

DTU





Daniel Olesen, DTU Space

30540 – Photogrammetry (4)

Note til 2023

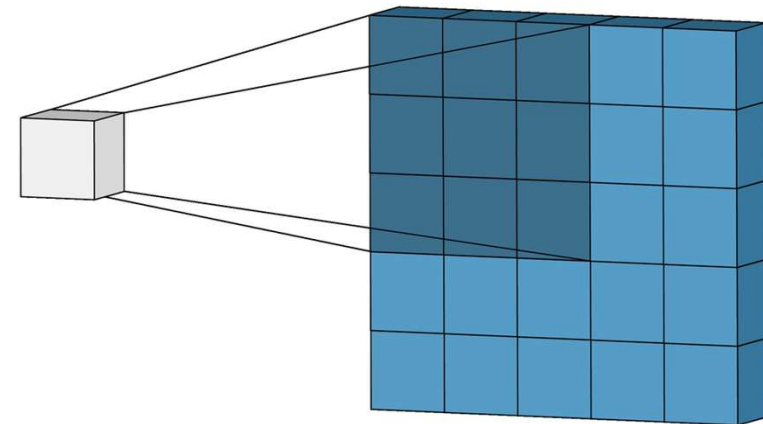
- Forklar hvad SSD (sum-squared distance) er for bedre at indikere hvordan Harris Corner detector virker
- Gå lidt mere i detaljer omkring scale-space extrema detection for SIFT
 - Mange spørgsmål i 2022 omkring hvordan denne del virkede.

Today's Lecture

- Kernel operations on images
 - Smoothing, sharpening, edge detection
- Image Features
 - Feature Detection (Principles of Harris Corners and SIFT)
 - Feature Description
 - Feature Matching
- Outlier Rejection
 - RANSAC

Kernel operations

- A kernel or convolutional matrix is commonly used for filter operations on images
- It turns out that convolving an image with different kernels (typically 3 x 3 matrices) can produce a number of different effects, e.g.,
 - Estimate image derivatives (edges)
 - Smoothing/blurring an image
 - Sharpening an image



filtered image kernel Input image

$$g(x, y) = k * f(x, y) = \sum_{d_x=-1}^1 \sum_{d_y=-1}^1 k(d_x, d_y) f(x - d_x, y - d_y)$$

[https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

2D convolution



filtered image \swarrow kernel \swarrow Input image \swarrow

$$g(x, y) = k * f(x, y) = \sum_{d_x=-1}^1 \sum_{d_y=-1}^1 k(d_x, d_y) f(x - d_x, y - d_y)$$

Example:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = a \cdot 9 + d \cdot 6 + g \cdot 3 + b \cdot 8 + e \cdot 5 + h \cdot 2 + i \cdot 1 + f \cdot 4 + c \cdot 7$$

Kernel functions (Sharpening and smoothening)

Operation	Kernel ω	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Gaussian blur 3×3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

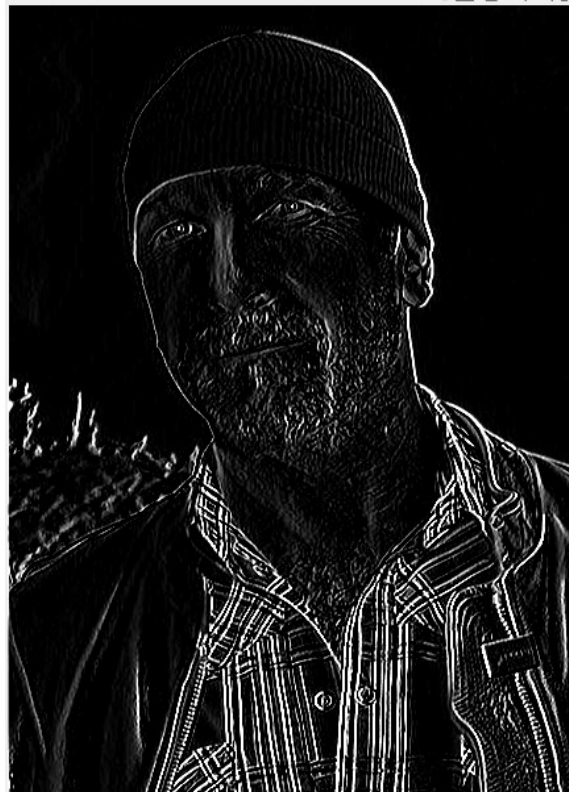
Kernel functions (Sobel Edge Detector)

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy



MATLAB: $I^* = \text{imfilter}(I, \text{kernel})$

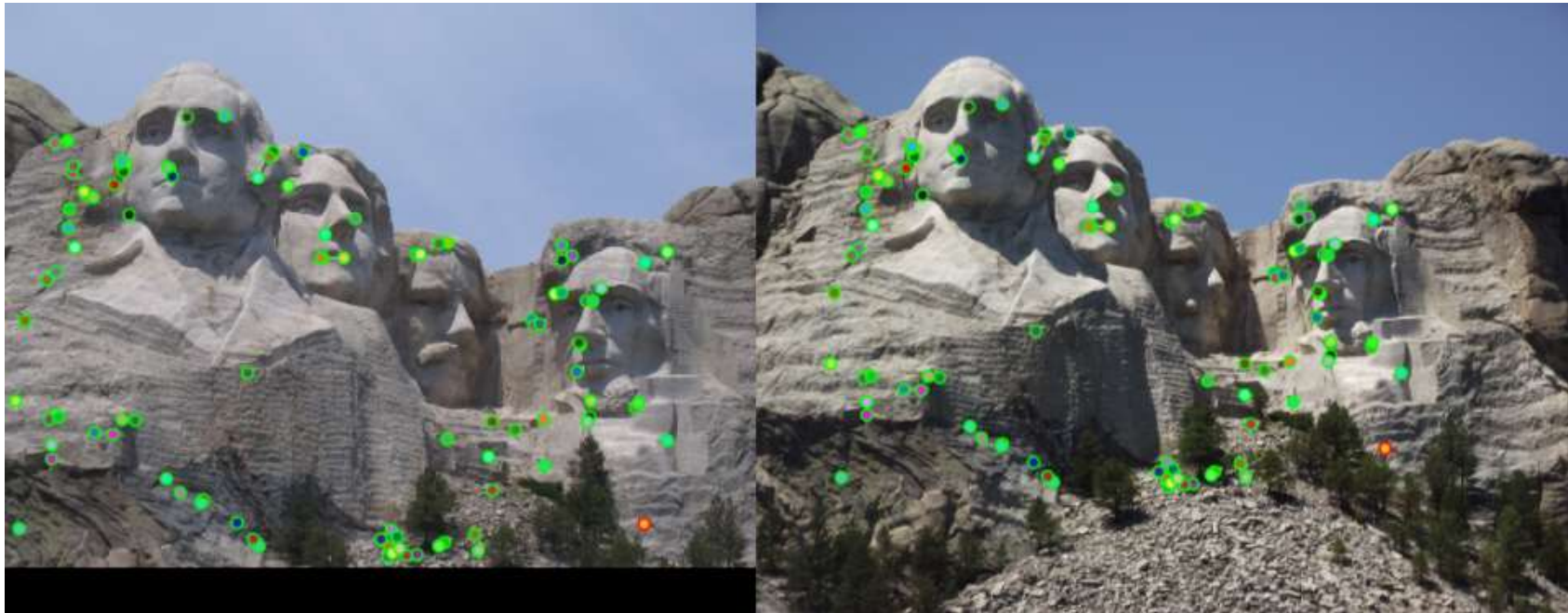
Class exercise: Gaussian blur and edge-detection (10-15 minutes)

