Perception – Information Extraction –

CSC398 Autonomous Robots

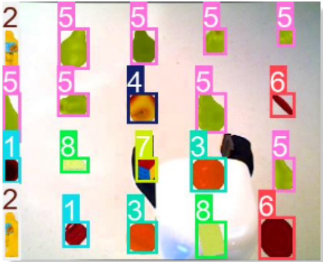
Ubbo Visser

Department of Computer Science
University of Miami

November 18, 2024

UNIVERSITY
OF MIAMI

Perception - Sensors for mobile robots



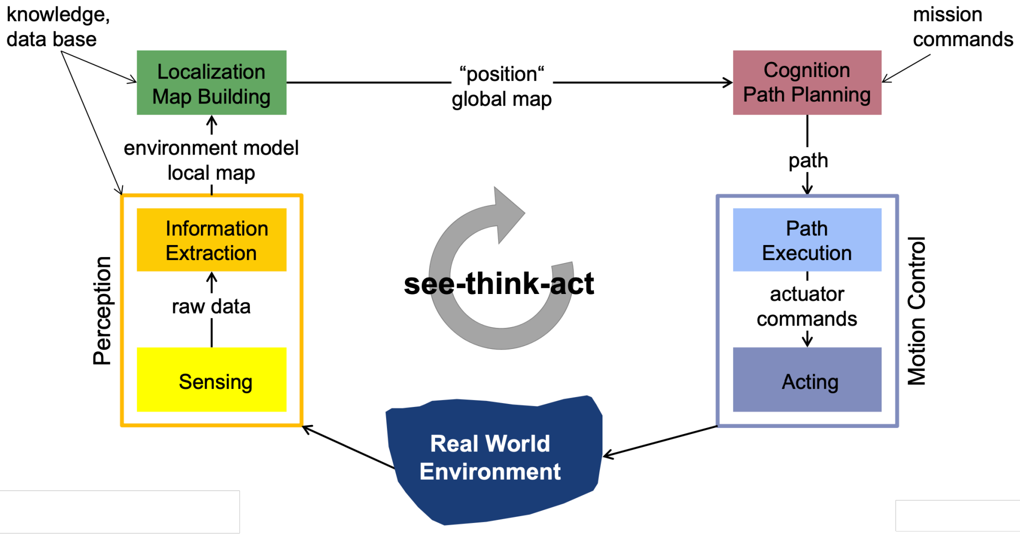
Aim

- Learn how to extract information from sensor measurements

Suggested Reading:

- *Introduction to Autonomous Mobile Robots* by Roland Siegwart, Illah Nourbakhsh, Davide Scaramuzza, The MIT Press, Sections: 4.1.3, 4.6.1 - 4.6.5, 4.7.1 - 4.7.4

Perception - Cognition - Action cycle



Source: Siegwart et. al (2018): Autonomous Mobile Robots, Lecture ETH Zürich

Object recognition

- **Object recognition: capability of naming discrete objects in the world**
- Why is it hard? Many reasons, including:
  - Real world is made of a jumble of objects, which all occlude one another and appear in different poses
  - There is a lot of variability intrinsic within each class (e.g., dogs)
- We will look at three methods:
  - Template matching
  - Bag of visual words
  - Neural network methods

Information extraction

- Next step is to extract **information** from images, such as
  - Geometric primitives (e.g., lines and circles): useful, for example, for robot localization and mapping
  - Object recognition and scene understanding: useful, for example, for localization within a topological map and for high-level reasoning

Information extraction

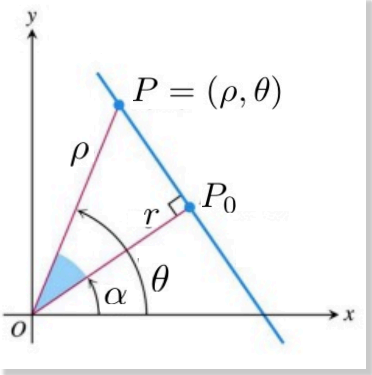
- **Geometric feature extraction:** extract geometric primitives from sensor data (e.g., range data)
- Examples: lines, circles, corners, planes, etc.
- We focus on line extraction from range data (a quite common task); other geometric feature extraction tasks are conceptually analogous
- The two main problems of line extraction from range data
    - Which points belong to which line? $\rightarrow$ *segmentation*
    - Given an association of points to a line, how do we estimate line parameters? $\rightarrow$ *fitting*

