Computer Vision

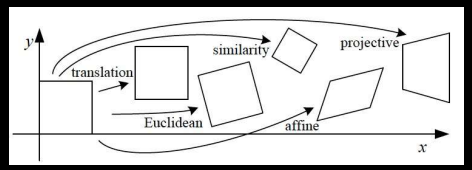
Introduction to "features"

Text resources

- Forsyth and Ponce: 5.3-5.4
  - Szeliski also covers this well – Section 4 – 4.1.1

The basic image point matching problem

- Suppose I have two images related by some transformation. Or have two images of the same object in different positions.
- How to find the transformation of image 1 that would align it with image 2?



We want *Local*(1) *Features*(2)

- Goal: Find points in an image that can be:
  - Found in other images
  - Found precisely – well localized
  - Found reliably – well matched

We want *Local*(1) *Features*(2)

Why?

- Want to compute a fundamental matrix to recover geometry
- Robotics/Vision: See how a bunch of points move from one frame to another. Allows computation of how camera moved -> depth -> moving objects
- Build a panorama…

Suppose you want to build a panorama

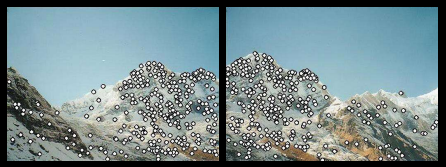
M. Brown and D. G. Lowe. Recognising Panoramas. ICCV 2003

How do we build panorama?

- We need to match (align) images

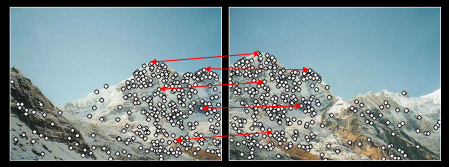
Matching with Features

- Detect features (feature points) in both images



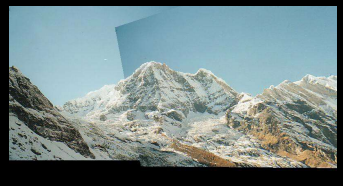
Matching with Features

- Detect features (feature points) in both images
- Match features - find corresponding pairs



Matching with Features

- Detect features (feature points) in both images
- Match features - find corresponding pairs
- Use these pairs to align images



Matching with Features

- Problem 1:
  - Detect the same point independently in both images

no chance to match!

We need a repeatable detector

Matching with Features

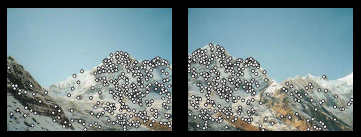
- Problem 2:
  - For each point correctly recognize the corresponding one

We need a reliable and distinctive ***descriptor***

More motivation…

- Feature points are used also for:
  - Image alignment (e.g. homography or fundamental matrix)
  - 3D reconstruction
  - Motion tracking
  - Object recognition
  - Indexing and database retrieval
  - Robot navigation
  - … other

Characteristics of good features



Characteristics of good features

Repeatability/Precision

- The same feature can be found in several images despite geometric and photometric transformations

Characteristics of good features

Saliency/Matchability

- Each feature has a distinctive description

Characteristics of good features

Compactness and efficiency

- Many fewer features than image pixels

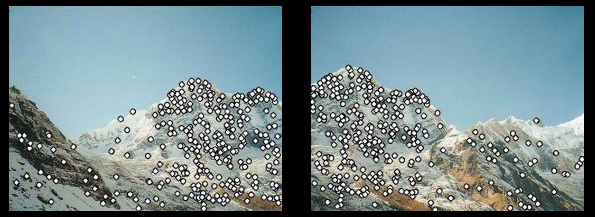
Characteristics of good features

Locality

- A feature occupies a relatively small area of the image; robust to clutter and occlusion

Finding corners

Feature points



Characteristics of good features

Repeatability/Precision

- The same feature can be found in several images despite geometric and photometric transformations

Characteristics of good features

Saliency/Matchability

- Each feature has a distinctive description

Characteristics of good features

Compactness and efficiency

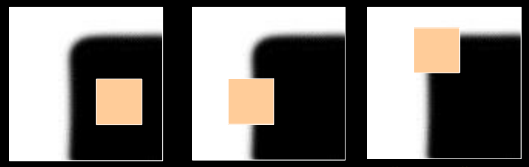
- Many fewer features than image pixels

Characteristics of good features

Locality

- A feature occupies a relatively small area of the image; robust to clutter and occlusion

Corner Detection: Basic Idea



"flat" region: no change in all directions

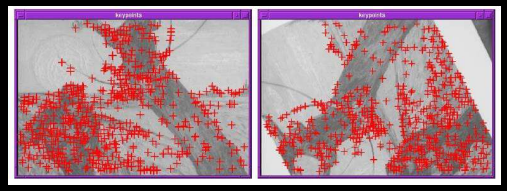
"edge": no change along the edge direction

"corner": significant change in all directions with small shift

Source: A. Efros

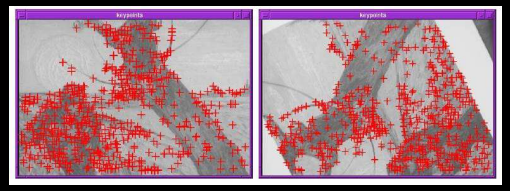
Finding Corners

- Key property: in the region around a corner, image gradient has two or more dominant directions



Finding Corners

C. Harris and M. Stephens. *"A Combined Corner and Edge Detector," Proceedings of the 4th Alvey Vision Conference*: **1988**

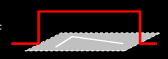
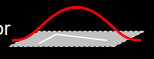


Corner Detection: Mathematics

Change in appearance for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

Window function | Shifted intensity | Intensity

Window function $w(x,y) =$  or 

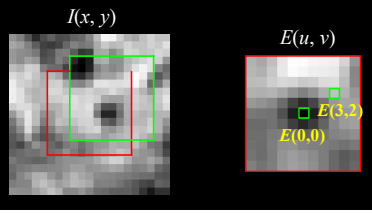
1 in window, 0 outside — Gaussian

Source: R. Szeliski

Corner Detection: Mathematics

Change in appearance for the shift [*u,v*]:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$



Corner Detection: Mathematics

Change in appearance for the shift [*u,v*]:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

We want to find out how this function behaves for ***small*** shifts (u,v near 0,0)

Corner Detection: Mathematics

Change in appearance for the shift [*u,v*]:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

Second-order Taylor expansion of $E(u,v)$ about (0,0) (local quadratic approximation for small *u,v*):

Corner Detection: Mathematics

Change in appearance for the shift [*u,v*]:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

$$F(\delta x) \approx F(0) + \delta x \cdot \frac{dF(0)}{dx} + \frac{1}{2}\delta x^2 \cdot \frac{d^2F(0)}{dx^2}$$

Corner Detection: Mathematics

Change in appearance for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

$$F(\delta x) \approx F(0) + \delta x \cdot \frac{dF(0)}{dx} + \frac{1}{2}\delta x^2 \cdot \frac{d^2F(0)}{dx^2}$$

$$E(u,v) \approx E(0,0) + \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E(u,v) \approx E(0,0) + \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E(u,v) \approx E(0,0) + \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Need these derivatives…

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E_u(u,v) = \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right] I_x(x+u,y+v)$$

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E_{uu}(u,v) = \sum_{x,y} 2w(x,y) I_x(x+u,y+v) I_x(x+u,y+v) + \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right] I_{xx}(x+u,y+v)$$

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E_{uv}(u,v) = \sum_{x,y} 2w(x,y) I_y(x+u,y+v) I_x(x+u,y+v) + \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right] I_{xy}(x+u,y+v)$$

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E(u,v) \approx E(0,0) + \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E_u(u,v) = \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right] I_x(x+u,y+v)$$

$$E_{uu}(u,v) = \sum_{x,y} 2w(x,y) I_x(x+u,y+v) I_x(x+u,y+v) + \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right] I_{xx}(x+u,y+v)$$

$$E_{uv}(u,v) = \sum_{x,y} 2w(x,y) I_y(x+u,y+v) I_x(x+u,y+v) + \sum_{x,y} 2w(x,y)\left[I(x+u,y+v) - I(x,y)\right] I_{xy}(x+u,y+v)$$