- Download the matlab file 'class_exercise.m' and the image 'puppy-gdcfea9a60_640.jpg' from DTU Learn
- In the script, fill out the kernels G_x and G_y as given from the previous slide
- Experiment with various values of sigma, i.e., 1, 2, 5 and 10 for the Gaussian smoother
 - What happens with the image?
 - What happens with the kernel size
- Discuss your findings in small groups

Definition of Image features

- Features are recognizable structures and serve as a compact and robust description of the environment.
- low-level features (geometric primitives) can be: lines, corner points, blobs, circles or polygons.
- high-level features (semantic objects) can be: signs, doors, tables, or trash cans.
- Features are used to be able to match points between different view-points (for e.g. mosaicing, triangulation, object detection/tracking etc.)

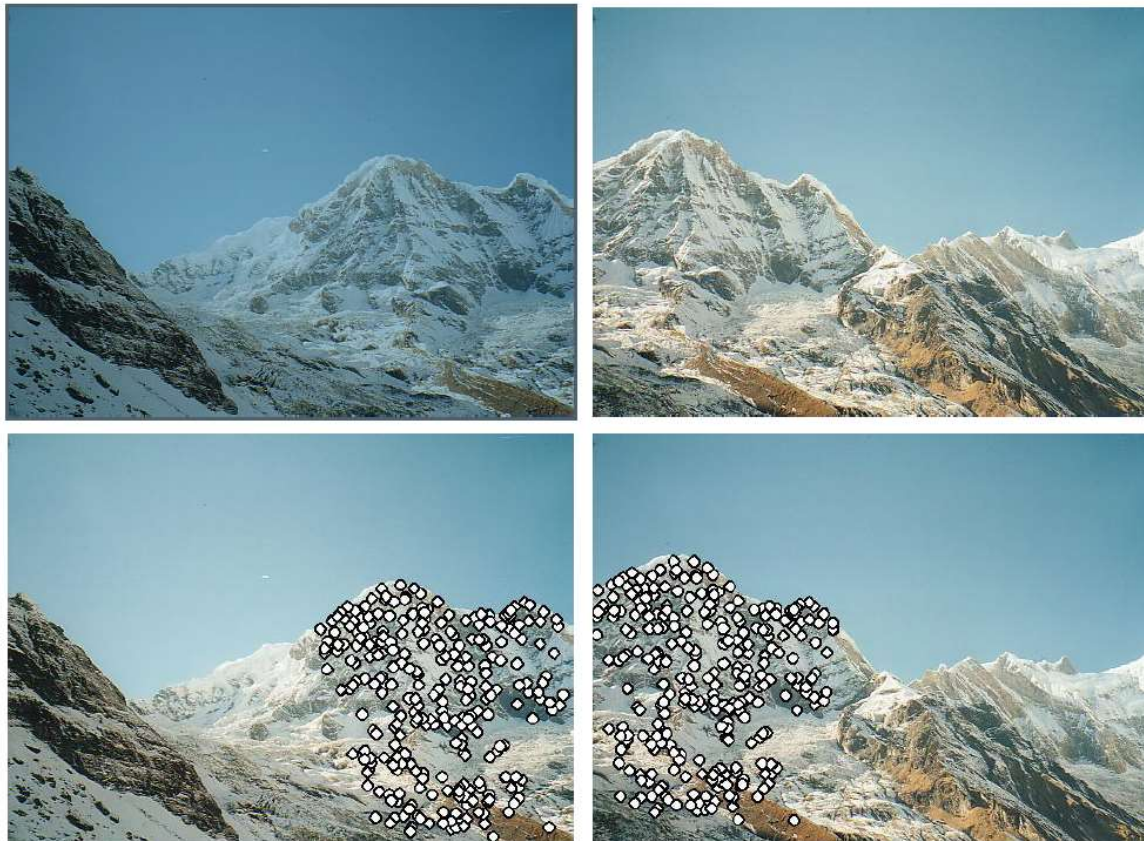
Image Features



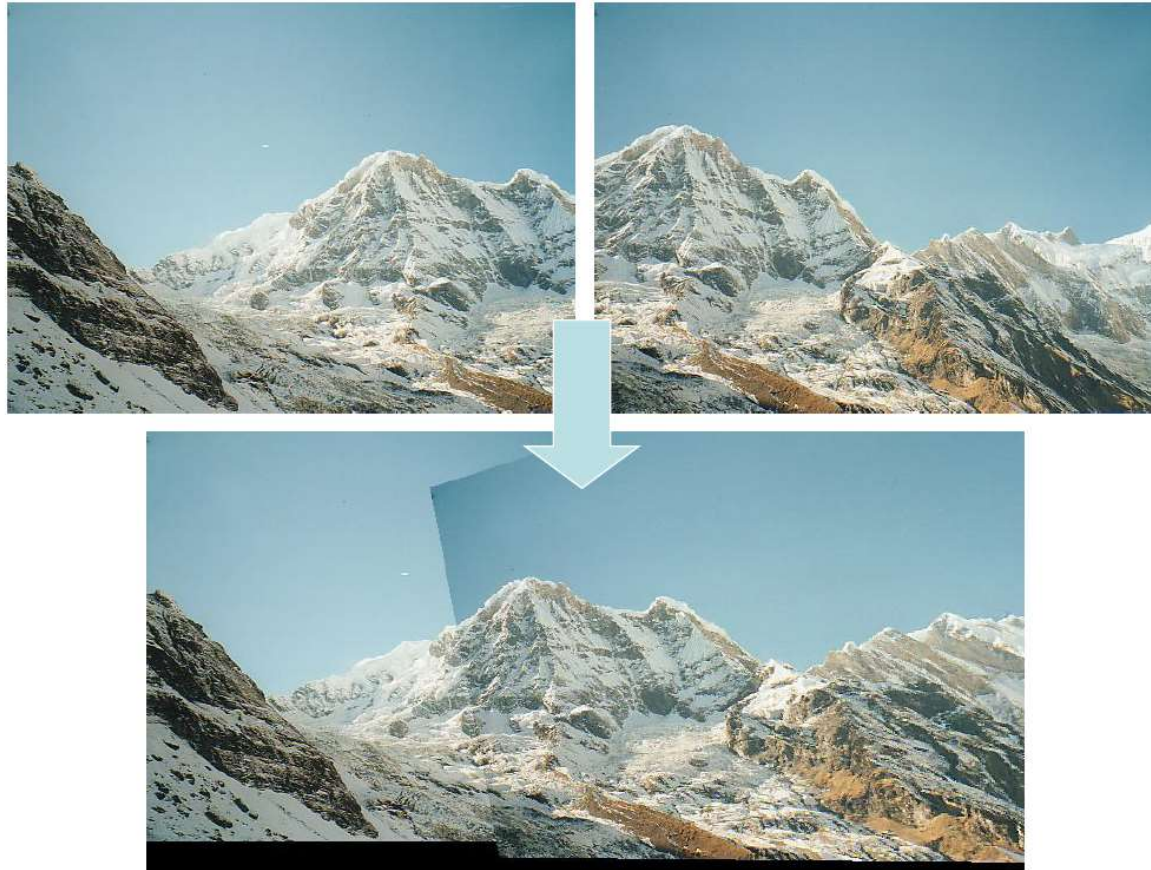
Examples of distinctive features in two images of Mt. Rushmore – notice the different size and illumination

