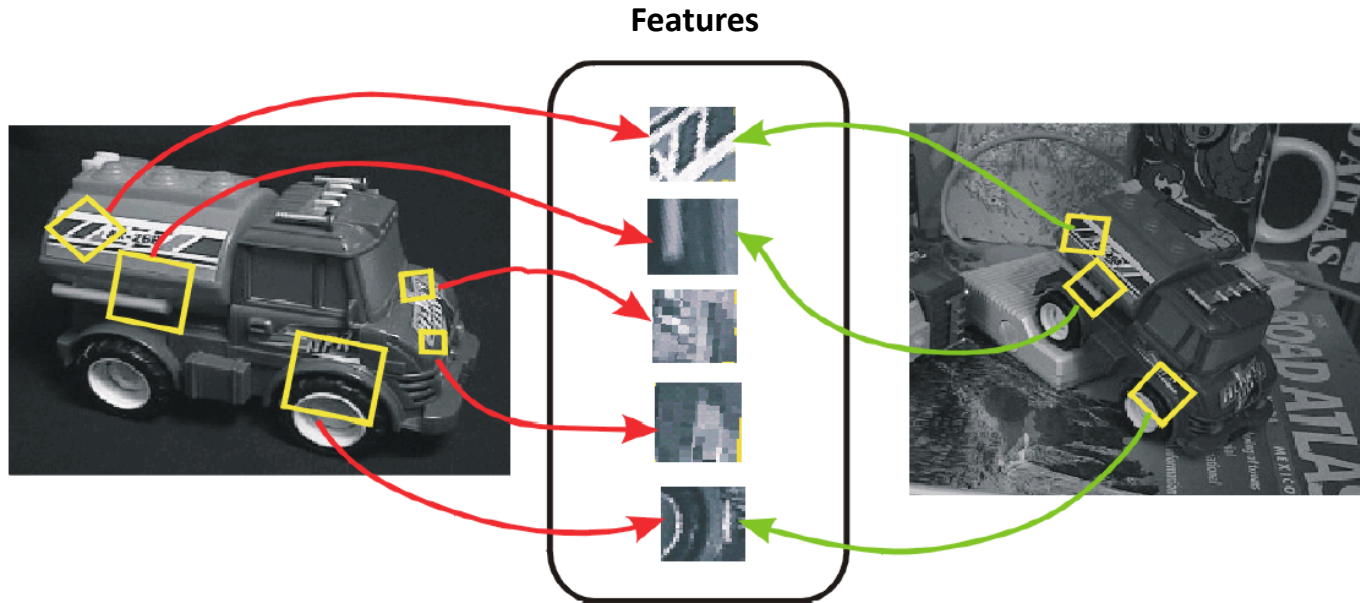
SIFT, SURF, HOG, KAZE,...

GLCM, LBP,...

More information about *Feature Detection* using *OpenCV*:

https://docs.opencv.org/4.4.0/db/d27/tutorial_py_table_of_contents_feature2d.html

From keypoint detection to feature description



Detection is *covariant*:

$$\text{features}(\text{transform}(\text{image})) = \text{transform}(\text{features}(\text{image}))$$

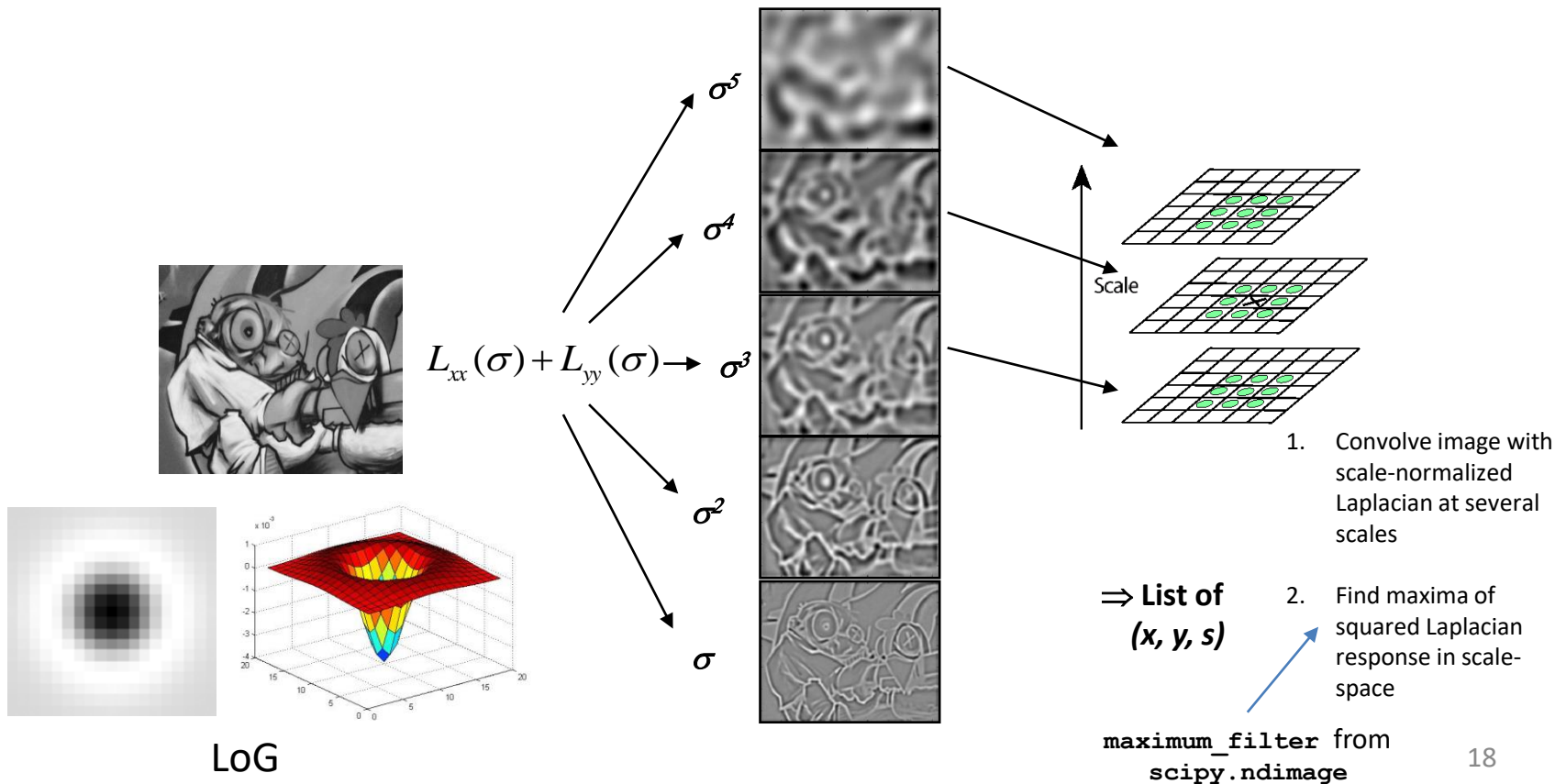
Description is *invariant*:

$$\text{features}(\text{transform}(\text{image})) = \text{features}(\text{image})$$

Exercise 1:

Blob detection using multiscale LoG

Exercise 1: Find local maxima in 3D position-scale space (2 points)

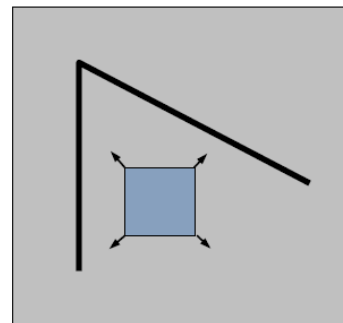
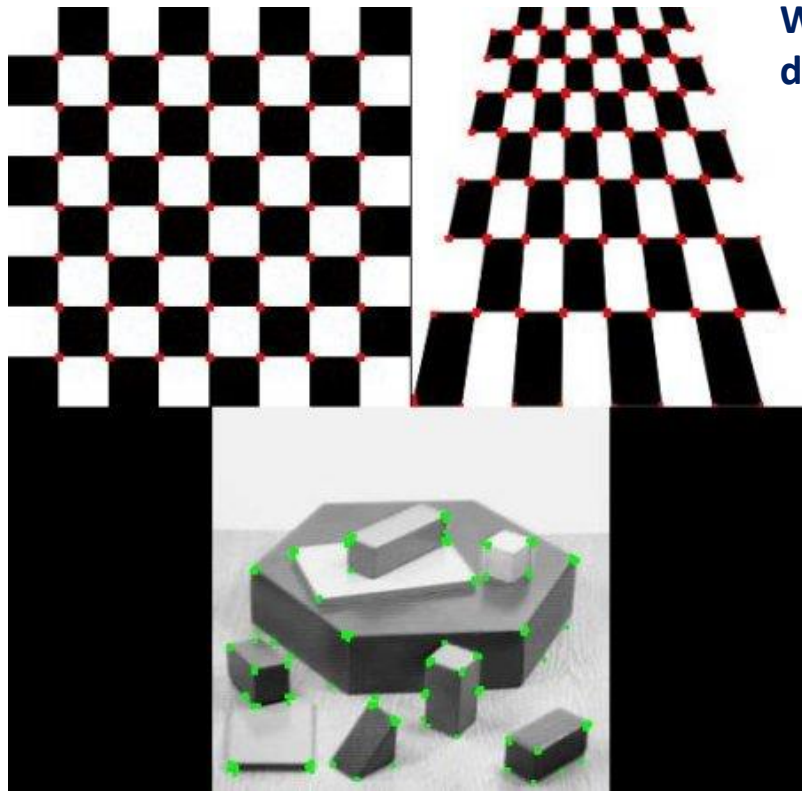


Exercise 2:

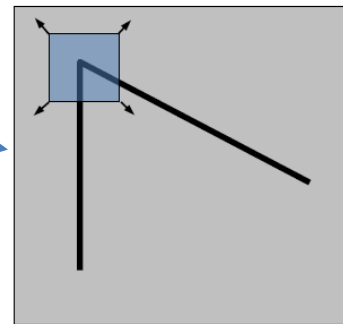
Harris corner detector

Harris Detector

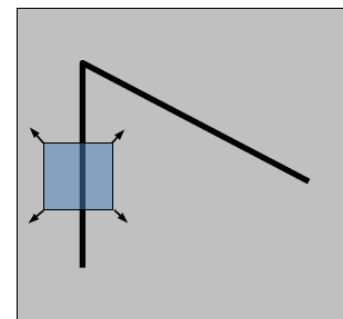
Distinctive region.
Wherever I move I
detect changes.



