Aalto University
School of Electrical
Engineering

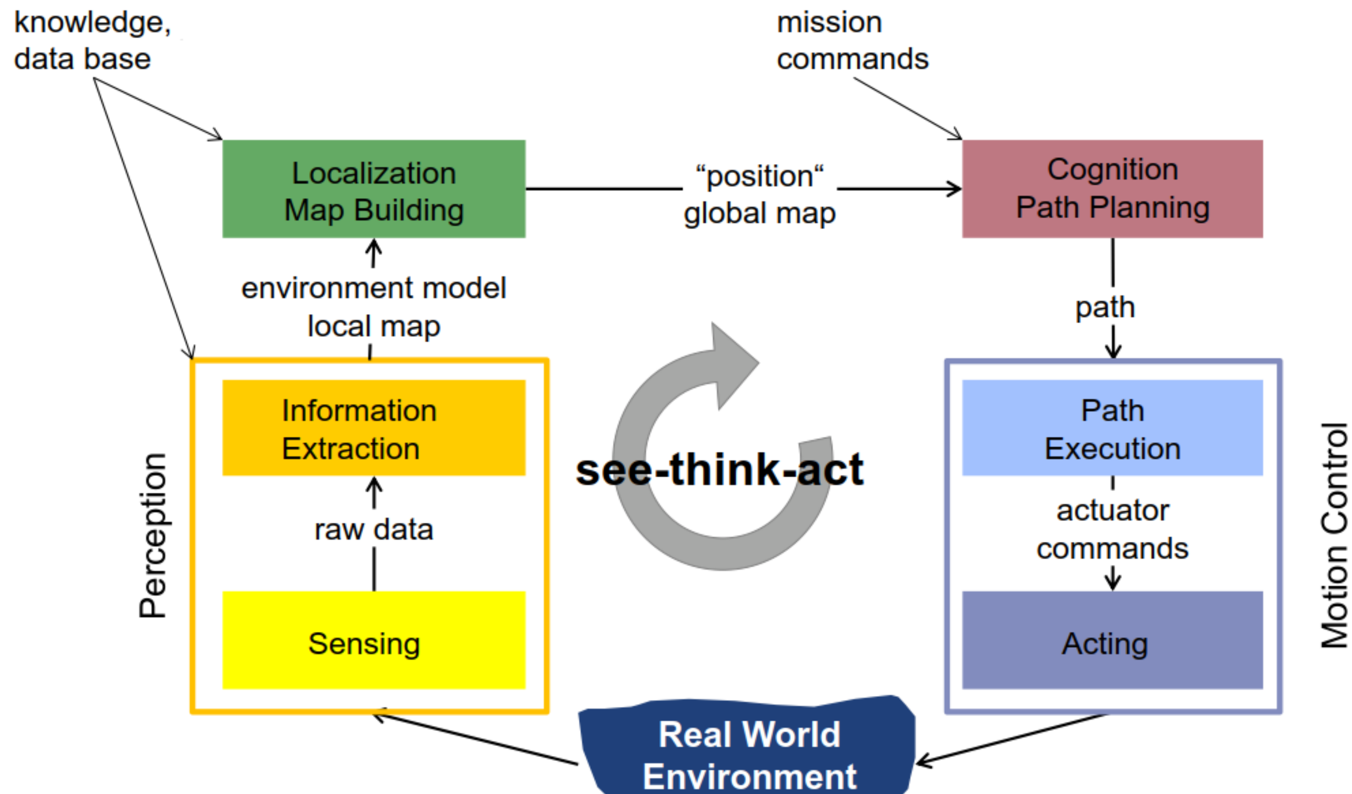
Visual localization and object recognition

Arto Visala

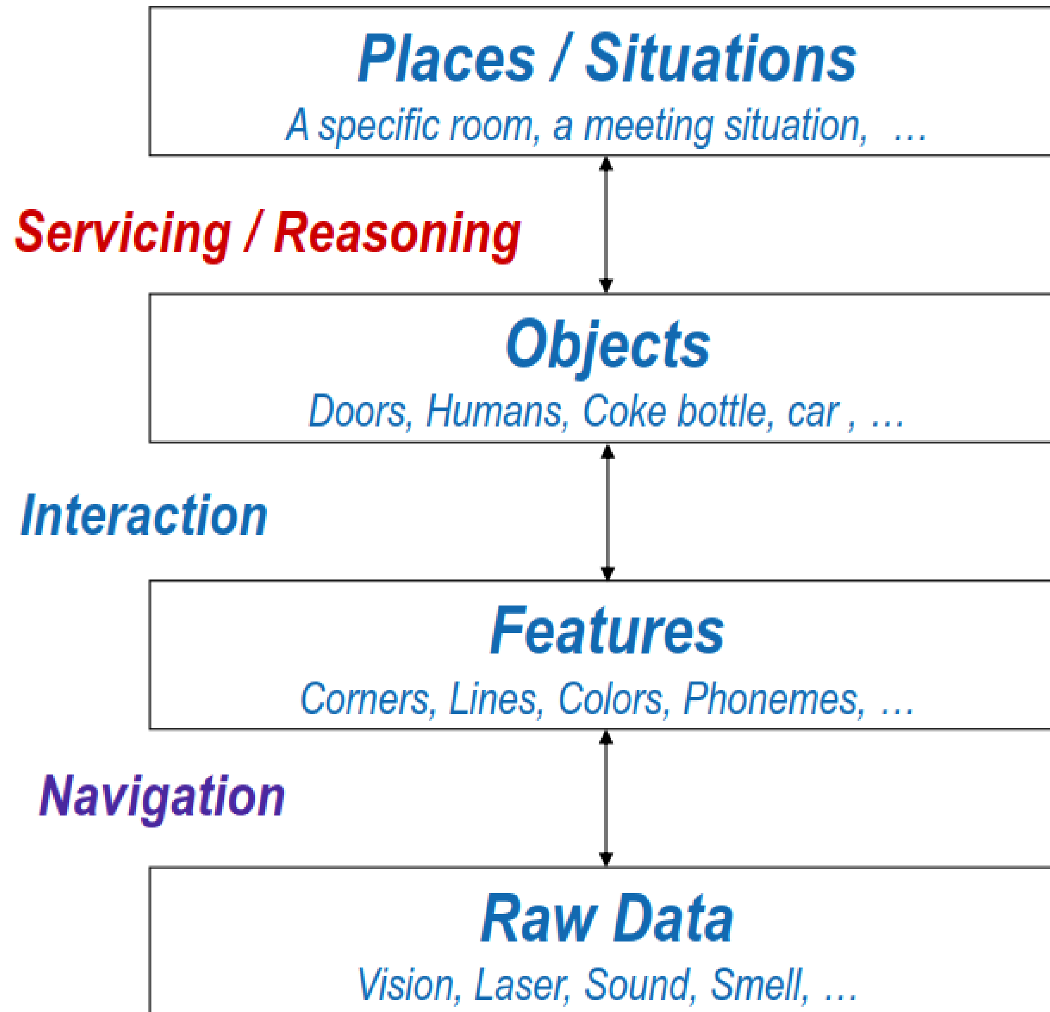
2.4.2019

Sensors and machine perception are the key components for modelling the environment

Mobile Robot Control Scheme

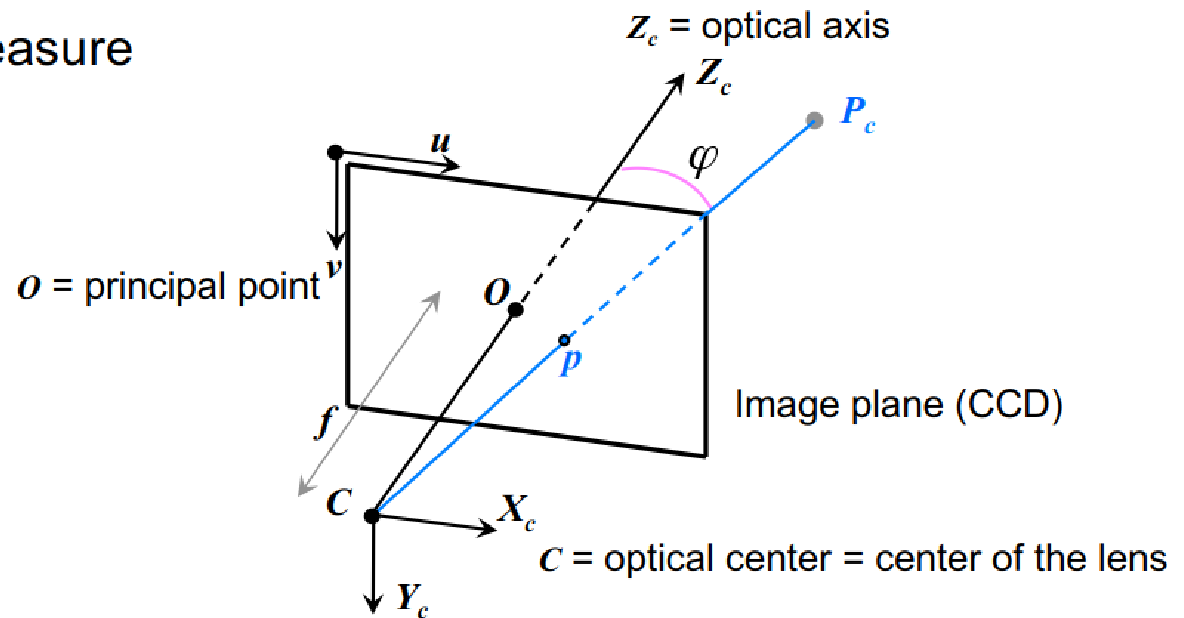


Perception for Mobile Robots



Perspective camera model

- For convenience, the image plane is usually represented in front of C such that the image preserves the same orientation (i.e. not flipped)
- A camera does not measure distances but angles!

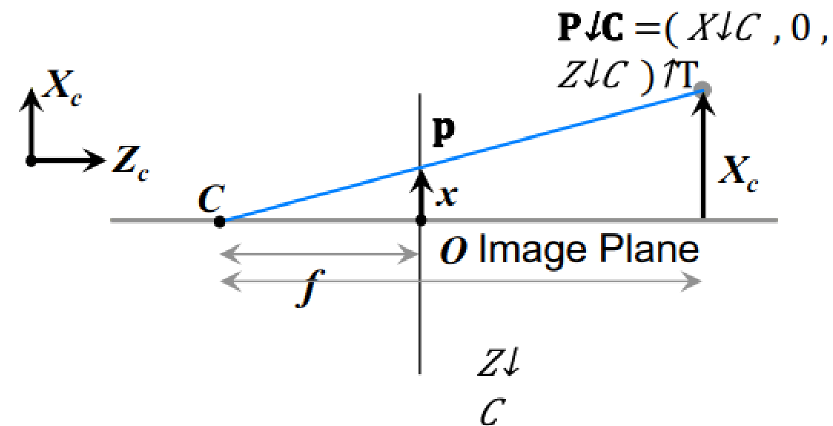


Perspective projection in camera model

- The Camera point $\mathbf{P}_C = (X_C, 0, Z_C)^T$ projects to $\mathbf{p}=(x, y)$ onto the image plane

- From similar triangles:

$$\frac{x}{f} = \frac{X_c}{Z_c} \Rightarrow x = \frac{fX_c}{Z_c}$$

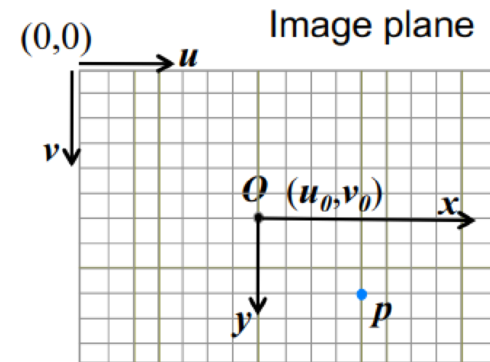


- Similarly, in the general case:

$$\frac{y}{f} = \frac{Y_c}{Z_c} \Rightarrow y = \frac{fY_c}{Z_c}$$

Perspective projection, from scene points to pixels

- To convert **p**, from the local image plane coordinates (x,y) to the pixel coordinates (u,v), we need to account for:
  - The pixel coordinates of the camera optical center $O=(u_0, v_0)$
  - Scale factor k for the pixel-size



$$u = u_0 + kx \Rightarrow u_0 + kfX_C / Z_C$$

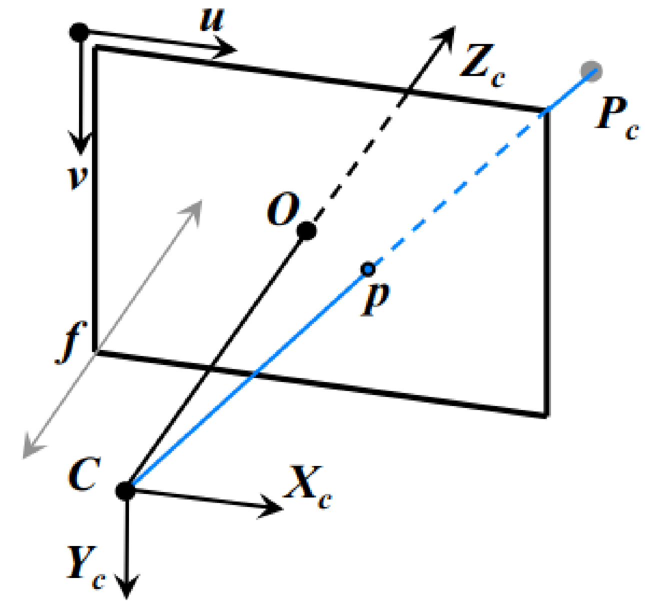
$$v = v_0 + ky \Rightarrow v_0 + kfY_C / Z_C$$

- Use Homogeneous Coordinates for linear mapping from 3D to 2D, by introducing an extra element (scale):

$$p = \begin{pmatrix} u \\ v \end{pmatrix} \quad \Rightarrow \quad \tilde{p} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Perspective projection, from scene points to pixels

- Expressed in matrix form and homogeneous coordinates:



$$\begin{bmatrix} \lambda u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} kf & 0 & u_0 \\ 0 & kf & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

$$= \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \alpha & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Focal length in pixels

Intrinsic parameters matrix

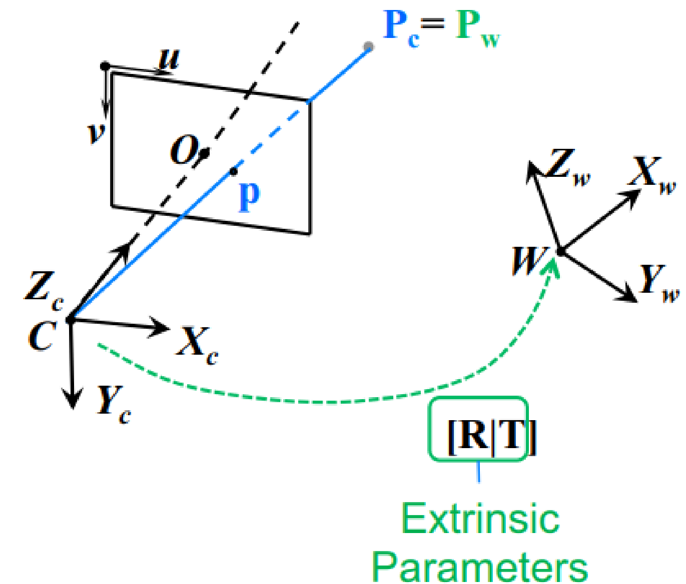
Perspective projection, from scene points to pixels

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} R & | & T \end{bmatrix} \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Perspective Projection Matrix

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R|T] \cdot \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

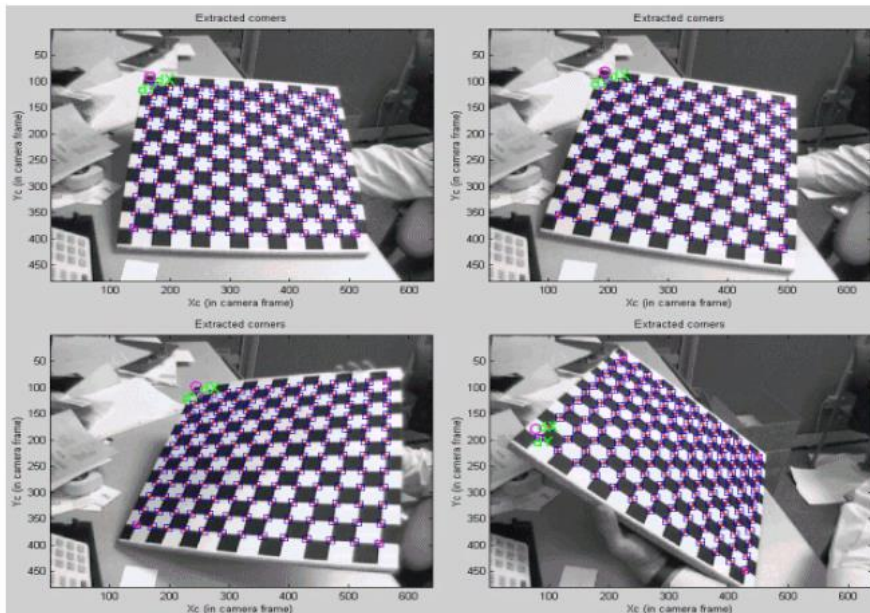


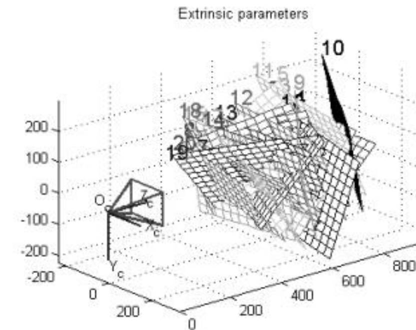
Camera Calibration

- Use camera model to interpret the projection from world to image plane
- Using known correspondences of $\boldsymbol{p} \Leftrightarrow \boldsymbol{P}$, we can compute the unknown parameters $\boldsymbol{K}$, $\boldsymbol{R}$, $\boldsymbol{T}$ by applying the perspective projection equation
- … so associate known, physical distances in the world to pixel-distances in image

