RRLAB
THE ROBOTICS RESEARCH LAB

Autonomous Mobile Robots (AMR)
5. Feature Extraction and Object Recognition

Prof. Karsten Berns
Robotics Research Lab
Department of Computer Science
University of Kaiserslautern, Germany

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Features

- Objects show features that can
  - be algorithmically extracted:
  - Edges, Corners, Texture
  - Shapes, Texture
- Special interest:
  - Stable Features
  - Invariant Features
  - Salient Features

Typical object features

Features that can be detected by robots' sensor systems

- geometric (size, shape) – laser scanners, cameras
- visual (color, texture, image features) – cameras
- physical (weight, temperature, motion) - IR-cameras, special-purpose sensors
- acoustic (noise, acoustic pattern) - microphones, sonar
- chemical (emission) - smell sensors

Visual Features

- Cameras are a commonly used sensor system in robotics
  - fast, cheap, rich of information
- Object properties that can be determined from images
  - shape (contour detection)
  - edges (edge filtering)
  - material (texture extraction, color thresholding)
  - motion (optical flow, temporal images)
  - distance (stereo-vision)

Visual Feature Hierarchy

Robot's Perception

- Object Classification
- Object Detection
- Object Tracking

High level Descriptor

- Key points (SIFT, SURF, BRISK)
- Texture (HOG, LBP)
- Distance image (grid, voxel)

Filtering and Preprocessing

- Corner
- Edge
- Geometric Features

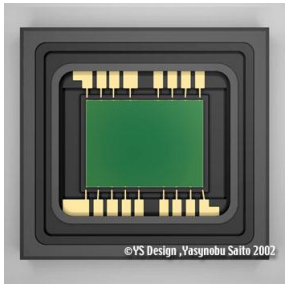
Raw Data from Sensors

- Color Image
- Gray Image
- Point Clouds (Stereo, Laser)

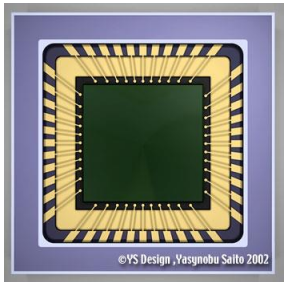
Digital camera

- Two common types
  - Charge Coupled Device (CCD)
  - CMOS

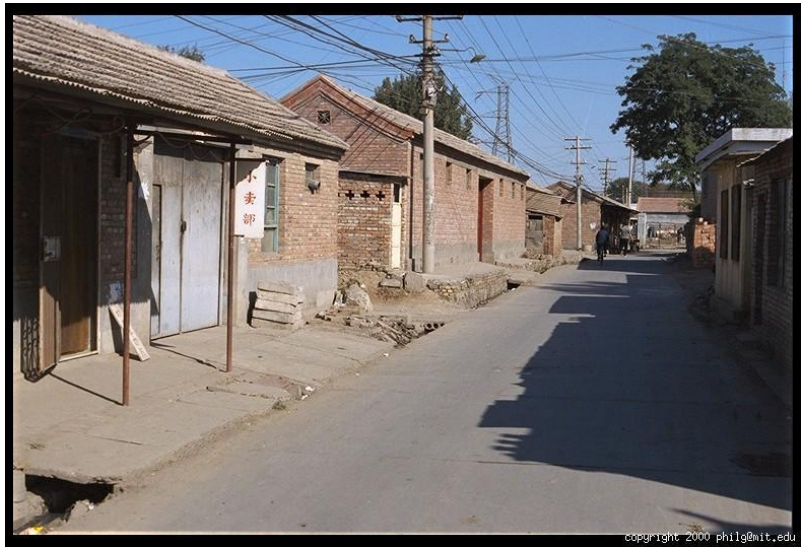
- CCD (charge coupled device)
  - Higher dynamic range
  - High uniformity
  - Lower noise



- CMOS (complementary metal Oxide semiconductor)
  - Lower voltage
  - Higher speed
  - Lower system complexity



Color Image



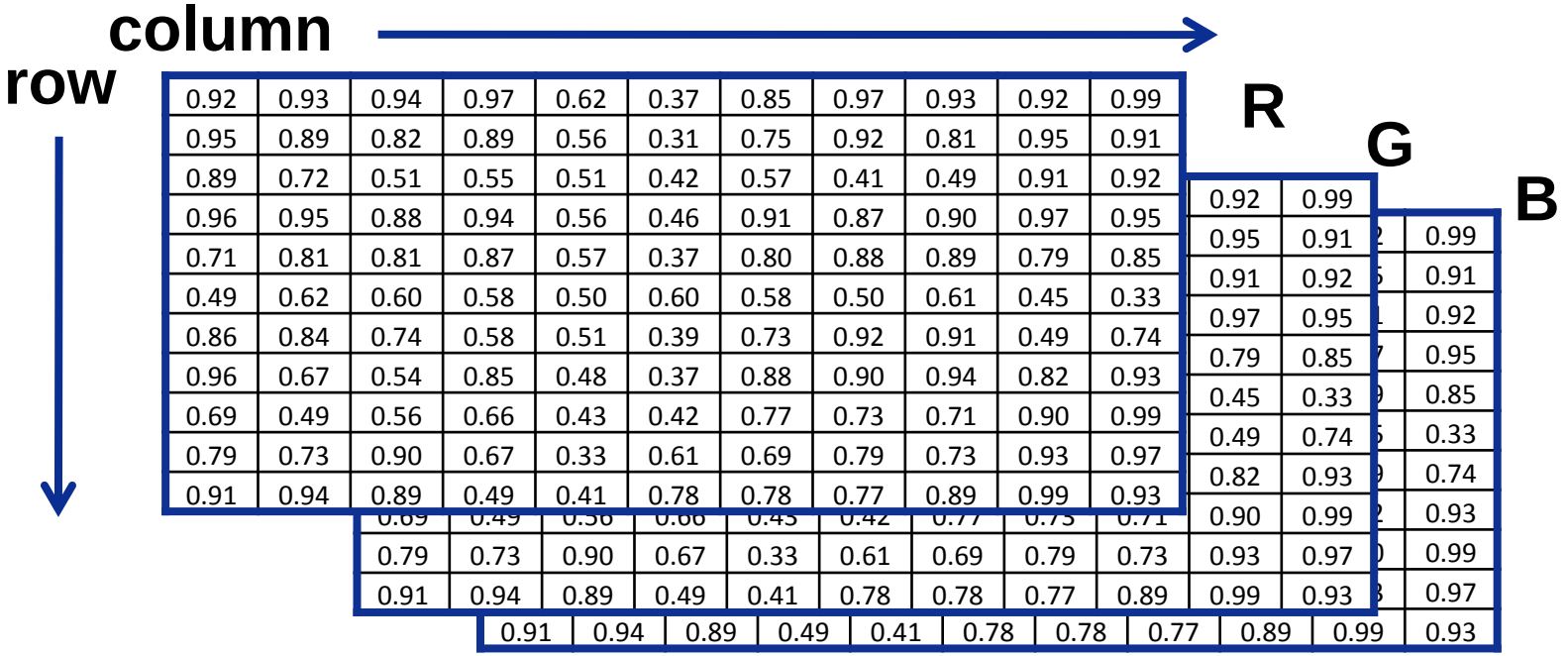
R

G

B

Images in Matlab

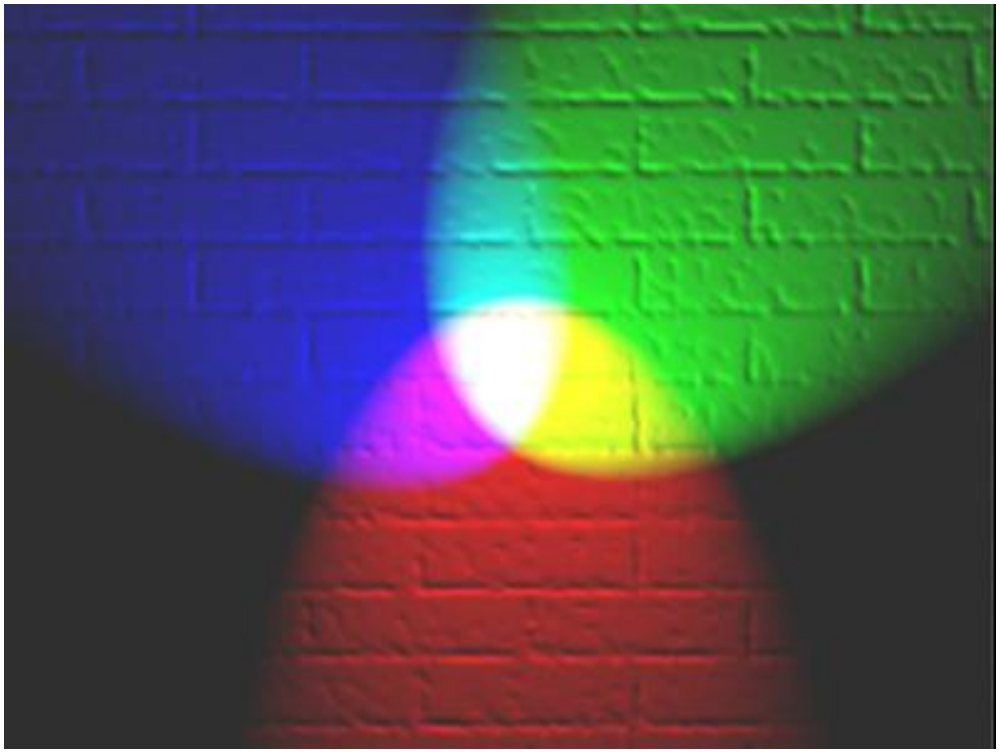
- Images represented as a matrix
- Suppose we have a NxM RGB image called "im"
  - im(1,1,1) = top-left pixel value in R-channel
  - im(y, x, b) = y pixels down, x pixels to right in the bth channel
  - im(N, M, 3) = bottom-right pixel in B-channel
- imread(filename) returns a uint8 image (values 0 to 255)
  - Convert to double format (values 0 to 1) with im2double



0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

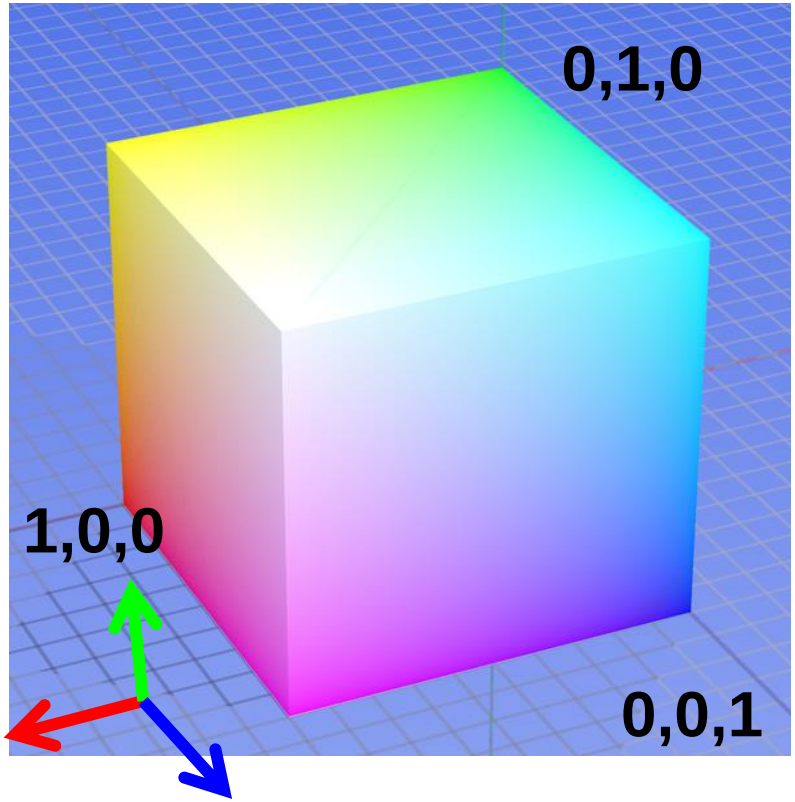
Color Spaces

- How can we represent color?



Color Spaces - RGB

Default color space



R
(G=0,B=0)

G
(R=0,B=0)

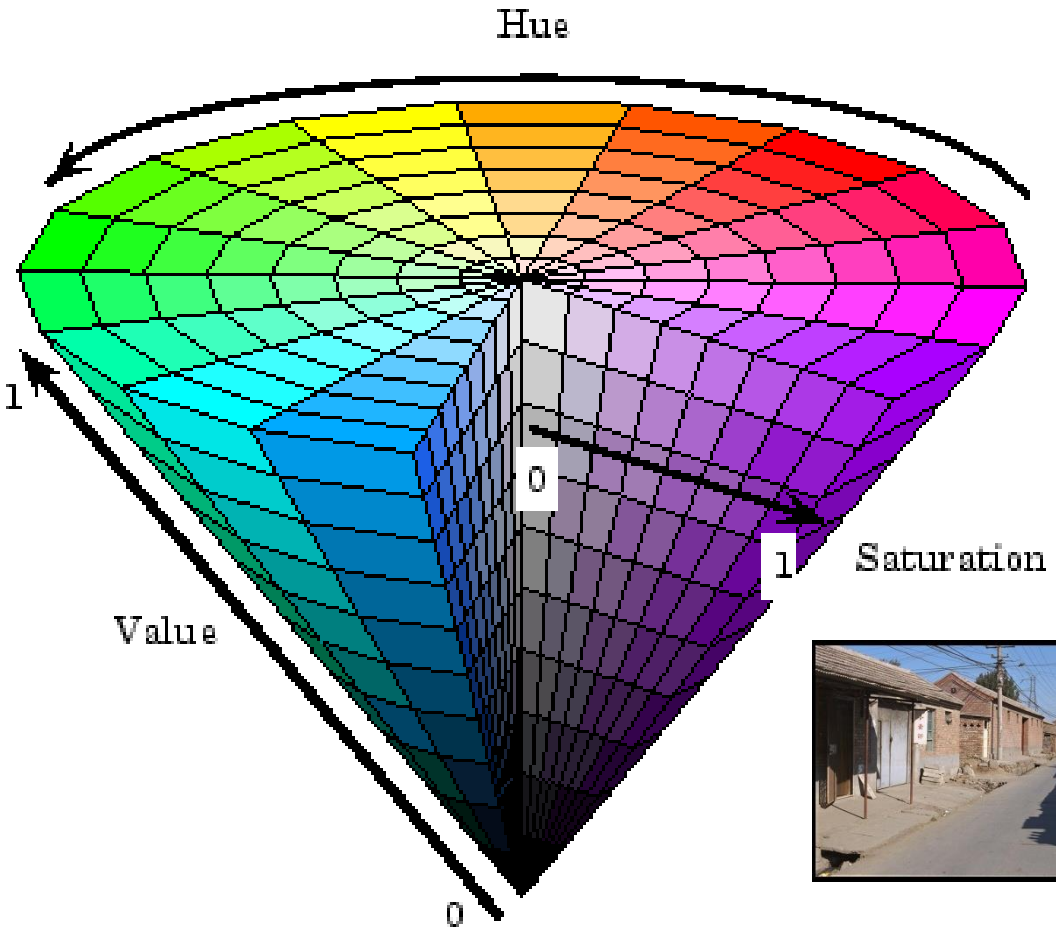
B
(R=0,G=0)

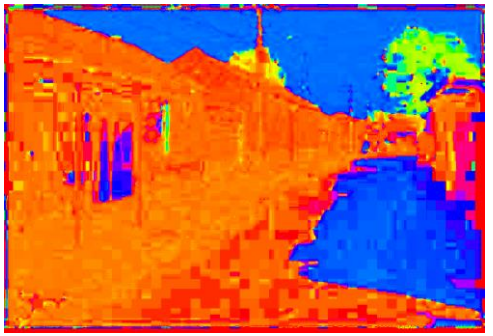
Some drawbacks

- Strongly correlated channels
- Non-perceptual

Color Spaces - HSV

Intuitive color space





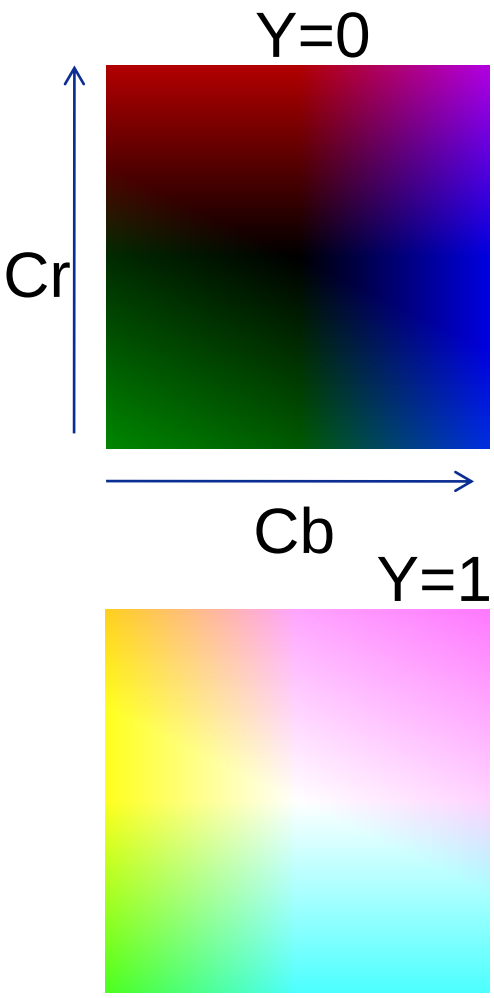
H
(S=1,V=1)

S
(H=0,V=1)

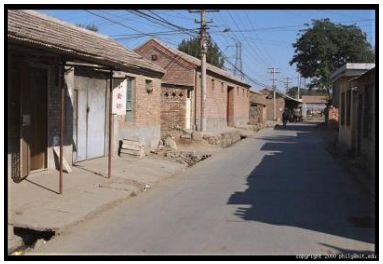
V
(H=0,S=0)

Color spaces: YCbCr

Fast to compute, good for compression, used by TV



Cr: green, red
Cb: green, blue
Y: brightness



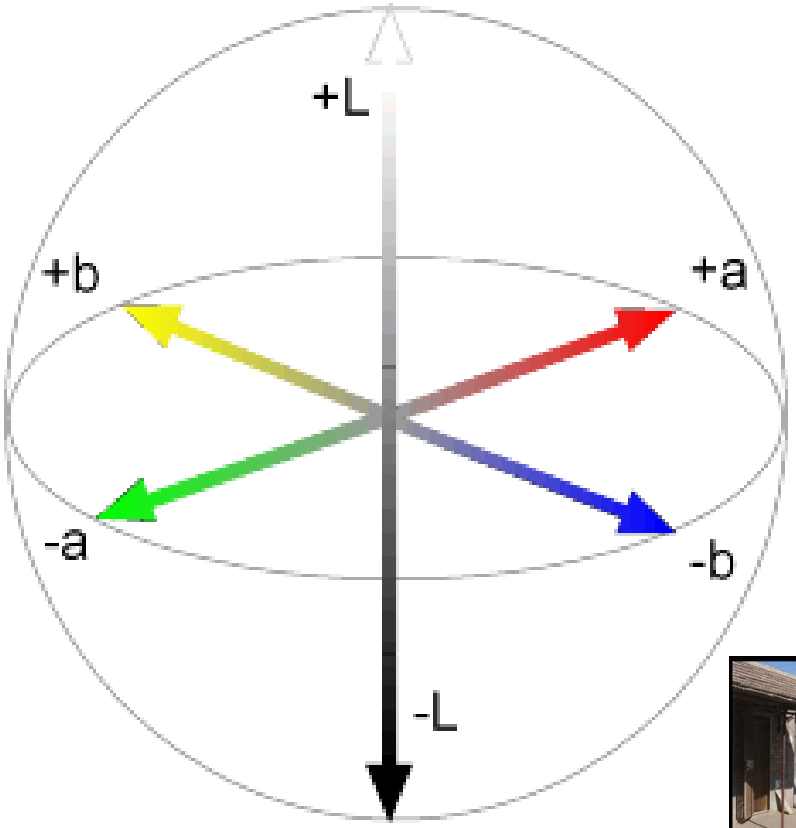
Y
(Cb=0.5,Cr=0.5)

Cb
(Y=0.5,Cr=0.5)

Cr
(Y=0.5,Cb=05)

Color Spaces - LAB

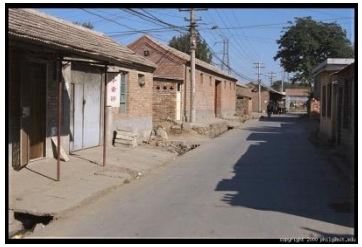
"Perceptually uniform"* color space



L
(a=0,b=0)

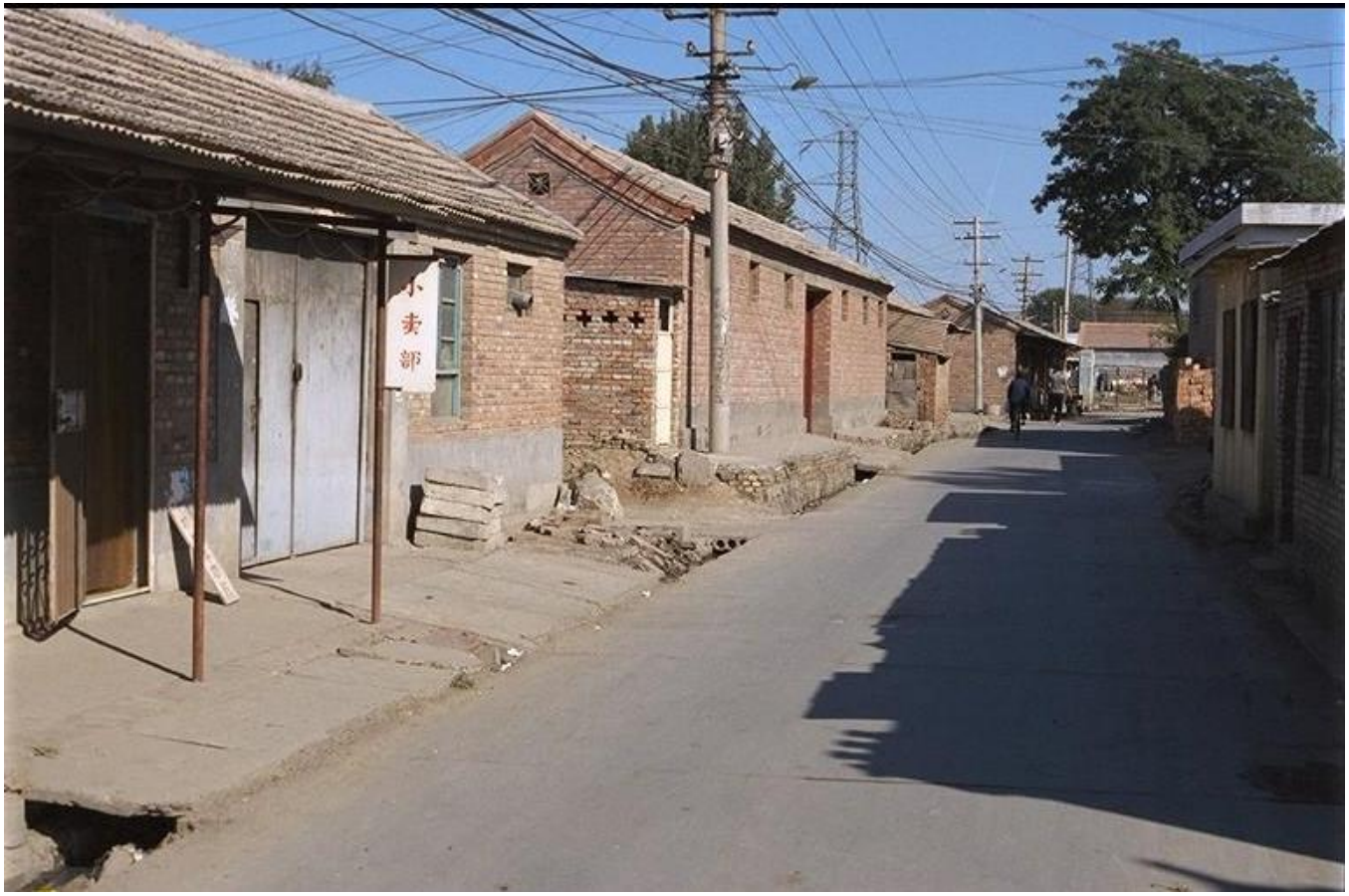
a
(L=65,b=0)

b
(L=65,a=0)



Gray Scale Image – Why?