Step #2: line fitting

- **Goal:** fit a line to a set of sensor measurements
- It is useful to work in polar coordinates:
  $x = p\cos\theta, \quad y = \sin\theta$
- Equation of a line in polar coordinates
  - Let $P = (p, \theta)$ be an arbitrary point on the line
  - Since P, P_0, O determine a right triangle

$$\boxed{p\cos(\theta - \alpha) = r} \quad \text{or} \quad x\cos\alpha + y\sin\alpha = r \tag{1}$$

- (r, α) are the parameters of the line

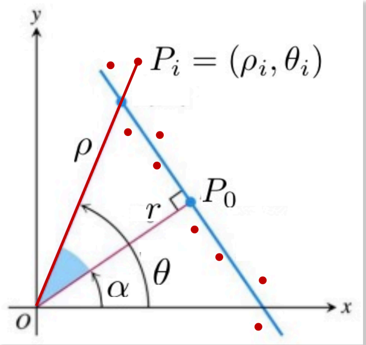


Step #2: Line Fitting

- Due to measurement errors, the equation of the line is only *approximately* satisfied:

$$p_i \cos(\theta_i - \alpha) = r + d_i \quad \longleftarrow \text{Error}$$

- Assume n measurement points represented in polar coordinates as (p_i, θ_i).
- Objective: Identify the line that best "fits" all the measurement points.



Step #2: Line Fitting

- Assume that all measurements have equal uncertainty.
- Find line parameters r, α that minimize the squared error:

$$S(r, \alpha) := \sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} (p_i \cos(\theta_i - \alpha) - r)^2$$

- Unweighted least squares

Step #2: Line Fitting

- Consider, now, the case where each measurement has its own, unique uncertainty
- For example, assume that the variance for each range measurement p_i is σ_i
- Associate with each measurement a weight, e.g., $w_i = 1/\sigma_i^2$
- Minimize

$$S(r, \alpha) := \sum_{i=1}^{n} w_i d_i^2 = \sum_{i=1}^{n} w_i (p_i \cos(\theta_i - \alpha) - r)^2$$

- Weighted least squares

Step #2: Line Fitting

- Assume that the n measurements are **independent**.
- Solution:

$$\alpha = \frac{1}{2}\,\text{atan2}\left(\frac{\sum_i w_i p_i^2 \sin 2\theta_i - \frac{2}{\sum_i w_i}\sum_i \sum_j w_i w_j p_i p_j \cos\theta_i \sin\theta_j}{\sum_i w_i p_i^2 \cos 2\theta_i - \frac{1}{\sum_i w_i}\sum_i \sum_j w_i w_j p_i p_j \cos(\theta_i + \theta_j)}\right) + \frac{\pi}{2}$$

$$r = \frac{\sum_i w_i p_i \cos(\theta - \alpha)}{\sum_i w_i}$$

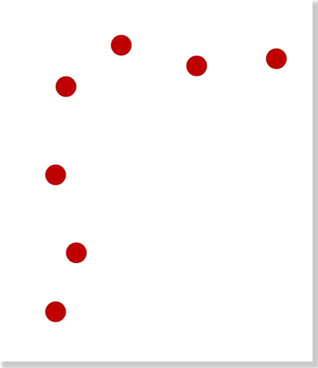
Step #1: Line Segmentation

- Several algorithms are available
- Here: three popular algorithms:
  - Split-and-merge
  - RANSAC
  - Hough-Transform

Split-and-Merge Algorithm

Most popular line extraction algorithm

```
Data: Set S consisting of all N points, a distance threshold d > 0
Output: L, a list of sets of points each resembling a line
L ← (S); i ← 1
while i ≤ len(L) do
    Fit a line (r, α) to the set L_i
    Detect the point P ∈ L_i with the maximum distance D to the line (r, α)
    if D < d then
        i ← i + 1
    else
        Split L_i at P into S_1 and S_2
        L_i ← S_1; L_{i+1} ← S_2
    end if
end while
Merge collinear sets in L
```



Split-and-Merge Algorithm

Most popular line extraction algorithm

Data: Set S consisting of all N points, a distance threshold $d > 0$

Output: L, a list of sets of points each resembling a line

$L \leftarrow (S)$; $i \leftarrow 1$

while $i \leq \text{len}(L)$ **do**

  Fit a line (r, α) to the set L_i

  Detect the point $P \in L_i$ with the maximum distance D to the line (r, α)

