Industrial Computer Vision

- Feature Matching

9th lecture, 2022.11.09

Lecturer: Youngbae Hwang

Contents

- Invariant feature matching
- RANSAC
- Bag of Visual Word

Image matching

by Diva Sian

by swashford

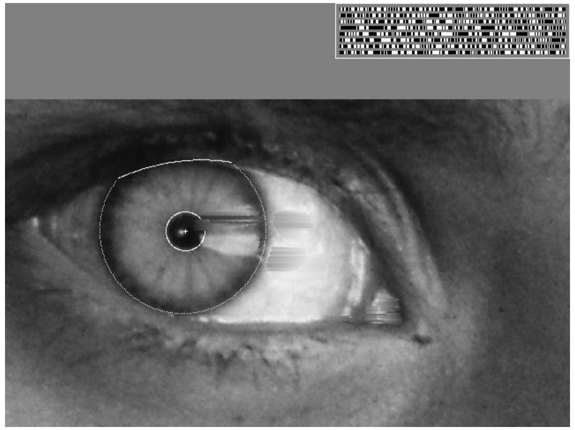
Harder case

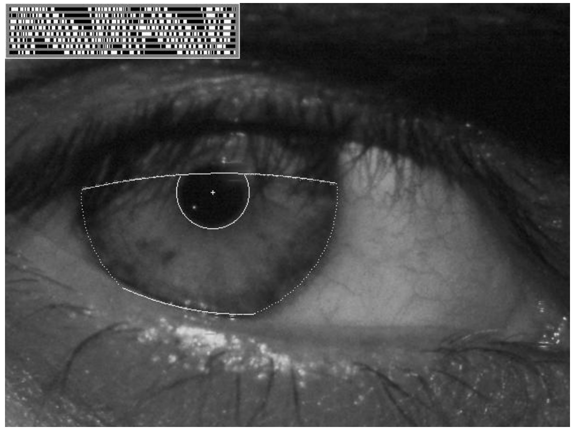
by Diva Sian

by scgbt

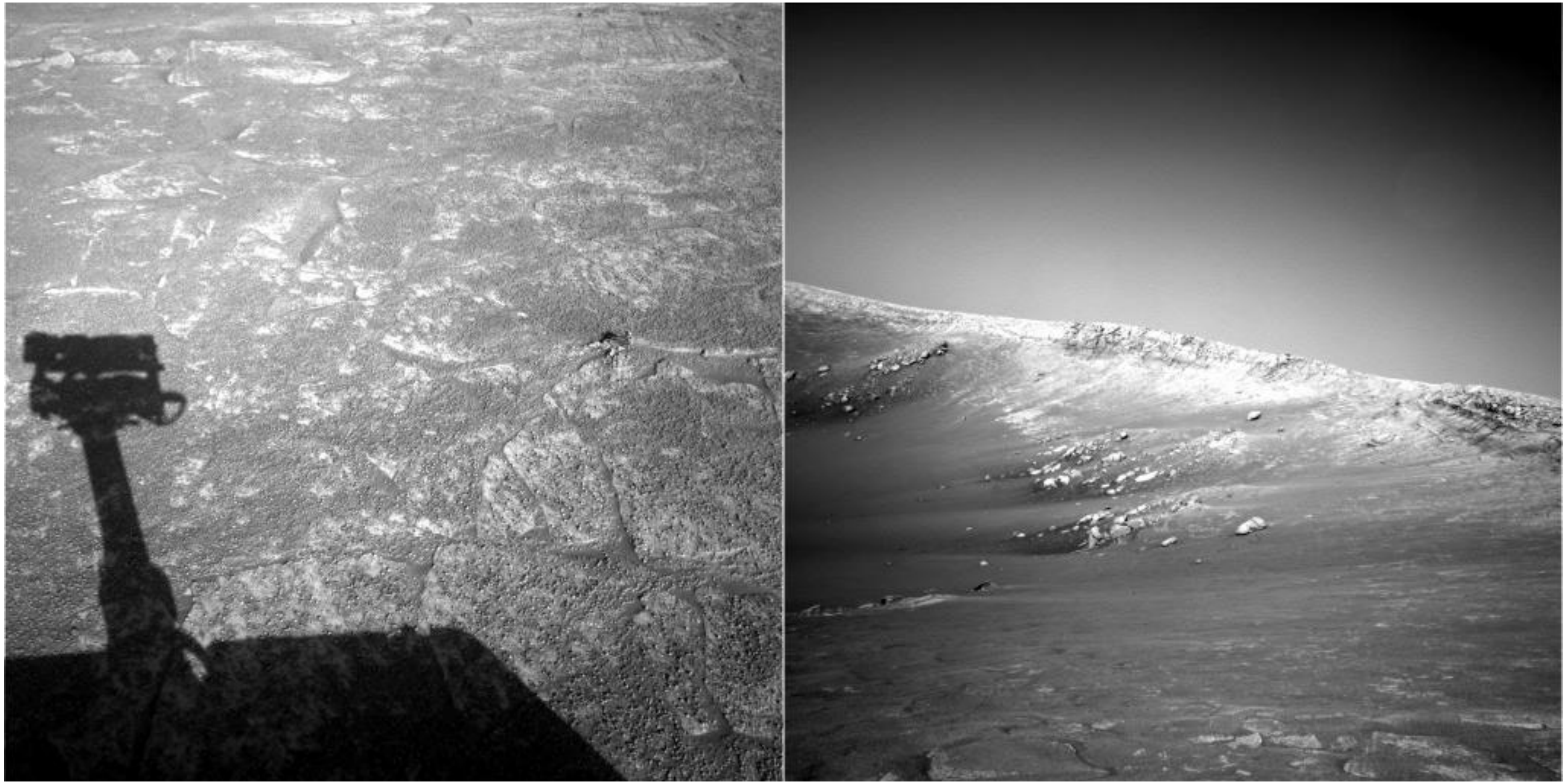
Even harder case

“*How the Afghan Girl was Identified by Her Iris Patterns*” **Read the story**



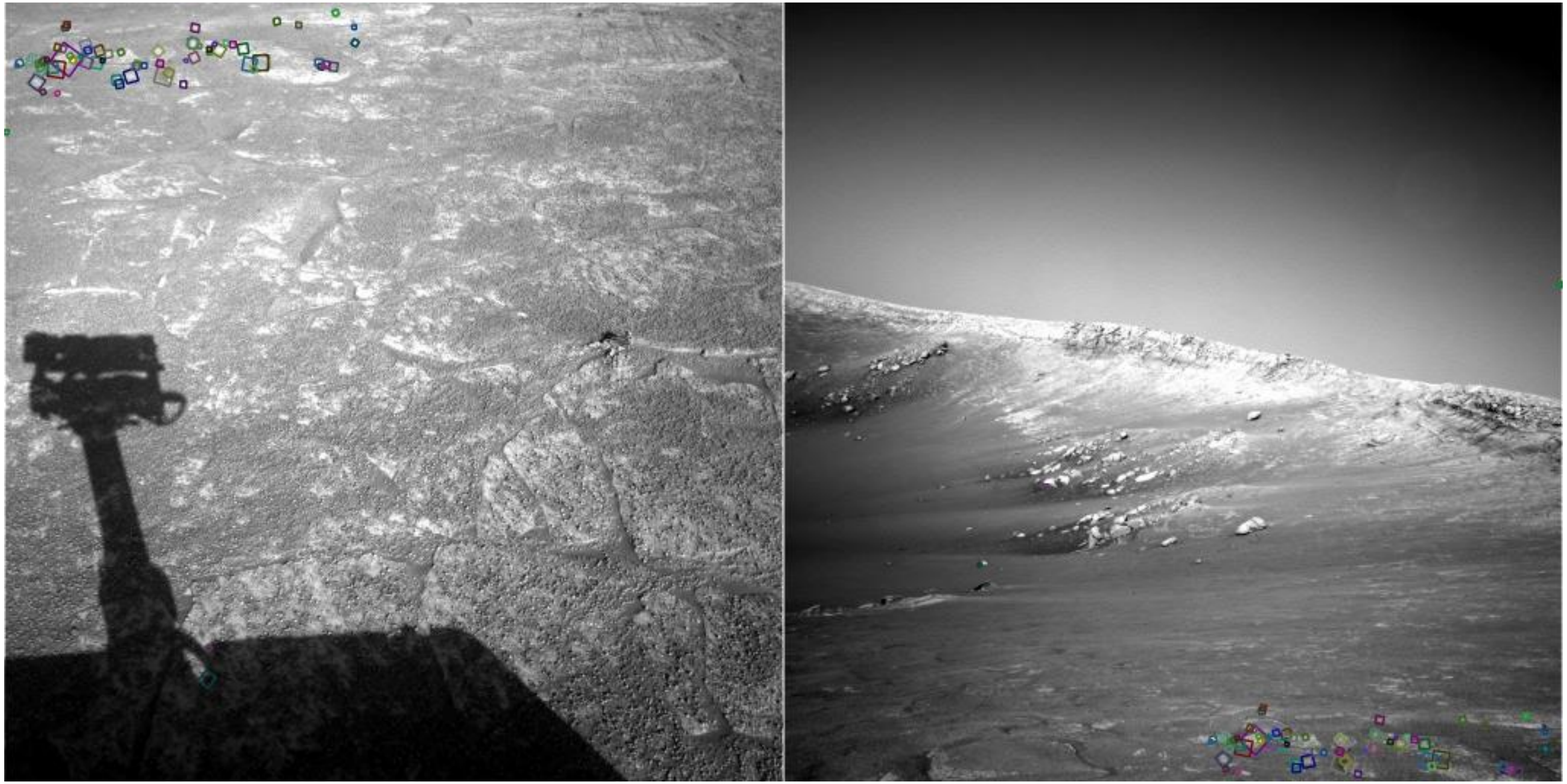


Harder still?



NASA Mars Rover images

Answer below (look for tiny colored squares…)



NASA Mars Rover images
with SIFT feature matches
Figure by Noah Snavely

Image Matching

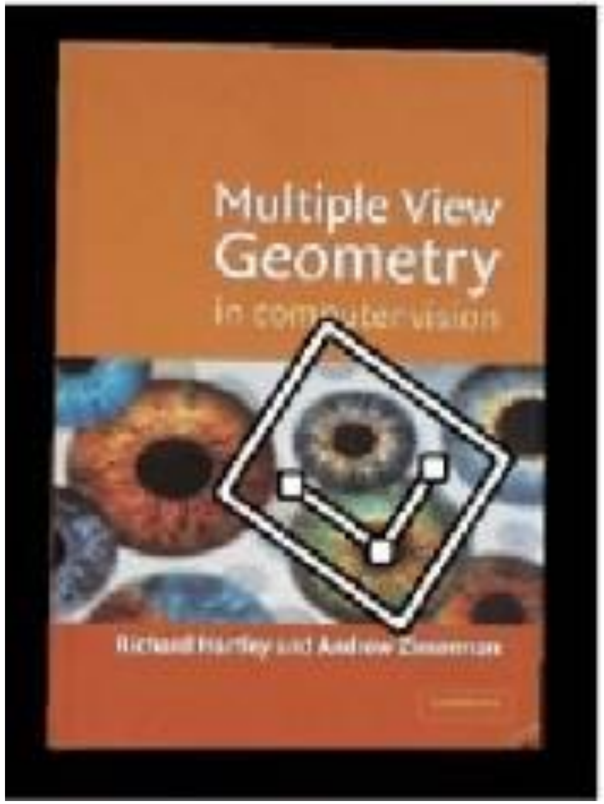


Image Matching

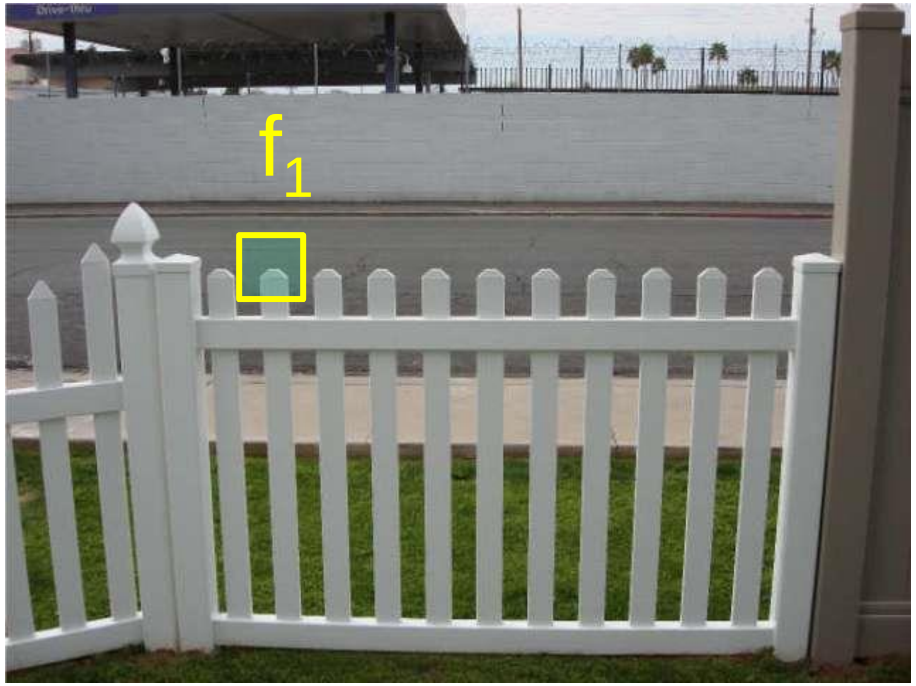
Feature matching

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

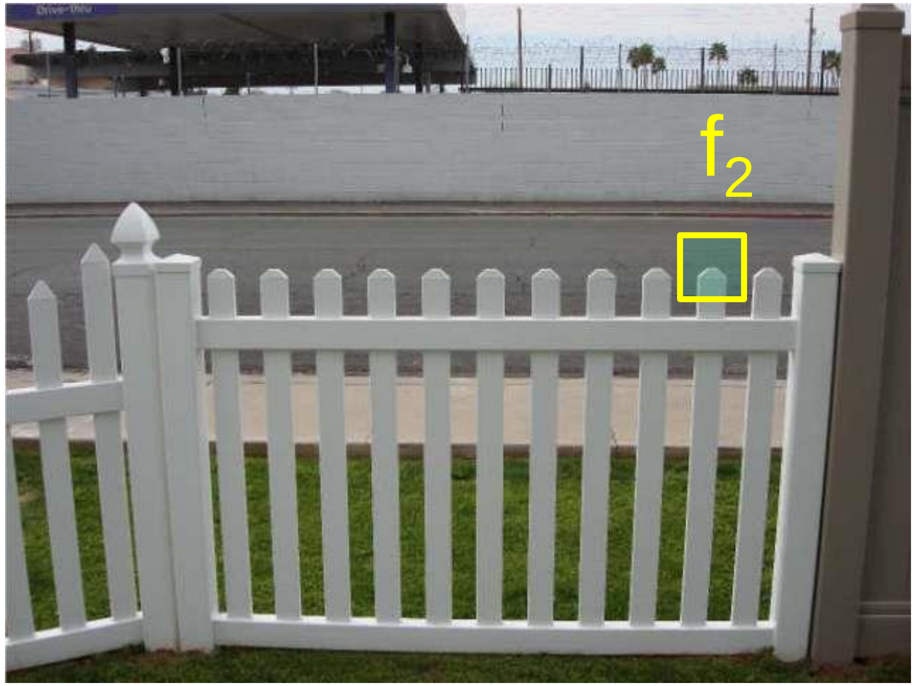
1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

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

- Simple approach is SSD(f_1, f_2)
  - sum of square differences between entries of the two descriptors
  - can give good scores to very ambiguous (bad) matches