Evaluate E and its derivatives at **(0,0)**:

$$E(u,v) \approx \underbrace{E(0,0)}_{=0} + \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E_u(0,0) = \sum_{x,y} 2w(x,y)\underbrace{\left[I(x,y) - I(x,y)\right]}_{=0} I_x(x,y)$$

$$E_{uu}(0,0) = \sum_{x,y} 2w(x,y) I_x(x,y) I_x(x,y) + \sum_{x,y} 2w(x,y)\underbrace{\left[I(x,y) - I(x,y)\right]}_{=0} I_{xx}(x,y)$$

$$E_{uv}(0,0) = \sum_{x,y} 2w(x,y) I_y(x,y) I_x(x,y) + \sum_{x,y} 2w(x,y)\underbrace{\left[I(x,y) - I(x,y)\right]}_{=0} I_{xy}(x,y)$$

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E(u,v) \approx E(0,0) + \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(0,0) = 0 \qquad E_{uu}(0,0) = \sum_{x,y} 2\, w(x,y) I_x(x,y) I_x(x,y)$$

$$E_u(0,0) = 0 \qquad E_{vv}(0,0) = \sum_{x,y} 2\, w(x,y) I_y(x,y) I_y(x,y)$$

$$E_v(0,0) = 0 \qquad E_{uv}(0,0) = \sum_{x,y} 2\, w(x,y) I_x(x,y) I_y(x,y)$$

Second-order Taylor expansion of $E(u,v)$ about (0,0):

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum_{x,y} w(x,y) I_x^2(x,y) & \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y) \\ \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y) & \sum_{x,y} w(x,y) I_y^2(x,y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(0,0) = 0 \qquad E_{uu}(0,0) = \sum_{x,y} 2\, w(x,y) I_x(x,y) I_x(x,y)$$

$$E_u(0,0) = 0 \qquad E_{vv}(0,0) = \sum_{x,y} 2\, w(x,y) I_y(x,y) I_y(x,y)$$

$$E_v(0,0) = 0 \qquad E_{uv}(0,0) = \sum_{x,y} 2\, w(x,y) I_x(x,y) I_y(x,y)$$

Corner Detection: Mathematics

The quadratic approximation simplifies to

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a *second moment matrix* computed from image derivatives:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

The second moment matrix M:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Each product is a rank 1 2x2

Can be written (without the weight):

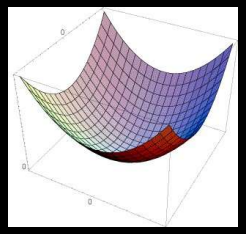
$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \left(\begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \right) = \sum \nabla I (\nabla I)^T$$

Interpreting the second moment matrix

The surface $E(u,v)$ is locally approximated by a quadratic form.

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

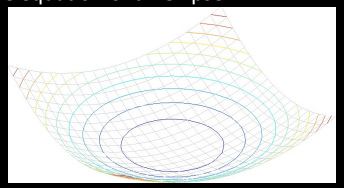


Interpreting the second moment matrix

Consider a constant "slice" of $E(u, v)$:

$$\Sigma I_x^2 u^2 + 2\Sigma I_x I_y uv + \Sigma I_y^2 v^2 = k$$

This is the equation of an ellipse.



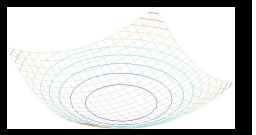
Interpreting the second moment matrix

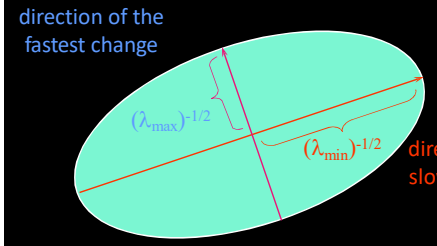
First, consider the axis-aligned case where gradients are either horizontal or vertical

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

If either λ is close to 0, then this is **not** a corner, so look for locations where both are large.

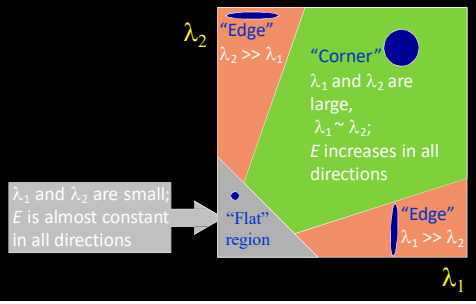
Interpreting the second moment matrix





Interpreting the eigenvalues

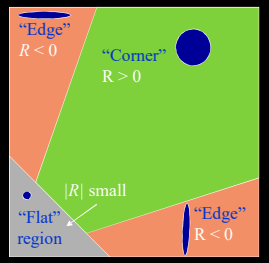
Classification of image points using eigenvalues of *M*:



Harris corner response function

$$R = \det(M) - \alpha\,\mathrm{trace}(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

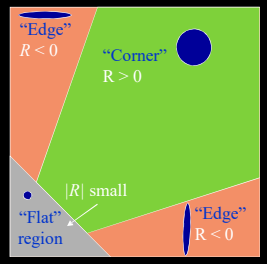
α: constant (0.04 to 0.06)



Harris corner response function

$$R = \det(M) - \alpha\,\mathrm{trace}(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

R depends only on eigenvalues of M, but don't compute them (no sqrt, so really fast even in the '80s).



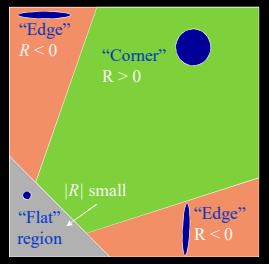
Harris corner response function

$$R = \det(M) - \alpha\,\mathrm{trace}(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

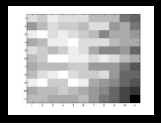
R is large for a corner

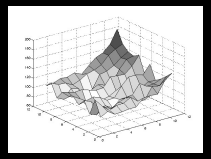
R is negative with large magnitude for an edge

|*R*| is small for a flat region



Low texture region

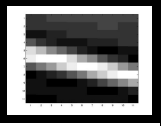


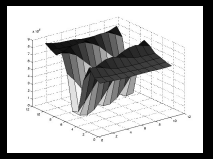


$$M = \sum \nabla I (\nabla I)^T$$

Gradients have small magnitude
=> small λ_1, small λ_2

Edge

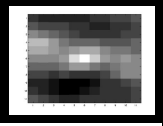


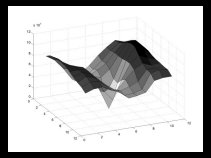


$$M = \sum \nabla I (\nabla I)^T$$

Large gradients, all the same
=> large λ_1, small λ_2

High textured region





$$M = \sum \nabla I (\nabla I)^T$$

Gradients different, large magnitudes
=> large λ_1, large λ_2

Harris Detector: Algorithm

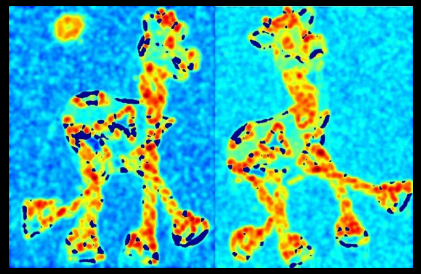
1. Compute derivatives at each pixel
2. Compute second moment matrix M in a Gaussian window around each pixel
3. Compute corner response function R
4. Threshold R
5. Find local maxima of response function (nonmaximum suppression)

C. Harris and M. Stephens. "A Combined Corner and Edge Detector"
Proceedings of the 4th Alvey Vision Conference: pages 147—151, 1988.

Harris Detector: Workflow

Harris Detector: Workflow

Compute corner response *R*

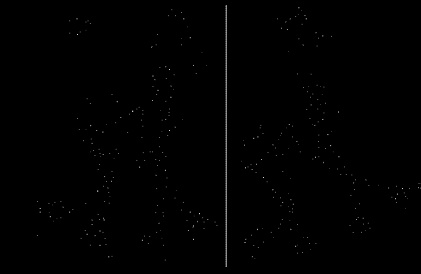


Harris Detector: Workflow

Find points with large corner response: R>threshold

Harris Detector: Workflow

Take only the points of local maxima of *R*



Harris Detector: Workflow

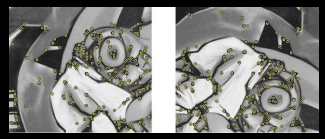
Other corners:

Shi-Tomasi ’94:

"Cornerness" = min (λ_1, λ_2) Find local maximums

cvGoodFeaturesToTrack(...)