“flat” region:
no change in all
directions



“corner”:
significant change in
all directions



“edge”:
no change along the
edge direction

Harris Detector: practical aspects

1) Build matrix H (*second moment matrix*, SMM):

$$H(u, v) = \begin{pmatrix} \sum_w w(x, y) \partial I_x^2 & \sum_w w(x, y) \partial I_x \partial I_y \\ \sum_w w(x, y) \partial I_x \partial I_y & \sum_w w(x, y) \partial I_y^2 \end{pmatrix}$$

Harris Detector: practical aspects

1) Build matrix H (*second moment matrix*, SMM):

Derivative with respect to X (squared)

$$H(u, v) = \begin{pmatrix} \sum_w w(x, y) \partial I_x^2 & \sum_w w(x, y) \partial I_x \partial I_y \\ \sum_w w(x, y) \partial I_x \partial I_y & \sum_w w(x, y) \partial I_y^2 \end{pmatrix}$$

Derivative with respect to X
multiplied by derivative with respect
to Y

Derivative with respect to X multiplied
by derivative with respect to Y

Derivative with respect to Y (squared)

Harris Detector: practical aspects

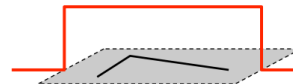
1) Build matrix H (*second moment matrix*, SMM):

$$H(u, v) = \begin{pmatrix} \sum_W w(x, y) \partial I_x^2 & \sum_W w(x, y) \partial I_x \partial I_y \\ \sum_W w(x, y) \partial I_x \partial I_y & \sum_W w(x, y) \partial I_y^2 \end{pmatrix}$$

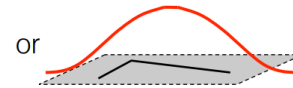
Note: each of the components of matrix $H(u, v)$ has the same size as the input image. Or in other words, now each pixel in the image will have three values associated with it.

Gaussian smoothing (σ_I)

Window function $w(x, y) =$



1 in window, 0 outside



or

Gaussian

Harris Detector: practical aspects

- 1) Build matrix H (*second moment matrix*, SMM)
- 2) Once we have H , we calculate its determinant and its trace.
This gives us a value for each pixel of the image (Harris map)

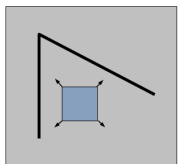
$$f_H = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{\text{Det}(\mathbf{H})}{\text{Trace}(\mathbf{H})} = \frac{\left(\sum_w w(x,y) \partial I_x^2 \cdot \sum_w w(x,y) \partial I_y^2 \right) - \left(\sum_w w(x,y) \partial I_x \partial I_y \cdot \sum_w w(x,y) \partial I_x \partial I_y \right)}{\left(\sum_w w(x,y) \partial I_x^2 + \sum_w w(x,y) \partial I_y^2 \right)}$$

Hadamard product

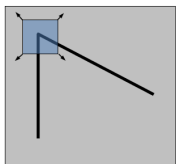
- 3) Remove from f_H the responses below a certain threshold (selected to keep a predefined number of *keypoints*): *Non-Maxima-Suppression*

Importance of the *scale invariance*

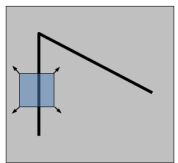
- Harris Detector



"flat" region:
no change in all
directions

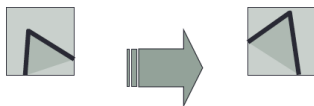


"corner":
significant change in
all directions

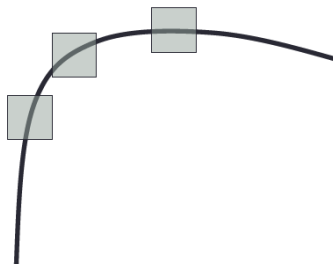


"edge":
no change along the
edge direction

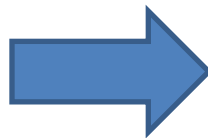
- Rotation invariance



- Not invariant to *image scale*!




All points will be
classified as **edges**



SIFT!!!!

Corner !

Exercise 2: point detection using the Harris detector (3 points)

- 1) Calculate the derivatives of the input image (e.g., $\sigma_D = 1.5$)
- 2) With these derivatives, calculate the Harris map (using $\sigma_I \approx 1.5 \cdot \sigma_D$)
- 3) *Non-Maxima-Suppression* using `corner_peaks(HarrisMaps, min_distance)` and a *threshold* (to filter out low responses).

Chosen by you
- 4) Calculate the orientation (in degrees) of the detected points
$$\theta = \frac{\arctan\left(\frac{\text{smooth}(dy)}{\text{smooth}(dx)}\right) \cdot 180}{\pi}$$
- 5) Display the *keypoints* with `cv2.drawKeypoints()`

Note: remember that you do not have to calculate any eigenvalue!!

Exercise 2: point detection using the Harris detector (3 points)

Harris Keypoints



Exercise 3:

Matching feature descriptors

Exercise 3: matching descriptors (2 points)

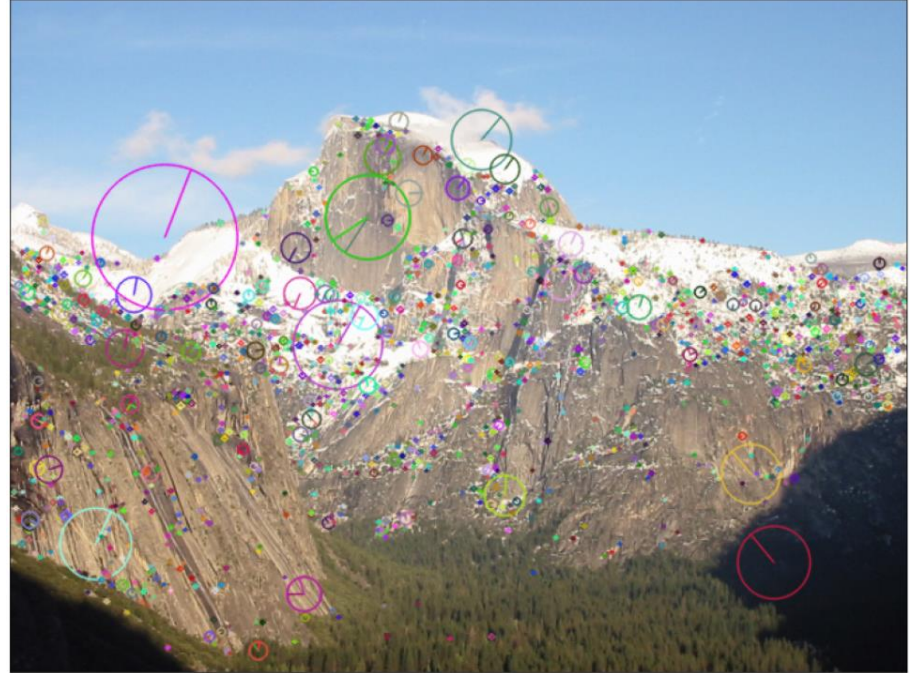
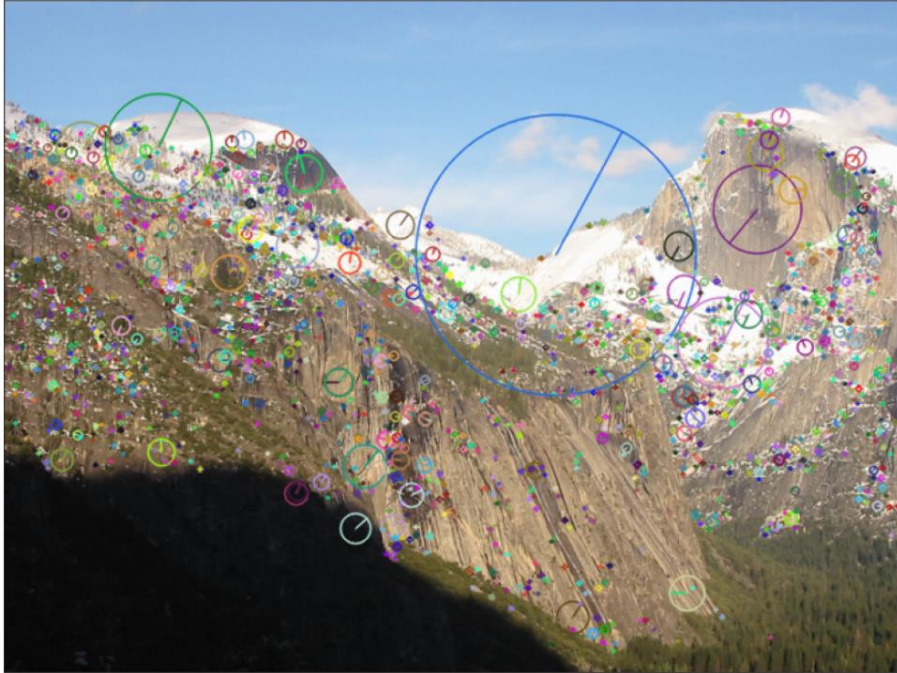
- Use SIFT (which provides a descriptor with 128 values) to calculate the matching to correspondence between images.

```
sift = cv2.SIFT_create()  
kpts1, desc1 = sift.detectAndCompute(img1, None)  
kpts2, desc2 = sift.detectAndCompute(img2, None)
```

https://docs.opencv.org/4.x/d7/d60/classcv_1_1SIFT.html

https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

Exercise 3: SIFT descriptors (2 points)



Exercise 3: SIFT descriptors (2 points)

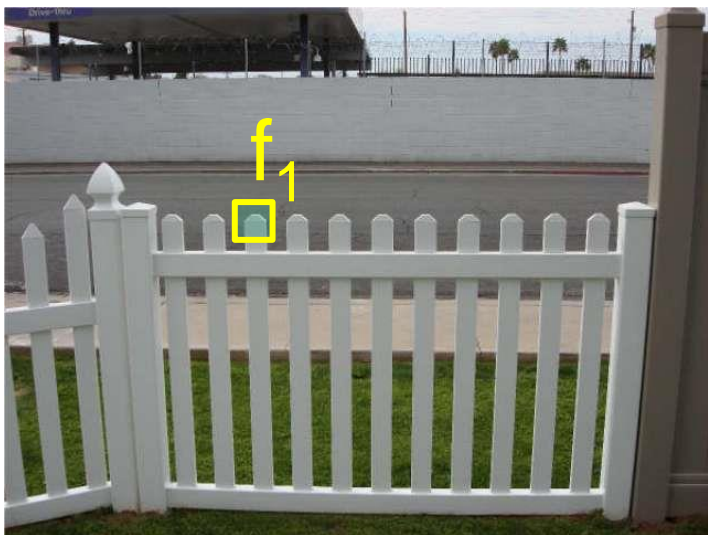
Given a feature in I_1 , how to find the best match in I_2 ?

