V4RL
VISION FOR ROBOTICS LAB

Perception IV:
Place Recognition & Line Extraction

Autonomous Mobile Robots

Margarita Chli
Martin Rufli, Roland Siegwart

Today's Lecture

- From object recognition to scene/place recognition
  Section **4.6**
  - The "Bag of Words" approach
  - Building & using the Vocabulary Tree
  - FABMAP & other place recognition methods from the state of the art
- Uncertainties: (mainly) Section **4.7**
  - Representation + Propagation – Section **4.1.3**
  - Line extraction from a point cloud
    - Split-and-merge
    - Line-Regression
    - RANSAC
    - Hough Transform

Optional Reading:

"Video Google", J. Sivic and A. Zisserman, ICCV 2003

"Scalable Recognition with a Vocabulary Tree", D. Nistér and H. Stewénius, CVPR 2006.

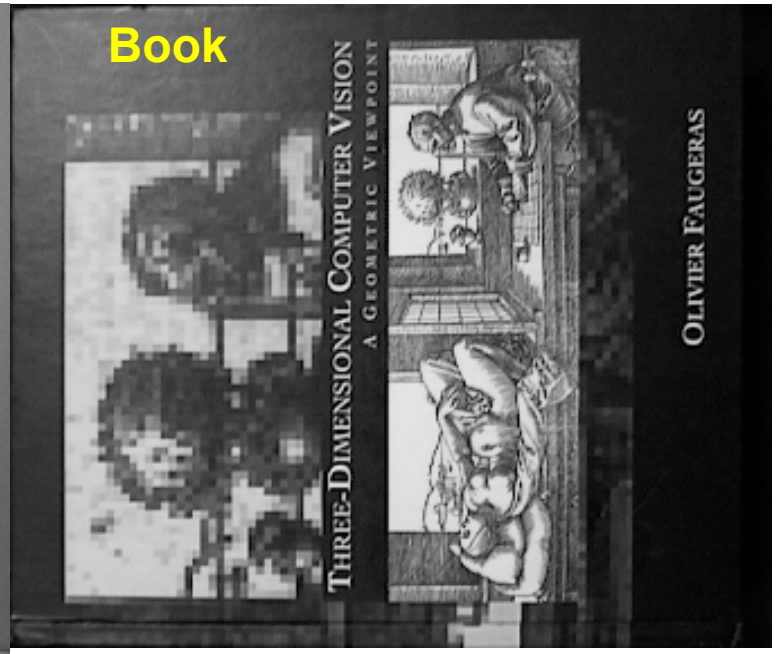
"FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance", M. Cummins and P. Newman, IJRR 2008.

"SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights", M. Milford and G. Wyeth, ICRA 2012

"Bags of Binary Words fro Fast Place Recognition in Image Sequences", D. Gálvez-López and Tardos, TRO 2012

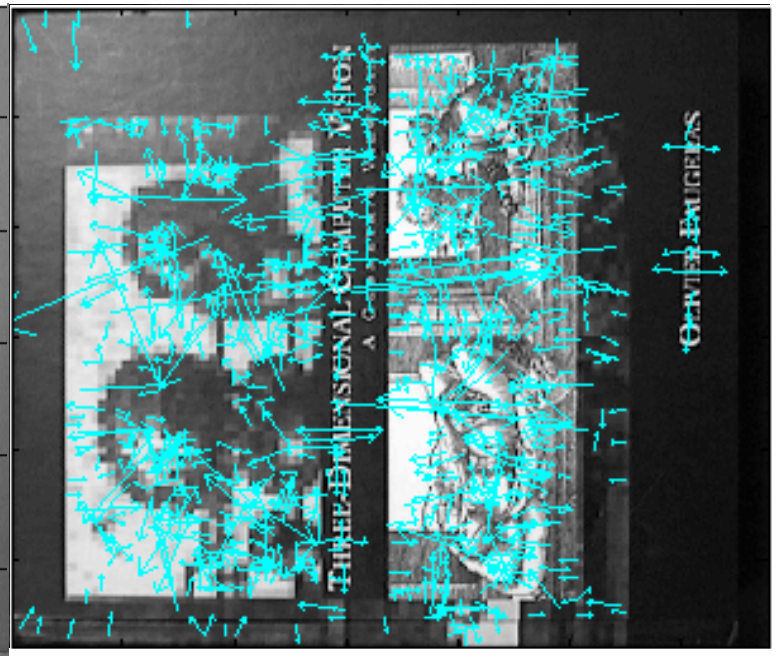
Object recognition

Q: Is this Book present in the Scene?



Object recognition | with SIFT features

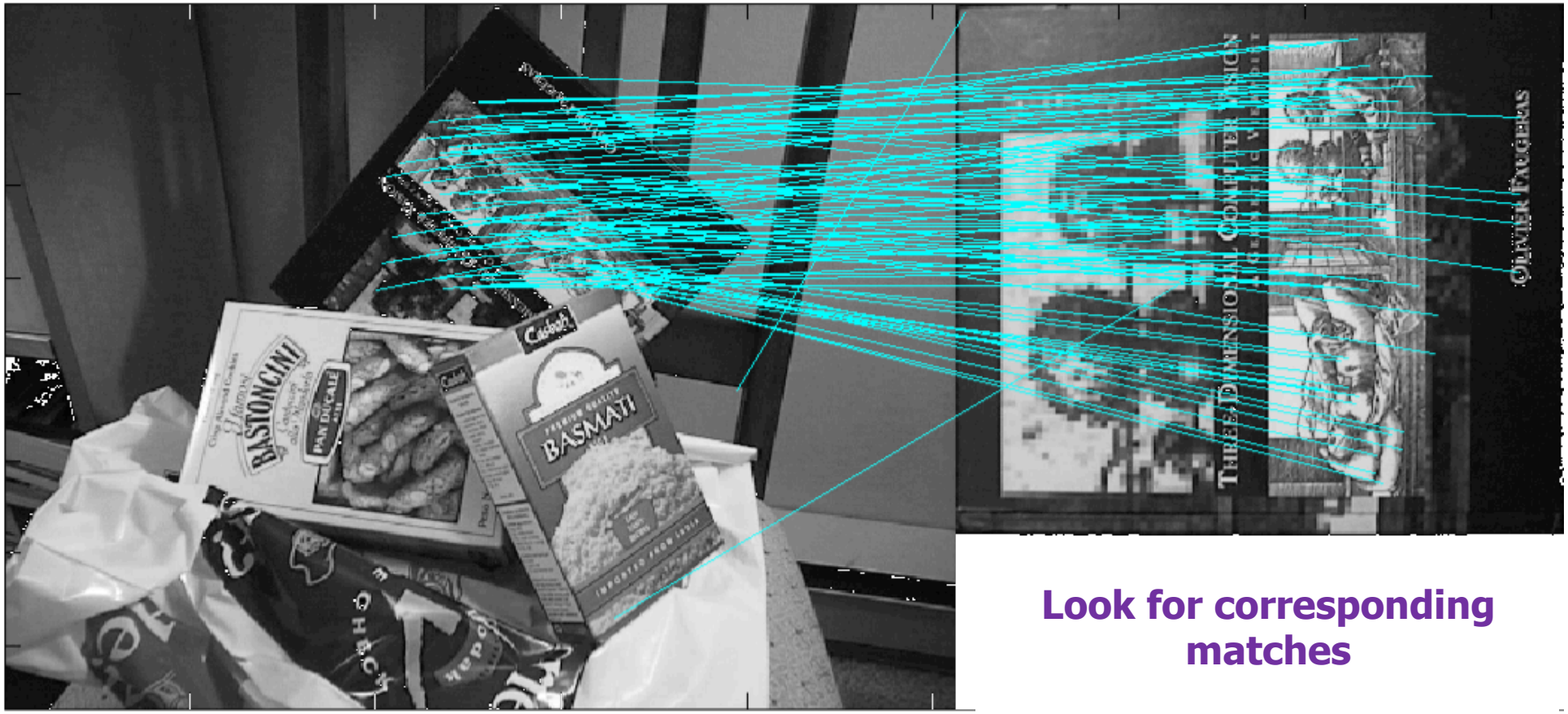
Q: Is this Book present in the Scene?



Extract keypoints in both images

Object recognition | with SIFT features

Q: Is this Book present in the Scene?



Look for corresponding matches

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

⇨ A: The Book is present in the Scene

Object recognition | taking this a step further…

- Find an object in an image

- Find an object in multiple images

- Find multiple objects in multiple images

BoW: "Bag of Words" for image retrieval

- Extension to scene/place recognition:
  - *Is this image in my database?*
  - Robot: Have I been to this place before?

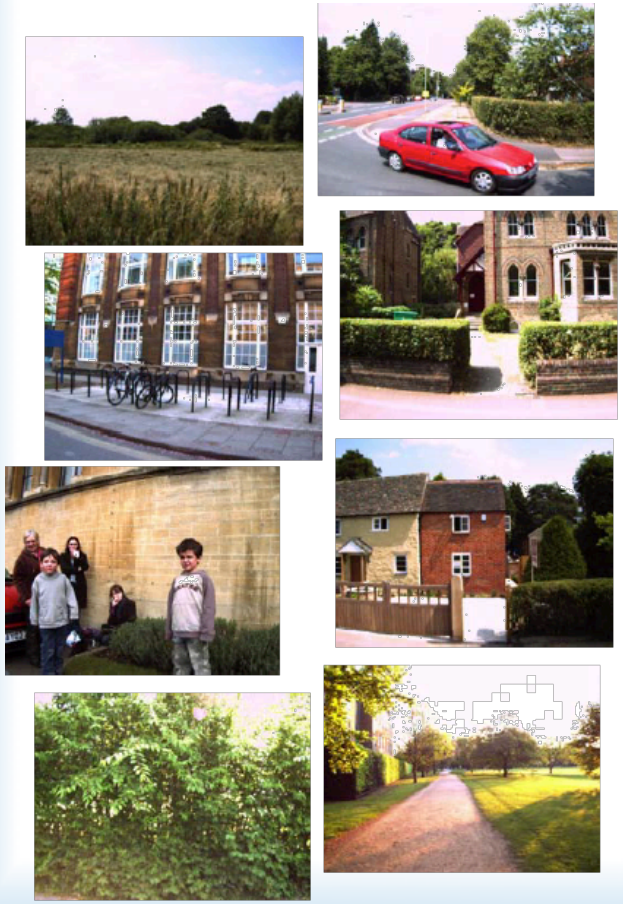
  ⇨ '**loop closure**' problem, '**kidnapped robot**' problem

- Use analogies from text retrieval:
  - Visual Words
  - Vocabulary of Visual Words
  - "Bag of Words" (BoW) approach

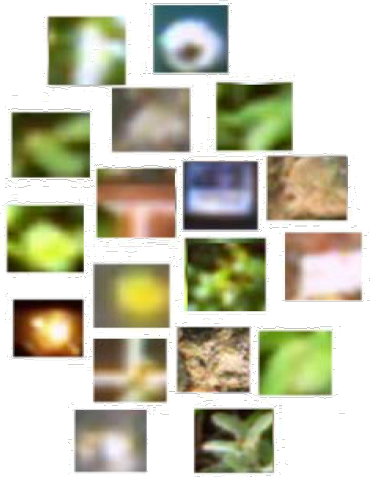
BoW | building the visual vocabulary

Images for this slide are courtesy of Mark Cummins

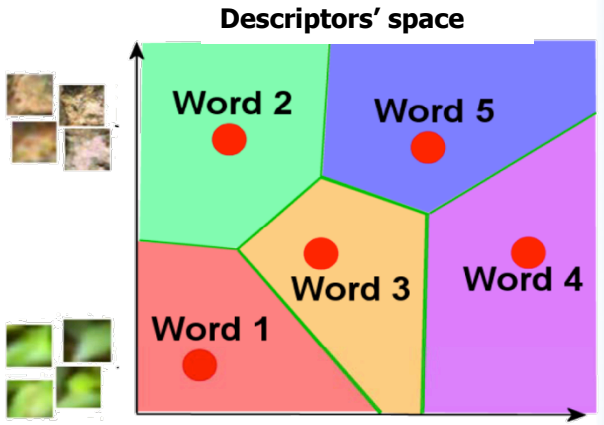
Image Collection



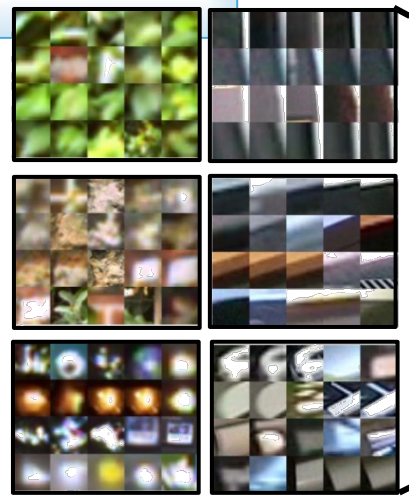
Extract Features

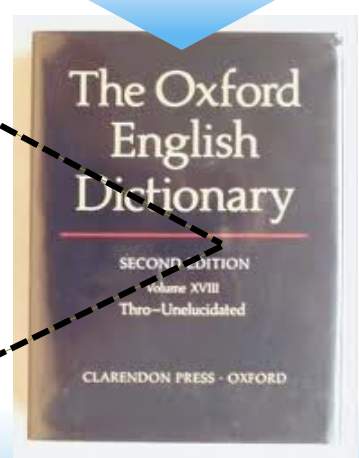


Cluster Descriptors



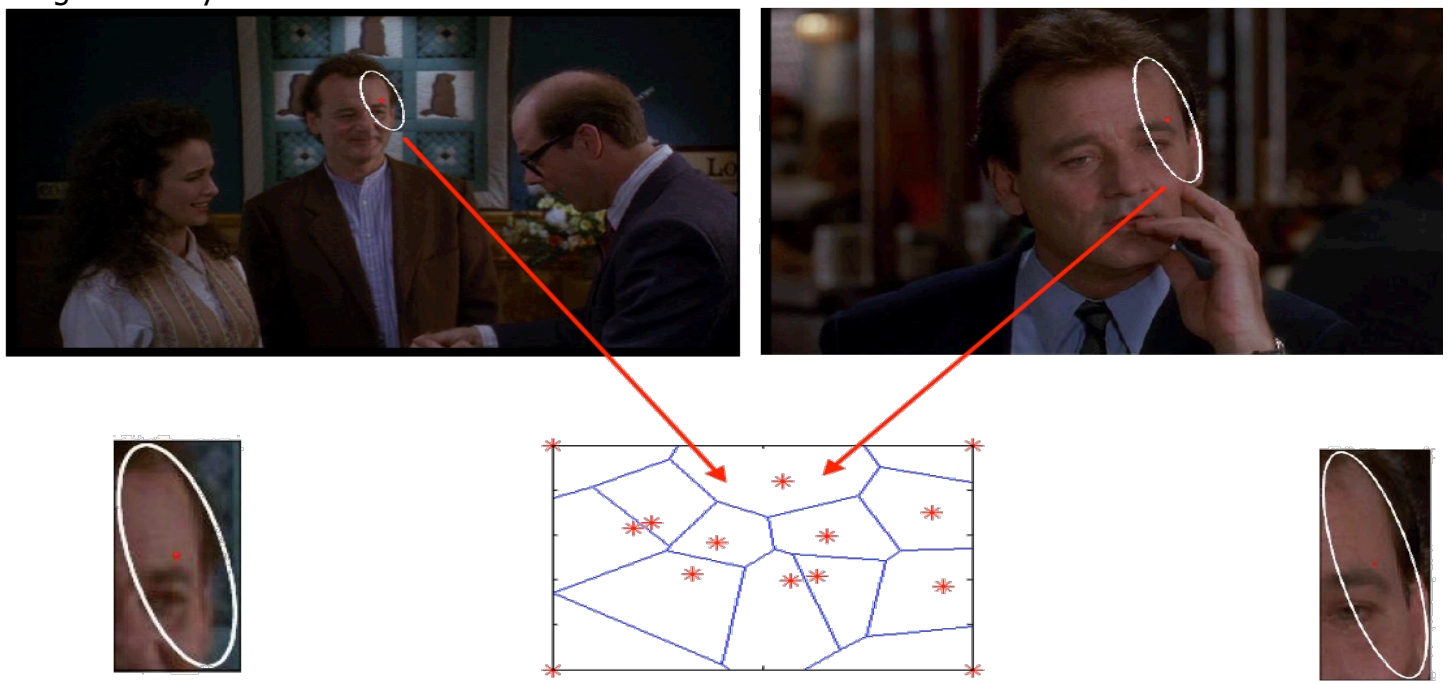
Examples of Visual Words:





BoW | Video Google: one image a thousand words?

Image courtesy of Andrew Zisserman



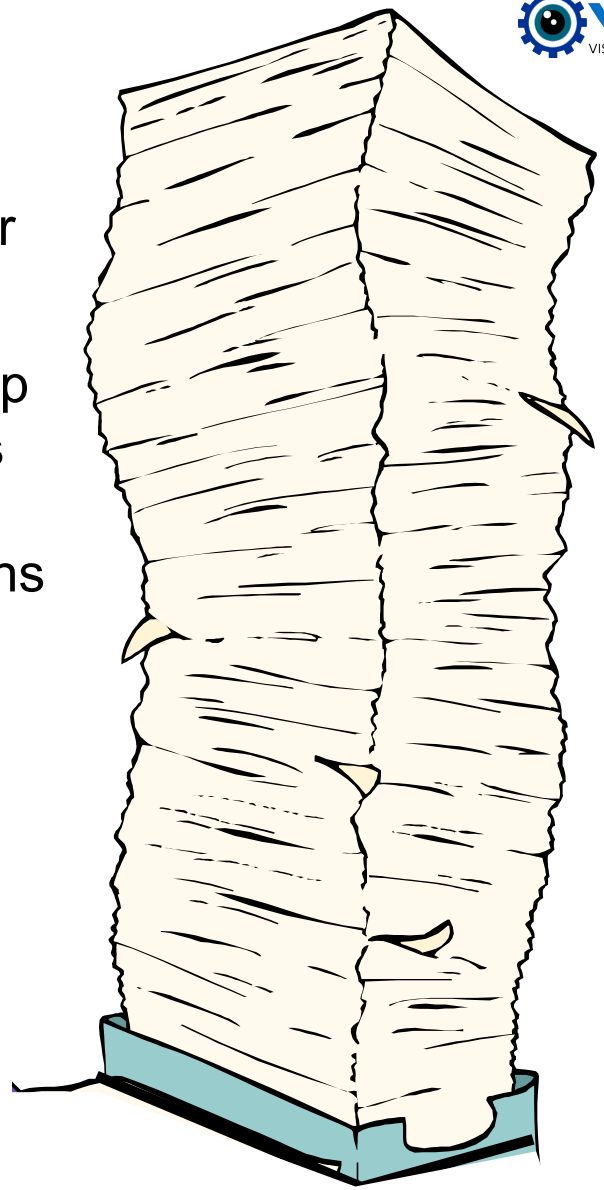
These features map to the same visual word

- "Video Google" [J.Sivic and A. Zisserman, ICCV 2003]
- Demo:

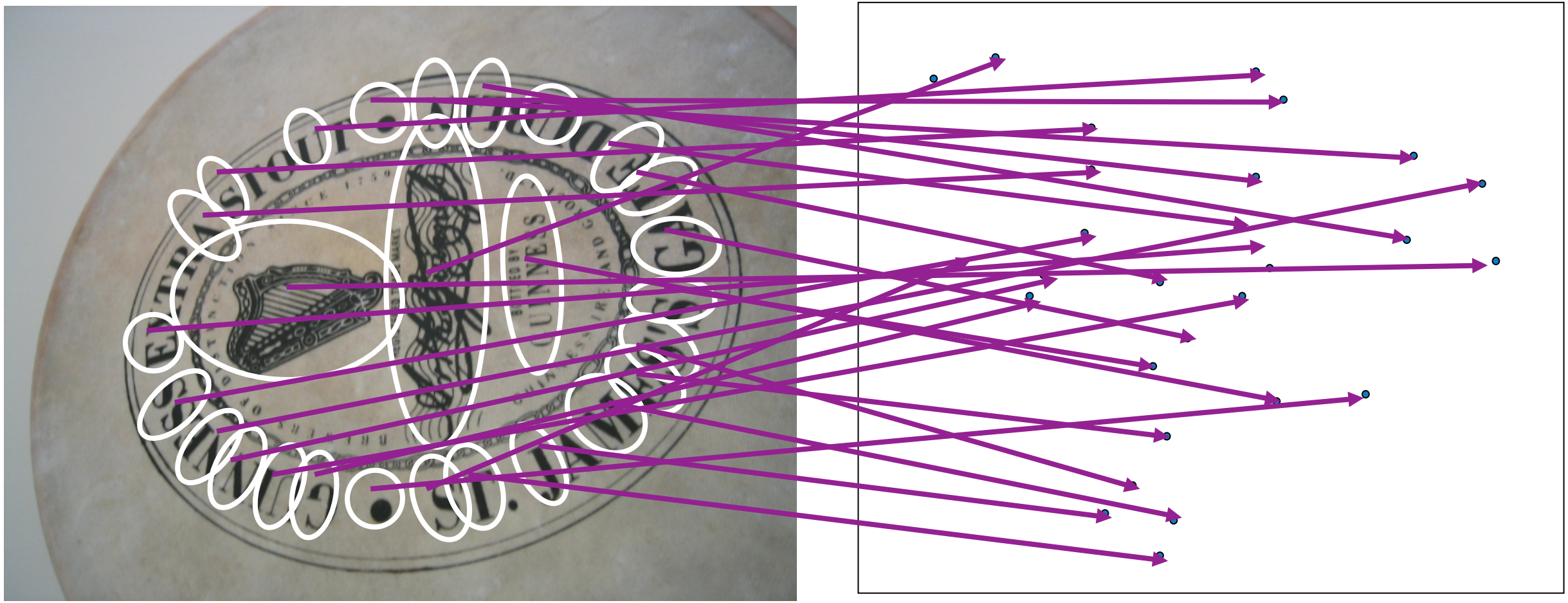
BoW | efficient image retrieval

- **Image retrieval** represents a more general problem of object or place recognition.
- We can describe a **scene as a collection of words** and look up in the database for **images with a similar collection of words**
- What if we need to find an object/scene in a database of millions of images?
  - Build **Vocabulary Tree** via hierarchical clustering
  - Use the **Inverted File system**:
    a way of efficient indexing
    (each node in the tree is associated with a list of images containing an instance of this node)

[Nistér and Stewénius, CVPR 2006]