<https://cclaassen3.github.io/computer-vision-local-feature-matching/>

Mosaicing (Image stitching)



Mosaicing (Image Stitching)



Low-level features: Edges

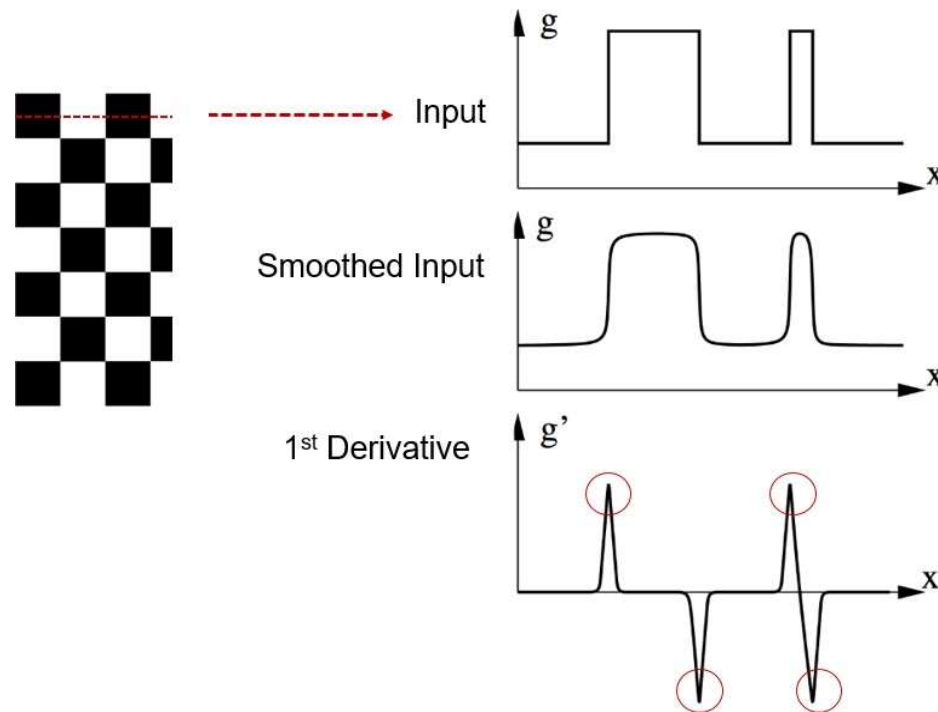
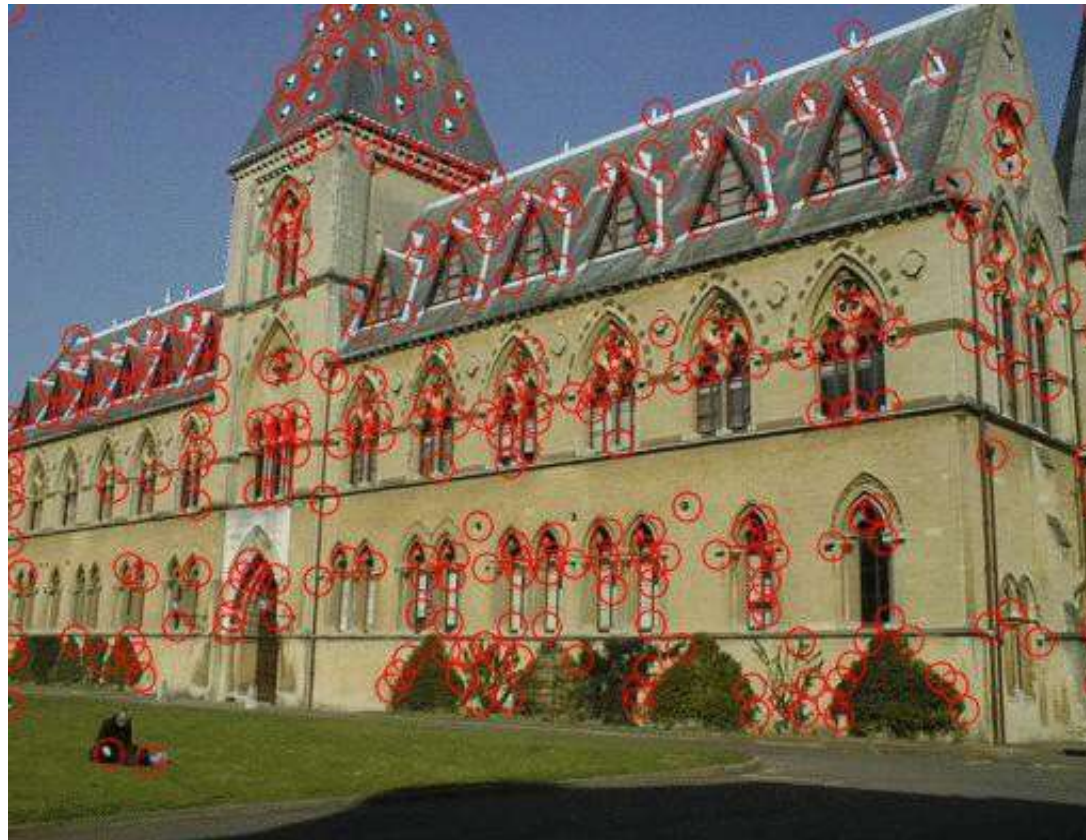


Figure from: Förstner, Scriptum Photogrammetry I, Chapter "Kantenextraktion".