1. Define distance function that compares two descriptors
2. Test all the features in I_2 , find the one with min distance

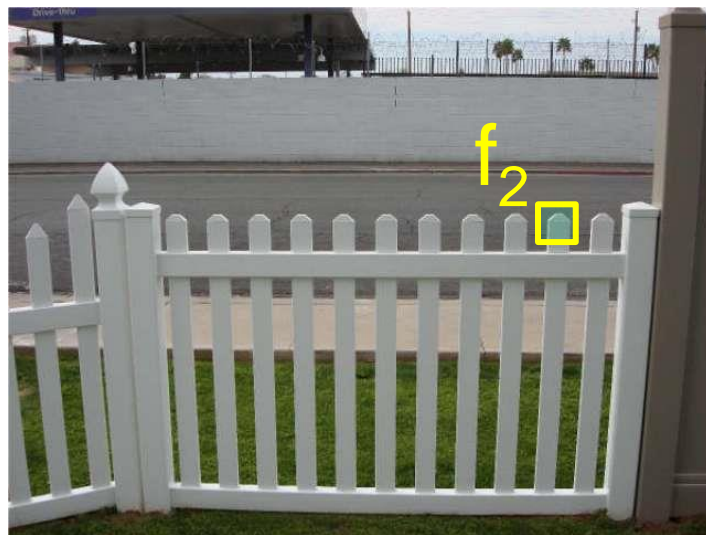
Exercise 3: SIFT descriptors (2 points)

How to define the difference between two features f_1, f_2 ?

- Simple approach (*brute force*): **L_2 distance**, $||f_1 - f_2||$
- can give small distances for ambiguous (incorrect) matches



I_1

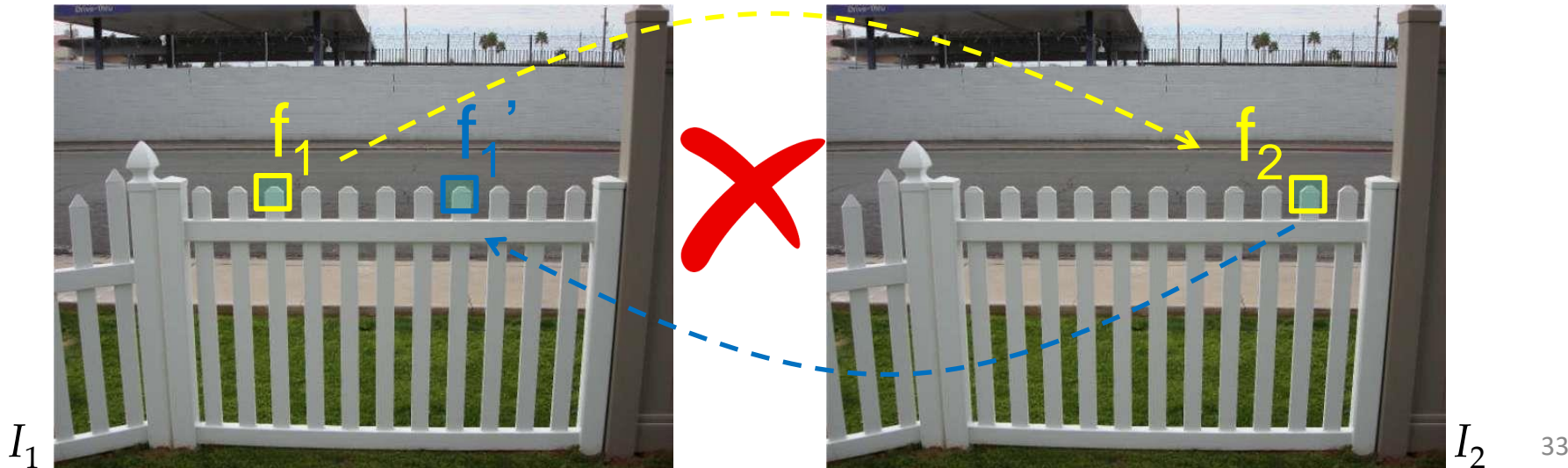


I_2

Exercise 3: SIFT descriptors (2 points)

How to define the difference between two features f_1, f_2 ?

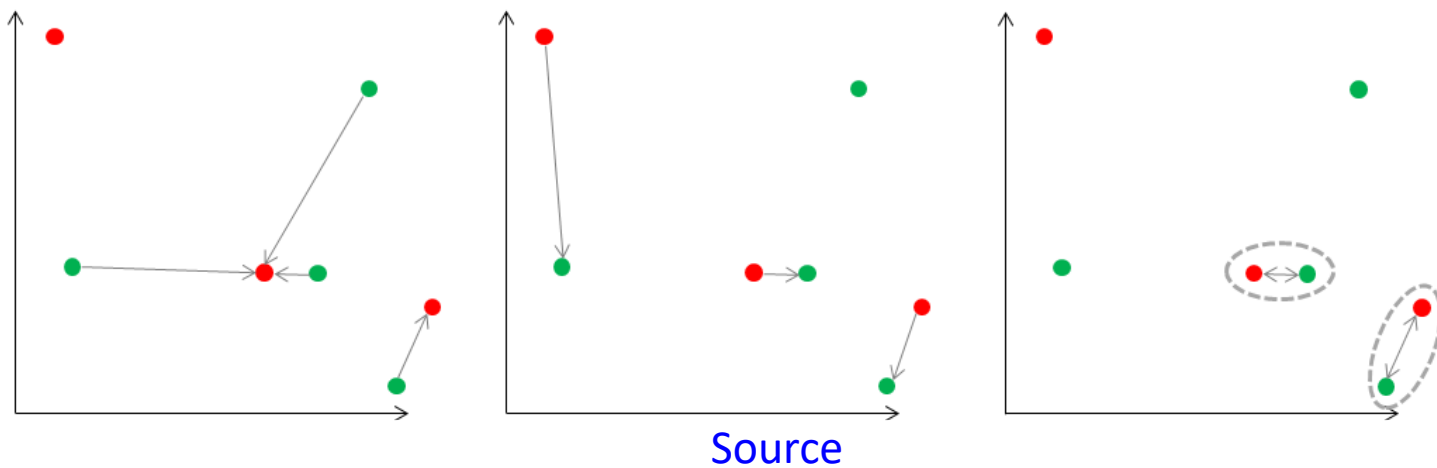
- Better approach (*brute force + cross-check*): **L_2 distance, $||f_1 - f_2||$**
- A match is considered valid only if we obtain the same pair of keypoints in both directions (that is, each keypoint is the best match of the other).



Exercise 3: SIFT descriptors (2 points)

How to define the difference between two features f_1, f_2 ?

- Better approach (*brute force + cross-check*): **L_2 distance, $||f_1 - f_2||$**
- A match is considered valid only if we obtain the same pair of keypoints in both directions (that is, each keypoint is the best match of the other).



Exercise 3: SIFT descriptors (2 points)

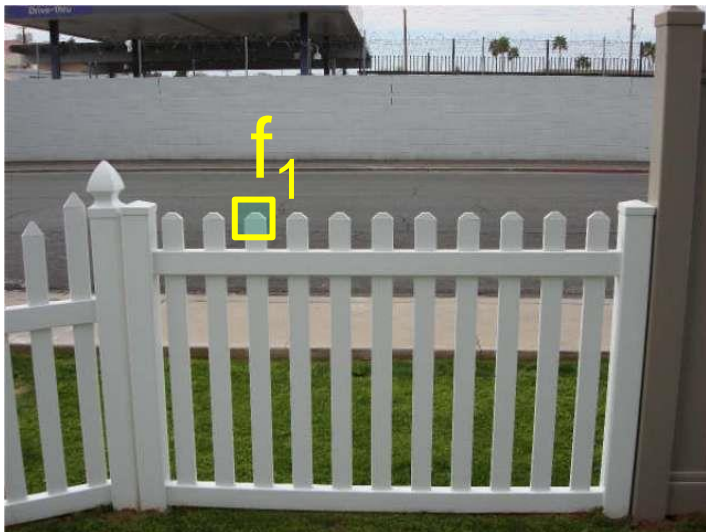
- How to define the difference between two features f_1, f_2 ?

- Better approach (*ratio test*): **ratio distance** = $\frac{\|f_1 - f_2\|}{\|f_1 - f_2'\|}$

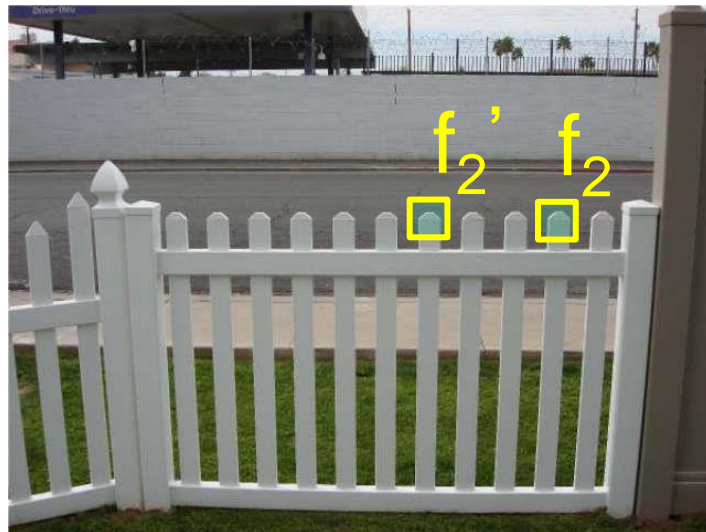
- f_2 is the best SSD match to f_1 in I_2
- f_2' is the 2nd best SSD match to f_1 in I_2
- gives large values for ambiguous matches

We want **this** to be clearly smaller than **this** to avoid ambiguous matches.