- Gray Image contains most information

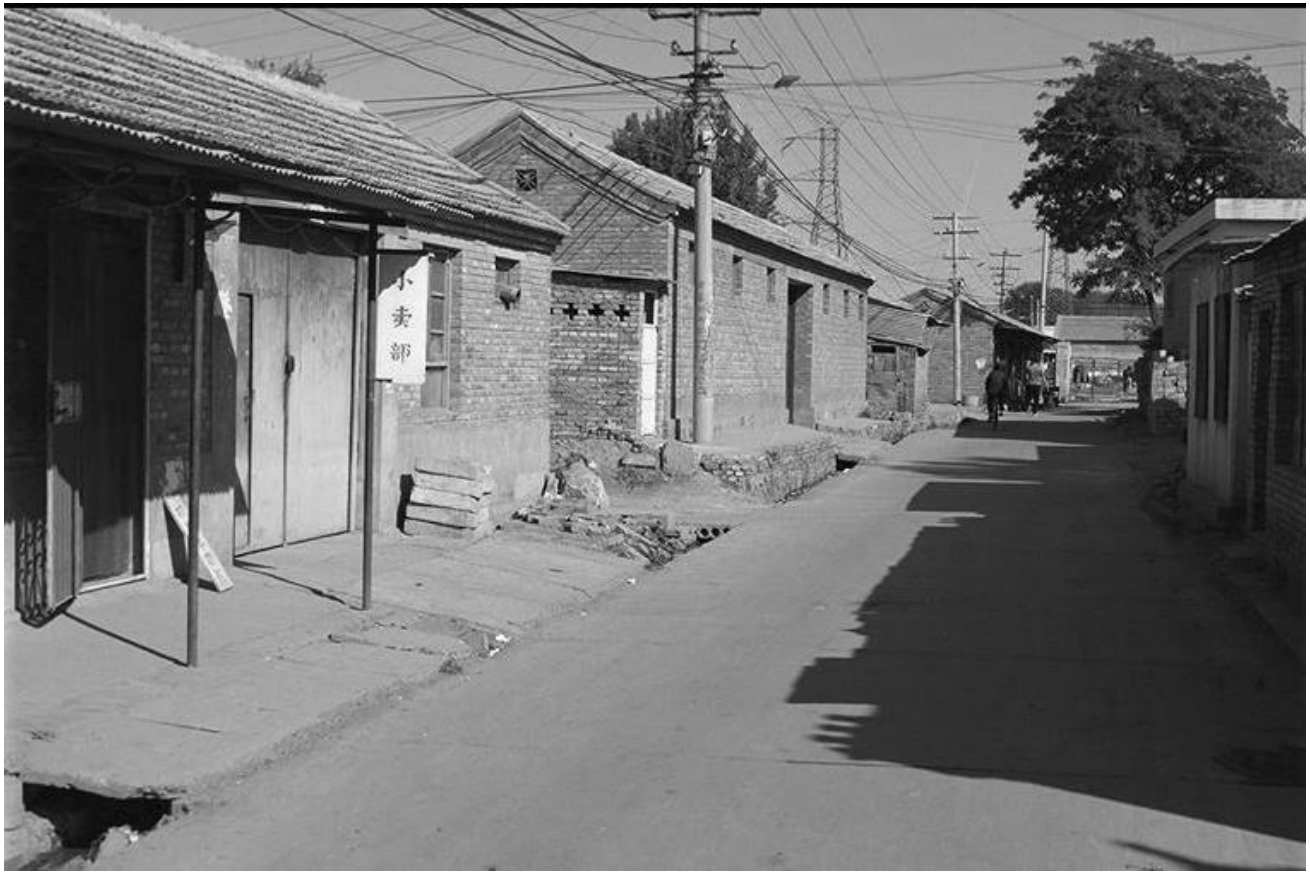


Original Image

Most information in intensity

Only color

Most information in intensity



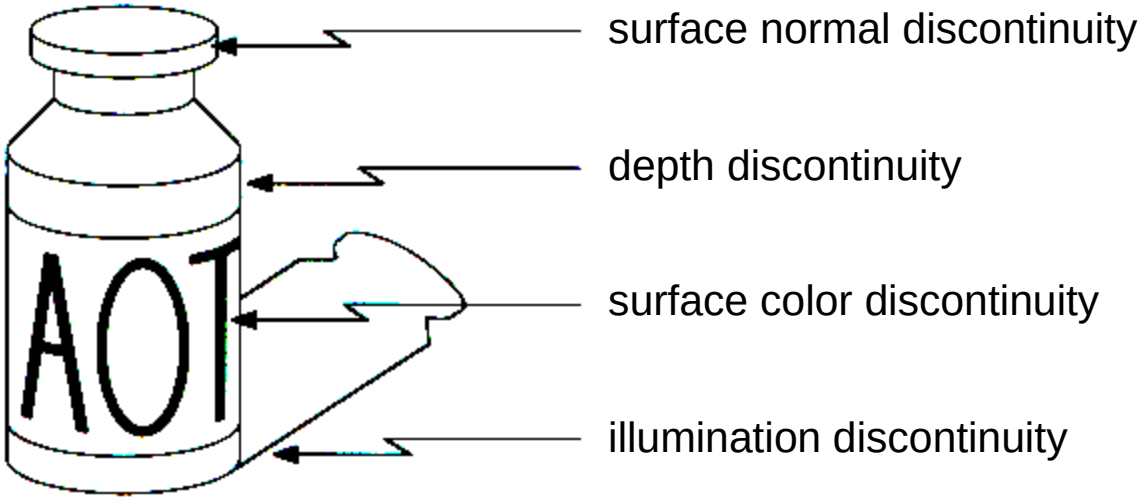
Only intensity

Edge Detection

- Identification of strong changes discontinuities in an image
  - Intuitively, most semantic and shape information encoded
  - More compact than pixels

Source: D. Lowe

Origin of Edges

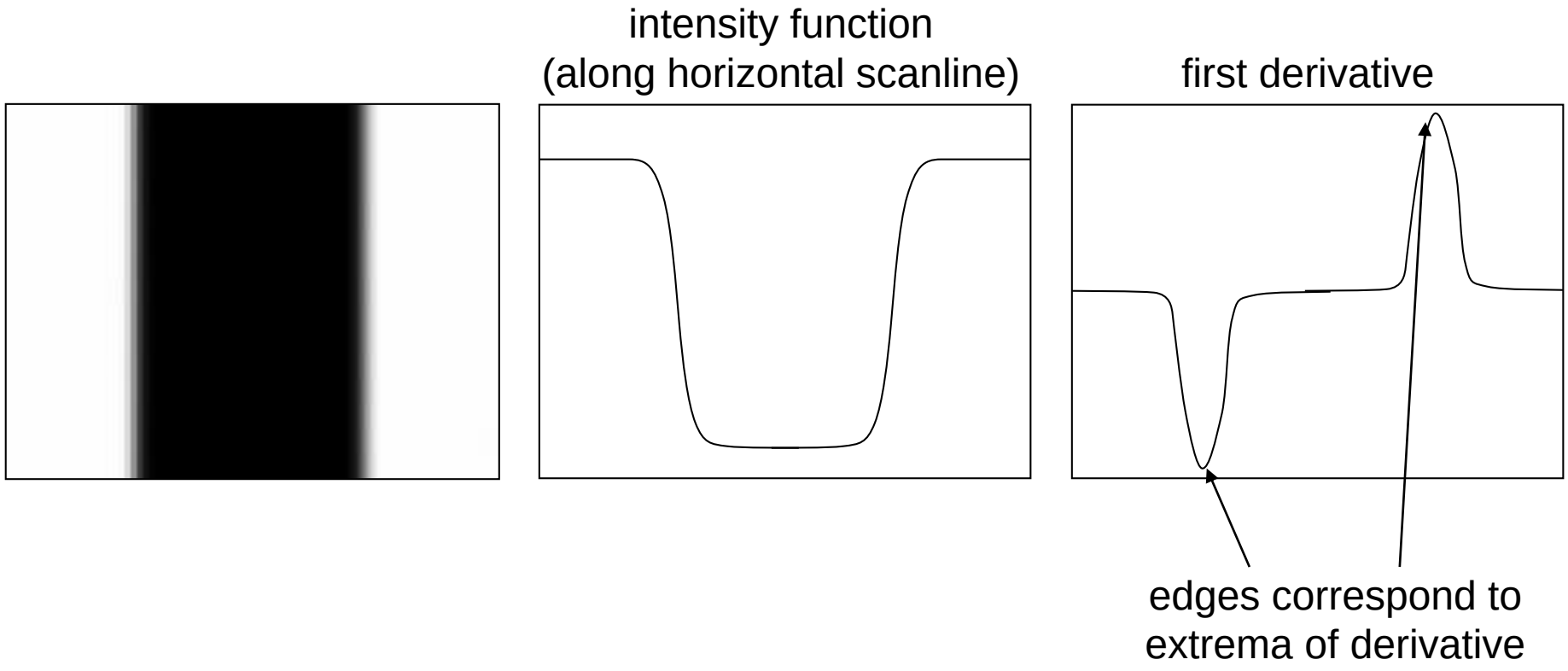


- Edges are caused by a variety of factors

Detecting Edges

- What's an edge?
  - intensity discontinuity (= rapid change)

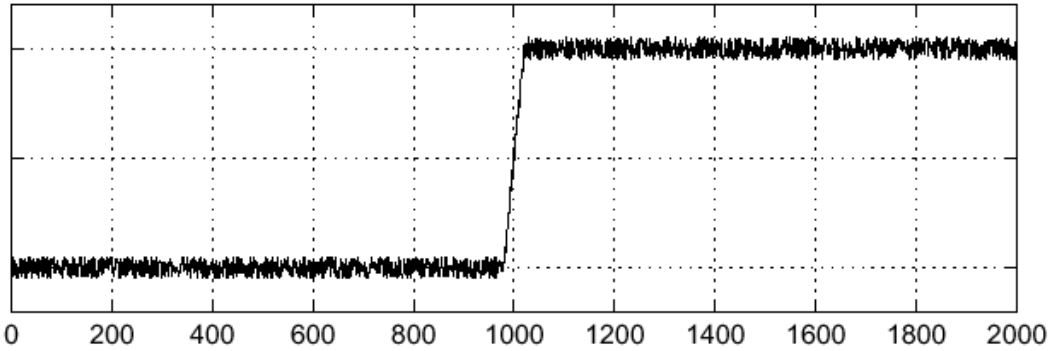
- How can we find large changes in intensity?
  - gradient operator seems like the right solution



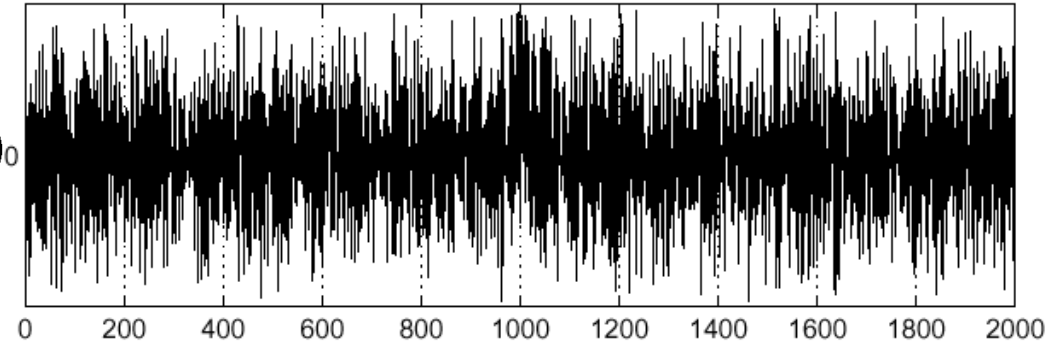
Effects of Noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal

$f(x)$



$\frac{d}{dx} f(x)$



=> Where is the edge?

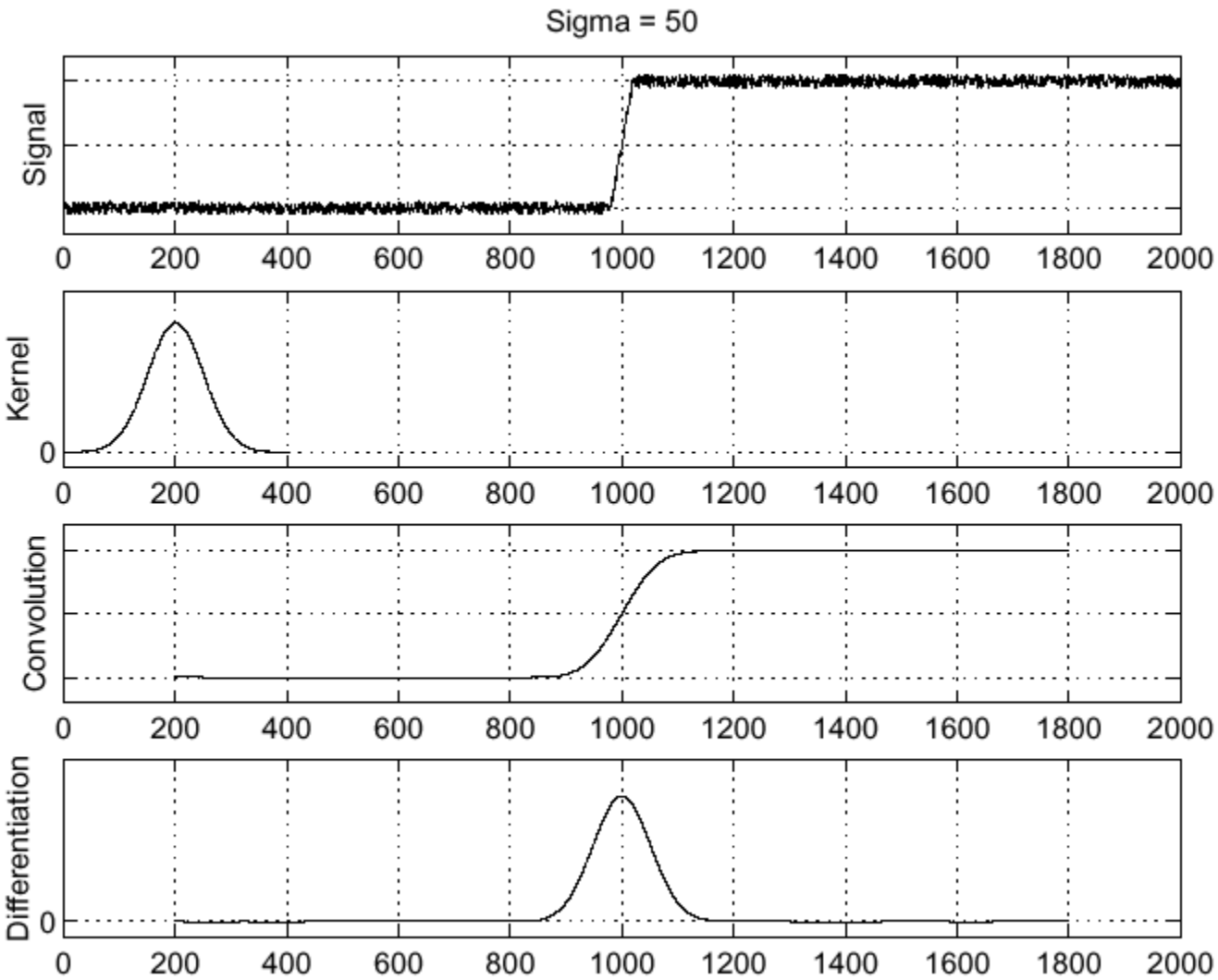
Convolution with Gauss-kernel

f

$$h = \frac{1}{\sigma\sqrt{2\pi}} . e^{-\frac{x^2}{2\sigma^2}}$$

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Edge position at peaks of $\frac{\partial}{\partial x}(h \star f)$

Derivative Theorem of Convolution

- Differentiation of convolution:

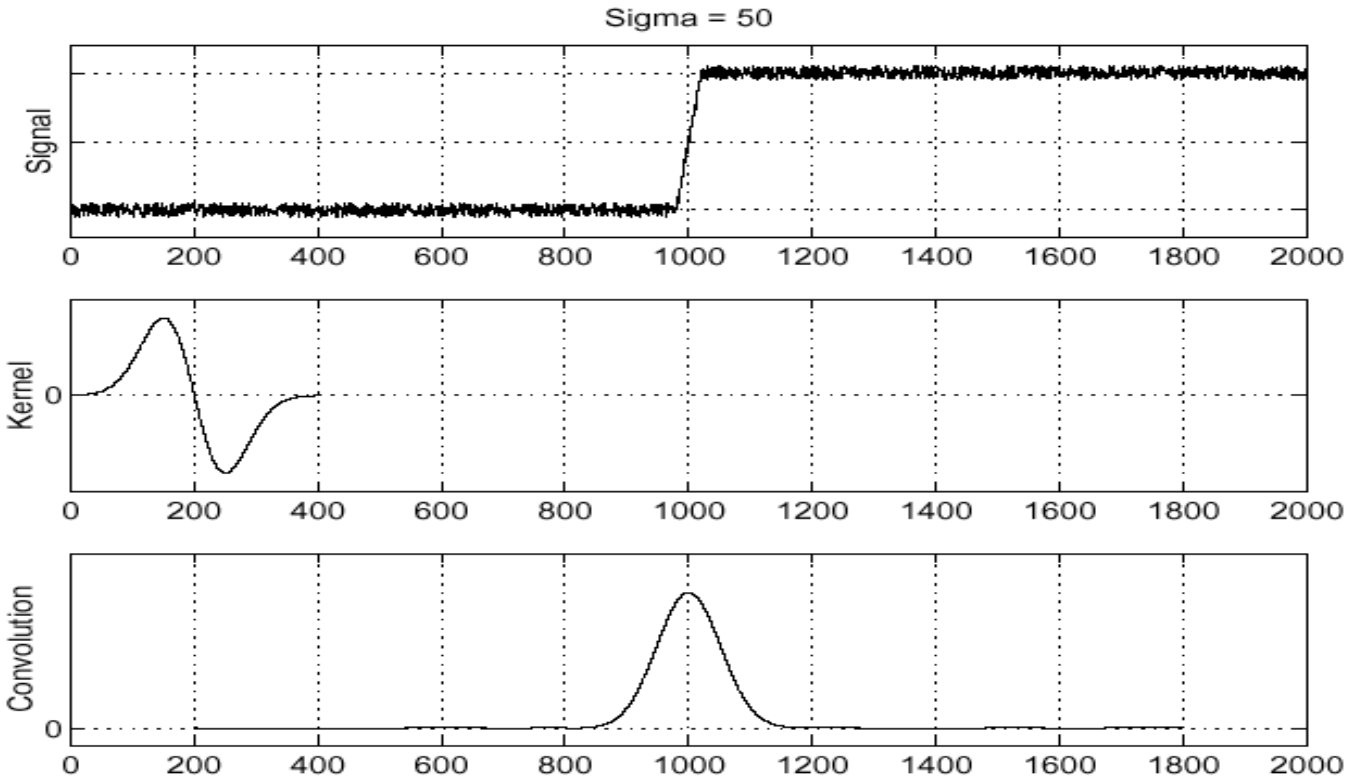
$$\frac{d}{dx}(f * h) = f * \frac{d}{dx} h$$

- Reduction of one operation:

f

$\frac{\partial}{\partial x} h$

$(\frac{\partial}{\partial x} h) \star f$



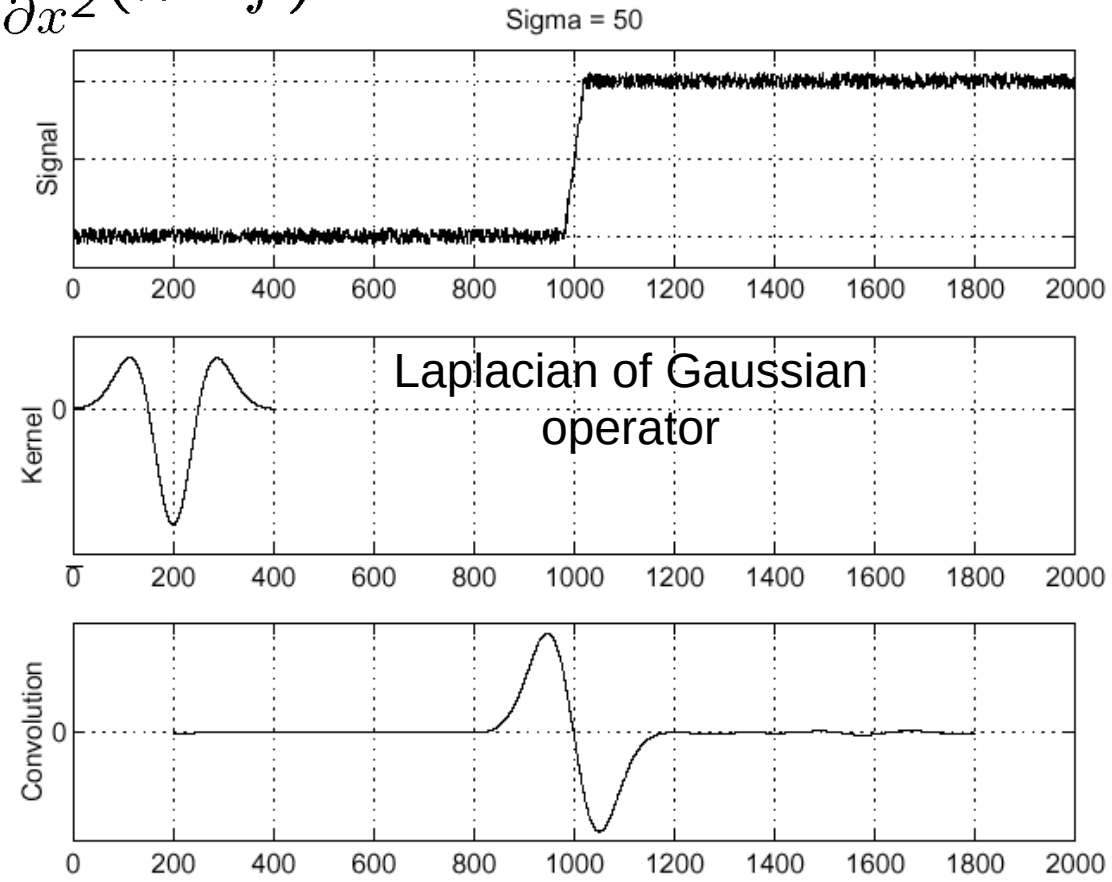
Laplacian of Gaussian

- Looking for maxima and minima of $s'(x)$ means looking for zero-crossings of $s''(x)$
- Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

$\frac{\partial^2}{\partial x^2}h$

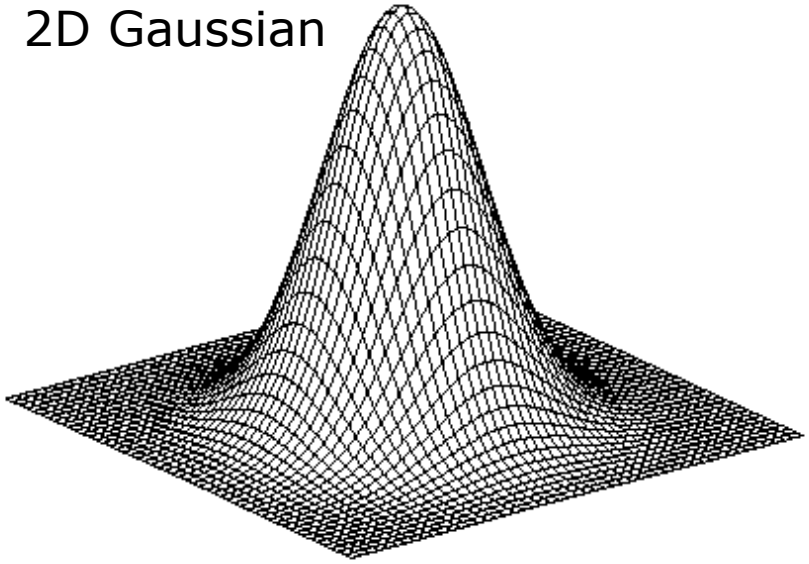
$(\frac{\partial^2}{\partial x^2}h) \star f$

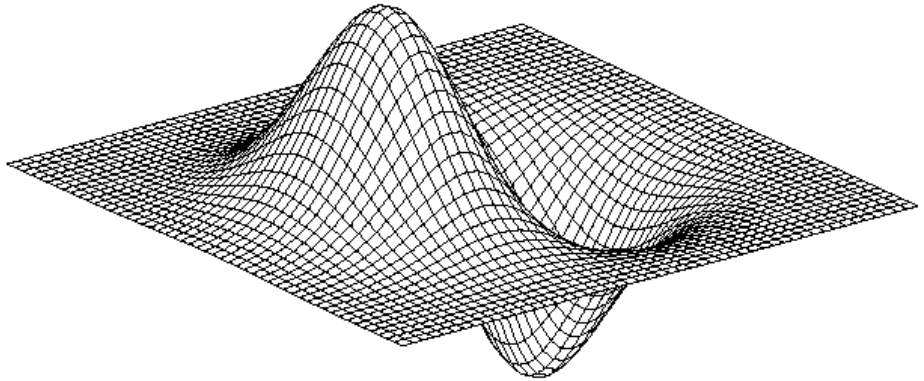


Two-dimensional Edge Detection

The scheme can easily be extended to work in two dimensions

2D Gaussian





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

$$\frac{\partial G(x,y,\sigma)}{\partial x} = -\frac{x}{2\pi\sigma^4} e - \frac{(x^2+y^2)}{2\sigma^2}$$

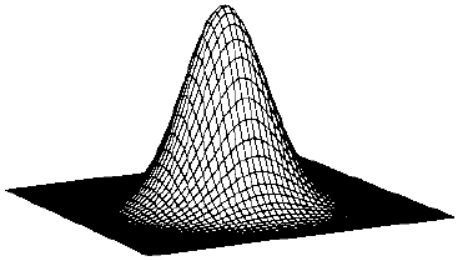
$$\frac{\partial G(x,y,\sigma)}{\partial y} = -\frac{y}{2\pi\sigma^4} e - \frac{(x^2+y^2)}{2\sigma^2}$$

Two-dimensional Laplacian of Gaussian

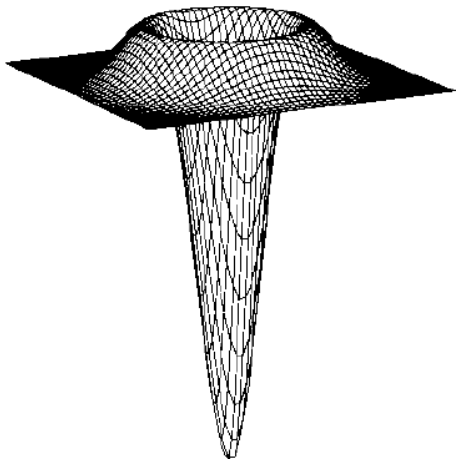
- To apply the search for zero-crossings the Laplacian of G_σ is needed

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

- Isotropic edge detection due to Marr and Hildreth [Marr80] ("Mexican Hat")



G_σ



$\nabla^2 G_\sigma$

Designing of Edge Detector

- Criteria for a good edge detector:
  - good detection: the optimal detector should find all real edges, ignoring noise or other artifacts
  - good localization
    - detected edges must be as close as possible to the real edges
    - the detector must return only one point for each real edge point
- Cues of edge detection
  - differences in color, intensity, or texture across the boundary
  - continuity and closure
  - high-level knowledge

Source: L. Fei-Fei

Canny Edge Detector

- This is probably the most widely used edge detector in computer vision
- Theoretical model: step-edges corrupted by additive Gaussian noise
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of *signal-to-noise ratio* and localization

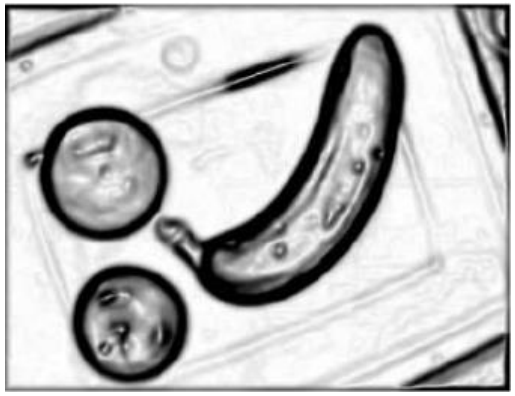
J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Source: L. Fei-Fei

Canny Edge Detection 1

Original Image

- An alternative is a directed edge filter that calculates both magnitude and direction, due to Canny[Canny86]:



Edge strength $|\nabla S|$

- Find the gradient of the smoothed image S (x; y) at every pixel:

$$\nabla S = \nabla(G_\sigma \circ I) = \begin{pmatrix} \frac{\partial G_\sigma}{\partial x} \circ I \\ \frac{\partial G_\sigma}{\partial y} \circ I \end{pmatrix}$$

- Approximation of gradient

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A}$$

Canny Edge Detection 2

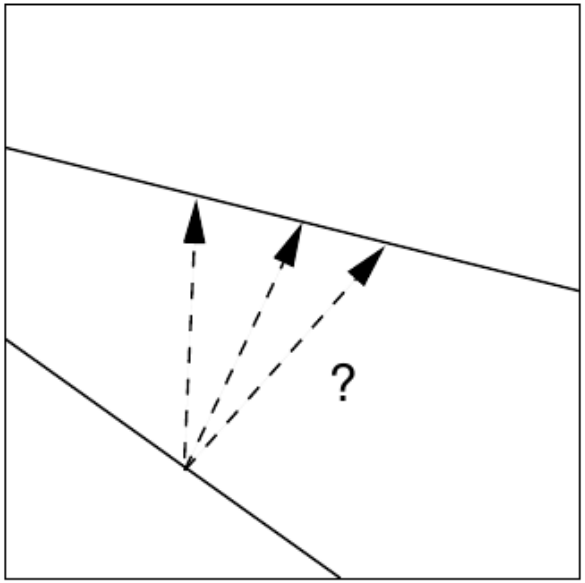
Non-maximal suppression

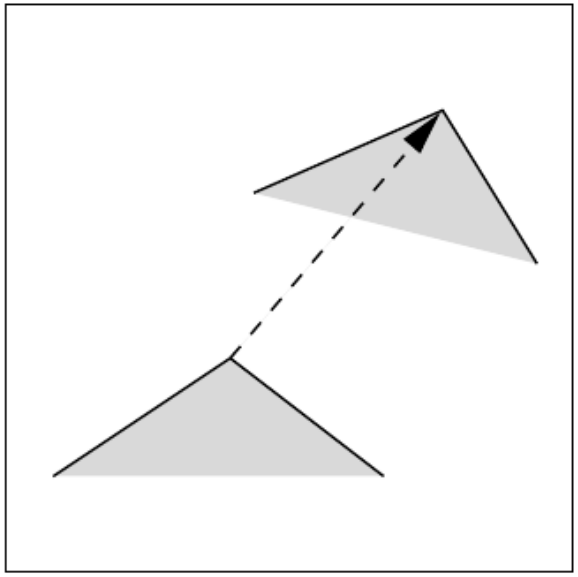
- Non-maximal suppression: Edge elements are located where $|\nabla S|$ is greater than local values of $|\nabla S|$ in the directions of $\pm\nabla S$

Thresholding

- Thresholding: Finally keep only those edges with $|\nabla S|$ above a certain value

Stability of Edges



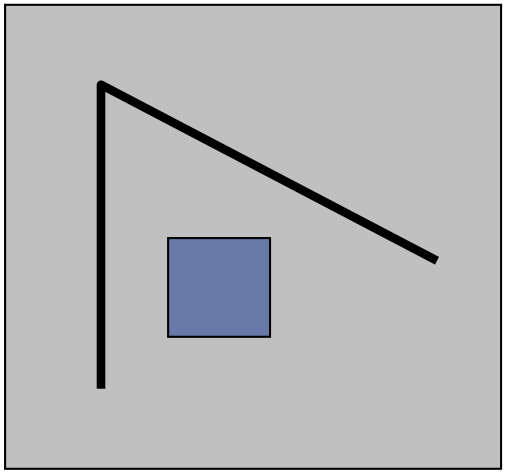


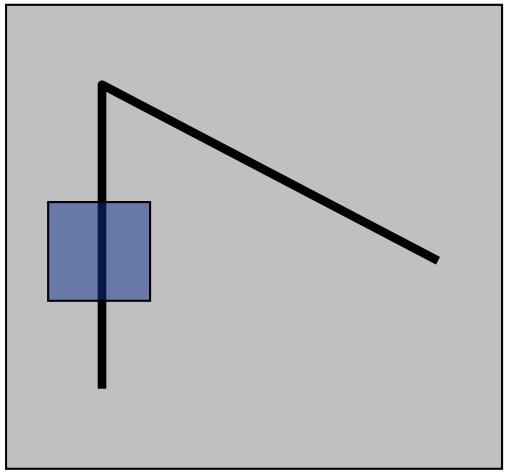
- Edges are good stable features if no motion is involved
- Motion of edges is ambiguous due to the aperture problem: When observing a long, moving edge, only motion normal to the edge can be measured!
- Corners are not subject to this problem

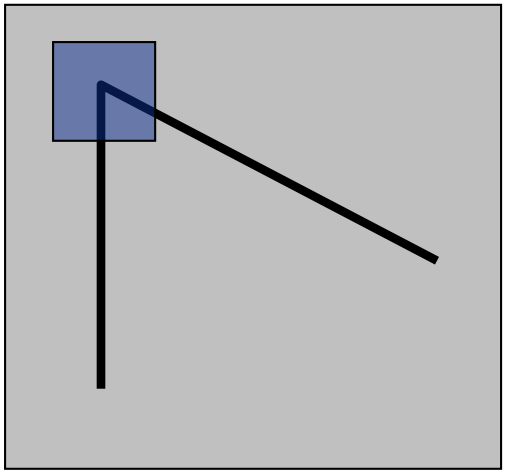
Corner Detection

Suppose we only consider a small window of pixels

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





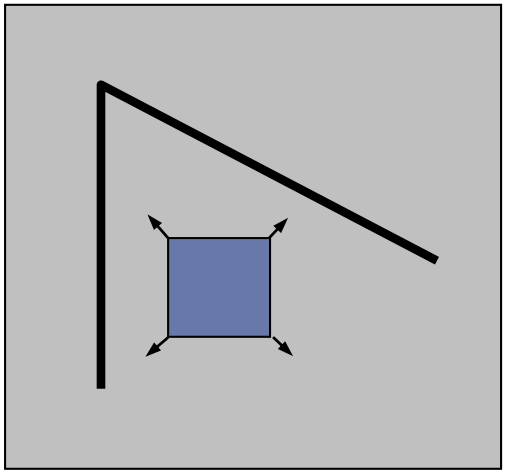


Corners can be used to detect good features to describe object, because there are significant changes around corners.