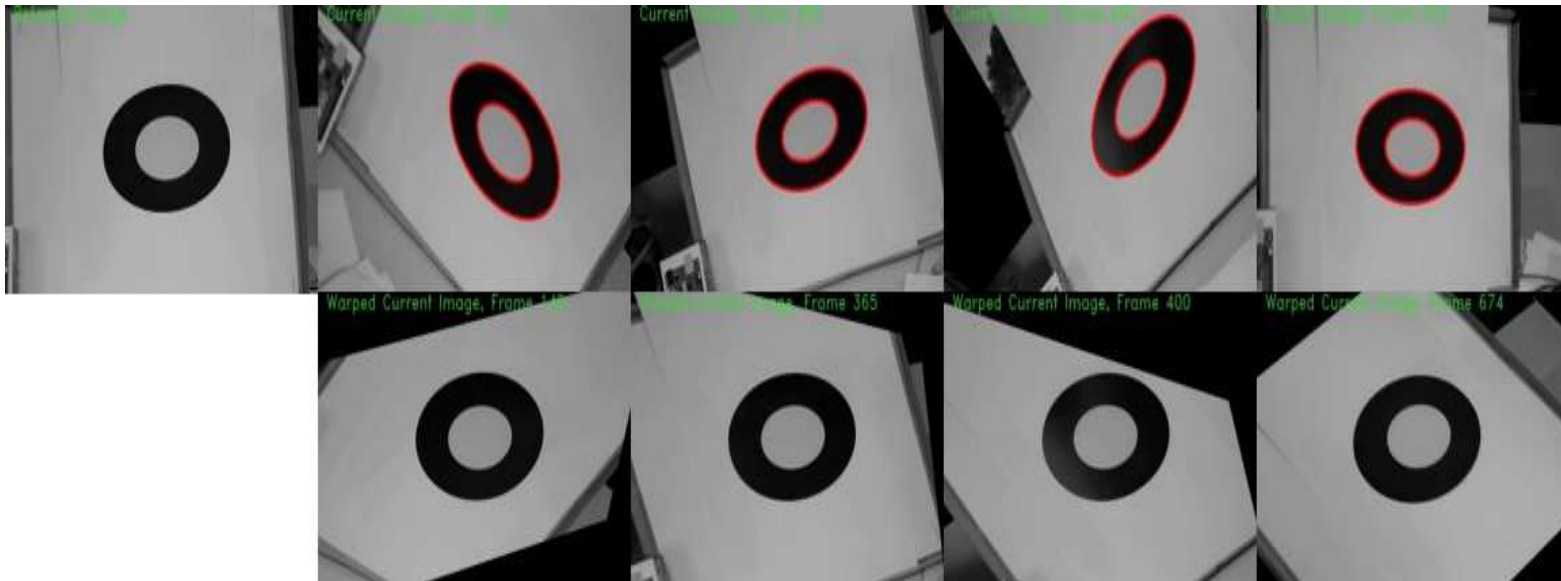
Vertical and horizontal edges



Low-level features: corner points



Low-level features: Conics



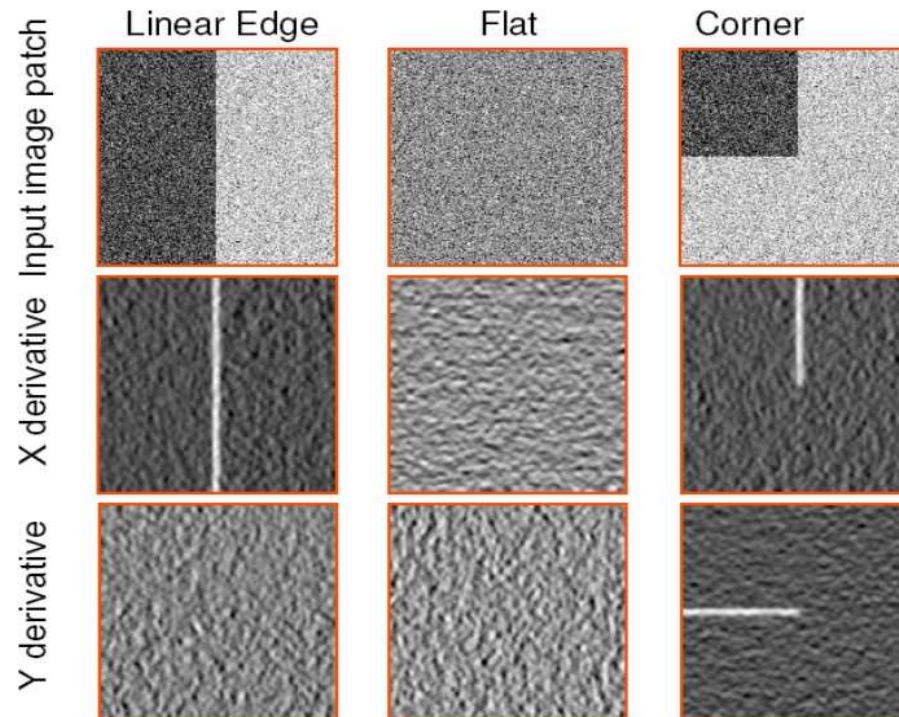
Feature Detection: Keypoint Detectors

Properties of the ideal feature detector:

- Repeatability: can be redetected regardless view/illumination changes: rotation, scale (zoom), and illumination invariant.
- Distinctiveness: easy to distinguish and match.
- Localization accuracy.
- Computational efficiency.
- Two major groups/categories of feature detectors:
 - Corner detectors: Harris[2], FAST[3], etc.
 - Blob feature detectors: SIFT

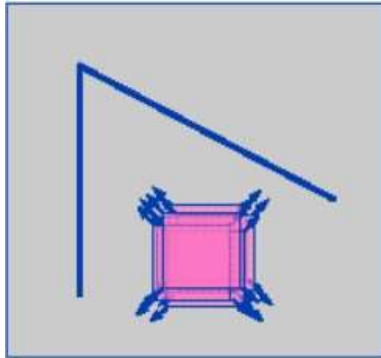
Corner detector: Harris

- A corner in an image can be defined as the intersection of two or more edges (lines).

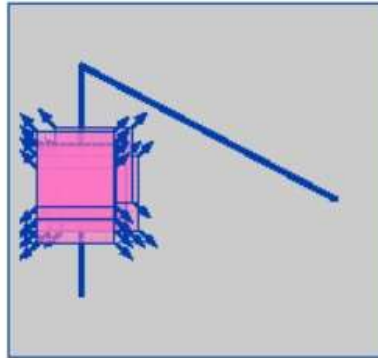


Corner detector: Harris

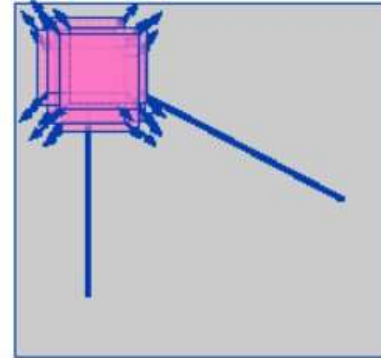
- A corner in an image can be defined as the intersection of two or more edges (lines).



