B. Feature Extraction and Matching

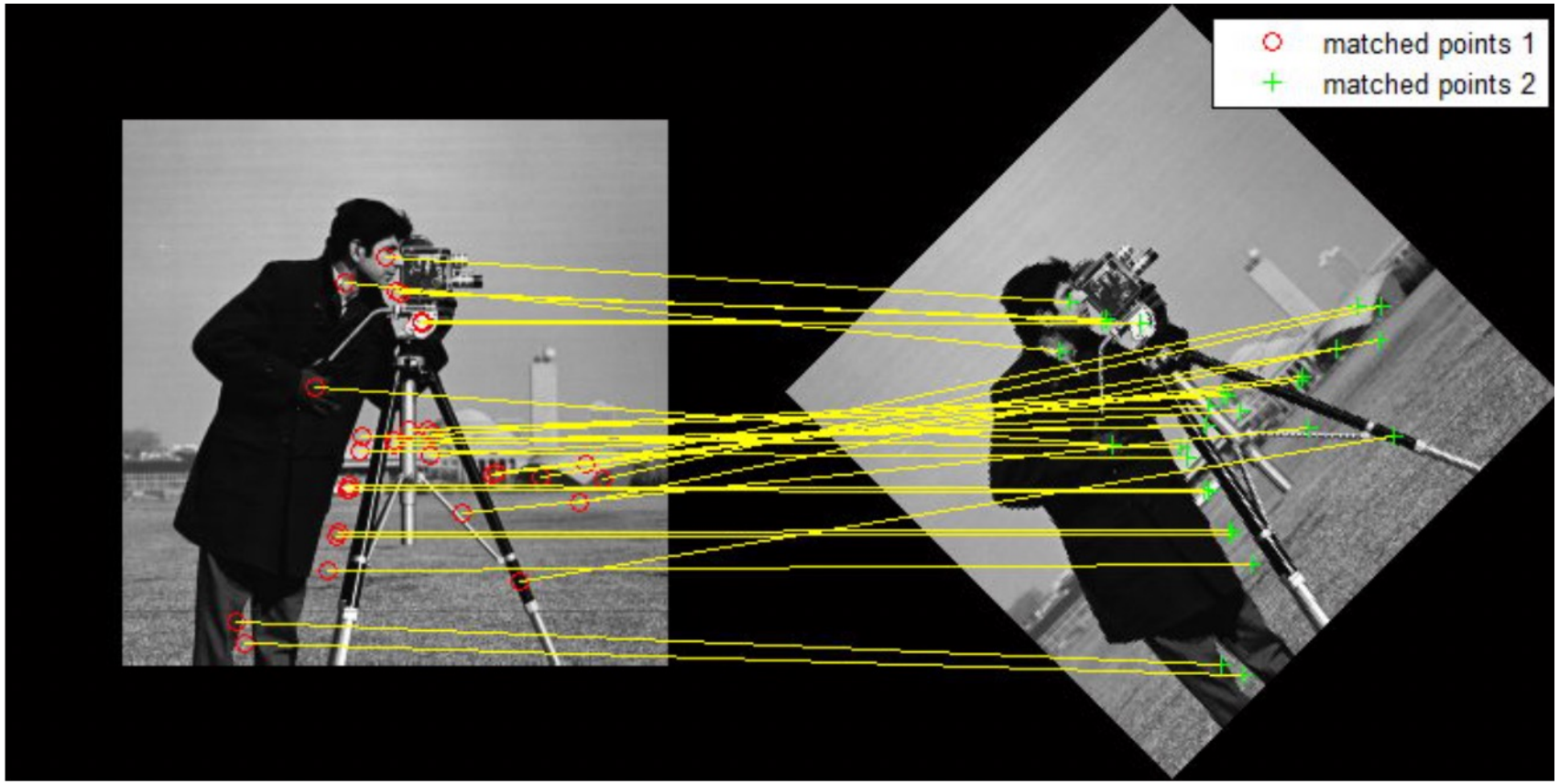
FAST, Harris Corners, SIFT and Applications

Instructor: Shaifali Parashar

Many Slides from Fei-Fei Li's lectures at Standford

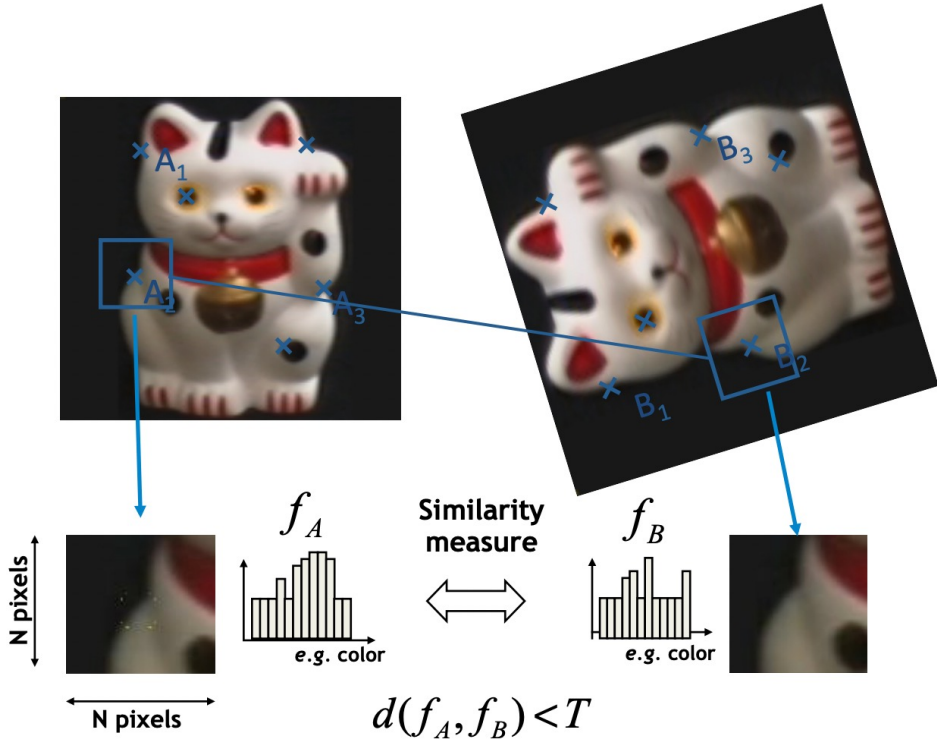
Why do we need features?

Finding same things across images: for reconstruction/tracking/generating panaromas …

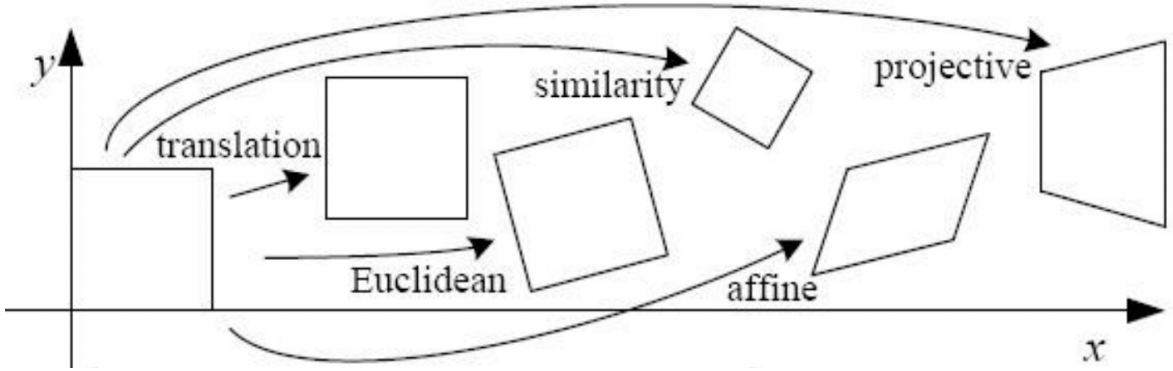


Feature Detection and Matching

1. Use a detector to detect same scene points independently in both images and find a set of distinctive keypoints
2. Extract and normalize the region content
3. Define a region around each keypoint
4. Compute local descriptor from normalized region
5. Find correspondences by matching local descriptors



What makes a good feature?



What makes a good feature?

Region extraction needs to be repeatable and accurate

Invariant to translation, rotation, scale changes

Robust or covariant to out-of-plane (affine) transformations

Robust to lighting variations, noise, blur, quantization

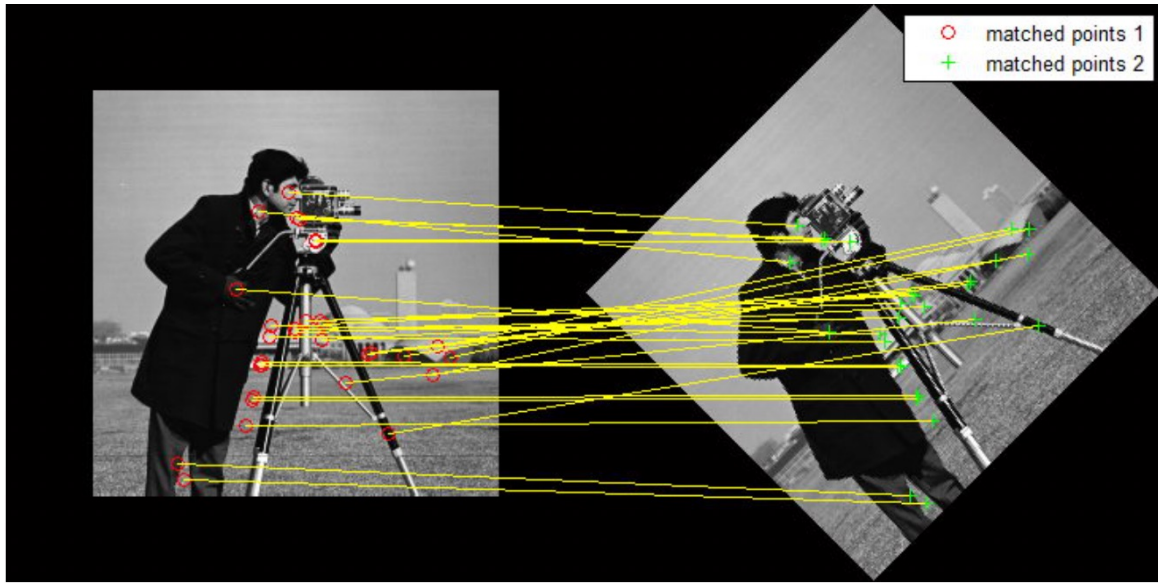
- Locality: Features are local, therefore robust to occlusion and clutter
- Quantity: We need sufficient regions to cover the object
- Distinctiveness: The regions should contain “interesting” structure
- Efficiency: Close to real-time performance

What makes a good feature?

Pixel Intensity

Pros: can get many, slight invariance to everting

Cons: Non-immune to photometry changes, noise, any strong geometric variation (even scale or rotation)

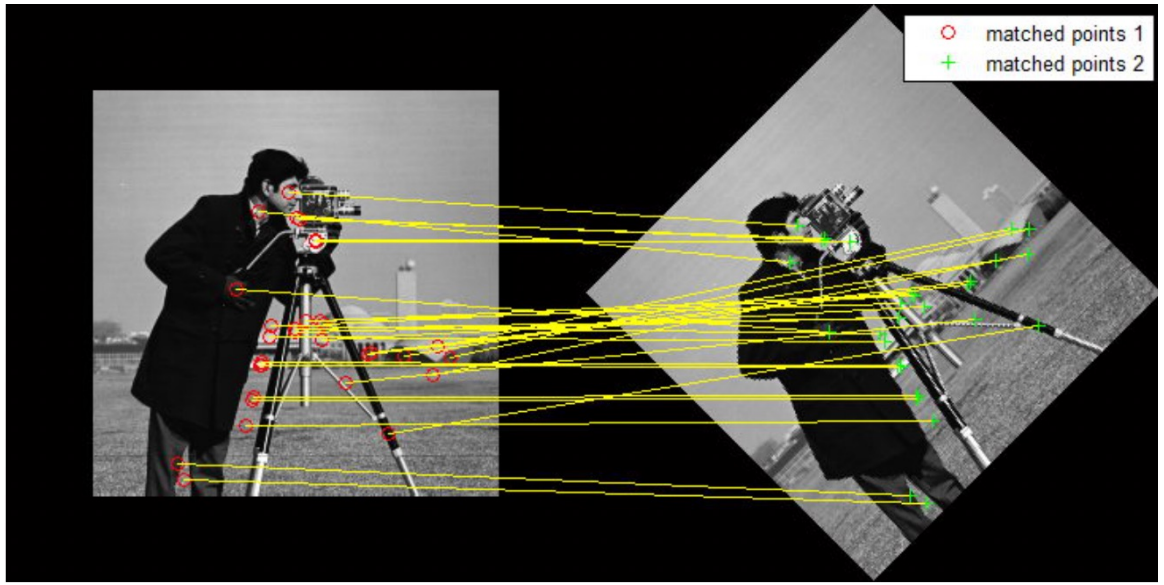


What makes a good feature?

Geometry

Pros: Can handle photometric changes, Invariances to some transformations, resistance to noise

Cons: Computationally expensive

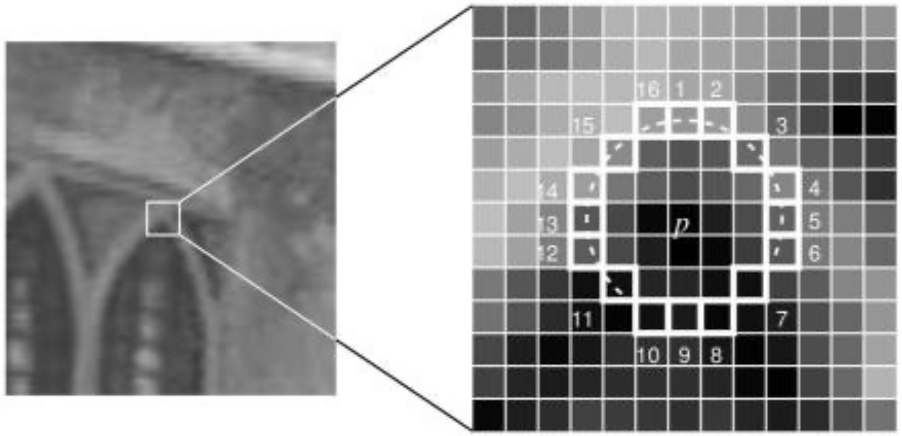


Features from Accelerated Segment Test (FAST)

Look for a 3 neighborhood circle (Bresenhem/mid point circle) around a point = 16 points

If a minimum of 12 points have intensity higher or smaller than the point, it's a good feature

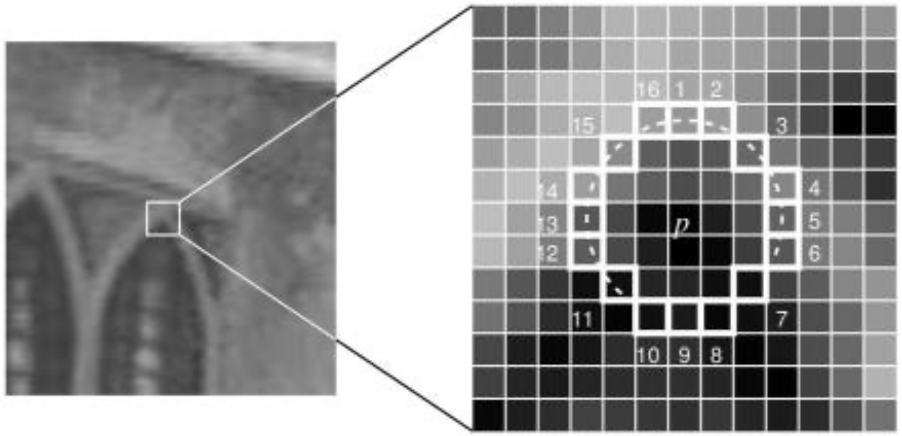
Good for detecting corners

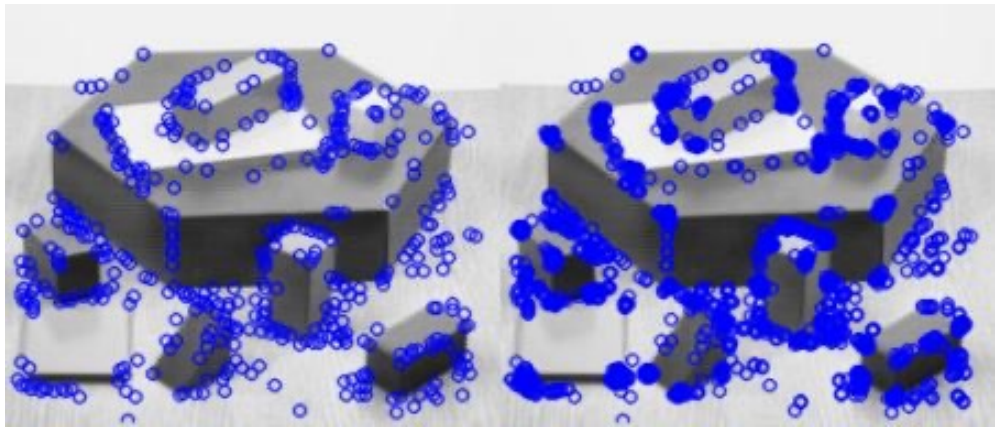


Features from Accelerated Segment Test (FAST)

Problems:

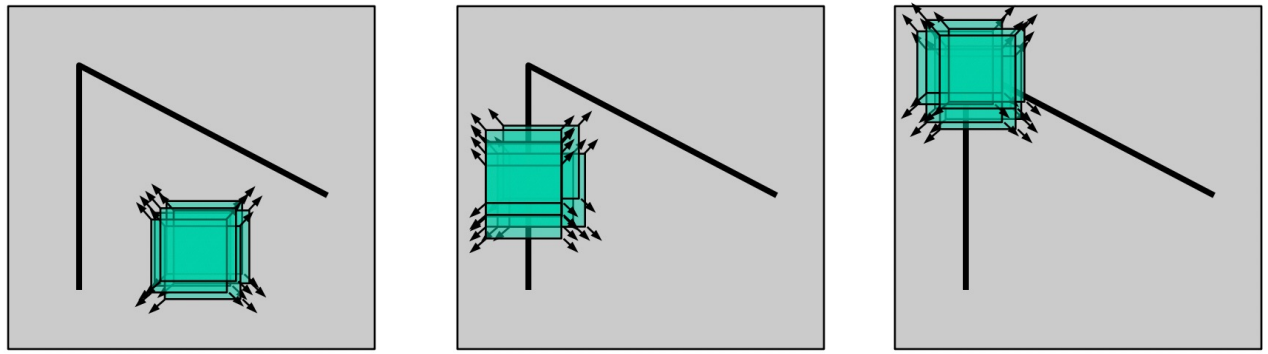
1. Slow (make fast by looking at pixels 1,5,9,13 or use machine learning)
2. Important to decide thresholds
3. Shadows/ small textual change can impact negatively
4. Photometric variations will impact it negatively





Harris Corners

- Localise the point through a small window
- Define precise location by shifting a window in any direction such that large change in pixels intensities are observed



"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

Harris Corners

Window-averaged squared change of intensity induced by shifting the image data by [u,v]:

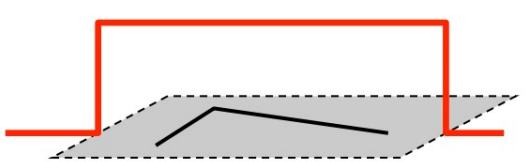
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

Window function

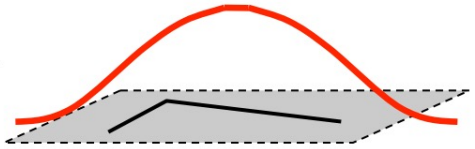
Shifted intensity

Intensity

Window function $w(x,y)$ =



or



1 in window, 0 outside

Gaussian

Darya Frolova, Denis Simakov The Weizmann Institute of Science
http://www.wisdom.weizmann.ac.il/~deniss/vision_spring04/files/InvariantFeatures.ppt

Harris Corners

$$E(u,v) \approx \sum_{x,y} w(x,y)[I(x,y) + uI_x + vI_y - I(x,y)]^2$$

$$= \sum_{x,y} w(x,y)[uI_x + vI_y]^2$$

$$= (u \quad v) \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

= M (structure tensor)

Its eigenanalysis reveals interesting things

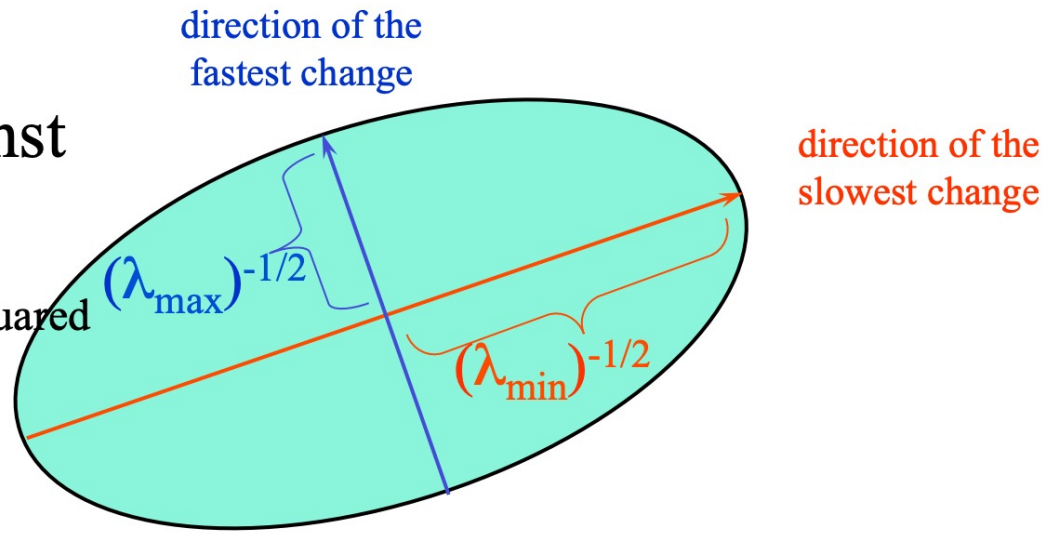
Harris Corners

Intensity change in shifting window: eigenvalue analysis

$$E(u,v) \cong \begin{bmatrix} u, v \end{bmatrix} \; M \; \begin{bmatrix} u \\ v \end{bmatrix}$$

λ_1, λ_2 – eigenvalues of M

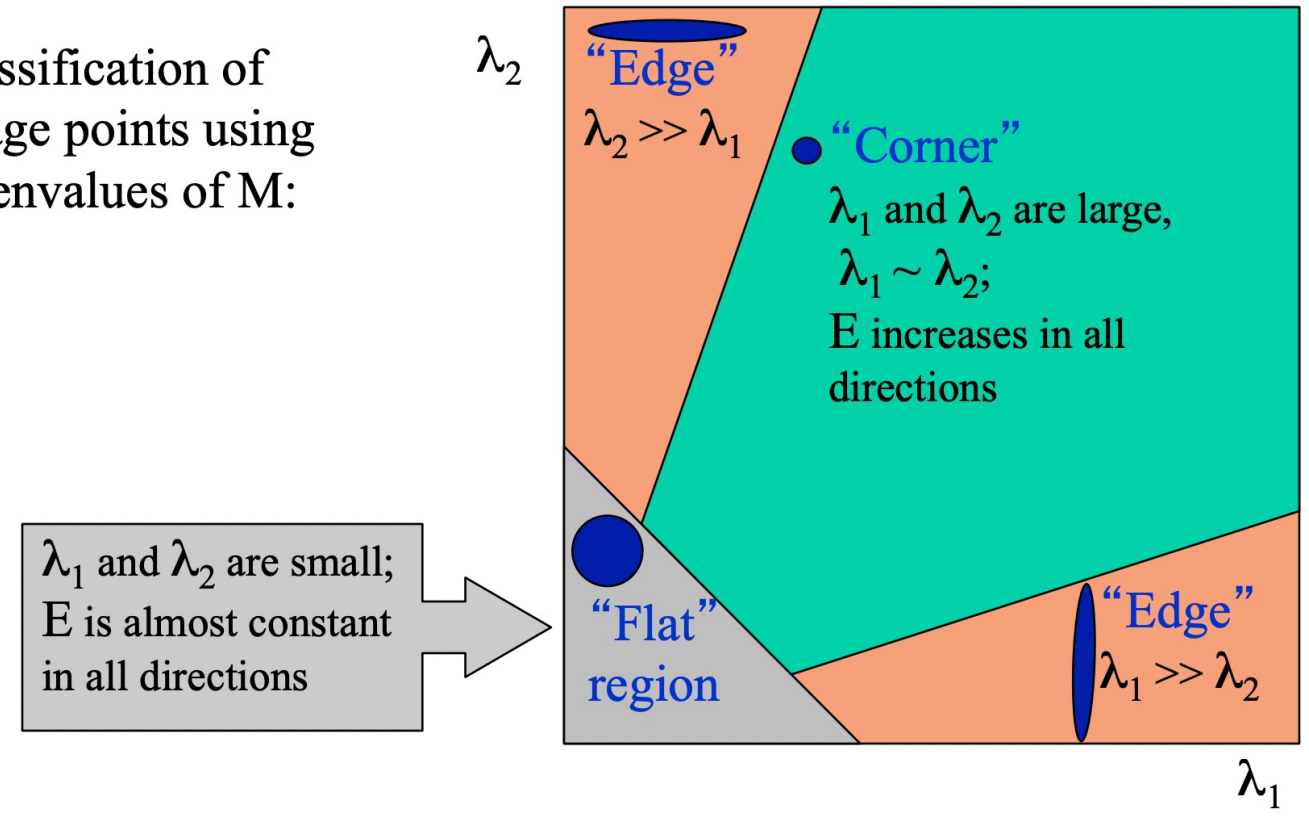
Ellipse E(u,v) = const



Iso-contour of the squared error, E(u,v)

http://www.wisdom.weizmann.ac.il/~deniss/vision_spring04/files/InvariantFeatures.ppt

Harris Corners



Instead of computing eigenvalues:
R = det(M) – a trace(M)^2

a = [0.04, 0.06]

Corner: R>0
Edge: R<0
Flat: abs(R) is small

Harris Corners

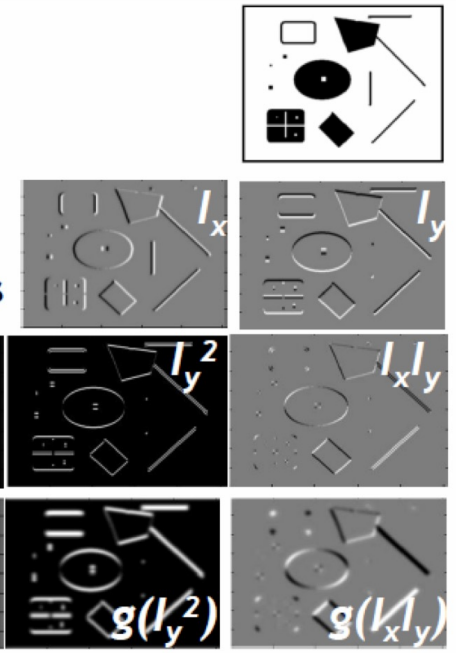
- Compute second moment matrix (autocorrelation matrix)

$$M(\sigma_I, \sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

1. Image derivatives

2. Square of derivatives

3. Gaussian filter $g(\sigma_I)$



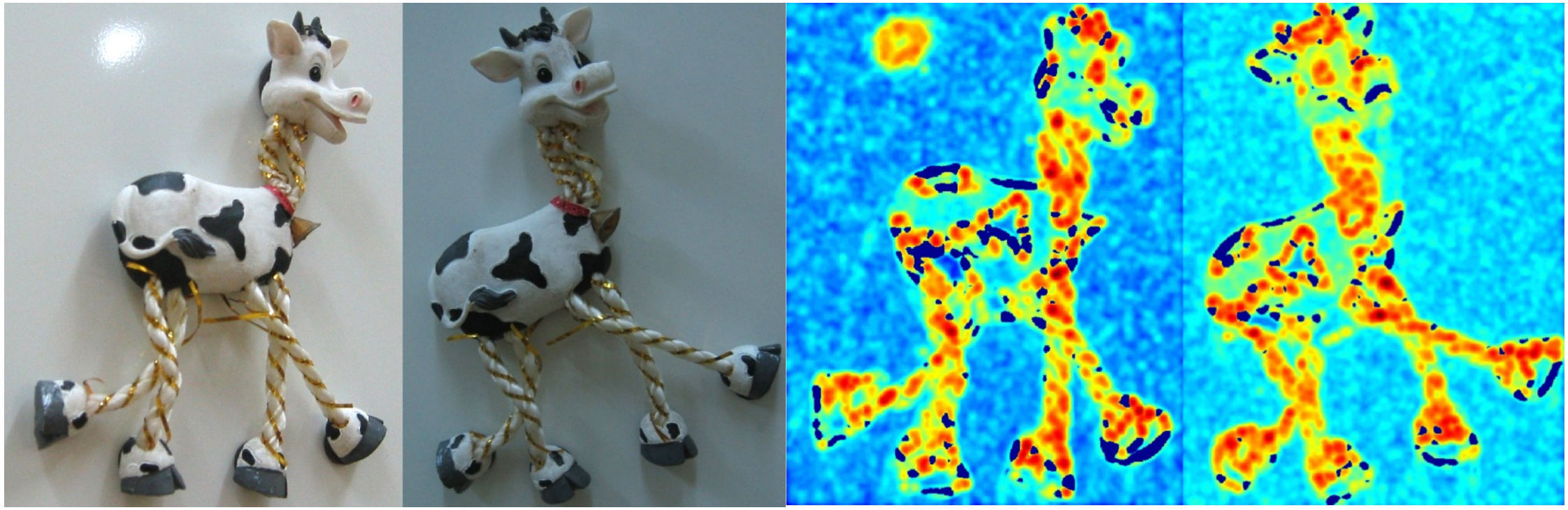
4. Cornerness function - two strong eigenvalues

$$R = \det[M(\sigma_I, \sigma_D)] - \alpha[\text{trace}(M(\sigma_I, \sigma_D))]^2$$
$$= g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$