Reportedly better for region undergoing affine deformations



Other corners:

- Brown, M., Szeliski, R., and Winder, S. (2005):

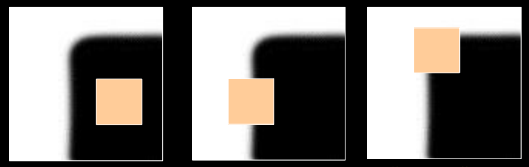
$$\frac{\det M}{\operatorname{tr} M} = \frac{\lambda_0 \lambda_1}{\lambda_0 + \lambda_1}$$

- There are others…

Computer Vision

Scale invariance

Recall: Corner Detection - Basic Idea



"flat" region: no change in all directions

"edge": no change along the edge direction

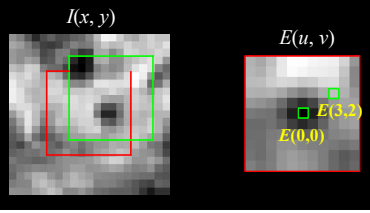
"corner": significant change in all directions with small shift

Source: A. Efros

Corner Detection: Mathematics

Change in appearance for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$



Corner Detection: Mathematics

The quadratic approximation simplifies to

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a *second moment matrix* computed from image derivatives:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

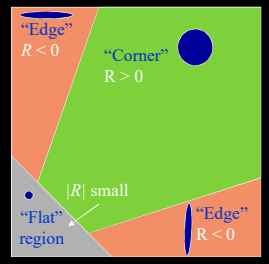
Harris corner response function

$$R = \det(M) - \alpha \,\mathrm{trace}(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

R is large for a corner

R is negative with large magnitude for an edge

$|R|$ is small for a flat region



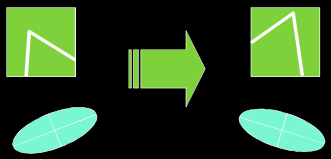
Harris Detector: Workflow

Harris Detector: Some Properties

Rotation invariance?

Harris Detector: Some Properties

Rotation invariance?

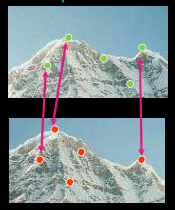


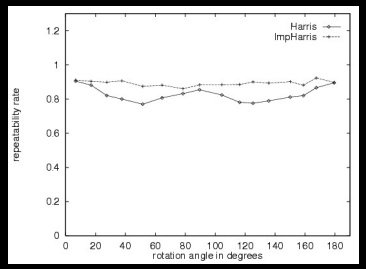
Ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner response R is invariant to image rotation

Repeatability rate:

$$\frac{\text{\# correspondences}}{\text{\# possible correspondences}}$$





C.Schmid et.al. "Evaluation of Interest Point Detectors". Int'l J. Computer Vision, 37(2), 2000

Evaluation plots are from this paper

International Journal of Computer Vision 37(2), 151–172, 2000

Evaluation of Interest Point Detectors

CORDELIA SCHMID, ROGER MOHR AND CHRISTIAN BAUCKHAGE
INRIA Rhône-Alpes, 655 av. de l'Europe, 38330 Montbonnot, France
Cordelia.Schmid@inrialpes.fr

Abstract. Many different low-level feature detectors exist and it is widely agreed that the evaluation of detectors is important. In this paper we introduce two evaluation criteria for interest points: repeatability rate and information content. Repeatability rate evaluates the geometric stability under different transformations. Information content measures the distinctiveness of features. Different interest point detectors are compared using these two criteria. We determine which detector gives the best results and show that it satisfies the criteria well.

Harris Detector: Some Properties

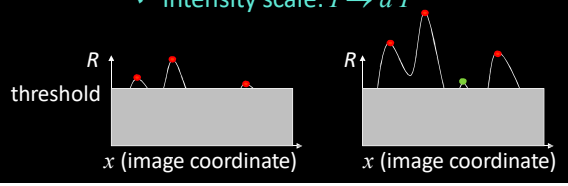
- Invariance to image intensity change?

Harris Detector: Some Properties

- Mostly invariant to additive and multiplicative intensity changes (threshold issue for multiplicative)
  - ✓ Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$

Harris Detector: Some Properties

- Mostly invariant to additive and multiplicative intensity changes (threshold issue for multiplicative)
  - ✓ Intensity scale: $I \rightarrow a\, I$

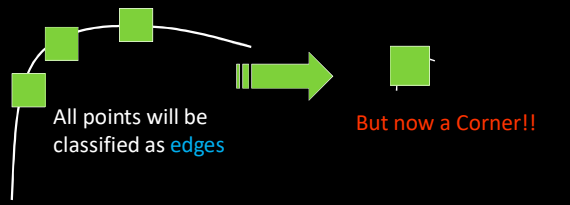


Harris Detector: Some Properties

- Invariant to image scale?

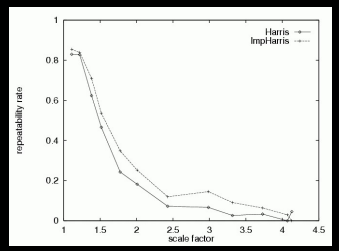
Harris Detector: Some Properties

- Not invariant to *image scale*!



Harris Detector: Some Properties

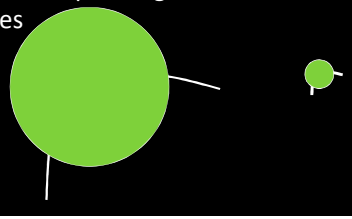
Not invariant to image scale:



IF we want scale invariance…*

Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images



Scale Invariant Detection

- The problem: how do we choose corresponding circles ***independently*** in each image?

Scale Invariant Detection

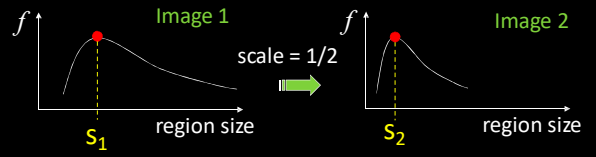
- Solution:
  - Design a function on the region (circle), which is "scale invariant" - not affected by the size but will be the same for "corresponding regions, " even if they are at different sizes/scales.

Example: Average intensity. For corresponding regions (even of different sizes) it will be the same.

Scale Invariant Detection

One method:

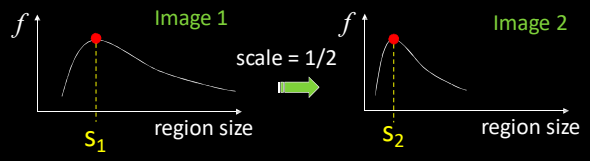
- At a point, compute the scale invariant function over different size neighborhoods (different scales).
- Choose the scale for each image at which the function is a maximum



Scale Invariant Detection

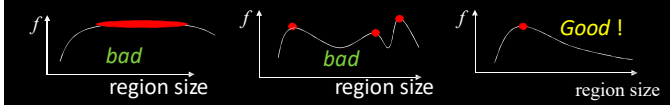
One method:

- Important: this scale invariant region size is found in each image independently



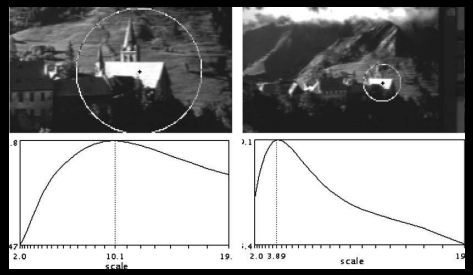
Scale Invariant Detection

- A "good" function for scale detection: has one stable sharp peak



For usual images: a good function would be a one which responds to contrast (sharp local intensity change)

Scale sensitive response



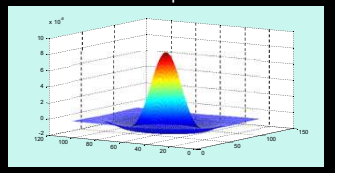
Scale Invariant Detection

Function is just application of a kernel: $f = \mathrm{Kernel} * \mathrm{Image}$

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian of Gaussian - LoG)

