Image matching

by Diva Sian

by swashford

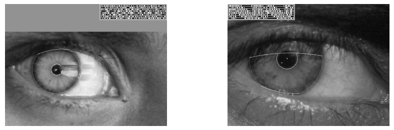
Harder case

by Diva Sian

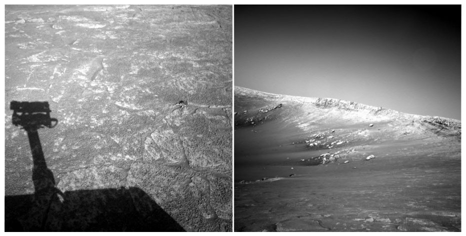
by scgbt

Even harder case

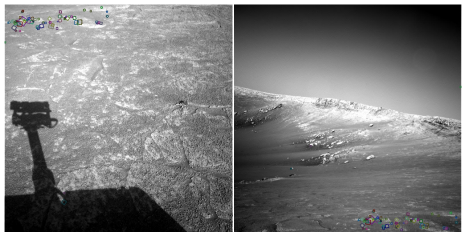
"How the Afghan Girl was Identified by Her Iris Patterns" **Read the story**



Harder still?


NASA Mars Rover images

Answer below (look for tiny colored squares…)


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

Features

All is Vanity, by C. Allan Gilbert, 1873-1929

Reading

- M. Brown et al. Multi-Image Matching using Multi-Scale Oriented Patches, CVPR 2005

Image Matching

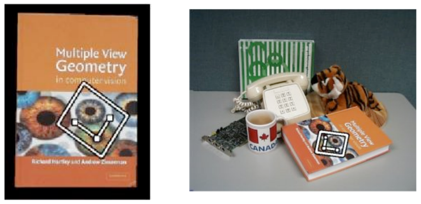
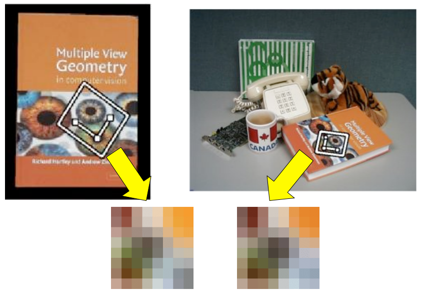
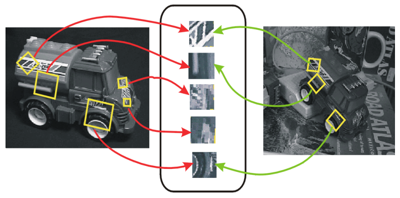


Image Matching



Invariant local features

Find features that are invariant to transformations
- geometric invariance: translation, rotation, scale
- photometric invariance: brightness, exposure, …


Feature Descriptors

Advantages of local features

Locality
- features are local, so robust to occlusion and clutter

Distinctiveness:
- can differentiate a large database of objects

Quantity
- hundreds or thousands in a single image

Efficiency
- real-time performance achievable

More motivation…

Feature points are used for:
- Image alignment (e.g., panoramas)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- … many others

What makes a good feature?

Snoop demo

Want uniqueness

Look for image regions that are unusual

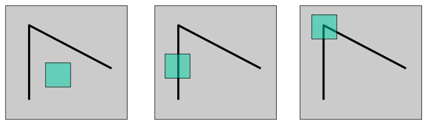
- Lead to unambiguous matches in other images

How to define “unusual”?

Local measures of uniqueness

Suppose we only consider a small window of pixels

- What defines whether a feature is a good or bad candidate?



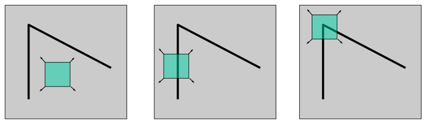
Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

High level idea is that corners are good. You want to find windows that contain strong gradients AND gradients oriented in more than one direction

Feature detection

Local measure of feature uniqueness

- How does the window change when you shift by a *small amount*?