I_1



I_2



Exercise 3: SIFT descriptors (2 points)

- **Ratio distance** = $\|f_1 - f_2\| / \|f_1 - f_2'\|$
 - f_2 is the best SSD match to f_1 in I_2
 - f_2' is the 2nd best SSD match to f_1 in I_2

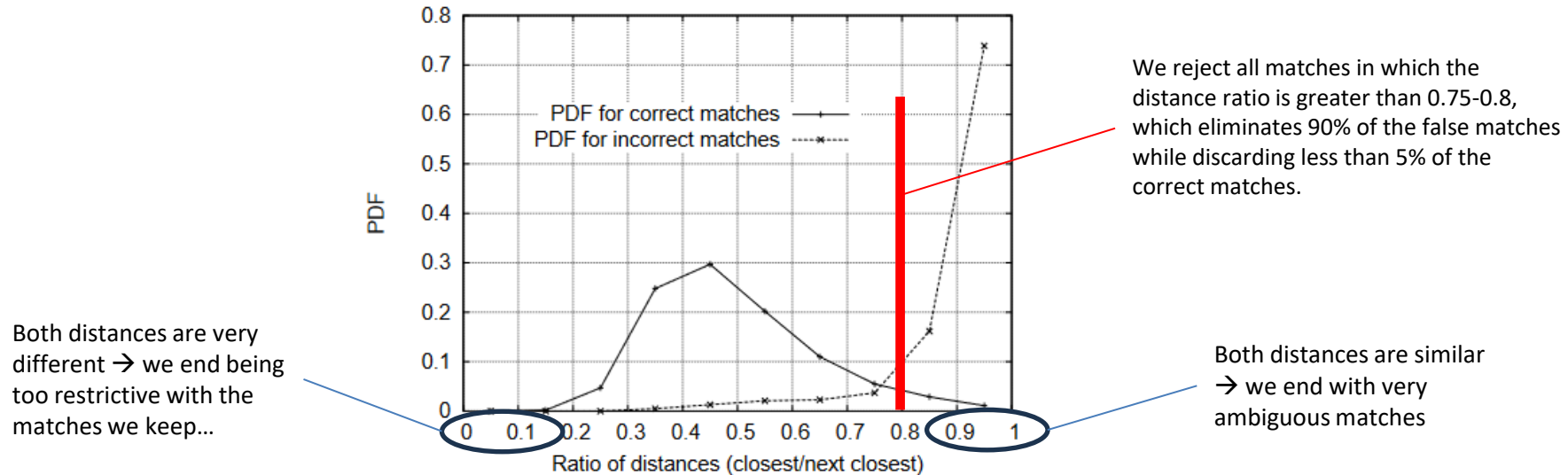
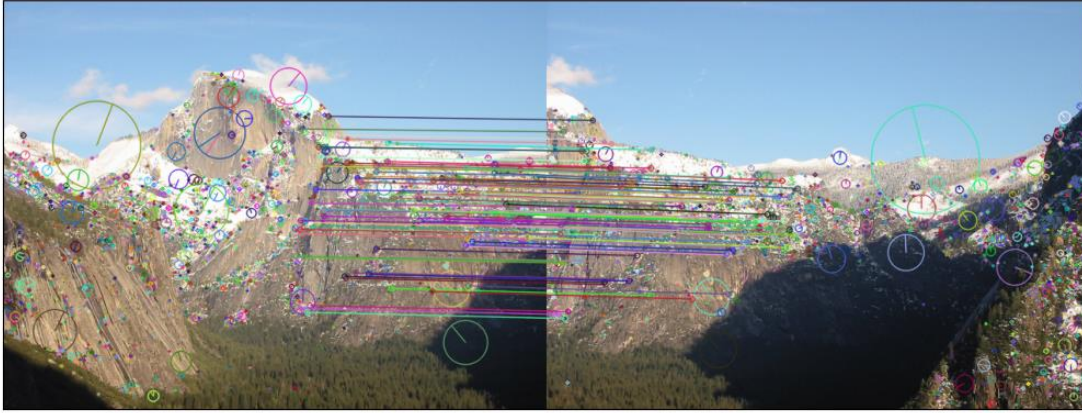


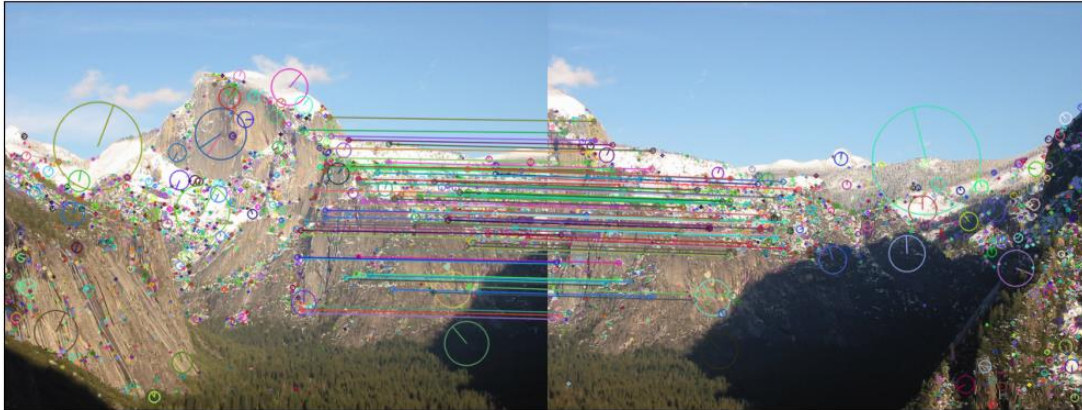
Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

Exercise 3: SIFT descriptors (2 points)

SIFT FEATURES: 100 Matches BruteForce+Crosscheck



SIFT FEATURES: 100 Matches 2NN



Cross-check:

- We keep a match when it represents the smallest distance (i.e. the best match) both from image A to image B, and from B to image A.

Lowe's ratio test:

- Distance to the best match divided by the distance to the second best.
- A threshold (usually 0.75) is used to discard matches that are not sufficiently unambiguous.

Exercise 3: matching descriptors (2 points)

- Now, you have to repeat the same process but employing Haralick features.
- Haralick texture features are implemented in the **mahotas** Python package

```
import mahotas
im=cv2.imread(get_image('zebra.jpg'),0)
features = mahotas.features.haralick(im).mean(axis=0)
```

- Which ones seem to be better for keypoint matching??

Exercise 4:

Image Stitching (Panorama Creation)

Creating Panoramas (*Image Stitching*)

- The process of combining multiple, partially overlapping images to produce a panorama or high-resolution image.



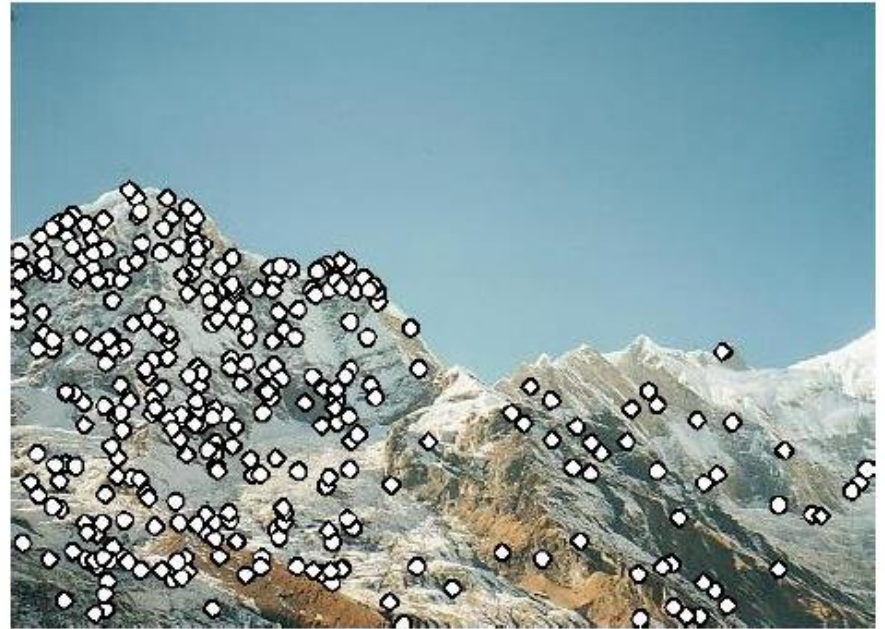
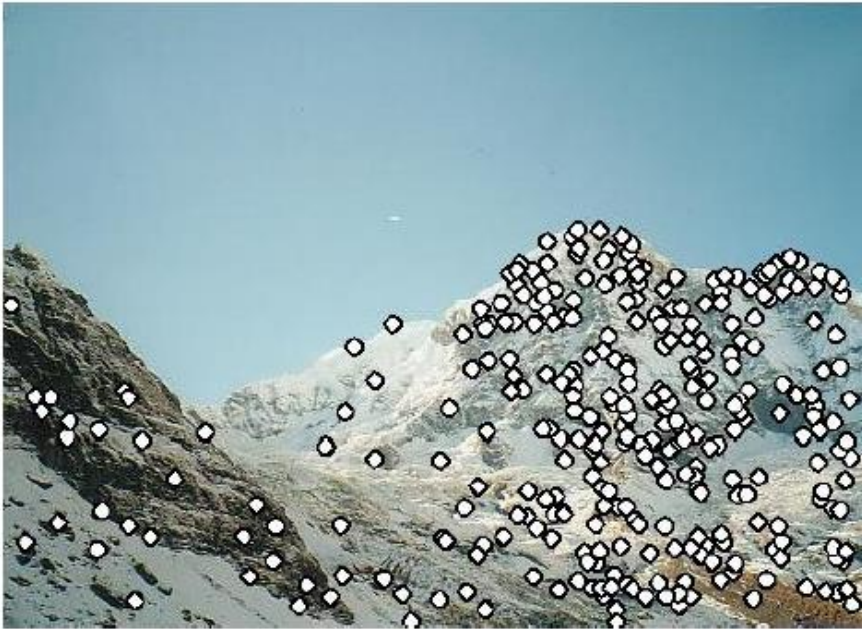
How do we create a panorama?