  if $D < d$ **then**

    $i \leftarrow i + 1$

  else

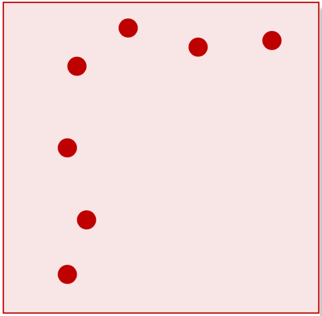
    Split L_i at P into S_1 and S_2

    $L_i \leftarrow S_1$; $L_{i+1} \leftarrow S_2$

  end if

end while

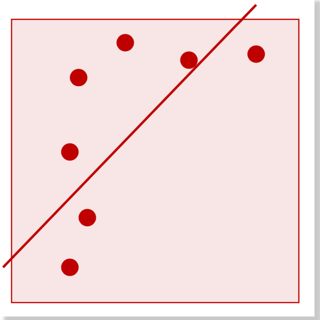
Merge collinear sets in L



Split-and-Merge Algorithm

Most popular line extraction algorithm

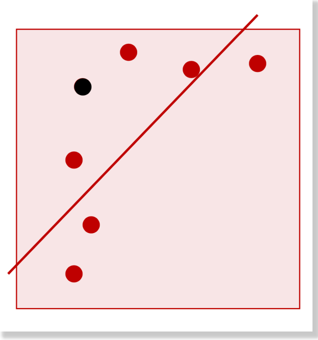
Data: Set S consisting of all N points, a distance threshold $d > 0$
Output: L, a list of sets of points each resembling a line
$L \leftarrow (S)$; $i \leftarrow 1$
while $i \leq \text{len}(L)$ **do**
    Fit a line (r, α) to the set L_i
    Detect the point $P \in L_i$ with the maximum distance D to the line (r, α)
    if $D < d$ **then**
        $i \leftarrow i + 1$
    else
        Split L_i at P into S_1 and S_2
        $L_i \leftarrow S_1$; $L_{i+1} \leftarrow S_2$
    end if
end while
Merge collinear sets in L



Split-and-Merge Algorithm

Most popular line extraction algorithm

Data: Set S consisting of all N points, a distance threshold $d > 0$
Output: L, a list of sets of points each resembling a line
$L \leftarrow (S)$; $i \leftarrow 1$
while $i \leq \text{len}(L)$ **do**
    Fit a line (r, α) to the set L_i
    Detect the point $P \in L_i$ with the maximum distance D to the line (r, α)
    if $D < d$ **then**
        $i \leftarrow i + 1$
    else
        Split L_i at P into S_1 and S_2
        $L_i \leftarrow S_1$; $L_{i+1} \leftarrow S_2$
    end if
end while
Merge collinear sets in L



Split-and-Merge Algorithm

Most popular line extraction algorithm

Data: Set S consisting of all N points, a distance threshold $d > 0$

Output: L, a list of sets of points each resembling a line

$L \leftarrow (S)$; $i \leftarrow 1$

while $i \leq \text{len}(L)$ **do**

    Fit a line (r, α) to the set L_i

    Detect the point $P \in L_i$ with the maximum distance D to the line (r, α)

    if $D < d$ **then**

        $i \leftarrow i + 1$

    else

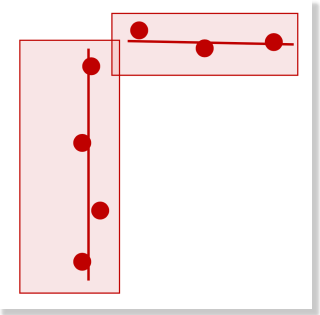
        Split L_i at P into S_1 and S_2

        $L_i \leftarrow S_1$; $L_{i+1} \leftarrow S_2$

    end if

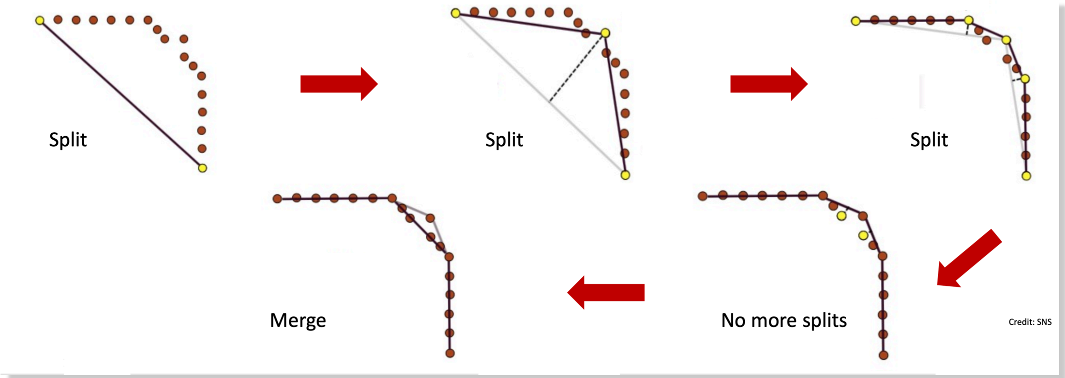
end while

Merge collinear sets in L



Split-and-merge: iterative-end-point-fit variant

Iterative-end-point-fit: split-and-merge where the line is constructed by simply connecting the first and last points (as opposed to least squares fit)