Laplacian of Gaussian



$\nabla^2 h_\sigma(u, v)$

Scale Invariant Detection

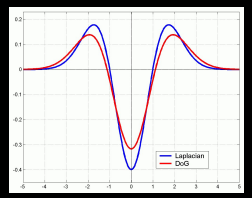
Function is just application of a kernel: $f = \text{Kernel} * \text{Image}$

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian of Gaussian - LoG)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

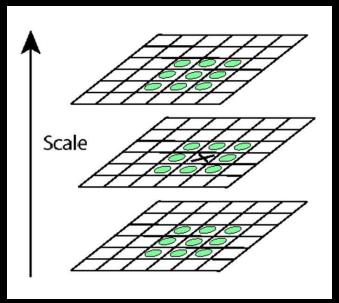
(Difference of Gaussians)



Note: both kernels are invariant to *scale* and *rotation*

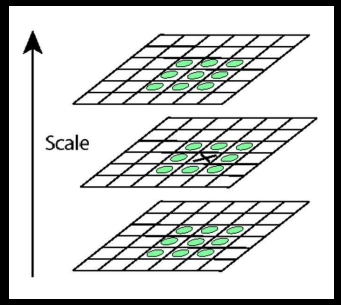
Key point localization

- General idea: find robust extremum (maximum or minimum) both in space and in scale.



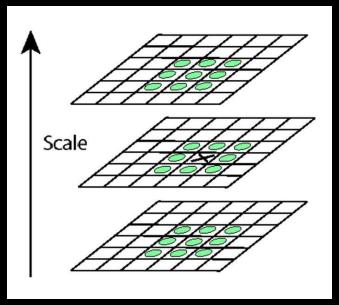
Key point localization

- SIFT: **S**cale **I**nvariant **F**eature **T**ransform
- Specific suggestion: use DoG pyramid to find maximum values (remember edge detection?) – then eliminate "edges" and pick only corners.

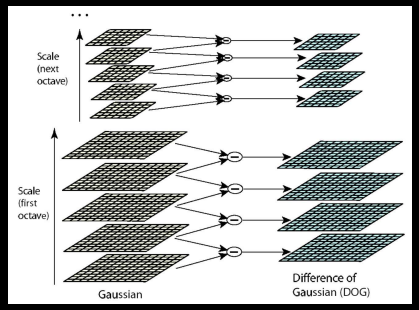


Key point localization

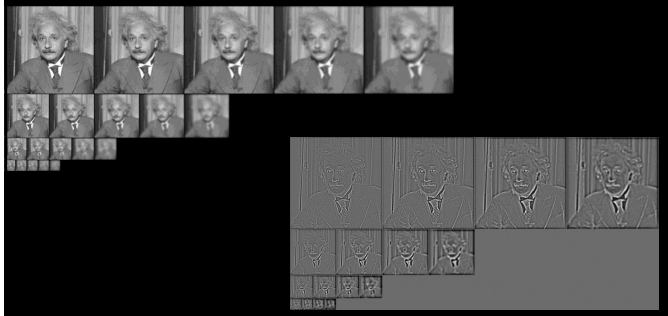
(Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below.)



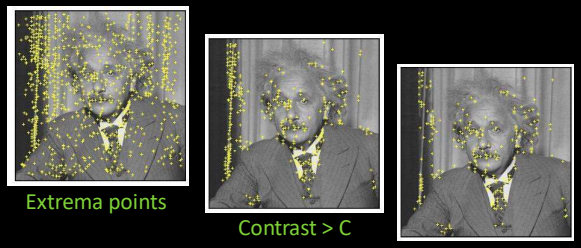
Scale space processed one octave at a time



Extrema at different scales



Remove low contrast, edge bound

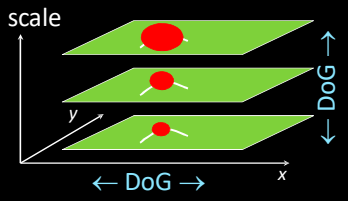


Scale Invariant Detectors

SIFT (Lowe)

Find local maximum of:

– Difference of Gaussians in space and scale



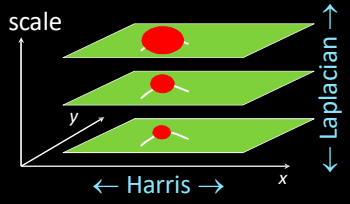
D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". IJCV 2004

Scale Invariant Detectors

Harris-Laplacian

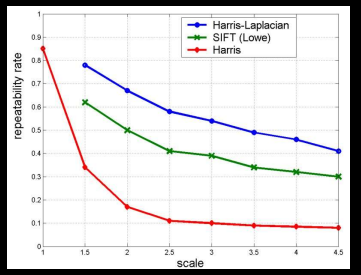
Find local maximum of:

- Harris corner detector in space (image coordinates)
- Laplacian in scale



K.Mikolajczyk, C.Schmid. "Indexing Based on Scale Invariant Interest Points". ICCV 2001

Scale Invariant Detectors



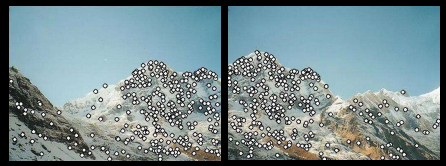
K.Mikolajczyk, C.Schmid. "Indexing Based on Scale Invariant Interest Points". ICCV 2001

Scale Invariance: Summary

- Given: Two images of the same scene with a *large* scale difference between them
- Goal: Find the same interest points independently in each image
- Solution: Search for maxima of suitable functions in scale and in space (over the image)

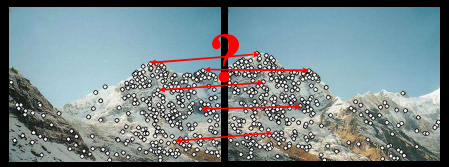
Point Descriptors

- We now know how to detect interest points



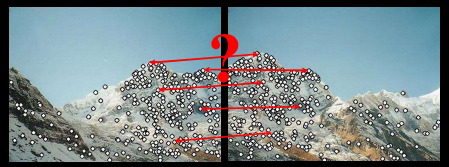
Point Descriptors

- Next question: How to match them?



Point Descriptors

- We need to describe them – a "*descriptor*"

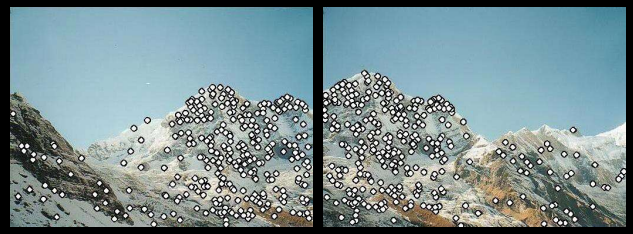


…Next!

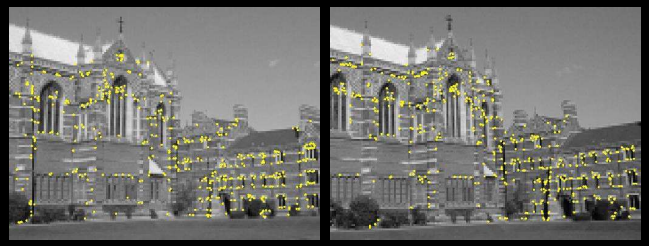
SIFT descriptor

Point Descriptors

- Last time: How to detect interest points



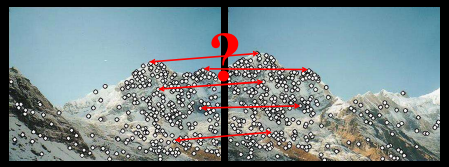
Harris detector



Interest points extracted with Harris (~ 500 points)

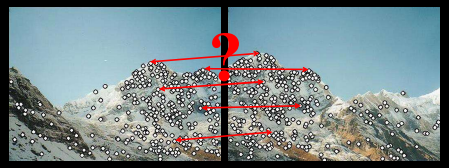
Point Descriptors

- Now: How to match them?