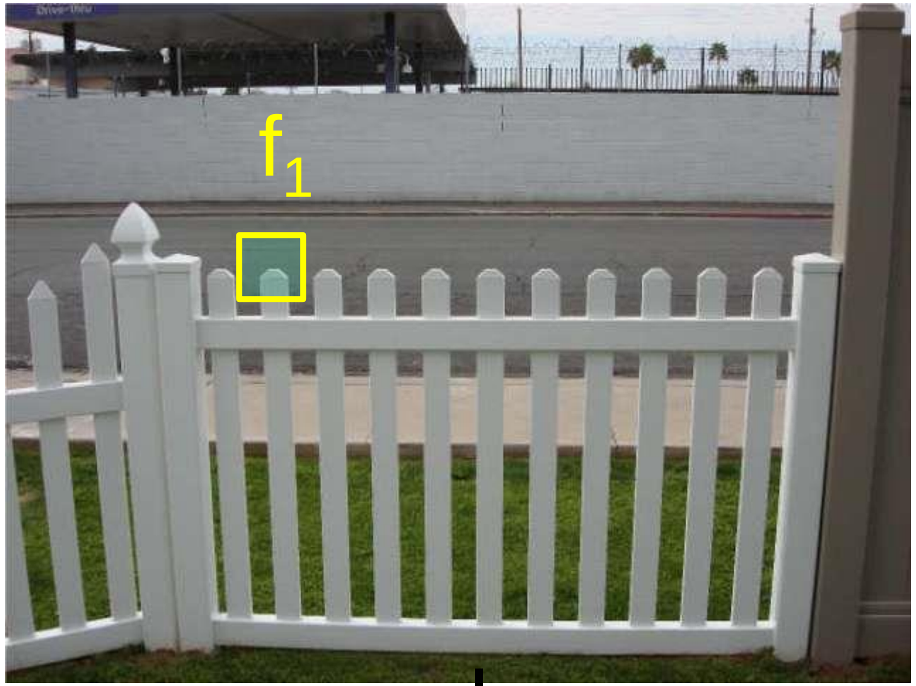
I_1



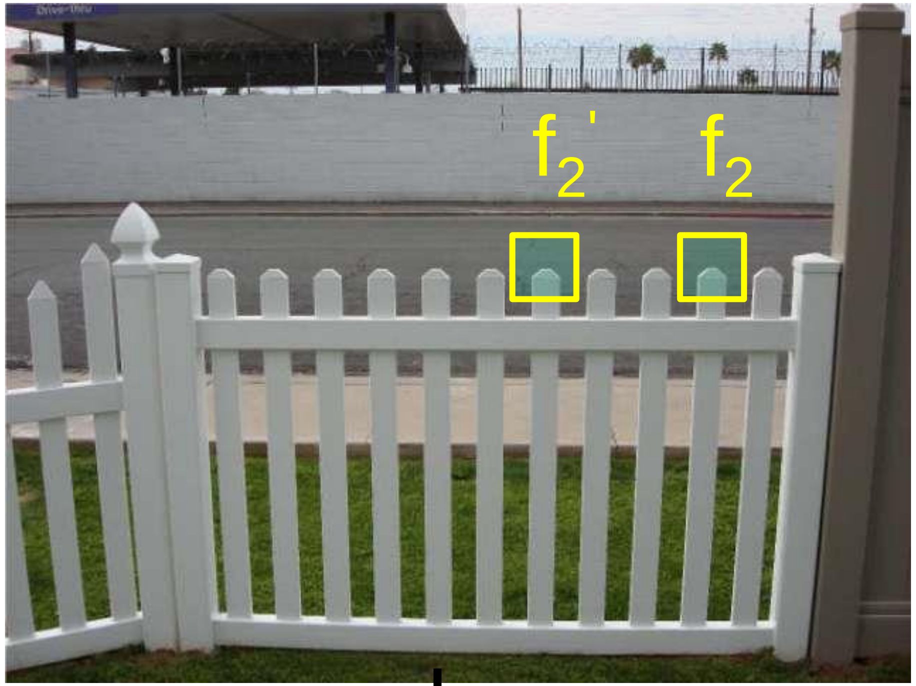
I_2

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

- Better approach: ratio distance = $SSD(f_1, f_2) / SSD(f_1, f_2')$
  - 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
  - gives small values for ambiguous matches



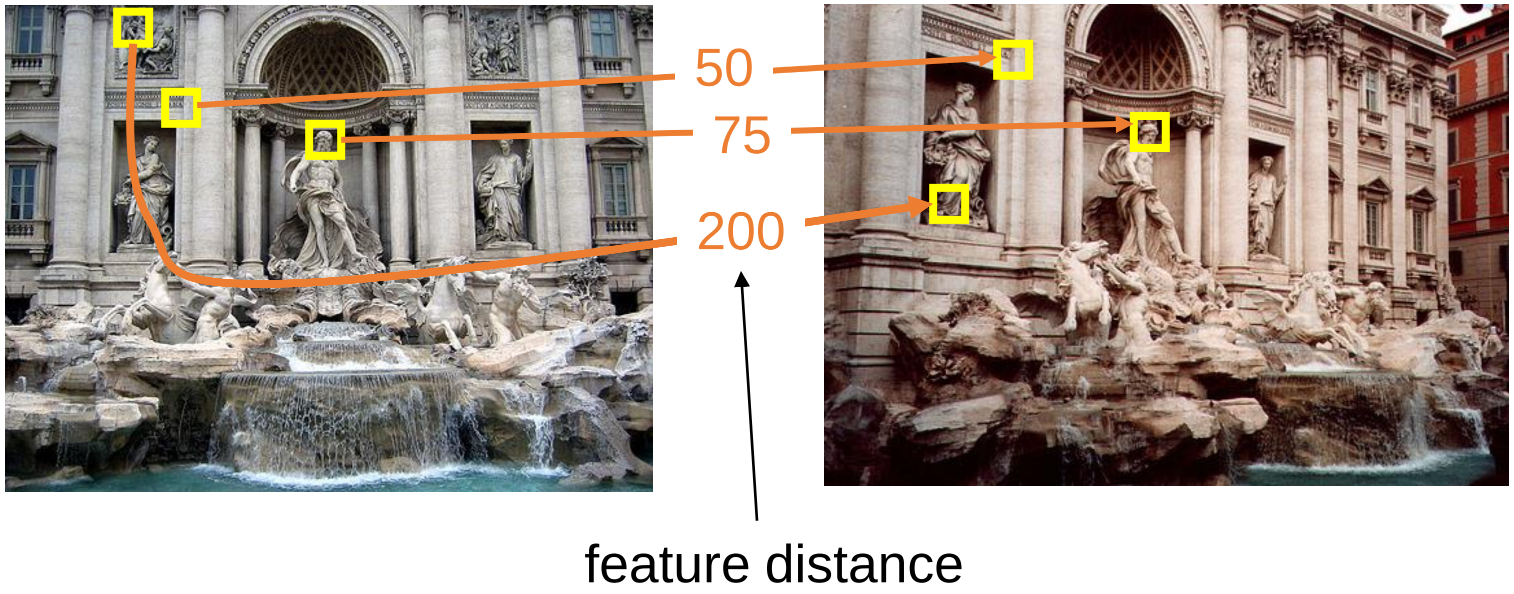
I_1



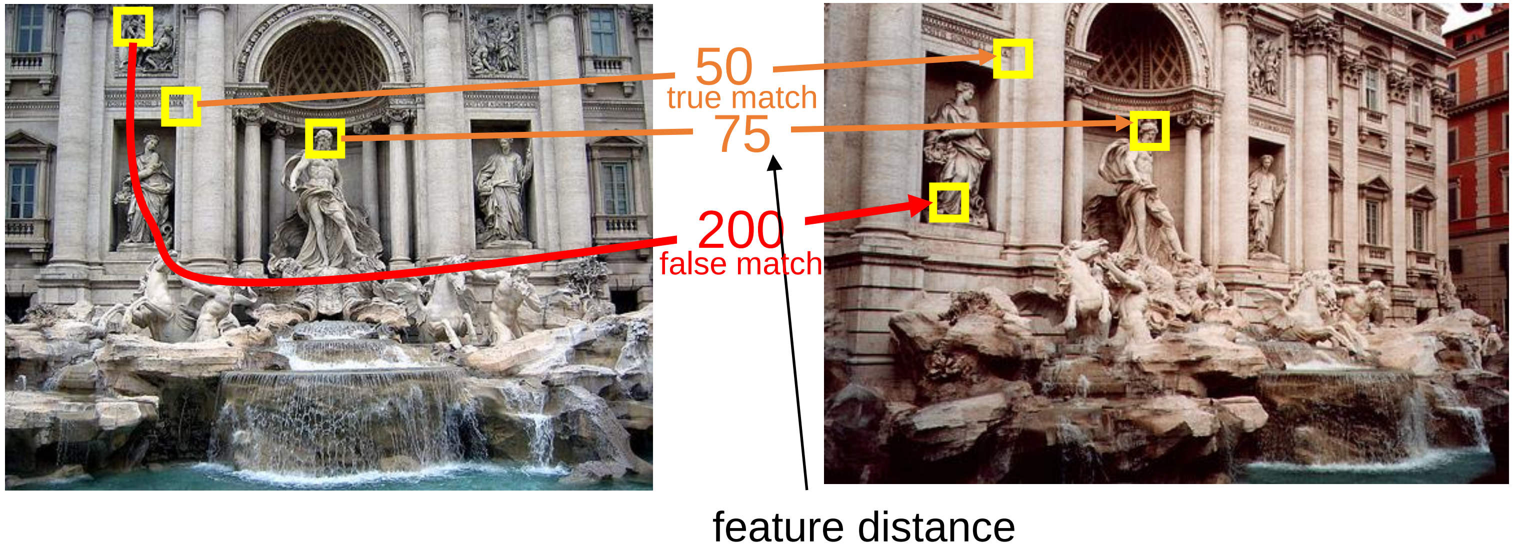
I_2

Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives



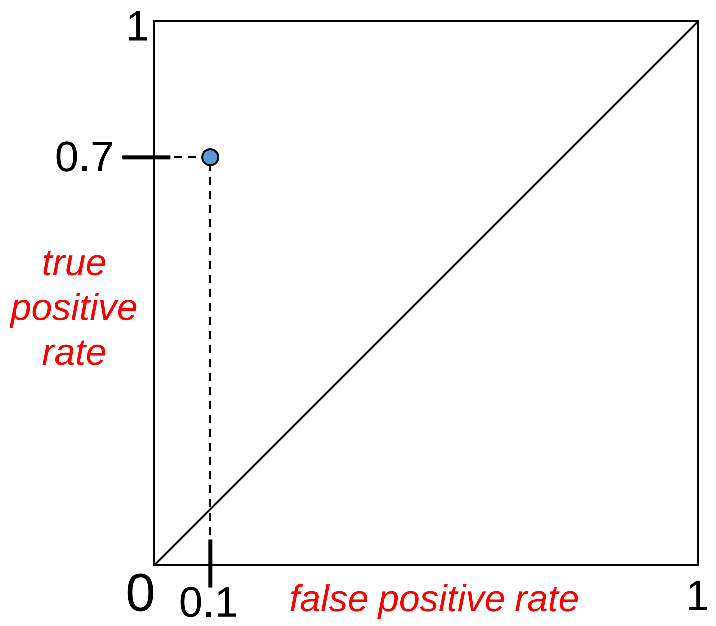
The distance threshold affects performance

- 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?

Evaluating the results

How can we measure the performance of a feature matcher?

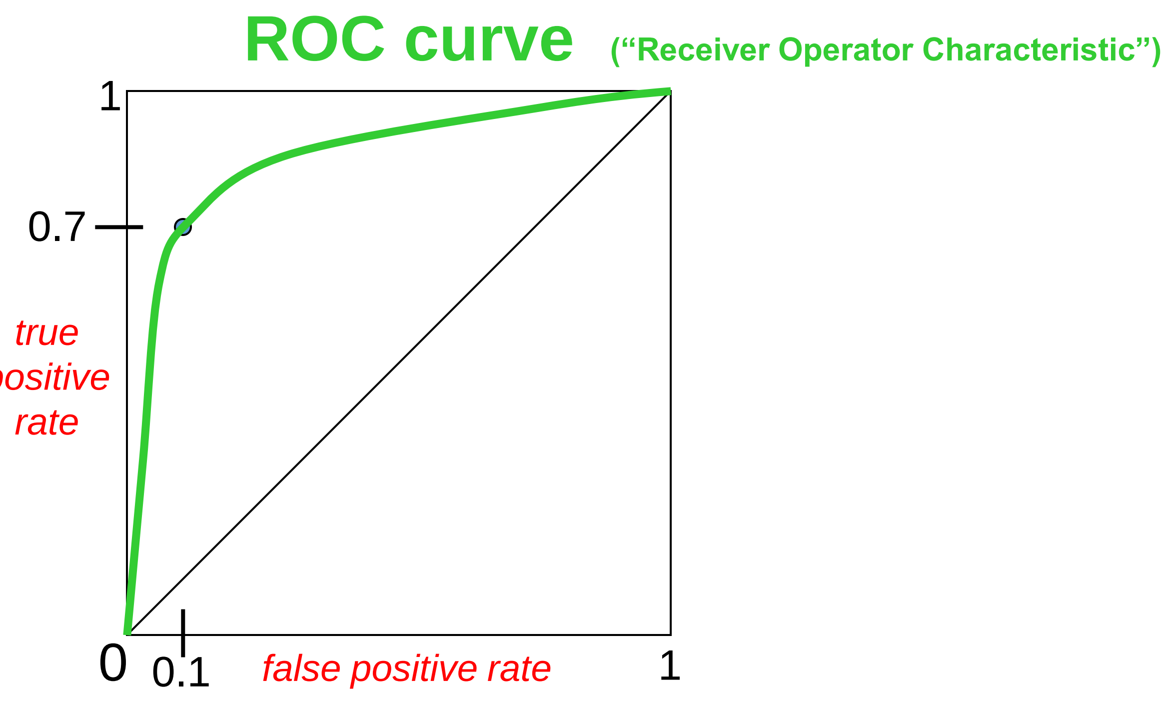
$$\text{true positive rate} = \frac{\text{\# true positives}}{\text{\# matching features (positives)}}$$



$$\text{false positive rate} = \frac{\text{\# false positives}}{\text{\# unmatched features (negatives)}}$$

Evaluating the results

How can we measure the performance of a feature matcher?



$$\text{true positive rate} = \frac{\text{\# true positives}}{\text{\# matching features (positives)}}$$

$$\text{false positive rate} = \frac{\text{\# false positives}}{\text{\# unmatched features (negatives)}}$$

ROC Curves

- Generated by counting # current/incorrect matches, for different threholds
- Want to maximize area under the curve (AUC)
- Useful for comparing different feature matching methods

Matching local features

Matching local features

Image 1

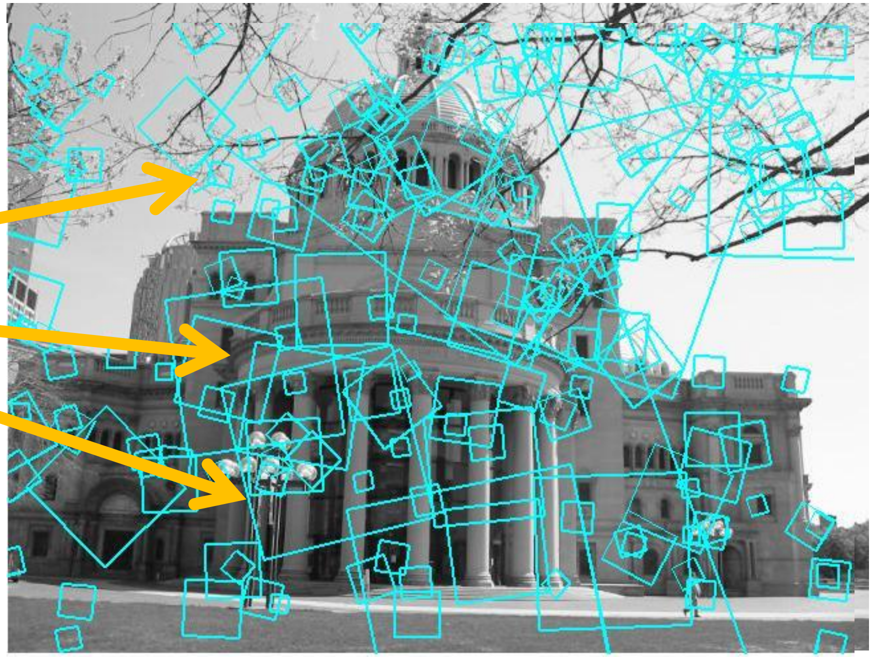


Image 2

To generate **candidate matches**, find patches that have the most similar appearance (e.g., lowest SSD)

Simplest approach: compare them all, take the closest (or closest k, or within a thresholded distance)

Ambiguous matches

Image 1

Image 2

At what SSD value do we have a good match?

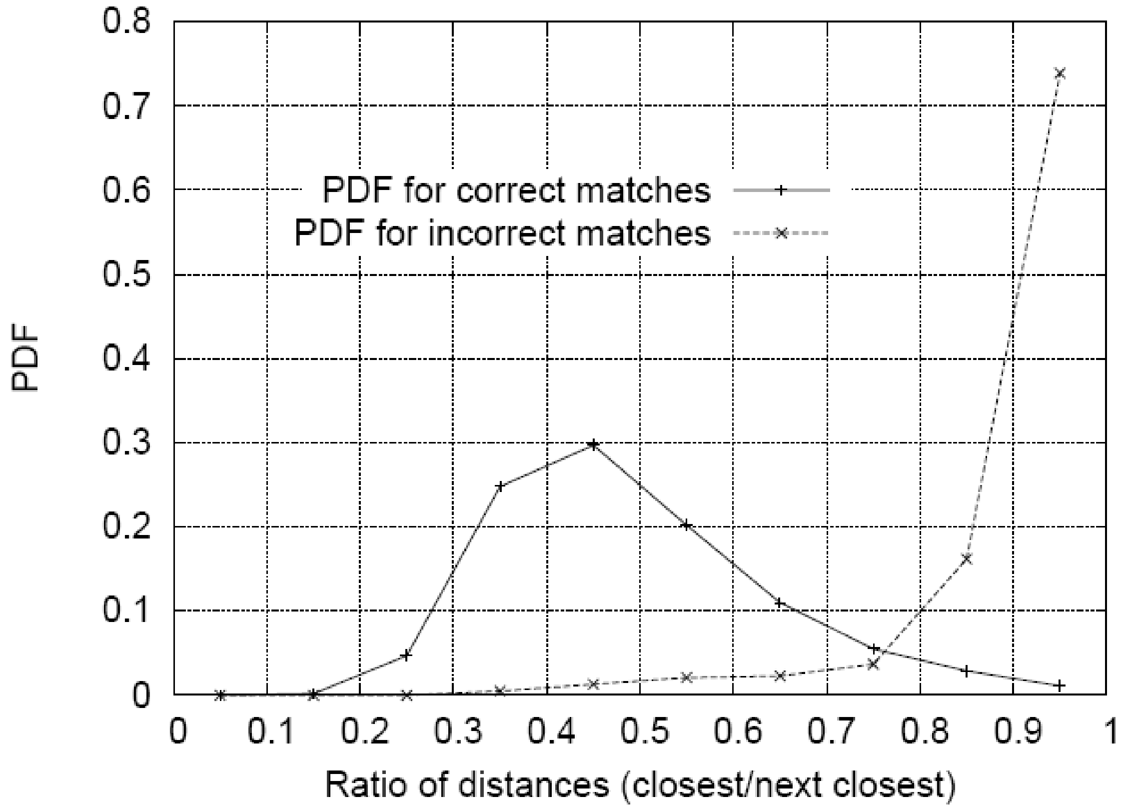
To add robustness to matching, can consider **ratio** :
distance to best match / distance to second best match

If low, first match looks good.