“flat” region:
no change in
all directions

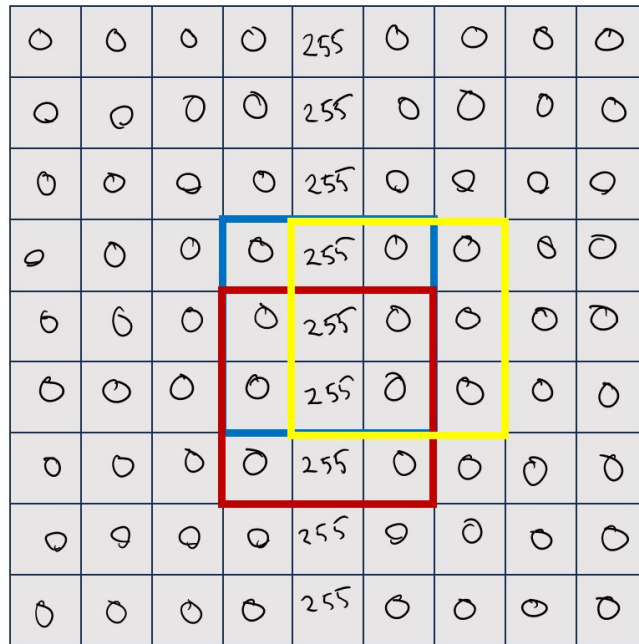


“edge”:
no change along
the edge direction



“corner”:
significant change
in all directions

Corner detector: Harris (Sliding window across an edge)



Blue square is the considered image patch. The red square shows the sliding window displaced vertically (-1) and the yellow square is the sliding window displaced horizontally (+1)

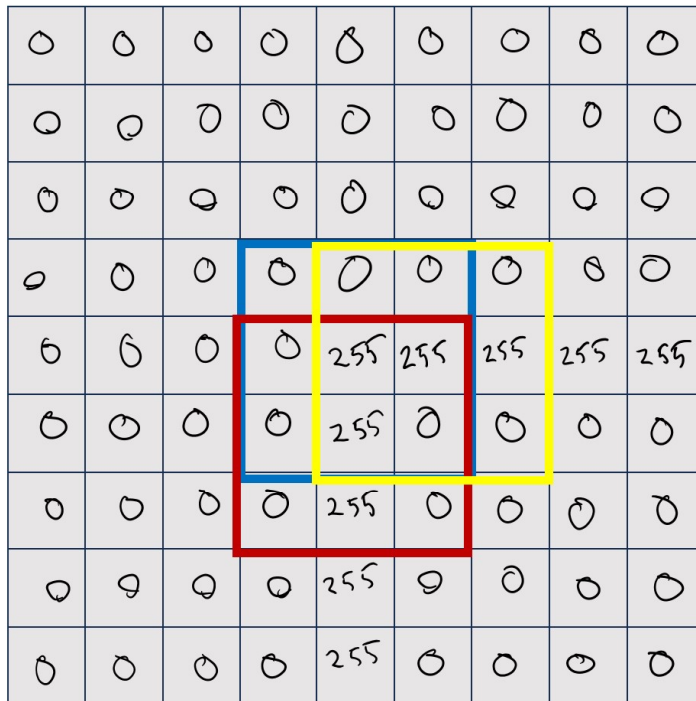
Difference between image patch and sliding window
(vertical movement)

$$\begin{bmatrix} 0 & 255 & 0 \\ 0 & 255 & 0 \\ 0 & 255 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 255 & 0 \\ 0 & 255 & 0 \\ 0 & 255 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Difference between image patch and sliding window
(horizontal movement)

$$\begin{bmatrix} 0 & 255 & 0 \\ 0 & 255 & 0 \\ 0 & 255 & 0 \end{bmatrix} - \begin{bmatrix} 255 & 0 & 0 \\ 255 & 0 & 0 \\ 255 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -255 & 255 & 0 \\ -255 & 255 & 0 \\ -255 & 255 & 0 \end{bmatrix}$$

Corner detector: Harris (Sliding window across a corner)



Blue square is the considered image patch. The red square shows the sliding window displaced vertically (-1) and the yellow square is the sliding window displaced horizontally (+1)

Difference between image patch and sliding window
(vertical movement)

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 255 & 255 \\ 0 & 255 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 255 & 255 \\ 0 & 255 & 0 \\ 0 & 255 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -255 & -255 \\ 0 & 0 & 255 \\ 0 & 0 & 0 \end{bmatrix}$$

Difference between image patch and sliding window
(horizontal movement)

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 255 & 255 \\ 0 & 255 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 255 & 255 & 255 \\ 255 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ -255 & 0 & 0 \\ -255 & 255 & 0 \end{bmatrix}$$

Corner detector: Harris

- We define a sliding window function which is swept across all pixels in the image.

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

Shifted intensity
 Pixel intensity

Where W is the window we are considering, $[u, v]$ are the displacement in x- and y-directions respectively. Using Taylor approximations, we can conclude that:

$$I(x + u, y + v) \approx I(x, y) + uI_x(x, y) + vI_y(x, y)$$

Where I_x, I_y are the x-and y-derivates of I . Hence we can write,

$$E(u, v) = [u \quad v] \sum_{x, y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = [u \quad v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

Corner detector: Harris

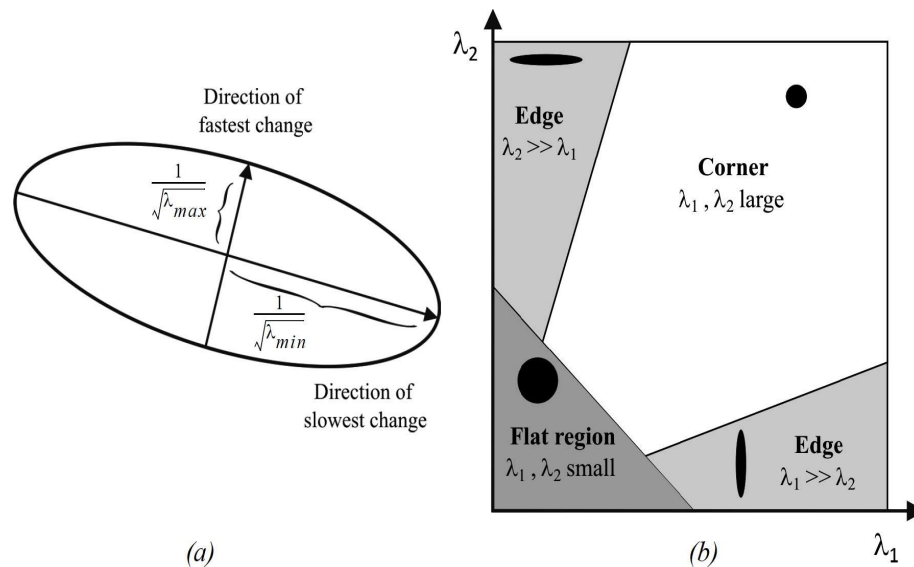


Figure: The Harris detector analyses the eigenvalues of M to verify whether current point is a corner or not

For easy computation, the following “cornerness function” is actually used:

$$C = \det(M) - k \cdot \text{trace}(M)^2$$

Harris corner point is then found as the local maximum.

