

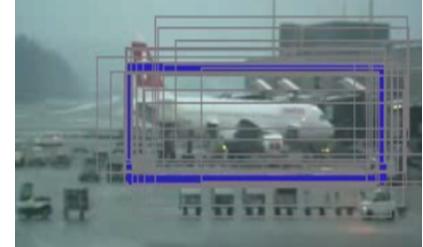
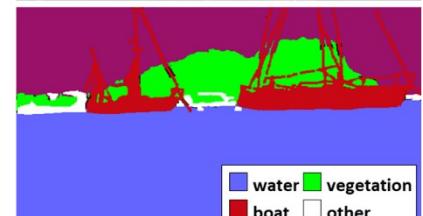
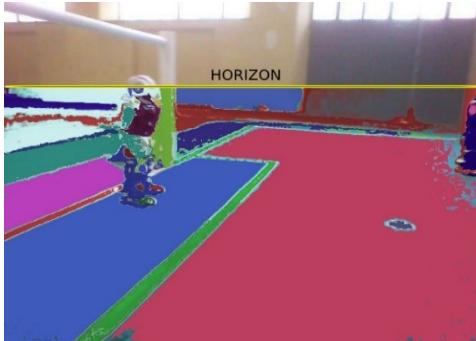
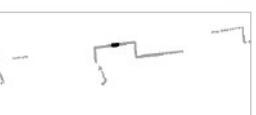


**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Visione e Percezione
A.A. 2019/2020

Feature Matching

Aprile 2020



■ water ■ vegetation
■ boat ■ other

Il corso

- Home page del corso
<http://web.unibas.it/bloisi/corsi/visione-e-percezione.html>
- Docente: Domenico Daniele Bloisi
- Periodo: Il semestre marzo 2020 – giugno 2020

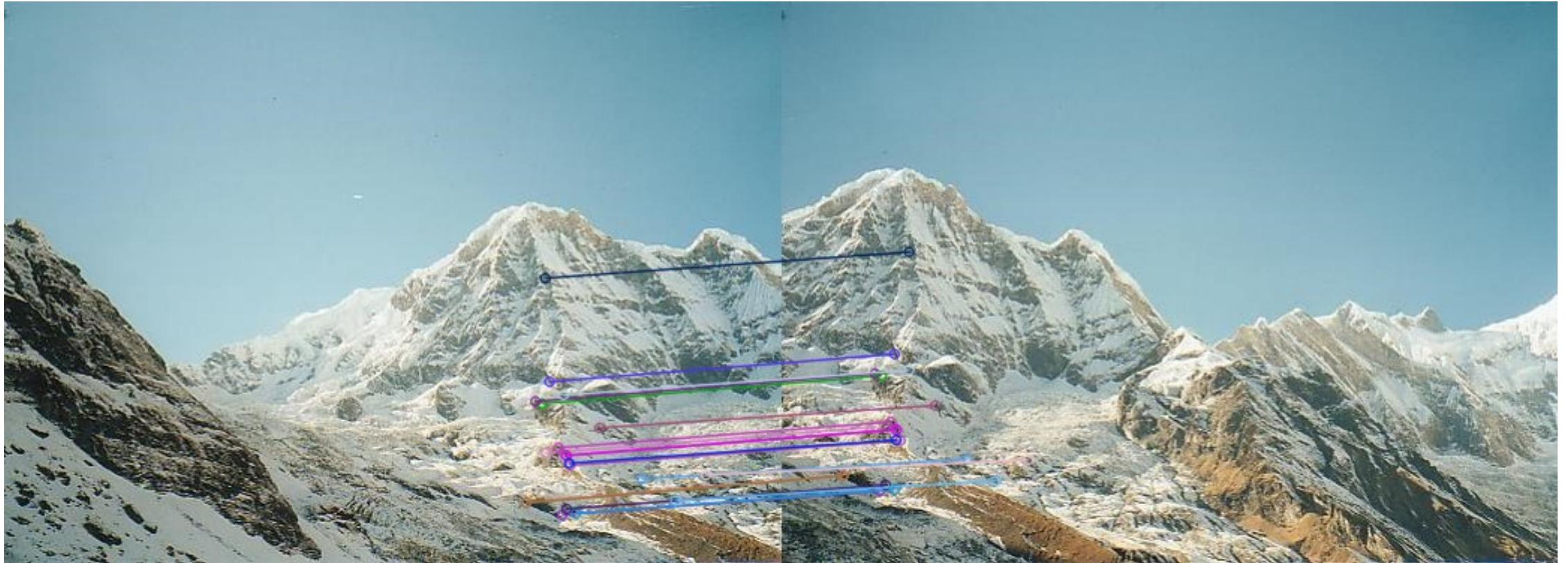
Martedì 17:00-19:00 (Aula GUGLIELMINI)

Mercoledì 8:30-10:30 (Aula GUGLIELMINI)

Riferimenti

- Queste slide sono adattate da
Noah Snavely - CS5670: Computer Vision
"Lecture 5: Feature descriptors and matching"
"Lecture 9: RANSAC"
- I contenuti fanno riferimento ai capitoli 3 e 4 del libro
"Computer Vision: Algorithms and Applications"
di Richard Szeliski, disponibile al seguente indirizzo
<http://szeliski.org/Book/>

Problem: Feature matching



Recap

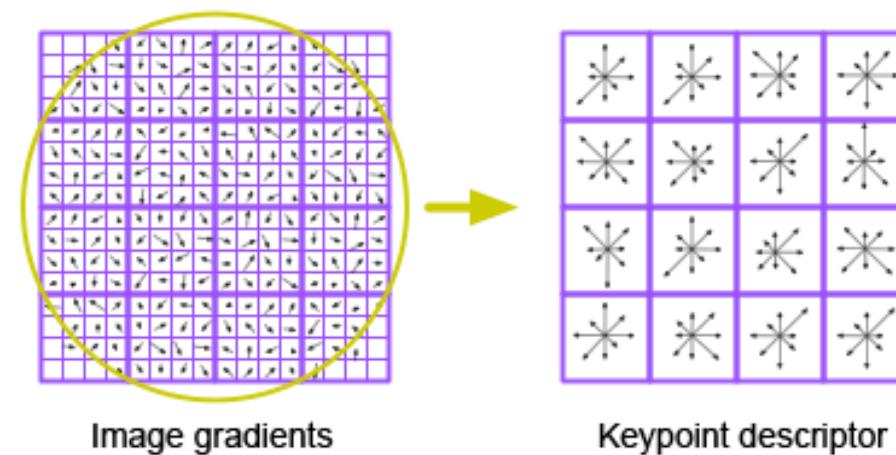
Keypoint detection: repeatable and distinctive

- Corners, blobs, stable regions
- Harris

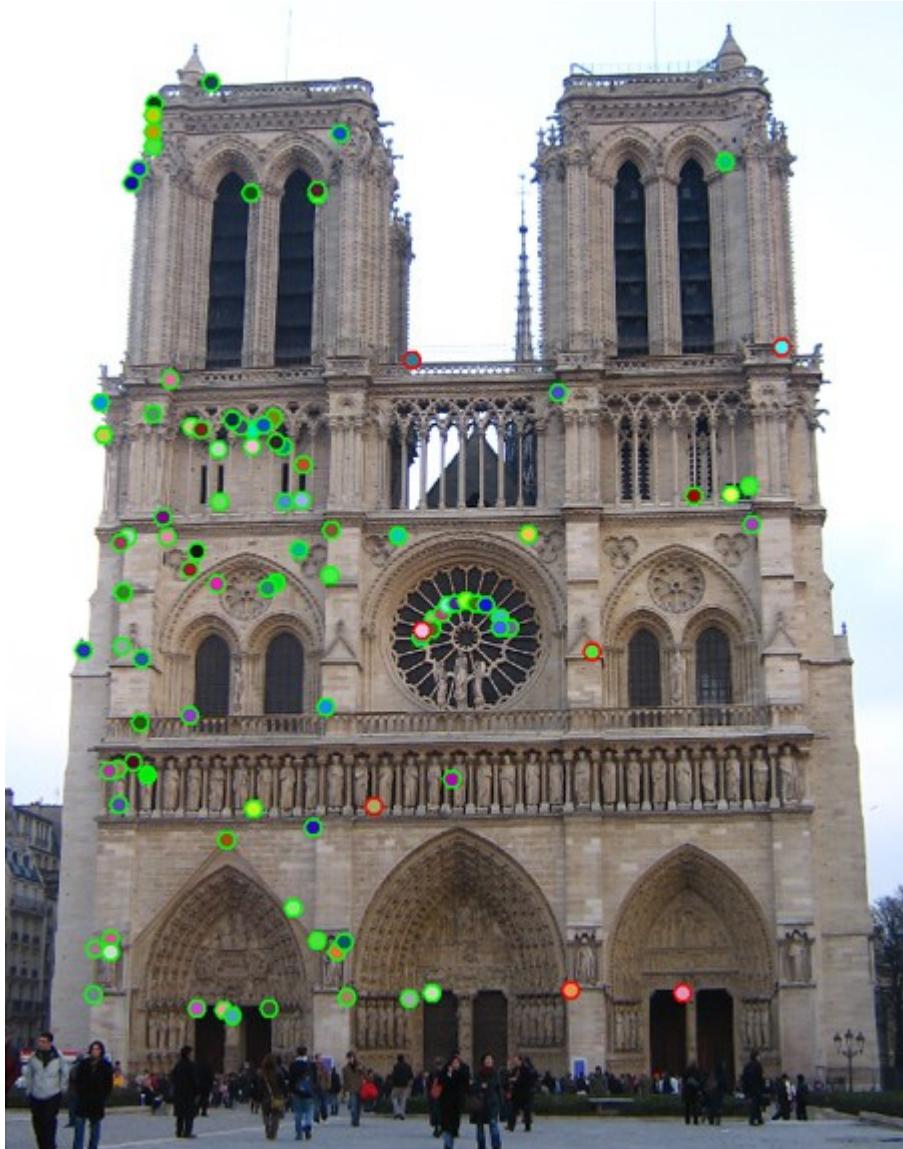


Descriptors: robust and selective

- spatial histograms of orientation
- SIFT and variants are typically good for stitching and recognition



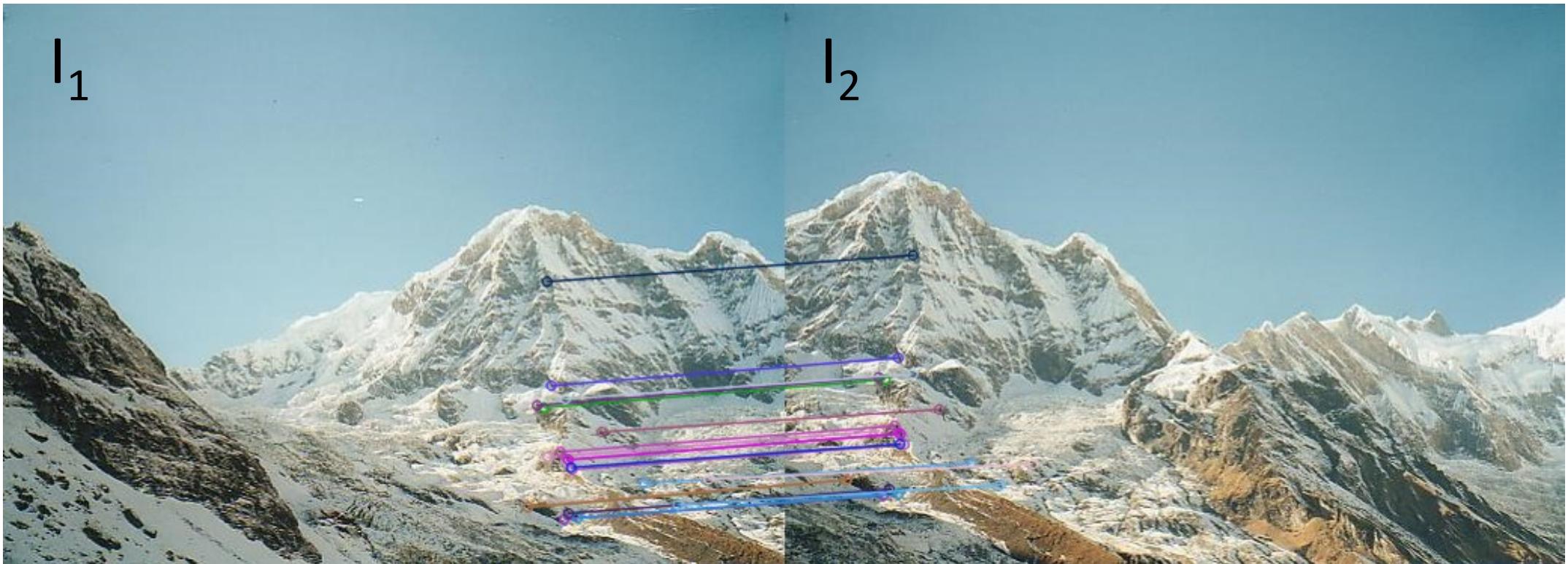
Which features match?



Features matching

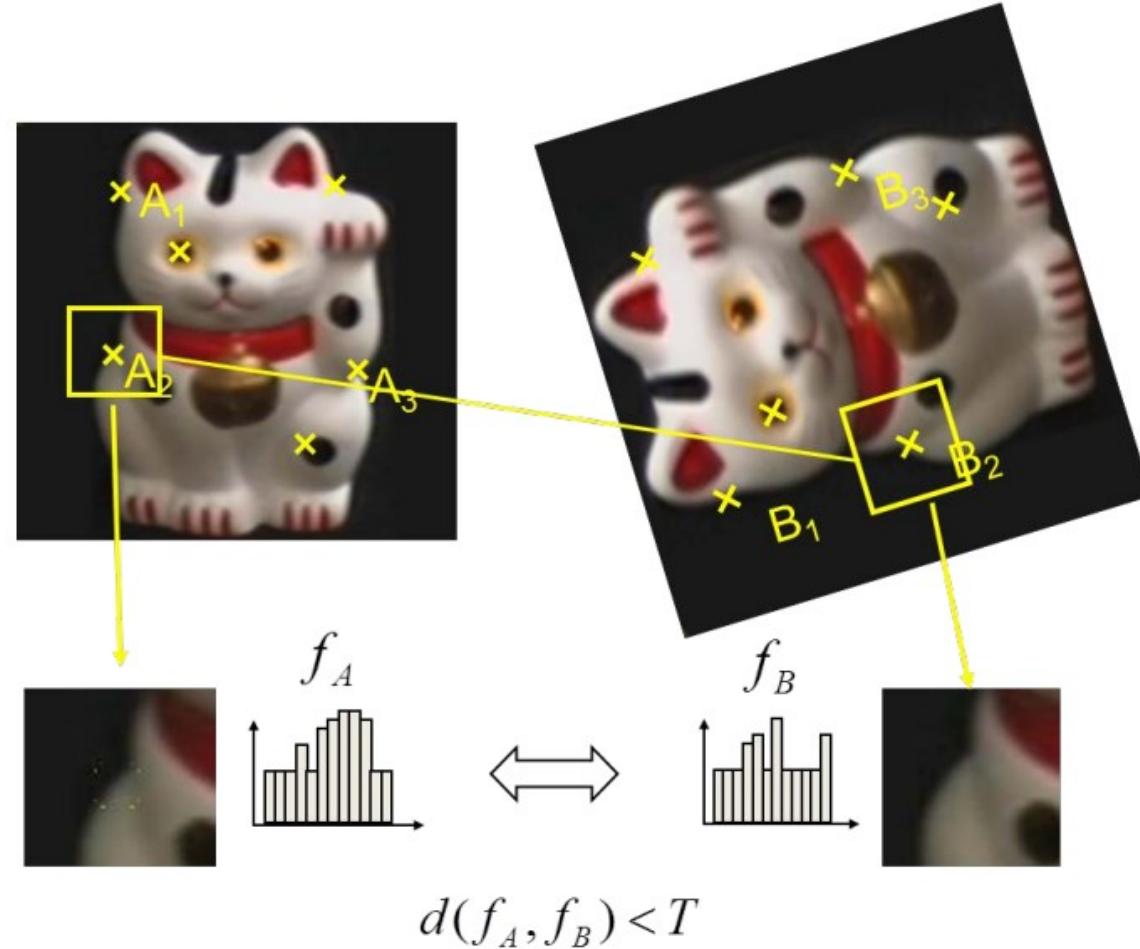
Given a feature in I_1 , how to find the best match in I_2 ?

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



Overview of point feature matching

1. Detect a set of distinct feature points
2. Define a patch around each point
3. Extract and normalize the patch
4. Compute a local descriptor
5. Match local descriptors



Source: Trym Vegard Haavardsholm

Distance between descriptors

- L_1 distance (SAD):

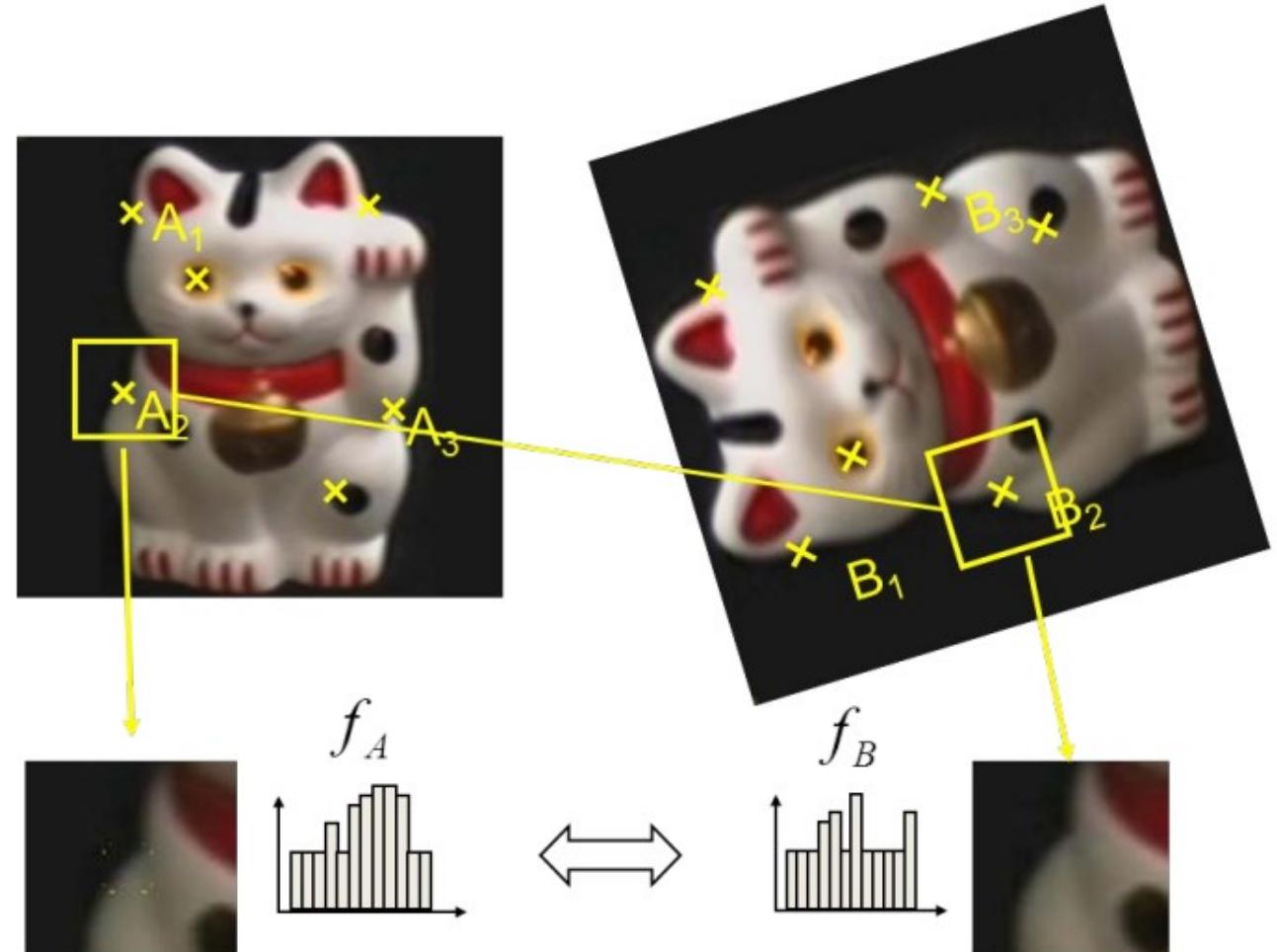
$$d(f_a, f_b) = \sum |f_a - f_b|$$

- L_2 distance (SSD):

$$d(f_a, f_b) = \sum (f_a - f_b)^2$$

- Hamming distance:

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



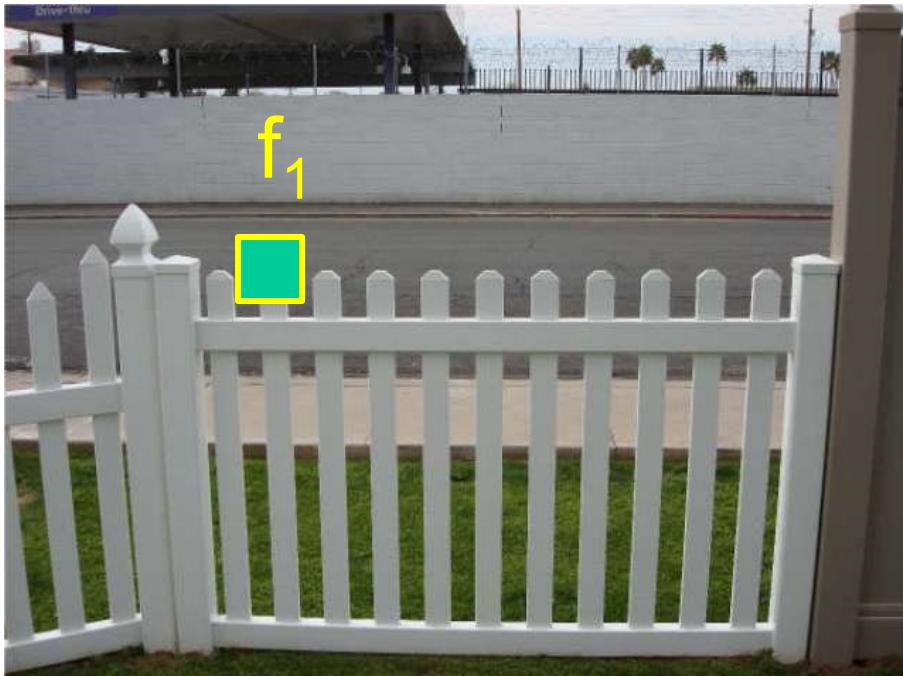
$$d(f_A, f_B) < T$$

Source: Trym Vegard Haavardsholm

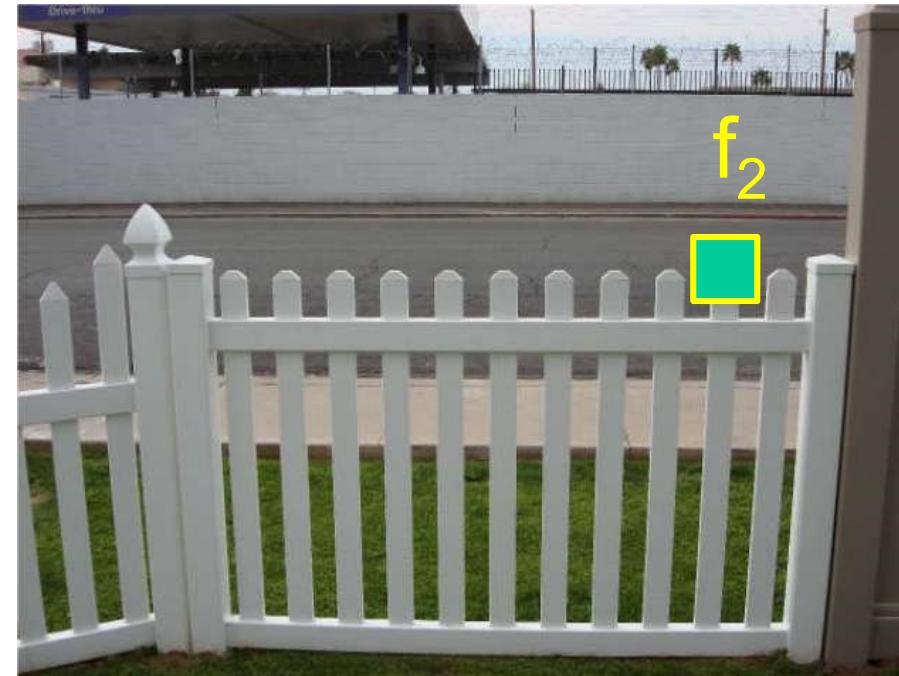
Features distance: SSD

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

- Simple approach: L_2 distance, $\|f_1 - f_2\|$
i.e., sum of square differences (SSD) between entries of the two descriptors
- can give small distances for ambiguous (incorrect) matches
i.e., does not provide a way to discard ambiguous (bad) matches



I_1

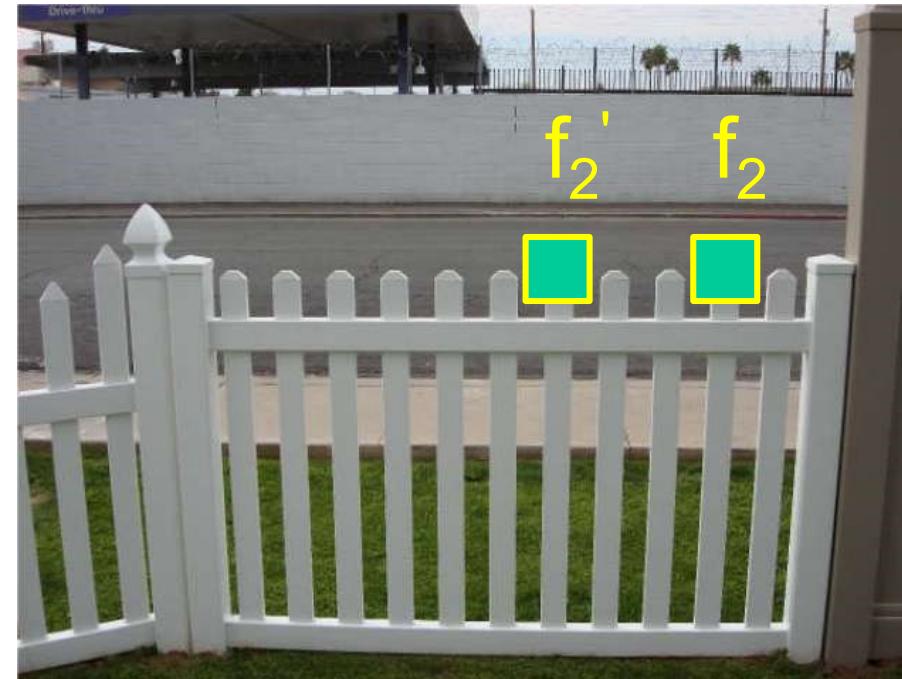
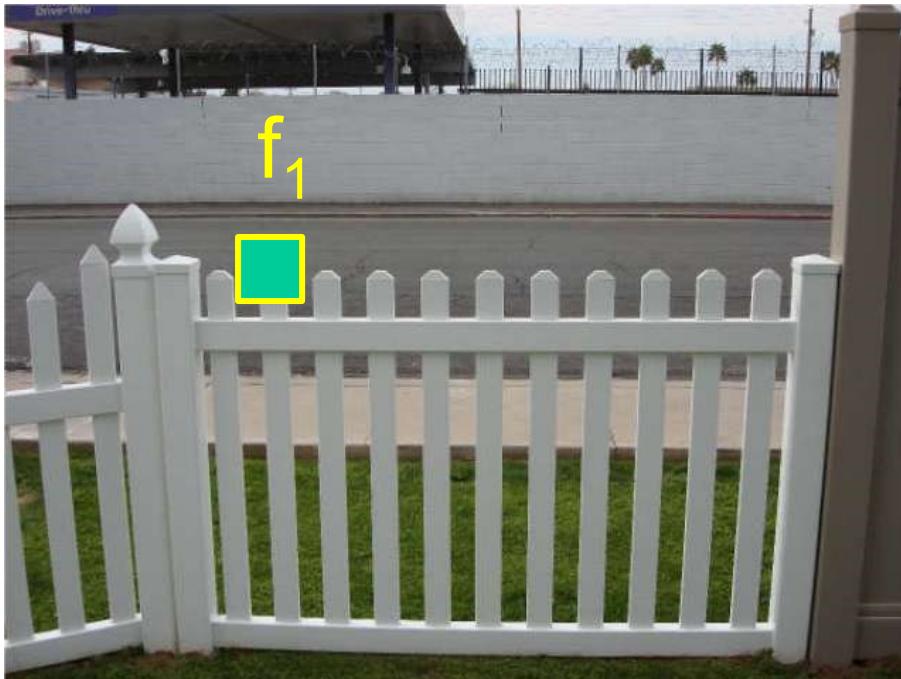


I_2

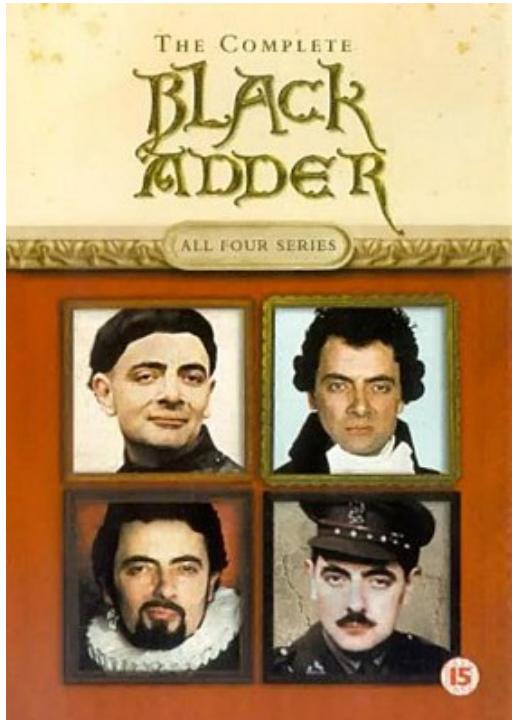
Features distance: Ratio of SSDs

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

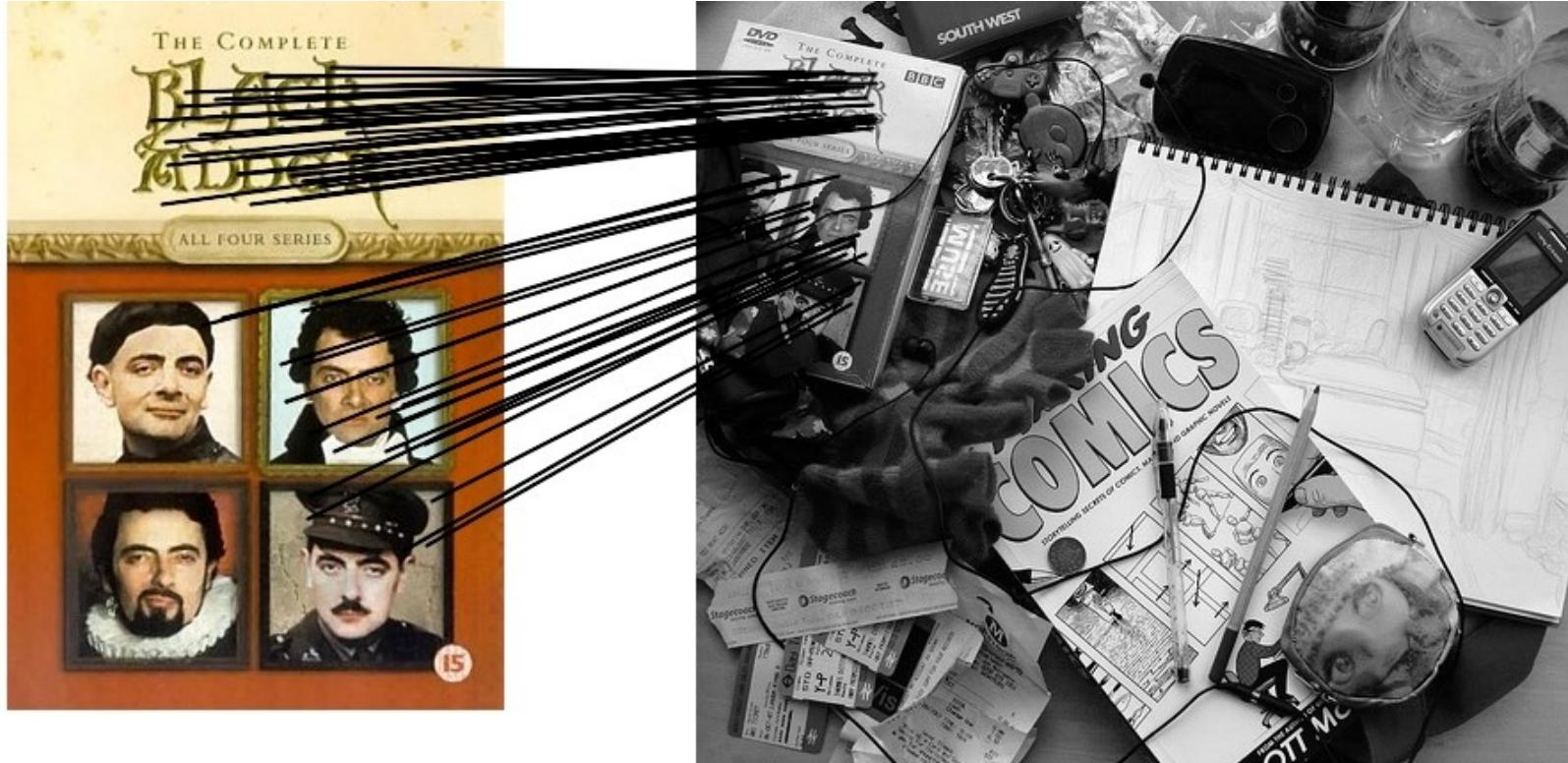
- Better approach: ratio distance = $\text{SSD}(f_1, f_2) / \text{SSD}(f_1, f_2')$
 - f_2 is best SSD match to f_1 in I_2
 - f_2' is 2nd best SSD match to f_1 in I_2
 - An ambiguous/bad match will have ratio close to 1
 - Look for unique matches which have low ratio



Feature matching example



Feature matching example



58 matches (thresholded by ratio score)

Example in Colab: ratio test

```
orb = cv.ORB_create()

kp_1, des_1 = orb.detectAndCompute(img_1, None)
kp_2, des_2 = orb.detectAndCompute(img_2, None)

bf = cv.BFMatcher()
matches = bf.knnMatch(des_1, des_2, k=2)

# Apply ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

img_3 = cv.drawMatchesKnn(img_1, kp_1, img_2, kp_2, good[:20], None, flags=2)

img_3_rgb = cv.cvtColor(img_3, cv.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(img_3_rgb)
plt.show()

cv.imwrite('matching.png', img_3)
```

`BFMatcher.knnMatch()` to get k best matches.
In this example, we will take k=2 so that we can apply ratio test

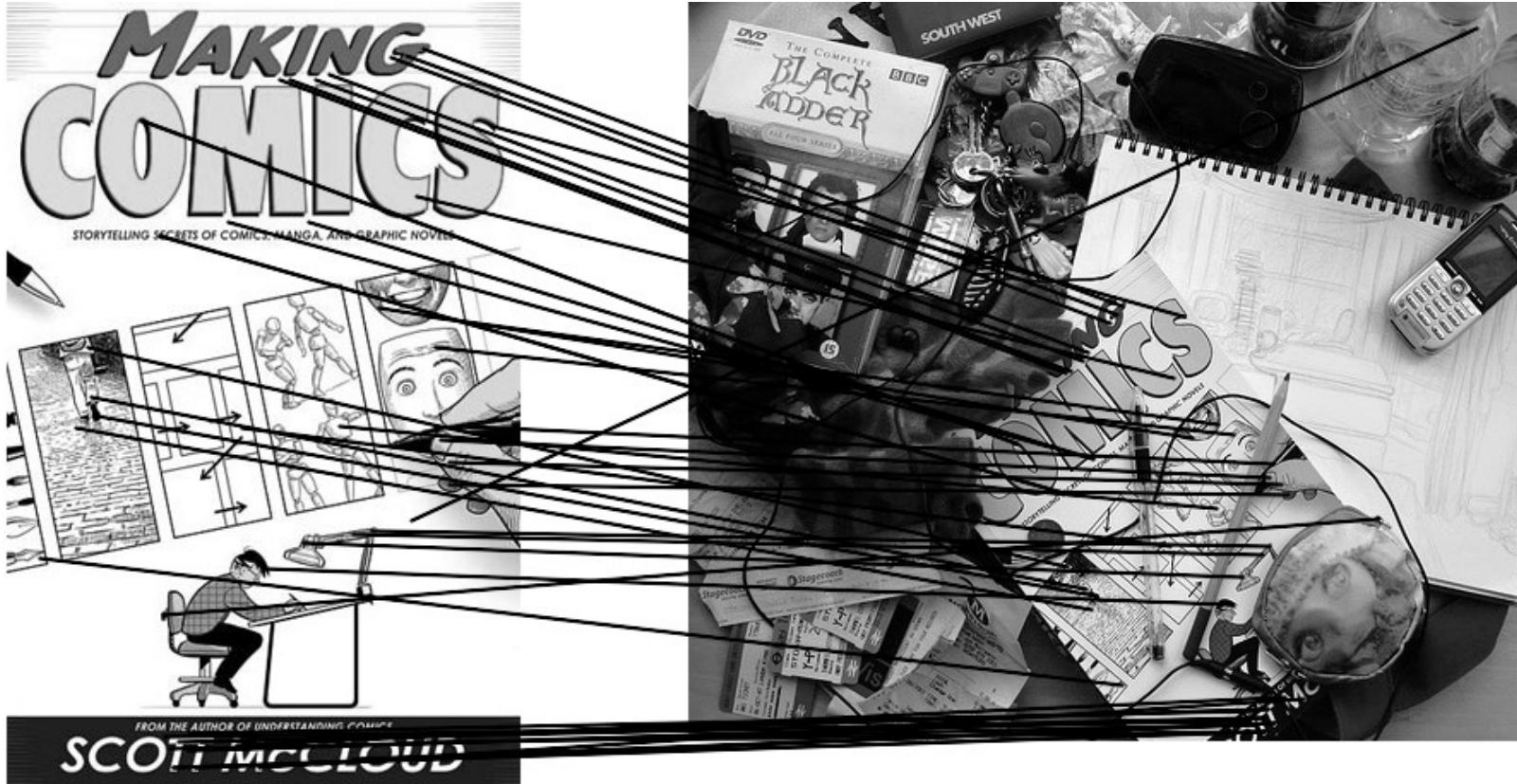
Example in Colab: ratio test results



Feature matching example

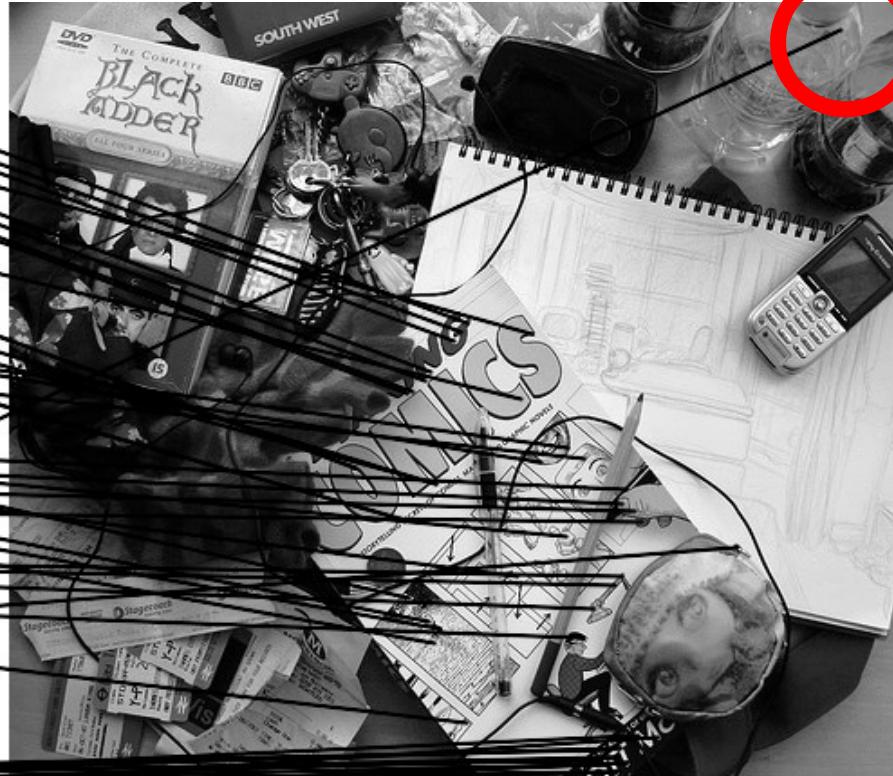
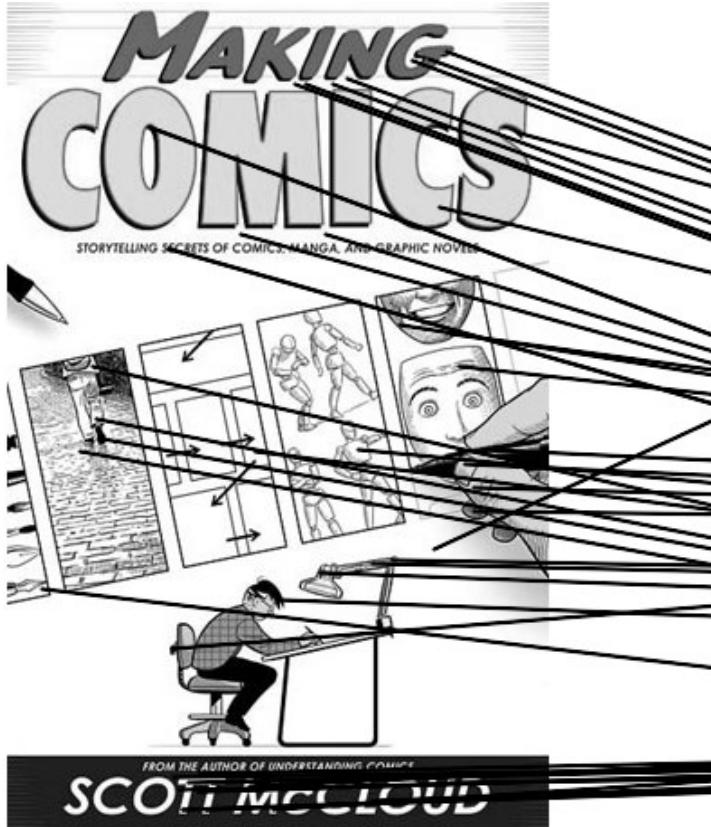