- We need to align images



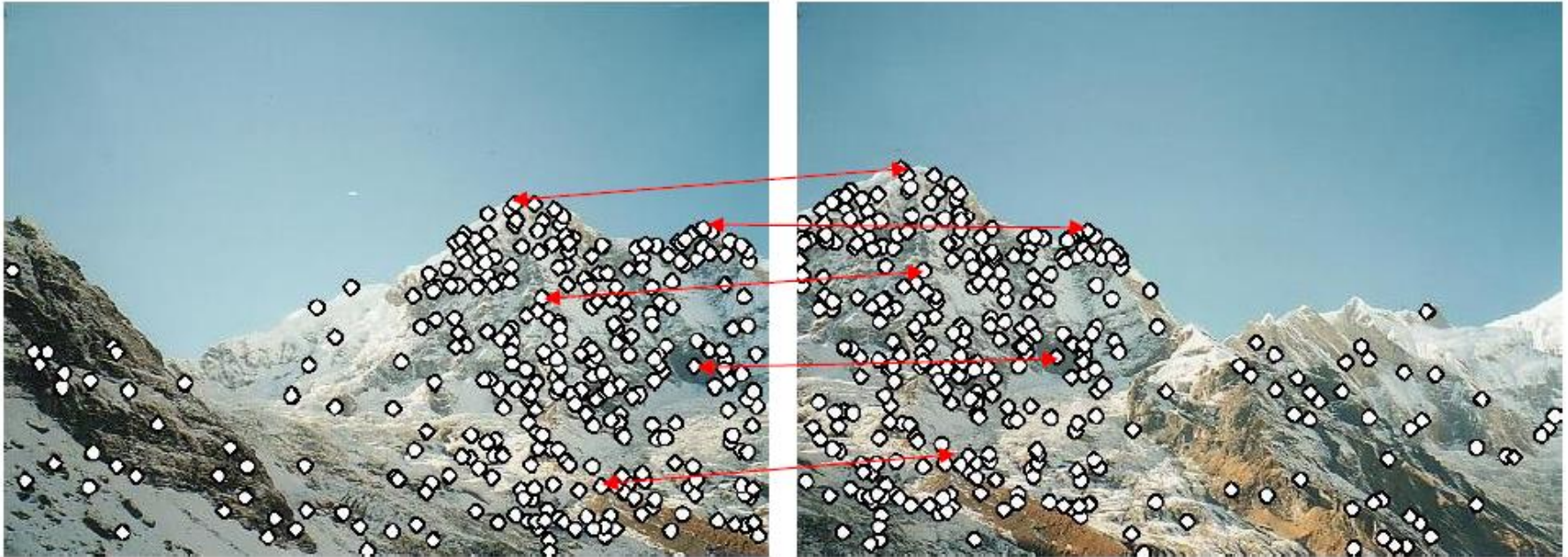
How do we create a panorama?

- We detect keypoints on both images



How do we create a panorama?

- We find corresponding pairs



How do we create a panorama?

- We use these pairs to align the images



Problem 1

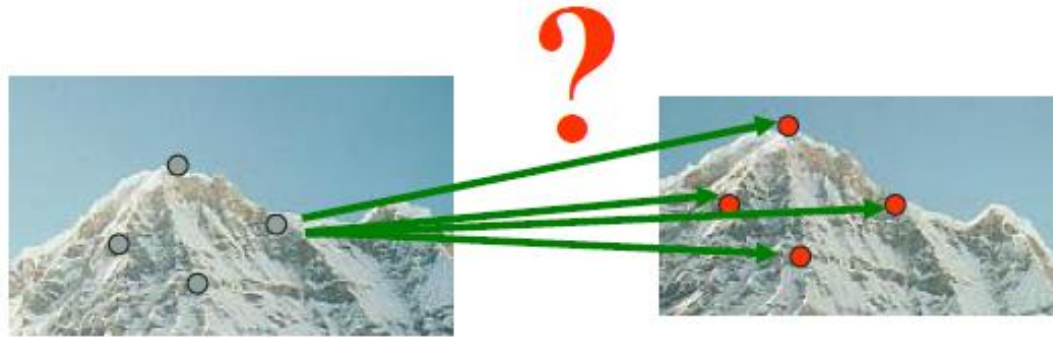
- We need to detect the same keypoint in both images
 - Repeatable/stable **detector**



no chance to match!

Problem 2

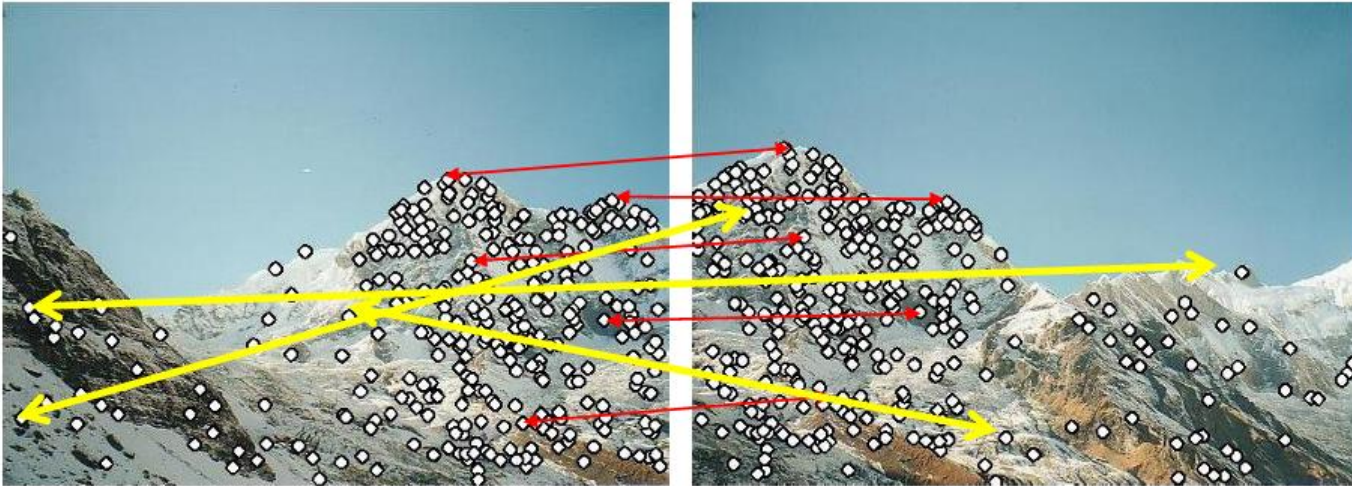
- For each point, we need to find its match in the other image
 - Reliable and distinctive **descriptor**



- Must provide some **invariance** to **geometric** and **photometric** differences between the two views.

Problem 3

- Need to estimate transformation between images, despite erroneous correspondences



RANSAC helps us in this regard!

Creating Panoramas

- This is where **SIFT** comes into play, as a **detector and descriptor**
- We create panoramas using **homographies**

Panoramic Stitching: views from rotating camera

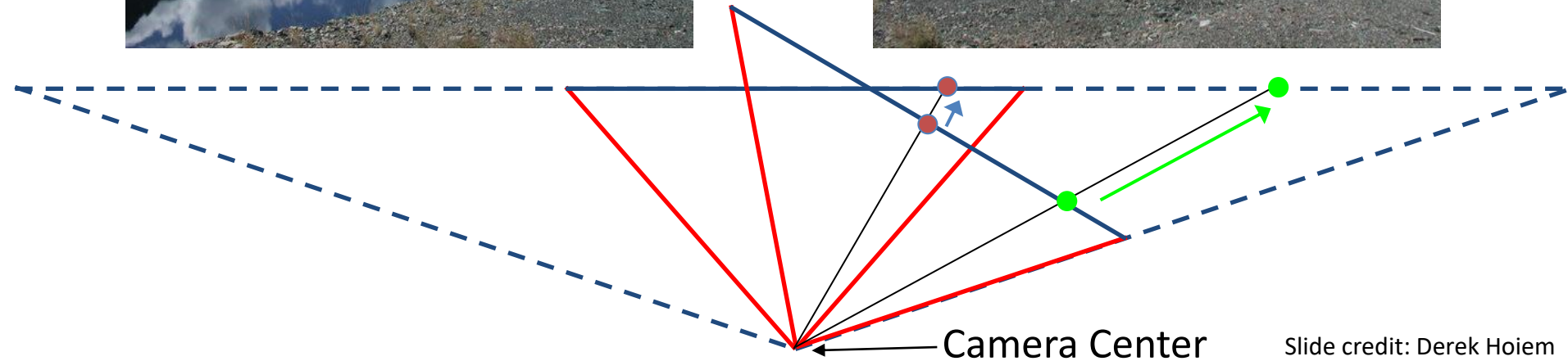
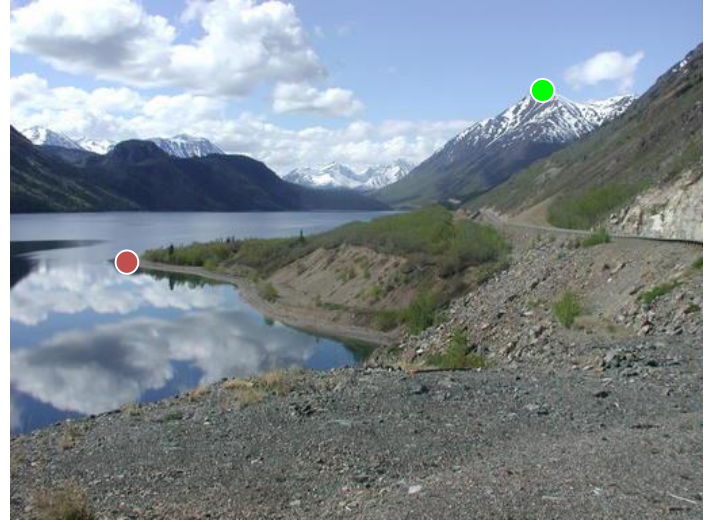
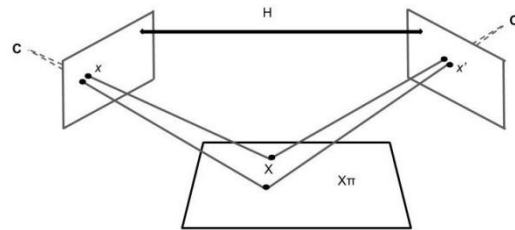
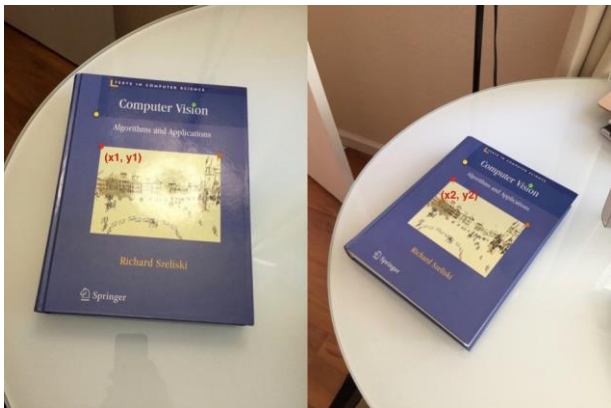


Image Stitching Algorithm Overview

1. Detect keypoints
2. Match keypoints
3. Estimate homography with four matched keypoints (using RANSAC to deal with outliers)
 - This helps us to find the geometric relationship between the images.
4. Project onto a surface and blend

Homographies

- Any two images of the same surface are related by a homography (*planar perspective map*).
- A homography is a transformation (3x3 matrix) that allows mapping (*map/warp*) points from one image onto the other.



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

Homographies

- Any two images of the same surface are related by a homography (*planar perspective map*).
- A homography is a transformation (3x3 matrix) that allows mapping (*map/warp*) points from one image onto the other.