Harris keypoints are **rotation/translation invariant, but not scale invariant.**

Corner detector: Harris

Original Image



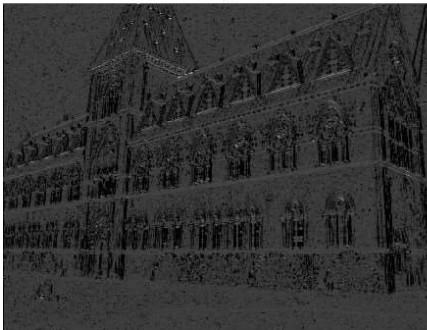
I_x



I_y



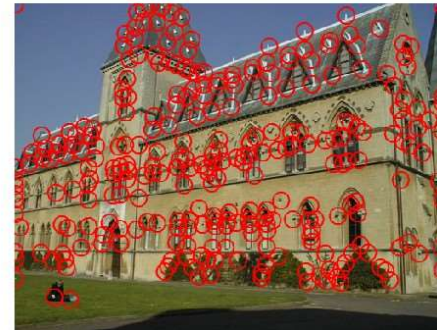
I_{xy}



C



Harris



Feature Descriptors

- How do we find corresponding features between images of the same objects?
 - We need to find a way to describe a feature point in a distinctive way and look for a match in corresponding images.



125	240	085	123	235
240	152	231	015	214
236	110	147	153	120
074	054	063	083	024
114	156	178	123	201

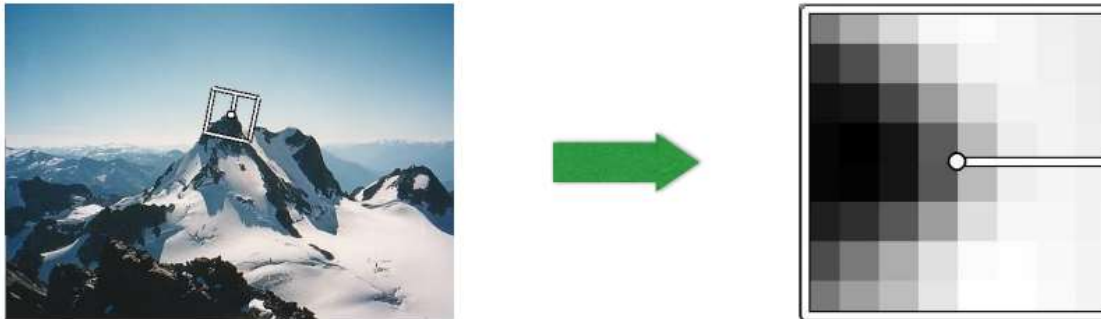
5 x 5 Image patch

Simple idea: Use intensity values in the close neighbourhood of the feature point (e.g. 5 x 5 patch) and uses this patch for finding matches in corresponding images.

What are the potential problems with this approach?

Feature Descriptors

- A simple intensity patch will not be invariant to rotation, scale and illumination.
- A better approach is to use pixel differences (image gradients) as this would make the descriptor more invariant to illumination changes.
- What about rotation invariance?
 - We could use the dominant gradient direction and normalize orientation of patch, i.e.



<http://www.cs.cmu.edu/~16385/s18/lectures/lecture6.pdf>

Scale-Invariant Feature Transform (SIFT)

- Scale Invariant Feature Transform(SIFT) is the most popular blob feature detector and descriptor, which is widely used in mapping and navigation, etc thanks to its robustness to rotation, scale and illumination changes.
- Main steps of the SIFT algorithm:
 - Perform multi-scale extrema detection
 - Find dominant orientation of keypoint patch
 - Generate keypoint descriptor

So, SIFT = keypoint detector + descriptor.

Find keypoint location and scale

- Build up scale space for input image (a)
- Compute Difference of Gaussian (DoG)
- Find local extrema across adjacent scales and select the highest response compared to adjacent pixels in the same scale and the surrounding scales.

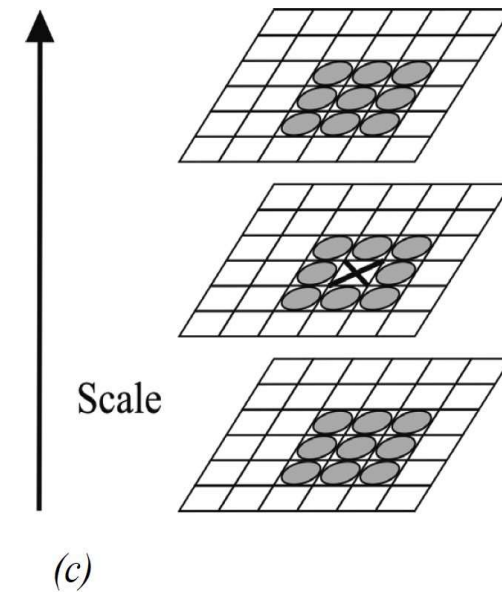
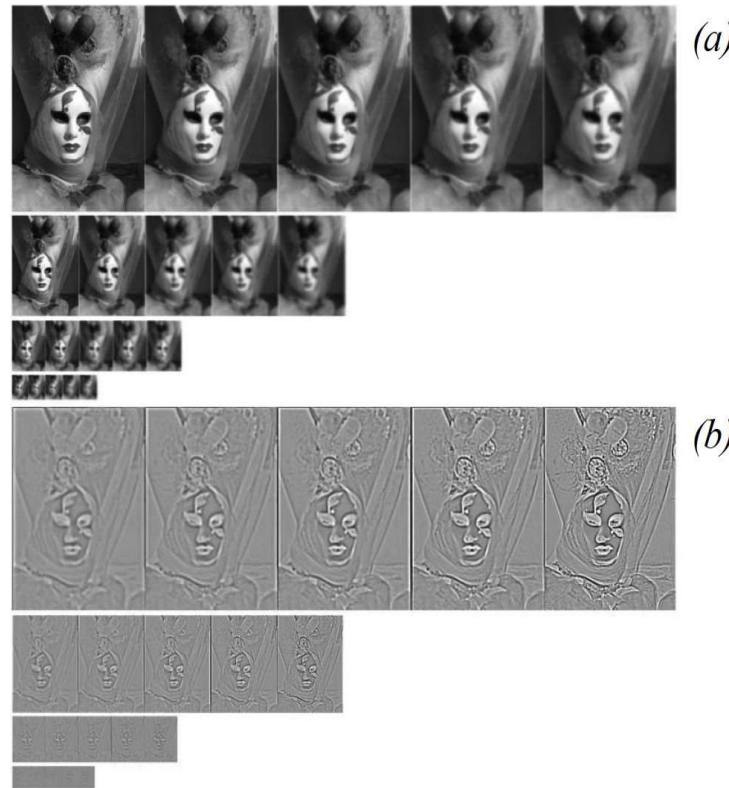
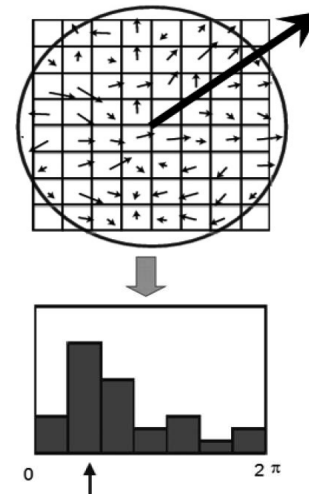


Figure: Scale space and DoG

Find Dominant Orientation

The dominant orientation is used to make the descriptor rotation invariant.