Projection Matrix

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \left[R \mid T \right] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$





Stereo Vision versus Structure from Motion

Stereo vision:

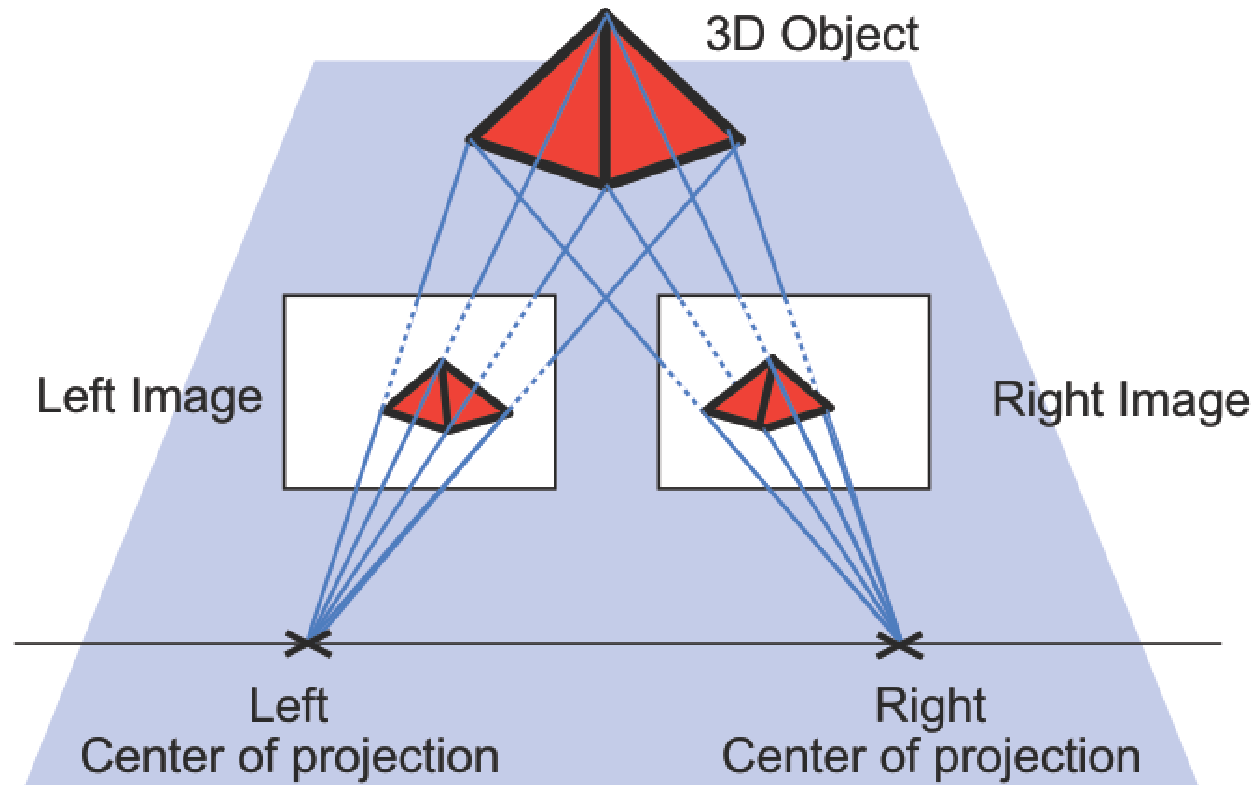
is the process of obtaining depth information from a pair of images coming from two cameras that look at the same scene from different but known positions

Structure from motion, Motion vision:

is the process of obtaining depth and motion information from a pair (sequence) of images coming from the same camera that looks at the same scene from different positions

Depth from Stereo

- From a single camera, we can only deduct the ray on which each image point lies
- With a stereo camera (binocular), we can solve for the intersection of the rays and recover the 3D structure

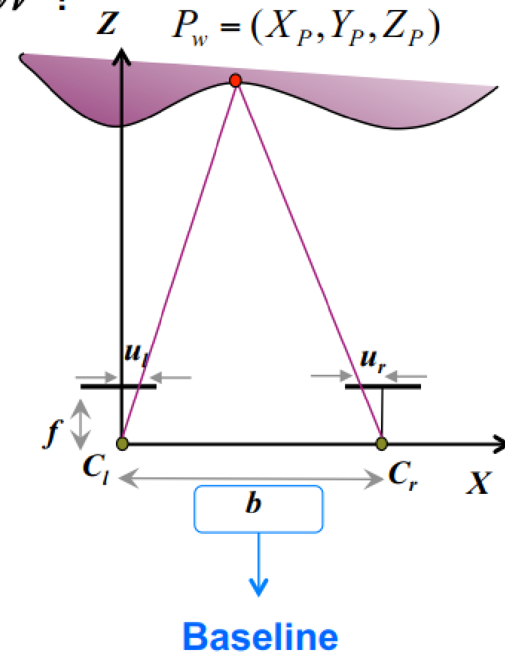


Stereo Vision, simplified case

- An ideal, simplified case assumes that both cameras are **identical** and **aligned** with the x-axis
- Can we find an expression for the depth Z_P of point P_W?
- From similar triangles:

$$\frac{f}{Z_P} = \frac{u_l}{X_P} \qquad \frac{f}{Z_P} = \frac{-u_r}{b - X_P} \quad \Rightarrow \quad Z_P = \frac{bf}{u_l - u_r}$$

$u_l - u_r$ → **Disparity**

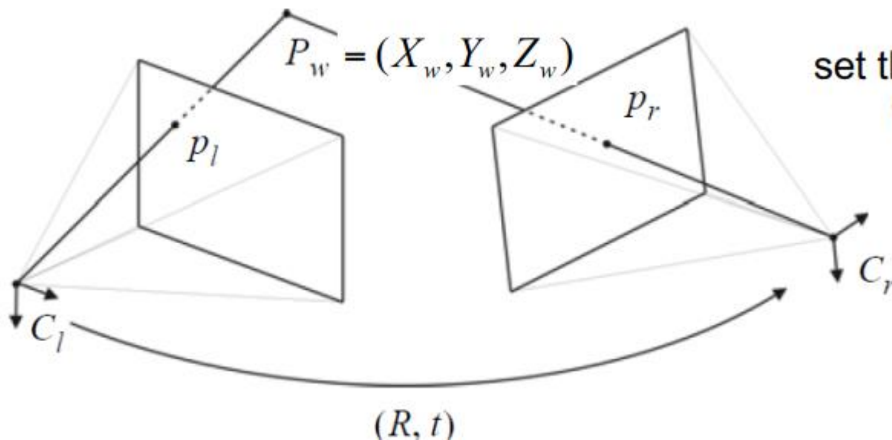


- **Disparity** is the difference in image location of the projection of a 3D point in two image planes
- **Baseline** is the distance between the two cameras

Disparity in inversely proportial to distance

Stereo Vision, general case

- To estimate the 3D position of $P \downarrow W$ we can construct the system of equations of the left and right camera
- Triangulation is the problem of determining the 3D position of a point given a set of corresponding image locations and known camera poses.



Left camera:
set the world frame to coincide with the left camera frame

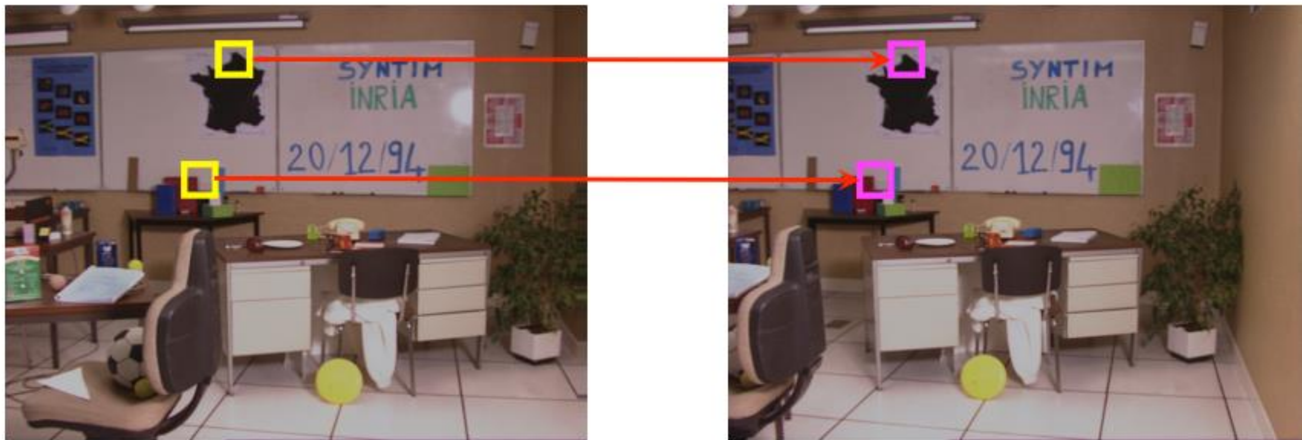
$$\tilde{p}_l = \lambda_l \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = K_l \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

Right camera:

$$\tilde{p}_r = \lambda_r \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = K_r R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + T$$

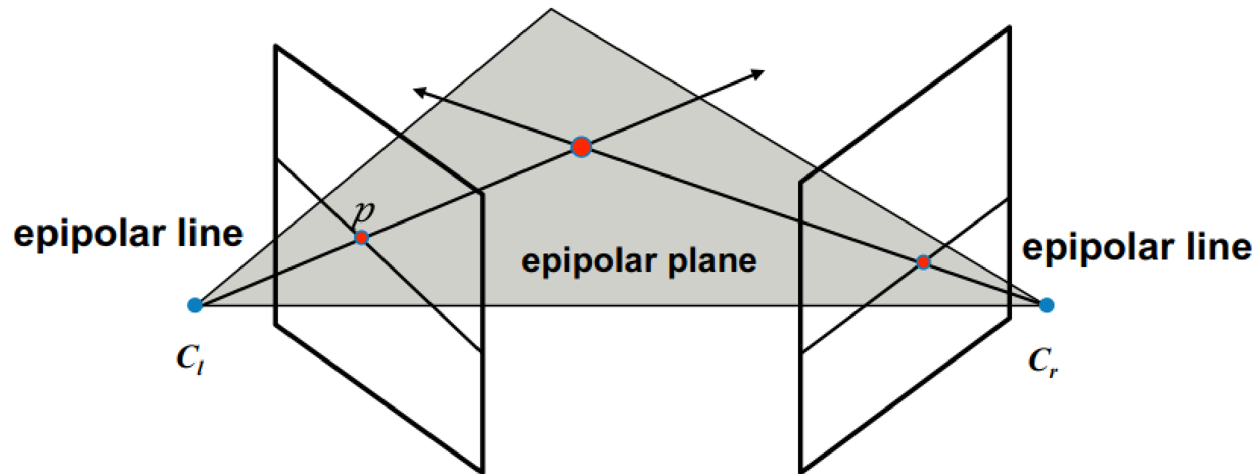
Stereo Vision, Correspondence

- Goal: identify corresponding points in the left and right images, which are the reprojection of the same 3D scene point
  - Typical similarity measures: Normalized Cross-Correlation (NCC) , Sum of Squared Differences (SSD), Sum of Absolute Differences (SAD), Census Transform
  - Exhaustive image search can be computationally very expensive! Can we make the correspondence search in 1D?



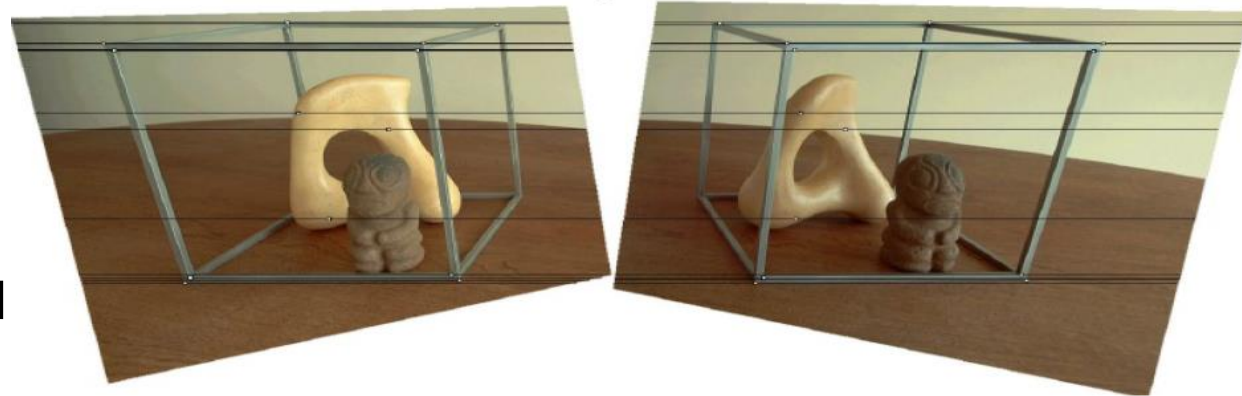
Stereo Vision, search in 1D with the epipolar constraint

- The epipolar plane is defined by the image point **p** and the optical centers
- Impose the epipolar constraint to aid matching: search for a correspondence along the epipolar line



Stereo Vision, Stereo Rectification

- Reprojects image planes onto a common plane parallel to the baseline
- It works by computing two homographies (image warping), one for each input image reprojection
- As a result, the new epipolar lines are horizontal and the scanlines of the left and right image are aligned



Stereo Vision, disparity map

- The disparity map holds the disparity value at every pixel:
  - Identify correspondent points of all image pixels in the original images
  - Compute the disparity $(u_l - u_r)$ for each pair of correspondences
- Usually visualized in gray-scale images
- Close objects experience bigger disparity; thus, they appear brighter in disparity map

Left image

Right image

Disparity Map

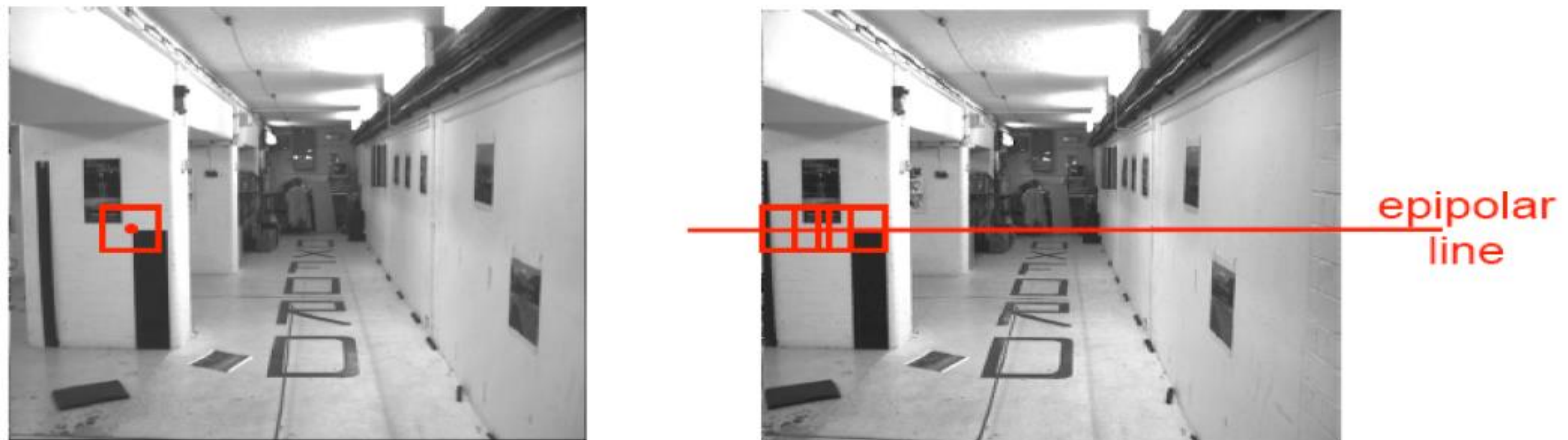
Stereo Vision, disparity map

- The disparity map holds the disparity value at every pixel:
  - Identify correspondent points of all image pixels in the original images
  - Compute the disparity $(u_l - u_r)$ for each pair of correspondences
- Usually visualized in gray-scale images
- Close objects experience bigger disparity; thus, they appear brighter in disparity map
- From the disparity, we can compute the depth Z as:

$$Z = \frac{bf}{u_l - u_r}$$

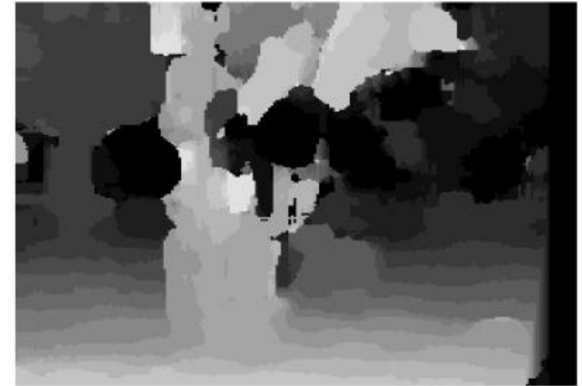
Correspondence problem

- To average noise effects, use a window around the point of interest
- Neighborhood of corresponding points are similar in intensity patterns
- **Similarity measures**:
  - Zero-Normalized Cross-Correlation (**ZNCC**)
  - Sum of Squared Differences (**SSD**),
  - Sum of Squared Differences (**SAD**)
  - **Census Transform** (Census descriptor plus Hamming distance)