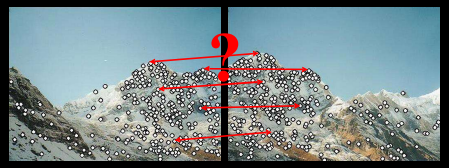
Point Descriptors

- We need to describe them – a "*descriptor*"



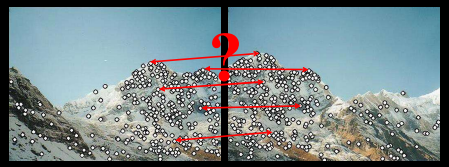
Criteria for Point Descriptors

- We want the descriptors to be the (almost) same in both image – *invariant.*



Criteria for Point Descriptors

- We also need the descriptors to be *distinctive*.



Simple solution?

- Harris gives good detection – and we also know the scale.
- Why not just use correlation to check the match of the window around the feature in image 1 with every feature in image 2?

Simple solution? *Not so good!*

- Not so good because:
  - Correlation is not rotation invariant - why do we want this?
  - Correlation is sensitive to photometric changes.
  - Normalized correlation is sensitive to non-linear photometric changes and even slight geometric ones.
  - Could be slow – check all features against all features

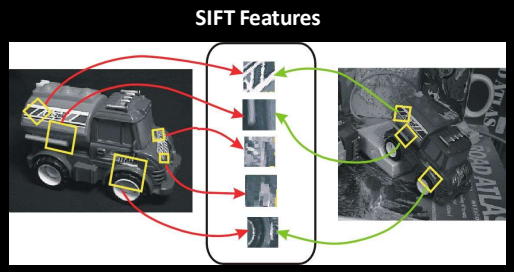
SIFT: Scale Invariant Feature Detection

- Motivation: The Harris operator was not invariant to scale and correlation was not invariant to rotation.
- For better image matching, Lowe's goals were:
  - To develop an interest operator – a *detector* – that is invariant to scale and rotation.
  - Also: create a descriptor that was robust to the variations corresponding to typical viewing conditions. *The descriptor is the most-used part of SIFT.*

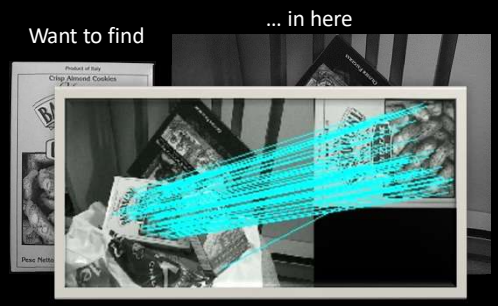
Idea of SIFT

- Image content is represented by a constellation of local features that are invariant to translation, rotation, scale, and other imaging parameters

Idea of SIFT



Another version of the problem…



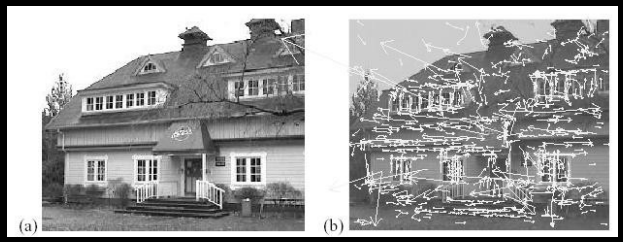
Overall SIFT Procedure

- Scale-space extrema detection
- Keypoint localization

Or use Harris-Laplace or other method

- Orientation assignment
- Keypoint description

Example of keypoint detection



(a) 233x189 image
(b) 832 DOG extrema

Overall SIFT Procedure

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment

   Compute best orientation(s) for each keypoint region.
4. Keypoint description

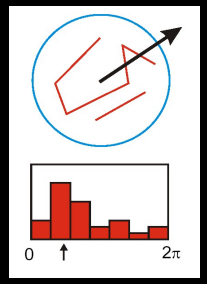
   Use local image gradients at selected scale and rotation to describe each keypoint region.

Descriptors Invariant to Rotation

- Find the dominant direction of gradient – that is the base orientation.

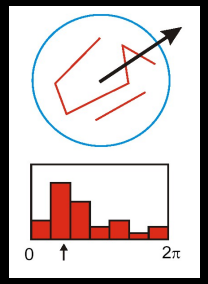
Compute image derivatives relative to this orientation

Orientation assignment



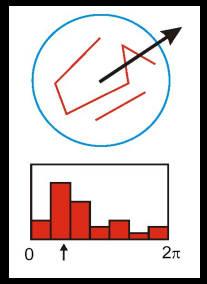
- Create histogram of local gradient directions at *selected* scale – 36 bins

Orientation assignment



- Assign canonical orientation at peak of smoothed histogram

Orientation assignment



- Each *keypoint* now specifies stable 2D coordinates (x, y, scale, orientation) – invariant to those.

4. Keypoint Descriptors

- Next is to compute a descriptor for the local image region about each keypoint that is:
  - Highly ***distinctive***
  - As ***invariant*** as possible to variations such as changes in viewpoint and illumination

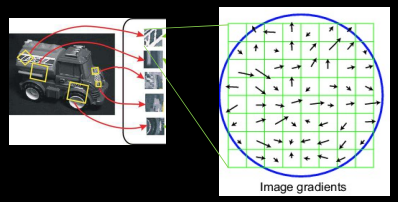
But first… normalization…

- Rotate the window to standard orientation
- Scale the window size based on the scale at which the point was found.

SIFT vector formation

Compute a feature vector based upon:

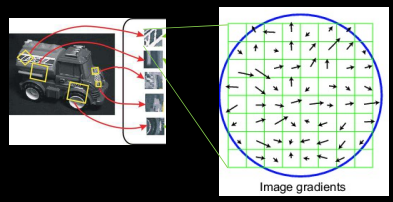
- histograms of gradients



SIFT vector formation

Compute a feature vector based upon:

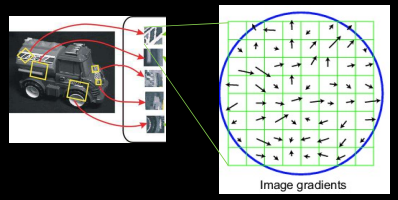
- weighted by a centered Gaussian,



SIFT vector formation

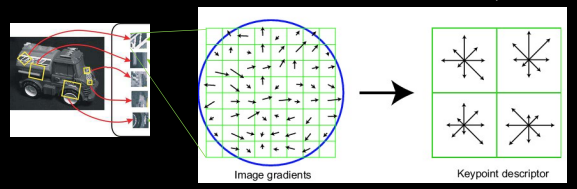
Compute a feature vector based upon:

- weighted by the magnitude of the gradient

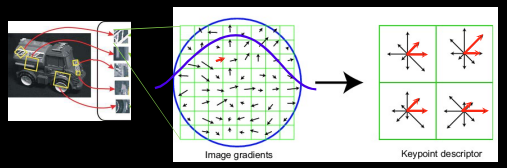


SIFT vector formation

showing only 2x2 here but it really is 4x4

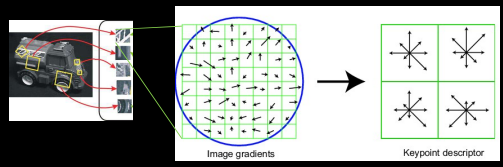


Ensure smoothness



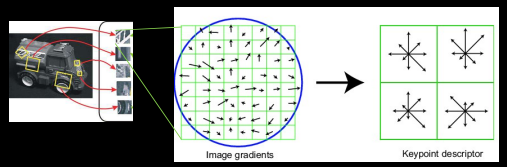
Reduce effect of illumination

- Clip gradient magnitudes to avoid excessive influence of high gradients
  - after rotation normalization, clamp gradients >0.2



Reduce effect of illumination

- 128-dim vector normalized to magnitude 1.0



Evaluating the SIFT descriptors

- Database images were subjected to rotation, scaling, affine stretch, brightness and contrast changes, and added noise.
- Feature point detectors and descriptors were compared before and after the distortions.
- Mostly looking for stability with respect to change.

