Visual Navigation for Flying Robots

Simultaneous Localization and Mapping (SLAM)

Dr. Jürgen Sturm

Organization: Exam Dates

- Registration deadline: June 30
- Course ends: July 19
- Examination dates: August 9+14 (Thu+Tue)
  - Oral team exam
  - Sign up for a time slot starting from now
  - List placed on blackboard in front of our secretary

VISNAV Oral Team Exam

Date and Time	Student Name	Student Name	Student Name
Tue, Aug. 9, 10am			
Tue, Aug. 9, 11am			
Tue, Aug. 9, 2pm			
Tue, Aug. 9, 3pm			
Tue, Aug. 9, 4pm			
Thu, Aug. 14, 10am			
Thu, Aug. 14, 11am			
Thu, Aug. 14, 2pm			
Thu, Aug. 14, 3pm			
Thu, Aug. 14, 4pm			

The SLAM Problem

SLAM is the process by which a robot **builds a map** of the environment and, at the same time, uses the map to **compute its location**

- Localization: inferring location given a map
- Mapping: inferring a map given a location

The SLAM Problem

Given:

- The robot's controls $\mathbf{u}_{1:t} =< \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_t >$
- (Relative) observations $\mathbf{z}_{1:t} =< \mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t >$

Wanted:

- Map of features $\mathbf{m} =< \mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_k >$
- Trajectory of the robot $\mathbf{x}_{1:t} =< \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t >$

SLAM Applications

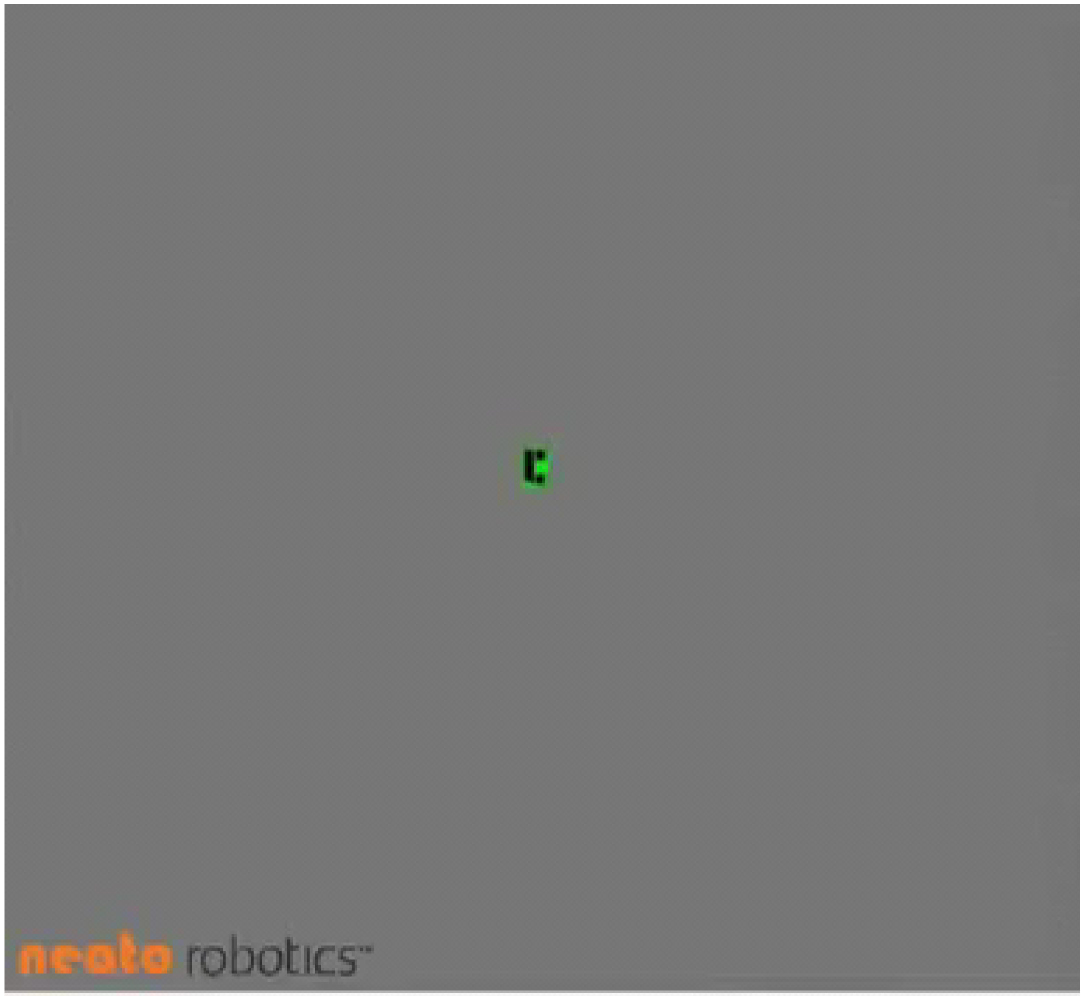
SLAM is central to a range of indoor, outdoor, in-air and underwater applications for both unmanned and autonomous vehicles.

Examples

- At home: vacuum cleaner, lawn mower
- Air: inspection, transportation, surveillance
- Underwater: reef/environmental monitoring
- Underground: search and rescue
- Space: terrain mapping, navigation

SLAM with Ceiling Camera (Samsung Hauzen RE70V, 2008)

SLAM with Laser + Line camera (Neato XV 11, 2010)



Localization, Path planning, Coverage (Neato XV11, $300)

SLAM vs. SfM

- In Robotics: Simultaneous Localization and Mapping (SLAM)
  - Laser scanner, ultrasound, monocular/stereo camera
  - Typically in combination with an odometry sensor
  - Typically pre-calibrated sensors
- In Computer Vision: Structure from Motion (SfM), sometimes: Structure and Motion
  - Monocular/stereo camera
  - Sometimes uncalibrated sensors (e.g., Flick images)

Agenda for Today

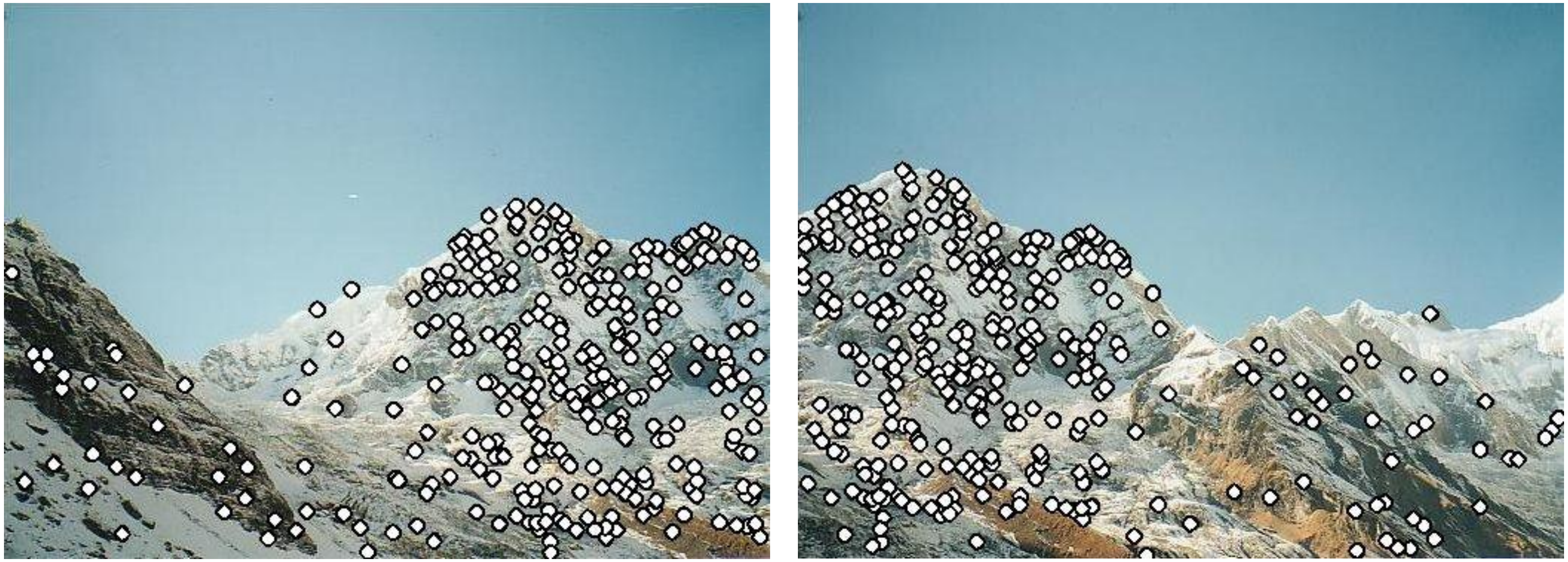
- **This week:** focus on monocular vision
  - Feature detection, descriptors and matching
  - Epipolar geometry
  - Robust estimation (RANSAC)
  - Examples (PTAM, Photo Tourism)
- **Next week:** focus on optimization (bundle adjustment), stereo cameras, Kinect
- **In two weeks:** map representations, mapping and (dense) 3D reconstruction

How Do We Build a Panorama Map?

- We need to match (align) images
- Global methods sensitive to occlusion, lighting, parallax effects
- How would you do it by eye?

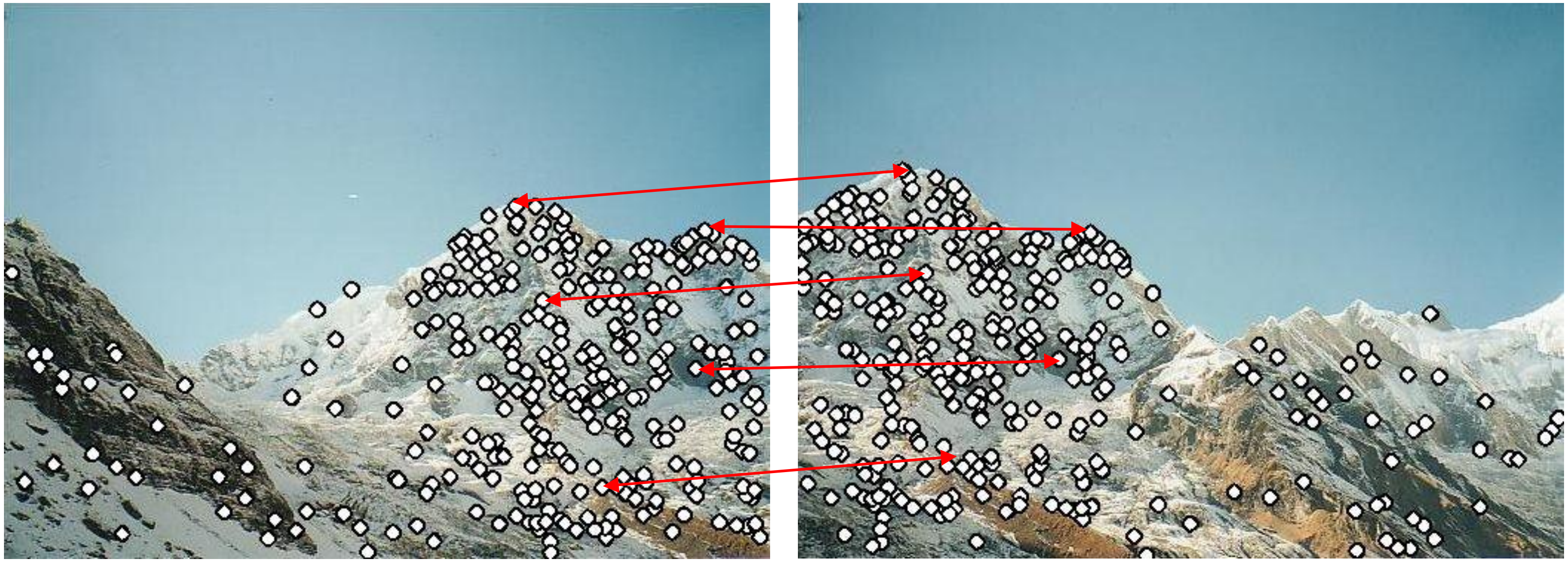
Matching with Features

- Detect features in both images



Matching with Features

- Detect features in both images
- Find corresponding pairs

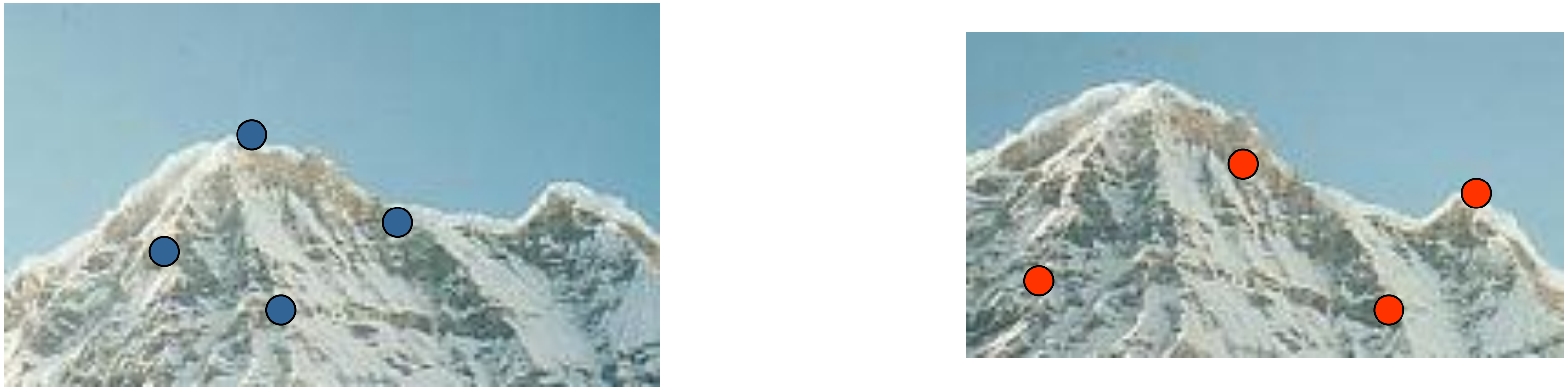


Matching with Features

- Detect features in both images
- Find corresponding pairs
- Use these pairs to align images

Matching with Features

- Problem 1:
  We need to detect the **same** point **independently** in both images

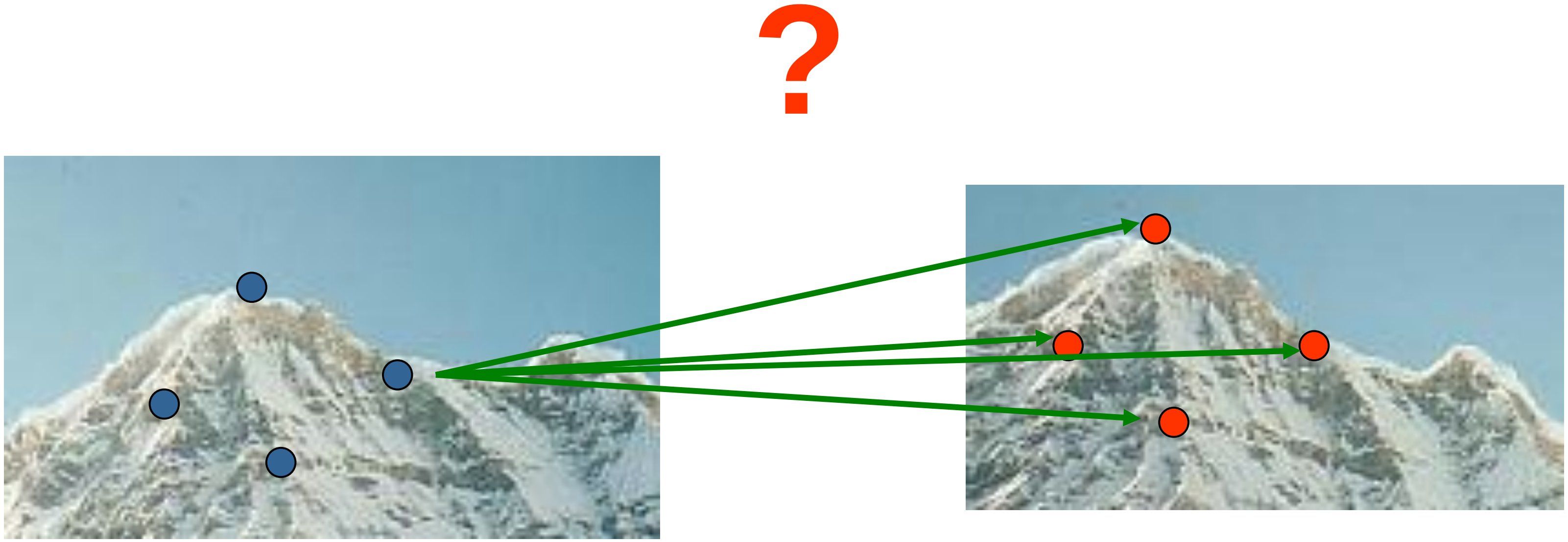


no chance to match!