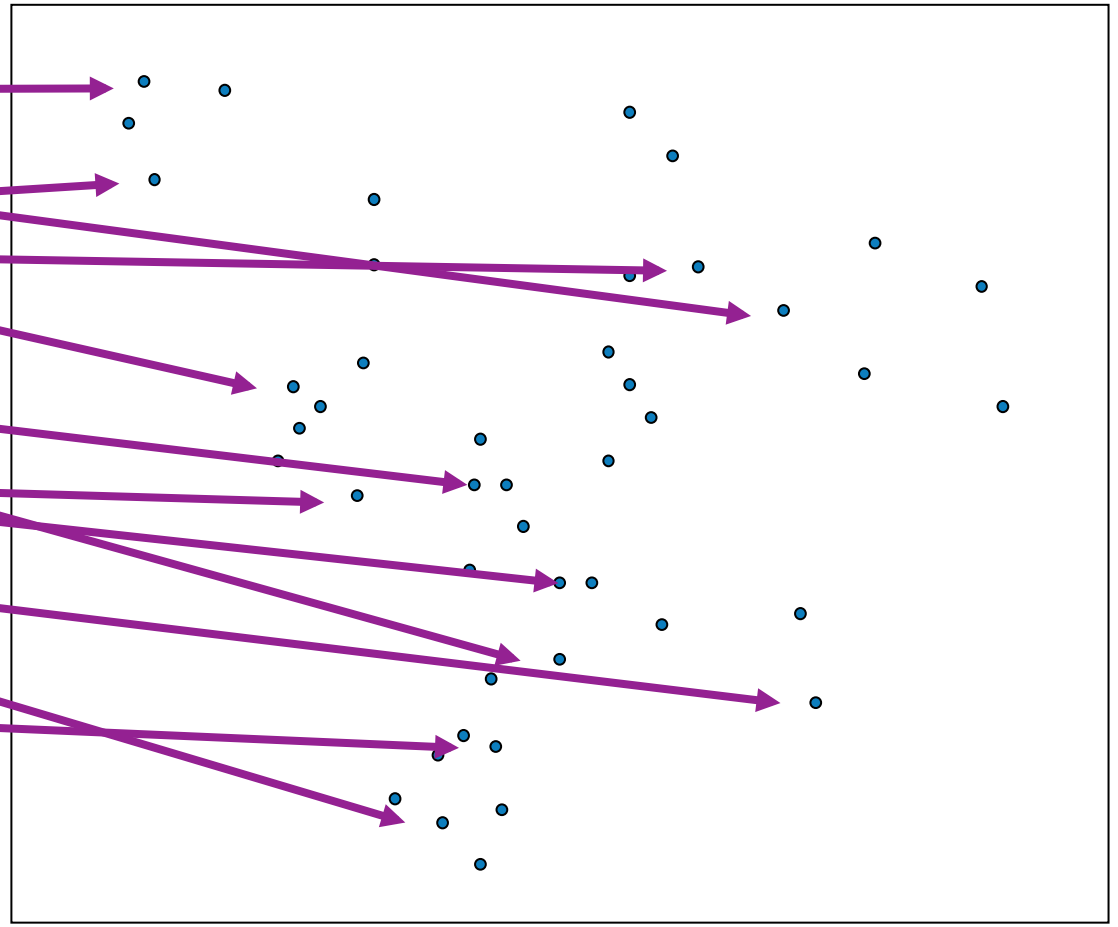


Vocabulary tree | extract features



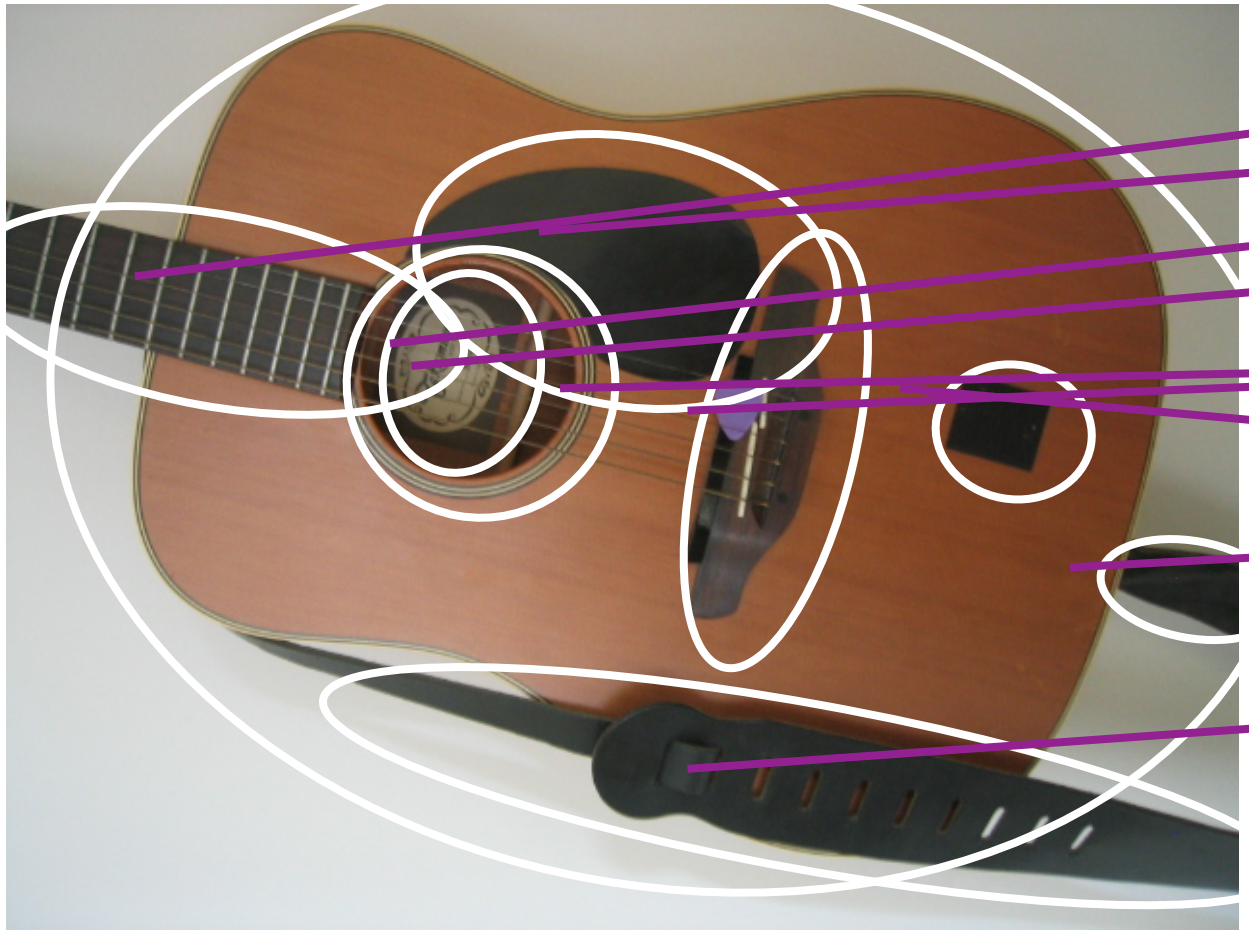
Based on D. Nister's slides

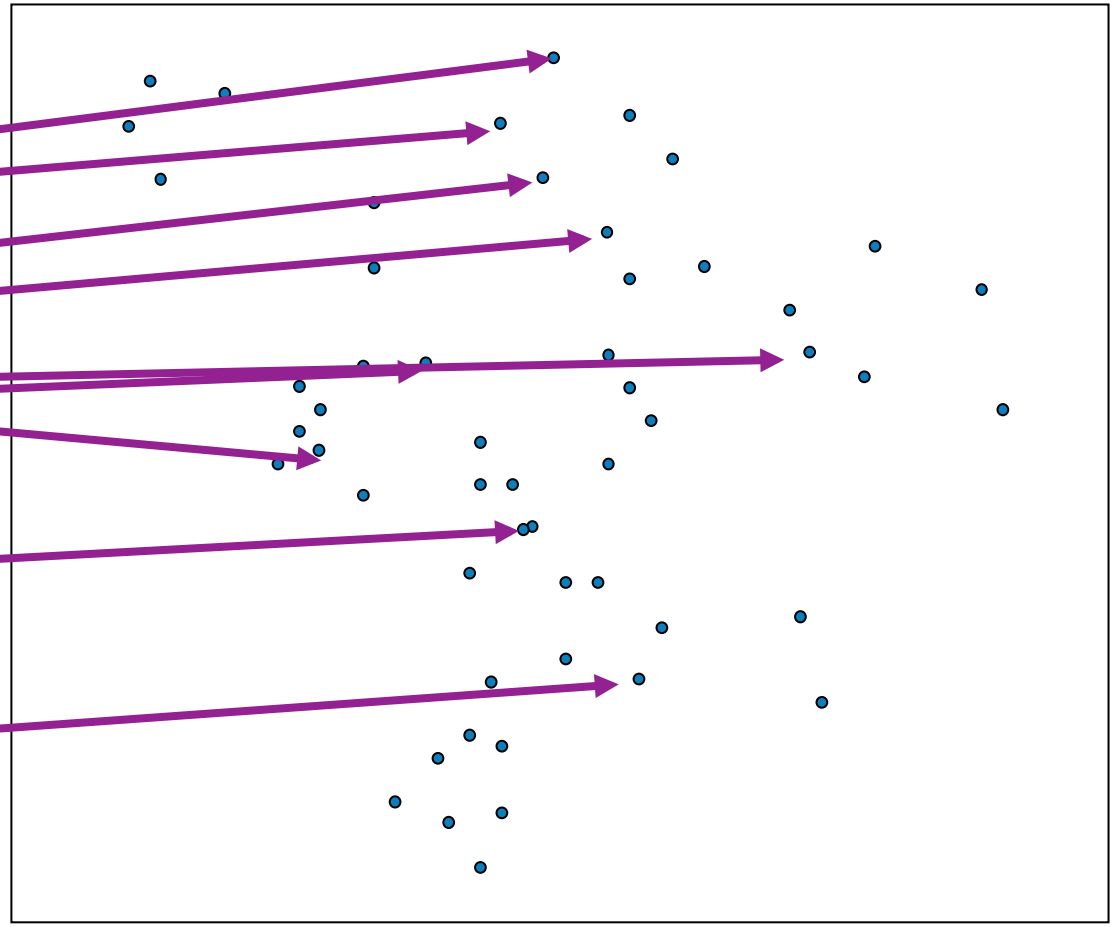
Vocabulary tree | extract features



Based on D. Nister's slides

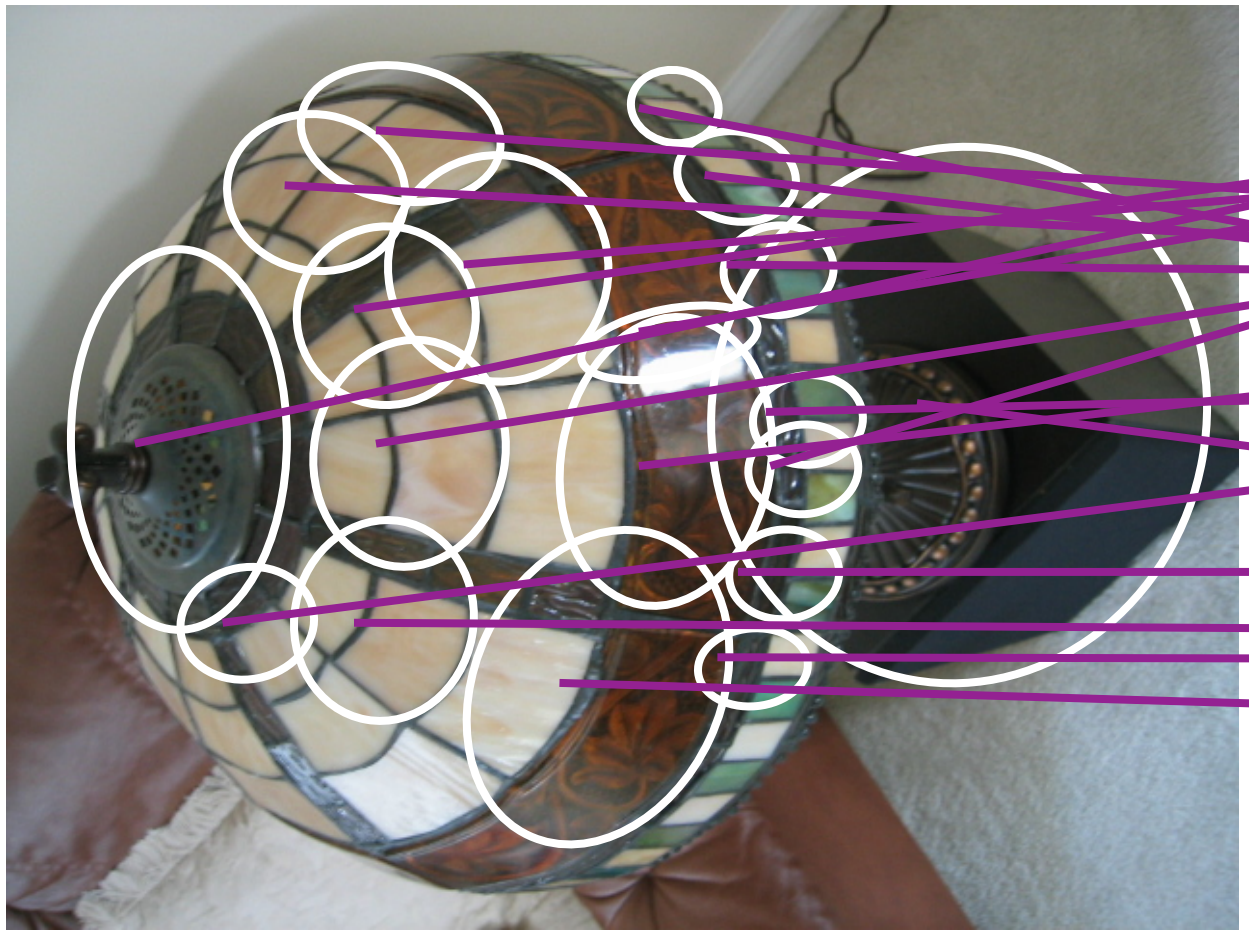
Vocabulary tree | extract features

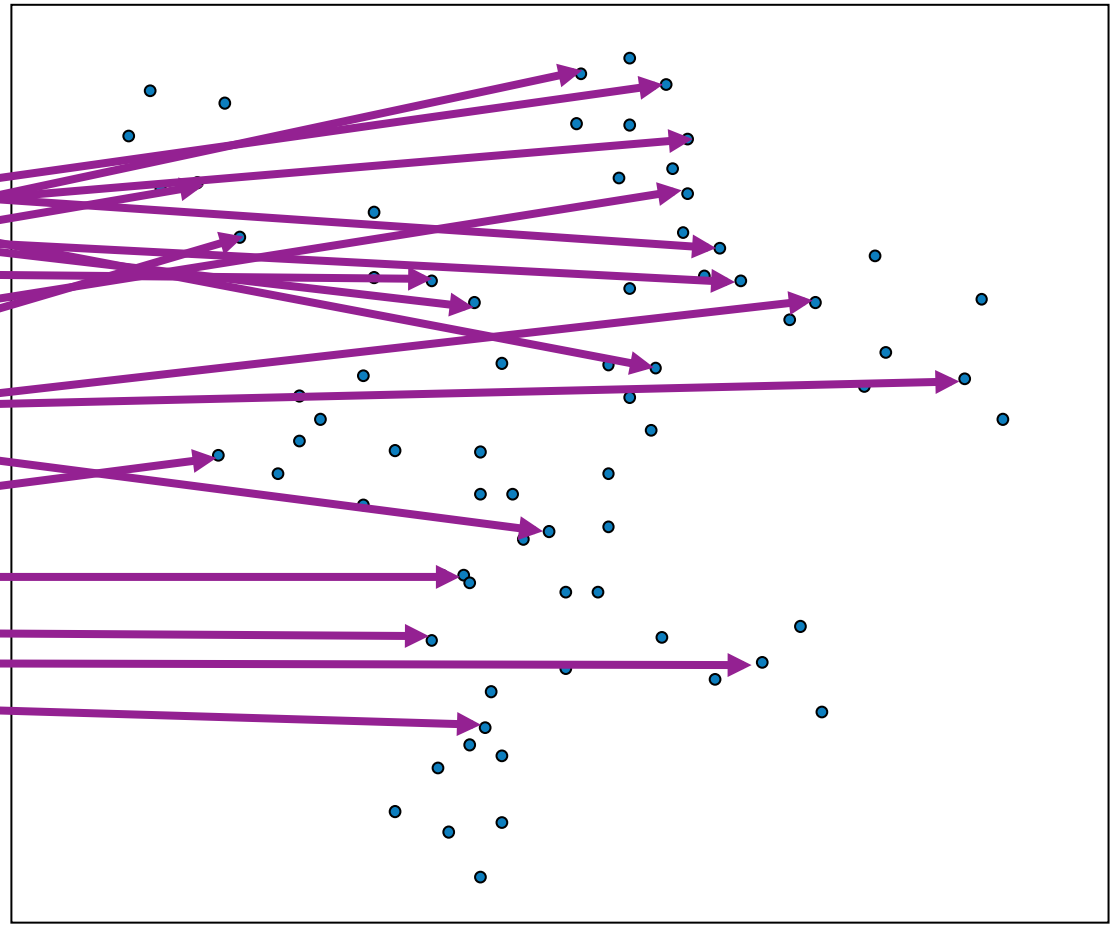




Based on D. Nister's slides

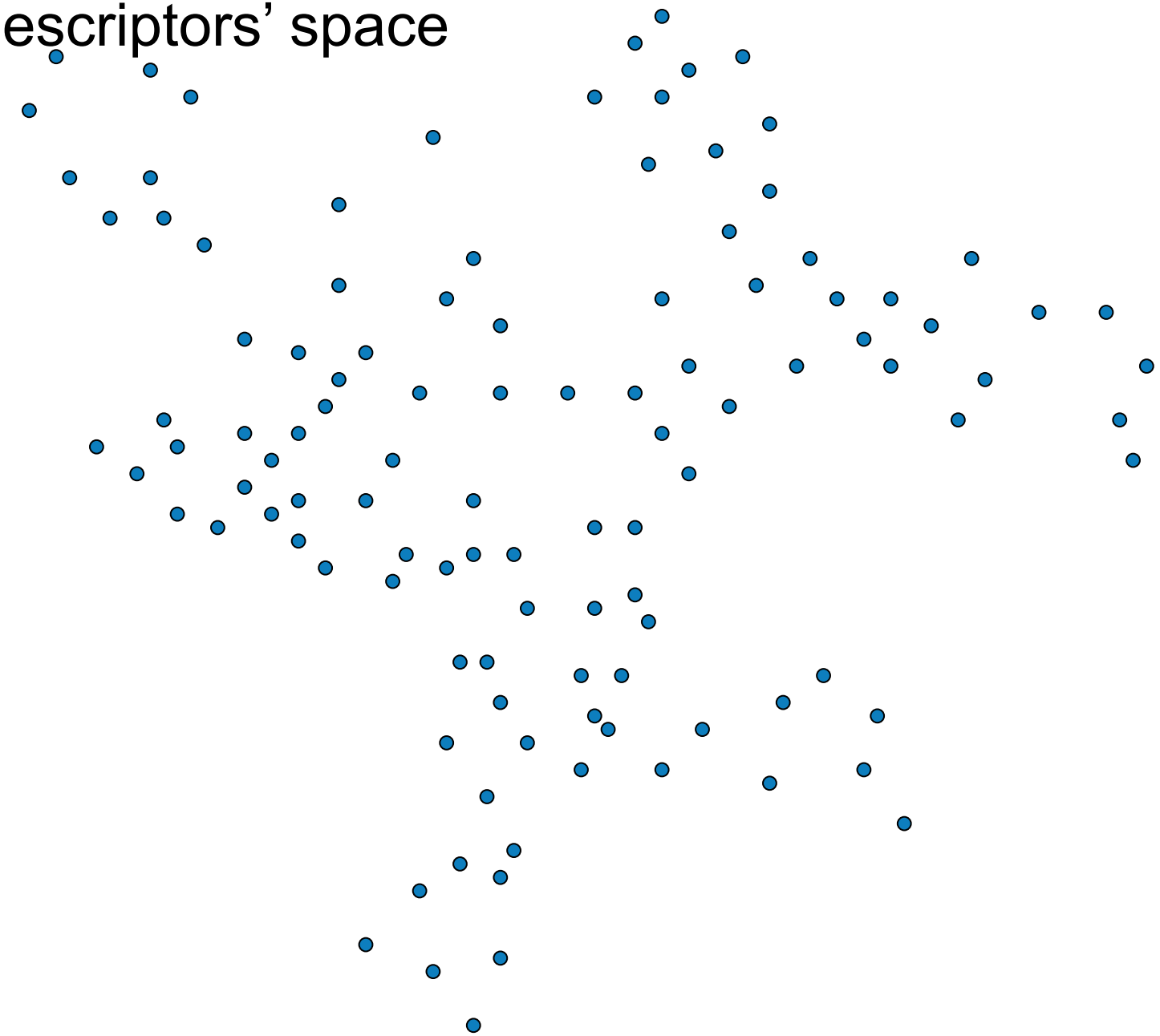
Vocabulary tree | extract features





Based on D. Nister's slides

Vocabulary tree | descriptors' space



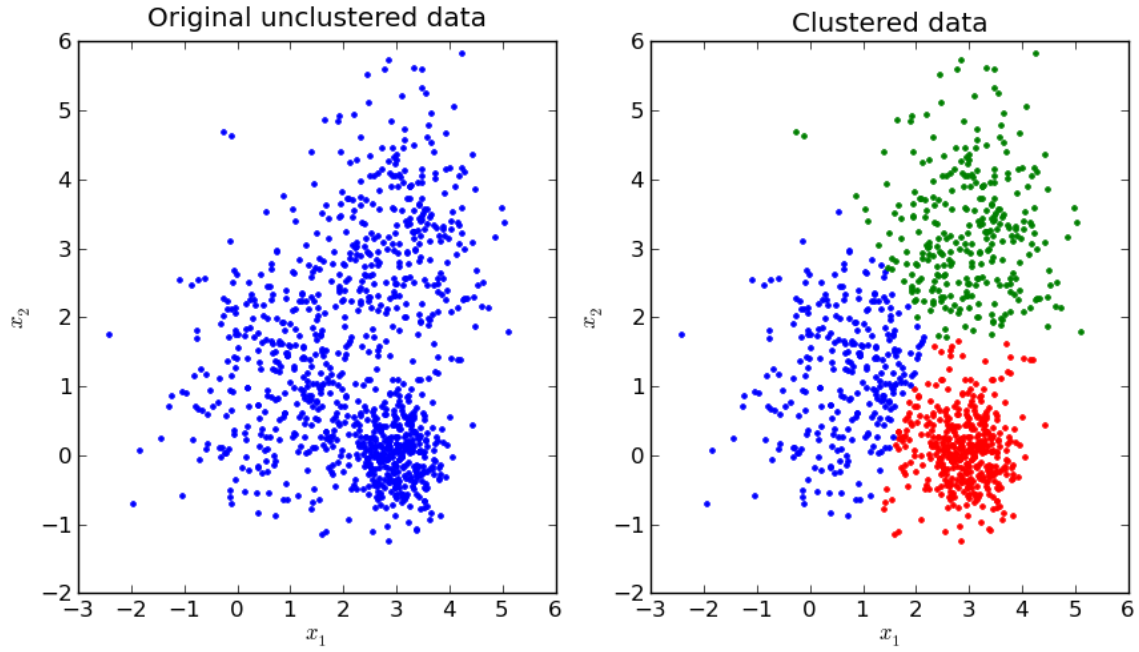
Based on D. Nister's slides

Vocabulary tree | k-means clustering, review

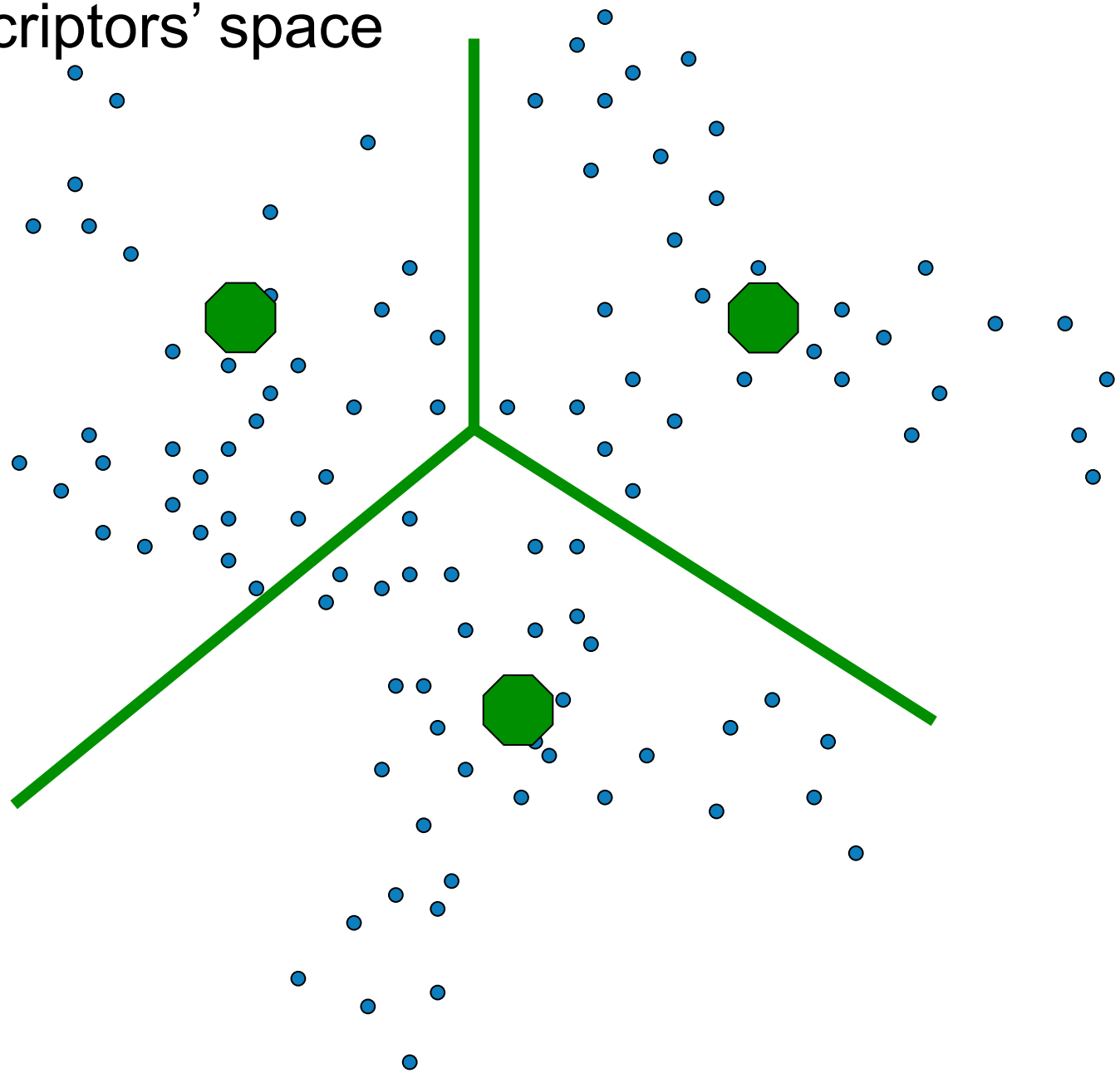
- **k-means clustering:** partitions a point cloud into ***k*** clusters, such that each point belongs to one cluster
- Minimizes the Sum of Squared Euclidean Distances between points and their nearest cluster-centers

Algorithm:

- `Randomly initialize` ***k*** `cluster centers`
- `until (convergence) do:`
  - `assign each data-point to its nearest cluster-center`
  - `Re-compute each cluster-center as the mean of all points assigned to each cluster`