Image **Warping**: Coordinate changing operator

vs

Image **Filtering**: Brightness changing operator

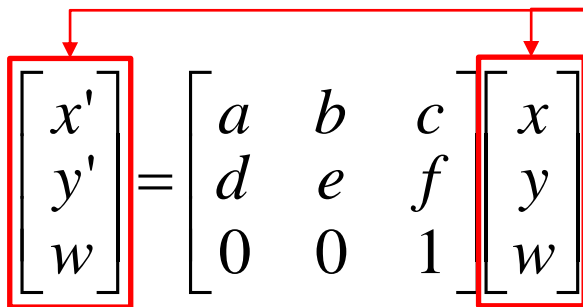
Homographies

- Any two images of the same surface are related by a homography (*planar perspective map*).
- A homography is a transformation (3x3 matrix) that allows mapping (*map/warp*) points from one image onto the other.
- Homographies are useful when:
 - We capture images of any 3D scene, but from the same viewpoint.
 - The scene is very far away (plane at infinity).
 - The scene points lie on a plane.

Affine Transformations

- Affine transformations are combinations of ...
 - Linear transformations (scale, rotation, shear, mirror), and

– Translations


$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

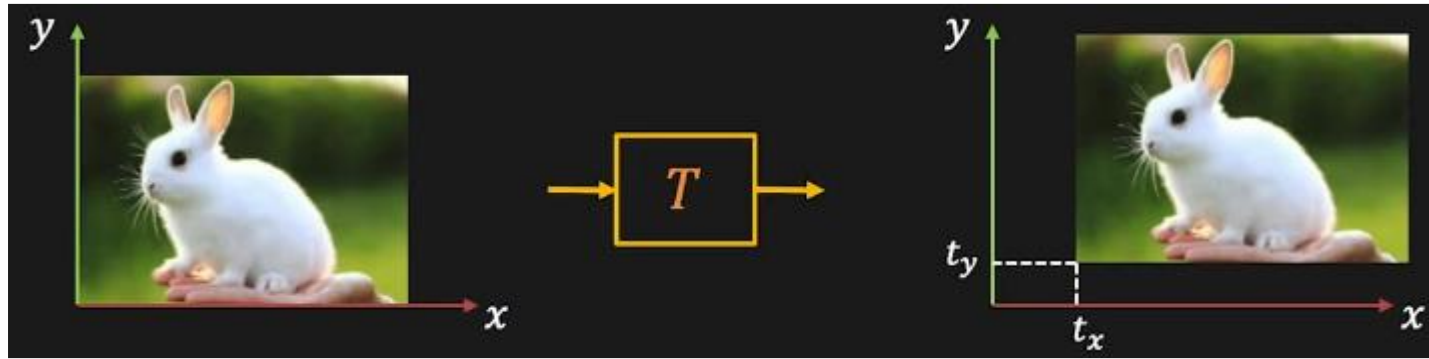
Homogeneous coordinates

w is a fictitious coordinate (no geometric meaning)

- We use this representation because translation cannot be expressed as a 2×2 matrix

Basic Affine Transformations

Translation:



There is no way to incorporate shifts t_x and t_y into a 2×2 transformation matrix. As a consequence, we use homogeneous coordinates (3×3 matrix).

$$x_2 = x_1 + t_x \quad y_2 = y_1 + t_y$$

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$x = \frac{\tilde{x}}{\tilde{z}} \quad y = \frac{\tilde{y}}{\tilde{z}}$$

$$\mathbf{p} \equiv \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \tilde{\mathbf{p}}$$

Basic Affine Transformations

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} 1 & m_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Skew

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Translation

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Rotation

Any transformation with last row $[0 \ 0 \ 1]$, we call it an *affine* transformation

Affine Transformations

- Affine transformations are combinations of ...
 - Linear transformations, and
 - Translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of affine transformations:
 - Origin does not necessarily map to origin
 - Because we have translation, i.e., if you have $x = 0$ and $y = 0$, you do not necessarily have $x' = 0$ and $y' = 0$.
 - Lines map to lines
 - Parallel lines remain parallel
 - Ratios are preserved
 - Closed under composition

$$\left. \begin{array}{l} \mathbf{p}_2 = T_{21}\mathbf{p}_1 \\ \mathbf{p}_3 = T_{32}\mathbf{p}_2 \\ \mathbf{p}_3 = T_{31}\mathbf{p}_1 \end{array} \right\} \mathbf{p}_3 = T_{32}\mathbf{p}_2 = T_{32}T_{21}\mathbf{p}_1 \Rightarrow T_{31} = T_{32}T_{21}$$

Projective Transformations a.k.a. Homographies a.k.a. Planar Perspective Maps

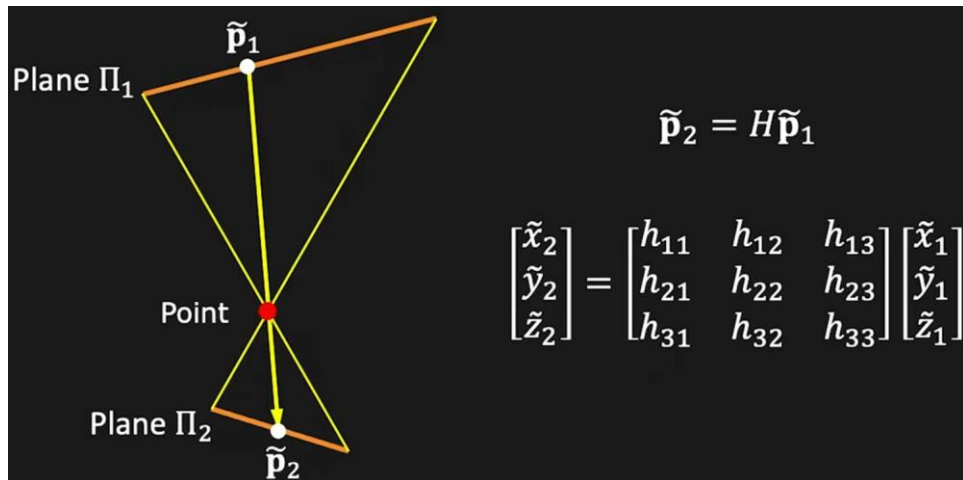
$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

What if we
don't restrict
this last row to
be $[0 \ 0 \ 1]$?

Called a *homography*
(or *planar perspective map*)

Now, **parallel lines do not
necessarily remain parallel!**

A projective matrix maps one
plane to another plane (but
maps it through a point)

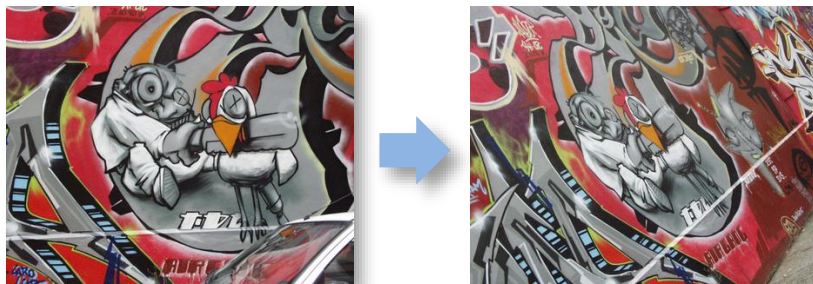
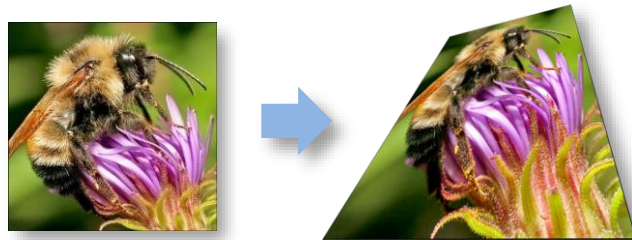


Projective Transformations a.k.a. Homographies a.k.a. Planar Perspective Maps

$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

What if we
don't restrict
this last row to
be $[0 \ 0 \ 1]$?

Called a *homography*
(or *planar perspective map*)



Projective Transformation

- Homography can only be defined up to scale
 - Since we are using homogeneous coordinates, we can multiply our coordinates $(\tilde{x}_1, \tilde{y}_1, \tilde{z}_1)$ by any scale factor (k) and we get a point which is equivalent...

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} \equiv k \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

$$\mathbf{p} \equiv \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{z}x \\ \tilde{z}y \\ \tilde{z} \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \tilde{\mathbf{p}}$$

$$x = \frac{\tilde{x}}{\tilde{z}} \quad y = \frac{\tilde{y}}{\tilde{z}}$$

$$x = \frac{\tilde{x}}{\tilde{z}} = \frac{k\tilde{x}}{k\tilde{z}}$$

Projective Transformation

- Homography can only be defined up to scale
 - Since we are using homogeneous coordinates, we can multiply our coordinates $(\tilde{x}_1, \tilde{y}_1, \tilde{z}_1)$ by any scale factor (k) and we get a point which is equivalent...

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} \equiv k \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

The homography matrix is a 3×3 matrix but with **8 DoF (degrees of freedom)** as it is estimated up to a scale.

Projective Transformation

- Homography can only be defined up to scale
 - Since we are using homogeneous coordinates, we can multiply our coordinates $(\tilde{x}_1, \tilde{y}_1, \tilde{z}_1)$ by any scale factor (k) and we get a point which is equivalent...