Feature matching example



51 matches (thresholded by ratio score)

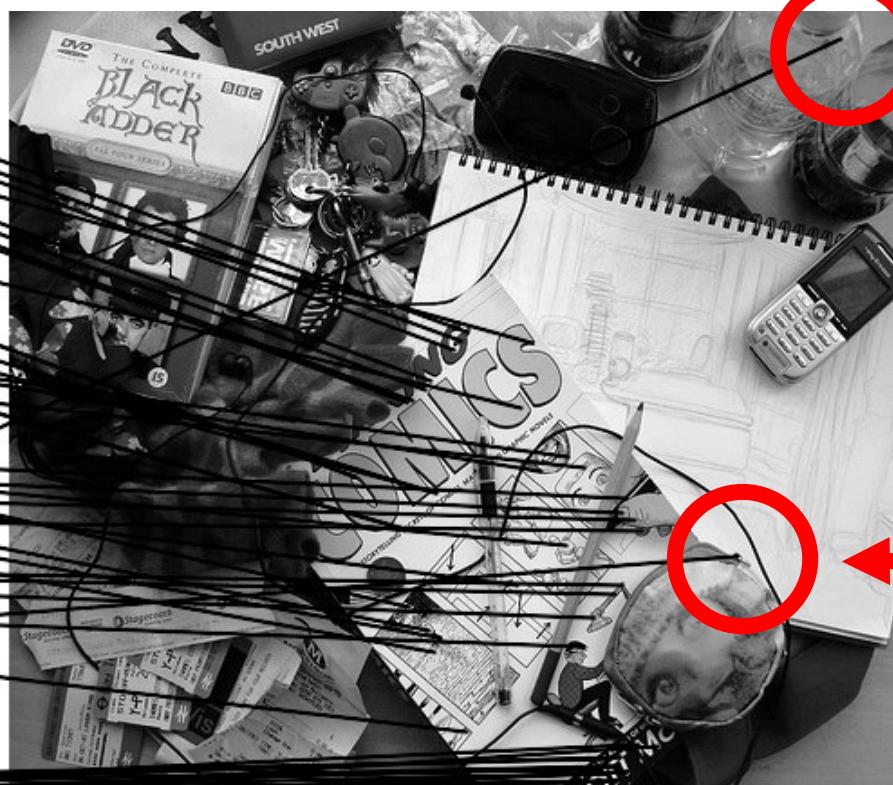
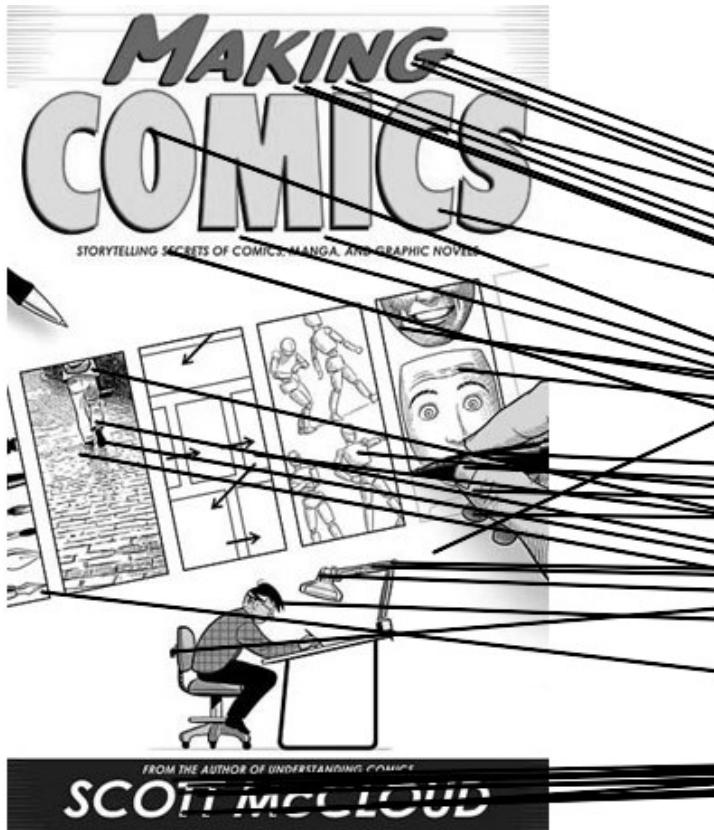
Feature matching example



outlier

51 matches (thresholded by ratio score)

Feature matching example



outlier

outlier

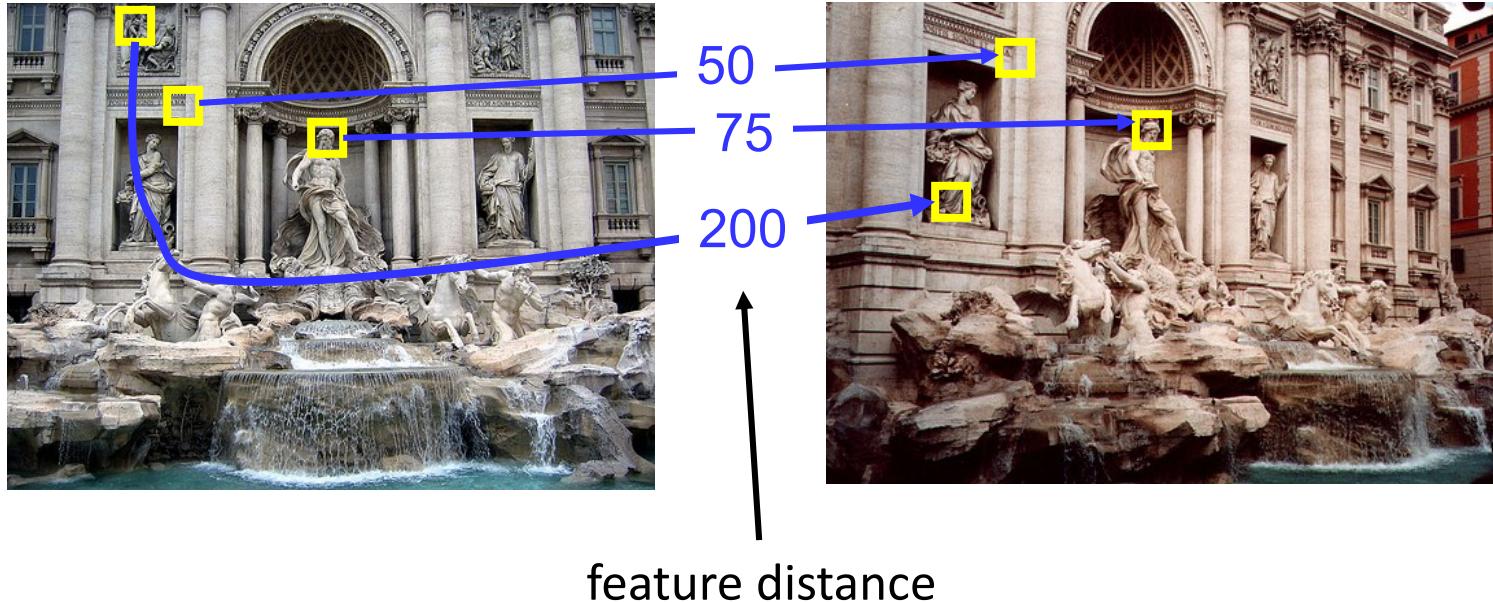
51 matches (thresholded by ratio score)

Evaluating the results

How can we measure the performance of a feature matcher?

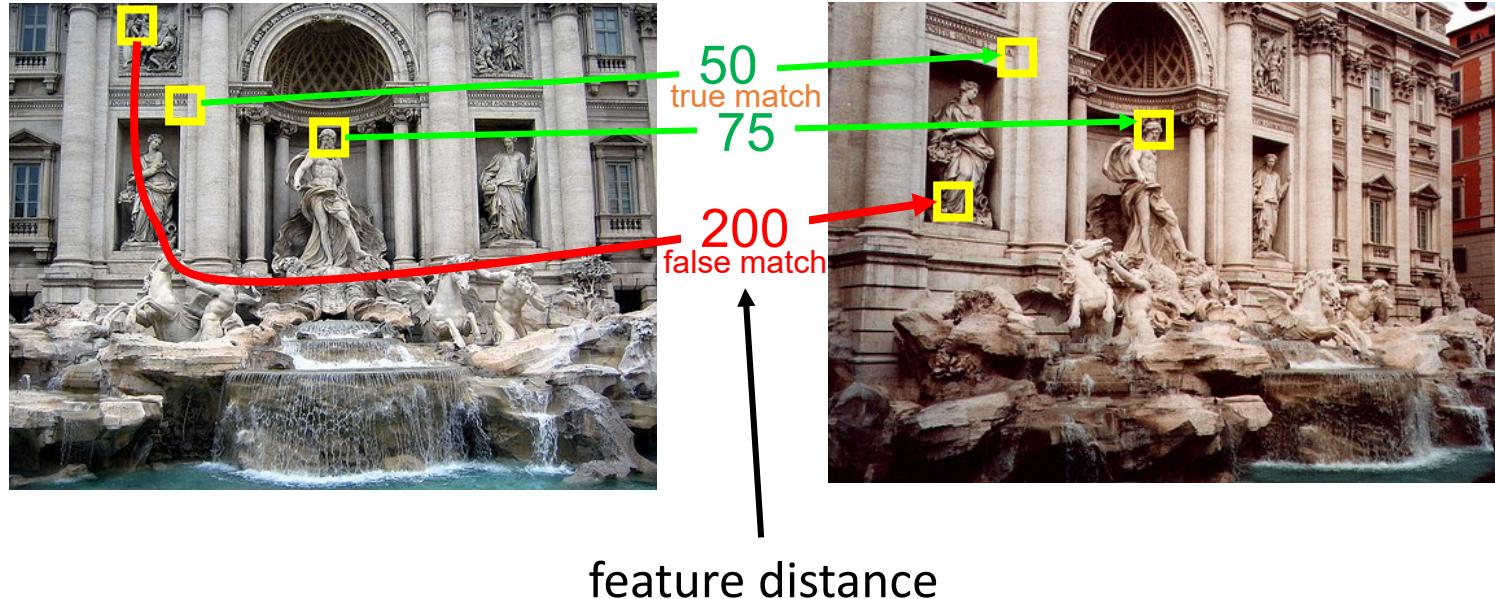
Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives

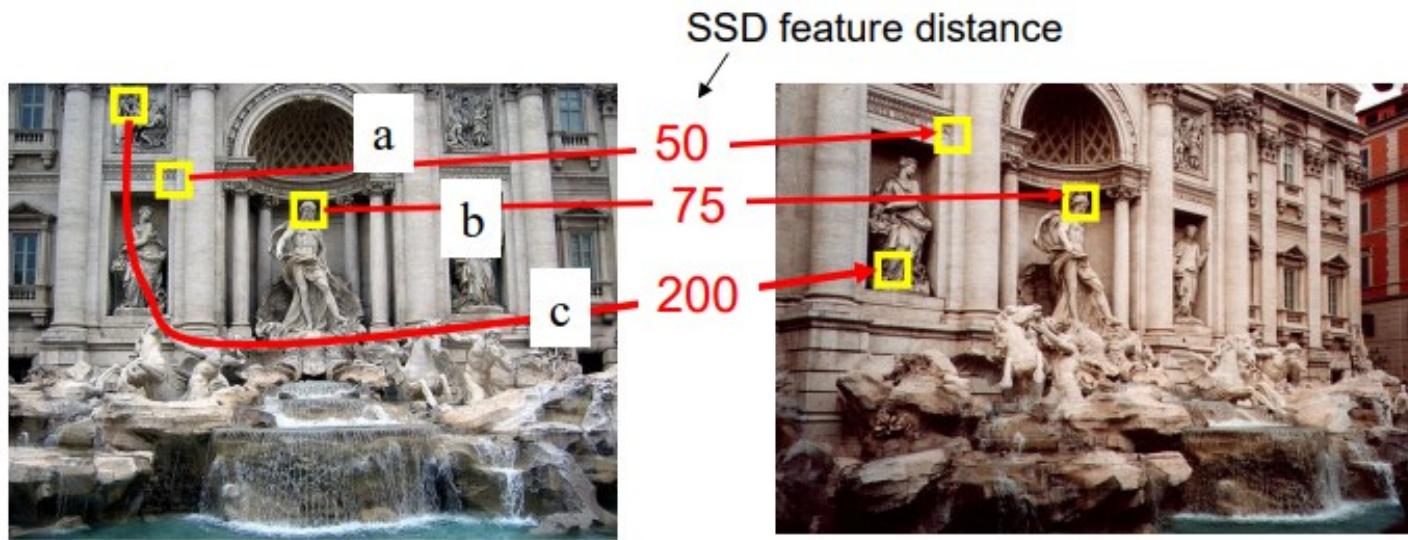
How can we measure the performance of a feature matcher?



The distance threshold affects performance

- **True positives** = # of detected matches that are correct
 - Suppose we want to maximize these—how to choose threshold?
- **False positives** = # of detected matches that are incorrect
 - Suppose we want to minimize these—how to choose threshold?

Effect of threshold T



Decision rule: Accept match if $\text{SSD} < T$

Example: **Large T**

$T = 250 \Rightarrow a, b, c$ are all accepted as matches

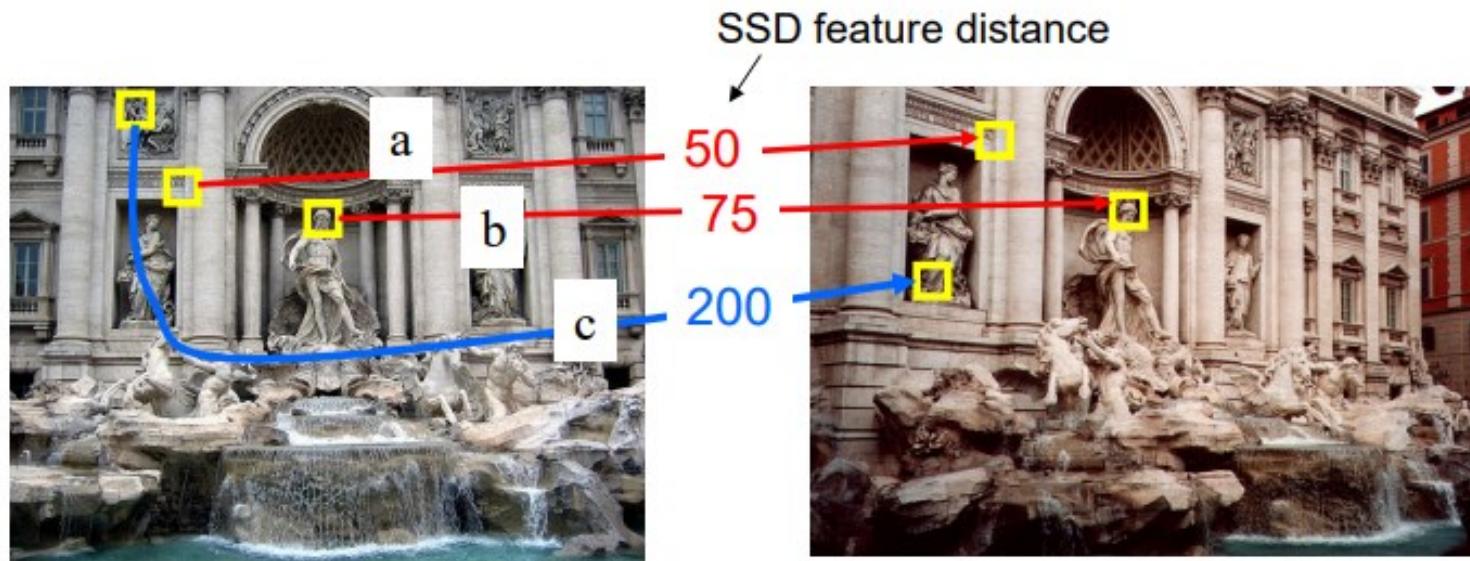
a and b are true matches (“**true positives**”)

- they are actually matches

c is a false match (“**false positive**”)

- actually not a match

Effect of threshold T



Decision rule: Accept match if $\text{SSD} < T$

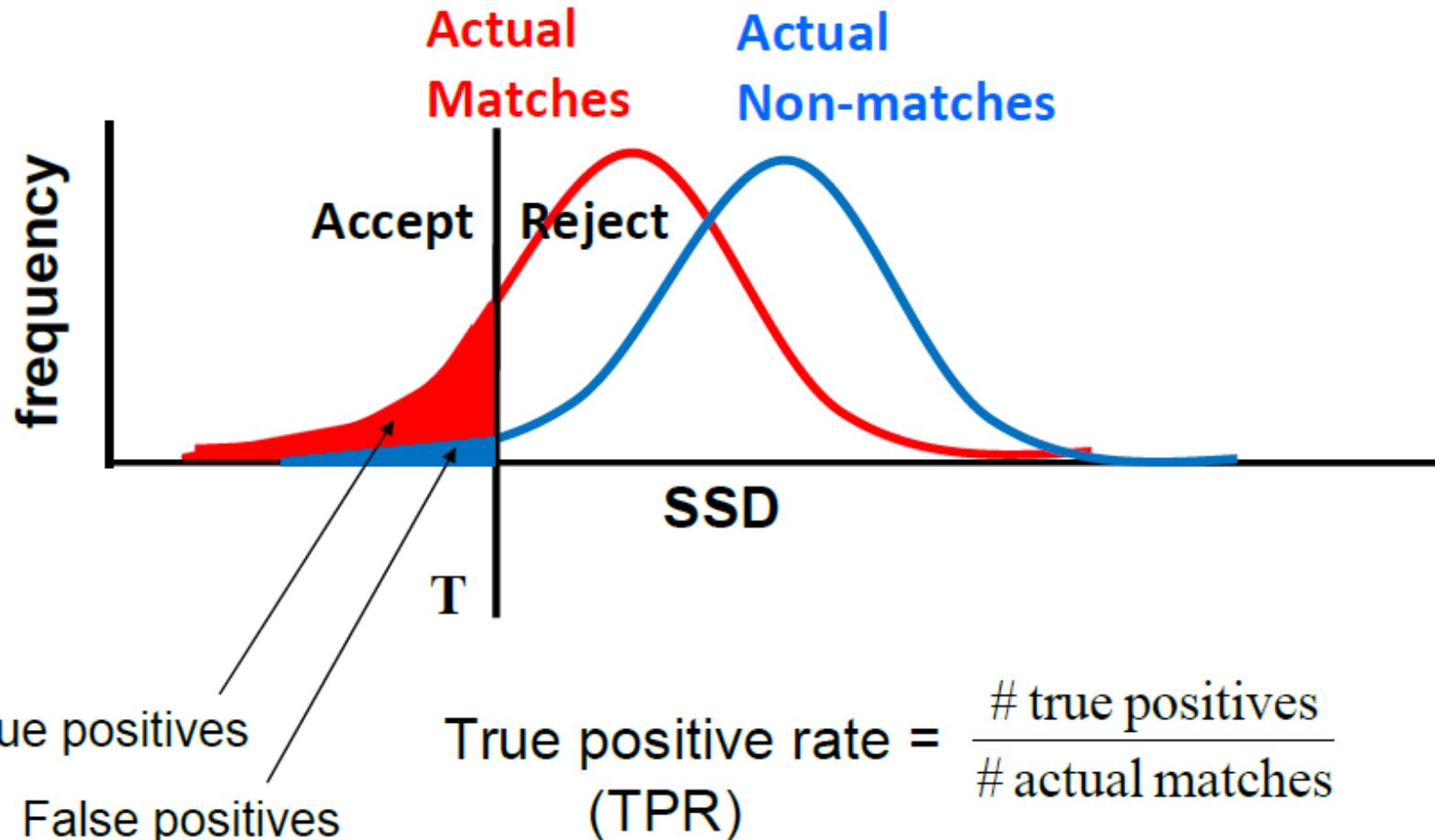
Example: **Smaller T**

$T = 100 \Rightarrow$ only a and b are accepted as matches

a and b are true matches (“true positives”)

c is no longer a “false positive” (it is a “true negative”)