“flat” region:
no change in all directions

“edge”:
no change along the edge direction

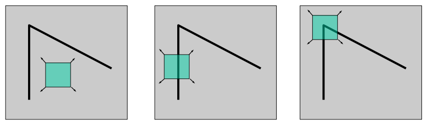
“corner”:
significant change in all directions

Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

Feature detection

Define

E(u,v) = amount of change when you shift the window by (u,v)



E(u,v) is small for **all** shifts

E(u,v) is small for **some** shifts

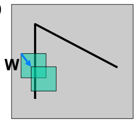
E(u,v) is small for **no** shifts

We want $\min_{(u,v)} E(u,v)$ to be ______

Feature detection: the math

Consider shifting the window **W** by (u,v)

- how do the pixels in **W** change?
- compare each pixel before and after by Sum of the Squared Differences (SSD)
- this defines an SSD "error" *E(u,v)*:



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

Small motion assumption

Taylor Series expansion of I:

$$I(x+u,y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approx is good

$$I(x+u,y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

$$\approx I(x,y) + [I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix}$$

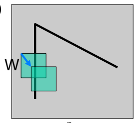
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide…

Feature detection: the math

Consider shifting the window W by (u,v)

- how do the pixels in W change?
- compare each pixel before and after by summing up the squared differences
- this defines an "error" of E(u,v):

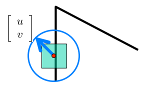


$$\begin{aligned} E(u,v) &= \sum_{(x,y)\in W} [I(x+u,y+v) - I(x,y)]^2 \\ &\approx \sum_{(x,y)\in W} \left[I(x,y) + [I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix} - I(x,y)\right]^2 \\ &\approx \sum_{(x,y)\in W} \left[[I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix}\right]^2 \end{aligned}$$

Feature detection: the math

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u \; v] \underbrace{\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$



For the example above

- You can move the center of the green window to anywhere on the blue unit circle
- Which directions will result in the largest and smallest E values?
- We can find these directions by looking at the eigenvectors of ***H***

Quick eigenvalue/eigenvector review

The **eigenvectors** of a matrix **A** are the vectors **x** that satisfy:

$$Ax = \lambda x$$

The scalar λ is the **eigenvalue** corresponding to **x**

- The eigenvalues are found by solving:

$$det(A - \lambda I) = 0$$

- In our case, ***A = H*** is a 2x2 matrix, so we have

$$det \begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} = 0$$

- The solution:

$$\lambda_{\pm} = \tfrac{1}{2} \left[(h_{11} + h_{22}) \pm \sqrt{4 h_{12} h_{21} + (h_{11} - h_{22})^2} \right]$$

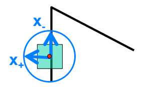
Once you know λ, you find **x** by solving

$$\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

Feature detection: the math

This can be rewritten:

$$E(u,v) = [u \ v] \underbrace{\left(\sum_{(x,y) \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \right)}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$



Eigenvalues and eigenvectors of H

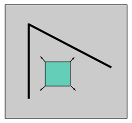
- Define shifts with the smallest and largest change (E value)
- x_+ = direction of **largest** increase in E.
- λ_+ = amount of increase in direction x_+
- x_- = direction of **smallest** increase in E.
- λ- = amount of increase in direction x_+

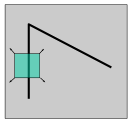
$$\begin{aligned} Hx_+ &= \lambda_+ x_+ \\ Hx_- &= \lambda_- x_- \end{aligned}$$

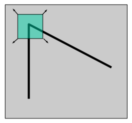
Feature detection

Local measure of feature uniqueness

- E(u,v) = amount of change when you shift the window by (u,v)


E(u,v) is small for **all** shifts

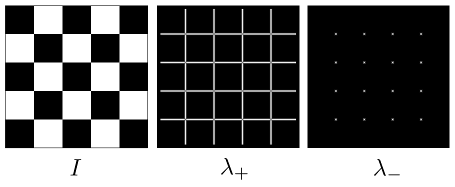

E(u,v) is small for **some** shifts


E(u,v) is small for **no** shifts

We want $\min_{(u,v)} E(u,v)$ to be large

$\min_{(u,v)} E(u,v)$ = ?