If high, could be ambiguous match.

Matching SIFT Descriptors

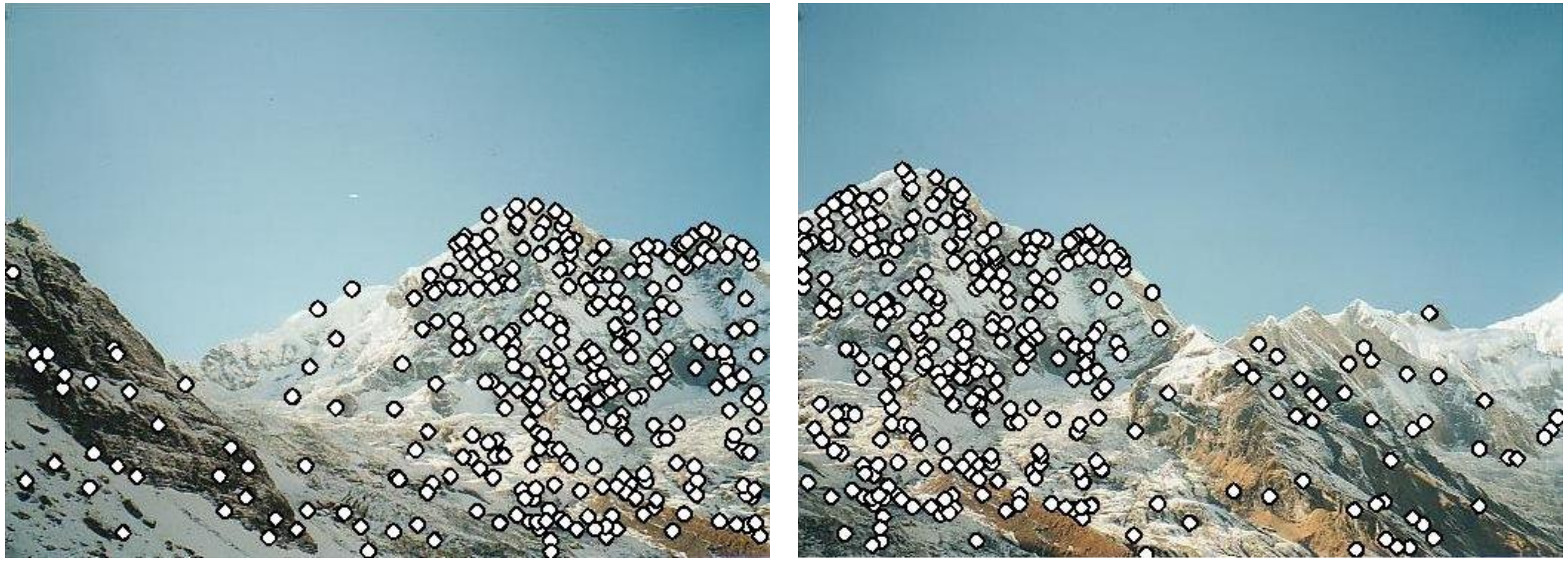
- Nearest neighbor (Euclidean distance)
- Threshold ratio of nearest to 2nd nearest descriptor



Recap: robust feature-based alignment


Source: L. Lazebnik

Recap: robust feature-based alignment



- Extract features

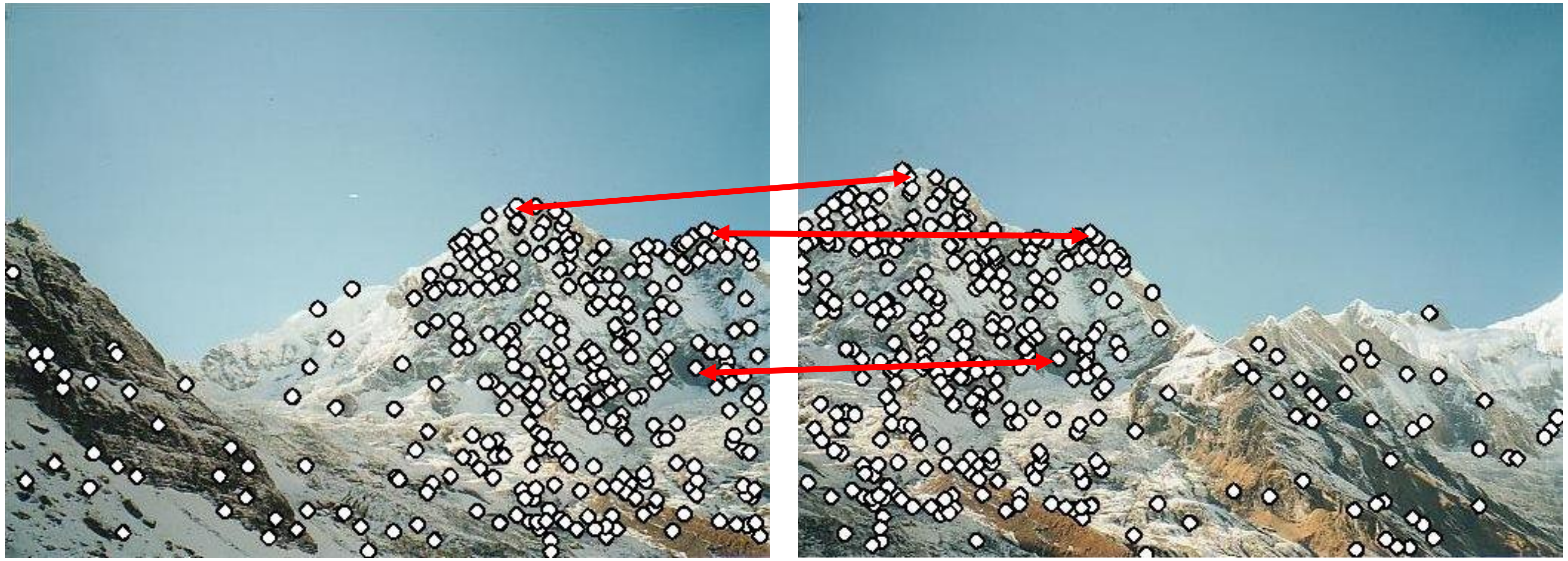
Source: L. Lazebnik

Recap: robust feature-based alignment

- Extract features
- Compute *putative matches*

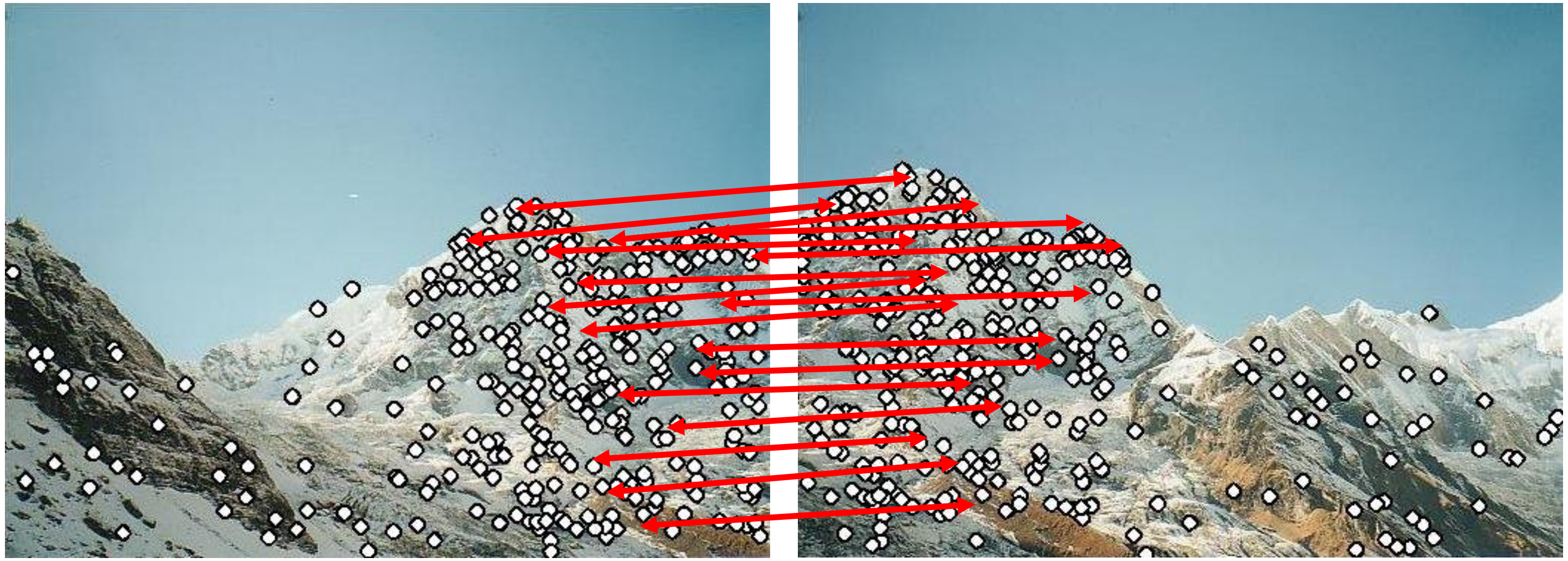
Source: L. Lazebnik

Recap: robust feature-based alignment



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)

Recap: robust feature-based alignment



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

Source: L. Lazebnik

Recap: robust feature-based alignment

- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

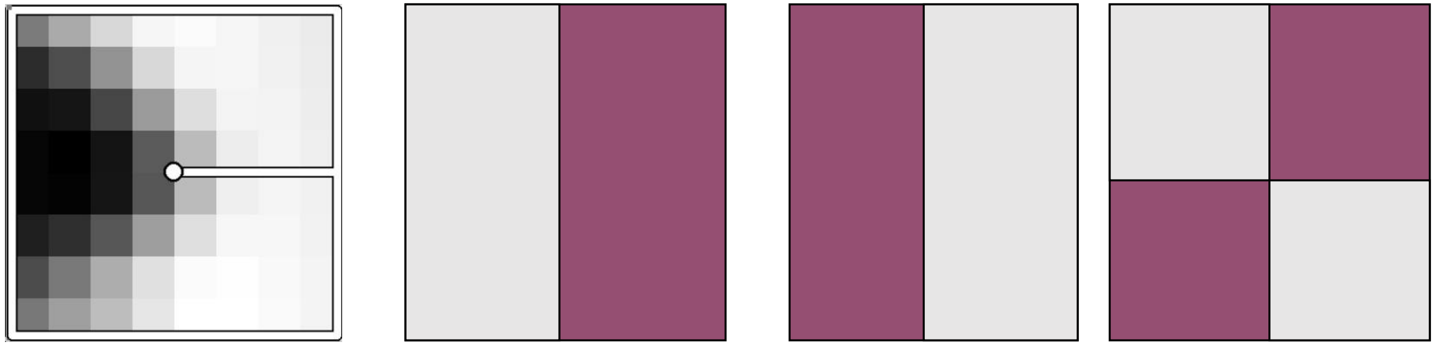
Source: L. Lazebnik

Feature matching

- Exhaustive search
  - for each feature in one image, look at *all* the other features in the other image(s)
- Hashing
  - compute a short descriptor from each feature vector, or hash longer descriptors (randomly)
- Nearest neighbor techniques
  - *k*-trees and their variants (Best Bin First)

Wavelet-based hashing

- Compute a short (3-vector) descriptor from an 8x8 patch using a Haar "wavelet"

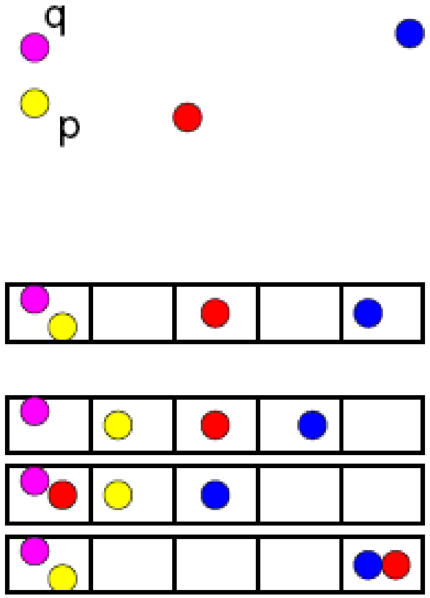


- Quantize each value into 10 (overlapping) bins (10^3 total entries)
- [Brown, Szeliski, Winder, CVPR'2005]

Locality sensitive hashing

[Indyk-Motwani'98]

- Idea: construct hash functions $g: R^d \rightarrow U$ such that for any points p,q:
  - If $D(p,q) \leq r$, then $Pr[g(p)=g(q)]$ is ~~"high"~~ "not-so-small"
  - If $D(p,q) > cr$, then $Pr[g(p)=g(q)]$ is "small"
- Then we can solve the problem by hashing



Nearest neighbor techniques

- *k*-D tree and
- Best Bin First (BBF)

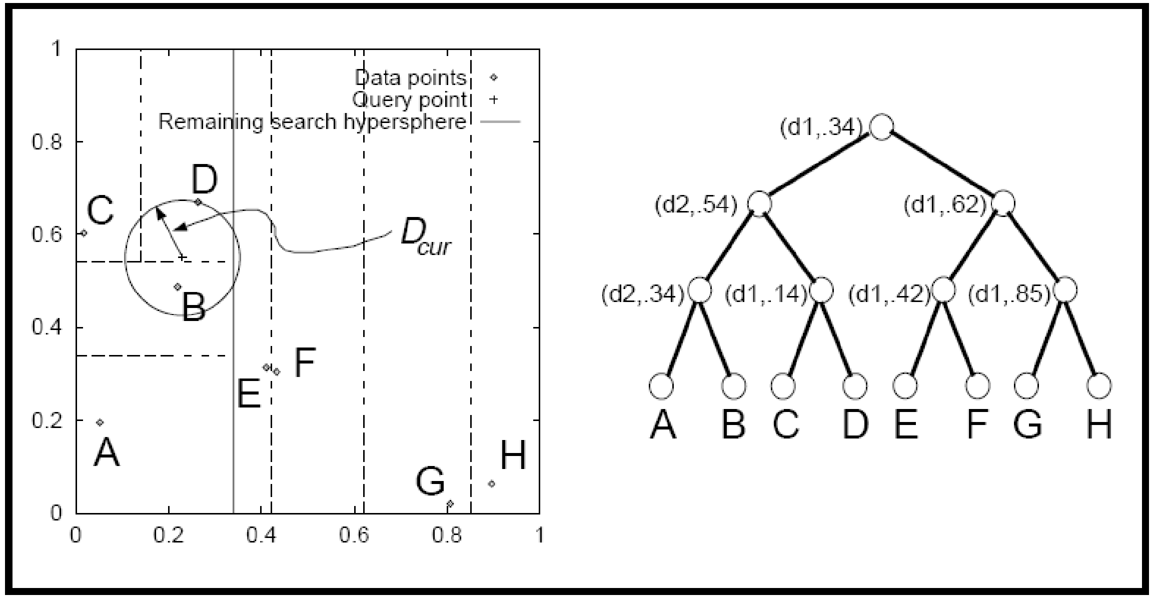
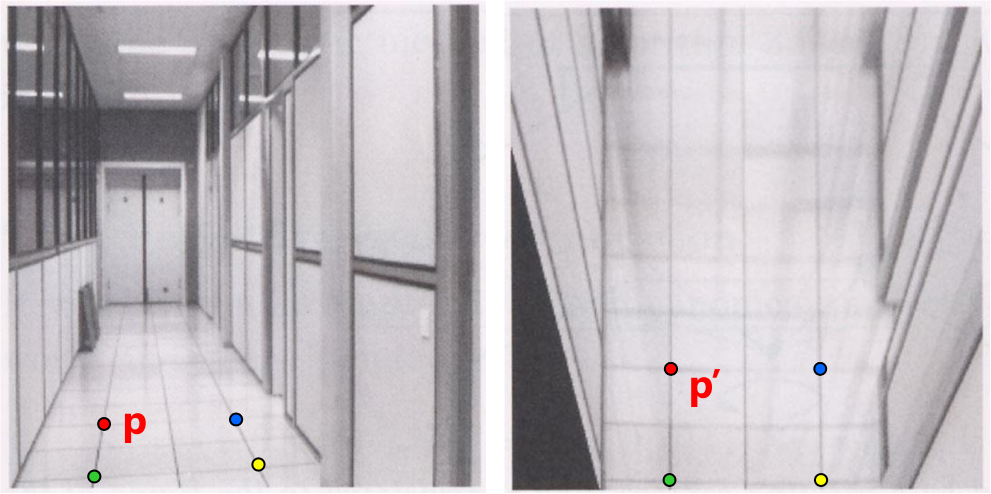


Figure 6: kd-tree with 8 data points labelled A-H, dimension of space k=2. On the right is the full tree, the leaf nodes containing the data points. Internal node information consists of the dimension of the cut plane and the value of the cut in that dimension. On the left is the 2D feature space carved into various sizes and shapes of bin, according to the distribution of the data points. The two representations are isomorphic. The situation shown on the left is after initial tree traversal to locate the bin for query point "+" (contains point D). In standard search, the closest nodes in the tree are examined first (starting at C). In BBF search, the closest bins to query point q are examined first (starting at B). The latter is more likely to maximize the overlap of (i) the hypersphere centered on q with radius D_{cur}, and (ii) the hyperrectangle of the bin to be searched. In this case, BBF search reduces the number of leaves to examine, since once point B is discovered, all other branches can be pruned.