→ We need a reliable detector

Matching with Features

- Problem 2:
  For each point correctly recognize the corresponding one



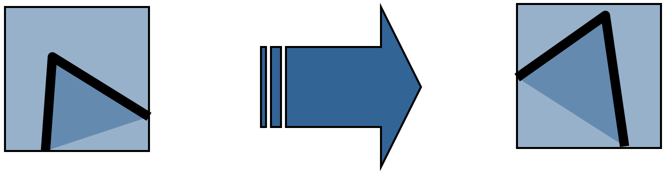
→ We need a reliable and distinctive descriptor

Ideal Feature Detector

- Always finds the same point on an object, regardless of changes to the image
- Insensitive (invariant) to changes in:
  - Scale
  - Lightning
  - Perspective imaging
  - Partial occlusion

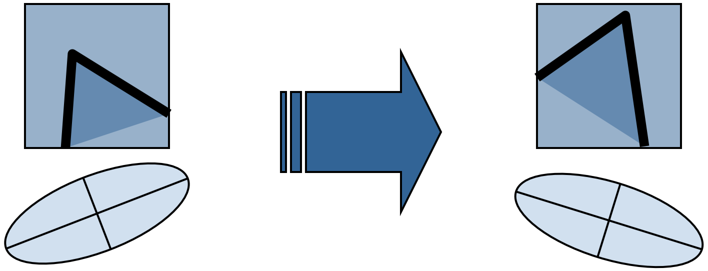
Harris Detector

- Rotation invariance?



Harris Detector

- Rotation invariance?

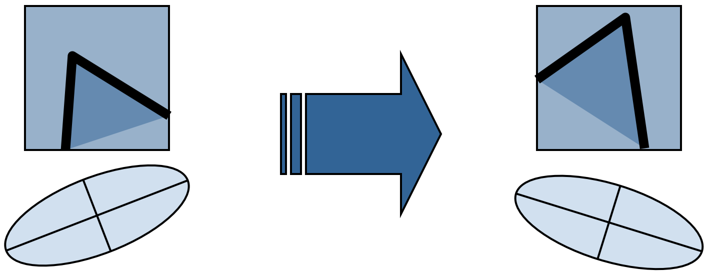


- Remember from last week

$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix} \qquad R = \lambda_1 \lambda_2 - \kappa \left(\lambda_1 + \lambda_2\right)^2$$

Harris Detector

- Rotation invariance



- Remember from last week

$$A = \begin{pmatrix} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{pmatrix} \quad R = \lambda_1 \lambda_2 - \kappa \left(\lambda_1 + \lambda_2\right)^2$$

- Ellipse rotates but its shape (i.e. eigenvalues) remains the same

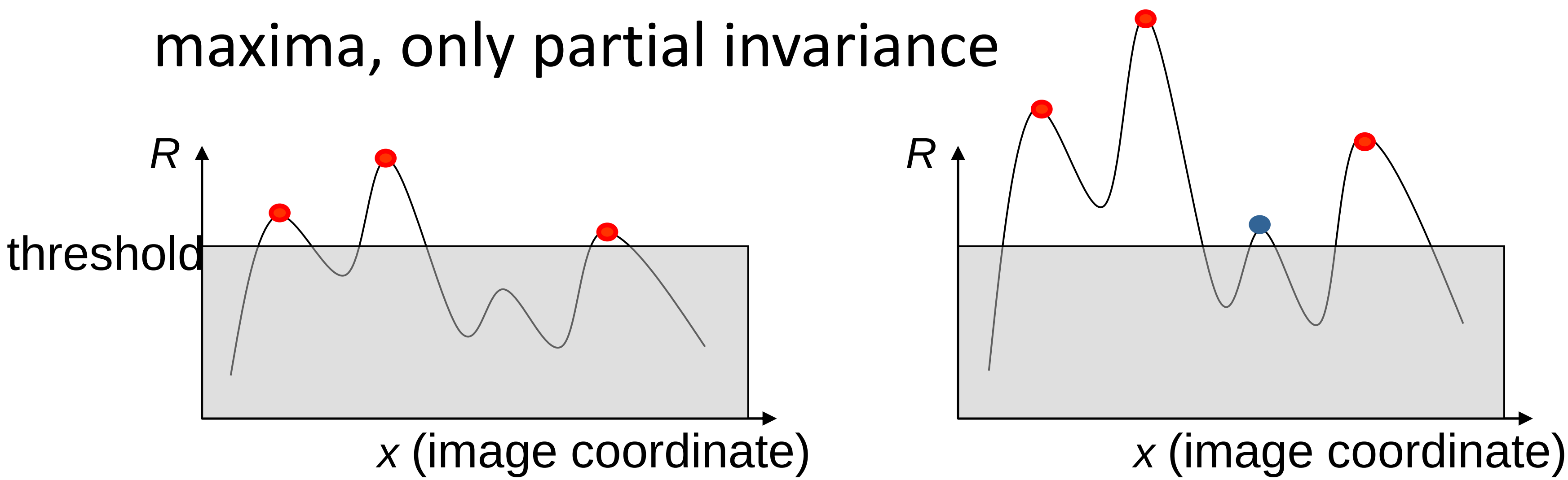
→ Corner response R is invariant to rotation

Harris Detector

- Invariance to intensity change?

Harris Detector

- Partial invariance to additive and multiplicative intensity changes
  - Only derivatives are used → invariance to intensity shift $I \rightarrow I + b$
  - Intensity scale $I \rightarrow aI$:
    Because of fixed intensity threshold on local maxima, only partial invariance

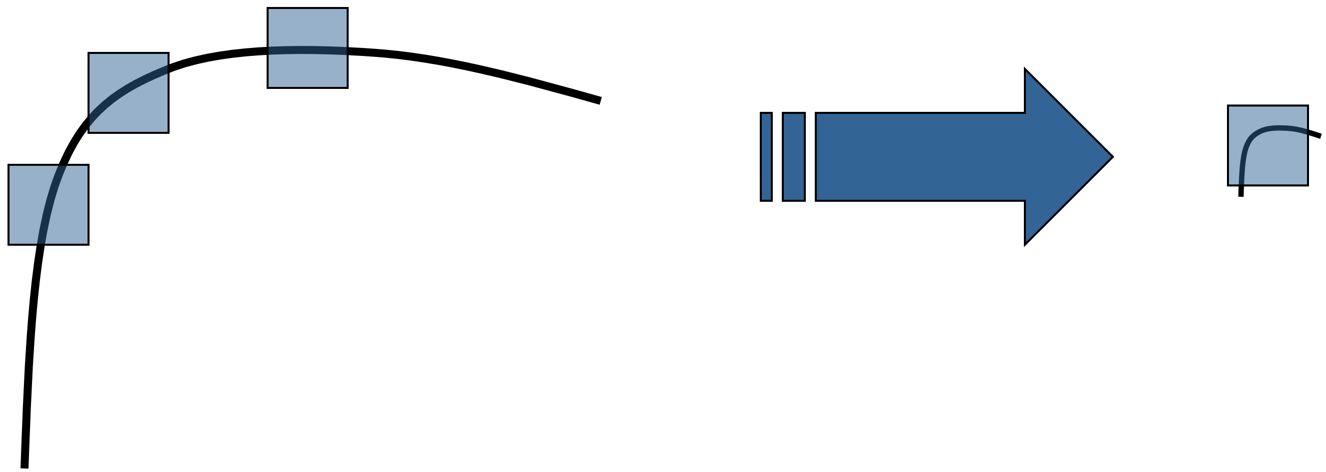


Harris Detector

- Invariant to scaling?

Harris Detector

- Not invariant to image scale

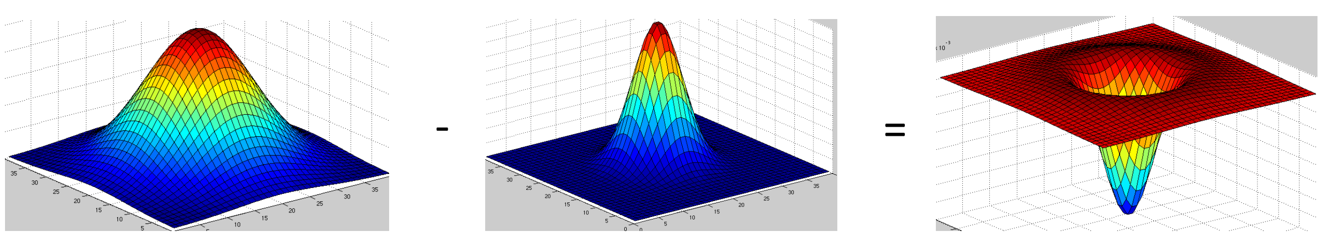


All points classified as edge

Point classified as corner

Difference Of Gaussians (DoG)

- Alternative corner detector that is additionally invariant to scale change
- Approach:
  - Run linear filter (diff. of two Gaussians, $\sigma_1 = 2\sigma_2$)
  - Do this at different scales
  - Search for a maximum both in space and scale



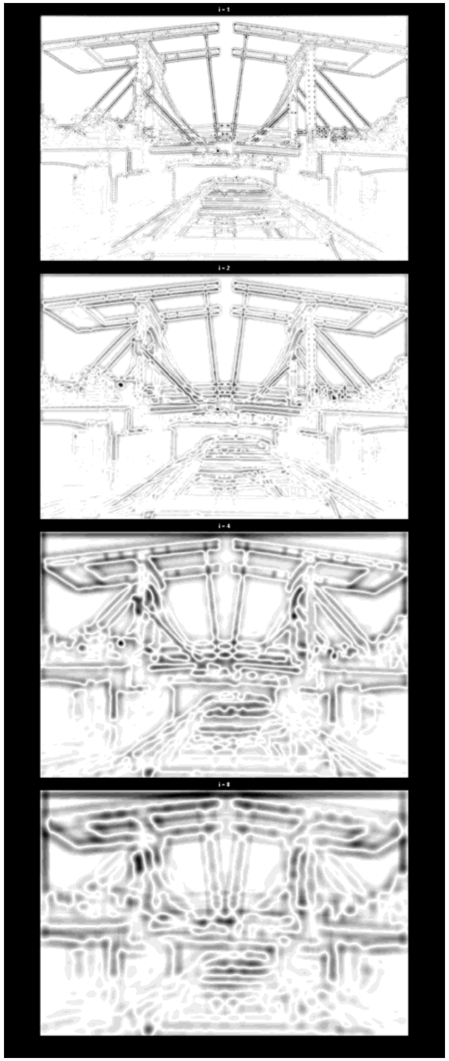
Example: Difference of Gaussians

$\sigma = 1$

$\sigma = 2$

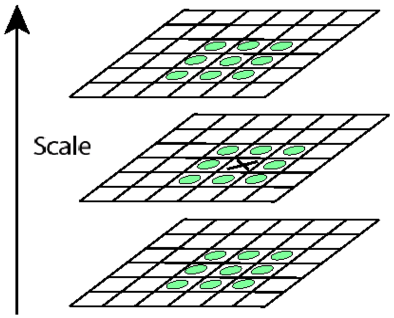
$\sigma = 4$

$\sigma = 8$

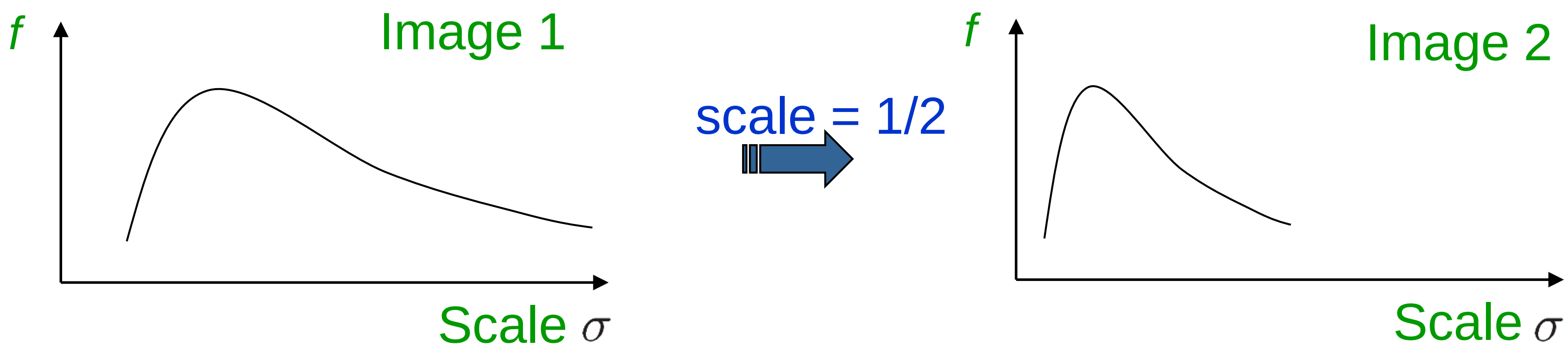


SIFT Detector

- Search for local maximum in space and scale



- Corner detections are invariant to scale change

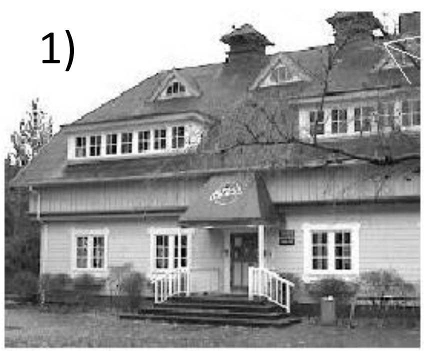


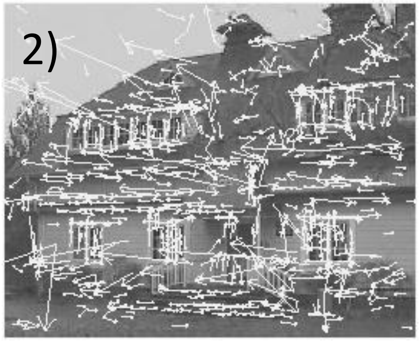
SIFT Detector

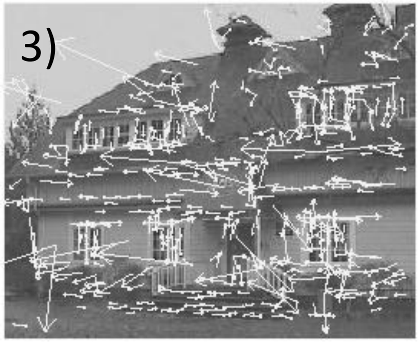
1. Detect maxima in scale-space
2. Non-maximum suppression
3. Eliminate edge points (check ratio of eigenvalues)
4. For each maximum, fit quadratic function and compute center at sub-pixel accuracy