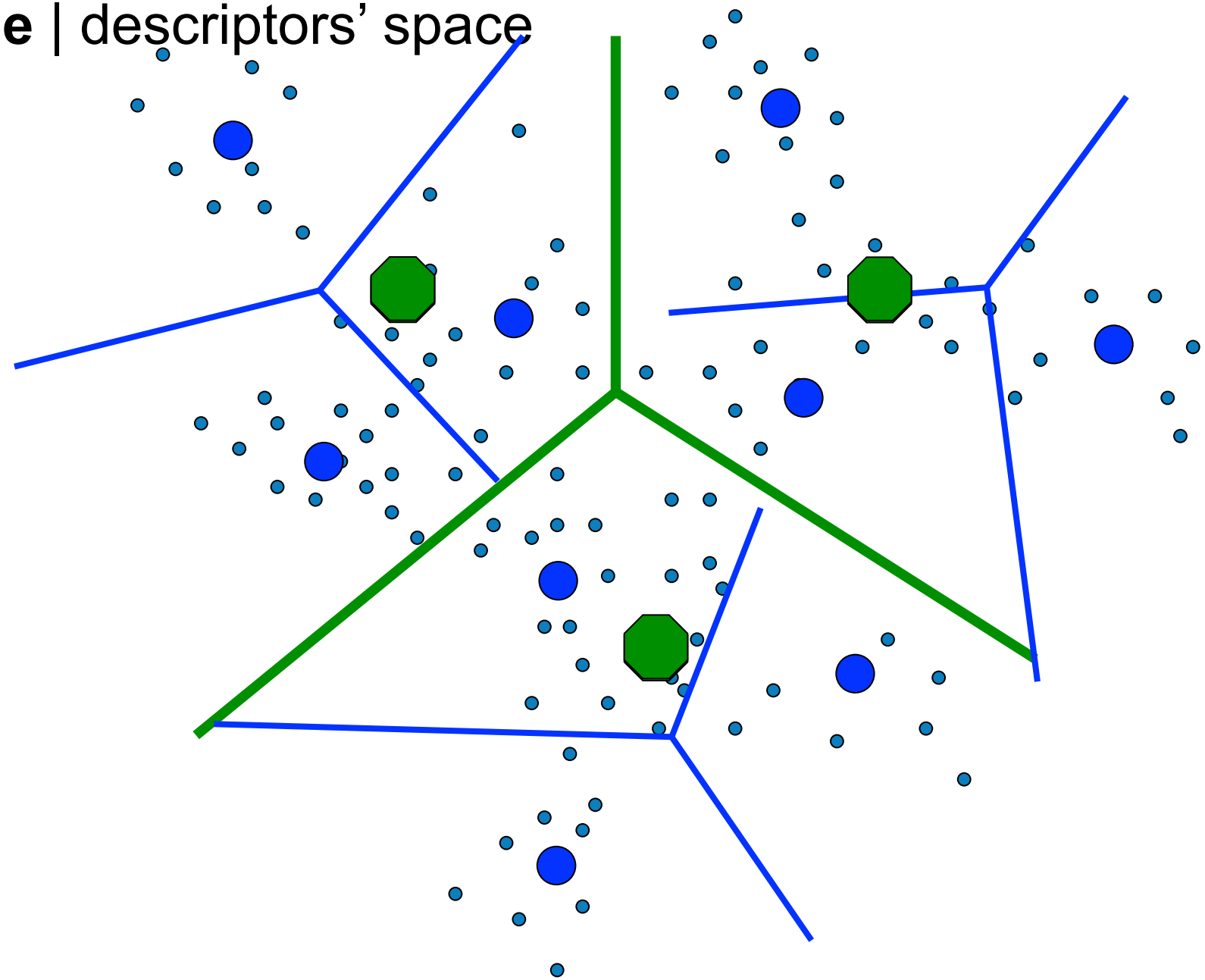


Vocabulary tree | descriptors' space



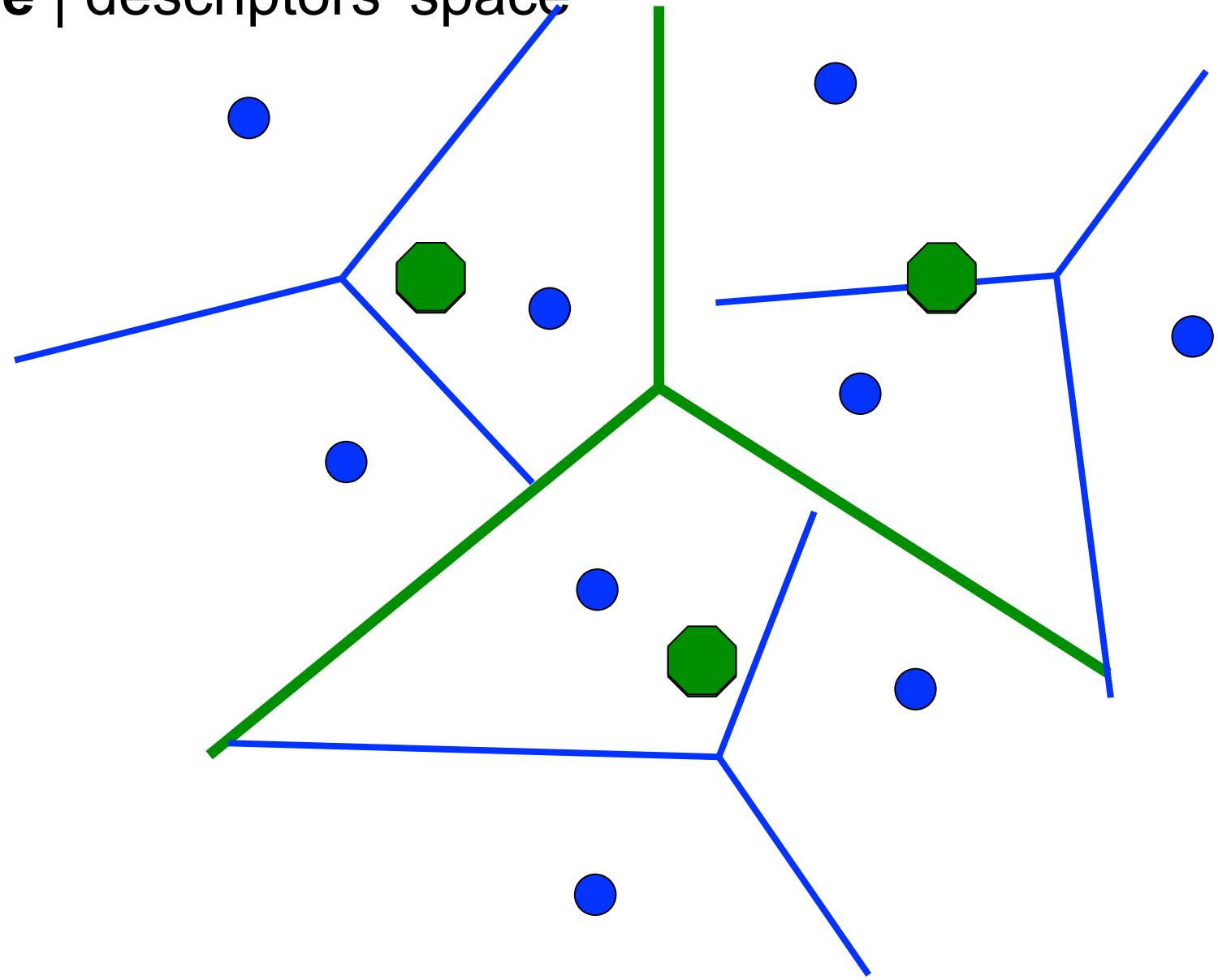
Based on D. Nister's slides

Vocabulary tree | descriptors' space



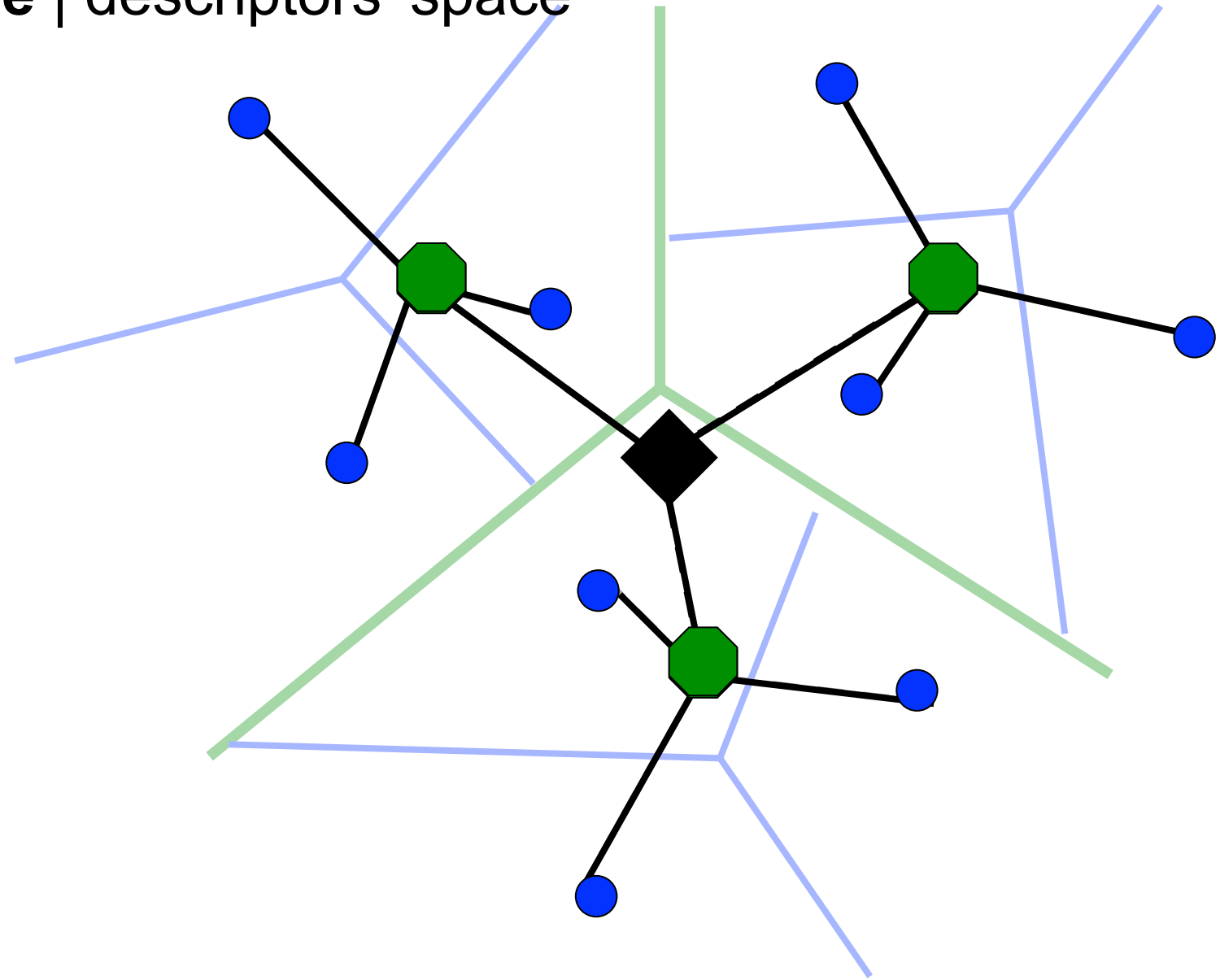
Based on D. Nister's slides

Vocabulary tree | descriptors' space



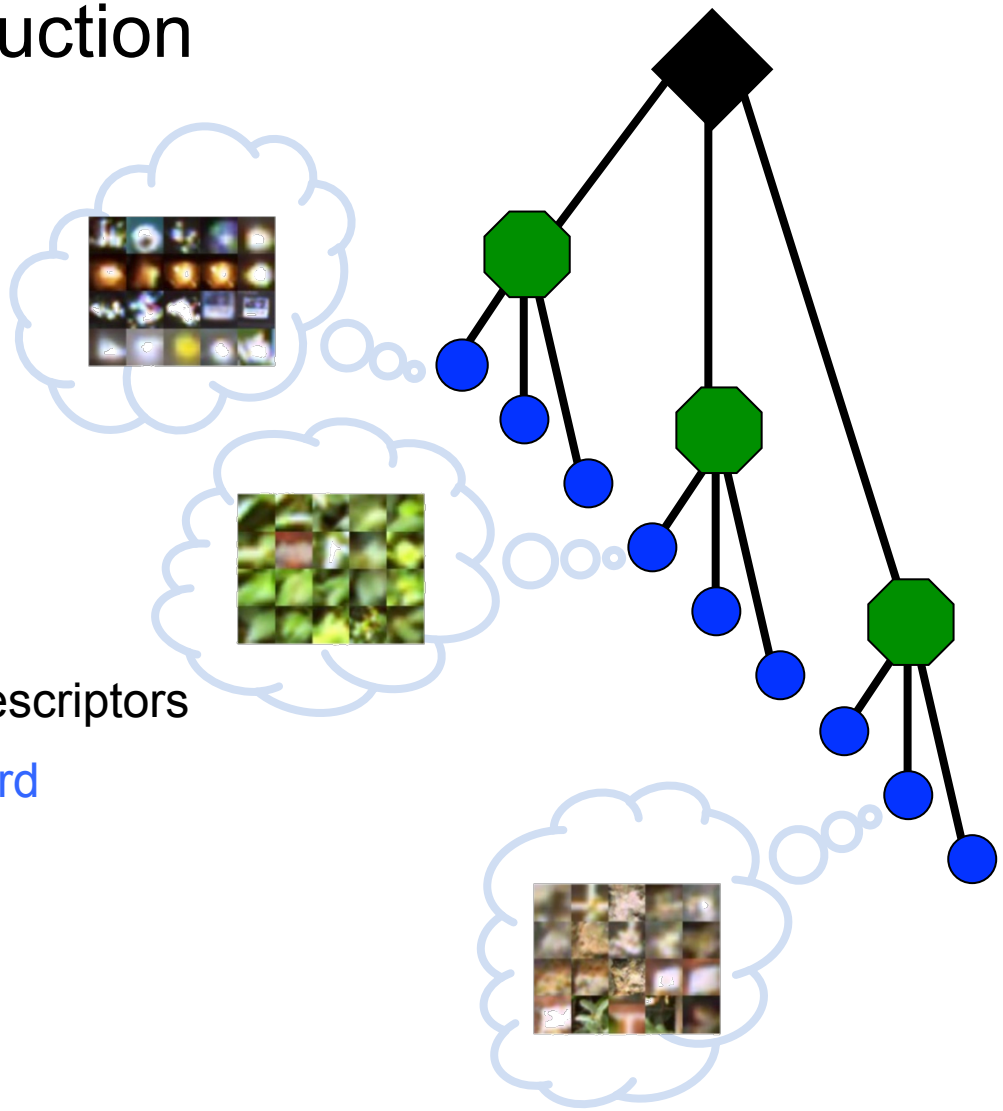
Based on D. Nister's slides

Vocabulary tree | descriptors' space



Based on D. Nister's slides

Vocabulary tree | construction

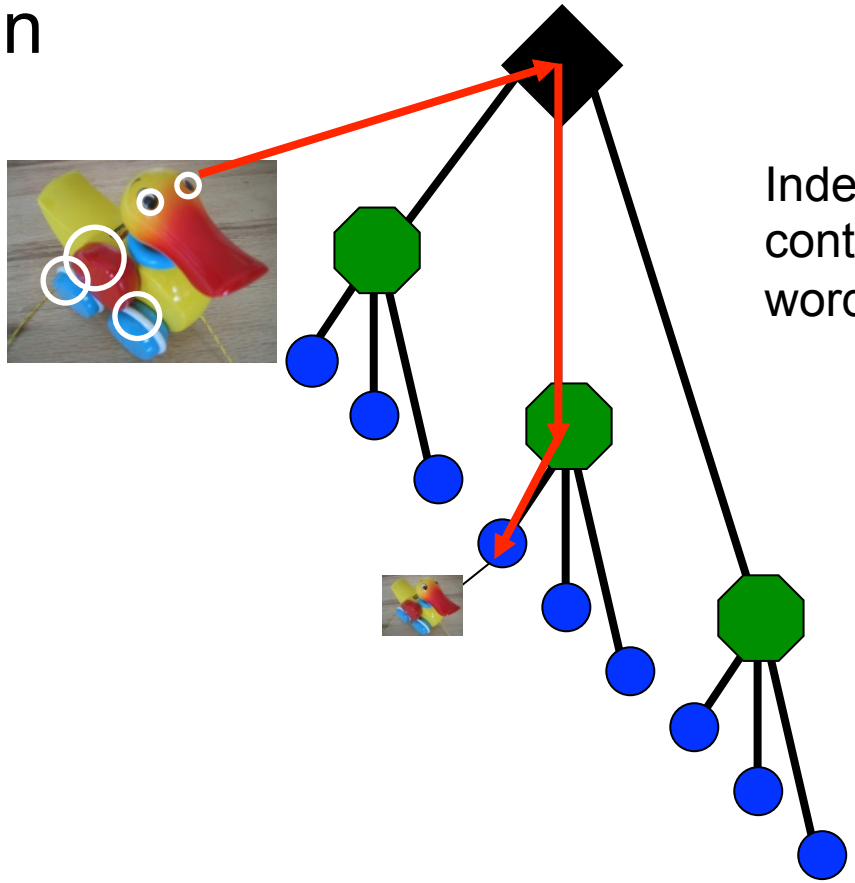


Each leaf represents a cluster of descriptors

⇨ each leaf represents a visual word

Vocabulary tree | construction

Model images

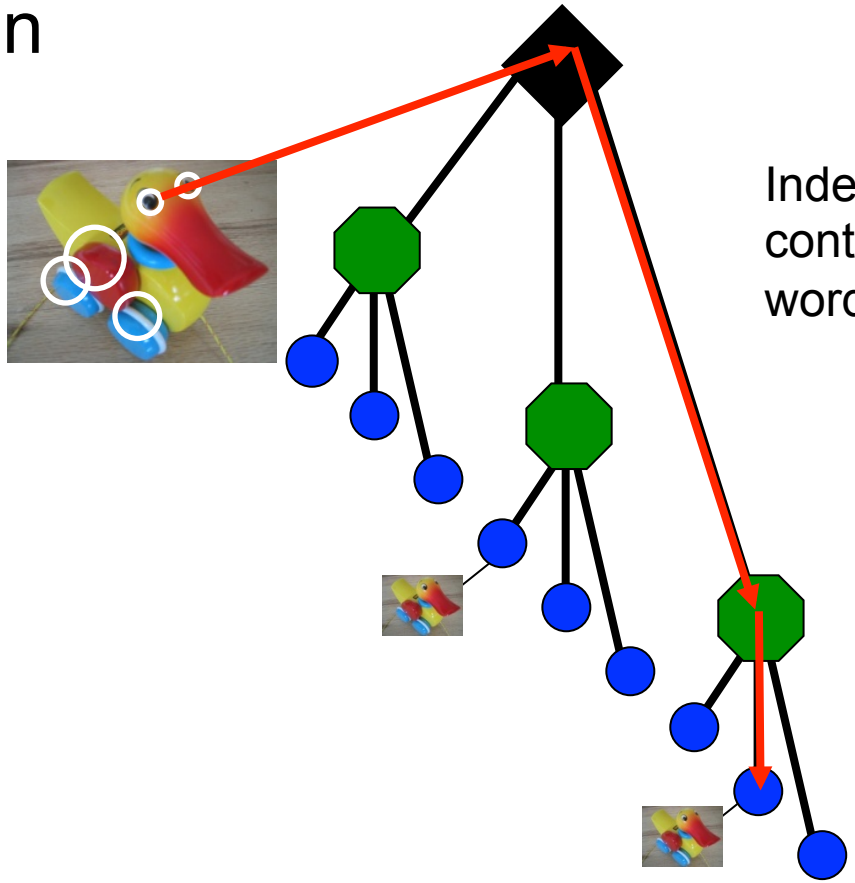


Index tree-leaves with images containing corresponding visual word

Based on D. Nister's slides

Vocabulary tree | construction

Model images

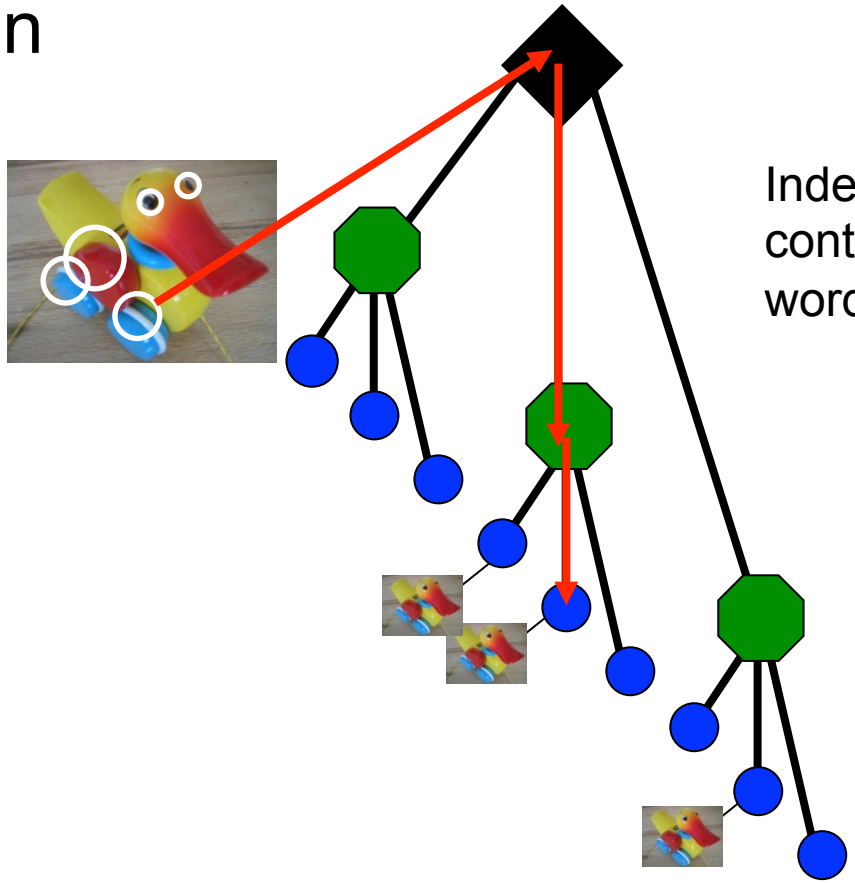


Index tree-leaves with images containing corresponding visual word

Based on D. Nister's slides

Vocabulary tree | construction

Model images

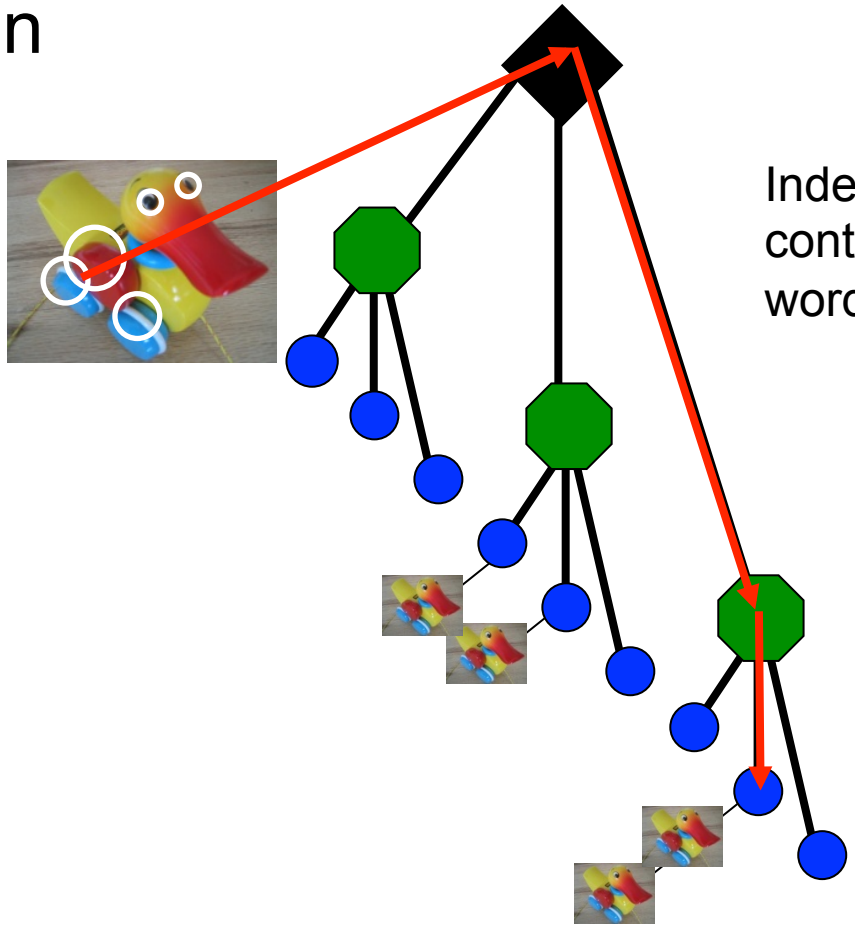


Index tree-leaves with images containing corresponding visual word

Based on D. Nister's slides

Vocabulary tree | construction

Model images

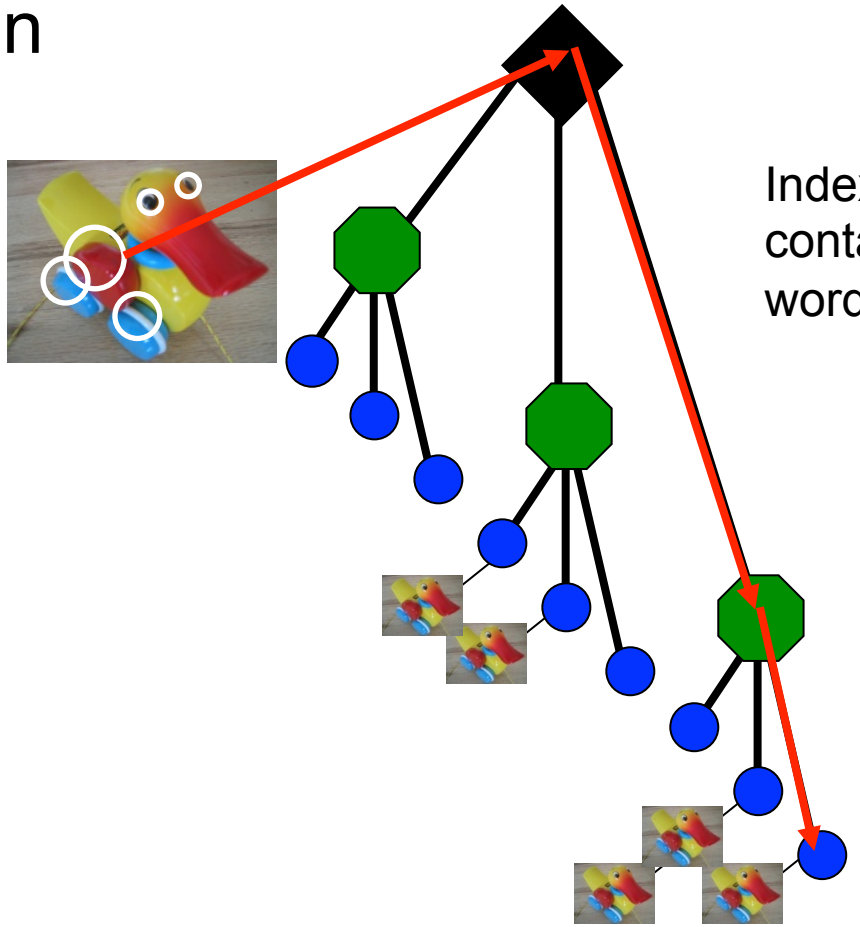


Index tree-leaves with images containing corresponding visual word

Based on D. Nister's slides

Vocabulary tree | construction

Model images

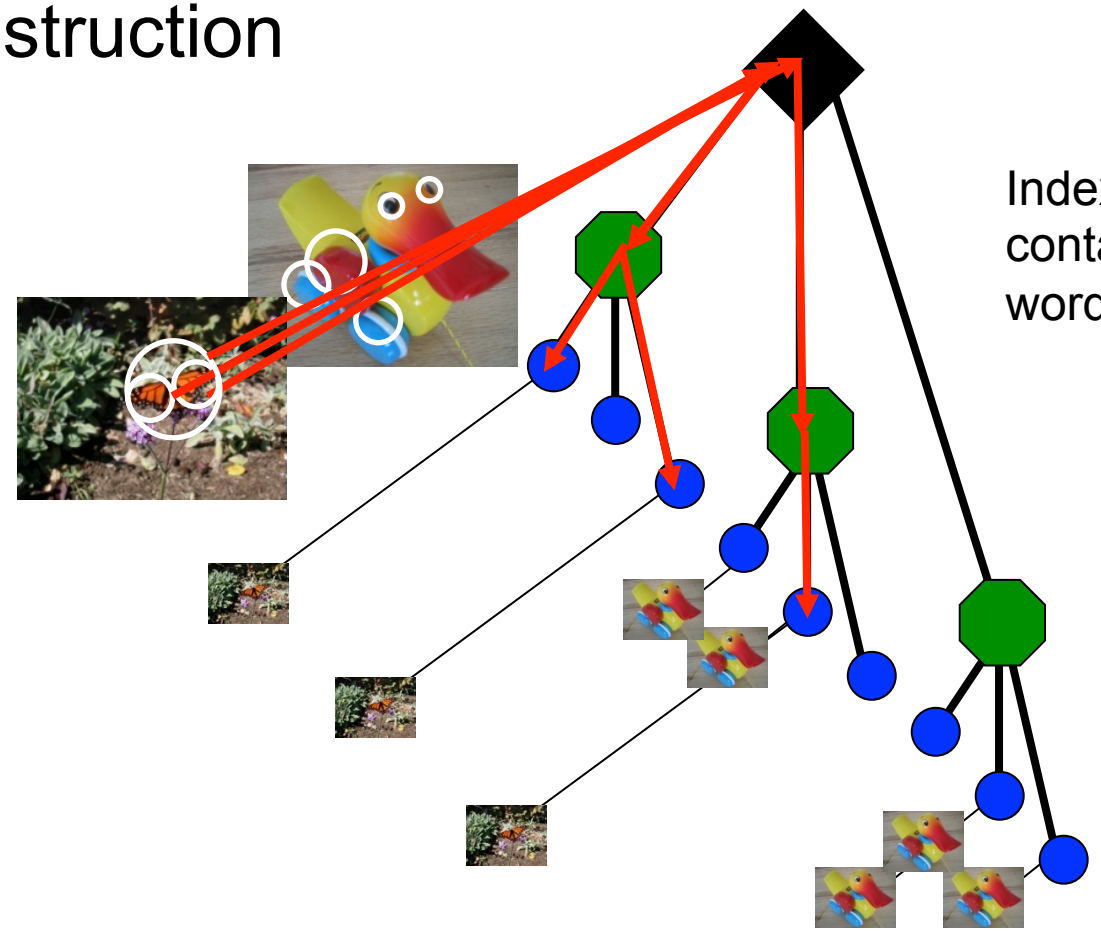


Index tree-leaves with images containing corresponding visual word

Based on D. Nister's slides

Vocabulary tree | construction

Model images

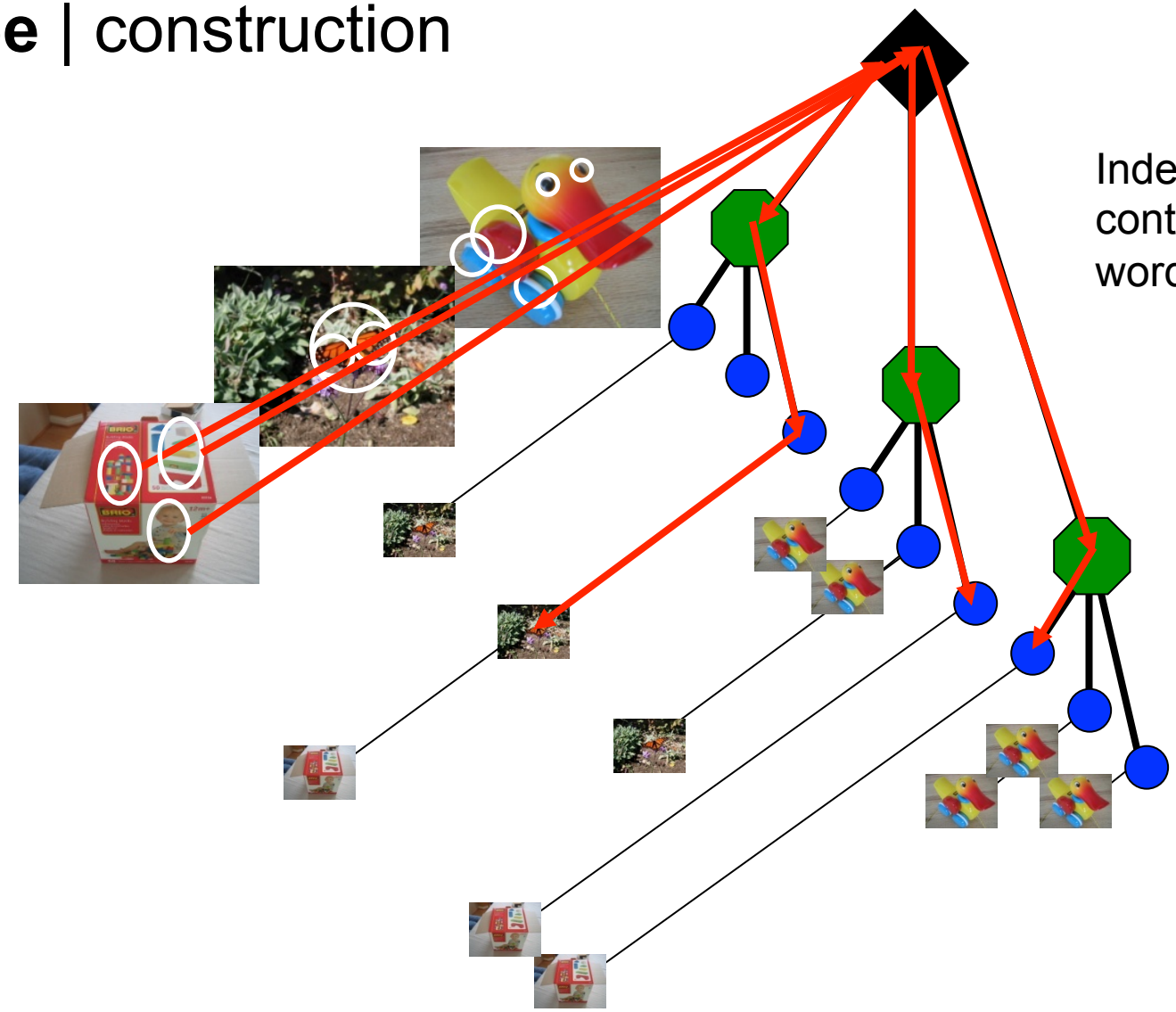


Index tree-leaves with images containing corresponding visual word

Based on D. Nister's slides

Vocabulary tree | construction

Model images

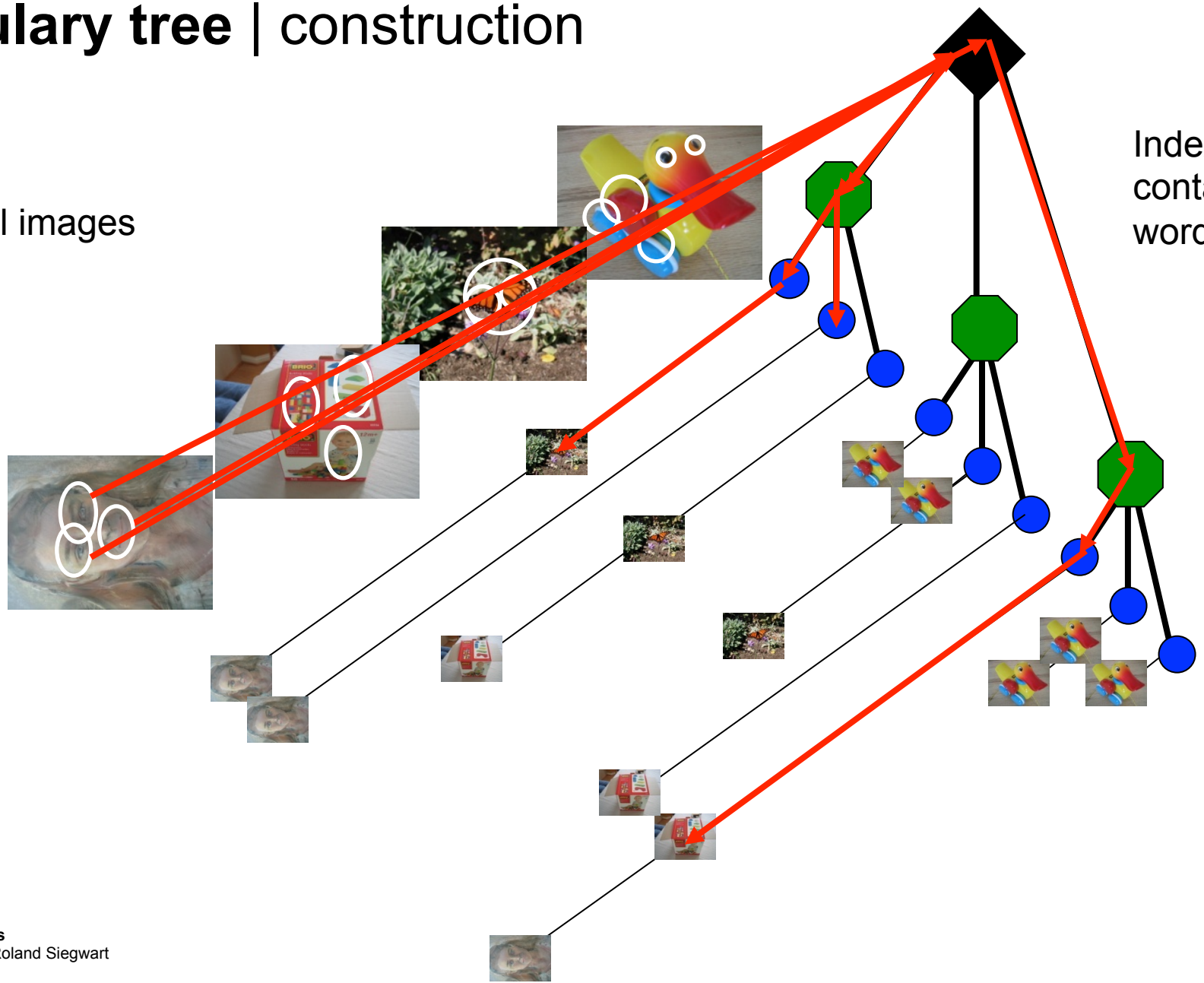


Index tree-leaves with images containing corresponding visual word

Based on D. Nister's slides

Vocabulary tree | construction

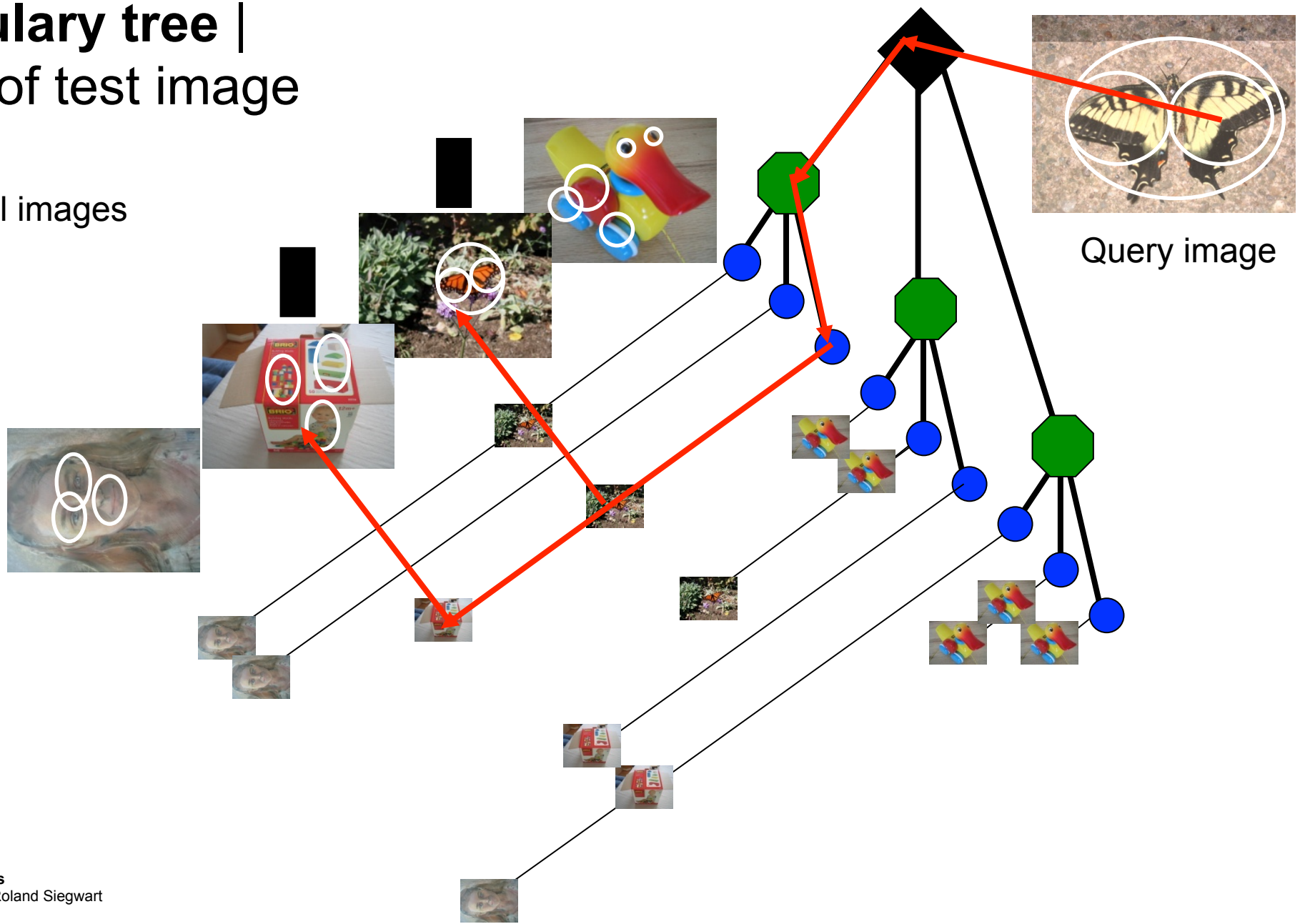
Model images



Index tree-leaves with images containing corresponding visual word

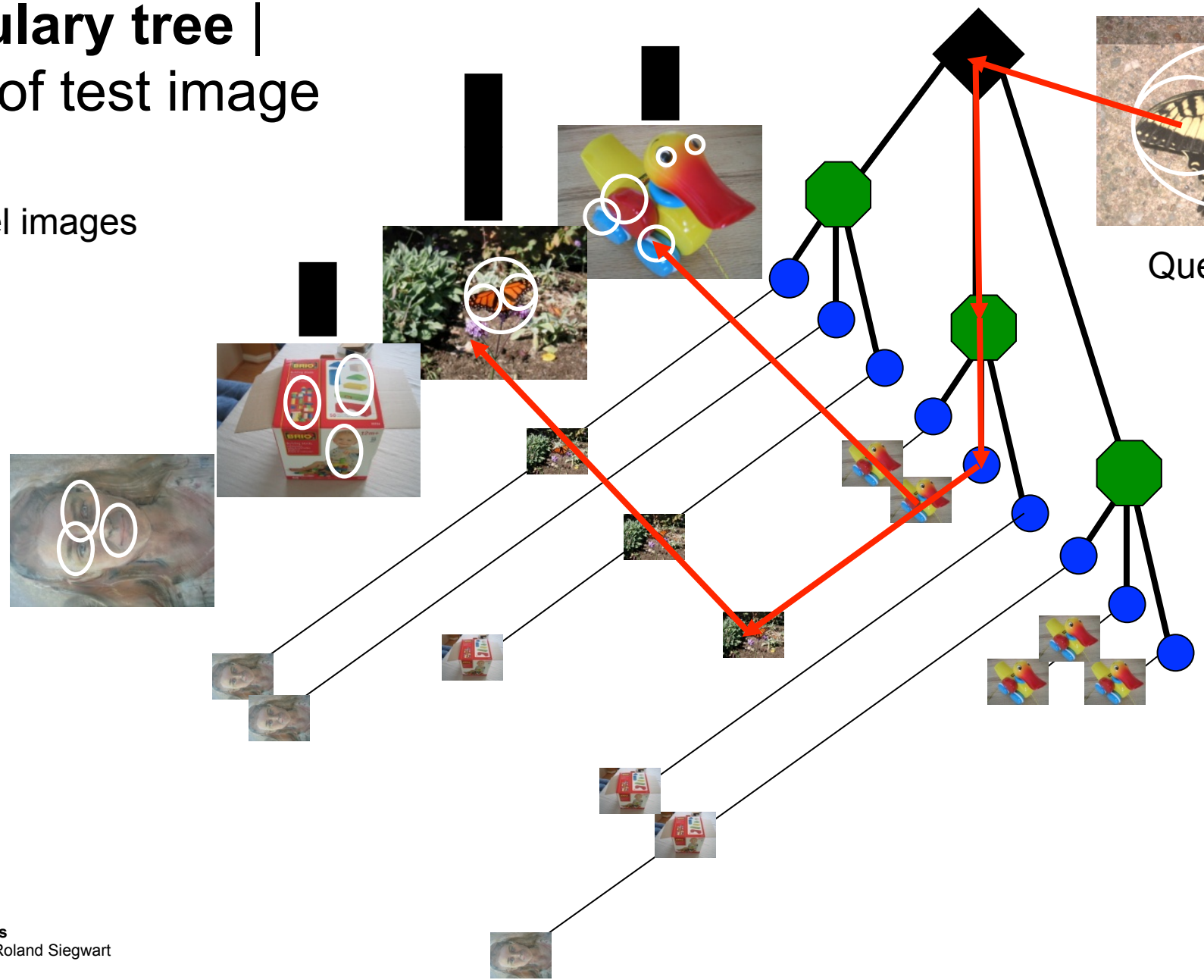
Based on D. Nister's slides

Vocabulary tree | lookup of test image



Vocabulary tree | lookup of test image

Model images

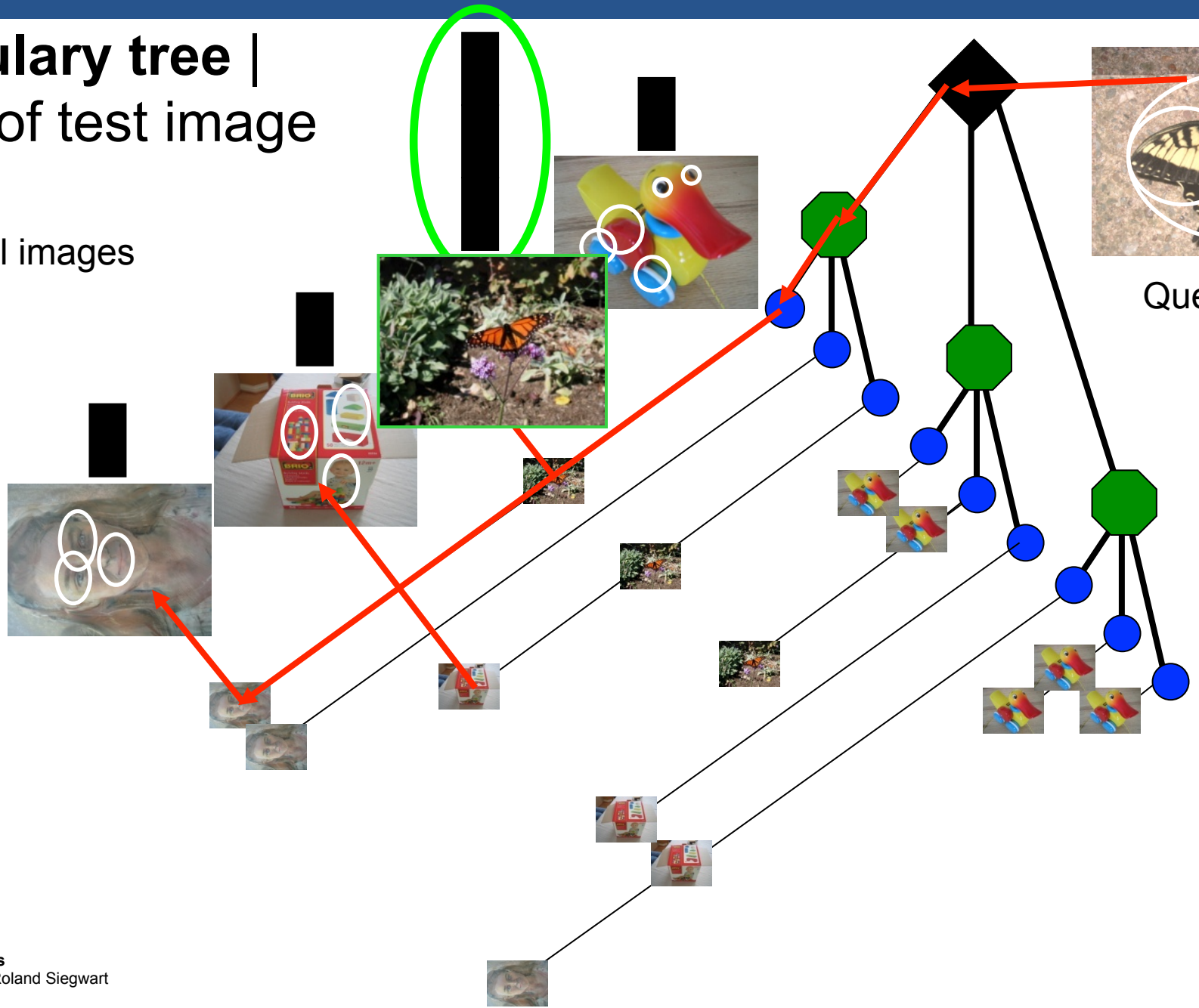


Query image

Based on D. Nister's slides

Vocabulary tree | lookup of test image

Model images

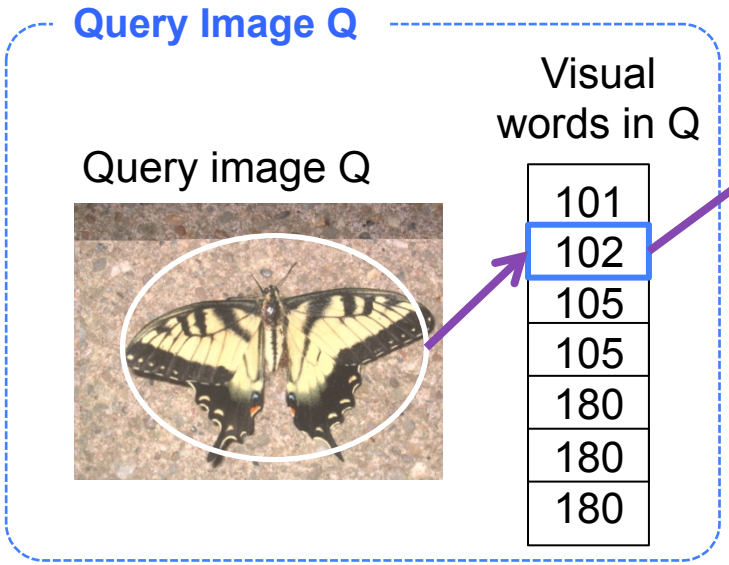


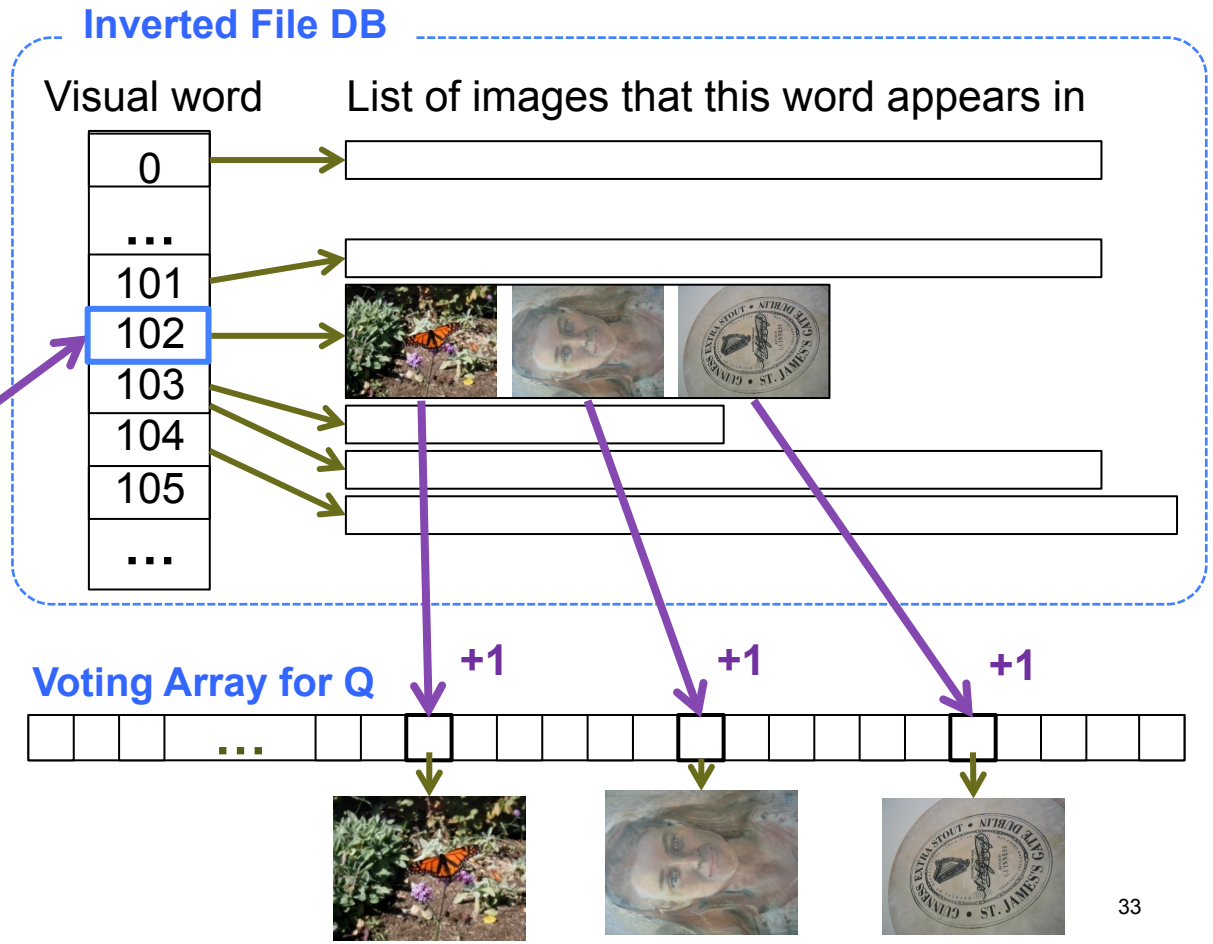
Query image

Based on D. Nister's slides

Vocabulary tree | inverted file index

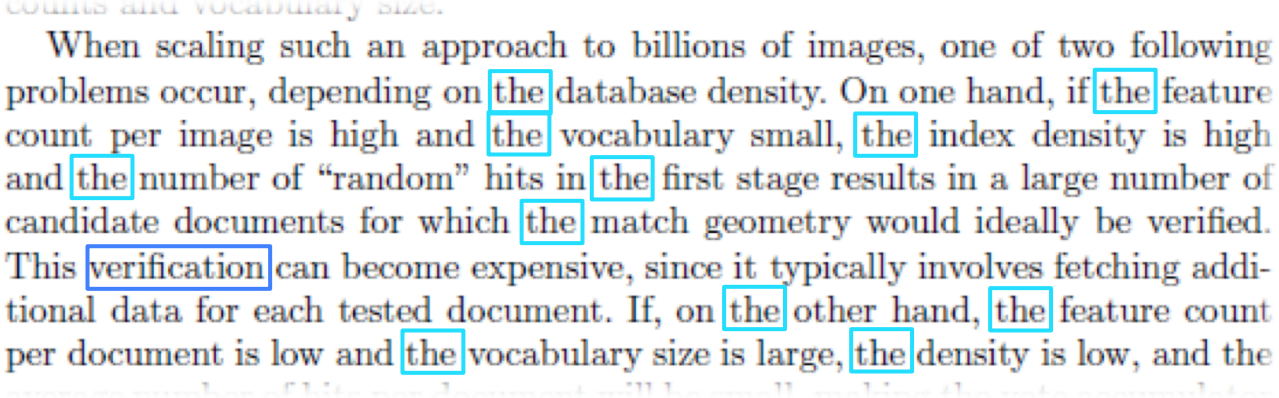
- Which image(s) look most like the Query Image Q ?
- An Inverted File DB lists all possible visual words
- Each word points to a list of images where this word occurs
- Voting array: has as many cells as the images in the DB – each word in query image votes for an image





Vocabulary tree | tf-idf