Indexing Without Invariants in 3D Object Recognition, Beis and Lowe, PAMI'99

Homographies



To unwarp (rectify) an image

- solve for homography **H** given **p** and **p'**
- solve equations of the form: w**p'** = **Hp**
  - linear in unknowns: w and coefficients of **H**
  - H is defined up to an arbitrary scale factor
  - how many points are necessary to solve for **H**?

Solving for homographies

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

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

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

Not linear!

$$\begin{aligned} x'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

Solving for homographies

$$
\begin{aligned}
x'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\
y'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12}
\end{aligned}
$$

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

Solving for homographies

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

$$\underset{2n \times 9}{\mathbf{A}} \qquad \underset{9}{\mathbf{h}} \qquad \underset{2n}{\mathbf{0}}$$

Defines a least squares problem: minimize $\|\mathbf{A}\mathbf{h} - \mathbf{0}\|^2$

- Since $\mathbf{h}$ is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenv alue
- Works with 4 or more points

Recap: Two Common Optimization Problems

Problem statement

$$\text{minimize} \quad ||\mathbf{Ax} - \mathbf{b}||^2$$

(least squares solution to $\mathbf{Ax} = \mathbf{b}$)

Solution

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

$\mathbf{x} = \mathbf{A}\backslash\mathbf{b}$ (matlab)

Problem statement

$$\text{minimize} \quad \mathbf{x}^T\mathbf{A}^T\mathbf{Ax} \quad \text{s.t. } \mathbf{x}^T\mathbf{x} = 1$$

(non−trivial lsq solution to $\mathbf{Ax} = 0$)

Solution

$$[\mathbf{v}, \lambda] = \text{eig}(\mathbf{A}^T\mathbf{A})$$

$$\lambda_1 < \lambda_{2..n}: \mathbf{x} = \mathbf{v}_1$$

Computing transformations

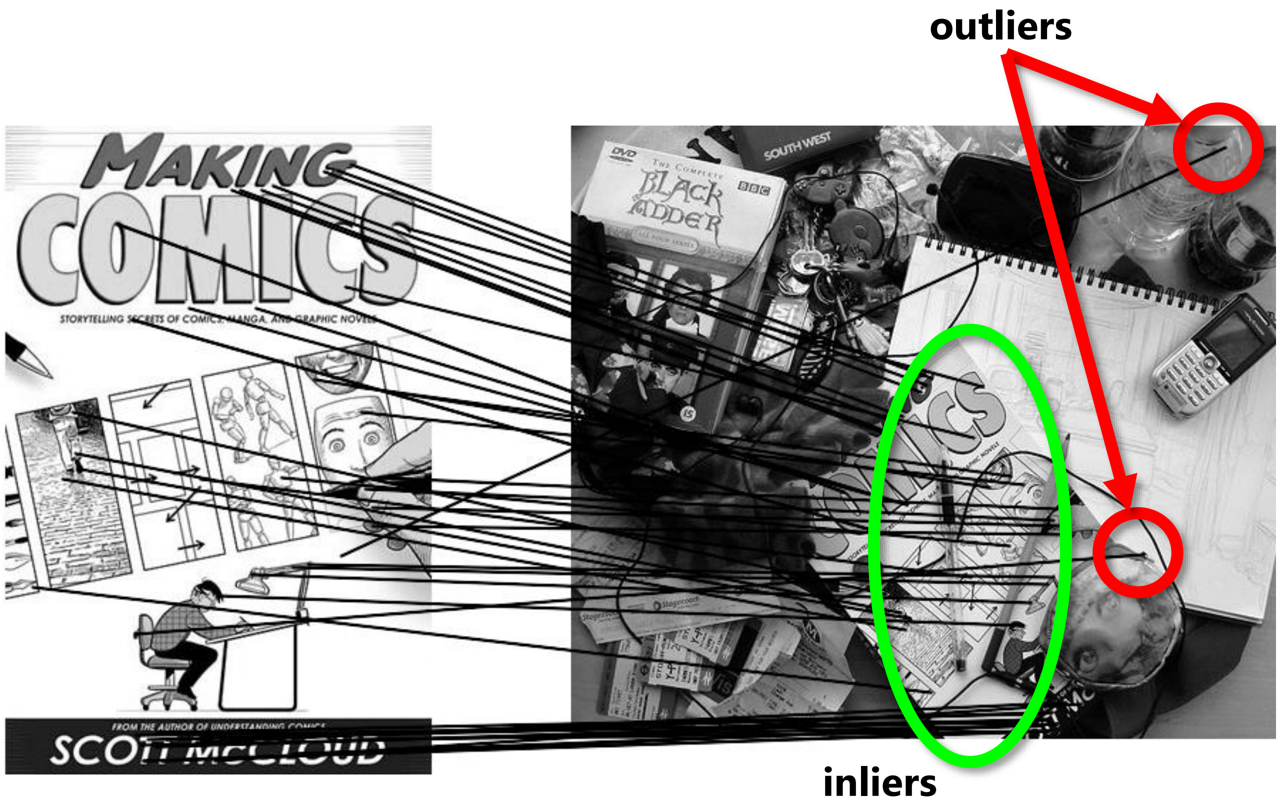
Image Alignment Algorithm

Given images A and B

1. Compute image features for A and B
2. Match features between A and B
3. Compute homography between A and B using least squares on set of match es

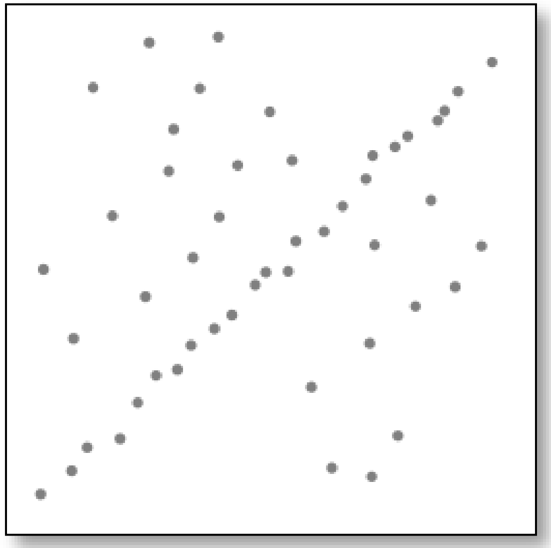
What could go wrong?

Outliers

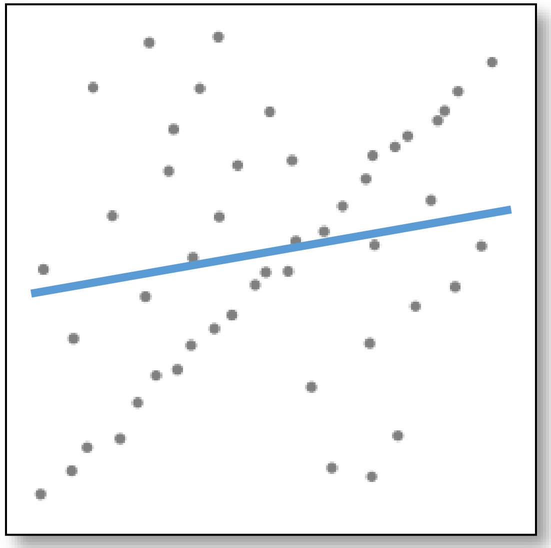

outliers
MAKING COMICS
STORYTELLING SECRETS OF COMICS, MANGA, AND GRAPHIC NOVELS
FROM THE AUTHOR OF UNDERSTANDING COMICS
SCOTT McCLOUD
SOUTH WEST
DVD
THE COMPLETE BLACK ADDER
BBC
inliers

Robustness

- Let's consider the problem of linear regression



Problem: Fit a line to these datapoints



Least squares fit

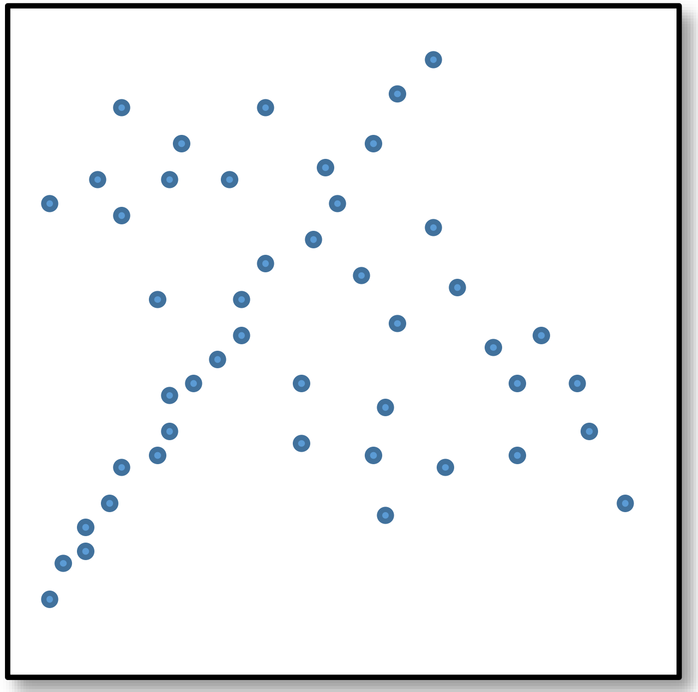
- How can we fix this?

Idea

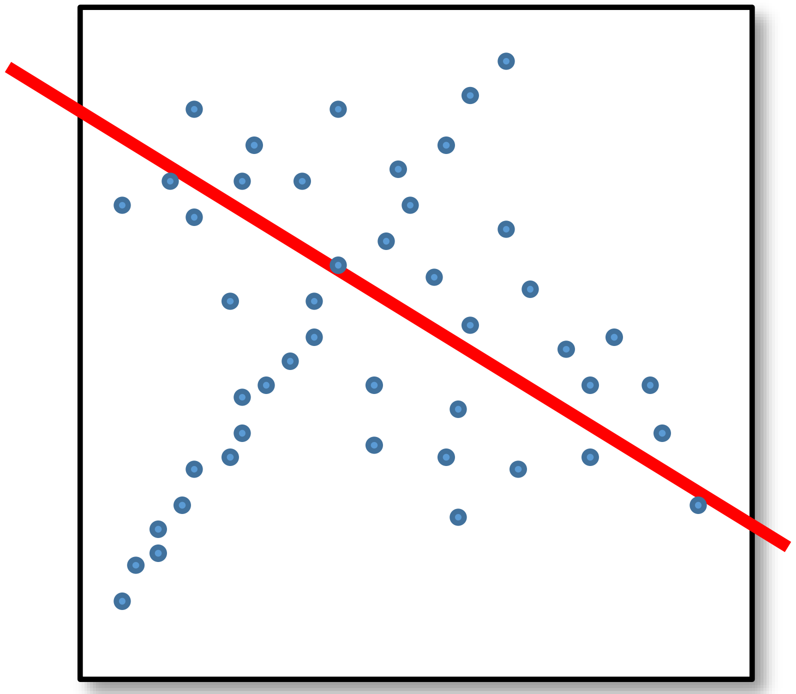
- Given a hypothesized line
- Count the number of points that “agree” with the line
  - “Agree” = within a small distance of the line
  - I.e., the **inliers** to that line

- For all possible lines, select the one with the largest number of inliers

Counting inliers

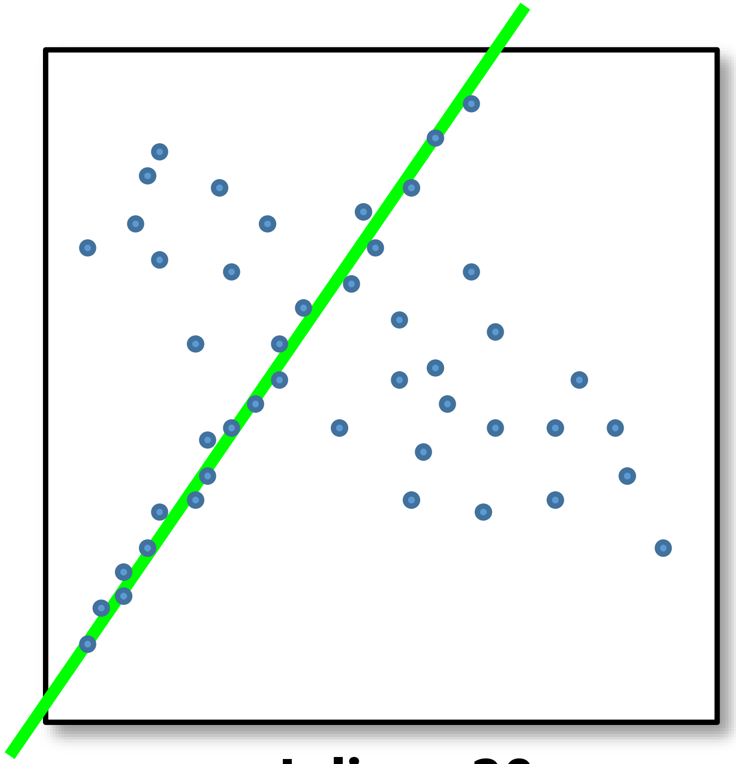


Counting inliers



Inliers: 3

Counting inliers

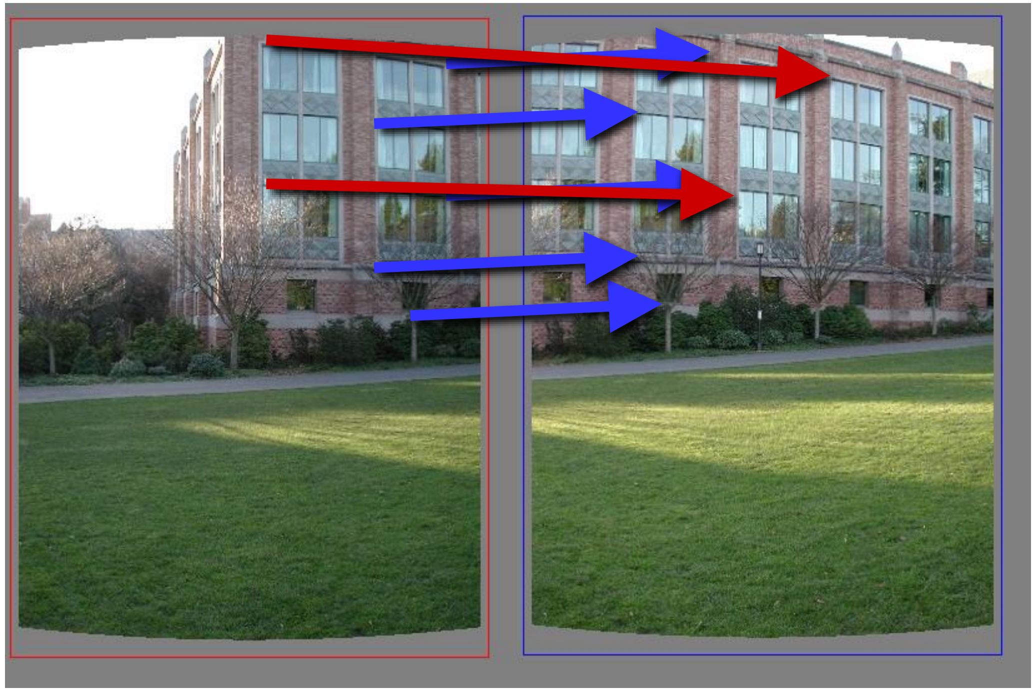


Inliers: 20

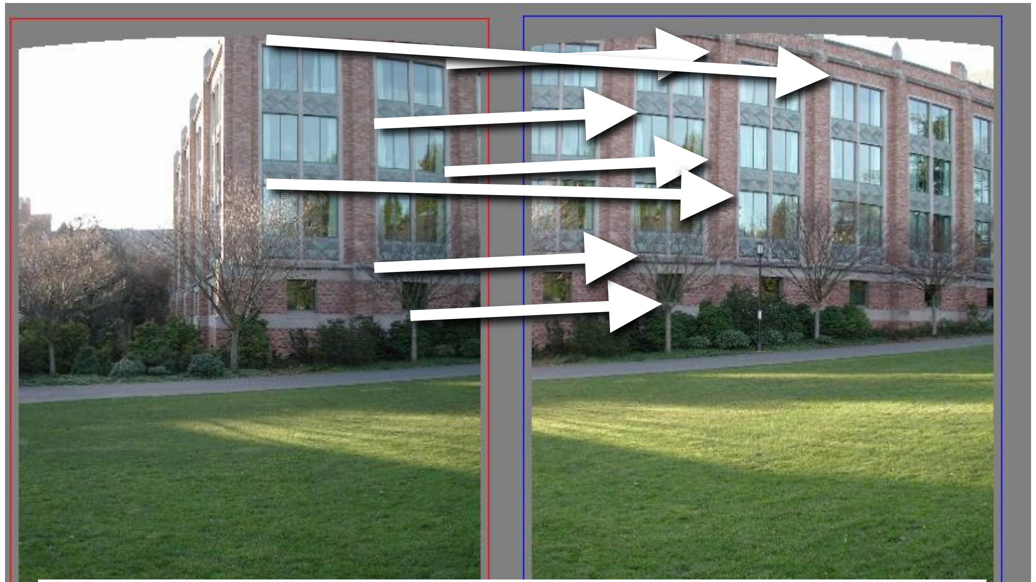
How do we find the best line?