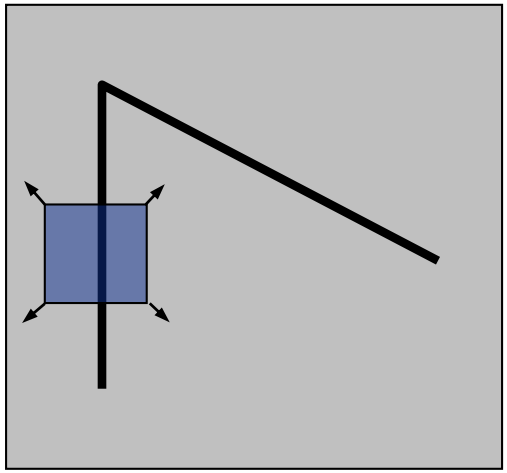
Corner Detection

Basic Idea

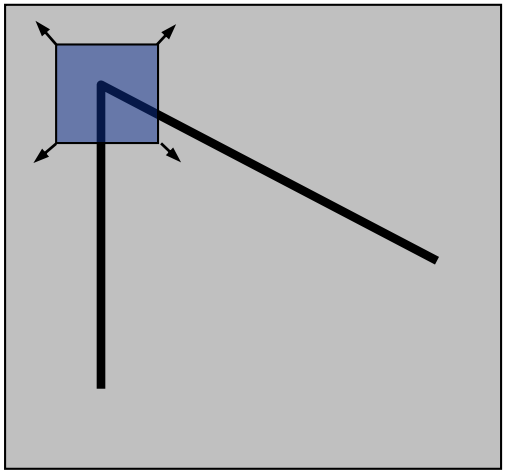
- Recognize the point by looking through a small window
- Shifting the window in *any direction* should give a large change in intensity



"flat" region:
no change in all directions



"edge":
no change along the edge direction

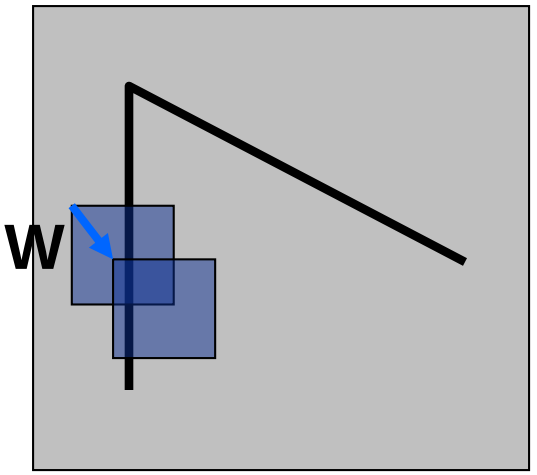


"corner":
significant change in all directions

Mathematical Model of Corners

Consider shifting the window W by (u,v)

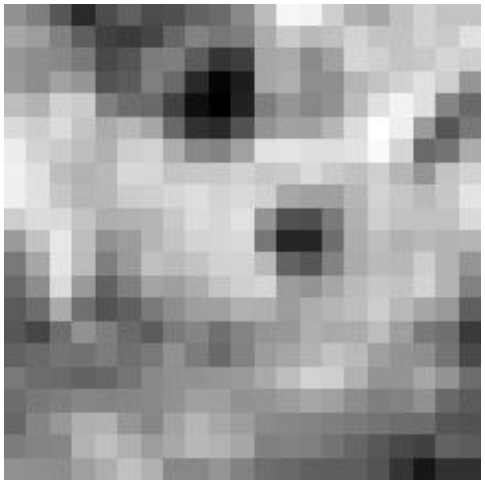
- how do the pixels in W change?
- compare each pixel before and after by summing up the squared differences (SSD)
- this defines an SSD "error" of E(u,v):



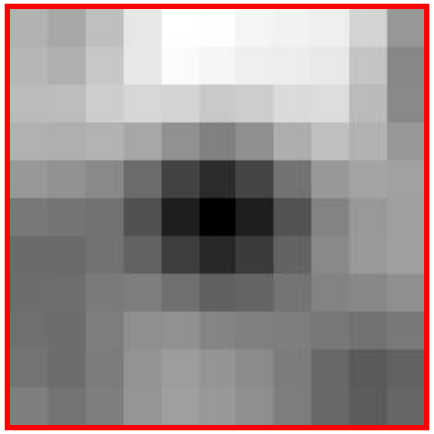
e.g. W is 5x5

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

$I(x, y)$



$E(u, v)$



Mathematical Model of Corners

Change in appearance of window $w(x,y)$ for the shift $[u,v]$:

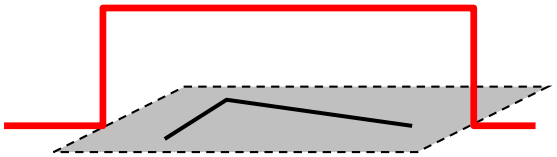
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Window function

Shifted intensity

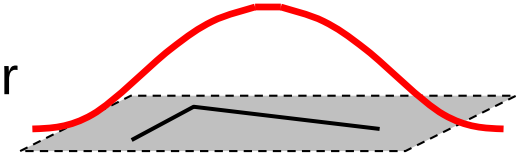
Intensity

Window function $w(x,y)$ =



1 in window, 0 outside

or



Gaussian

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 approximation is adequate

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

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

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

Plugging this into the formula on the previous slide…

Corner detection: Math

Consider shifting the window W by (u,v)

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

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

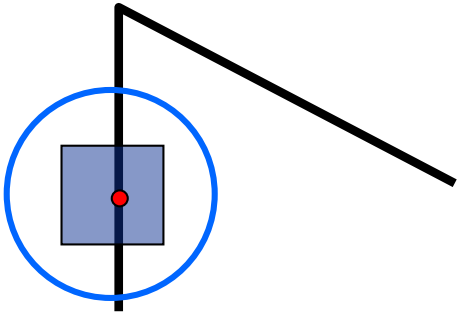
$$= \sum_{(x,y)\in w} w(x,y)\left[\begin{bmatrix} I_x & I_y \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}\right]^2$$

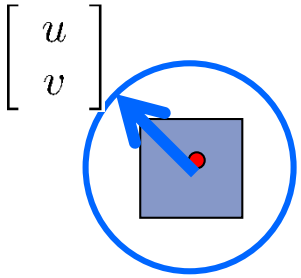
$$= \sum_{(x,y)\in w} w(x,y)\begin{bmatrix} u & v \end{bmatrix}\begin{bmatrix} I_x^2 & I_xI_y \\ I_yI_x & I_y^2 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

$$= \begin{bmatrix} u & v \end{bmatrix}\underbrace{\sum_{(x,y)\in w} w(x,y)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}}_{H}\begin{bmatrix} u \\ v \end{bmatrix}$$

Feature detection: Math

H is called Hessian - “structure tensor of the local neighborhood of (x, y)”





- 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

Eigenvalue

- **Eigenvectors** of a matrix **A** are the vectors **x** that satisfy:

$$Ax = \lambda x$$

- Scalar λ is the **eigenvalue** corresponding to **x**
  - eigenvalues are found by solving:

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

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

  - The solution:

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

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

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

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

Harris Corner Detection

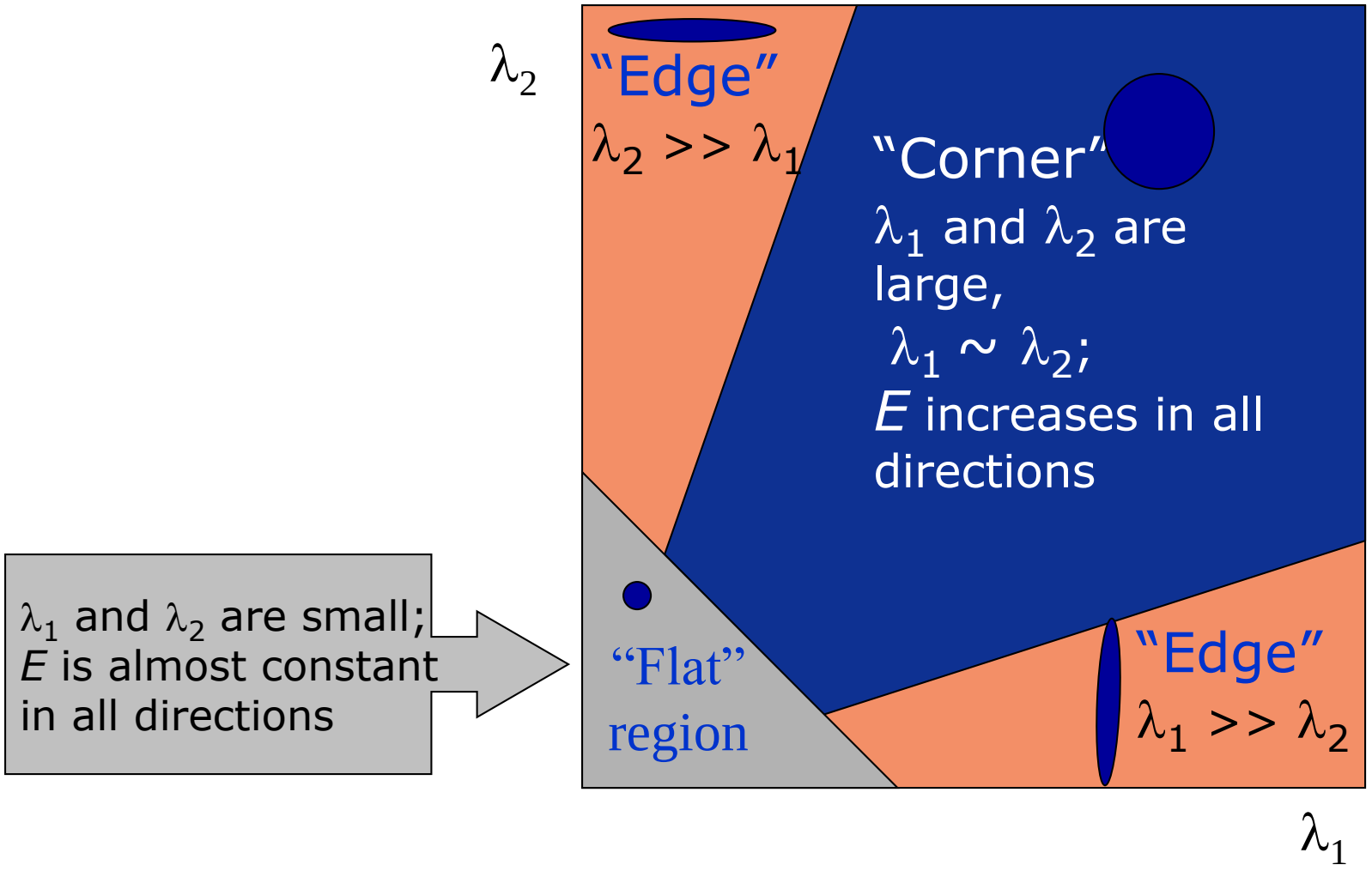
- Let λ_1, λ_2 be the Eigenvalues of Matrix H:
  - No Structure: $\lambda_1 \approx \lambda_2 \approx 0$ => (smooth variation)
  - 1D Structure: $\lambda 1 \approx 0$ (direction of edge),$\lambda_2 \ggg 0$ (normal)
  - 2D Structure: $\lambda_1 \ggg 0, \lambda_2 \ggg 0$ (corner)

- Algorithm
  - compute H matrix for each image pixel
  - find points whose surrounding window give large corner response (R> threshold)
  - take the points of local maxima, i.e., perform non-maximum suppression

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

Interpreting the Eigenvalues

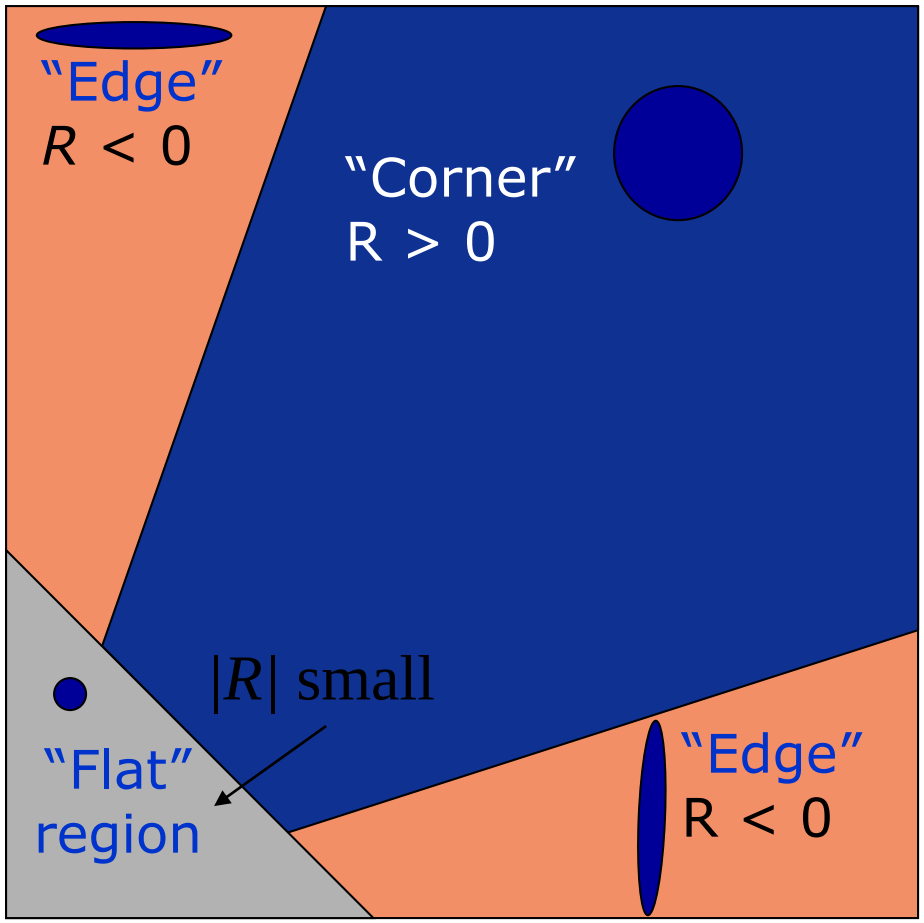
Classification of image points using eigenvalues of *M*:



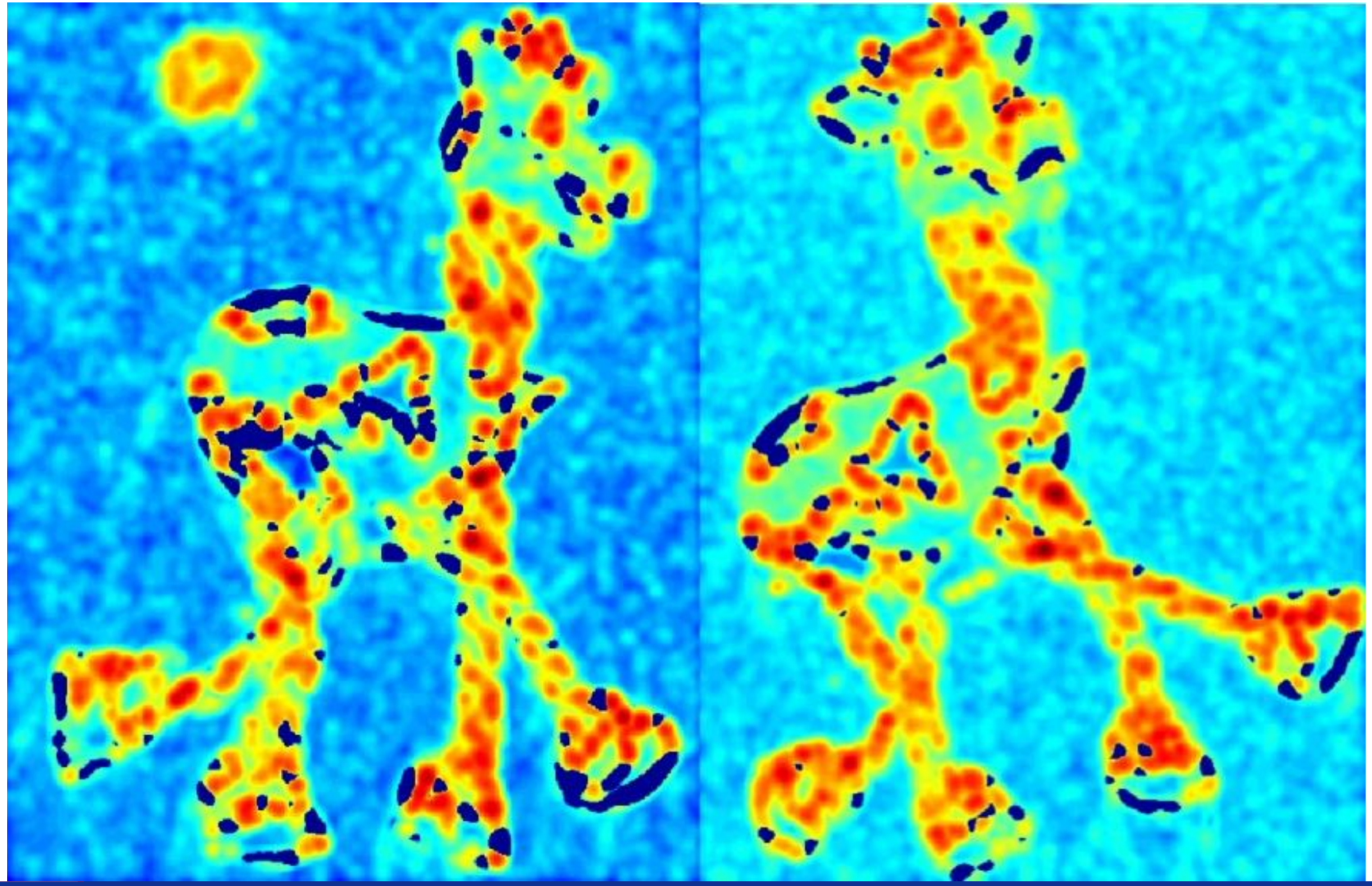
Corner Response Function

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

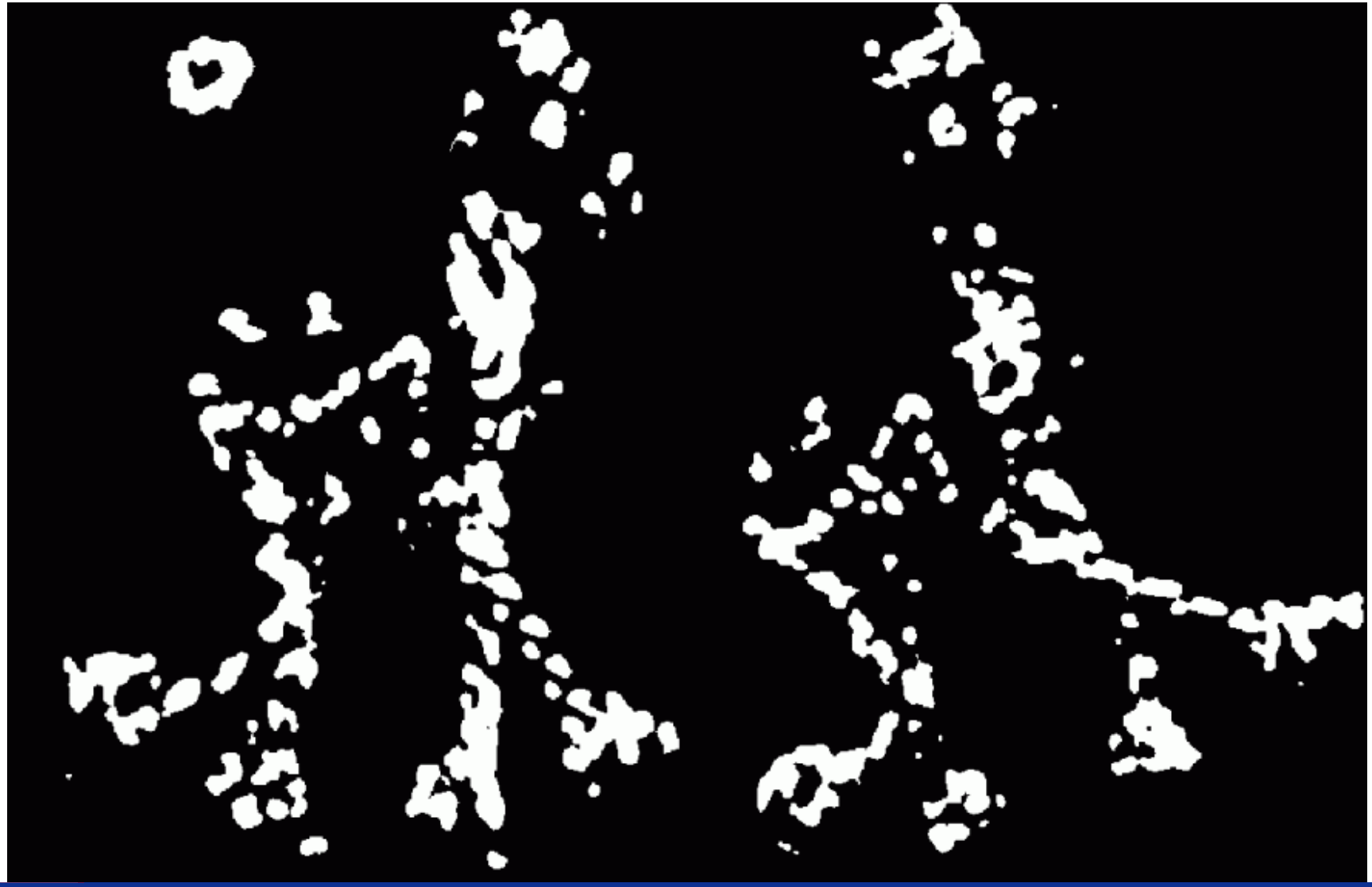
a: constant (0.04 to 0.06)



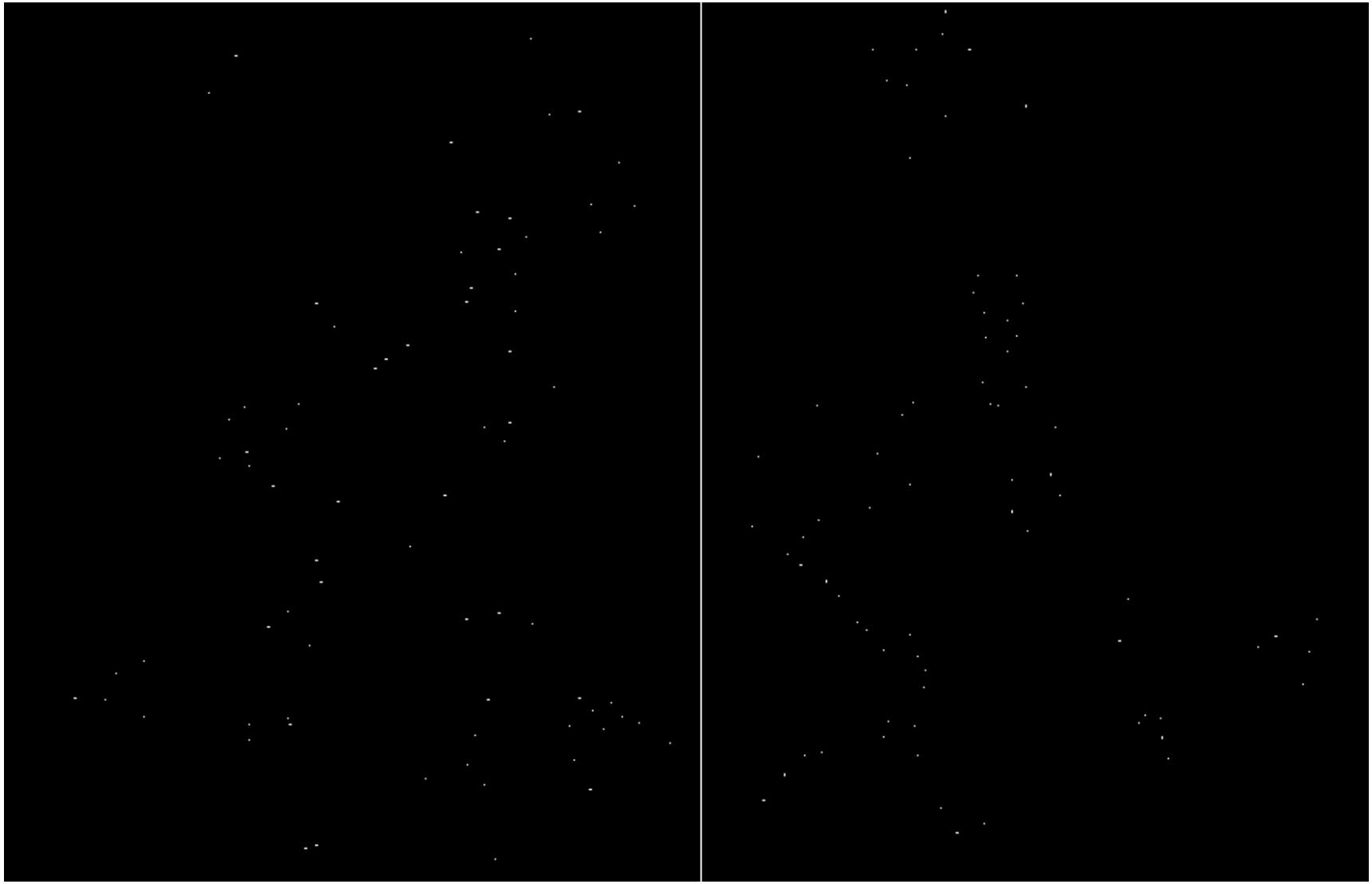
R value (red high, blue low)