Effects of window size W

W = 3

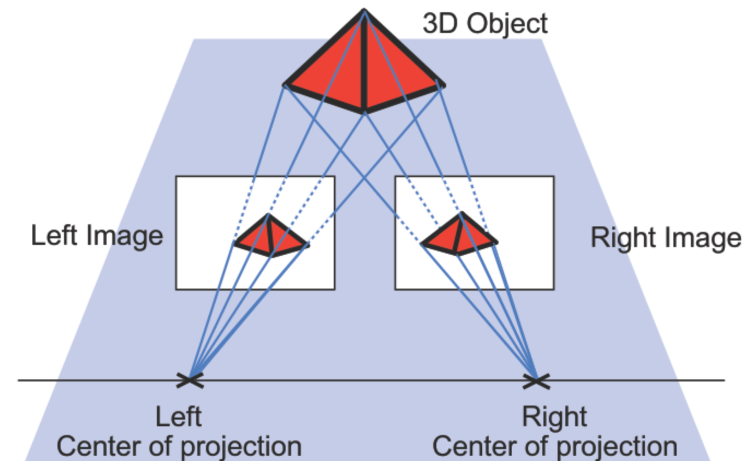


W = 20

- Smaller window
  - + More detail
  - – More noise
- Larger window
  - + Smoother disparity maps
  - – Less detail

Stereo Vision, summary

1. Stereo camera calibration -> compute camera relative pose
2. Epipolar rectification -> align images & epipolar lines
3. Search for correspondences
4. Output: compute stereo triangulation or disparity map
5. Consider how baseline & image resolution affect accuracy of depth estimates



Filtering, Edges, and Point-features

- Convolution for filter

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G = H * F$$

Notation for convolution operator

- Correlation for matching

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

$$G = H \otimes F$$

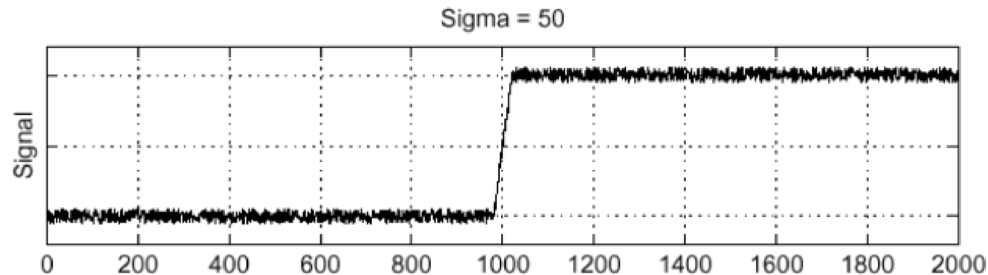
- We can use correlation for template matching to detect locations similar to templates

Derivative Theorem of Convolution in 1D

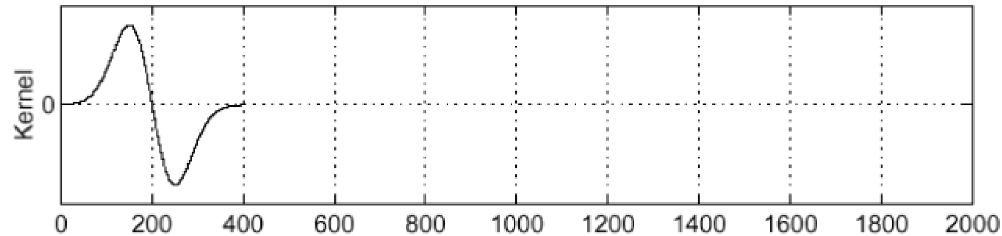
- Gaussiam smoothing + derivative filtering

- $s'(x) = \frac{d}{dx}\left(G_\sigma(x) * I(x)\right) = G'_\sigma(x) * I(x)$
- This saves us one operation:

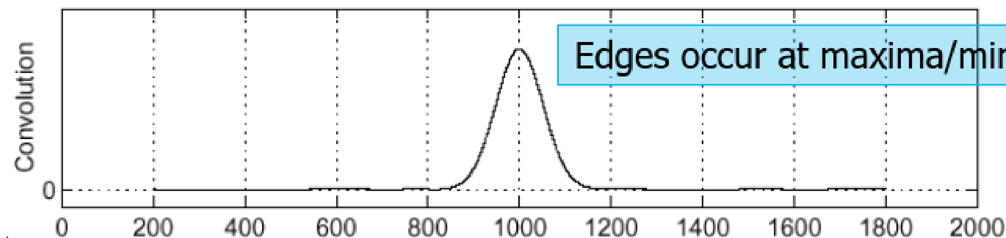
$I(x)$



$G'_\sigma(x) = \frac{d}{dx} G_\sigma(x)$



$s'(x) = G'_\sigma(x) * I(x)$

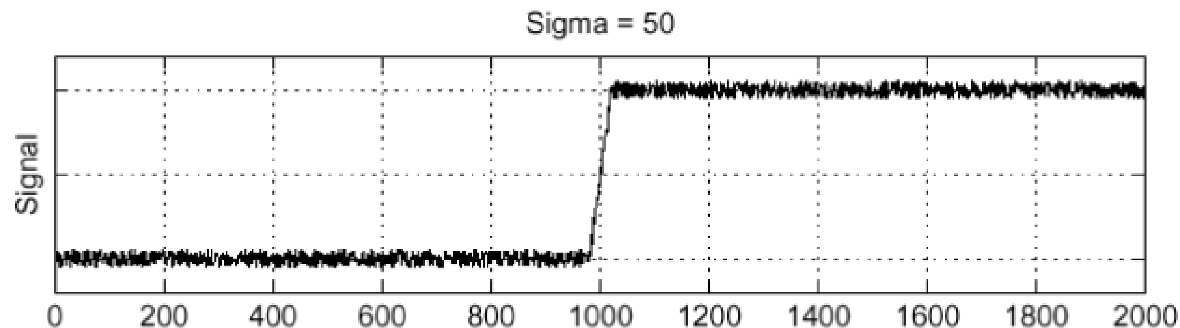


Edges occur at maxima/minima of $s'(x)$

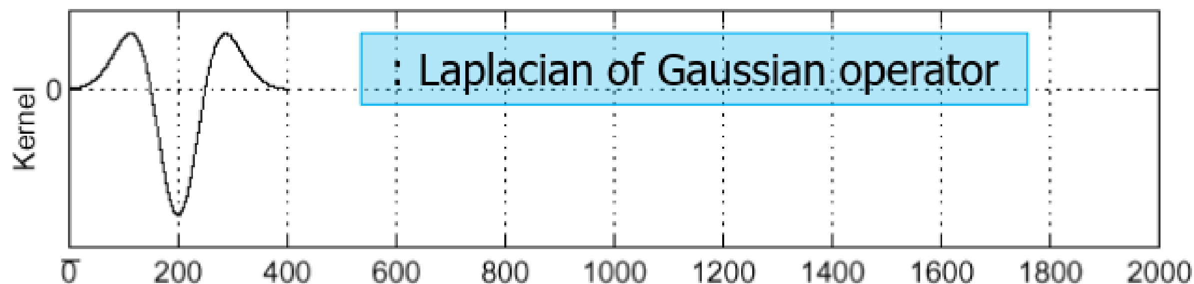
Zero-crossings with Laplacian in 1D

- Gaussiam smoothing + Laplacian filtering in one operation

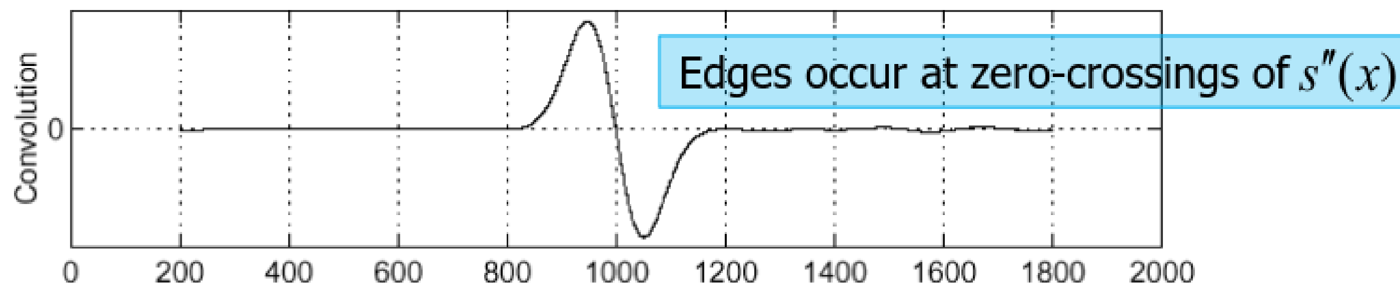
$$I(x)$$



$$G''_\sigma(x) = \frac{d^2}{dx^2} G_\sigma(x)$$



$$s''(x) = G''_\sigma(x) * I(x)$$



2D Edge Detection

- Find gradient of smoothed image in both directions

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

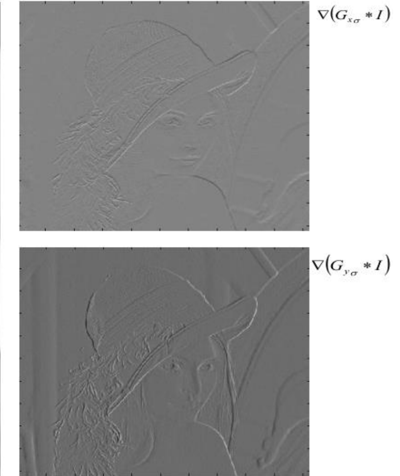
- Discard pixels with $|\nabla S|$ (i.e. edge strength) below a certain threshold
- **Non-maxima suppression**: identify local maxima of $|\nabla S|$
  ⇨ detected edges

2D Edge Detection with Canny edge detector

$\nabla S = \nabla(G_\sigma * I)$

I : original image (Lena image)

$|\nabla S|$: Edge strength



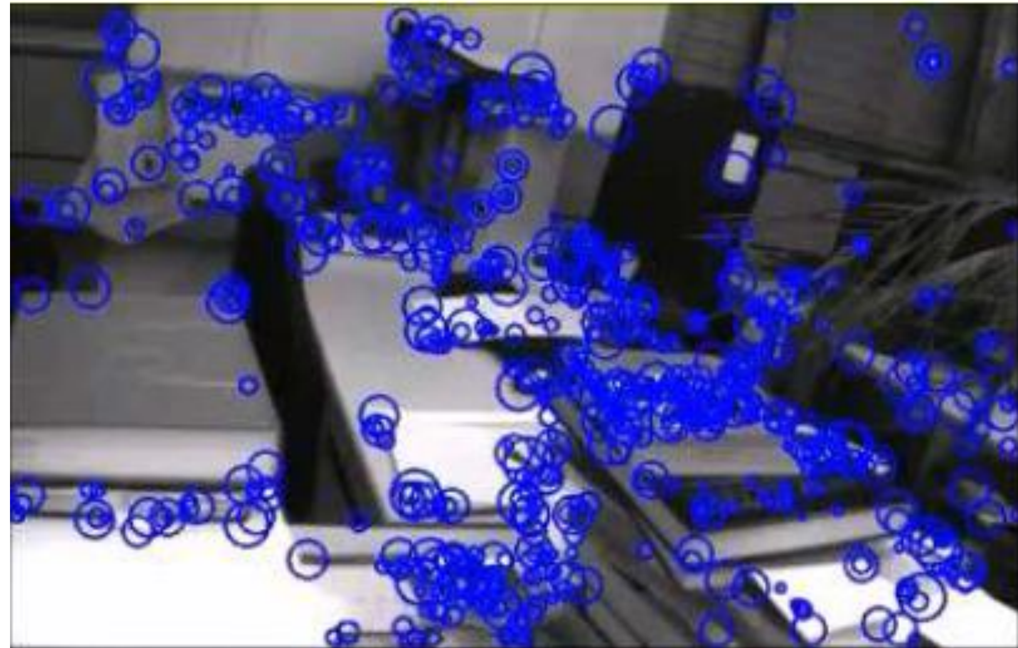
Thresholding $|\nabla S|$

Thinning: non-maximal suppression ⇨ edge image

Aalto University
School of Electrical Engineering

Point-feature extraction

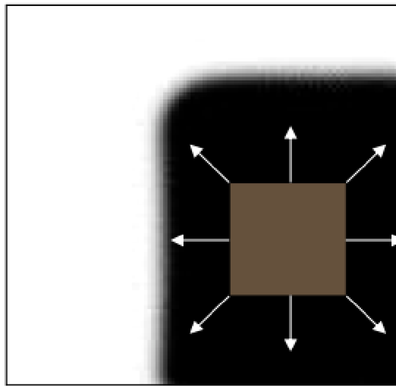
- Harris corners
- SIFT features
- and more recent algorithms from the state of the art
- Application: visual odometry
- Videos from the Robotics and Perception Group: http://rpg.ifi.uzh.ch



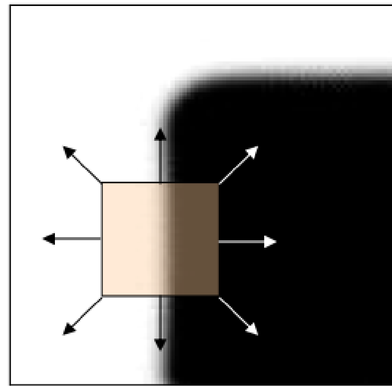
Corner Detection

Corner Detection

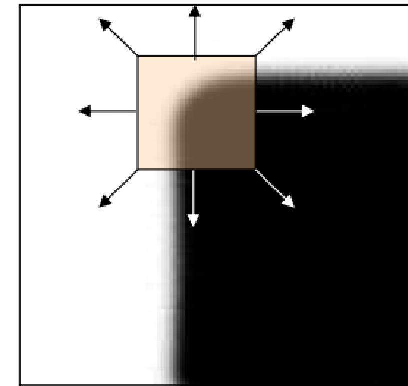
- Shifting a window in **any direction** should give a **large change** of intensity in at least 2 directions



"flat" region:
no intensity change



"edge":
no change along the edge direction



"corner":
significant change in at least 2 directions

How do we implement Harris corner detector

- Two image patches of size ***P*** one centered at (x, y) and one centered at $(x+\Delta x, y+\Delta y)$

- The Sum of Squared Differences between them is:

$$SSD(\Delta x, \Delta y) = \sum_{x,y \in P} \left(I(x,y) - I(x+\Delta x, y+\Delta y) \right)^2$$

- Let $I_x = \frac{\partial I(x,y)}{\partial x}$ and $I_y = \frac{\partial I(x,y)}{\partial y}$. Approximating with a 1st order Taylor expansion:

$$I(x+\Delta x, y+\Delta y) \approx I(x,y) + I_x(x,y)\Delta x + I_y(x,y)\Delta y$$

- This produces the approximation

$$SSD(\Delta x, \Delta y) \approx \sum_{x,y \in P} \left(I_x(x,y)\Delta x + I_y(x,y)\Delta y \right)^2$$

How do we implement Harris corner detector

$$SSD(\Delta x, \Delta y) \approx \sum_{x,y \in P} \left(I_x(x,y)\Delta x + I_y(x,y)\Delta y \right)^2$$

- This can be written in a matrix form as

$$SSD(\Delta x, \Delta y) \approx \sum [\Delta x \quad \Delta y] \begin{bmatrix} I_x{}^2 & I_x I_y \\ I_x I_y & I_y{}^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$\Rightarrow SSD(\Delta x, \Delta y) \approx \sum [\Delta x \quad \Delta y] M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$M = \sum_{x,y \in P} \begin{bmatrix} I_x{}^2 & I_x I_y \\ I_x I_y & I_y{}^2 \end{bmatrix} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \quad I_y]$$

2nd moment matrix

Alternative ways to write this matrix

How do we implement Harris corner detector

$$SSD(\Delta x, \Delta y) \approx \sum_{x,y \in P} \left(I_x(x,y)\Delta x + I_y(x,y)\Delta y \right)^2$$

- This can be written in a matrix form as

$$SSD(\Delta x, \Delta y) \approx \sum \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} I_x^{\ 2} & I_x I_y \\ I_x I_y & I_y^{\ 2} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$\Rightarrow SSD(\Delta x, \Delta y) \approx \sum \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$M = \sum_{x,y \in P} \begin{bmatrix} I_x^{\ 2} & I_x I_y \\ I_x I_y & I_y^{\ 2} \end{bmatrix} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix}$$

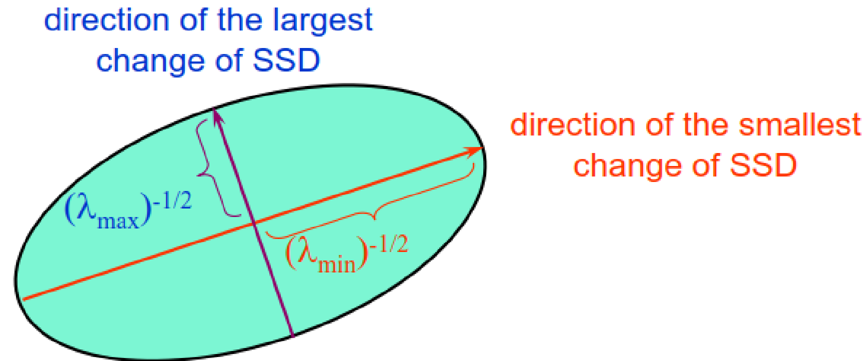
2nd moment matrix

Alternative ways to write this matrix

Harris corner detector, Interpretation of matrix M

-

- Since M is symmetric, it can always be decomposed into $M = R^{-1}\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$
- We can visualize $[\Delta x \quad \Delta y] M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = const$ as an ellipse with axis lengths determined by the ***eigenvalues*** and the two axes' orientations determined by *R* (i.e., the **eigenvectors** of M)
- The two eigenvectors identify the two orthogonal directions of largest and smallest changes of SSD

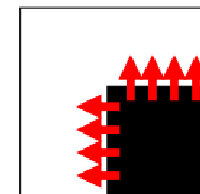


Harris corner detector, Interpretation of matrix M

What does this matrix M reveal?