RANSAC

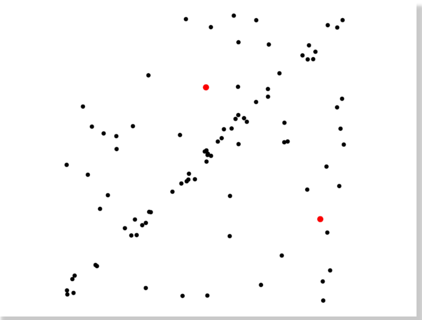
- RANSAC: **Ran**dom **Sa**mple **C**onsensus
- General method to estimate parameters of a model from a set of observed data in the presence of outliers, where outliers should not influence the estimates of the values
- Typical applications in robotics: line extraction from 2D range data, plane extraction from 3D point clouds, feature matching for structure from motion, etc.
- RANSAC is **iterative** and **non-deterministic**: the probability of finding a set free of outliers increases as more iterations are used

RANSAC

Data: Set S consisting of all N points
Output: Set with the maximum number of inliers (and corresponding fitting line)
for $i = 1$ to k **do**
    Randomly select two points from S
    Fit line l_i through the two selected points
    Compute the distance of all other points to line l_i
    Construct the *inlier set* by counting the number of points with distance to the line less than γ
    Store line l_i and associated set of inliers
end for
Choose the set with the maximum number of inliers

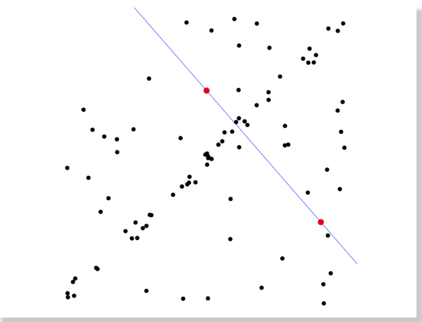
RANSAC

Data: Set S consisting of all N points
Output: Set with the maximum number of inliers (and corresponding fitting line)
for $i = 1$ to k **do**
    Randomly select two points from S
    Fit line l_i through the two selected points
    Compute the distance of all other points to line l_i
    Construct the *inlier set* by counting the number of points with distance to the line less than γ
    Store line l_i and associated set of inliers
end for
Choose the set with the maximum number of inliers



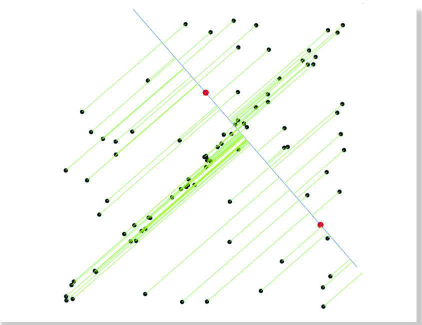
RANSAC

Data: Set S consisting of all N points
Output: Set with the maximum number of inliers
(and corresponding fitting line)
for $i = 1$ to k **do**
    Randomly select two points from S
    Fit line l_i through the two selected points
    Compute the distance of all other points to line l_i
    Construct the *inlier set* by counting the number
        of points with distance to the line less than γ
    Store line l_i and associated set of inliers
end for
Choose the set with the maximum number of inliers



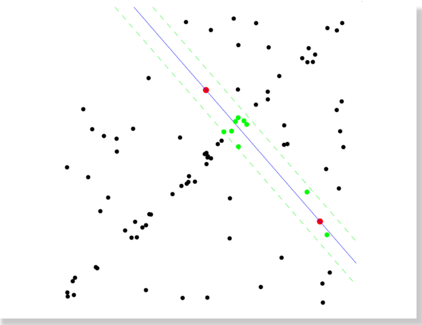
RANSAC

Data: Set S consisting of all N points
Output: Set with the maximum number of inliers (and corresponding fitting line)
for $i = 1$ to k **do**
    Randomly select two points from S
    Fit line l_i through the two selected points
    Compute the distance of all other points to line l_i
    Construct the *inlier set* by counting the number of points with distance to the line less than γ
    Store line l_i and associated set of inliers
end for
Choose the set with the maximum number of inliers