Threshold (R > value)



Non-maximal suppression



Harris features (in red)

Features from Accelerated Segment Test

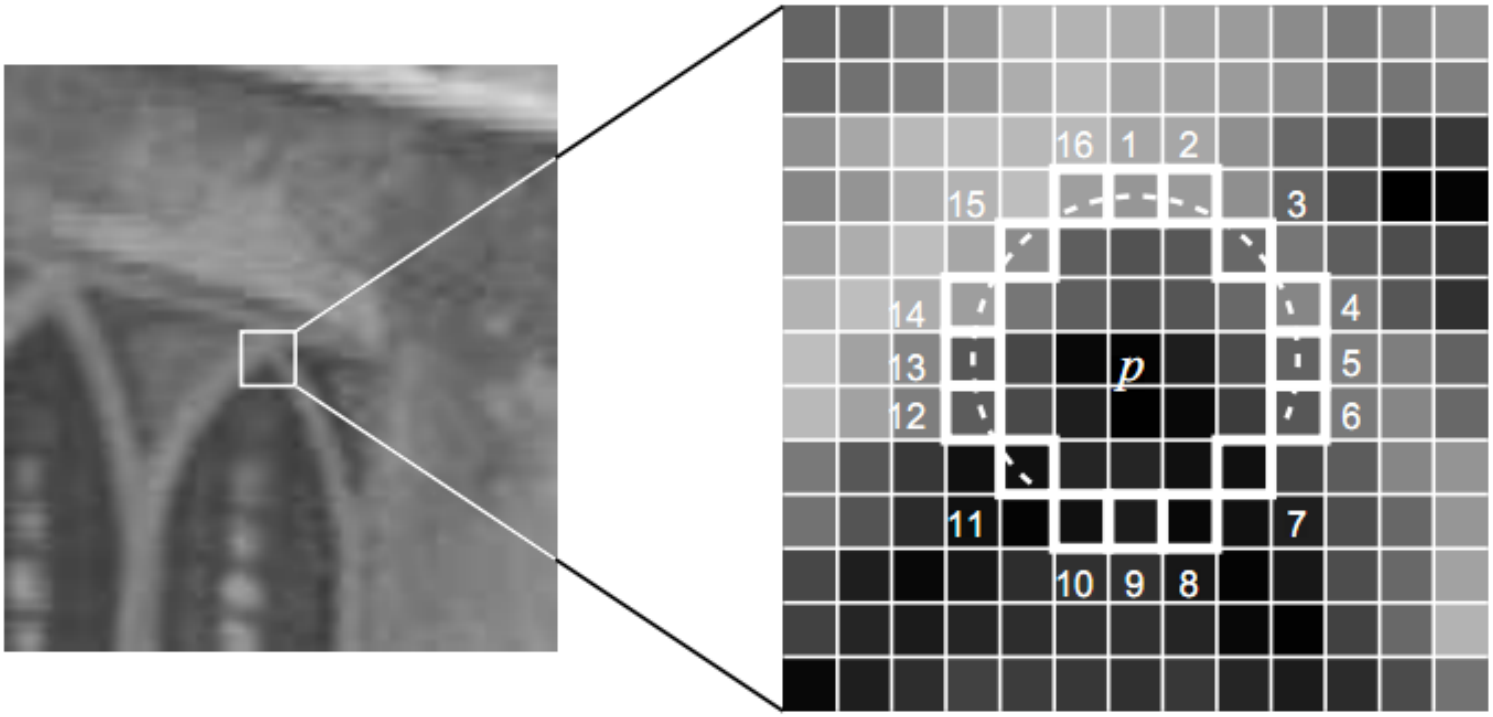
- A very efficient algorithm for corner detection is called Features from Accelerated Segment Test (FAST)

- For a corner candidate c in the image the following is calculated:
  - Considering the 16 pixels at a radius of 3 pixels around c (see image next slide)

  - Finding the longest uninterrupted sequence of pixels, whose intensity is either greater than that of c plus a threshold or less that that of c minus the same threshold

  - If length of sequence is at least 12 pixels, then c is a corner

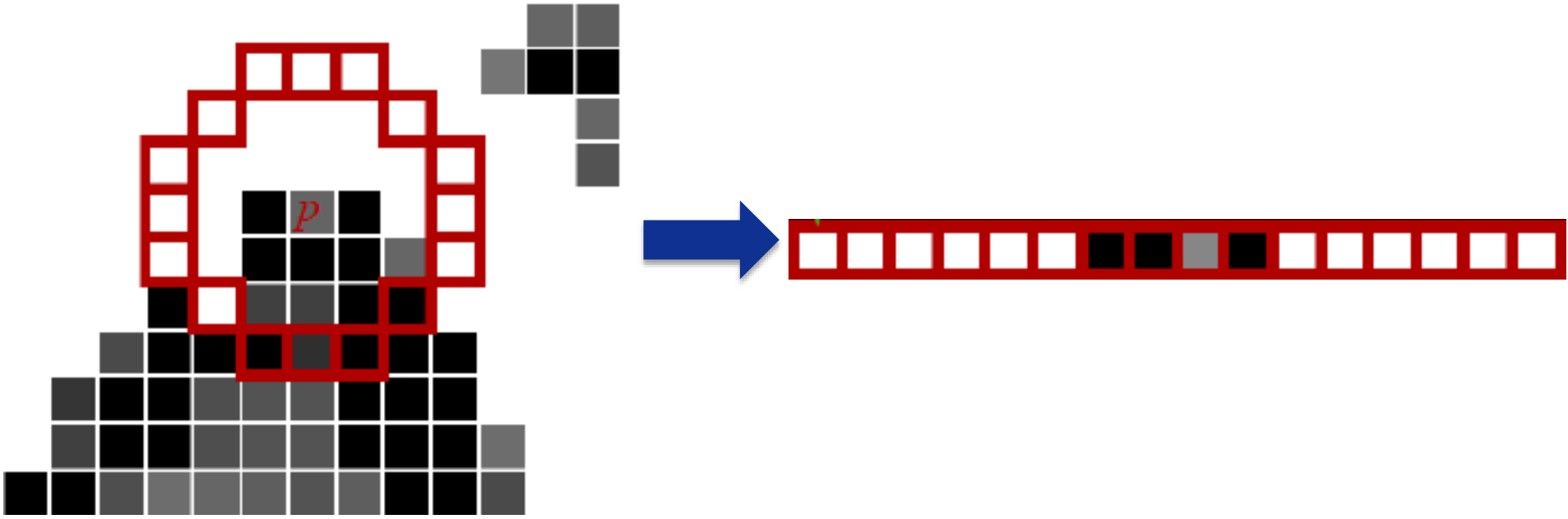
FAST



FAST

- The FAST algorithm can be made even faster by first checking only pixels 1, 5, 9, and 13. If at least three of them fulfill the intensity condition, we can immediately rule out that the given point is a corner.
- In order to avoid detecting multiple corners in the neighborhood near the same pixel, we can require a minimum distance between corners.
- If two corners are too close, we only keep the one with the highest corner score.
- Such a score can be computed as the sum of intensity differences between c and the pixels in sequence.

FAST Version2

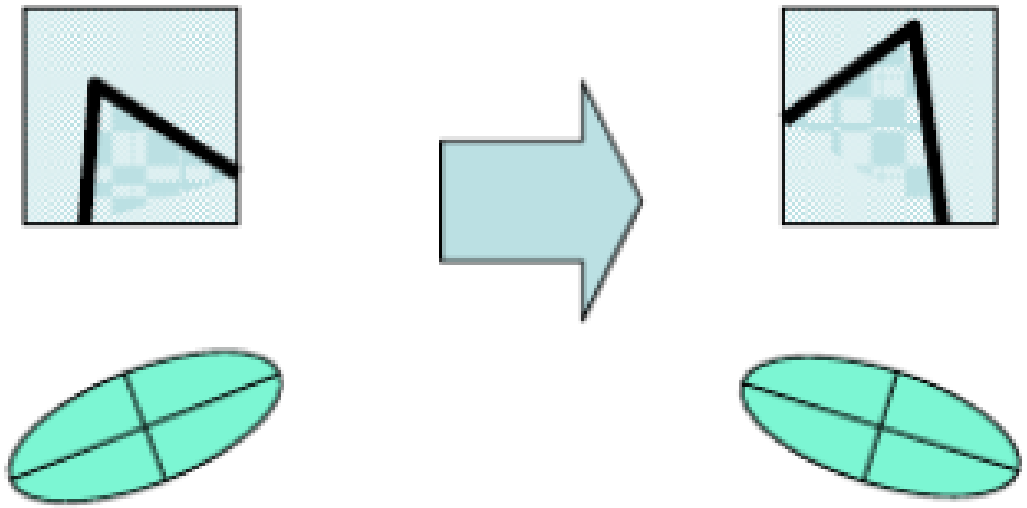


- Pixels are either:
  - much brighter
  - much darker
  - similar
- Presenting ring as a vector
- Classify vector

Conclusion to FAST

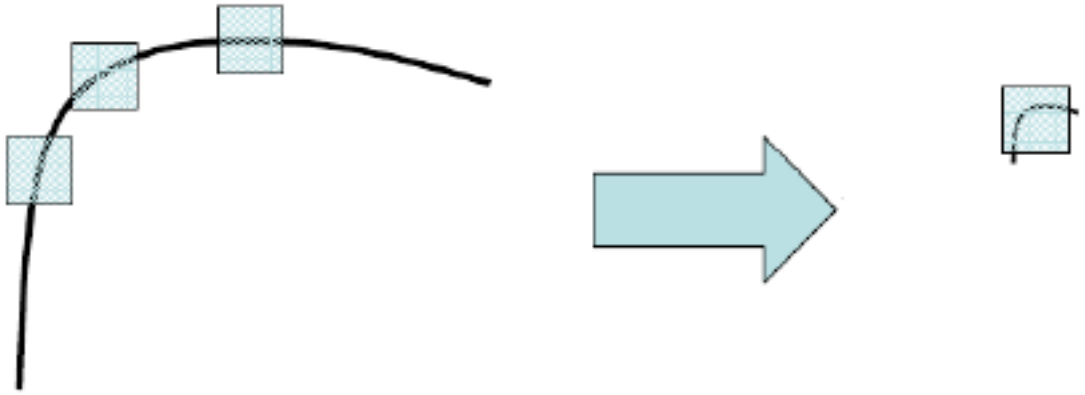
- Pros
  - Very fast
  - High quality feature detection
- Cons
  - Not robust to high level noise
  - Dependent on a threshold

Stability of Corners 1



- Corners are invariant with respect to rotation during rotation the ellipsis changes, but the Eigenvalues remain the same

Stability of Corners 2

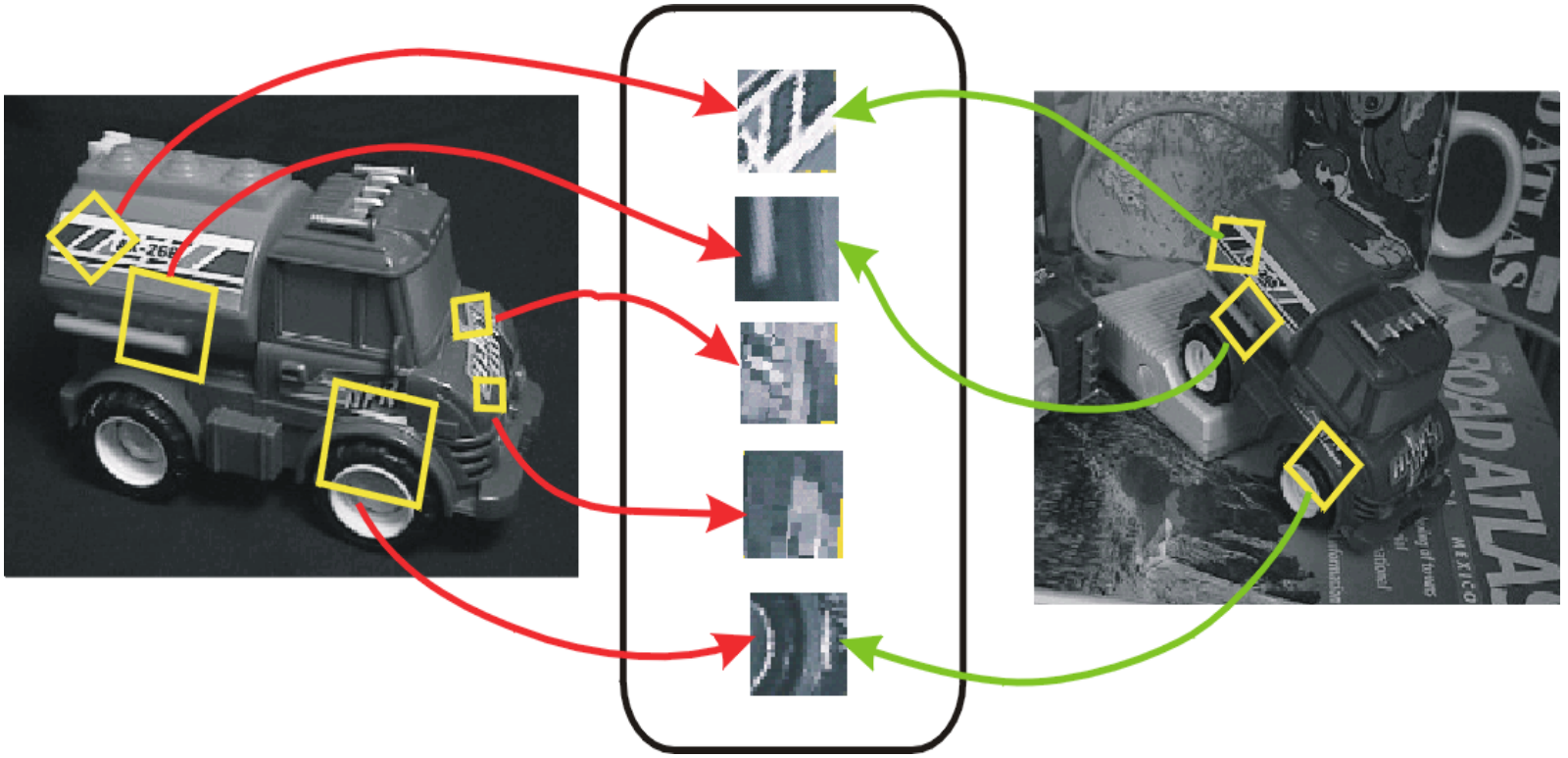


- Corners are not invariant with respect to scale
- After downscaling some edges might appear as corners

Invariant Local Features

Find features that are invariant to transformations

- geometric invariance: translation, rotation, scale
- photometric invariance: brightness, exposure, …



Advantages of Local Features

Locality

- features are local, e.g. robust to occlusion and clutter

Distinctiveness:

- can differentiate objects of a large dataset

Quantity

- hundreds or thousands in a single image

Efficiency

- calculation in real-time

Generality

- exploit different types of features in different situations

More motivation…

- Feature points are used for:
  - image alignment (e.g., mosaics)
  - 3D reconstruction
  - motion tracking
  - object recognition
  - indexing and database retrieval
  - robot navigation
  - … other

What is a useful Feature?

- Uniqueness
  - select image regions that are unusual
  - lead to unambiguous matches in other images
  - how to define “unusual”?
  - **corner detection can be used to solve “uniqueness” problem**

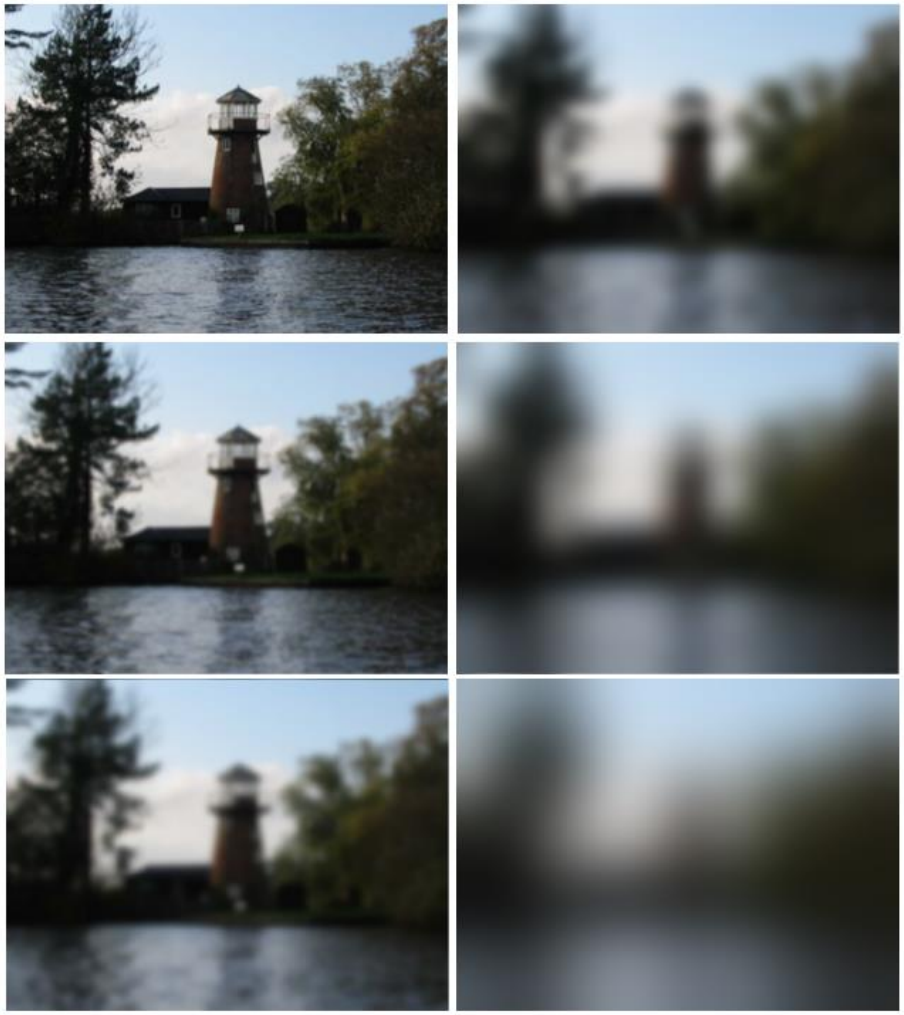
- Invariance due to
  - rotation and shift
  - brightness
  - Scale

=> Corners are robust to change of rotation and intensity but not to the change of scaling.

Blobs and Corners

- Blobs are regions of continuous intensity in every direction
- Can be detected like-wise looking for extrema of Laplacian of Gaussian
- Apart from that, they have similar properties like corners
- Corners are blob center at relevant resolution

Scale Space Representation (L)



- scale independence by looking at different resolutions
- infinite resolutions of one image as 3D function:

$$L(x,y,t) = G(x,y,t) \circ I(x,y)$$

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

$$t = \sigma^2$$

- L defines the scale space
- Blob/corner is local max/min (black+/white-) of $\nabla^2 L = L_{xx} + L_{yy}$

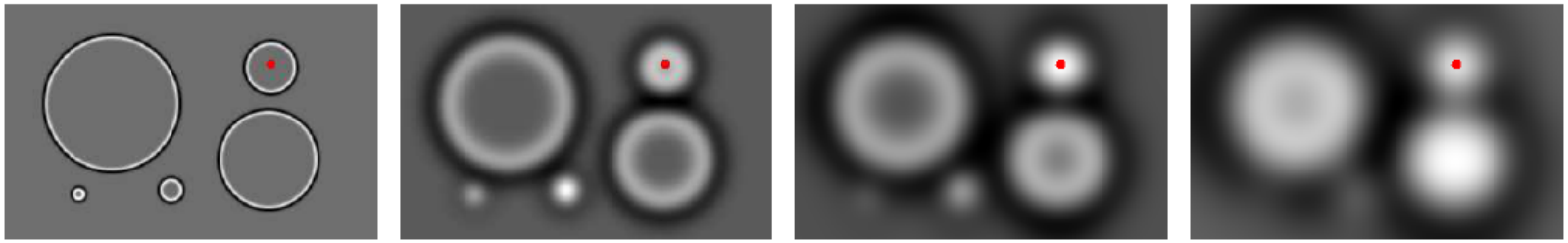
(Laplacian of Gaussian (LoG))

Blob/corner as extrema $(\hat{x}, \hat{y}; \hat{t}) = \text{argmaxminlocal}_{(x,y;t)}(\nabla^2_{norm} L(x,y;t))$

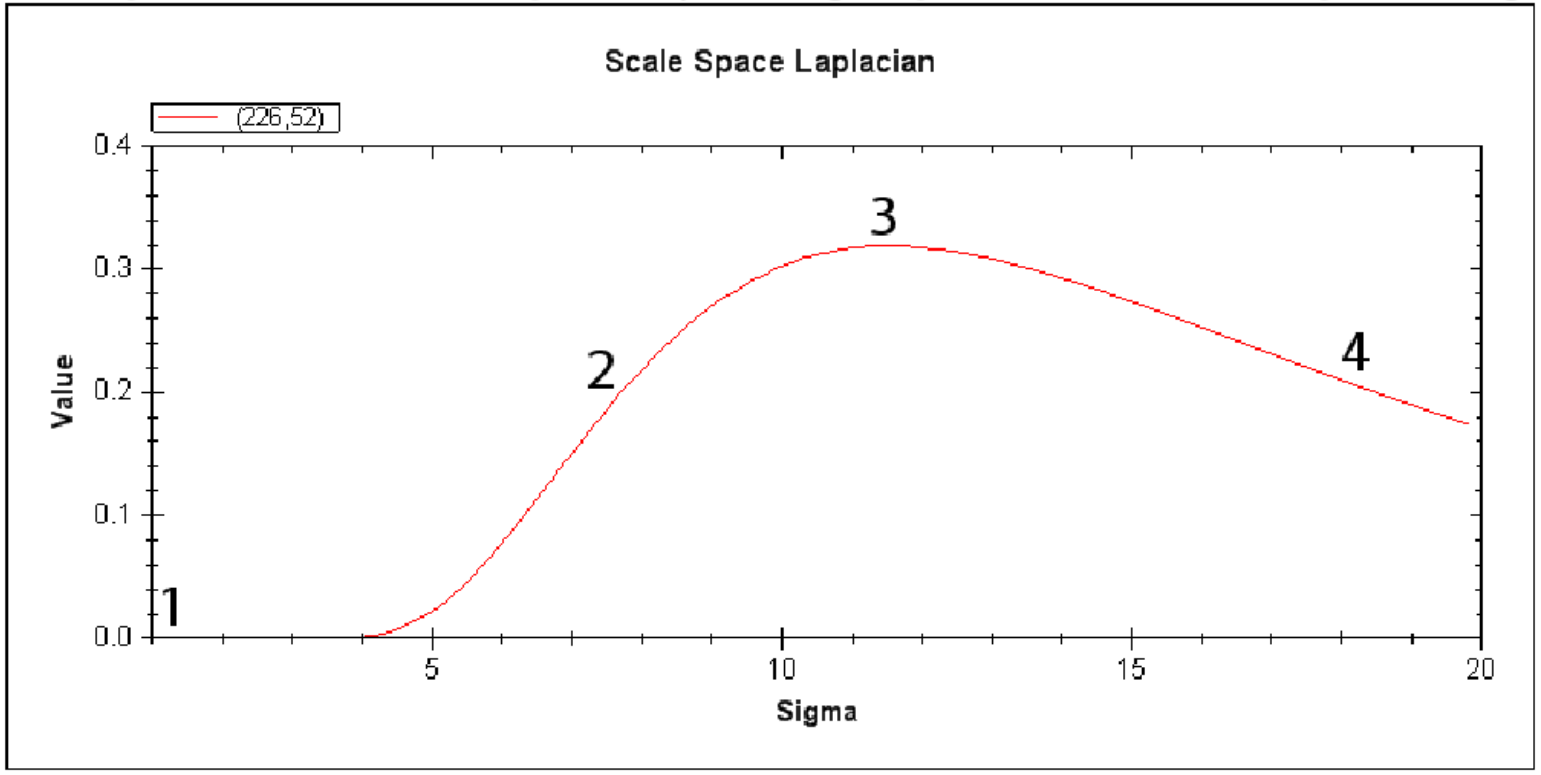
Sampling of Scale Space

- looking at complete scale space is impossible
- choose particular resolutions and examine scales from σ to 2σ (1 Octave)