- **tf-idf:** **t**erm **f**requency-**i**nverse **d**ocument **f**requency
- measures the importance of a visual word inside a document (as part of a document DB)

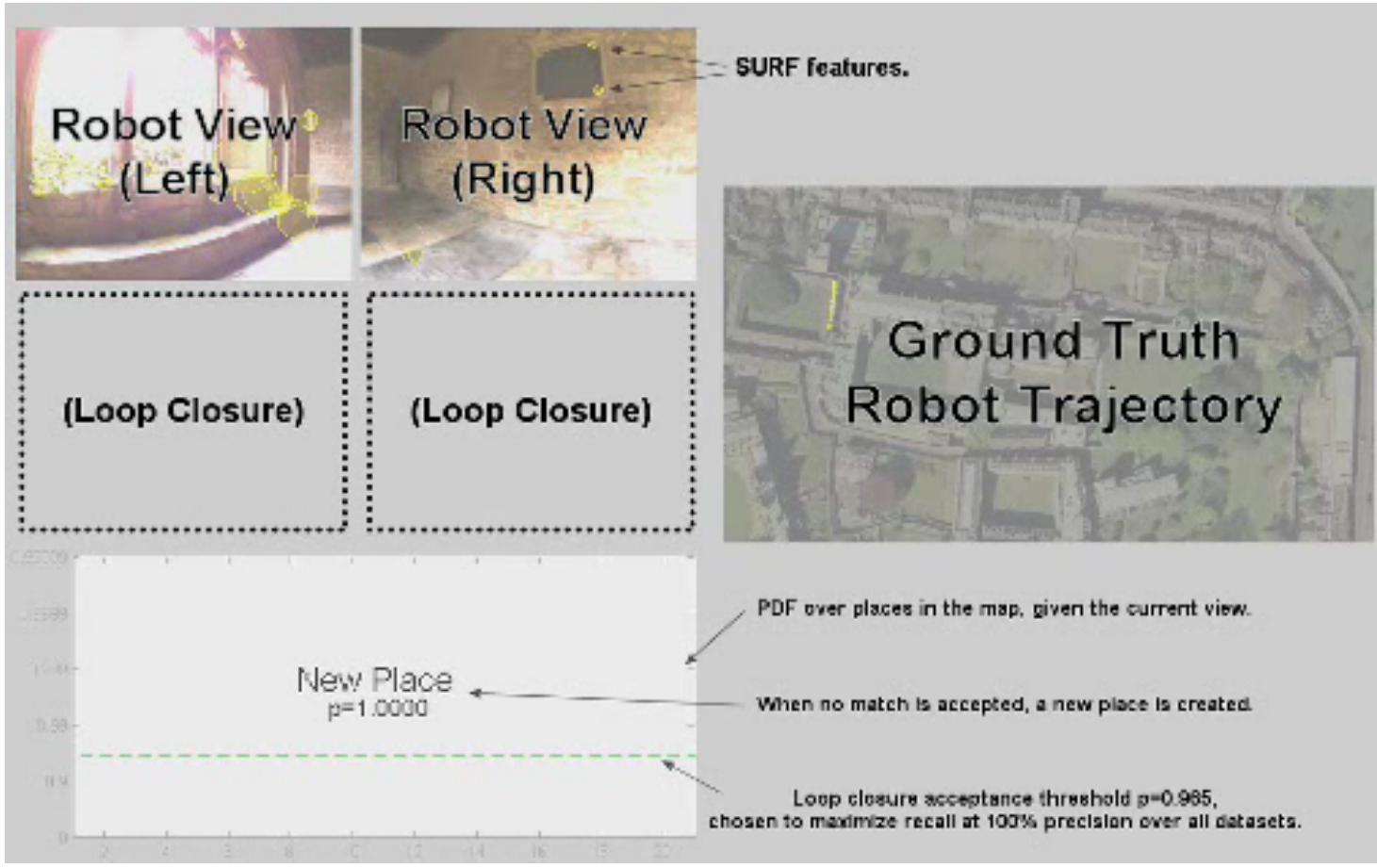


When scaling such an approach to billions of images, one of two following problems occur, depending on the database density. On one hand, if the feature count per image is high and the vocabulary small, the index density is high and the number of "random" hits in the first stage results in a large number of candidate documents for which the match geometry would ideally be verified. This verification can become expensive, since it typically involves fetching additional data for each tested document. If, on the other hand, the feature count per document is low and the vocabulary size is large, the density is low, and the

- term frequency: frequency of word w_i in image j: $tf_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}}$
- inverse document frequency: $idf_i = \log \frac{|D|}{|\{d : w_i \in d\}|}$ ← No. all images (documents); ← No. all images containing w_i
- tf-idf of word w_i in image j is: $= tf_{ij} \cdot idf_i$
- Use it to weigh the importance of each word when voting for corresponding image

Place recognition | FABMAP [Cummins and Newman IJRR 2011]

- Place recognition for robot localization
- Use training images to build the visual vocabulary, using SURF features
- Probabilistic model of the world:
  - Consider the *world* as a set of discrete places
  - *Place* = a set of consecutive images
- At a new frame, compute:
  - P(being at a known place)
  - P(being at a new place)
- Captures the dependencies of visual words to distinguish the most characteristic structure of each scene
  (using the Chow-Liu tree)
- Code available online:
  http://www.robots.ox.ac.uk/~mjc/Software.htm



Place recognition | FABMAP examples

p = probability of images coming from the same place

Place recognition | FABMAP examples

p = probability of images coming from the same place

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

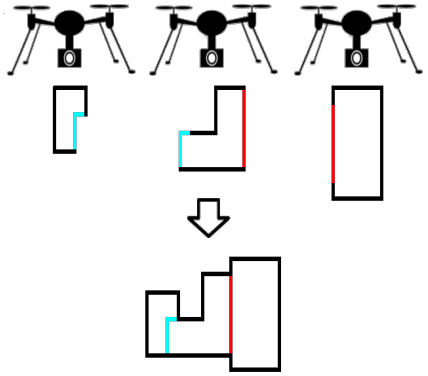
Place recognition | SeqSLAM [Milford & Wyeth, ICRA 2012]

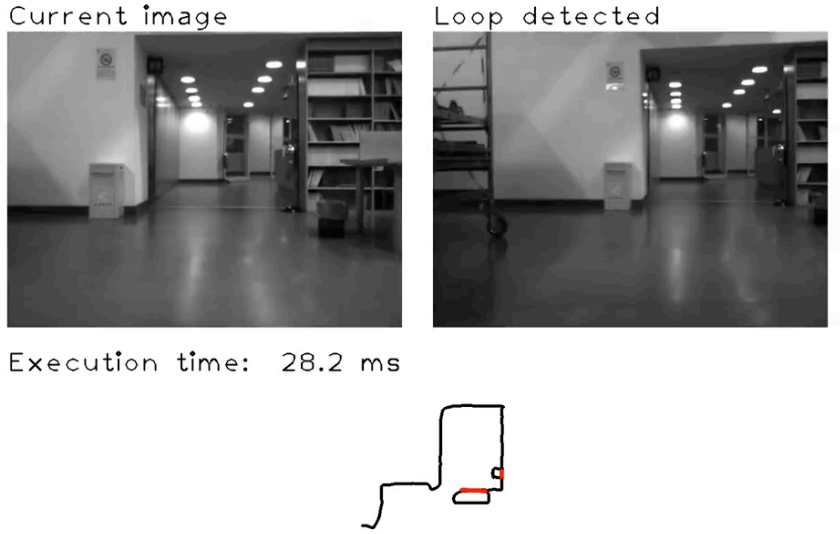
SeqSLAM: Visual Route-Based Navigation for Sunny Summer Days and Stormy Winter Nights

- Works with extreme illumination changes!
- Whole-image descriptor
- Image-preprocessing:
  - Crop,
  - Patch-normalize,
  - Down-sample
- Build image-difference matrix
- Recognizes loop when scene is visited:
  - at the same speed,
  - from the same viewpoint
  - Relies on long sequence of matching frames

Place Recognition | bags of binary words

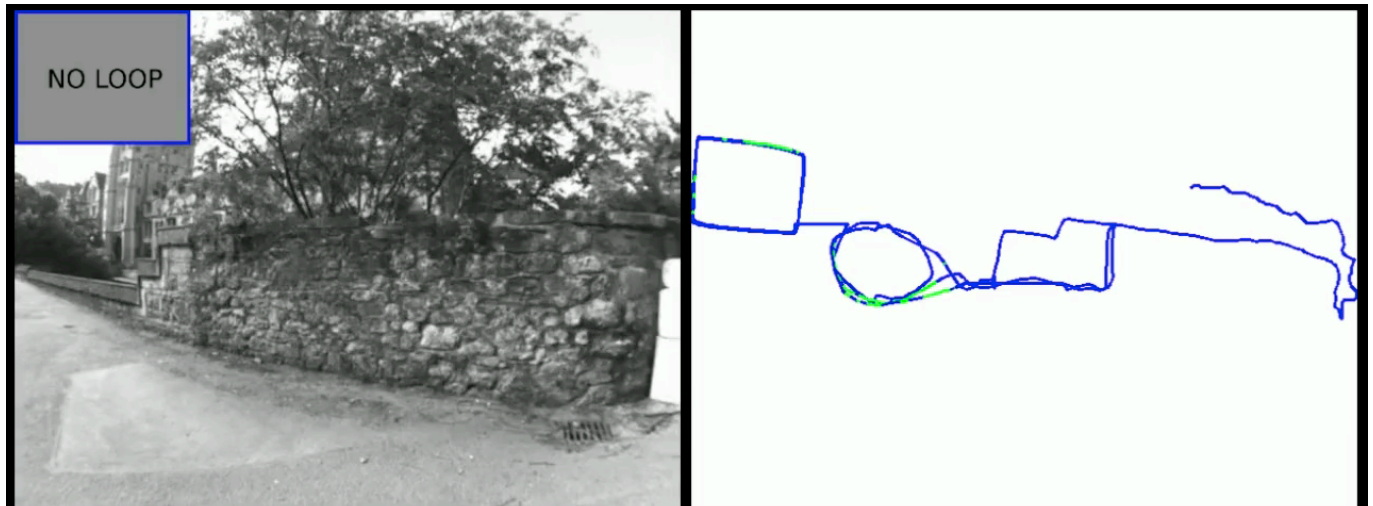
- **Bag of Binary Words (BoBW)**: [Gálvez-López & Tardós, TRO 2012]
  - FAST Keypoints + BRIEF descriptor
  - Very fast
  - Works for up-right images – Not viewpoint-invariant