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

Translations

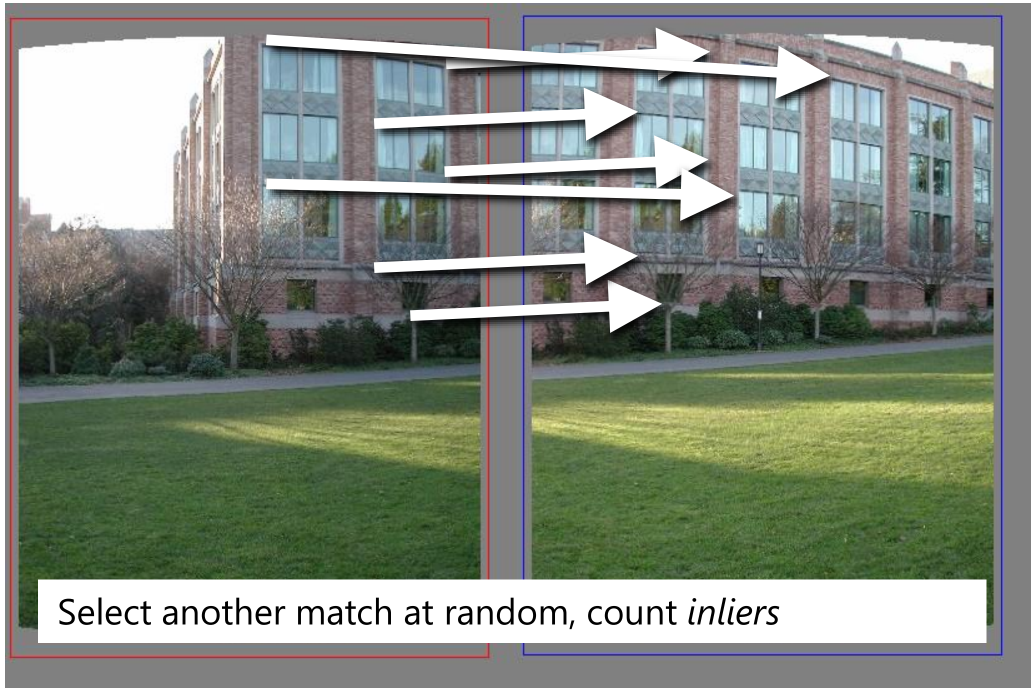


RAndom SAmple Consensus

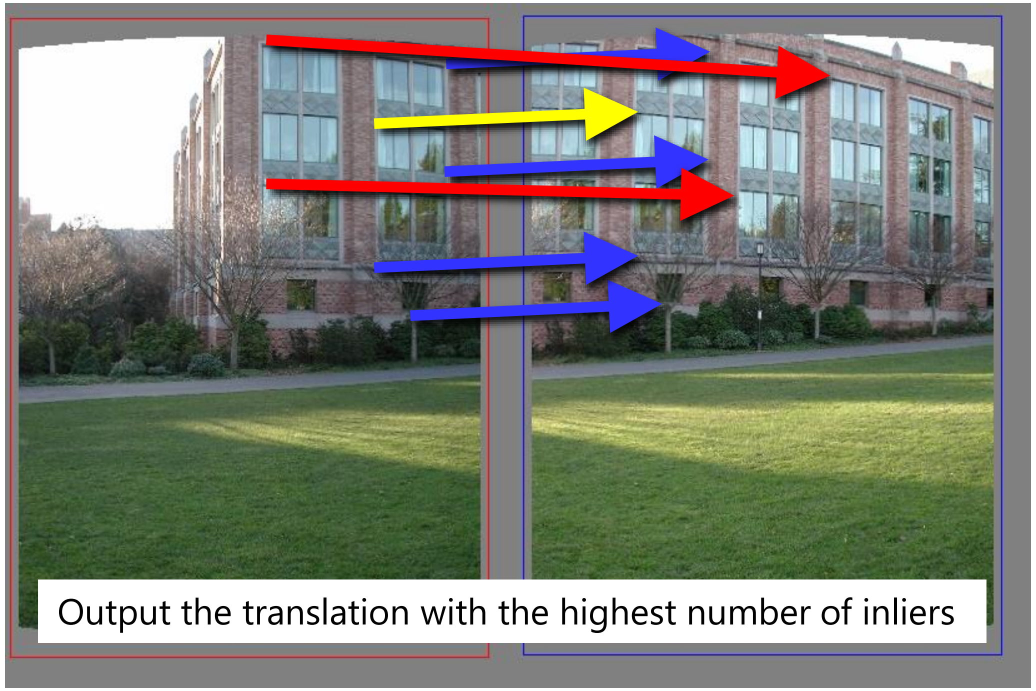


Select *one* match at random, count *inliers*

RAndom SAmple Consensus



RAndom SAmple Consensus



RANSAC

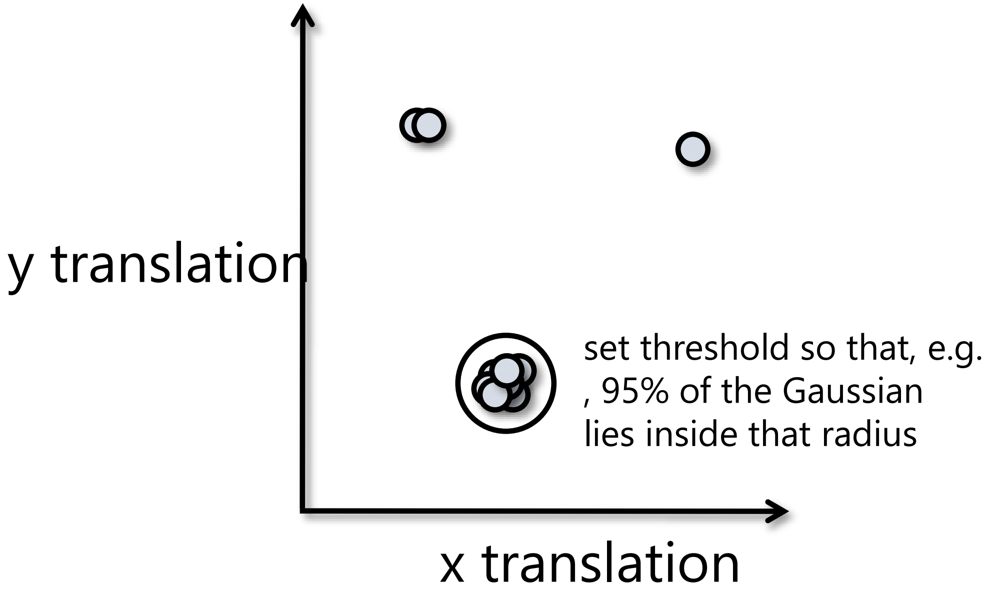
- Idea:
  - All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other
    - RANSAC only has guarantees if there are < 50% outliers
  - "All good matches are alike; every bad match is bad in its own way."

– Tolstoy via Alyosha Efros

RANSAC

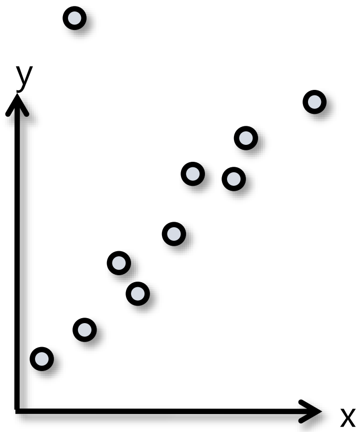
- **Inlier threshold** related to the amount of noise we expect in inliers
  - Often model noise as Gaussian w/ some standard deviation (e.g. 3 pixels)
- **Number of rounds** related to the percentage of outliers we expect, and the probability of success we'd like to guarantee
  - Suppose there are 20%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

RANSAC: Another view

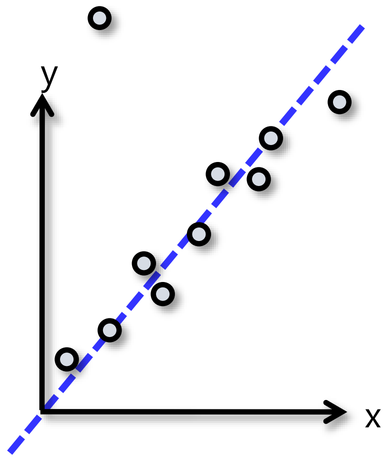


RANSAC

- Back to linear regression
- How do we generate a hypothesis?



- Back to linear regression
- How do we generate a hypothesis?



- General version:
  1. Randomly choose *s* samples
     - Typically *s* = minimum sample size that lets you fit a model
  2. Fit a model (e.g., line) to those samples
  3. Count the number of inliers that approximately fit the m odel
  4. Repeat *N* times
  5. Choose the model that has the largest set of inliers

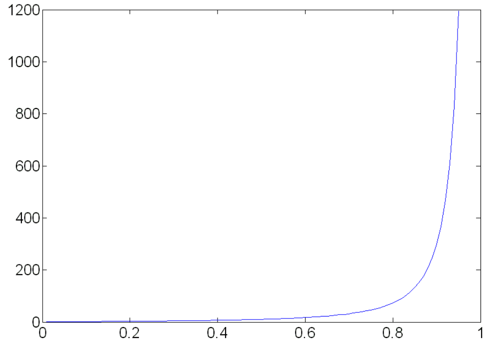
How many rounds?

- If we have to choose *s* samples each time
  - with an outlier ratio *e*
  - and we want the right answer with probability *p*

$$N \geq \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

	proportion of outliers *e*						
s	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

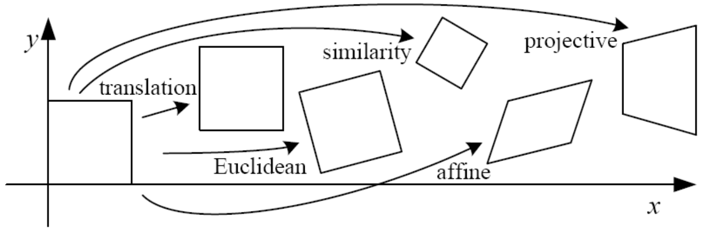
p = 0.99



Source: M. Pollefeys

How big is *s*?

- For alignment, depends on the motion model
  - Here, each sample is a correspondence (pair of matchi ng points)

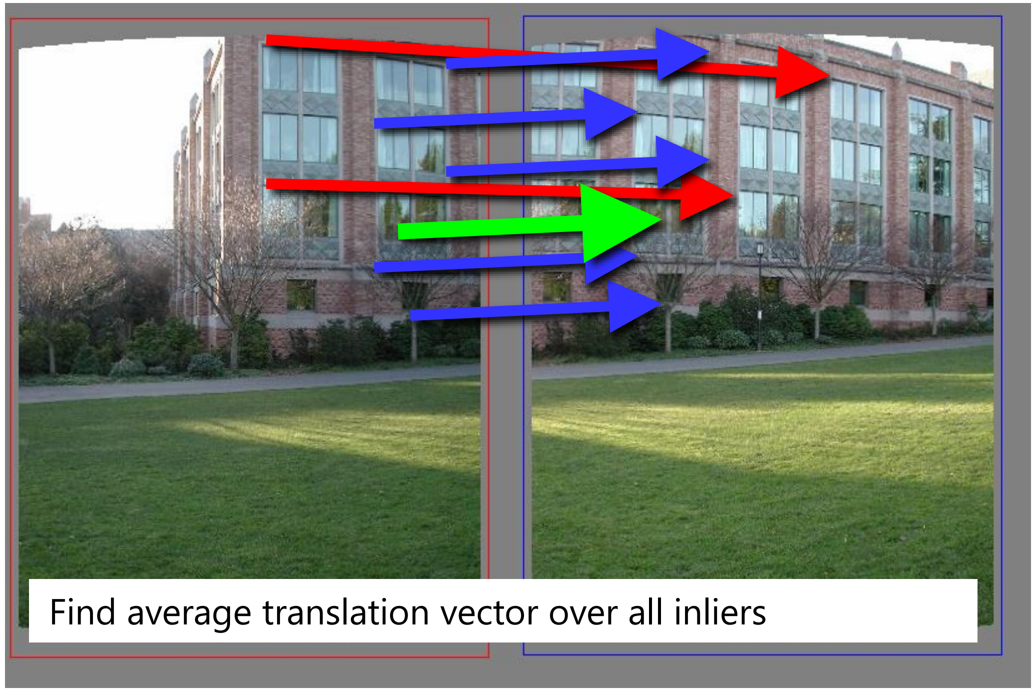


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{2\times 3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times 3}$	8	straight lines	

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

Final step: least squares fit



RANSAC

- An example of a “voting”-based fitting scheme
- Each hypothesis gets voted on by each data point, best hypothesis wins

- There are many other types of voting schemes
  - E.g., Hough transforms…

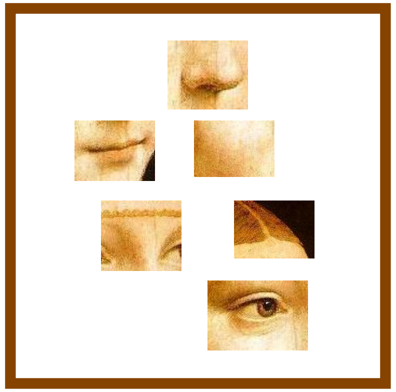
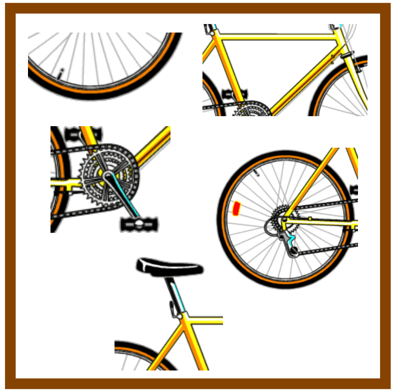
Dictionary Learning:
Learn Visual Words using clustering

Encode:
build Bags-of-Words (BOW) vectors for each image

Classify:
Train and test data using BOWs

Dictionary Learning:
Learn Visual Words using clustering

1. extract features (e.g., SIFT) from images

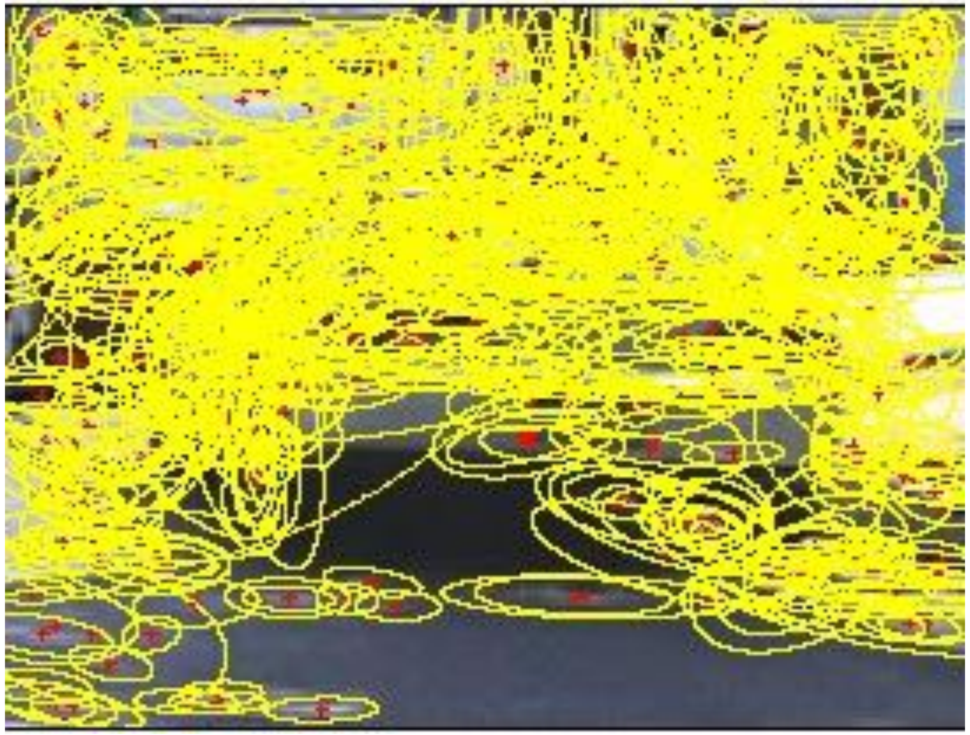



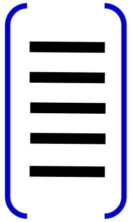

Dictionary Learning:
Learn Visual Words using clustering

2. Learn visual dictionary (e.g., K-means clustering)

What kinds of features can we extract?