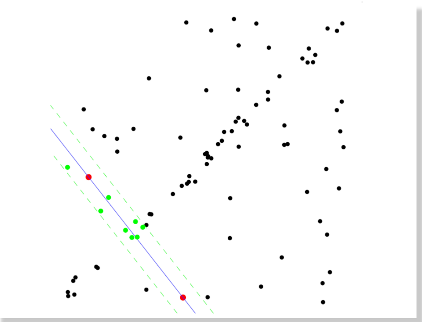
RANSAC

1: **Data:** Set S consisting of all N points
2: **Output:** Set with the maximum number of inliers (and corresponding fitting line)
3: **for** $i = 1$ to k **do**
4: Randomly select two points from S
5: Fit line l_i through the two selected points
6: Compute the distance of all other points to line l_i
7: Construct the *inlier set* by counting the number of points with distance to the line less than γ
8: Store line l_i and associated set of inliers
9: **end for**
10: Choose the set with the maximum number of inliers



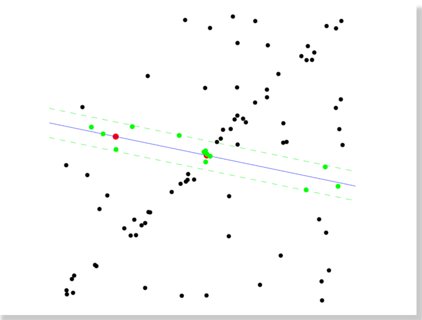
RANSAC

1: **Data:** Set S consisting of all N points
2: **Output:** Set with the maximum number of inliers (and corresponding fitting line)
3: **for** $i = 1$ to k **do**
4: Randomly select two points from S
5: Fit line l_i through the two selected points
6: Compute the distance of all other points to line l_i
7: Construct the *inlier set* by counting the number of points with distance to the line less than γ
8: Store line l_i and associated set of inliers
9: **end for**
10: Choose the set with the maximum number of inliers



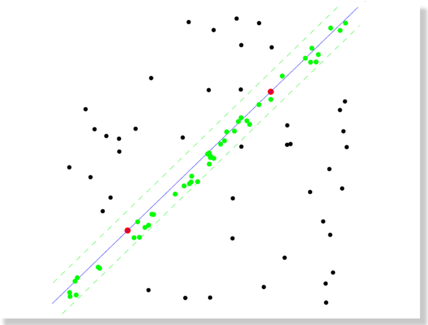
RANSAC

Data: Set S consisting of all N points
Output: Set with the maximum number of inliers (and corresponding fitting line)
for $i = 1$ to k **do**
    Randomly select two points from S
    Fit line l_i through the two selected points
    Compute the distance of all other points to line l_i
    Construct the *inlier set* by counting the number of points with distance to the line less than γ
    Store line l_i and associated set of inliers
end for
Choose the set with the maximum number of inliers



RANSAC

Data: Set S consisting of all N points
Output: Set with the maximum number of inliers (and corresponding fitting line)
for $i = 1$ to k **do**
    Randomly select two points from S
    Fit line l_i through the two selected points
    Compute the distance of all other points to line l_i
    Construct the *inlier set* by counting the number of points with distance to the line less than γ
    Store line l_i and associated set of inliers
end for
Choose the set with the maximum number of inliers



RANSAC iterations

- In principle, one would need to check all possible combinations of 2 points in dataset
- If $|S| = N$, number of combinations is $\frac{N(N-1)}{2} \rightarrow$ too many
- However, if we have a rough estimate of the percentage of inliers, we do not need to check all combinations...

RANSAC iterations: statistical characterization

- Let w be the percentage of inliers in the dataset, i.e.,

$$w = \frac{\#\text{of inliers}}{N}$$

- Let p be the desired probability of finding a set of points free of outliers (typically, $p = 0.99$)
- Assumption: 2 points chosen for line estimation I selected independently
    - $P(\text{both points selected are inliers}) = w^2$
    - $P(\text{at least one of the selected points is an outlier}) = 1 - w^2$
    - $P(\text{RANSAC never selects two points that are both inliers}) = (1 - w^2)^k$

RANSAC iterations: statistical characterization

- Then, the minimum number of iterations $\bar{k}$ to find an outlier-free set with probability, at least p is:

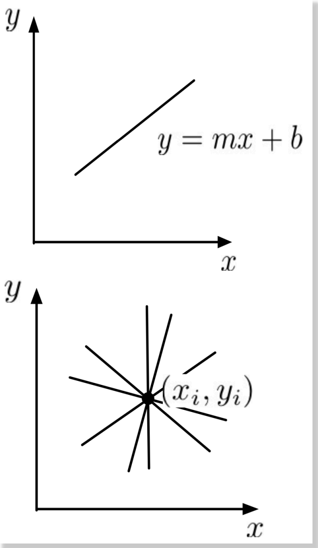
$$1 - p = (1 - w^2)^{\bar{k}} \Rightarrow \bar{k} = \frac{log(1 - p)}{log(1 - w^2)}$$