- Work towards general & computationally feasible Place Recognition
- BoBW with BRISK features towards place rec. tolerant to viewpoint changes

Work with Fabiola Maffra



Place Recognition | towards robust performance

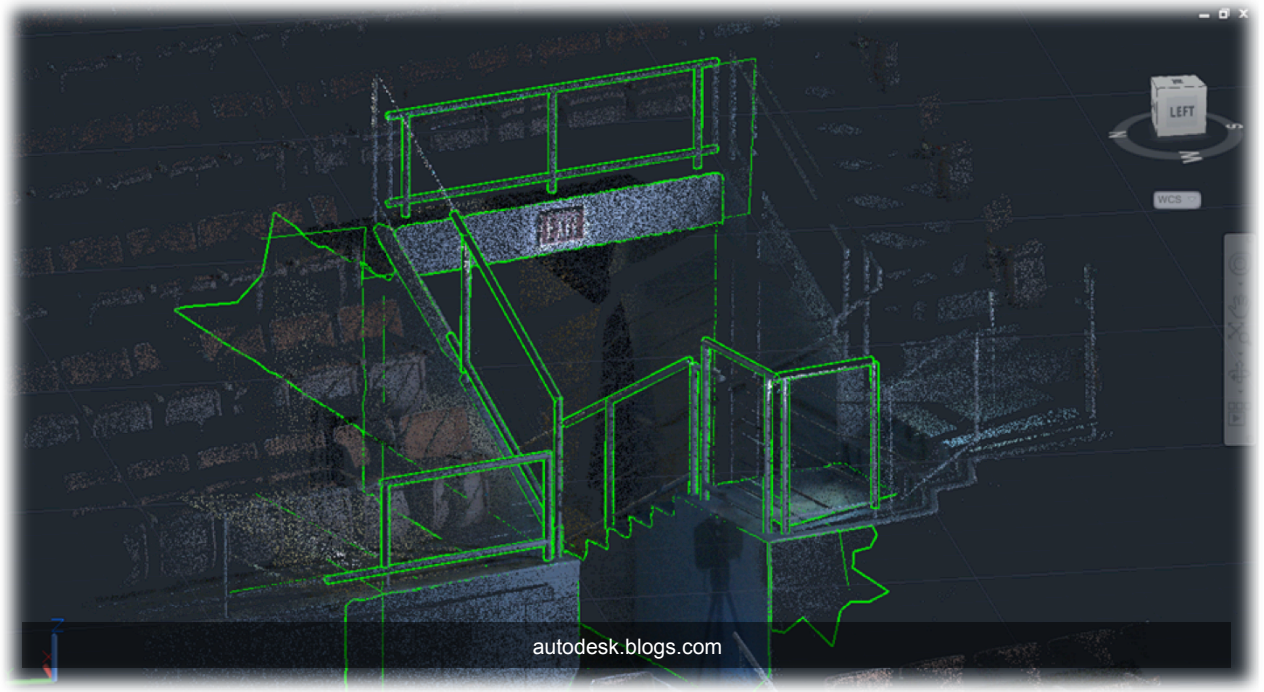
- Visual Vocabulary holds **appearance information,** but discards the spatial relationships between features

  ⇨ each image is considered as a "bag of words"

  ⇨ 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**
  - for example: test the *k* most similar images to the query image for geometric consistency (e.g. using RANSAC)

- Uncertainties – Representation & Propagation
- Line Extraction from point clouds



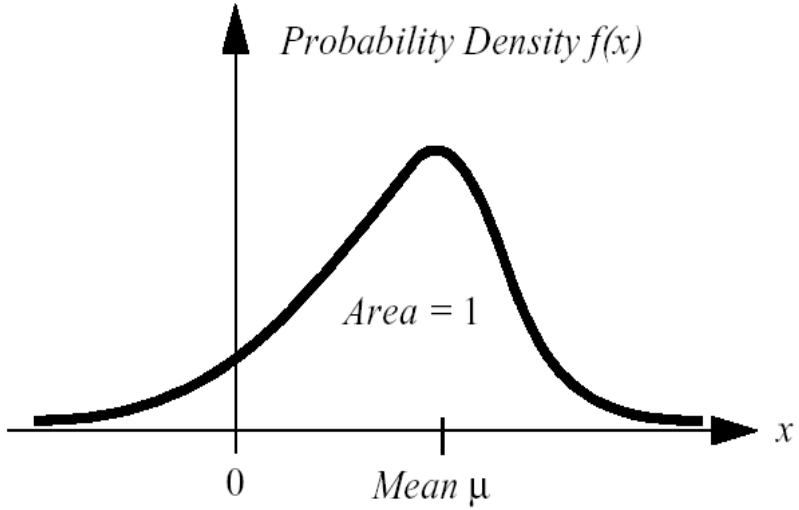
Uncertainty representation | importance

Section 4.1.3 of the book

- Sensing in the real world is **always uncertain**
  - How can uncertainty be **represented** or quantified?
  - How does uncertainty **propagate**?
    i.e. given uncertain inputs into a system, what is the uncertainty in the output?
  - What is the merit of all this for mobile robotics?

Uncertainty representation

- Use a Probability Density Function (PDF) to characterize the statistical properties of a variable X:



$$\int_{-\infty}^{\infty} f(x)dx = 1$$

- Expected value of Variable X:

$$\mu = E[X] = \int_{-\infty}^{\infty} xf(x)dx$$

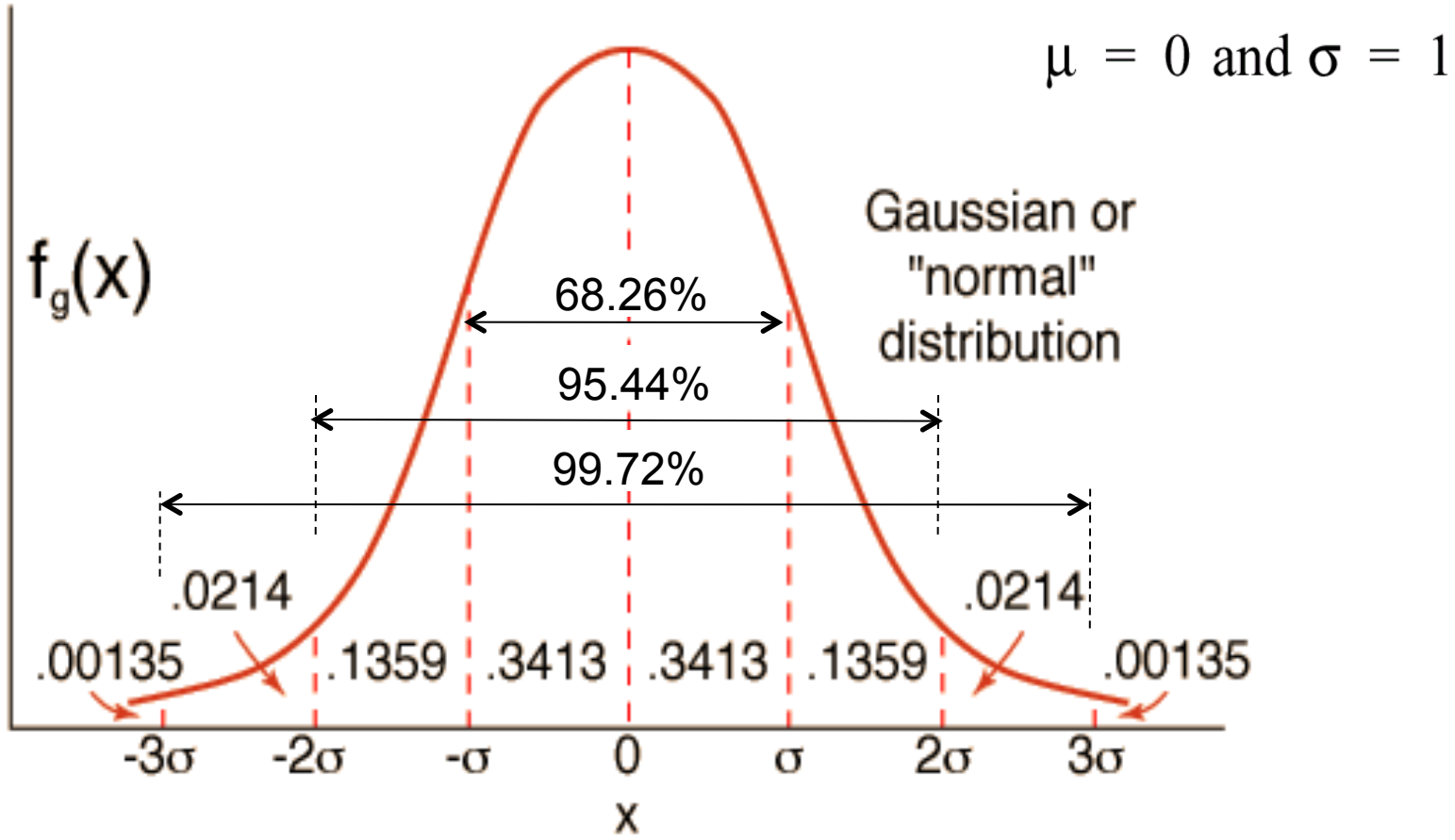
- Variance of variable X:

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx$$

Uncertainty representation | gaussian distribution

- Most common PDF for characterizing uncertainties: Gaussian

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

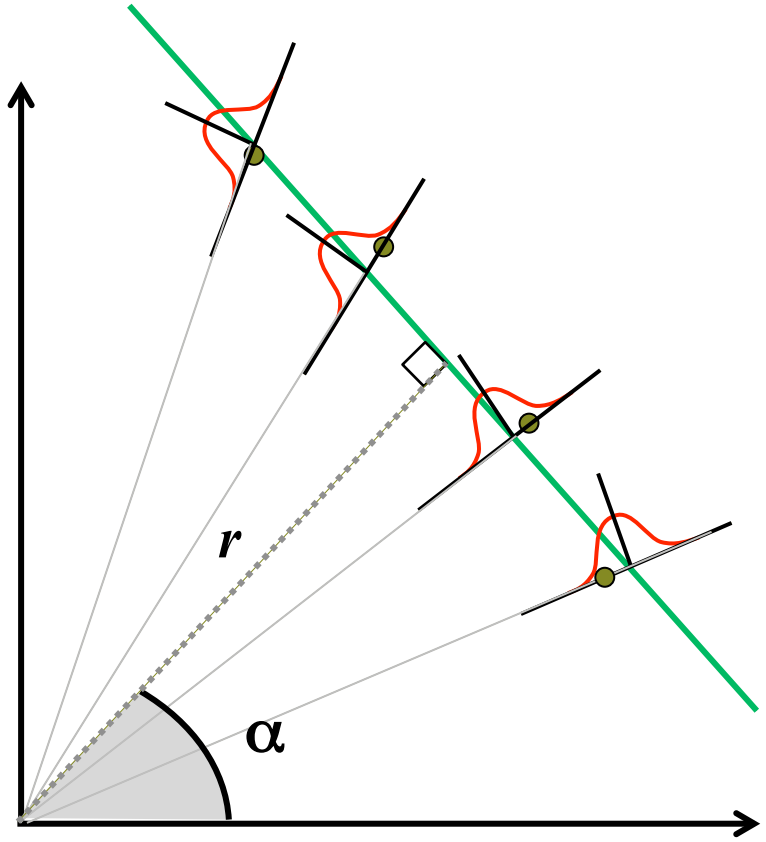


Uncertainty representation | the error propagation law

- Imagine extracting a line based on point measurements with uncertainties.

- Model parameters in polar coordinates

  [(r, α) uniquely identifies a line]

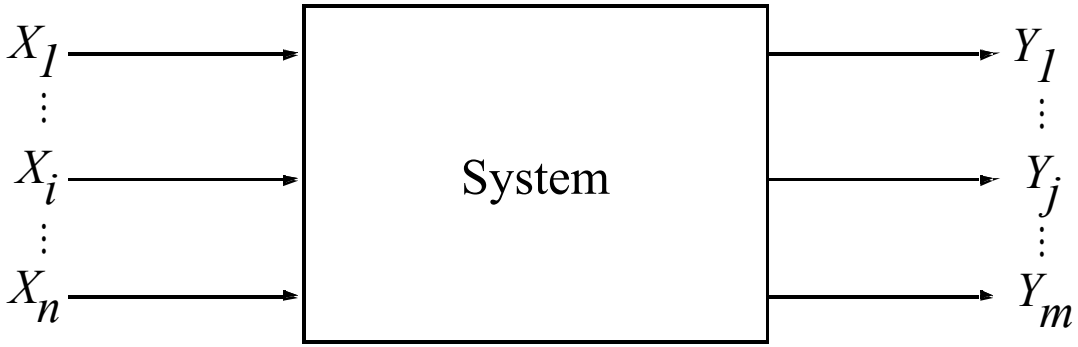


The question:

- What is the **uncertainty** **of the extracted line** knowing the uncertainties of the measurement points that contribute to it ?

Uncertainty representation | the error propagation law

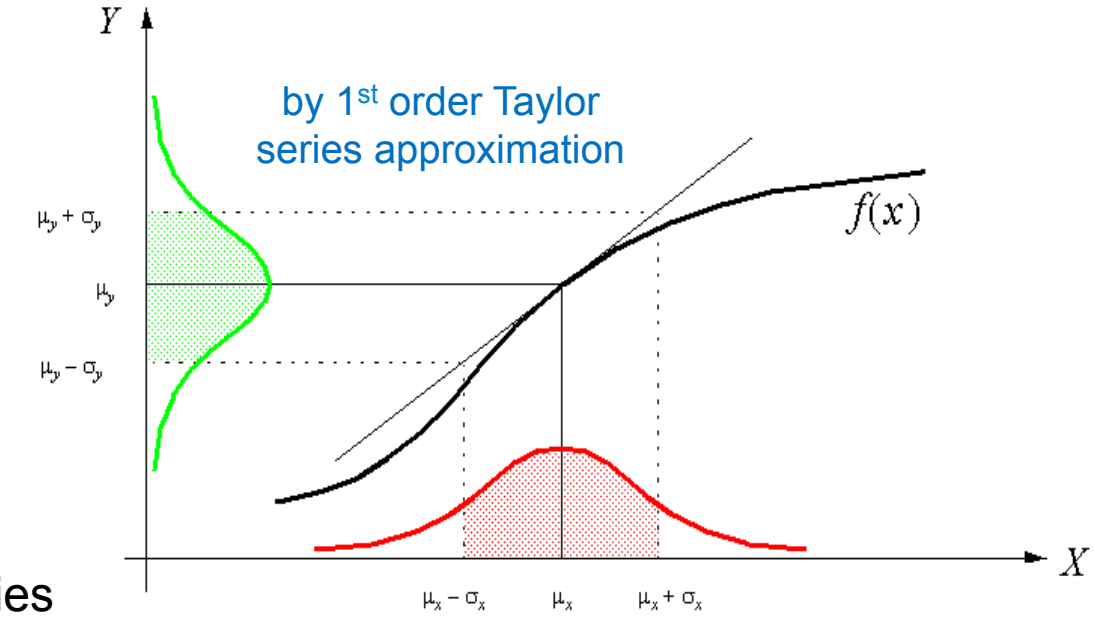
Error propagation in a multiple-input mutliple-output system with *n* inputs and *m* outputs



$$Y_j = f_j(X_1 \ldots X_n)$$

Uncertainty representation | the error propagation law

- 1D case of a nonlinear error propagation problem:



- It can be shown that the output covariance matrix C_Y is given by the error propagation law:

$$C_{YY} = F_{YX} C_{XX} F_{XY} \qquad , F_{XY} = F_{YX}^T$$

where

- C_{XX}: covariance matrix representing the input uncertainties
- C_{YY}: covariance matrix representing the propagated uncertainties for the outputs.
- F_{YX}: is the **Jacobian** matrix defined as:

$$F_{YX} = \begin{pmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_m}{\partial X_1} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_m}{\partial X_1} & \cdots & \frac{\partial f_m}{\partial X_n} \end{pmatrix}$$

- Defines the orientation of the tangent line/plane/hyper-plane at a given point

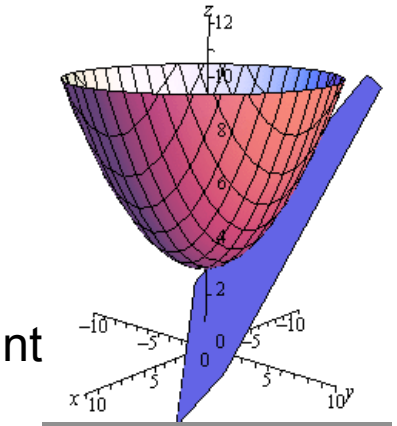


Image courtesy of K. Arras

Uncertainty representation | line extraction

- Point-Line distance

$$\rho_i \cos(\theta_i - \alpha) - r = d_i$$

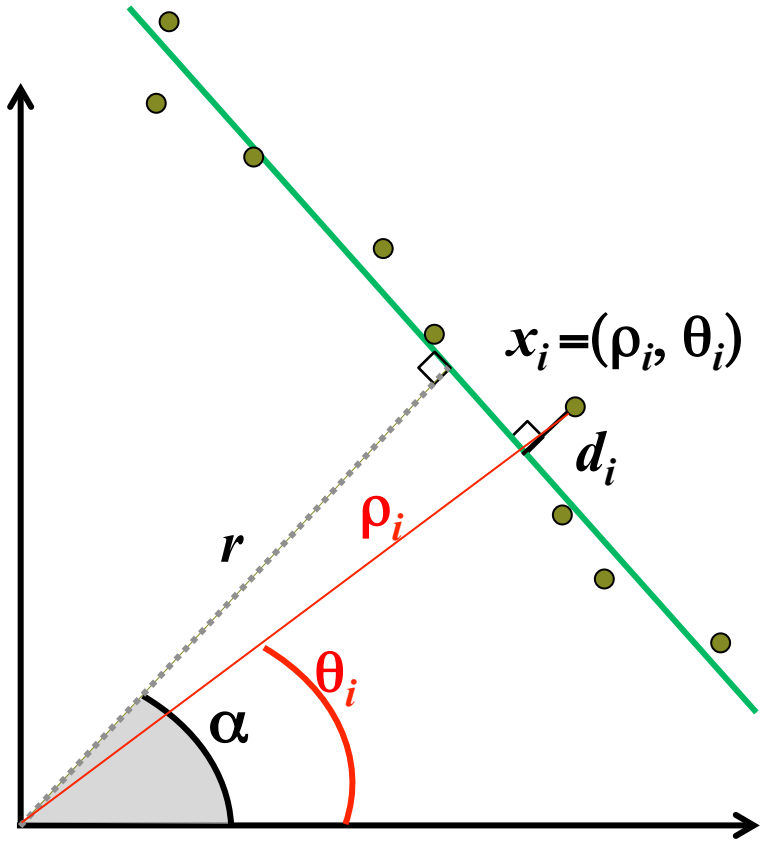
- If each measurement is equally uncertain then sum of sq. errors:

$$S = \sum_i d_i^2 = \sum_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$

- Goal: minimize S when selecting (r, α) ⇨ solve the system

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0$$

- **"Unweighted Least Squares"**



Uncertainty representation | line extraction

- Point-Line distance

$$\rho_i \cos(\theta_i - \alpha) - r = d_i$$

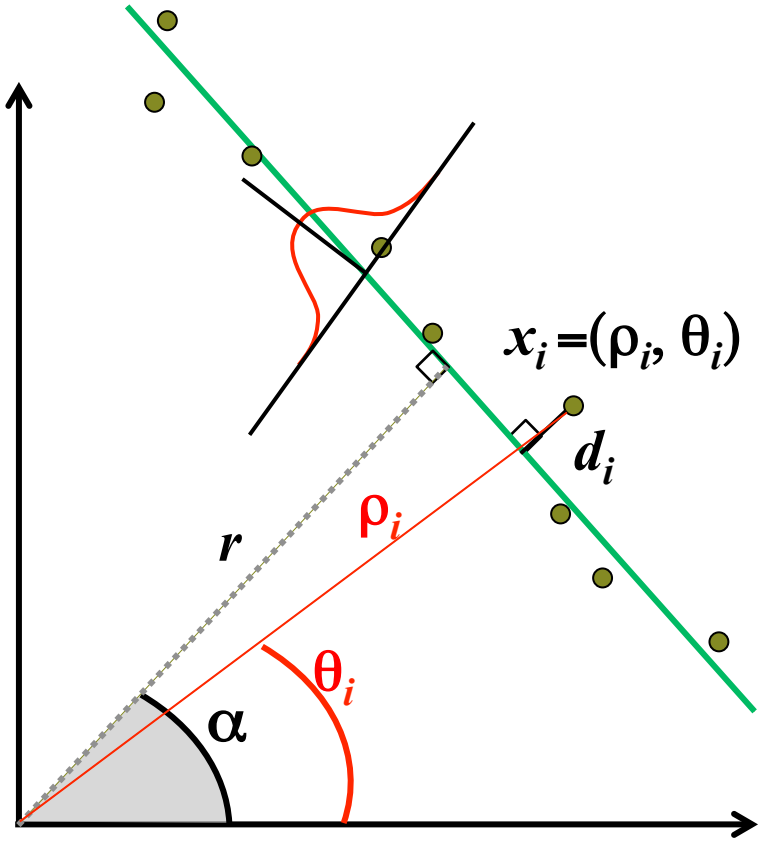
- Each sensor measurement, may have its own, unique uncertainty σ_i^2

$$S = \sum w_i d_i^2 = \sum w_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$

$$w_i = 1/\sigma_i^2$$

- **"Weighted Least Squares"**

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0$$



Uncertainty representation | line extraction

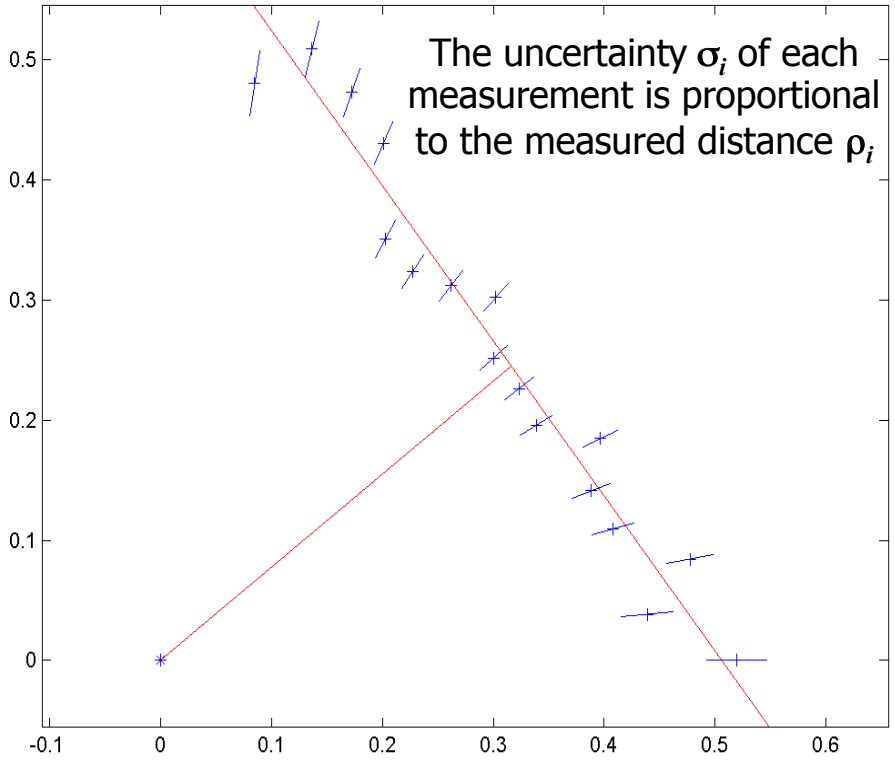
- Weighted least squares and solving the system:

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0$$

- Gives the line parameters:

$$\alpha = \frac{1}{2}\text{atan}\left(\frac{\sum w_i \rho_i^2 \sin 2\theta_i - \frac{2}{\sum w_i}\sum\sum w_i w_j \rho_i \rho_j \cos\theta_i \sin\theta_j}{\sum w_i \rho_i^2 \cos 2\theta_i - \frac{1}{\sum w_i}\sum\sum w_i w_j \rho_i \rho_j \cos(\theta_i + \theta_j)}\right)$$

$$r = \frac{\sum w_i \rho_i \cos(\theta_i - \alpha)}{\sum w_i}$$



- If $\begin{cases} \rho_i \sim N(\hat{\rho}_i, \sigma_{\rho_i}^2) \\ \theta_i \sim N(\hat{\theta}_i, \sigma_{\theta_i}^2) \end{cases}$ what is the uncertainty in the line (r, α) ?

Uncertainty representation | line extraction

The uncertainty of **each measurement** $x_i = (\rho_i, \theta_i)$ is described by the covariance matrix:

Assuming that ρ_i, θ_i are independent

$$C_{xx_i} = \begin{bmatrix} \sigma^2_{\rho_i} & 0 \\ 0 & \sigma^2_{\theta_i} \end{bmatrix}$$

The uncertainty in the **line** (r, α) is described by the covariance matrix: $C_{ll} = \begin{bmatrix} \sigma^2_{\alpha} & \sigma_{\alpha r} \\ \sigma_{r\alpha} & \sigma^2_{r} \end{bmatrix} = ?$

Define:

$$C_{xx} = \begin{bmatrix} diag(\sigma^2_{\rho}) & 0 \\ 0 & diag(\sigma^2_{\theta}) \end{bmatrix} = \begin{bmatrix} \dots & 0 & 0 & . & 0 & 0 & . \\ . & \sigma^2_{\rho_i} & 0 & . & 0 & 0 & . \\ . & 0 & \sigma^2_{\rho_{i+1}} & . & 0 & 0 & . \\ . & . & . & \dots & . & . & . \\ . & 0 & 0 & . & \sigma^2_{\theta_i} & 0 & . \\ . & 0 & 0 & . & 0 & \sigma^2_{\theta_i+1} & . \\ . & . & . & . & . & . & \dots \end{bmatrix}$$

$2n \times 2n$

Error Propagation Law

$$C_{ll} = F_{lx} C_{xx} F_{xl}$$

Jacobian:

$$F_{lx} = \begin{bmatrix} \dots & \frac{\partial \alpha}{\partial \rho_i} & \frac{\partial \alpha}{\partial \rho_{i+1}} & \dots & \frac{\partial \alpha}{\partial \theta_i} & \frac{\partial \alpha}{\partial \theta_{i+1}} & \dots \\ \dots & \frac{\partial r}{\partial \rho_i} & \frac{\partial r}{\partial \rho_{i+1}} & \dots & \frac{\partial r}{\partial \theta_i} & \frac{\partial r}{\partial \theta_{i+1}} & \dots \end{bmatrix}$$

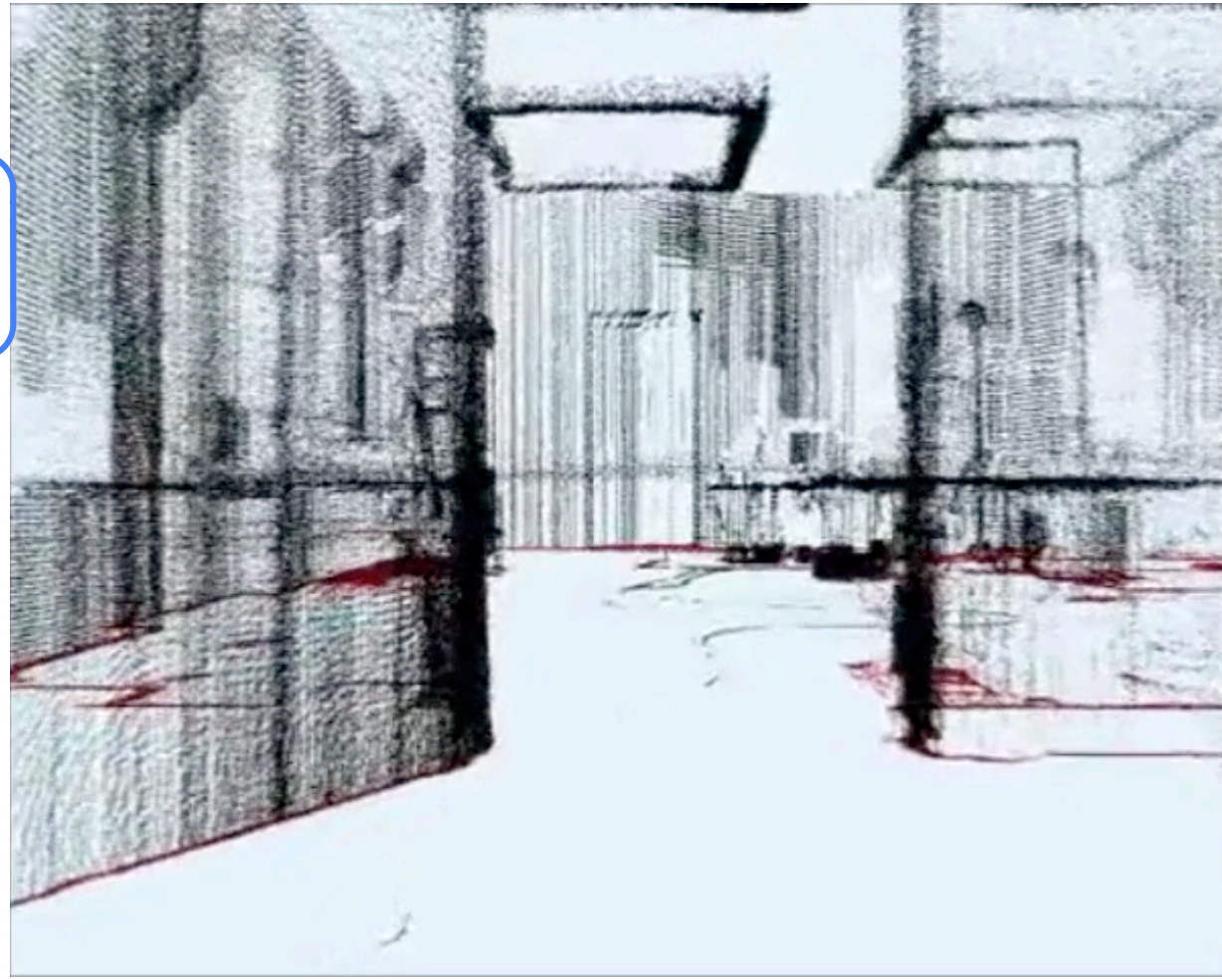
Line extraction from a point cloud

Extract lines from a point cloud (e.g. range scan)

- Three main problems:
  - How many lines are there?
  - **Segmentation**: Which points belong to which line?
  - **Line Fitting/Extraction**: Given points that belong to a line, how to estimate the line parameters?