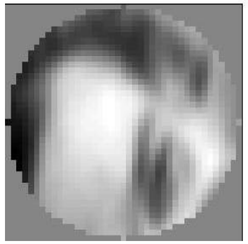
- Regular grid
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005
- Interest point detector
  - Csurka et al. 2004
  - Fei-Fei & Perona, 2005
  - Sivic et al. 2005
- Other methods
  - Random sampling (Vidal-Naquet & Ullman, 2002)
  - Segmentation-based patches (Barnard et al. 2003)



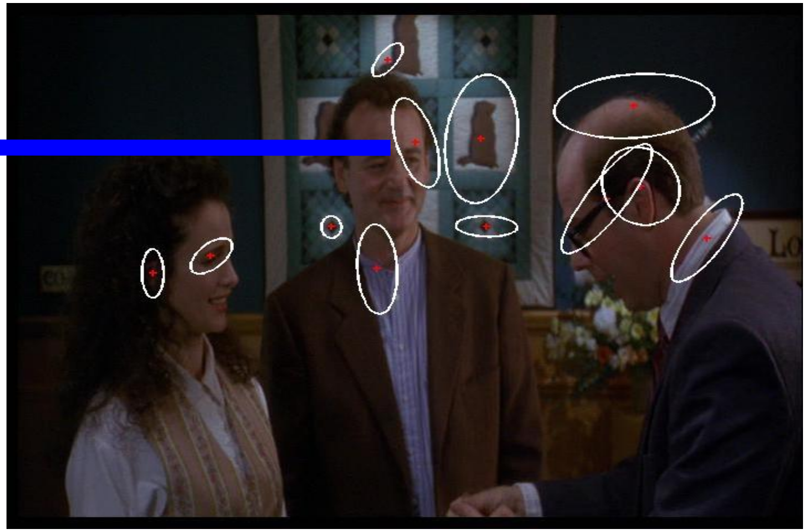


Compute SIFT descriptor

[Lowe'99]



Normalize patch

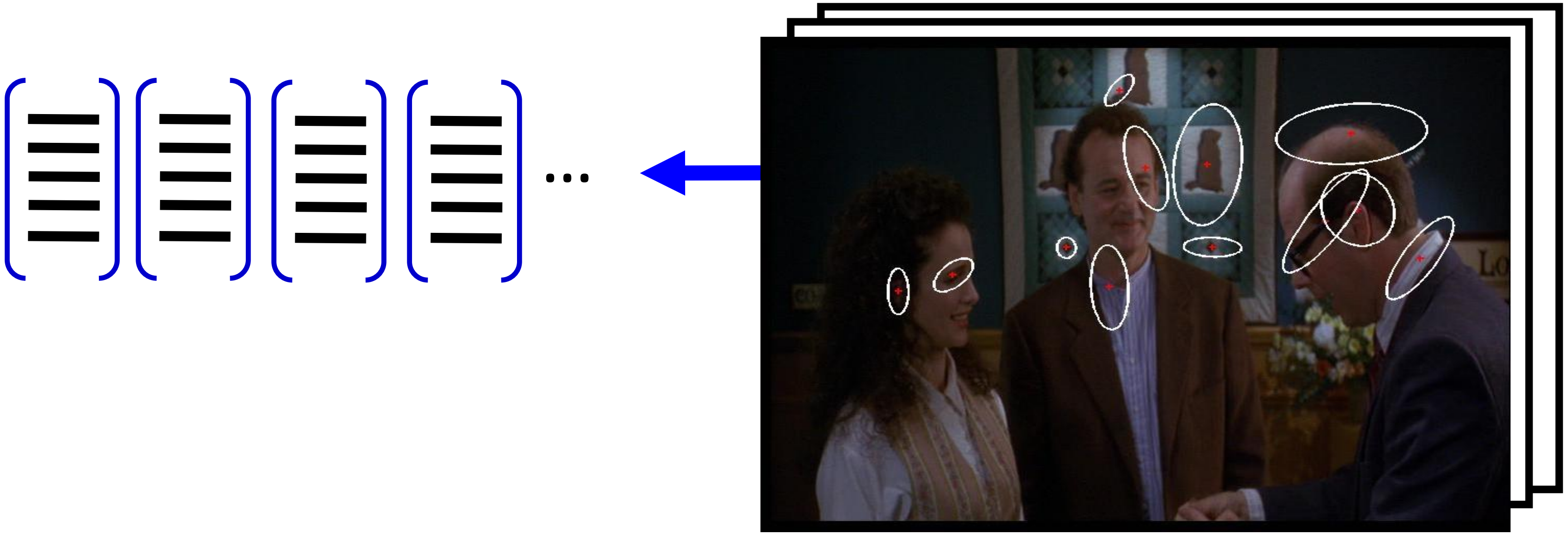


Detect patches

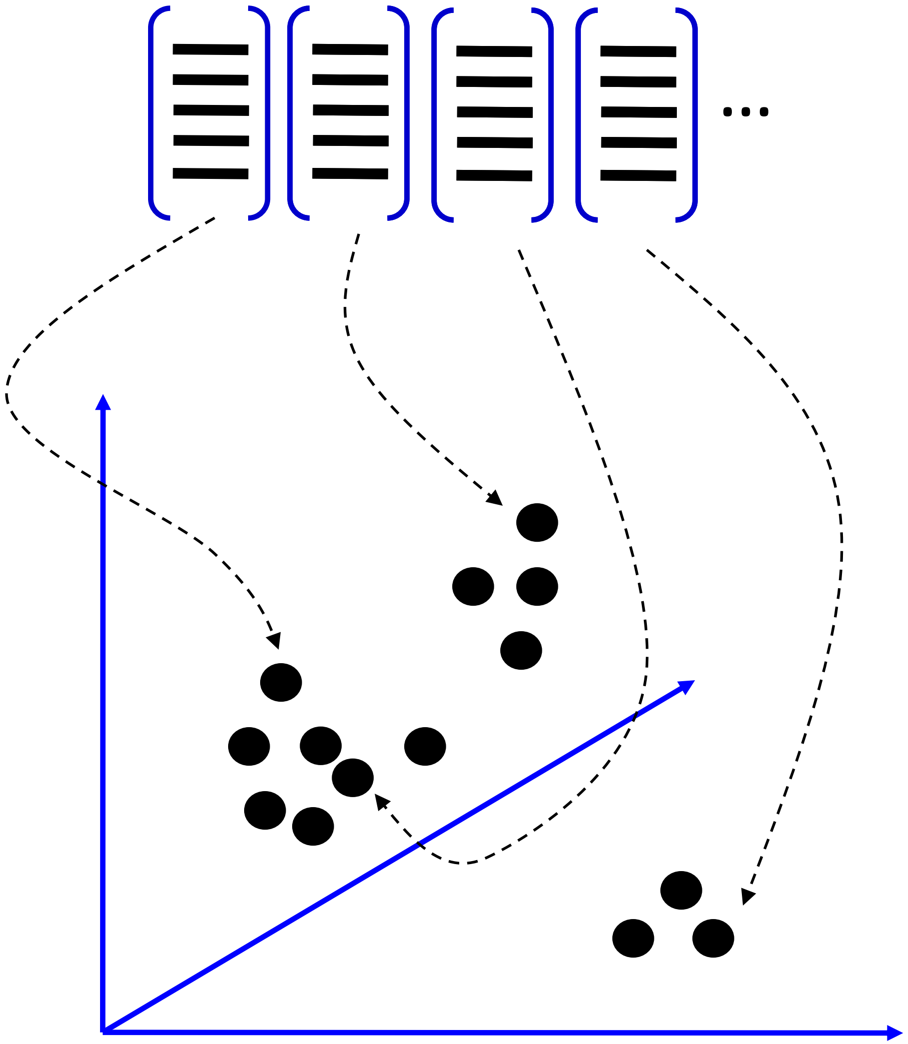
[Mikojaczyk and Schmid '02]

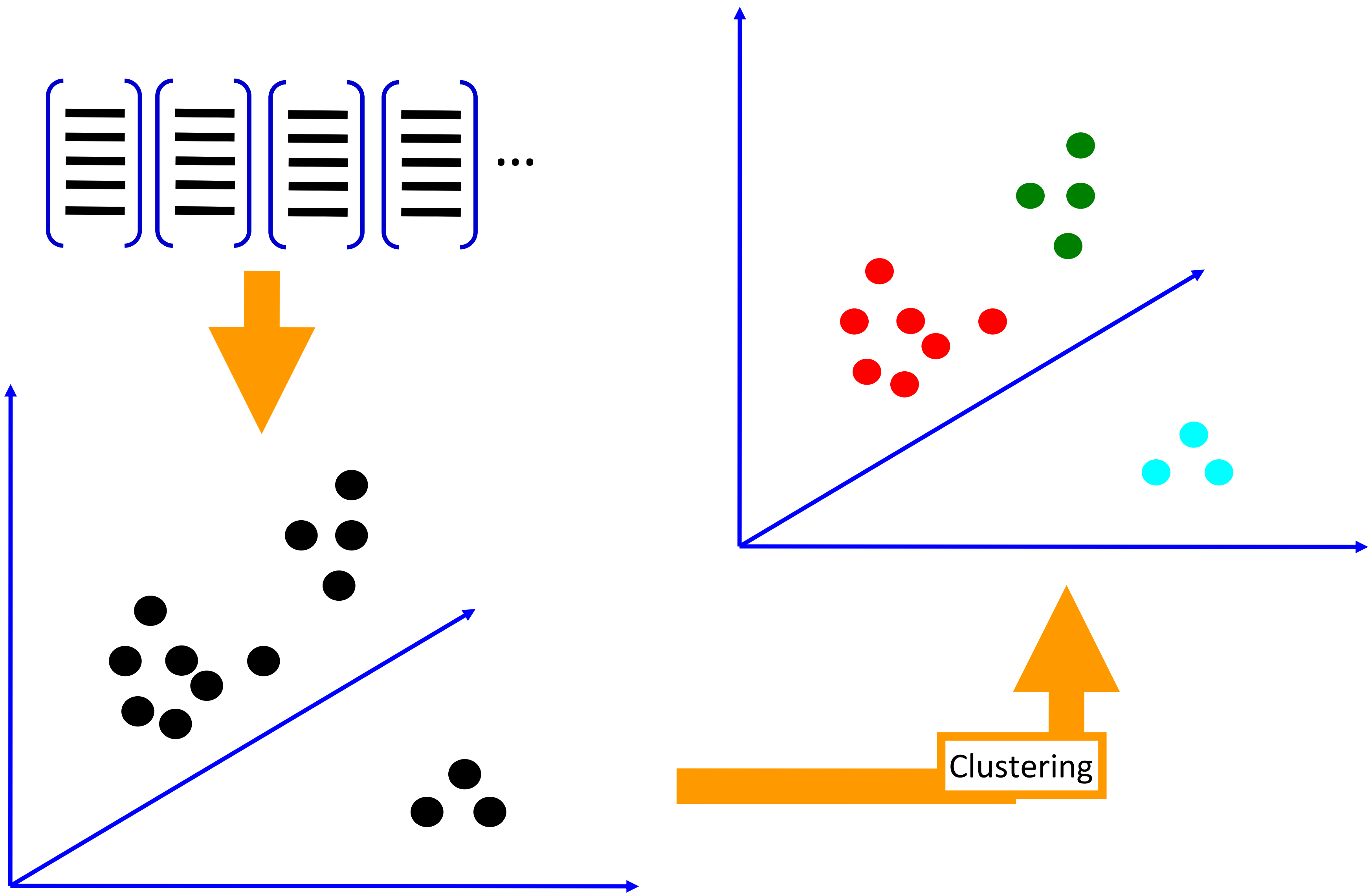
[Mata, Chum, Urban & Pajdla, '02]

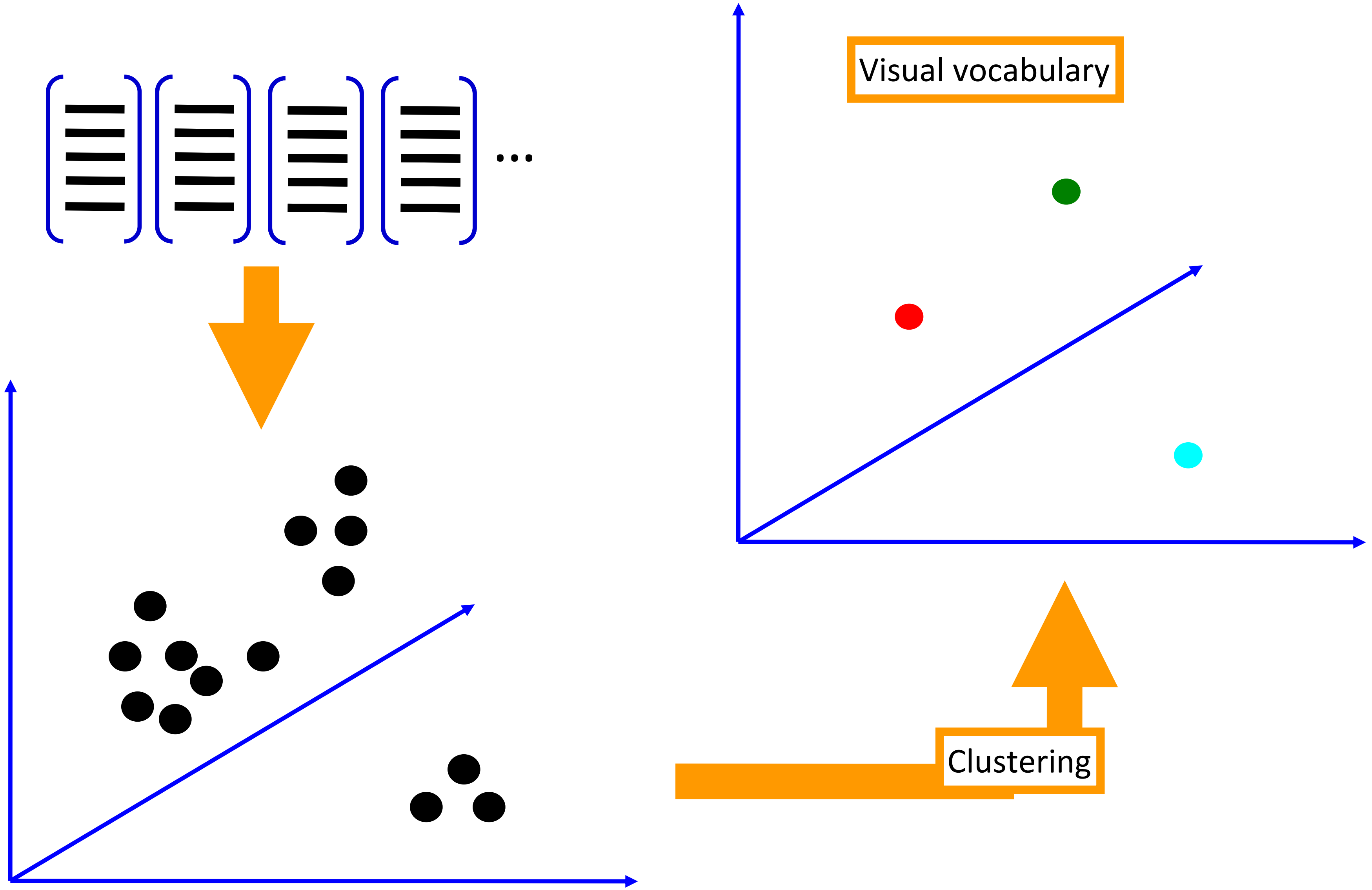
[Sivic & Zisserman, '03]



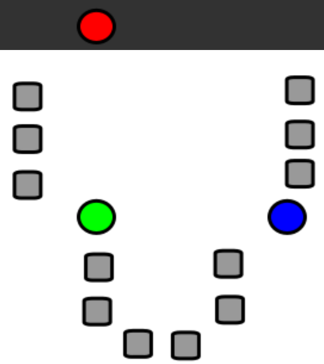
How do we learn the dictionary?



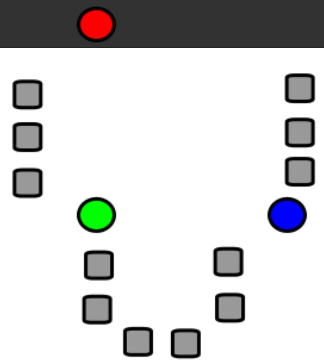

...
Clustering


Visual vocabulary
…
Clustering

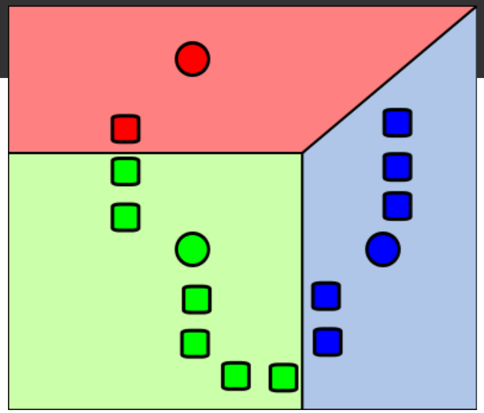
K-means clustering



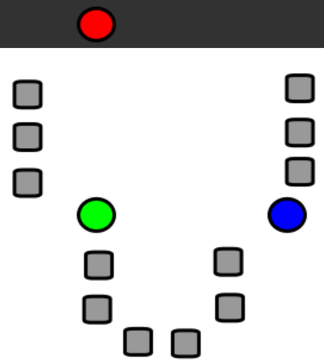
1. Select initial centroids at random



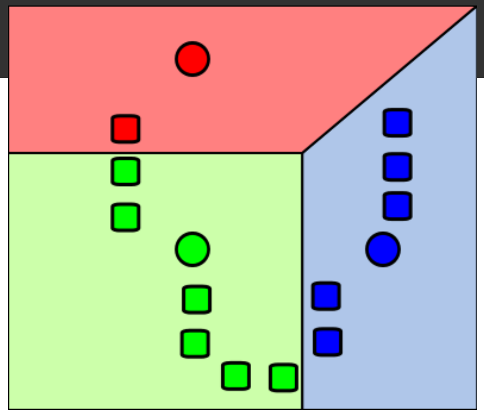
1. Select initial centroids at random



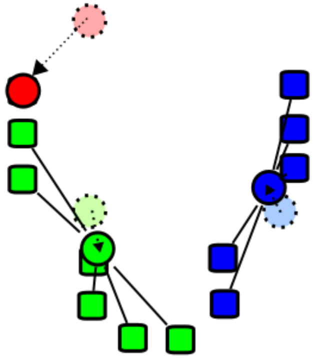
2. Assign each object to the cluster with the nearest centroid.