Example

1. Input image 233x189 pixel
2. 832 candidates DoG minima/maxima (visualization indicate scale, orient., location)
3. 536 keypoints remain after thresholding on minimum contrast and principal curvature

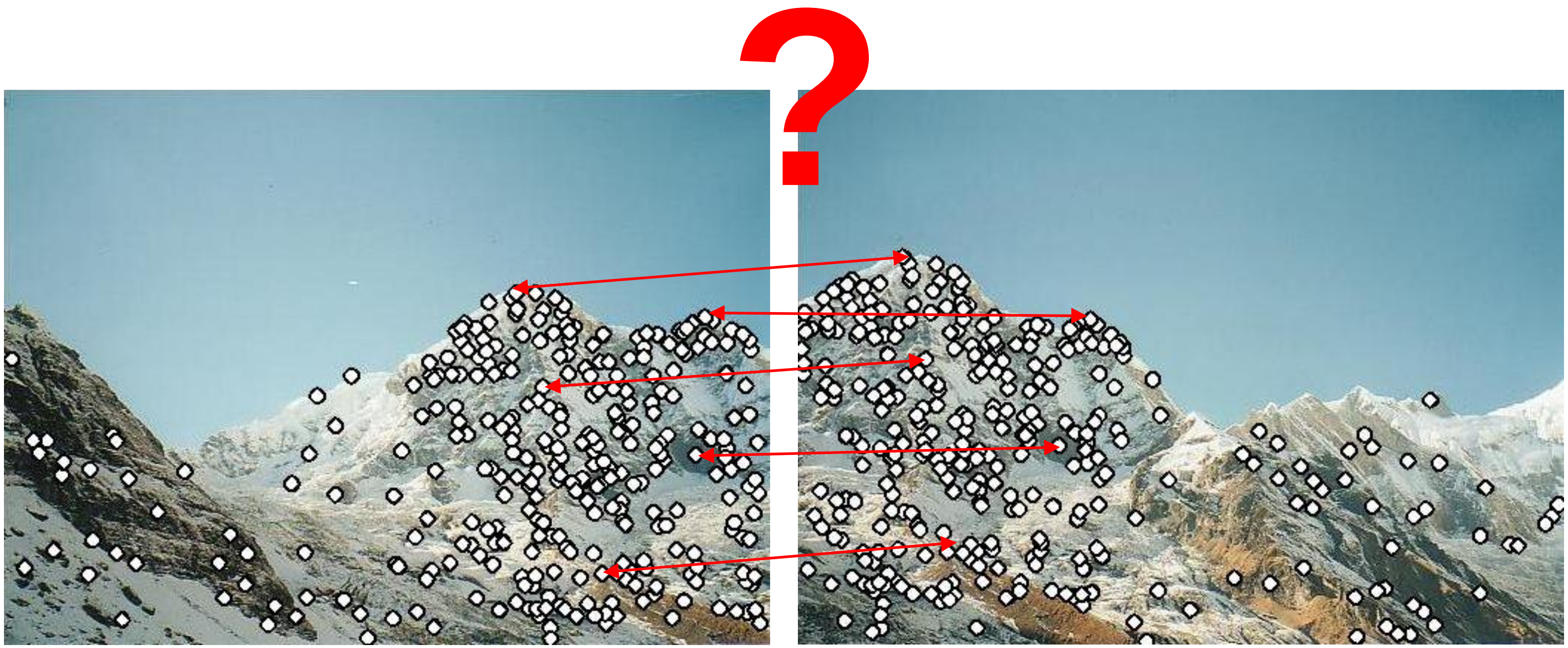






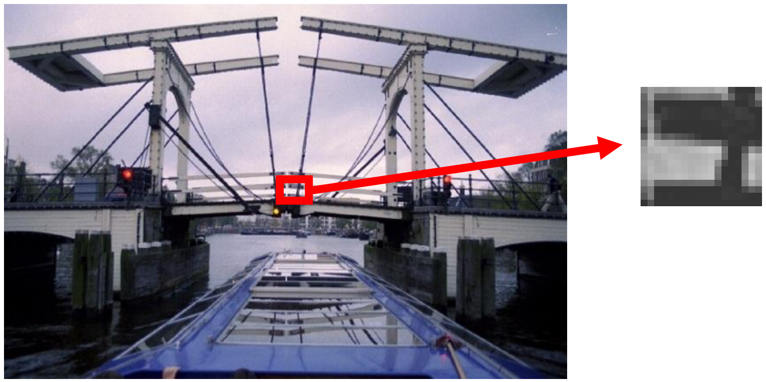
Feature Matching

- Now, we know how to find **repeatable** corners
- Next question: How can we match them?

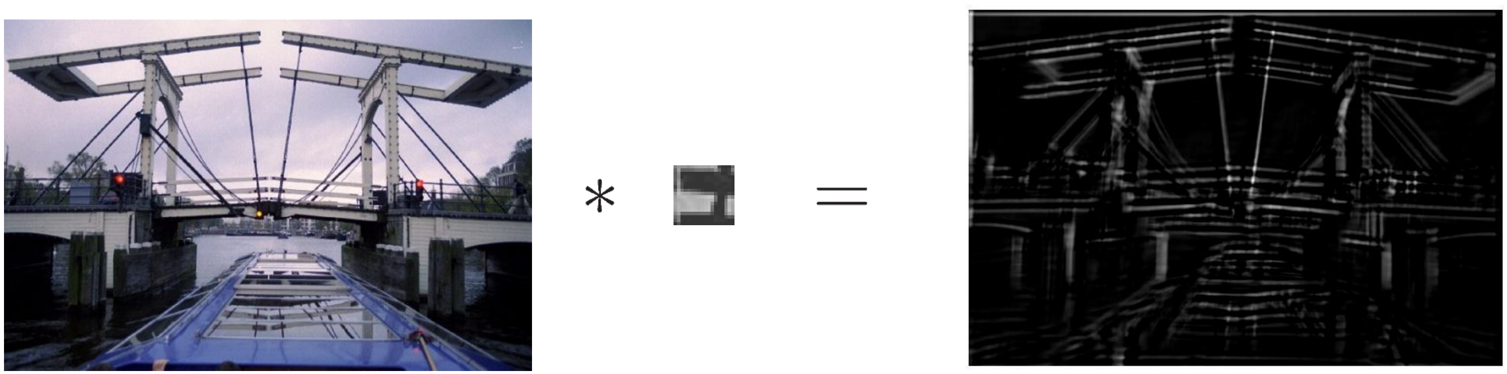


Template Convolution

- Extract a small as a template



- Convolve image with this template



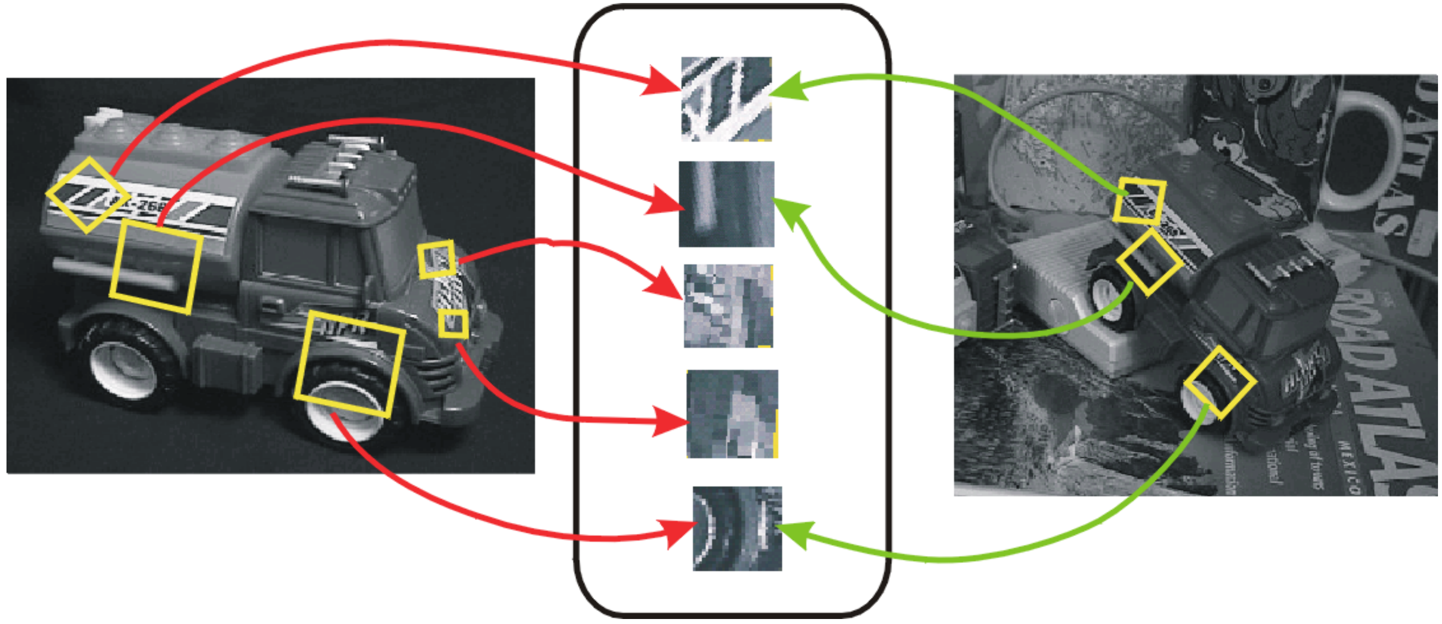
Template Convolution

Invariances

- Scaling: No
- Rotation: No (maybe rotate template?)
- Illumination: No (use bias/gain model?)
- Perspective projection: Not really

Scale Invariant Feature Transform (SIFT)

- Lowe, 2004: Transform patches into a canonical form that is invariant to translation, rotation, scale, and other imaging parameters



SIFT Features

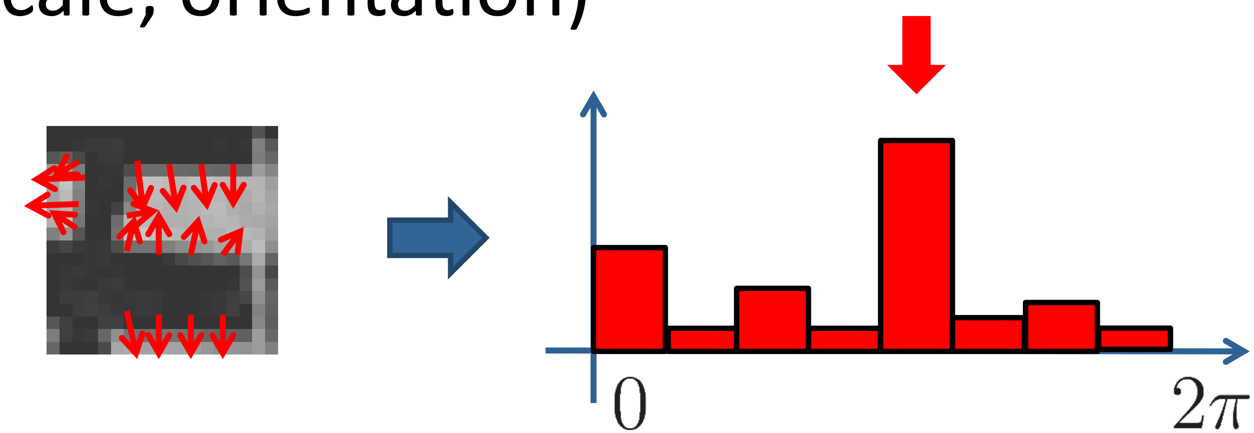
Scale Invariant Feature Transform (SIFT)

Approach

1. Find SIFT corners (position + scale)
2. Find dominant orientation and de-rotate patch
3. Extract SIFT descriptor (histograms over gradient directions)

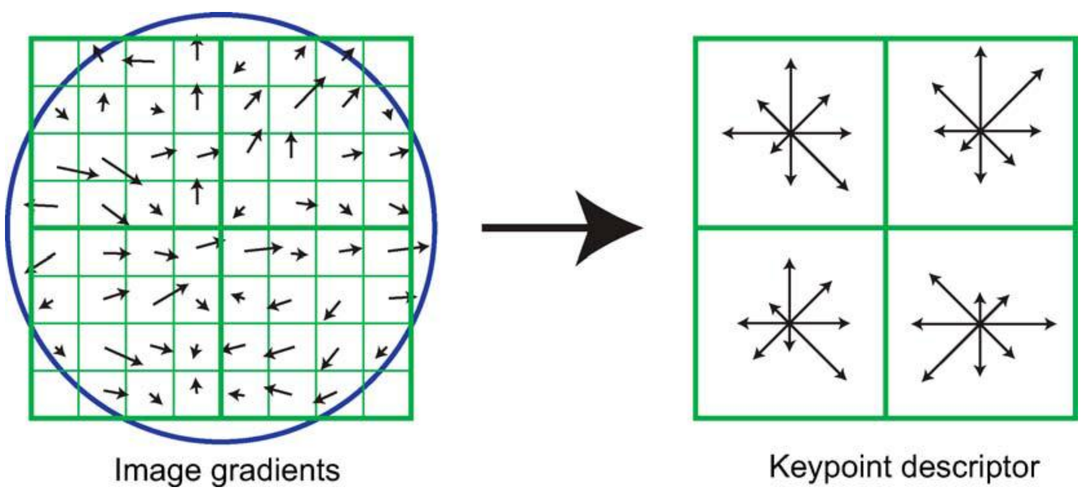
Select Dominant Orientation

- Create a histogram of local gradient directions computed at selected scale (36 bins)
- Assign canonical orientation at peak of smoothed histogram
- Each key now specifies stable 2D coordinates (x, y, scale, orientation)



SIFT Descriptor

- Compute image gradients over 16x16 window (green), weight with Gaussian kernel (blue)
- Create 4x4 arrays of orientation histograms, each consisting of 8 bins
- In total, SIFT descriptor has 128 dimensions



Feature Matching

Given features in I_1, how to find best match in I_2?