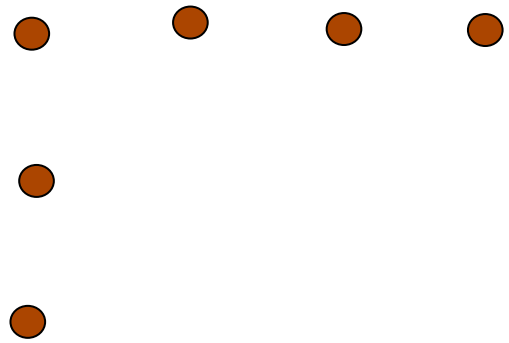
- Algorithms we will see:
  1. Split-and-merge
  2. Linear regression
  3. RANSAC
  4. Hough-Transform

http://www.youtube.com/watch?v=wV8frjLqtIA&feature=related



Line extraction | 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.



Line extraction | 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.

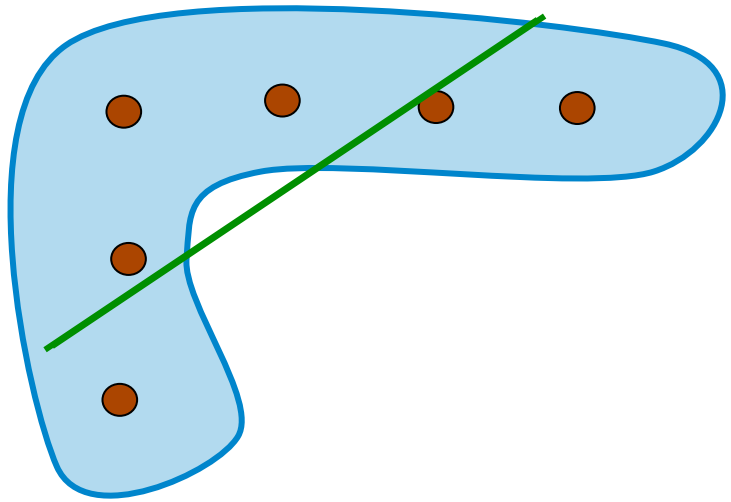
Let **S** be the set of all data points

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



Line extraction | 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.

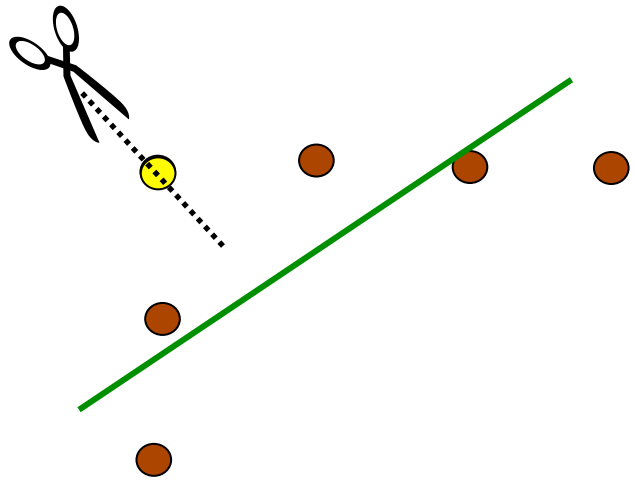
Let **S** be the set of all data points

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



Line extraction | 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.

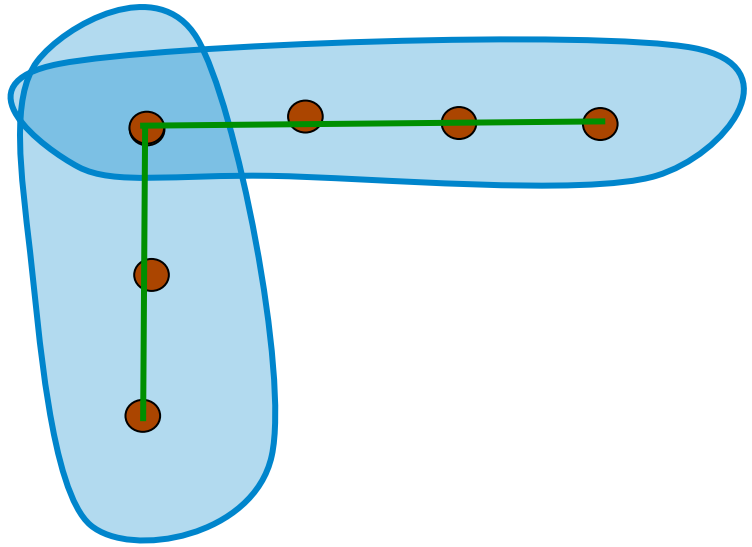
Let **S** be the set of all data points

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

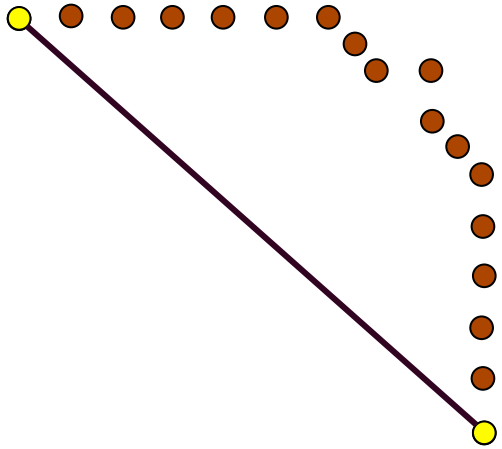
Merge

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



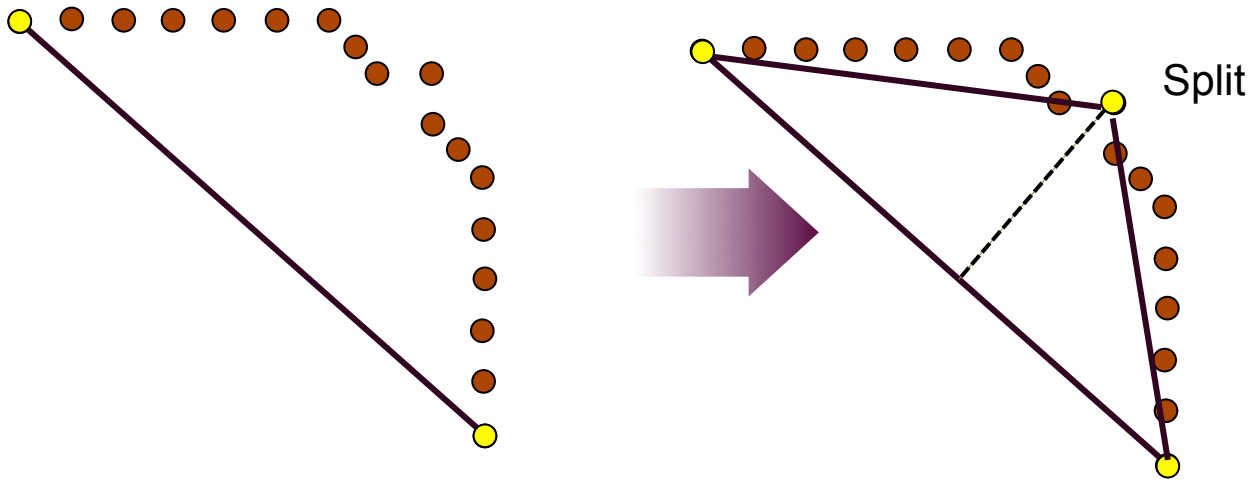
Line extraction | 1. split-and-merge (iterative end-point-fit)

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



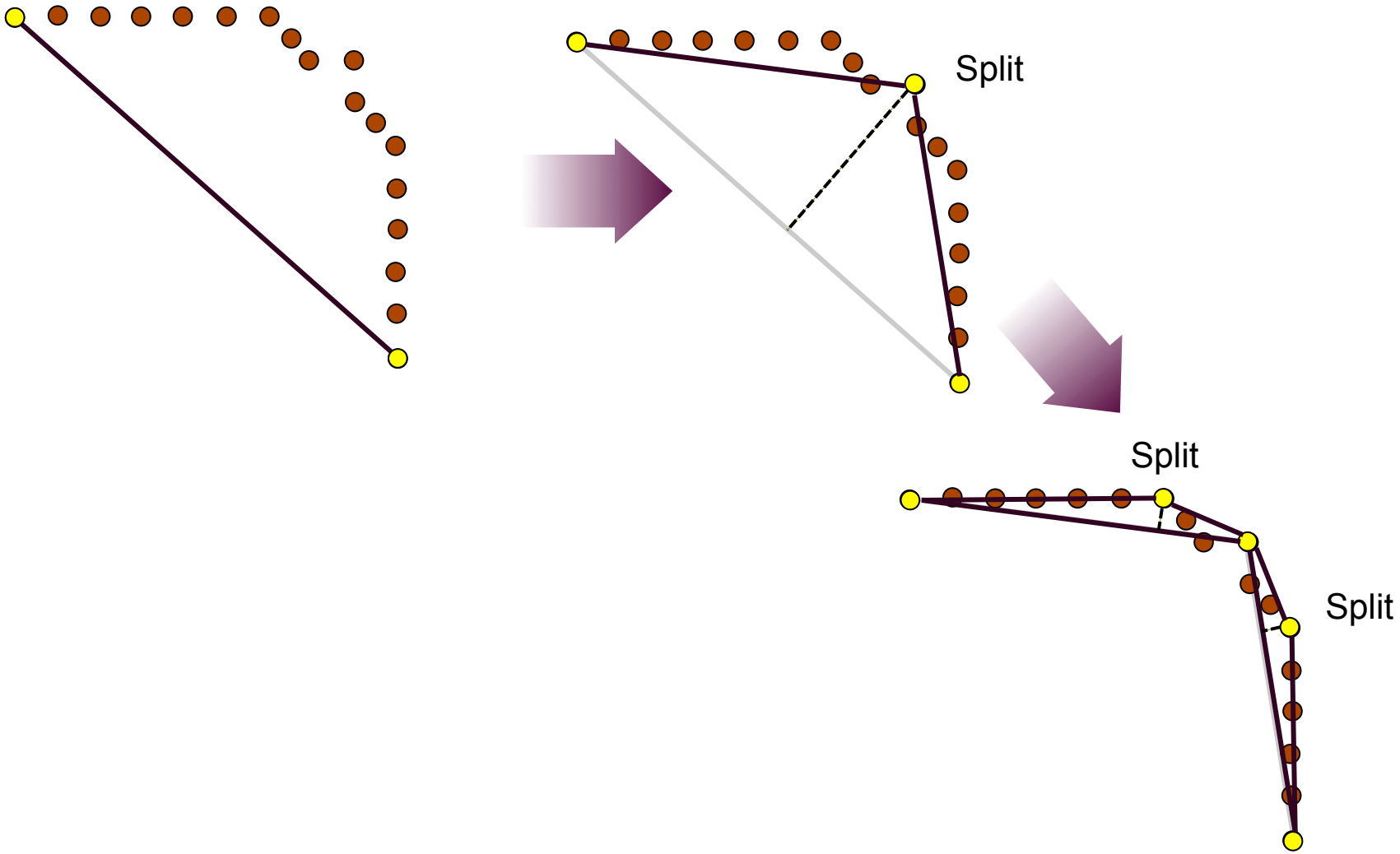
Line extraction | 1. split-and-merge (iterative end-point-fit)

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



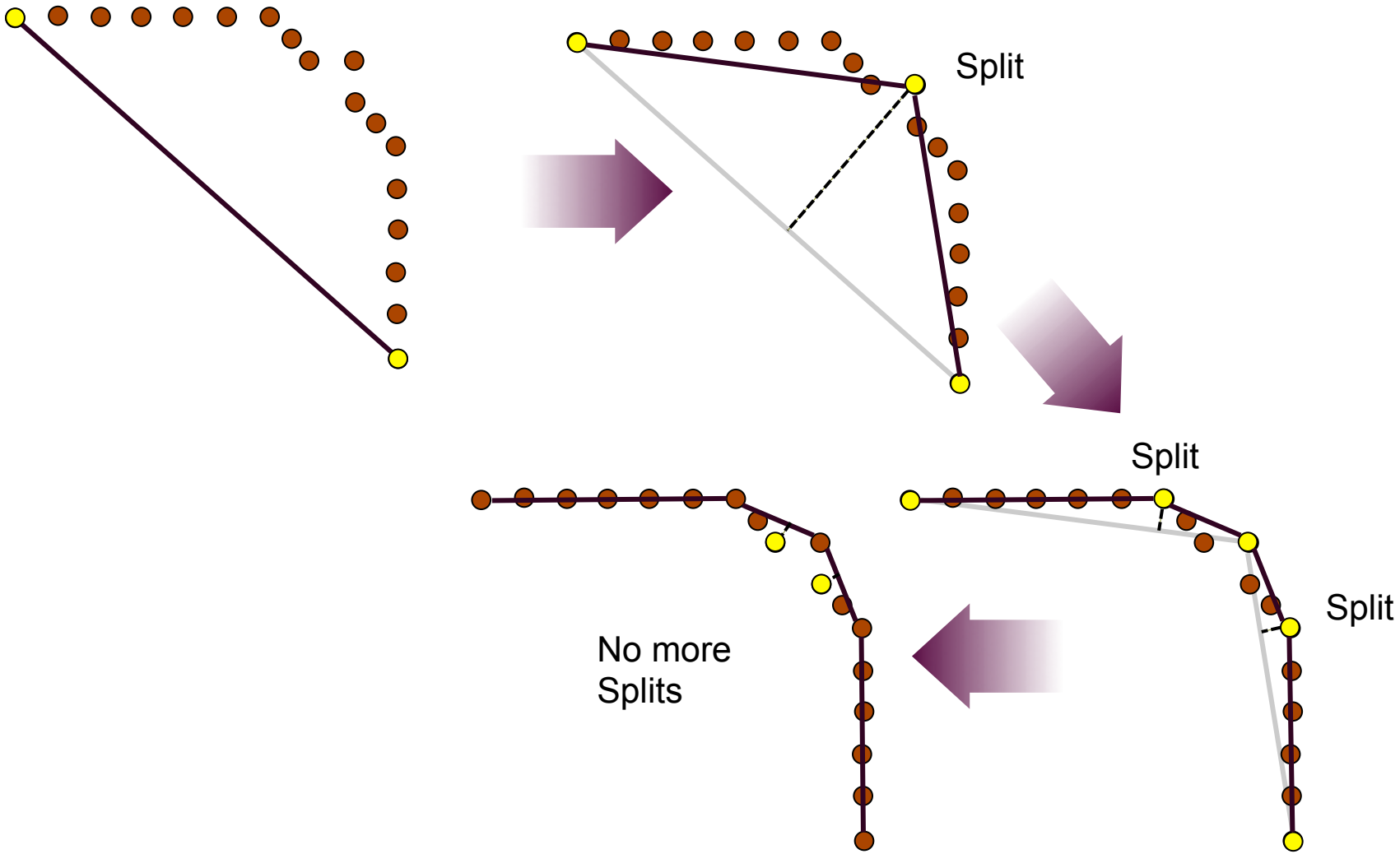
Line extraction | 1. split-and-merge (iterative end-point-fit)

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



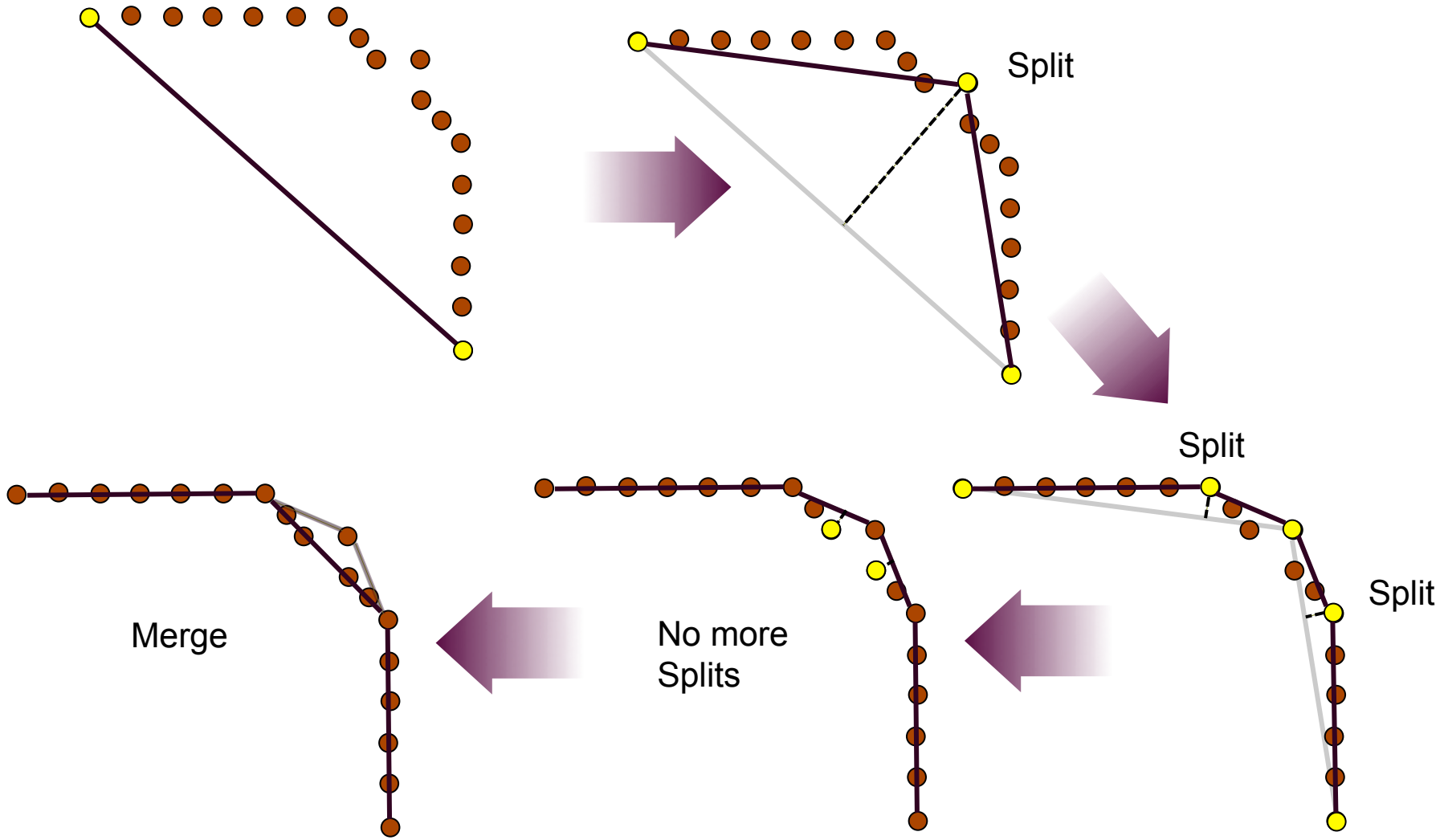
Line extraction | 1. split-and-merge (iterative end-point-fit)

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



Line extraction | 1. split-and-merge (iterative end-point-fit)

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

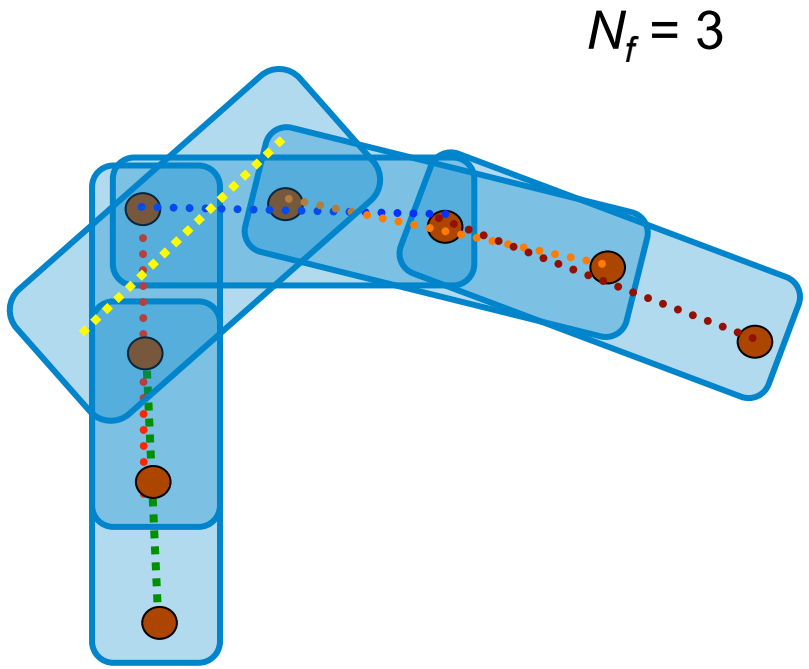


Line extraction | 2. line-regression

- "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 + re-compute line parameters for each segment

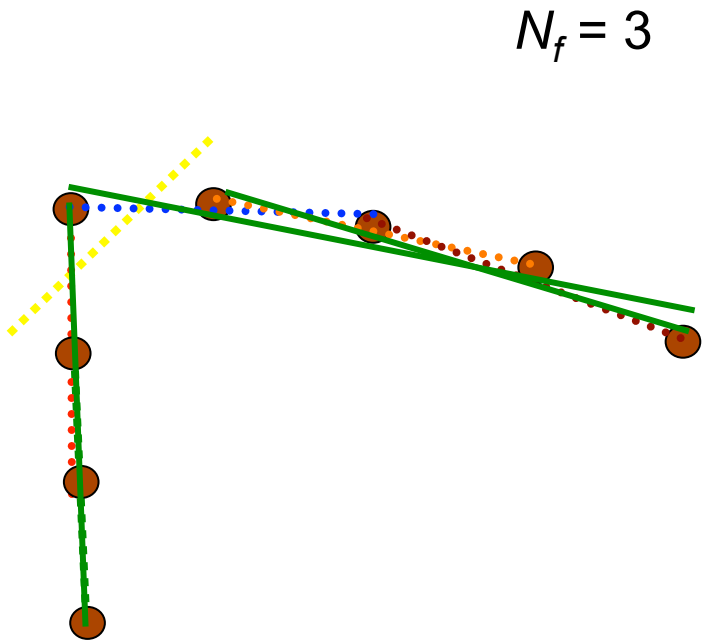


Line extraction | 2. line-regression

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

Line-Regression

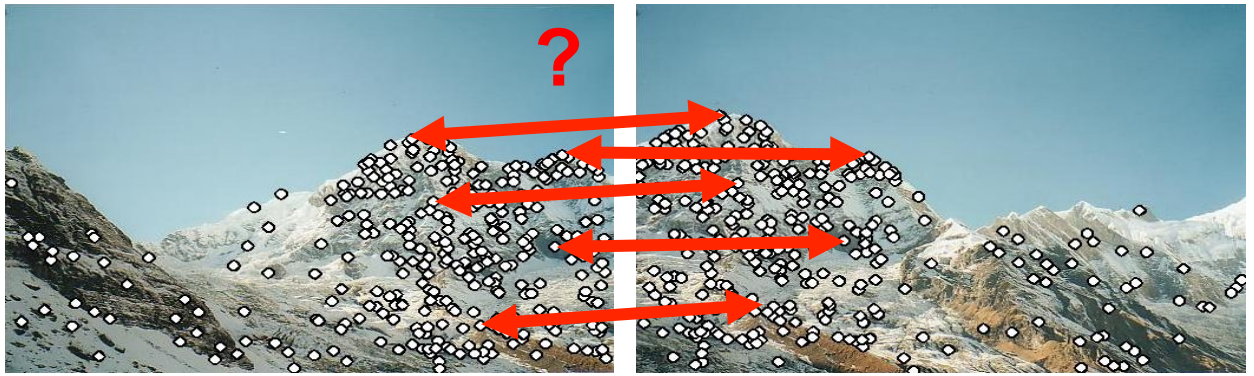
- Initialize sliding window size N_f
- Fit a line to every N_f consecutive points (i.e. in each window)
- Merge overlapping line segments + re-compute line parameters for each segment



Line extraction | 3. RANSAC

M. Fischler & R. C.Bolles. Random sample consensus: A paradigm for model fitting with applicatlons to image analysis and automated cartography. Graphics and Image Processing, 1981.

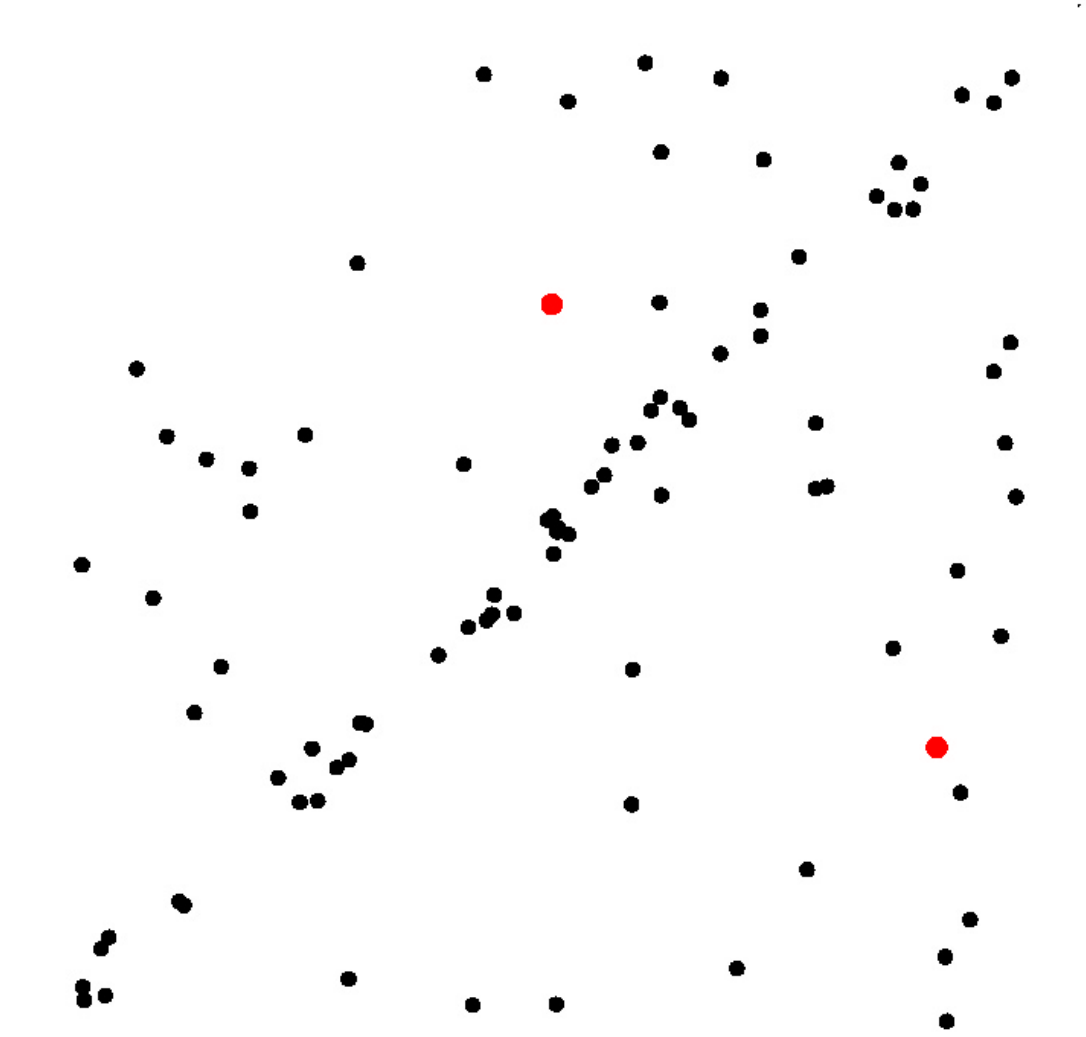
- **RANSAC** = **RAN**dom **SA**mple **C**onsensus.
- A generic & robust fitting algorithm of models in the presence of outliers (i.e. points which do not satisfy a model)
- Can be applied in general to any problem where the goal is to **identify the inliers which satisfy a predefined model**.