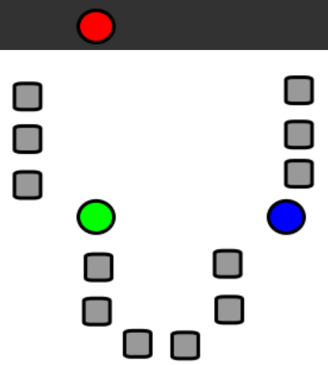
1. Select initial centroids at random



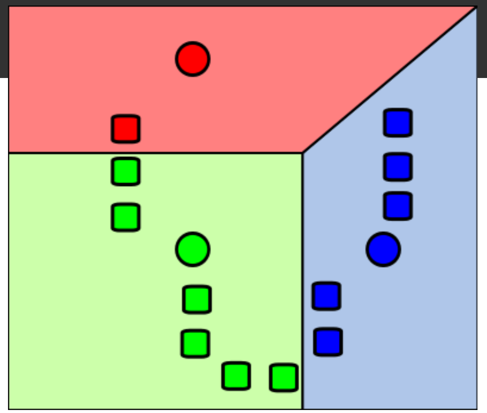
2. Assign each object to the cluster with the nearest centroid.



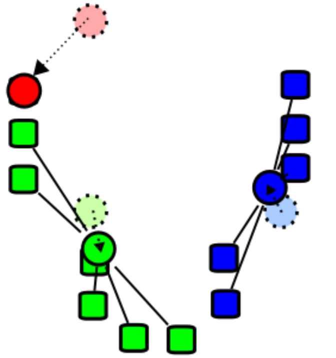
3. Compute each centroid as the mean of the objects assigned to it (go to 2)



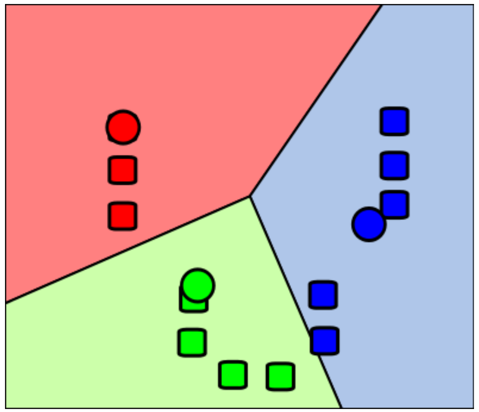
1. Select initial centroids at random



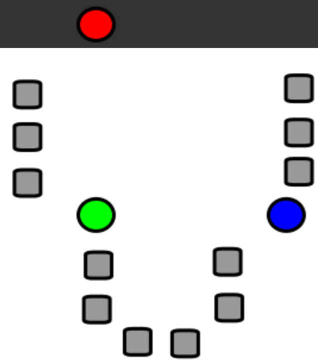
2. Assign each object to the cluster with the nearest centroid.



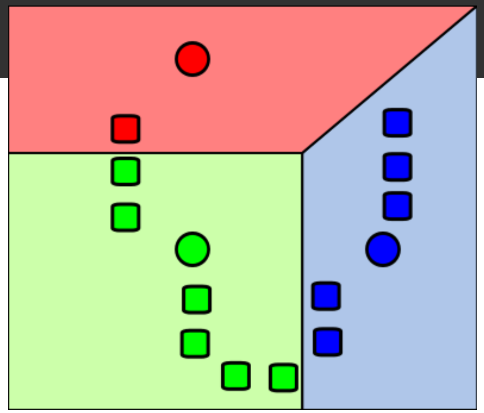
3. Compute each centroid as the mean of the objects assigned to it (go to 2)



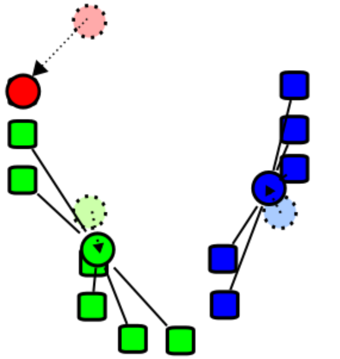
2. Assign each object to the cluster with the nearest centroid.



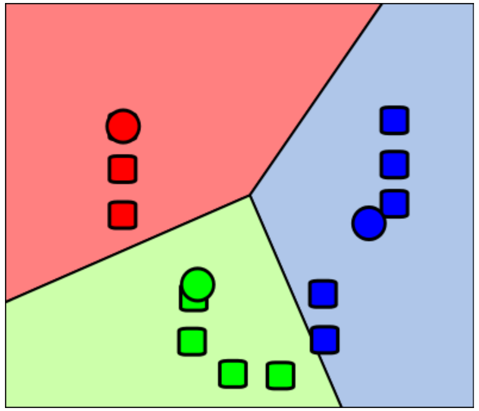
1. Select initial centroids at random



2. Assign each object to the cluster with the nearest centroid.



3. Compute each centroid as the mean of the objects assigned to it (go to 2)



2. Assign each object to the cluster with the nearest centroid.

Repeat previous 2 steps until no change

K-means Clustering

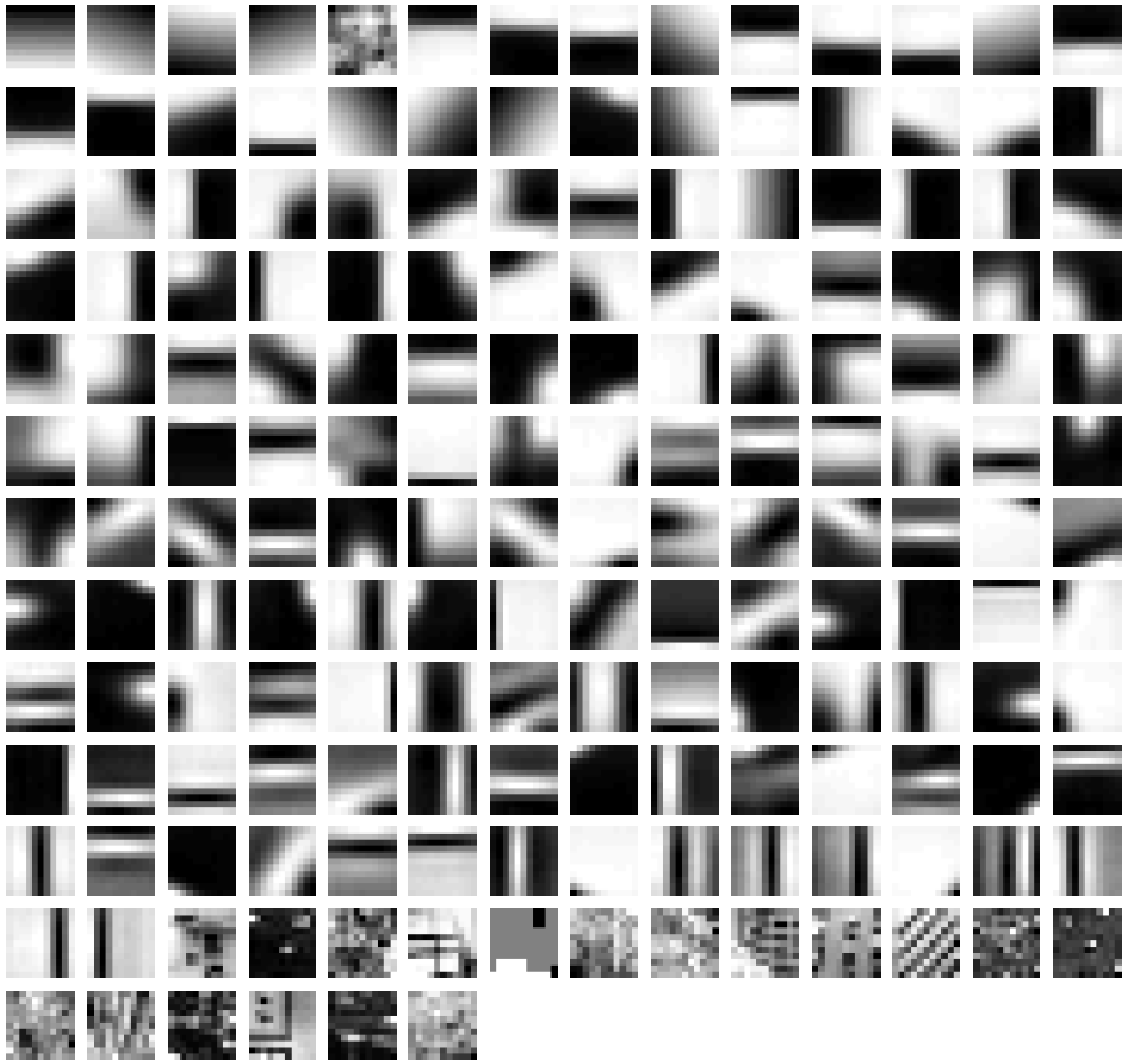
Given k:

1. Select initial centroids at random.
2. Assign each object to the cluster with the nearest centroid.
3. Compute each centroid as the mean of the objects assigned to it.
4. Repeat previous 2 steps until no change.

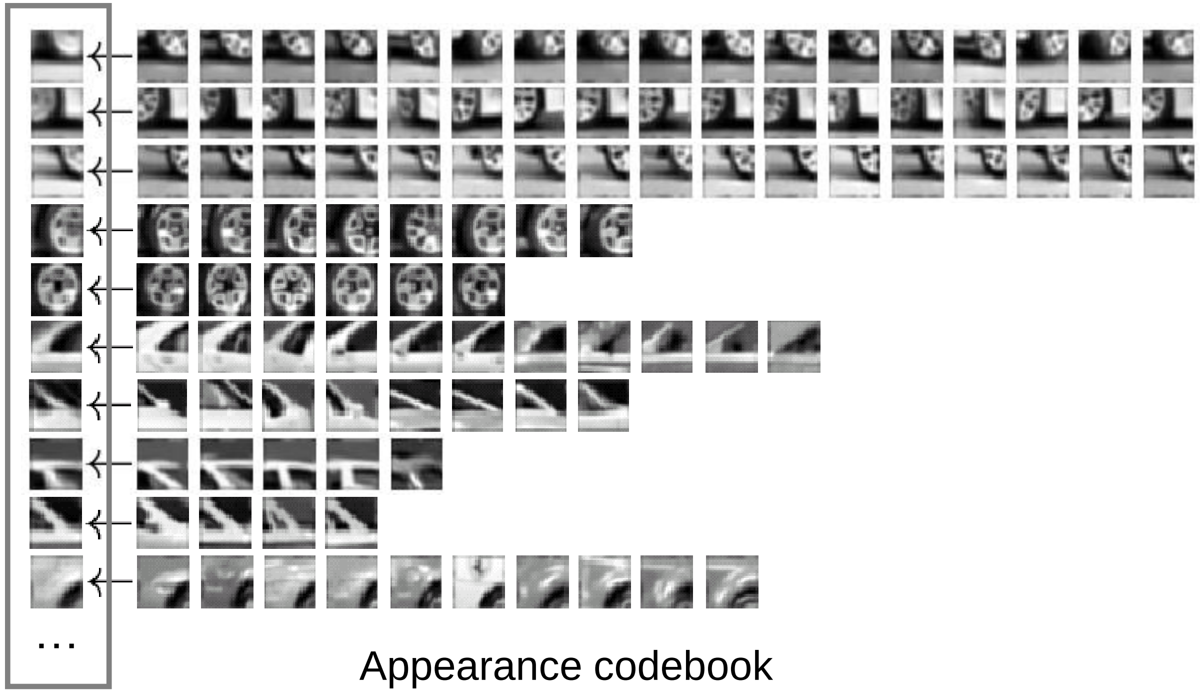
*From what **data** should I learn the dictionary?*

- Dictionary can be learned on separate training set
- Provided the training set is sufficiently representative, the dictionary will be "universal"

Example visual dictionary



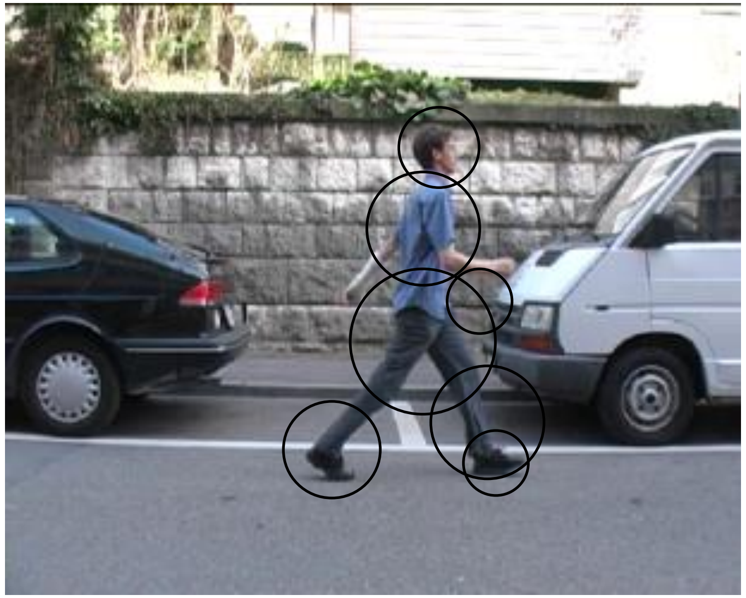
Example dictionary

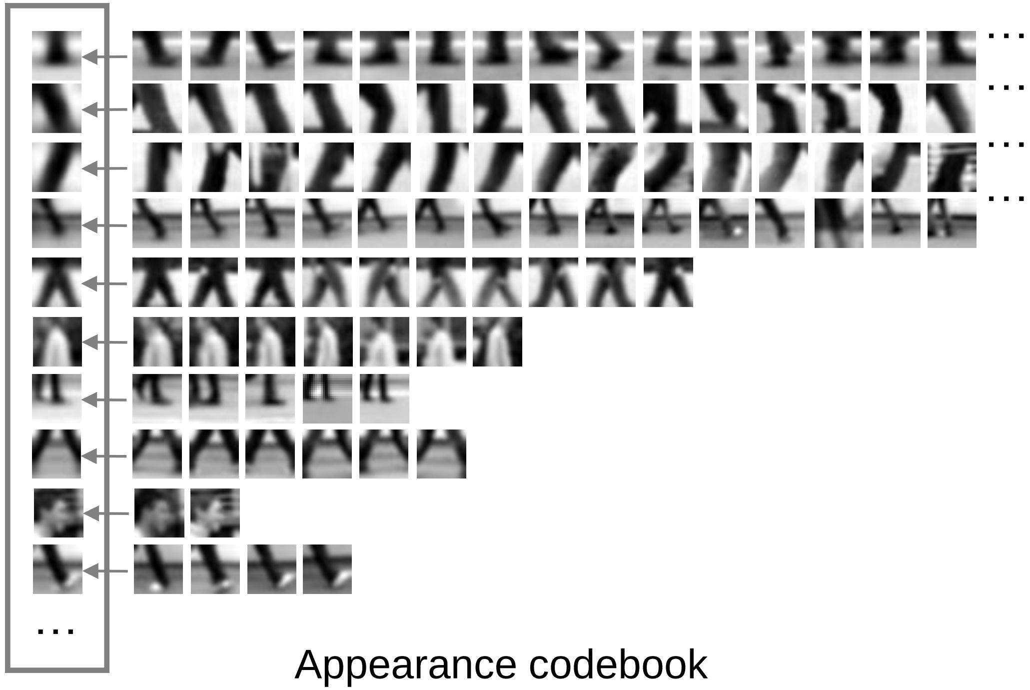


Appearance codebook

Source: B. Leibe

Another dictionary





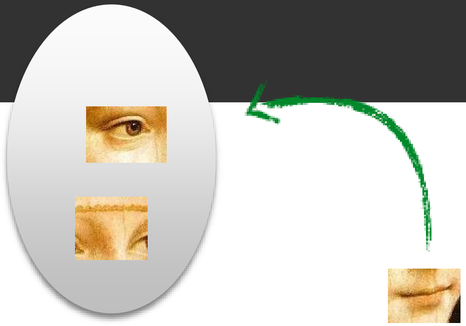
Appearance codebook

Source: B. Leibe

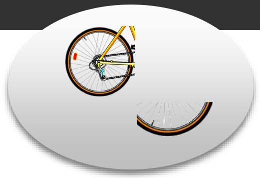
Dictionary Learning:
Learn Visual Words using clustering

Encode:
build Bags-of-Words (BOW) vectors
for each image

Classify:
Train and test data using BOWs

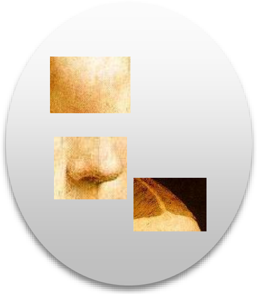


1. Quantization: image features gets associated to a visual word (nearest cluster center)



Encode:

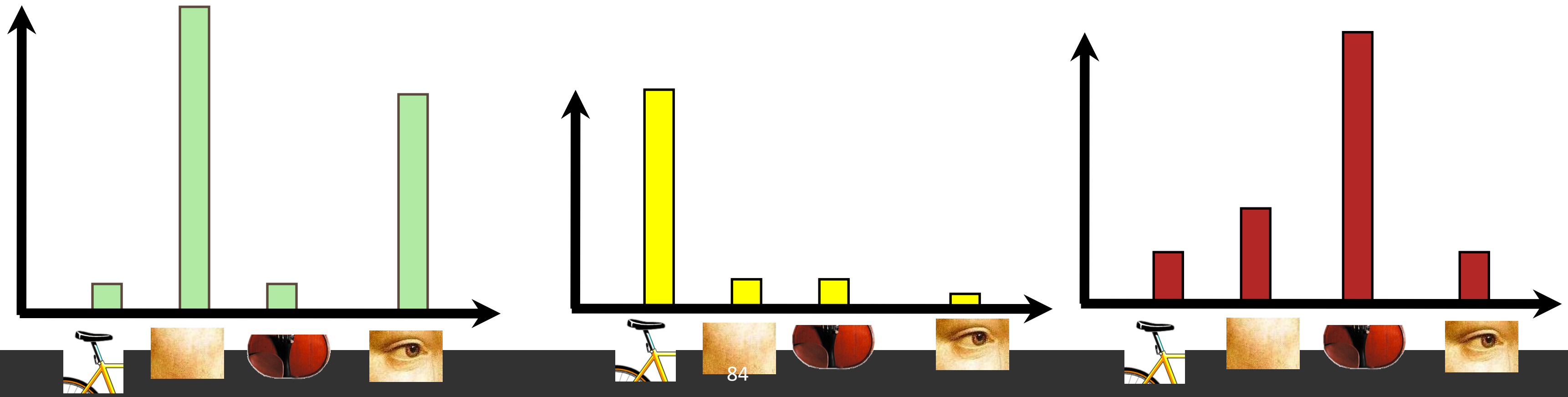
build Bags-of-Words (BOW) vectors for each image