- Thus if we know w (at least approximately), after $\bar{k}$ iterations RANSAC will find a set free of outliers with probability p
- Note:
  - $\bar{k}$ depends only on w, not on N!
  - More advanced versions of RANSAC estimate w adaptively

Hough Transform

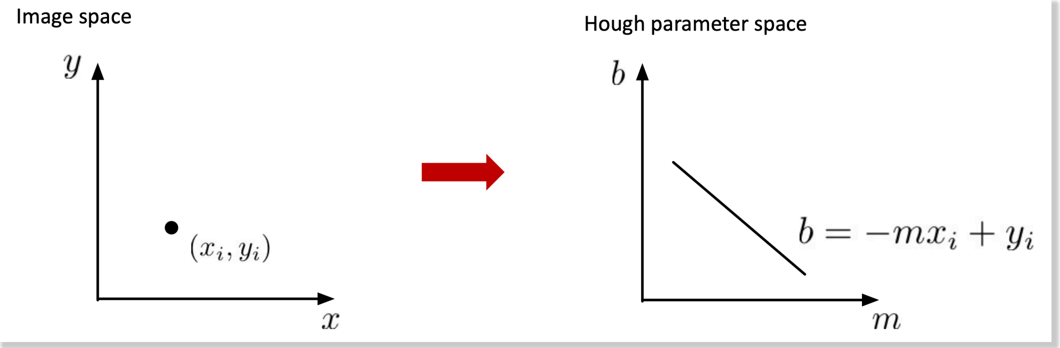
- **Key idea:** Each point votes for a *set* of plausible line parameters.
- A line has two parameters: (m, b).
- Given a point (x_i, y_i), the lines that could pass through this point are all (m, b) satisfying:

$$y_i = mx_i + b, \quad \text{or} \quad b = -mx_i + y_i$$



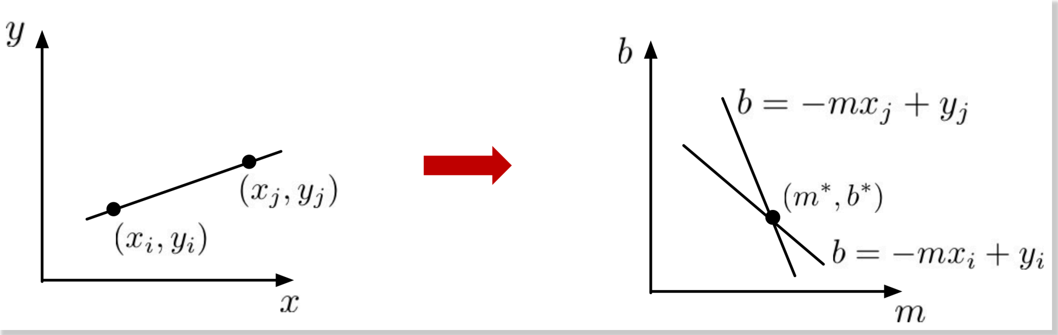
Hough Transform

- A point in image space maps into a line in *Hough space*



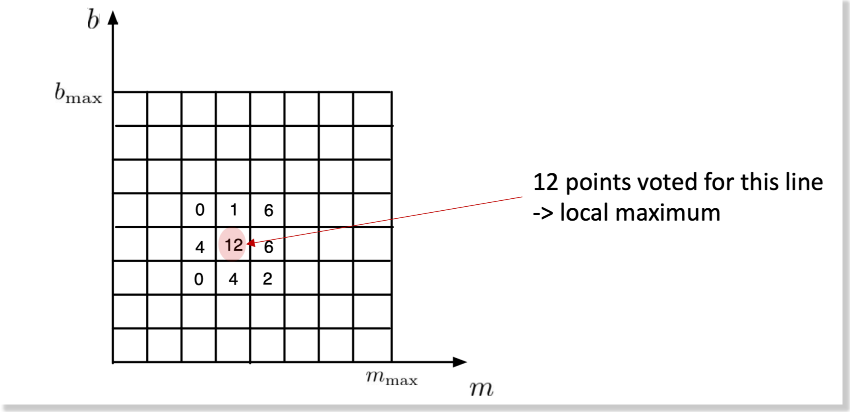
Hough Transform

- **Key fact:** all points on a line in image space yield lines in the parameter space which intersects at a *common point*, $(m*, b*)$