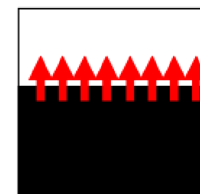
- First, consider an axis-aligned corner:

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$



Corner

- This means dominant gradient directions align with x or y axis
- If either λ_1 or λ_2 is close to 0, then this is **not** a corner:



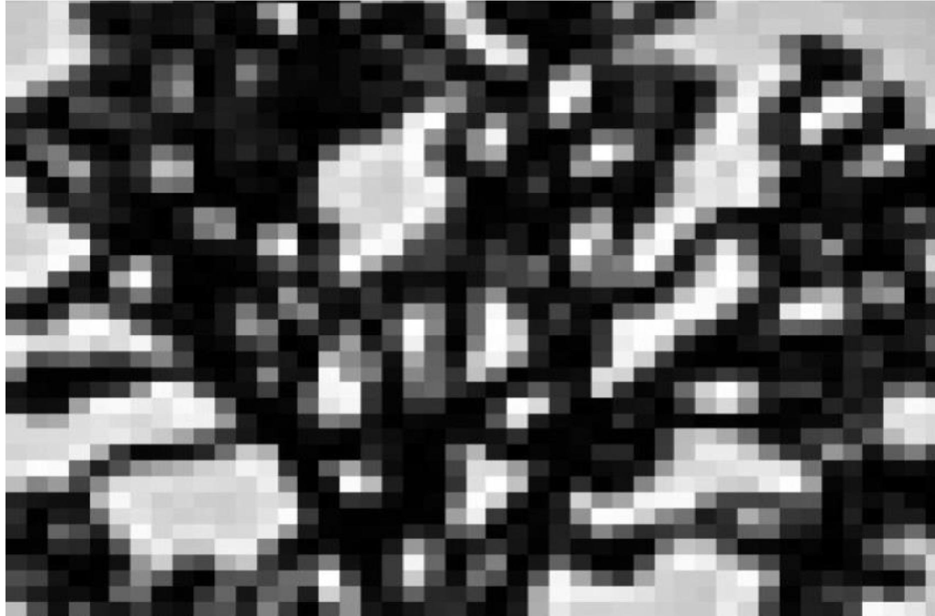
Edge

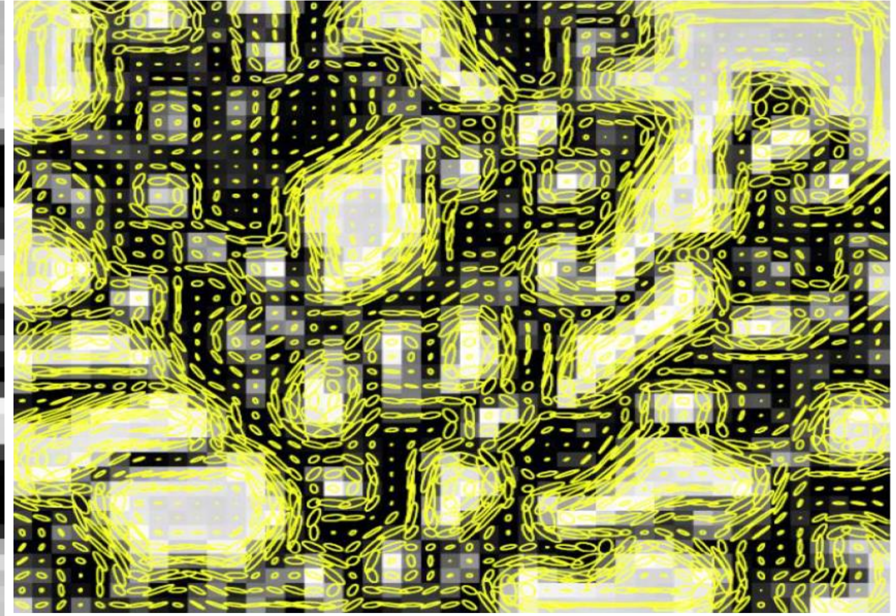
$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Flat region

Harris corner detector, Visualization of 2nd moment matrices

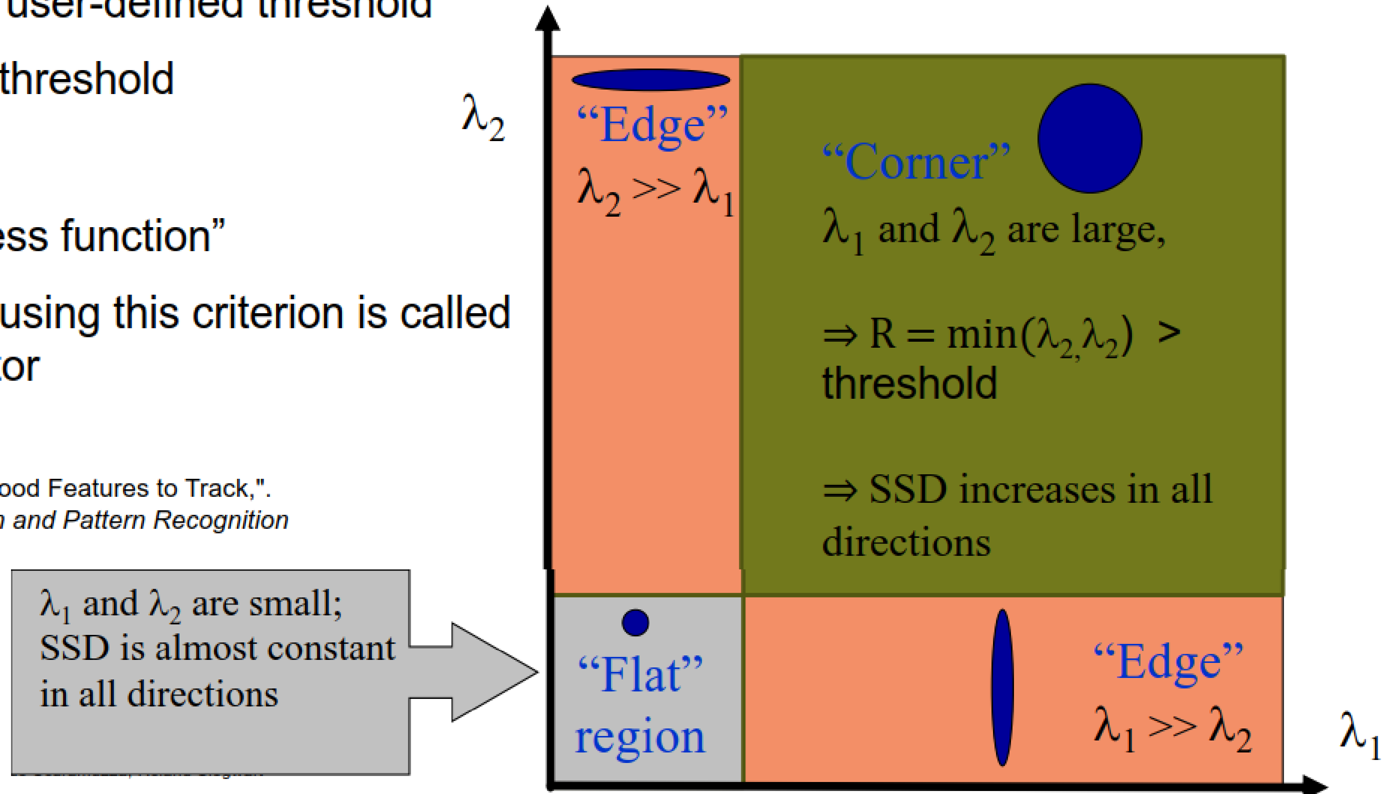




Harris corner detector, Interpreting the eigenvalues

- Classification of image points using eigenvalues of *M*
- A corner can then be identified by checking whether the minimum of the two eigenvalues of M is larger than a certain user-defined threshold
- $\Rightarrow$ R = $\min(\lambda_2, \lambda_2)$ > threshold
- R is called "cornerness function"
- The corner detector using this criterion is called «Shi-Tomasi» detector

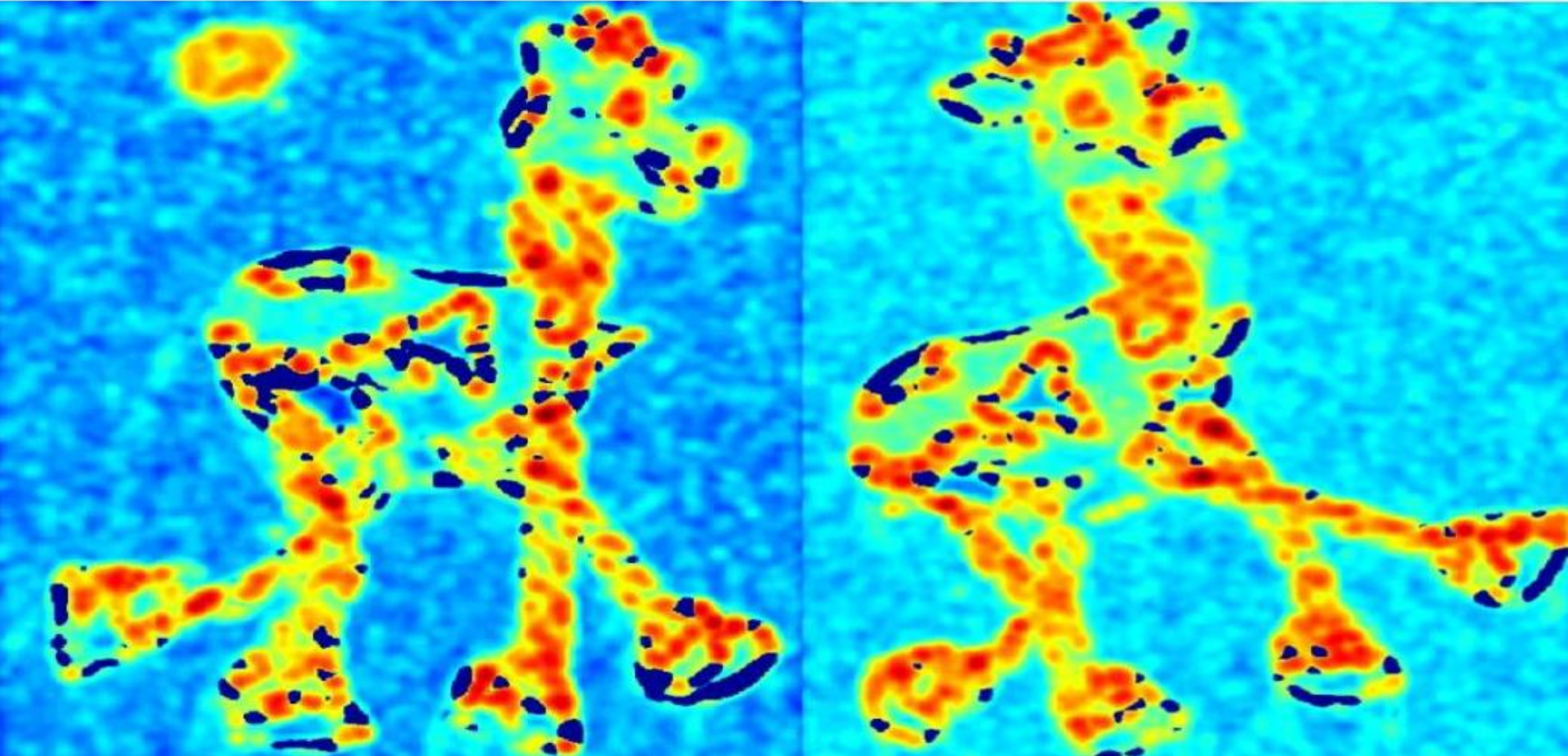
J. Shi and C. Tomasi (June 1994). "Good Features to Track,". *IEEE Conference on Computer Vision and Pattern Recognition*



Harris corner detector: Workflow by ETH

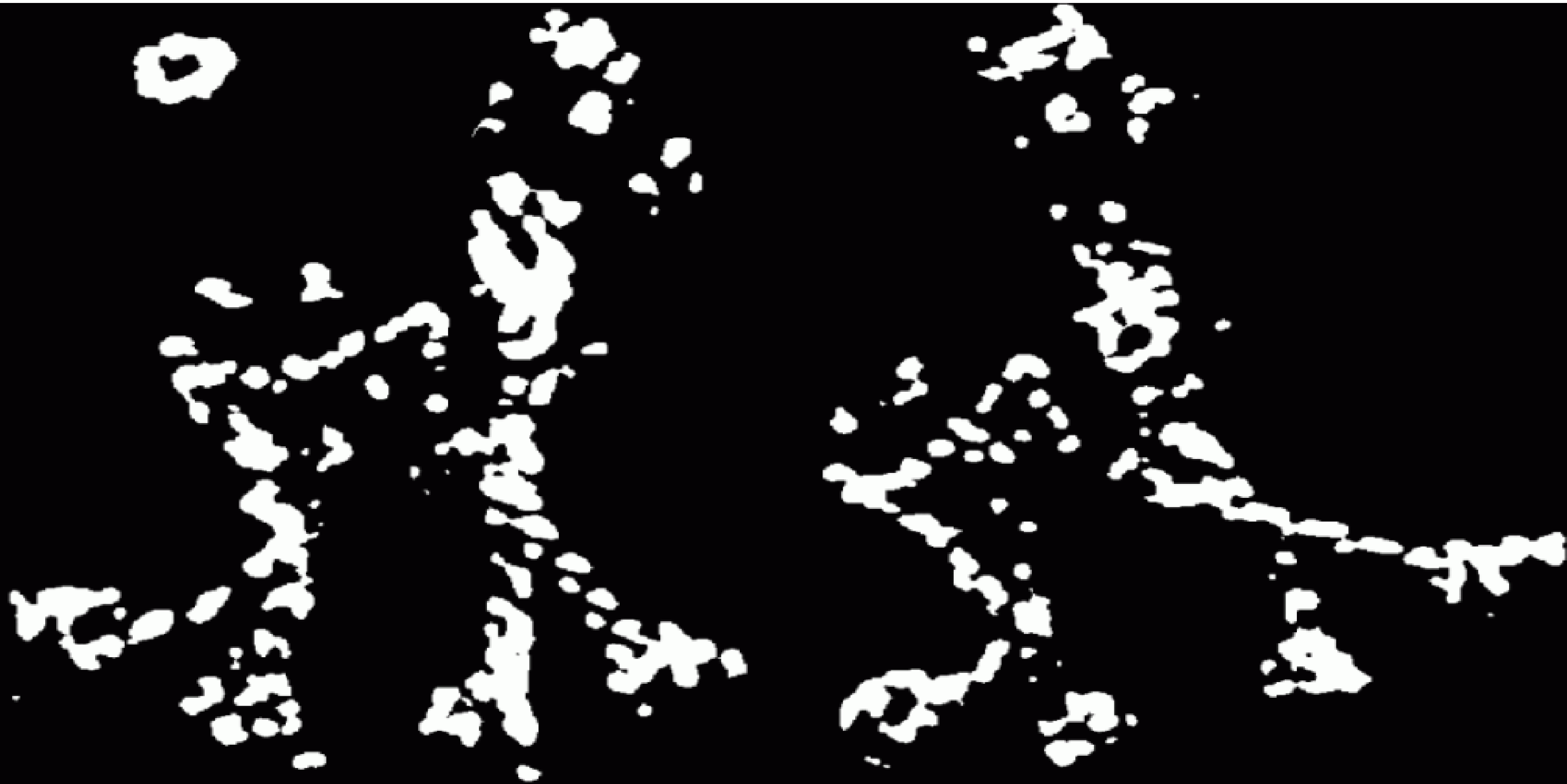
Harris corner detector: Workflow by ETH

- Compute corner response C



Harris corner detector: Workflow by ETH

- Find points with large corner response: $C >$ threshold



Harris corner detector: Workflow by ETH

- Corner response C is invariant to image rotation,.
- Probably the most widely used and known corner detector

Blob Detection

Blob features

A blob is an image pattern that differs from its neighbors in intensity and texture (e.g., a circle, a star, an ellipse, or **any particular patch which is not a corner**!)