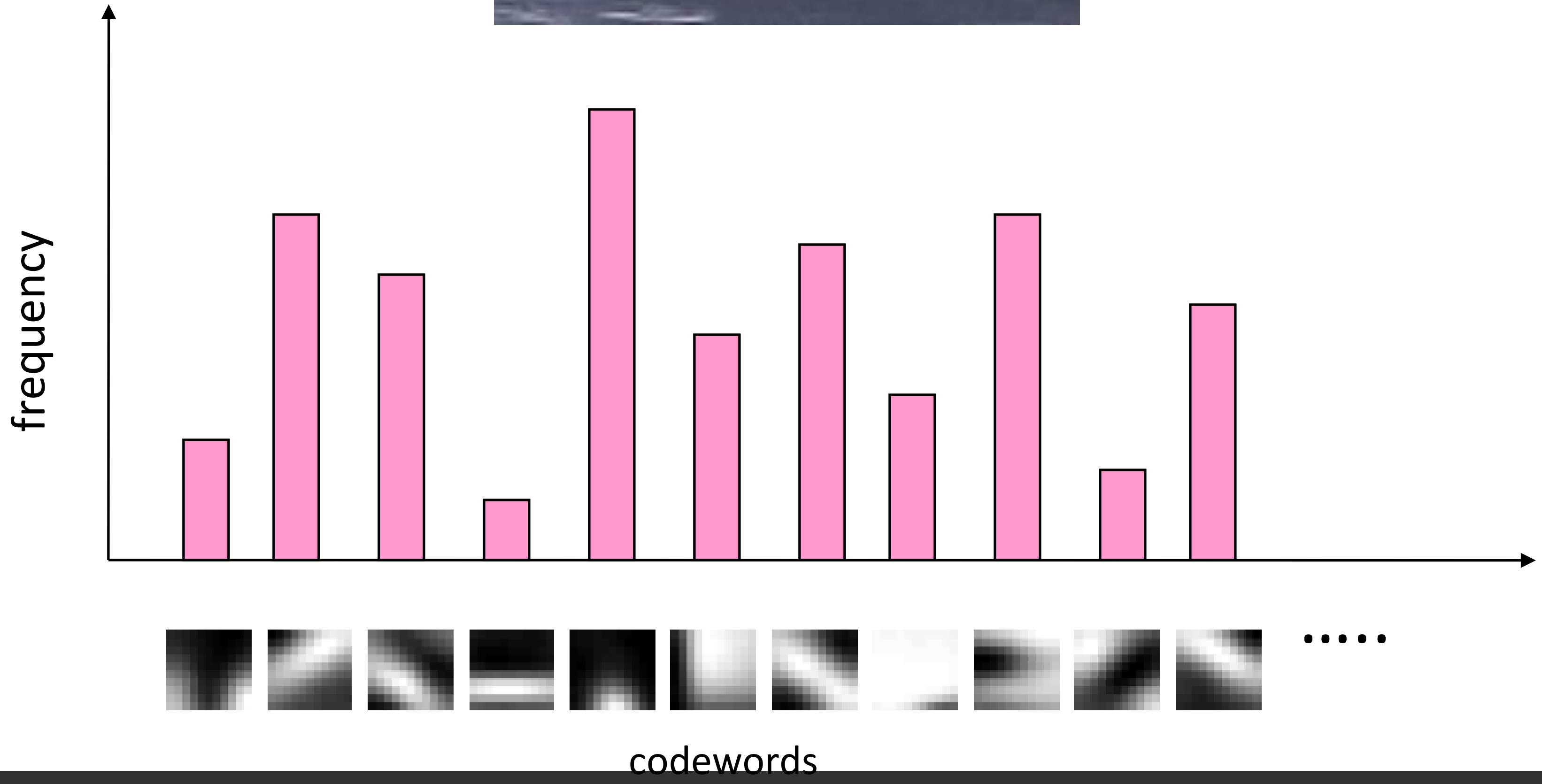


Encode:

build Bags-of-Words (BOW) vectors for each image

2. Histogram: count the number of visual word occurrences

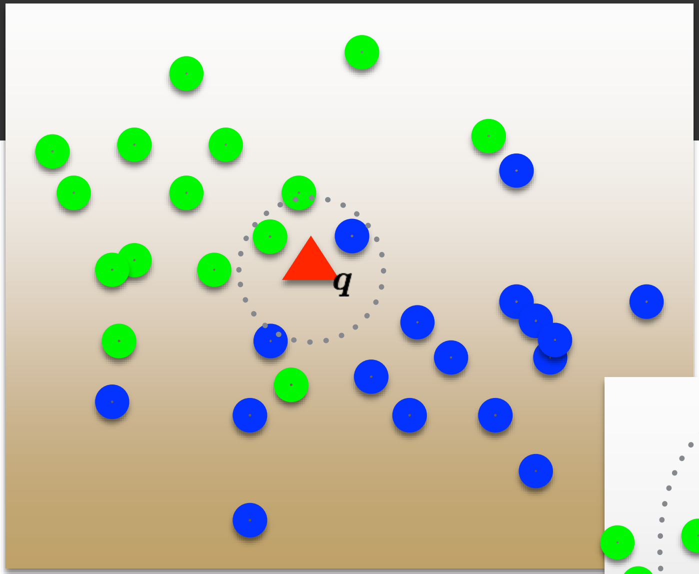



frequency
codewords

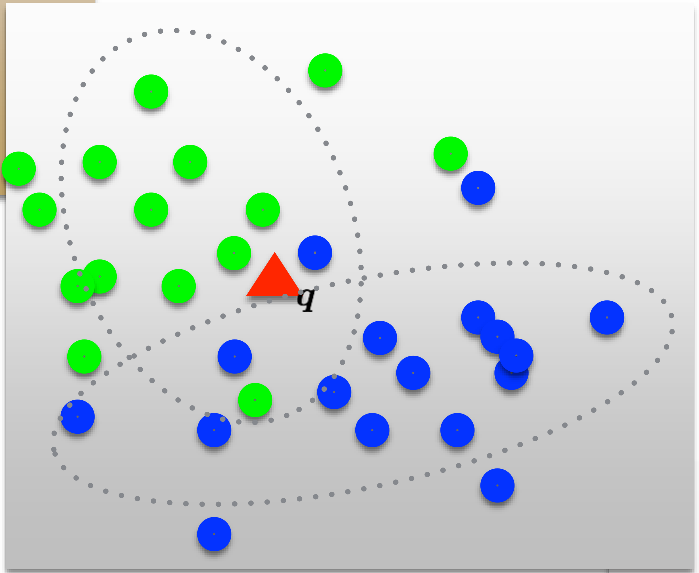
Dictionary Learning:
Learn Visual Words using clustering

Encode:
build Bags-of-Words (BOW) vectors
for each image

Classify:
Train and test data using BOWs

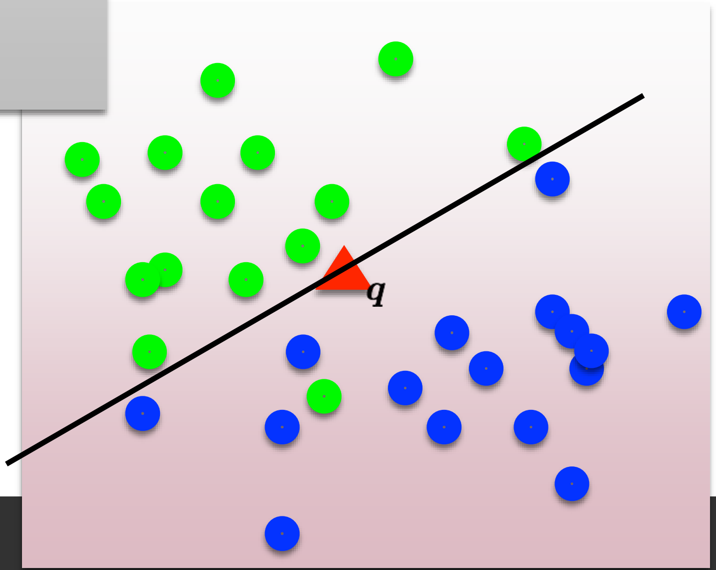


K nearest neighbors



Naïve Bayes

Support Vector Machine



SURF, BRIEF, ORB feature

```python
import cv2
import numpy as np

img = cv2.imread('../data/scenetext01.jpg', cv2.IMREAD_COLOR)

surf = cv2.xfeatures2d.SURF_create(10000)
surf.setExtended(True)
surf.setNOctaves(3)
surf.setNOctaveLayers(10)
surf.setUpright(False)

keyPoints, descriptors = surf.detectAndCompute(img, None)

show_img = cv2.drawKeypoints(img, keyPoints, None, (255, 0, 0),
                             cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imshow('SURF descriptors', show_img)
cv2.waitKey()
cv2.destroyAllWindows()
```

```python
brief = cv2.xfeatures2d.BriefDescriptorExtractor_create(32, True)

keyPoints, descriptors = brief.compute(img, keyPoints)

show_img = cv2.drawKeypoints(img, keyPoints, None, (0, 255, 0),
                             cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imshow('BRIEF descriptors', show_img)
cv2.waitKey()
cv2.destroyAllWindows()

orb = cv2.ORB_create()
orb.setMaxFeatures(200)

keyPoints = orb.detect(img, None)
keyPoints, descriptors = orb.compute(img, keyPoints)

show_img = cv2.drawKeypoints(img, keyPoints, None, (0, 0, 255),
                             cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imshow('ORB descriptors', show_img)
cv2.waitKey()
cv2.destroyAllWindows()
```

Finding correspondences between descriptors

```python
import cv2
import numpy as np

def video_keypoints(matcher, cap=cv2.VideoCapture("../data/traffic.mp4"),
                    detector=cv2.ORB_create(40)):
    cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
    while True:
        status_cap, frame = cap.read()
        frame = cv2.resize(frame, (0, 0), fx=0.5, fy=0.5)
        if not status_cap:
            break
        if (cap.get(cv2.CAP_PROP_POS_FRAMES) - 1) % 40 == 0:
            key_frame = np.copy(frame)
            key_points_1, descriptors_1 = detector.detectAndCompute(frame, None)
        else:
            key_points_2, descriptors_2 = detector.detectAndCompute(frame, None)
            matches = matcher.match(descriptors_2, descriptors_1)
            frame = cv2.drawMatches(frame, key_points_2, key_frame, key_points_1,
                                    matches, None,
                                    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS |
                                    cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
        cv2.imshow('Keypoints matching', frame)
        if cv2.waitKey(300) == 27:
            break

    cv2.destroyAllWindows()
```

```python
bf_matcher = cv2.BFMatcher_create(cv2.NORM_HAMMING2, True)
video_keypoints(bf_matcher)

flann_kd_matcher = cv2.FlannBasedMatcher()
video_keypoints(flann_kd_matcher, detector=cv2.xfeatures2d.SURF_create(20000))

FLANN_INDEX_LSH = 6
index_params = dict(algorithm=FLANN_INDEX_LSH, table_number=20, key_size=15, multi_probe_level=2)
search_params = dict(checks=10)

flann_kd_matcher = cv2.FlannBasedMatcher(index_params, search_params)
video_keypoints(flann_kd_matcher)

FLANN_INDEX_COMPOSITE = 3
index_params = dict(algorithm=FLANN_INDEX_COMPOSITE, trees=16)
search_params = dict(checks=10)

flann_kd_matcher = cv2.FlannBasedMatcher(index_params, search_params)
video_keypoints(flann_kd_matcher, detector=cv2.xfeatures2d.SURF_create(20000))
```

Feature matching with consistency check and ratio test

```python
import cv2
import matplotlib.pyplot as plt
import numpy as np
import math

img0 = cv2.imread('../data/Lena.png')
M = np.array([[math.cos(np.pi/12), -math.sin(np.pi/12), 0],
              [math.sin(np.pi/12), math.cos(np.pi/12), 0],
              [0,0,1]])
Moff = np.eye(3)
Moff[0,2] = -img0.shape[1]/2
Moff[1,2] = -img0.shape[0]/2
print(np.linalg.inv(Moff)@M@Moff)
img1 = cv2.warpPerspective(img0, np.linalg.inv(Moff)@M@Moff,
                           (img0.shape[1], img0.shape[0]), borderMode=cv2.BORDER_REPLICATE)
cv2.imwrite('../data/Lena_rotated.png', img1)

img0 = cv2.imread('../data/Lena.png', cv2.IMREAD_GRAYSCALE)
img1 = cv2.imread('../data/Lena_rotated.png', cv2.IMREAD_GRAYSCALE)

detector = cv2.ORB_create(100)
kps0, fea0 = detector.detectAndCompute(img0, None)
kps1, fea1 = detector.detectAndCompute(img1, None)
```

```python
matcher = cv2.BFMatcher_create(cv2.NORM_HAMMING, False)
matches01 = matcher.knnMatch(fea0, fea1, k=2)
matches10 = matcher.knnMatch(fea1, fea0, k=2)

def ratio_test(matches, ratio_thr):
    good_matches = []
    for m in matches:
        ratio = m[0].distance / m[1].distance
        if ratio < ratio_thr:
            good_matches.append(m[0])
    return good_matches

RATIO_THR = 0.7  # Lower values mean more aggressive filtering.
good_matches01 = ratio_test(matches01, RATIO_THR)
good_matches10 = ratio_test(matches10, RATIO_THR)

good_matches10_ = {(m.trainIdx, m.queryIdx) for m in good_matches10}
final_matches = [m for m in good_matches01 if (m.queryIdx, m.trainIdx) in good_matches10_]

dbg_img = cv2.drawMatches(img0, kps0, img1, kps1, final_matches, None)
plt.figure()
plt.imshow(dbg_img[:,:,[2,1,0]])
plt.tight_layout()
plt.show()
```

Model based fitting using RANSAC

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

img0 = cv2.imread('../data/Lena.png', cv2.IMREAD_GRAYSCALE)
img1 = cv2.imread('../data/Lena_rotated.png', cv2.IMREAD_GRAYSCALE)

detector = cv2.ORB_create(100)
kps0, fea0 = detector.detectAndCompute(img0, None)
kps1, fea1 = detector.detectAndCompute(img1, None)
matcher = cv2.BFMatcher_create(cv2.NORM_HAMMING, False)
matches = matcher.match(fea0, fea1)

pts0 = np.float32([kps0[m.queryIdx].pt for m in matches]).reshape(-1,2)
pts1 = np.float32([kps1[m.trainIdx].pt for m in matches]).reshape(-1,2)
H, mask = cv2.findHomography(pts0, pts1, cv2.RANSAC, 3.0)

plt.figure()
plt.subplot(211)
plt.axis('off')
plt.title('all matches')
dbg_img = cv2.drawMatches(img0, kps0, img1, kps1, matches, None)
plt.imshow(dbg_img[:,:,[2,1,0]])
plt.subplot(212)
plt.axis('off')
plt.title('filtered matches')
dbg_img = cv2.drawMatches(img0, kps0, img1, kps1, [m for i,m in enumerate(matches) if mask[i]], None)
plt.imshow(dbg_img[:,:,[2,1,0]])
plt.tight_layout()
plt.show()
```

BoW model for global image descriptor

```python
import cv2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
matplotlib.rc('font', size=18)

img0 = cv2.imread('../data/people.jpg', cv2.IMREAD_GRAYSCALE)
img1 = cv2.imread('../data/face.jpeg', cv2.IMREAD_GRAYSCALE)

detector = cv2.ORB_create(500)
_, fea0 = detector.detectAndCompute(img0, None)
_, fea1 = detector.detectAndCompute(img1, None)
descr_type = fea0.dtype

bow_trainer = cv2.BOWKMeansTrainer(50)
bow_trainer.add(np.float32(fea0))
bow_trainer.add(np.float32(fea1))
vocab = bow_trainer.cluster().astype(descr_type)

bow_descr = cv2.BOWImgDescriptorExtractor(detector, cv2.BFMatcher(cv2.NORM_HAMMING))
bow_descr.setVocabulary(vocab)

img = cv2.imread('../data/Lena.png', cv2.IMREAD_GRAYSCALE)
kps = detector.detect(img, None)
descr = bow_descr.compute(img, kps)

plt.figure(figsize=(10,8))
plt.title('image BoW descriptor')
plt.bar(np.arange(len(descr[0])), descr[0])
plt.xlabel('vocabulary element')
plt.ylabel('frequency')
plt.tight_layout()
plt.show()
```

실습문제

- (1) 두 장의 영상에 대해서 SURF detector를 이용해서 feature를 추출하고, SURF, ORB, BRIEF descriptor 를 추출해서 matching을 한 후에 대응관계를 출력하시오. RANSAC을 통해서 homography를 구하고 outlier를 제거하시오.

- (2) 인터넷에서 다양한 영상(2개 이상의 클래스)을 수집해서 BoW를 학습하고 이에 대한 히스토그램을 그려보시오.