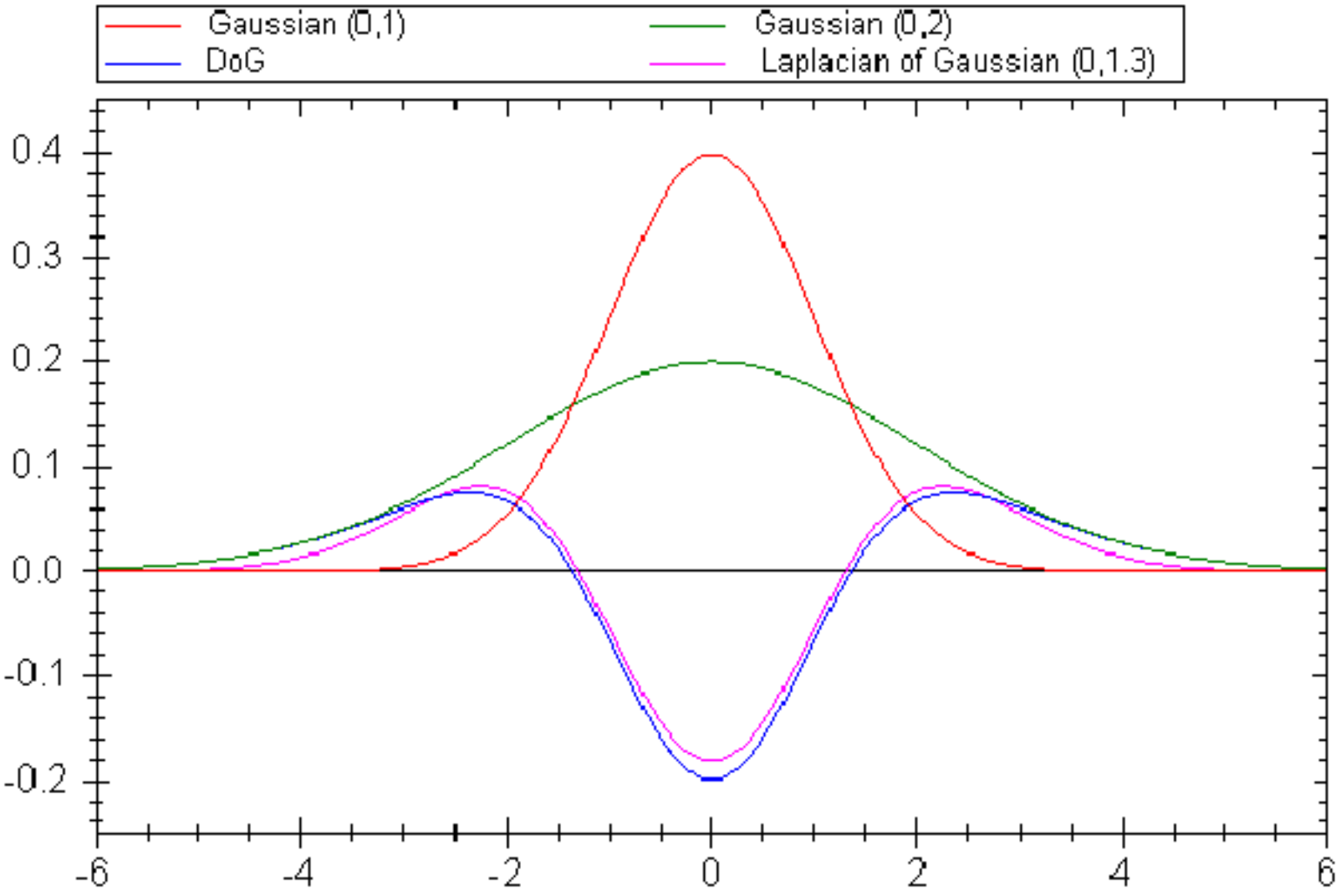
Scale Space Laplacian



$1: \sigma = 1 \quad 2: \sigma = 7 \quad 3: \sigma = 12 \quad 4: \sigma = 18$

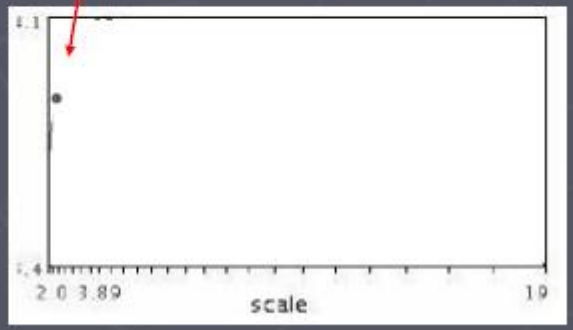


Difference of Gaussian



Automatic scale selection

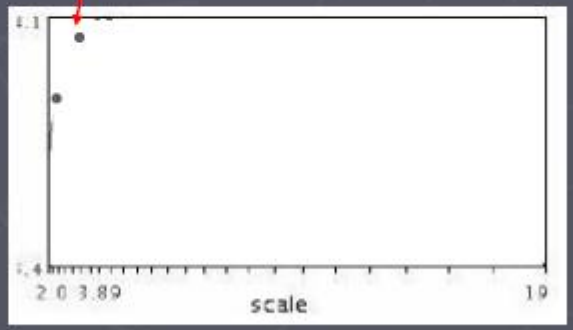
Lindeberg et al., 1996



$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

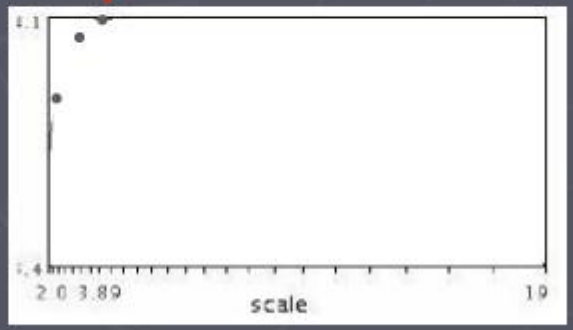
Slide from Tinne Tuytelaars

Automatic scale selection



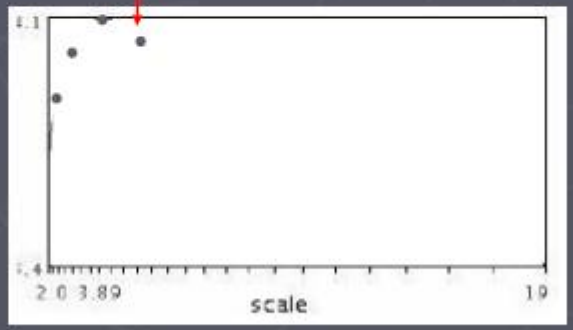
$f(I_{i_1 \ldots i_m}(x,\sigma))$

Automatic scale selection



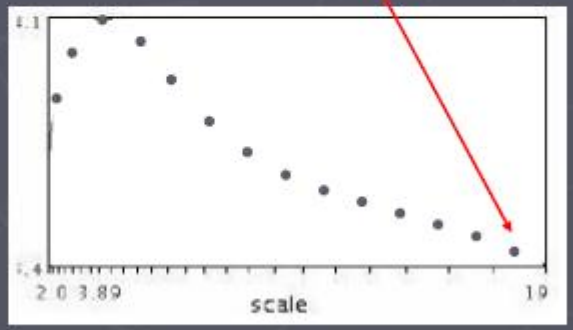
$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection



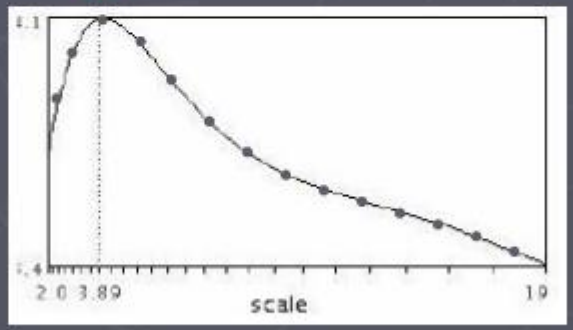
$f(I_{i_1 \ldots i_m}(x,\sigma))$

Automatic scale selection



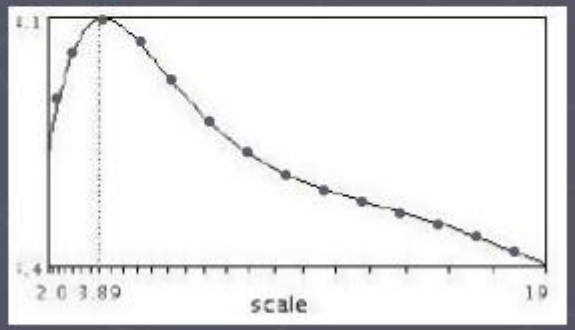
$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection

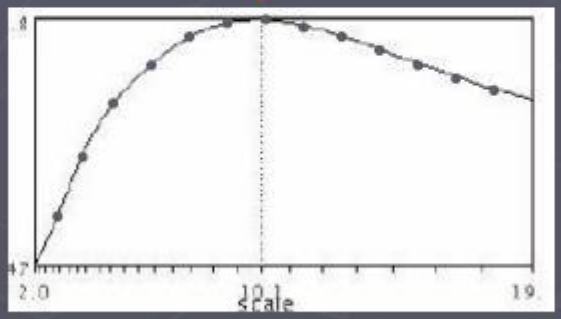


$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection



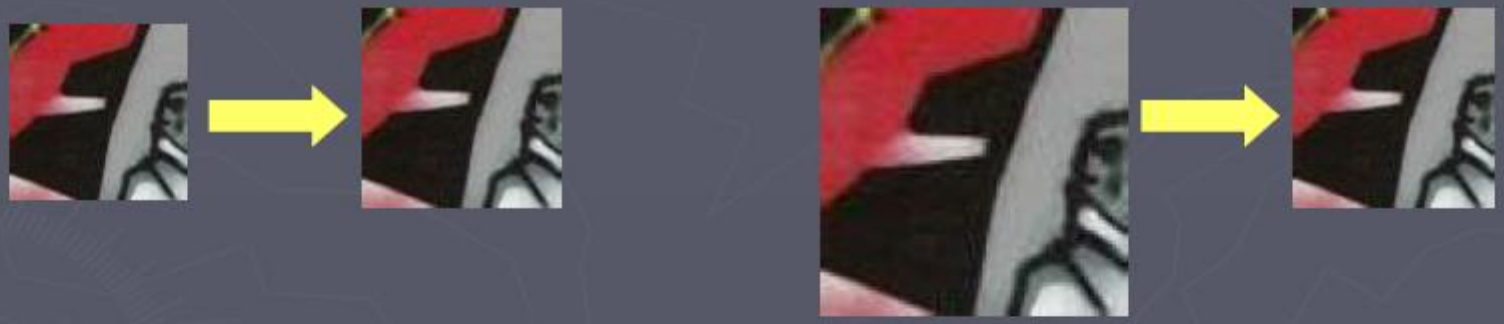
$f(I_{i_1 \ldots i_m}(x,\sigma))$



$f(I_{i_1 \ldots i_m}(x',\sigma'))$

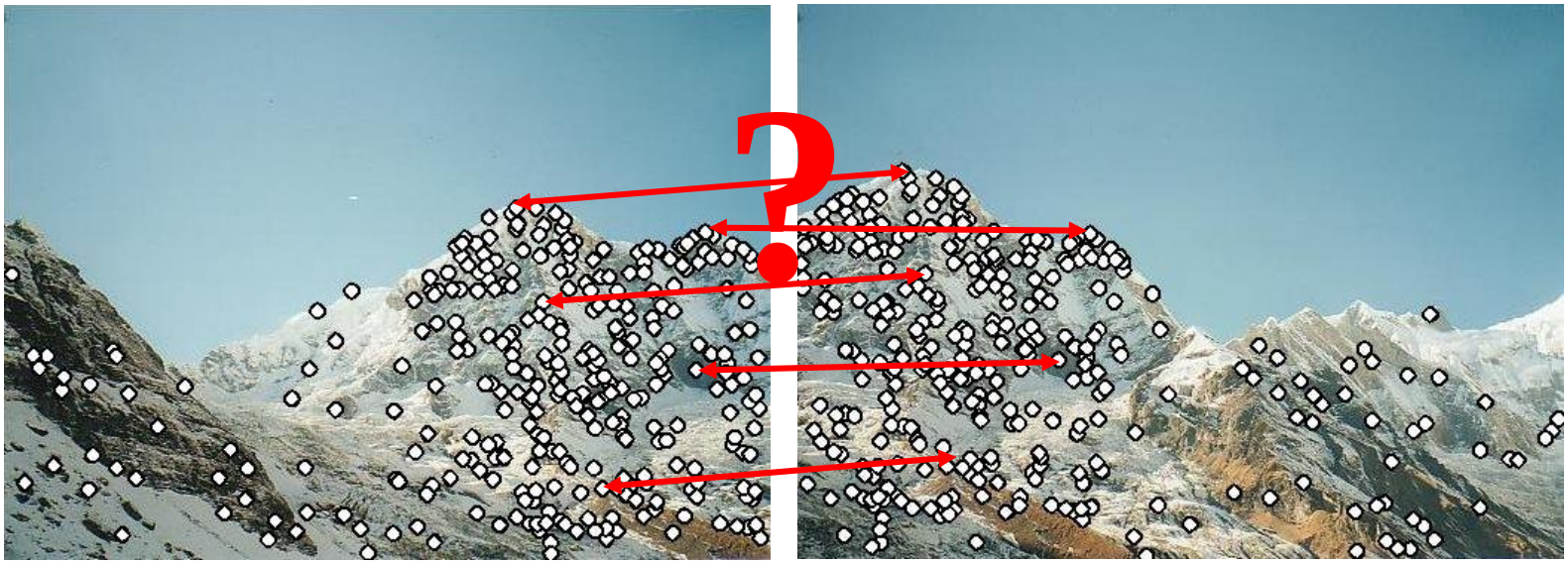
Automatic scale selection

Normalize: rescale to fixed size

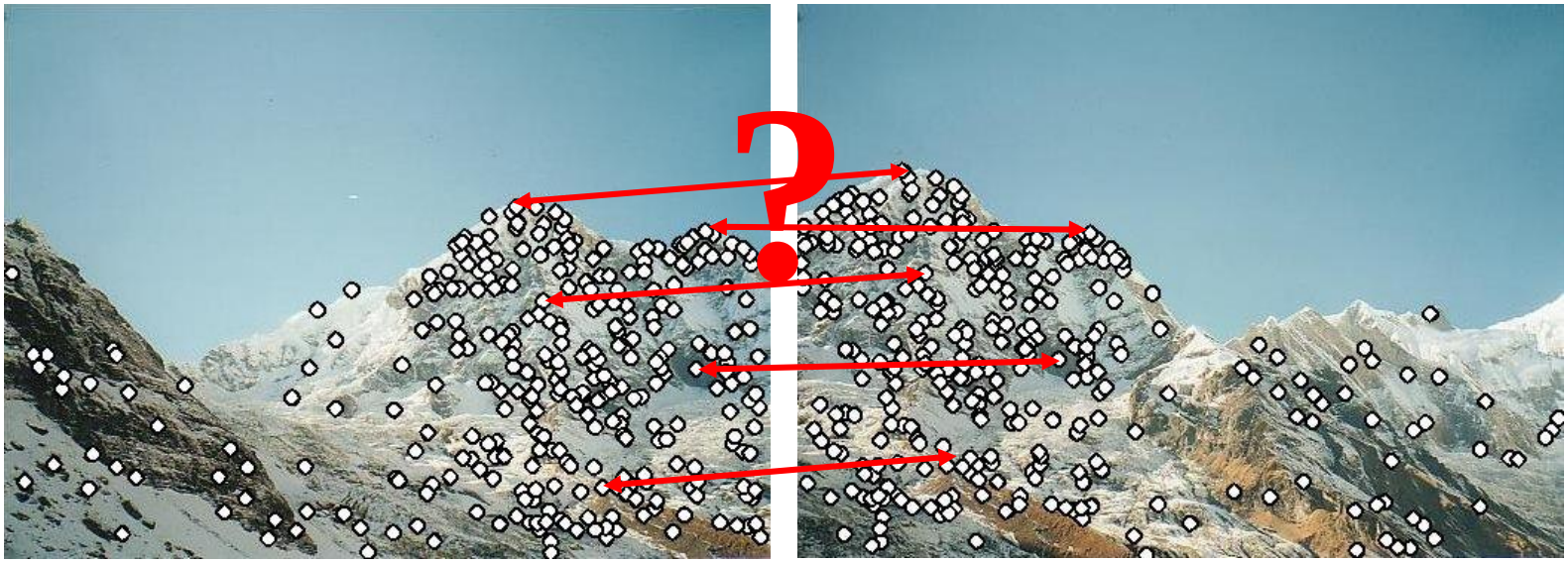


Feature Descriptors

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



Feature descriptors



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?

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 illumination/contrast changes

How to achieve Invariance

- Make sure your detector is invariant
  - Corner is invariant to translation and rotation
  - Scale is more complex
    - common approach is to detect features at many scales using a Gaussian pyramid
    - more sophisticated methods find “the best scale” to represent each feature (e.g., SIFT)

Rotation Invariance

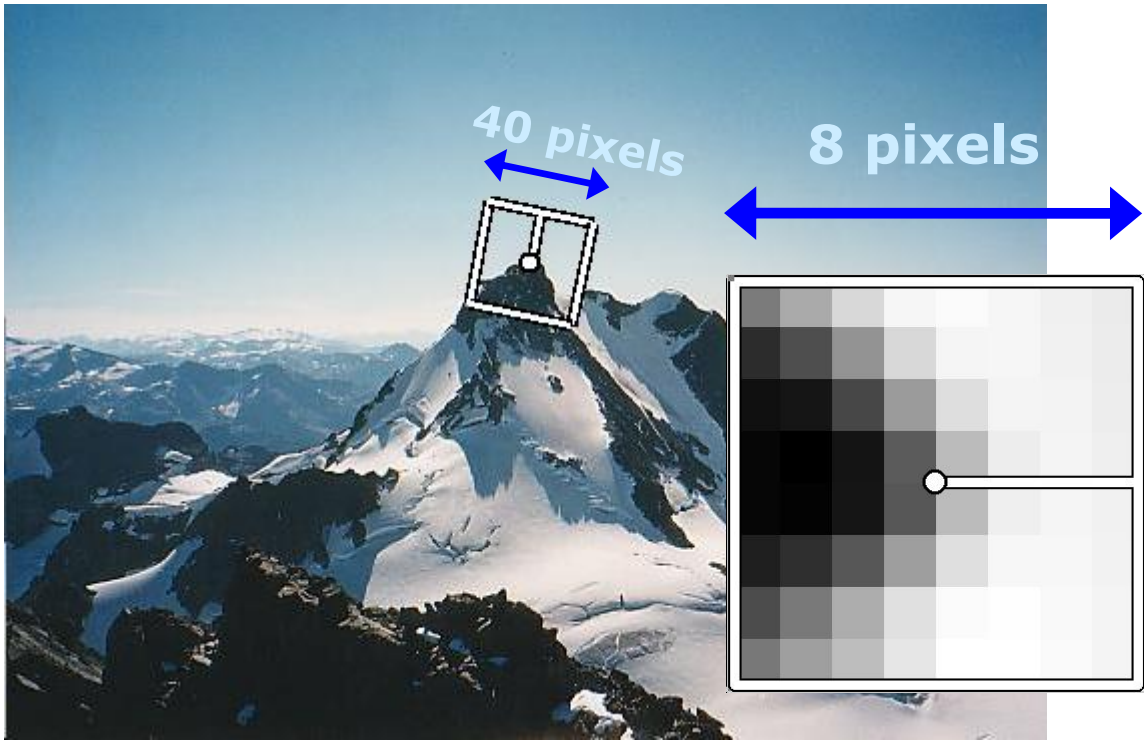
Find dominant orientation of the image patch

- How to estimate this angle?
  - use Hessian Matrix
  - A lot of other methods available
- Rotate the patch

Multi-scale Oriented Patches Descriptor

Take 40x40 square window around detected feature

- Scale to 1/5 size (using pre-filtering)
- 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

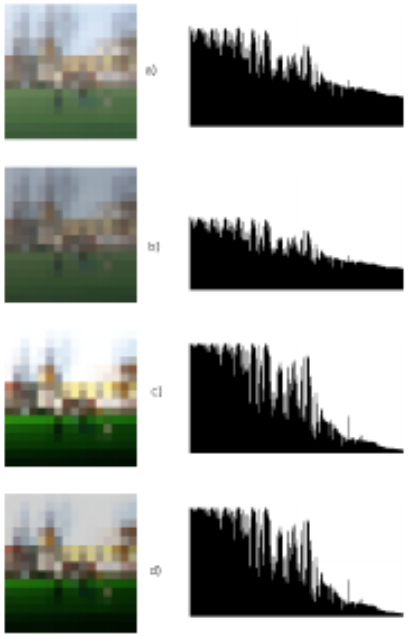


Matching Intensity Patches I

- Simplest way:
  - store n intensity values for a patch of n pixels.
  - use cross-correlation to find a match

$$c(P_1, P2) = \sum_{i}^{n} P_1[i]P_2[i]$$

- Matching Intensity Patches



a) Original patch

b) Brightness decreased
c = 0.26272

c) Contrast increased
c = 0.38041

d) Various changes
c = 0.29758

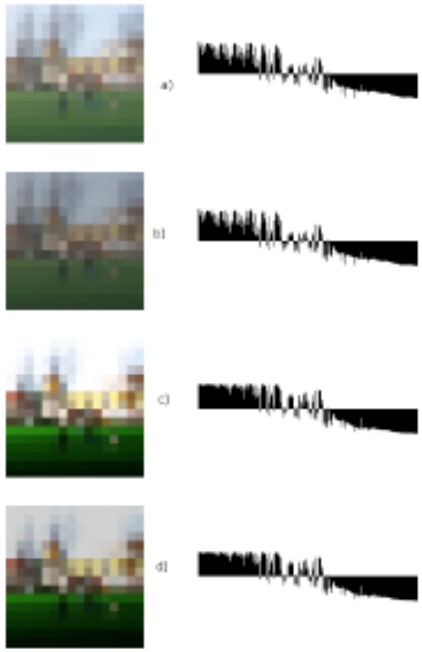
Matching Zero-Normalized Patches I

- Immunity to brightness change: zero mean

$$Z(x,y) = I(x,y) - \mu, \qquad \mu = \frac{1}{n}\sum_{x,y} I(x,y)$$

- Immunity to contrast change: unit variance

$$ZN(x,y) = \frac{Z(x,y)}{\sigma}, \qquad \sigma^2 = \frac{1}{n}\sum_{x,y} Z(x,y)^2$$



a) Original patch

b) Brightness decreased
c = 0.99999

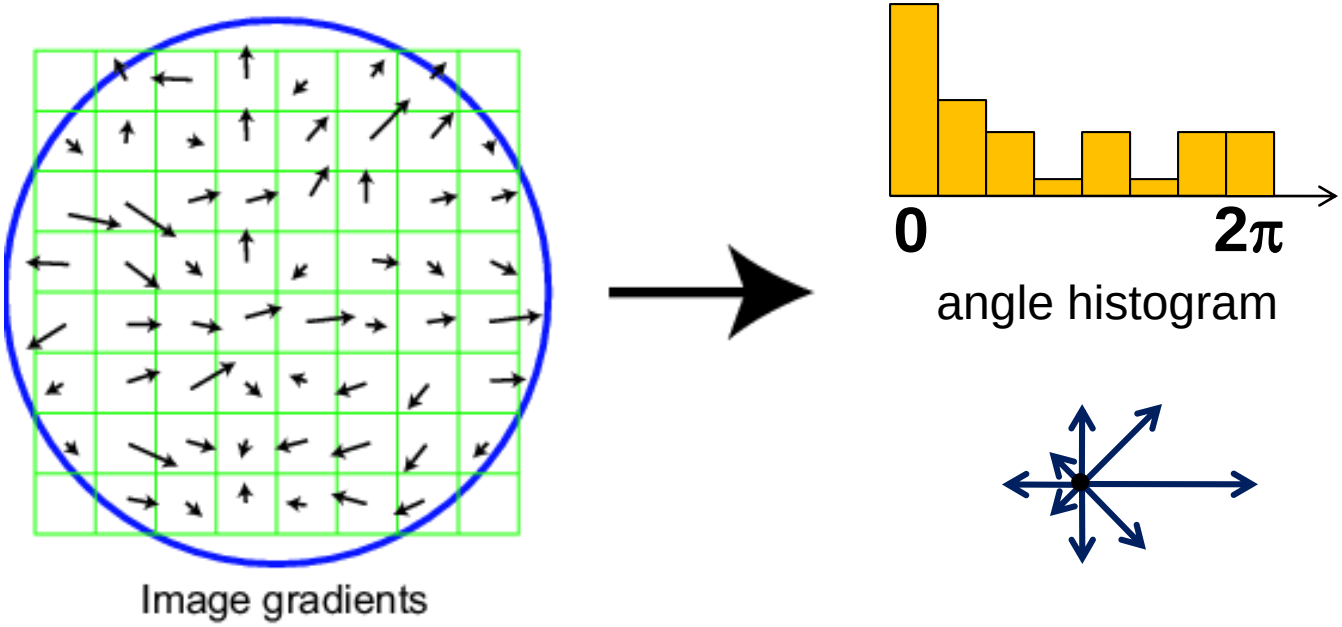
c) Contrast increased
c = 0.96987

d) Various changes
c = 0.98501

Scale Invariant Feature Transform (SIFT)

Basic idea:

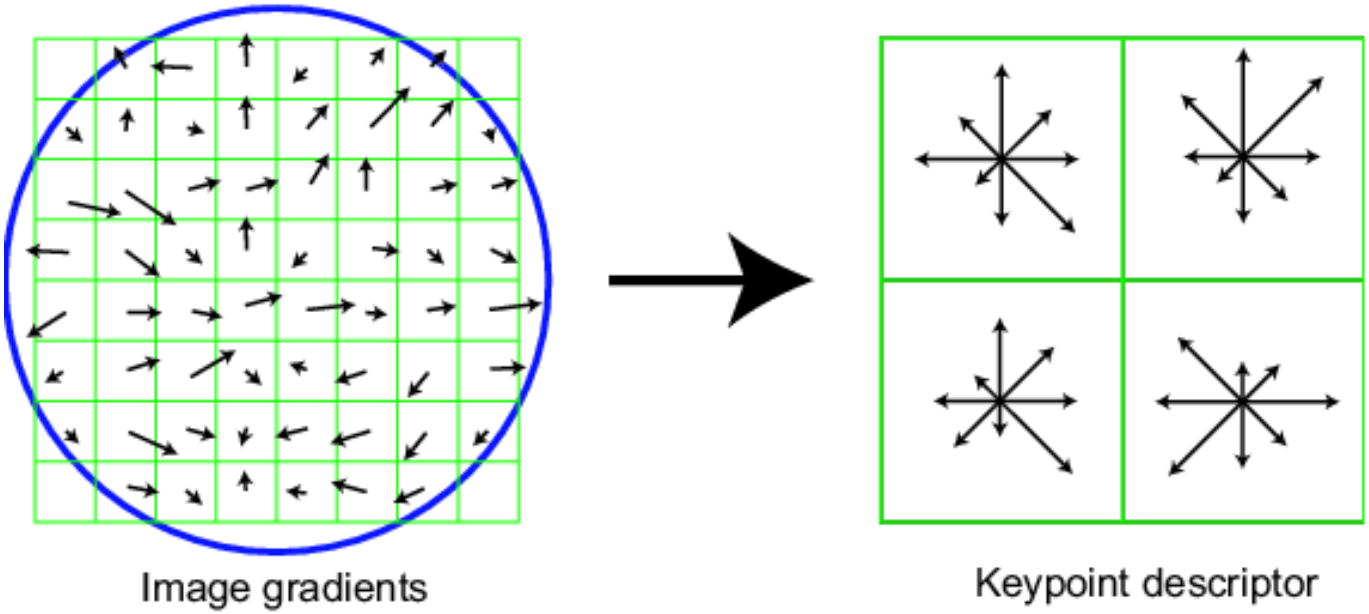
- Take 16x16 square window around detected feature
- Compute edge orientation for each pixel
- Delete small edges (threshold gradient magnitude)
- Create histogram of remaining edge orientations



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

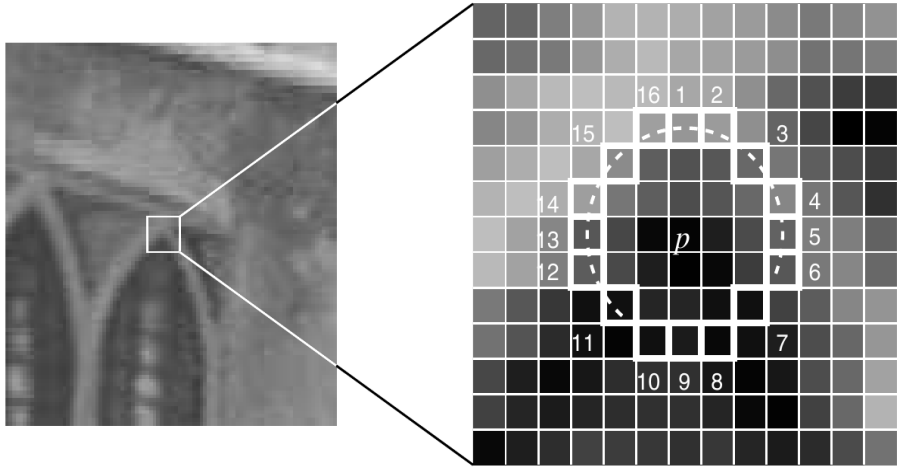


Properties of SIFT

- 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)
  - Lots of code available
    - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT
    - opencv

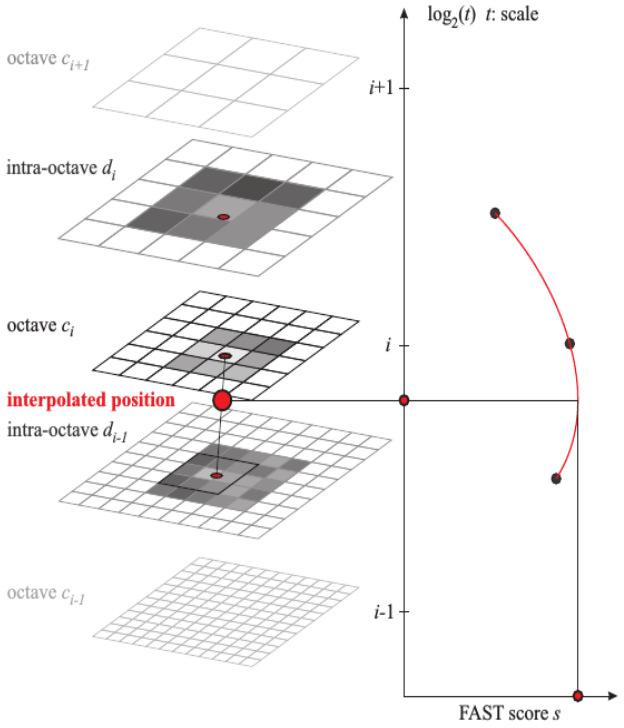
BRISK Key Points Detection

FAST keypoints detection



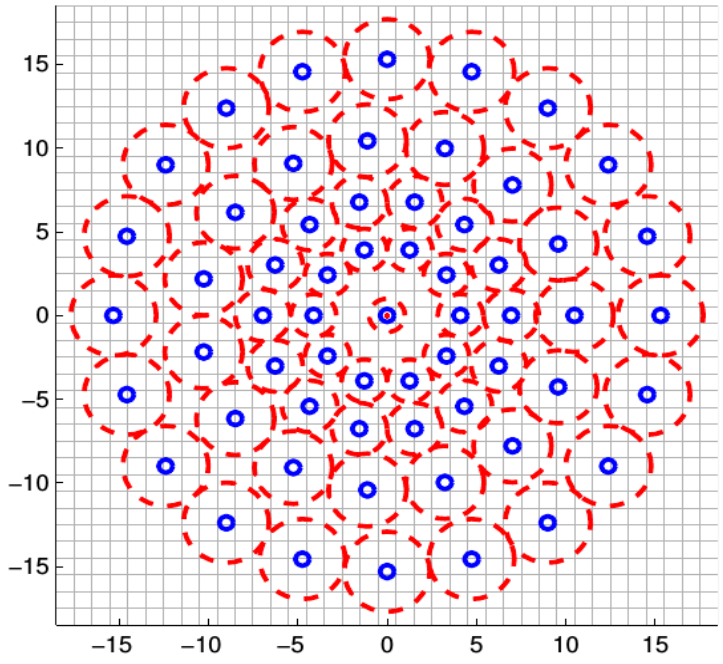
-Threshold are needed to control the number of keypoints produced.