5. Perform non-maximum suppression

Slide credit: Krystian Mikolajczyk

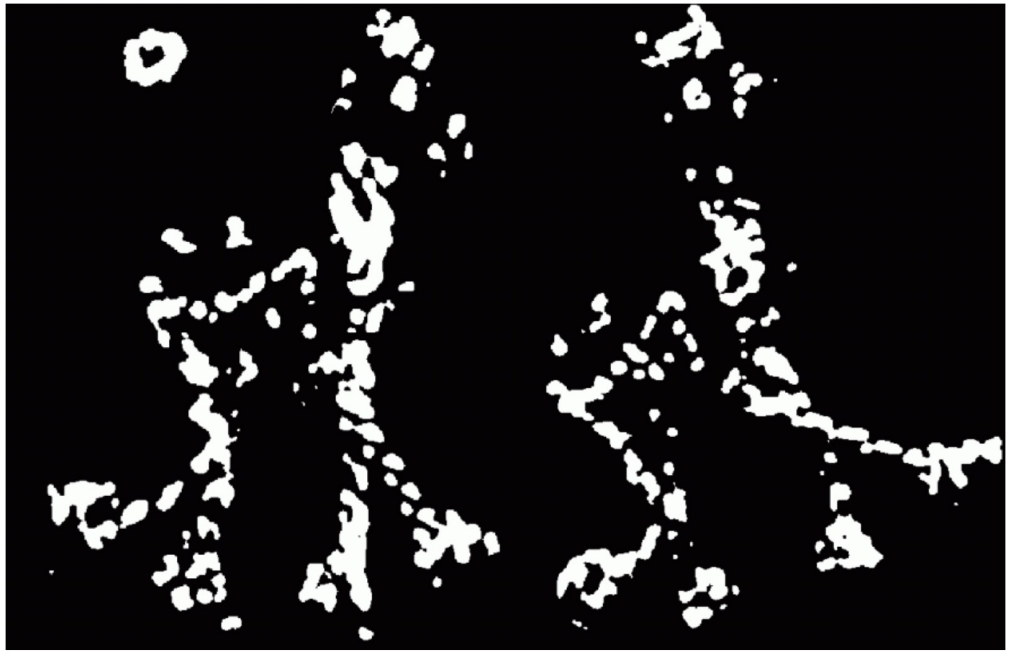
Harris Corners



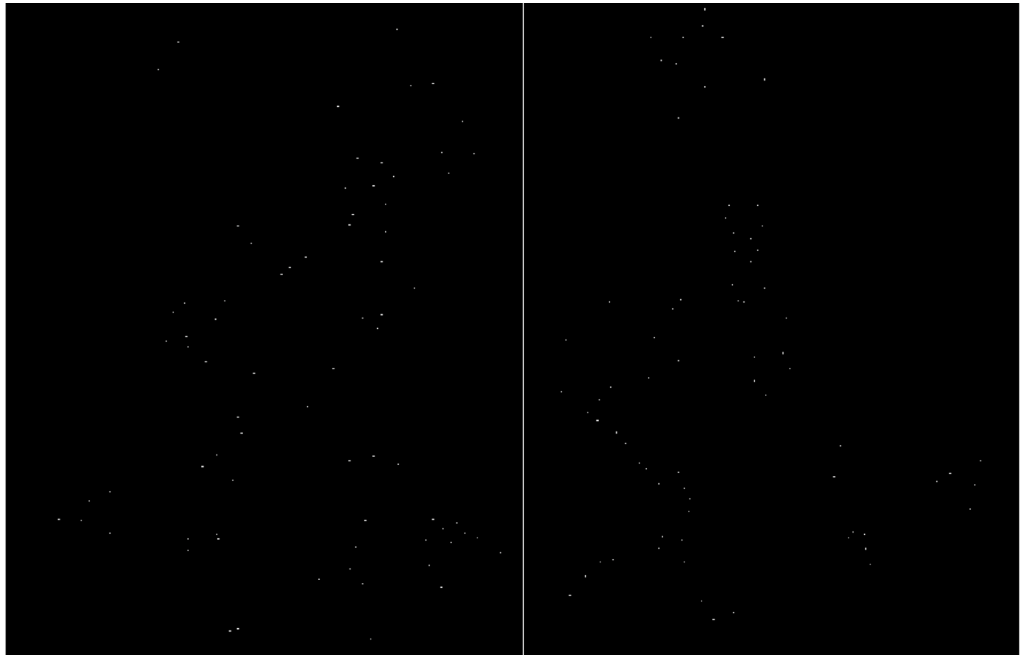
Computed R

Harris Corners

Find points with large corner response:
R>threshold



Take only the points of local maxima of R

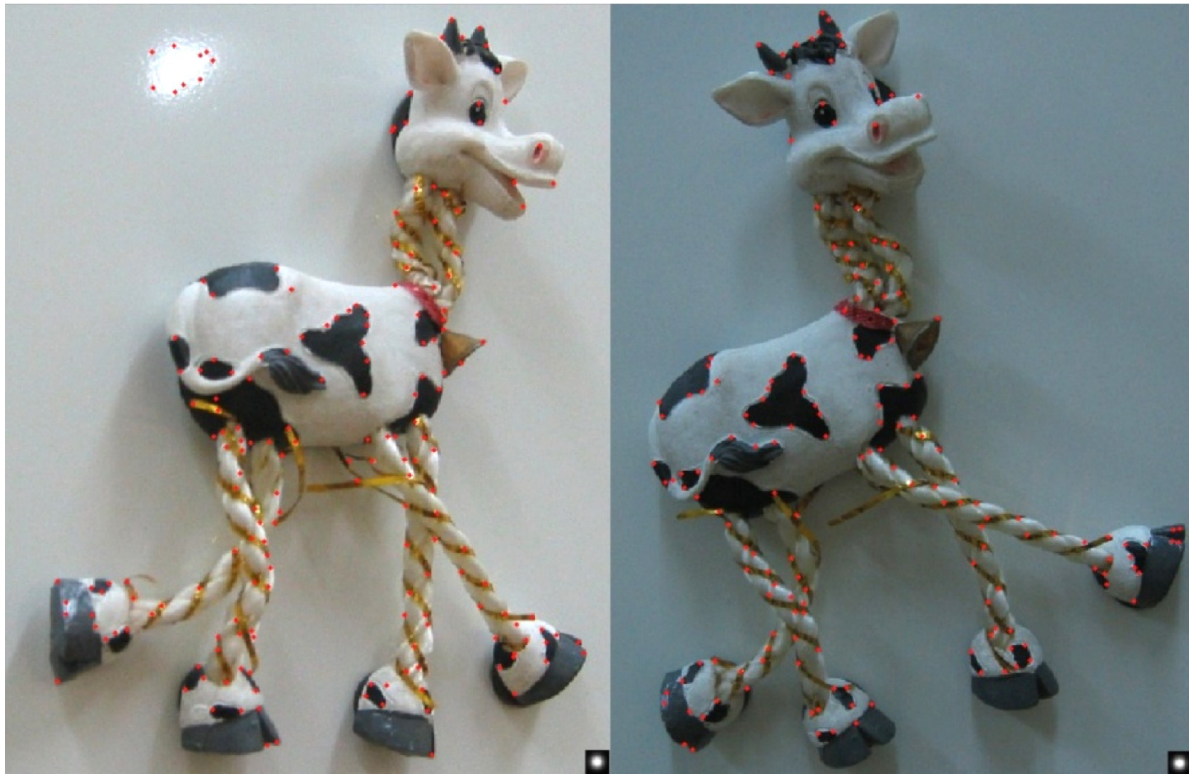


Harris Corners

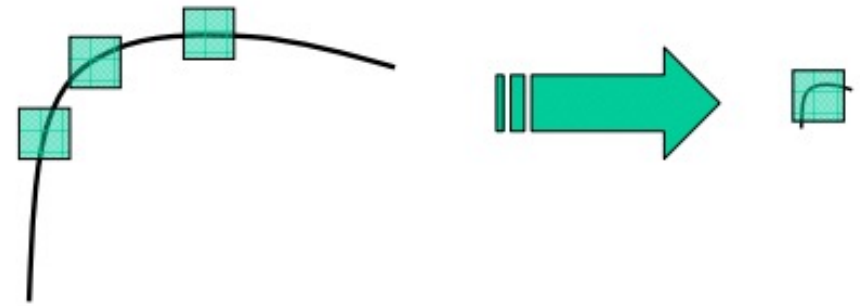
We want corner locations to be invariant to photometric transformations and covariant to geometric transformations

Invariance: image is transformed and corner locations do not change

Covariance: if we have two transformed versions of the same image, features should be detected in corresponding locations



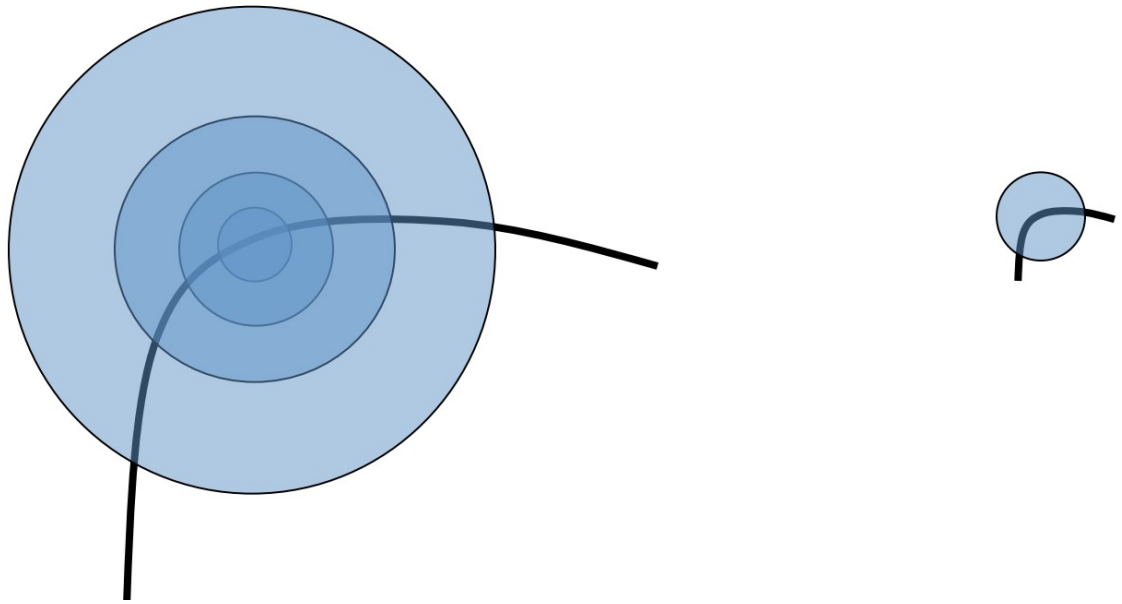
Harris Corners



Invariant to rotation, translation, photometric intensity changes (slightly).
Scale? No.

Scale Invariant Detection

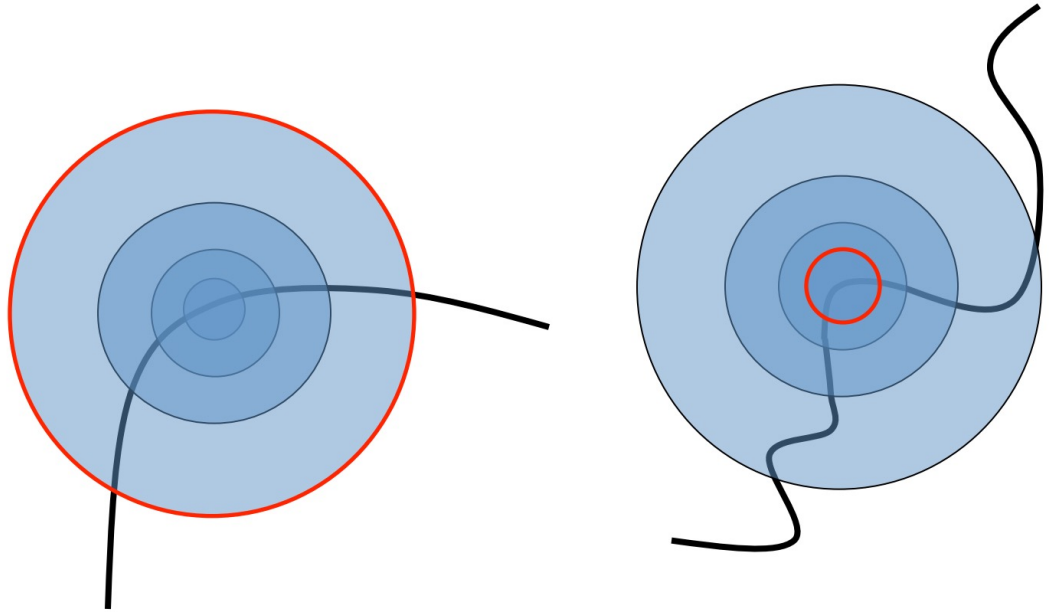
- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images



Scale Invariant Detection

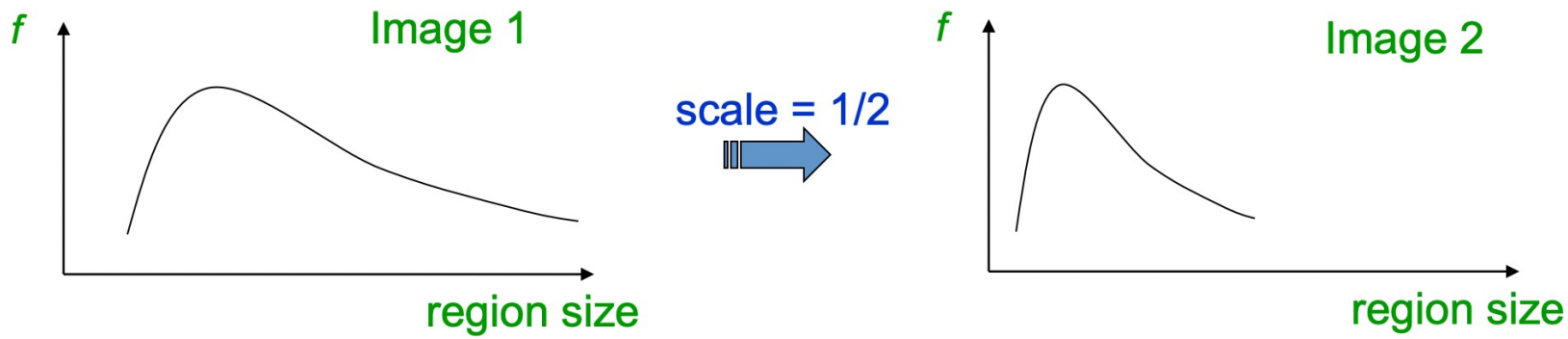
- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images

How to choose these regions automatically?



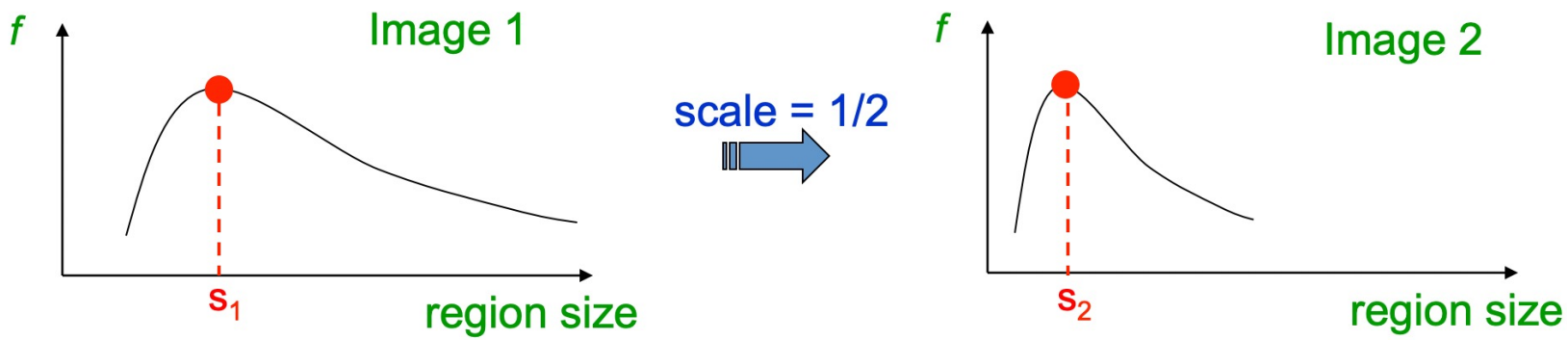
Scale Invariant Detection

- Design a function on the region (circle), which is "scale invariant" (the same for corresponding regions, even if they are at different scales)
- Image intensity (invariant to scale, of course):For a point in one image, we can consider it as a function of region size (circle radius)



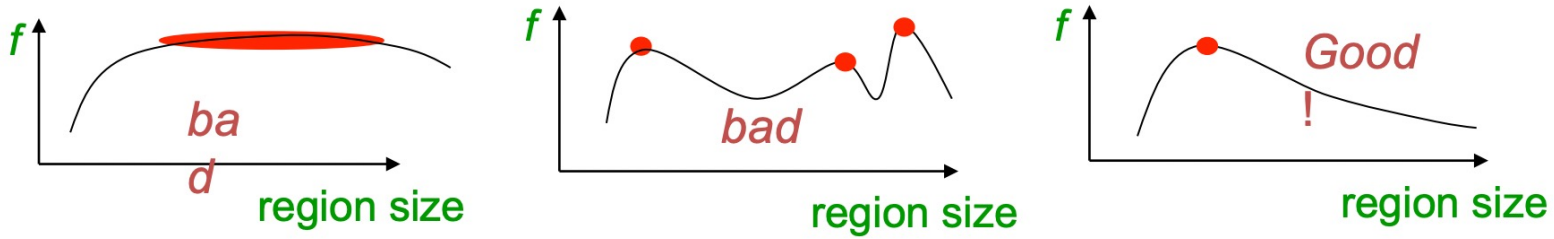
Scale Invariant Detection

- Take a local maximum of image intensity
- Region size, for which the maximum is achieved, is normally covariant with image scale
- Find these regions independently in each image



Scale Invariant Detection

- A good function should have 1 stable peak
- Should react to sharp intensity changes



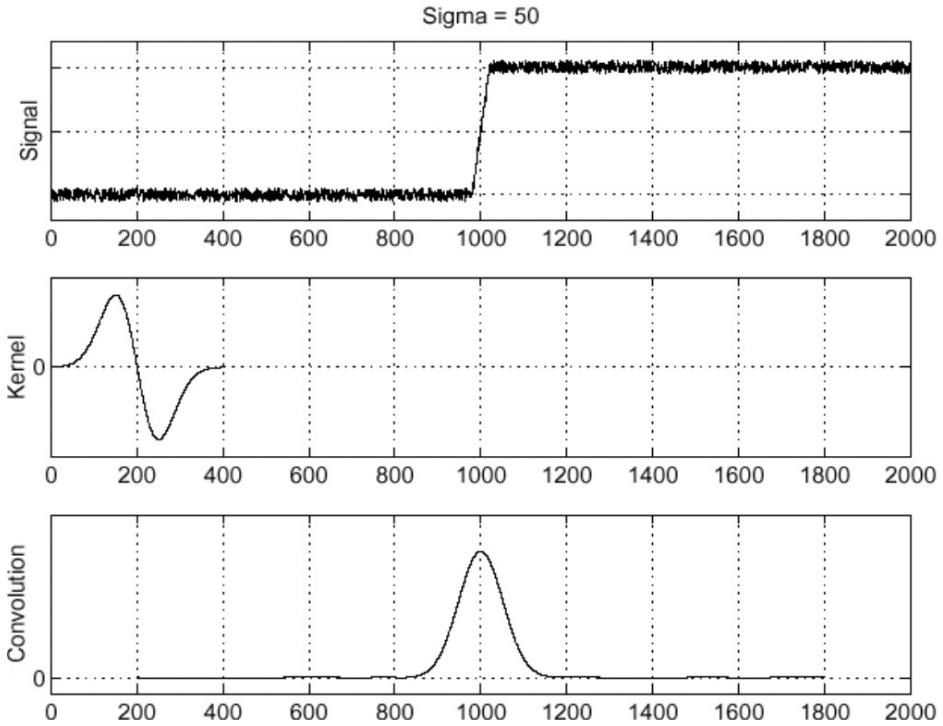
Scale Invariant Detection

- Remember edge detection

f

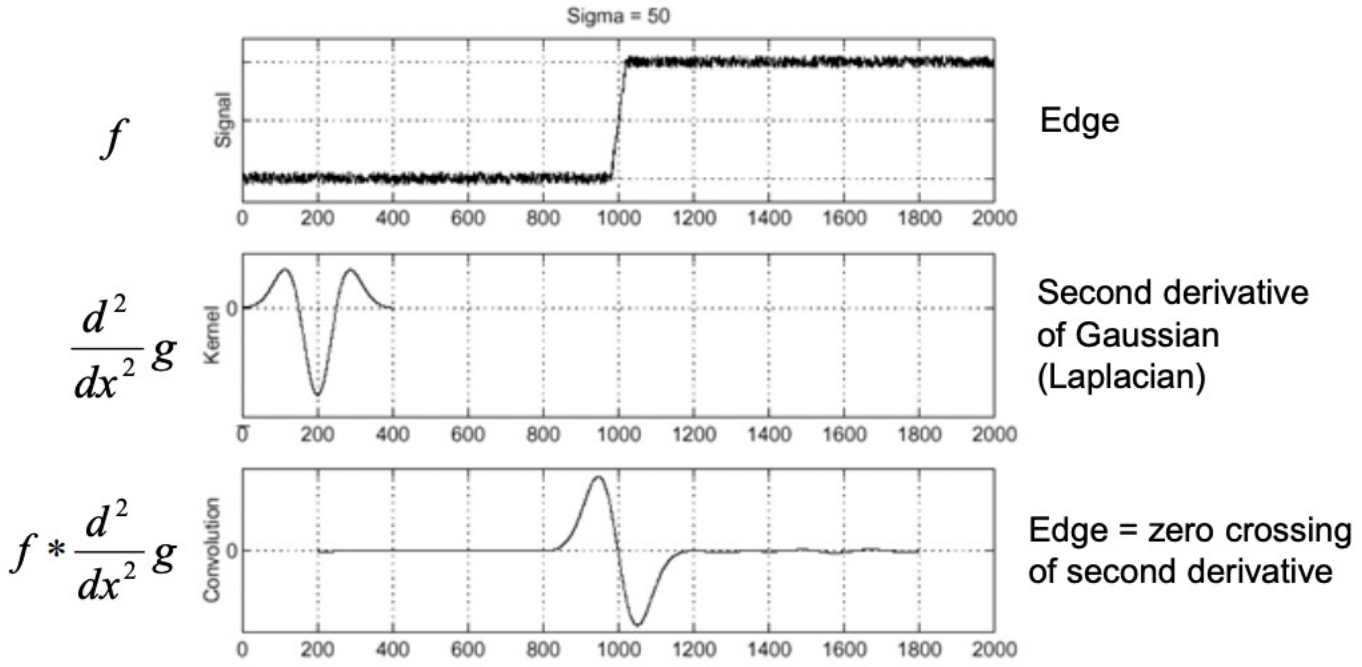
$\frac{d}{dx} g$

$f * \frac{d}{dx} g$



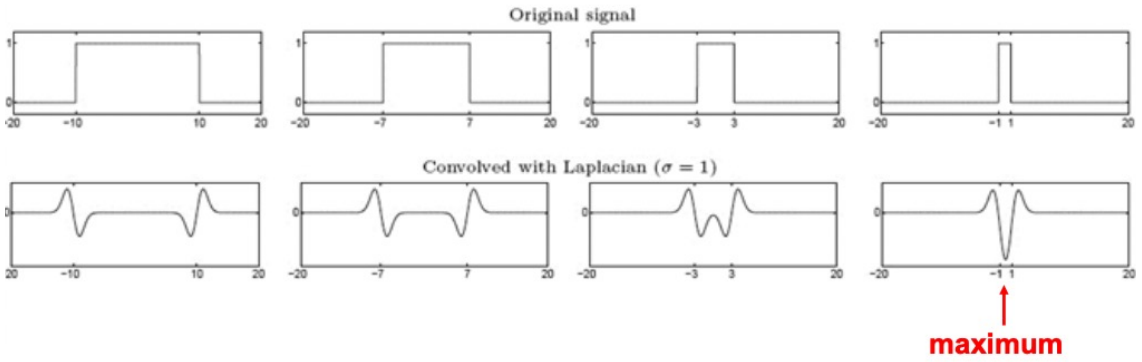
Scale Invariant Detection

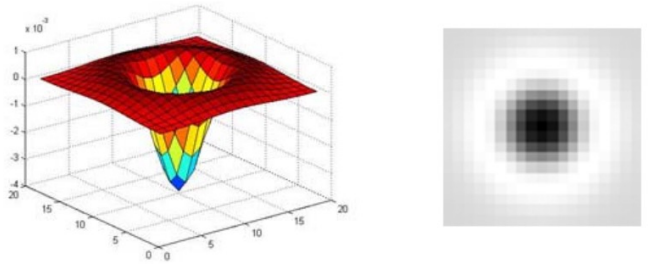
- Remember edge detection



Scale Invariant Detection

- Laplacians are good candidate

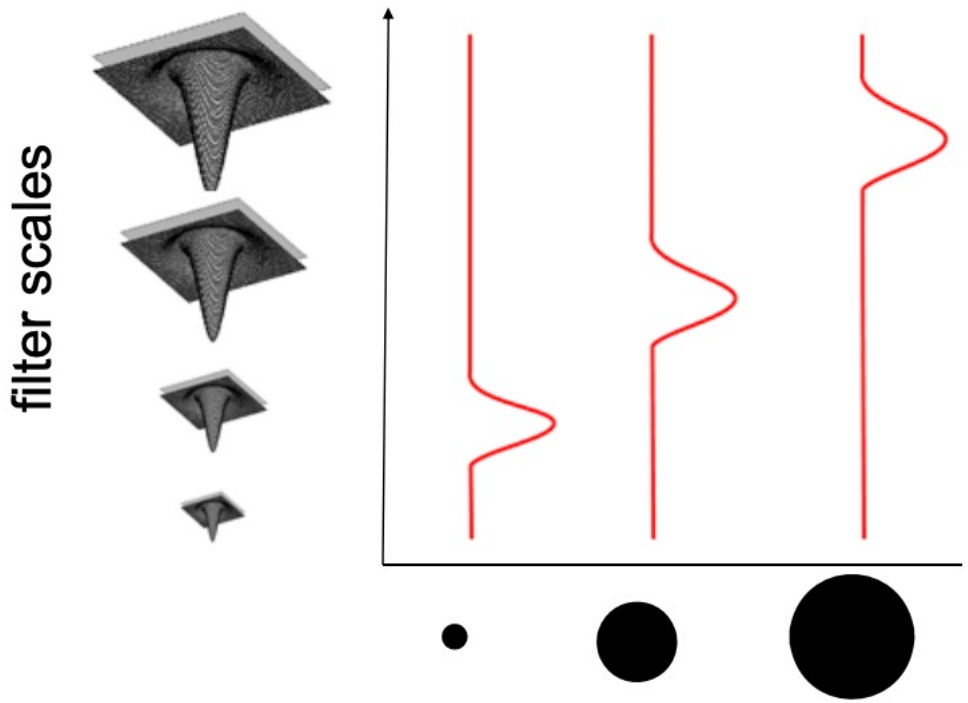


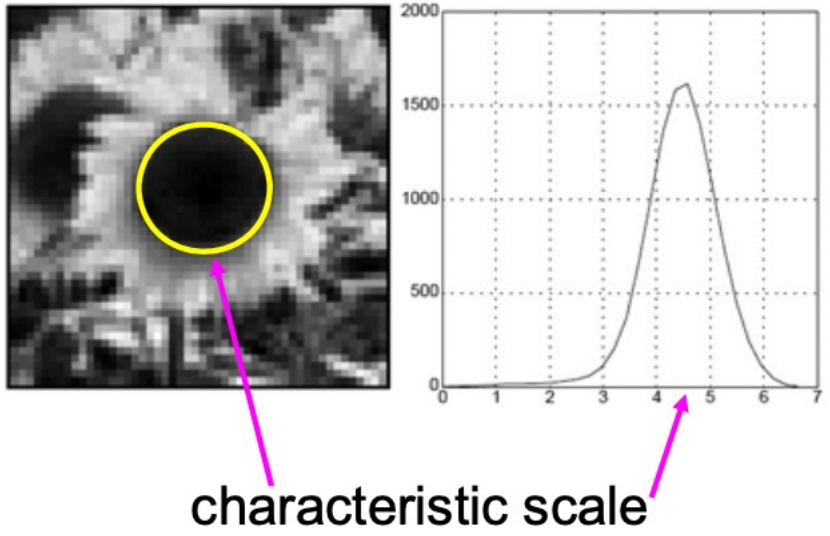


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

Scale Invariant Detection

- Characteristic scale: scale that produces a peak





Scale Invariant Detection

- Interest points are local maximas in both position and scale
  - **Functions for determining scale** $f = \text{Kernel} * \text{Image}$

Kernels:

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

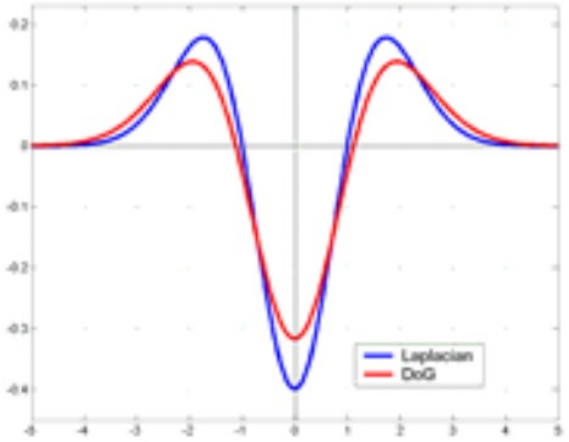
(Laplacian)

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

(Difference of Gaussians)

where Gaussian

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

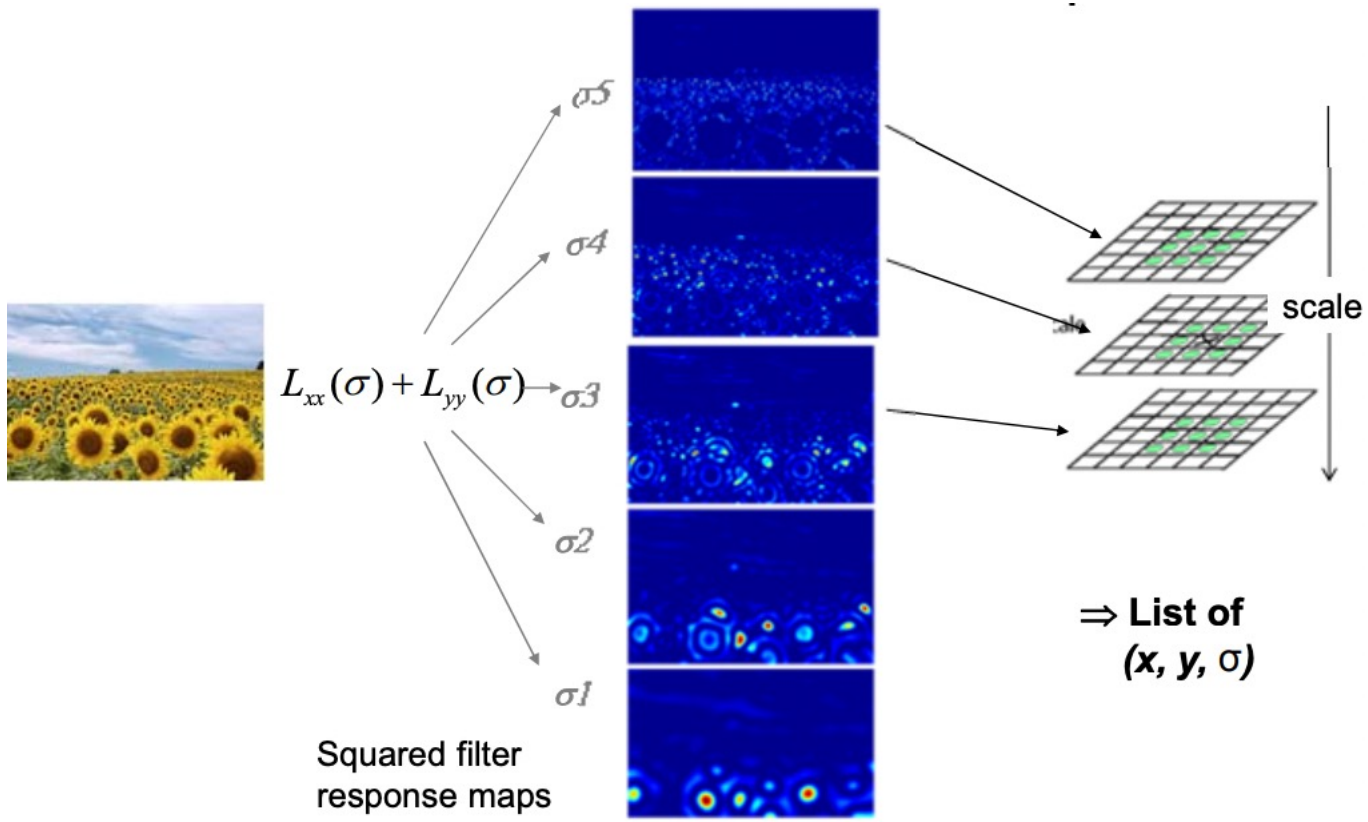


Note: both kernels are invariant to *scale* and *rotation*

DoG is computationally efficient that Laplacians

Scale Invariant Detection

- Interest points are local maximas in both position and scale

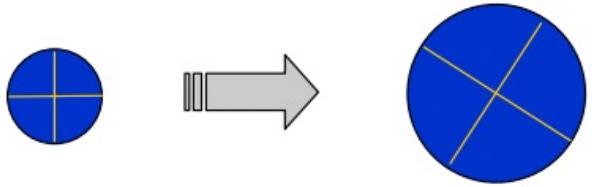


Scale Invariant Detection

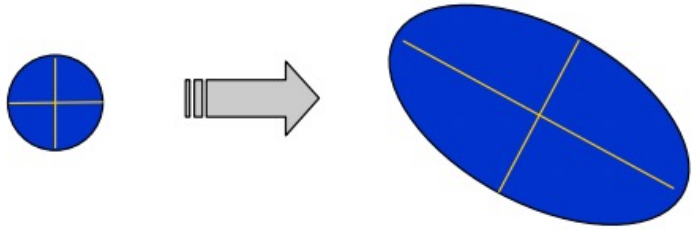
- Given: two images of the same scene with a large scale difference between them
- Goal: find the same interest points independently in each image
- Solution: search for maxima of suitable functions in scale and in space (over the image)

Affine Invariant Detection

- **Above we considered:**
  Similarity transform (rotation + uniform scale)

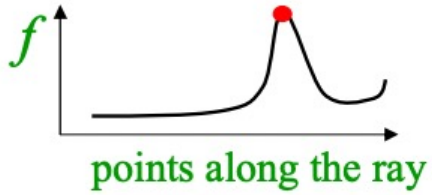


- Now we go on to:
  Affine transform (rotation + non-uniform scale)



Affine Invariant Detection

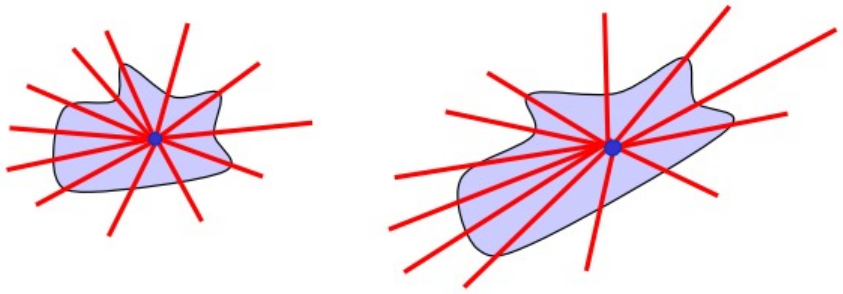
- **Take a local intensity extremum as initial point**
- **Go along every ray starting from this point and stop when extremum of function f is reached**



$$f(t) = \frac{\left|I(t) - I_0\right|}{\frac{1}{t}\int_o^t \left|I(t) - I_0\right| dt}$$

- We will obtain approximately corresponding regions

Remark: we search for scale in every direction



T.Tuytelaars, L.V.Gool. "Wide Baseline Stereo Matching Based on Local, Affinely Invariant Regions", BMVC 2000.