- Define distance function that compares two features
- Test all the features in I_2, find the one with the minimal distance

Feature Distance

How to define the difference between features?

- Simple approach is Euclidean distance (or SSD)

$$\mathrm{d}(\mathbf{d}_1, \mathbf{d}_2) = \|\mathbf{d}_1 - \mathbf{d}_2\|$$

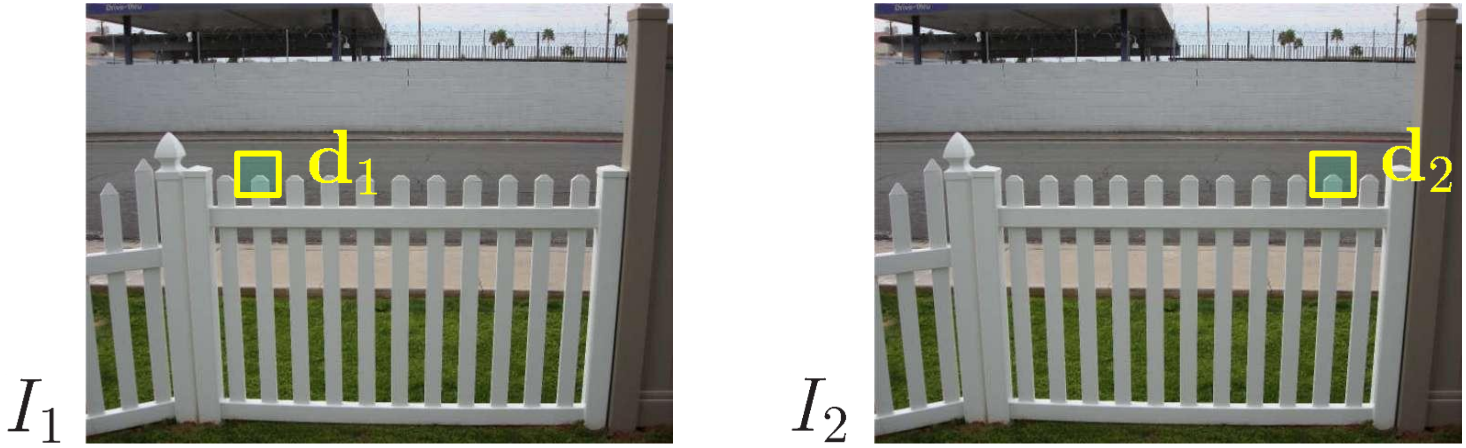
Feature Distance

How to define the difference between features?

- Simple approach is Euclidean distance (or SSD)

$$\mathrm{d}(\mathbf{d}_1, \mathbf{d}_2) = \|\mathbf{d}_1 - \mathbf{d}_2\|$$

- Problem: can give good scores to ambiguous (bad) matches

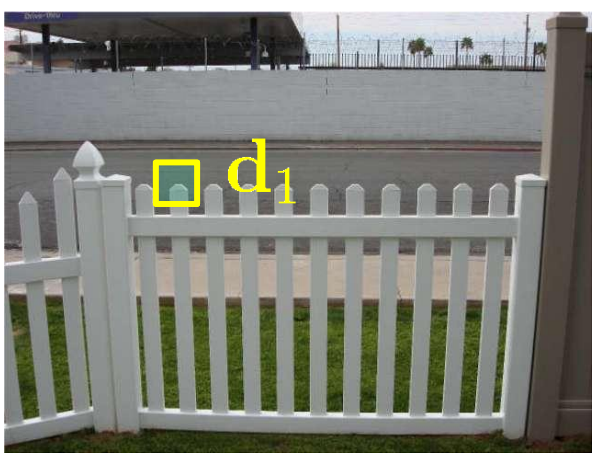


Feature Distance

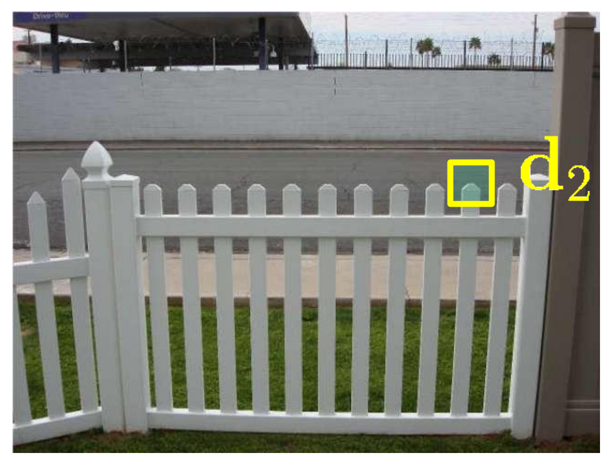
How to define the difference between features?

- Better approach $d(\mathbf{d}_1, \mathbf{d}_2) = \|\mathbf{d}_1 - \mathbf{d}_2\| / \|\mathbf{d}_1 - \mathbf{d}_2'\|$
  with $\mathbf{d}_2$ best matching feature from I_2
  $\mathbf{d}_2'$ second best matching feature from I_2
- Gives small values for ambiguous matches

I_1



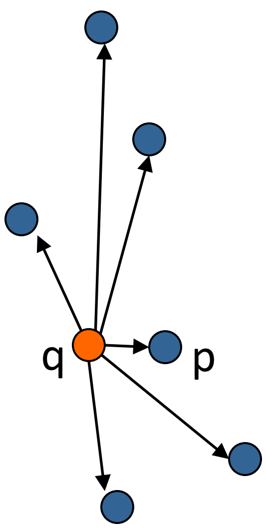
I_2



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

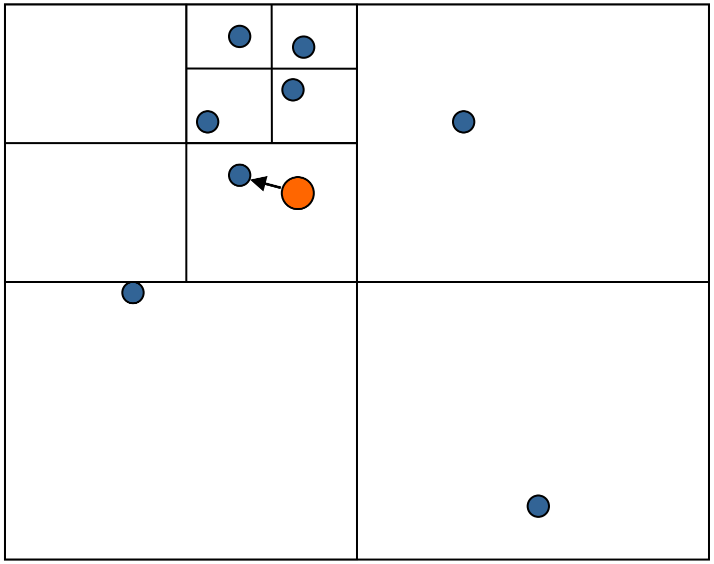
- Exhaustive search $\mathrm{O}(n^2)$



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

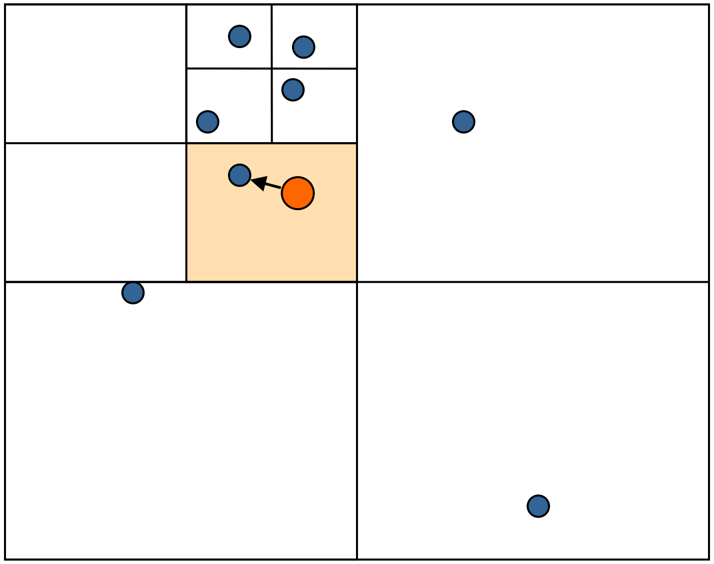
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

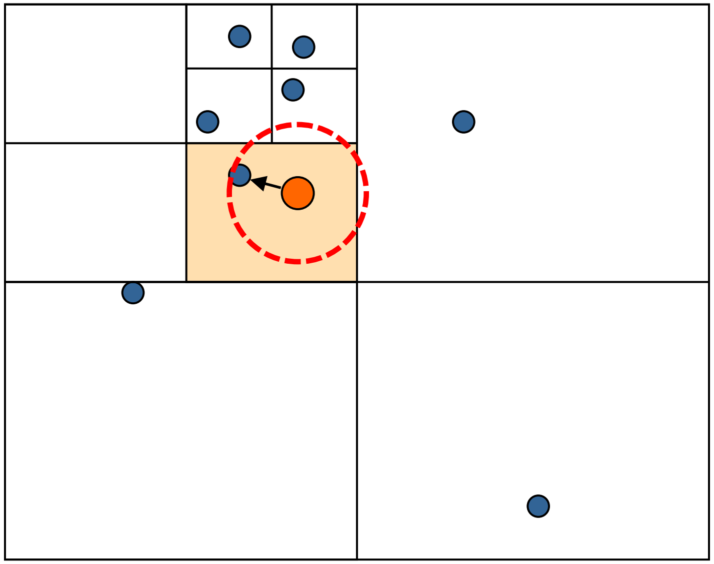
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

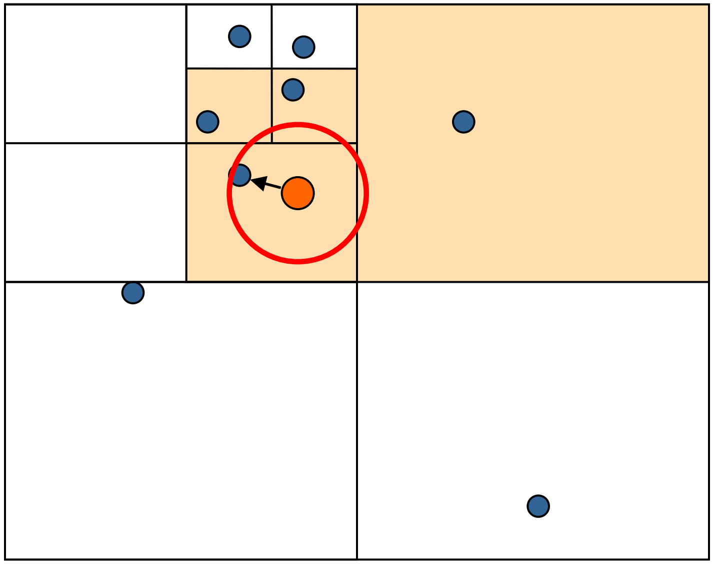
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

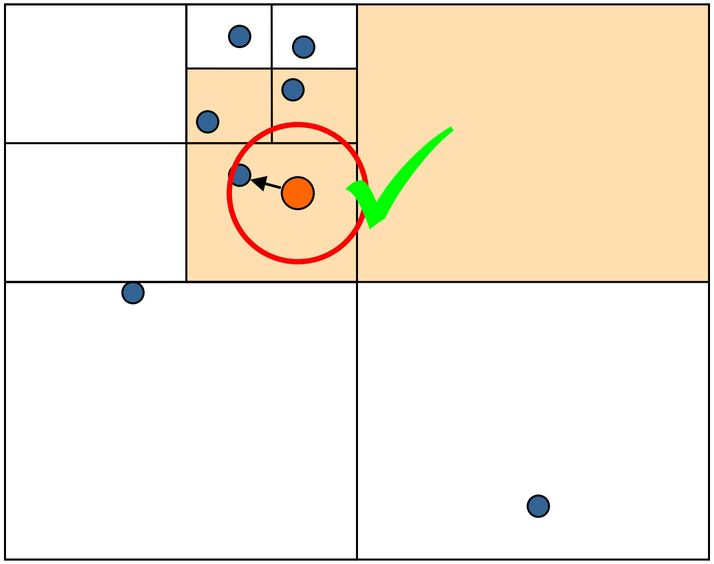
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

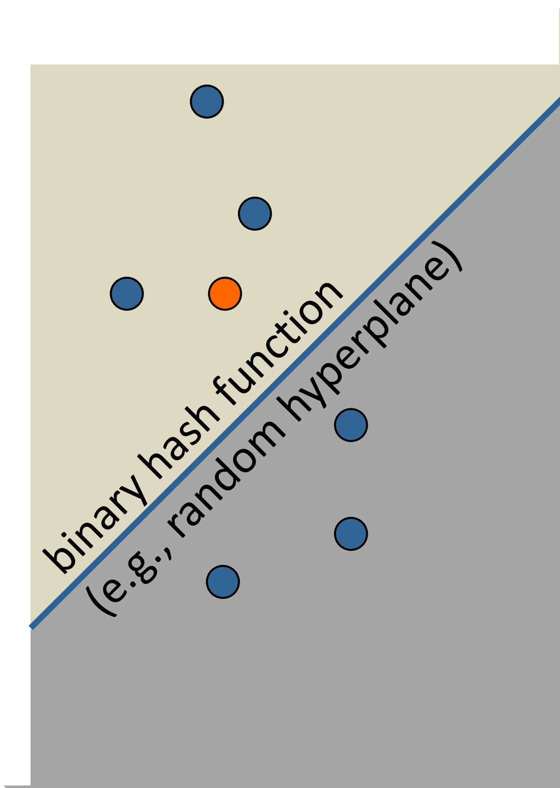
- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found
  - Best-bin-first: use priority queue for unchecked leafs



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

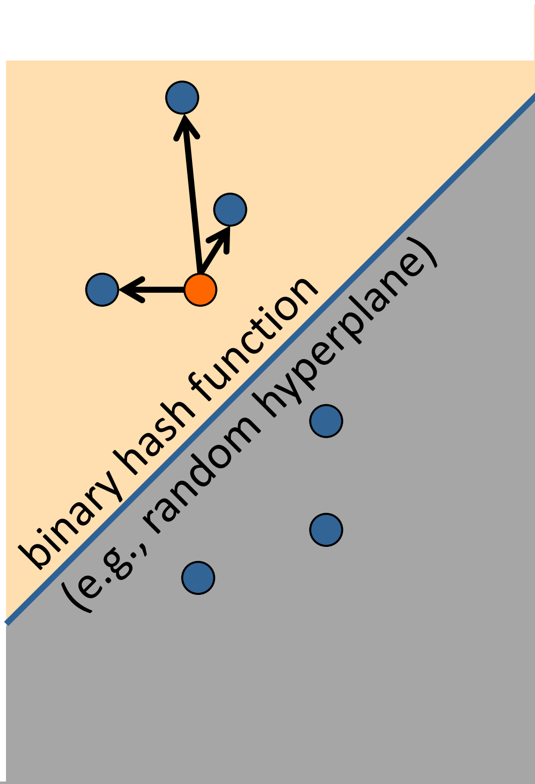
- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
- Approximate search
  - Locality sensitive hashing
  - Approximate nearest neighbor