- SIFT compute a gradient magnitude and orientation based on a patch around the keypoint.
- The patch is hereafter normalized according to the dominant direction in order to become rotation invariant.



(a)

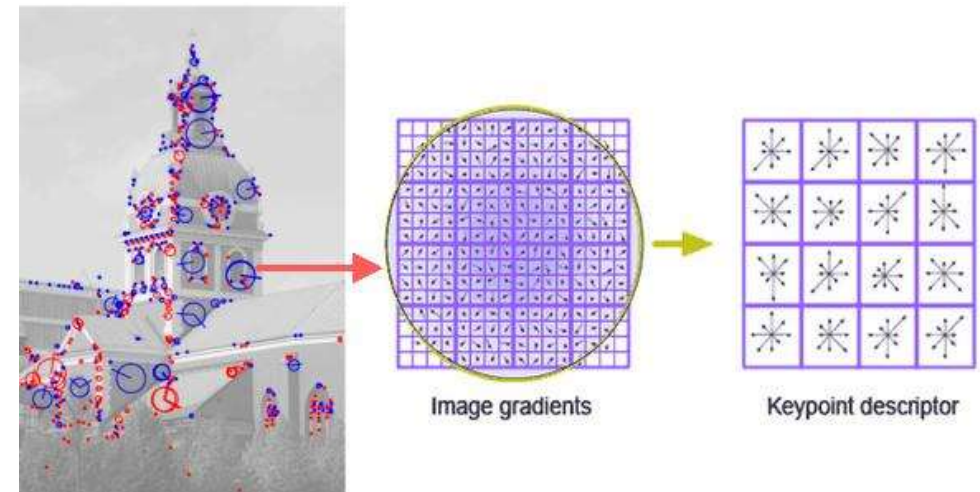


(b)

Figure: Histogram of orientation and examples of SIFT features with different scales and dominant orientations

SIFT feature descriptor

- The feature descriptor used by SIFT is based on an image patch of 16×16 pixels, distributed into 4×4 cell.
- A histogram of gradient with eight orientation bins is computed within each of the cells.
- The descriptor is then simply constructed by combining the orientation histograms of the 16 (4×4) cells and concatenating them to form a $4 \times 4 \times 8 = 128$ vector.



Summary

	Corner detector	Blob detector	Rotation invariant	Scale invariant	Affine invariant	Repeatability	Localization accuracy	Robustness	Efficiency
Harris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
Harris-Laplacian	x	x	x	x		+++	+++	++	+
Harris-Affine	x	x	x	x	x	+++	+++	++	++
SUSAN	x		x			++	++	++	+++
FAST	x		x			++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
MSER		x	x	x	x	+++	+	+++	+++
SURF		x	x	x	x	++	++	++	++

Figure: Comparison of feature detectors: properties and performance

Feature matching

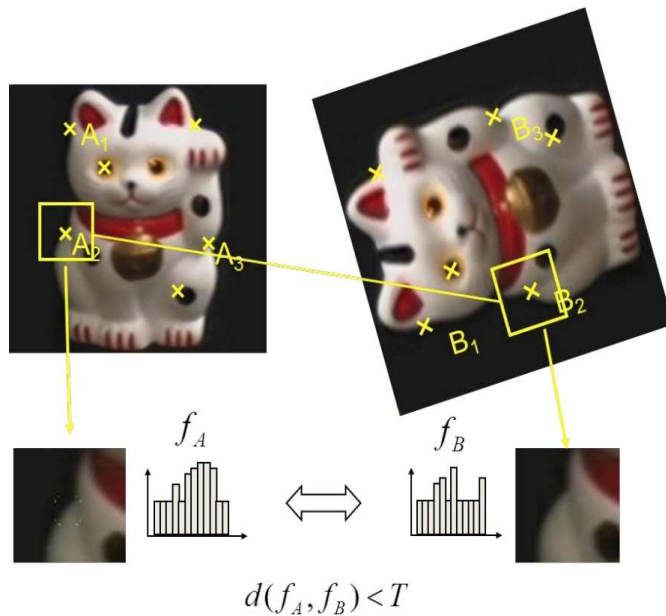


Figure: Principle of feature matching[\[11\]](#).

Different distance functions for comparing the similarity of two descriptors:

- L₂ distance, for descriptors with floating value (SIFT):

$$d(f_a, f_b) = \sum_i \|(f_a(i) - f_b(i))\|_2$$

- Hamming distance, for descriptors with binary value (BRIEF):

$$d(f_a, f_b) = \sum_i \text{XOR}(f_a(i), f_b(i))$$

- others: L₁ distance, etc.

Outlier Rejection - Ransac

Why do we need RANSAC?

- Feature matching result often contains a certain amount of false matches (outliers). Outliers will severely deteriorate or skew the estimation result.
- Generally speaking, RANdom SAMple Consensus (RANSAC) is an iterative method for estimating model parameters from observations containing outliers. The model could be a line, a parabola, an ellipse, etc.

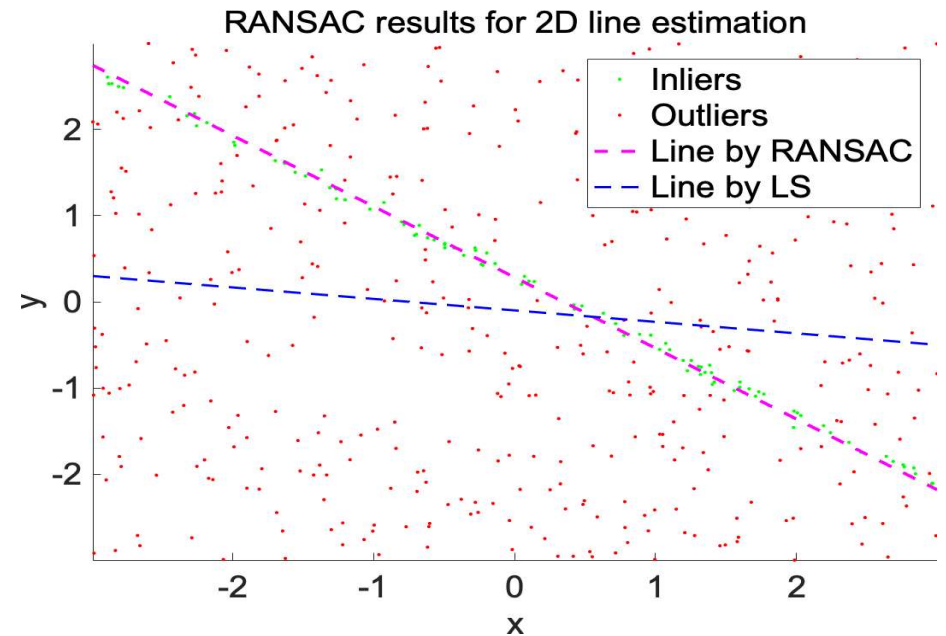


Figure: Line estimation results.