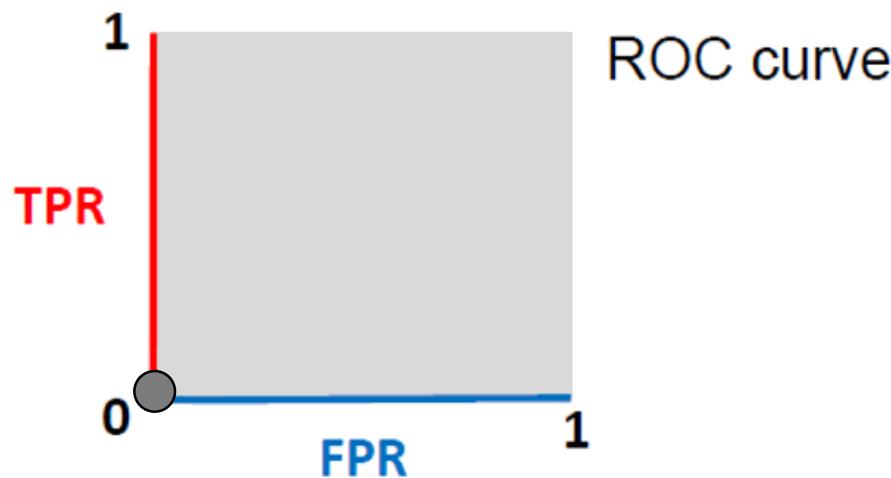
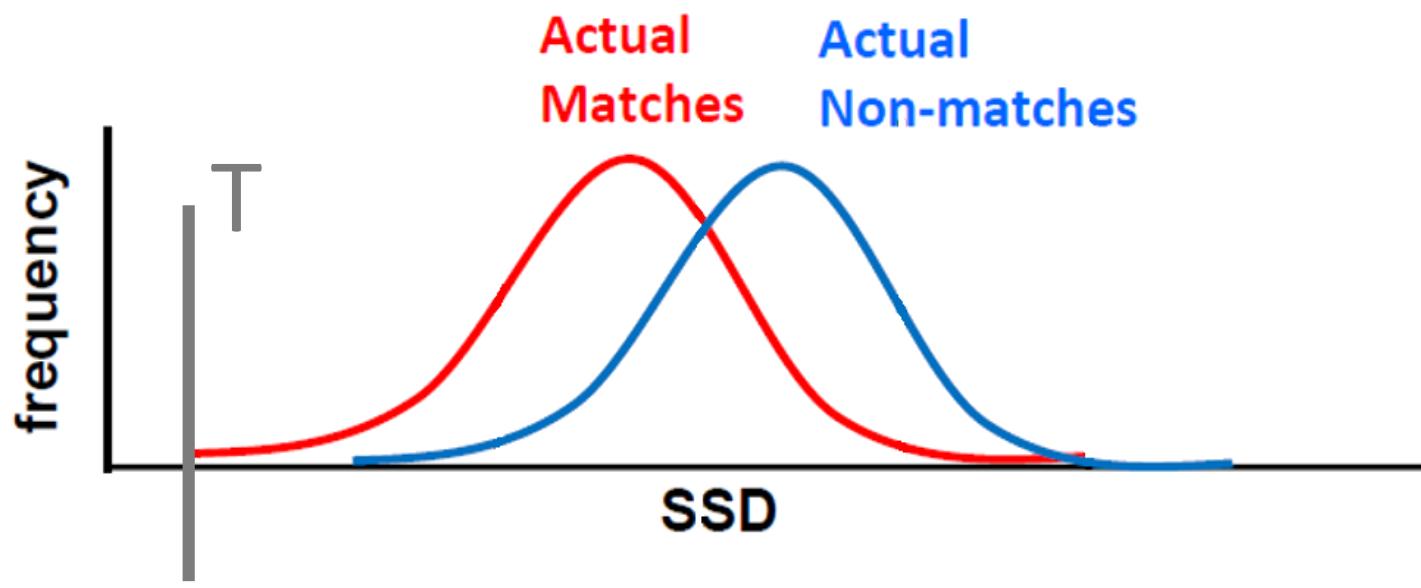
True positives and false positives



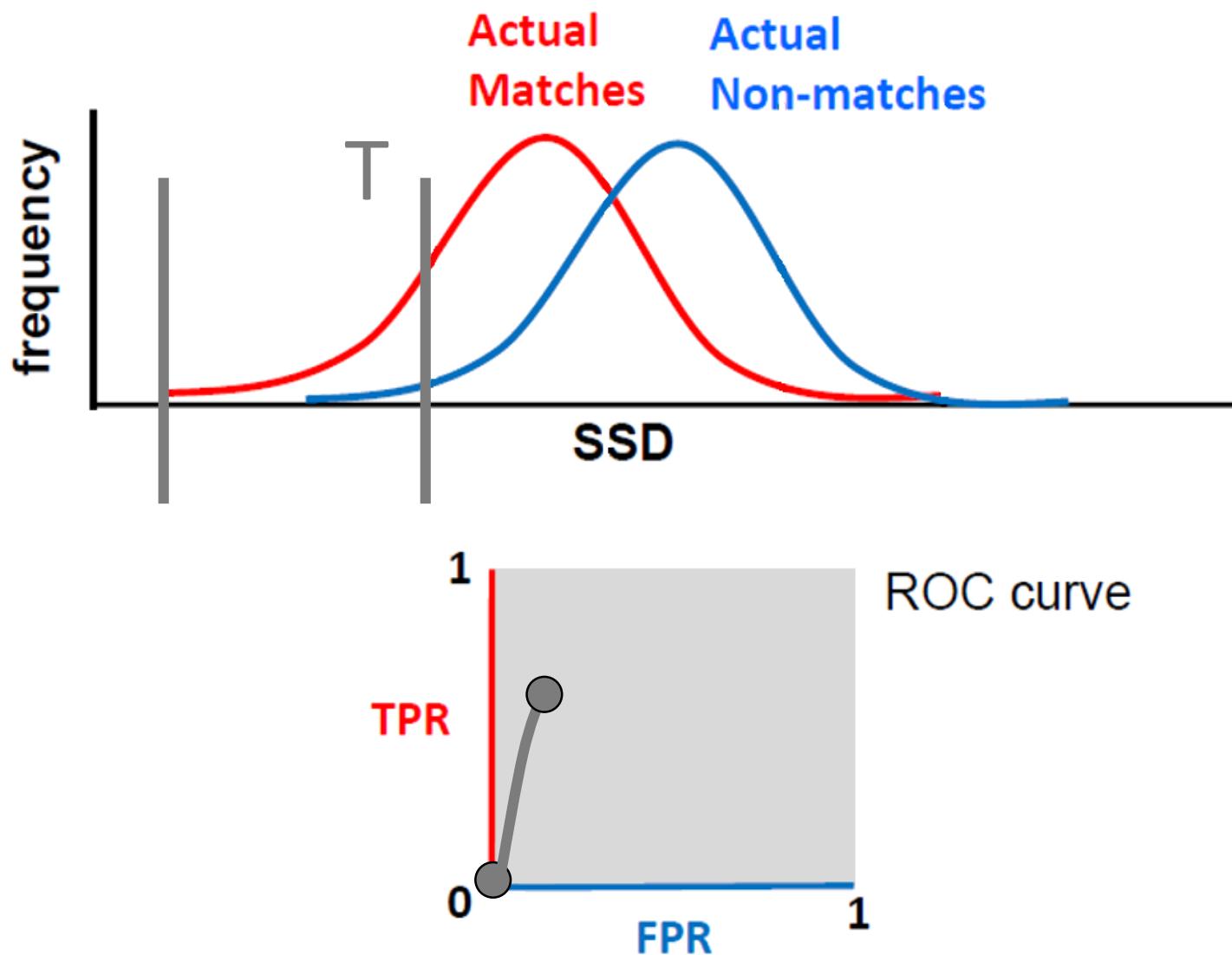
True positive rate = $\frac{\# \text{ true positives}}{\# \text{ actual matches}}$

False positive rate = $\frac{\# \text{ false positives}}{\# \text{ actual nonmatches}}$

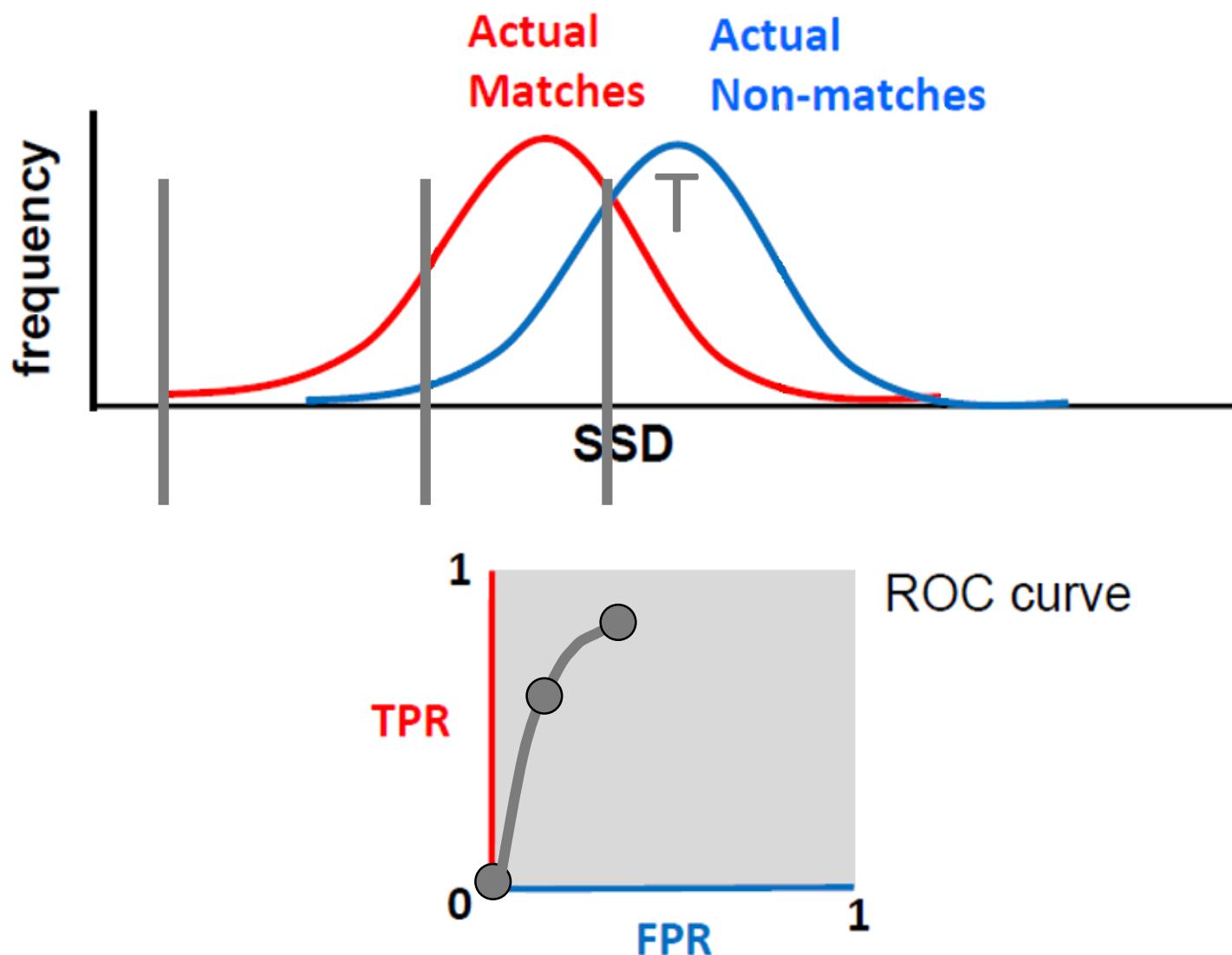
Receiver Operating Characteristic (ROC) curve



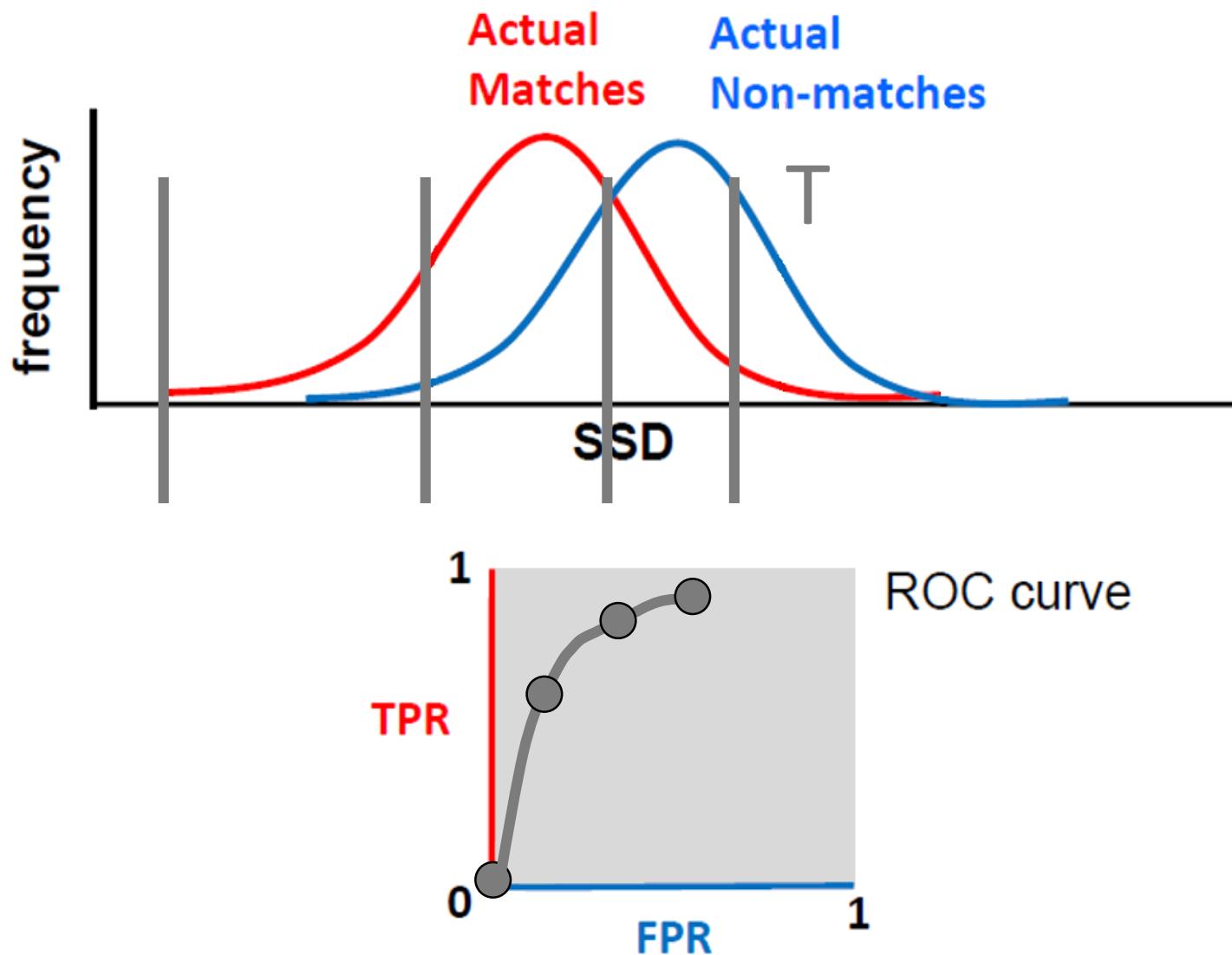
Receiver Operating Characteristic (ROC) curve



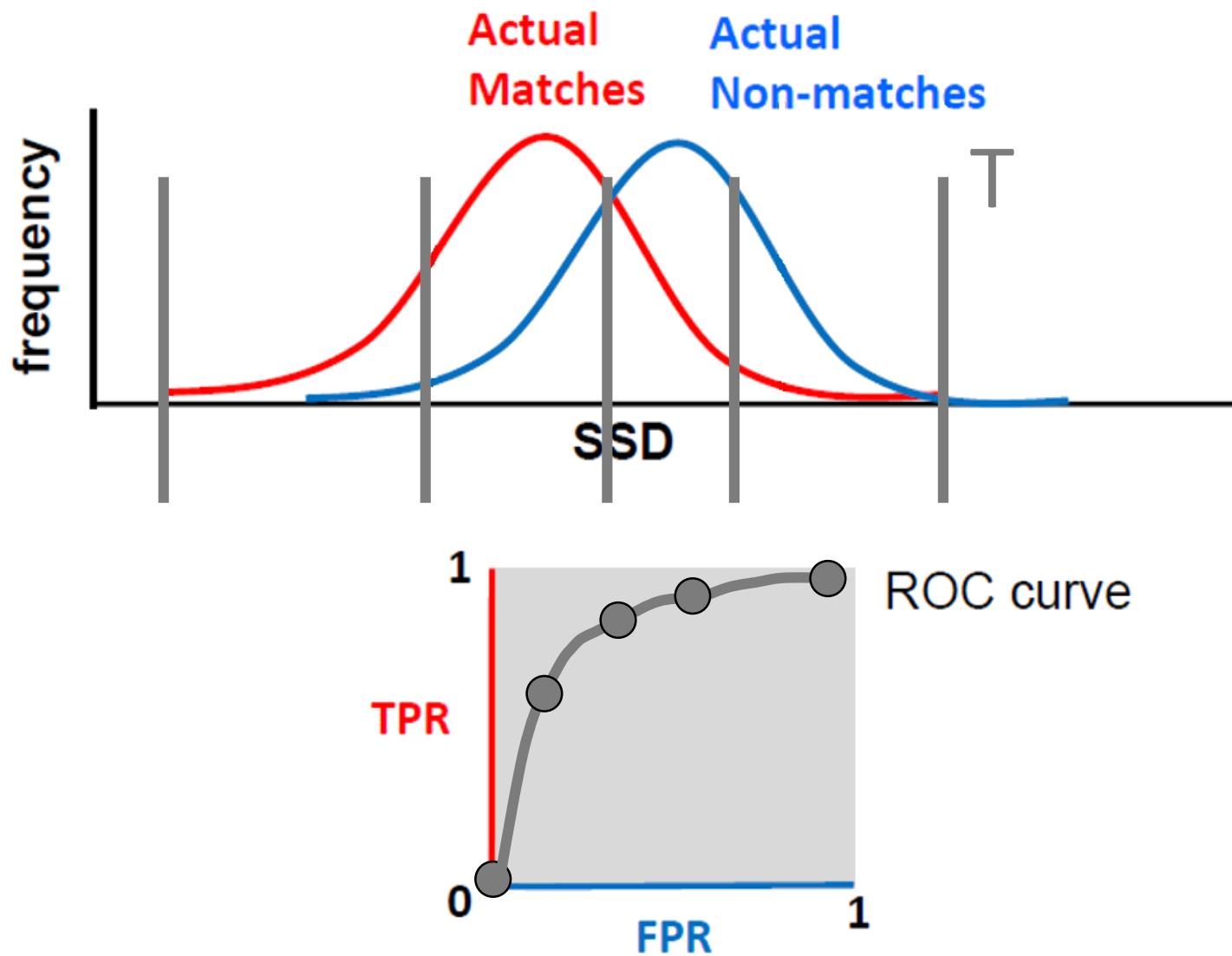
Receiver Operating Characteristic (ROC) curve



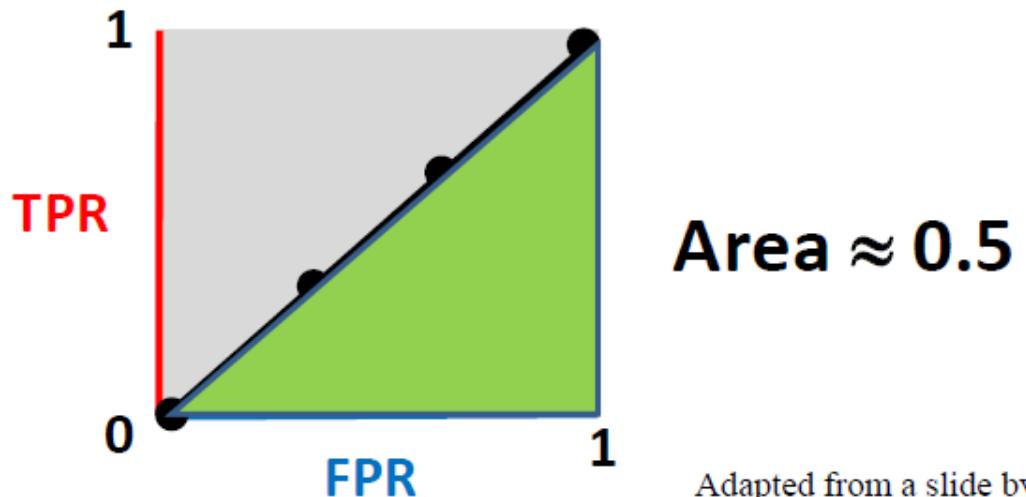
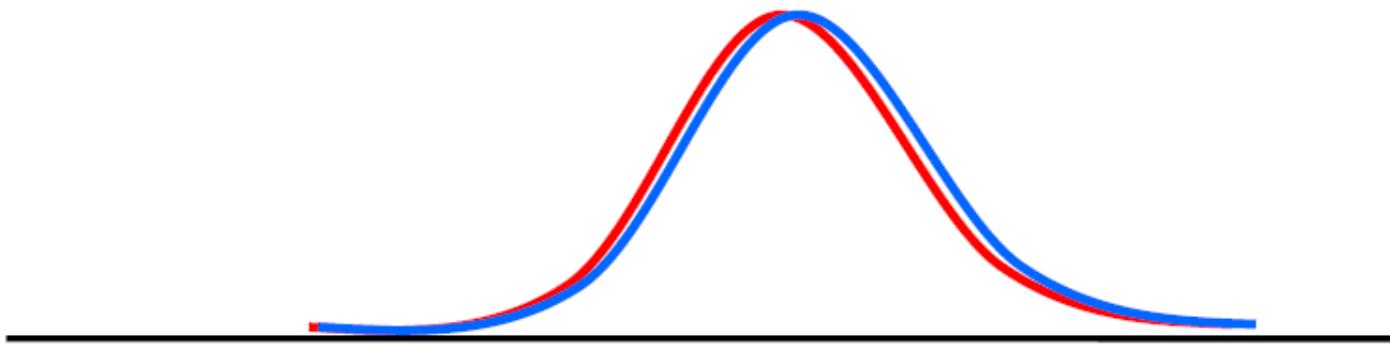
Receiver Operating Characteristic (ROC) curve



Receiver Operating Characteristic (ROC) curve

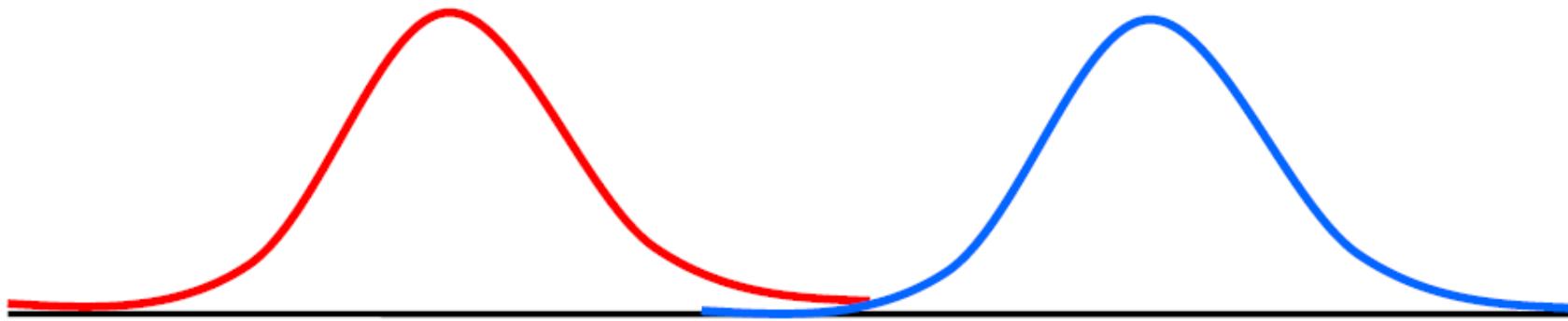


If the features selected were bad...