Sensitivity to parameters

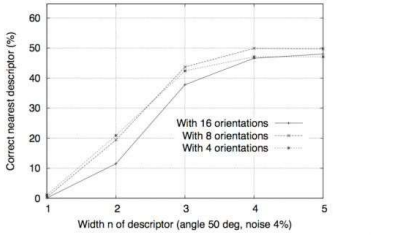
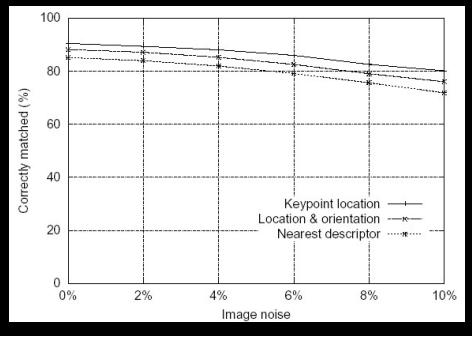


Figure 8: This graph shows the percent of keypoints giving the correct match to a database of 40,000 keypoints as a function of width of the $n \times n$ keypoint descriptor and the number of orientations in each histogram. The graph is computed for images with affine viewpoint change of 50 degrees and addition of 4% noise.

Feature stability to noise

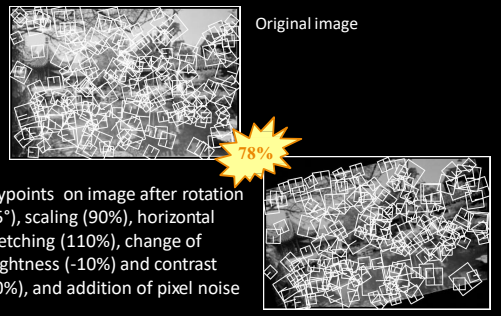


Experimental results - summary

Image transformation	Location and scale match	Orientation match
Decrease constrast by 1.2	89.0 %	86.6 %
Decrease intensity by 0.2	88.5 %	85.9 %
Rotate by 20°	85.4 %	81.0 %
Scale by 0.7	85.1 %	80.3 %
Stretch by 1.2	83.5 %	76.1 %
Stretch by 1.5	77.7 %	65.0 %
Add 10% pixel noise	90.3 %	88.4 %
All previous	**78.6 %**	**71.8 %**

20 different images, around 15,000 keys

Experimental results

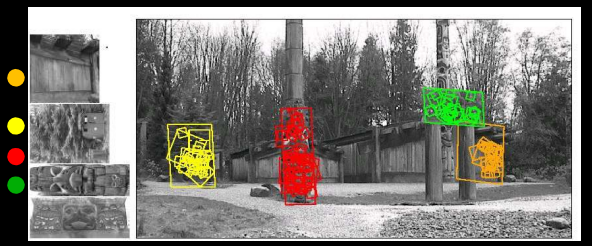


Original image

Keypoints on image after rotation (15˚), scaling (90%), horizontal stretching (110%), change of brightness (-10%) and contrast (90%), and addition of pixel noise

SIFT matching object pieces (for location)

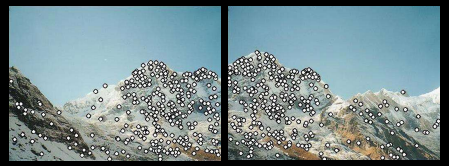
SIFT matching object pieces (for location)



Matching feature points (a little)

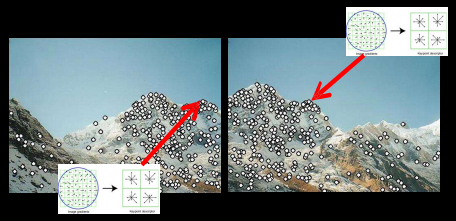
Feature Points

- We know how to detect points



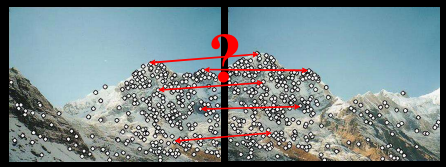
Feature Points

- We know how to describe them



Feature Points

- Next question: How to match them?



How to match feature points?

- Could just do nearest-neighbor search
  - [OMS students: You will!]
- But that's really expensive…SIFT tests have 10,000's of points!

Nearest-neighbor matching to feature database

- Better: Hypotheses are generated by *approximate nearest neighbor* matching of each feature to vectors in the database
  - SIFT uses best-bin-first (Beis & Lowe, 97) modification to k-d tree algorithm
  - Use heap data structure to identify bins in order by their distance from query point

Nearest-neighbor matching to feature database

- Result: Can give speedup by factor of *100-1000* while finding nearest neighbor (of interest) 95% of the time

Nearest neighbor techniques

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

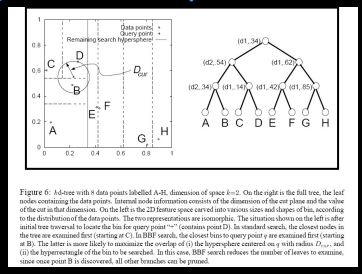
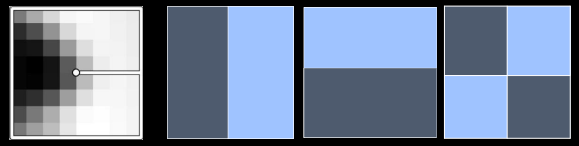


Figure 6: A *k*-d tree with 8 data points labelled A-H, dimension of space *k*=2. On the right is the full tree, the leaf nodes containing the data points. Internal node information consists of the dimension of the cut plane and the value of the cut in that dimension. On the left is the 2D feature space 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 minimize 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

Wavelet-based hashing

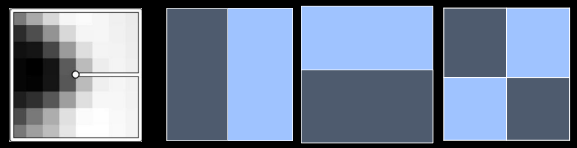
Compute a short (3-vector) descriptor from the neightborhood using a Haar "wavelet"



[Brown, Szeliski, Winder, CVPR'2005]

Wavelet-based hashing

Quantize each value into 10 (overlapping) bins (10^3 total entries)



[Brown, Szeliski, Winder, CVPR'2005]

3D Object Recognition

Train:

1. Extract outlines with background subtraction
2. Compute "keypoints" – interest points and descriptors.

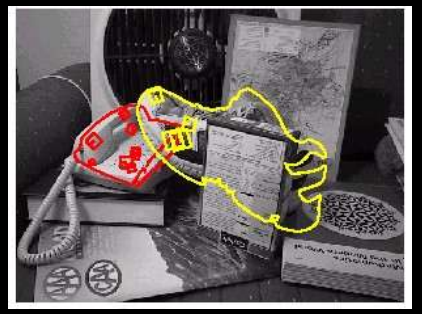
3D Object Recognition

Test:

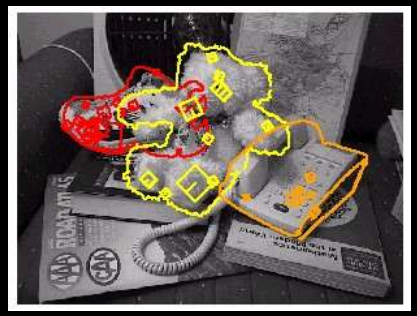
1. Find possible matches.
2. Search for consistent solution – such as *affine*. *(How many points?!?!?)*

Results

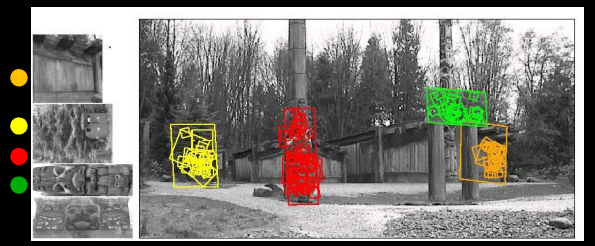
Recognition under occlusion



Recognition under occlusion



Locating object pieces



(From last lesson)

SIFT in Sony Aibo (Evolution Robotics)

SIFT usage:

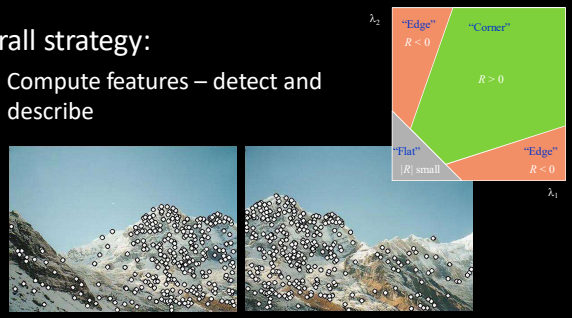
- Recognize charging station
- Communicate with visual cards