- 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.
- RANSAC is **iterative** and **non-deterministic** ⇨ the probability to find a set free of outliers increases as more iterations are used
- Drawback: a non-deterministic method, results are different between runs.

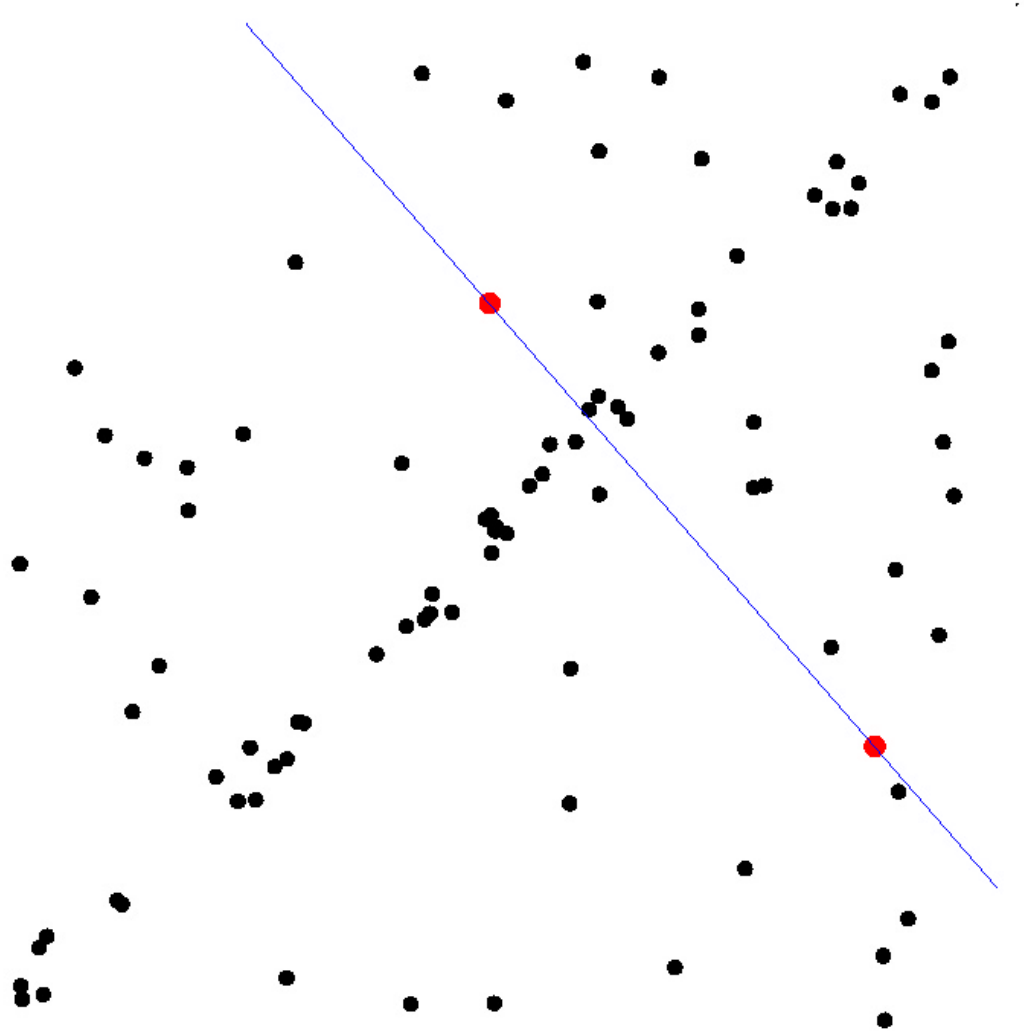
Line extraction | 3. RANSAC

Line extraction | 3. RANSAC



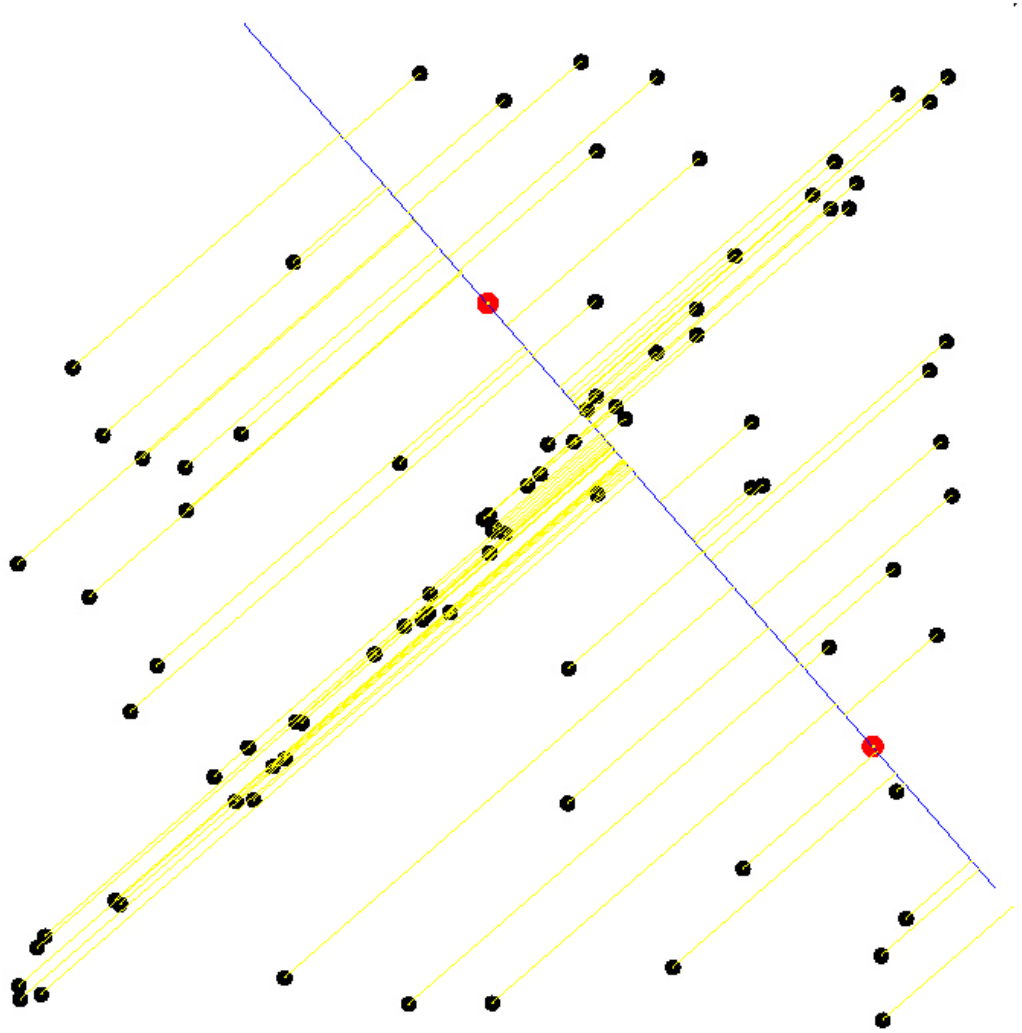
- **Select sample of 2 points at random**

Line extraction | 3. RANSAC



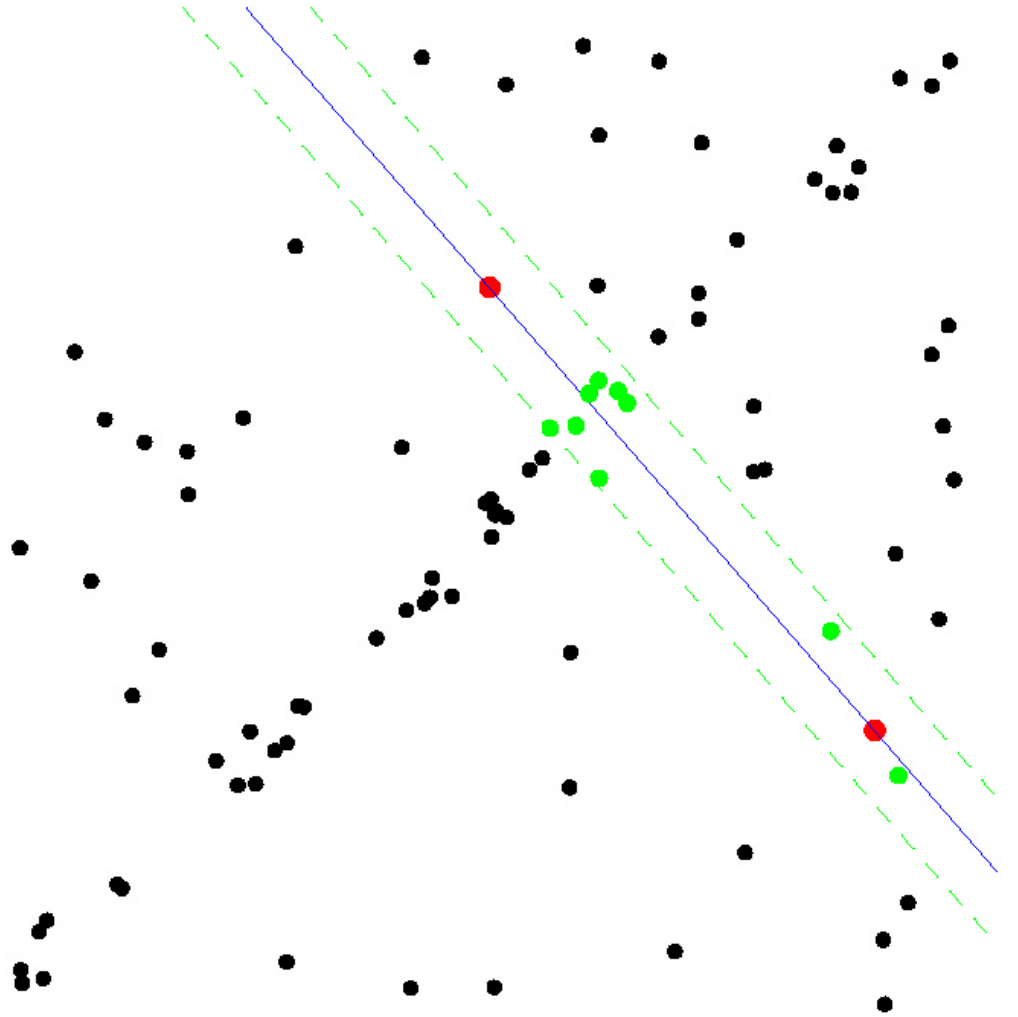
- Select sample of 2 points at random
- **Calculate model parameters that fit the data in the sample**

Line extraction | 3. RANSAC



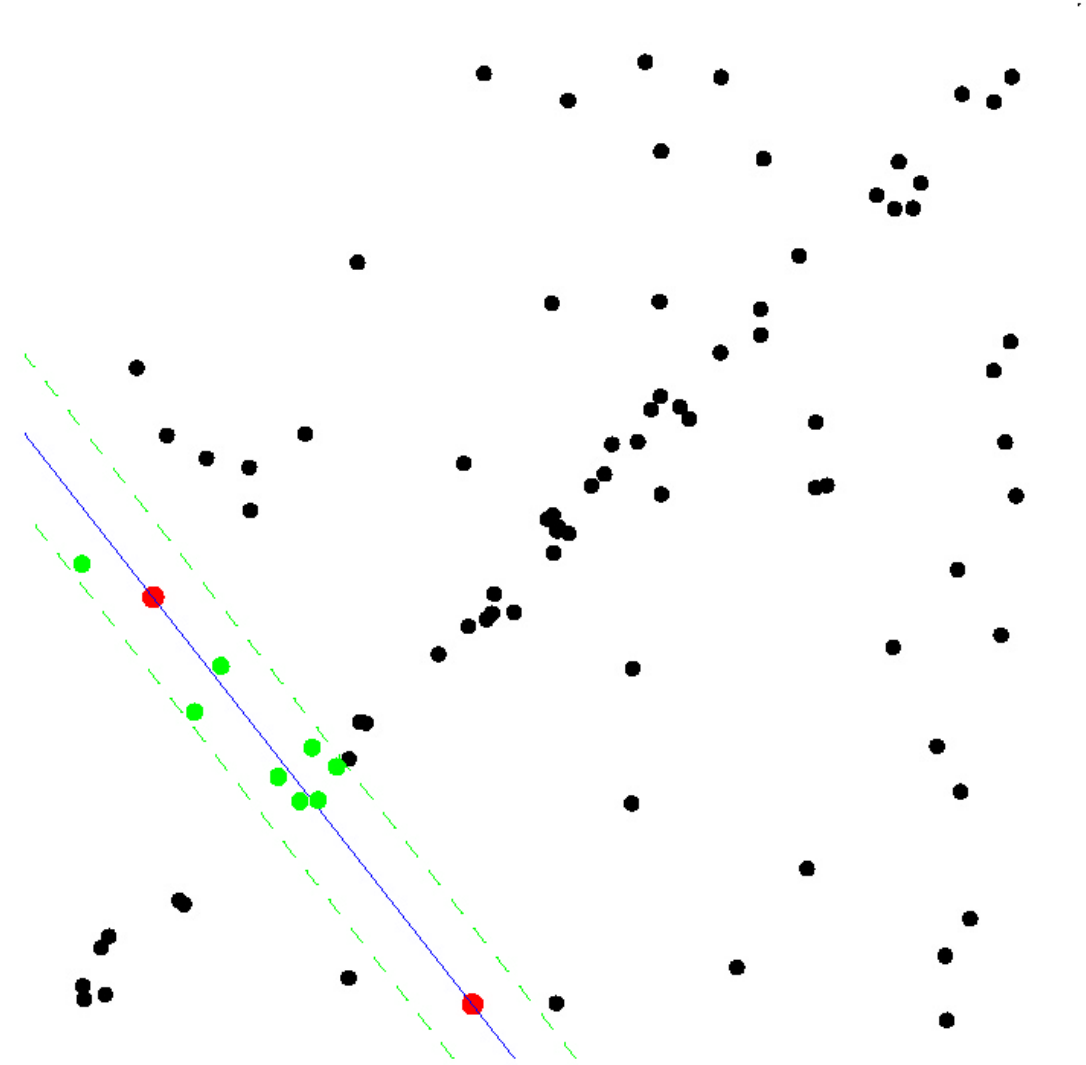
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- **Calculate error function for each data point**

Line extraction | 3. RANSAC



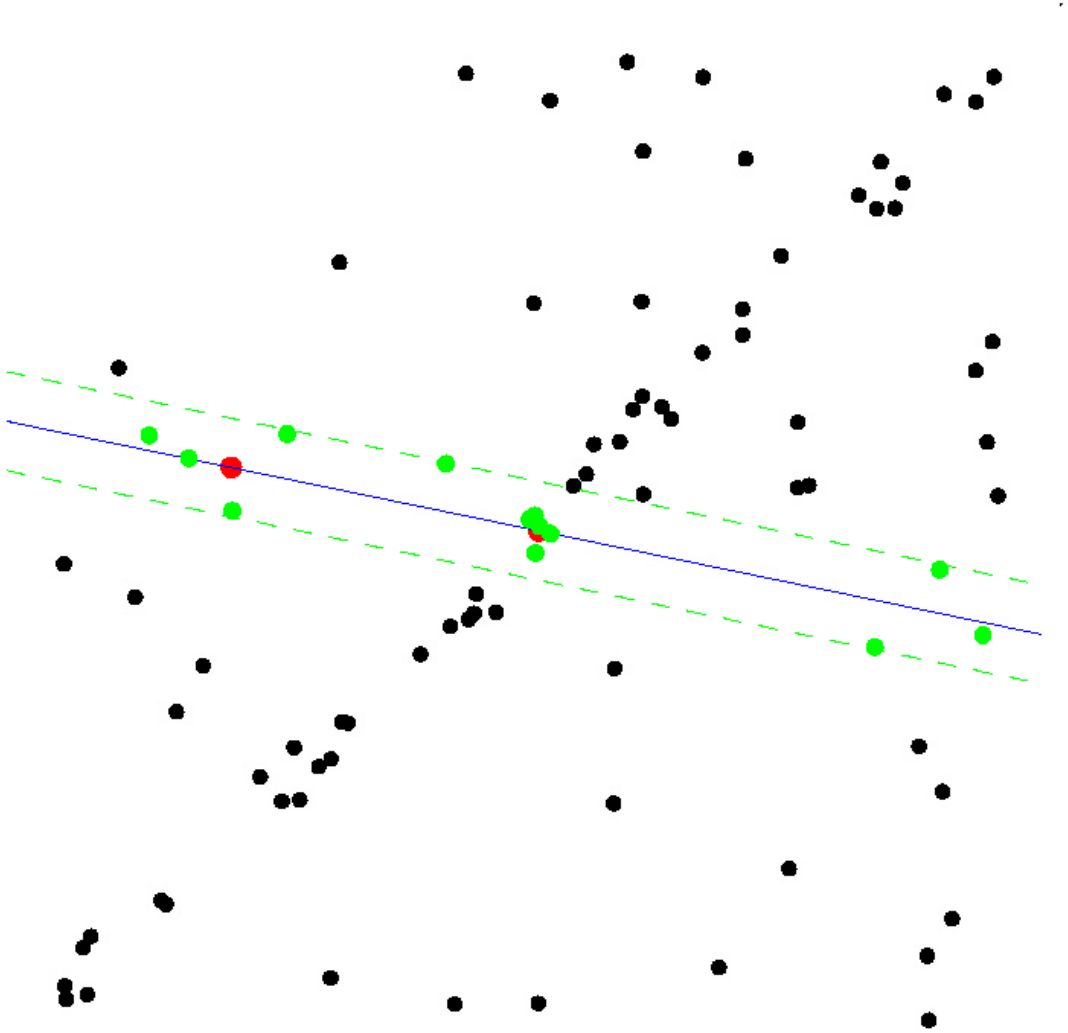
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- **Select data that supports current hypothesis**

Line extraction | 3. RANSAC



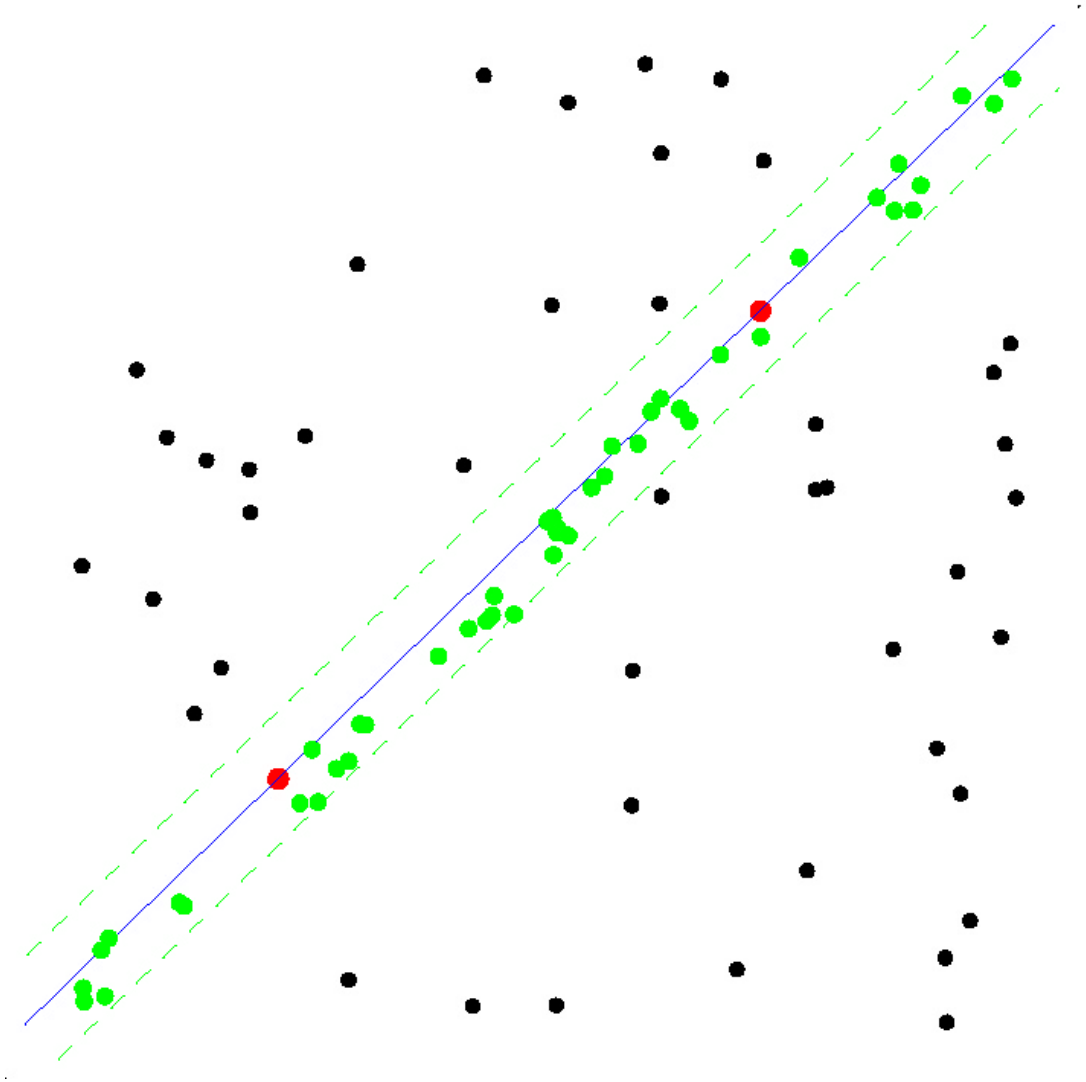
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that supports current hypothesis
- **Repeat sampling**

Line extraction | 3. 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**

Line extraction | 3. 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**

Set with the maximum number of inliers obtained after k iterations

Line extraction | 3. 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 points to fit a line through ⇨ 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

Line extraction | 3. RANSAC

How many iterations does RANSAC need?

- w := number of inliers / N

  N := tot. no. data points

  ⇨ w : fraction of inliers in the dataset ⇨ w = P(selecting an inlier-point out of the dataset)

- Let p := P(selecting a minimal set of points free of outliers)

- *Assumption*: the 2 points necessary to estimate a line are selected independently

  ⇨ w^2 = P(both selected points are inliers)

  ⇨ $1-w^2$ = P(at least one of these two points is an outlier)

- Let k := no. RANSAC iterations executed so far

  ⇨ $(1-w^2)^k$ = P(RANSAC never selects two points that are both inliers)

  ⇨ $1-p = (1-w^2)^k$ and therefore :

$$k = \frac{\log(1-p)}{\log(1-w^2)}$$

Line extraction | 3. RANSAC

How many iterations does RANSAC need?

- The number of iterations k is: $$k = \frac{\log(1-p)}{\log(1-w^2)}$$

⇨ knowing the fraction of inliers w, after k RANSAC iterations we will have a probability p of finding a set of points free of outliers

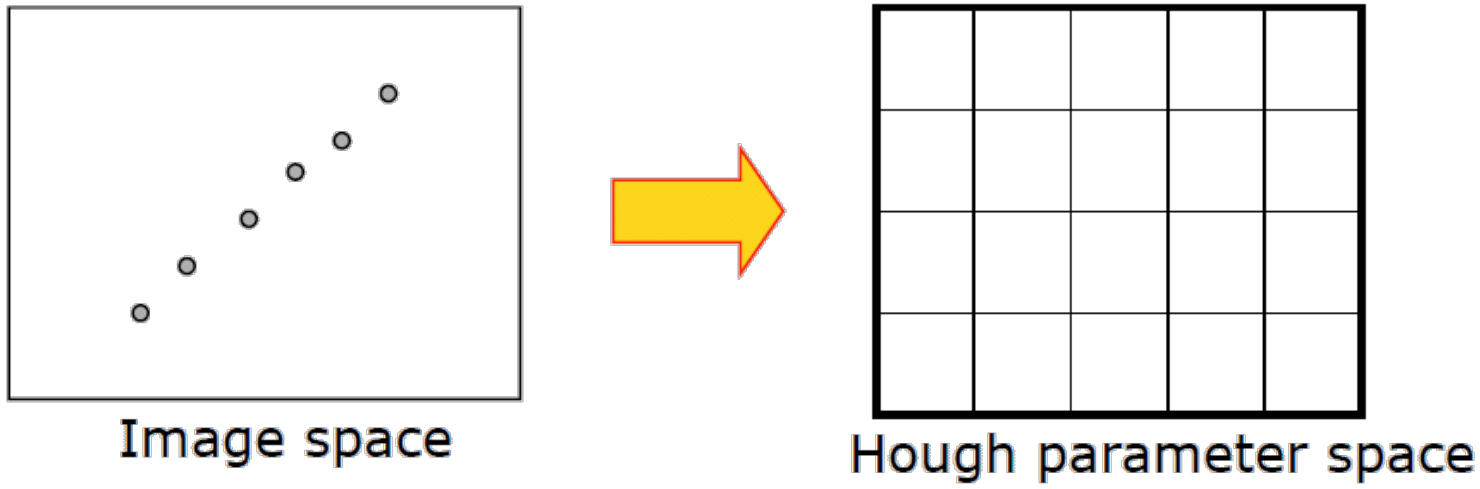
- Example: if we want a probability of success p=99% and we know that w=50% ⇨ k=16 iterations – these are dramatically fewer than the number of all possible combinations!

- Notice: the number of points does not influence the estimated number of iterations, only w does!