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} \equiv k \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

3x3 matrix but with **8 DoF (degrees of freedom)** → **4 correspondence points are sufficient** (each correspondence solves two constraints), but...
we generally have much more matches. 😊

Direct Linear Transformation (DLT): Solving for homographies

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

This is just a quick summary of DLT. For more information, please, refer to [Hartley and Zisserman's book, Ch. 4](#), or to [Robert Collins' notes](#)

$$x'_i = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$y'_i = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

Converting from homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

$$x'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02}$$

$$y'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{12}$$

Direct Linear Transformation (DLT):

Solving for homographies

$$x'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02}$$

$$y'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{12}$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i & -y'_i \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Each pair of points gives us two equations

Direct Linear Transformation (DLT): Solving for homographies

$$\begin{array}{c}
 \text{Match 1} \\
 \dots \\
 \text{Match n}
 \end{array}
 \begin{bmatrix}
 x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\
 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\
 & & & & & & \vdots & & \\
 x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\
 0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n
 \end{bmatrix}
 \begin{bmatrix}
 h_{00} \\
 h_{01} \\
 h_{02} \\
 h_{10} \\
 h_{11} \\
 h_{12} \\
 h_{20} \\
 h_{21} \\
 h_{22}
 \end{bmatrix}
 =
 \begin{bmatrix}
 0 \\
 0 \\
 0 \\
 \vdots \\
 0 \\
 0
 \end{bmatrix}$$

\mathbf{A}
 $2n \times 9$
Known

\mathbf{h}
 9
Unknown

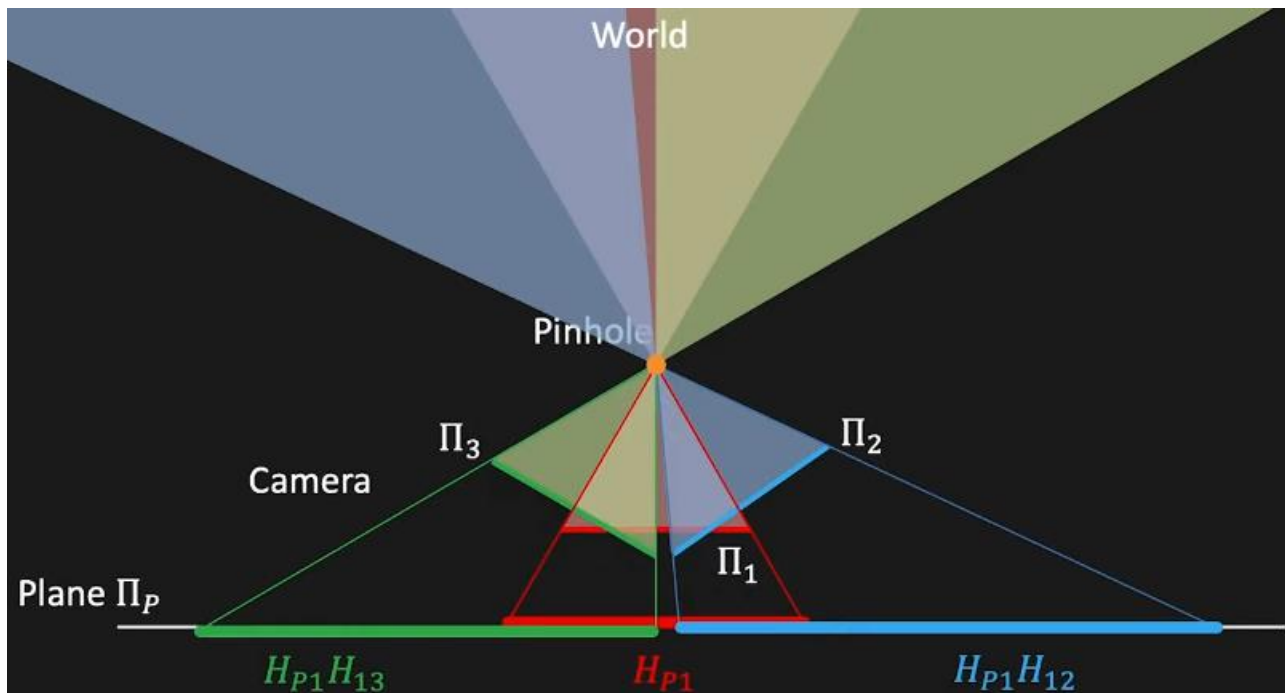
$\mathbf{0}$
 $2n$

This defines a least squares problem:

$$\min_{\mathbf{h}} \|\mathbf{A}\mathbf{h}\|^2 \text{ such that } \|\mathbf{h}\|^2 = 1$$

Solution (constrained least squares): $\mathbf{A}^T \mathbf{A} \mathbf{h} = \lambda \mathbf{h}$ **Eigenvalue Problem.** Eigenvector \mathbf{h} with smallest eigenvalue λ of matrix $\mathbf{A}^T \mathbf{A}$ is our solution.

Homography composition



If your images share the same center of projection (also called camera center) you can map all images to a single plane by computing the homographies between the images.

Map plane 2 to plane 1 (H_{12}), and then map this to plane P (matrix multiplication with H_{P1}).

Recap

- To compute a homography:

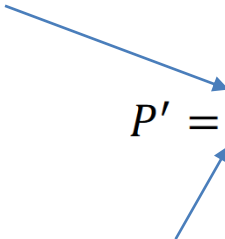
1. Detect and match keypoints
2. Convert them to homogeneous coordinates
3. Estimate homography using DLT and RANSAC

$$p = \begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- To apply a homography:

1. Convert to homogeneous coordinates
2. Multiply by the homography matrix
3. Convert back to heterogeneous coordinates

What is the size of the homography matrix? 3x3

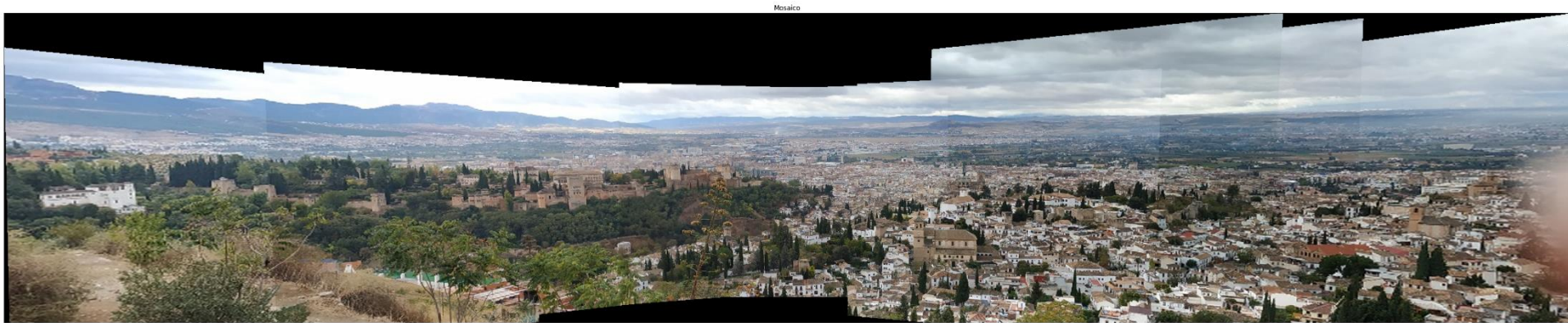

$$P' = H \cdot P$$

$$P' = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \Rightarrow p' = \begin{bmatrix} x'/w' \\ y'/w' \end{bmatrix}$$

How many degrees of freedom does the homography matrix have? 8

Exercise 4: calculation of homographies and composition of a mosaic (3 points)

- This is the same as in the previous exercise, but now, together with the calculation of the correspondences, we apply the corresponding transformation (homography) to align them.

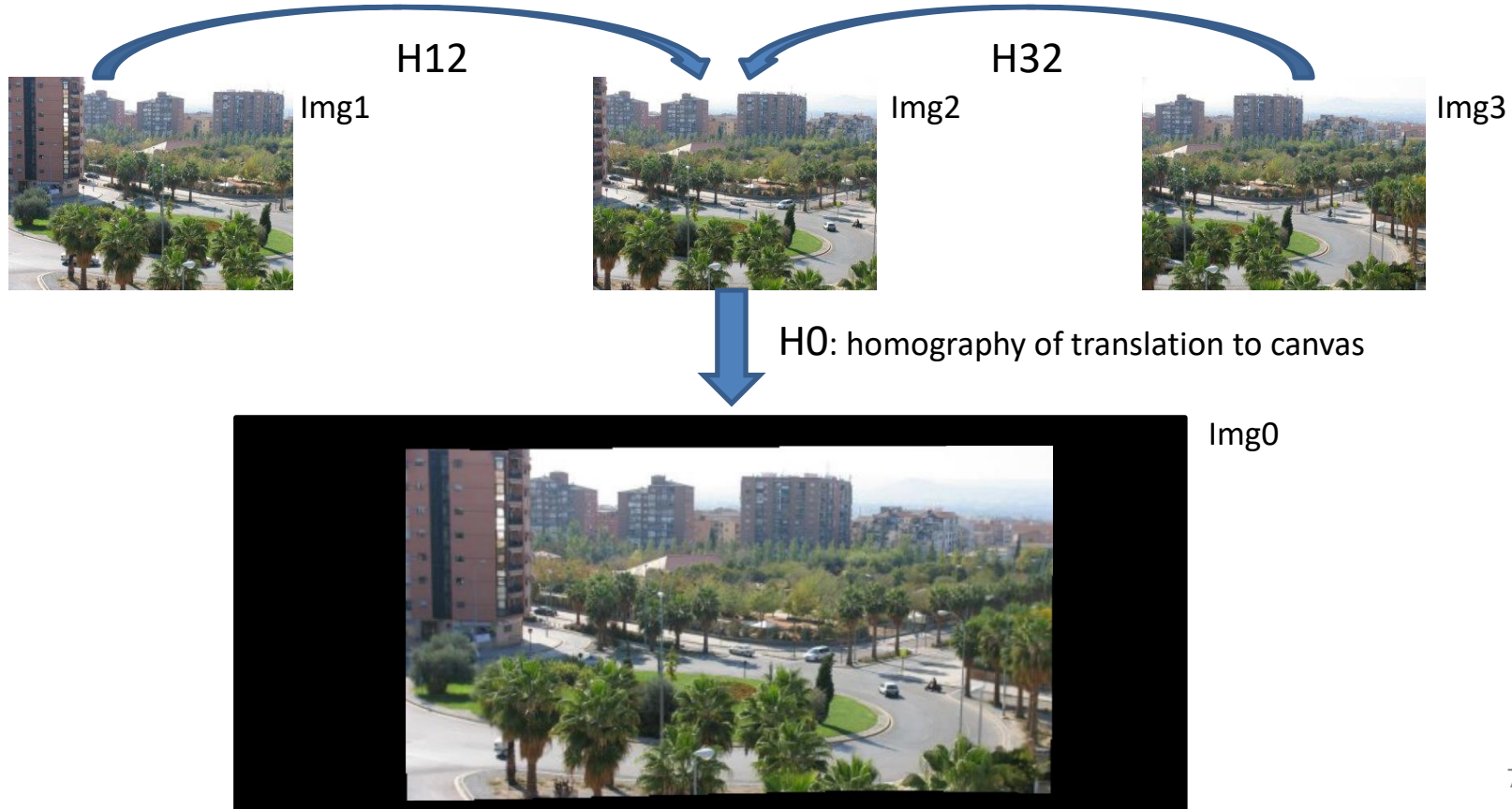


Exercise 4: calculation of homographies and composition of a mosaic (3 points)