Affine Invariant Detection

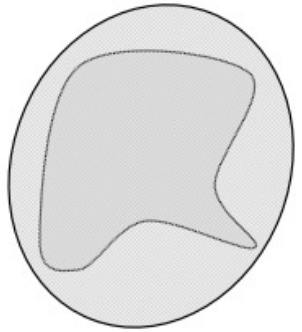
- **The regions found may not exactly correspond, so we approximate them with ellipses**

- Geometric Moments:

$$m_{pq} = \int_{\mathbb{R}^2} x^p y^q f(x,y)\,dx\,dy$$

Fact: moments m_{pq} uniquely determine the function f

Taking f to be the characteristic function of a region (1 inside, 0 outside), moments of orders up to 2 allow to approximate the region by an ellipse

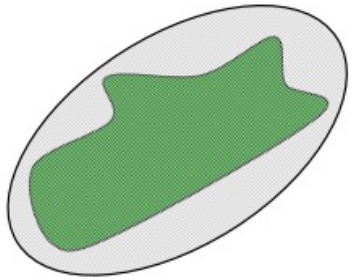


This ellipse will have the same moments of orders up to 2 as the original region

Affine Invariant Detection

- Covariance matrix of region points defines an ellipse:

$$q = Ap$$



$$p^T \Sigma_1^{-1} p = 1$$

$$q^T \Sigma_2^{-1} q = 1$$

$$\Sigma_1 = \left\langle pp^T \right\rangle_{\text{region 1}}$$

$$\Sigma_2 = \left\langle qq^T \right\rangle_{\text{region 2}}$$

($p = [x, y]^T$ is relative to the center of mass)

$$\Sigma_2 = A \Sigma_1 A^T$$

Ellipses, computed for corresponding regions, also correspond!

Affine Invariant Detection

- **Under affine transformation, we do not know in advance shapes of the corresponding regions**
- **Ellipse given by geometric covariance matrix of a region robustly approximates this region**
- **For corresponding regions ellipses also correspond.**

Methods:

1. Search for extremum along rays [Tuytelaars, Van Gool]:
2. Maximally Stable Extremal Regions [Matas et.al.]

Affine Invariant Detection

- **Algorithm summary (detection of affine invariant region):**

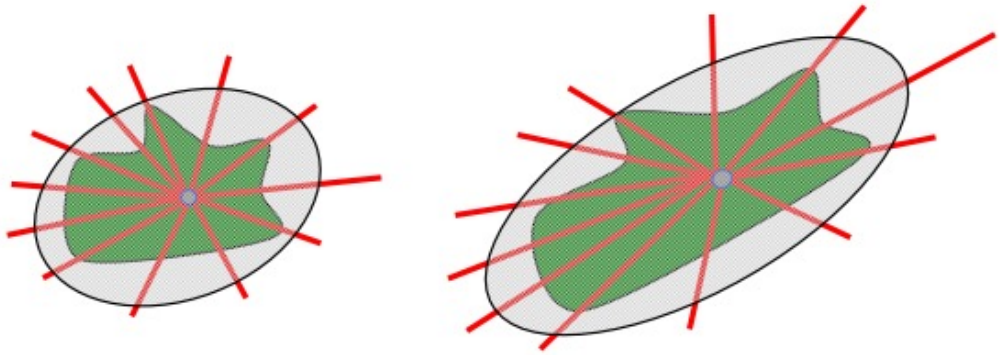
  Start from a *local intensity extremum* point

  Go in *every direction* until the point of extremum of some function f

  Curve connecting the points is the region boundary

  Compute *geometric moments* of orders up to 2 for this region

  Replace the region with *ellipse*



Comparison of Keypoint Detection

Table 7.1 Overview of feature detectors.

Feature Detector	Corner	Blob	Region	Rotation invariant	Scale invariant	Affine invariant	Localization Repeatability	Localization accuracy	Robustness	Efficiency
Harris	√			√			+++	+++	+++	++
Hessian		√		√			++	++	++	+
SUSAN	√			√			++	++	++	+++
Harris-Laplace	√	(√)		√	√		+++	+++	++	+
Hessian-Laplace	(√)	√		√	√		+++	+++	+++	+
DoG	(√)	√		√	√		++	++	++	++
SURF	(√)	√		√	√		++	++	++	+++
Harris-Affine	√	(√)		√	√	√	+++	+++	++	++
Hessian-Affine	(√)	√		√	√	√	+++	+++	+++	++
Salient Regions	(√)	√		√	√	(√)	+	+	++	+
Edge-based	√			√	√	√	+++	+++	+	+
MSER			√	√	√	√	+++	+++	++	+++
Intensity-based			√	√	√	√	++	++	++	++
Superpixels			√	√	(√)	(√)	+	+	+	+

Tuytelaars Mikolajczyk 2008

Takeaway

What do you want it for?

- Precise localization in x-y: Harris
- Good localization in scale: Difference of Gaussian

Best choice often application dependent

- Harris-/Hessian-Laplace/DoG work well for many natural categories

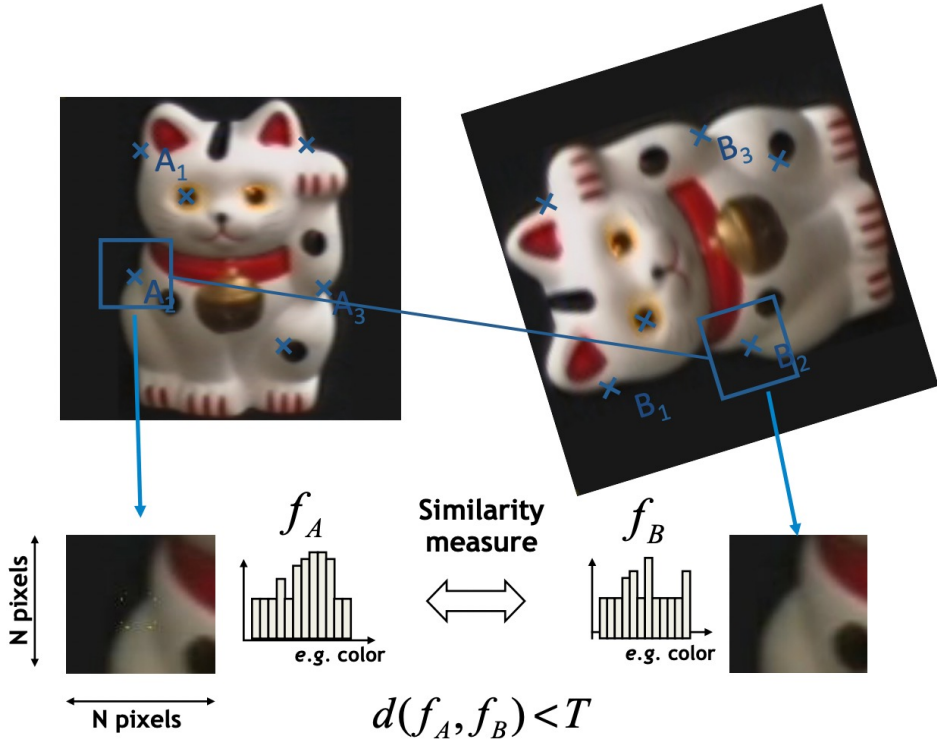
Extensive evaluations/comparisons available in [Mikolajczyk et al., IJCV-2005, PAMI-2005]

For most local feature detectors, executables are available online:
http://robots.ox.ac.uk/~vgg/research/affine
http://www.cs.ubc.ca/~lowe/keypoints/
http://www.vision.ee.ethz.ch/~surf

Feature Detection and Matching (Descriptor matching)

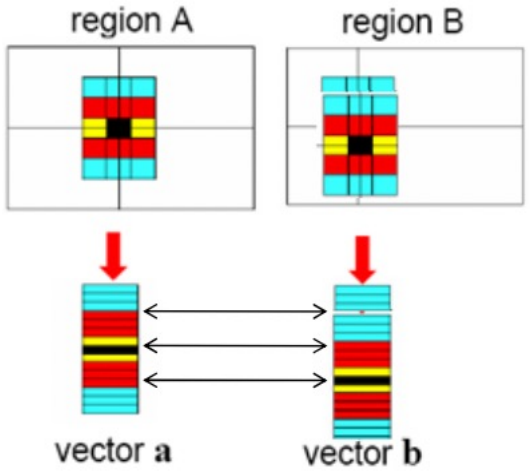
1. Use a detector to detect same scene points independently in both images and find a set of distinctive keypoints
2. Extract and normalize the region content
3. Define a region around each keypoint
4. Compute local descriptor from normalized region
5. Find correspondences by matching local descriptors

Once the detector has been decided, we need to look for a descriptor, so that the detected points can be represented distinctively and matched properly



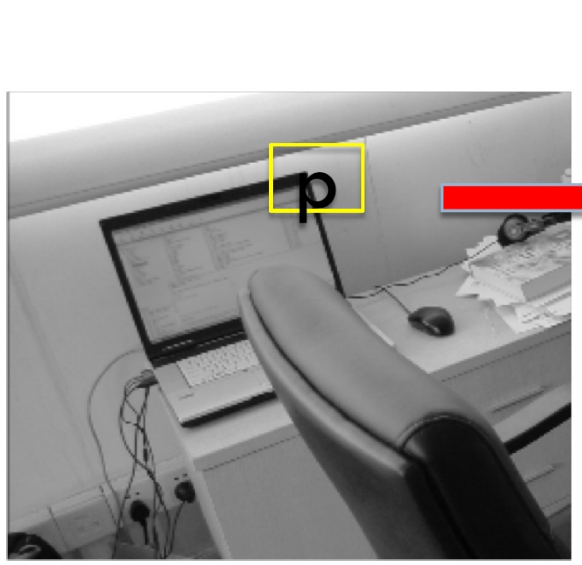
Patches as Descriptor

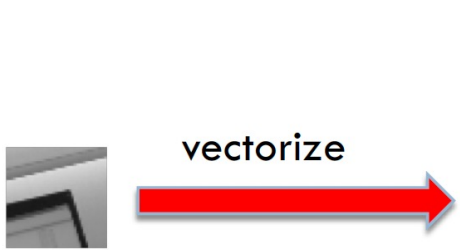
- The simplest way to describe the neighborhood around an interest point is to write down the list of intensities to form a feature vector. But this is very sensitive to even small shifts, rotations.



Patches as Descriptors

- Consider a region around a feature point
- Stack the region into a vector
- Use correlation to match





vectorize

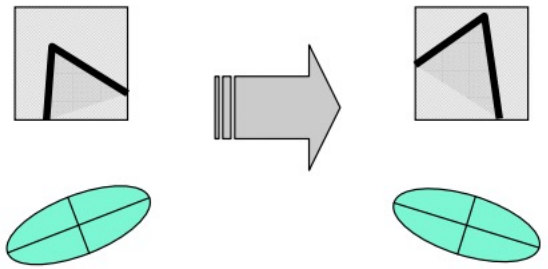
$$\begin{bmatrix} v1 \\ v2 \\ \vdots \\ v_n \end{bmatrix}$$

$$\rho = \frac{E[\mathbf{v}_1 - E(\mathbf{v}_1)][\mathbf{v}_2 - E(\mathbf{v}_2)]}{\sqrt{D(\mathbf{v}_1)}\sqrt{D(\mathbf{v}_2)}}$$

Descriptors Invariant to Rotation

- **Harris corner response measure:**
  depends only on the eigenvalues of the matrix *M*

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



- **Image moments in polar coordinates**

$$m_{kl} = \iint r^k e^{-i\theta l} I(r,\theta) dr d\theta$$

Rotation in polar coordinates is translation of the angle:

$$\theta \rightarrow \theta + \theta_0$$

This transformation changes only the phase of the moments, but not its magnitude

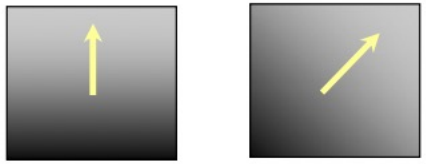
Rotation invariant descriptor consists of magnitudes of moments: $|m_{kl}|$

Matching is done by comparing vectors $[|m_{kl}|]_{k,l}$

Descriptors Invariant to Rotation

- **Find local orientation**

Dominant direction of gradient



- Compute image derivatives relative to this orientation

Descriptors Invariant to Scale

- **Use the scale determined by detector to compute descriptor in a normalized frame**

For example:

- moments integrated over an adapted window
- derivatives adapted to scale: sI_x

Descriptors Invariant to Affine transformation

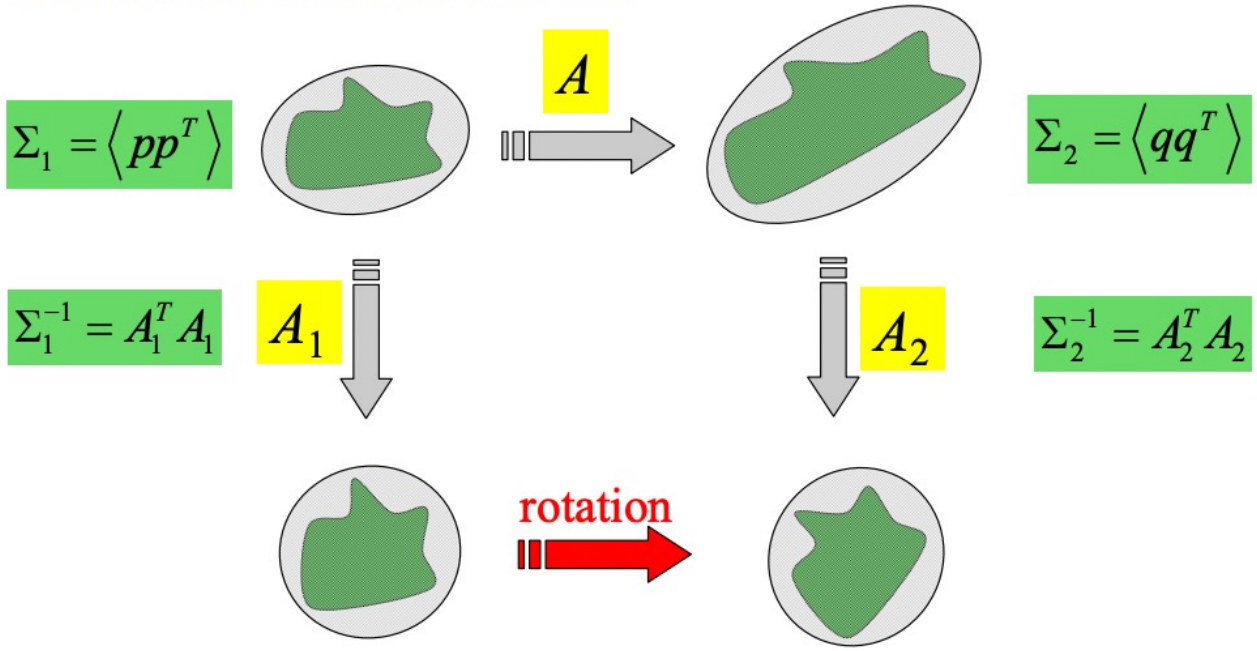
- **Affine invariant color moments**

$$m_{pq}^{abc} = \int_{region} x^p y^q R^a(x,y) G^b(x,y) B^c(x,y) dxdy$$

Different combinations of these moments are fully affine invariant

Also invariant to affine transformation of intensity $I \rightarrow a\,I + b$

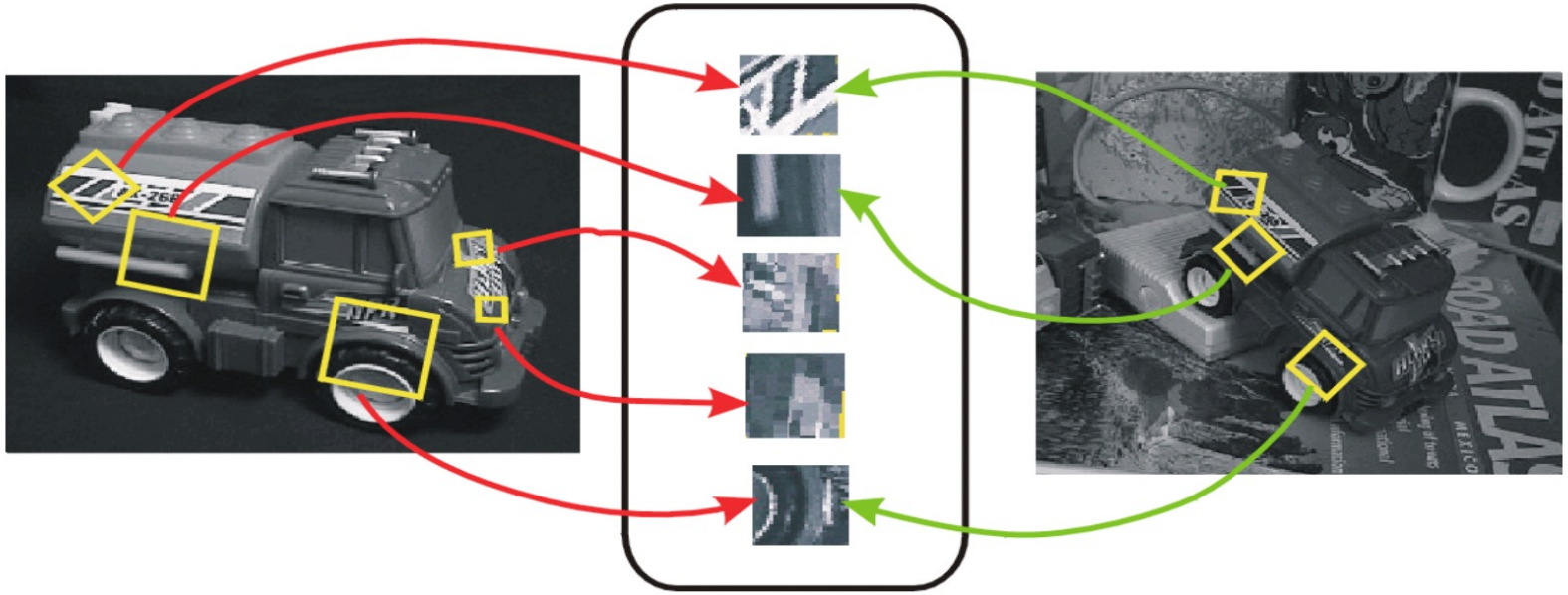
- **Find affine normalized frame**



- Compute rotational invariant descriptor in this normalized frame

Scale Invariant Feature Transform- SIFT Descriptor

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



SIFT

- Uses DoG to detect key points and the associated characteristic scales
- Motivation: The Harris operator was not invariant to scale and correlation was not invariant to rotation.
- For better image matching, Lowe's goals were:

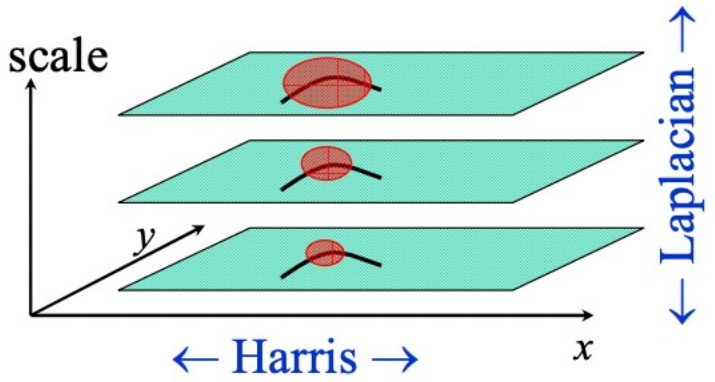
1. To develop an interest operator – a detector – that is invariant to scale and rotation.
2. Also: create a descriptor that was robust to the variations corresponding to typical viewing conditions.

- The descriptor isthe most-used part of SIFT.

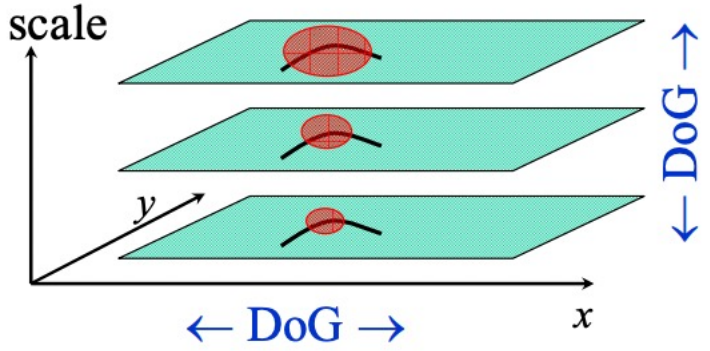
Prof. David Lowe
University of British Columbia, Canada

Scale Invariant Detection

- **Harris-Laplacian**[1]
  Find local maximum of:
  - Harris corner detector in space (image coordinates)
  - Laplacian in scale



- **SIFT (Lowe)**[2]
  Find local maximum of:
  - Difference of Gaussians in space and scale



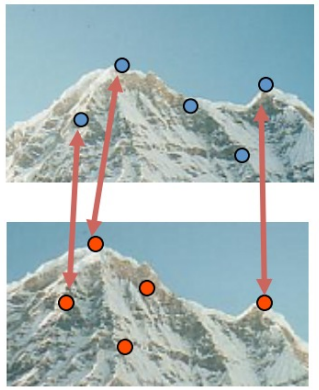
[1] K.Mikolajczyk, C.Schmid. "Indexing Based on Scale Invariant Interest Points". ICCV 2001
[2] D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". Accepted to IJCV 2004

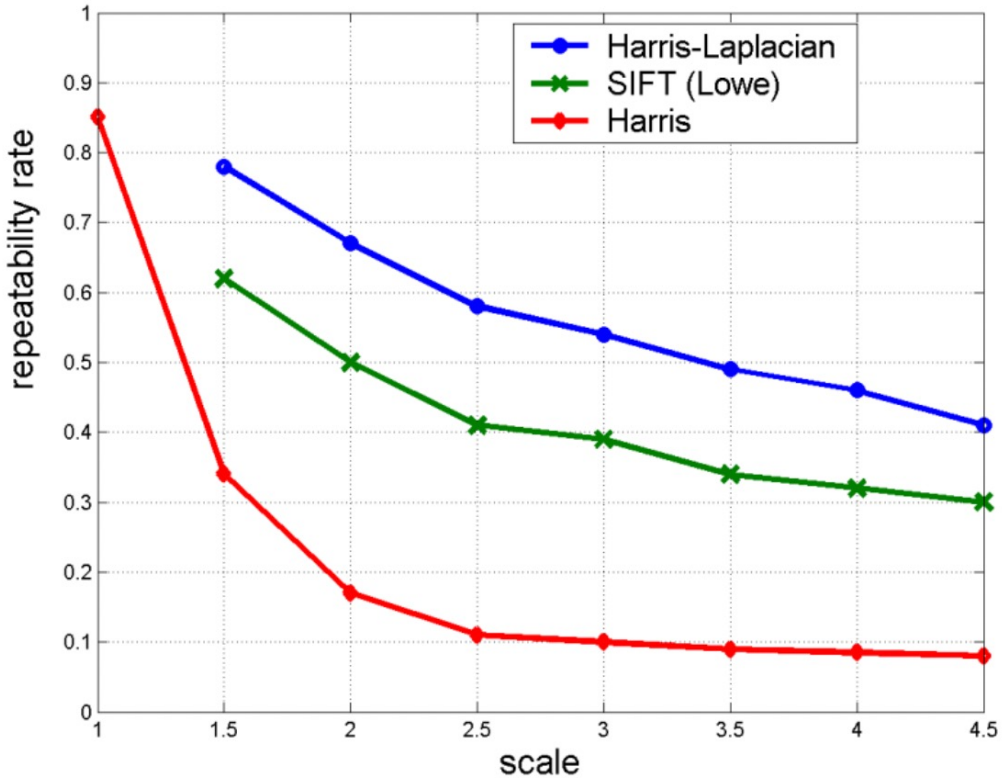
Scale Invariant Detection

- Experimental evaluation of detectors w.r.t. scale change

Repeatability rate:

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





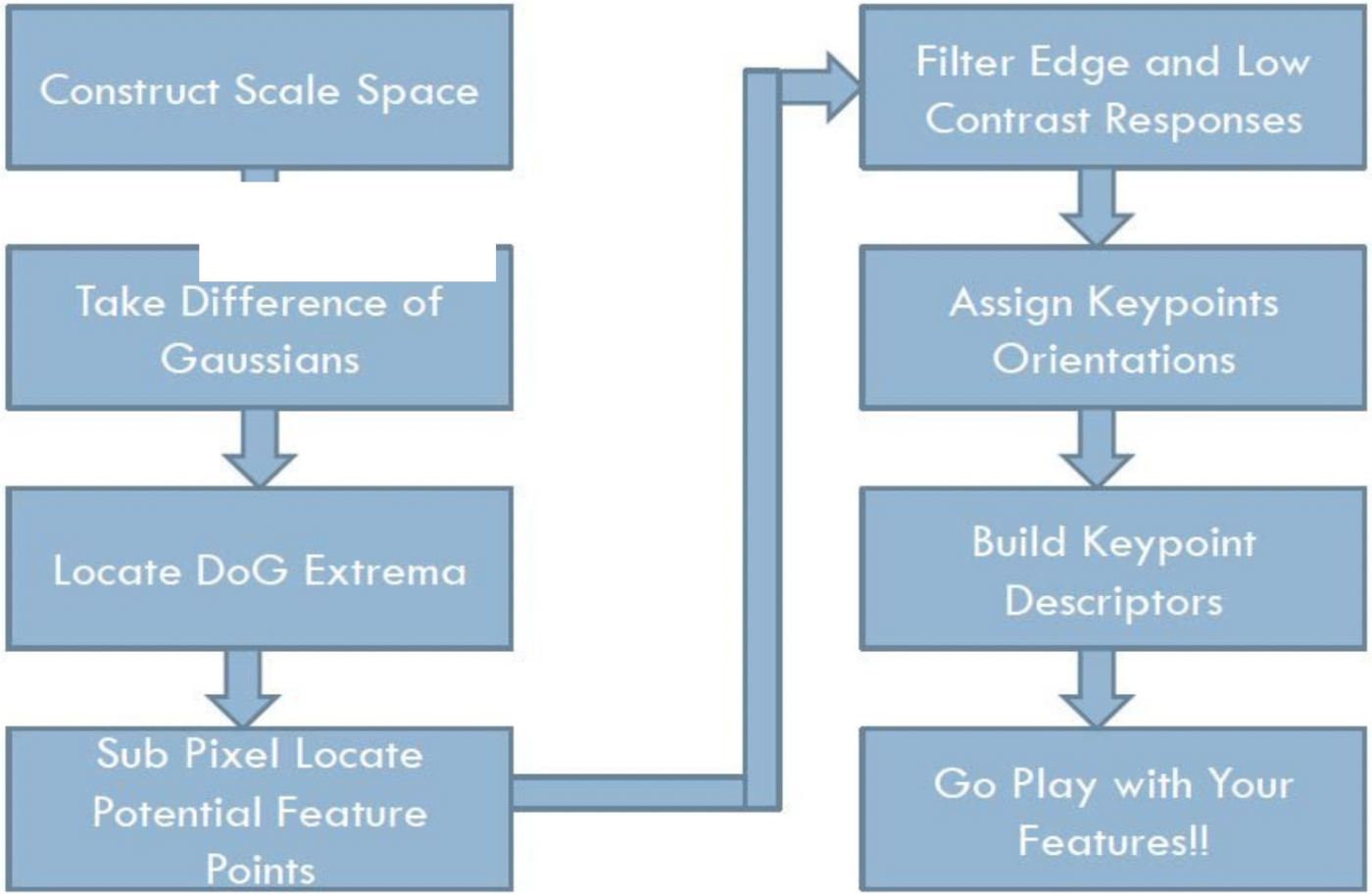
Advantages of Invariant Local Features

- Locality: features are local, so robust to occlusion and clutter (no prior segmentation)
- Distinctiveness: individual features can be matched to a large database of objects
- Quantity: many features can be generated for even small objects
- Efficiency: close to real-time performance
- Extensibility: can easily be extended to wide range of differing feature types, with each adding robustness

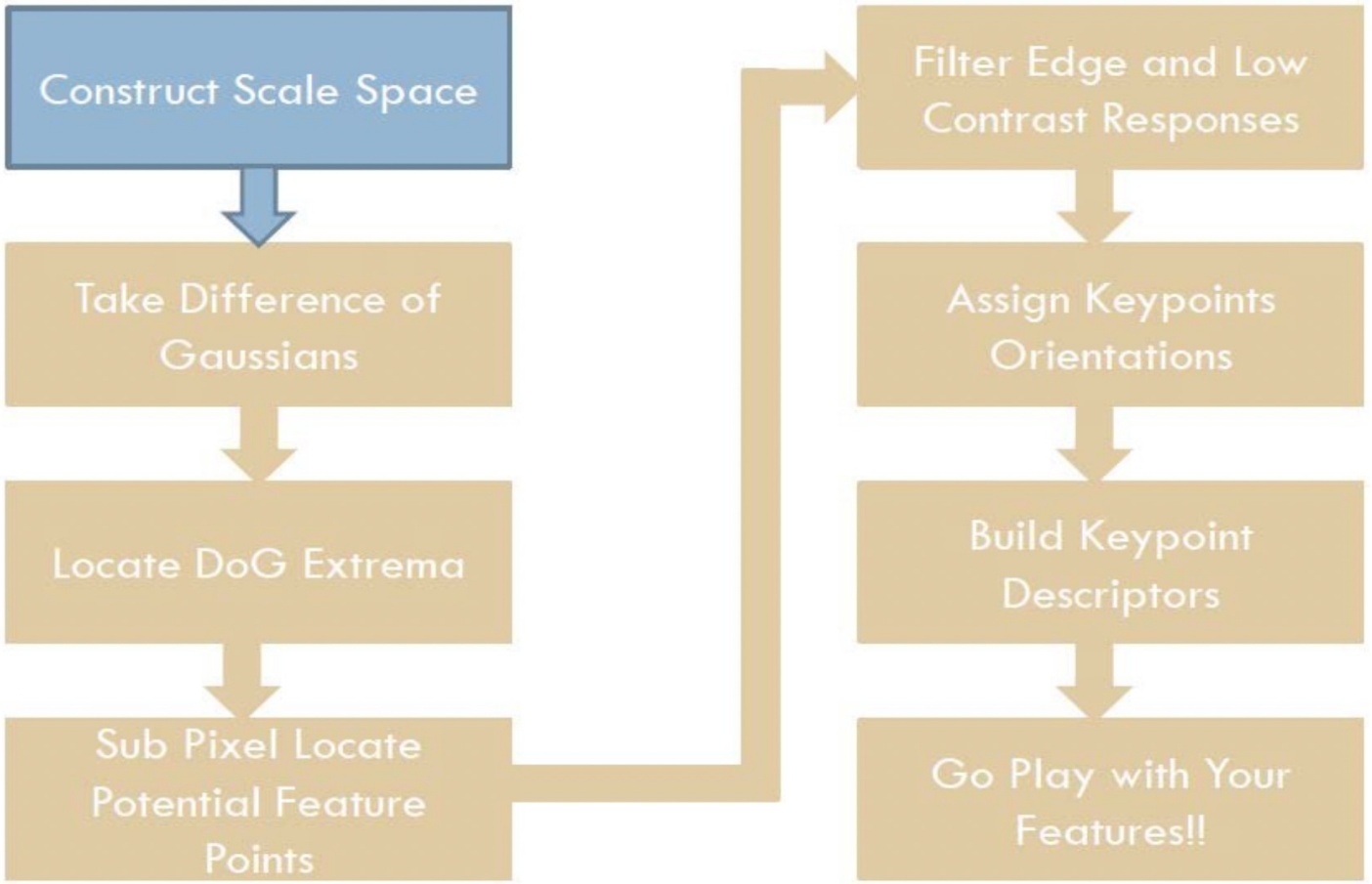
Scale Invariance- Before SIFT

- Requires a method to repeatably select points in location and scale
- The only reasonable scale-space kernel is a Gaussian (Koenderink, 1984; Lindeberg,
- An efficient choice is to detect peaks in the difference of Gaussian pyramid (Burt & Adelson, 1983; Crowley & Parker, 1984 – but examining more scales)
- Difference-of-Gaussian with constant ratio of scales is a close approximation to Lindeberg's scale-normalized Laplacian (can be shown from the heat diffusion equation)