Adapted from a slide by Shin Kira

If the features selected were good...



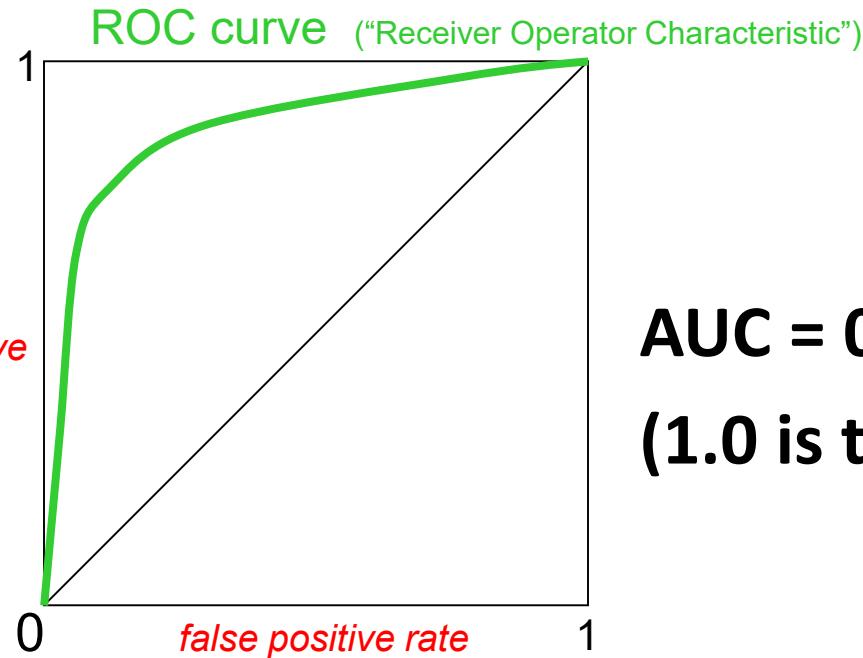
Adapted from a slide by Shin Kira

Area under the curve

Single number: Area Under the Curve (AUC)

$$\frac{\# \text{ true positives}}{\# \text{ matching features (positives)}} \quad \text{recall}$$

*true
positive
rate*



$$\frac{\# \text{ false positives}}{\# \text{ unmatched features (negatives)}} \quad 1 - \text{specificity}$$

AUC = 0.87
(1.0 is the best)

Feature matching: example using ORB

The screenshot shows a Jupyter Notebook interface with the following details:

- File:** matching.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, CODE, TEXT, CELL, CELL.
- Table of contents:** Code snippets, Files (selected).
- Files:** UPLOAD, REFRESH, sample_data, montagna-1.jpg, montagna-2.jpg.
- Code Cell:** Contains Python code for feature matching using ORB.
- Code Content:**

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img_1 = cv.imread("montagna-1.jpg")
gray_1 = cv.cvtColor(img_1, cv.COLOR_BGR2GRAY)

img_2 = cv.imread("montagna-2.jpg")
gray_2 = cv.cvtColor(img_2, cv.COLOR_BGR2GRAY)

orb = cv.ORB_create()

kp_1, des_1 = orb.detectAndCompute(img_1, None)
kp_2, des_2 = orb.detectAndCompute(img_2, None)

bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

matches = bf.match(des_1, des_2)

matches = sorted(matches, key = lambda x:x.distance)

img_3 = cv.drawMatches(img_1, kp_1, img_2, kp_2, matches[:20], None, flags=2)

img_3_rgb = cv.cvtColor(img_3, cv.COLOR_GRAY2RGB)

plt.axis('off')
plt.imshow(img_3_rgb)
plt.show()

cv.imwrite('matching.png', img_3)
```
- Output:** A red play button icon is located above the code cell.
- Links:** Two blue links are shown on the right side: <https://dbloisi.github.io/corsi/images/montagna-1.jpg> and <https://dbloisi.github.io/corsi/images/montagna-2.jpg>.
- Disk:** Disk usage bar showing 28.08 GB available.

ORB



```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img_1 = cv.imread("montagna-1.jpg")
gray_1 = cv.cvtColor(img_1, cv.COLOR_BGR2GRAY)

img_2 = cv.imread("montagna-2.jpg")
gray_2 = cv.cvtColor(img_2, cv.COLOR_BGR2GRAY)

orb = cv.ORB_create()

kp_1, des_1 = orb.detectAndCompute(img_1, None)
kp_2, des_2 = orb.detectAndCompute(img_2, None)
```

brute force matching

Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

Hamming distance:

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



```
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

matches = bf.match(des_1, des_2)
```

Cross check test

- Choose matches (f_a, f_b) so that
 - f_b is the best match for f_a in I_b
 - And f_a is the best match for f_b in I_a
- Alternative to ratio test



Cross check test

- Choose matches (f_a, f_b) so that
 - f_b is the best match for f_a in I_b
 - And f_a is the best match for f_b in I_a
- Alternative to ratio test



Sorting

Matches are sorted in ascending order of their distances so that best matches (with low distance) come to front.

```
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

matches = bf.match(des_1, des_2)

matches = sorted(matches, key = lambda x:x.distance)
```



In Python, le funzioni lambda, dette anche funzioni anonime, sono funzioni che vengono usate per un periodo di tempo limitato e sono legate a funzioni di più alto livello

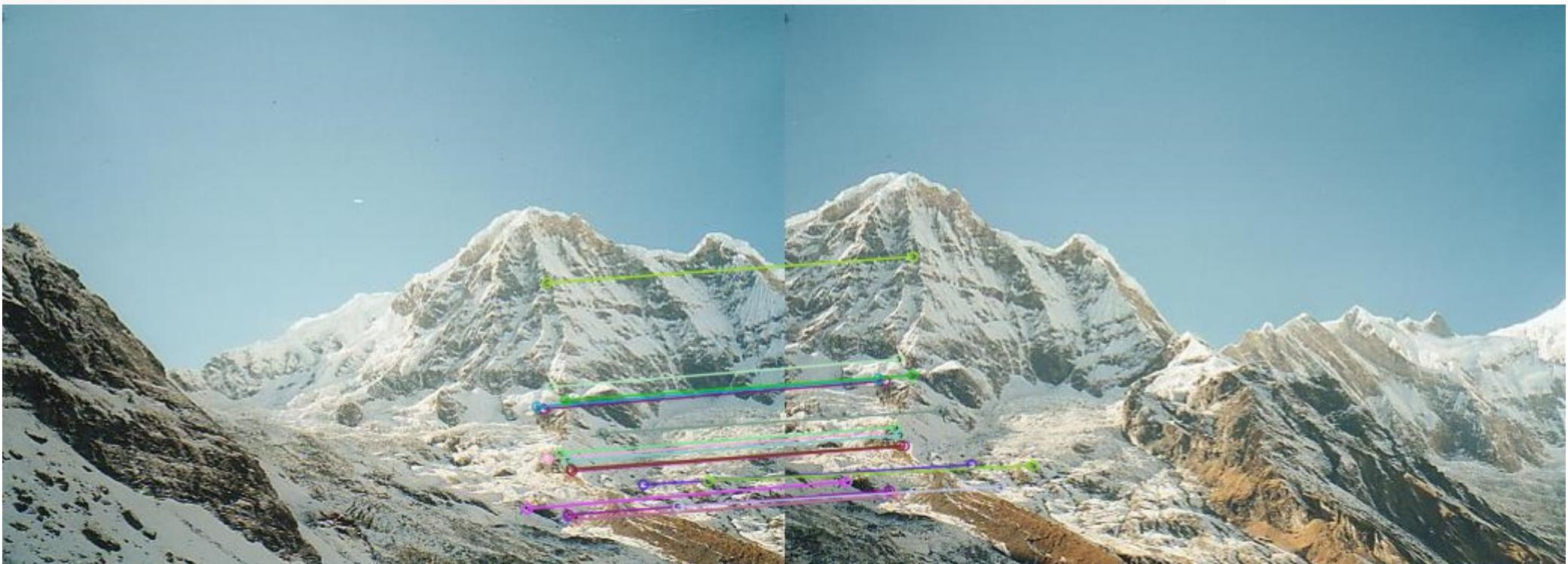
Result

```
img_3 = cv.drawMatches(img_1, kp_1, img_2, kp_2, matches[:20], None, flags=2)

img_3_rgb = cv.cvtColor(img_3, cv.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(img_3_rgb)
plt.show()