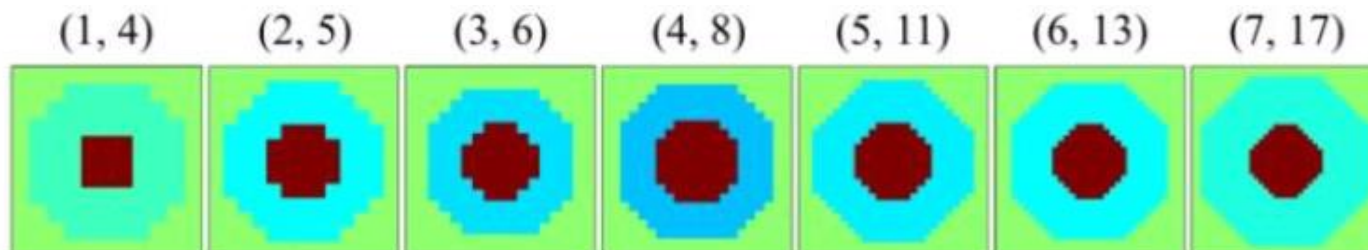
- Has less localization accuracy than a corner (e.g., what's the center of a blob?)
- It's more distinctive than a blob

The most popular blob detectors are

- LoG: Laplacian of Gaussian operator
- DoG: Difference of Gaussian
- SIFT (it uses DoG features)
- SURF (it's an fast implementation of SIFT)
- CenSurE
- MSER

CenSurE features

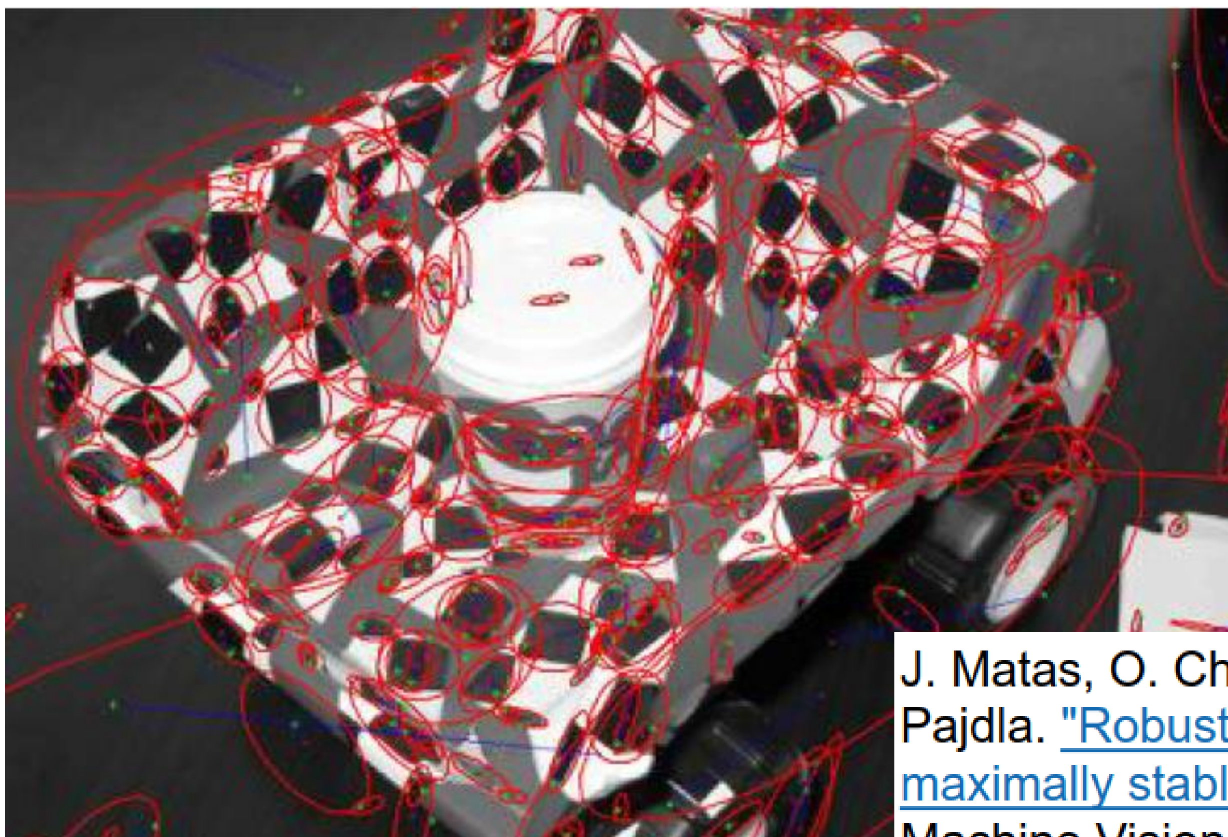
Approximates LoG/DoG by octagonal box filters



M. Agrawal, K. Konolige, M. R. Blas. CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching ECCV 2008

MSER blob detector

Looks for elliptical regions of uniform color



J. Matas, O. Chum, M. Urban, and T. Pajdla. "Robust wide baseline stereo from maximally stable extremal regions." British Machine Vision Conference, 2002.

SIFT features

SIFT (Scale Invariant Feature Transform) is an approach for detecting and describing regions of interest (blobs) in an image developed by D. Lowe (Univ. of Bristish Columbia, Canada) in 2004 and today used in most vision applications (Google image search, image retrieval, place recognition, and consumer cameras)

After 11 years since its invention, SIFT is still the best performing and most robust feature descriptor; SURF, BRIEF, BRISK are suboptimal (way more efficient than SIFT but not as robust to changes in view point)!

Things you should remember:

- SIFT detects DoG features
- SIFT is scale invariant: the same features can be re-detected from images taken with significant distance from each other (i.e., re-scaled versions of the image)
- SIFT is also invariant to orientation and changes of view-point (up to 60 degrees)
- SIFT introduces a “descriptor” based on gradient orientations, which is more robust that just using pixel intensities

SIFT features summary

SIFT features are reasonably invariant to changes in:

- Rotation
- Scaling
- Small changes in viewpoint,
- Illumination

Very powerful in capturing and describing distinctive features but also computationally demanding

SIFT feature detector Demo:

for Matlab, Win, and Linux (freeware)

http://www.cs.ubc.ca/~lowe/keypoints/

http://www.vlfeat.org/~vedaldi/code/sift.html

Make your own panorama with AUTOSTITCH (freeware):

http://www.cs.ubc.ca/~mbrown/autostitch/autostitch.html

Open-source code for FAST, BRIEF, BRISK and many more, available at the OpenCV library

Place Recognition, Line Extraction

- **Place recognition using Vocabulary Tree**
- Line extraction from images
- Line extraction from laser data

Q: Is this Book present in the Scene?

Most of the Book’s keypoints are present in the Scene

⇨ A: The Book is present in the Scene

Taking this a step further… becomes computationaly unfeasable

- Find an object in an image

?

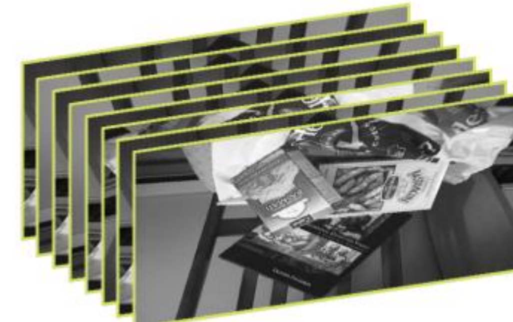
- Find an object in multiple images

?

- Find multiple objects in multiple images

As the number of images increases, feature based object recognition becomes computationaly unfeasable

?



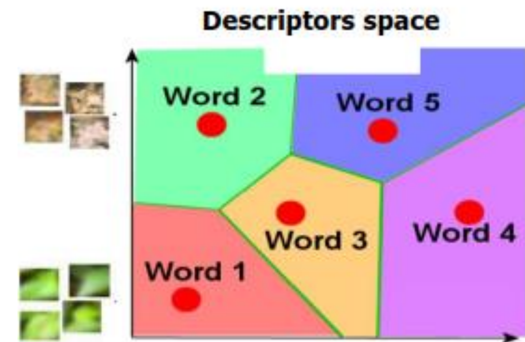
Robot: Have I been to this place before?

- Building the Visual Vocabulary

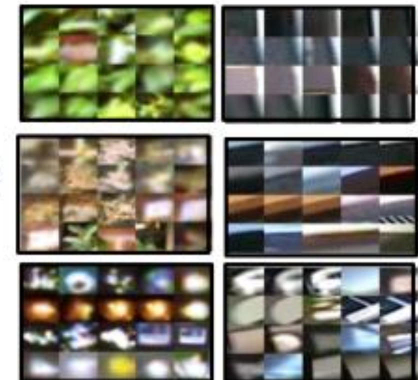
Image Collection

Extract Features

Cluster Descriptors



Examples of Visual Words:



Inverted File index

Inverted File DB

Visual word | List of images that this word appears in

0
...
101
102
103
104
105
...

Query Image Q

Query image Q

Visual words in Q

101
102
105
105
180
180
180

Voting Array for Q

+1 +1 +1

...

Efficient Place/Object Recognition

- If we have millions of visual words, how do we efficiently associate an image feature to the visual word it belongs to?
- In principle, we would have to compare each feature with all visual words

- How can we do this efficiently?
  - Build **Vocabulary Tree** via hierarchical clustering

[Nister and Stewenius, CVPR 2006]

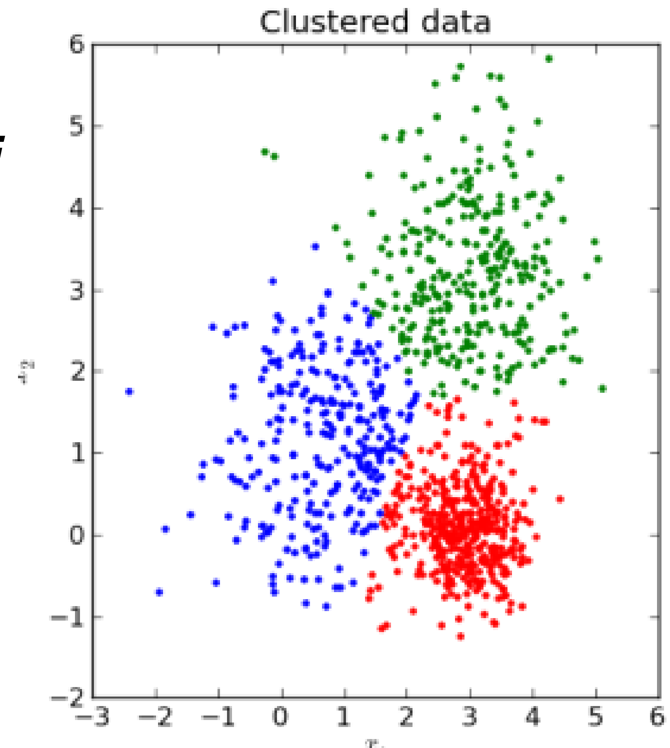
K-means clustering - Review

- k-means clustering is an algorithm to partition n observations into k clusters in which each observation $\boldsymbol{x}j$ belongs to the cluster with center $\boldsymbol{m}i$
- It minimizes the sum of squared Euclidean distances between points $\boldsymbol{x}j$ and their nearest cluster centers $\boldsymbol{m}i$

$$D(X, M) = \sum_{i=1}^{k} \sum_{j=1}^{n} (x_j - m_i)^2$$

Algorithm:

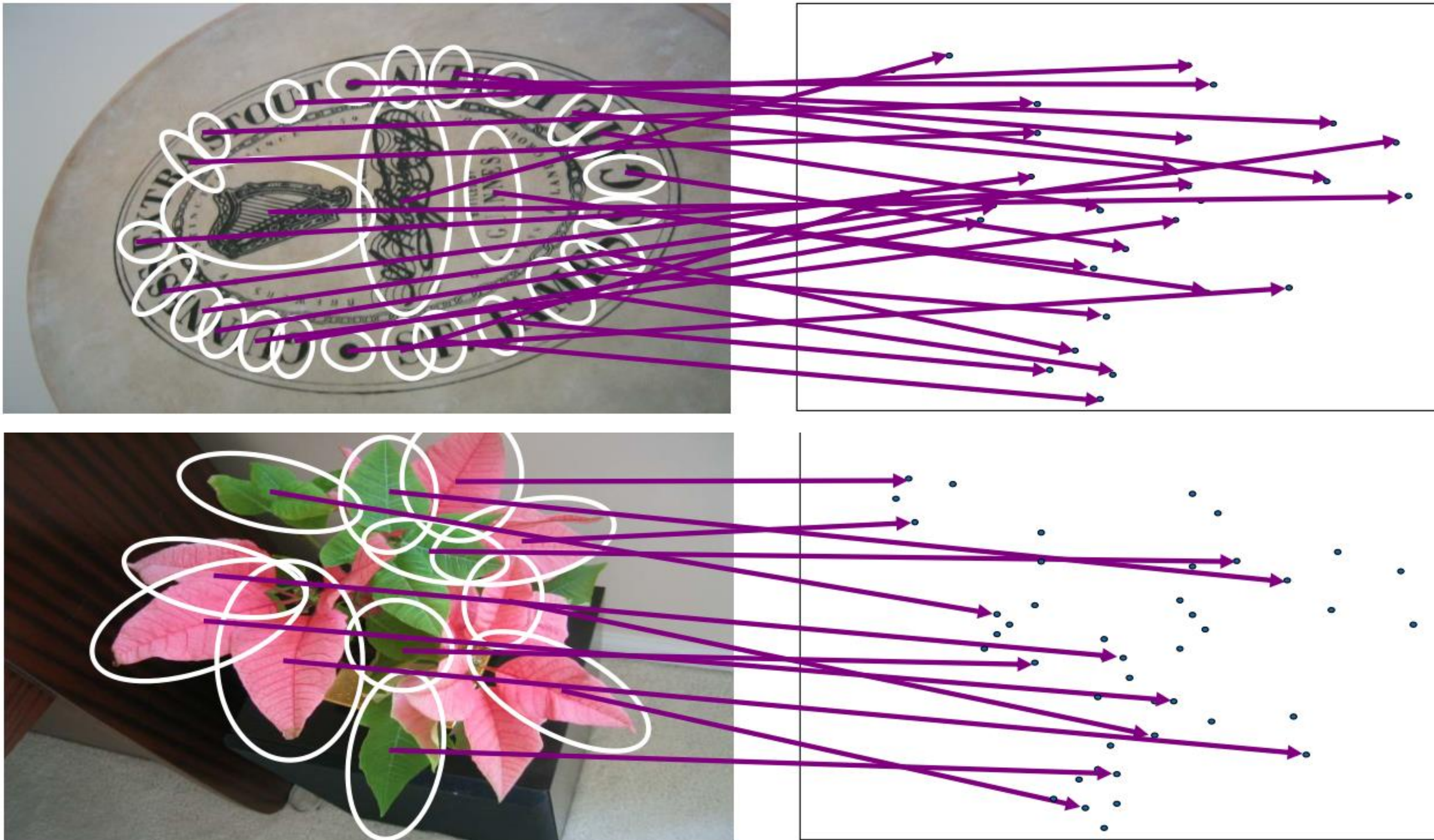
- Randomly initialize k cluster centers
- Iterate until convergence:
  - Assign each data point $\boldsymbol{x}j$ to the nearest center $\boldsymbol{m}i$
  - Recompute each cluster center as the mean of all points assigned to it