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
- Approximate search
  - Locality sensitive hashing
  - Approximate nearest neighbor



Efficient Matching

For feature matching, we need to answer a large number of **nearest neighbor queries**

- Exhaustive search $\mathrm{O}(n^2)$
- Indexing (k-d tree)
- Approximate search
- Vocabulary trees

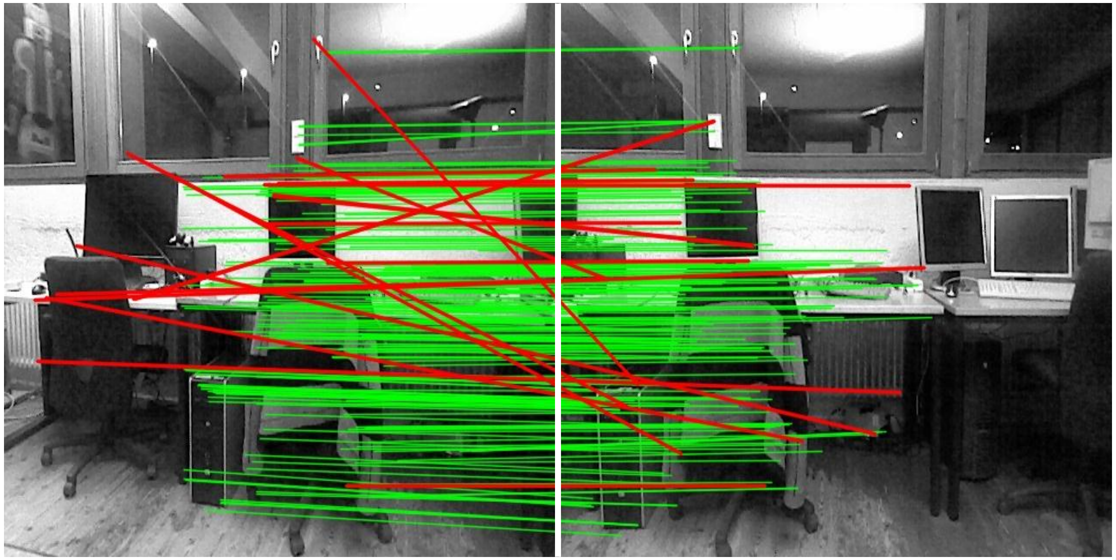
Other Descriptors

- SIFT (Scale Invariant Feature Transform) [Lowe, 2004]
- SURF (Speeded Up Robust Feature) [Bay et al., 2008]
- BRIEF (Binary robust independent elementary features) [Calonder et al., 2010]
- ORB (Oriented FAST and Rotated Brief) [Rublee et al, 2011]
- …

Example: RGB-D SLAM

[Engelhard et al., 2011; Endres et al. 2012]

- Feature descriptor: SURF
- Feature matching: FLANN (approximate nearest neighbor)



I_1 I_2

Structure From Motion (SfM)

- Now we can compute point correspondences
- What can we use them for?

Four Important SfM Problems

- Camera calibration
  Known 3D points, observe corresponding 2D points, compute camera pose
- Point triangulation
  Known camera poses, observe 2D point correspondences, compute 3D point
- Motion estimation (epipolar geometry)
  Observe 2D point correspondences, compute camera pose (up to scale)
- Bundle adjustment (next week!)
  Observe 2D point correspondences, compute camera pose and 3D points (up to scale)

Camera Calibration

- **Given:** n 2D/3D correspondences $\mathbf{x}_i \leftrightarrow \mathbf{p}_i$

- **Wanted:** $M = K(R \;\; \mathbf{t})$
  such that $\tilde{\mathbf{x}}_i = M\mathbf{p}_i$

- The algorithm has two parts:
  1. Compute $M \in \mathbb{R}^{3\times 4}$
  2. Decompose M into $K, R, \mathbf{t}$ via QR decomposition

Step 1: Estimate M

- $\tilde{\mathbf{x}}_i = M\mathbf{p}_i$
- Each correspondence generates two equations

$$x = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}W}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}W} \qquad y = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}W}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}W}$$

- Multiplying out gives equations **linear** in the elements of M

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34}W)x = m_{11}X + m_{12}Y + m_{13}Z + m_{14}W$$

$$(m_{31}X + m_{32}Y + m_{33}Z + m_{34}W)y_j = m_{21}X + m_{22}Y + m_{23}Z + m_{24}W$$

- Re-arrange in matrix form…

Step 1: Estimate M

- Re-arranged in matrix form

$$\begin{pmatrix} X & Y & Z & 1 & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -x \\ 0 & 0 & 0 & 0 & X & Y & Z & 1 & -yX & -yY & -yZ & -y \end{pmatrix} \mathbf{m} = \mathbf{0}$$

  with $\mathbf{m} = (m_{11} \quad m_{12} \quad \dots \quad m_{34}) \in \mathbb{R}^{12}$

- Concatenate equations for n≥6 correspondences

$$A\mathbf{m} = \mathbf{0}$$

- Wanted vector $\mathbf{m}$ is in the null space of A
- Initial solution using SVD (vector with least singular value), refine using non-linear min.

Step 2: Recover K,R,t

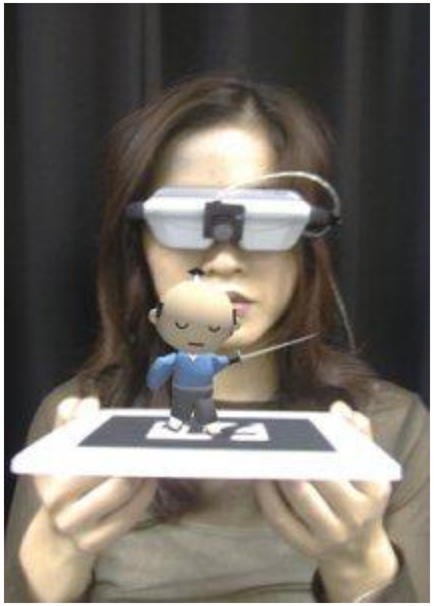
- Remember $M = K(R \;\; \mathbf{t})$
- The first 3x3 submatrix is the product of an upper triangular and orthogonal (rot.) matrix

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Procedure:

1. Factor M into KR using QR decomposition
2. Compute translation as $\mathbf{t} = K^{-1}(p_{14}, p_{24}, p_{34})^\top$

Example: ARToolkit Markers (1999)

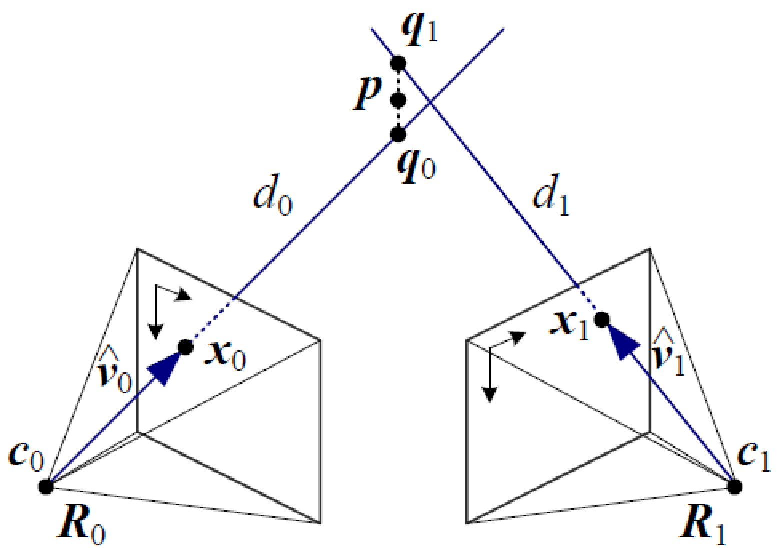


1. Threshold image
2. Detect edges and fit lines
3. Intersect lines to obtain corners
4. Estimate projection matrix M
5. Extract camera pose R,t (assume K is known)

The final error between measured and projected points is typically less than 0.02 pixels

Triangulation

- **Given:** cameras $\{M_j = K_j(R_j \ \mathbf{t}_j)\}$
  point correspondence $\mathbf{x}_0, \mathbf{x}_1$
- **Wanted:** Corresponding 3D point $\mathbf{p}$



Triangulation

- Where do we expect to see $\mathbf{p} = (X \ Y \ Z \ W)^\top$?

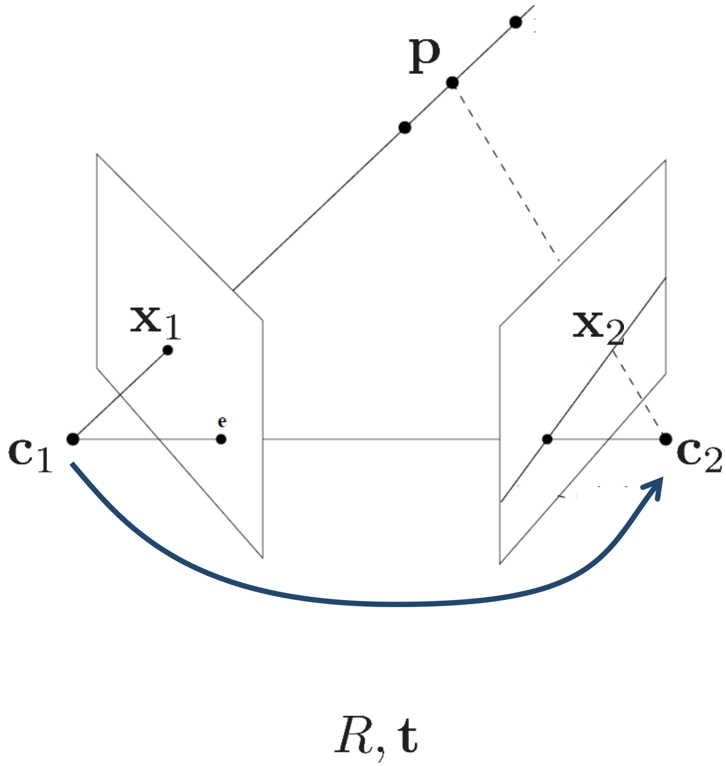
$$\hat{x} = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}W}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}W} \qquad \hat{y} = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}W}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}W}$$

- Minimize the residuals (e.g., using least squares)

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_j d(\mathbf{x}_j, \hat{\mathbf{x}}_j)^2$$

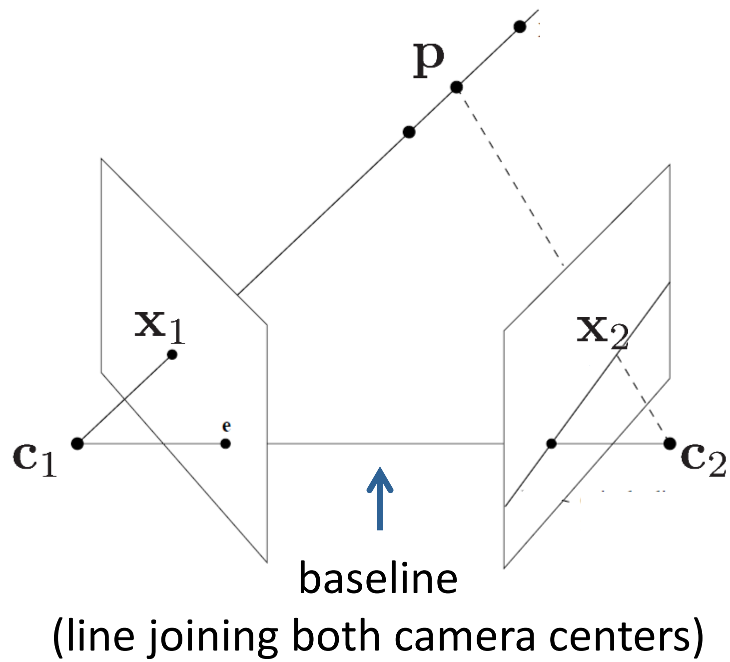
Epipolar Geometry

- Consider two cameras that observe a 3D world point



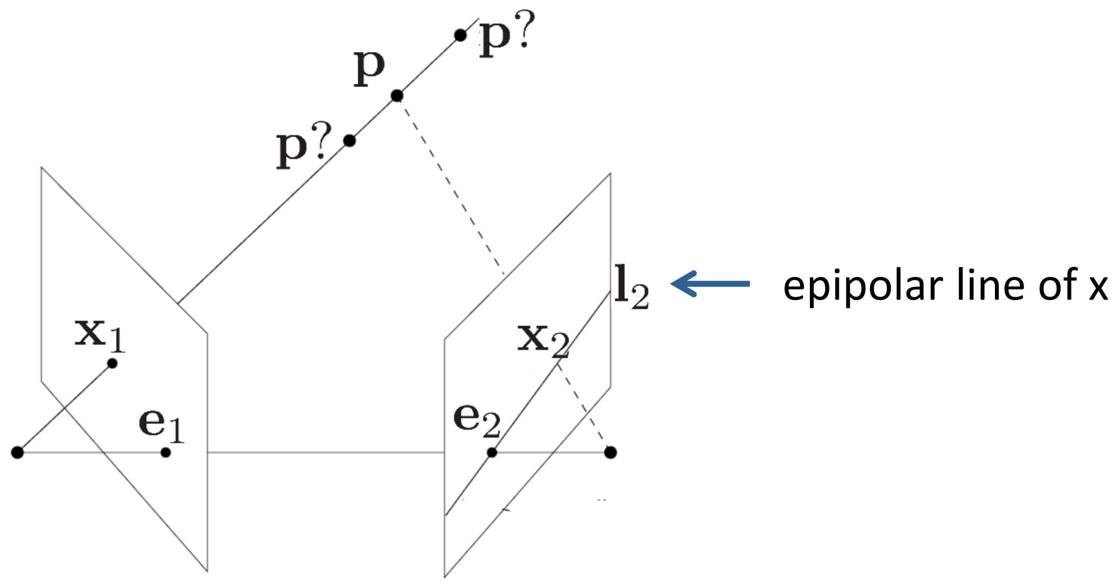
Epipolar Geometry

- The line connecting both camera centers is called the **baseline**



Epipolar Geometry

- Given the image of a point in one view, what can we say about its position in another?