cv.imwrite('matching.png', img_3)
```



SIFT Example



Univ4.jpg



Univ3.jpg

SIFT Example



SIFT vs ORB

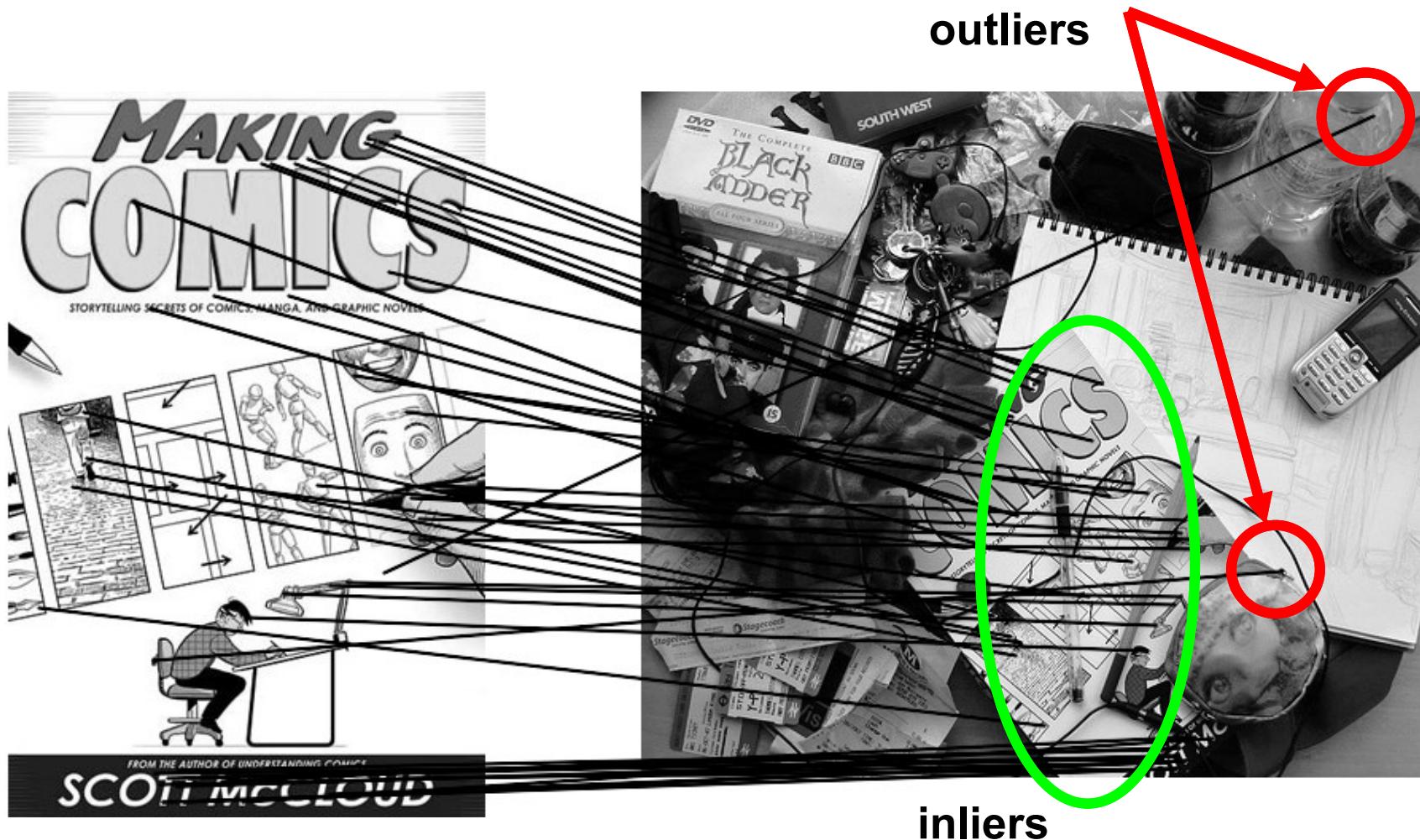


<https://dbloisi.github.io/corsi/lezionivep/sift.ipynb>



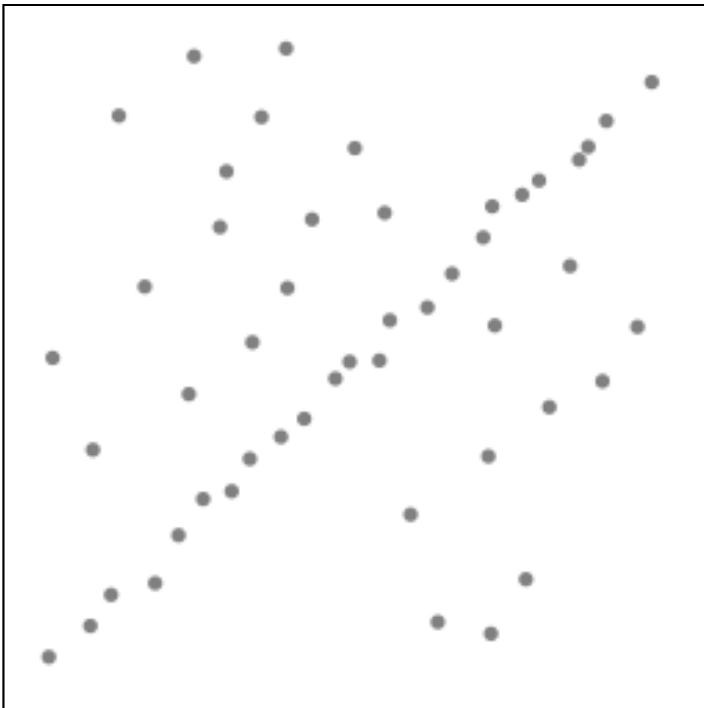
<https://dbloisi.github.io/corsi/lezionivep/orb.ipynb>

Excluding outliers



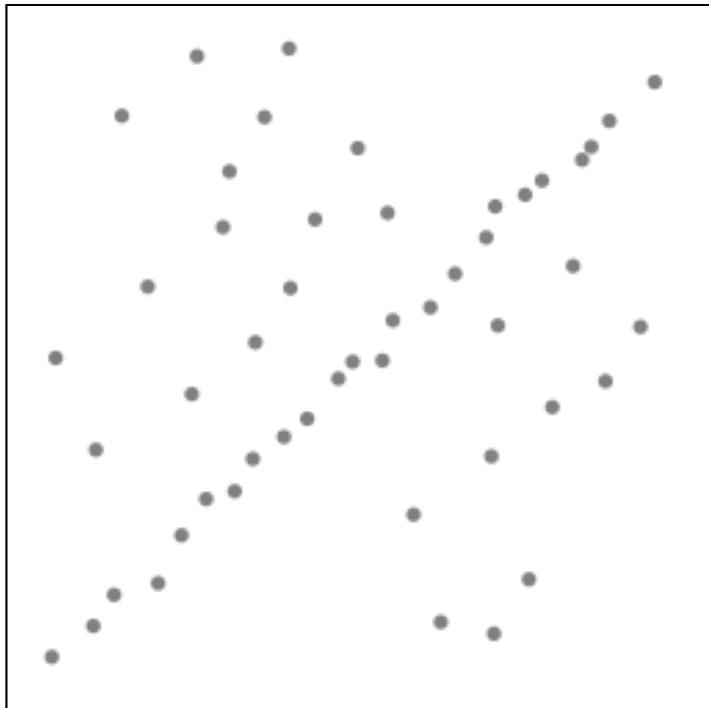
Robustness

Problem: Fit a line to these datapoints

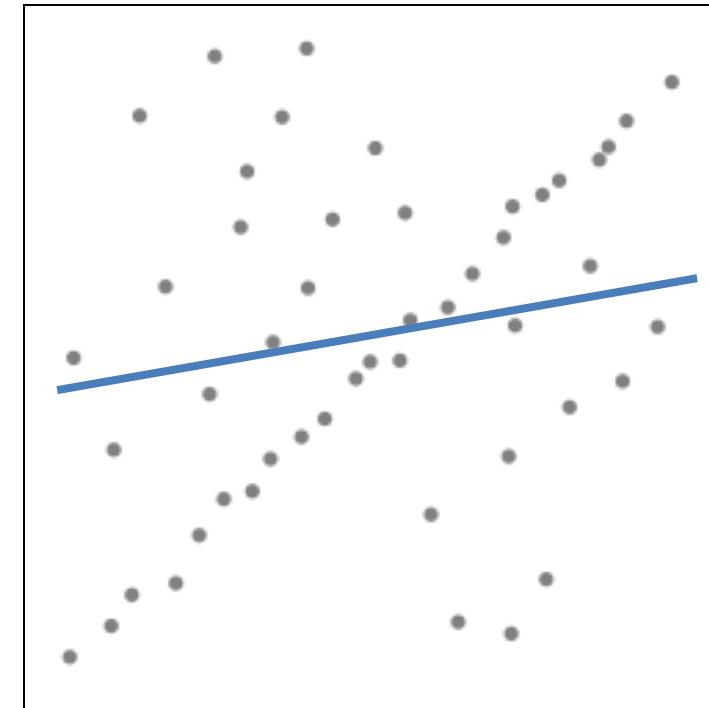


Robustness

Problem: Fit a line to these datapoints

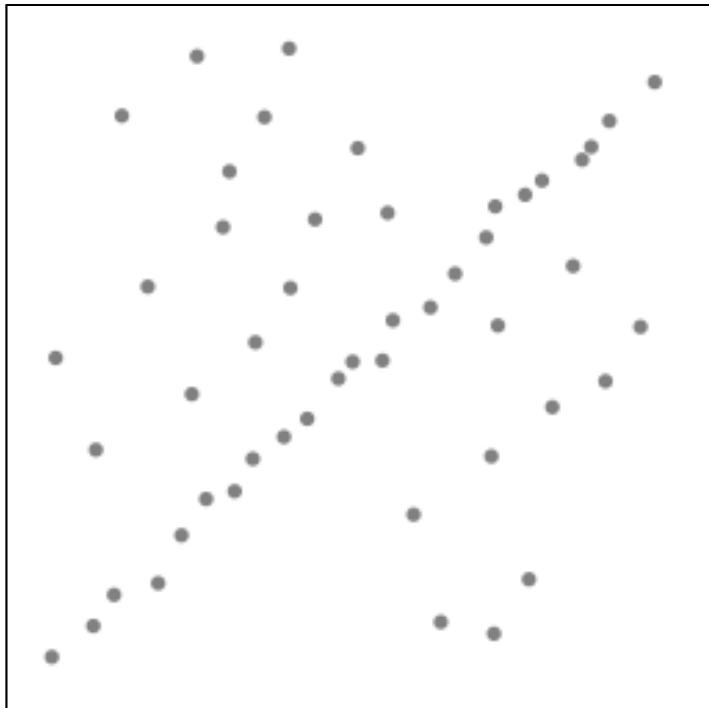


Least squares fit

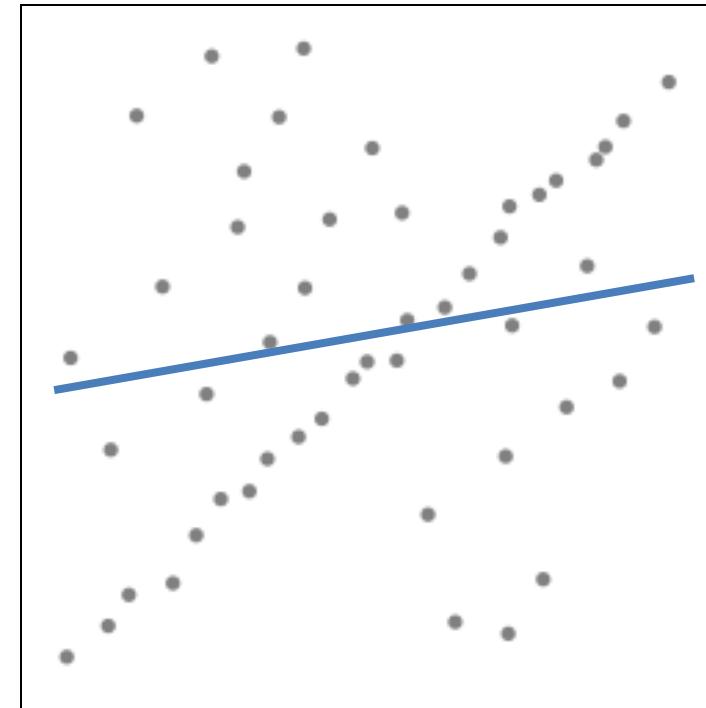


Robustness

Problem: Fit a line to these datapoints



Least squares fit

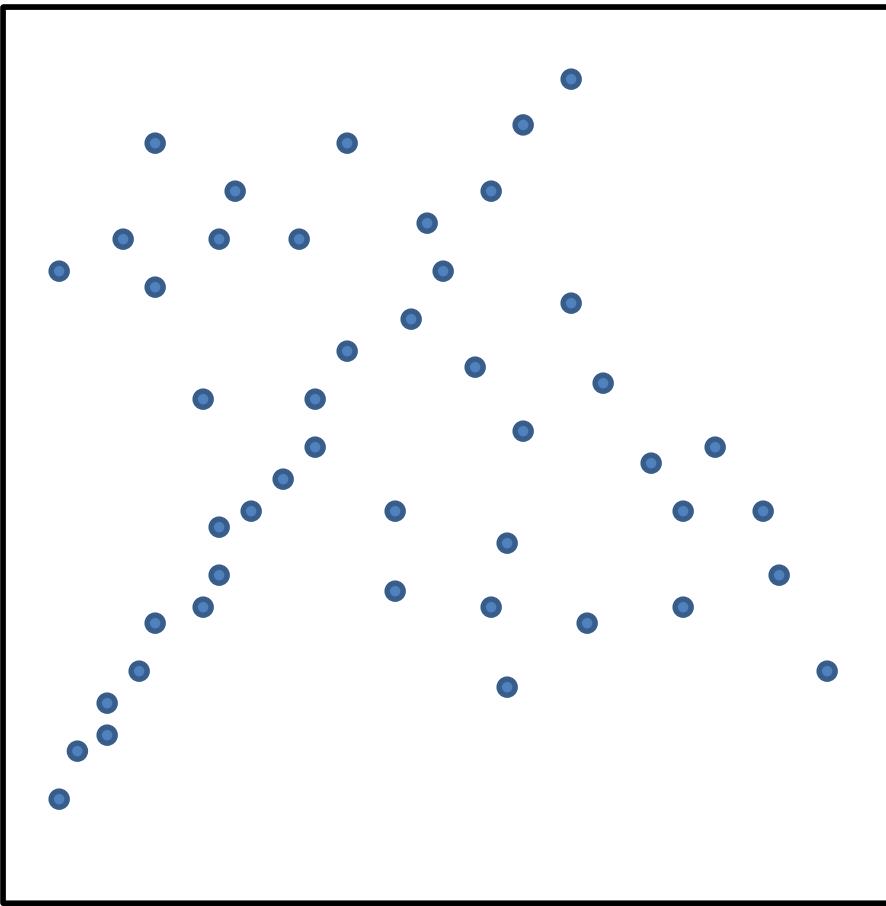


How can we fix this?

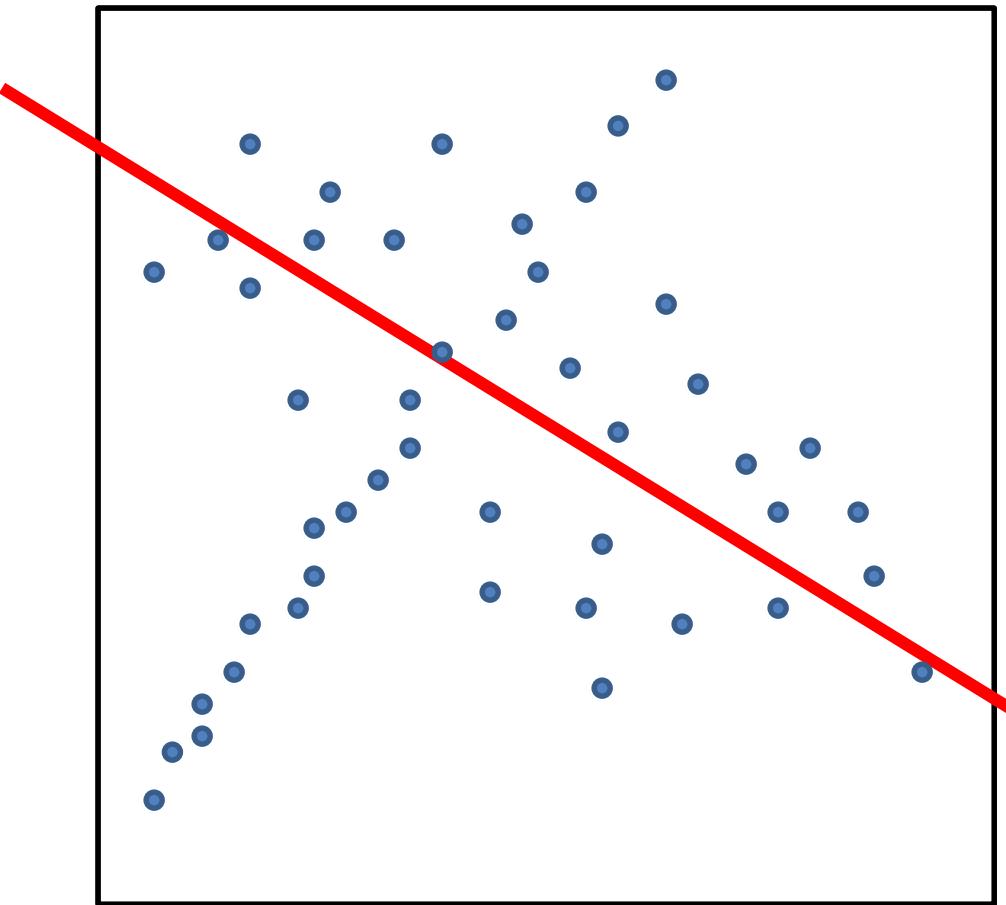
Idea

- Given a hypothesized line
- Count the number of points that “agree” with the line
 - “Agree” = within a small distance of the line
 - I.e., the **inliers** to that line
- For all possible lines, select the one with the largest number of inliers

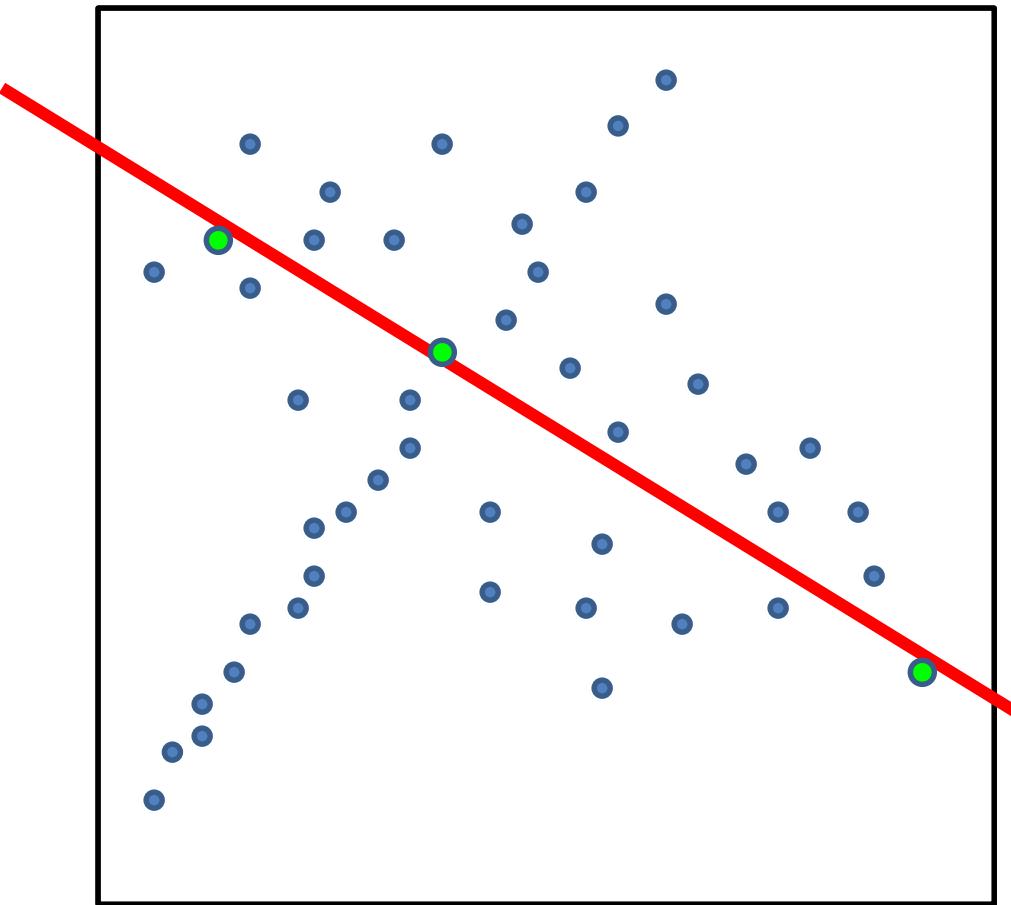
Counting inliers



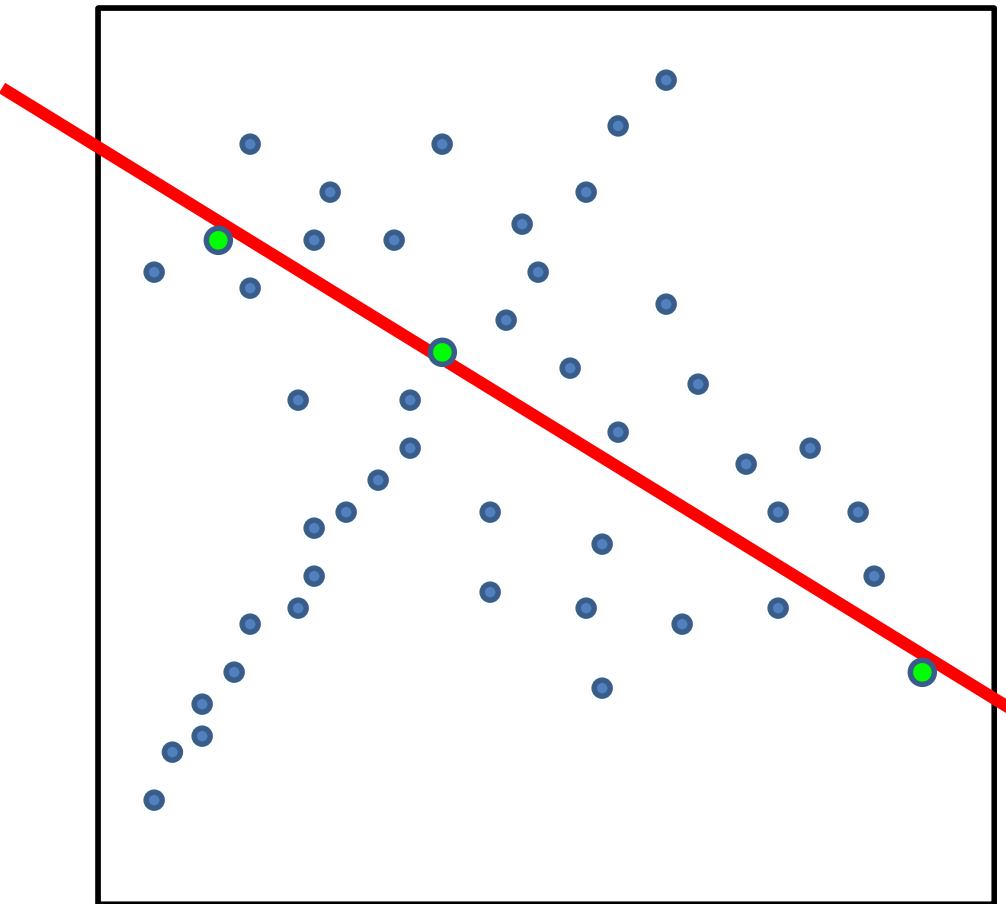
Counting inliers



Counting inliers

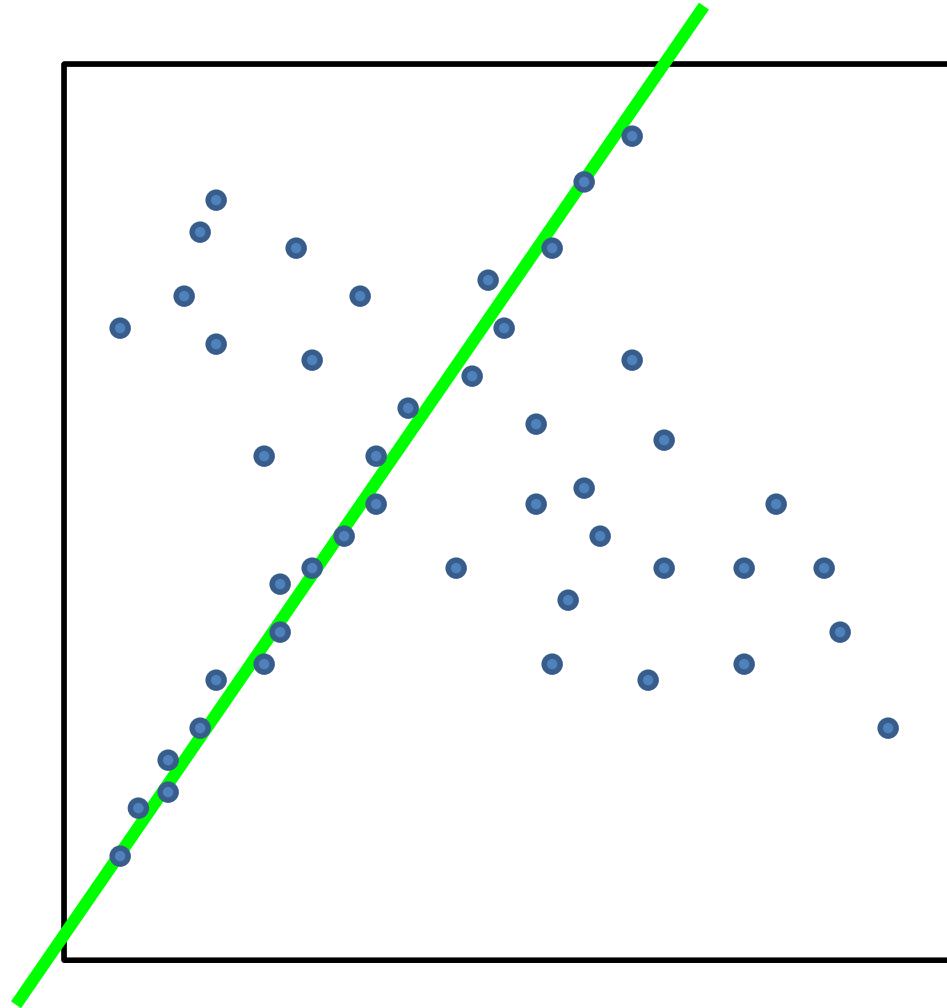