BRISK keypoints detection



-Combined the advantages of SIFT and SURF detection methods.

BRISK Descriptor



The local gradient of keypoint pair (P_i , P_j) can be represented as g :

$$\boldsymbol{g}(P_i, P_j) = (P_j - P_i) \cdot \frac{I(P_j, \sigma_j) - I(P_i, \sigma_i)}{||P_j - P_i||^2}$$

$$\mathcal{S} = \{(P_i, P_j) \in \mathcal{A} | \; ||P_j - P_i|| < \delta_{max}\} \subseteq \mathcal{A}$$

$$\mathcal{L} = \{(P_i, P_j) \in \mathcal{A} | \; ||P_j - P_i|| > \delta_{min}\} \subseteq \mathcal{A}$$

$$\boldsymbol{g} = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \cdot \sum_{(P_i, P_j) \in \mathcal{L}} \boldsymbol{g}(P_i, P_j)$$

Based on g the orientation angle α can be determined.
Each bit of the binary BRISK descriptor can be determined as following:

$$b = \begin{cases} 1, & I(P_j^\alpha, \sigma_j) > I(P_i^\alpha, \sigma_i) \\ 0, & otherwise \end{cases} \forall (P_i^\alpha, P_j^\alpha) \in \mathcal{S}$$

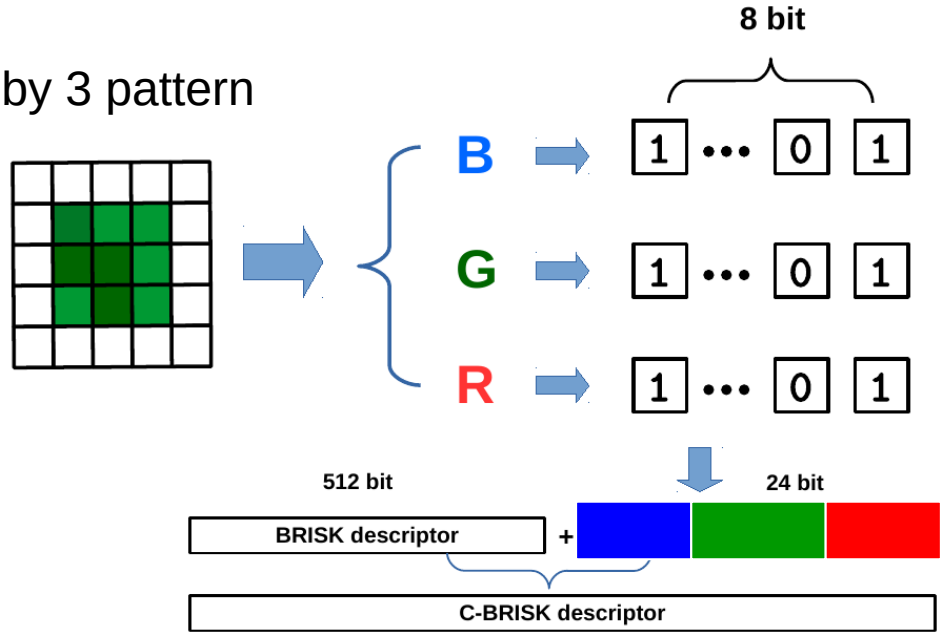
C-BRISK Descriptor

For color image:

$$m(R,G,B) = \frac{\sum_{i=1}^{n} V_{(R,G,B)}}{n}$$

m(R,G,B): Median value of R,G,B value

3 by 3 pattern



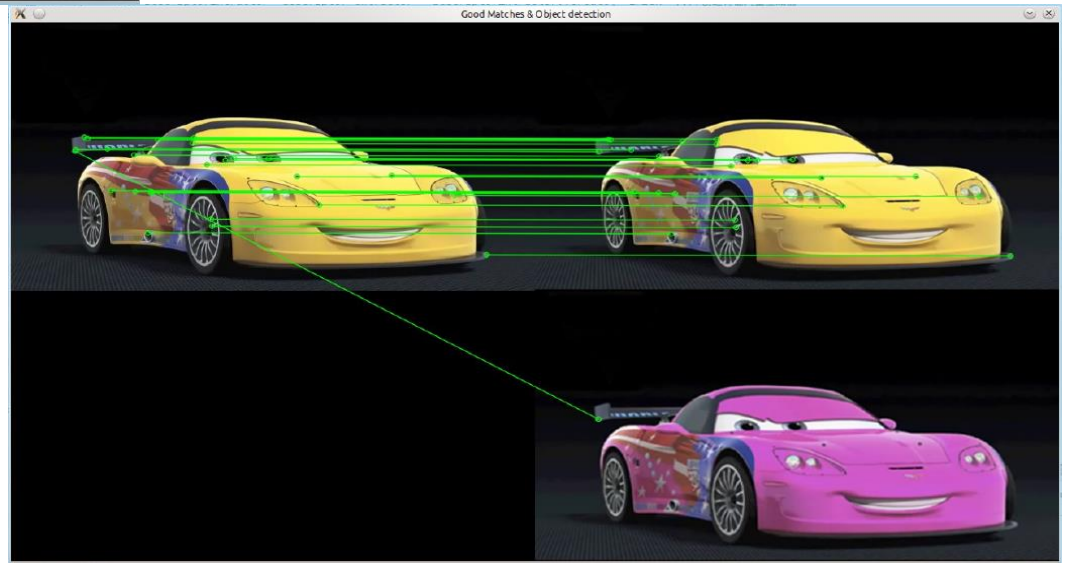
For gray image:

C-BRISK change the m(R,G,B) to the median value of gray values of the 3 by 3 pattern and keep the length of C-BRISK at 536 bits.

BRISK VS C-BRISK

BRISK

C-BRISK



Key Points Detection Test

SIFT

SURF

BRISK

Feature Matching

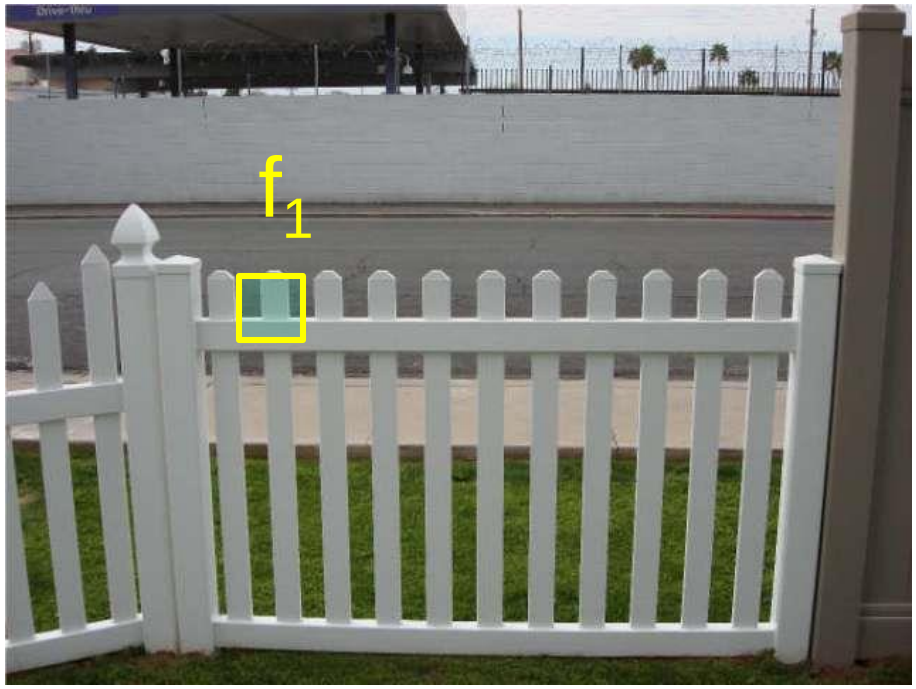
Given a feature in I_1, how to find the best match in I_2?

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

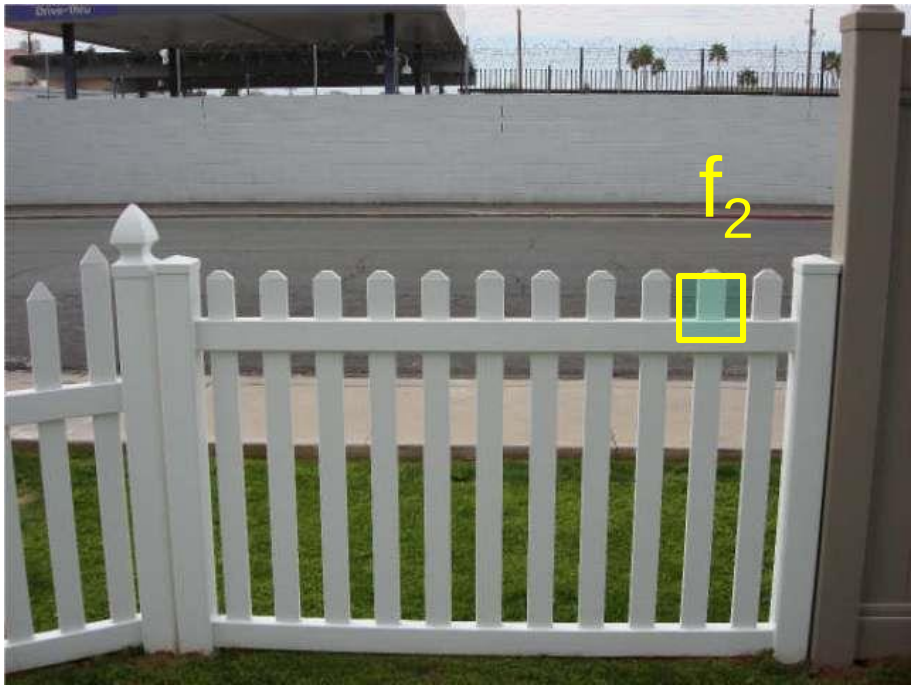
Feature Distance

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 deliver good scores to very ambiguous (bad) matches



I_1

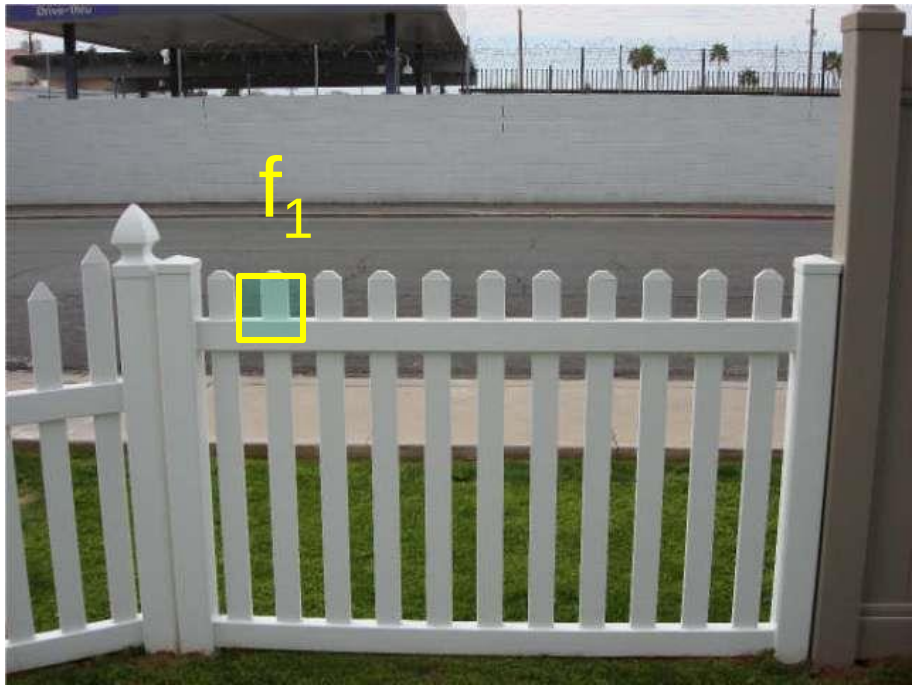


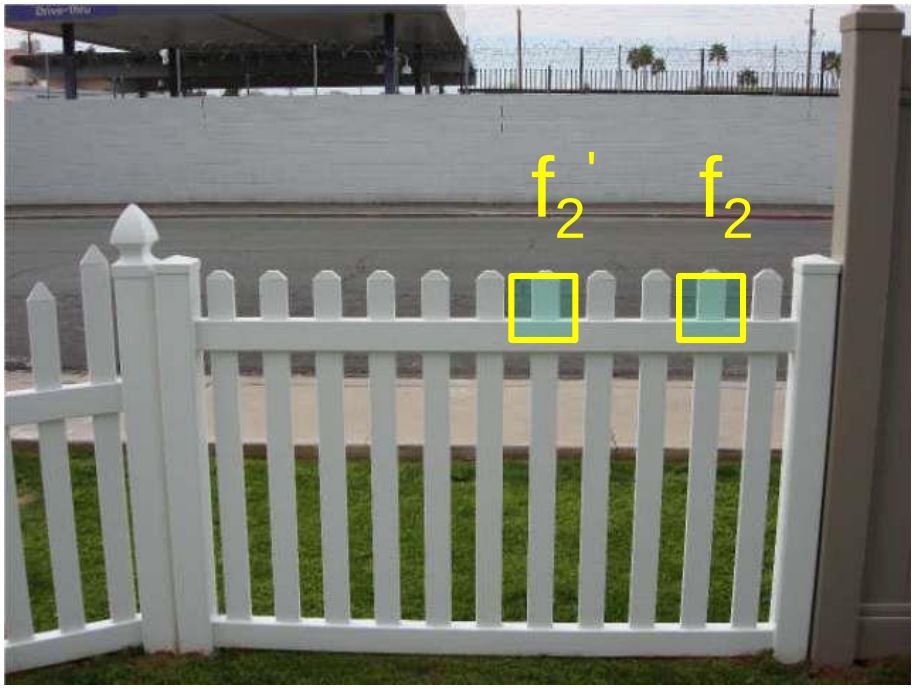
I_2

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 small values for ambiguous matches

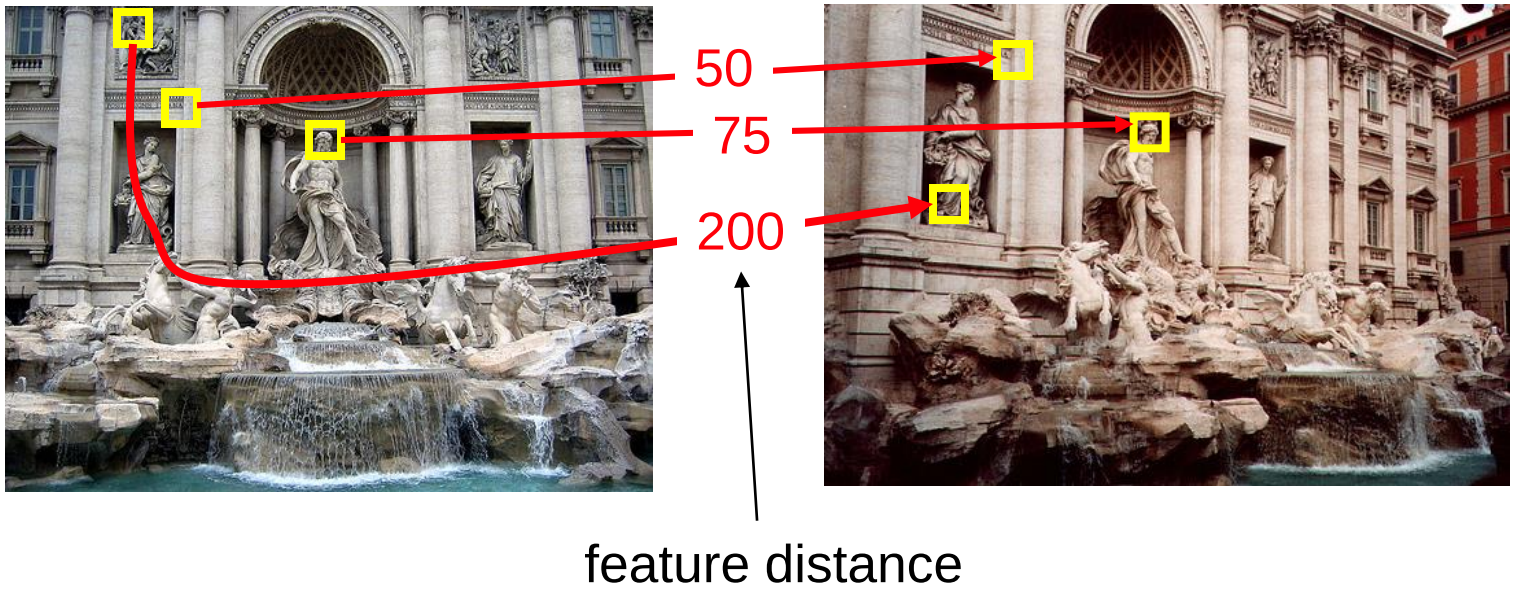




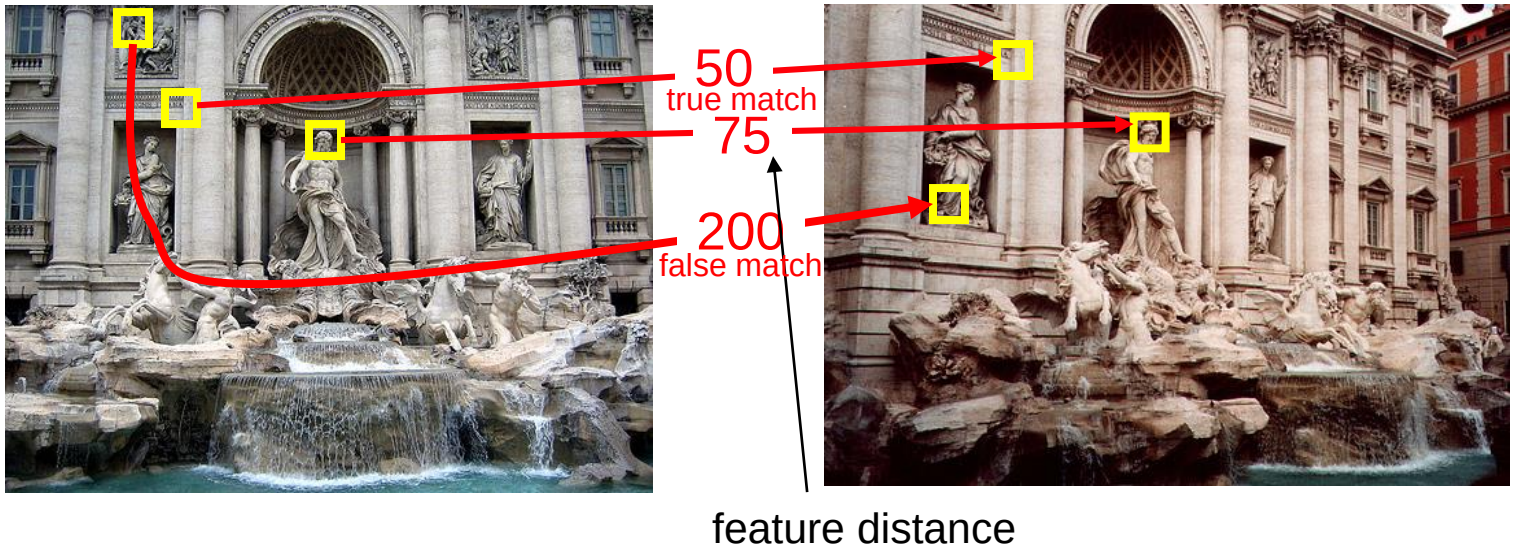
I_1 I_2

Evaluating the Results

How can we measure the performance of a feature matcher?



True/false Positives



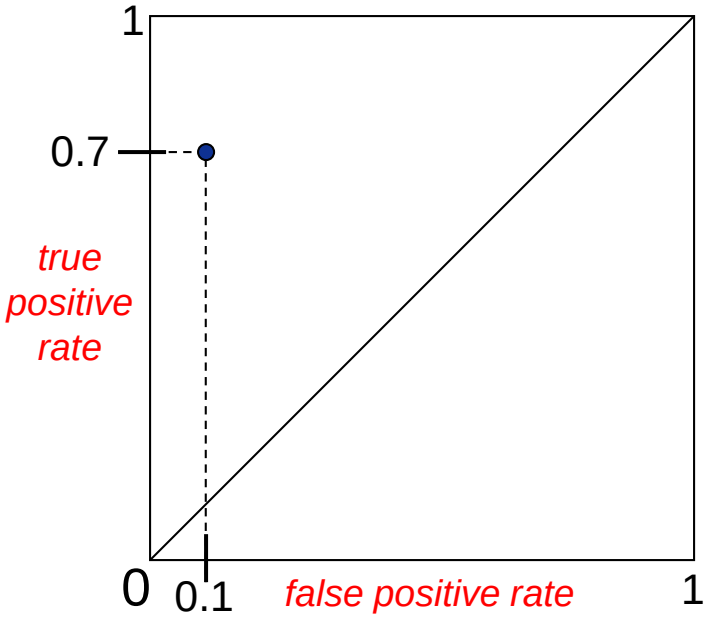
The distance threshold affects performance

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

Evaluating the Results

How can we measure the performance of a feature matcher?

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

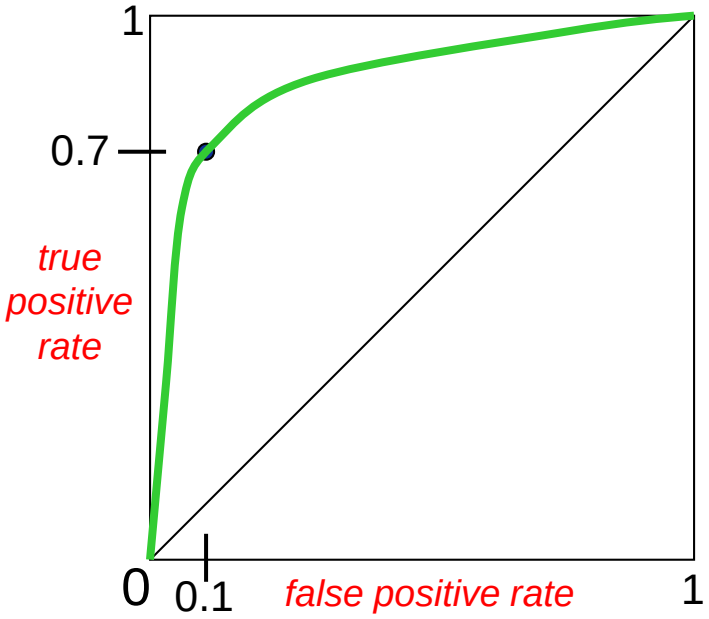


$$\frac{\text{\# false positives}}{\text{\# unmatched features (negatives)}}$$

Evaluating the Results

How can we measure the performance of a feature matcher?

ROC curve (“Receiver Operator Characteristic”)



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

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

ROC Curves

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

Object Recognition (David Lowe)

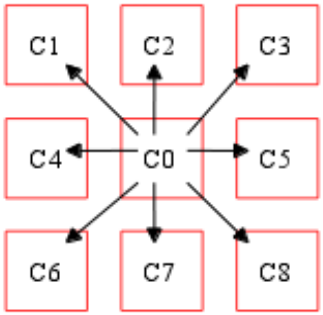
BRISK based Vehicle Tracking

Object Detection

Feature Representation

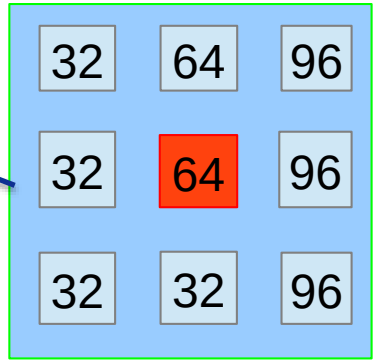
Discriminative Classifier

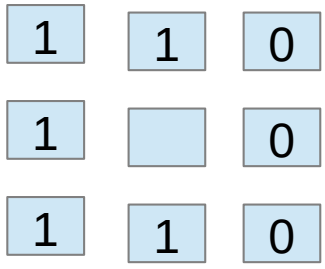
LBP descriptor 1

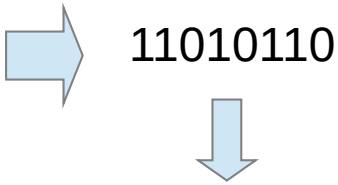


1. Edge detection
2. Pixel value comparison with 8 neighbors
3. $(c_1c_2c_3c_4c_5c_6c_7c_8)$ combine these eight values (from 0 to 255)
4. 11010110 = 214

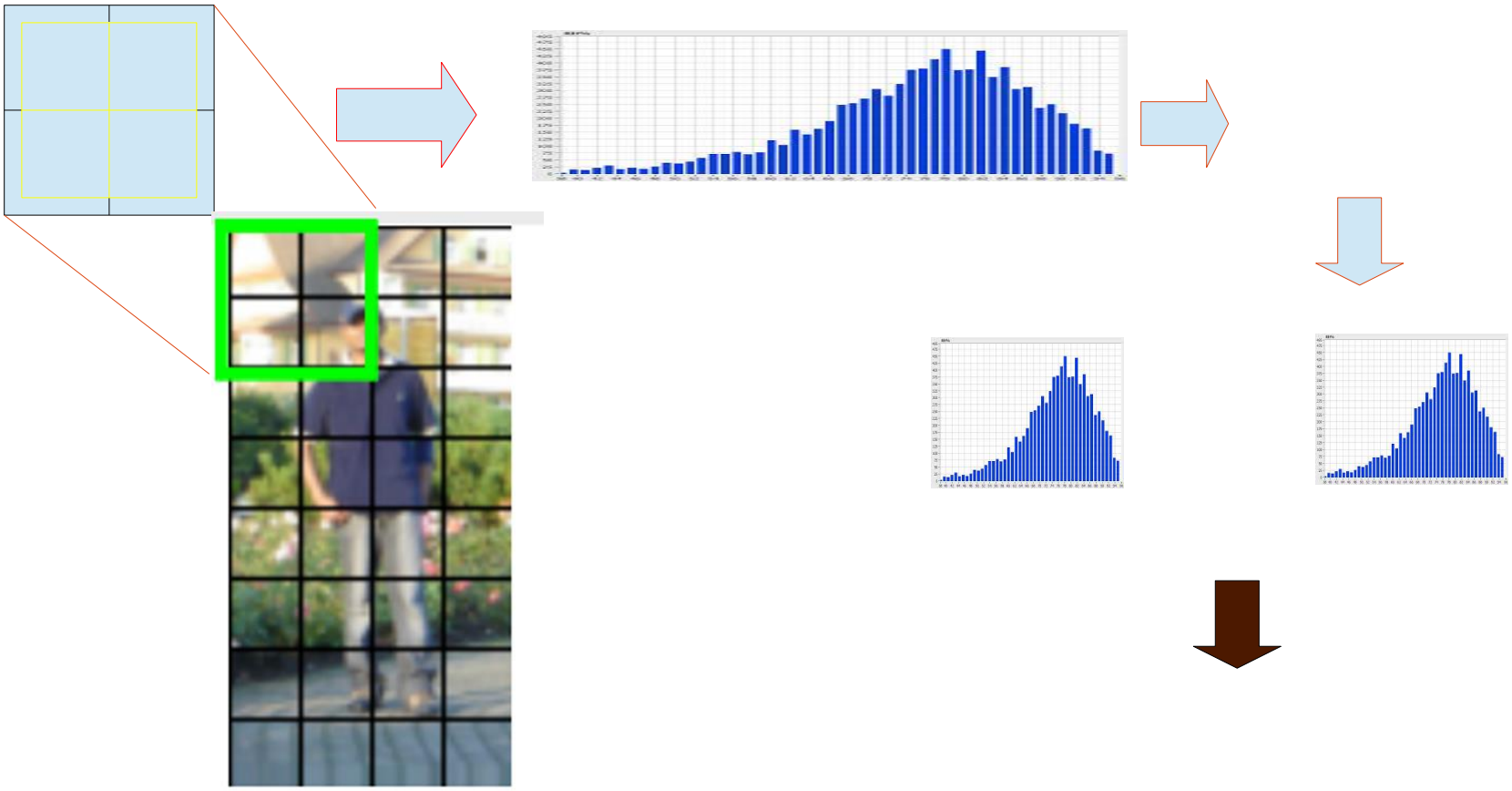
Sobel Edge







LBP Descriptor 2



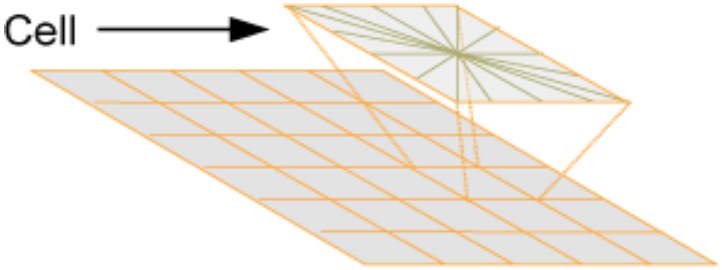
Patch size 108 by 36 divided into 9x4 small grids dimension is 24x256=6144