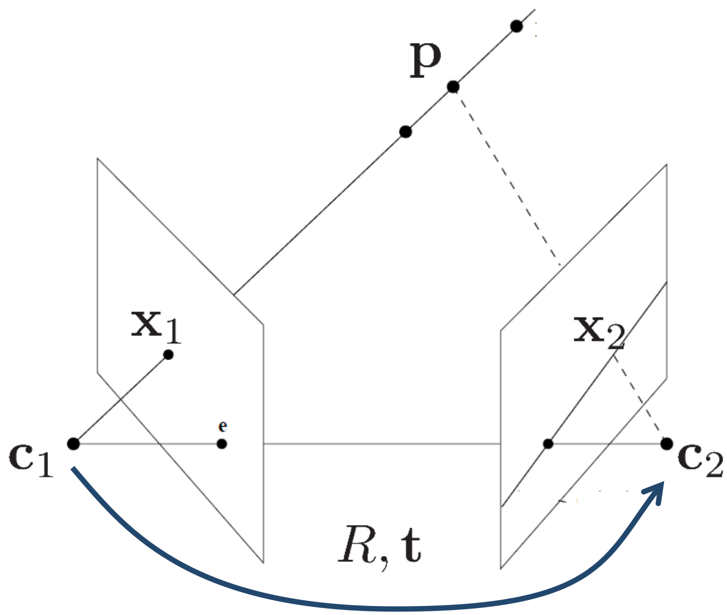


- A point in one image "generates" a line in another image (called the **epipolar line**)

Epipolar Geometry

- Left line in left camera frame $\mathbf{p}_1 = d_1 \hat{\mathbf{x}}_1$
- Right line in right camera frame $\mathbf{p}_2 = d_2 \hat{\mathbf{x}}_2$

where $\hat{\mathbf{x}}_\mathbf{j} = K^{-1} \bar{\mathbf{x}}_j$ are the (local) ray directions



Epipolar Geometry

- Left line in **right** camera frame $\mathbf{p}'_1 = Rd_1\hat{\mathbf{x}}_1 + t$
- Right line in right camera frame $\mathbf{p}_2 = d_2\hat{\mathbf{x}}_2$

where $\hat{\mathbf{x}}_\mathbf{j} = K^{-1}\bar{\mathbf{x}}_j$ are the (local) ray directions

- Intersection of both lines

$$d_2\hat{\mathbf{x}}_2 = Rd_1\hat{\mathbf{x}}_1 + \mathbf{t} \qquad \Big| [\mathbf{t}]_\times \cdot$$

$$d_2[\mathbf{t}]_\times\hat{\mathbf{x}}_2 = d_1[\mathbf{t}]_\times R\hat{\mathbf{x}}_1 + \underbrace{\cancel{[\mathbf{t}]_\times\mathbf{t}}}_{=0} \qquad \Big| \hat{\mathbf{x}}_2^\top \cdot$$

$$\underbrace{\cancel{d_2\hat{\mathbf{x}}_2^\top[\mathbf{t}]_\times\hat{\mathbf{x}}_2}}_{0=} = d_1\hat{\mathbf{x}}_2^\top[\mathbf{t}]_\times R\hat{\mathbf{x}}_1$$

$$0 = \hat{\mathbf{x}}_2^\top[\mathbf{t}]_\times R\hat{\mathbf{x}}_1$$

$$\boxed{0 = \hat{\mathbf{x}}_2^\top E\hat{\mathbf{x}}_1}$$

this is called the epipolar constraint

Epipolar Geometry

Note: The epipolar constraint holds for **every** pair of corresponding points $\mathbf{x}_1, \mathbf{x}_2$

$$\hat{\mathbf{x}}_2^\top E \hat{\mathbf{x}}_1 = 0$$

where E is called the essential matrix

$$E = [\mathbf{t}]_\times R \in \mathbb{R}^{3\times 3}$$

8-Point Algorithm: General Idea

1. Estimate the essential matrix E from at least eight point correspondences
2. Recover the relative pose R,t from E (up to scale)

Step 1: Estimate E

- Epipolar constraint $\hat{\mathbf{x}}_2^\top E \hat{\mathbf{x}}_1 = 0$

- Written out (with $\mathbf{x}_j = (x_j, y_j, 1)^\top$)

$$\begin{array}{ccccccc} x_1 x_2 e_{11} & + & y_1 x_2 e_{12} & + & x_2 e_{13} & + & \\ x_1 y_2 e_{21} & + & y_1 y_2 e_{22} & + & y_2 e_{23} & + & \\ x_1 e_{31} & + & y_1 e_{32} & + & 1 e_{33} & = & 0 \end{array}$$

- Stack the elements into two vectors

$$\left.\begin{array}{l} \mathbf{z} = (x_1 x_2 \;\; y_1 x_2 \;\; \ldots \;\; 1)^\top \\ \mathbf{e} = (e_{11} \;\; e_{12} \;\; \ldots \;\; e_{33})^\top \end{array}\right\} \mathbf{z}^\top \mathbf{e} = 0$$

Step 1: Estimate E

- Each correspondence gives us one constraint

$$\left.\begin{aligned} \mathbf{z}_1^\top \mathbf{e} &= 0 \\ \mathbf{z}_2^\top \mathbf{e} &= 0 \\ &\vdots \\ \mathbf{z}_n^\top \mathbf{e} &= 0 \end{aligned}\right\} \quad Z\mathbf{e} = \mathbf{0}$$

- Linear system with *n* equations
- e is in the null-space of Z
- Solve using SVD (assuming $\|\mathbf{e}\| = 1$)

Normalized 8-Point Algorithm

[Hartley 1997]

- Noise in the point observations is unequally distributed in the constraints, e.g.,

double noise

$$
\begin{array}{lllllll}
x_1 x_2 e_{11} & + & y_1 x_2 e_{12} & + & x_2 e_{13} & + & \\
x_1 y_2 e_{21} & + & y_1 y_2 e_{22} & + & y_2 e_{23} & + & \\
x_1 e_{31} & + & y_1 e_{32} & + & 1 e_{33} & = & 0
\end{array}
$$

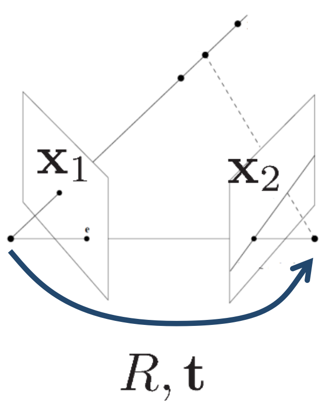
normal noise

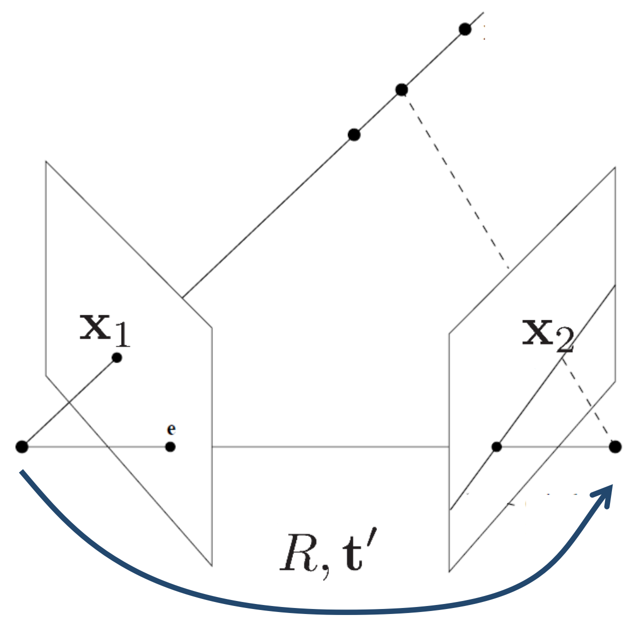
noise free

- Estimation is sensitive to scaling
- Normalize all points to have zero mean and unit variance

Step 2: Recover R,t

- **Note:** The absolute distance between the two cameras can never be recovered from pure images measurements alone!!!
- Illustration





- We can only recover the translation $\hat{\mathbf{t}}$ up to scale

Step 2a: Recover t

- Remember: $E = [\mathbf{t}]_\times R$
- Therefore, $\mathbf{t}^\top$ is in the null space of E

$$\mathbf{t}^\top E = \underbrace{\mathbf{t}^\top [\mathbf{t}]_\times}_{=0} R = 0$$

→ Recover $\hat{\mathbf{t}}$ (up to scale) using SVD

$$E = [\hat{\mathbf{t}}]_\times R = U \Sigma V^\top$$

$$= \begin{pmatrix} \mathbf{u}_0 & \mathbf{u}_1 & \boxed{\hat{\mathbf{t}}} \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ & & \boxed{0} \end{pmatrix} \begin{pmatrix} \mathbf{v}_0^\top & \mathbf{v}_1^\top & \mathbf{v}_2^\top \end{pmatrix}$$

Step 2b: Recover R

Remember, the cross-product $[\hat{\mathbf{t}}]_\times$

- ... projects a vector onto a set of orthogonal basis vectors including $\hat{\mathbf{t}}$
- ... zeros out the $\hat{\mathbf{t}}$ component
- ... rotates the other two by 90°

$$
\begin{aligned}
[\hat{\mathbf{t}}]_\times &= S Z R_{90^\circ} S^\top \\
&= (\mathbf{s}_0 \ \mathbf{s}_1 \ \hat{\mathbf{t}}) \begin{pmatrix} 1 & & \\ & 1 & \\ & & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 & \\ 1 & 0 & \\ & & 1 \end{pmatrix} \begin{pmatrix} \mathbf{s}_0 \\ \mathbf{s}_1 \\ \hat{\mathbf{t}} \end{pmatrix}
\end{aligned}
$$

Step 2b: Recover R

- Plug this into the essential matrix equation

$$E = [\mathbf{t}]_\times R = SZR_{90^\circ}S^\top R = U\Sigma V^\top$$

=

=

- By identifying $S = U$ and $Z = \Sigma$, we obtain

$$R_{90^\circ}U^\top R = V^\top$$

$$\boxed{R = UR_{90^\circ}^\top V^\top}$$

Summary: 8-Point Algorithm

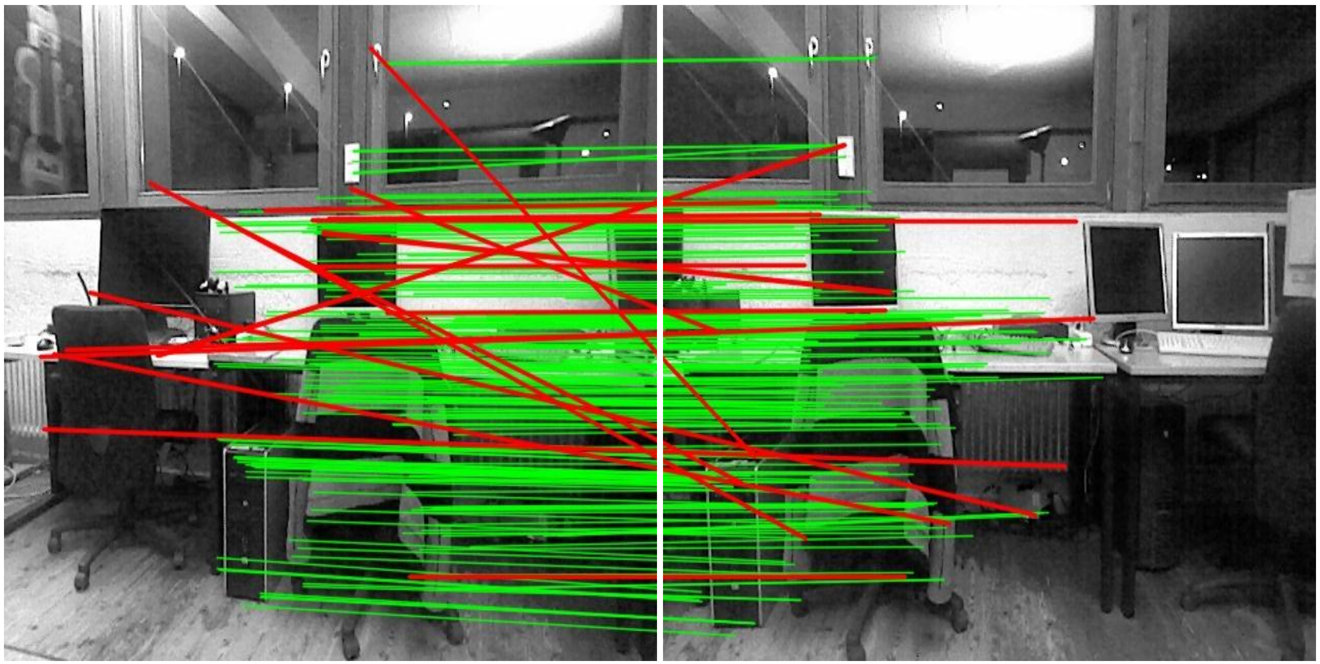
Given: Image pair

Find: Camera motion R,t (up to scale)

- Compute correspondences
- Compute essential matrix
- Extract camera motion

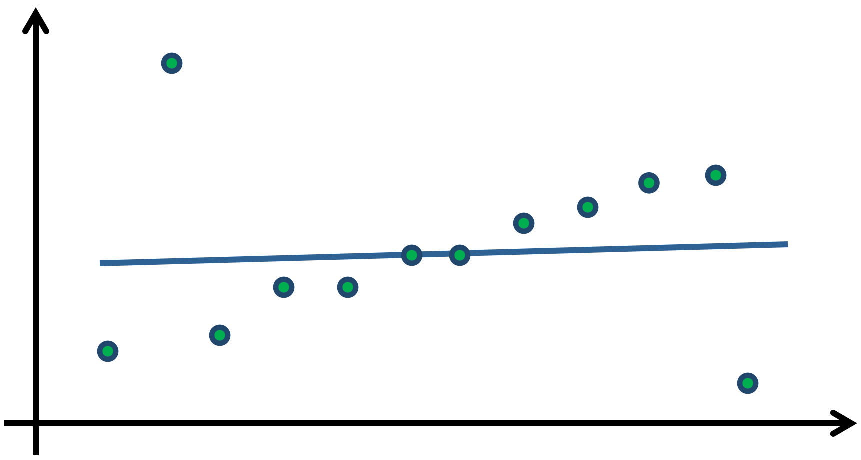
How To Deal With Outliers?