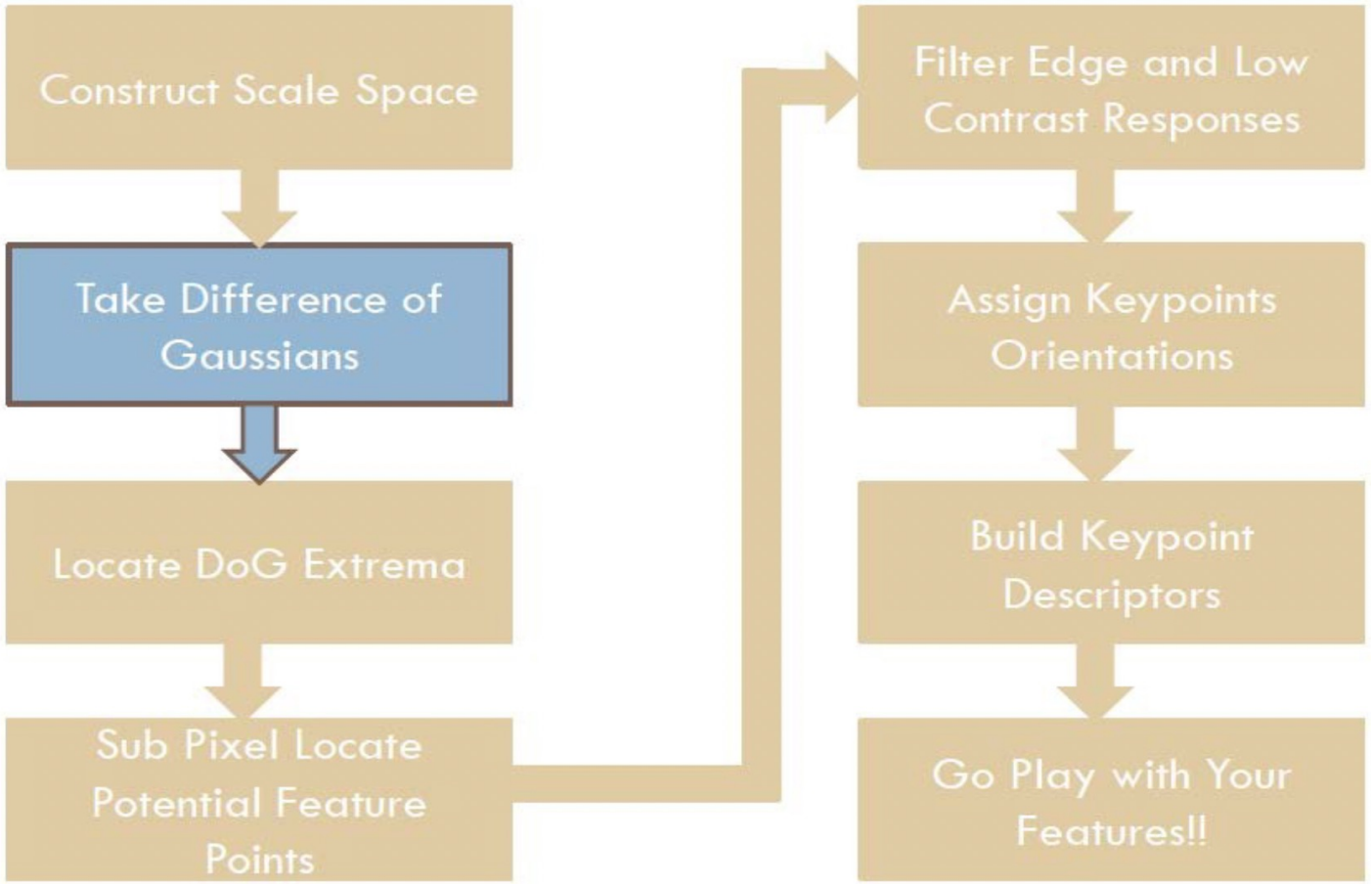
SIFT


Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

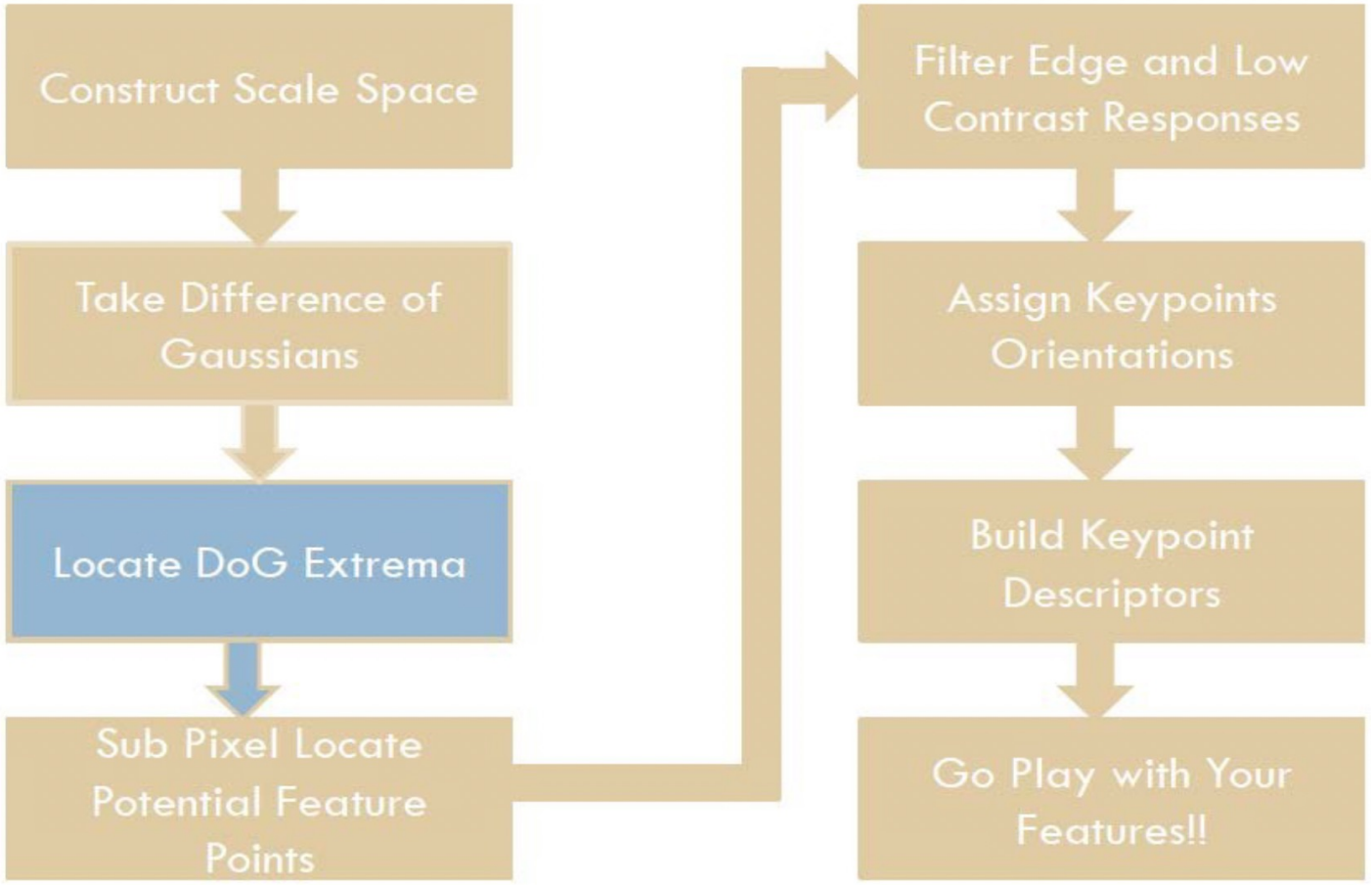
SIFT


Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

SIFT


Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

SIFT

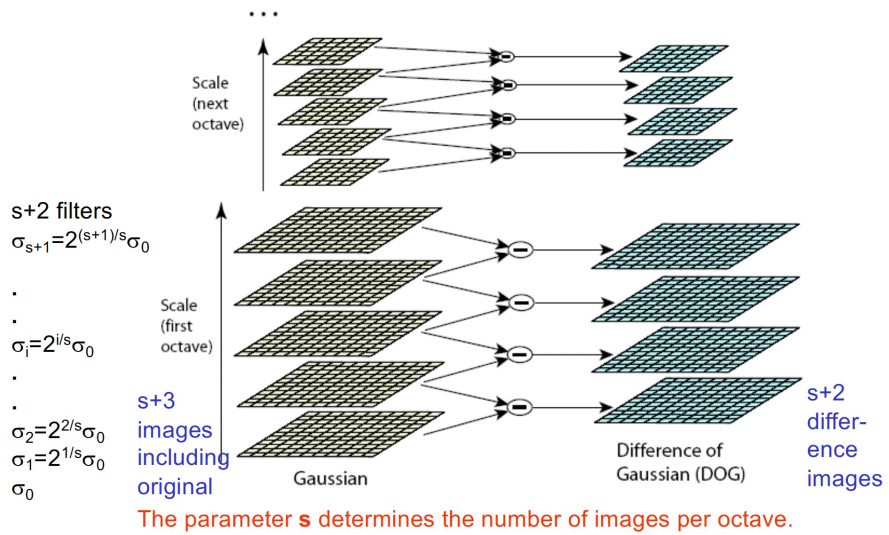

Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

SIFT-Lowe's Pyramid

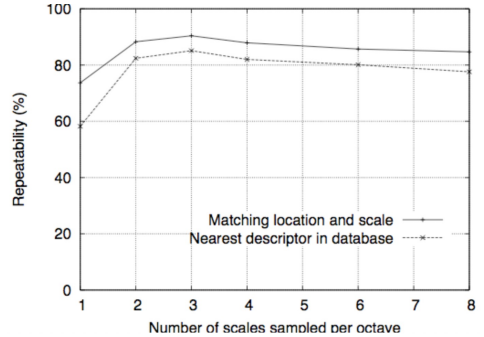
- Scale space is separated into octaves:

1. Octave 1 uses scale s
2. Octave 2 uses scale 2s, etc.

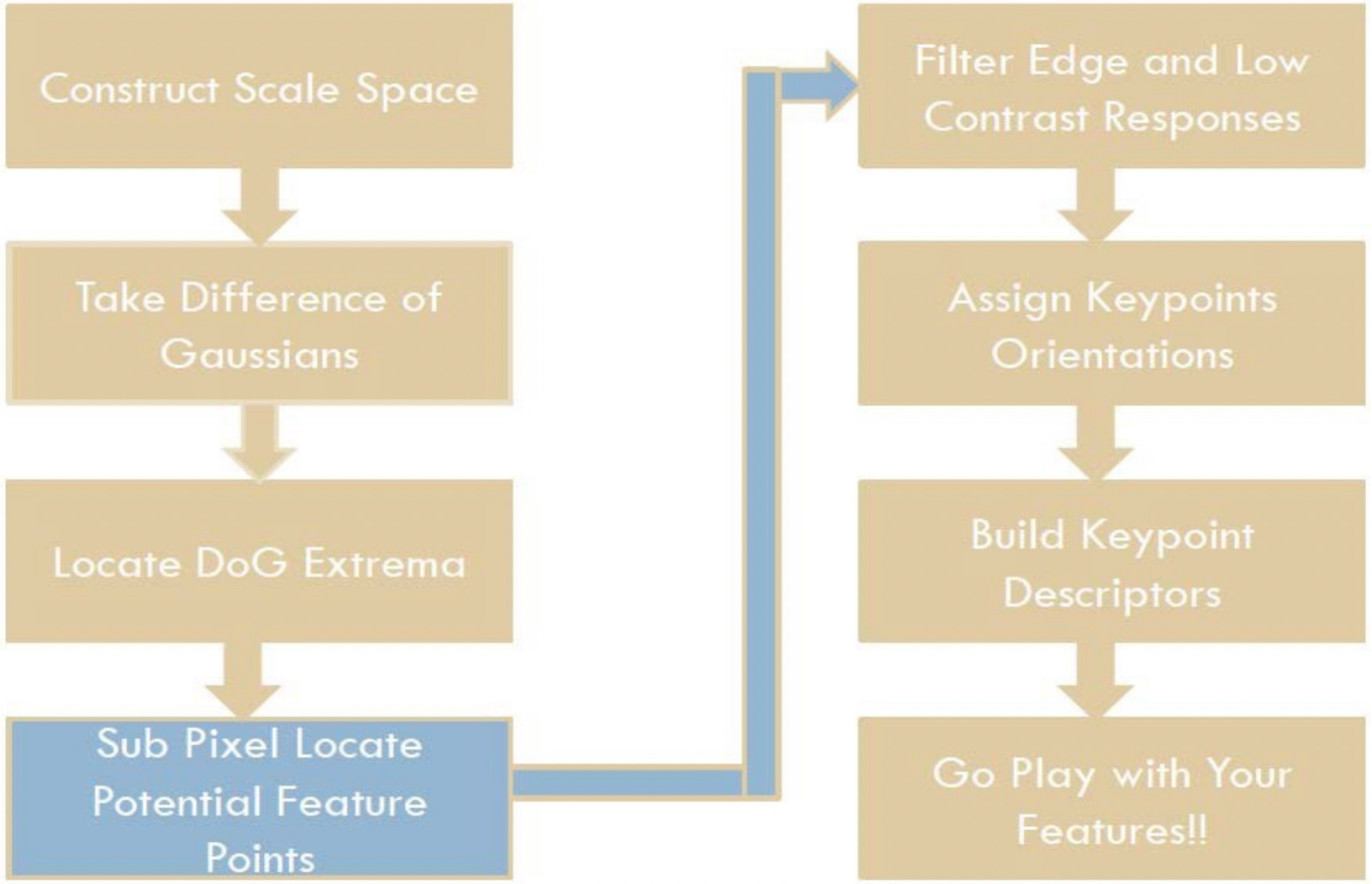
- In each octave, the initial image is repeatedly convolved with Gaussians to produce a set of scale space images.
- Adjacent Gaussians are subtracted to produce the DOG
- After each octave, the Gaussian image is down-sampled by a factor of 2 to produce an image 1/4 the size to start the next level.



S=3, by experimentation

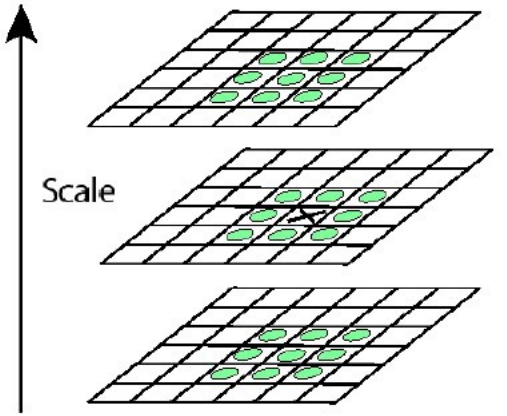


SIFT

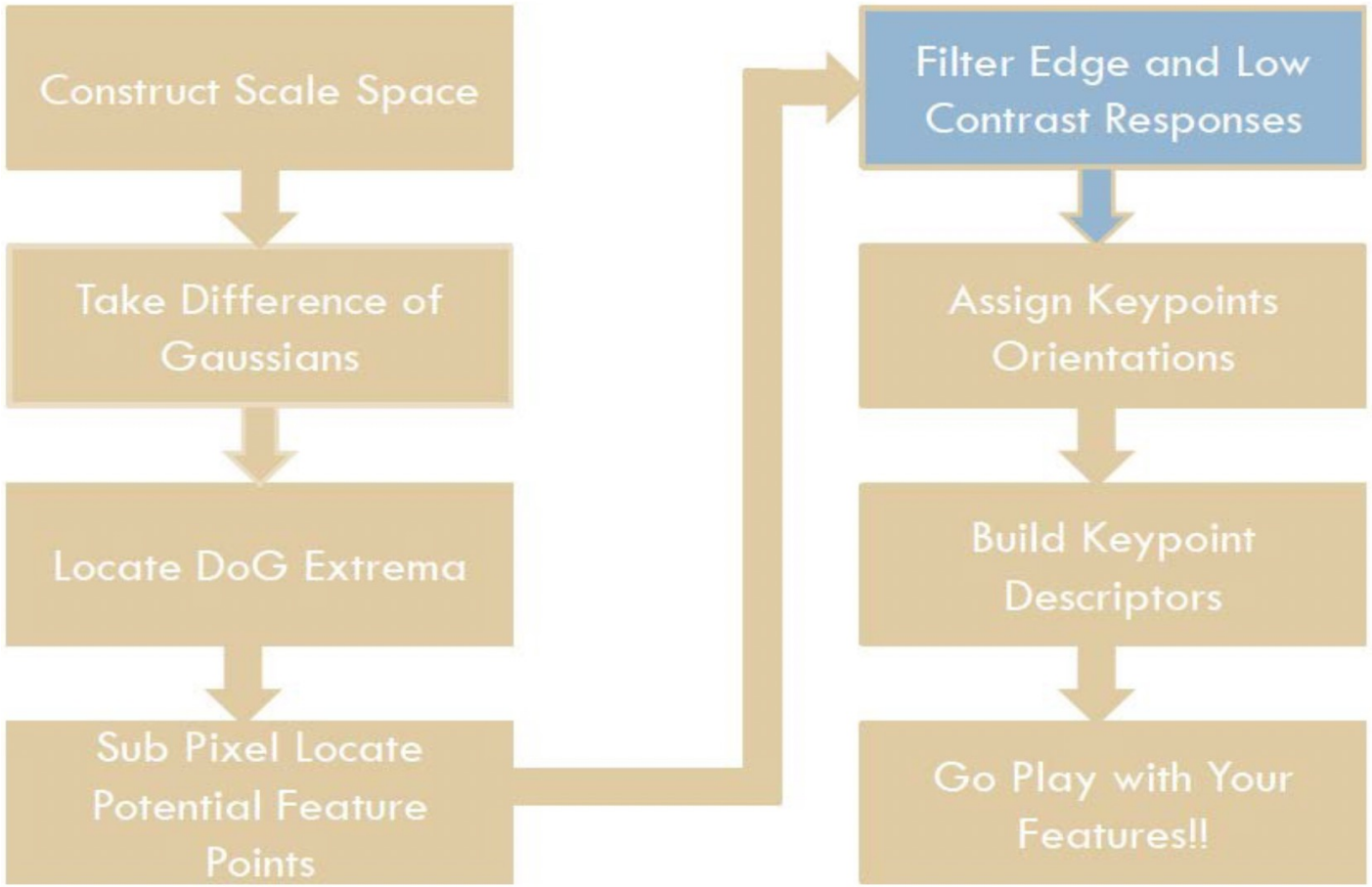

Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

SIFT-Keypoint Localisation

- Scan each DoG image. Look for all neighbouring points including scale.
- Identify min and max- 26 comparisons
- Output both location and scale



SIFT


Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

SIFT-Keypoint Filtering

- Obtain location, scale, and ratio of principal curvatures for each keypoint.
- Unique keypoint found at location and scale of a central sample point or fit a 3D quadratic function to improve interpolation accuracy.
- Use Hessian matrix to eliminate edge responses.

- Reject flats:
  - $|D(\hat{\mathbf{x}})| < 0.03$
- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

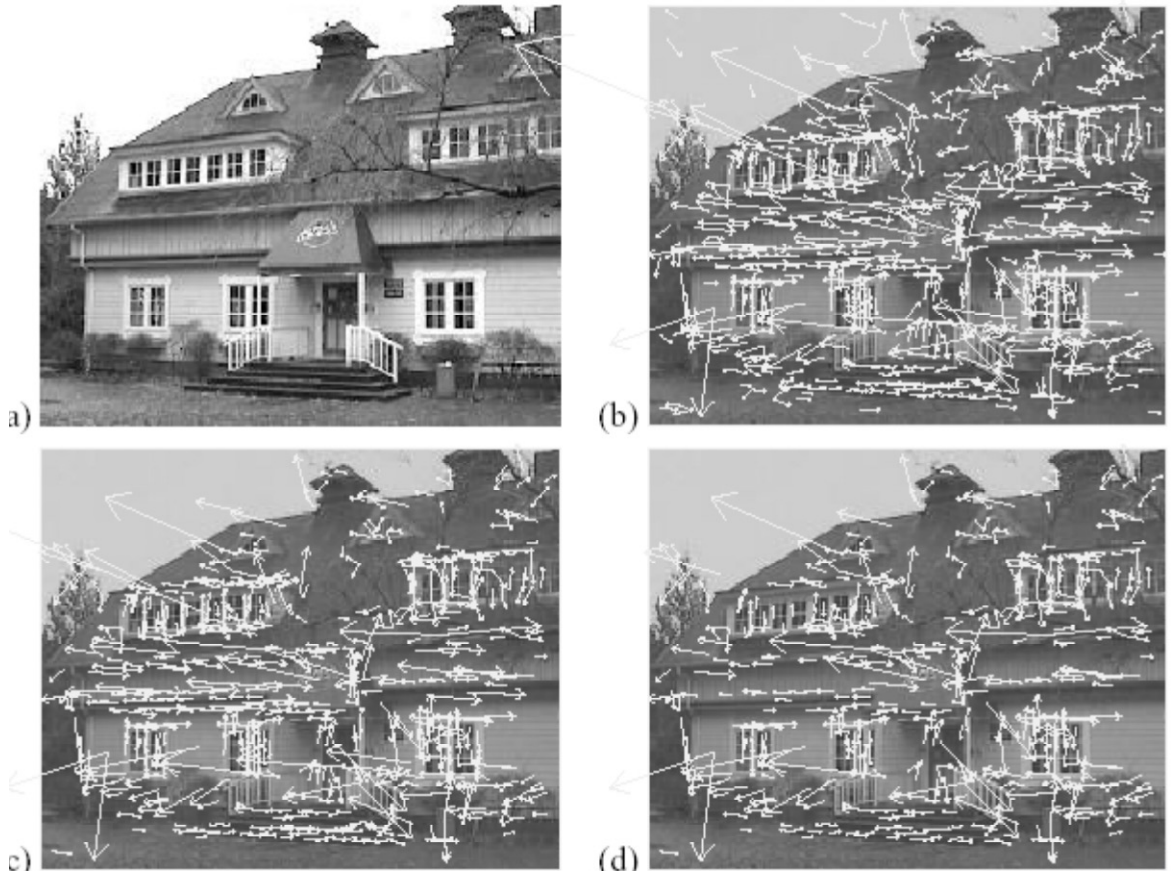
Let $r = \alpha/\beta$. So $\alpha = r\beta$

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

$(r+1)^2/r$ is at a min when the 2 eigenvalues are equal.

  - $r < 10$

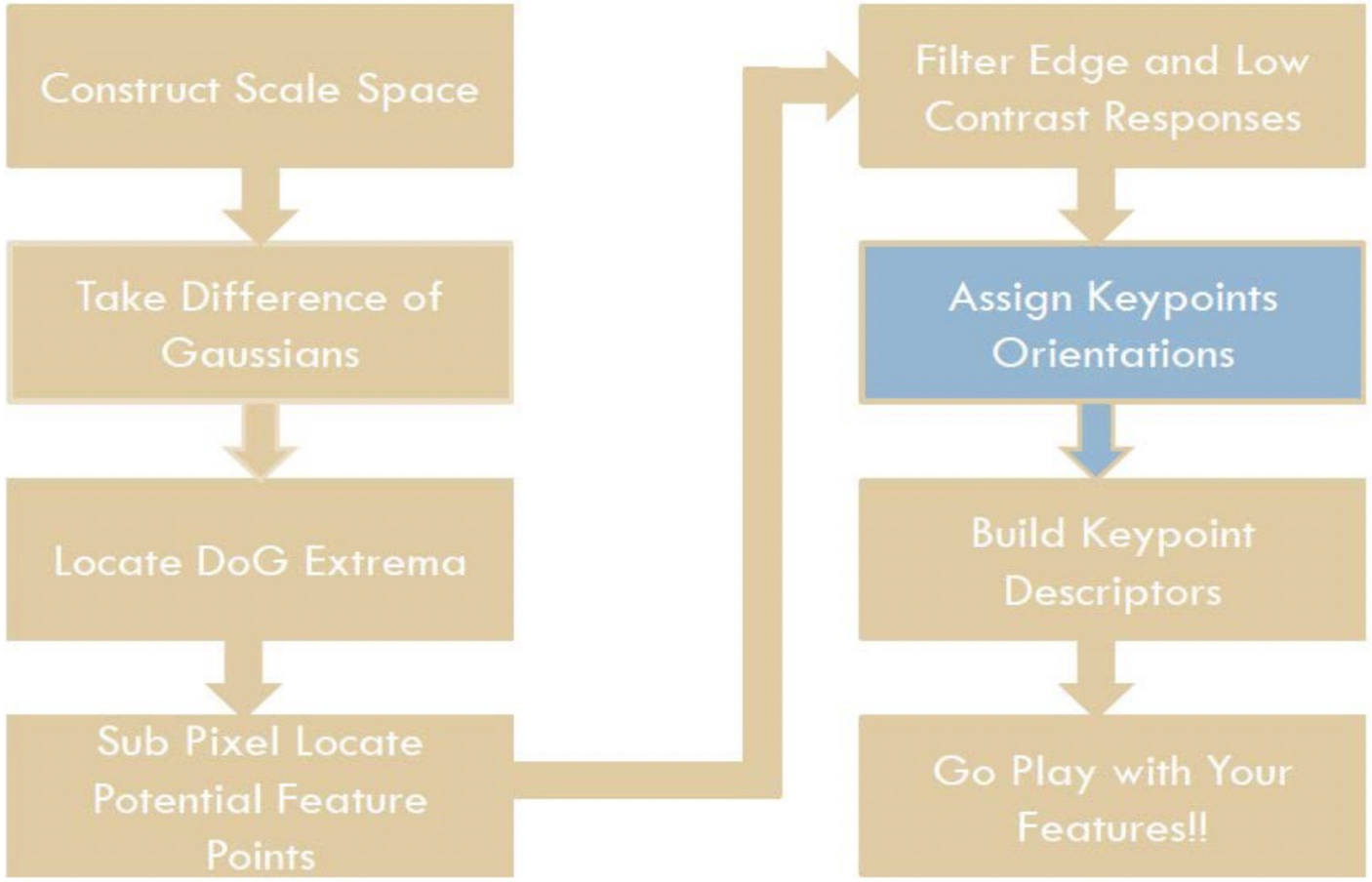
SIFT-Keypoint Filtering



(a) 233x189 image
(b) 832 DOG extrema
(c) 729 left after peak value threshold
(d) 536 left after testing ratio of principle curvatures

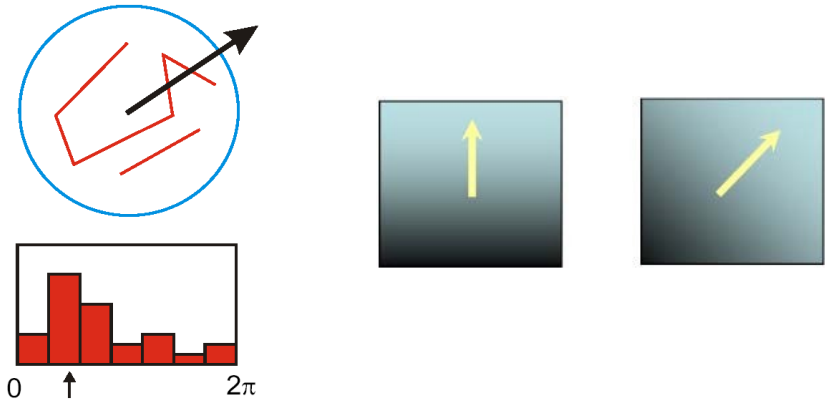
Vectors indicate scale, orientation and location.

SIFT


Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

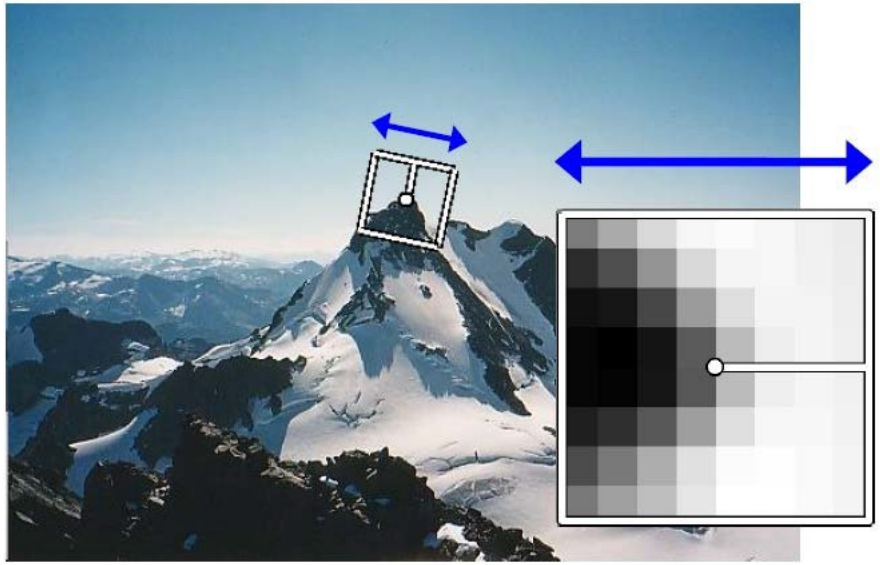
SIFT-Keypoint Orientations

- Compute orientation histogram
- Select dominant orientation. If 2 major orientations, keep both.