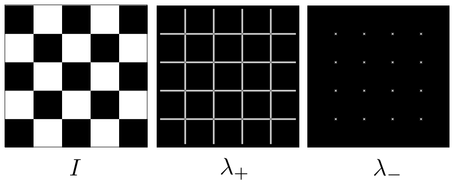
Eigenvalues of **H**



Feature detection summary

Here's what you do

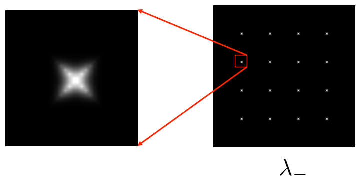
- Compute the gradient at each point in the image
- Create the ***H*** matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ- > threshold)
- Choose those points where λ- is a local maximum as features



Feature detection summary

Here's what you do

- Compute the gradient at each point in the image
- Create the ***H*** matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ- > threshold)
- Choose those points where λ- is a local maximum as features



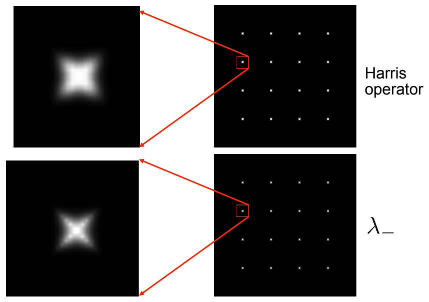
The Harris operator

λ- is a variant of the "Harris operator" for feature detection

$$f = \frac{\lambda_-\lambda_+}{\lambda_- + \lambda_+} = \frac{determinant(H)}{trace(H)}$$

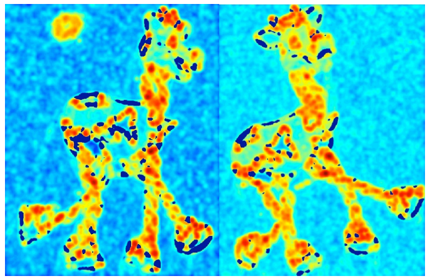
- The *trace* is the sum of the diagonals, i.e., *trace(H)* = $h_{11} + h_{22}$
- Very similar to λ- but less expensive (no square root)
- Called the "Harris Corner Detector" or "Harris Operator"
- Lots of other detectors, this is one of the most popular

The Harris operator

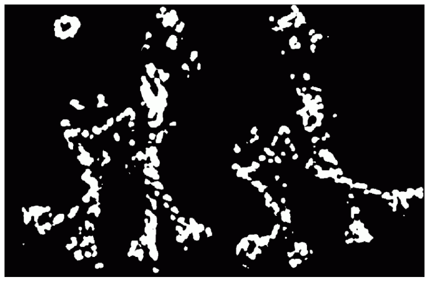


Harris detector example

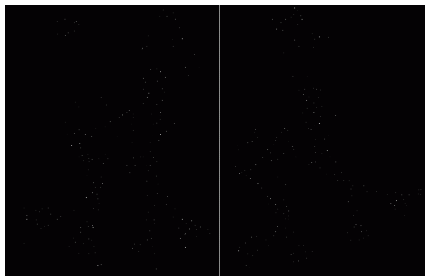
f value (red high, blue low)



Threshold (f > value)



Find local maxima of f



Harris features (in red)

Invariance

Suppose you **rotate** the image by some angle

- Will you still pick up the same features?

What if you change the brightness?

Scale?

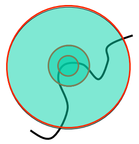
Rotation: yes (with some caveats—image resampling errors…)

Change brightness: probably, but will have to adjust thresholds

Scale: no, generally.

Scale invariant detection

Suppose you're looking for corners



Key idea: find scale that gives local maximum of f

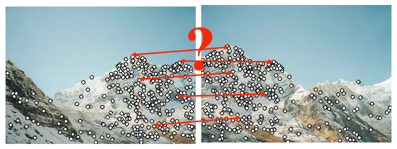
- f is a local maximum in both position and scale
- Common definition of f: Laplacian
  (or difference between two Gaussian filtered images with different sigmas)

Show other scales and how the feature looks less corner like

Feature descriptors

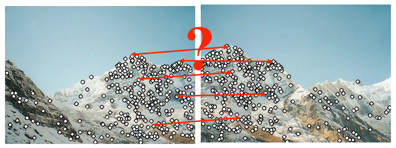
We know how to detect good points

Next question: **How to match them?**



Feature descriptors

We know how to detect good points
Next question: **How to match them?**



Lots of possibilities (this is a popular research area)

- Simple option: match square windows around the point
- State of the art approach: SIFT
  - David Lowe, UBC http://www.cs.ubc.ca/~lowe/keypoints/

Invariance

Suppose we are comparing two images I_1 and I_2

- I_2 may be a transformed version of I_1
- What kinds of transformations are we likely to encounter in practice?

Invariance

Suppose we are comparing two images I_1 and I_2

- I_2 may be a transformed version of I_1
- What kinds of transformations are we likely to encounter in practice?

We'd like to find the same features regardless of the transformation

- This is called transformational ***invariance***
- Most feature methods are designed to be invariant to
  - Translation, 2D rotation, scale
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transformations (some are fully affine invariant)
  - Limited illumination/contrast changes

How to achieve invariance

Need both of the following:

1. Make sure your detector is invariant
   - Harris is invariant to translation and rotation
   - Scale is trickier
     - SIFT uses automatic scale selection (previous slides)
     - simpler approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS) and add them all to database
2. Design an invariant feature *descriptor*
   - A descriptor captures the information in a region around the detected feature point
   - The simplest descriptor: a square window of pixels
     - What's this invariant to?
   - Let's look at some better approaches…

Rotation invariance for feature descriptors

Find dominant orientation of the image window

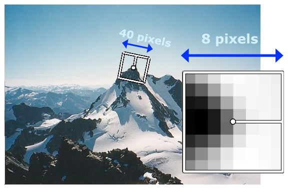
- This is given by $\mathbf{x}_+$, the eigenvector of $\mathbf{H}$ corresponding to λ_+
  - λ_+ is the *larger* eigenvalue
- Rotate the window according to this angle

Figure by Matthew Brown

Multiscale Oriented PatcheS descriptor

Take 40x40 square window around detected feature

- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window



Adapted from slide by Matthew Brown

Detections at multiple scales

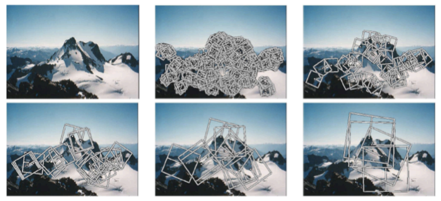
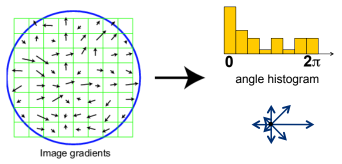


Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

Scale Invariant Feature Transform

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

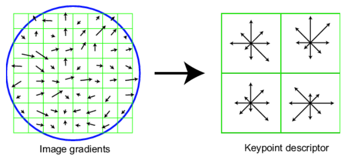


Adapted from slide by David Lowe

SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

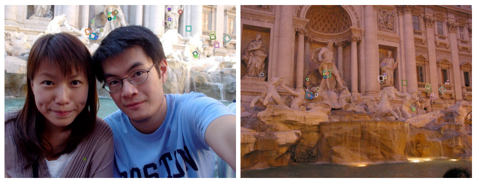


Adapted from slide by David Lowe

Properties of SIFT

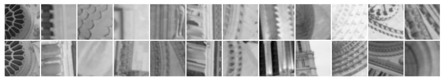
Extraordinarily robust matching technique

- Can handle changes in viewpoint
  - Up to about 60 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available
  - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT



When does SIFT fail?

Patches SIFT thought were the same but aren't:



Feature matching

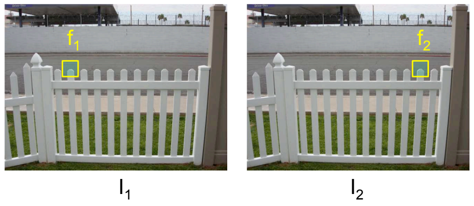
Given a feature in I_1, how to find the best match in I_2?

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

Feature distance

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

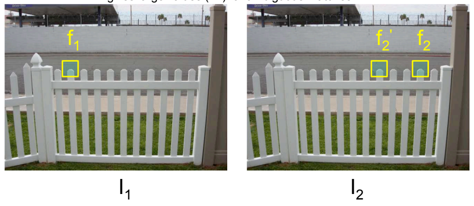
- Simple approach is SSD(f_1, f_2)
  - sum of square differences between entries of the two descriptors
  - can give good scores to very ambiguous (bad) matches



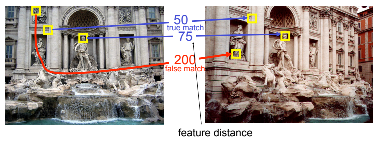
Feature distance

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

- Better approach: ratio distance = SSD(f_1, f_2) / SSD(f_1, f_2')
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2nd best SSD match to f_1 in I_2
  - gives large values (~1) for ambiguous matches



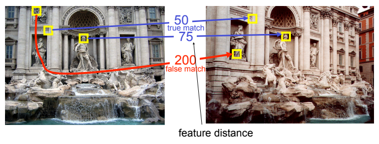
Eliminating bad matches



Throw out features with distance > threshold

- How to choose the threshold?

True/false positives



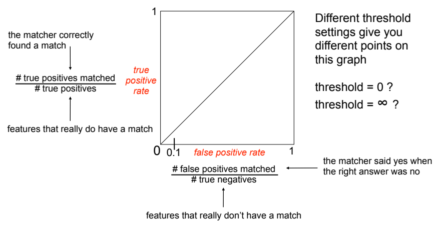
Throw out features with distance > threshold

The threshold affects performance

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

Evaluating the results

How can we measure the performance of a feature matcher?



the matcher correctly found a match
# true positives matched / # true positives
features that really do have a match
true positive rate
1
0
0.1
false positive rate
1
# false positives matched / # true negatives
the matcher said yes when the right answer was no
features that really don't have a match

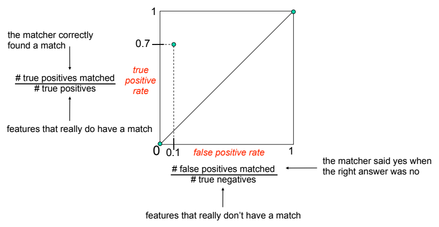
Different threshold settings give you different points on this graph

threshold = 0 ?

threshold = ∞ ?

Evaluating the results

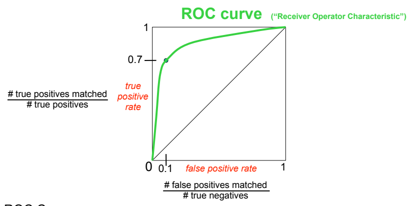
How can we measure the performance of a feature matcher?



the matcher correctly found a match
# true positives matched / # true positives
features that really do have a match
true positive rate
1
0.7
0
0.1
false positive rate
1
# false positives matched / # true negatives
the matcher said yes when the right answer was no
features that really don't have a match

Evaluating the results

How can we measure the performance of a feature matcher?



ROC Curves

- Generated by counting # current/incorrect matches, for different threholds
- Want to maximize area under the curve (AUC)
- Useful for comparing different feature matching methods
- For more info: http://en.wikipedia.org/wiki/Receiver_operating_characteristic

Tons of applications

Features are used for:

- Image alignment (e.g., panoramas)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- …