- In practice we need only a rough estimate of w.
  More advanced variants of RANSAC estimate the fraction of inliers and adaptively change it on every iteration

Line extraction | 4. Hough-Transform

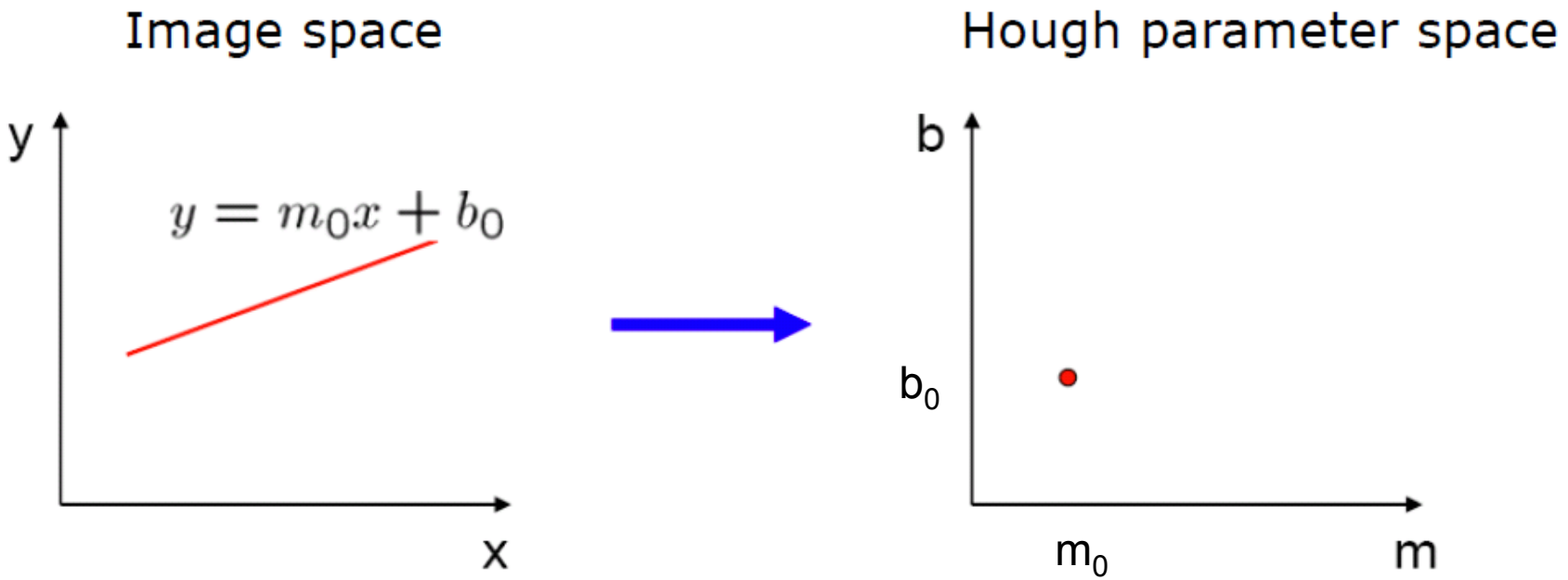
- Points vote for plausible line parameters
- Hough-Transform: maps image-space into Hough-space
- **Hough-space**: voting accummulator, parametrized w.r.t. line characteristics



1. P. Hough, Machine Analysis of Bubble Chamber Pictures, Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959
2. J. Richard, O. Duda, P.E. Hart (April 1971). "Use of the Hough Transformation to Detect Lines and Curves in Pictures". Artificial Intelligence Center (SRI International)

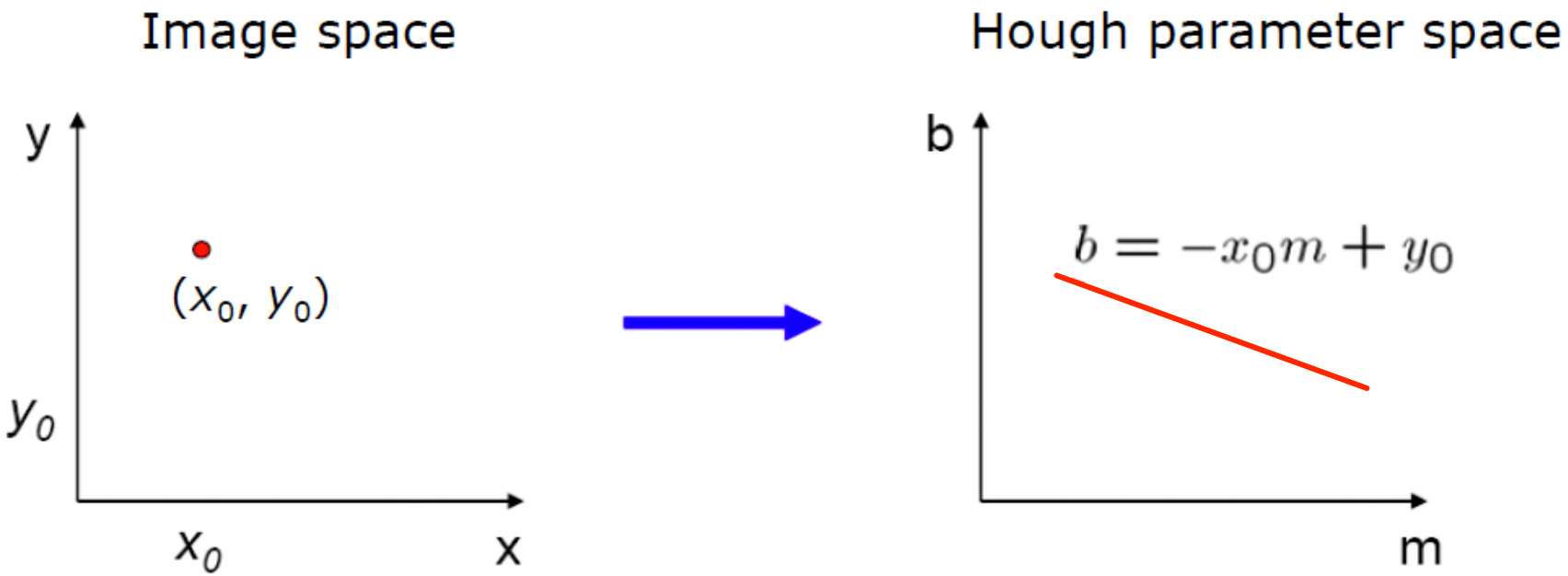
Line extraction | 4. Hough-Transform

- A line in the image corresponds to a point in Hough space



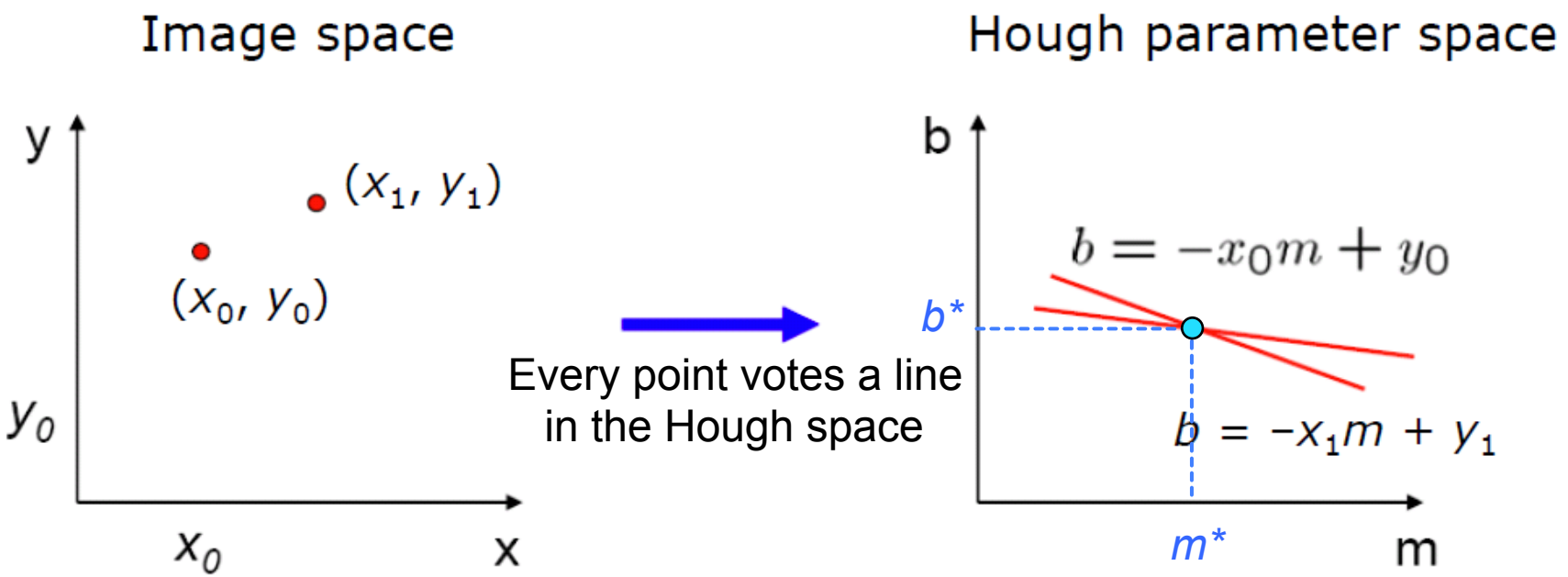
Line extraction | 4. Hough-Transform

- What does a point (x_0, y_0) in the image space map to in the Hough space?

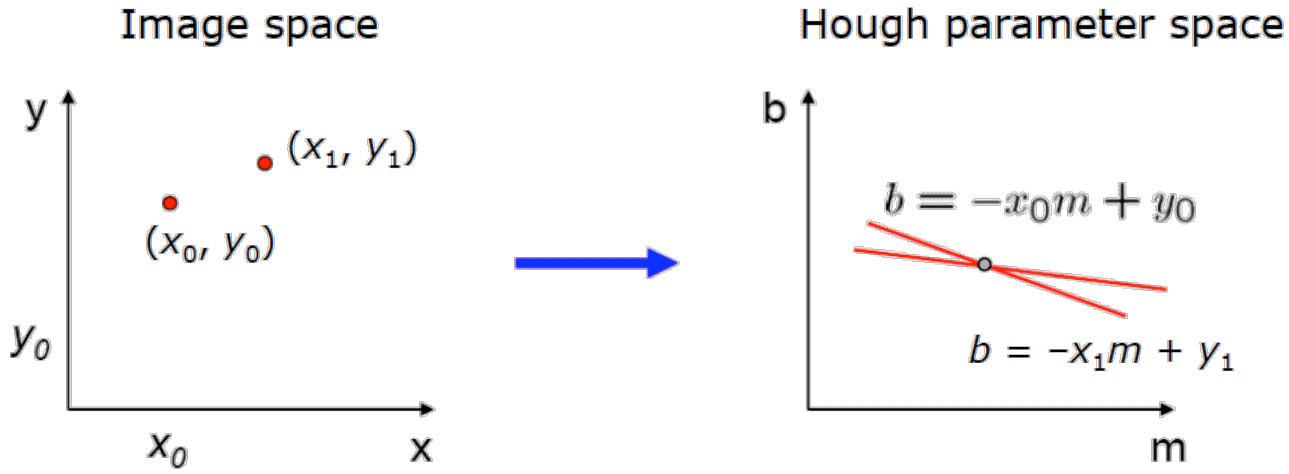


Line extraction | 4. Hough-Transform

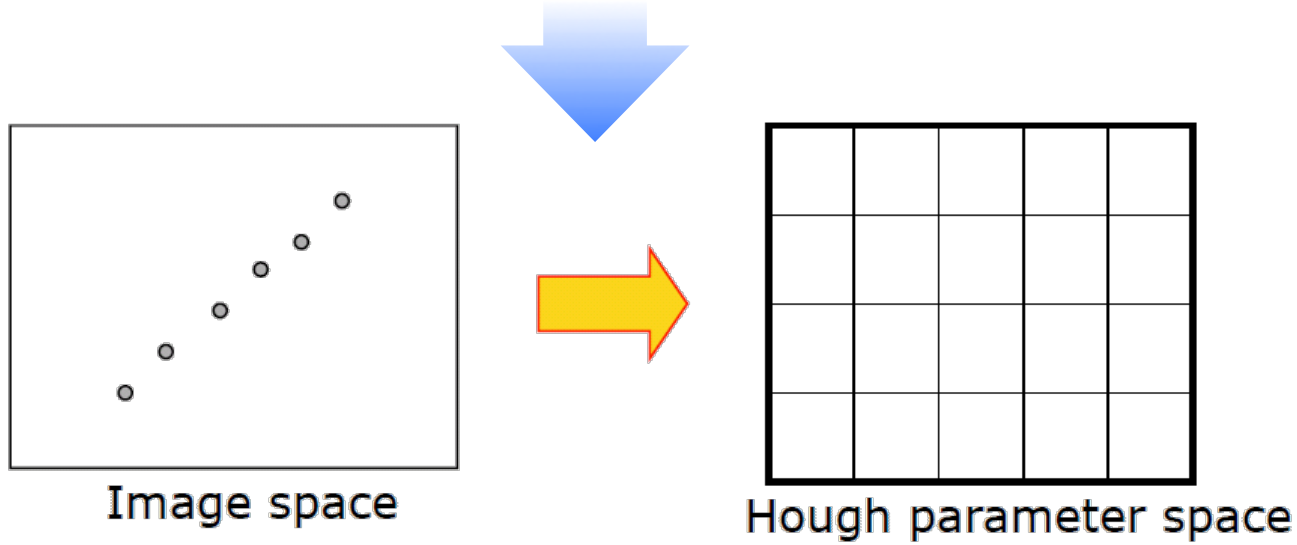
- Where is the line that contains both (x_0, y_0) and (x_1, y_1)?
  - It is the intersection of the lines $b = -x_0 m + y_0$ and $b = -x_1 m + y_1$



Line extraction | 4. Hough-Transform



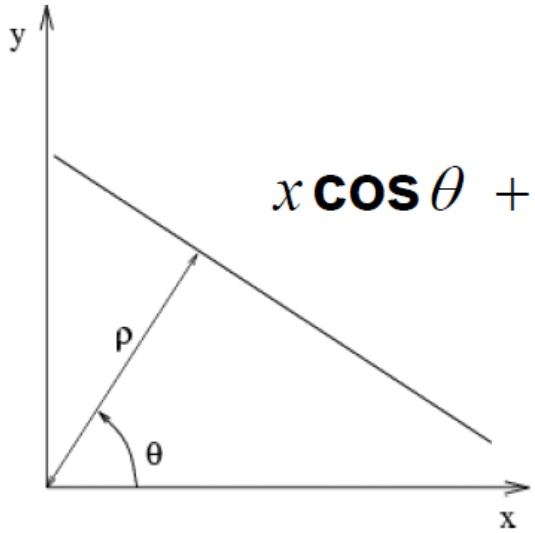
- Each point in image space, votes for line-parameters in Hough parameter space



Line extraction | 4. Hough-Transform

- Problems with the (m,b) space:
  - Unbounded parameter domain
  - How to represent lines aligned with the axes?

- Alternative: polar representation



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

Each point in image space will map to a sinusoid in the (ρ,θ) parameter space

Line extraction | 4. Hough-Transform

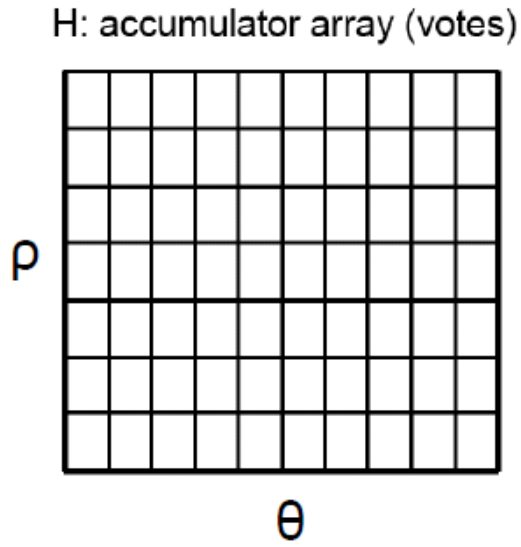
1. Initialize accumulator H to all zeros
2. **for** each edge point (x,y) in the image

   for all θ in [0,180]

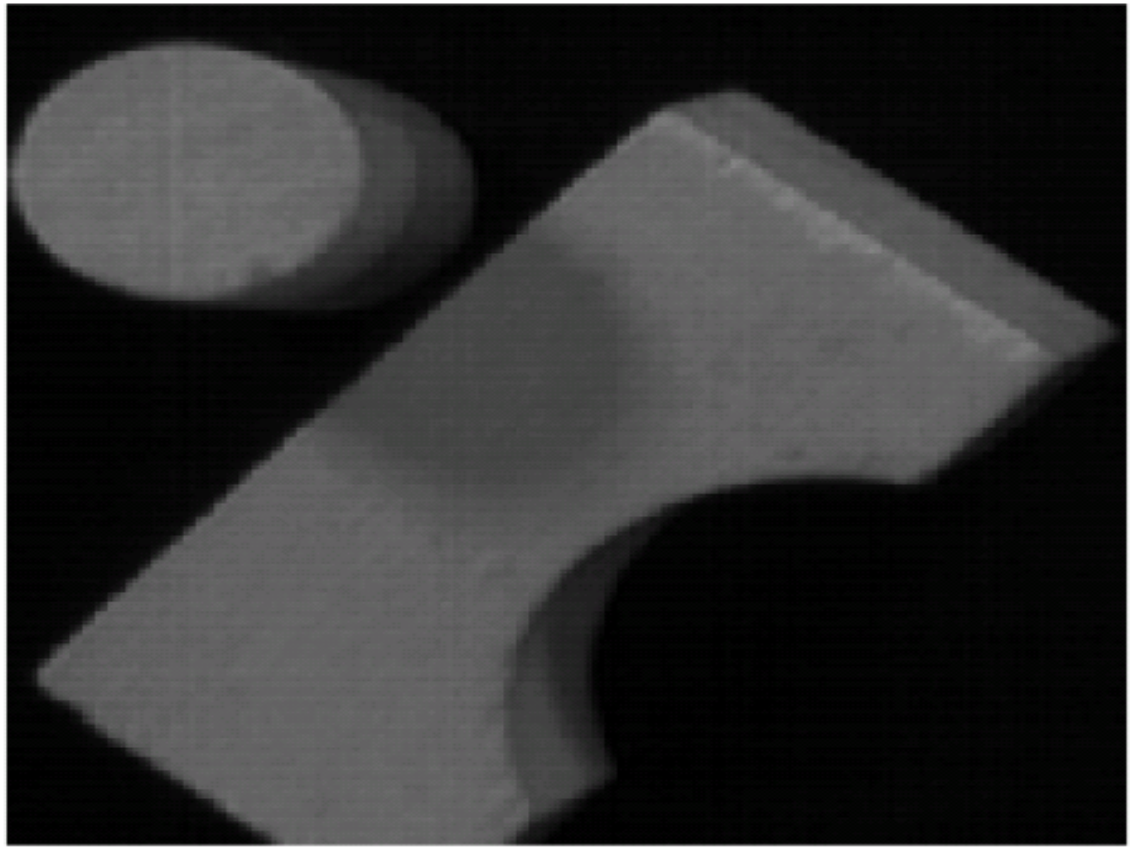
   - Compute ρ = x cos θ + y sin θ
   - H(θ, ρ) = H(θ, ρ) + 1

   end

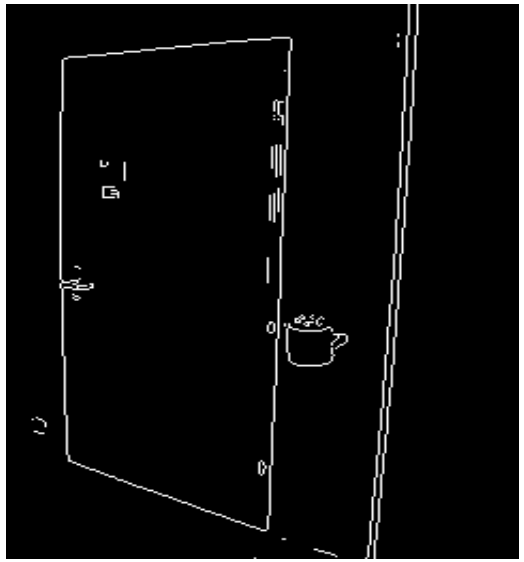
   end
3. Find the values of (θ, ρ) where H(θ, ρ) is a local maximum
4. The detected line in the image is given by: ρ = x cos θ + y sin θ

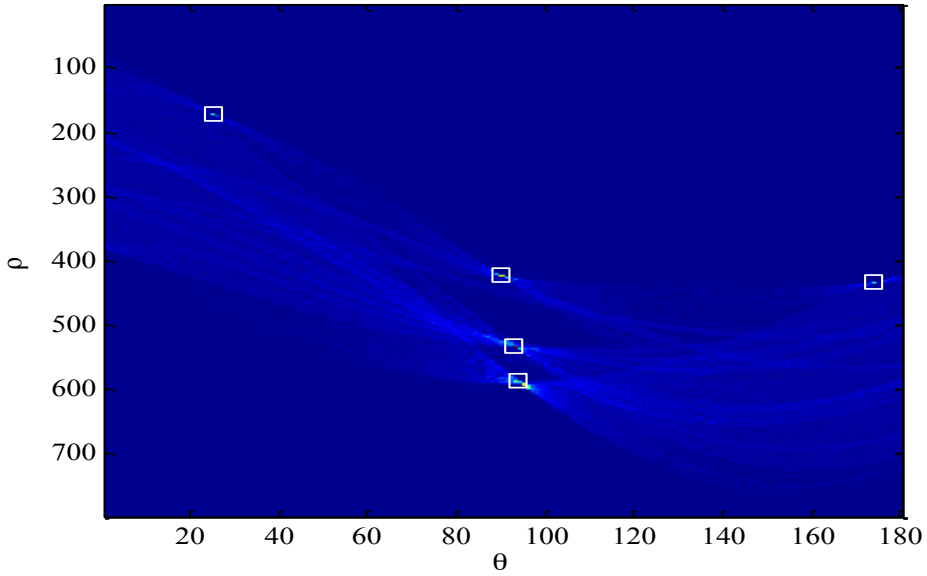


Line extraction | 4. Hough-Transform: examples

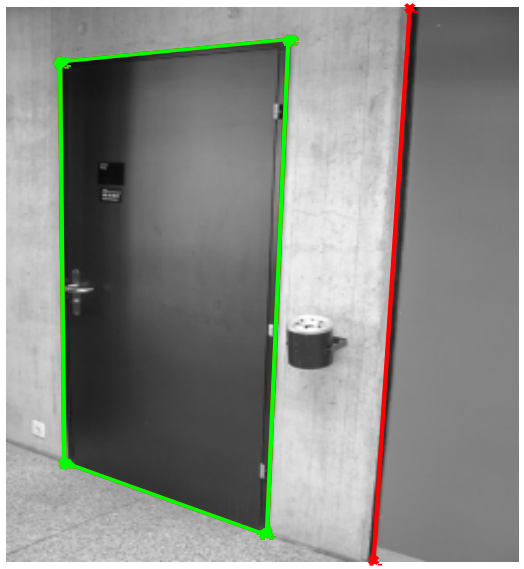


Line extraction | 4. Hough-Transform: examples

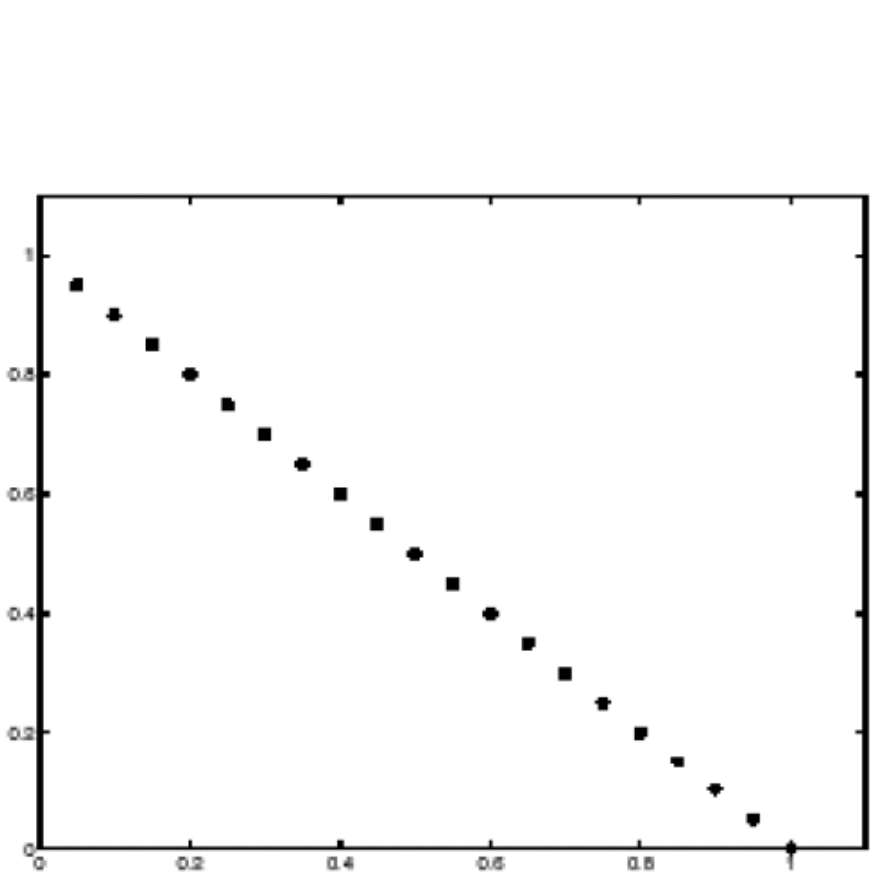




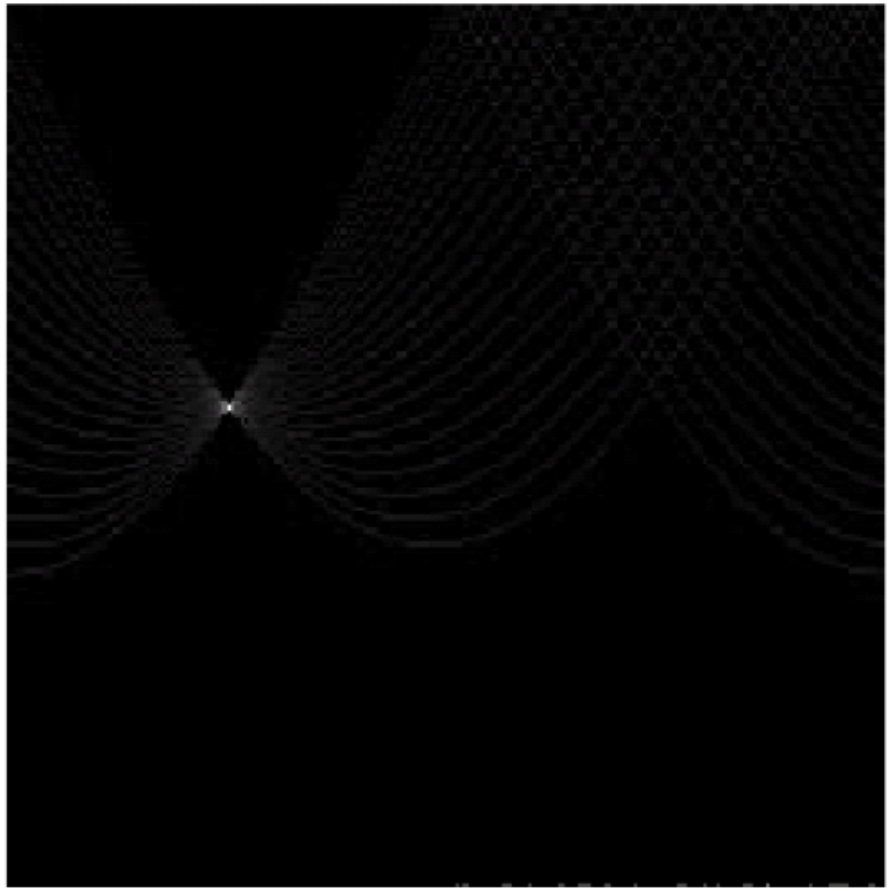
Hough Transform



Line extraction | 4. Hough-Transform: examples



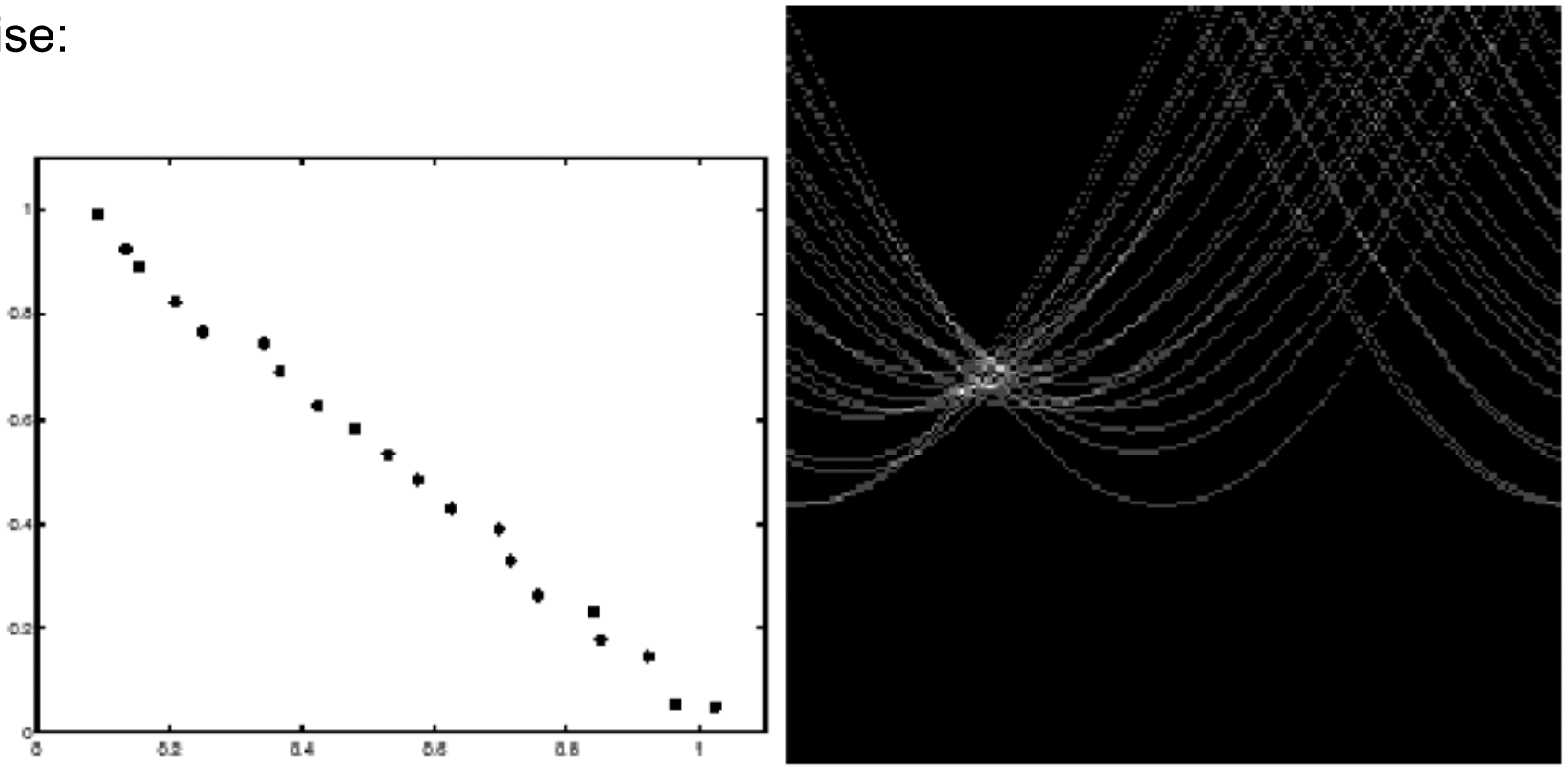
features



votes

Line extraction | 4. Hough-Transform: examples

Effect of Noise:



features votes

- Peak gets fuzzy and hard to locate

Line extraction | comparison

- Split-and-merge, Incremental and Line-Regression: fastest – best applied on laser scans
- Deterministic & make use of the sequential ordering of raw scan points
  (: points captured according to the rotation direction of the laser beam)
- If applied on randomly captured points only last 3 algorithms would segment all lines.
- RANSAC, Hough-Transform and EM produce greater precision ⇨ more robust to outliers

Comparison by [Nguyen et al. IROS 2005]

	Complexity	Speed (Hz)	False positives	Precision
Split-and-Merge	N logN	1500	10%	+++
Incremental	S N	600	6%	+++
Line-Regression	N N_f	400	10%	+++
RANSAC	S N k	30	30%	++++
Hough-Transform	S N N_C + S N_R N_C	10	30%	++++
Expectation Maximization	S N_1 N_2 N	1	50%	++++