RANSAC Pipeline

- Randomly select n samples from observations, n is the minimum number of samples needed for model estimation, e.g. $n = 2$ for 2D line, $n = 3$ for 3D plane, etc.
- Model estimation, could be LS, SVD, etc.
- Compute Consensus: apply a model-specific loss function to each observation and the model obtained, the response could serve as the consensus, e.g. point-line distance, point-plane distance.
- Classifier inliers and outliers using a predefined threshold and log the model with the maximum number of inliers.
- Iterate

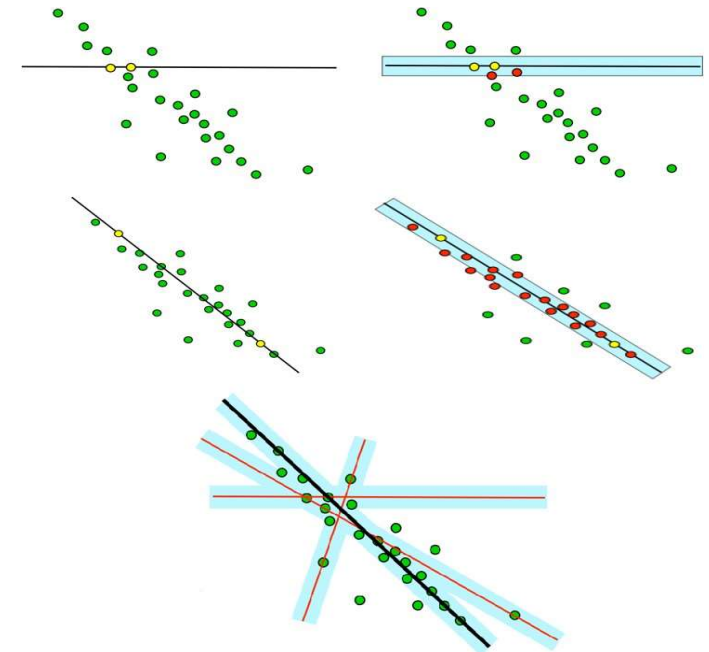


Figure: RANSAC Procedure.

How many iterations to choose?

Let p_{out} be the probability that one point is an outlier, n be the minimum number of samples for model estimation, N be the number of iterations, p be the desired probability that we get a good sample:

$$p = 1 - (1 - (1 - p_{out})^n)^N$$

- $1 - p_{out}$: Probability of an inlier.
- $(1 - p_{out})^n$: Probability of choosing n inliers.
- $(1 - (1 - p_{out})^n)$: Probability of at least one item in the sample being outliers for one iteration.
- $(1 - (1 - p_{out})^n)^N$: Probability that N iterations are contaminated.
- $1 - (1 - (1 - p_{out})^n)^N$: Probability that at least one iteration is not contaminated.

Usually, we set p and then compute N backwardly as $N = \frac{\log(1-p)}{\log(1-(1-p_{out})^n)}$.

How many iterations to choose?

A look-up table for N when p is set to 0.99.

n	p_{out}						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Feature matching before RANSAC



Figure: Feature matching before applying RANSAC.

As you can see, false matches get hypothesized. Let's apply RANSAC + epipolar constraint.

Feature matching after RANSAC

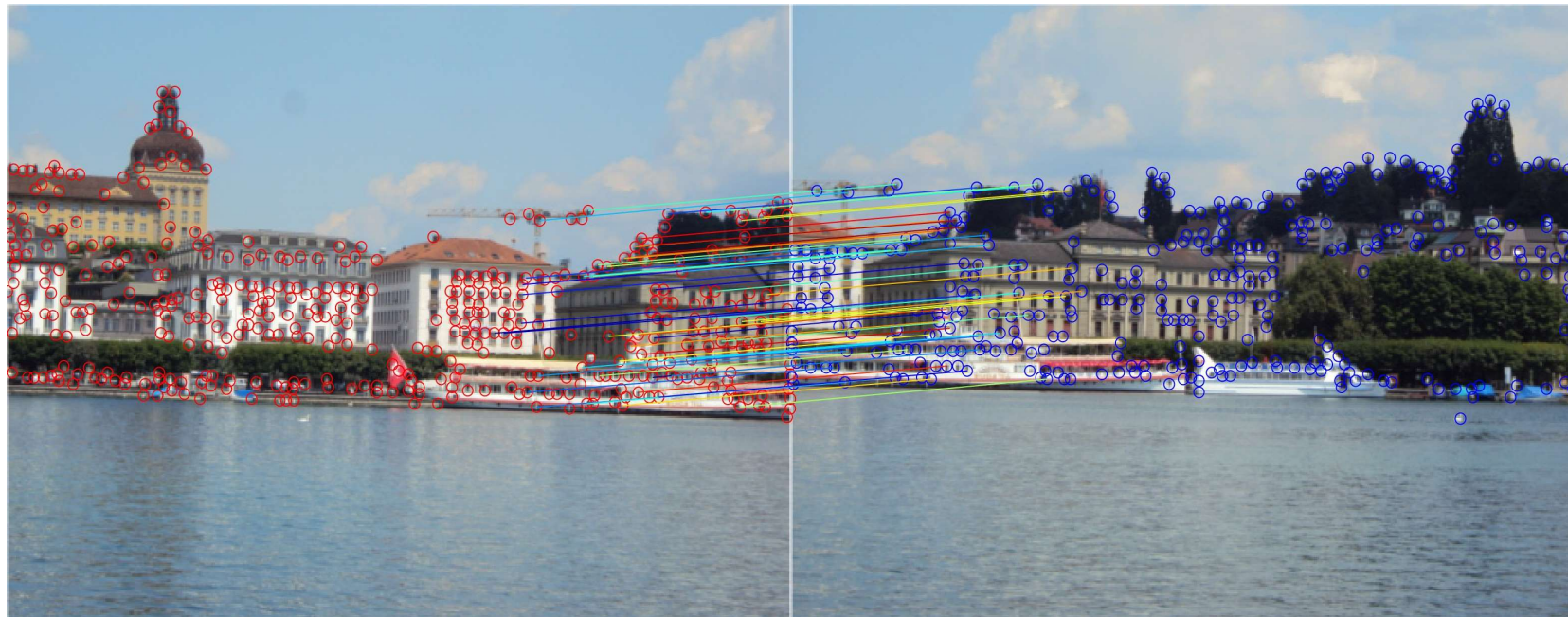


Figure: Feature matching after applying RANSAC.

As you can see, false matches are removed.