Robust error functions

Feature-based alignment to find transforms

Overall strategy:

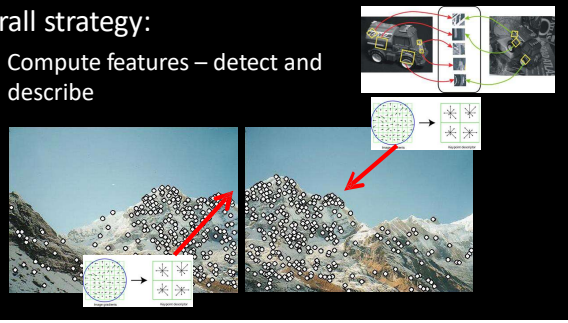
1. Compute features – detect and describe



Feature-based alignment to find transforms

Overall strategy:

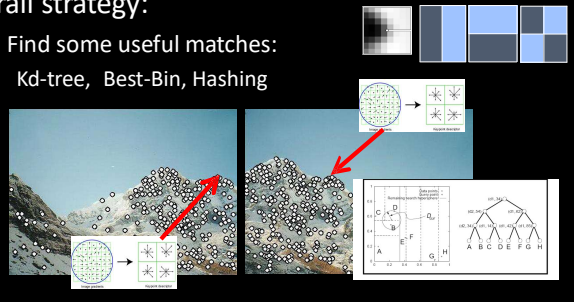
1. Compute features – detect and describe



Feature-based alignment to find transforms

Overall strategy:

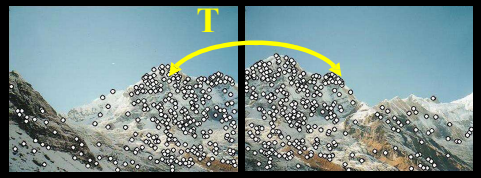
2. Find some useful matches:
   Kd-tree, Best-Bin, Hashing



Feature-based alignment to find transforms

Overall strategy:

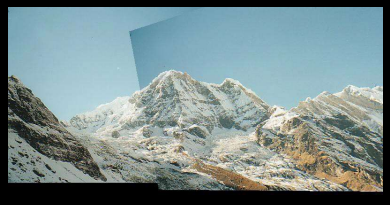
3. Compute and apply the best transformation:
   e.g. affine, translation, or homography



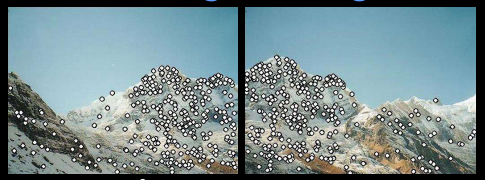
Feature-based alignment to find transforms

Overall strategy:

3. Compute and apply the best transformation:
   e.g. affine, translation, or homography



Feature-based alignment algorithm

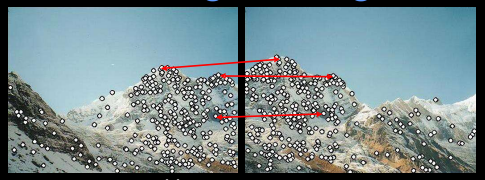


1. Extract features

Feature-based alignment algorithm

2. Compute ***putative*** *matches* – e.g. "closest descriptor"
   Kd-tree, best bin, etc…

Feature-based alignment algorithm



3. Loop until happy:
   - *Hypothesize* transformation *T* from some matches
   - *Verify* transformation (search for other matches consistent with *T*) – mark best

Feature-based alignment algorithm

4. Apply best transformation.

How to get “putative” matches?

Feature matching

- Exhaustive search
- Hashing
- Nearest neighbor techniques

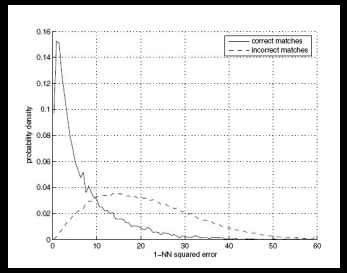
…. but these give the best match. How do we know it’s a good one?

Feature-space outlier rejection

- Let’s not match all features, but only these that have “similar enough” matches?
- How can we do it?
  - SSD(patch1,patch2) < threshold
  - How to set threshold?

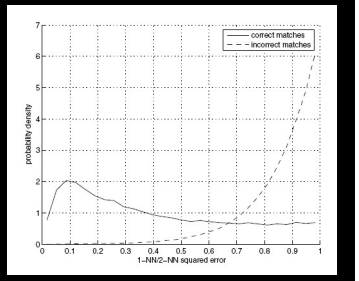
Feature-space outlier rejection

- How to set threshold?



A better way [Lowe, 1999]:

- 1-NN: SSD of the closest match
- 2-NN: SSD of the second-closest match
- Look at how much better 1-NN is than 2-NN, e.g. 1-NN/2-NN



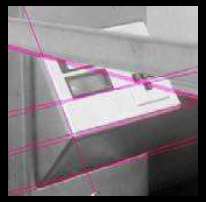
Feature matching

- Exhaustive search
- Hashing
- Nearest neighbor techniques
- But…remember the distinctive vs invariant competition? Implies:

- *Problem: Even when pick best match, still lots (and lots) of wrong matches – "outliers". What should we do?*

Model Fitting

- Choose a parametric model to represent a set of features – *remember this???*



simple model: lines

simple model: circles

Fitting: Issues

Case study: Line detection

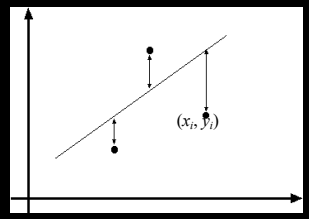
- **Noise** in the measured feature locations
- **Extraneous data**: clutter (outliers), multiple lines
- **Missing data**: occlusions

Slide: S. Lazebnik

Typical least squares line fitting

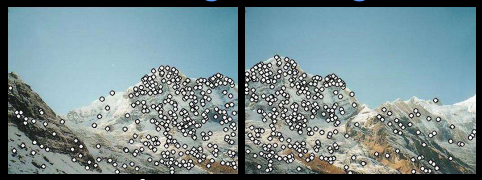
- Data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Line equation: $y_i = m\,x_i + b$
- Find (m, b) to minimize:

$$E = \sum_{i=1}^{n} (y_i - m\,x_i - b)^2$$



RANSAC

Feature-based alignment algorithm



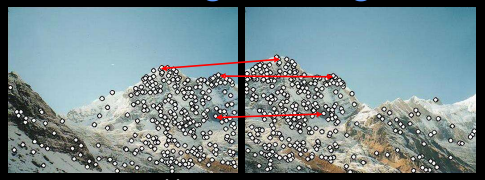
1. Extract features

Feature-based alignment algorithm

2. Compute ***putative*** *matches* – e.g. "closest descriptor"

   Kd-tree, best bin, etc…

Feature-based alignment algorithm

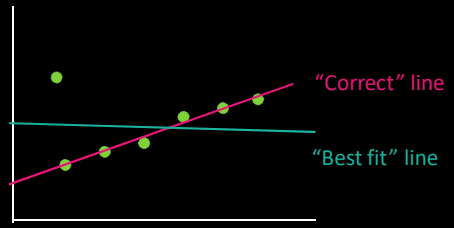


3. Loop until happy:
   - *Hypothesize* transformation *T* from some matches
   - *Verify* transformation (search for other matches ***consistent*** with *T*) – mark best

"Find consistent matches"?