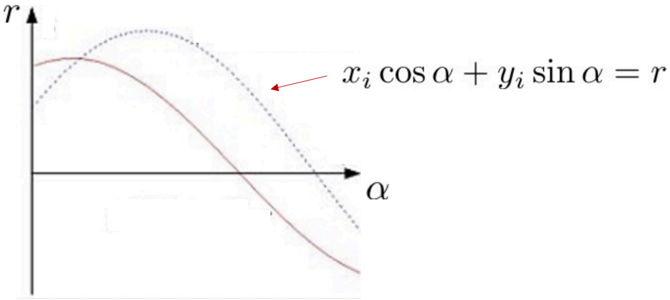
Hough transform algorithm

1: initialize accumulator array $H(m, b)$ to zero
2: for each point (x_i, y_i), increment all cells that satisfy $b = -x_i m + y_i$
3: local Maxima in array $H(m, b)$ corresponds to lines



Hough transform algorithm: polar coordinate representation

- Equation of a line in polar coordinates $x \cos \alpha + y \sin \alpha = r$
- The parameter space transform of a point is a sinusoidal curve

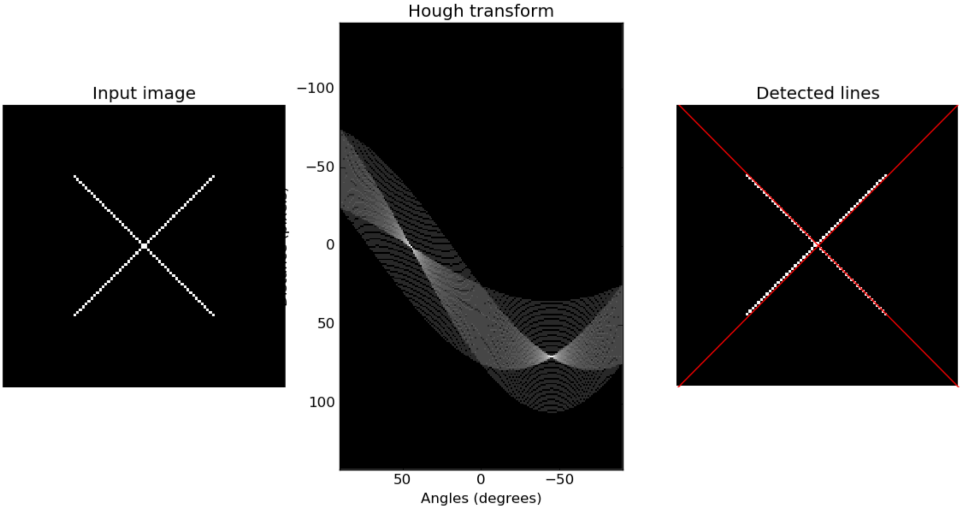


- Avoids infinite slope
- Constant resolution

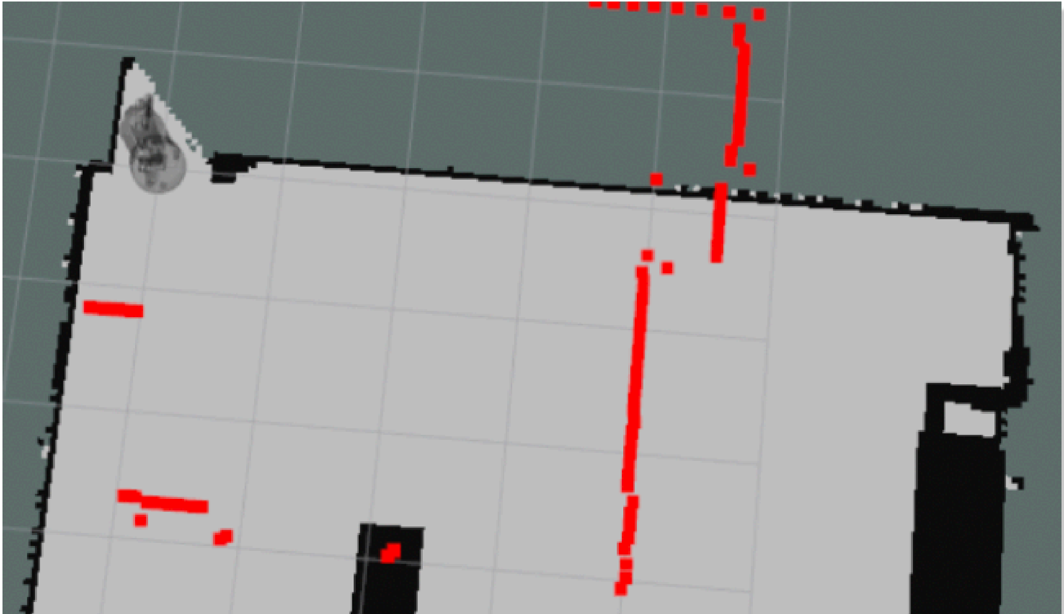
Hough Transform Algorithm, Revised

Data: Set S consisting of N points
Output: Line fitting the points in S
Initialize $n_\alpha \times n_r$ accumulator H with zeros
for $(x_i, y_i) \in S$ **do**
    for $\alpha \in \{\alpha_1, \ldots \alpha_{n_\alpha}\}$ **do**
        compute $r = x_i \cos\alpha + y_i \sin\alpha$;