Counting inliers

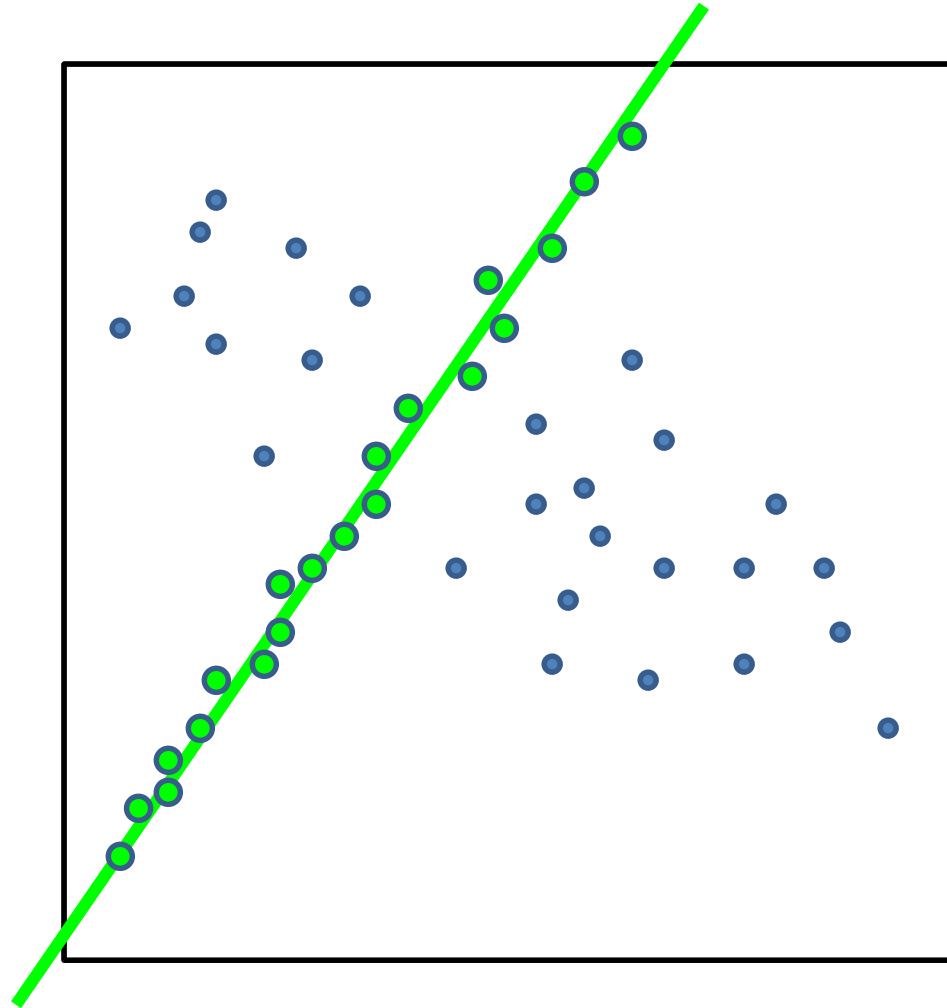


Inliers: 3

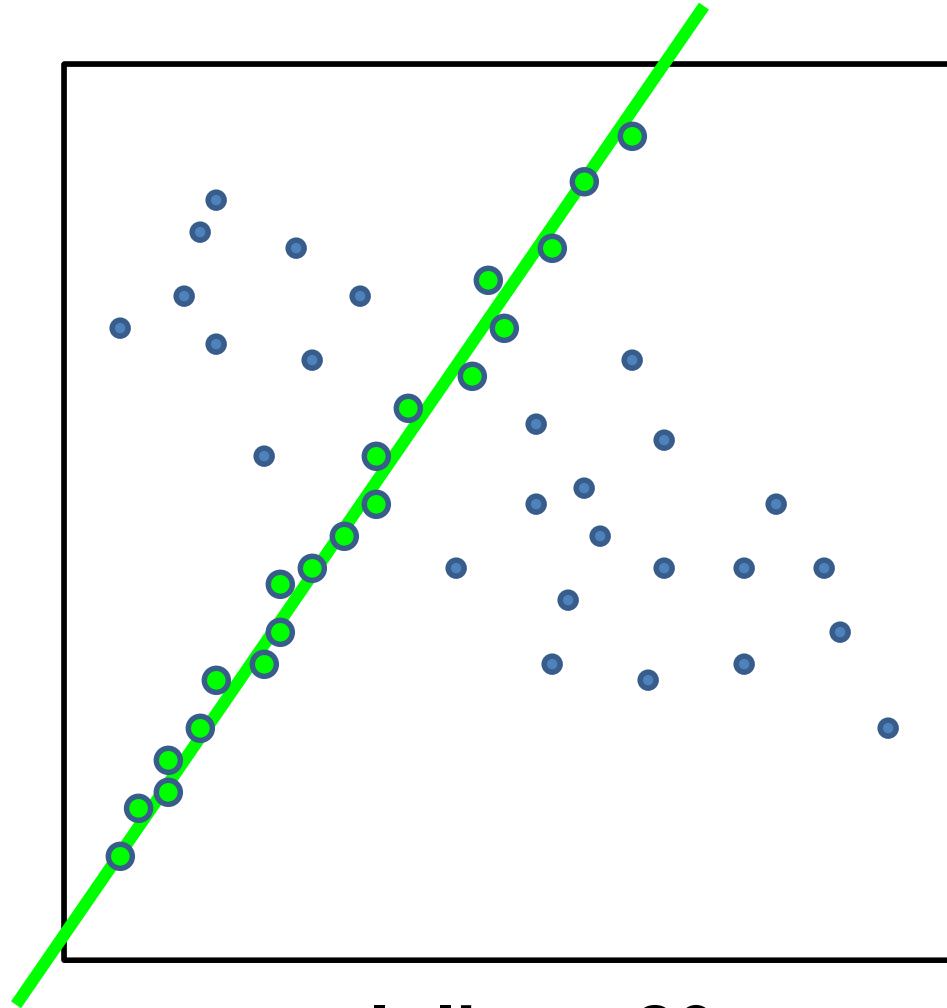
Counting inliers



Counting inliers



Counting inliers



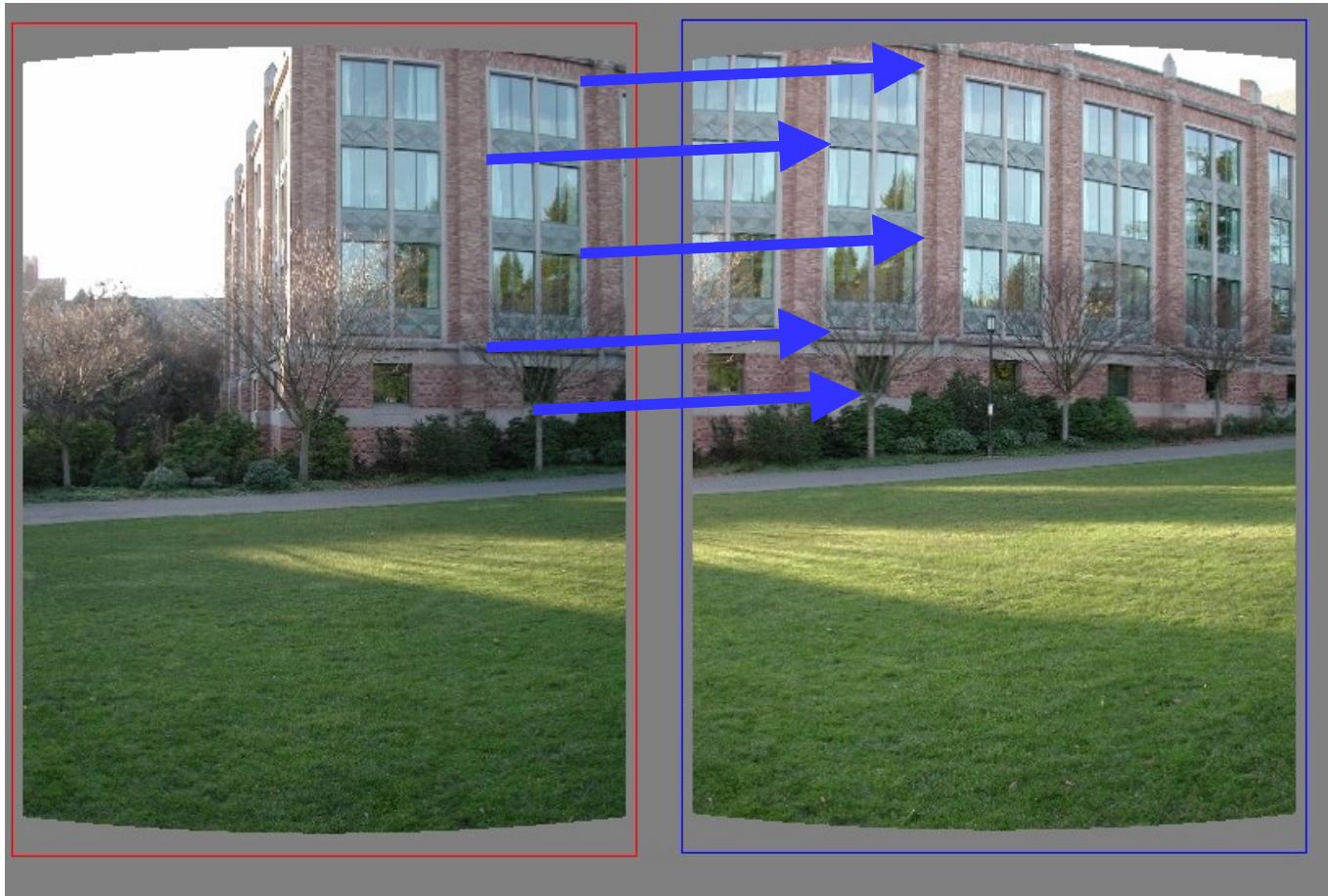
How do we find the best line?

- Unlike least-squares, no simple closed-form solution
- Hypothesize-and-test
 - Try out many lines, keep the best one
 - Which lines?

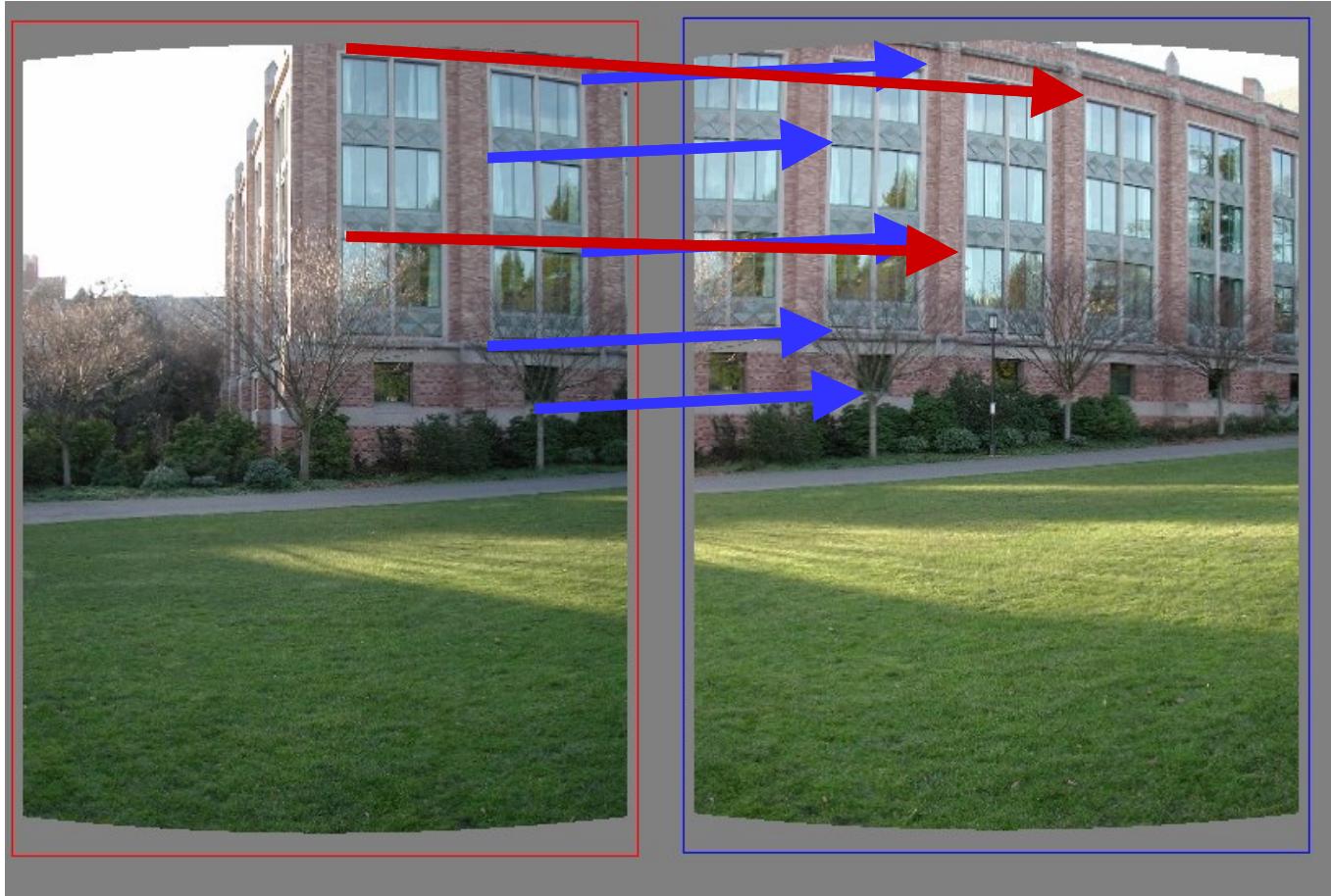
Translations



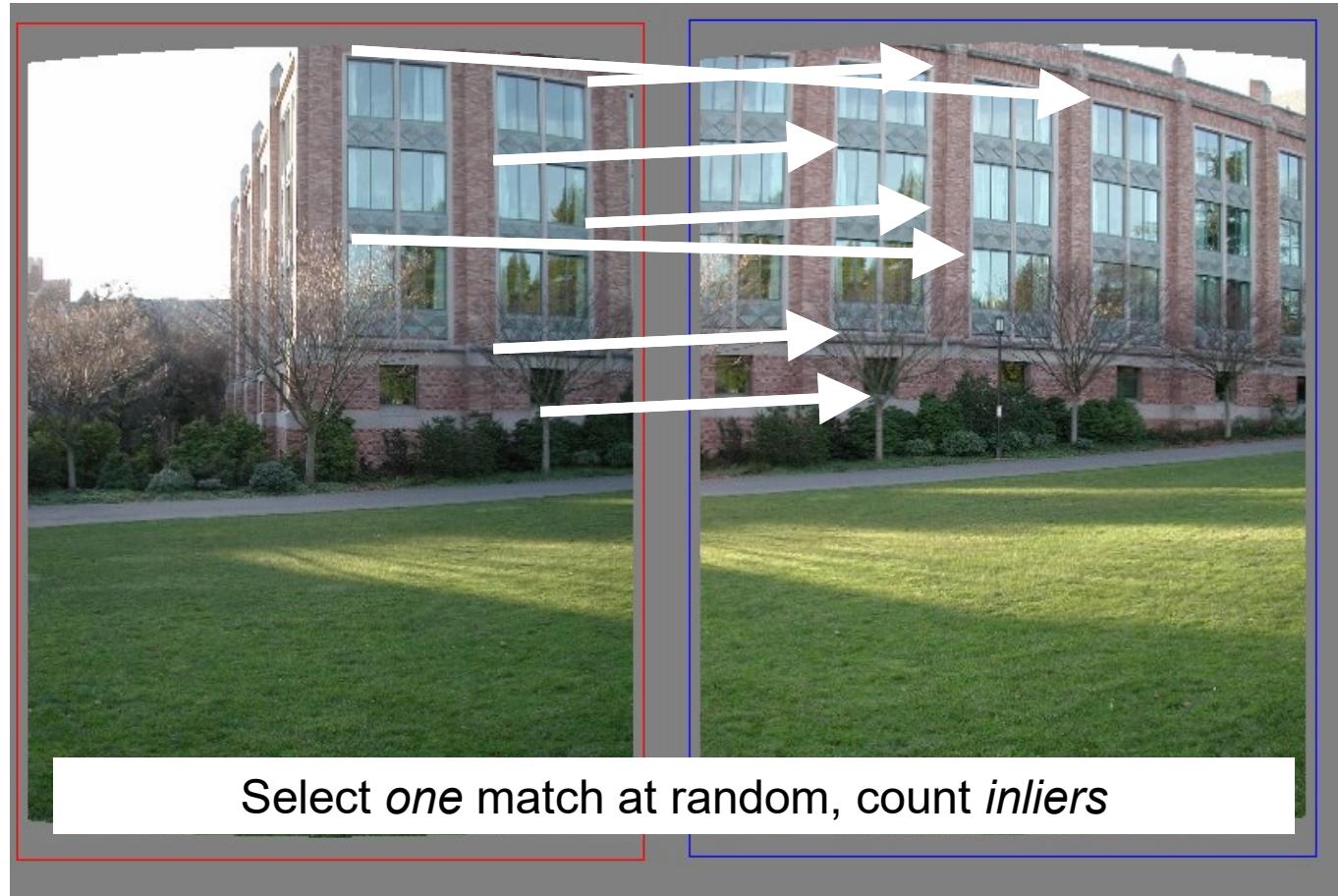
Translations



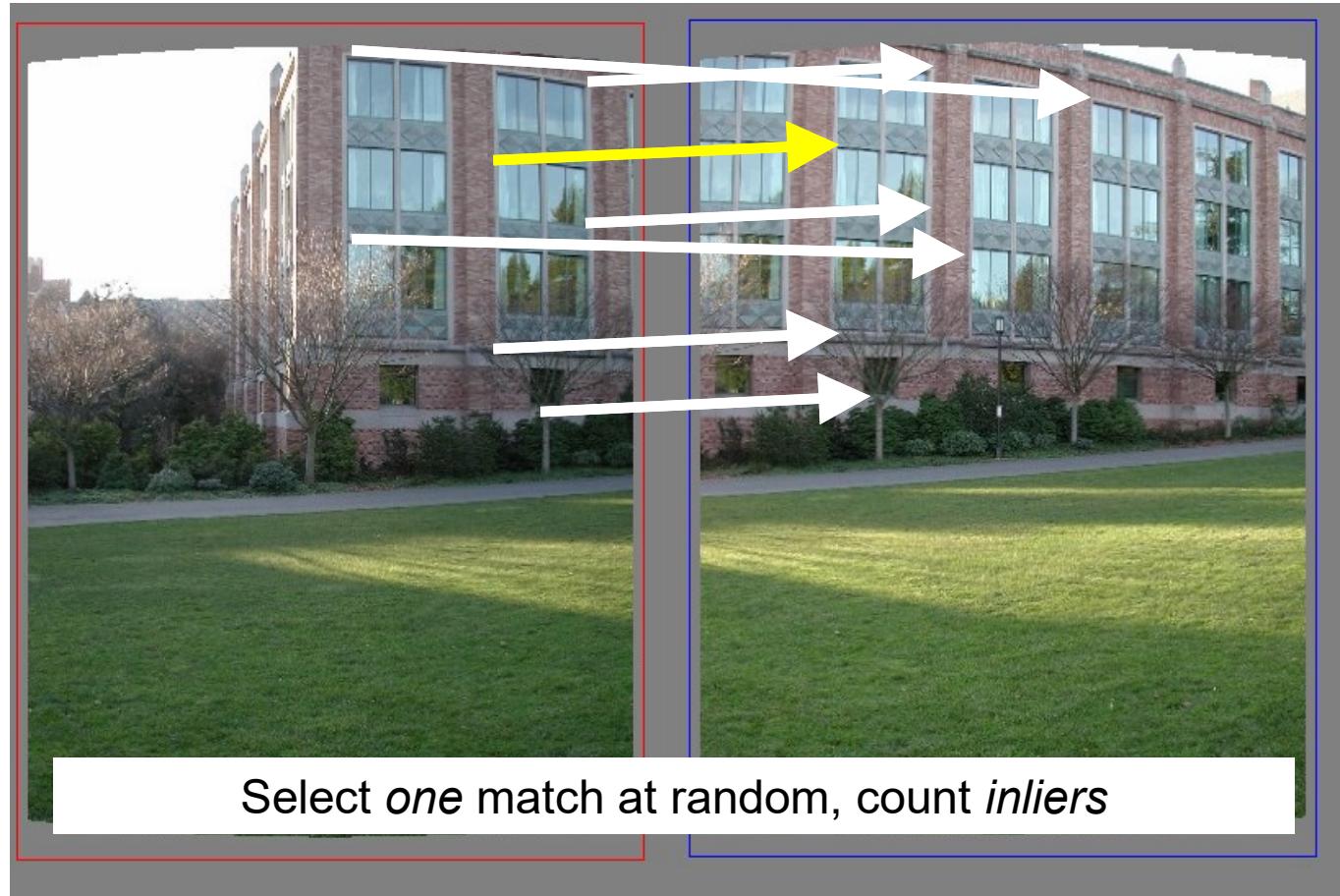
Translations



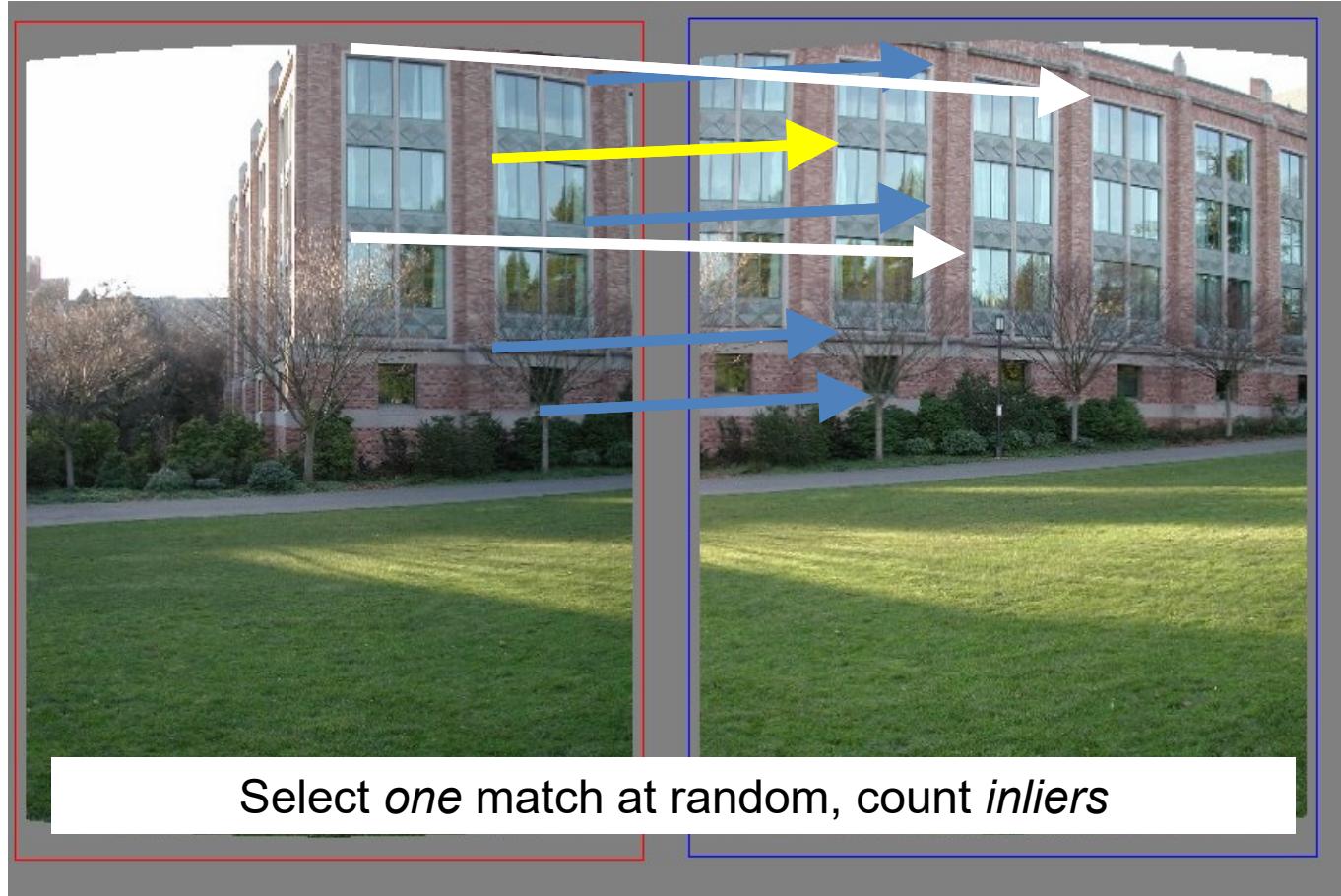
RANdom SAmple Consensus



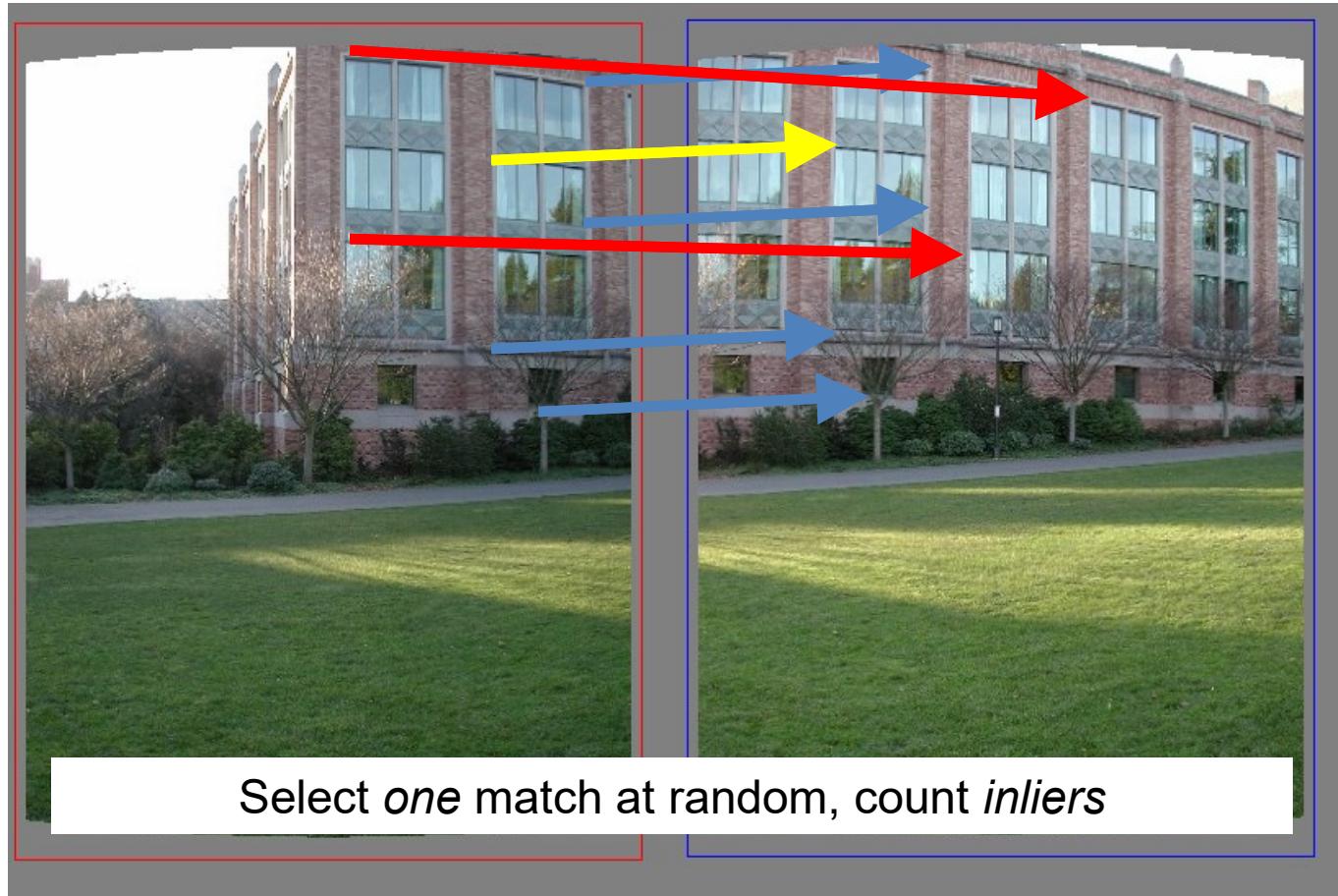
RANdom SAmple Consensus



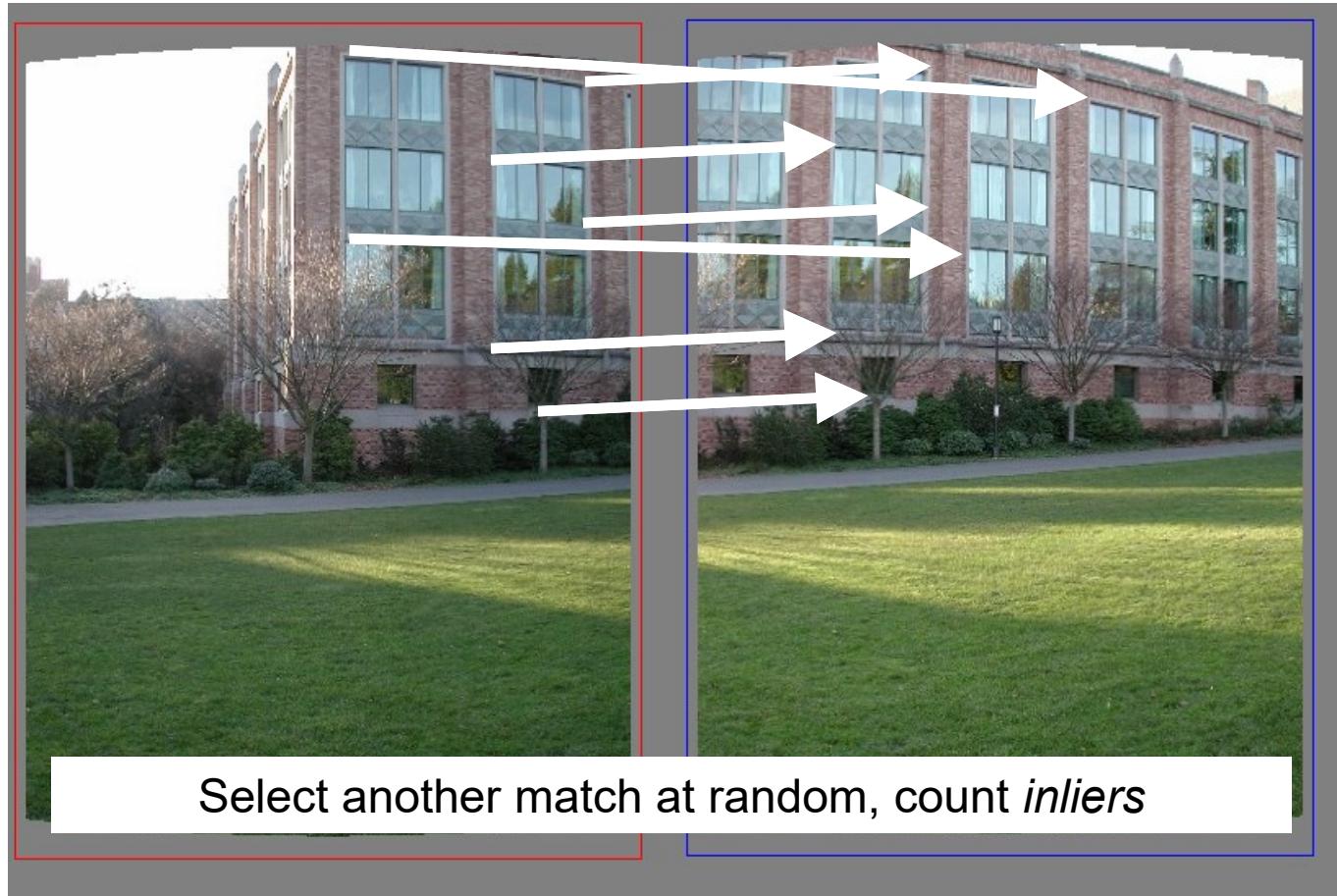
RANdom SAmple Consensus



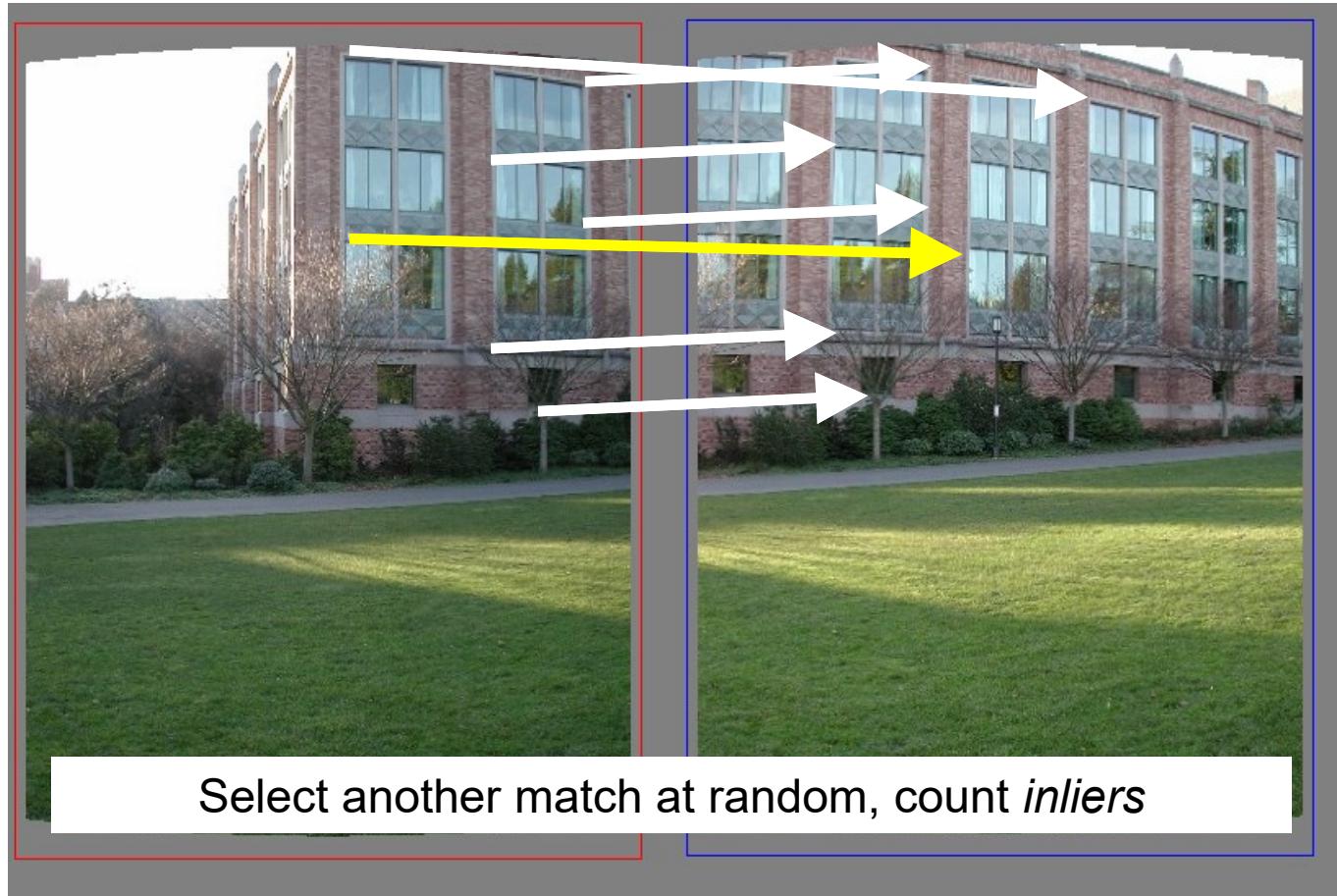
RANdom SAmple Consensus



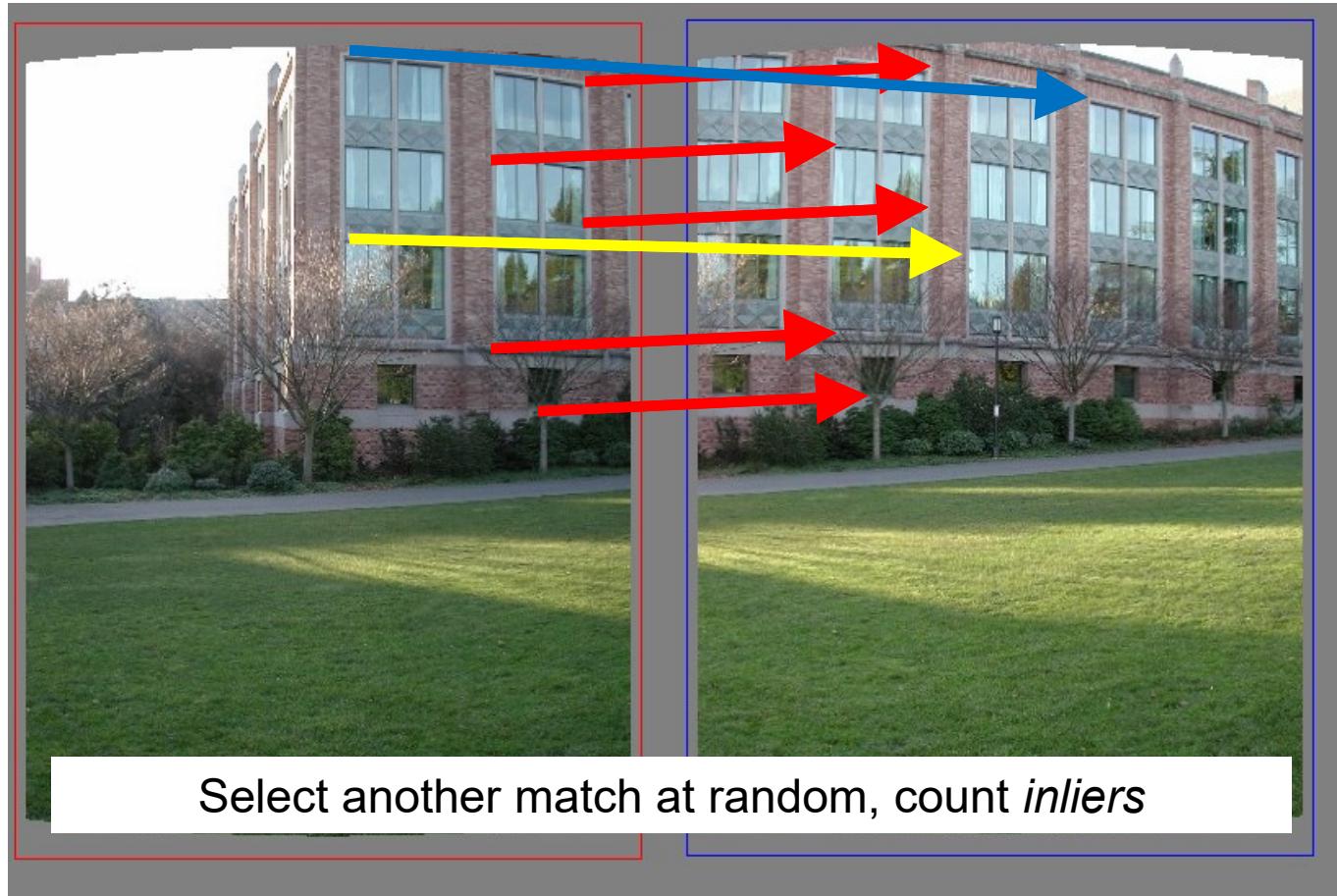
RANdom SAMPle CONSENSUS



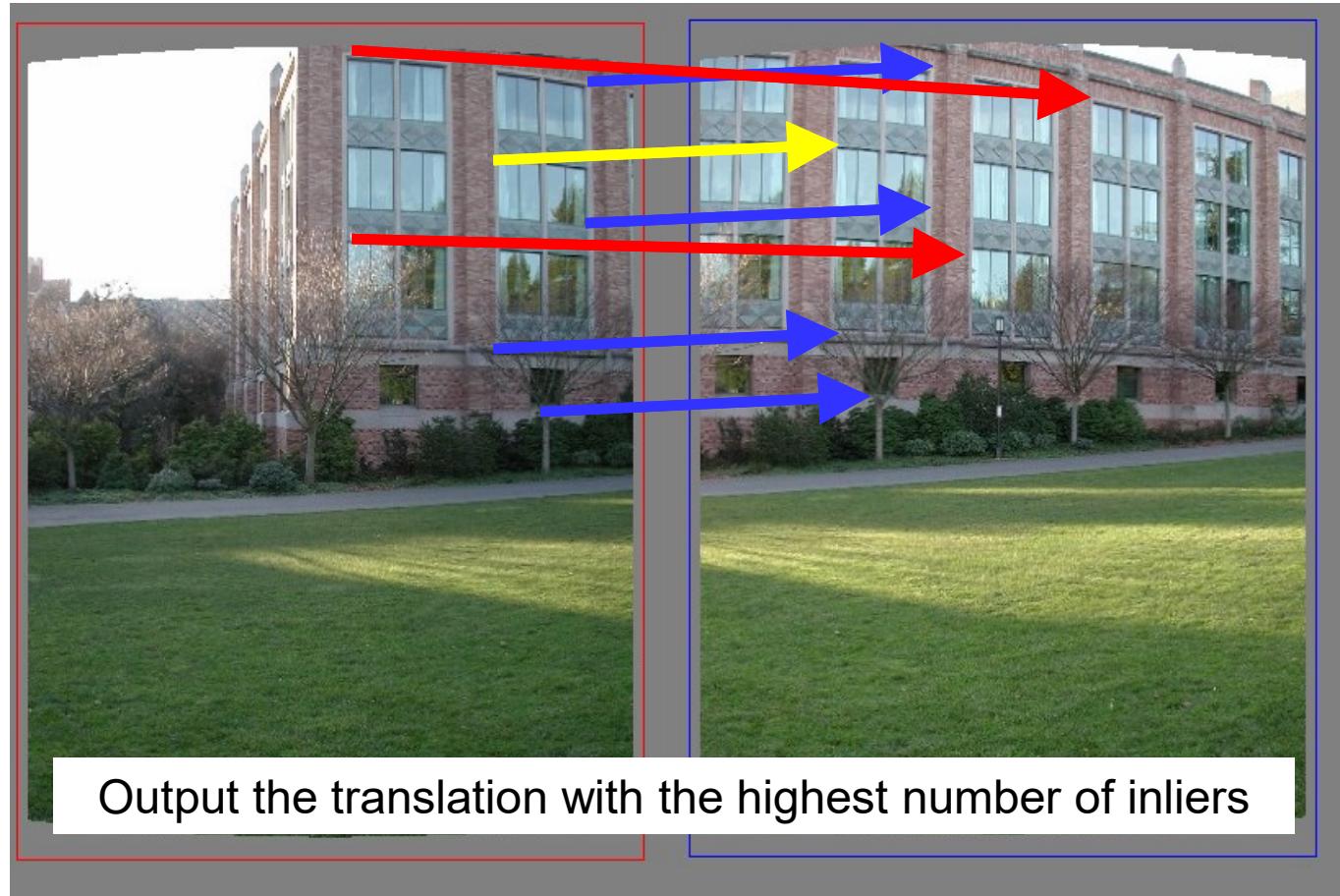
RANdom SAMPle CONSENSUS



RANdom SAMPle CONSENSUS



RANdom SAmple Consensus



RANSAC

- Idea:
 - All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other
- RANSAC only has guarantees if there are < 50% outliers

RANSAC

- **Inlier threshold** related to the amount of noise we expect in inliers