- Some "best" matches are correct
- Some are not. And the "not" are not part of any other consistent match…
- Need to find the right ones so can compute the pose/transform/fundamental… *the model.*
- Today*:* Random Sample Consensus (RANSAC)

Simple Example: Fitting a line



RANSAC: Main idea

- Fitting a line (model) is easy if we know which points belong and which do not. (duh…)
- If we had a proposed line (model), we could probably guess which points belong to that line (model): *inliers*.
- RANdom SAmple Consensus: randomly pick some points to define your line (model). Repeat enough times until you find a good line (model) – on with many inliers.
- Fischler & Bolles 1981 – Copes with a large proportion of outliers

RANSAC

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of ***inliers*** within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence

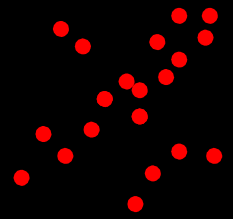
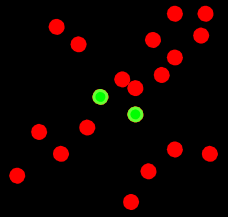


Illustration by Savarese

RANSAC

Algorithm:

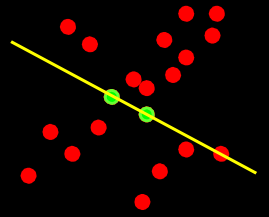
1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of ***inliers*** within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence



RANSAC

Algorithm:

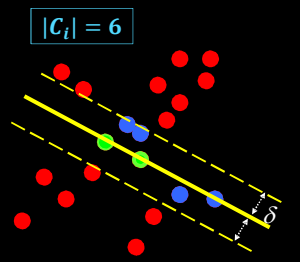
1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of ***inliers*** within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence



RANSAC

Algorithm:

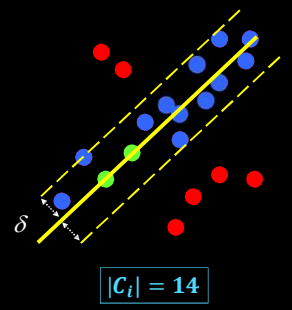
1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of ***inliers*** within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence



RANSAC

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of ***inliers*** within a preset threshold of the model
4. **Repeat** 1-3 until the best model is found with high confidence



RANSAC for general model

A given model type has a *minimal set* – the smallest number of samples from which the model can be computed.

- Line: 2 points

RANSAC for general model

Image transformations are models. Minimal set of point pairs/matches:

- Translation: pick one pair of matched points
- Homography (for plane) – pick 4 point pairs
- Fundamental matrix – pick 8 point pairs (really 7 but lets not go there)

RANSAC for general model

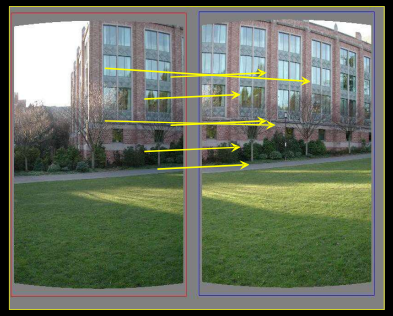
General RANSAC algorithm

- Randomly select Spoints (or point pairs) to form a *sample*
- Instantiate the model
- Get consensus set C_i : The points within error bounds (distance threshold) of the model
- If $|C_i| > T$, terminate and return model
- Repeat for N trials, return model with max $|C_i|$

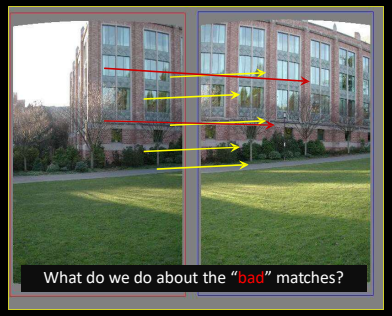
Choosing the parameters

1. Initial number of points in the minimal set s
   - Typically minimum number needed to fit the model
2. Distance threshold t

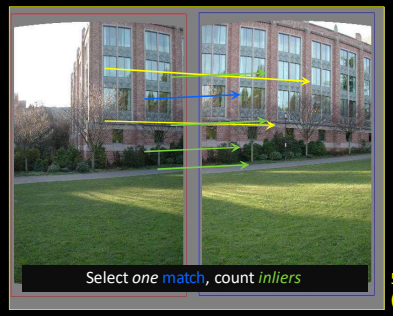
Matching features



Matching features

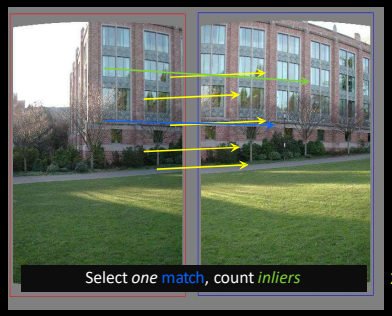


RAndom SAmple Consensus (1)



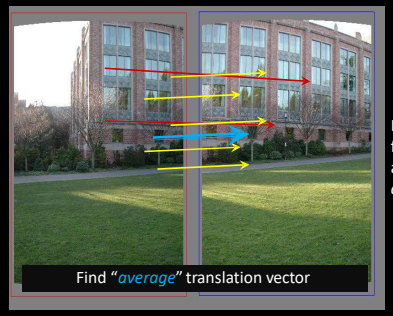
5 total inliers
(including selected)

RAndom SAmple Consensus (2)



2 total inliers

Least squares fit



Note: We use the first set of matches as it had stronger *consensus* (5 inliers)

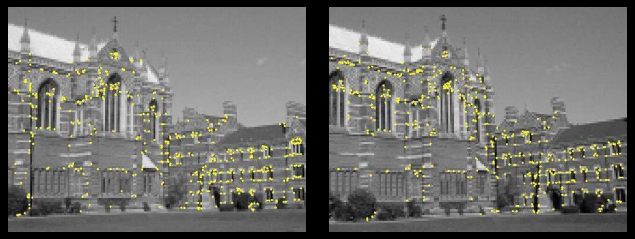
RANSAC for estimating homography

RANSAC loop:

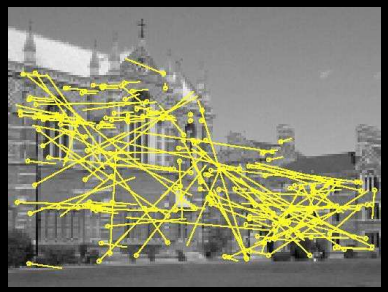
1. Select four feature pairs (at random)
2. Compute homography $\boldsymbol{H}$ (exact)
3. Compute *inliers* where $\mathrm{SSD}(\mathrm{p}', \boldsymbol{H}\,\mathrm{p}i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares $\boldsymbol{H}$ estimate on all of the inliers

RANSAC for recognition

RANSAC for finding fundamental matrix



Putative matches (motion) by cross-correlation

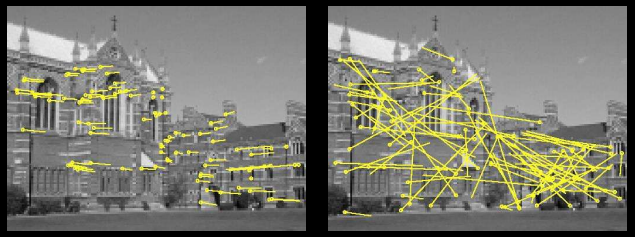


188 matches

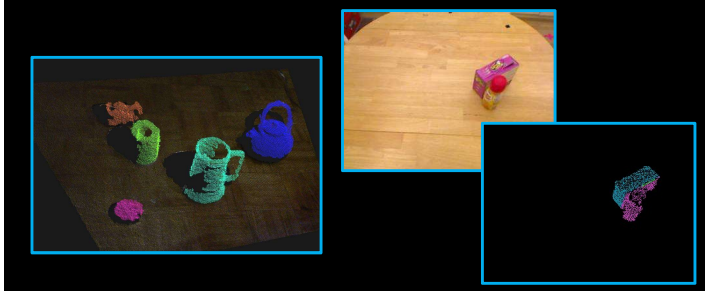
RANSAC for fundamental matrix

Inliers (99)

Outliers (89)



Point cloud planes



Find the plane and object in realtime

RANSAC: Conclusions

The good…

- Simple and general
- Applicable to many different problems, often works well in practice
- Robust to large numbers of outliers
- Applicable for larger number of parameters than Hough transform
- Parameters are easier to choose than Hough transform

RANSAC: Conclusions

The not-so-good…

- Computational time grows quickly with the number of model parameters
- Not as good for getting multiple fits
- Really not good for approximate models

RANSAC: Conclusions

Common applications

- Computing a homography (e.g., image stitching) or other image transform
- Estimating fundamental matrix (relating two views)
- *Pretty much every problem in robot vision*