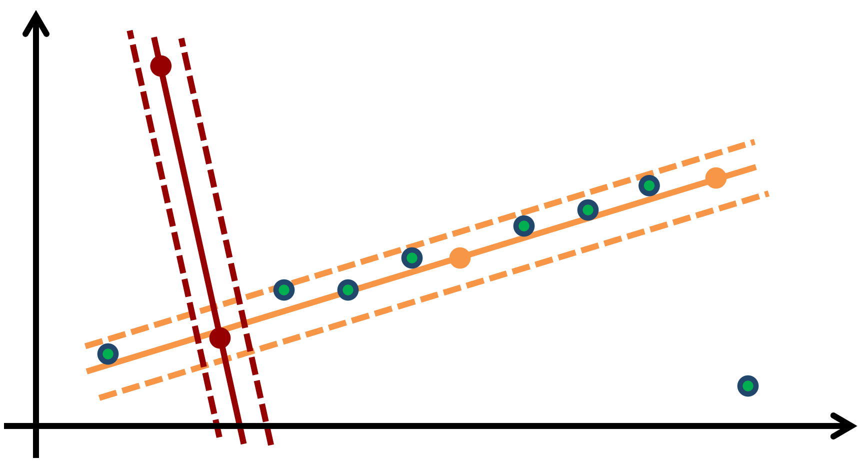


Problem: No matter how good the feature descriptor/matcher is, there is always a chance for bad point correspondences (=outliers)

Robust Estimation

Example: Fit a line to 2D data containing outliers





There are two problems

1. **Fit** the line to the data $\arg\min_l \sum_i d_i^2$
2. **Classify** the data into inliers (valid points) and outliers (using some threshold)

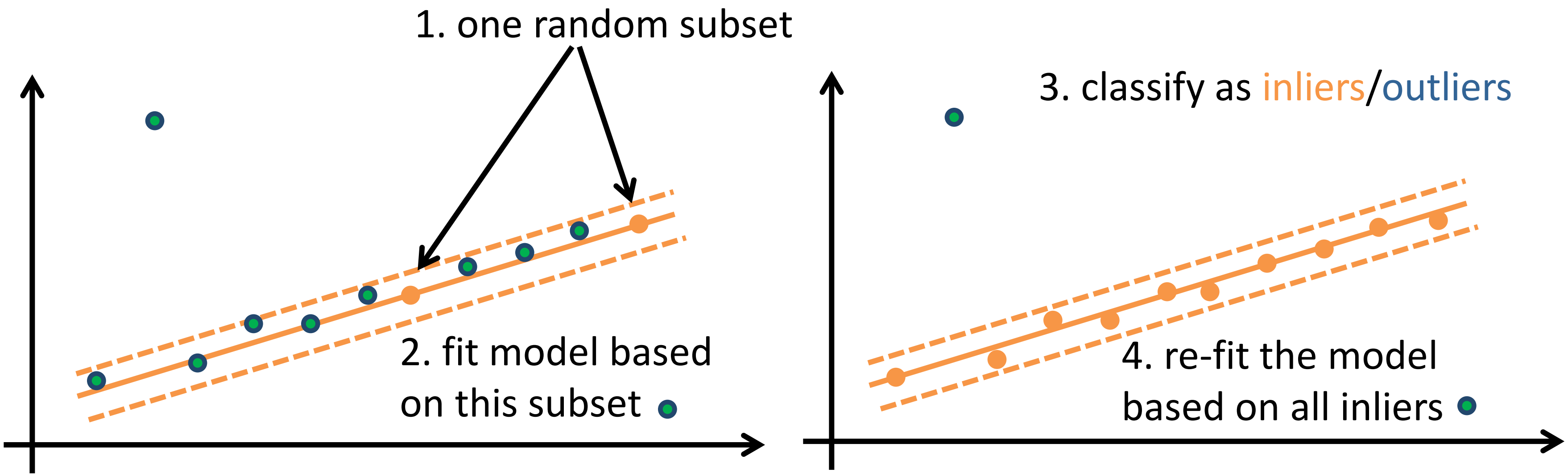
RANdom SAmple Consensus (RANSAC)
[Fischler and Bolles, 1981]

Goal: Robustly fit a model to a data set S which contains outliers

Algorithm:

1. Randomly select a (minimal) subset of data points and instantiate the model from it
2. Using this model, classify the all data points as inliers or outliers
3. Repeat 1&2 for N iterations
4. Select the largest inlier set, and re-estimate the model from all points in this set

RANdom SAmple Consensus (RANSAC)



- RANSAC is used very widely
- Many improvements/variants, e.g., MLESAC:

$$\arg\min_l \sum_i \rho(d_i) \text{ with } \rho(d) = \begin{cases} d^2 & \text{if } d \leq e\,(inlier) \\ e^2 & \text{if } d > e\,(outlier) \end{cases}$$

How Many Samples?

- For probability p of having no outliers, we need

to sample $N = \dfrac{\log(1-p)}{\log(1-(1-\epsilon)^s)}$ subsets

for subset size s and outlier ratio ϵ

- E.g., for p=0.95:

Sample size	Proportion of outliers ϵ						
s	5%	10%	20%	25%	30%	40%	50%
2	2	2	3	4	5	7	11
3	2	3	5	6	8	13	23
4	2	3	6	8	11	22	47
5	3	4	8	12	17	38	95
6	3	4	10	16	24	63	191
7	3	5	13	21	35	106	382
8	3	6	17	29	51	177	766

Two Examples

- PTAM

  G. Klein and D. Murray, Parallel Tracking and Mapping for Small AR Workspaces, International Symposium on Mixed and Augmented Reality (ISMAR), 2007

  http://www.robots.ox.ac.uk/~gk/publications/KleinMurray2007ISMAR.pdf

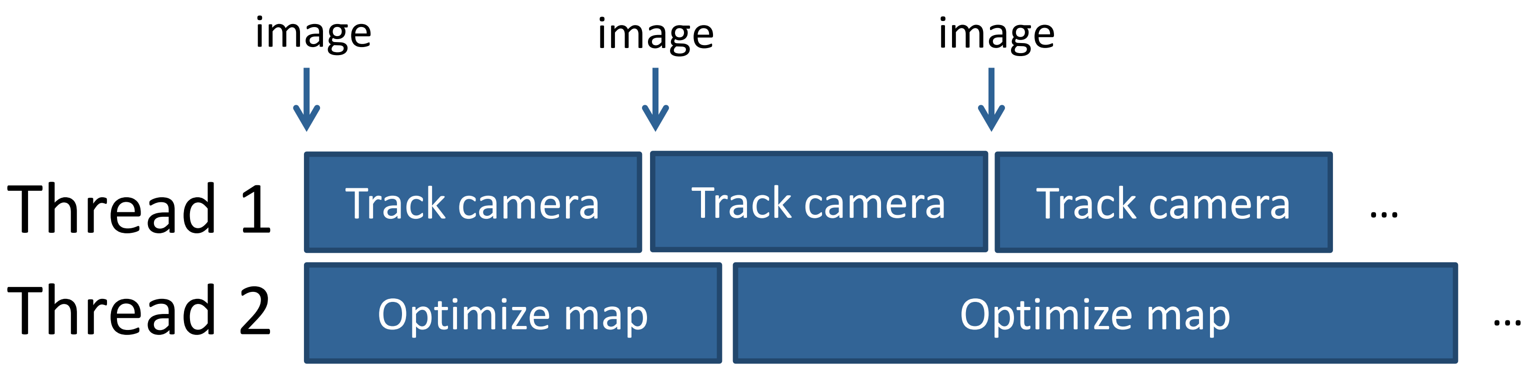
- Photo Tourism

  N. Snavely, S. M. Seitz, R. Szeliski, Photo tourism: Exploring photo collections in 3D, ACM Transactions on Graphics (SIGGRAPH), 2006

  http://phototour.cs.washington.edu/Photo_Tourism.pdf

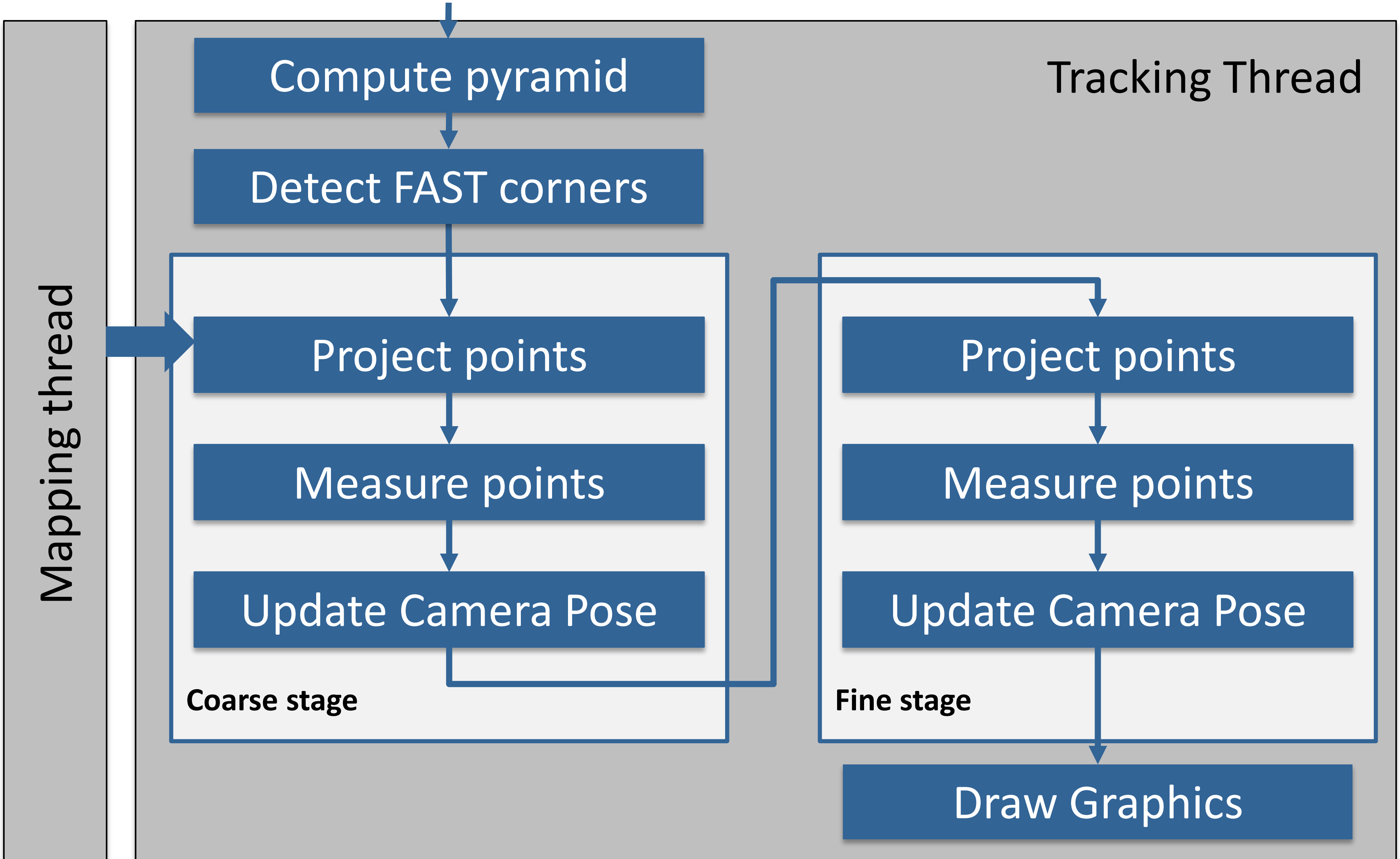
PTAM (2007)

- Architecture optimized for dual cores



- Tracking thread runs in real-time (30Hz)
- Mapping thread is not real-time

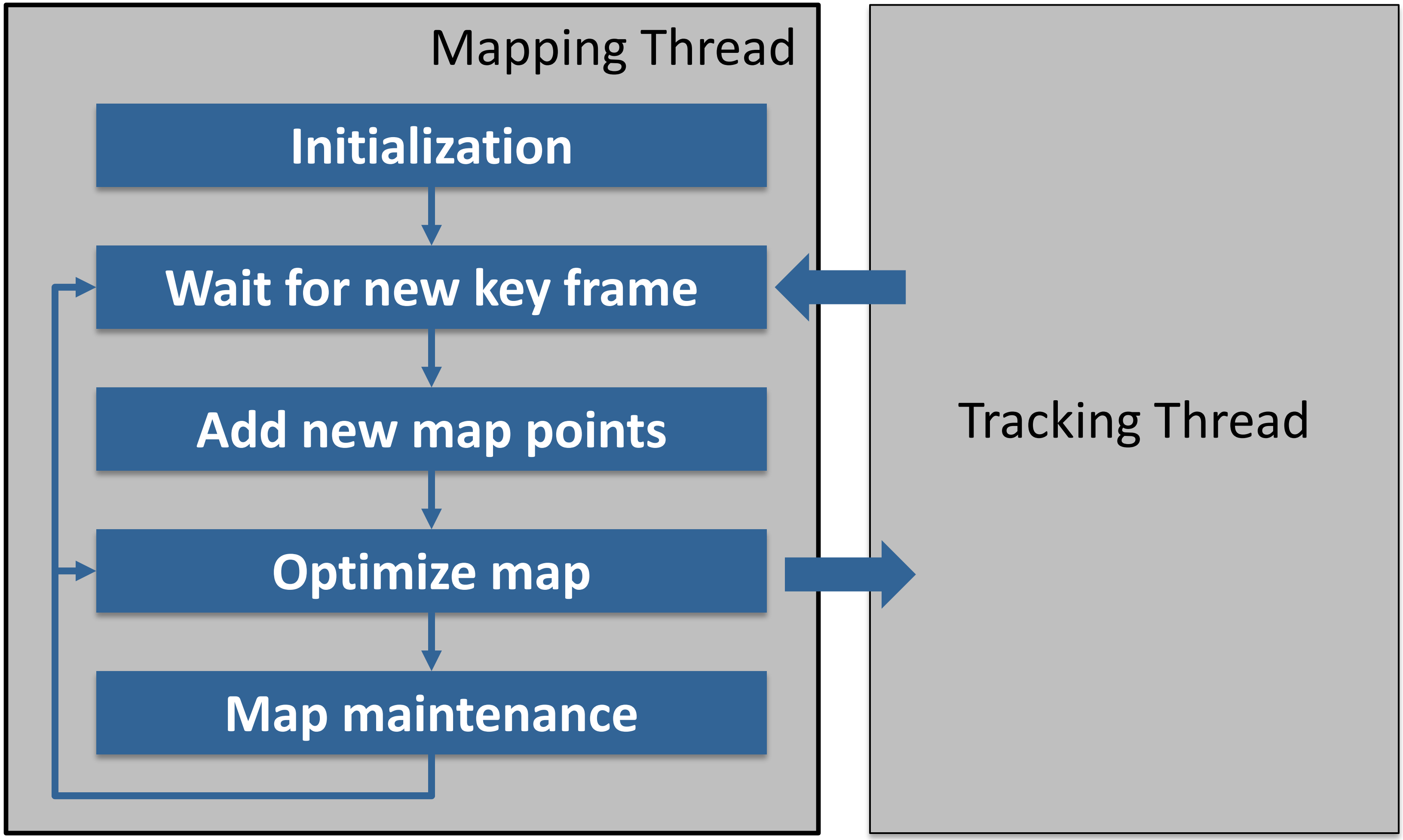
PTAM – Tracking Thread



PTAM – Feature Tracking

- Generate 8x8 matching template (warped from key frame to current pose estimate)
- Search a fixed radius around projected position
  - Using SSD
  - Only search at FAST corner points

PTAM – Mapping Thread



PTAM – Example Timings

- Tracking thread

Total	19.2 ms
Key frame preparation	2.2 ms
Feature Projection	3.5 ms
Patch search	9.8 ms
Iterative pose update	3.7 ms

- Mapping thread

Key frames	2-49	50-99	100-149
Local Bundle Adjustment	170 ms	270 ms	440 ms
Global Bundle Adjustment	380 ms	1.7 s	6.9 s