 - Often model noise as Gaussian w/ some standard deviation (e.g. 3 pixels)
- **Number of rounds** related to the percentage of outliers we expect, and the probability of success we'd like to guarantee
 - Suppose there are 20% outliers, and we want to find the correct answer with 99% probability
 - How many rounds do we need?

RANSAC pros and cons

- **Pros**

- Simple and general
- Applicable to many different problems
- Often works well in practice

- **Cons**

- Parameters to tune
- Sometimes too many iterations are required
- Can fail for extremely low inlier ratios
- We can often do better than brute-force sampling

Panoramas

- Now we know how to create panoramas!

- Given two images:

- Step 1: Detect features
- Step 2: Match features
- Step 3: Compute a homography using RANSAC
- Step 4: Combine the images together (somehow)



What if we have more than two images?

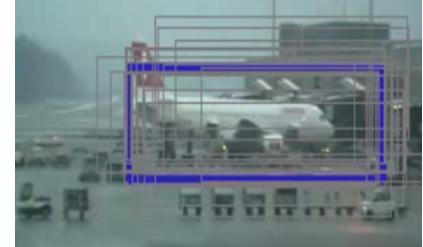
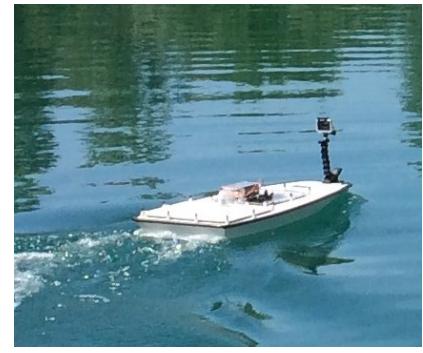
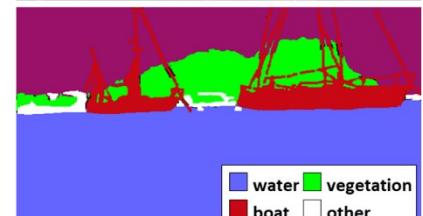
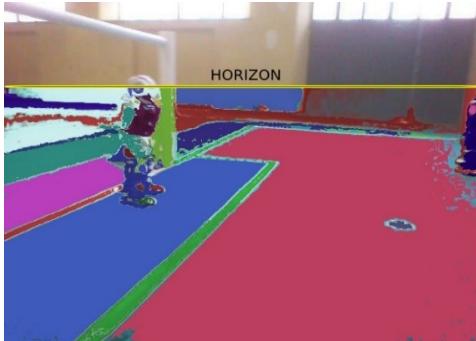
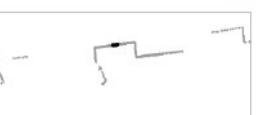


**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Visione e Percezione
A.A. 2019/2020

Feature Matching

Aprile 2020



■ water ■ vegetation
■ boat ■ other