HOG Descriptor 1

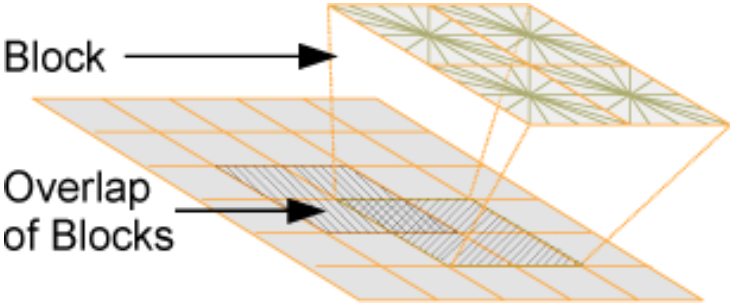
- Compute gradients on an image region of 64x128 pixels

- Compute histograms on “cells” of typically 8x8 pixels (i.e. 8x16 cells)

- Normalize histograms within overlapping blocks of cells (typically 2x2 cells, i.e. 7x15 blocks)



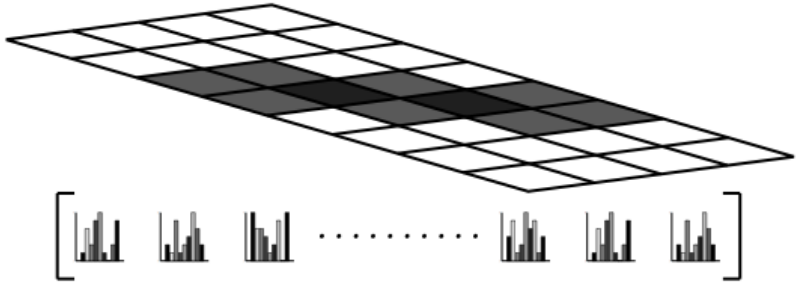
- Concatenate histograms

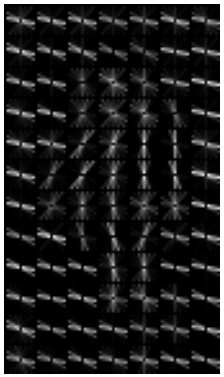


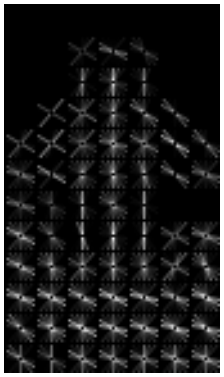
HOG Descriptor 2

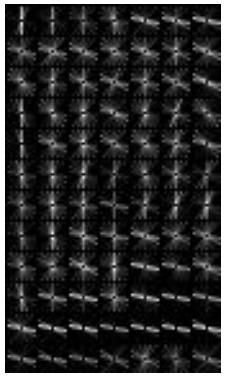
- Concatenation of Blocks

- Visualization:



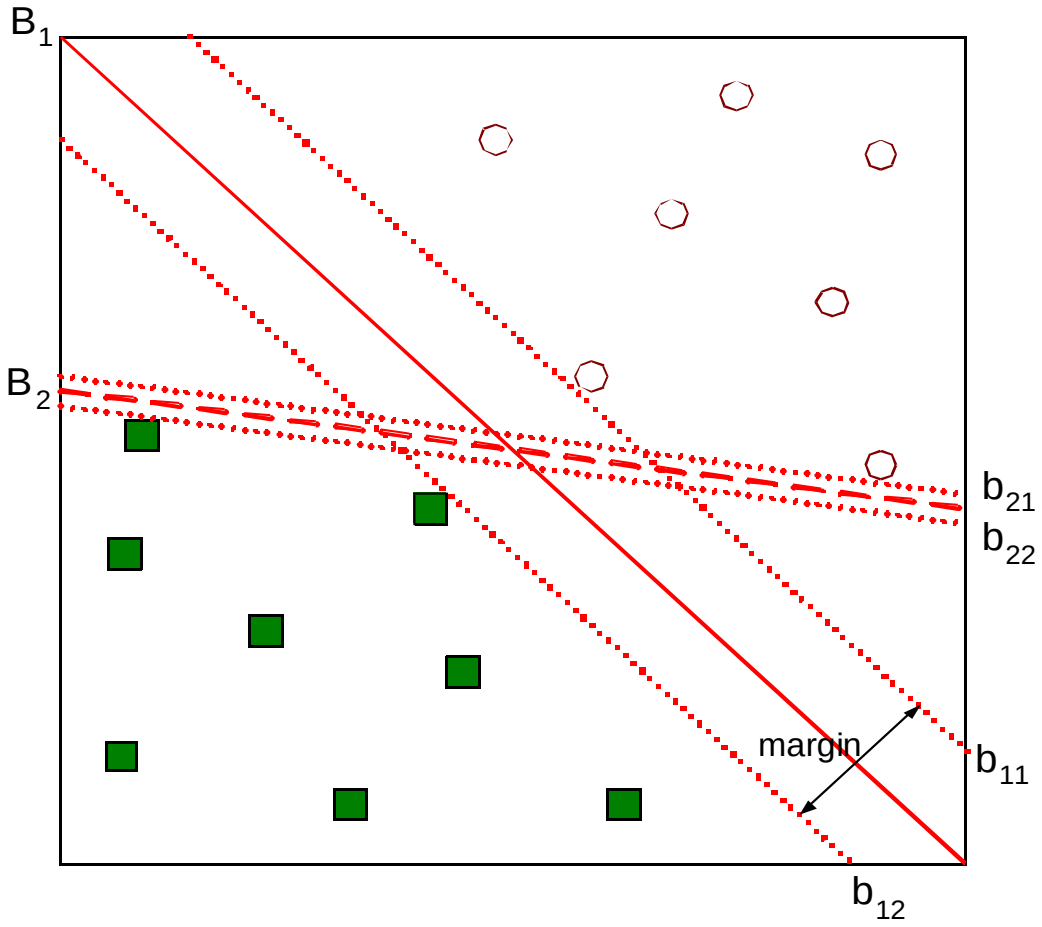




Training with SVM

1. SVM training is to find a hyper plane which can maximizes the margin.

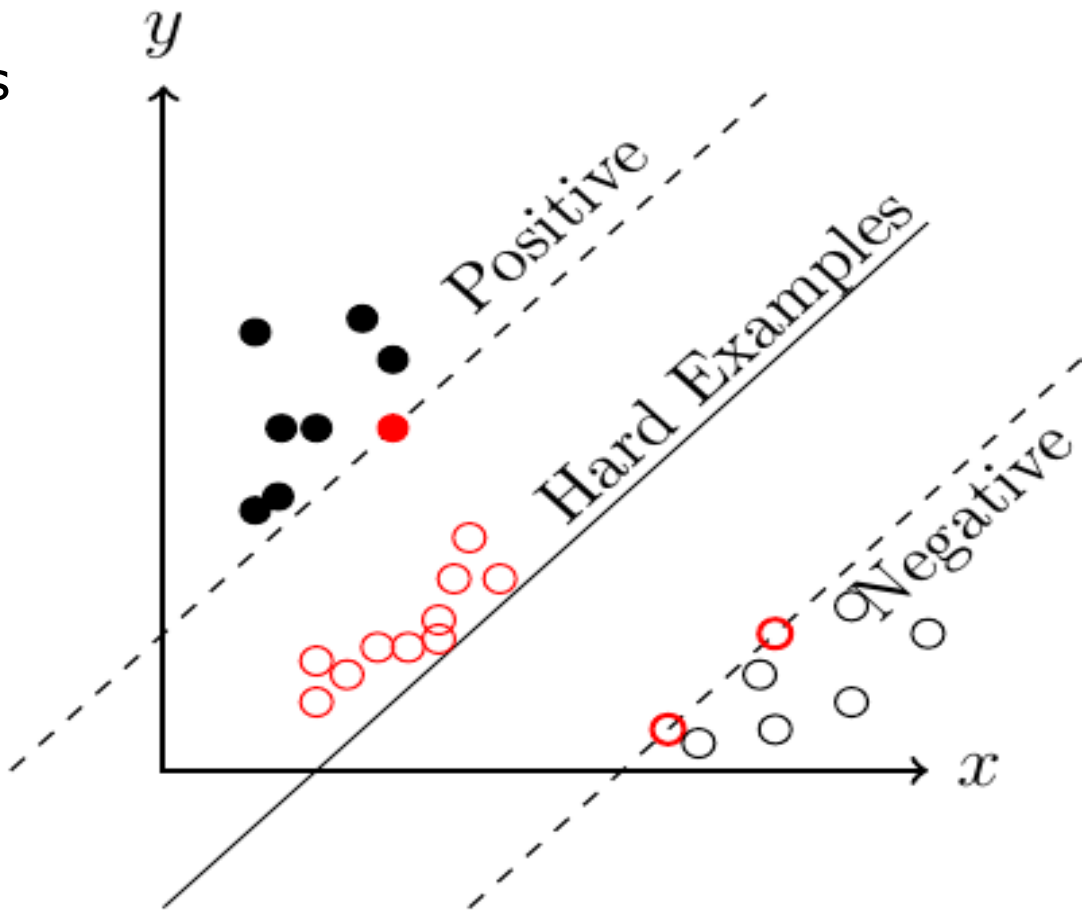
2. B_1 is better than B_2



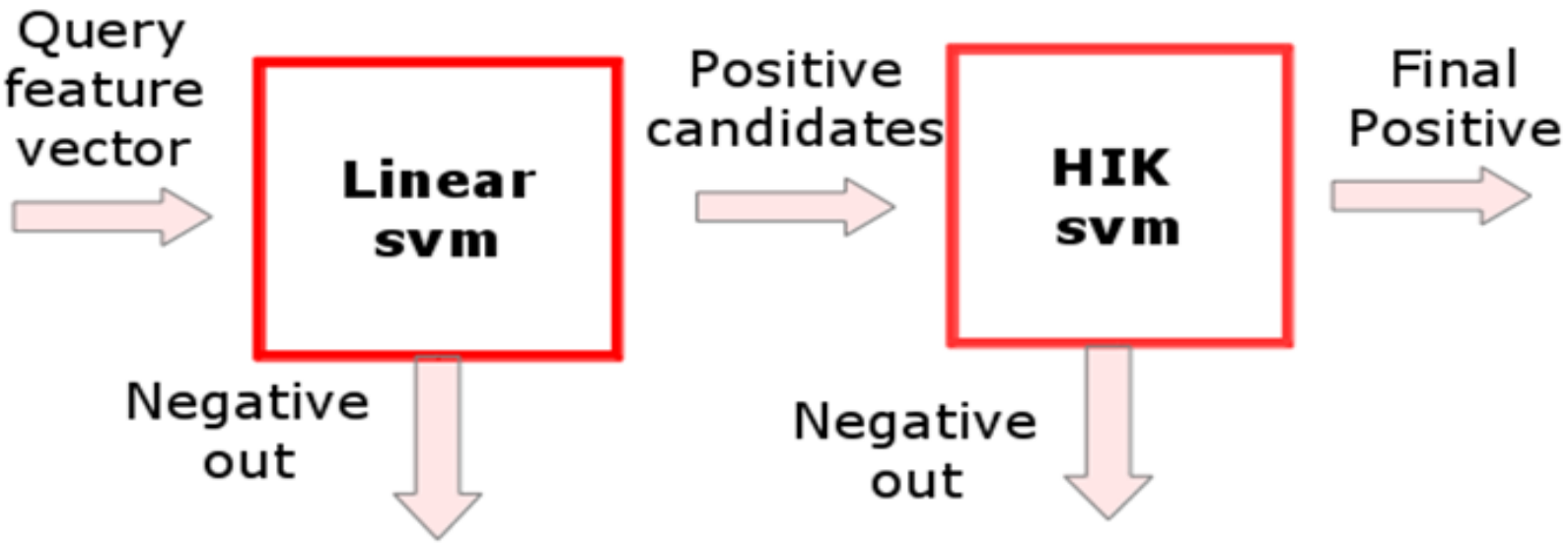
Training with SVM

1. Real problems (object detection) is not always SVM solvable. Especially for non-rigid object like human beings.

2. There are always unpredictable samples which are located in critical region. (**Hard Examples**)



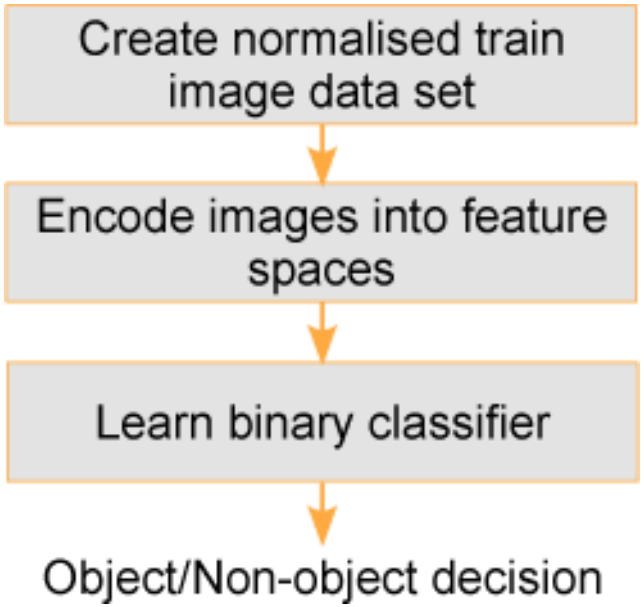
Cascaded Classifier



1. Linear svm is fast, but not accurate enough.
2. Histogram intersection kernel SVM is slow, but much more accurate.
3. Combination of these two classifier is our compromised solution.

Detector: Learning Phase

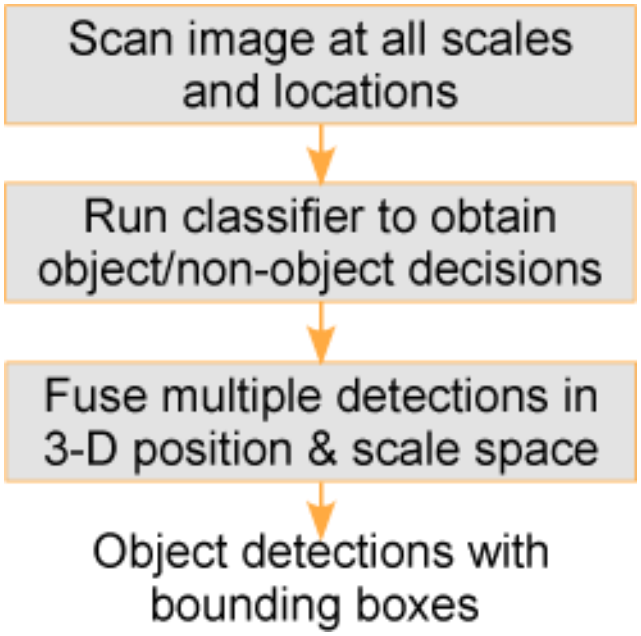
1. Learning

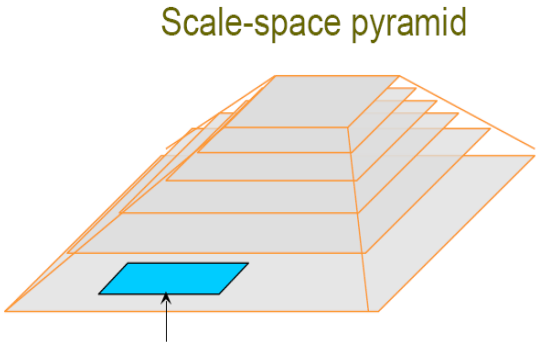


- Set of cropped images containing pedestrians in normal environment
- Global descriptor rather than local features
- Using linear SVM

Detector: Detection Phase

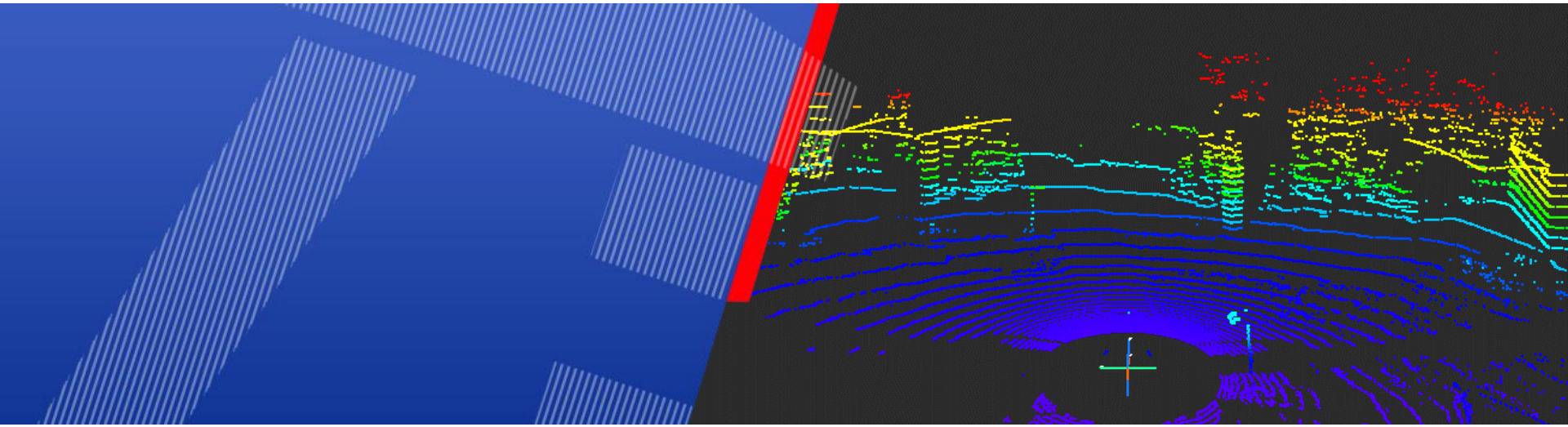
2. Detection





- Sliding window over each scale
- Simple SVM prediction

Point Cloud Processing



Point Clouds

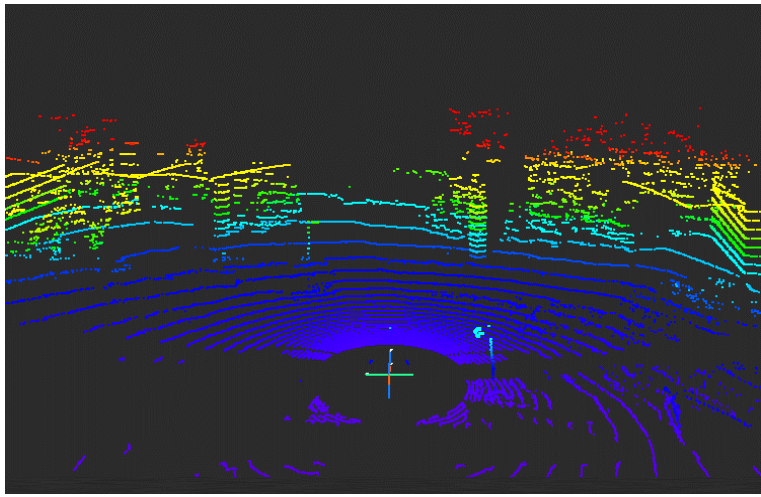
- A point cloud P is usually defined as a set of Cartesian points $P = \{p_1, p_2, \dots, p_n\} \mid p_i = (x_i, y_i, z_i)^T$
- The order of the points within the data structure representing the cloud is usually strongly dependent on the sensor which was used to gather the data
  - For devices with imaging sensors (stereo cameras, time-of-flight cameras) it is often line or column like
  - For 2D laser range finders the points are sorted according the angle of the polar representation
  - For 3D laser scanners its often related to the spherical coordinates
- To find the neighboring points of an arbitrary point p_i the set has to represented in an appropriate data structure like a k-d tree or an octree

Point Cloud Generation

- Different working principles
  - Active (sensor is emitting light, typically in the infrared range) and passive (sensor uses existing ambient light)
- Accuracies
  - Laser range finders > time of flight cameras > stereo cameras

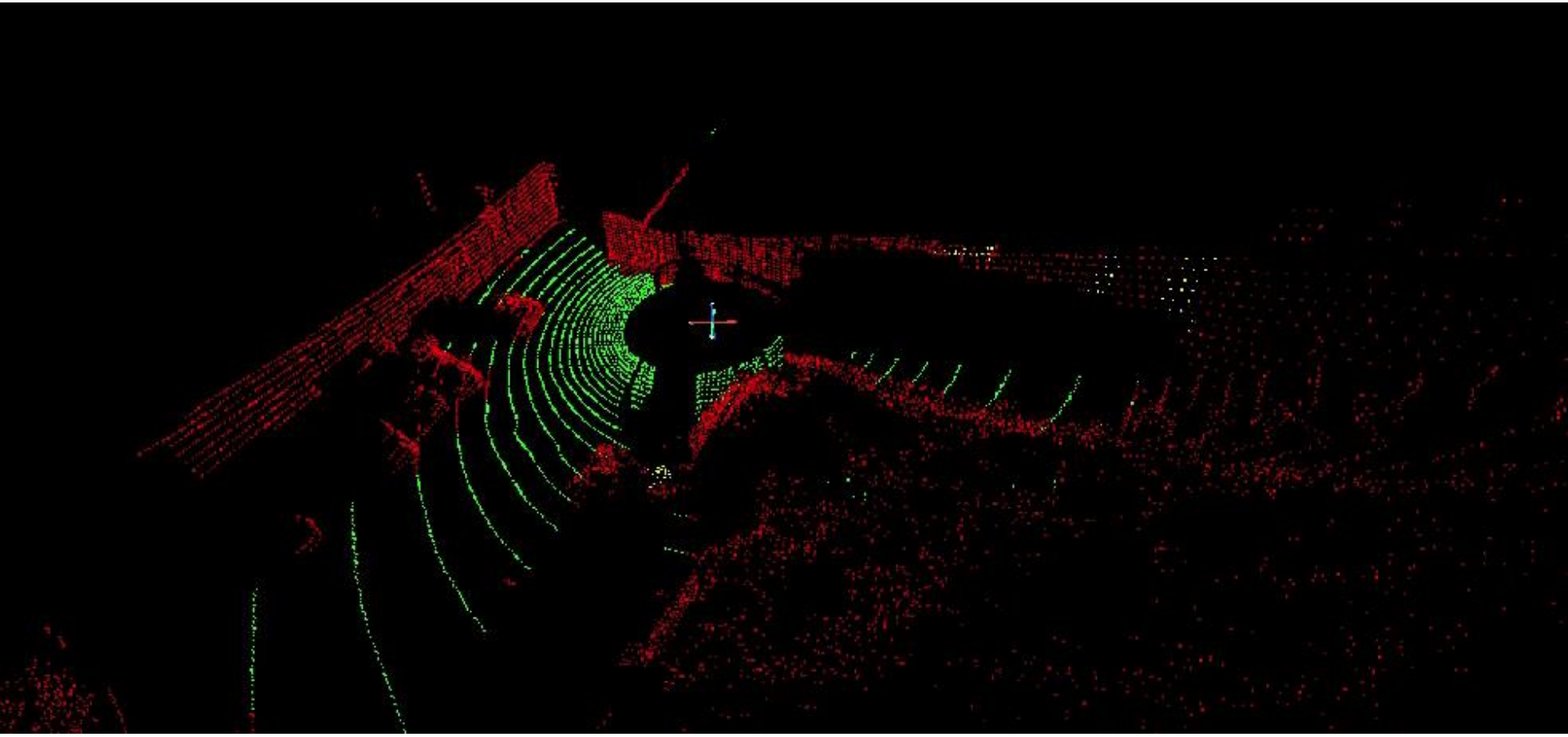
3D Laser Range Finders

- Example: Velodyne HDL-32E
  - ~70000 distance measurements per frame
  - Field of view (H x V): 360° x 40°
  - Update rate 10Hz
  - Ethernet connection, UDP
  - Output: 3D polar distance data



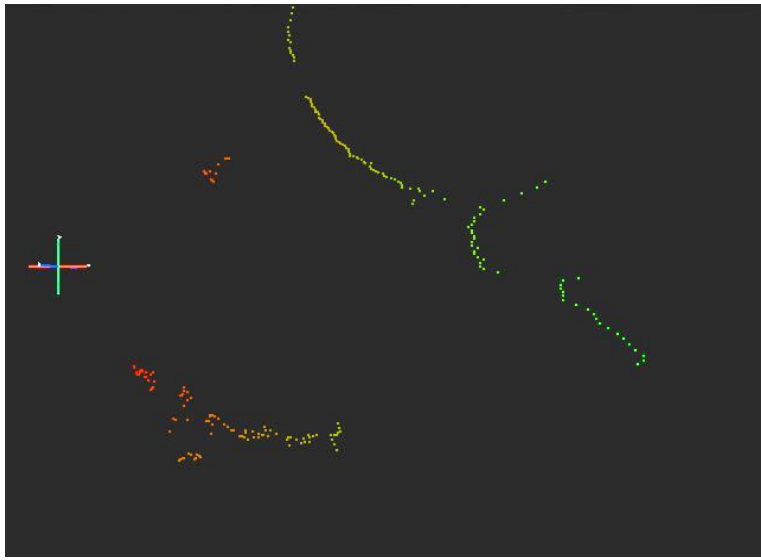
RRLAB
THE ROBOTICS RESEARCH LAB

3D Laser Range Finder



2D Laser Range Finders

- Example: SICK LMS151
  - 541 distance measurements per frame
  - Field of view (H): 270°
  - Angular resolution: 0.5°
  - Update rate 50Hz
  - Ethernet connection, TCP
  - Output: 2D polar data