PTAM Video

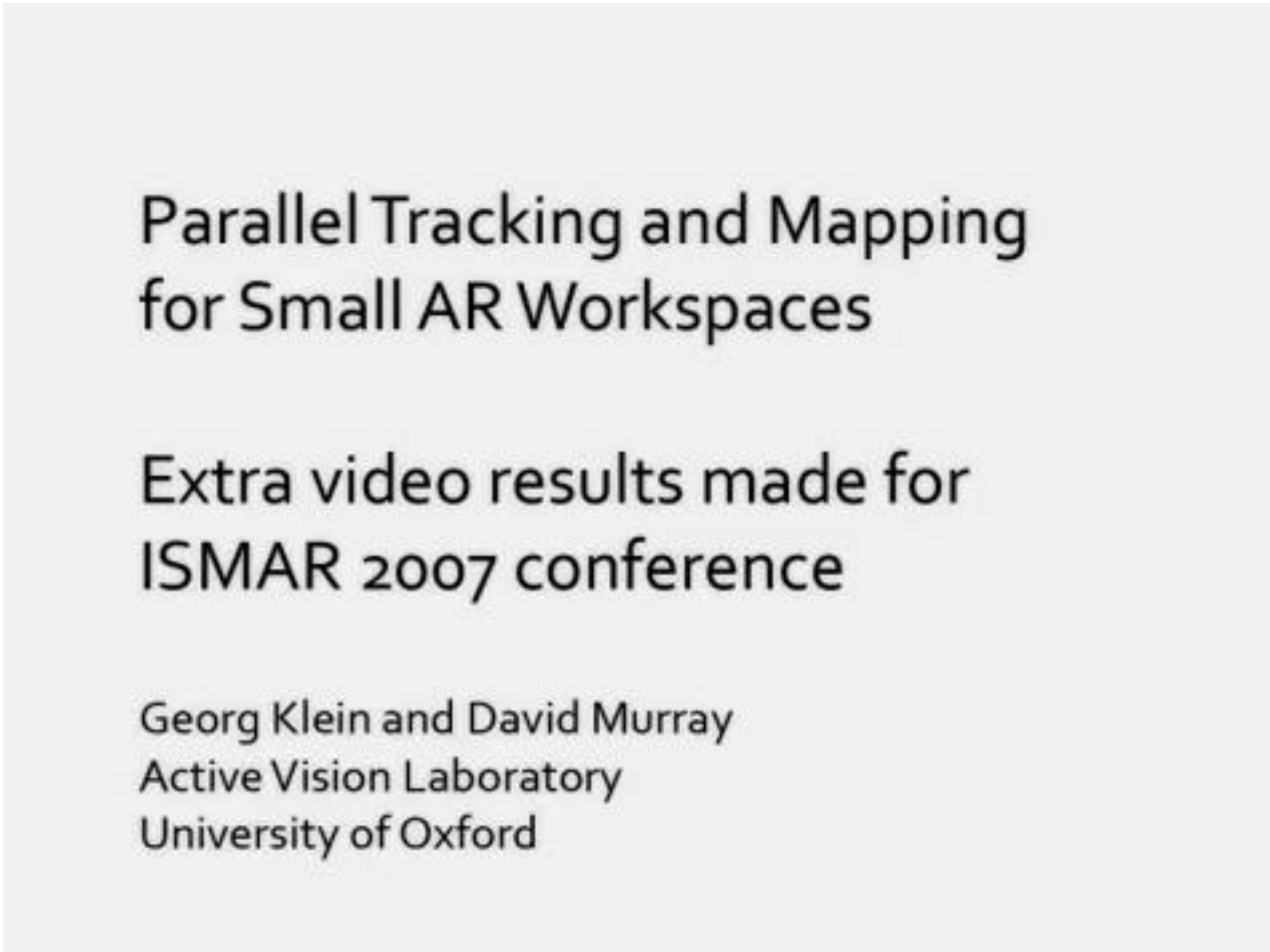
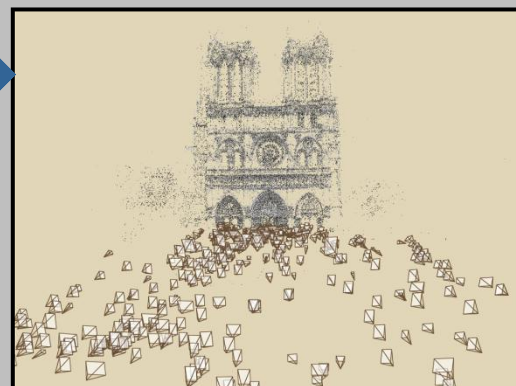

Parallel Tracking and Mapping
for Small AR Workspaces
Extra video results made for
ISMAR 2007 conference
Georg Klein and David Murray
Active Vision Laboratory
University of Oxford

Photo Tourism (2006)

- Overview

Input Photographs (from Flickr)

Scene Reconstruction



Relative camera positions and orientations

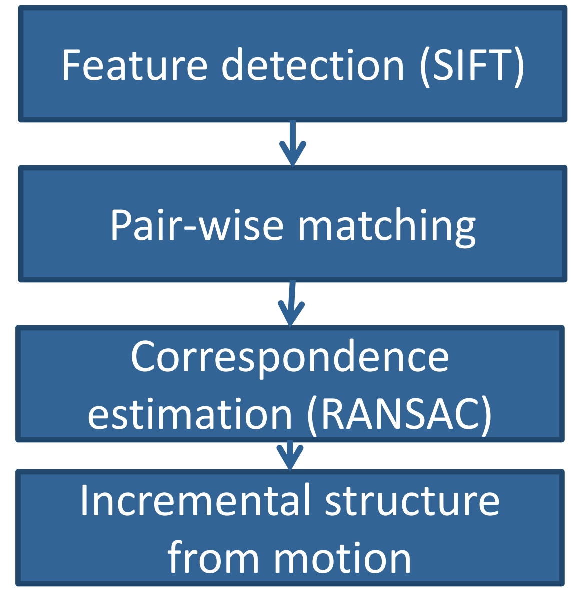
Point cloud

Sparse correspondence

Photo Explorer

Photo Tourism – Scene Reconstruction

- Processing pipeline



- Automatically estimate
  - Position, orientation and focal length of all cameras
  - 3D positions of point features

Photo Tourism – Input Images

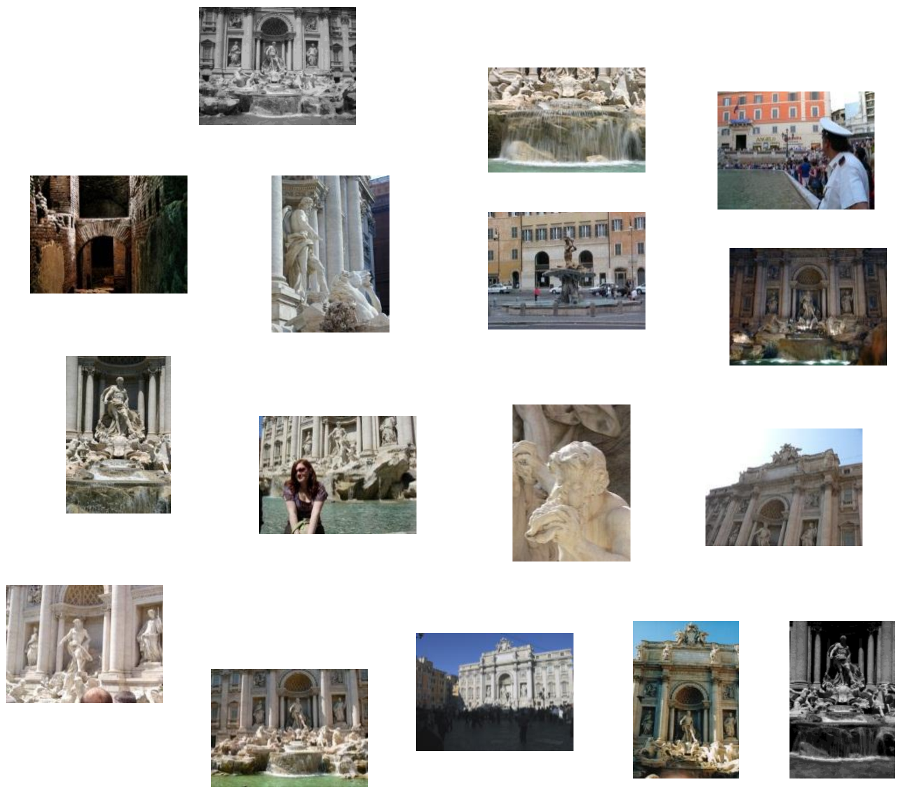


Photo Tourism – Feature Detection

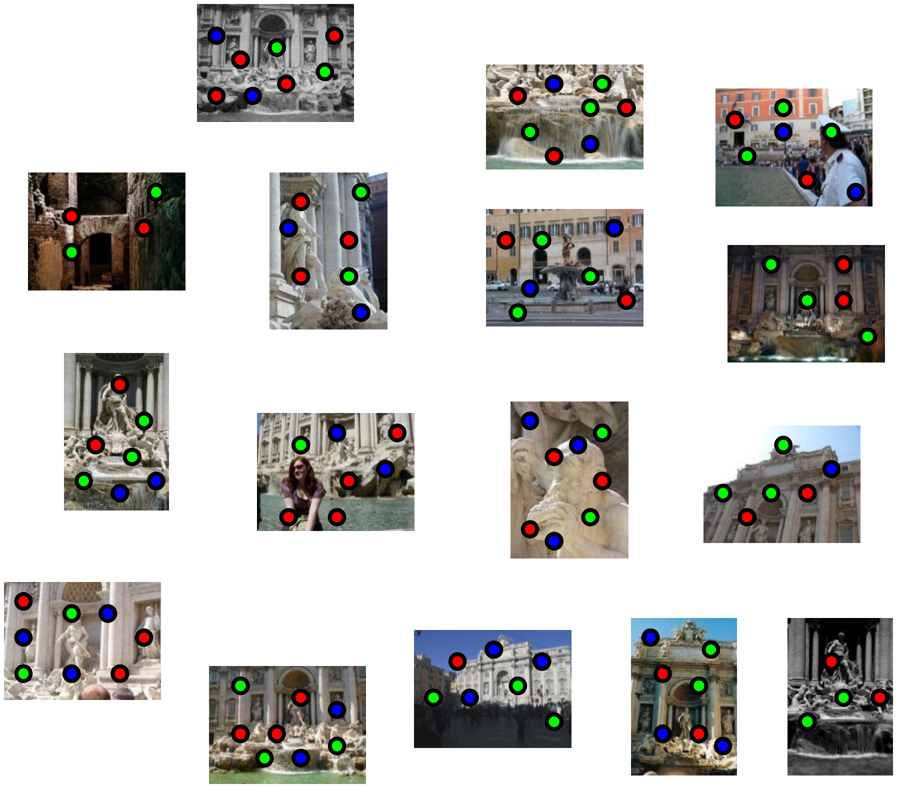
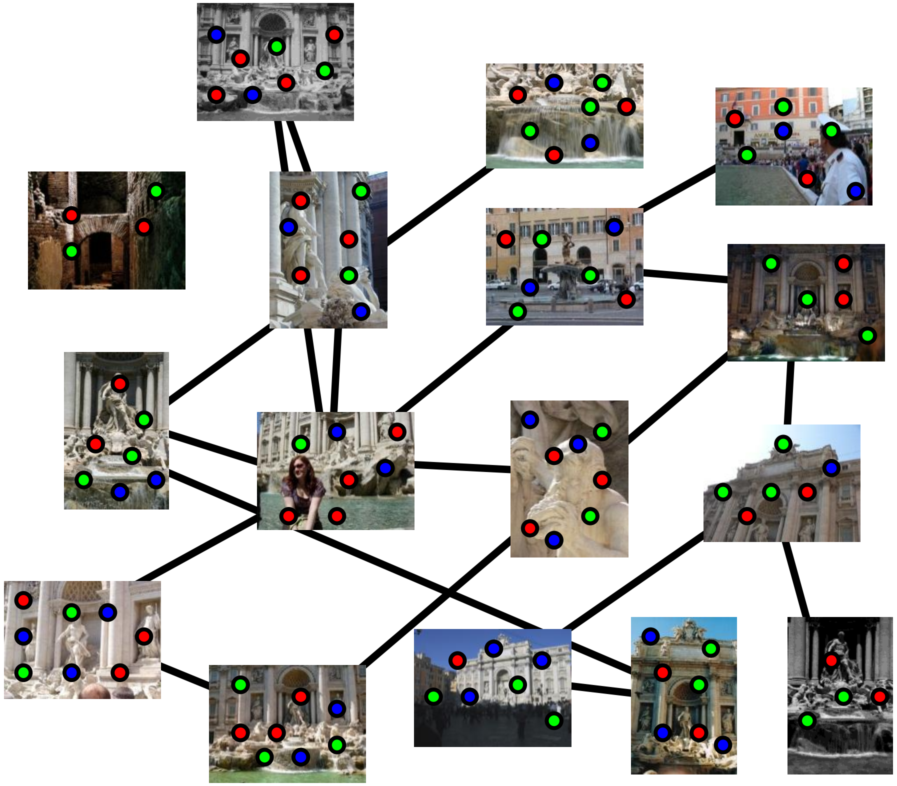


Photo Tourism – Feature Matching



Incremental Structure From Motion

- To help get good initializations, start with two images only (compute pose, triangulate points)
- Non-linear optimization
- Iteratively add more images

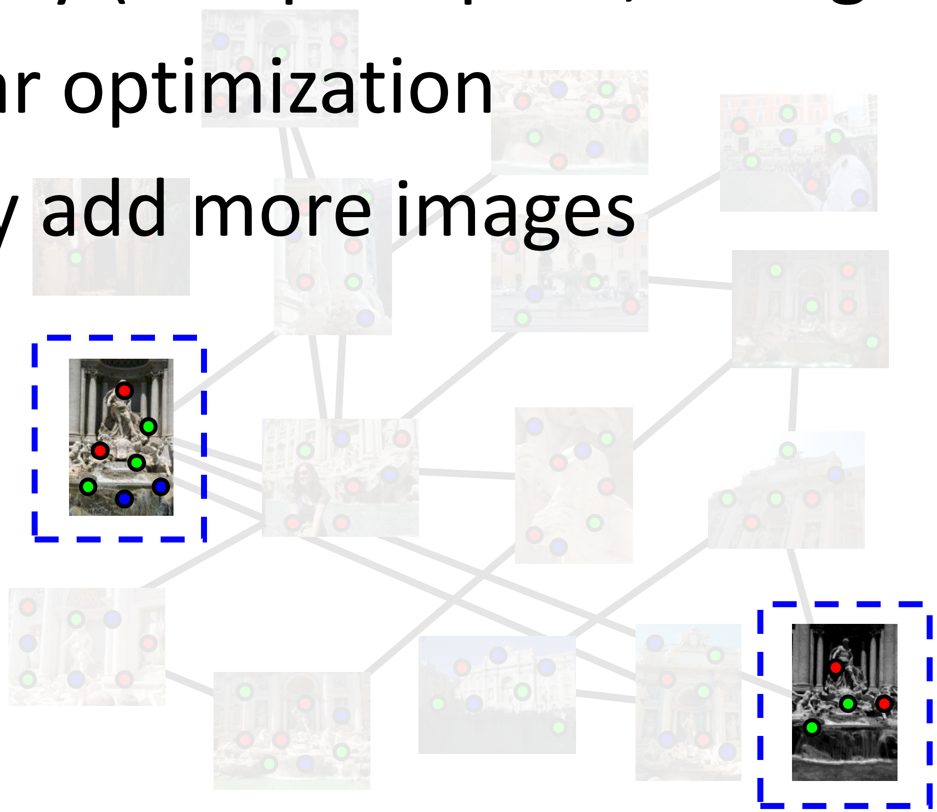


Photo Tourism – Video

Lessons Learned Today

- … how to detect and match feature points
- … how to compute the camera pose and to triangulate points
- ... how to deal with outliers