        $H[\alpha, r] \leftarrow H[\alpha, r] + 1$;
    end for
end for
Choose $(\alpha*, r*)$ that corresponds to largest count in H;
Return line defined by $(\alpha*, r*)$

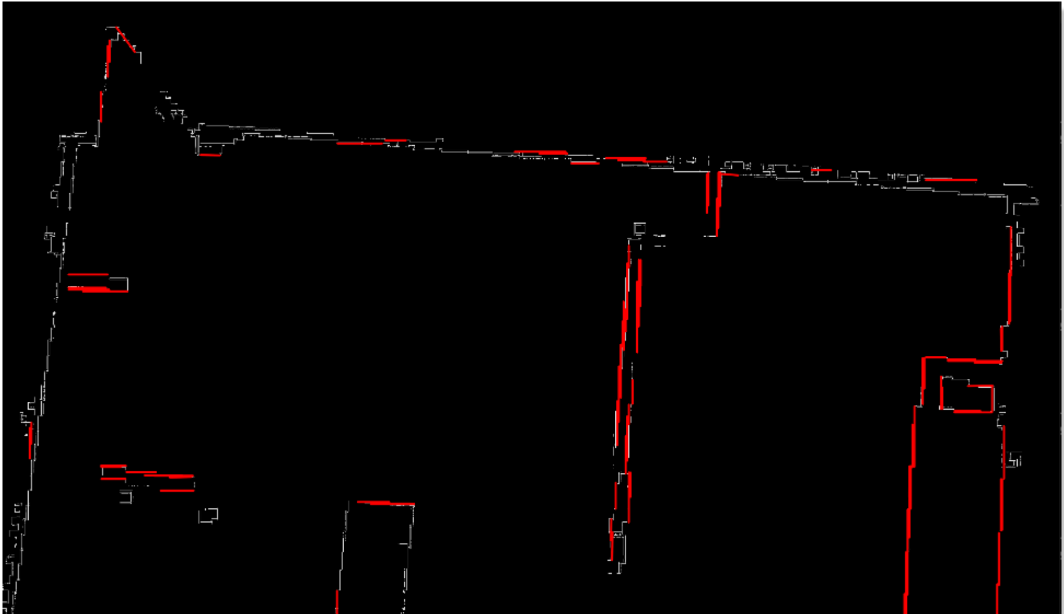
Hough transform: example



Hough transform: example



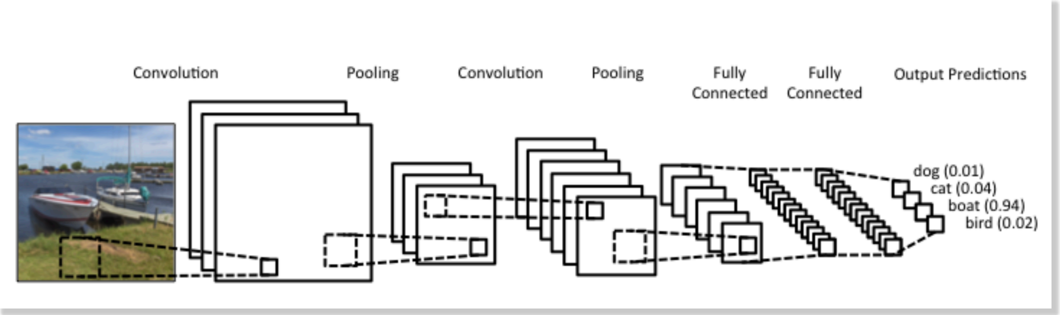
Hough transform: example



Object recognition

- Object recognition: capability of naming discrete objects in the world
- Why is it hard? Many reasons, including:
  - Real world is made of a jumble of objects, which all occlude one another and appear in different poses
  - There is a lot of variability intrinsic within each class (e.g., dogs)
- Here, we will look at three methods:
  - Template matching
  - Bag of visual words
  - Neural network methods

Geometric feature extraction



Acknowledgements

Acknowledgement

This slide deck is based on material from the Stanford ASL and ETH Zürich

References