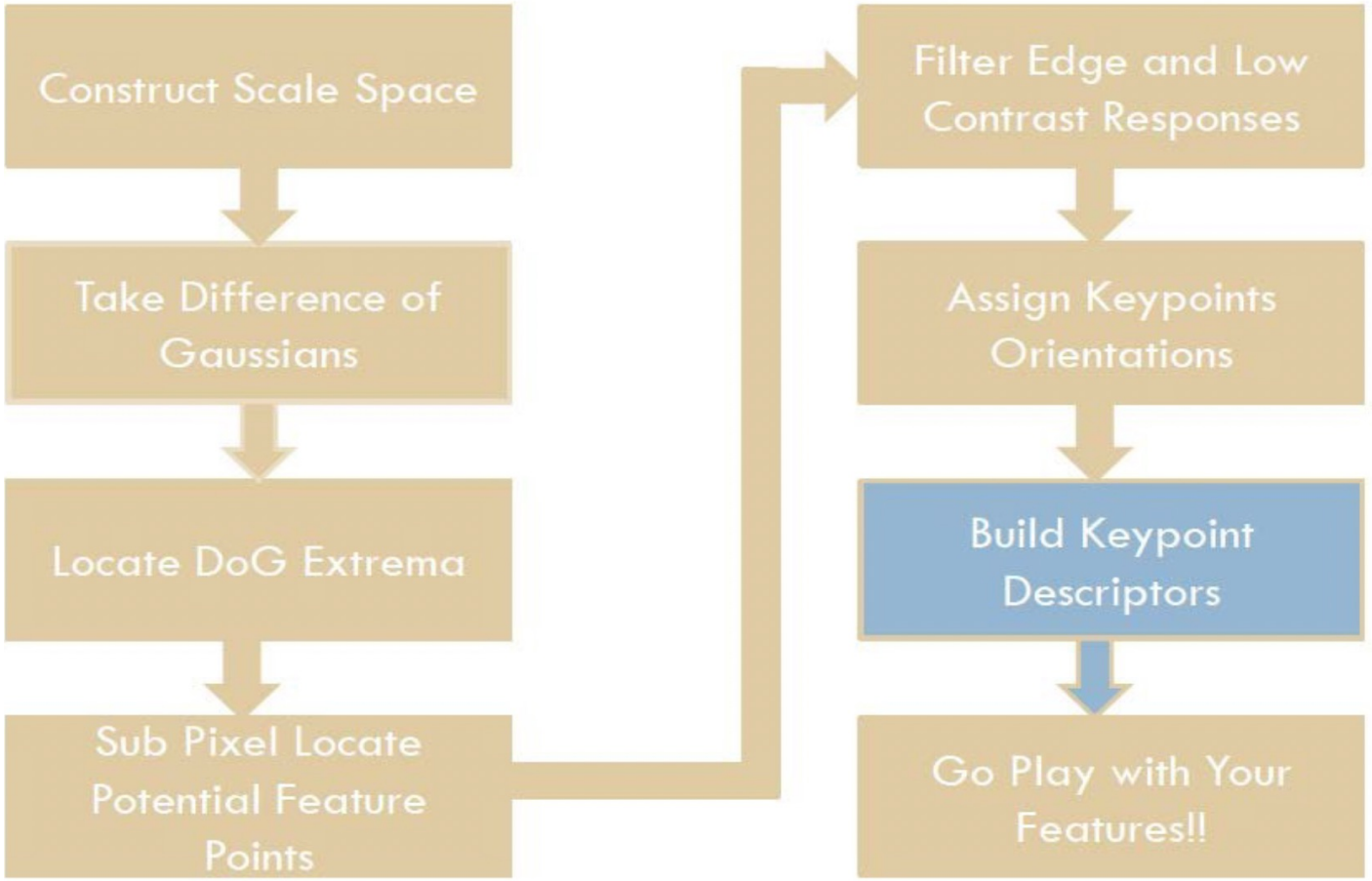


- Normalization: rotate the patch to the selected orientation

Now, keypoints have location, scale and orientation. Next, compute a descriptor that is distinctive and invariant to changes in viewpoints and illumination



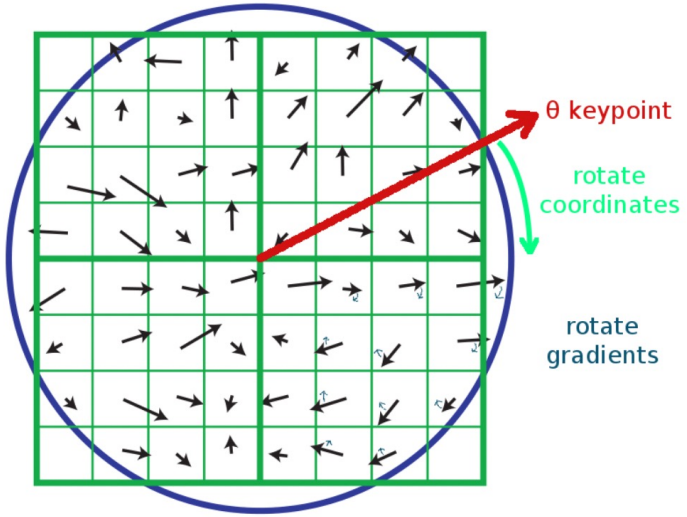
SIFT


Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

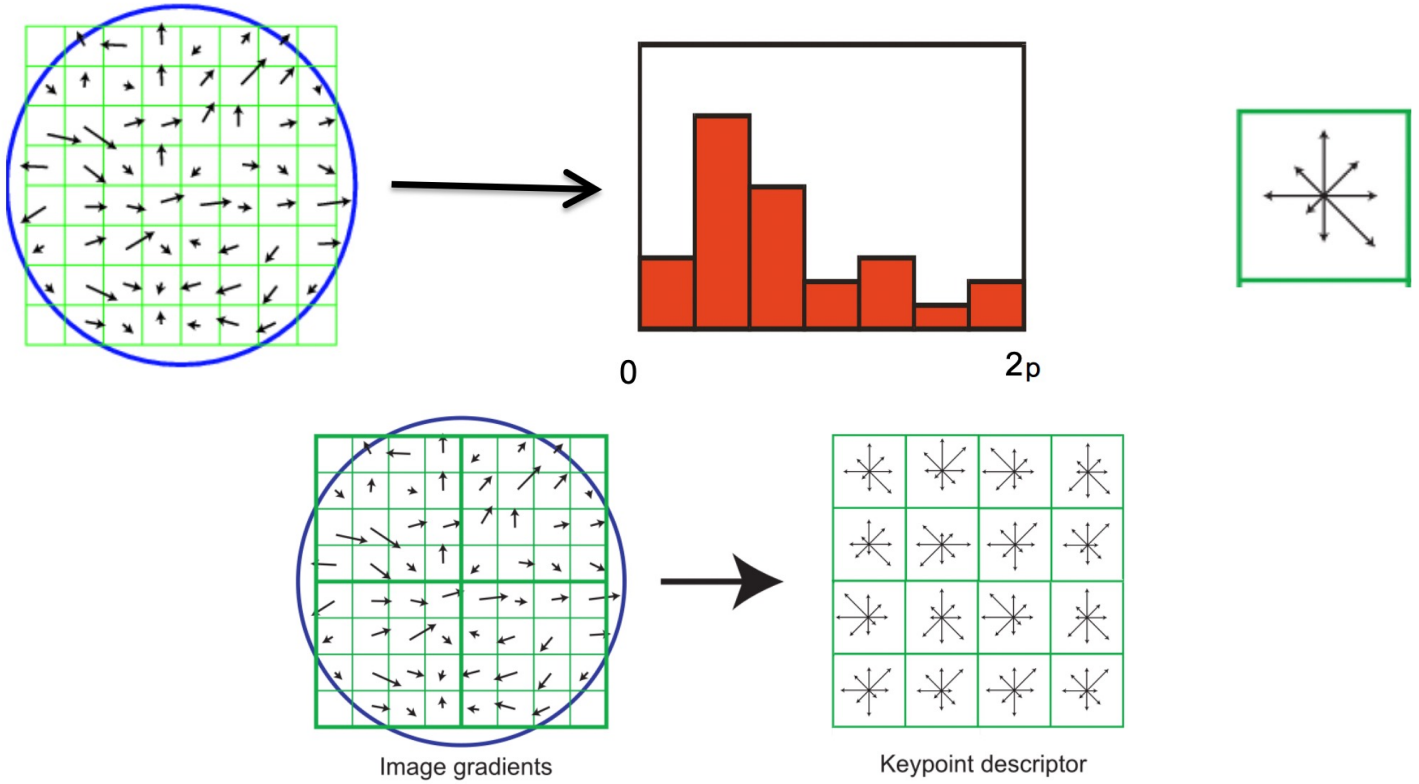
SIFT-Descriptor

- Use the blurred image associated with the keypoint's scale
- Take image gradients over the keypoint neighborhood.
- To become rotation invariant, rotate the gradient directions AND locations by (-keypoint orientation)

Now we've cancelled out rotation and have gradients expressed at locations relative to keypoint orientation θ



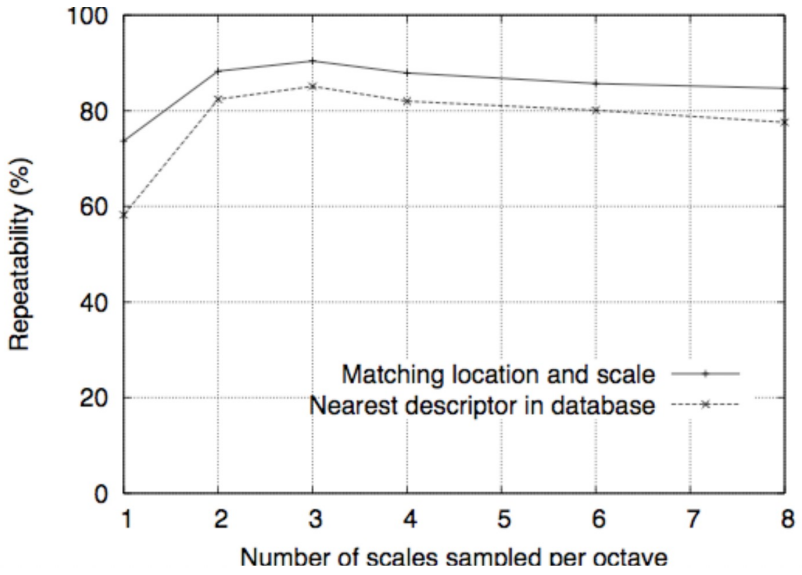
SIFT Descriptor

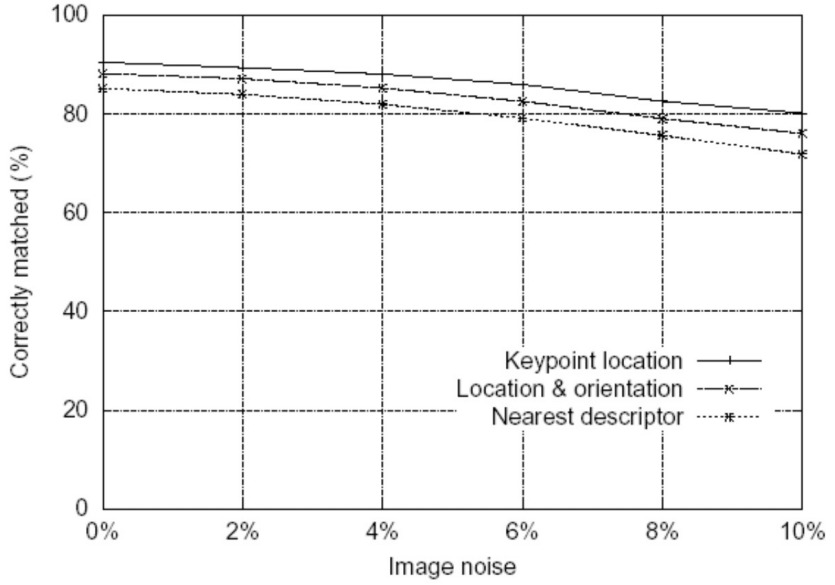


- Don’t use precise locations. Use blobs
- Divide into 4X4 regions and create histograms
- Put the rotated gradients into their local orientation histograms (8 bins)

SIFT Descriptor

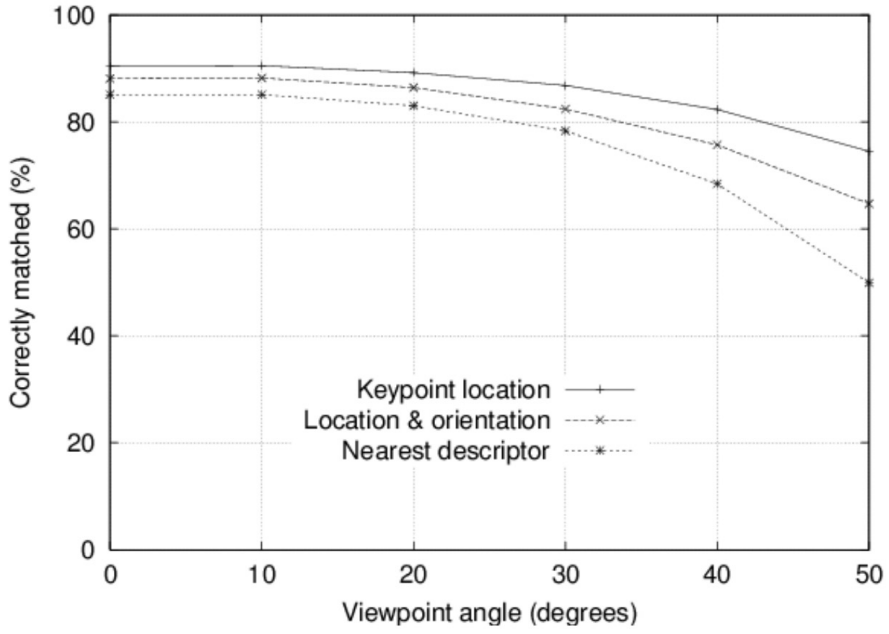
- 8 orientation bins per histogram, and a 4x4 histogram array, yields 8 x 4x4 = 128 numbers.
- Find correspondences by minimizing Euclidean Distances

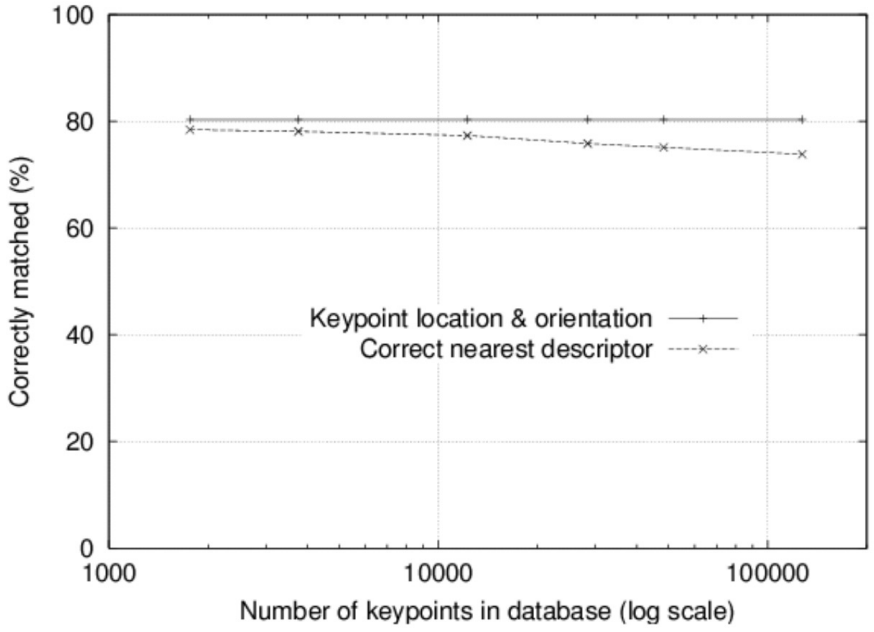




SIFT Descriptor

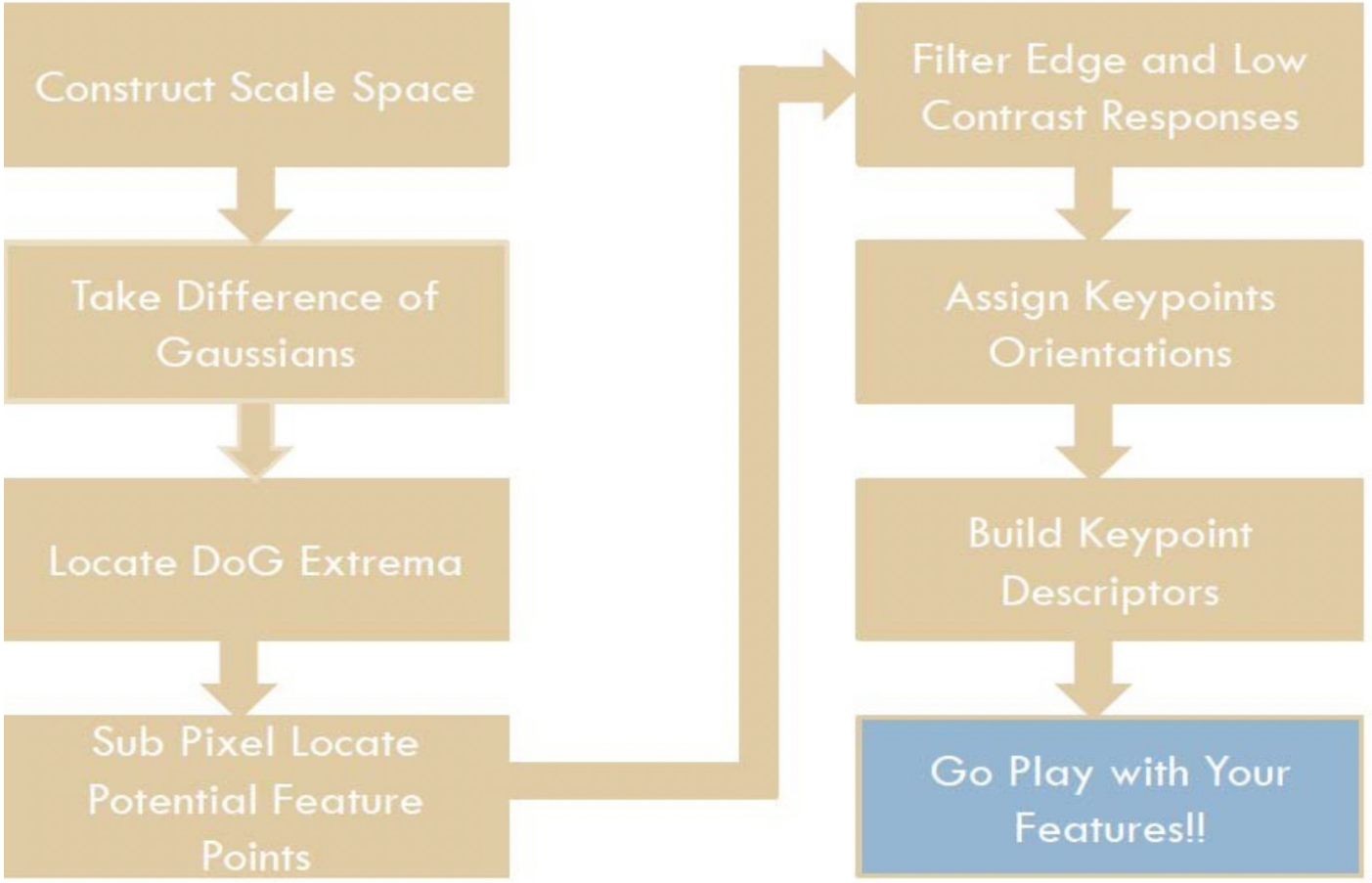
- 8 orientation bins per histogram, and a 4x4 histogram array, yields 8 x 4x4 = 128 numbers.
- Find correspondences by minimizing Euclidean Distances



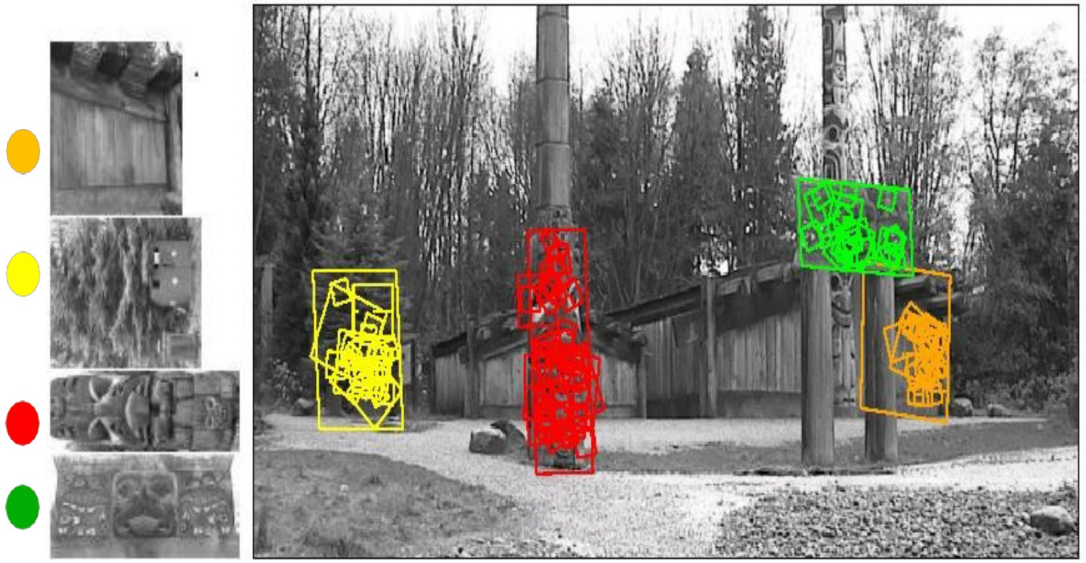


Affine change upto 20 degrees is ok

SIFT


Construct Scale Space
Take Difference of Gaussians
Locate DoG Extrema
Sub Pixel Locate Potential Feature Points
Filter Edge and Low Contrast Responses
Assign Keypoints Orientations
Build Keypoint Descriptors
Go Play with Your Features!!

SIFT Descriptor



Feature Matching

Given a feature in I1, how to find the best match in I2?

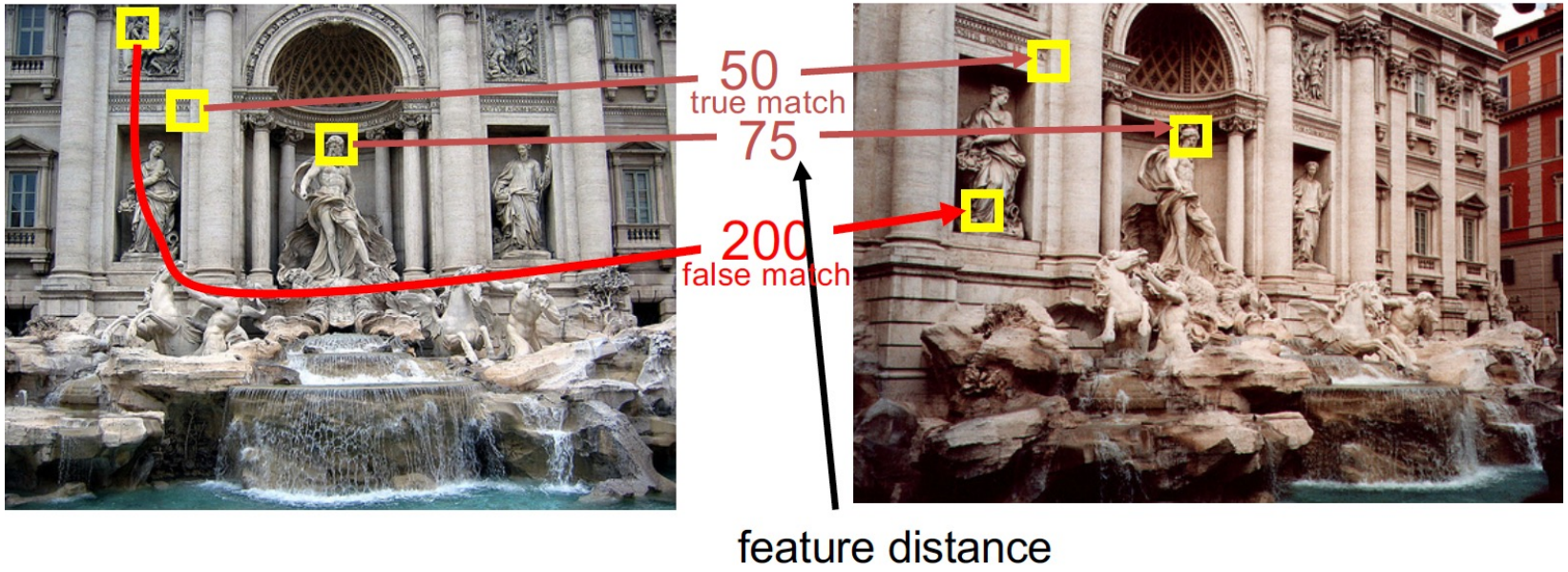
1. Define distance function that compares two descriptors
2. Test all the features in I2, find the one with min distance

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

I_1 I_2

Feature Matching- Evaluating Performance

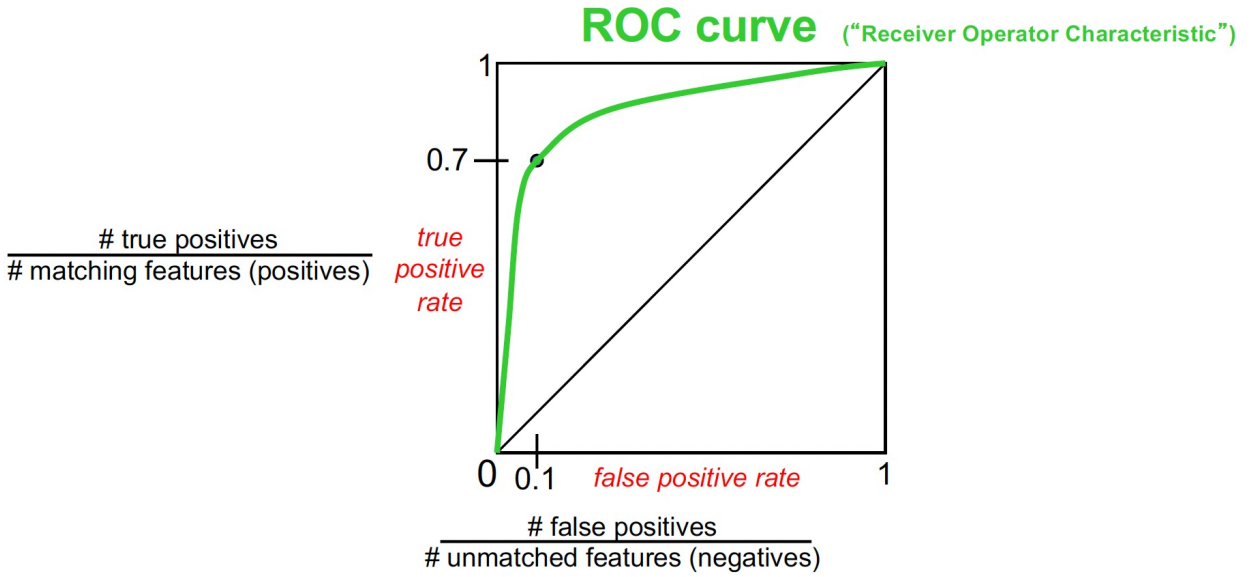
The distance threshold affects performance
– True positives = # of detected matches that are correct
– False positives = # of detected matches that are incorrect



Feature Matching- Evaluating Performance

ROC Curves

- Generated by counting # current/incorrect matches, for different thresholds
- 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



SIFT-Implementation

VLFeat.org

Search

Home Download Tutorials Applications Documentation

Tutorials > SIFT detector and descriptor

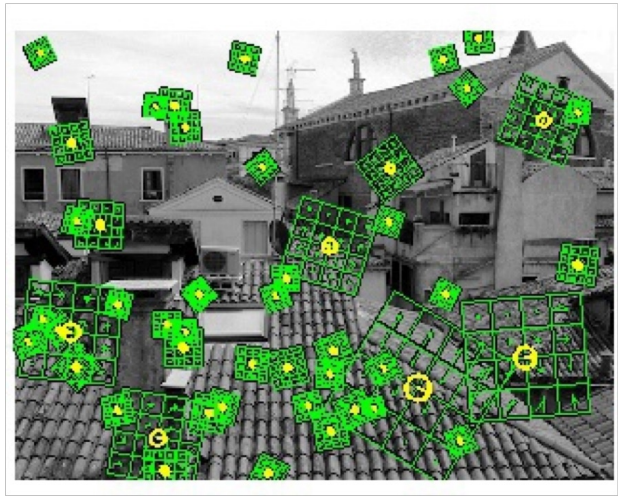
The *Scale-Invariant Feature Transform (SIFT)* bundles a feature detector and a feature descriptor. The detector extracts from an image a number of frames (attributed regions) in a way which is consistent with (some) variations of the illumination, viewpoint and other viewing conditions. The descriptor associates to the regions a signature which identifies their appearance compactly and robustly. For a more in-depth description of the algorithm, see our API reference for SIFT.

Table of Contents

Extracting frames and descriptors

Both the detector and descriptor are accessible by the **vl_sift** MATLAB command (there is a similar command line utility). Open MATLAB and load a test image

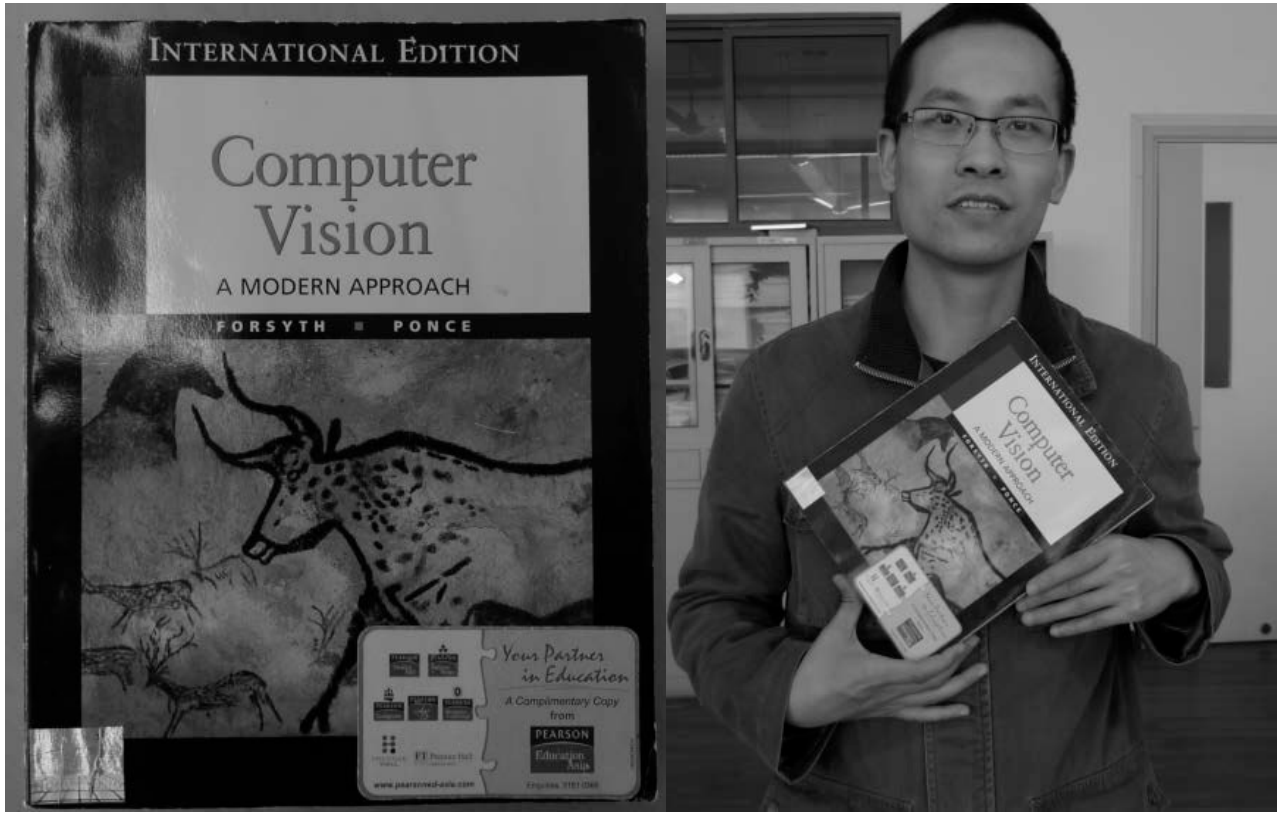
```
I = vl_impattern('roofs1') ;
image(I) ;
```



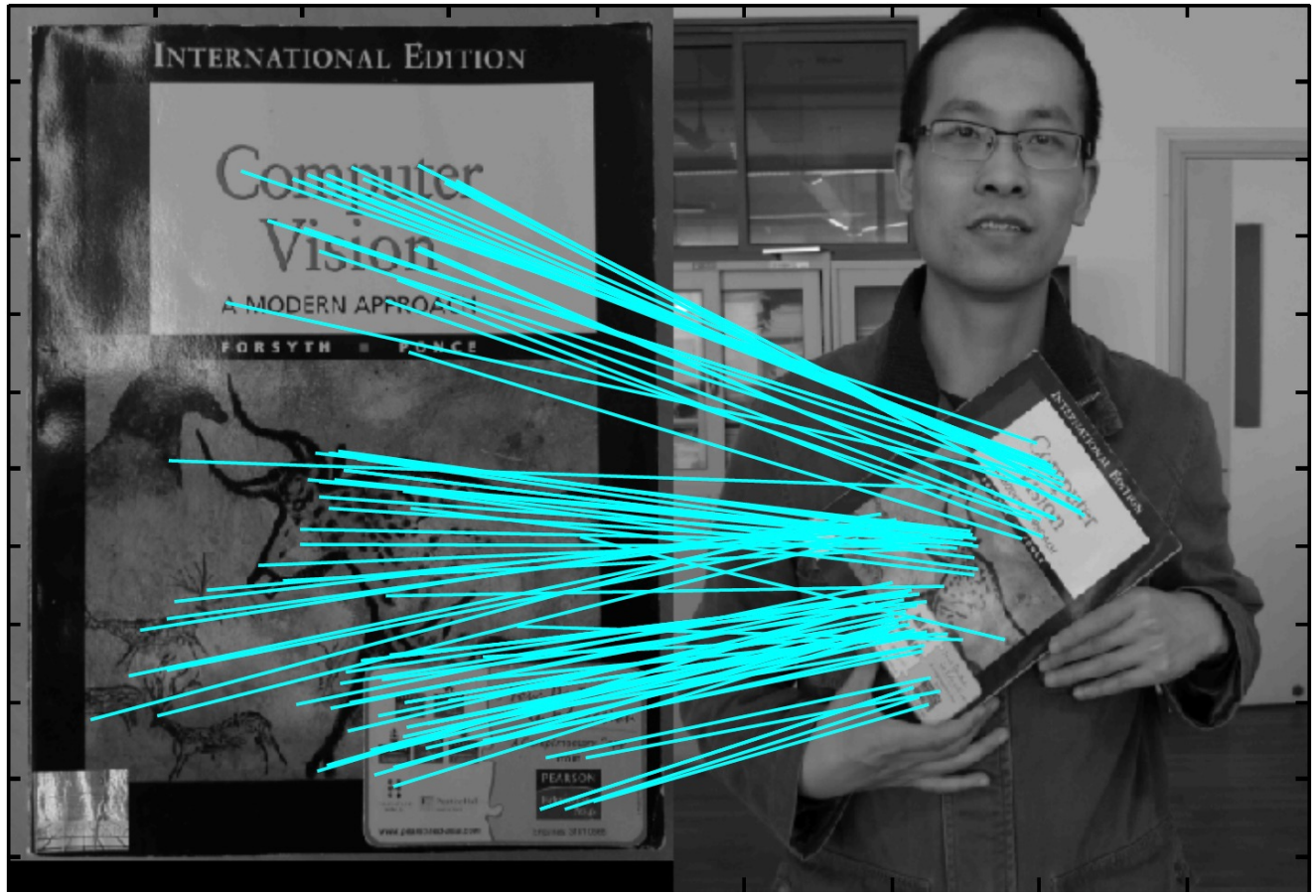
Applications

- Object recognition
- Face Analysis and Recognition
- Robot localization and mapping
- Panorama stitching
- 3D scene modeling, recognition and tracking
- Analyzing the human brain in 3D magnetic resonance images

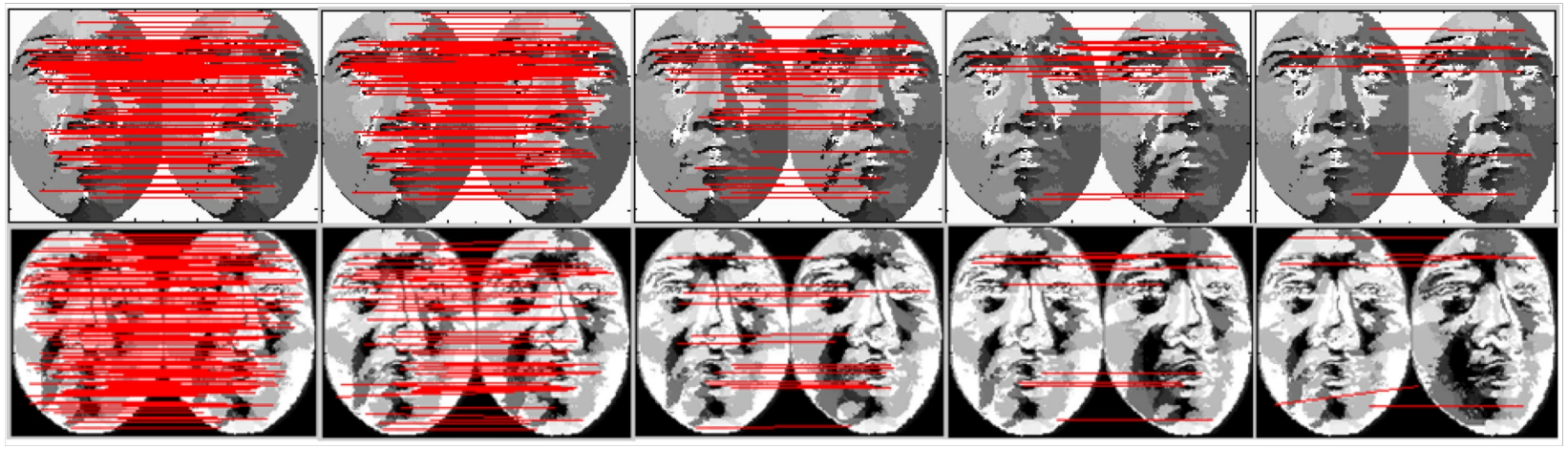
Applications-Object Detection



Applications-Object Detection

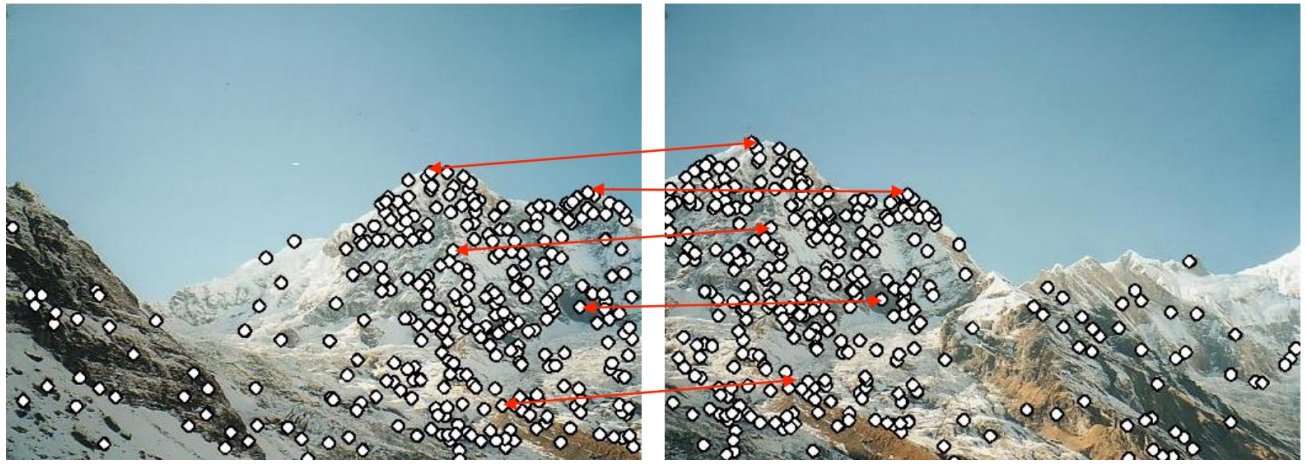


Applications-Face Recognition



D.Huang, M.Ardabilian, Y.Wang, L.Chen, IEEE TIFS, 2012

Applications-Image Stitiching



Find matches and estimate transformation between images
We will learn this in the class today

C. Feature Tracking and Optical Flow

Instructor: Shaifali Parashar

Many Slides from Fei-Fei Li's lectures at Standford

Feature Tracking & Optical Flow

Feature tracking: Extract visual features (corners, textured areas) and “track” them over multiple frames

Optical Flow: Recover image motion at each pixel from spatio-temporal image brightness variations

Optical flow = apparent motion of brightness patterns

Optical Flow

Why bother with motion?

G. Johansson, “Visual Perception of Biological Motion and a Model For Its Analysis", *Perception and Psychophysics 14, 201-211, 1973.*

Feature Tracking & Optical Flow

Challenges:

1. Figure out which features can be tracked
2. Efficiently track across frames
3. Some points may change appearance over time (e.g., due to rotation, moving into shadows, etc.)
4. Drift: small errors can accumulate as appearance model is updated
5. Points may appear or disappear: need to be able to add/delete tracked points

Feature Tracking & Optical Flow

Feature tracking: Extract visual features (corners, textured areas) and “track” them over multiple frames

Problems: Sparse, inexact feature alignment, low accuracy

Optical Flow: Recover image motion at each pixel from spatio-temporal image brightness variations

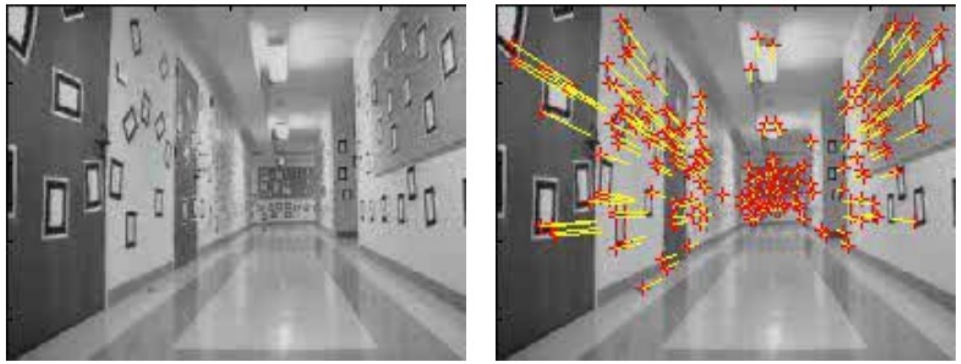
Advantage: Scale/rotation invariant, lighting invariant, can handle large movements (not really, not accurately for sure)

Registration using Lucas-Kanade Method

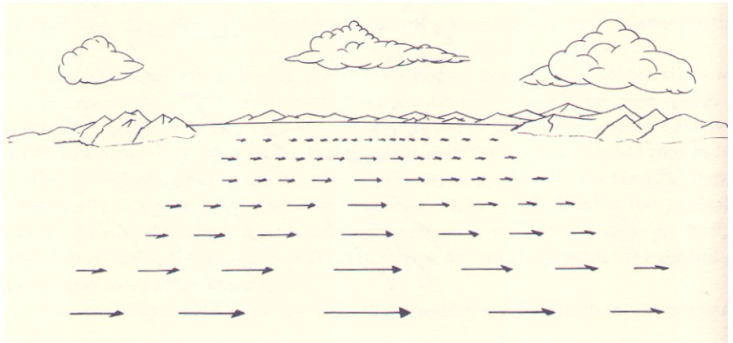
An iterative image registration technique with an application to stereo vision

Feature Tracking & Optical Flow

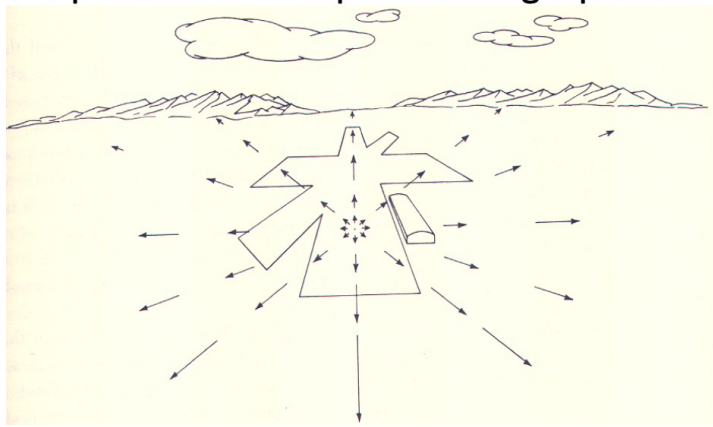
Feature-tracking



Optical flow from the side window of a car

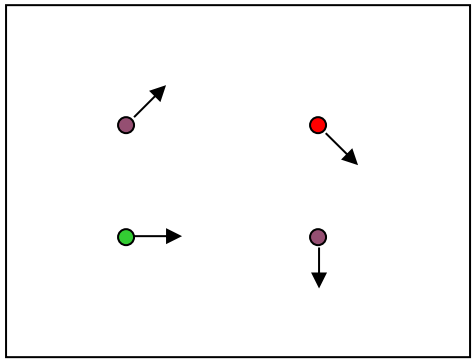


Optical flow for a pilot landing a plane

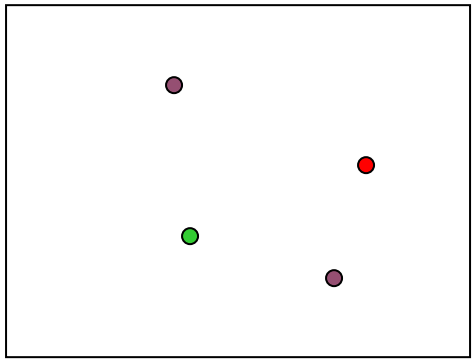


Applications:
3D reconstruction, Object segmentation, Motion Segmentation, Tracking, Super-resolution, Activity recognition

Optical Flow



$I(x,y,t)$



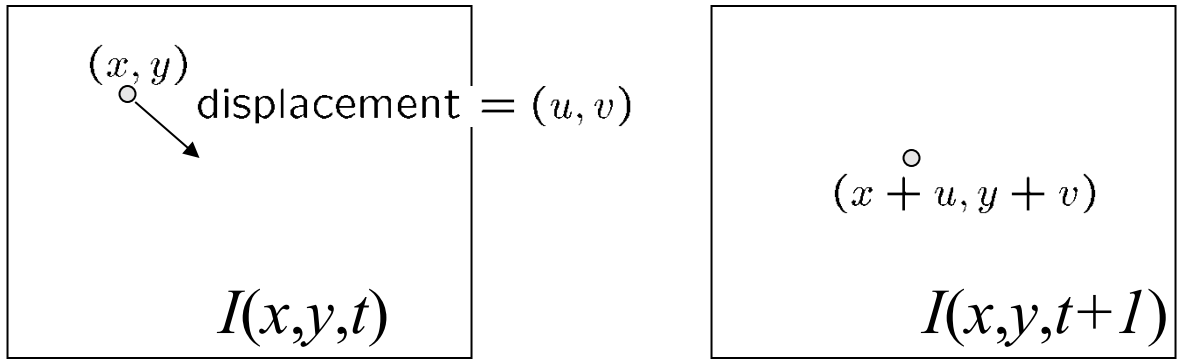
$I(x,y,t+1)$

Given two subsequent frames, estimate the point translation

Key assumptions of Lucas-Kanade Tracker

- **Brightness constancy:** projection of the same point looks the same in every frame
- **Small motion:** points do not move very far
- **Spatial coherence:** points move like their neighbors

Optical Flow: Brightness Constancy



Brightness Constancy Equation: $I(x,y,t) = I(x+u, y+v, t+1)$

Take Taylor expansion of *I(x+u, y+v, t+1)* at *(x,y,t)* to linearize the right side:

Image derivative along x Difference over frames

$$I(x+u,y+v,t+1) \approx I(x,y,t) + I_x \cdot u + I_y \cdot v + I_t$$

$$I(x+u,y+v,t+1) - I(x,y,t) = I_x \cdot u + I_y \cdot v + I_t$$

So: $$I_x \cdot u + I_y \cdot v + I_t \approx 0 \quad \rightarrow \nabla \mathrm{I} \cdot \left[\mathrm{u}\ \mathrm{v}\right]^{\mathrm{T}} + \mathrm{I_t} = 0$$

Optical Flow: Brightness Constancy

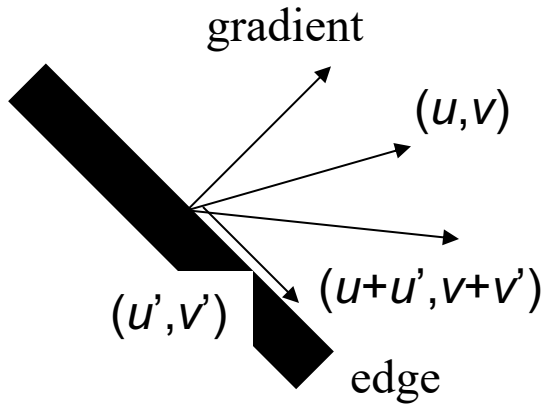
$$\nabla I \cdot \begin{bmatrix} u & v \end{bmatrix}^T + I_t = 0$$

1 equation, 2 unknowns (u,v), can't solve

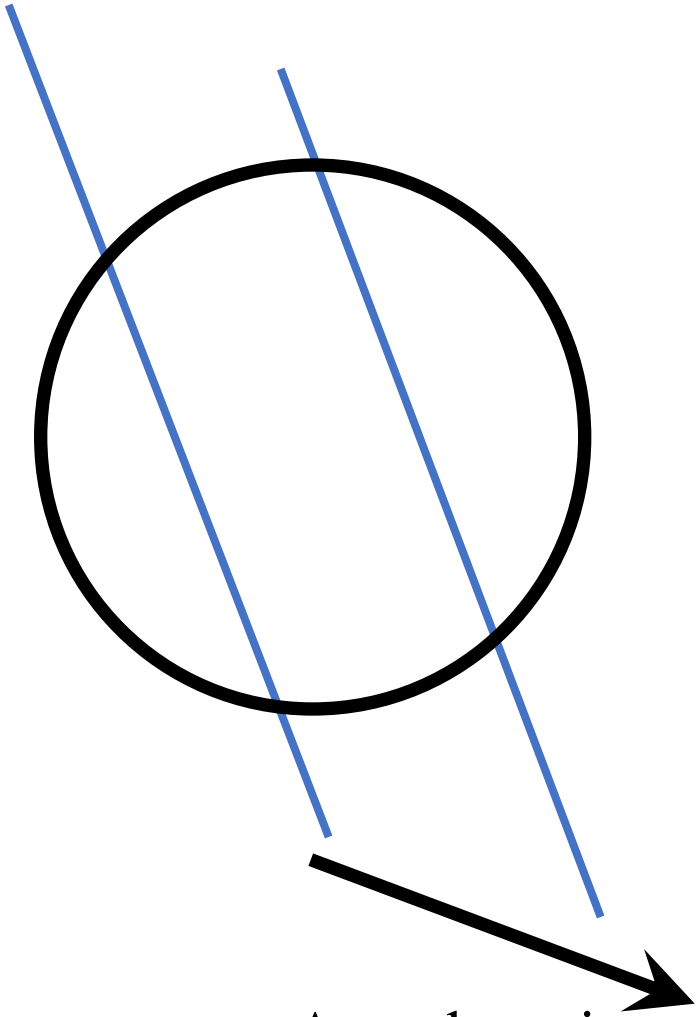
The component of the motion perpendicular to the gradient (i.e., parallel to the edge) cannot be measured

If (u, v) satisfies the equation, so does (u+u', v+v') if

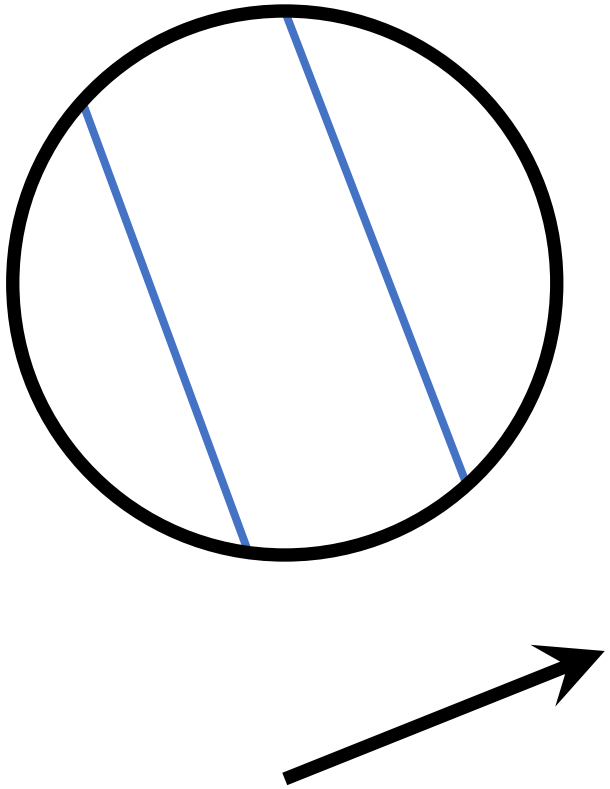
$$\nabla I \cdot \begin{bmatrix} u' & v' \end{bmatrix}^T = 0$$



Optical Flow: Aperture Problem

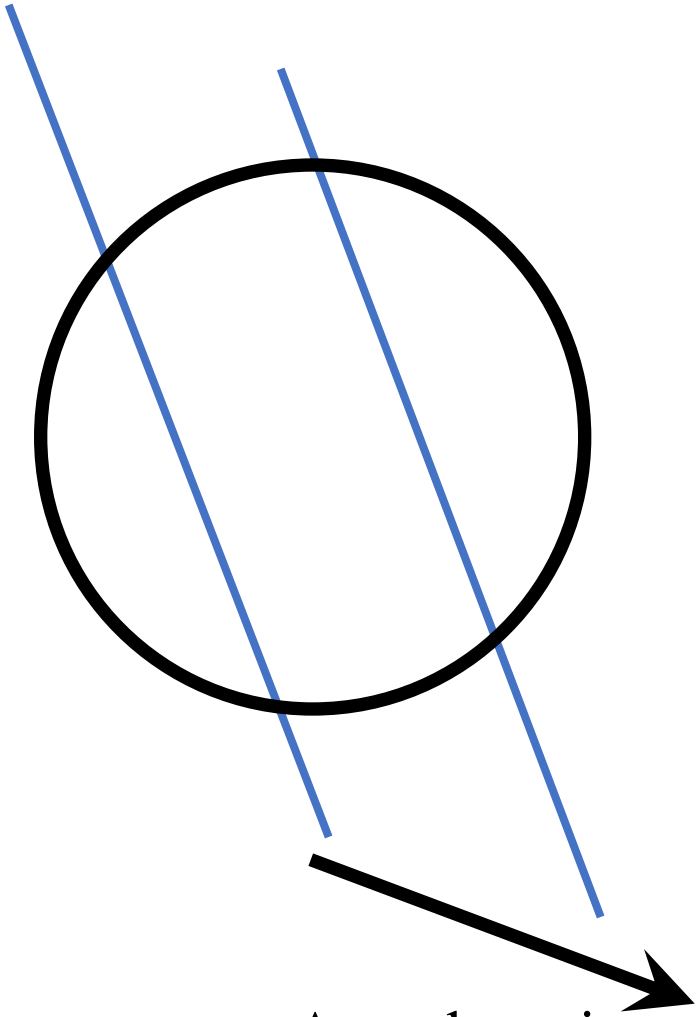


Actual motion

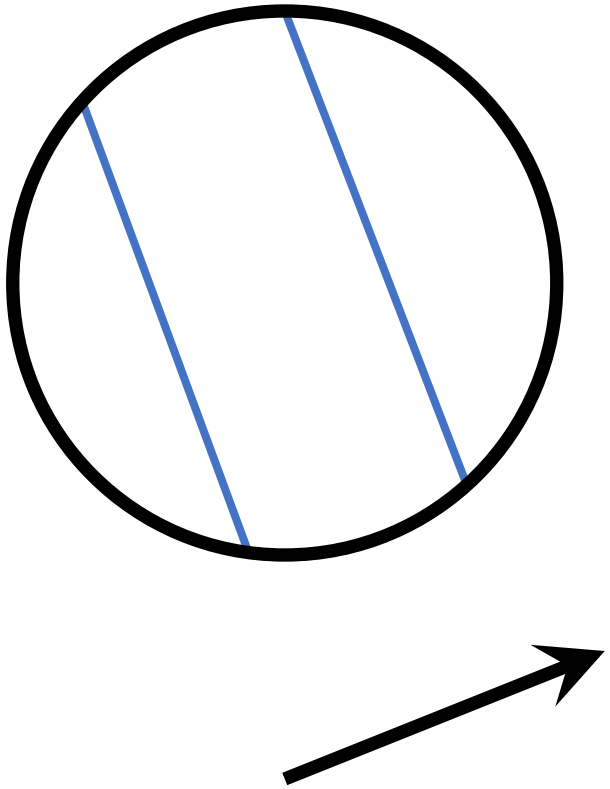


Perceived motion

Optical Flow: Aperture Problem



Actual motion



Perceived motion

Solution:
Look at neighbors

Optical Flow: Barber Pole Illusion

http://en.wikipedia.org/wiki/Barberpole_illusion

Optical Flow: Barber Pole Illusion

http://en.wikipedia.org/wiki/Barberpole_illusion

Optical Flow: Solving the issue

Get more equations for a pixel.

Spatial coherence constraint: Assume the pixel's neighbors have the same (u,v)

If we use a 5x5 window, that gives us 25 equations per pixel

$$0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u\ v]$$

$$\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix}$$

$$\underset{25\text{x}2}{A}\ \underset{2\text{x}1}{d} = \underset{25\text{x}1}{b}$$

$$(A^T A)\ d = A^T b$$

Optical Flow: Solving the issue

When is this solvable? I.e., what are good points to track? Remember: Harris Corners

- $\mathbf{A^TA}$ should be invertible
- $\mathbf{A^TA}$ should not be too small due to noise
  - eigenvalues λ_1 and λ_2 of $\mathbf{A^TA}$ should not be too small
- $\mathbf{A^TA}$ should be well-conditioned
  - λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

Low-texture region

– gradients are small magnitude
– small eigenvalues

Edge

– gradients are large and small
– large and small eigenvalue

High-texture region

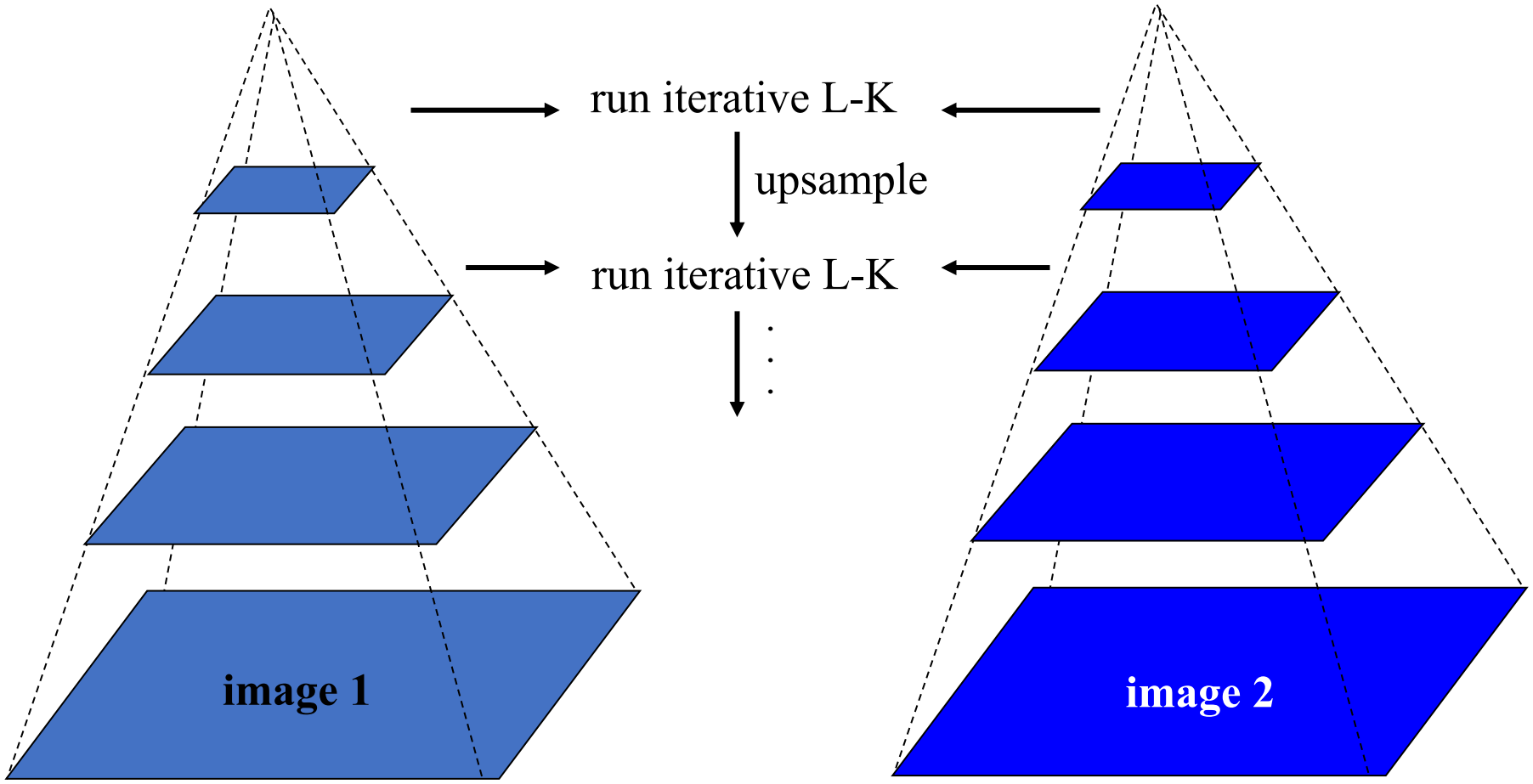
– gradients are different, large magnitudes
– large eigenvalues

Optical Flow- Lucas-Kanade

Assumptions:

1. Brightness consistency
2. Spatial coherence
3. Small motion (not really!)

Dealing with larger movements: coarse-to-fine registration



Gaussian pyramid of image 1 (t) **Gaussian pyramid of image 2 (t+1)**

Optical Flow: State of the Art

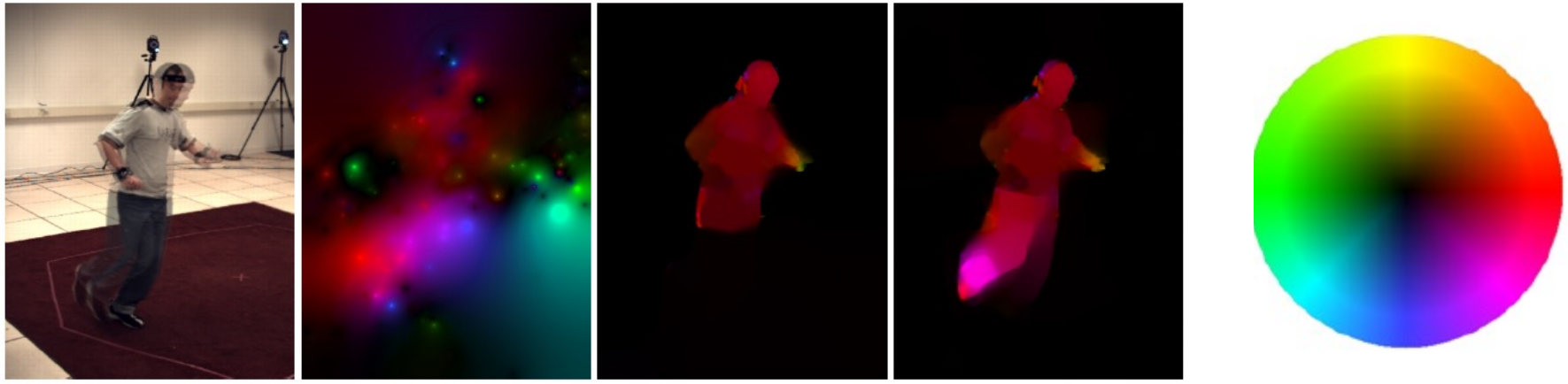
Start with something similar to Lucas-Kanade

+ gradient constancy

+ energy minimization with smoothing term

+ region matching

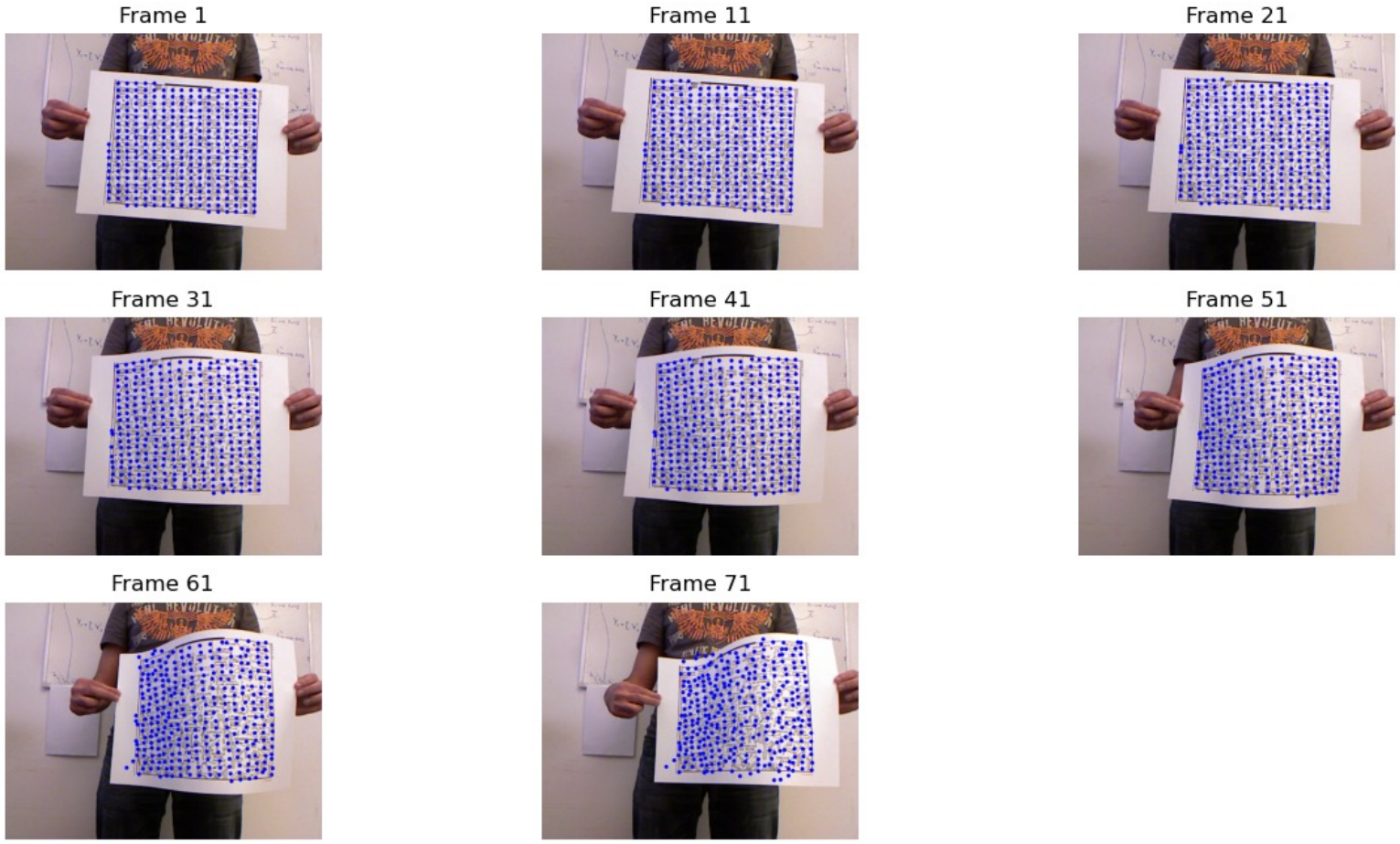
+ keypoint matching (long-range)



Region-based +Pixel-based +Keypoint-based

Large displacement optical flow, Brox et al., CVPR 2009

Optical Flow: State of the Art



PIPS: ECCV-2022

Feature Tracking: Shi-Tomasi Feature Tracker

Find good features using eigenvalues of second-moment matrix (e.g., Harris detector or threshold on the smallest eigenvalue)

- Key idea: "good" features to track are the ones whose motion can be estimated reliably

Track from frame to frame with Lucas-Kanade

- This amounts to assuming a translation model for frame-to-frame feature movement

Check consistency of tracks by *affine* registration to the first observed instance of the feature

- Affine model is more accurate for larger displacements
- Comparing to the first frame helps to minimize drift

J. Shi and C. Tomasi. Good Features to Track. CVPR 1994.

Feature Tracking: Shi-Tomasi Feature Tracker

Figure 1: Three frame details from Woody Allen's *Manhattan*. The details are from the 1st, 11th, and 21st frames of a subsequence from the movie.

Figure 2: The traffic sign windows from frames 1,6,11,16,21 as tracked (top), and warped by the computed deformation matrices (bottom).