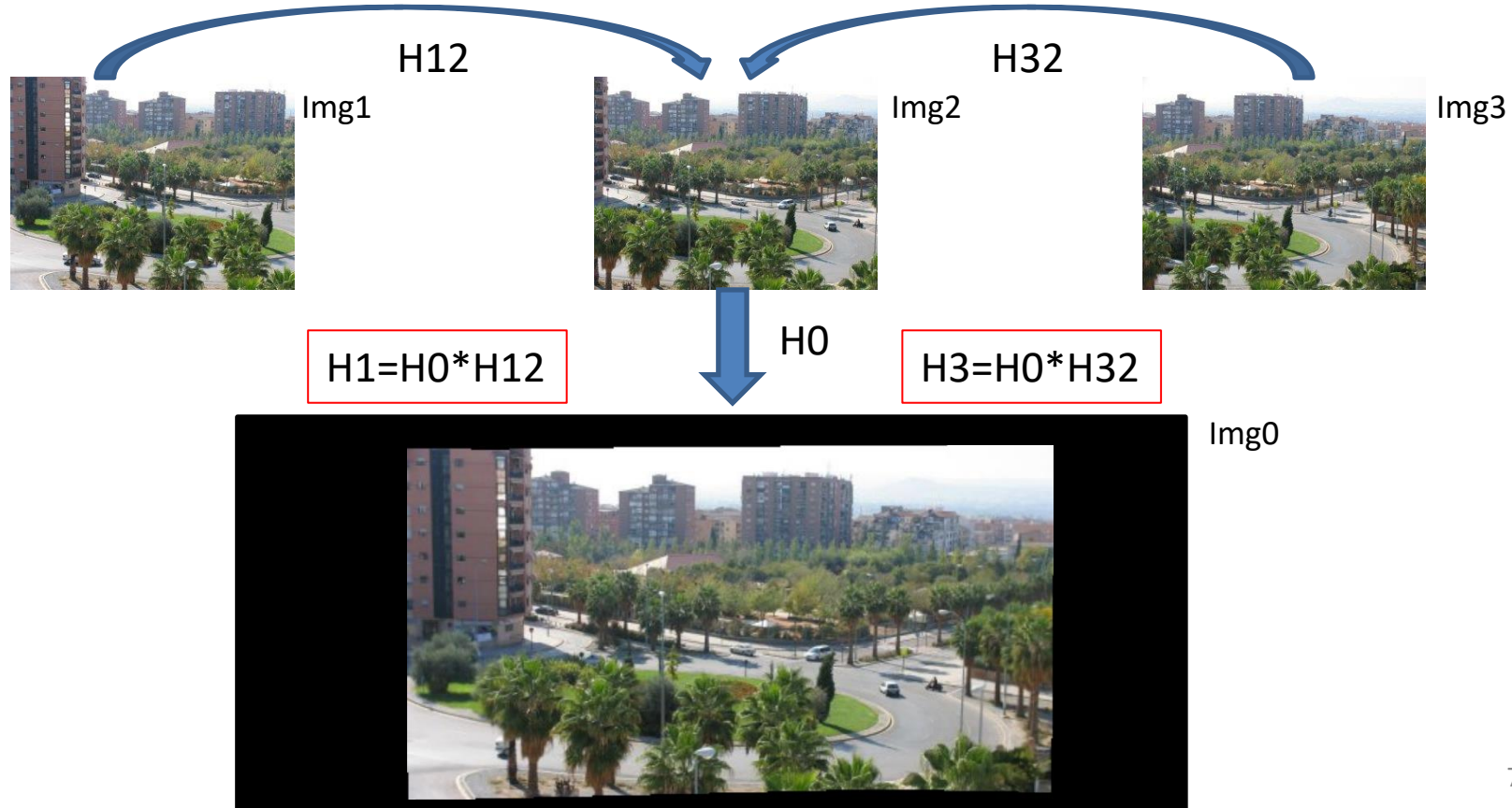
- a) Extract the SIFT KeyPoints and establish a set of points in correspondences between each two overlapping images
- b) Estimate the homography between the images from these correspondences → `cv2.findHomography()`
- c) Compose and visualize the resulting mosaic → `cv2.warpPerspective()` with `cv.BORDER_TRANSPARENT`



Exercise 4: calculation of homographies and composition of a mosaic (3 points)



Exercise 4: calculation of homographies and composition of a mosaic (3 points)



Exercise 4: calculation of homographies and composition of a mosaic (3 points)

- The base homography (H_0 , mere traslation to the centre of the canvas) can be calculated manually. You do not need to calculate keypoints and we provide it directly (*axesHomography()*)

```
H0 = np.array([[1, 0, canvas_width//2 - image.shape[1]//2],  
               [0, 1, canvas_height//2 - image.shape[0]//2],  
               [0, 0, 1]])
```

Note: Probably, if the panorama is cut off, it is because this homography is not correct or the canvas is not big enough...

Exercise 4: calculation of homographies and composition of a mosaic (3 points)

- To improve the final quality of the resulting panorama, the excess black bands should be removed.
 - `cv2.boundingRect(img)` is used in `blackOut(img)`

https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html



Exercise 4: calculation of homographies and composition of a mosaic (3 points)

- About *cv2.warpPerspective()*
 - Applies a perspective transformation to an image

Image to transform

The output/destination image is our canvas

```
cv2.warpPerspective(image_left, H_left2canvas, (width_canvas, height_canvas), dst=canvas,  
                    flags=cv2.INTER_CUBIC, borderMode = cv2.BORDER_TRANSPARENT)
```

H_left2canvas is the homography resulting from the composition of bringing the image of the left to the centre (H_left2center) and from the centre to the canvas (H0) → $H_left2canvas = H0 * H_left2center$

Note: the order in which we «fill» the canvas is important.
We want to insert the least transformed image at the end.

Exercise 4: calculation of homographies and composition of a mosaic (3 points)

Keys Ideas:

- **Composition of homographies:** The transformation to be applied can be the composition of several homographies, and the more homographies are involved, the more the image will be deformed.
 - We should try to combine as few homographies as possible → **we should go from the extremes to the central image, and not from one extreme to the other.**

Note: the “central” image does not necessarily have to be the one in the centre (of the ordered list of photos). Imagine, for example, that when taking pictures from left to right, there are many more pictures on one side than on the other. In that case the mosaic would be more distorted on the side with fewer pictures.

Exercise 4: calculation of homographies and composition of a mosaic (3 points)

Keys Ideas:

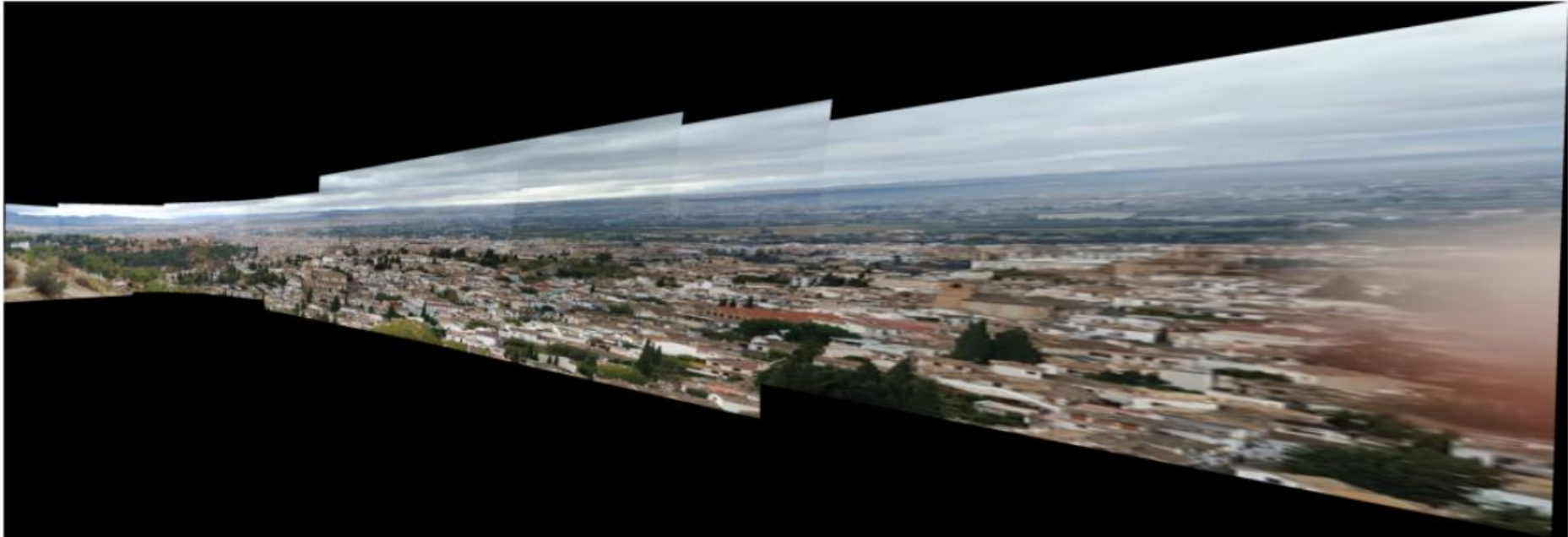
- **Composition of homographies:** The transformation to be applied can be the composition of several homographies, and the more homographies are involved, the more the image will be deformed.
 - We should try to combine as few homographies as possible → **we should go from the extremes to the central image, and not from one extreme to the other.**
- **Filling the canvas:** We should try to get the highest quality mosaic we can.
 - **We must “paste” the images from the ends/boundaries to the centre of the canvas, and paste, as the last image, the central image** (which will only require a translation)

Exercise 4: calculation of homographies and composition of a mosaic (3 points)



Exercise 4: calculation of homographies and composition of a mosaic (3 points)

What happens if we do not choose the “central” image well?



Exercise 4: calculation of homographies and composition of a mosaic (3 points)

What happens if we do not choose the “central” image well?



Useful References

- Basic concepts of the homography explained with code:
https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html
- Image Matching: <https://ai.stanford.edu/~syyeong/cvweb/tutorial2.html>
- Local features: detection and description.
https://www.cs.utexas.edu/~grauman/courses/trento2011/slides/Monday_LocalFeatures.pdf
- Relevant OpenCV documentation:
 - https://docs.opencv.org/4.5.4/d7/d60/classcv_1_1SIFT.html
 - https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html
 - https://docs.opencv.org/4.4.0/db/d27/tutorial_py_table_of_contents_feature2d.html
 - <https://theailearner.com/tag/cv2-integral/>
 - https://docs.opencv.org/master/d7/d1b/group_imgproc_misc.html

Computer Vision: Assignment 3

Feature Extraction and Image Stitching

Pablo Mesejo and Víctor Vargas

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA