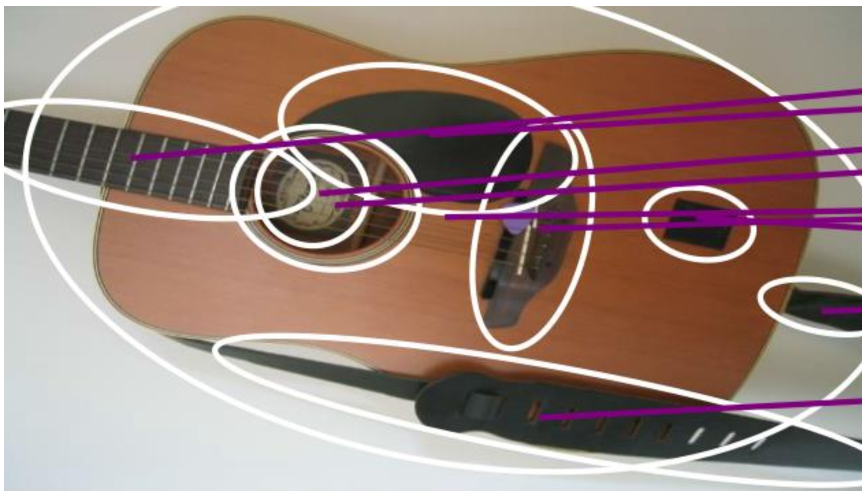
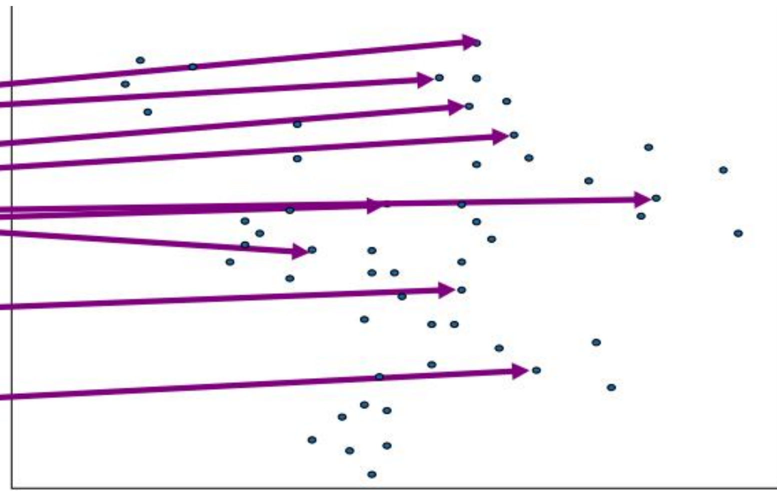
K-means clustering - Demo

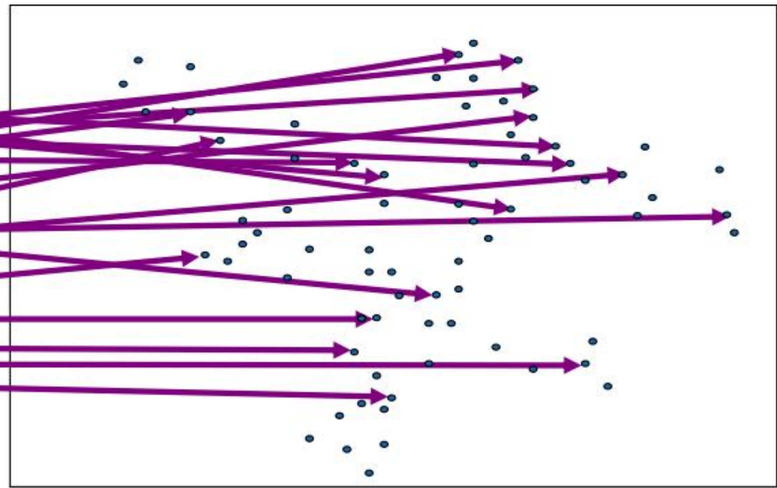
Source: http://shabal.in/visuals/kmeans/1.html

Recognition with K-tree – Populate the descriptor space

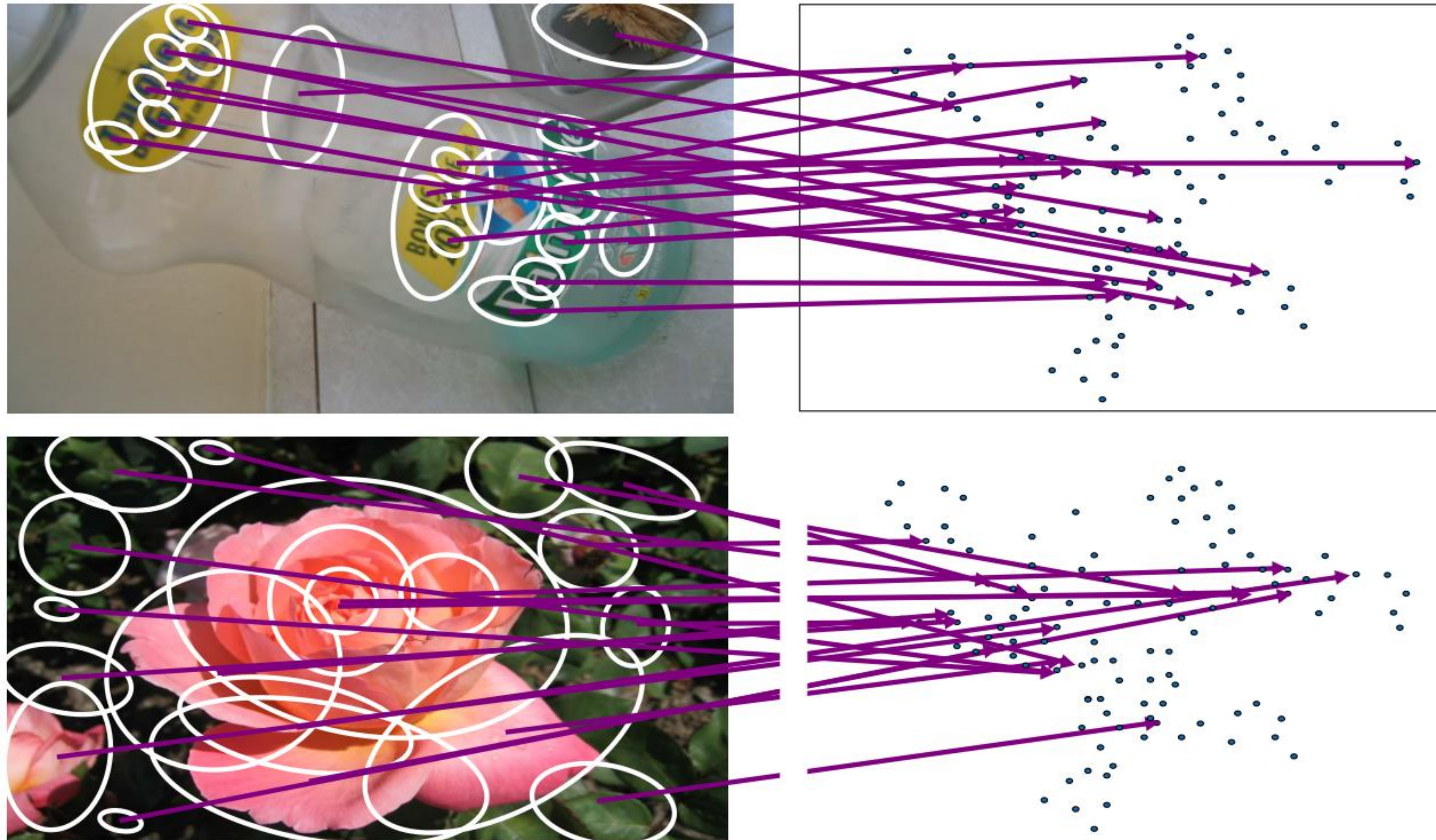


Recognition with K-tree – Populate the descriptor space

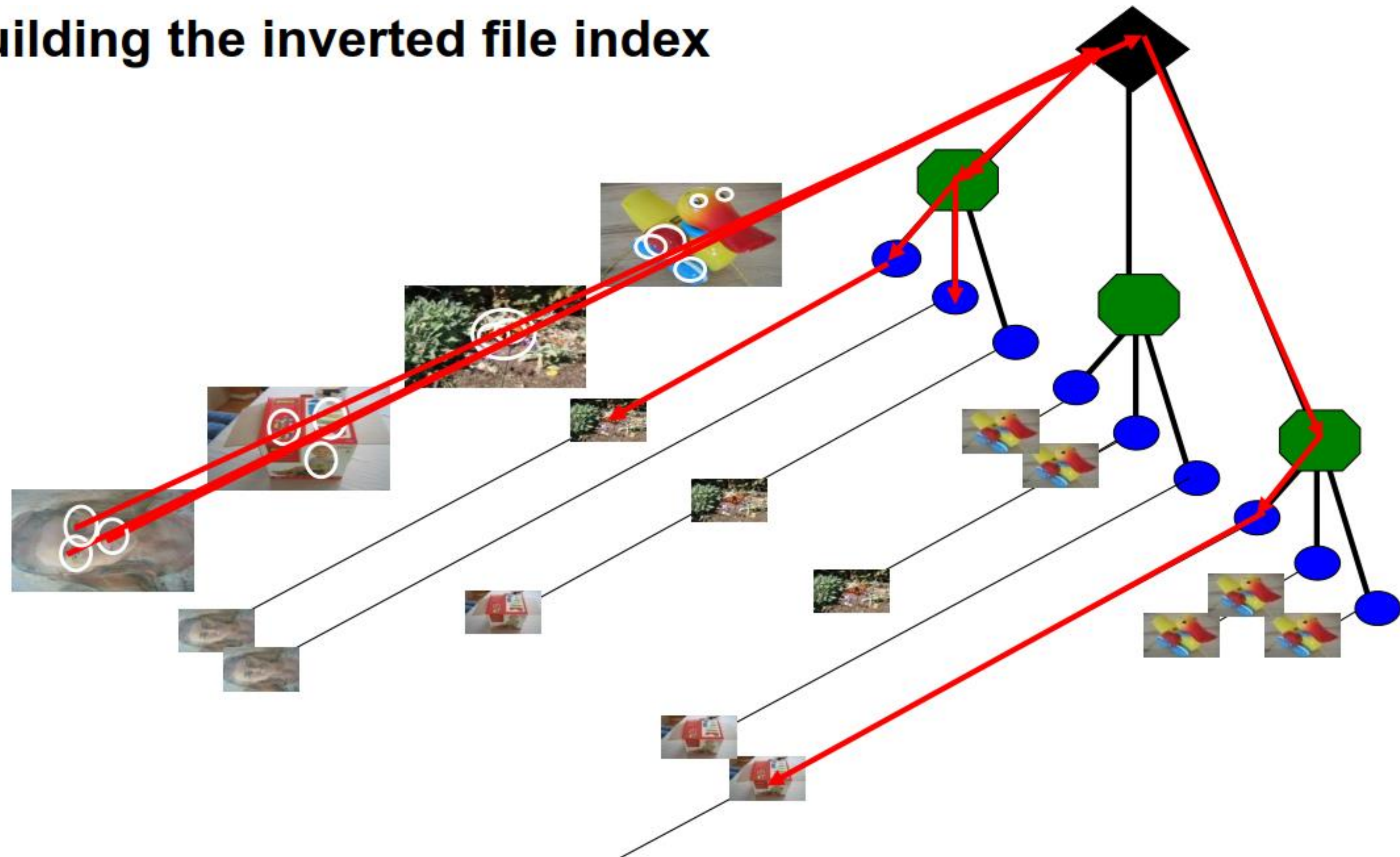





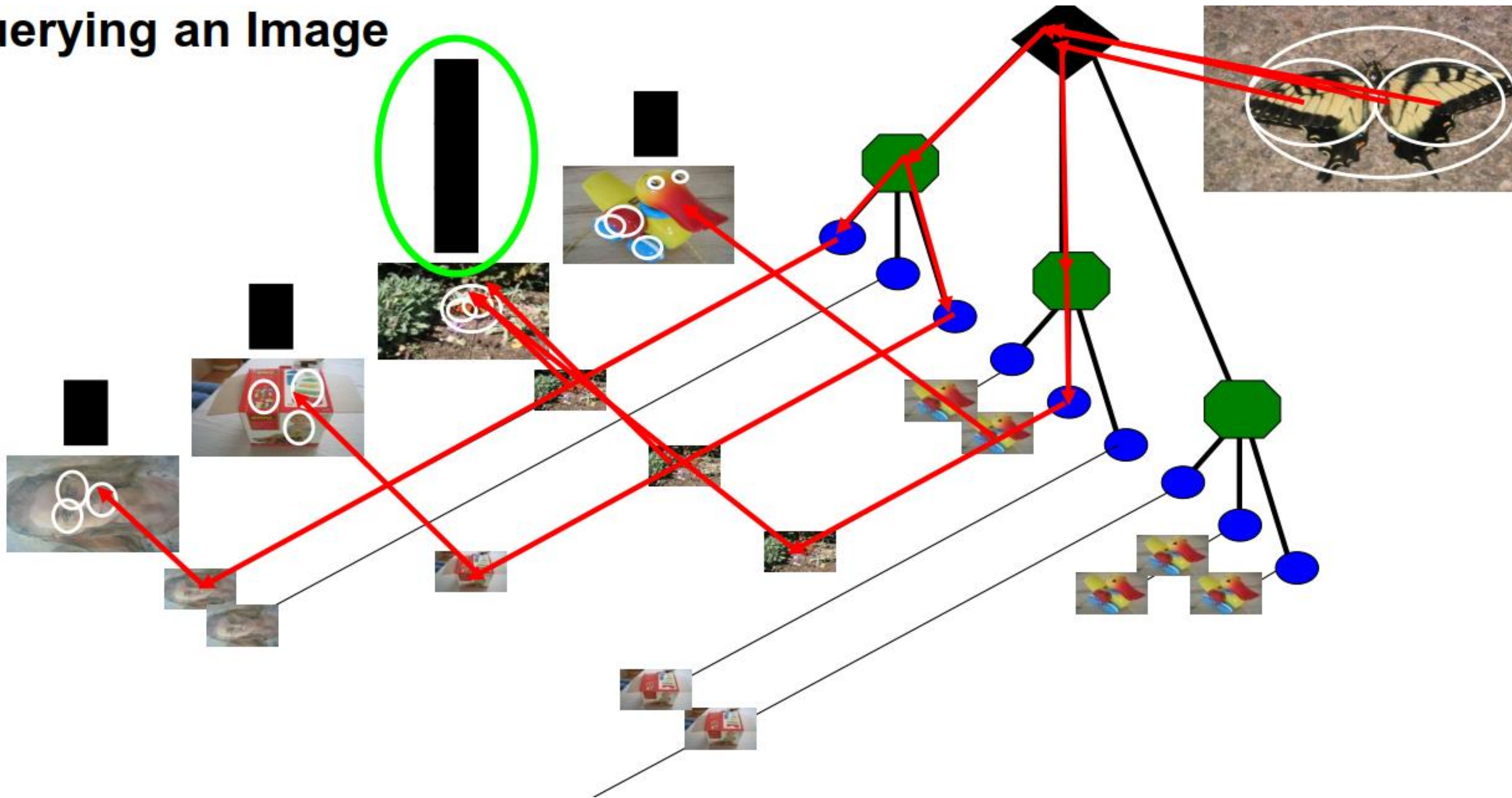
Recognition with K-tree – Populate the descriptor space



By Klustering



Querying an Image



FABMAP [Cummins and Newman IJRR 2011]

- Place recognition for robot localization
- Use training images to build the visual vocabulary
- At a new frame, compute:
  - P(being at a known place)
  - P(being at a new place)
- Captures the dependencies of words to distinguish the most characteristic structure of each scene (using the Chow-Liu tree)
- Binaries available online: http://www.robots.ox.ac.uk/~mjc/Software.htm

FABMAP example

robots.ox.ac.uk/~mjc/appearance_based_results.htm

(a) p=0.53 (b) p=0.30 (c) p=0.31

- p = probability of images coming from the same place

FABMAP example

robots.ox.ac.uk/~mjc/appearance_based_results.htm

(a) p=0.9989 (b) p=0.999 (c) p=0.999

- p = probability of images coming from the same place

Robust object/scene recognition

- Visual Vocabulary holds appearance information but discards the spatial relationships between features
- Two images with the same features shuffled around in the image will be a 100% match when using only appearance information.
- If different arrangements of the same features are expected then one might use **geometric verification**
  - Test the k most similar images to the query image for geometric consistency (e.g. using RANSAC)
  - Further reading (out of the scope of this course):
  - [Cummins and Newman, IJRR 2011]
  - [Stewenius et al, ECCV 2012]

Line extraction from images

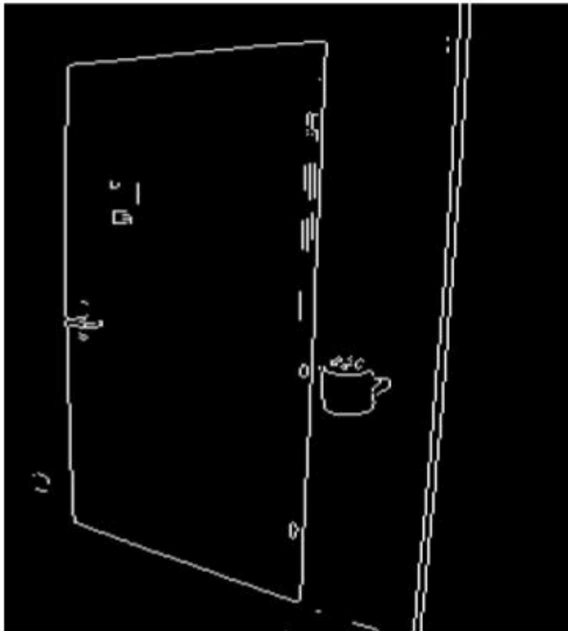
Suppose that you have been commissioned to implement a lane detection for a car driving-assistance system. How would you proceed?

Classical reference; Ernst D Dicksmanns: Dynamic vision for control of motion, Springer

Line extraction

How do we extract lines from edges? Two popular line extraction algorithms:

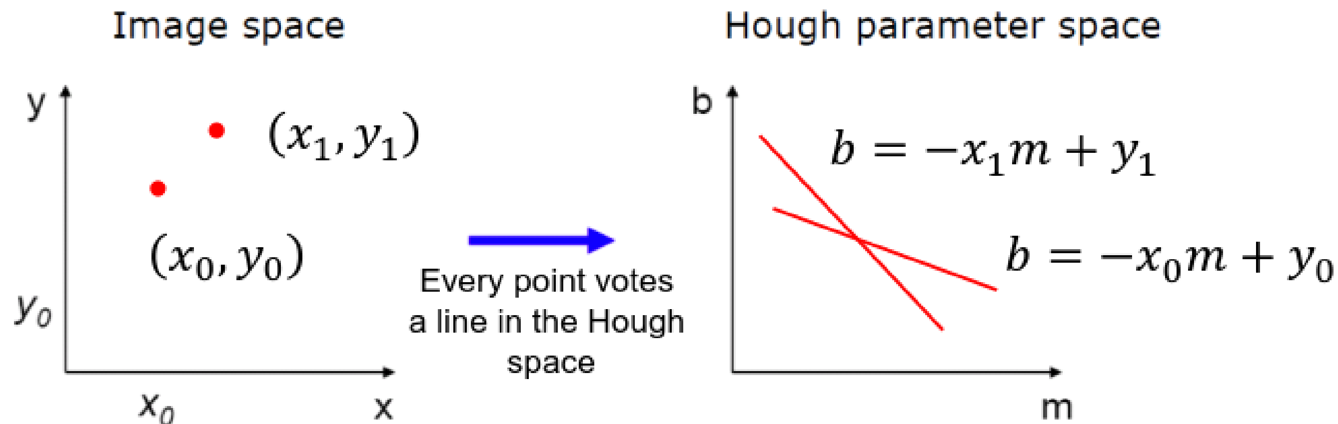
1. Hough transform (used also to detect circles, ellipses, and any sort of shape)
2. RANSAC (Random Sample Consensus)



Hough transform

How do we extract lines from edges? Two popular line extraction algorithms:

- Finds lines from a binary edge image using a voting procedure
- The voting space (or accumulator) is called Hough space



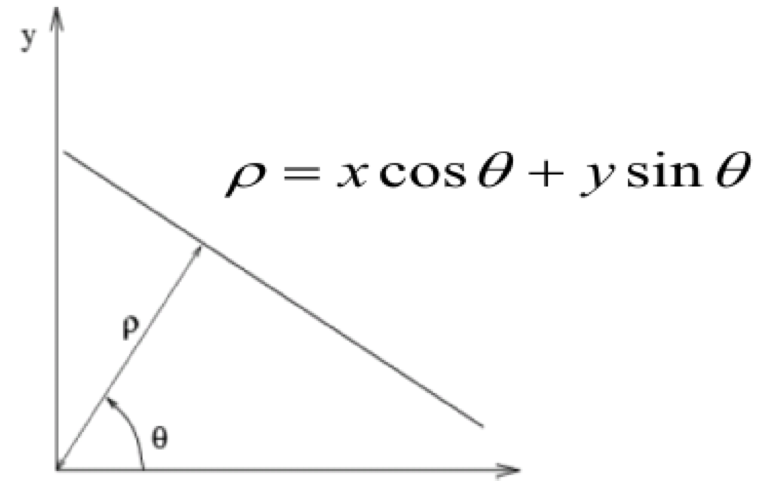
- Each point in image space, votes for line-parameters in Hough parameter space

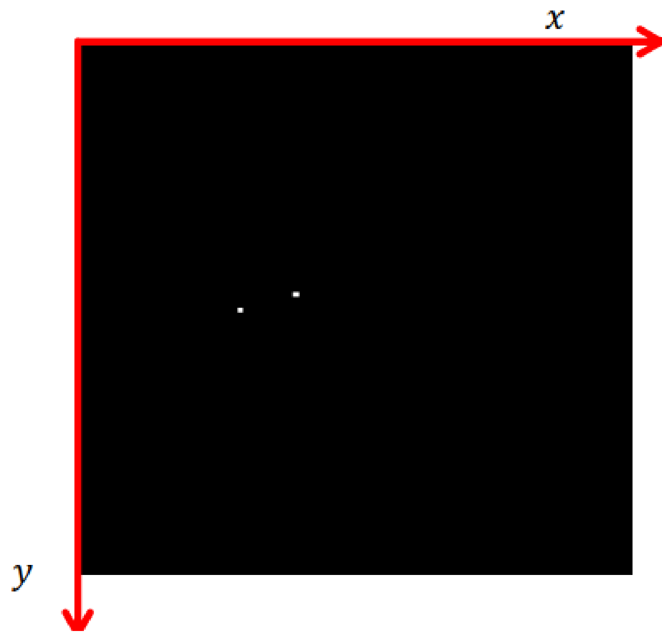
Hough transform

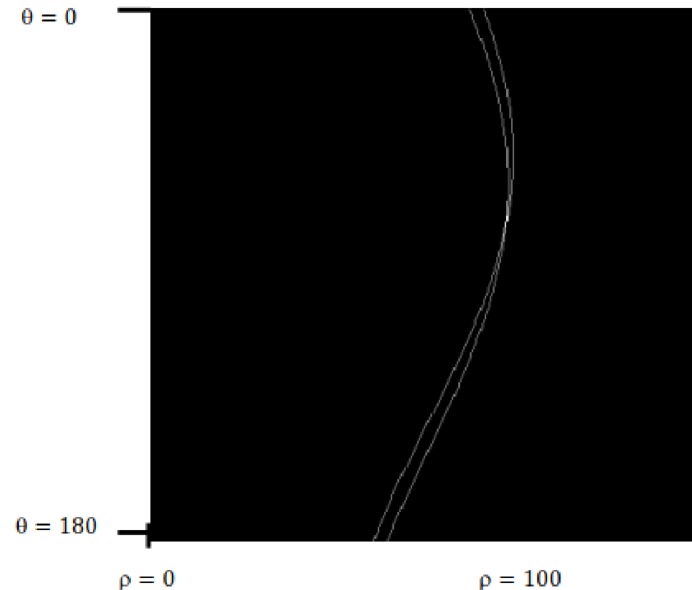
Problems with the (m, b) space:
Unbounded parameter domain

- m, b can assume any value in [−∞, +∞]

Alternative line representation: polar representation







Hough transform

1. Initialize: set all accumulator cells to zero

2. for each edge point (x,y) in the image

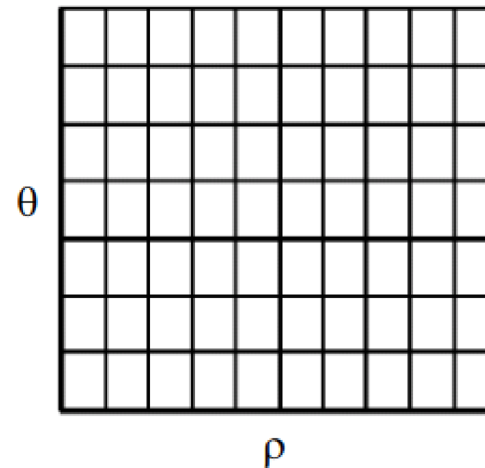
  for all θ in [0 : step : 180]

    Compute $\rho = x\cos\theta + y\sin\theta$

    $H(\theta,\rho) = H(\theta,\rho) + 1$

  end

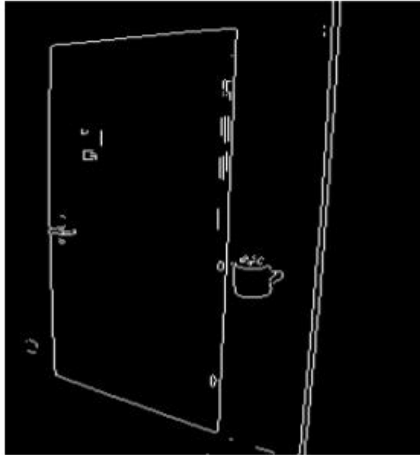
end

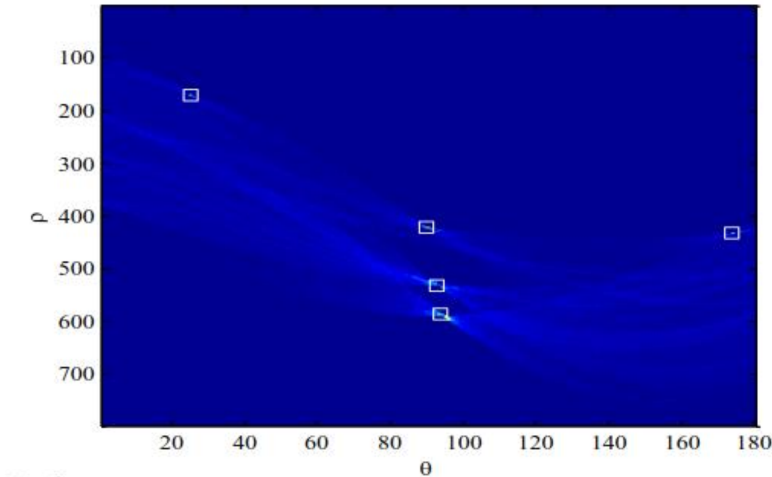


3. Find the values of (θ,ρ) where $H(\theta,\rho)$ is a local maximum

4. The detected line in the image is given by: $\rho = x\cos\theta + y\sin\theta$

Hough transform, examples





Hough Transform

Notice, however, that the Hough only find the parameters of the line, not the ends of it.

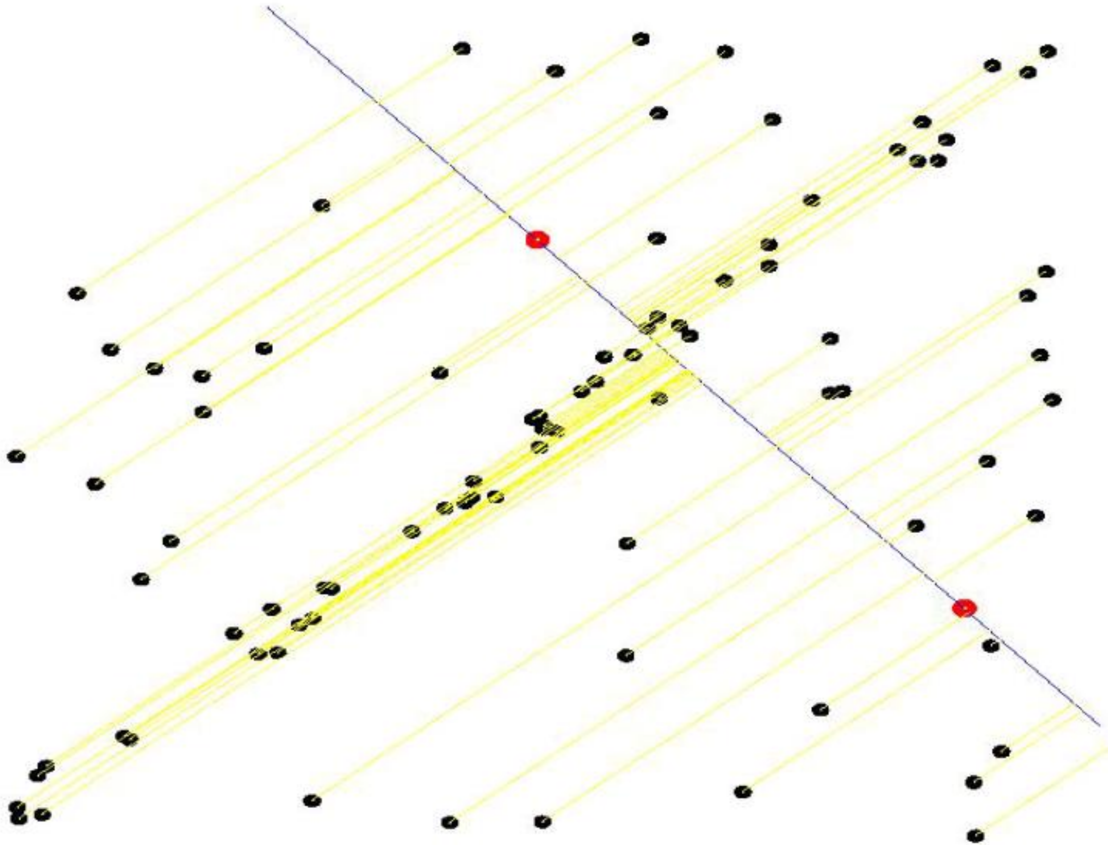
Hough is suitable for extracting other geometric forms having finite number of parameters, circles etc.

RANSAC (RAndom SAmple Consensus)

- RANSAC has become the standard method for model fitting in the presence of outliers (very noisy points or wrong data)
- It can be applied to line fitting but also to thousands of different problems where the goal is to estimate the parameters of a model from noisy data (e.g., camera calibration, structure from motion, DLT, homography, etc.)
- Let’s now focus on line extraction

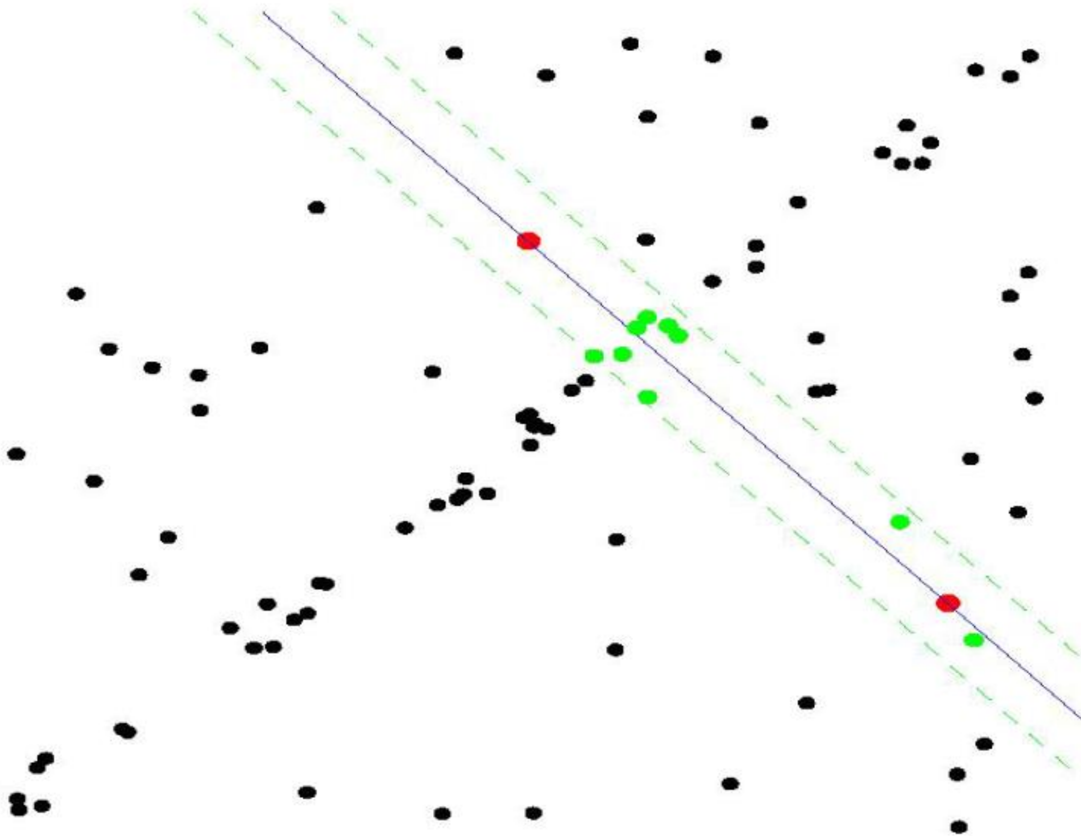
M. A.Fischler and R. C.Bolles. Random sample consensus: A paradigm for model fitting with applicatlons to image analysis and automated cartography. Graphics and Image Processing, 24(6):381–395, 1981.

RANSAC



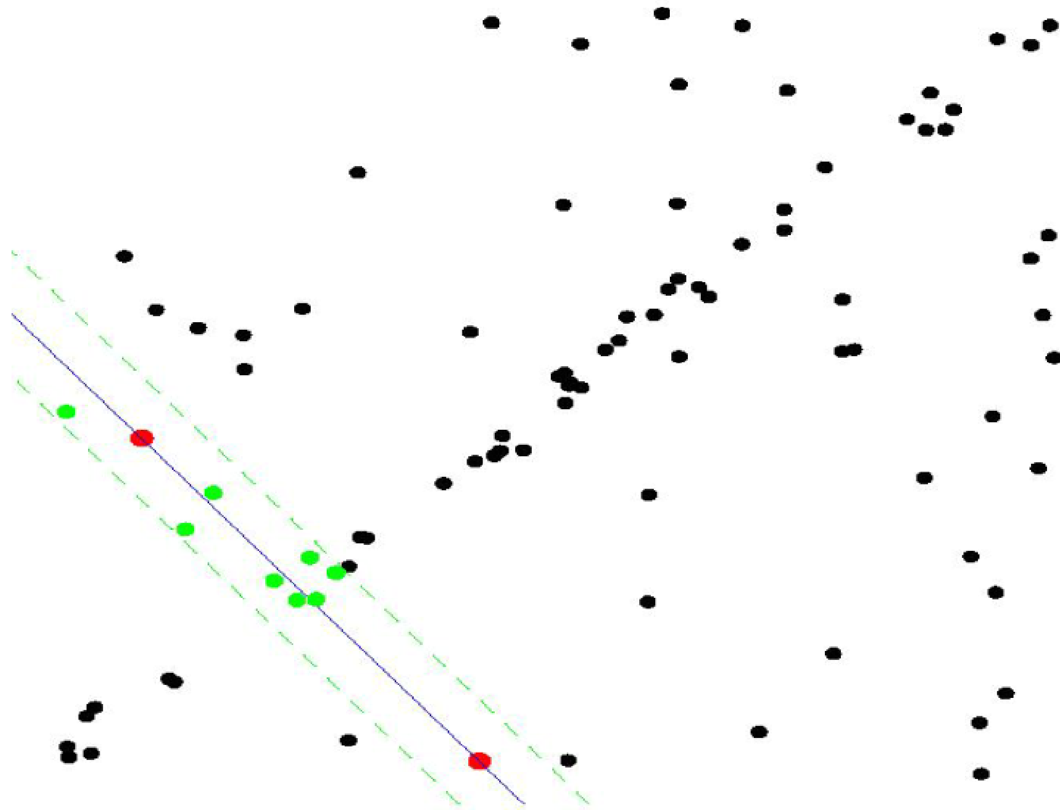
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- **Calculate error function for each data point**

RANSAC



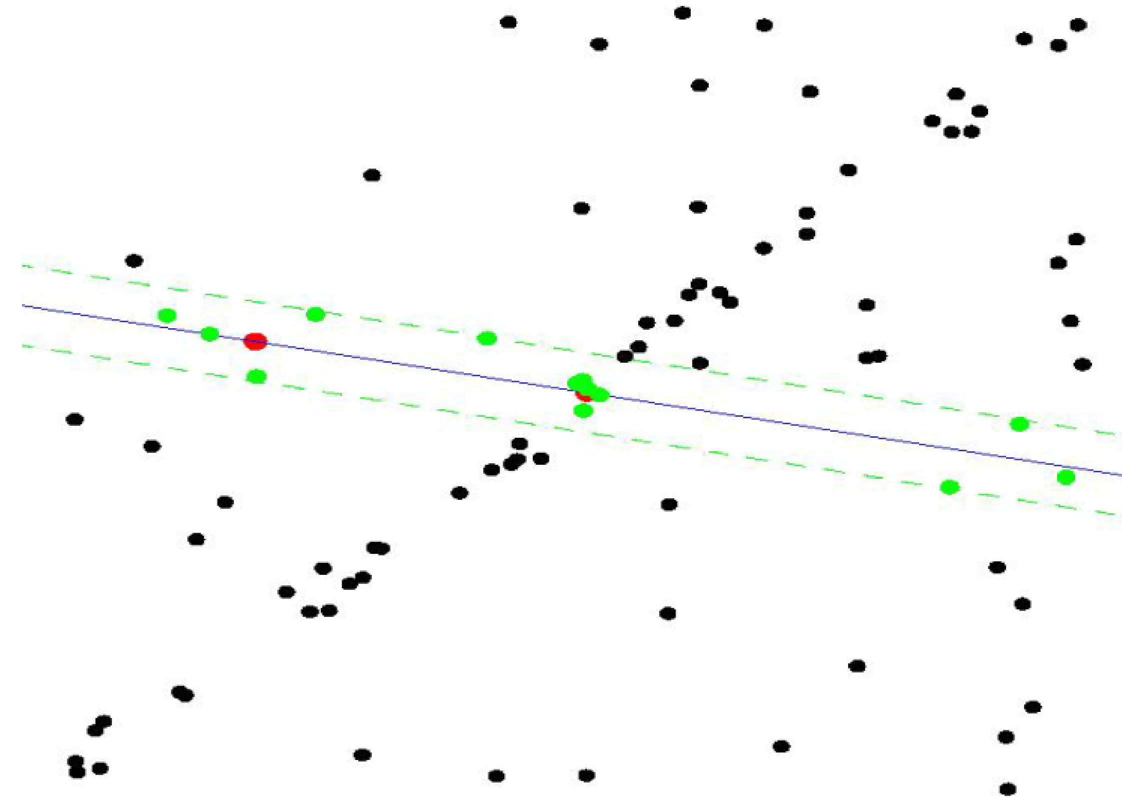
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- **Select data that supports current hypothesis**

RANSAC



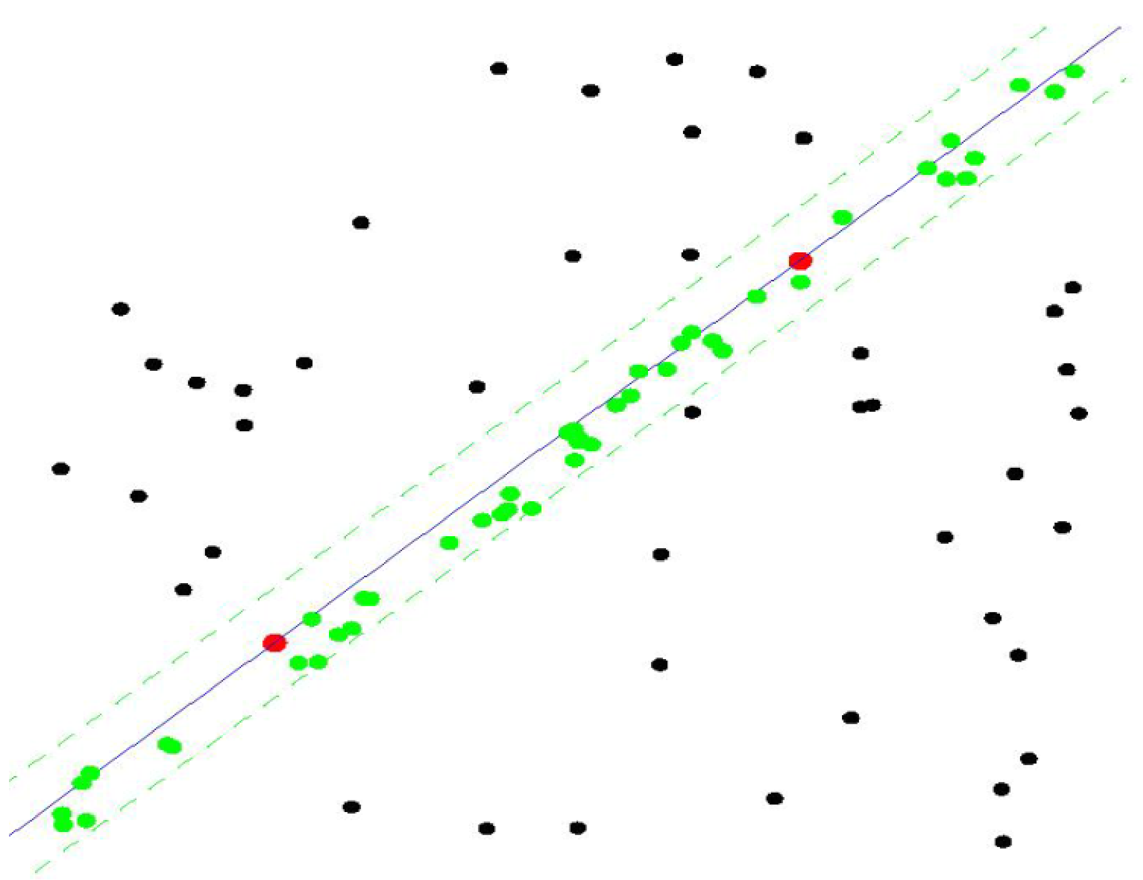
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that supports current hypothesis
- **Repeat sampling**

RANSAC



- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that supports current hypothesis
- **Repeat sampling**

RANSAC



Set with the maximum number of inliers obtained after k iterations

RANSAC

How many iterations does RANSAC need?

- Ideally: check all possible combinations of 2 points in a dataset of N points.
- Number of all pairwise combinations: N(N-1)/2

  => computationally unfeasible if N is too large.

  example: 10’000 edge points => need to check all 10’000*9999/2= 50 million combinations!
- Do we really need to check all combinations or can we stop after some iterations?
  – Checking a subset of combinations is enough if we have a rough estimate of the percentage of inliers in our dataset
- This can be done in a probabilistic way

RANSAC- Algorithm

Let A be a set of N points

1. **repeat**
2. Randomly select a sample of 2 points from A
3. Fit a line through the 2 points
4. Compute the distances of all other points to this line
5. Construct the inlier set (i.e. count the number of points whose distance < d)
6. Store these inliers
7. **until** maximum number of iterations k reached
8. **The set with the maximum number of inliers is chosen as a solution to the problem**

RANSAC is really robust in eliminating outlayers.

Typical applications in robotics are: line extraction from 2D range data, plane extraction from 3D data, feature matching, structure from motion, camera calibration, homography estimation, etc.

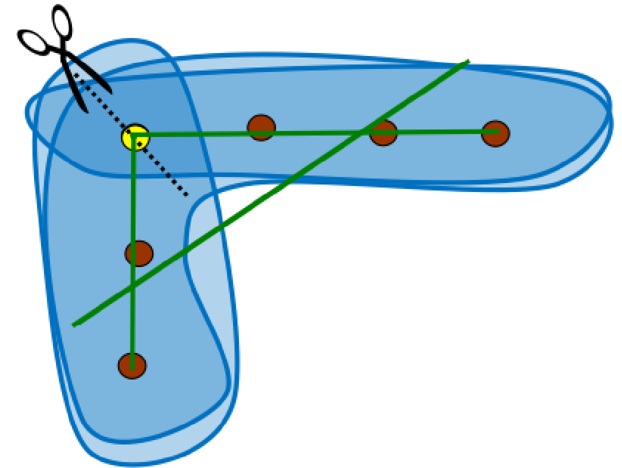
Algorithm 1: Split-and-Merge (standard)

- Popular algorithm, originates from Computer Vision.
- A recursive procedure of fitting and splitting.
- A slightly different version, called Iterative end-point-fit, simply connects the end points for line fitting.

Let **S** be the set of all data points

Split

- Fit a line to points in current set S
- Find the most distant point to the line
- If distance > threshold => split set & repeat with left and right point sets

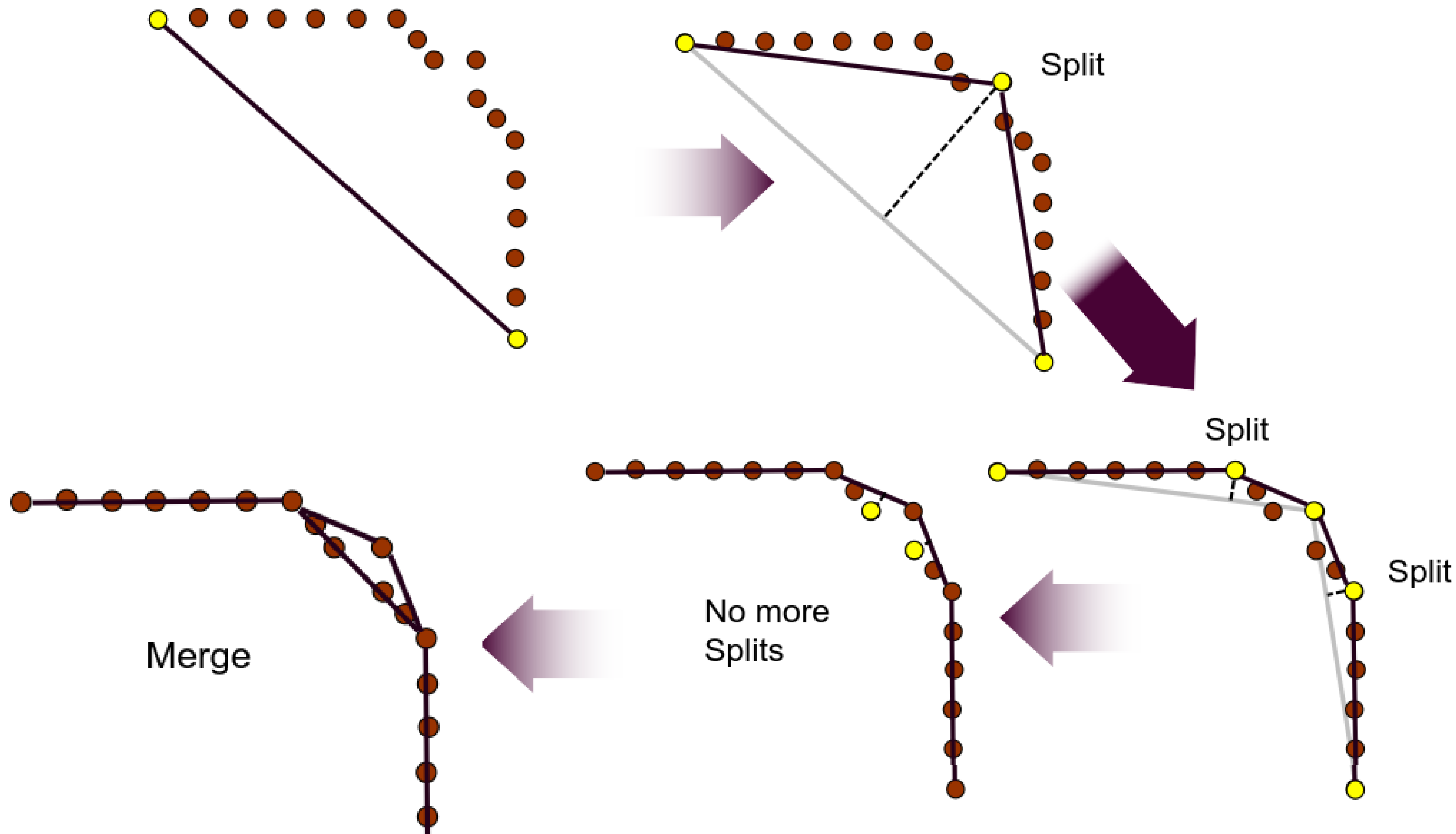


Merge

- If two consecutive segments are collinear enough, obtain the common line and find the most distant point
- If distance <= threshold, merge both segments

Algorithm 1: Split-and-Merge (iterative end-point-fit)

- Iterative end-point-fit: simply connects the end points for line fitting

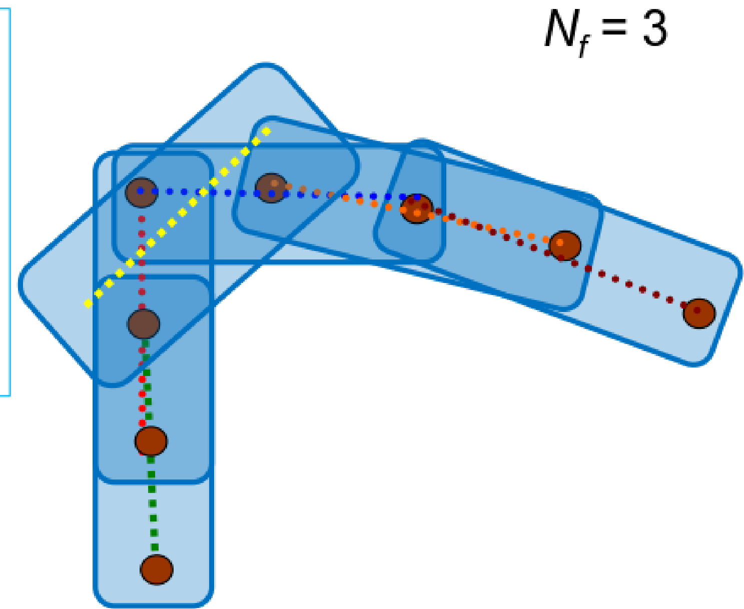


Algorithm 2: Line-Regression

- "Sliding window" of size Nf points
- Fit line-segment to all points in each window

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Merge overlapping line segments + recompute line parameters for each segment



Algorithm 2: Line-Regression

- "Sliding window" of size N_f points
- Fit line-segment to all points in each window

Line-Regression

- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Merge overlapping line segments + recompute line parameters for each segment