J. Shi and C. Tomasi. Good Features to Track. CVPR 1994.

Feature Tracking: Shi-Tomasi Feature Tracker

1. Find a good point to track (harris corner)
2. Use intensity second moment matrix and difference across frames to find displacement
3. Iterate and use coarse-to-fine search to deal with larger movements
4. When creating long tracks, check appearance of registered patch against appearance of initial patch to find points that have drifted

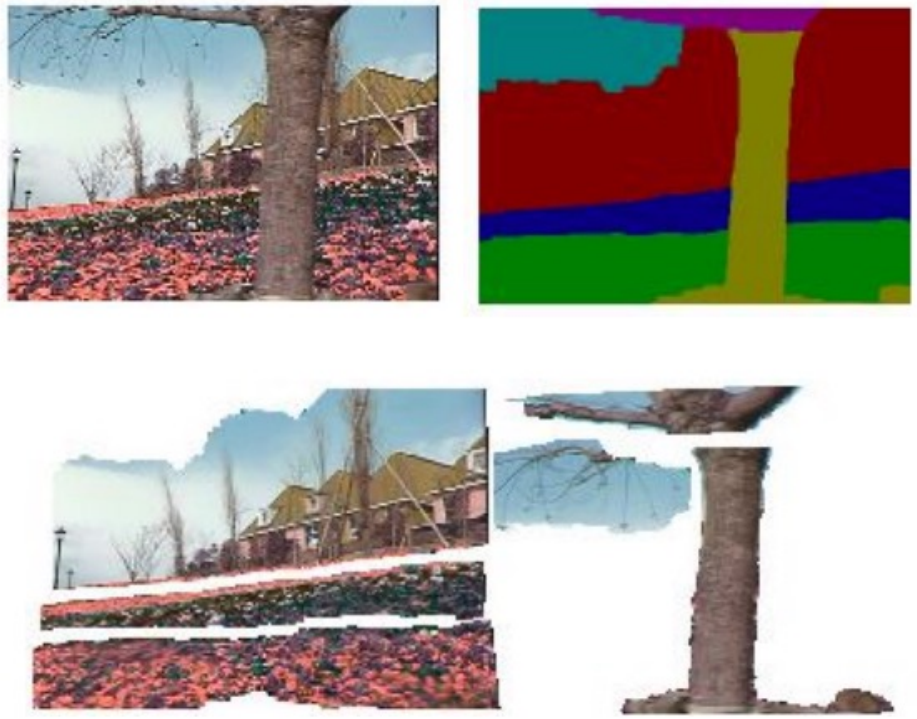
Implementation issues:

- Window size
  - Small window more sensitive to noise and may miss larger motions (without pyramid)
  - Large window more likely to cross an occlusion boundary (and it's slower)
  - 15x15 to 31x31 seems typical
- Weighting the window
  - Common to apply weights so that center matters more (e.g., with Gaussian)

J. Shi and C. Tomasi. Good Features to Track. CVPR 1994.

Motion Segmentation

- Create layers (with coherent affine motion) and track them



J. Wang and E. Adelson. Layered Representation for Motion Analysis. *CVPR 1993*.

Affine Motion

$$u(x,y) = a_1 + a_2 x + a_3 y$$
$$v(x,y) = a_4 + a_5 x + a_6 y$$

- Substituting into the brightness constancy equation:

$$I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \approx 0$$

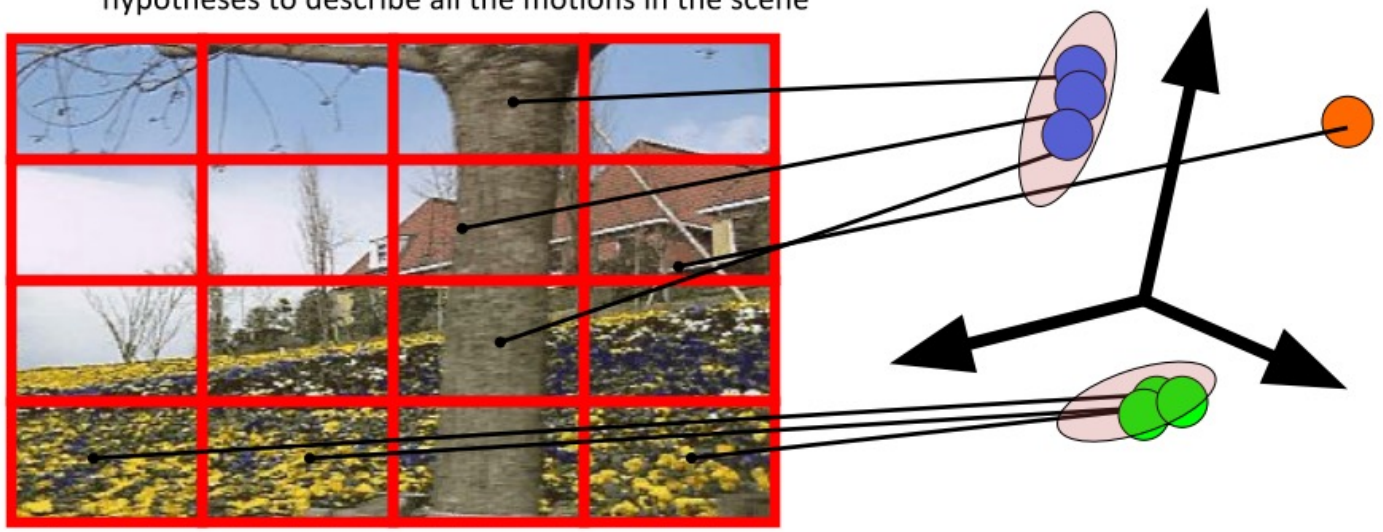
- Each pixel provides 1 linear constraint in 6 unknowns
- Least squares minimization:

$$Err(\vec{a}) = \sum \left[I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \right]^2$$

Source: Silvio Savarese

How to estimate layers?

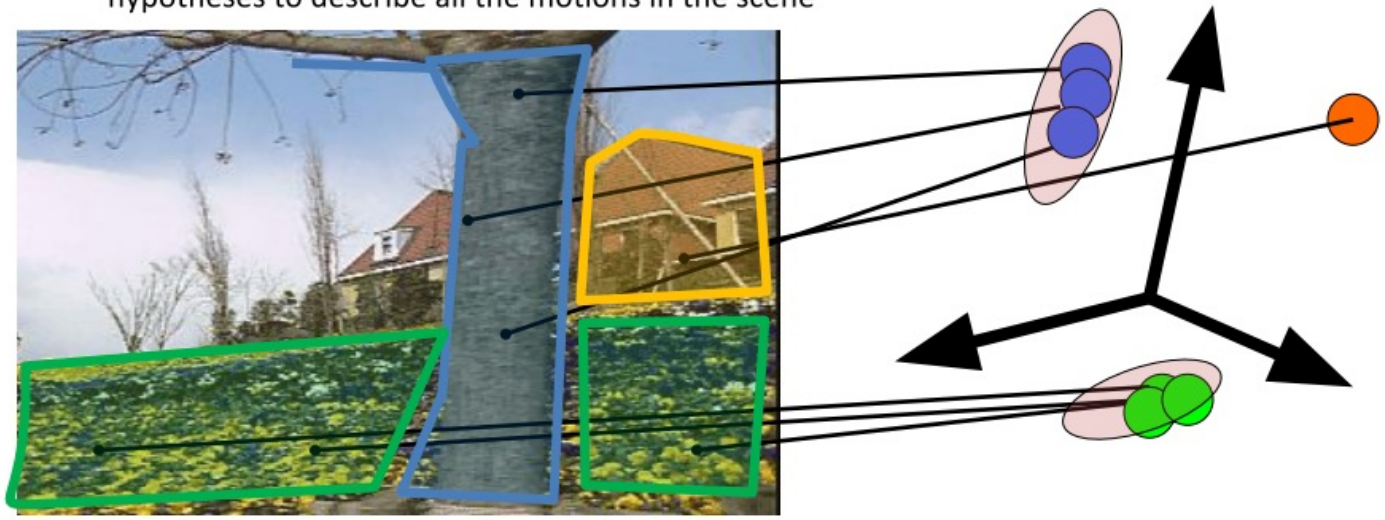
1. Obtain a set of initial affine motion hypotheses
   - Divide the image into blocks and estimate affine motion parameters in each block by least squares
   - Eliminate hypotheses with high residual error
2. Map into motion parameter space
3. Perform k-means clustering on affine motion parameters
   - Merge clusters that are close and retain the largest clusters to obtain a smaller set of hypotheses to describe all the motions in the scene



Source: Silvio Savarese

How to estimate layers?

1. Obtain a set of initial affine motion hypotheses
   - Divide the image into blocks and estimate affine motion parameters in each block by least squares
   - Eliminate hypotheses with high residual error
2. Map into motion parameter space
3. Perform k-means clustering on affine motion parameters
   - Merge clusters that are close and retain the largest clusters to obtain a smaller set of hypotheses to describe all the motions in the scene



Source: Silvio Savarese

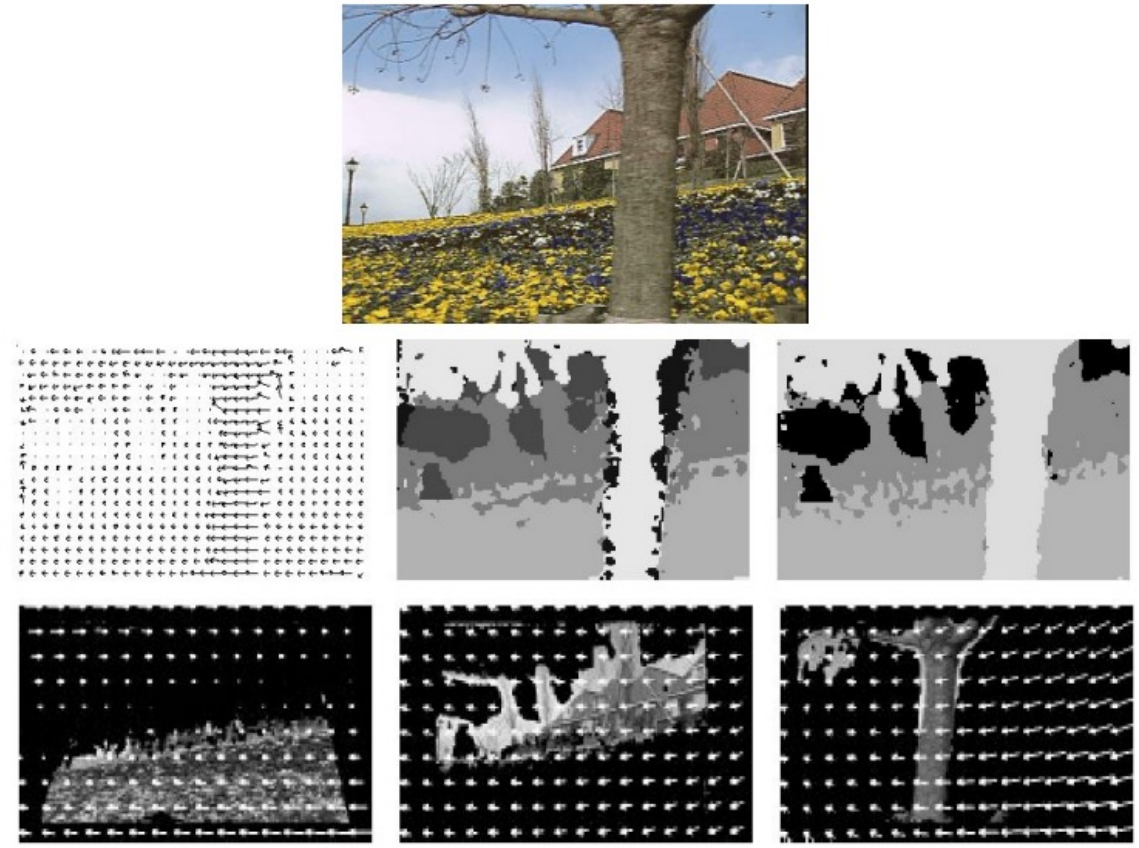
How to estimate layers?

1. Obtain a set of initial affine motion hypotheses
   - Divide the image into blocks and estimate affine motion parameters in each block by least squares
   - Eliminate hypotheses with high residual error
   - Map into motion parameter space
   - Perform k-means clustering on affine motion parameters
   - Merge clusters that are close and retain the largest clusters to obtain a smaller set of hypotheses to describe all the motions in the scene

2. Iterate until convergence:
   - Assign each pixel to best hypothesis
   - Pixels with high residual error remain unassigned
   - Perform region filtering to enforce spatial constraints
   - Re-estimate affine motions in each region

Source: Silvio Savarese

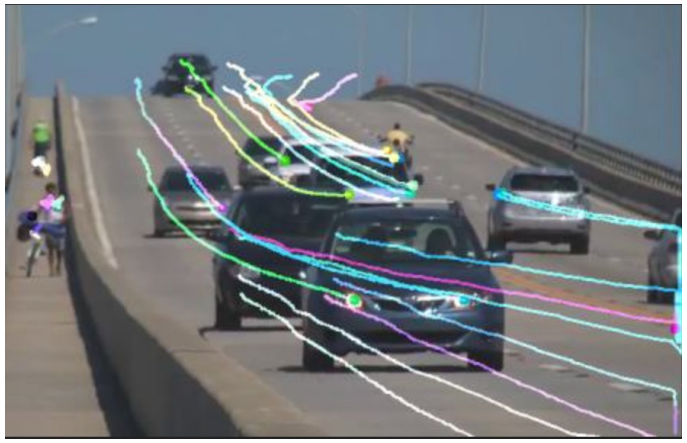
How to estimate layers?



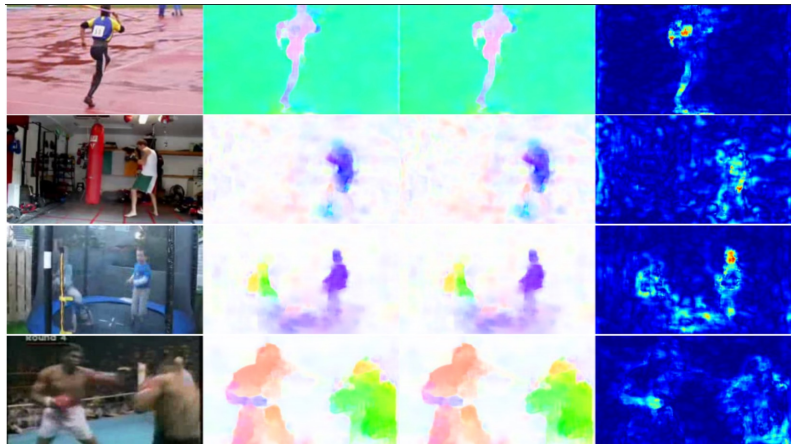
J. Wang and E. Adelson. Layered Representation for Motion Analysis. *CVPR 1993.*

Source: Silvio Savarese

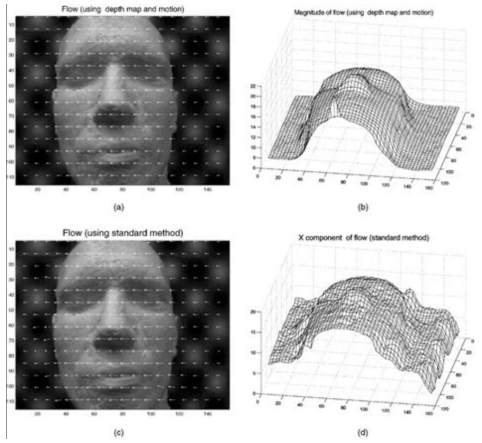
Applications-Optical Flow



Trajectory estimation



Motion estimation



3D Reconstruction

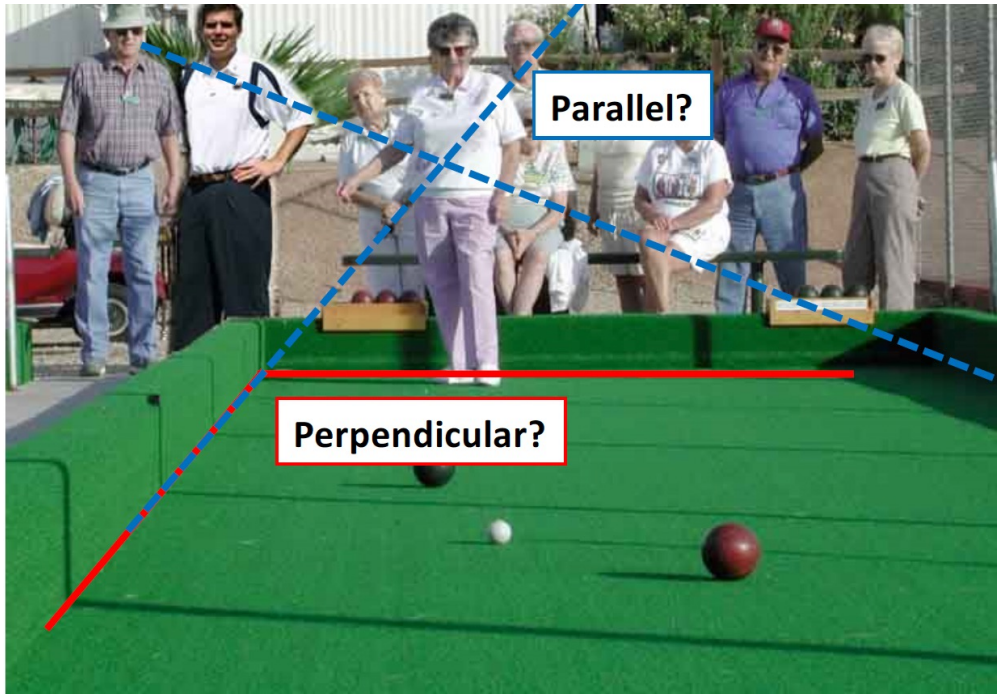
D. Building Panaromas

Instructor: Shaifali Parashar

Many Slides from Fei-Fei Li's lectures at Standford

Images formation

Lengths, angles are lost
Parallelism is lost
Collinearity is preserved

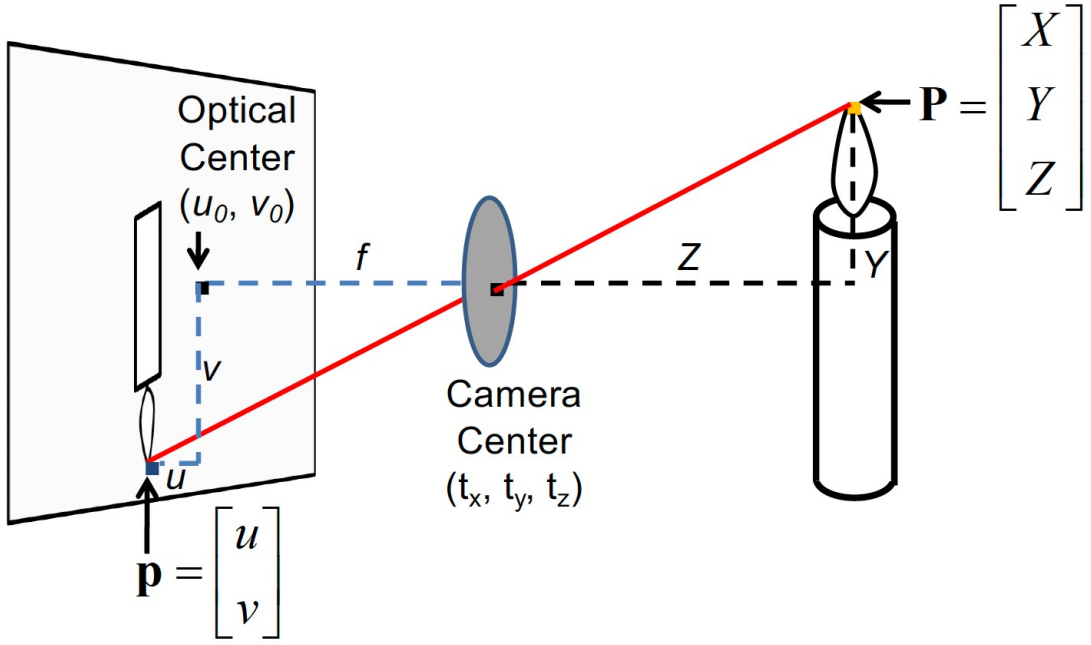


Images formation-Pin hole camera

$$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -f\dfrac{X}{Z} \\ -f\dfrac{Y}{Z} \end{bmatrix}$$

In fact, we invert and shift the image.

p ----> $[u_0, v_0]$ - **p**

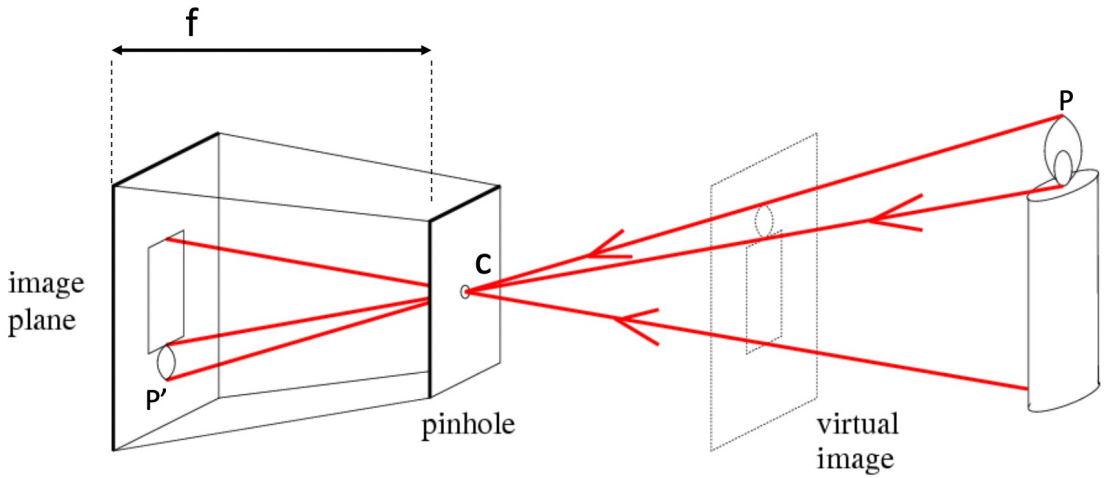


Homogeneous coordinates

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous image coordinates

homogeneous scene coordinates



Converting from homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w) \qquad \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$$

Also invariant to scale, multiplication with a scalar results movement along a ray

Another problem solved by homogeneous coordinates

Intersection of parallel lines

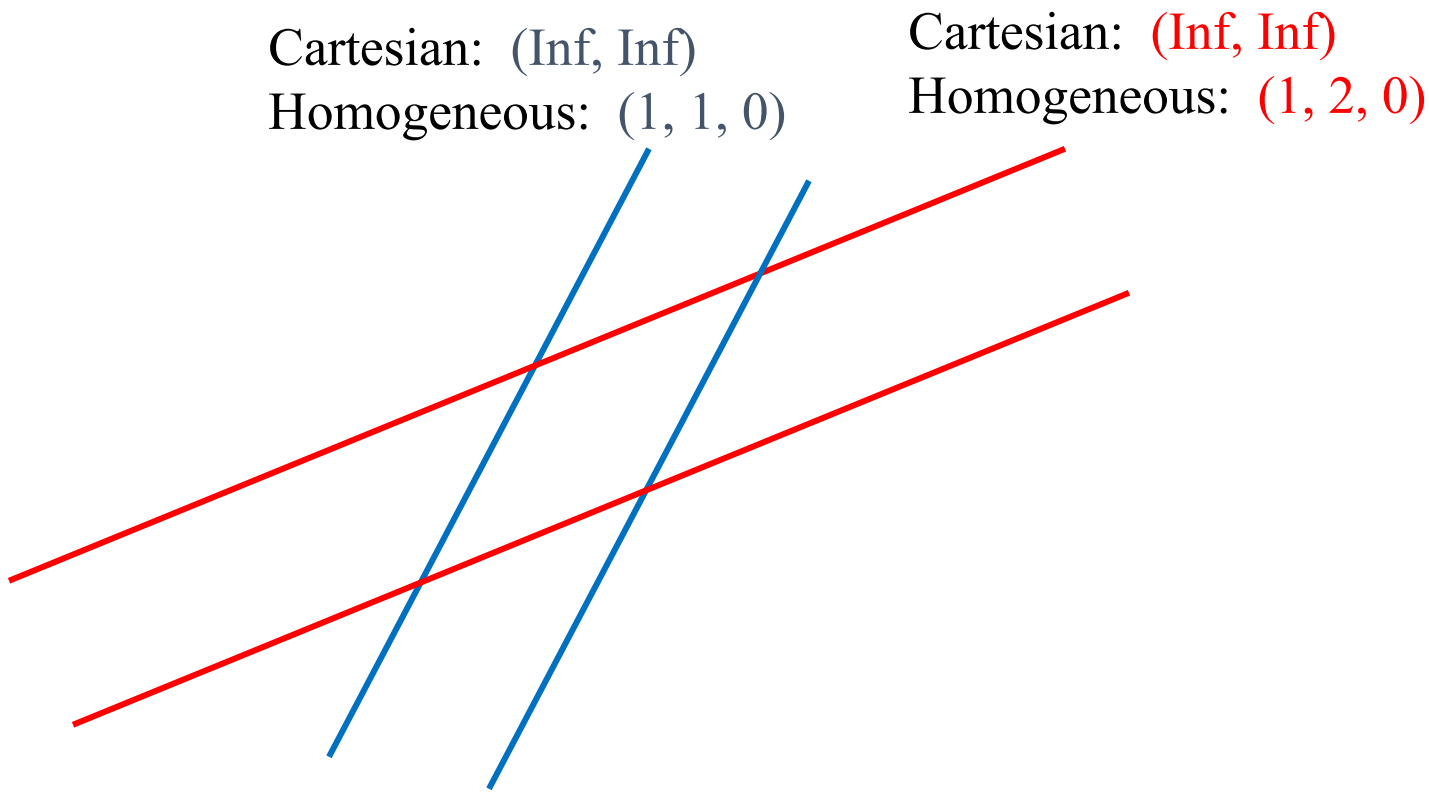
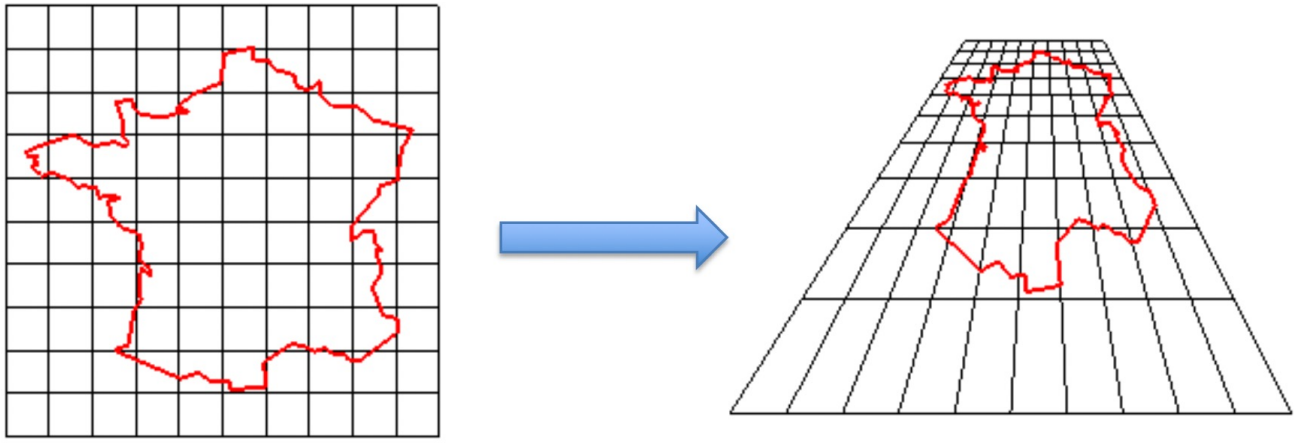


Image Transformations (Homography)



$$c\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Image transformations → Projective transformation → Homography

Homography Estimation

- If point correspondences (x,y)<-->(x',y') are known

We can find a 3X3 transformation, **A** such that
x' = **Ax**

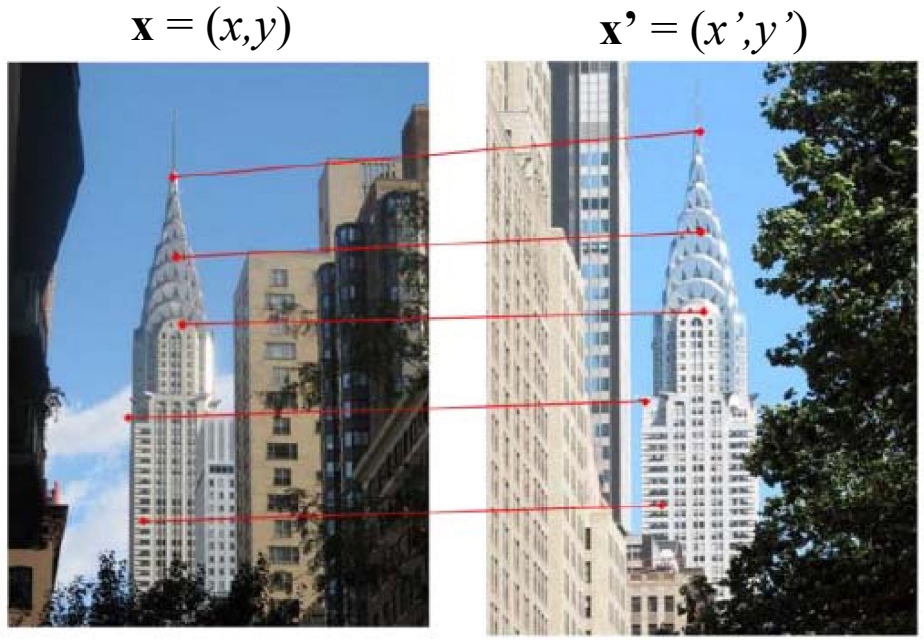
A = $[a_1, a_2, a_3; a_4, a_5, a_6; a_7, a_8, a_9]$

Remember from first lecture on transformations, a 2D projective transformation (most generic transformation) has 8 dof

Assume $a_9 = 1$, therefore

$$x' = \frac{a_1 x + a_2 y + a_3}{a_7 x + a_8 y + 1}$$

$$y' = \frac{a_4 x + a_5 y + a_6}{a_7 x + a_8 y + 1}$$



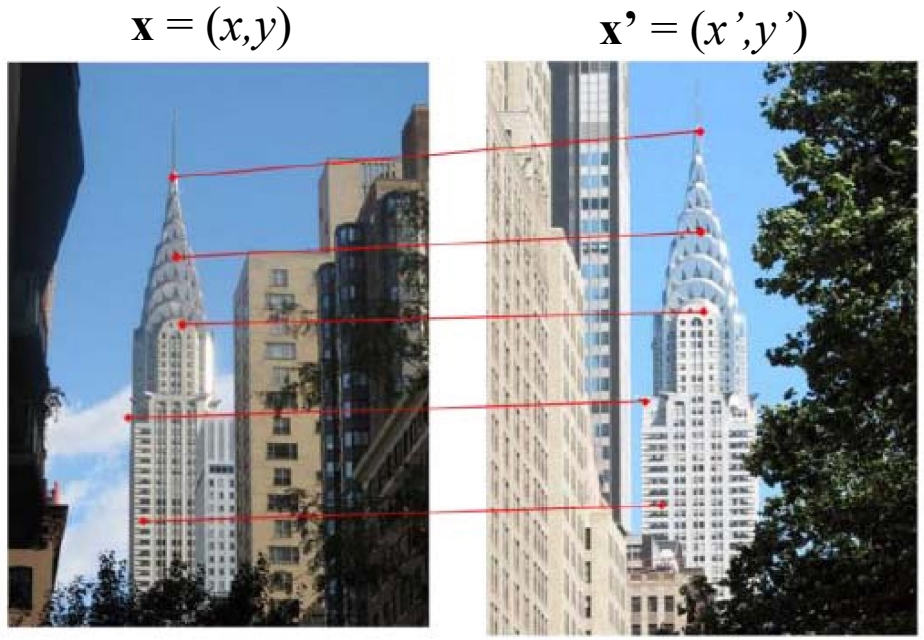
Homography Estimation

Assume $a_9 = 1$, therefore

$$x' = \frac{a_1 x + a_2 y + a_3}{a_7 x + a_8 y + 1}$$

$$y' = \frac{a_4 x + a_5 y + a_6}{a_7 x + a_8 y + 1}$$

$$a_7 x' x + a_8 x' y + x' = a_1 x + a_2 y + a_3$$
$$a_7 y' x + a_8 y' y + y' = a_7 x + a_8 y + a_6$$
$$x' = a_1 x + a_2 y + a_3 - a_7 x' x - a_8 x' y$$
$$y' = a_7 x + a_8 y + a_6 - a_7 y' x - a_8 y' y$$



Segregate the equations into matrices/vectors of knowns and unknowns

Homography Estimation

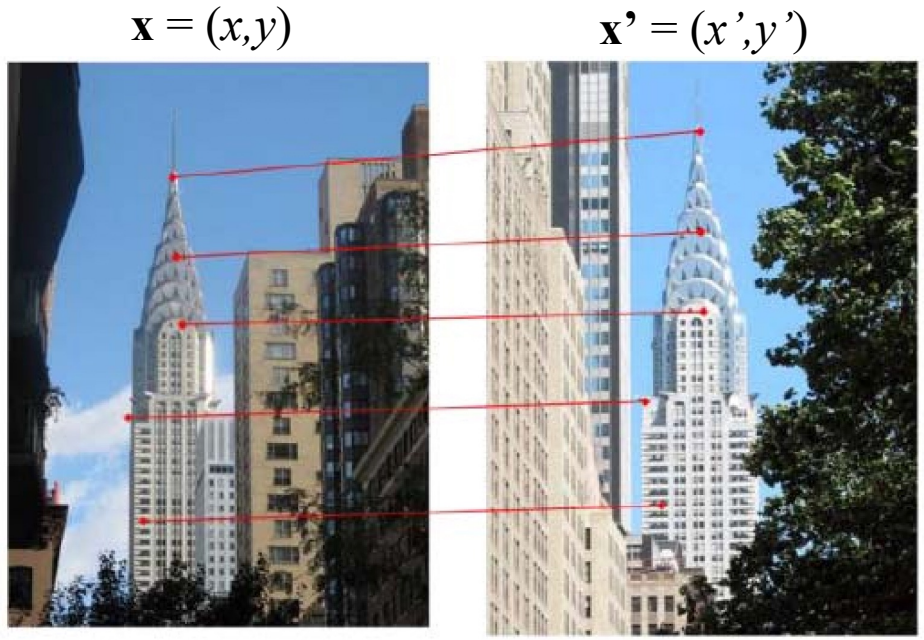
$$a_7 x'x + a_8 x'y + x' = a_1 x + a_2 y + a_3$$
$$a_7 y'x + a_8 y'y + y' = a_7 x + a_8 y + a_6$$
$$x' = a_1 x + a_2 y + a_3 - a_7 x'x - a_8 x'y$$
$$y' = a_7 x + a_8 y + a_6 - a_7 y'x - a_8 y'y$$

Two rows for each point *i*

$$\begin{bmatrix} \vdots & & & & & & & \\ x_i & y_i & 1 & 0 & 0 & 0 & -x_i x_i' & -y_i x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_i y_i' & -y_i y_i' \\ \vdots & & & & & & & \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{bmatrix} = \begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}$$

$$\mathbf{T}_{8X8}\mathbf{a}_{8X1} = \mathbf{B}_{8X1}$$

Linear Least squares: $a = (\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{B}$



Homography using Direct Linear Transformation (DLT)

Don't assume $a_9 = 1$, add it to the system

In order to obtain a non-trivial solution,

we obtain **a :**

min || **T'a** || such that ||**a**|| =1

Two rows for each point *i*

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i x_i' & -y_i x_i' \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_i y_i' & -y_i y_i' \\ & & & & \vdots & & & \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{bmatrix} = \begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}$$

a_9

$\mathbf{T'}_{8X9}\mathbf{a}_{9X1} = \mathbf{0}_{9X1}$

$\mathbf{T'} = [\mathbf{T},\mathbf{-B}]$

Homography using Direct Linear Transformation (DLT)

Don't assume $a_9 = 1$, add it to the system

In order to obtain a non-trivial solution,

we obtain **a :**

min || **T'a** || such that ||**a**|| =1

Solution: SVD of **T'**. The eigenvector corresponding to smallest eigenvalue of $\mathbf{T'}^{T}\mathbf{T'}$ is the solution.