2D Laser Range Finder

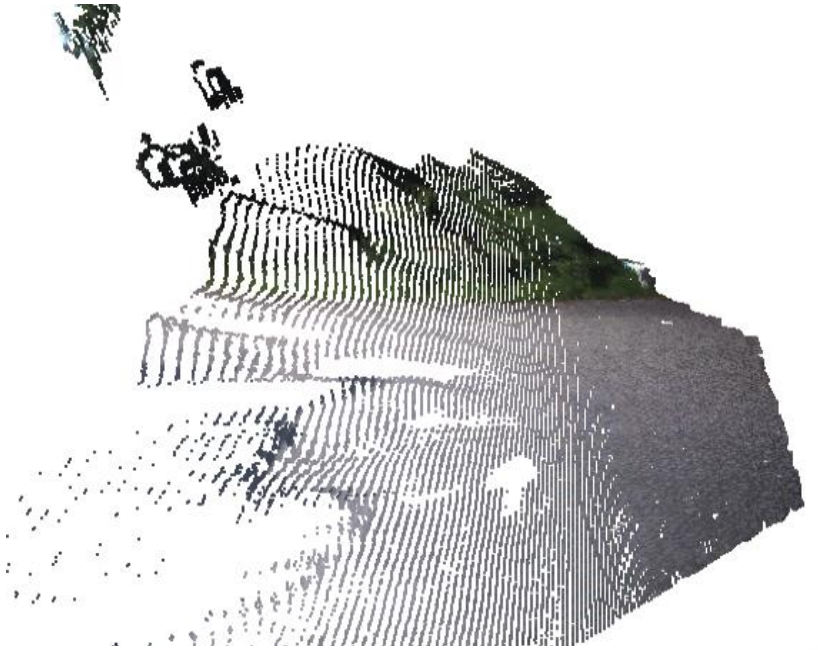
Stereo Cameras

- Example: Point Grey Bumblebee2
  - Resolution 1024 x 768px
  - Field of view (H x V): 97° x 73°
  - Frame rate 20fps
  - Baseline 12cm
  - FireWire connection
  - Output: 2 images (interlaced, Bayer decoding required)
  - **Post-processing: undistortion, stereo matching, triangulation**

Stereo Camera – Example Output

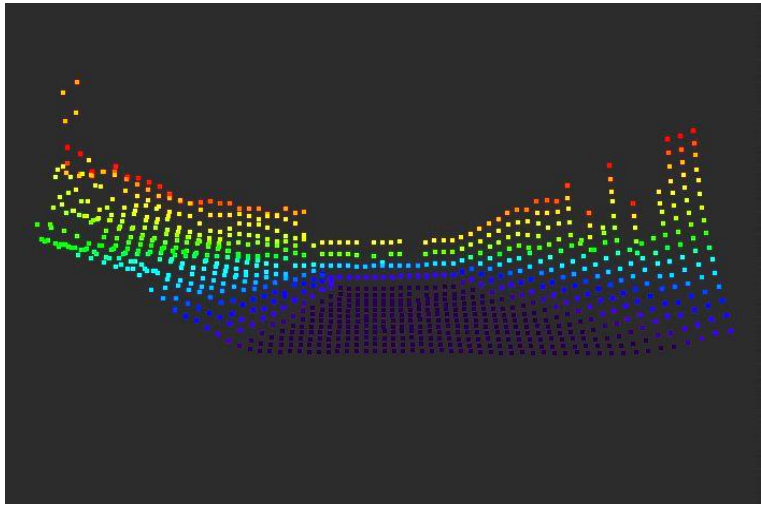
Stereo Camera – Precision

- Stereo point cloud has plane like appearance (based on the disparity computation/levels)
- Measurement precision depends on baseline, focal length, resolution of the cameras and the matching algorithm
- Graph shows precision for a camera with a baseline of 15cm, 640x480 pixel resolution, SGBM stereo matching

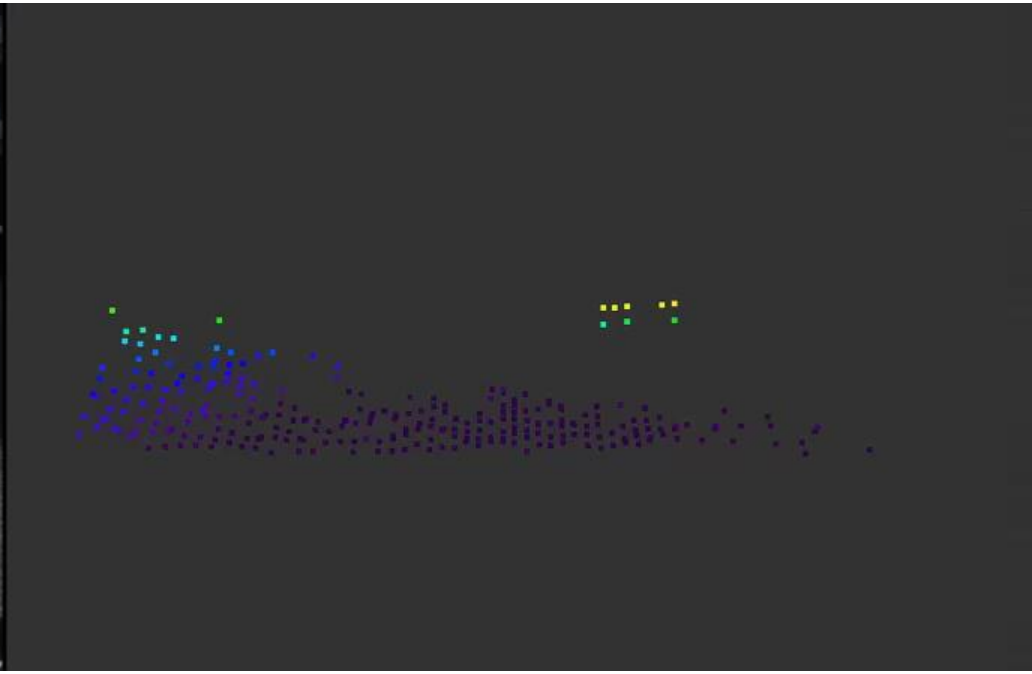


Time-of-Flight (ToF)-Cameras

- Example: ifm electronic O3M151
  - 64x16 distance measurements per frame
  - Field of view (H x V): 70° x 23°
  - Frame rate 25fps
  - Ethernet connection, UDP
  - Output: 3D Cartesian distance data

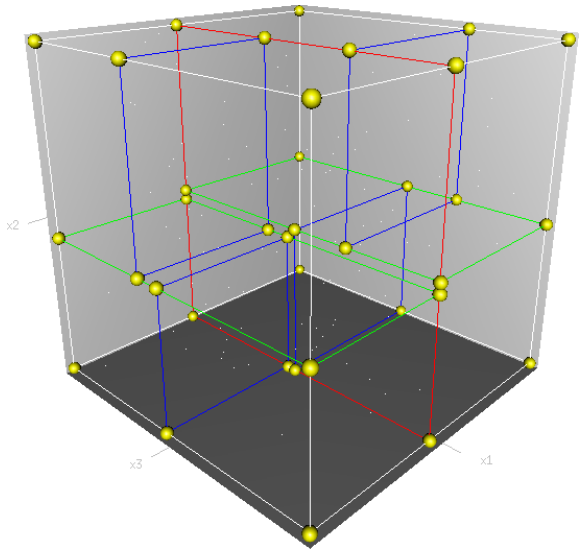


ToF-Camera – Example Output



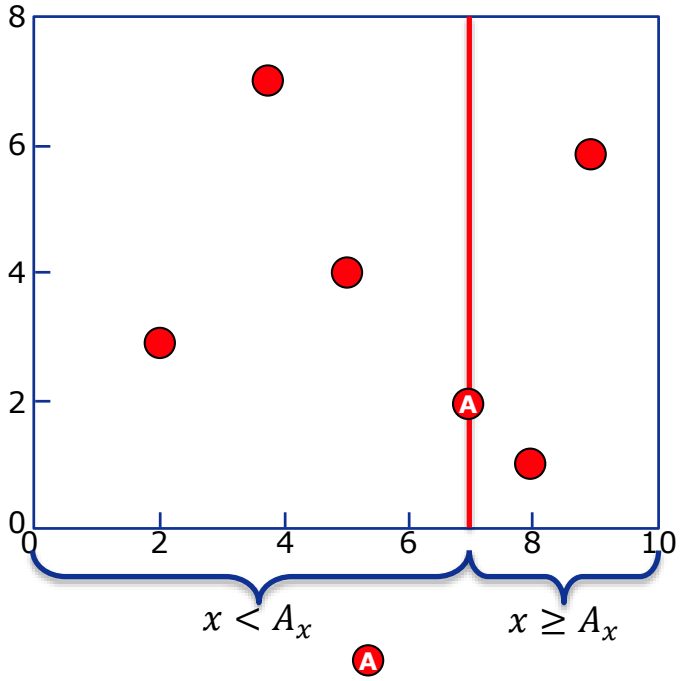
Sorting a Point Cloud – k-d Tree

- Space-partitioning data structure
  - For nearest neighbor search, range searches
  - In a k-d tree, each level splits the children along a specific dimension (axis)
    - Therefore, a hyperplane perpendicular to the corresponding axis is used



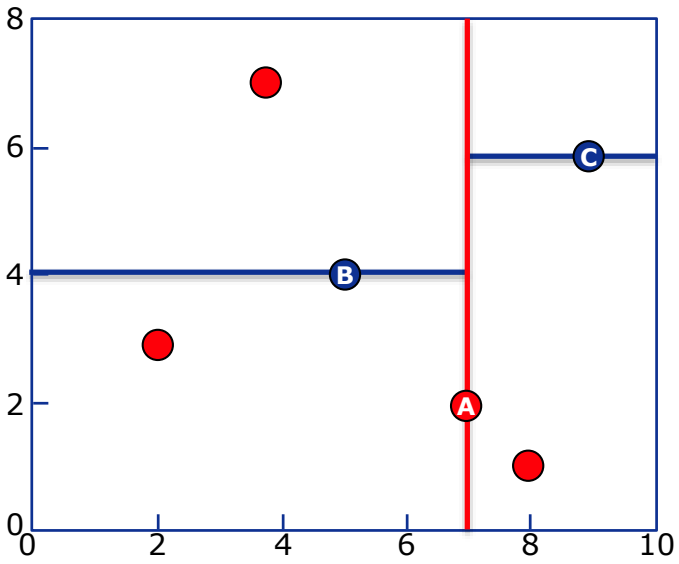
K-d Tree 2D Construction Example

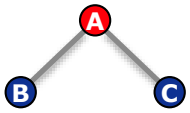
1. Start with x-axis, construct a hyperplane (in this case a line) perpendicular to the axis, at the x-value greater then the median
2. Sort all points with smaller x-values to the left branch of the tree, all points with larger or equal values to the right



K-d Tree 2D Construction Example

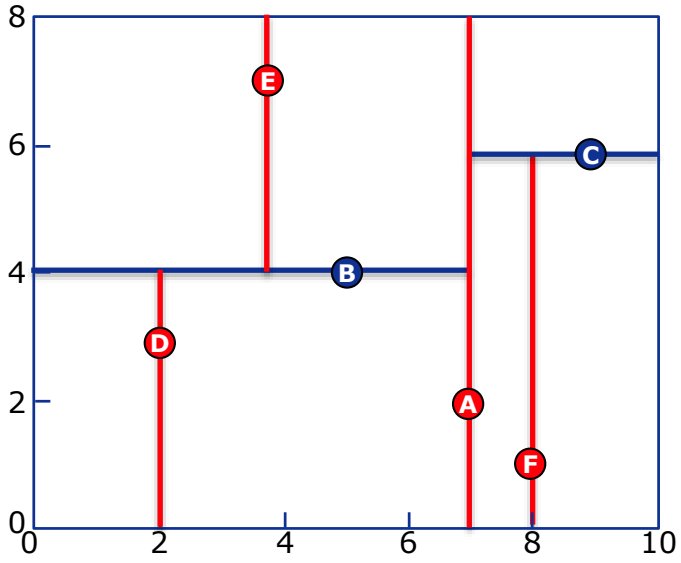
3. In each half-space (left and right): construct a hyperplane perpendicular to the y-axis, greather then the median of all y-values
4. In the tree, place the smaller values to the left, the larger or equal ones to the right

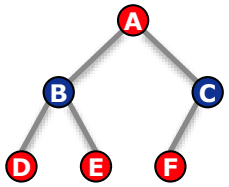




K-d Tree 2D Construction Example

5. Continue with the x-axis (in the 3D space z-axis), construct a hyperplane perpedicular to the axis in each half-space of B and C (top and bottom)
6. Add the points as leaves and terminate as no points are remaining



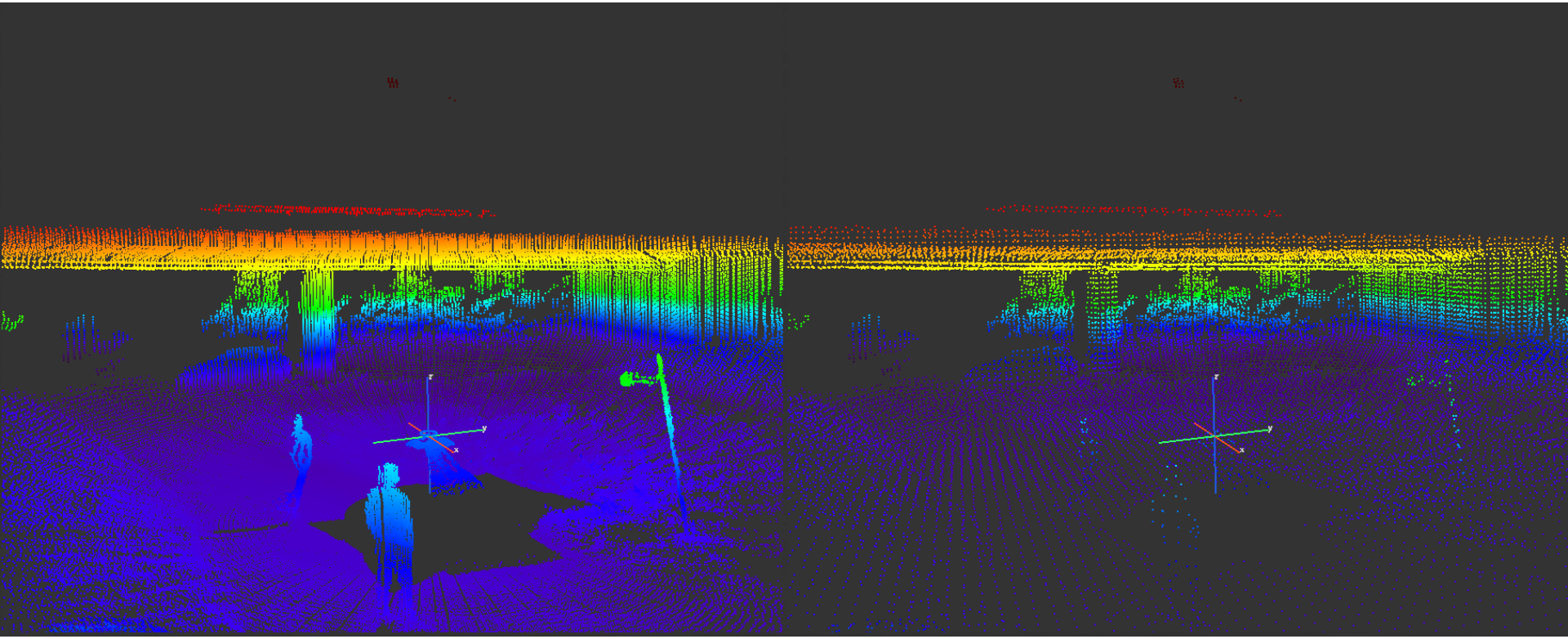


Downsampling: Voxel Grid Filter

- If the density of the point cloud exceeds the computational power, the cloud can be downsampled (reducing the number of points) to speed up further steps
- Approach
  - Subdivide the space into small boxes of parametrized sizes - this is called a 3D voxel grid
  - Sort the points into the voxels
  - If the voxel is not empty:
    - Var1: Use the center of the box as a new point
    - Var2: Use the centroid of all points within the voxel as a new point (more accurate, slower)

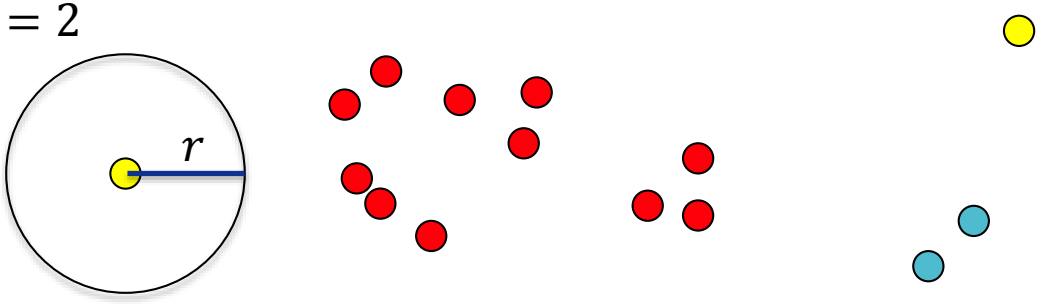
Example

- Parameters and Results
  - Voxel size: 20cm × 20cm × 20cm



Outlier Removal

- Outliers, e.g. cause by measurement errors or varying densities of the point cloud can lead to erroneous interpretations when analyzing (e.g. ground plane estimation, extracting surface normals)
- Approach 1: Neighborhood based
  - For each point: search for all neighbors within a certain radius r (Euclidian distance)
  - Mark point as outlier if number of neighbors is below a given threshold k
    - Example: yellow: outlier for $k = 1$, blue+yellow outlier for $k = 2$

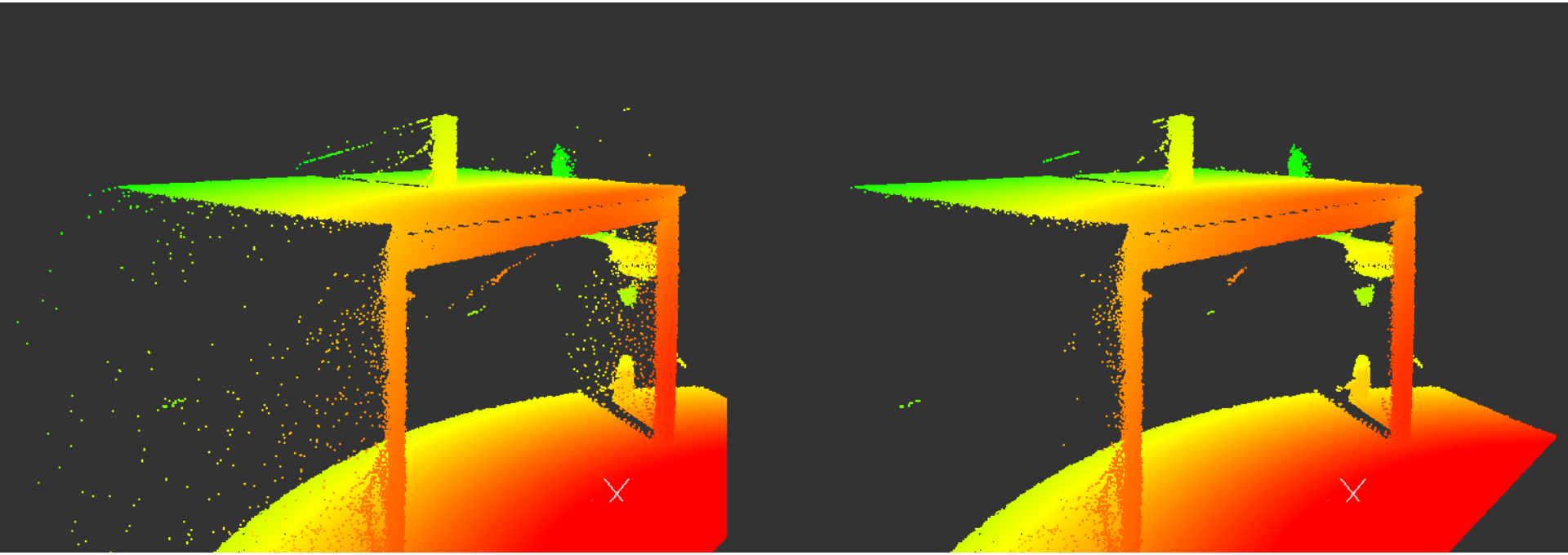


Outlier Removal (cont.)

- Approach 2: Assuming a Gaussian distribution
  - First iteration:
    - Find k nearest neighbors of each point p_i
    - Compute the Euclidian distances to all neighbors, determine the mean value μ_i of the distances, add the value to a distances vector $\vec{d}$
  - Calculate the mean μ and the standard deviation σ of all elements in $\vec{d}$
  - Define a distance threshold: $t = \mu + \alpha \cdot \sigma$, where α is a parameter
  - Second iteration: Mark all points as outliers where $\mu_i > t$ and remove them from the cloud

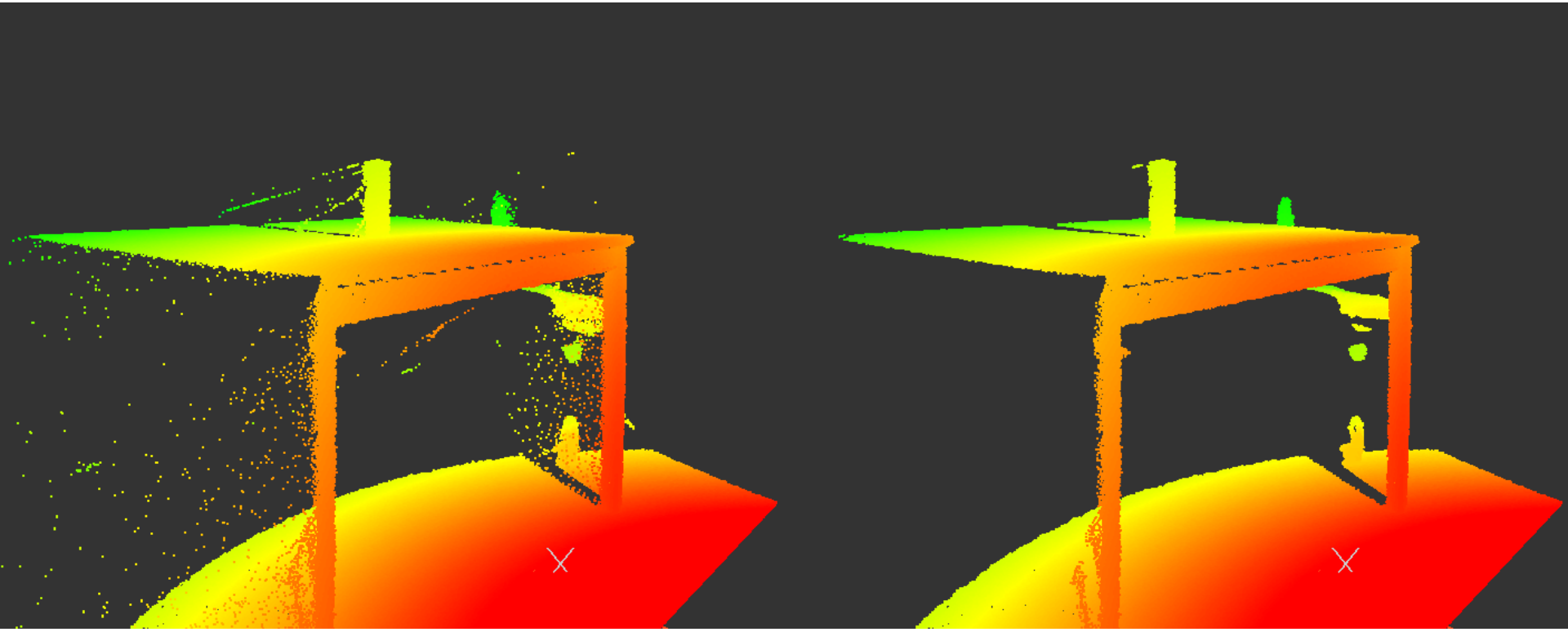
Outlier Removal Example Approach 1

- Input points: 460400
- Output points: 459072
- $k = 5,\ r = 0.02$



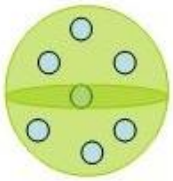
Outlier Removal Example Approach 2

- Input points: 460400
- Output points: 451410
- $k = 50, \alpha = 1$



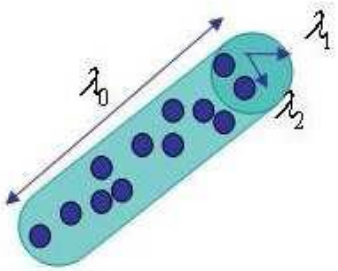
Principal Component Analysis (PCA)

- Construct a covariance matrix C from n given points p_i
  - $C = \frac{1}{n}\sum_{i=0}^{n}(p_i - \bar{p}) \cdot (p_i - \bar{p})^T$
  - where $\bar{p}$ is the centroid of all points p_i
- Determine the eigenvalues λ_j and the eigenvectors $\vec{v}_j$ of C
  - $C \cdot \vec{v}_j = \lambda_j \cdot \vec{v}_j$
  - where $j \in \{0,1,2\}$ for 3D Cartesian data
- Possible distributions of the points if $\lambda_0 \geq \lambda_1 \geq \lambda_2$
  - $\lambda_0 \approx \lambda_1 \approx \lambda_2$: No principal direction, e.g. sphere like distribution

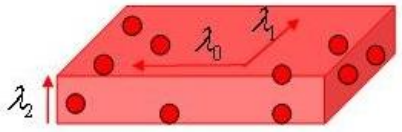


Principal Component Analysis (cont.)

- $\lambda_0 \gg \lambda_1 \approx \lambda_2$: a cylindrical distribution can be observed, of one eigenvalue is significantly larger then the two other ones



- $\lambda_0 \approx \lambda_1 \gg \lambda_2$: the points represent a plane, if two eigenvalues are significantly larger then one



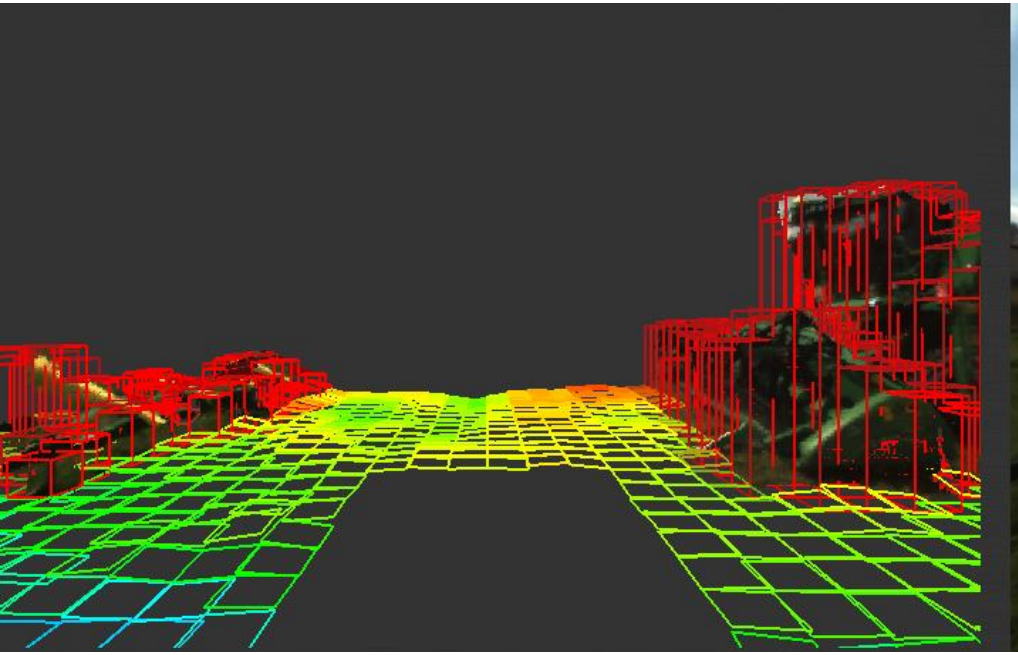
Normal Estimation

- Normals are a powerful criterion to estimate surface shapes and properties and are often used as features for further classification steps
- Approach
  - For each point p_i: find all neighbors within a sphere of given radius
  - Perform a PCA for p_i together with its neighbors
  - Set normal $\vec{n}_i = \vec{v}_2$, where $\vec{v}_2$ is the eigenvector of the smallest eigenvalue $\lambda_2 (\leq \lambda_1 \leq \lambda_0)$
  - Check if the normal is correctly oriented (flip it otherwise)

$$\vec{n}_i \cdot \left(v_p - p_i\right) > 0$$

    - where v_p is the position of the sensor

Obstacle Detection - Example



Obstacle Detection - Example

Coming Next

Localization Systems