In order to get a stable solution, Normalise the data before DLT.

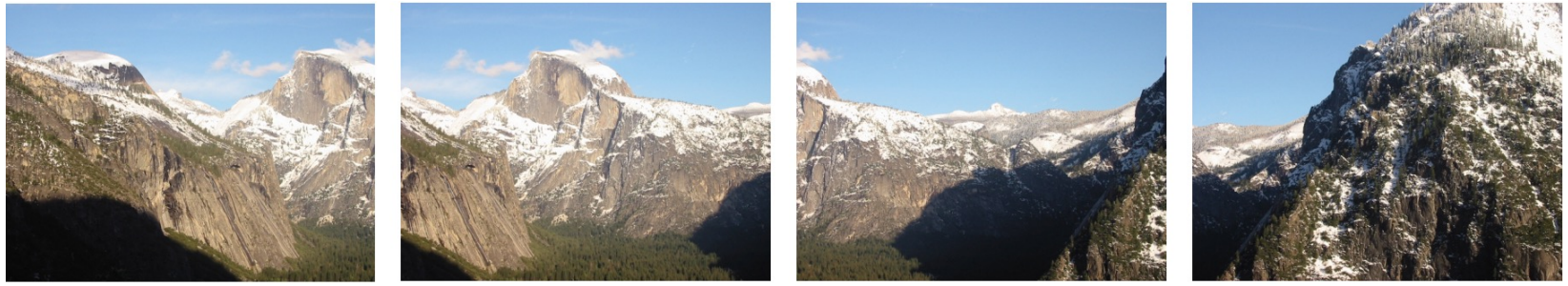
a) Translate for zero mean

b) b) Scale so that average distance to origin is ~sqrt(2)

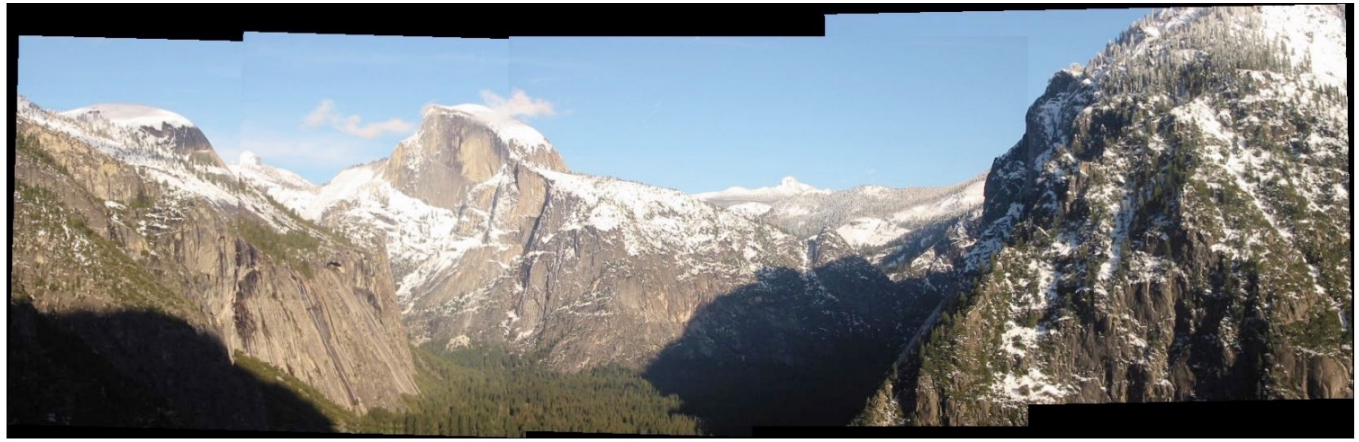
https://www.cs.cmu.edu/~16385/s17/Slides/10.2_2D_Alignment__DLT.pdf

Panoramic Image Stitching

Given a set of images



Obtain:



Problem with Linear Least Squares (LLS) fitting

1. Noisy data and outliers

Solution:

a) Voting. Check all possible combinations. Cycle through features, cast votes for model parameters. Bad idea due to high computation cost.

b) RANdom SAmple Consensus (RANSAC)

RANSAC

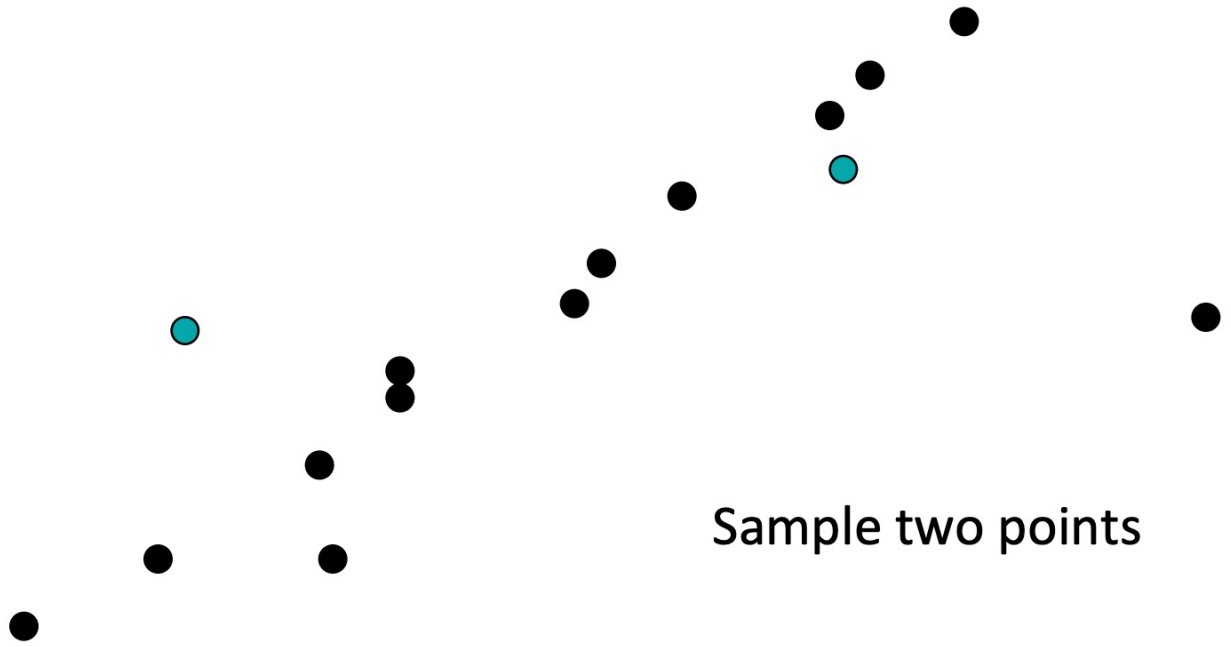
Look for ‘only’ inliers.

1. Randomly select a seed group to estimate transformation
2. Compute transformation
3. Compute inliers
4. If inliers are large, recompute transformation

Keep transformation with most inliers

RANSAC: line fitting example

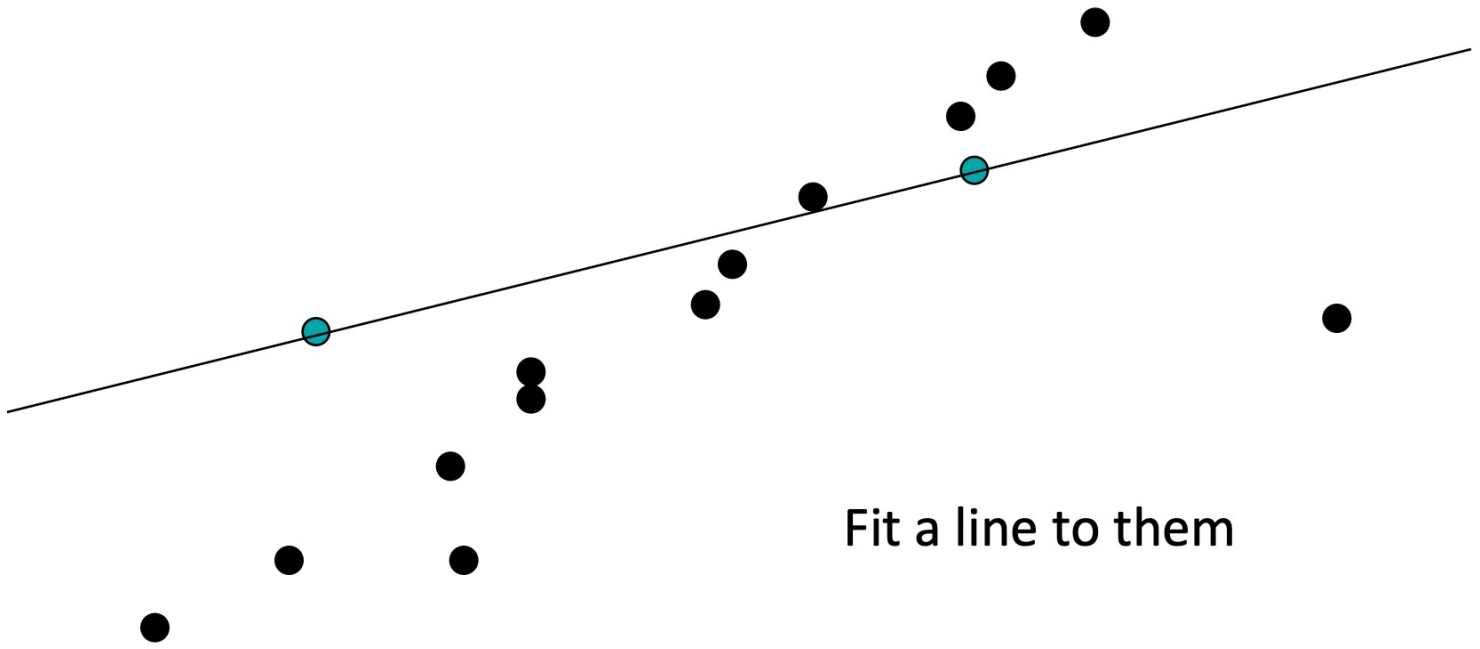
Estimate the best line



Slide credit: Jinxiang Chai

RANSAC: line fitting example

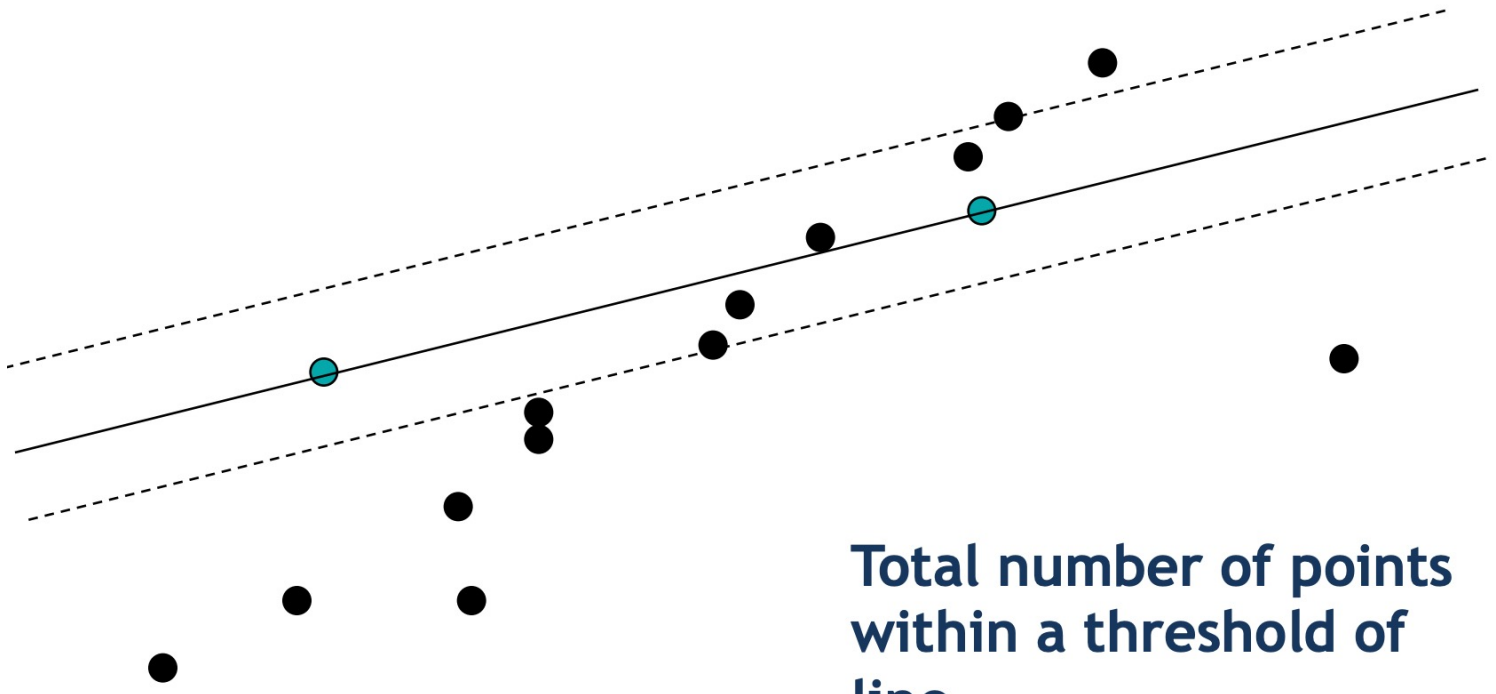
Estimate the best line



Slide credit: Jinxiang Chai

RANSAC: line fitting example

Estimate the best line

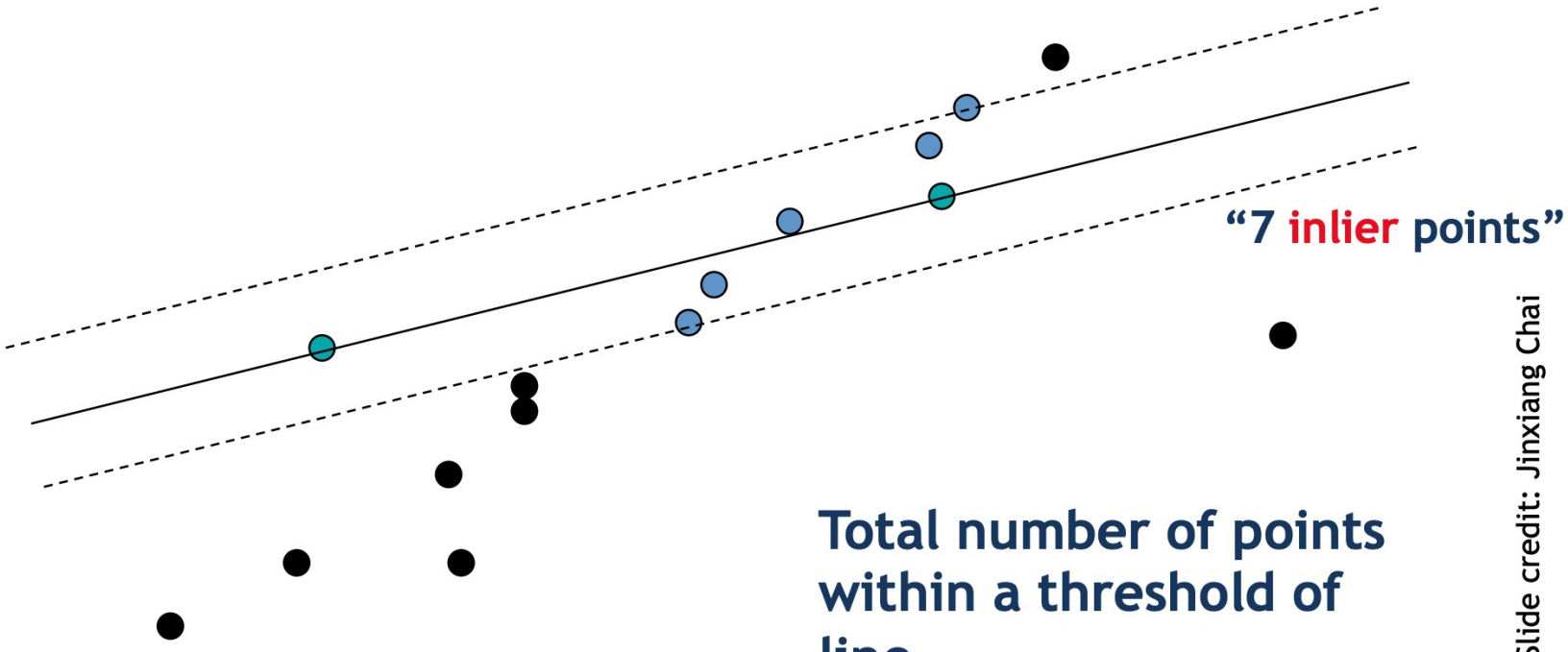


Total number of points within a threshold of line.

Slide credit: Jinxiang Chai

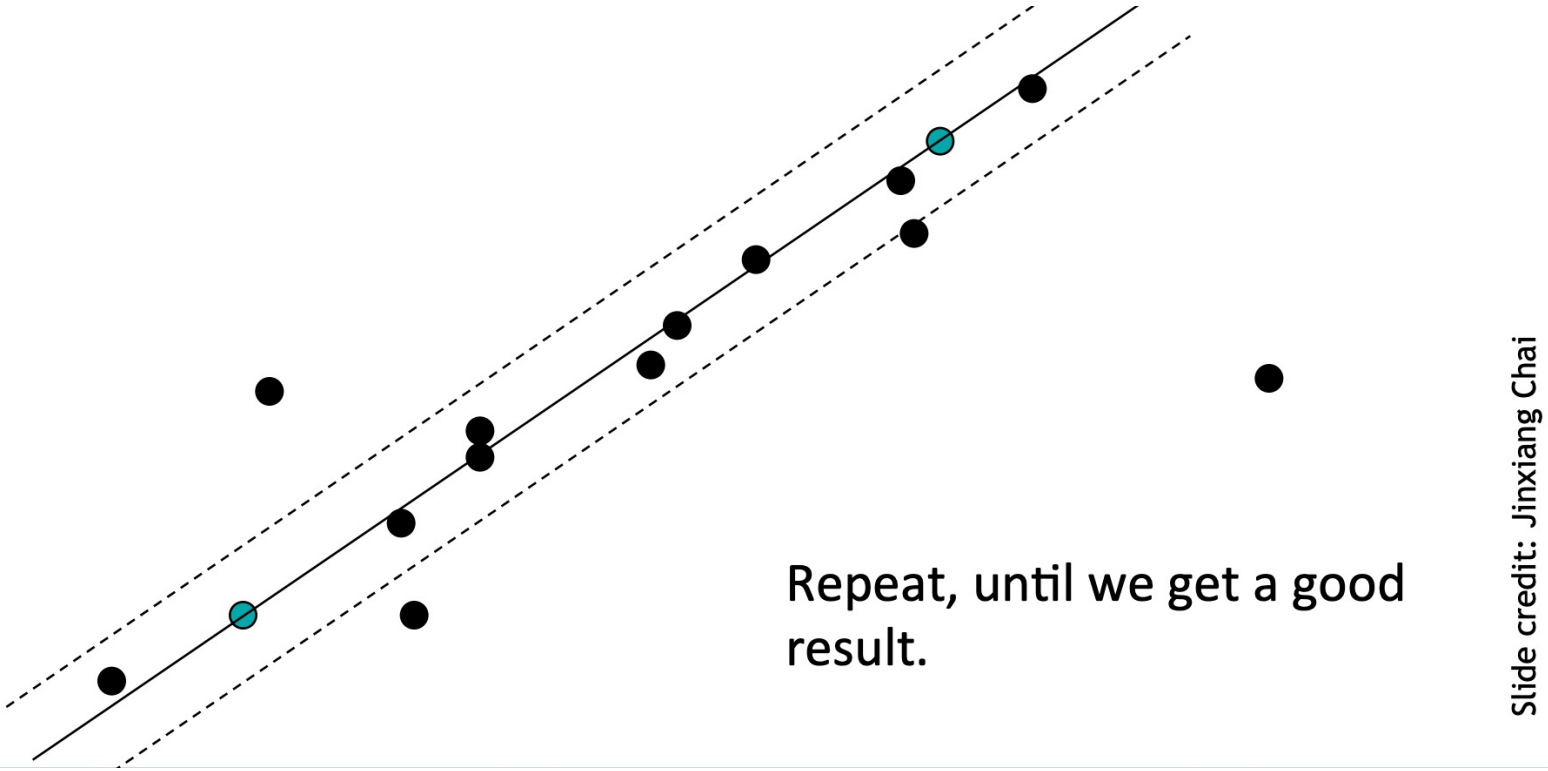
RANSAC: line fitting example

Estimate the best line



RANSAC: line fitting example

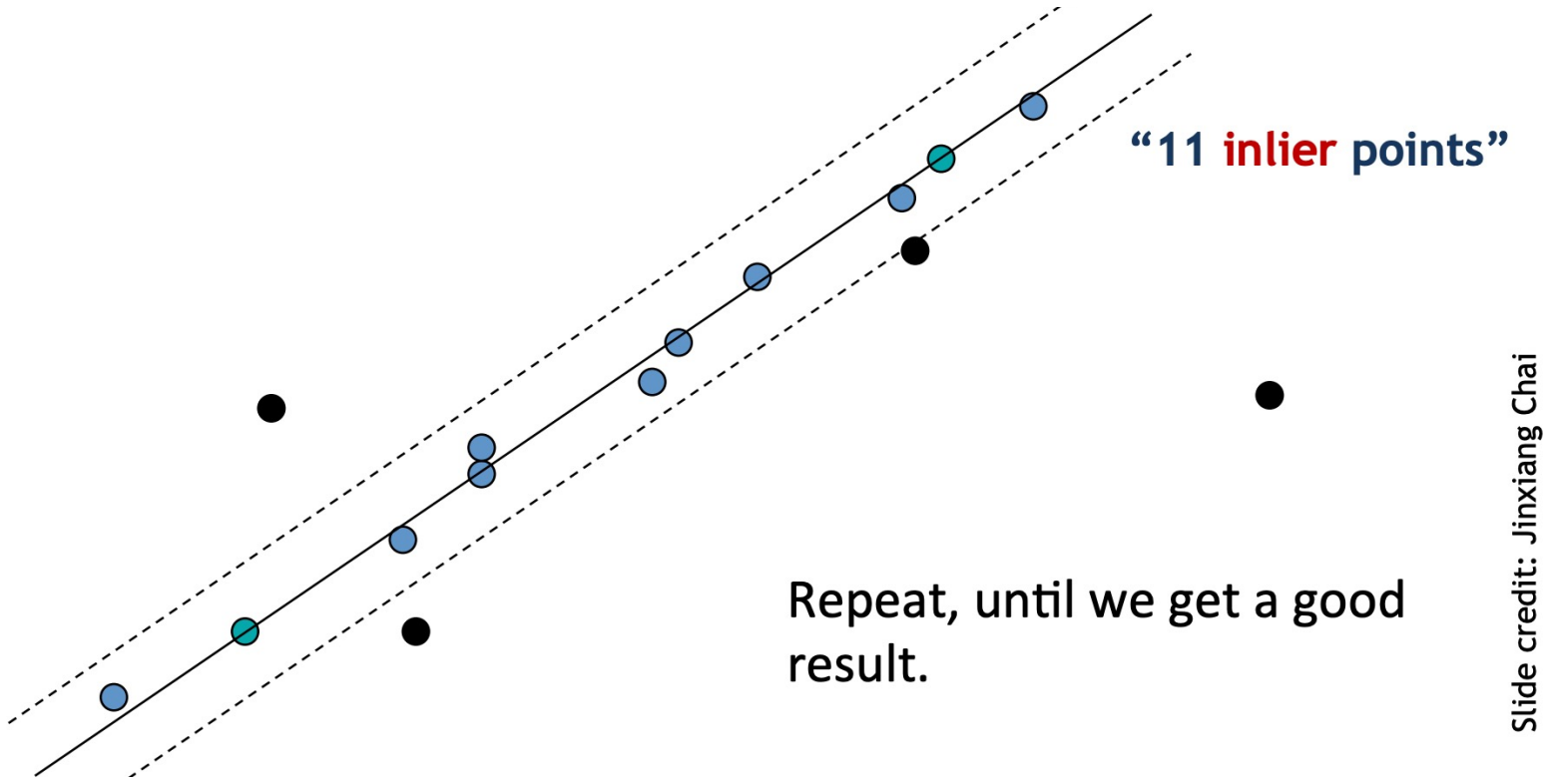
Estimate the best line



Slide credit: Jinxiang Chai

RANSAC: line fitting example

Estimate the best line



Slide credit: Jinxiang Chai

RANSAC: line-fitting example

After RANSAC, use all inliers to improve transformation. This may change the transformation, so perform another reclassification of inliers-outliers.

RANSAC is fast but it can accommodate relatively small number of outliers.

Voting is efficient but computationally expensive.

RANSAC: line-fitting example

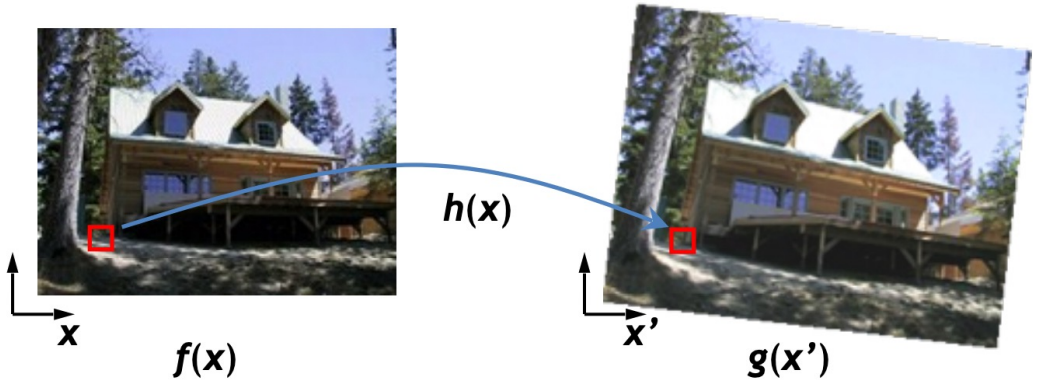
After RANSAC, use all inliers to improve transformation. This may change the transformation, so perform another reclassification of inliers-outliers.

RANSAC is fast but it can accommodate relatively small number of outliers.

Voting is efficient but computationally expensive.

Image Warping: Forward

Given a transformation x'=h(x) and source image f(x), how do we compute a transformed image g(x')=f(h(x))?



- What if pixel lands "between" two pixels?
- Answer: add "contribution" to several pixels, normalize later (*splatting*)

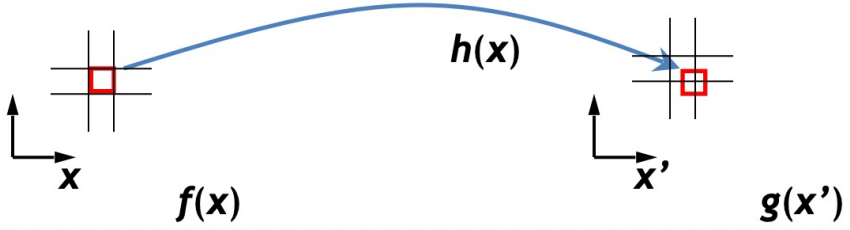
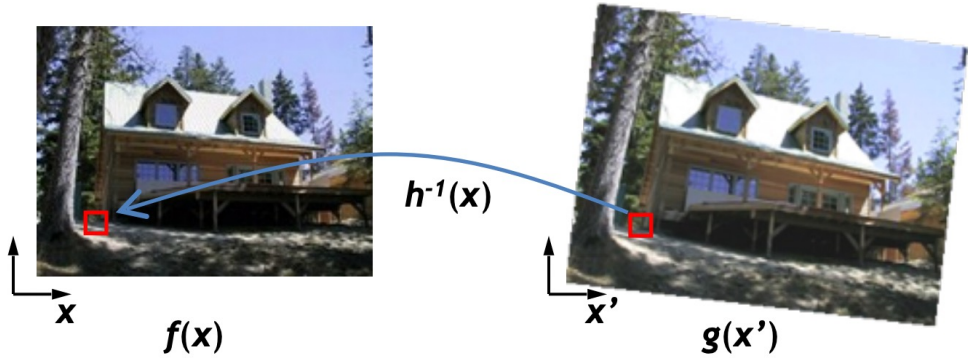
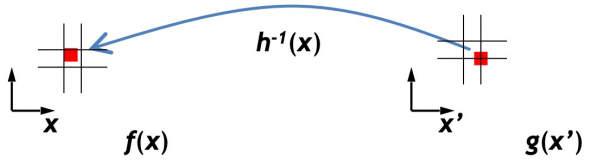


Image Warping: Backward

Get each pixel g(x') from its corresponding location x' = h(x) in f(x) ?

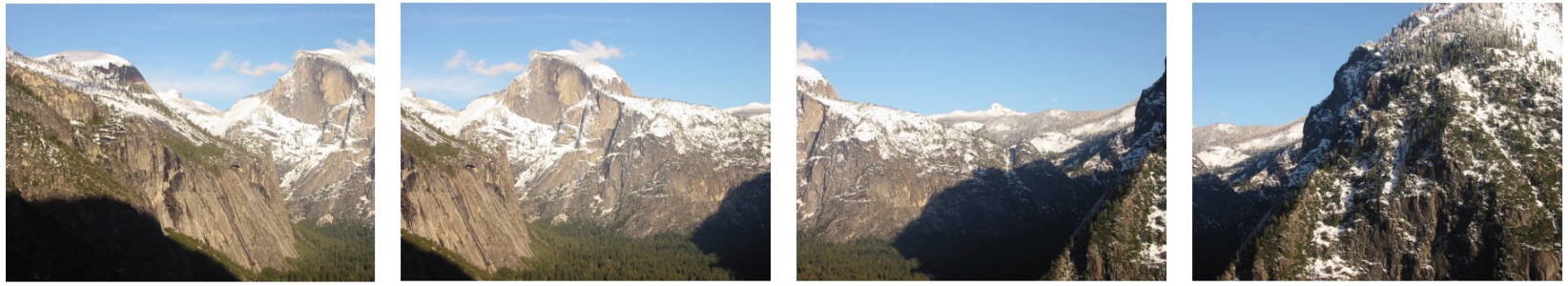


- What if pixel comes from "between" two pixels?
- Answer: *resample* color value from *interpolated* source image



Panoramic Image Stitching

Given a set of images



Obtain: