AMME4710: COMPUTER VISION AND IMAGE PROCESSING WEEK 4

Dr. Mitch Bryson

School of Aerospace, Mechanical and Mechatronic Engineering, University of Sydney

Last Week

- Basic Image Processing and Applications:
  - Morphological operations
  - Image filtering
  - Edge detection

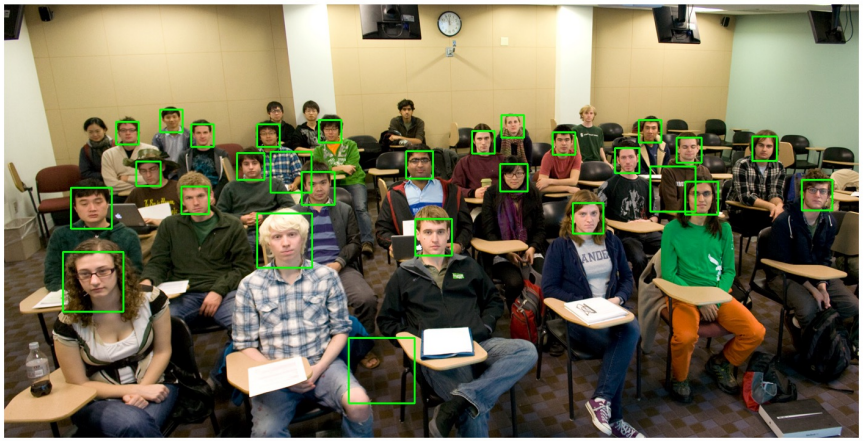
This Week's Lecture

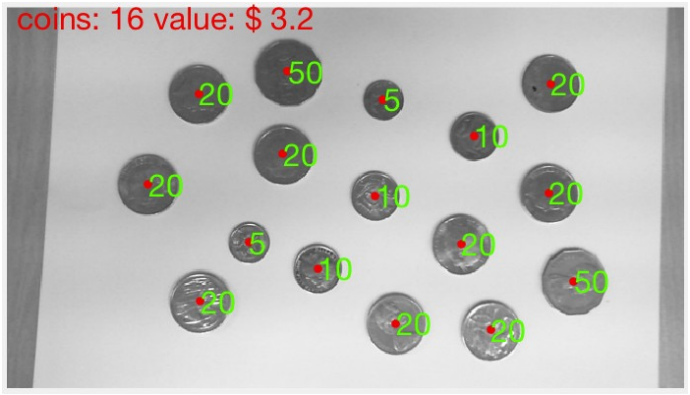
- Shapes, Features and Object Detection

- Learning Objectives:
  - To explore algorithms for basic object detection: the Hough transform and RANdom SAmple Concensus (RANSAC)
  - To introduce the concept of image features and explore methods for using features in the form of corner and interest points

Object Detection

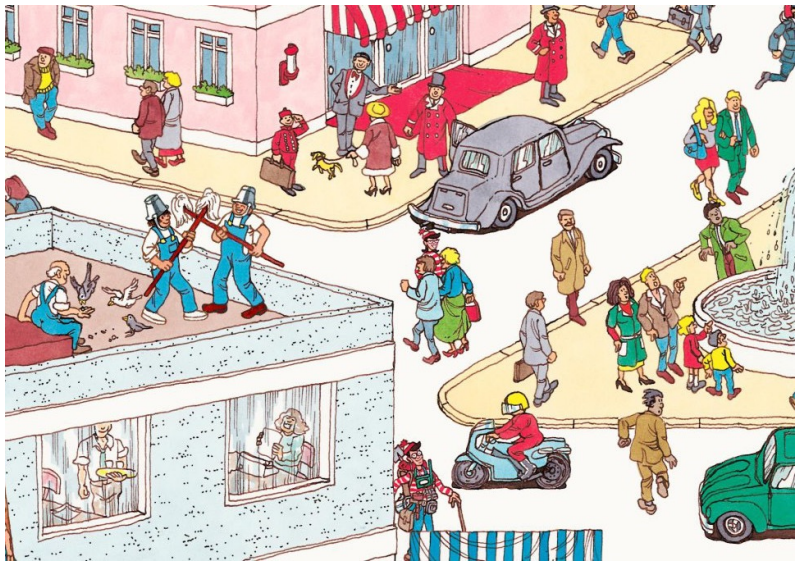
- Object detection is the process of finding instances real-world objects in images and localizing their position in image-coordinates
- Popular applications for object detection in computer vision include face detection, people detection, car/pedestrian detection





Template Matching

- One very simple form of object detection is **template matching**
- Problem is to find the location/s in a **source image** that best match with a **template image** that represents image data associated with an object we expect to find
- The template image is compared to every location in the source image (i.e. by non-linear filtering) to find the best match

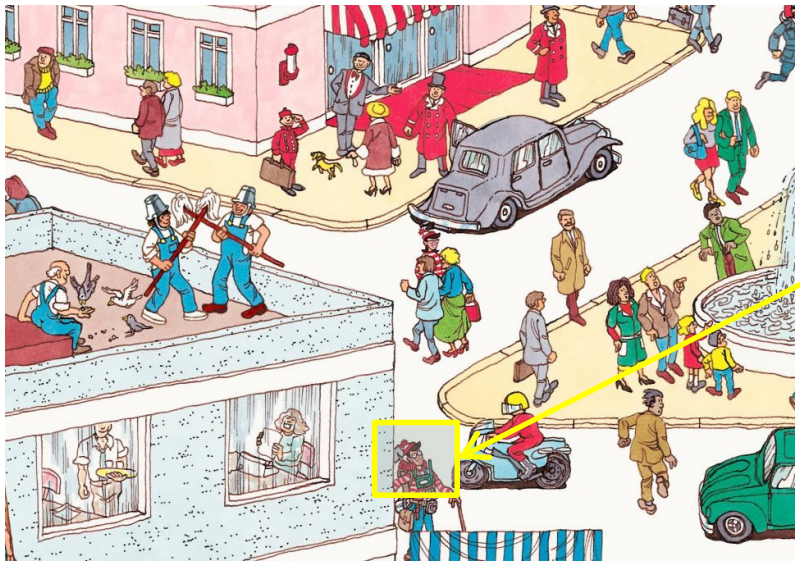


Source Image

Template Image

Template Matching

- One very simple form of object detection is **template matching**
- Problem is to find the location/s in a **source image** that best match with a **template image** that represents image data associated with an object we expect to find
- The template image is compared to every location in the source image (i.e. by non-linear filtering) to find the best match



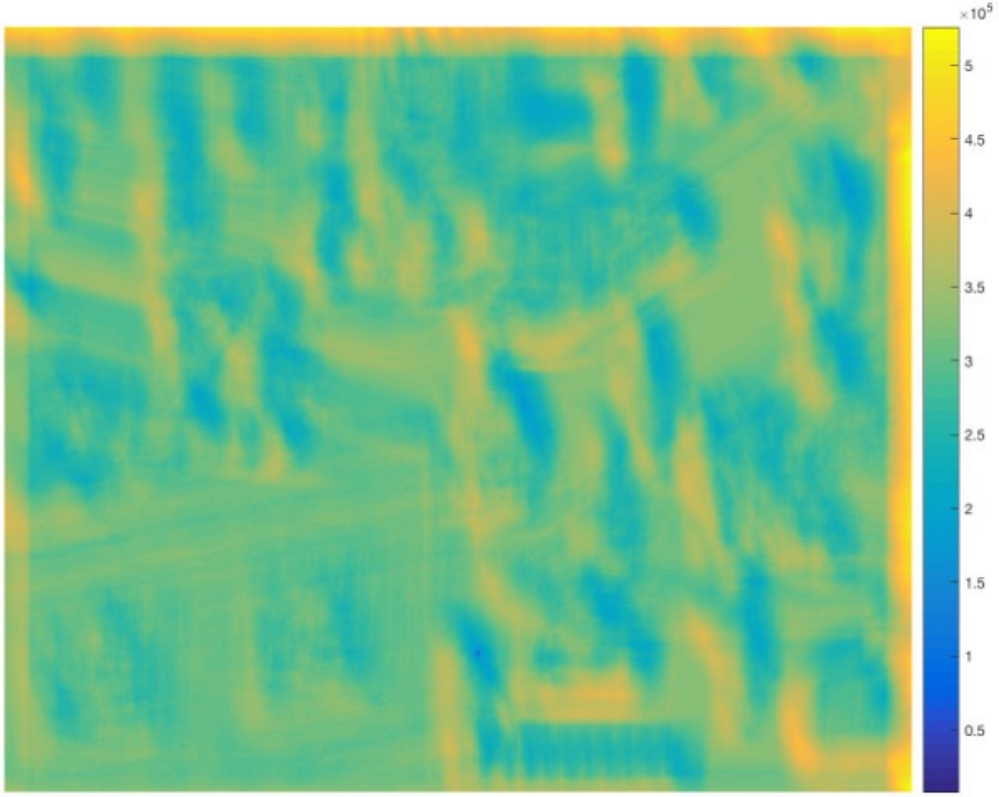
Source Image

Template Image

Template Matching

- Various cost functions R are used when comparing pixels in the source and template images (i.e. sum of absolute differences, sum of squared differences)
- Template matching is only effective if we have good image data that represents out object: we usually need to know it's scale and orientation (or else exhaustively search these extra dimensions)

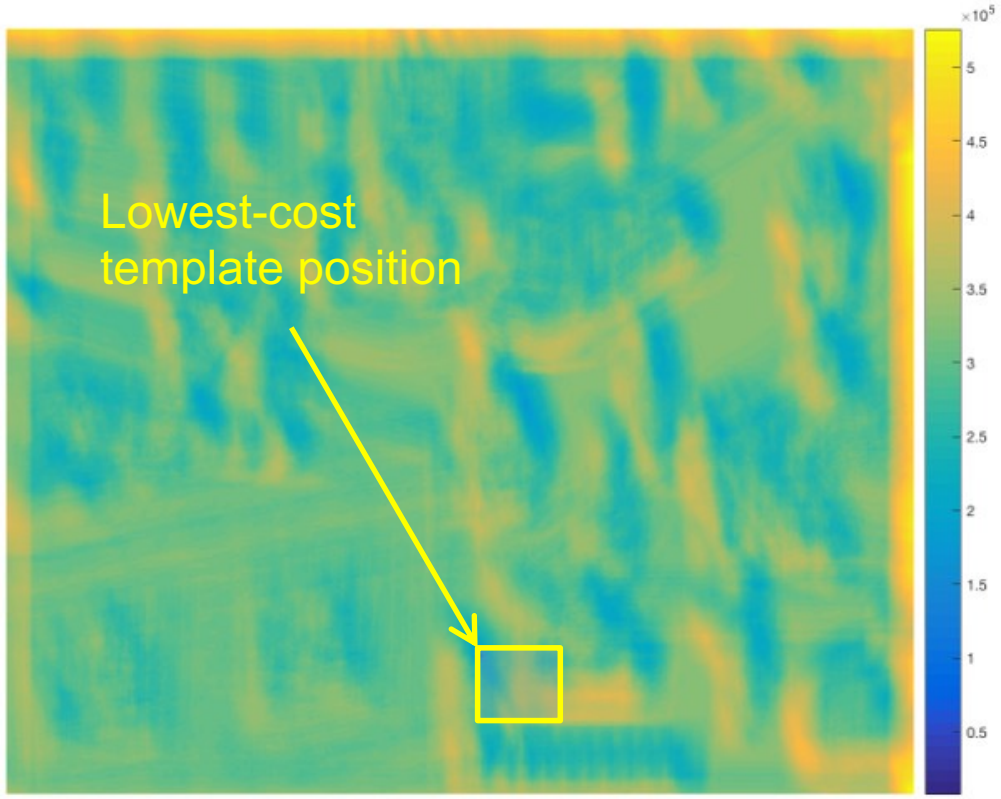
$$R(i,j) = \sum_k \sum_l |h(k,l) - f(i+k, j+l)|$$



Template Matching

- Various cost functions R are used when comparing pixels in the source and template images (i.e. sum of absolute differences, sum of squared differences)
- Template matching is only effective if we have good image data that represents our object: we usually need to know it's scale and orientation (or else exhaustively search these extra dimensions)

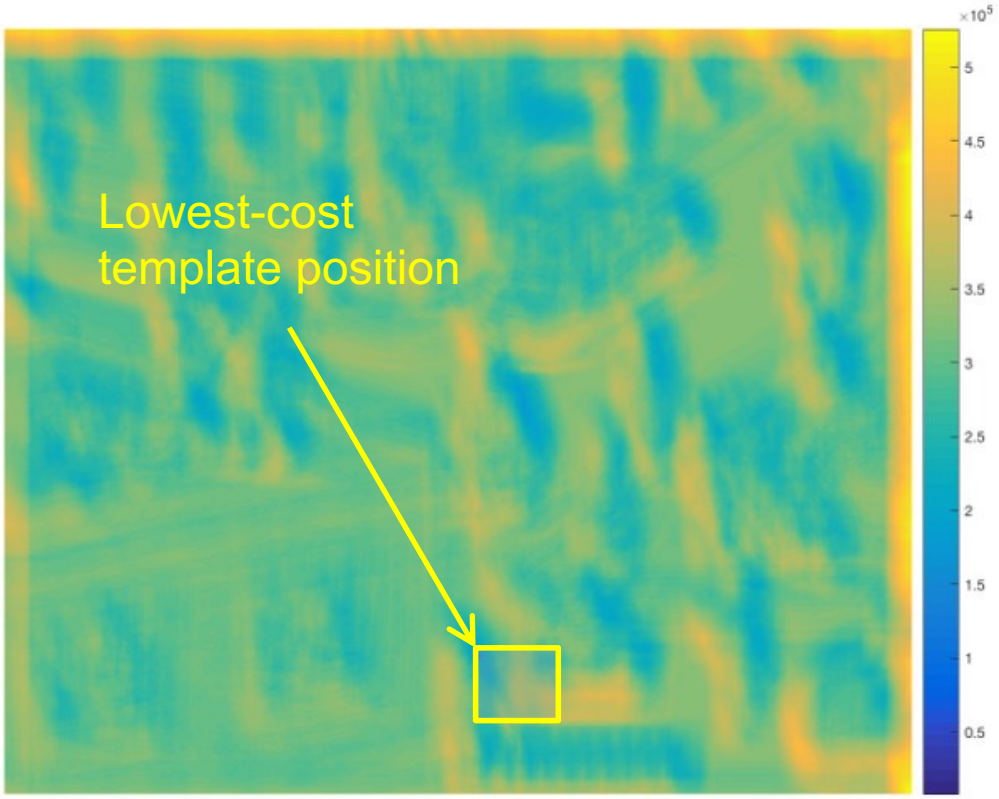
$$R(i,j) = \sum_k \sum_l |h(k,l) - f(i+k, j+l)|$$



Template Matching

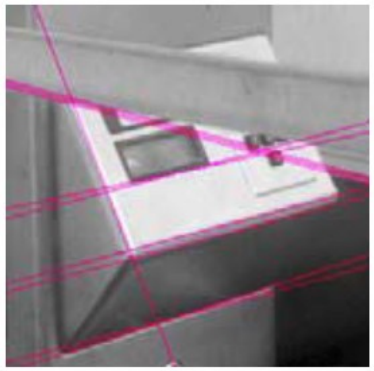
- Various cost functions R are used when comparing pixels in the source and template images (i.e. sum of absolute differences, sum of squared differences)
- Template matching is only effective if we have good image data that represents our object: we usually need to know it's scale and orientation (or else exhaustively search these extra dimensions)

$$R(i,j) = \sum_k \sum_l |h(k,l) - f(i+k, j+l)|$$



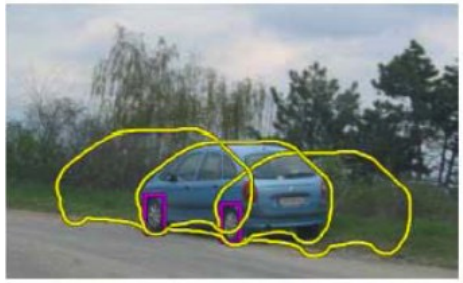
Searching for Objects using Models

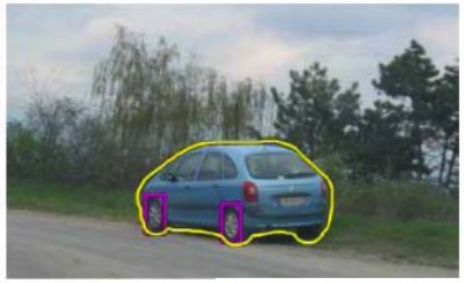
- A more robust approach stems from modelling an object's appearance based on elements that don't depend on the precise arrangement of pixels in a template:
  - Simple geometric components: lines, circles, ellipses etc.
  - Distinctive regions of intensity or colour
  - Textural properties
- Object detection methods must then be:
  - Robust to noise, slight changes in colour/lighting, clutter
  - Be able to deal with variations in the position, orientation and scale of the object in an image
  - Able to find objects based on these properties in a computationally tractable way



simple model: lines

simple model: circles





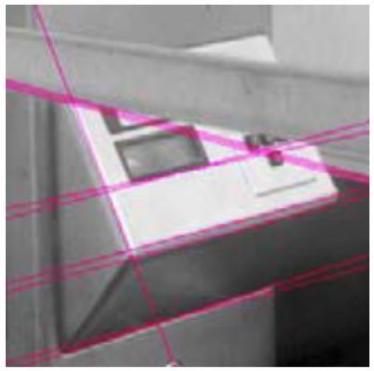
complicated model: car

Source: K. Grauman

Searching for Objects using Models

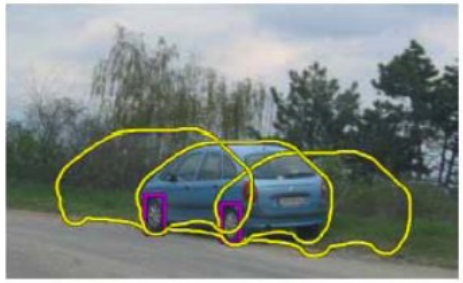
- A more robust approach stems from modelling an object's appearance based on elements that don't depend on the precise arrangement of pixels in a template:
  - Simple geometric components: lines, circles, ellipses etc.
  - Distinctive regions of intensity or colour
  - Textural properties

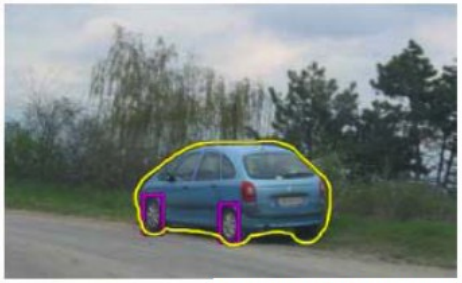
- Object detection methods must then be:
  - Robust to noise, slight changes in colour/lighting, clutter
  - Be able to deal with variations in the position, orientation and scale of the object in an image
  - Able to find objects based on these properties in a computationally tractable way



simple model: lines

simple model: circles



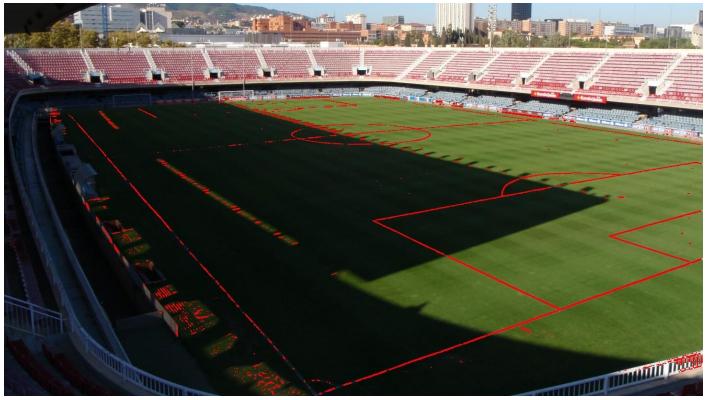


complicated model: car

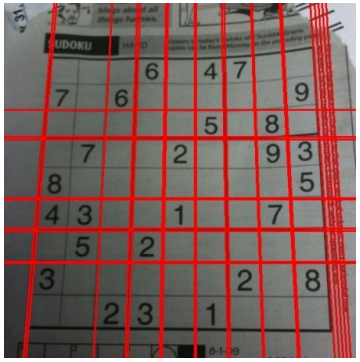
Source: K. Grauman

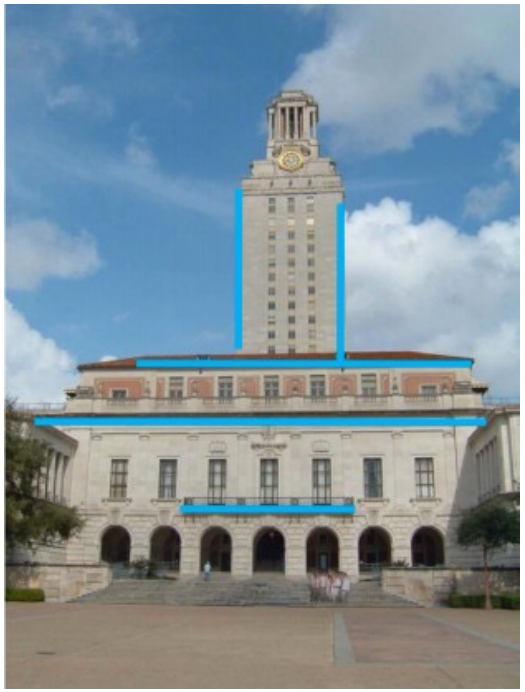
Finding objects based on geometry: lines

- One of the most simple geometric primitives that may assist in detecting objects is a **line**



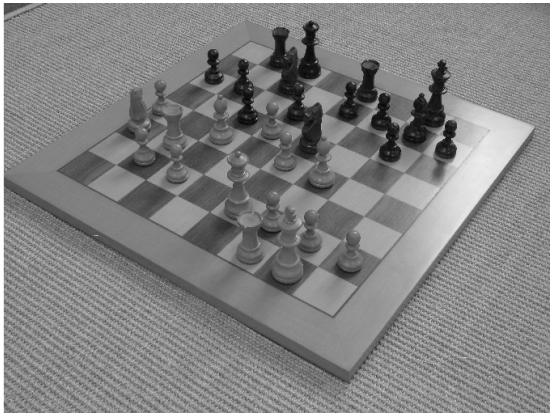
- A line differs from an edge: lines are specifically straight and are modelled as one contiguous object that exists over a region in the image that is typically much large than a few pixels



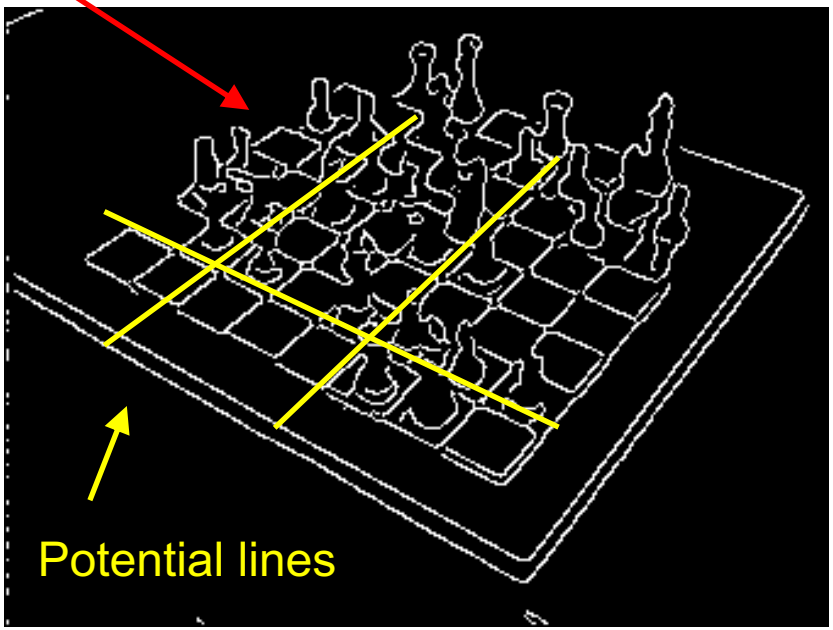


Difficulties with finding lines in images

- One way to find lines is to start with edges



Edge Detected Image

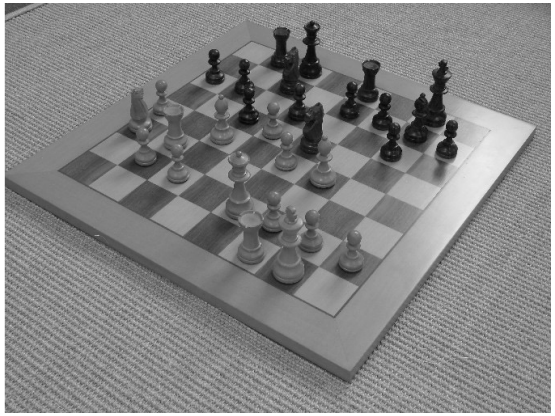


Difficulties with finding lines in images

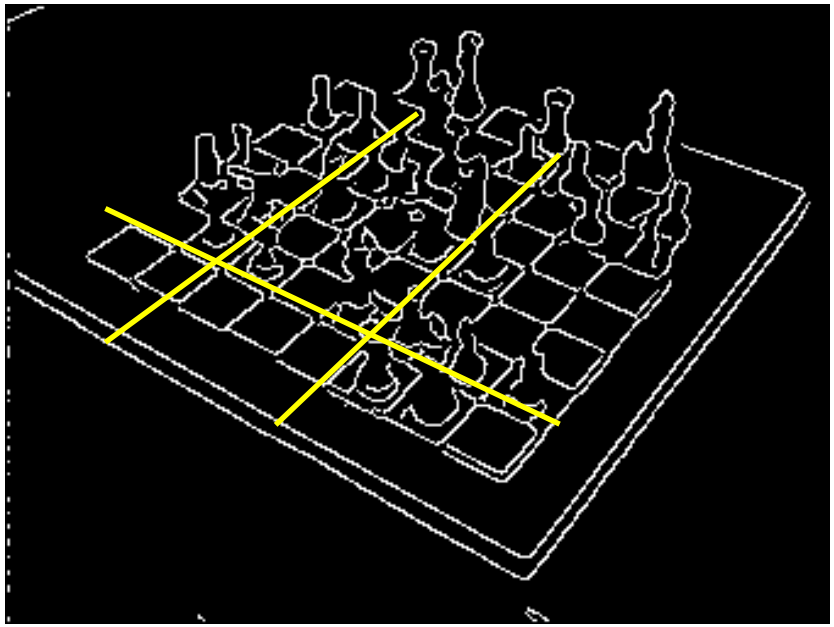
- One way to find lines is to start with edges
- We can define a line based on a set of parameters (m,b) such that:

$$y = mx + b$$

where m is the slope and b in the intercept



We could then "fit" lines to the edge data: good lines intercept with a lot of edges: but it's infeasible to manually check every possible set of parameters we could imagine

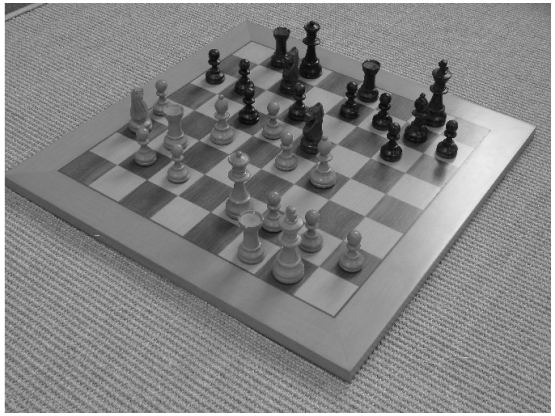


Difficulties with finding lines in images

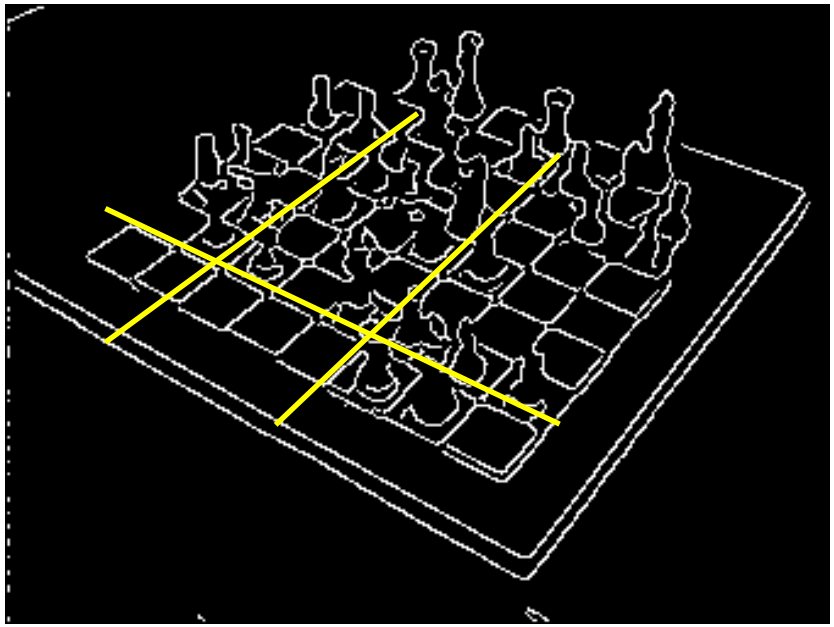
- One way to find lines is to start with edges
- We can define a line based on a set of parameters (m,b) such that:

$$y = mx + b$$

where m is the slope and b in the intercept



We could then "fit" lines to the edge data: good lines intercept with a lot of edges: but it's infeasible to manually check every possible set of parameters we could imagine

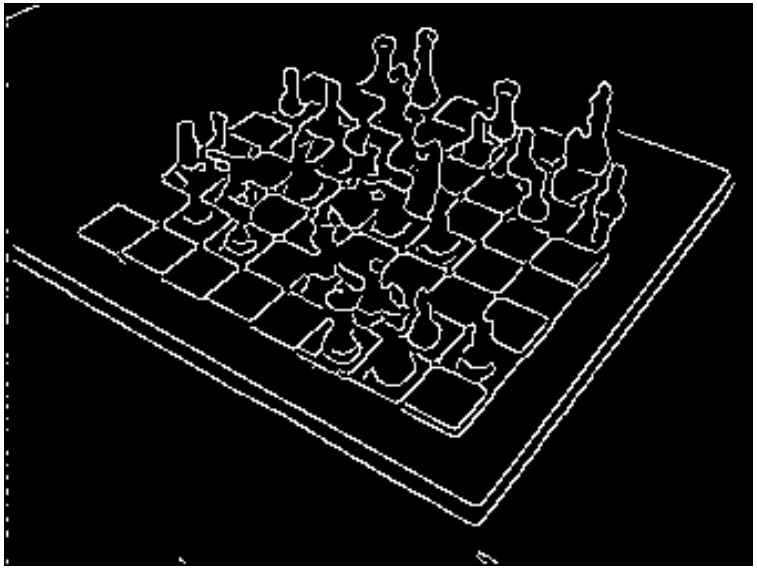


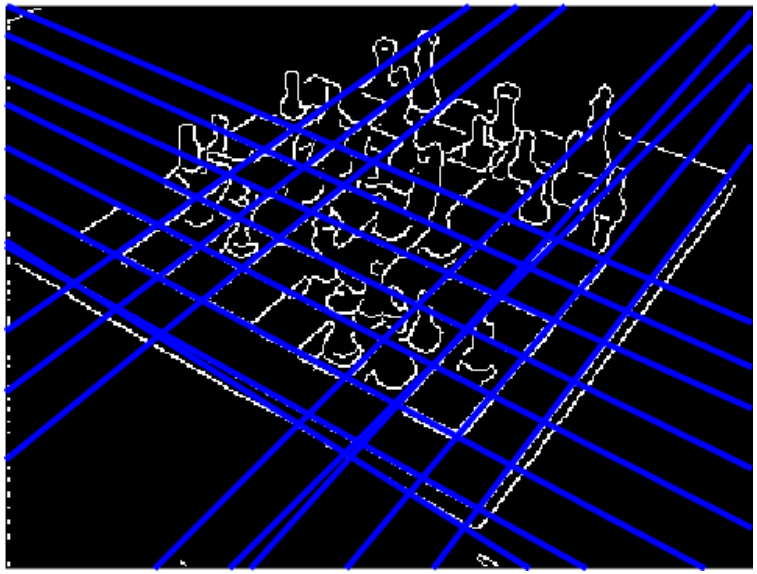
- Remaining difficulties:
  - **Clutter**: Many edge points don't correspond to lines, how do we sort out which do?
  - **Partial data**: the edges from one side of the board to the other form a line, but it is broken and occluded by the pieces

The Hough Transform

- The Hough Transform is a **voting technique** that can be used to address these issues

- The idea of the approach is:
  1) for each edge pixel, determine all the possible lines (in parametric space) that are compatible with this pixel and cast a “vote” for these parameter sets
  2) Find the sets of parameters which have received the most votes

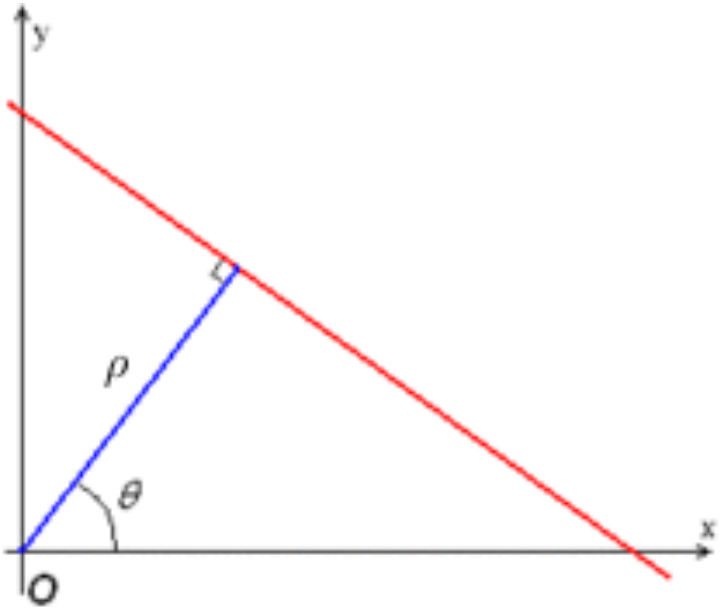




Hough Transform: Parametric Model for a line

- The slope/intercept model for a line is not great: parameters can take on arbitrary large/small values and vertical lines have infinite slope/no defined intercept
- The Hough transform uses the Hesse normal form to parameterise lines using two parameters ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$):

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

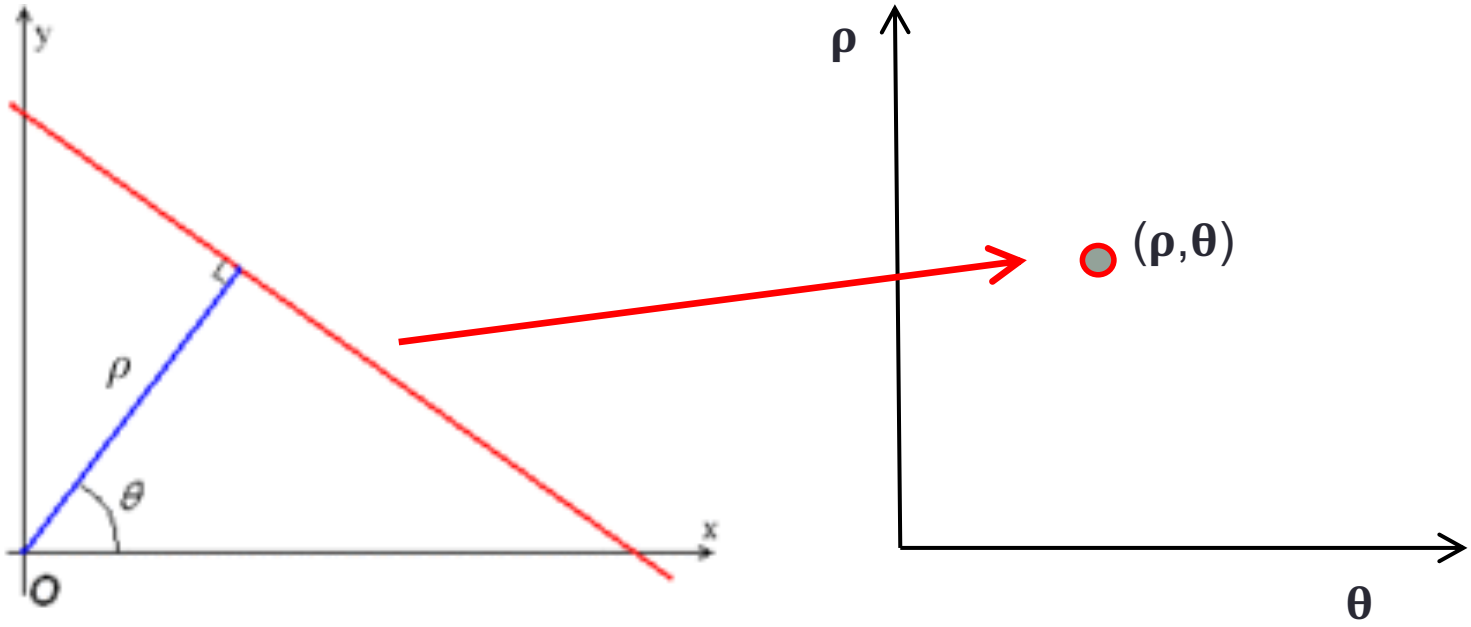


- $\boldsymbol{\theta}$: angle of vector perpendicular to the line (between +/- 90 degrees)
- $\boldsymbol{\rho}$: perpendicular distance to line from origin (positive when perpendicular point has positive x-coordinate, negative otherwise)

- Advantages of ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$): vertical/horizontal lines completely defined, finite limits on $\boldsymbol{\theta}$, and finite limits (in practice) on $\boldsymbol{\rho}$, as we don't consider lines outside of the image boundaries

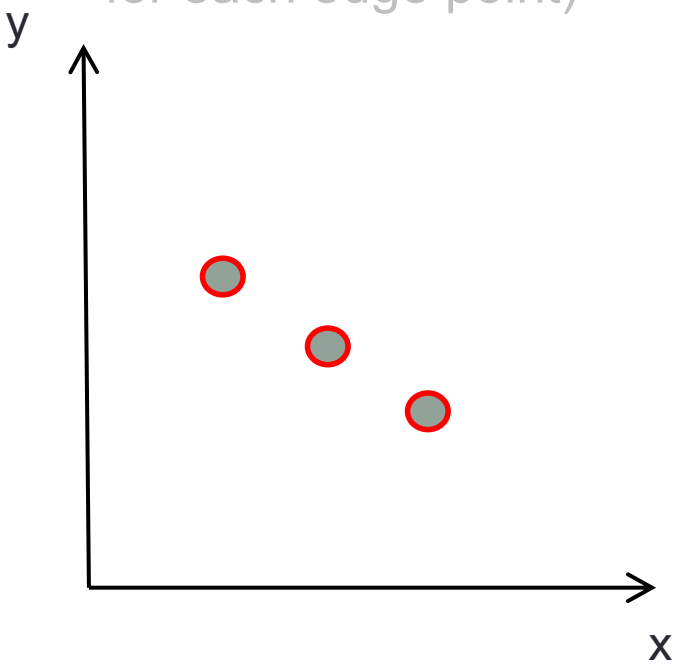
Hough Transform: Parametric Model for a line

- Each line in (x,y) space corresponds to a single point in (**ρ**,**θ**) space:



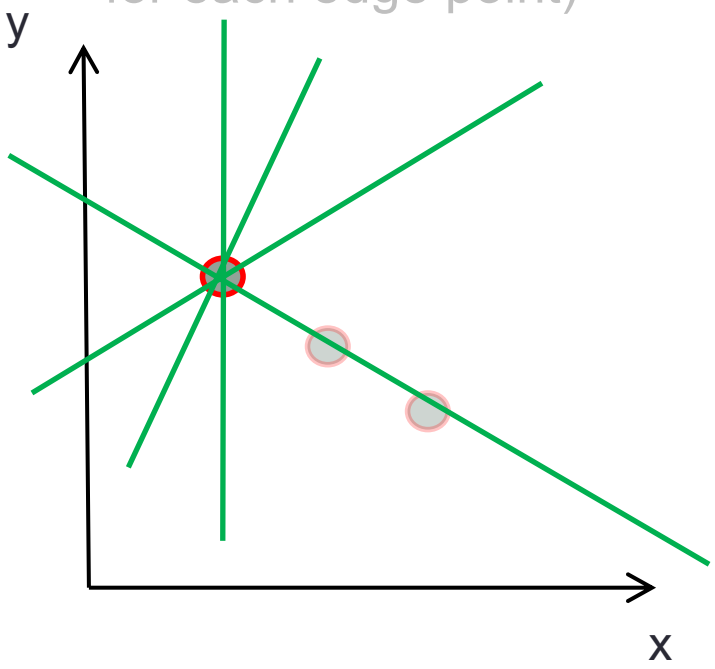
Hough Transform: Voting

- To find lines in an edge image, each edge pixel votes for all points in (ρ,θ) space that are compatible with it
- Consider three edge points (pixels) in image space (x,y):
- For each point, there are a range of different compatible lines (i.e. all lines that pass through the point):
- We will define a discrete set of line orientations and calculate a value of ρ for each θ to form (ρ,θ) pairs (in this case six different models for each edge point)



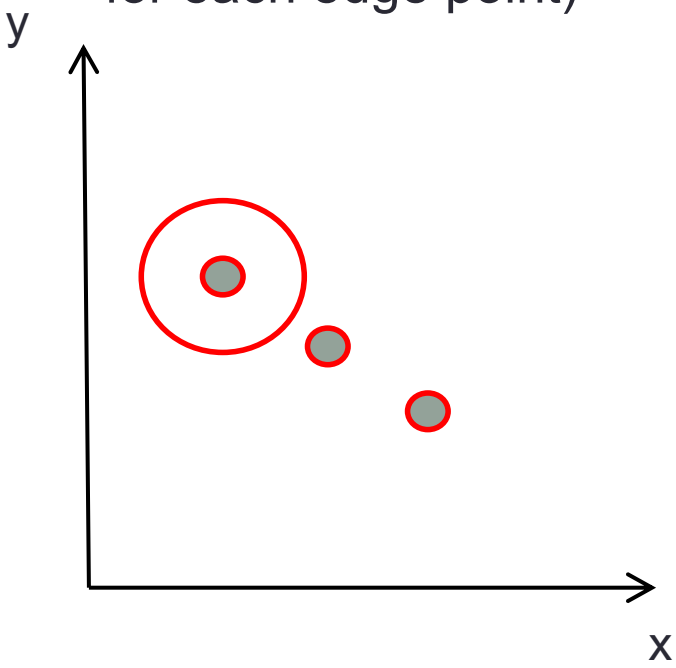
Hough Transform: Voting

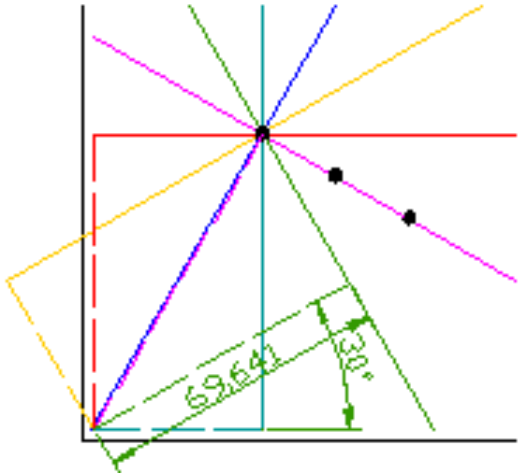
- To find lines in an edge image, each edge pixel votes for all points in (ρ,θ) space that are compatible with it
- Consider three edge points (pixels) in image space (x,y):
- For each point, there are a range of different compatible lines (i.e. all lines that pass through the point):
- We will define a discrete set of line orientations and calculate a value of ρ for each θ to form (ρ,θ) pairs (in this case six different models for each edge point)



Hough Transform: Voting

- To find lines in an edge image, each edge pixel votes for all points in (ρ,θ) space that are compatible with it
- Consider three edge points (pixels) in image space (x,y):
- For each point, there are a range of different compatible lines (i.e. all lines that pass through the point):
- We will define a discrete set of line orientations and calculate a value of ρ for each θ to form (ρ,θ) pairs (in this case six different models for each edge point)

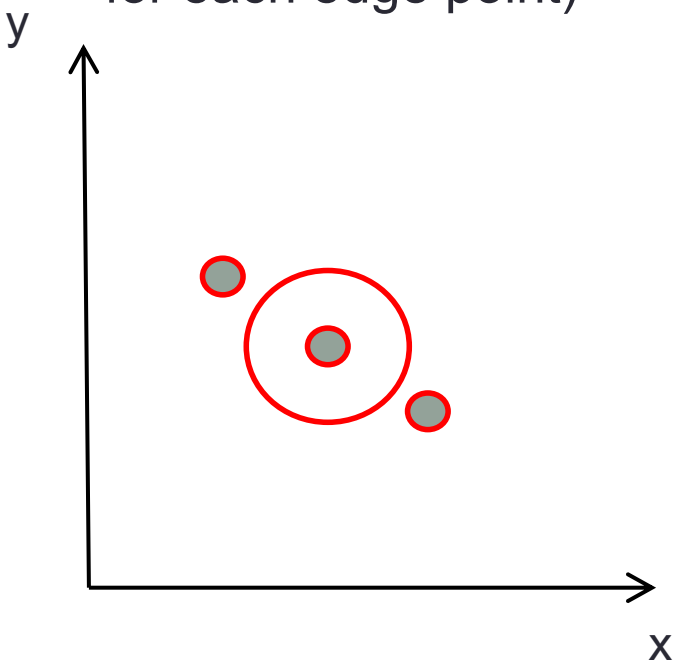


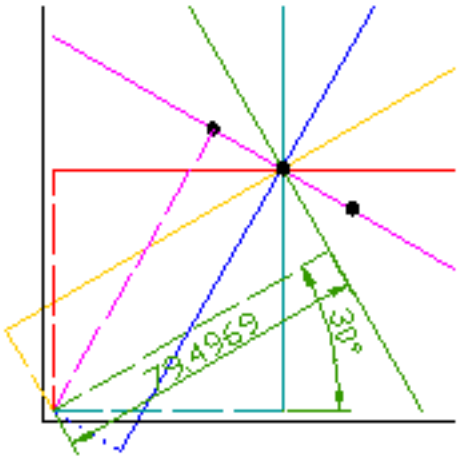


θ	ρ
Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

Hough Transform: Voting

- To find lines in an edge image, each edge pixel votes for all points in (ρ,θ) space that are compatible with it
- Consider three edge points (pixels) in image space (x,y):
- For each point, there are a range of different compatible lines (i.e. all lines that pass through the point):
- We will define a discrete set of line orientations and calculate a value of ρ for each θ to form (ρ,θ) pairs (in this case six different models for each edge point)

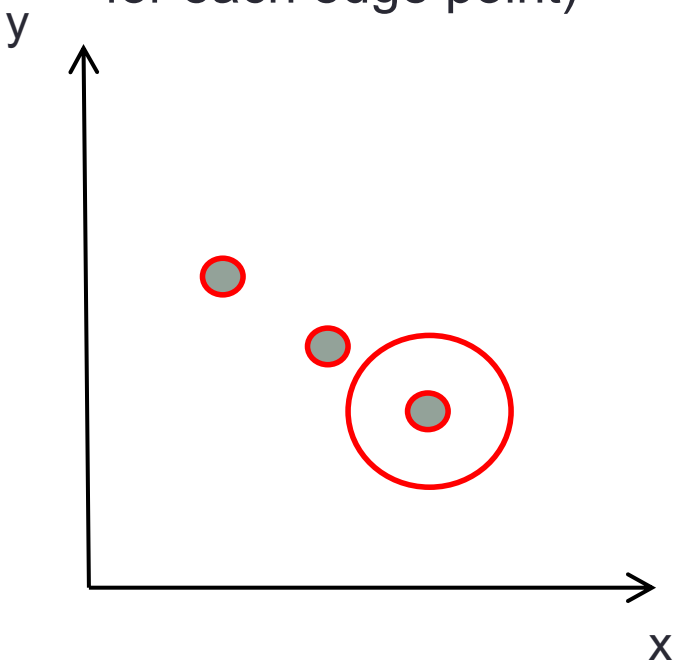


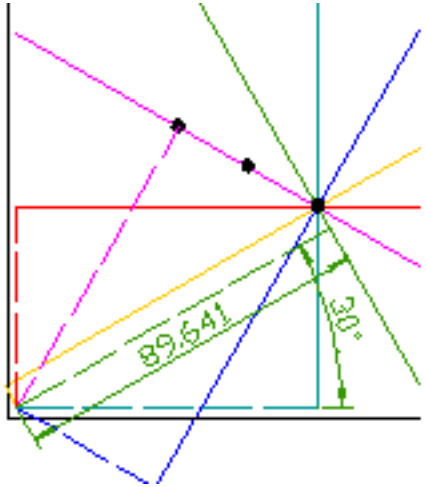


θ	ρ
Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	-19.5

Hough Transform: Voting

- To find lines in an edge image, each edge pixel votes for all points in (ρ,θ) space that are compatible with it
- Consider three edge points (pixels) in image space (x,y):
- For each point, there are a range of different compatible lines (i.e. all lines that pass through the point):
- We will define a discrete set of line orientations and calculate a value of ρ for each θ to form (ρ,θ) pairs (in this case six different models for each edge point)

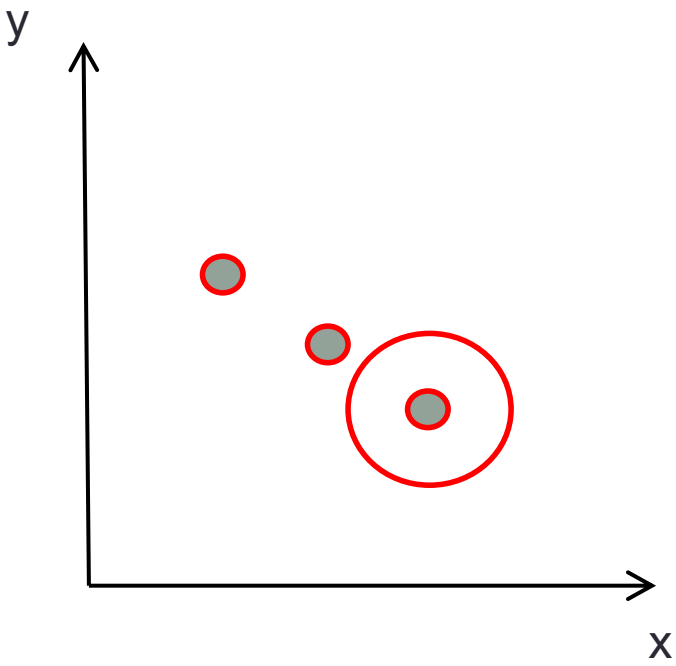


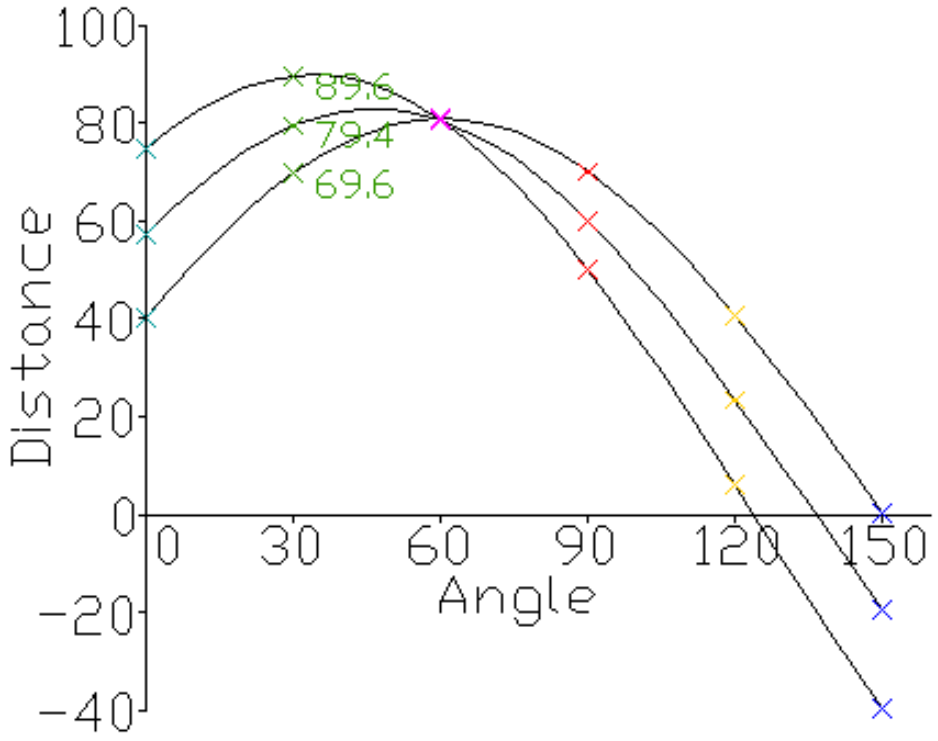


θ	ρ
Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	-39.6

Hough Transform: Voting

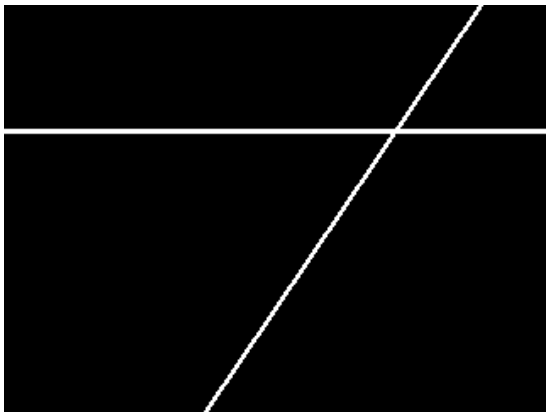
- We can now plot our votes in (ρ,θ) space:
  - Each edge point creates a sine wave in (ρ,θ) space
  - Notice that for all three edge points in this case, the same (ρ,θ) combination at (80, 60°) has been voted for, and our sine waves converge on this point



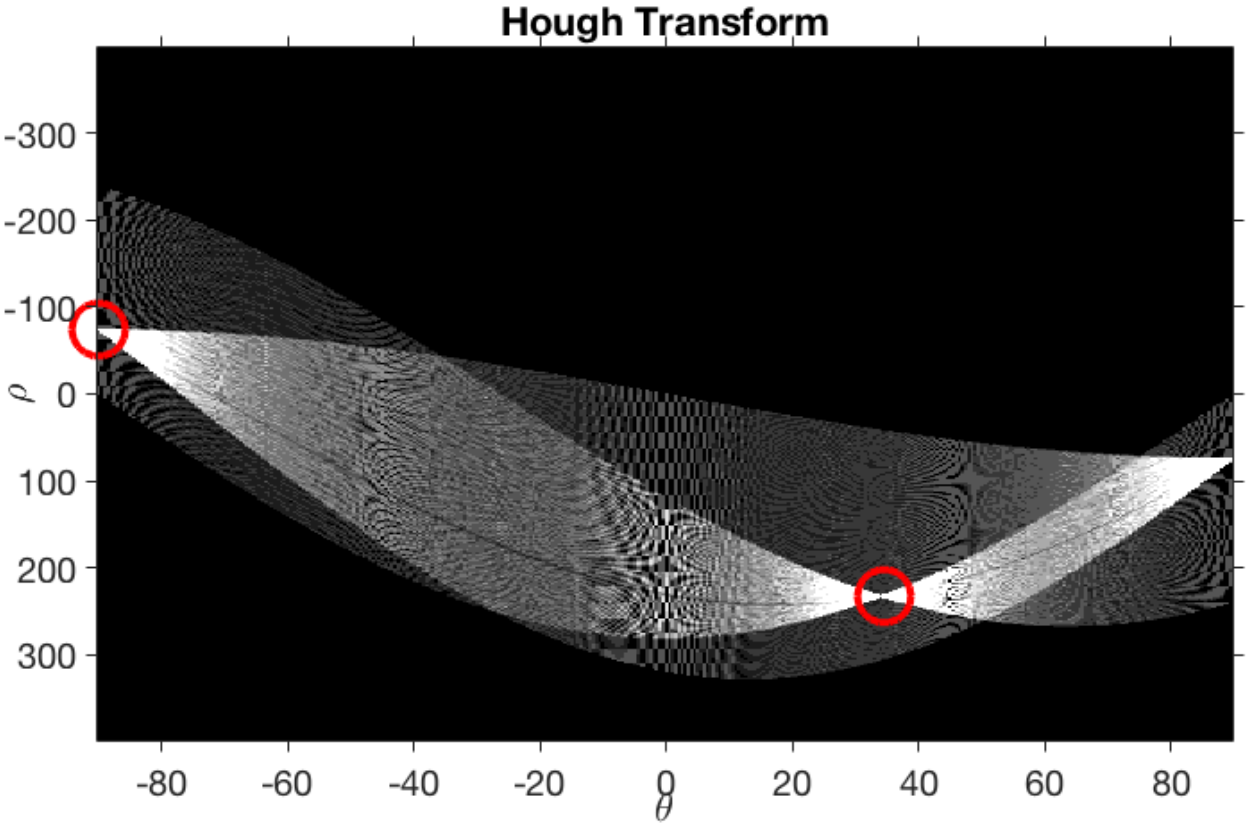


Hough Transform: Voting

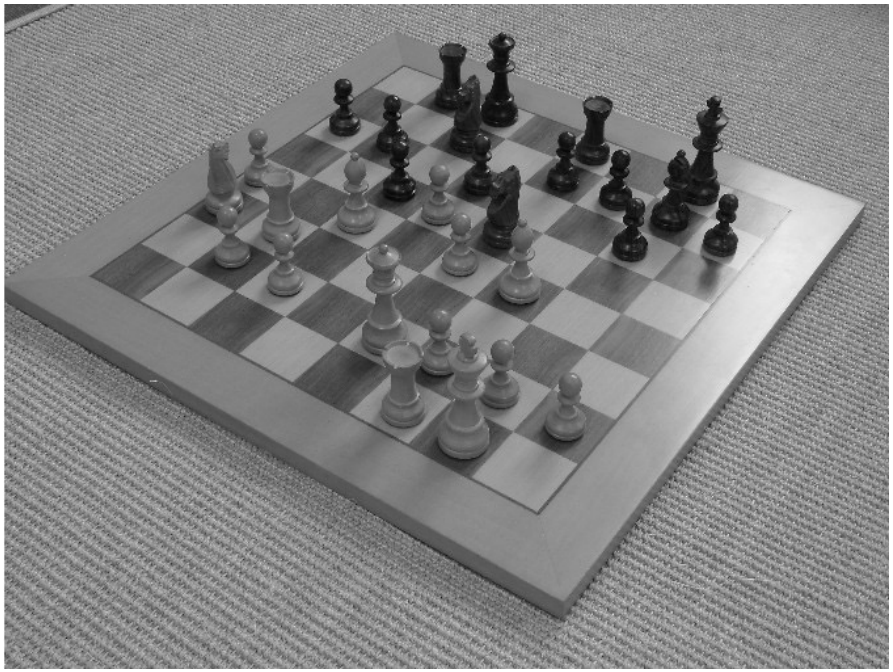
- Consider a real image with a large number of edge points/pixels: in practice we compute a 2D histogram of (ρ,θ) combinations with discrete bins for (ρ,θ)
- If we count the values in each (ρ,θ) bin, peaks will correspond to strong evidence for the presence of a line: we grab all (ρ,θ) values over a particular threshold and these are our lines



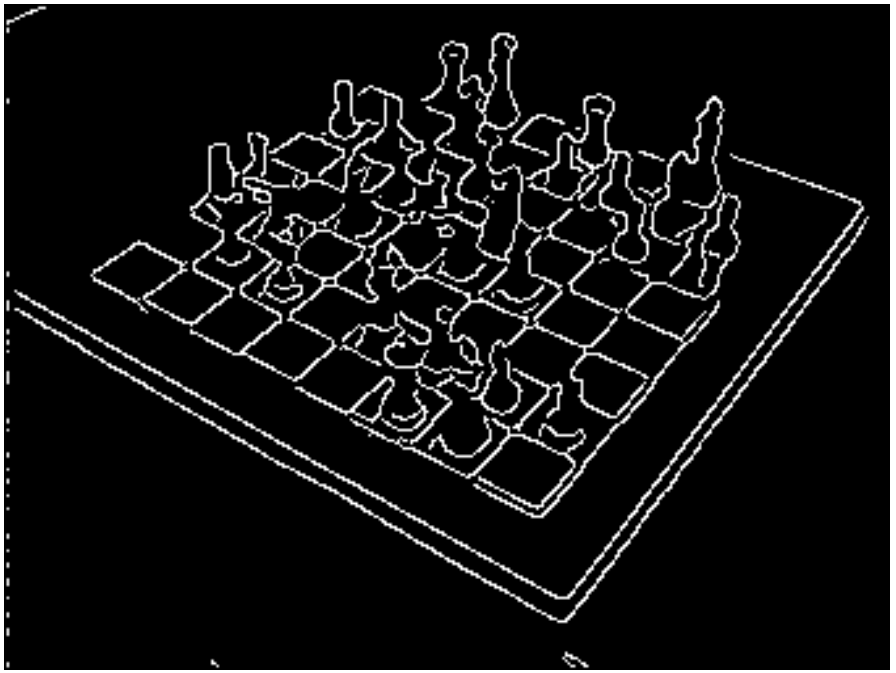
Simple Edge Image



Hough Transform: Example

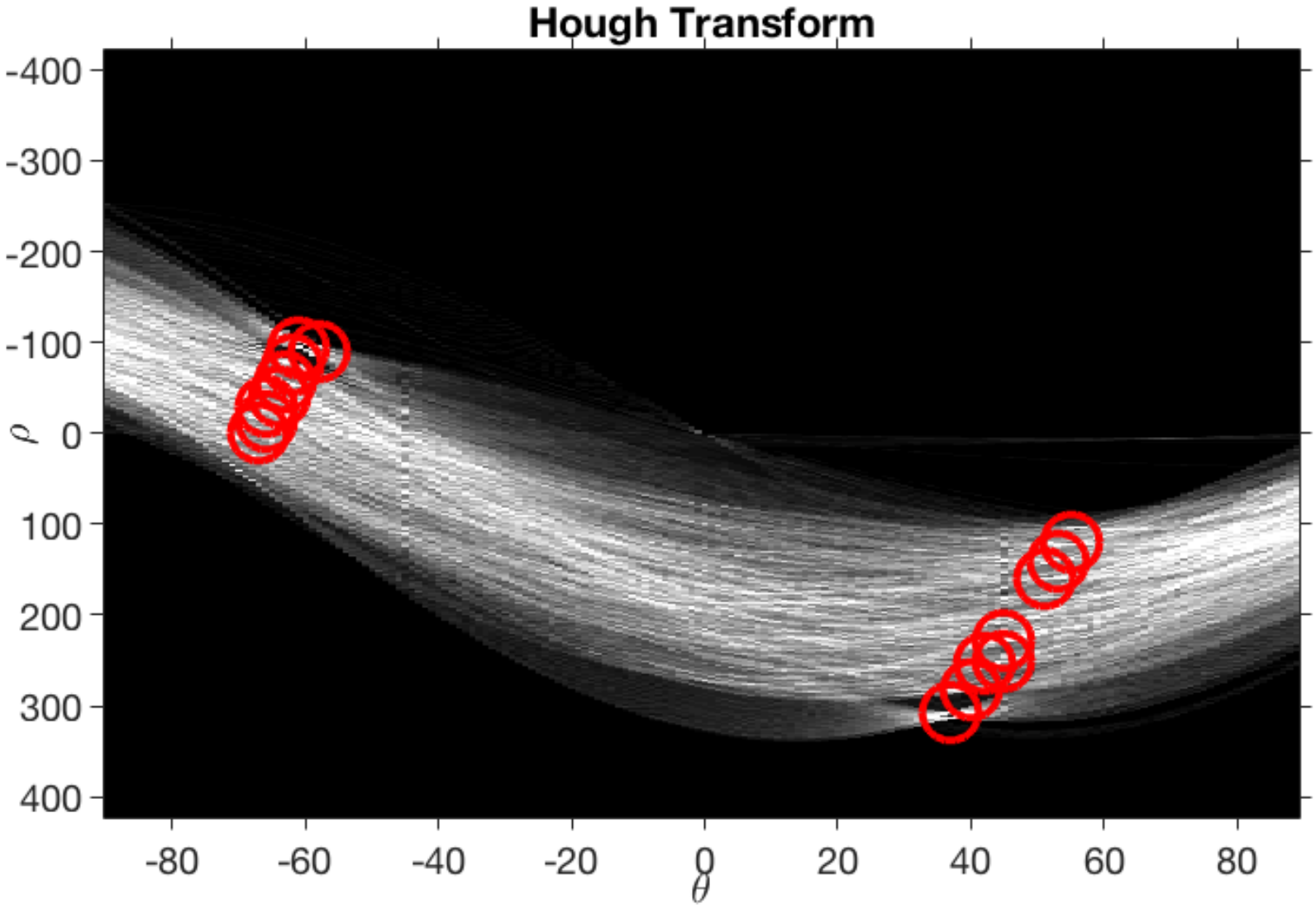


Original Image

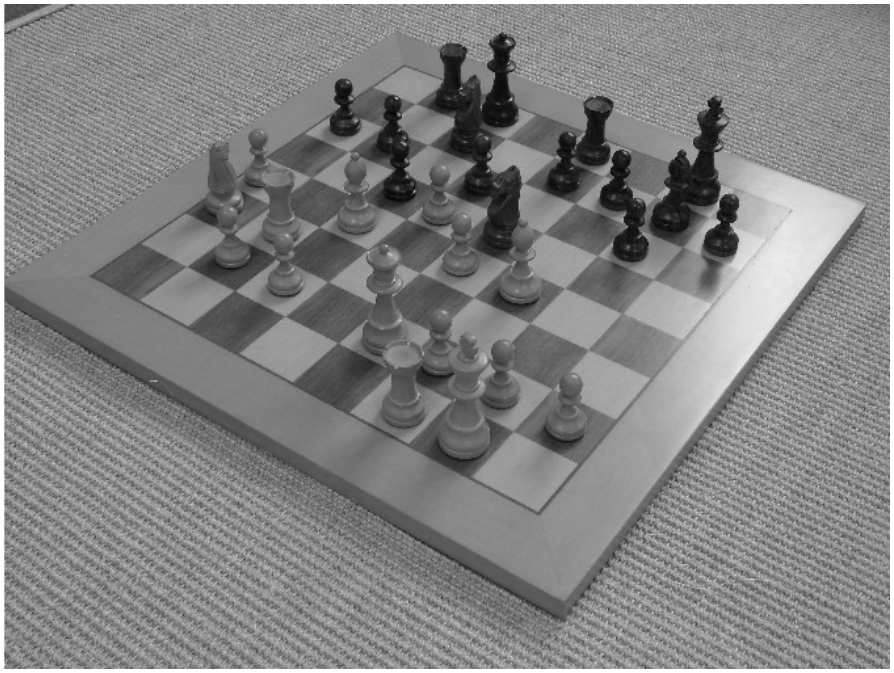


Edge Image Calculated using
Canny Edge Detection

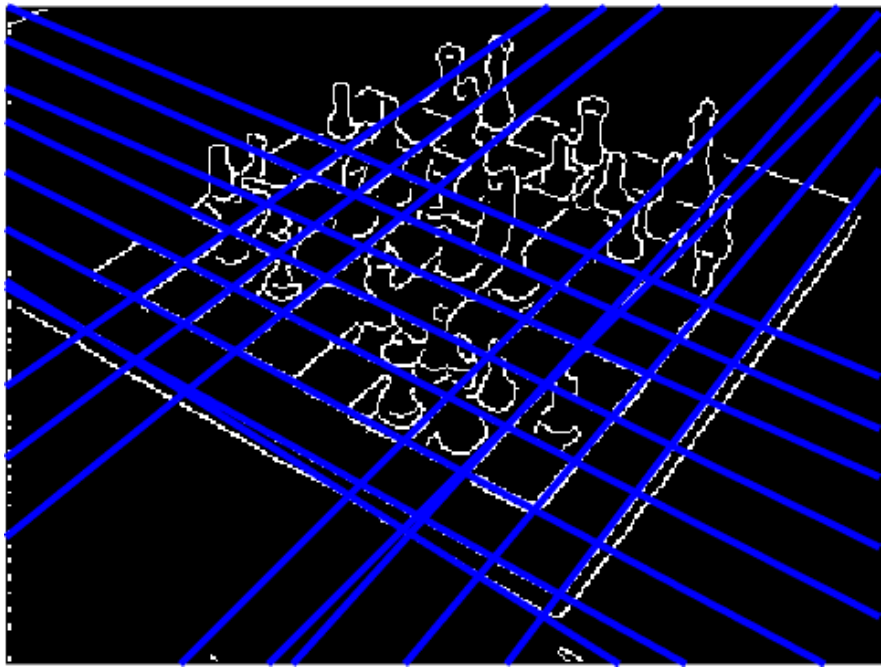
Hough Transform: Example



Hough Transform: Example



Original Image



Lines corresponding to chess board

- Once lines have been computed as ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$) points, x-y coordinates can be recovered using:

$$y = \frac{\rho - x\cos\theta}{\sin\theta} \qquad \text{or} \qquad x = \frac{\rho - y\sin\theta}{\cos\theta}$$

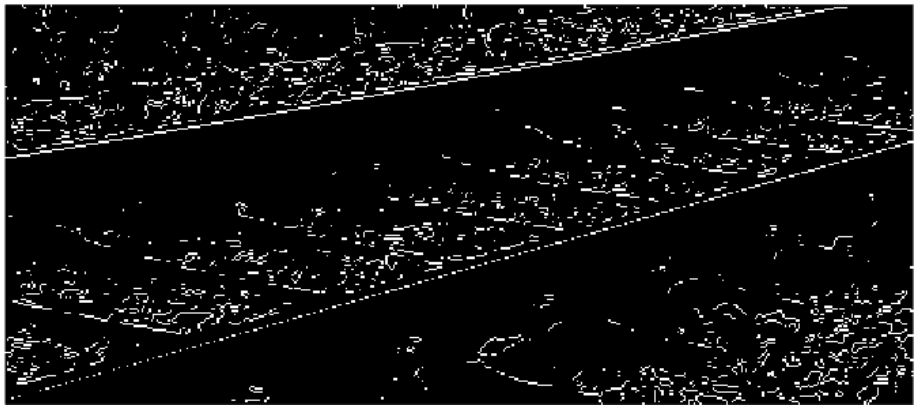
- The start and end points of lines can be computed by running along (x,y) coordinates and tracking edge pixels until an appropriately-sized gap appears

Hough transform guided by edge direction

- Standard Hough transform: each edge pixel casts several votes for θ (line angle)
  - End up with a lot of "clutter" in the parameter matrix
  - For most edge finding techniques, we know the direction of the edge based on the gradient information at each image location
  - We can use the edge direction to guide the values of θ for each pixel's vote by restricting the range of θ to be some small deviation about edge/gradient direction

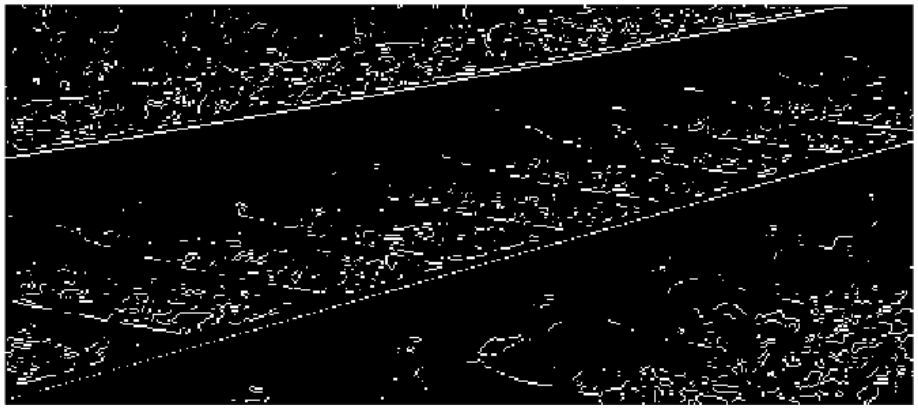
Hough transform guided by edge direction

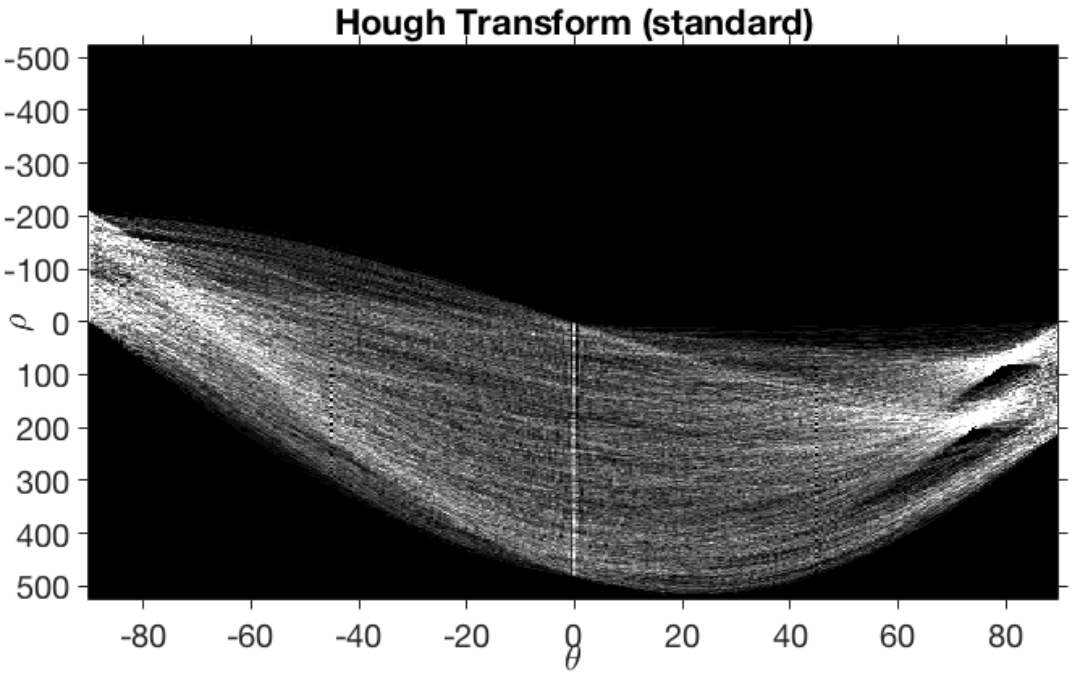
- Standard Hough transform: each edge pixel casts several votes for θ (line angle)
  - End up with a lot of "clutter" in the parameter matrix
  - For most edge finding techniques, we know the direction of the edge based on the gradient information at each image location
  - We can use the edge direction to guide the values of θ for each pixel's vote by restricting the range of θ to be some small deviation about edge/gradient direction



Hough transform guided by edge direction

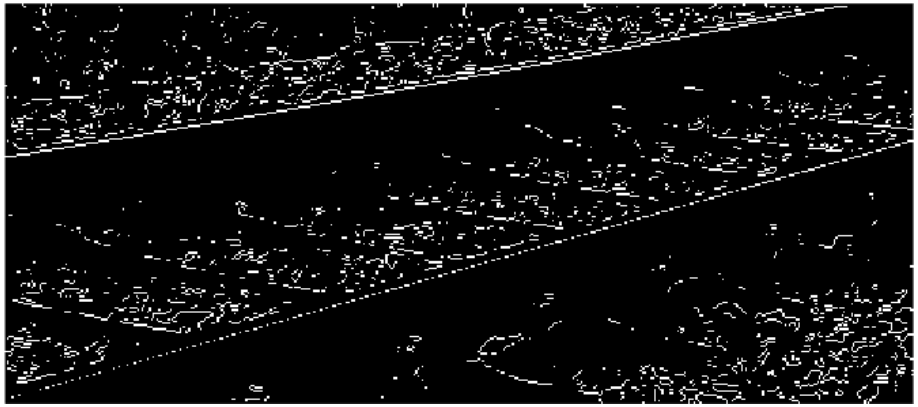
- Standard Hough transform: each edge pixel casts several votes for θ (line angle)
  - End up with a lot of "clutter" in the parameter matrix
  - For most edge finding techniques, we know the direction of the edge based on the gradient information at each image location
  - We can use the edge direction to guide the values of θ for each pixel's vote by restricting the range of θ to be some small deviation about edge/gradient direction

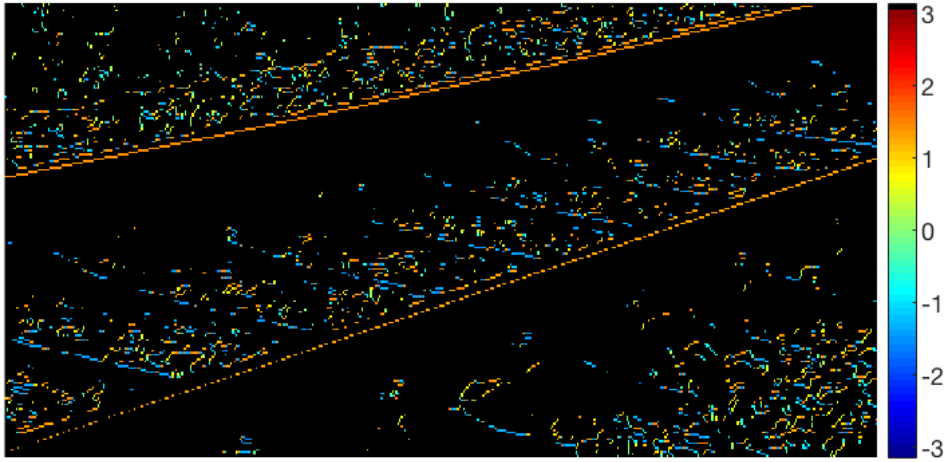




Hough transform guided by edge direction

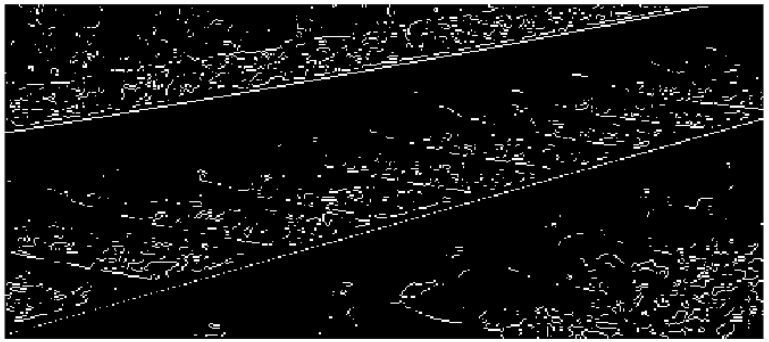
- Standard Hough transform: each edge pixel casts several votes for θ (line angle)
  - End up with a lot of "clutter" in the parameter matrix
  - For most edge finding techniques, we know the direction of the edge based on the gradient information at each image location
  - We can use the edge direction to guide the values of θ for each pixel's vote by restricting the range of θ to be some small deviation about edge/gradient direction

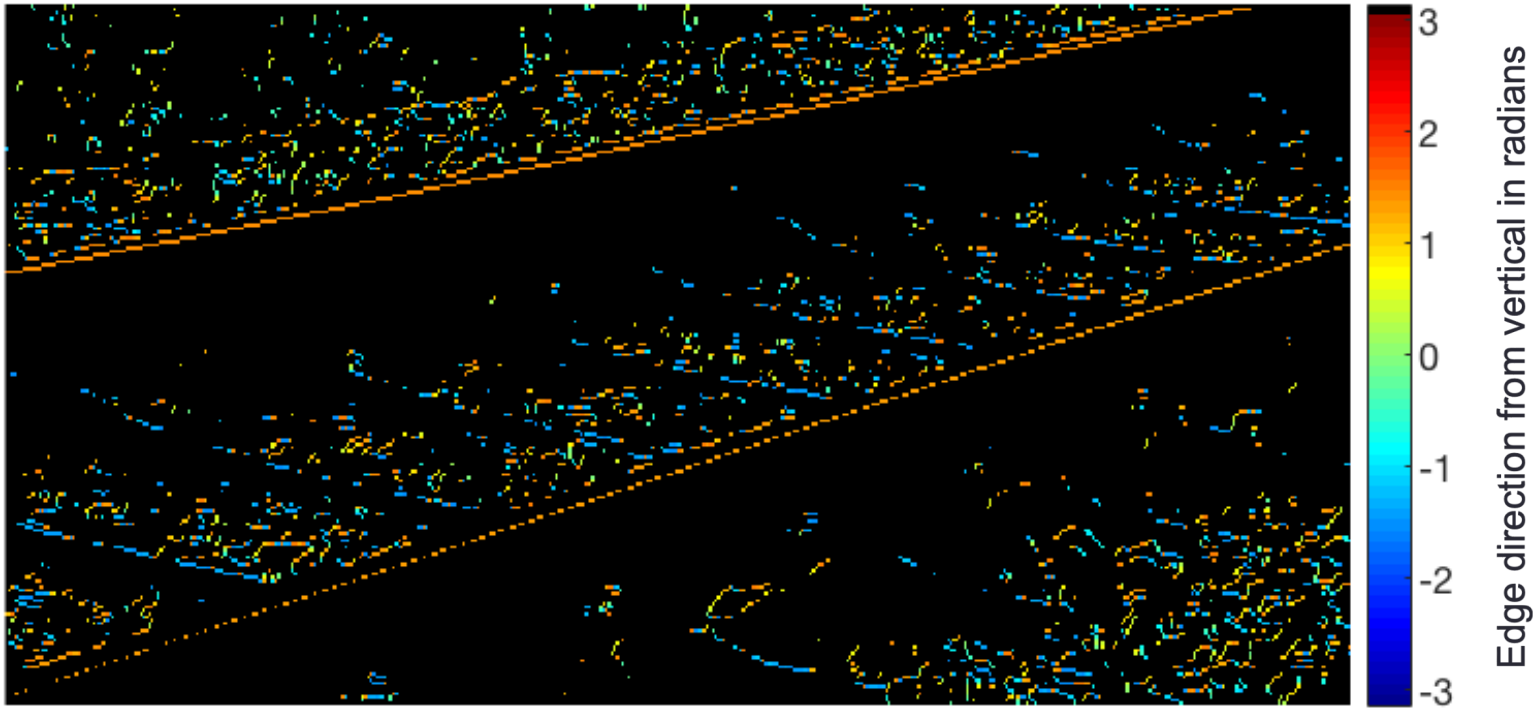




Edge Direction

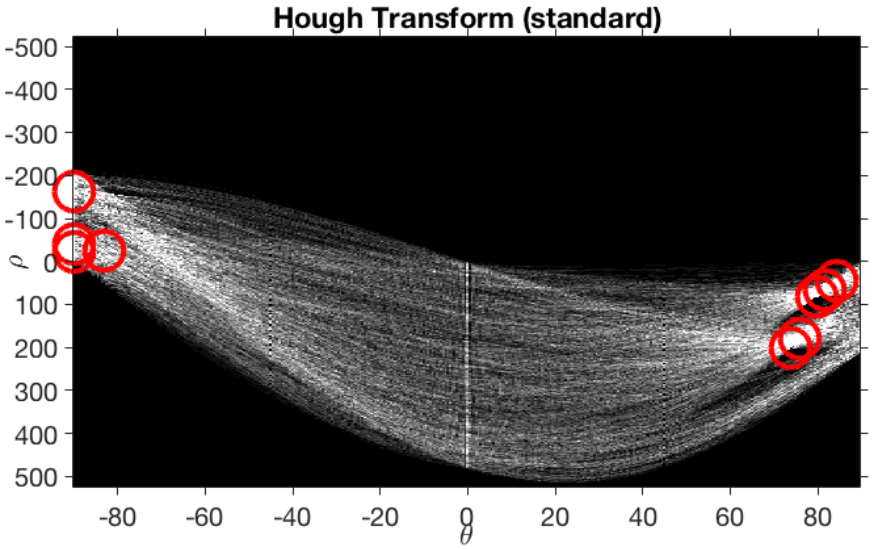
Hough transform guided by edge direction

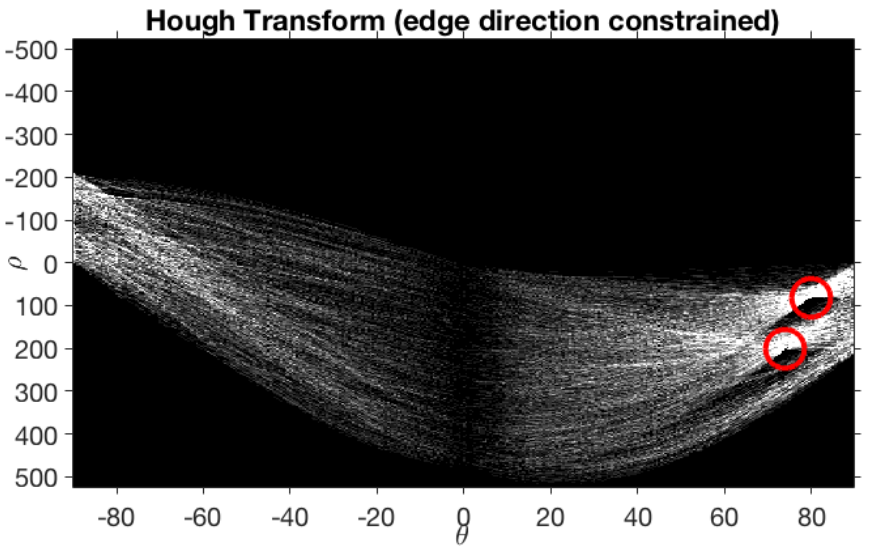


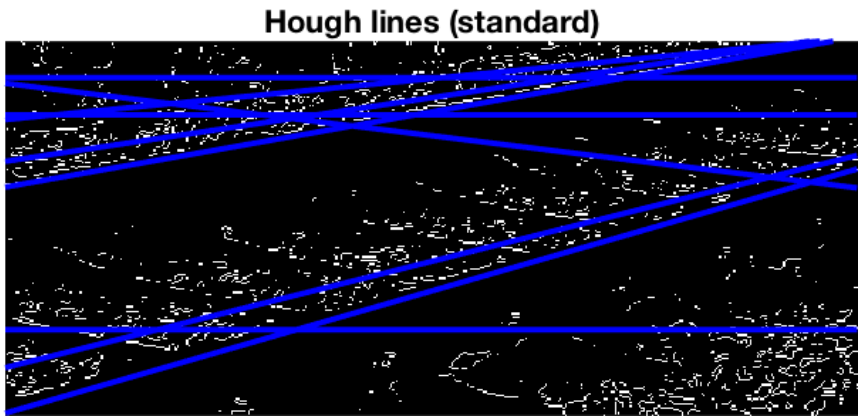


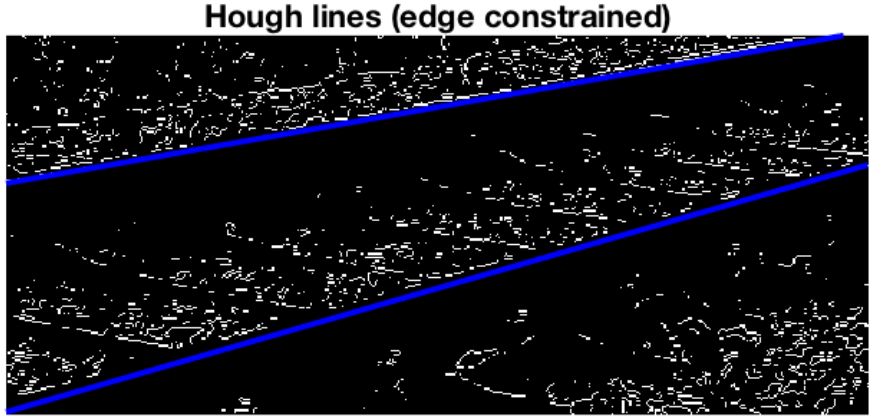
Sobel edges coloured based direction

Hough transform guided by edge direction





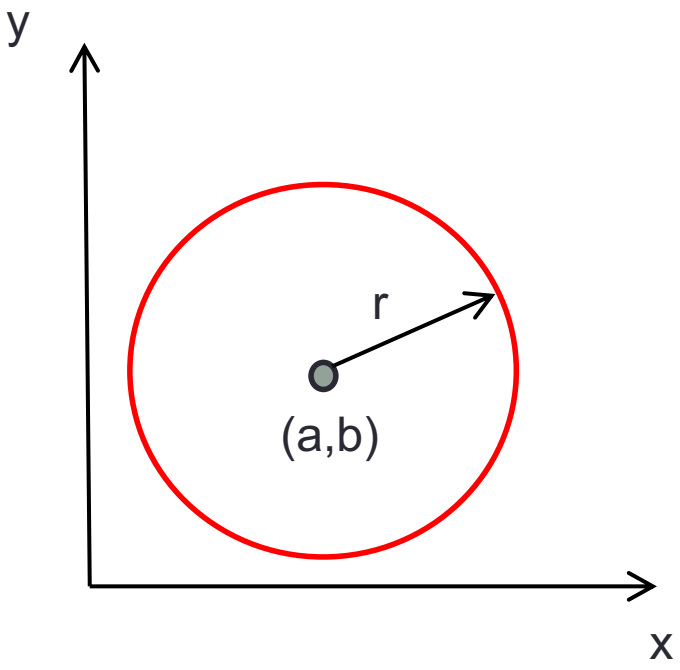




The Circular Hough Transform

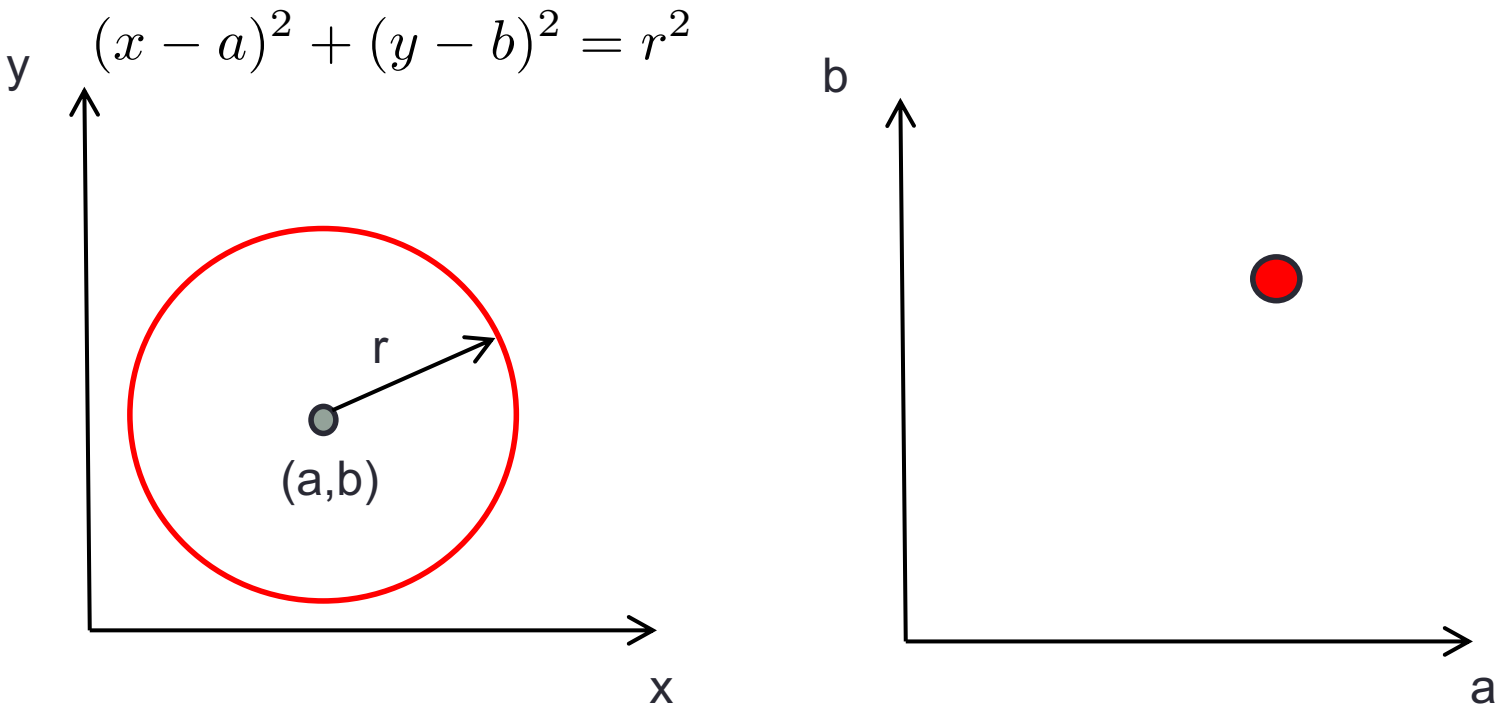
- The same idea of parameter space voting that is used for lines can also be used for detecting/fitting other geometric shapes, such as circles
- For circles of a fixed radius r, we can parameterise the circle based on it's center (a,b):

$$(x-a)^2 + (y-b)^2 = r^2$$



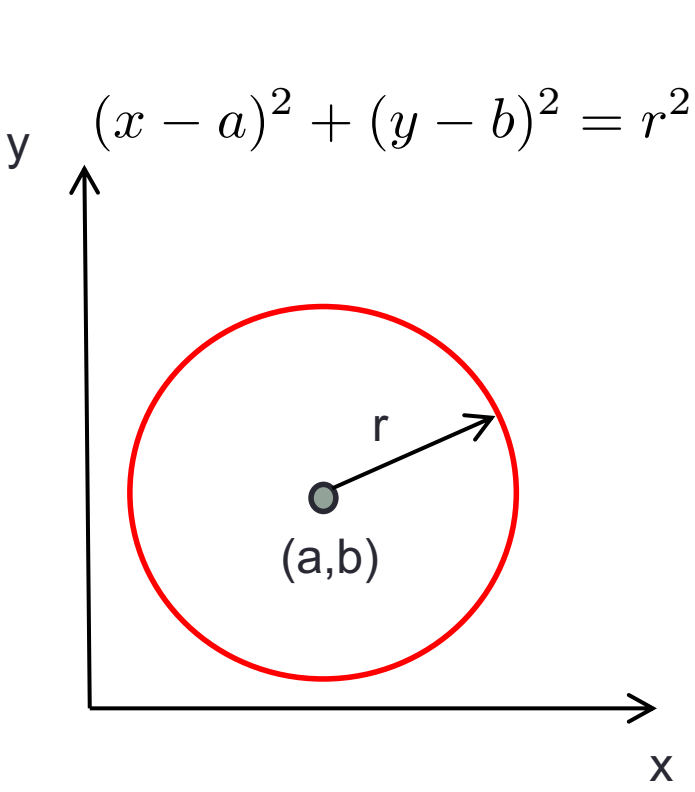
The Circular Hough Transform

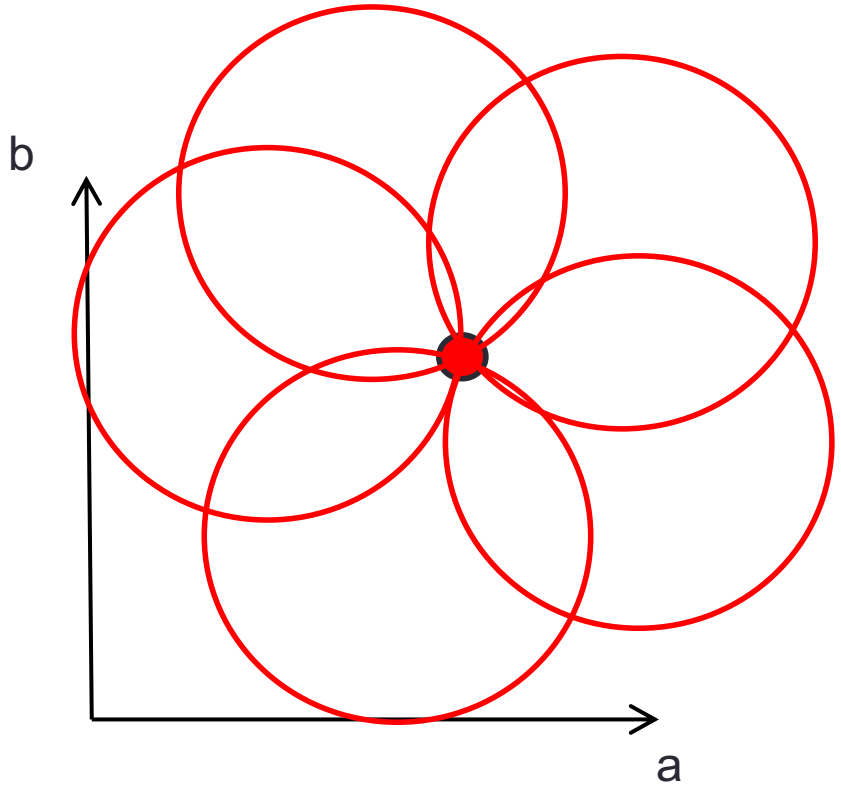
- Consider some edge points: for each edge point we can vote for several circles that intersect with this point: we then cast votes in (a,b) space for each of these circles
- To account for an unknown r, we need to include r in the parameterisation (r,a,b) and include an extra dimension in our histogram



The Circular Hough Transform

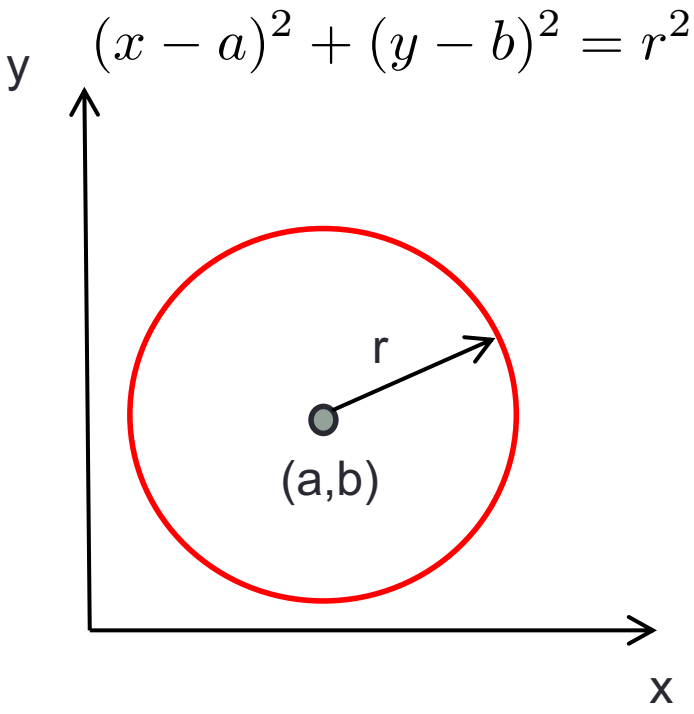
- Consider some edge points: for each edge point we can vote for several circles that intersect with this point: we then cast votes in (a,b) space for each of these circles
- To account for an unknown r, we need to include r in the parameterisation (r,a,b) and include an extra dimension in our histogram

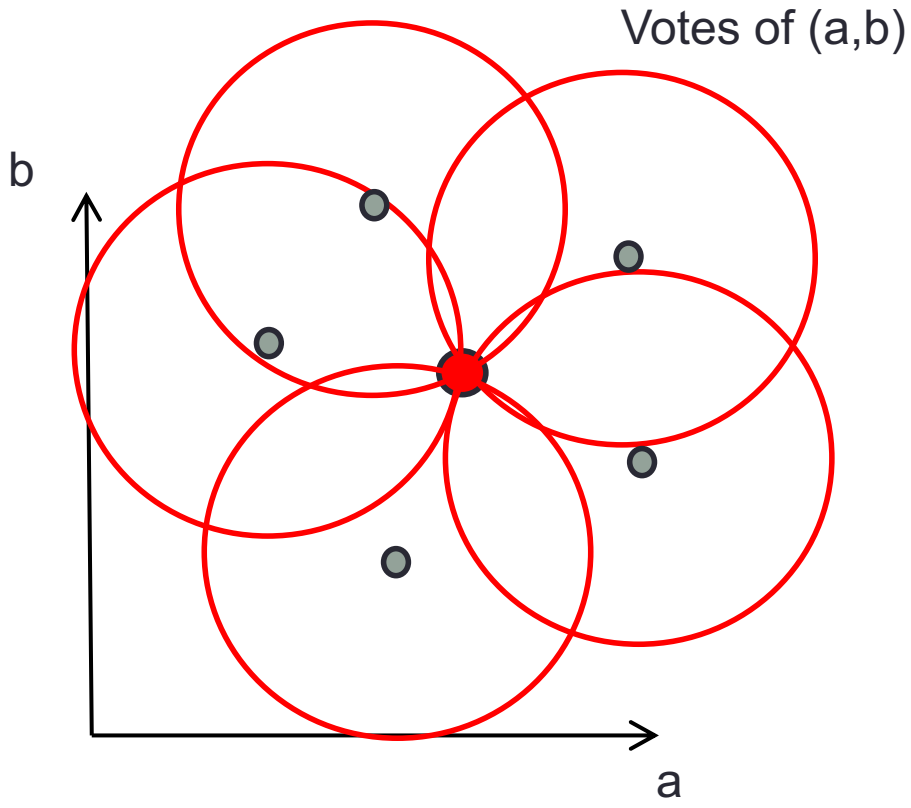




The Circular Hough Transform

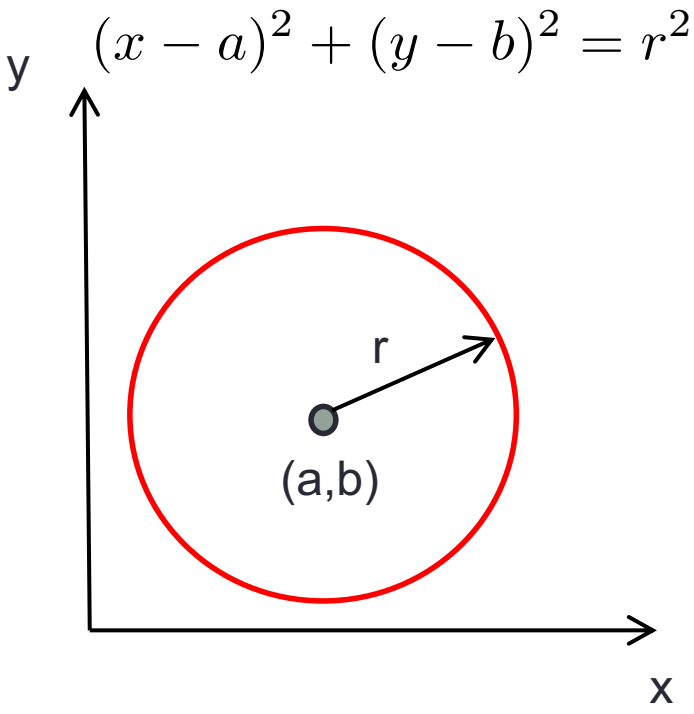
- Consider some edge points: for each edge point we can vote for several circles that intersect with this point: we then cast votes in (a,b) space for each of these circles
- To account for an unknown r, we need to include r in the parameterisation (r,a,b) and include an extra dimension in our histogram

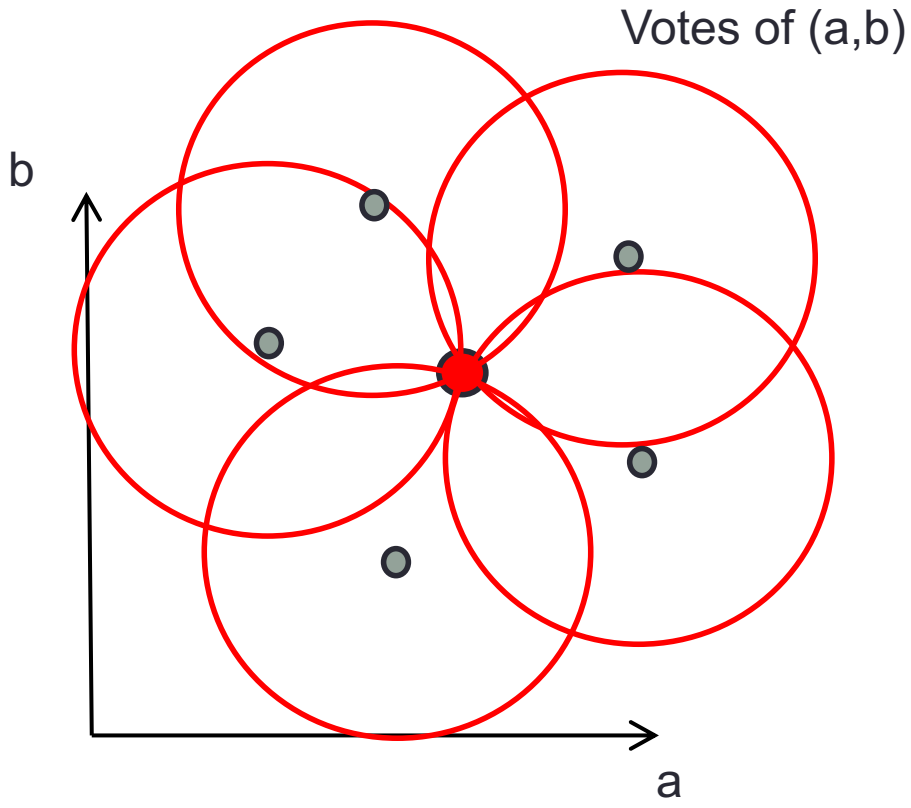




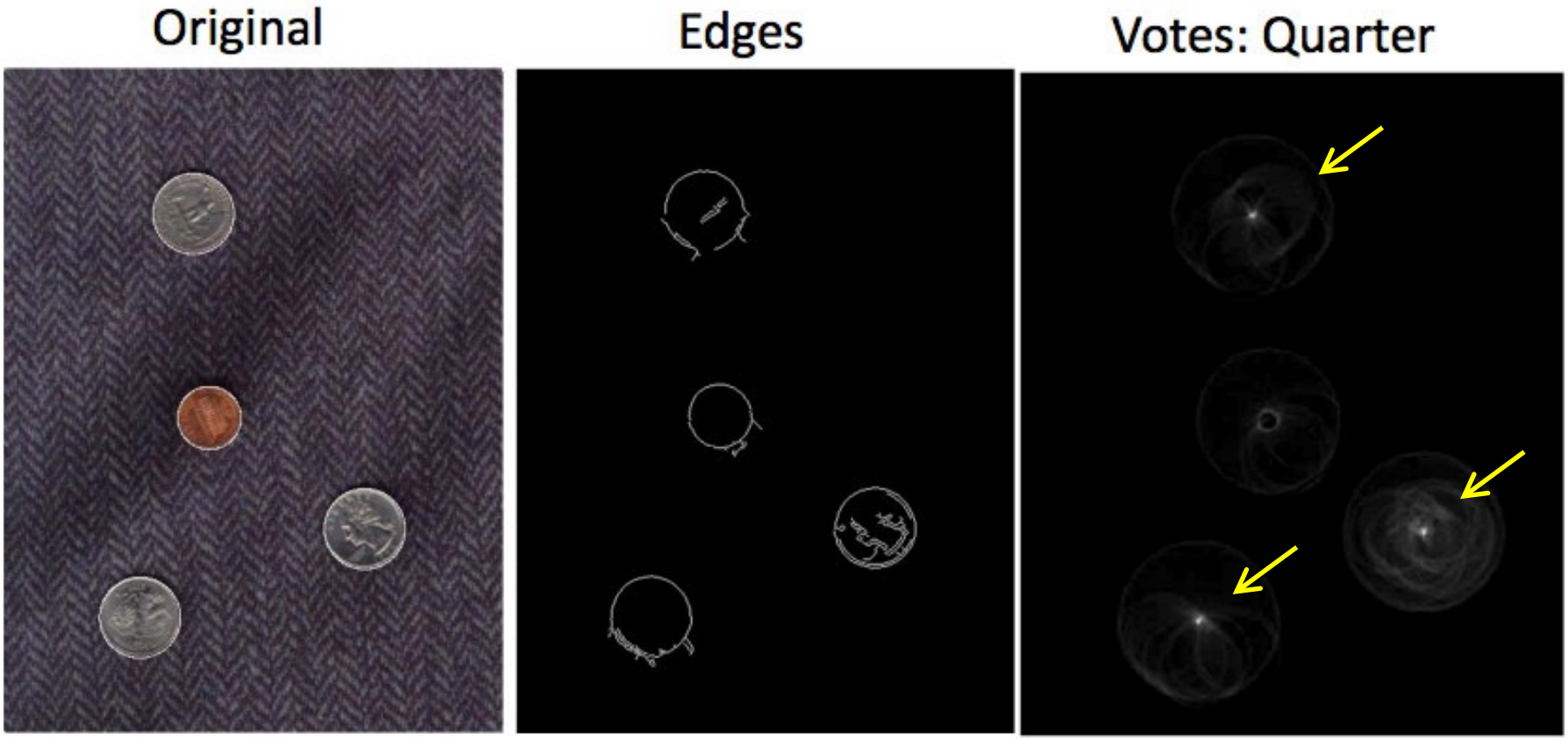
The Circular Hough Transform

- Consider some edge points: for each edge point we can vote for several circles that intersect with this point: we then cast votes in (a,b) space for each of these circles
- To account for an unknown r, we need to include r in the parameterisation (r,a,b) and include an extra dimension in our histogram



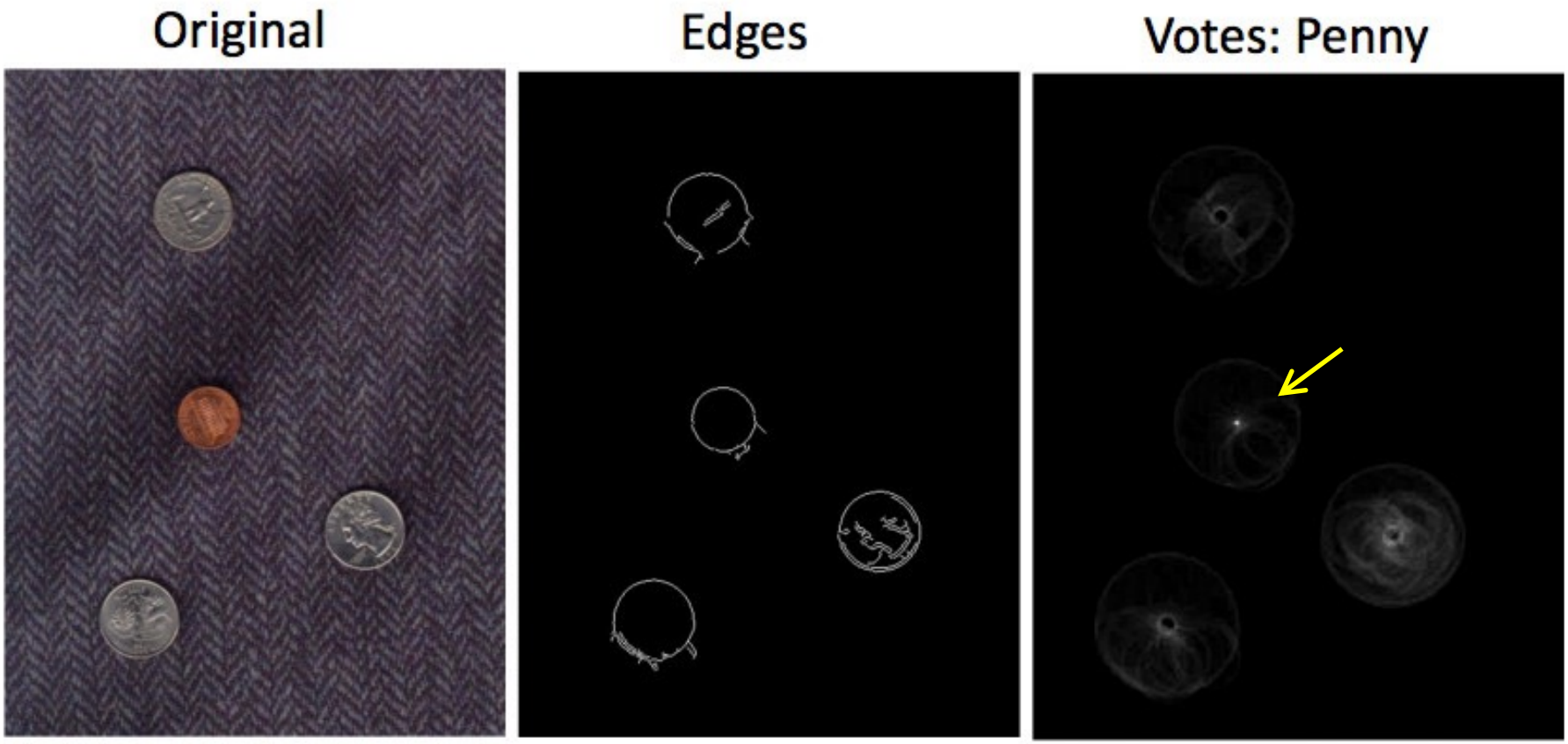


Circular Hough Transform: Example



(a,b) parameter votes for radius chosen to match size of a quarter

Circular Hough Transform: Example



(a,b) parameter votes for radius chosen to match size of a penny

Using Hough Transforms in the MATLAB Image Processing Toolbox

- MATLAB provide implementations of Hough transform functions for both line and circle detection:
  - [H,T,R] = **hough**(bw)
    - Computes the Hough transform of a binary image bw and returns a 2D histogram H of ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$) counts and the values used for ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$) bins in T and R
  - P = **houghpeaks**(H,N)
    - Computes the N highest peaks of the Hough transform H
  - lines = **houghlines**(bw,T,R,P)
    - Extracts lines that include start and end points using ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$) points stored in P and the edge image data itself in bw by tracking continuous gap-free regions along lines
  - centers = **imfindcircles**(bw, radius) or centers = **imfindcircles**(A, [min_radius, max_radius]))
    - Finds circles using a Circular Hough transform of either a fixed radius or over a range of radii values

Advantages/Disadvantages of Voting Methods

- Advantages:
  - All points are processed independently, so can cope with occlusion, gaps
  - Some robustness to noise: noise points unlikely to contribute consistently to any single bin
  - Can detect multiple instances of a model in a single pass

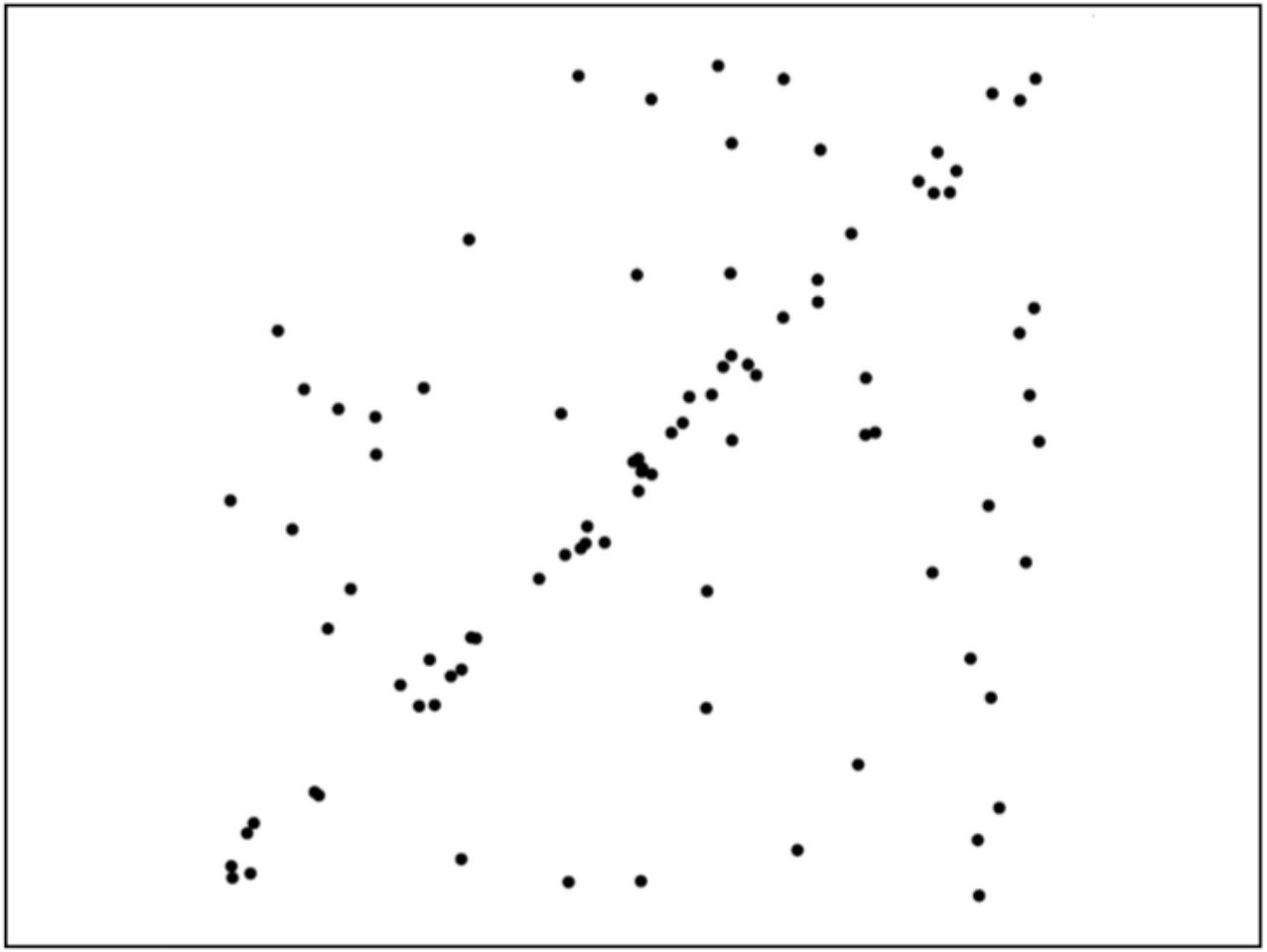
- Disadvantages:
  - Difficult to implement for models that have many parameters (need to maintain an N-D grid to store votes), computational complexity increases exponentially with number of parameters
  - Non-target shapes can produce spurious peaks in parameter space
  - Quantisation of Parameters: can be tricky to pick a good grid size

Model Fitting: RANSAC

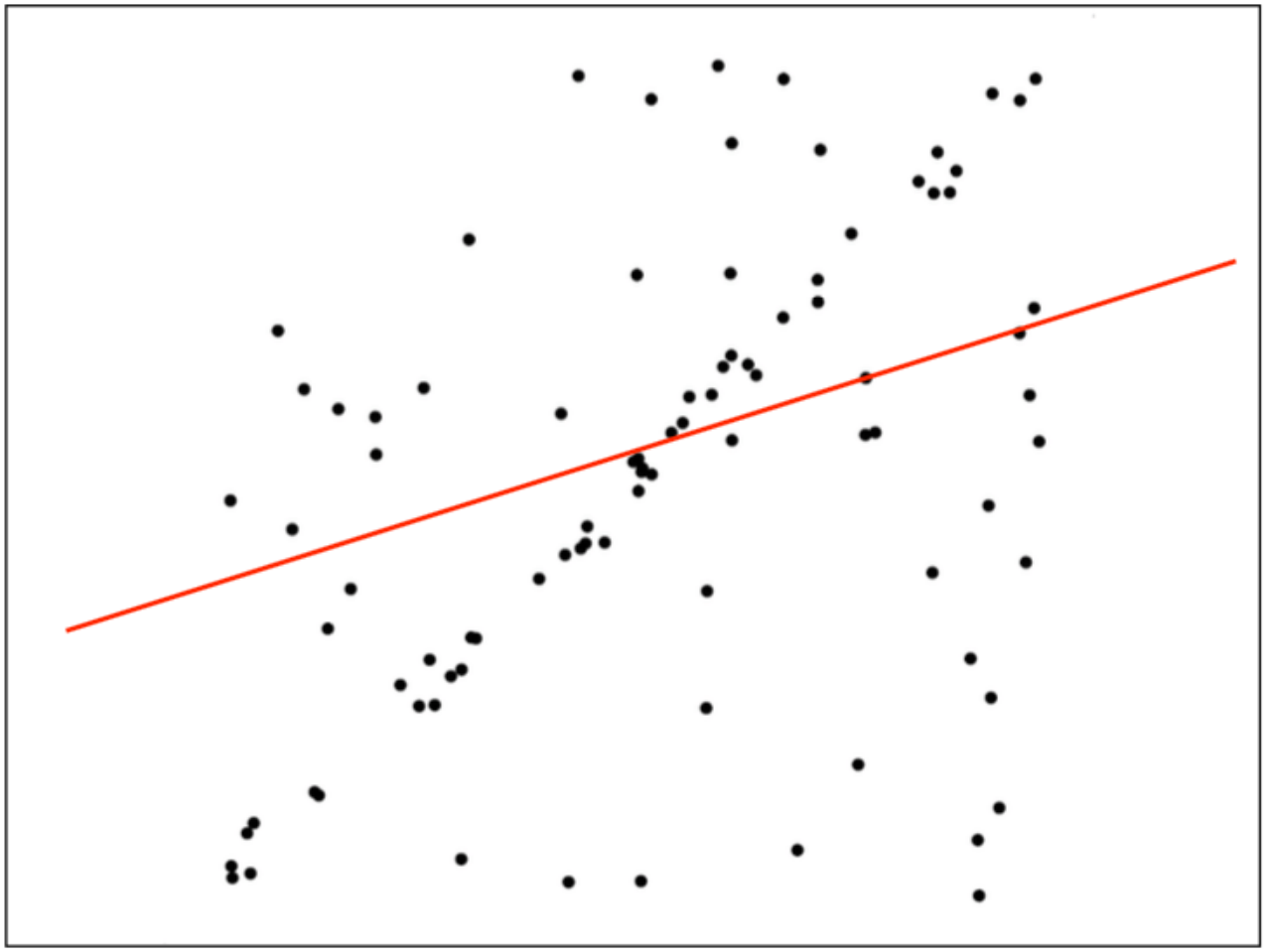
- One alternative to voting schemes is the **RANdom SAmple Consensus (RANSAC)** algorithm:

- The idea of RANSAC is to:
  - Select a random subset of n edge points/pixels and use these to compute a single set of model parameters (i.e. ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$) or (a,b))
  - Check how many of the other edge points/pixel "agree" with this model
  - Iterate over different random subsets until one or several "acceptable" models are found

Model Fitting: RANSAC



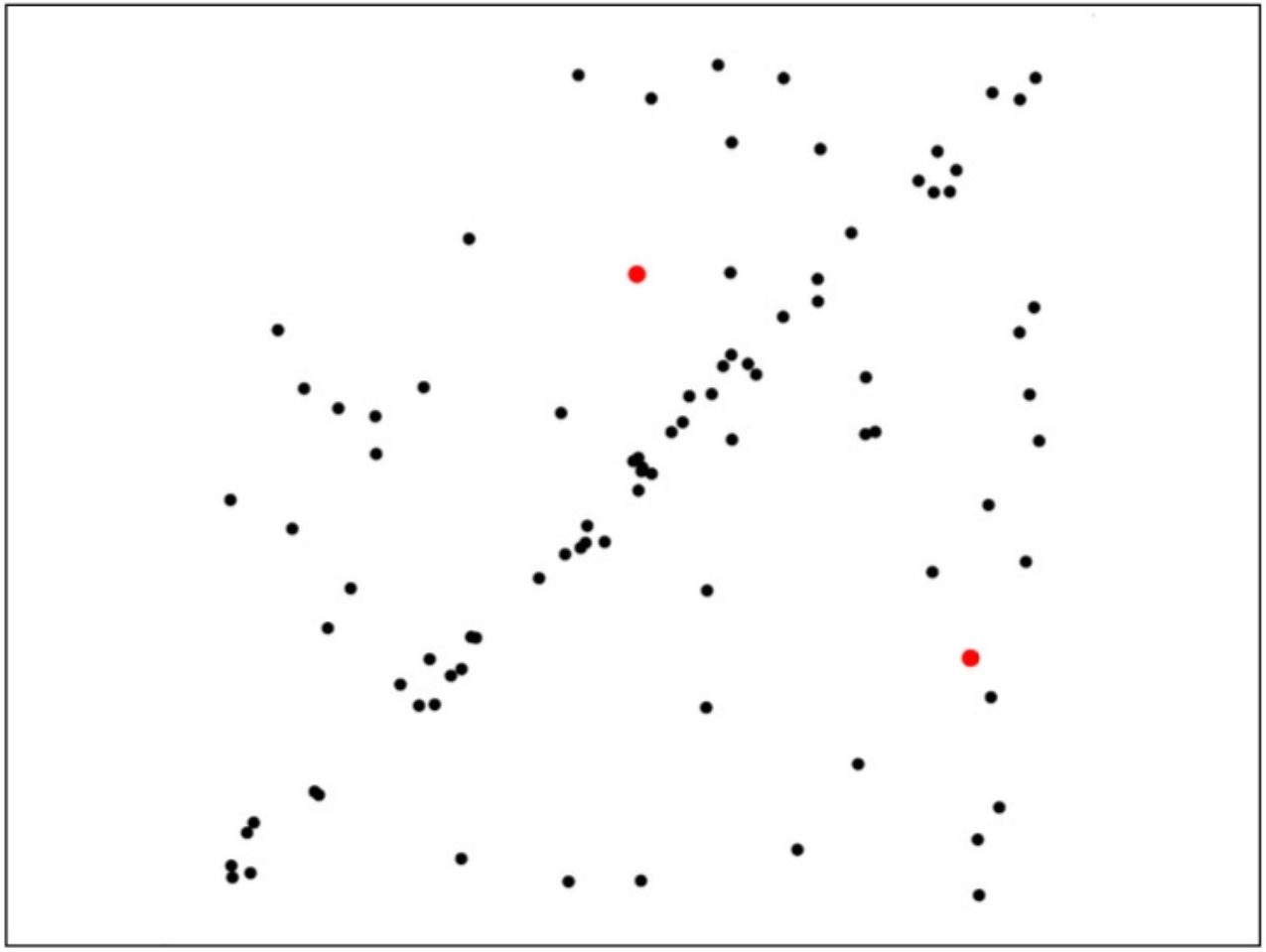
Model Fitting: RANSAC



Least-squares fit

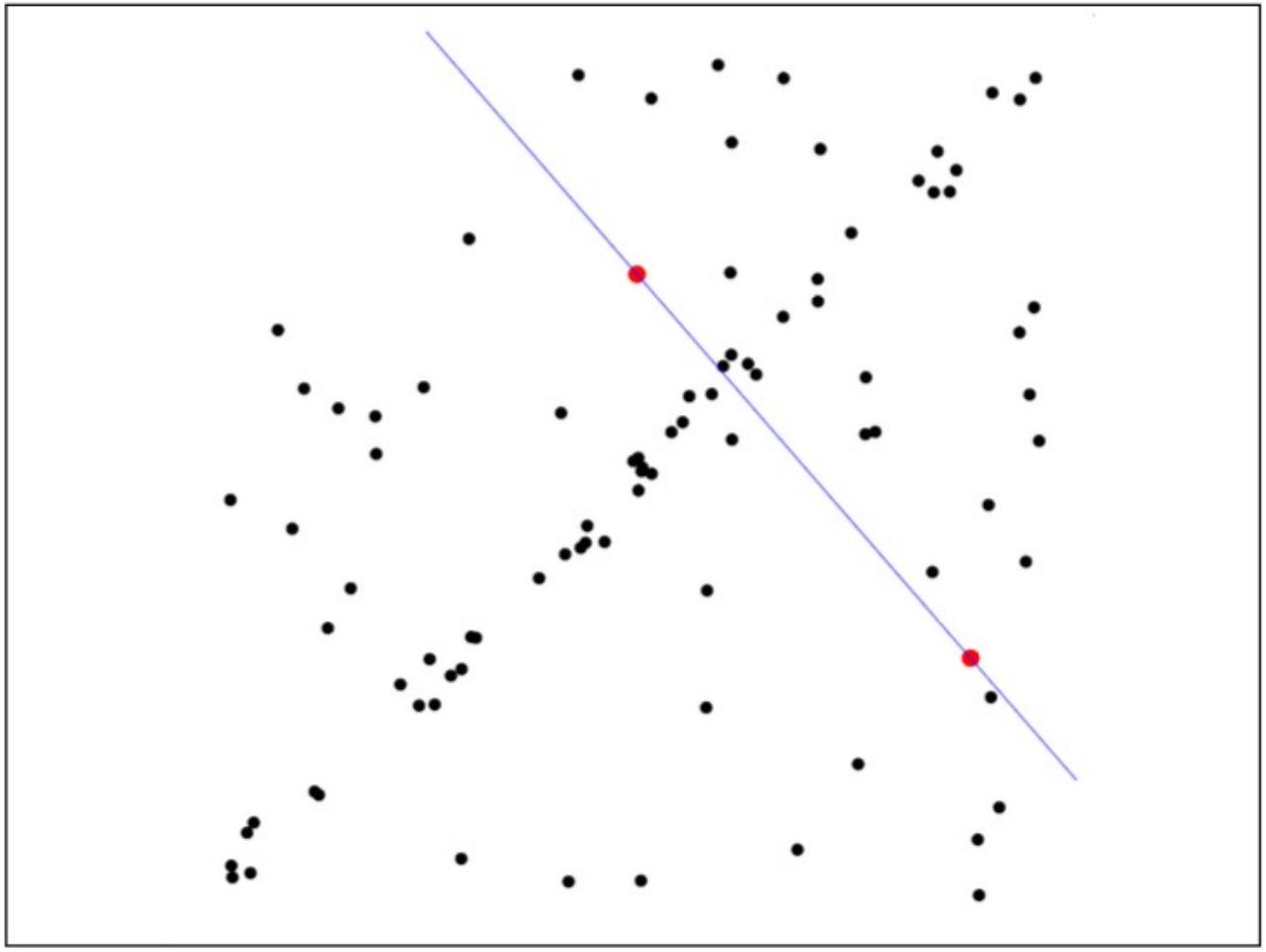
Using all points to compute one set of parameters (m,b) or ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$) that minimse the sum of squared residual distances to the line

Model Fitting: RANSAC



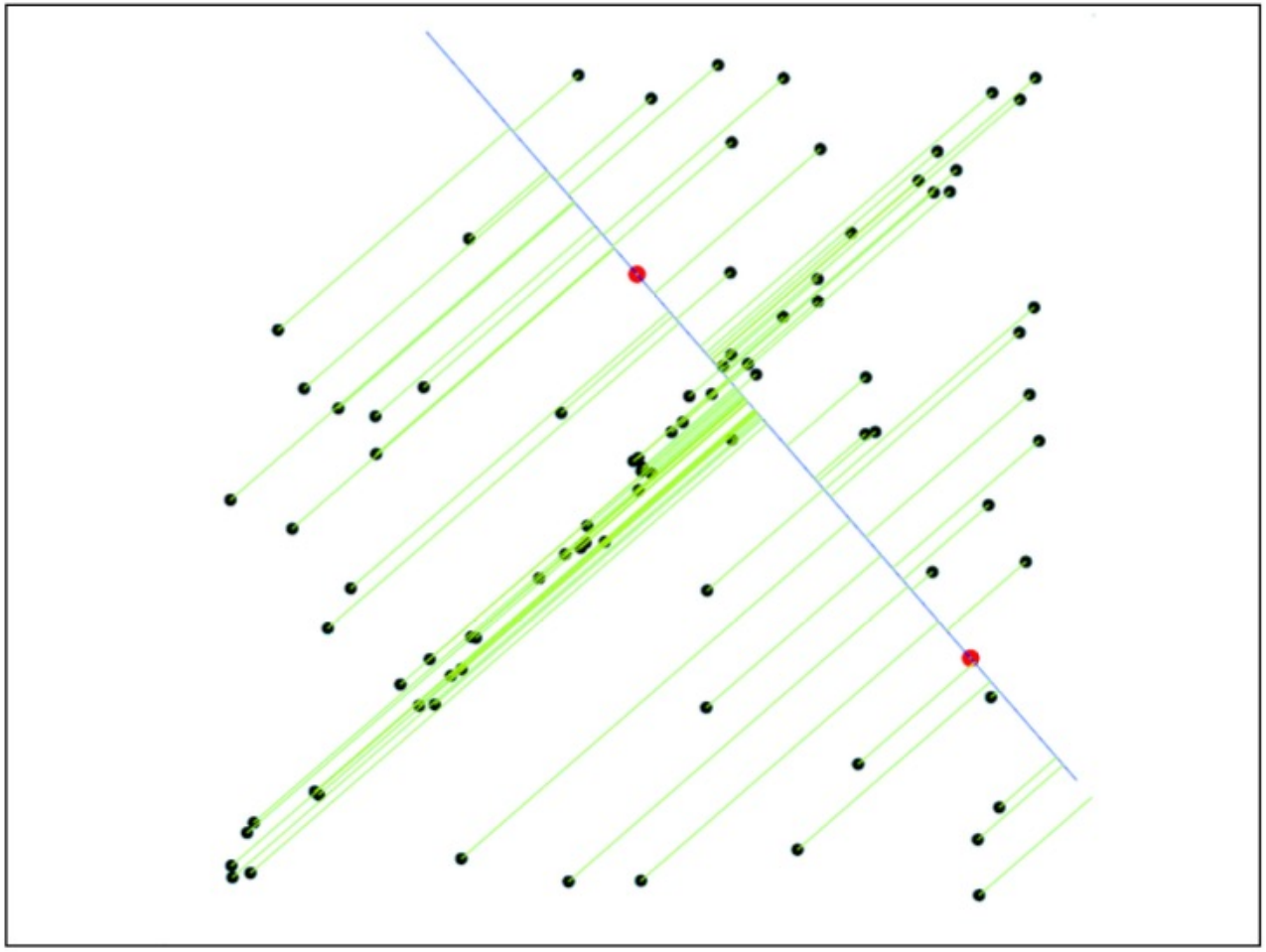
1. Randomly select minimal subset of points

Model Fitting: RANSAC



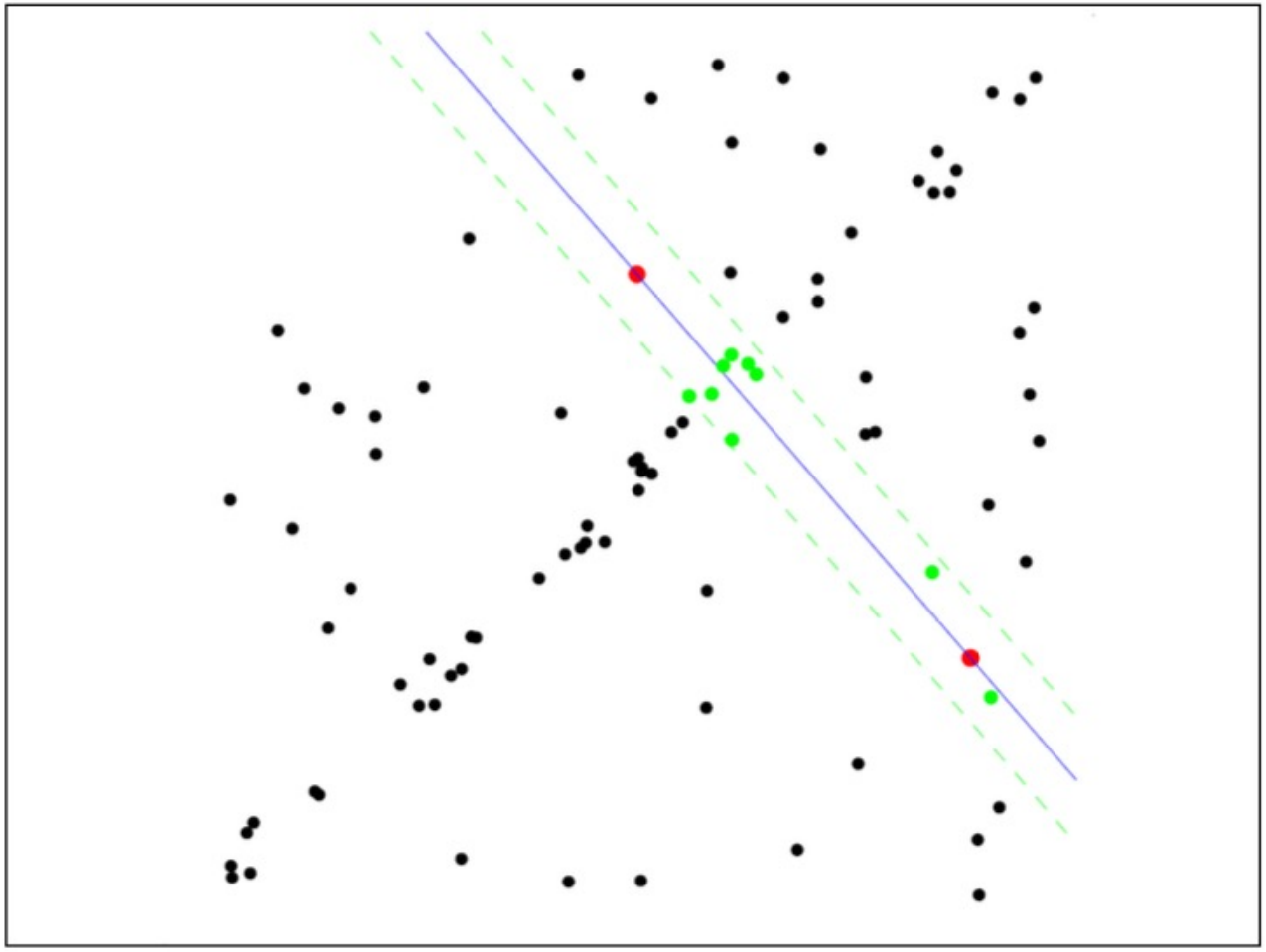
1. Randomly select minimal subset of points
2. Hypothesize a model

Model Fitting: RANSAC



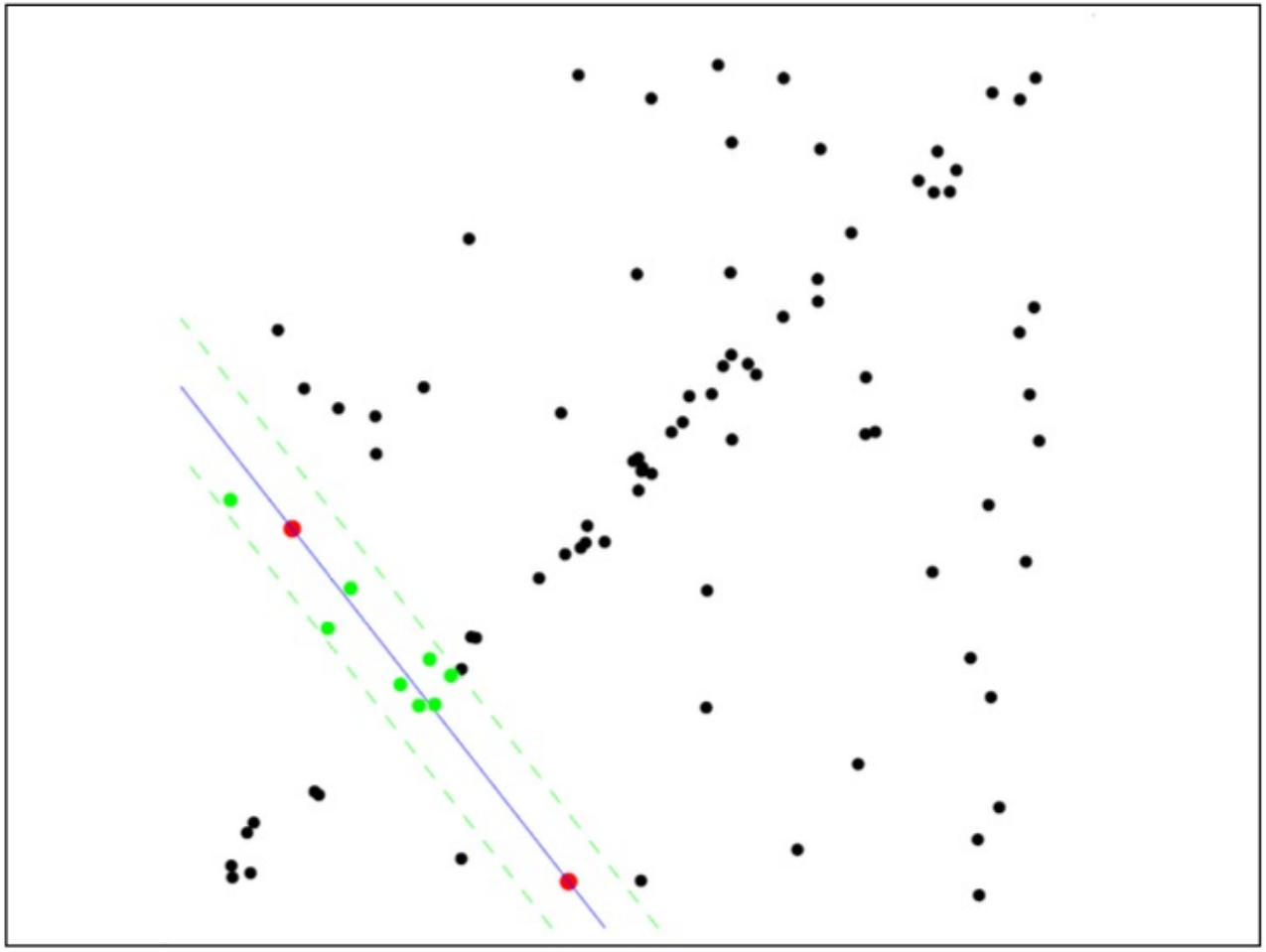
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

Model Fitting: RANSAC



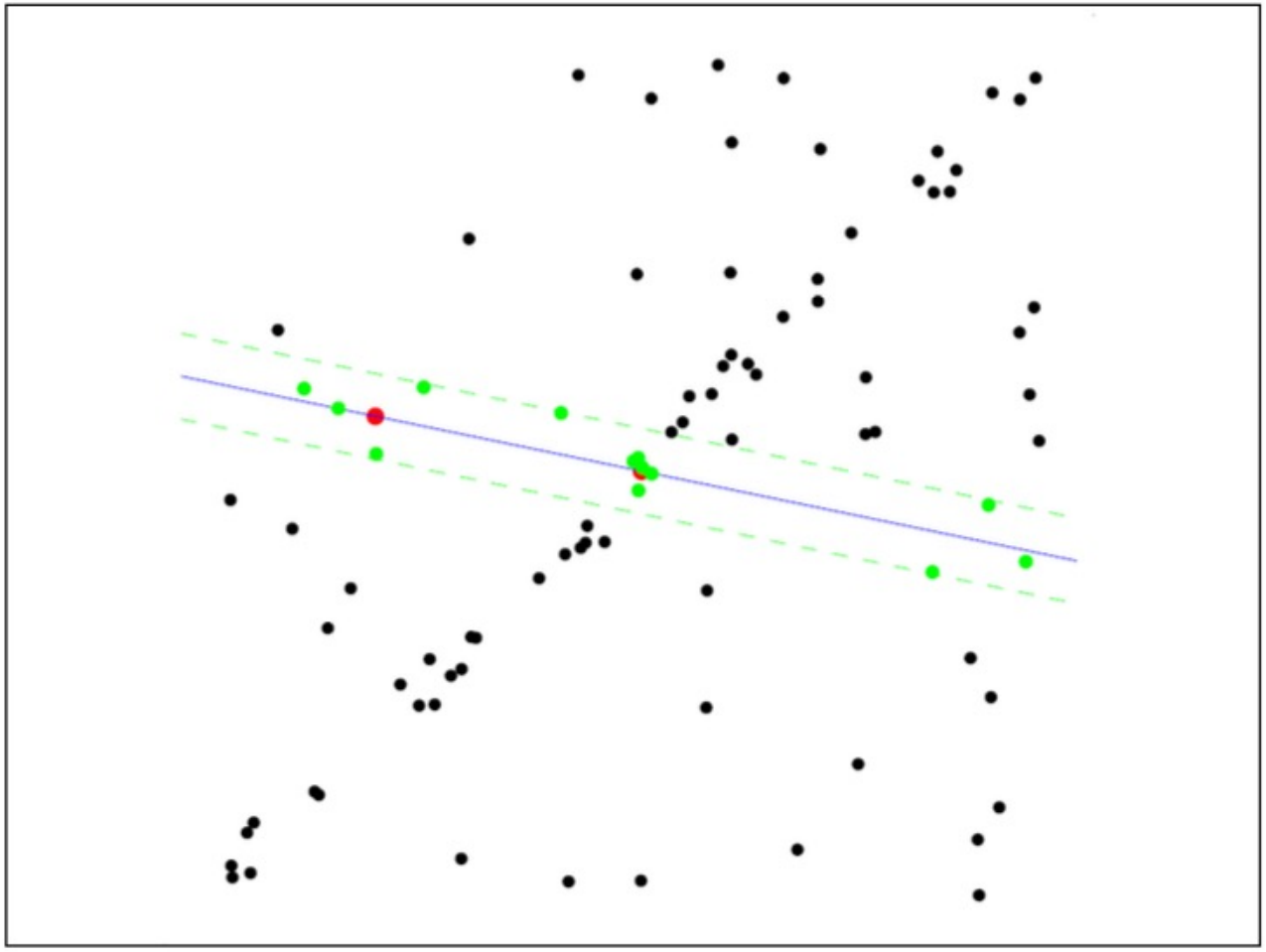
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

Model Fitting: RANSAC



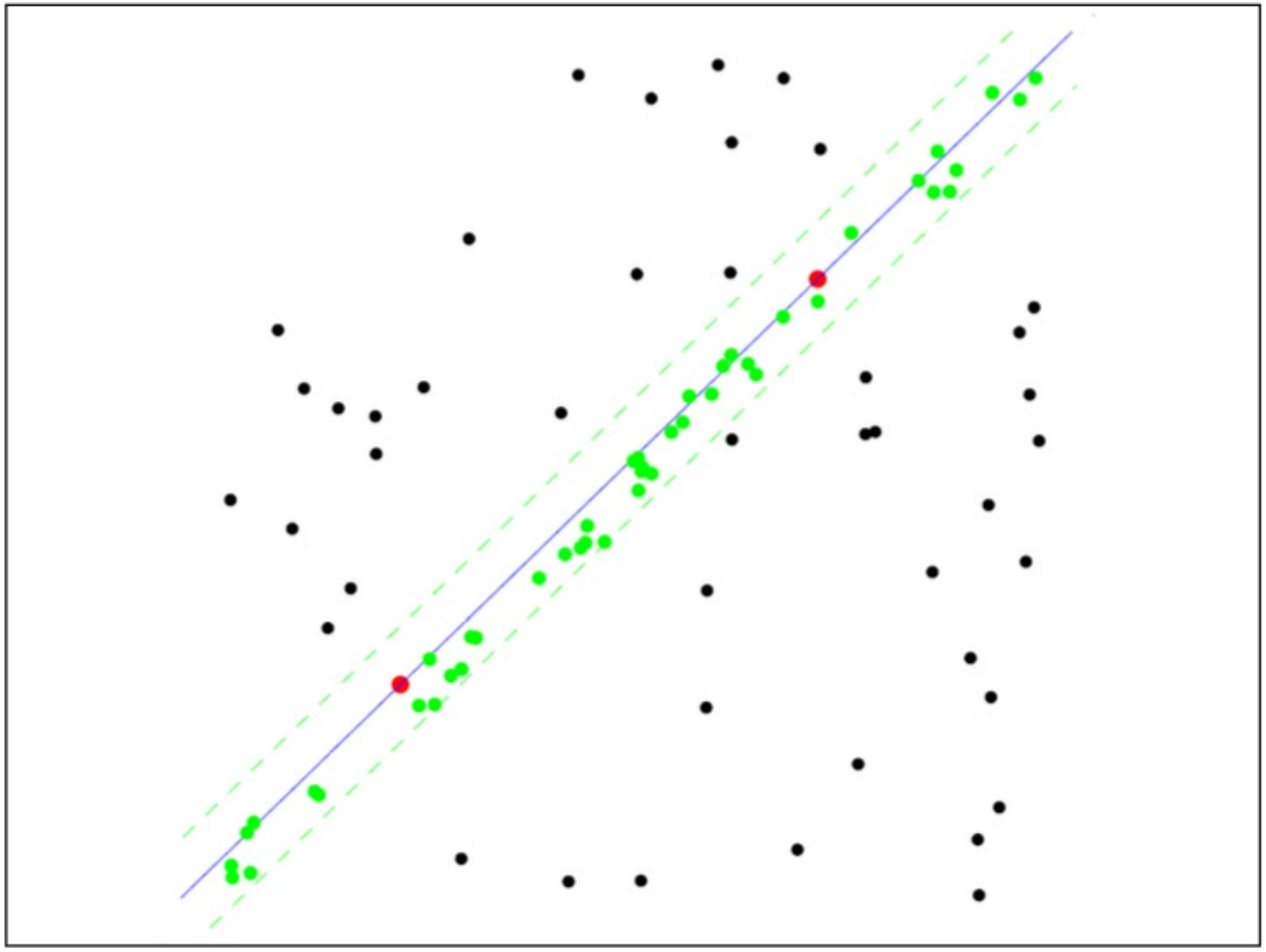
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Model Fitting: RANSAC



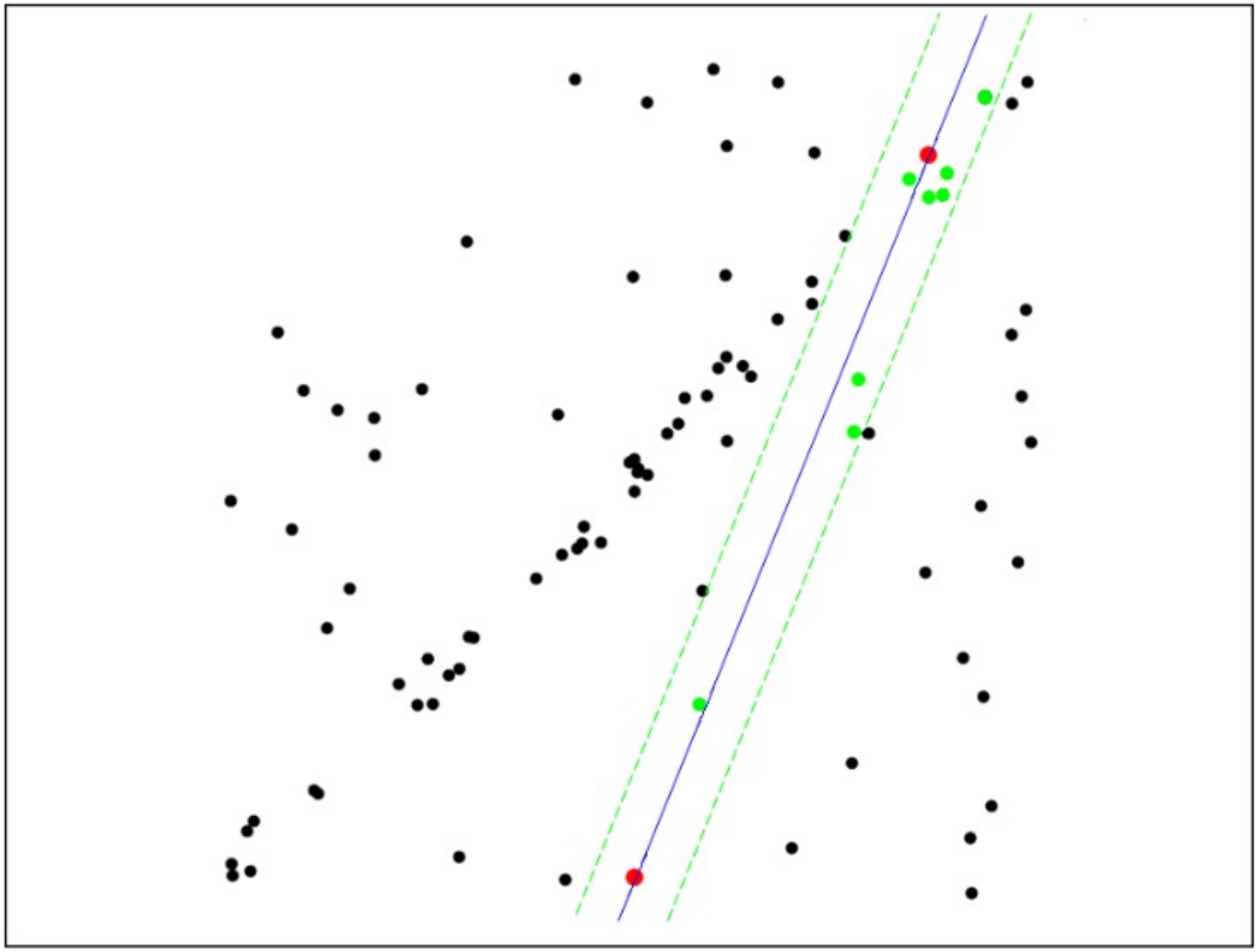
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Model Fitting: RANSAC



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Model Fitting: RANSAC



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Model Fitting: RANSAC

1) Draw n points/pixels uniformly at random

2) Fit model to these n points:
   a) i.e. for lines (n = 2), compute the line parameters (either (m,b) or (ρ,θ)) from the two points

3) Find inliers to this line among the remaining points (i.e. points whose distance from the line is less than a threshold t)

4) If there are d or more inliers, accept the line and possibly refit using all inliers (i.e. via a least-squares approach)

5) Repeat the procedure from 1) N times

Model Fitting: RANSAC

- The intuition behind RANSAC is that if we repeat this procedure N times, where N is a “large” number, then the chances we have detected the correct model is “high”
- In reality, the relationship between N and the probability that we have selected the correct model depends on the proportion of outlier points in the data

Model Fitting: RANSAC

- The intuition behind RANSAC is that if we repeat this procedure N times, where N is a “large” number, then the chances we have detected the correct model is “high”
- In reality, the relationship between N and the probability that we have selected the correct model depends on the proportion of outlier points in the data
- Suppose our model requires at least n points and we iterate N times in the presence of outliers where w is the fraction of inlier points:
- Probability that a single sample of points is correct: w^n
- Probability that all N subsets of n points contain at least one outlier: $(1 - w^n)^N$

Model Fitting: RANSAC

- We can therefore calculate the number of iterations we should make N to achieve a certain level of confidence p that we have found our inlier set (and hence calculated the model)

Sample size	Proportion of outliers						
n	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Required N (iterations) to ensure inlier set found at p = 0.99

Advantages/Disadvantages of RANSAC

- Advantages:
  - General method suited for a wide range of model fitting problems
  - Easy to implement and easy to calculate its failure rate
  - Can deal with models that have higher-dimensions without lots of memory/computational time

- Disadvantages:
  - Only handles a moderate percentage of outliers without cost blowing up (compared to voting methods)
  - Defining a threshold t is not particularly easy when noise in the point increases

5 minute break

Image Interest Points

- Interest points (sometimes referred to as feature points) are points or small local regions in the image that have properties that make them distinctive from other regions
- Interest points may be small local regions in an image that have specific colour, texture or shape

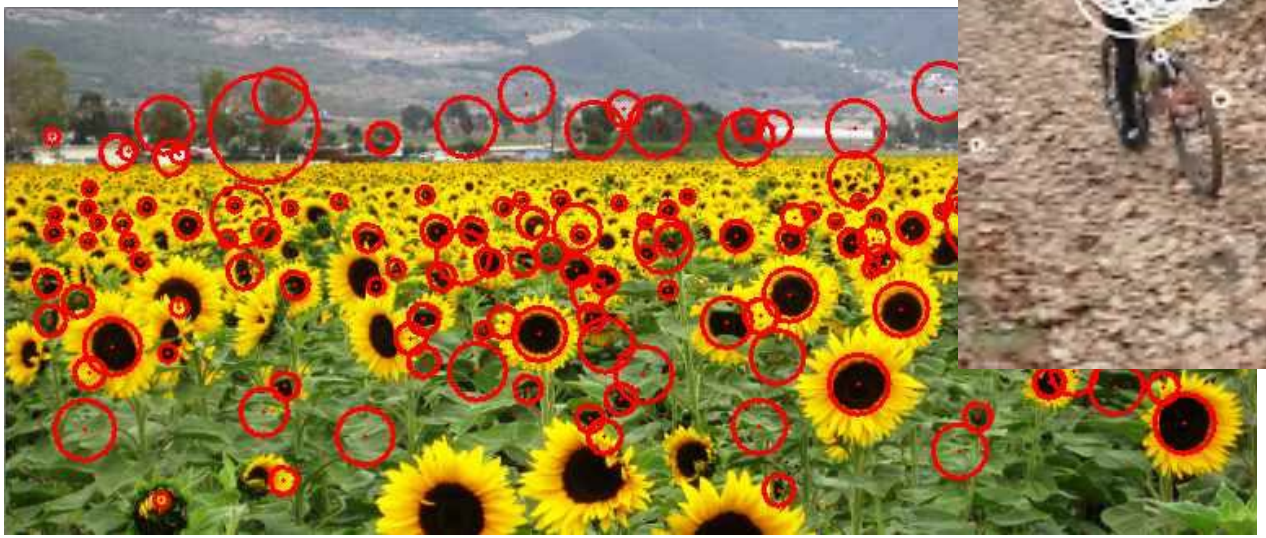


Image Interest Points and Feature-based Object Detection

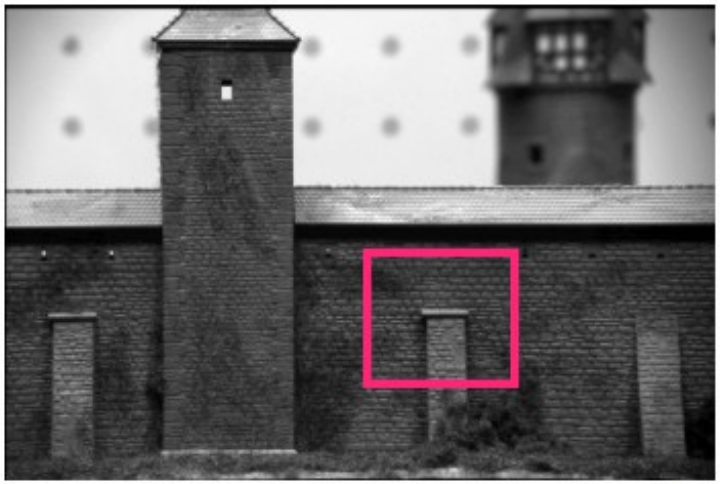
- Interests points can also form the basis for object recognition by matching collections of uniquely identifiable points across two images

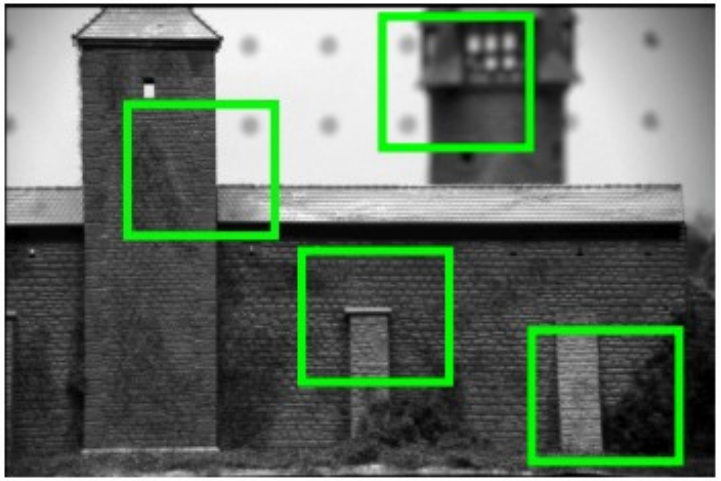
J. Sivic, A. Zisserman, "Video Google: A text retrieval approach to object matching in videos", International Conference on Computer Vision, 2003

D. Lowe, “Distinctive image features from scale-invariant keypoints”, International Journal of Computer Vision, 2004

Image features for matching

- Good features to match should be relatively small, relatively stable against changes in illumination and image orientation and most importantly visually distinct

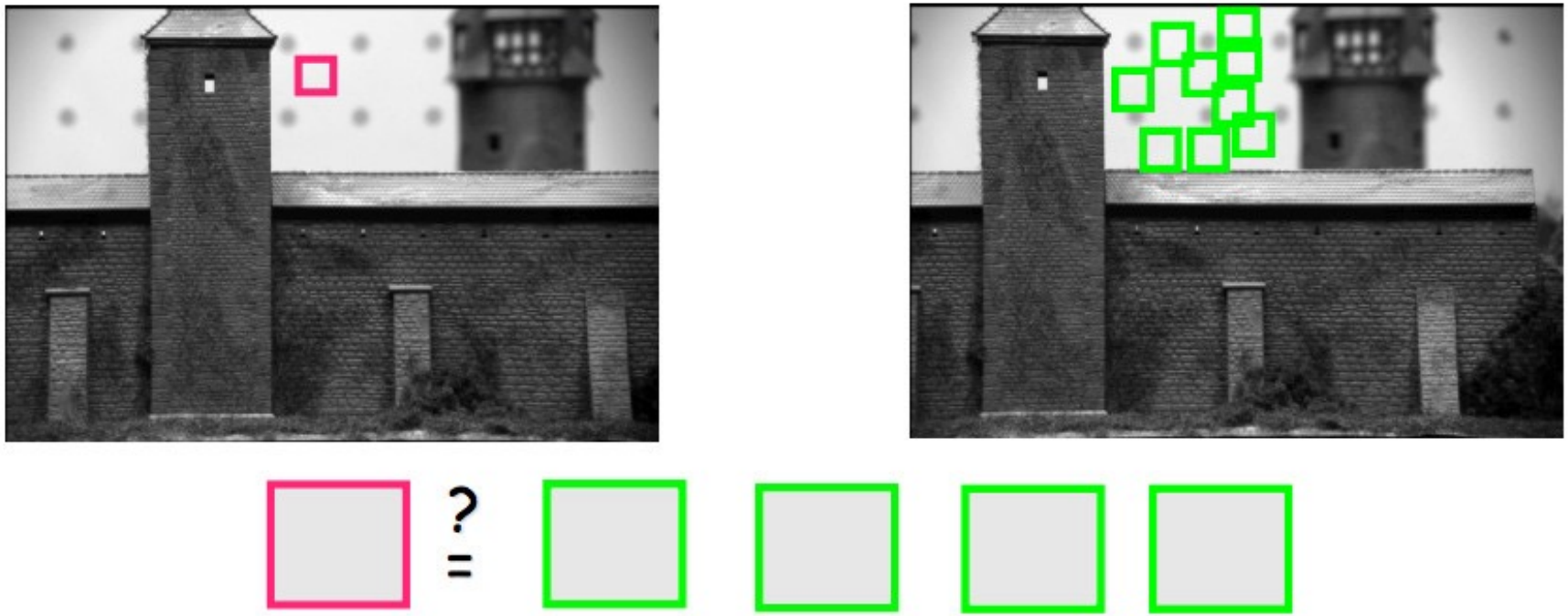




Easy to find matches

Image features for matching

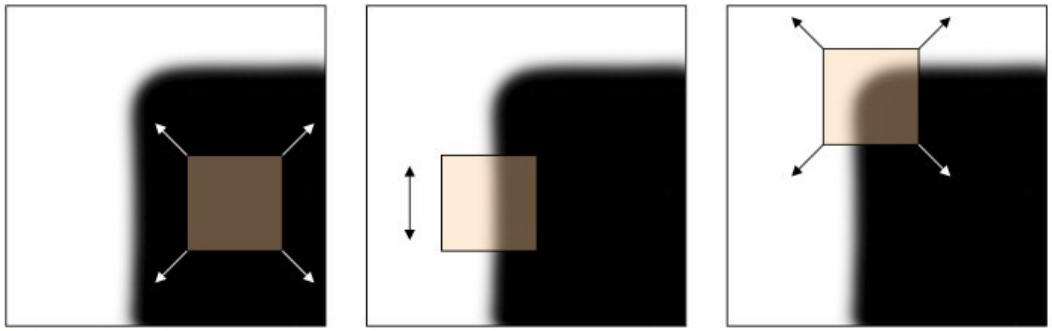
- Good features to match should be relatively small, relatively stable against changes in illumination and image orientation and most importantly visually distinct



Difficult to find matches

Image Corner Points

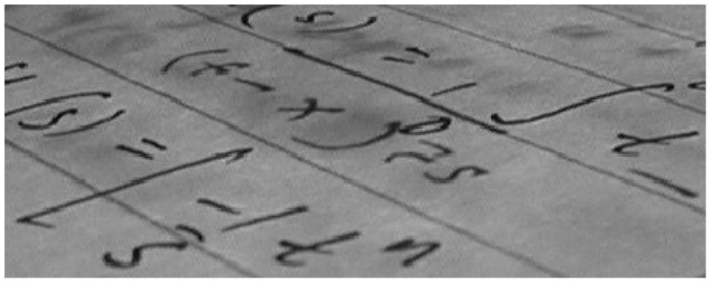
- Corner points are a type of interest point defined by a sharp change in image intensity that occurs across two different directions in the image space
  - Unlike edges, for which the sharp change usually happens just in one direction

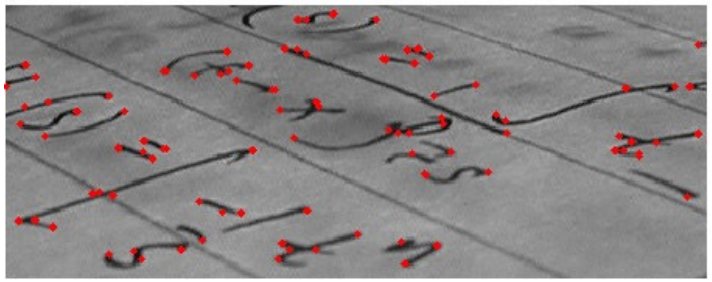


"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

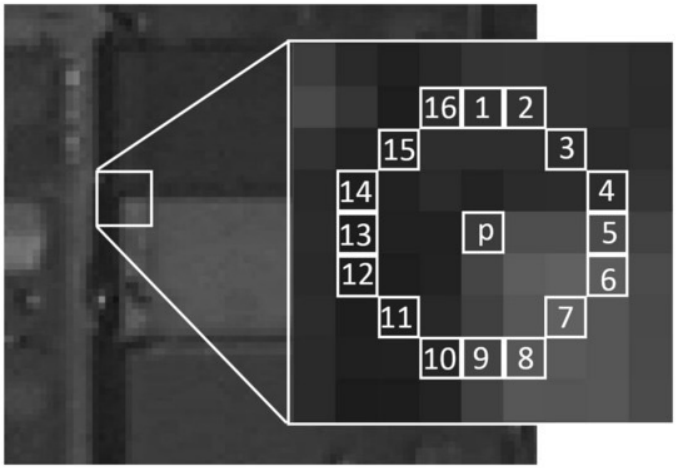




- Sharp changes in different directions make corner points easy to localise and distinct from other regions of the image

Finding and Describing Corner Points

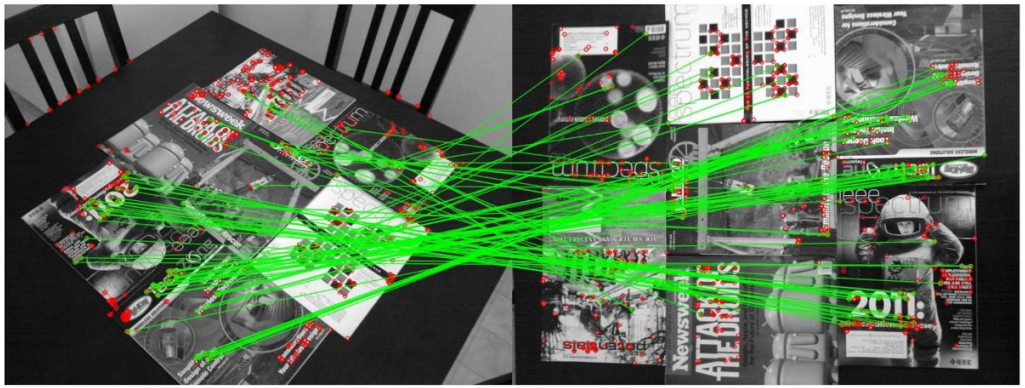
- Popular corner detection algorithms include:
  - Harris Corner Points
  - Features from Accelerated Segment Test (FAST)



FAST corner detection process
E. Rosten, T. Drummond, "Machine Learning for High-speed Corner Detection", ECCV 2006.

- Features typically combine a detector with a descriptor: a summary of the local image region around the feature point that can be used for matching

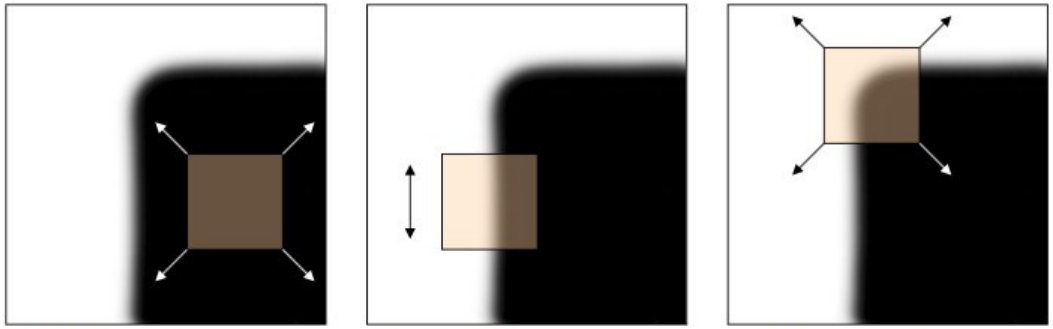
- Examples include:
  - Scale Invariant Feature Transform (SIFT)
  - Speeded Up Robust Features (SURF)
  - Oriented FAST and rotated BRIEF (ORB)



ORB features: combines the FAST detector with BRIEF descriptor
E. Rublee, V. Rabaud, K. Konolige, G. Bradski, "ORB: an efficient alternative to SIFT or SURF", ICCV 2011

Detecting Corners: Harris Corners

- The Harris corner detector* is a simple algorithm for defining “cornerness” and locating regions in the image that make the best corners
- One intuitive notion of “cornerness” is that if we chose an image patch I(u,v), for any small shift I(u+x,v+y) in location in any direction (x,y) we should always end up with an image patch that is quite “different” to our original (i.e. changes sharply)



“flat” region: no change in all directions

“edge”: no change along the edge direction

“corner”: significant change in all directions

*C. Harris and M. Stephens, “A Combined Corner and Edge Detector”, 4th Alvey Vision Conference, pp. 147—151, 1988

Detecting Corners: Harris Corners

- The Harris corner detector* is a simple algorithm for defining "cornerness" and locating regions in the image that make the best corners
- One intuitive notion of "cornerness" is that if we chose an image patch I(u,v), for any small shift I(u+x,v+y) in location in any direction (x,y) we should always end up with an image patch that is quite "different" to our original (i.e. changes sharply)
- Consider an image I and a small patch in the image defined over an area (u,v): if we shift by (x,y) we can compute the weighted sum of square differences between the original and shifted locations:

$$S(x,y) = \sum_u \sum_v w(u,v)\,(I(u+x,v+y) - I(u,v))^2$$

Where w(u,v) is a weight matrix that represents a kernel region over which the response S(x,y) is calculated

*C. Harris and M. Stephens, "A Combined Corner and Edge Detector", 4th Alvey Vision Conference, pp. 147—151, 1988

Detecting Corners: Harris Corners

- We can approximate the image patch at a shifted location using a first-order Taylor series expansion:

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

- Where I_x and I_y are the partial derivatives (gradients) of I in the x and y directions, therefore:

$$S(x,y) \approx \sum_u \sum_v w(u,v) \left(I_x(u,v)x + I_y(u,v)y\right)^2$$

- Which we can represent in matrix form as:

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} M \begin{pmatrix} x \\ y \end{pmatrix}$$

Where $$M = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x(u,v)^2 & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y(u,v)^2 \end{bmatrix}$$

*C. Harris and M. Stephens, “A Combined Corner and Edge Detector”, 4th Alvey Vision Conference, pp. 147—151, 1988

Detecting Corners: Harris Corners

- The Harris Matrix M is indicative of the magnitude of S(x,y) when moving in different directions:

$$M = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x(u,v)^2 & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y(u,v)^2 \end{bmatrix}$$

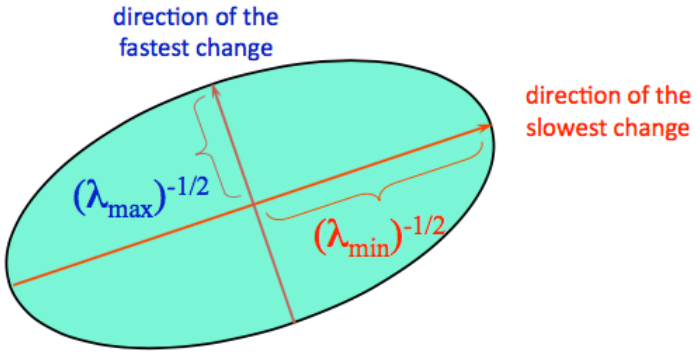
- If we take an eigen decomposition of M, the eigenvalues correspond to maximal and minimal coefficients of image intensity change, where the eigenvectors in R represent the directions of the principle axes

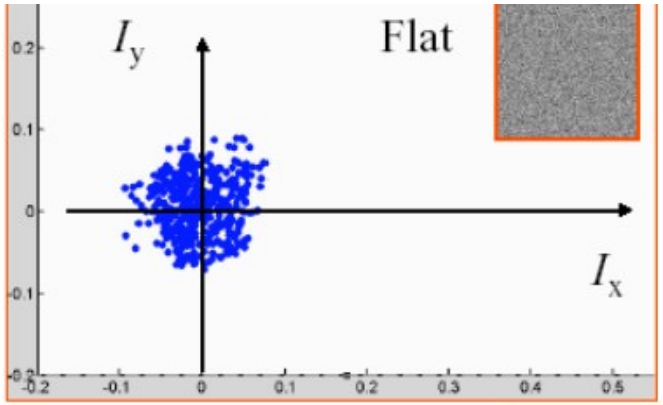
$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

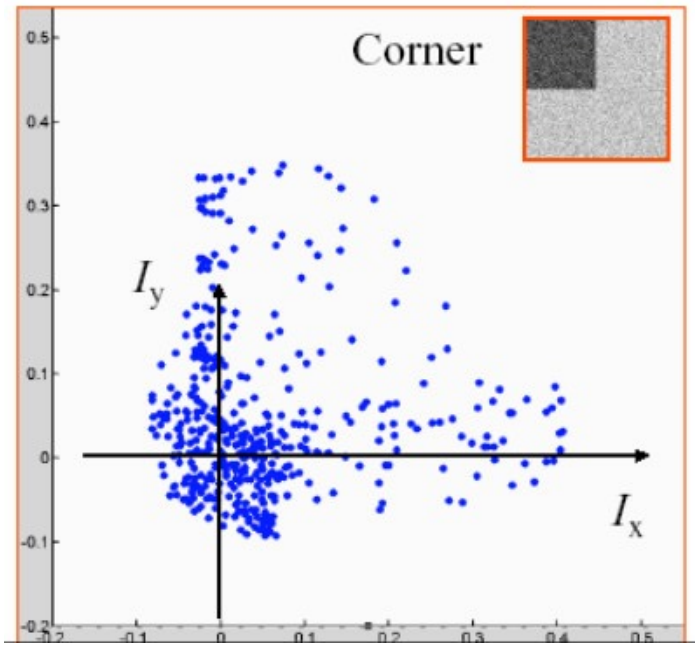
*C. Harris and M. Stephens, “A Combined Corner and Edge Detector”, 4th Alvey Vision Conference, pp. 147—151, 1988

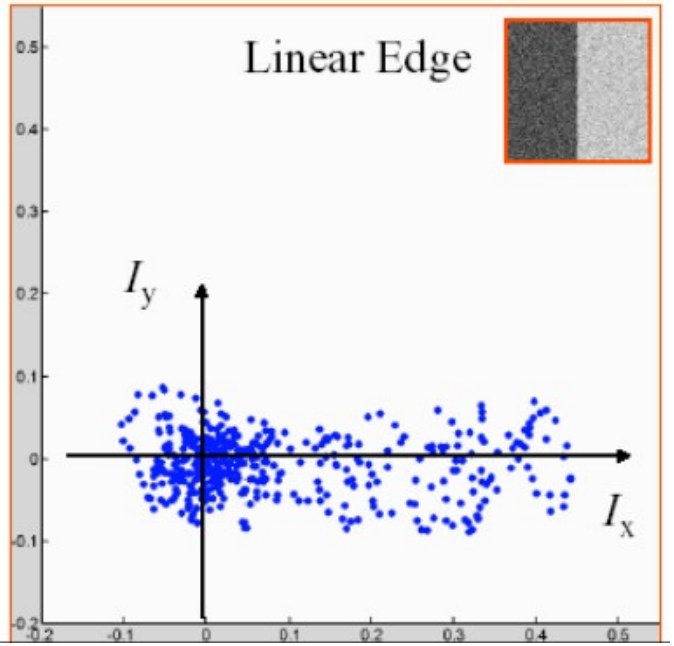
Detecting Corners: Harris Corners

- Consider the distribution of pixel gradient values for several example image patches: M represents a covariance matrix of these values







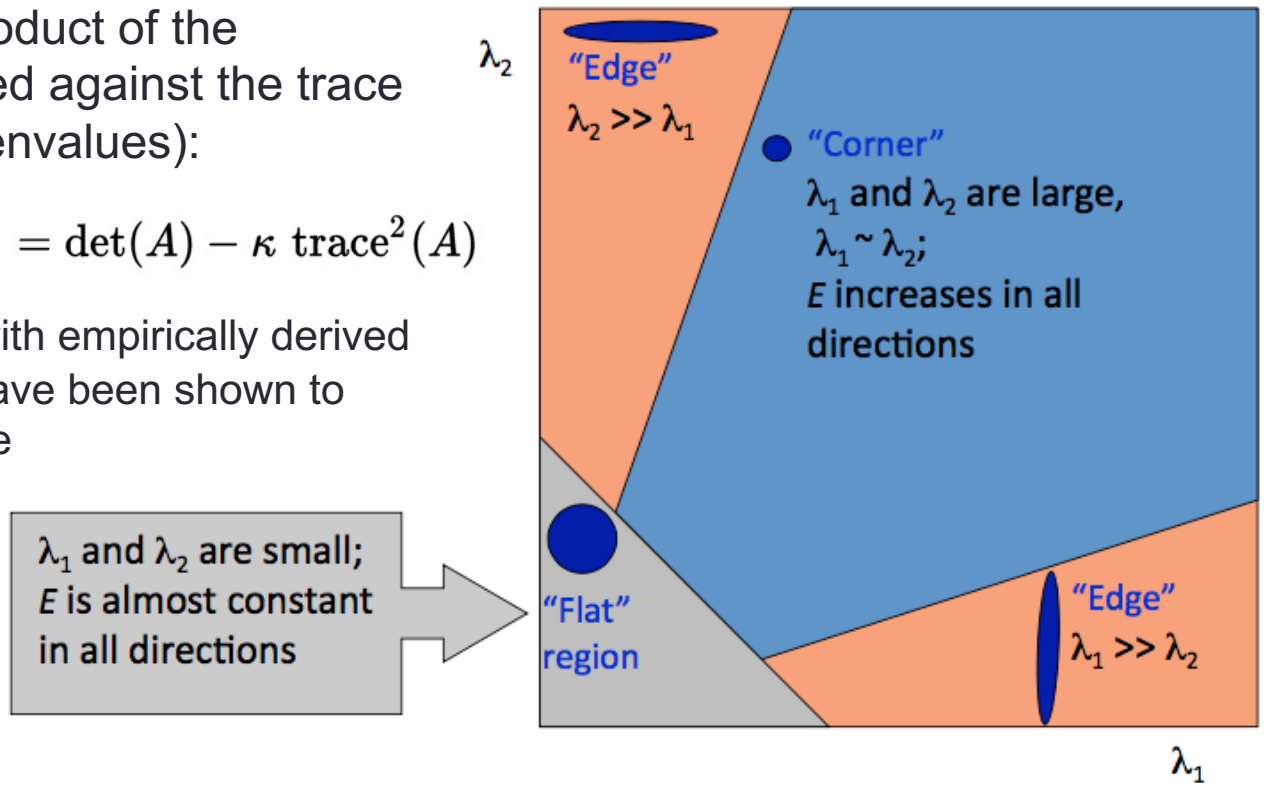


Detecting Corners: Harris Corners

- For good corners, the deviation in similarity as we move in any direction (x,y) should be maximal: hence **both** eigenvalues of M should be large for the image patch to be considered a good corner
- A single measure for **both** eigenvalues being large is to maximise the determinant of M (product of the eigenvalues) balanced against the trace of M (sum of the eigenvalues):

$$M_c = \lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2 = \det(A) - \kappa \, \mathrm{trace}^2(A)$$

Where κ is a constant with empirically derived values of 0.04 to 0.15 have been shown to work well in the literature



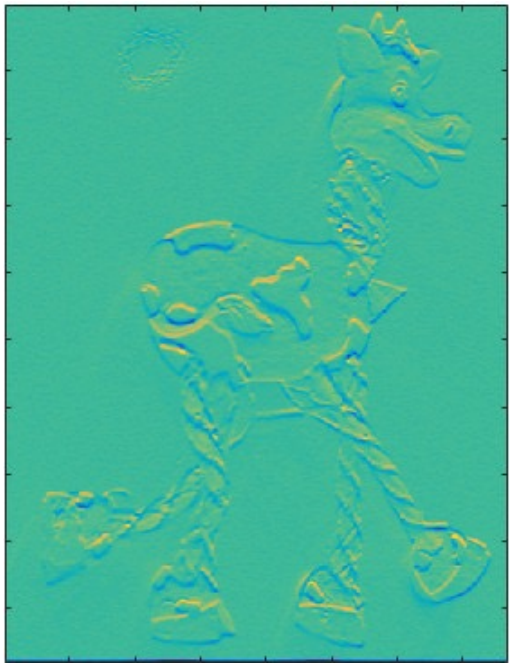
*C. Harris and M. Stephens, "A Combined Corner and Edge Detector", 4th Alvey Vision Conference, pp. 147—151, 1988

Harris Corners: Overview

1) Compute derivatives (gradients) at each pixel
2) Compute second moment matrix *M* in a window around each pixel using gradients
3) Compute corner response function *Mc* from M
4) Find points with large corner response (Mc > threshold)
5) Take the points of local maxima of Mc

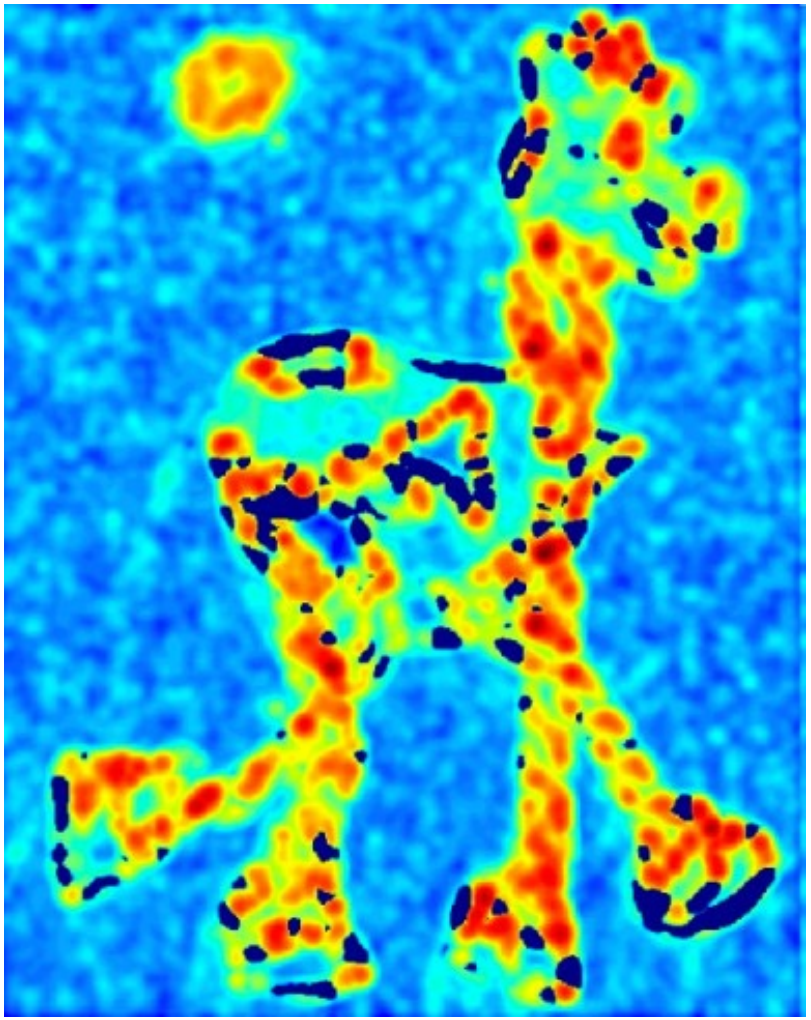
Harris Corners: Overview

1) Compute derivatives (gradients) at each pixel
2) Compute second moment matrix *M* in a window around each pixel using gradients
3) Compute corner response function *Mc* from M
4) Find points with large corner response (Mc > threshold)
5) Take the points of local maxima of Mc



Harris Corners: Overview

1) Compute derivatives (gradients) at each pixel
2) Compute second moment matrix *M* in a window around each pixel using gradients
3) Compute corner response function *Mc* from M
4) Find points with large corner response (Mc > threshold)
5) Take the points of local maxima of Mc



Harris Corners: Overview

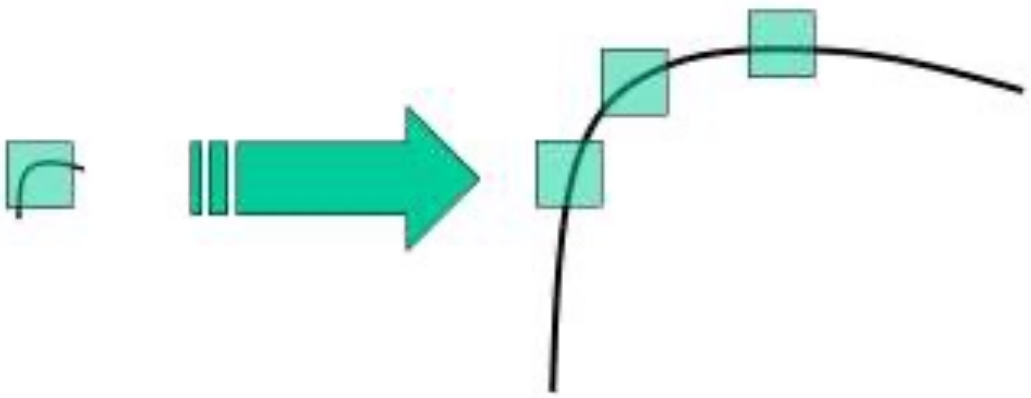
1) Compute derivatives (gradients) at each pixel
2) Compute second moment matrix *M* in a window around each pixel using gradients
3) Compute corner response function *Mc* from M
4) Find points with large corner response (Mc > threshold)
5) Take the points of local maxima of Mc

Harris Corners: Overview

1) Compute derivatives (gradients) at each pixel
2) Compute second moment matrix *M* in a window around each pixel using gradients
3) Compute corner response function *Mc* from M
4) Find points with large corner response (Mc > threshold)
5) Take the points of local maxima of Mc

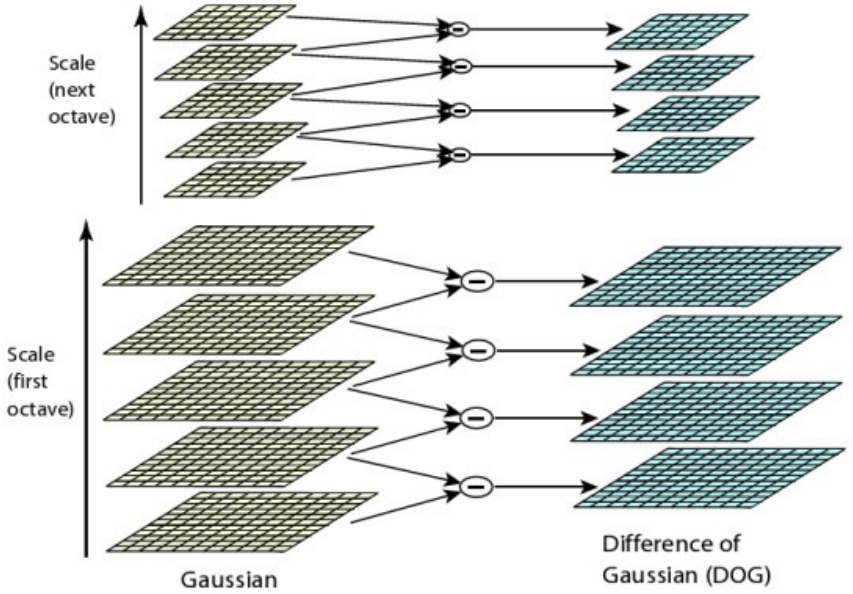
Properties of Harris Corners

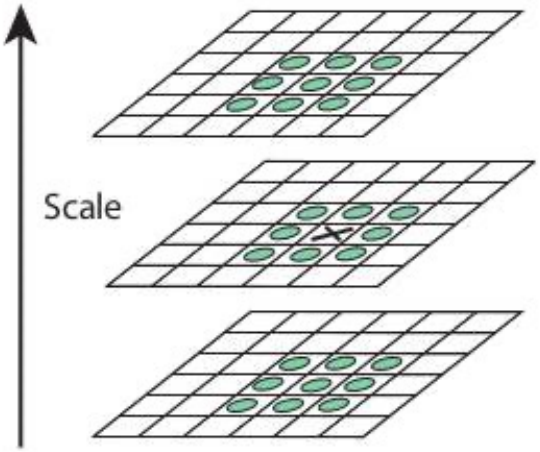
- Harris corners are invariant to translation and rotation of an image: we still detect the same points if we rotate the image
- Harris corners are not invariant to changes in scale: if we change the resolution of the image or size of the patch, we get different corners:



Scale Invariant Features: SIFT and SURF

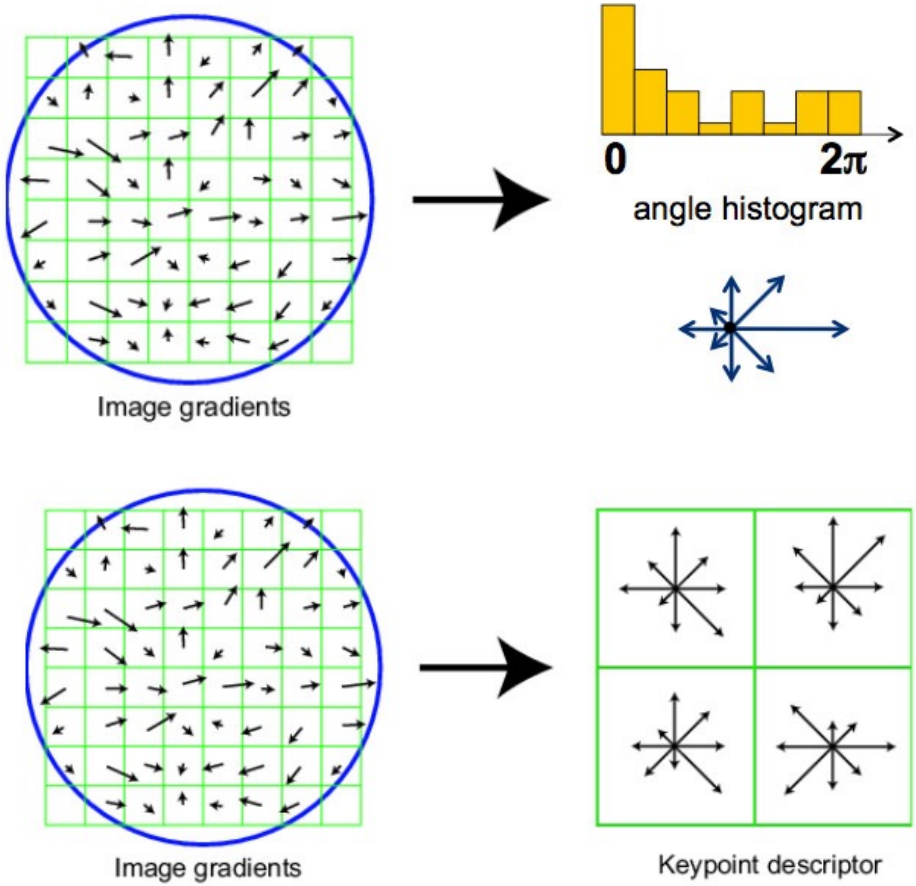
- Scale-Invariant Feature Transform (SIFT) and Speeded-Up Robust Features (SURF) are examples of two corner-like features that are invariant to scale
- These algorithms begin by looking for potential features by building a “pyramid” of images at successively blurred images and looking at differences in the image intensities between subsequent blurring levels
- The local image region at this local position AND scale becomes a potential interest point





Feature Descriptors: SIFT

- For SIFT, histograms are then constructed using gradient information within a 16x16 pixel patch of the image data at the selected scale:
  - A 4x4 grid of cells (2x2 case shown below) is taken from the 16x16 patch
  - An angle histogram is computed for each cell using 8 bins
  - 16 cells * 8 orientations = 128 dimensional descriptor
- The angle histograms are measured with respect to the principle axes of the Hessian matrix, so as to achieve rotational invariance of the descriptor



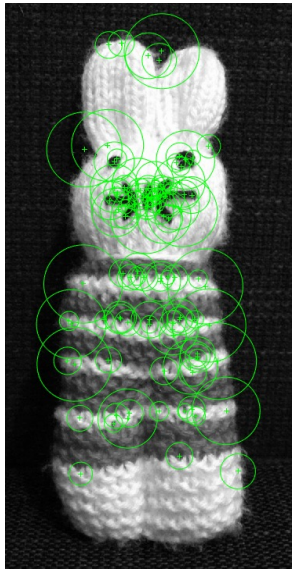
Use of SIFT in Object Detection

- The resulting descriptor/feature vector has been shown to be very robust to noise and viewing angle and is distinct to the local image intensity information associated with the patch
- SIFT and SURF features hence become useful for object detection: we can detect a series of interest points in both a source image and template image and look for our object by finding matching sets of correspondences between images

Use of SIFT in Object Detection

- The resulting descriptor/feature vector has been shown to be very robust to noise and viewing angle and is distinct to the local image intensity information associated with the patch
- SIFT and SURF features hence become useful for object detection: we can detect a series of interest points in both a source image and template image and look for our object by finding matching sets of correspondences between images

Object Template Image



SIFT Features

Use of SIFT in Object Detection

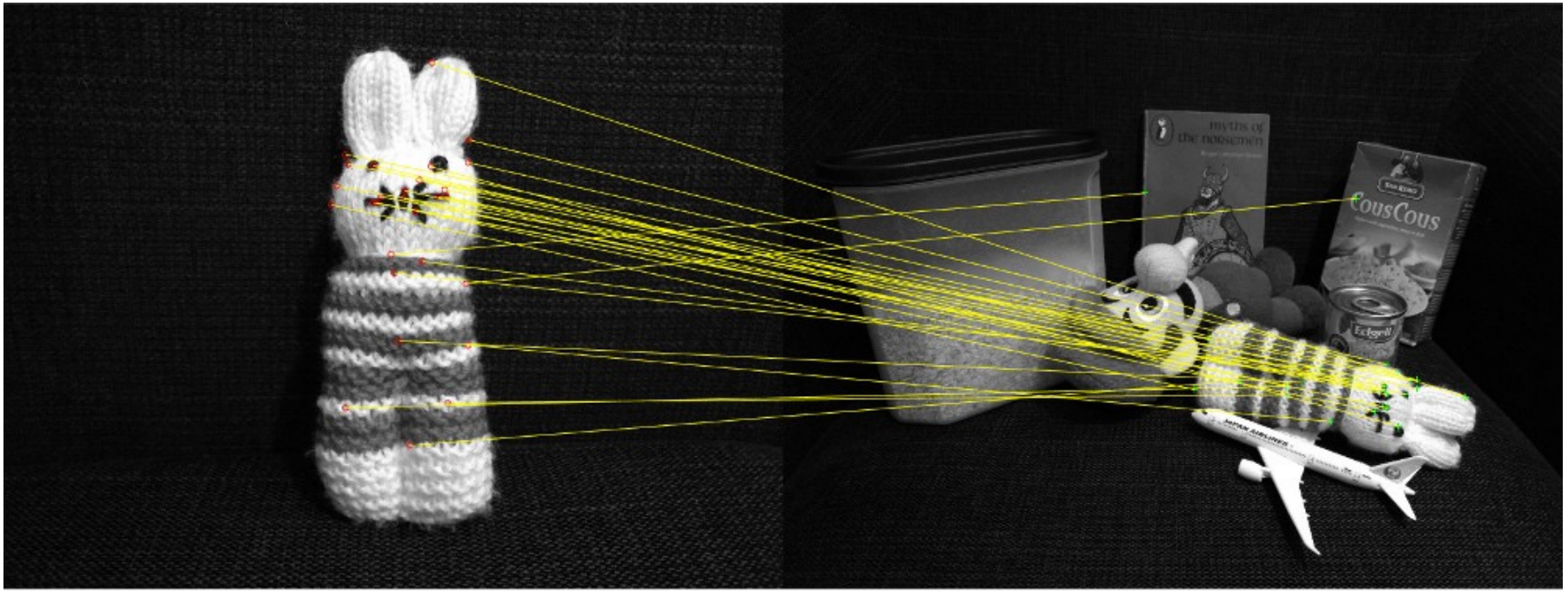
- The resulting descriptor/feature vector has been shown to be very robust to noise and viewing angle and is distinct to the local image intensity information associated with the patch
- SIFT and SURF features hence become useful for object detection: we can detect a series of interest points in both a source image and template image and look for our object by finding matching sets of correspondences between images

Scene Image

SIFT Features

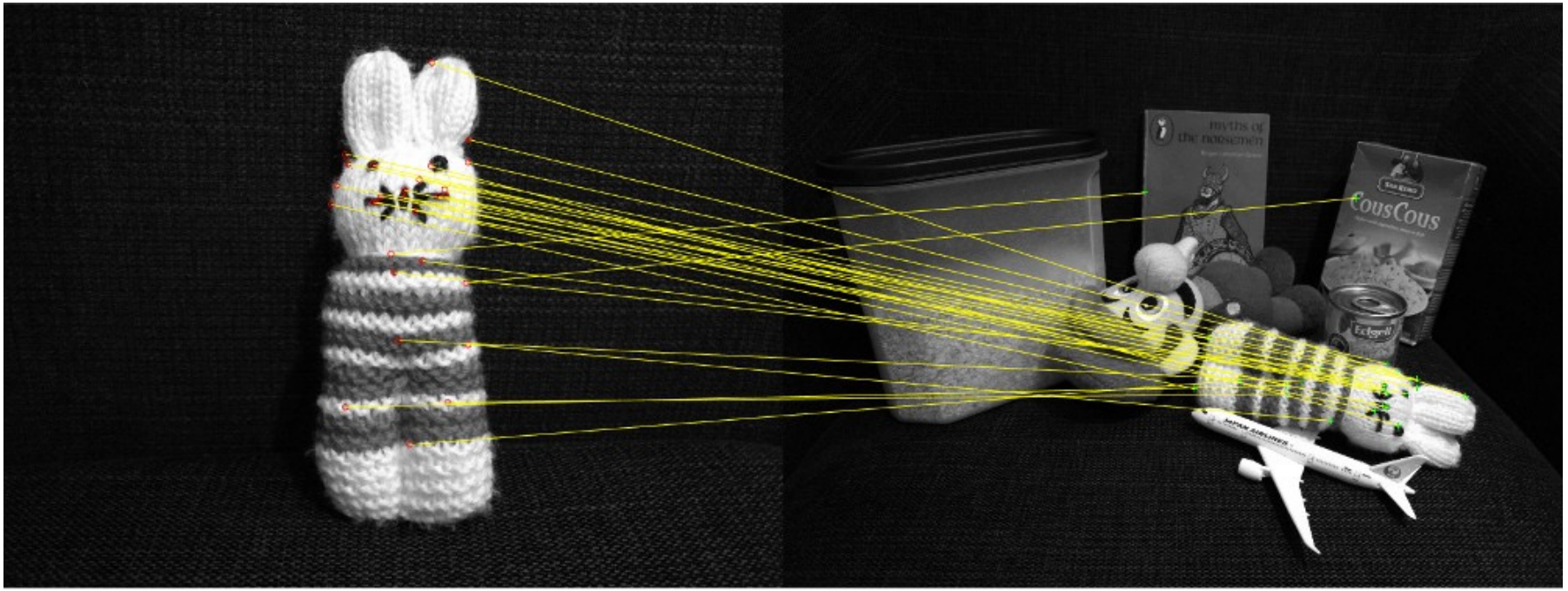
Use of SIFT in Object Detection

1) Detect SIFT points in both a template image and a scene image
2) Find correspondences between features:
    1) for each image in the template, compute the Sum of Squared Difference (SSD) between the template feature's 128D descriptor and each of the descriptors in the set of scene features
    2) Only keep matches below a threshold and reject pairs for which a template feature matched well with more than one source feature



Cleaning up incorrect associations

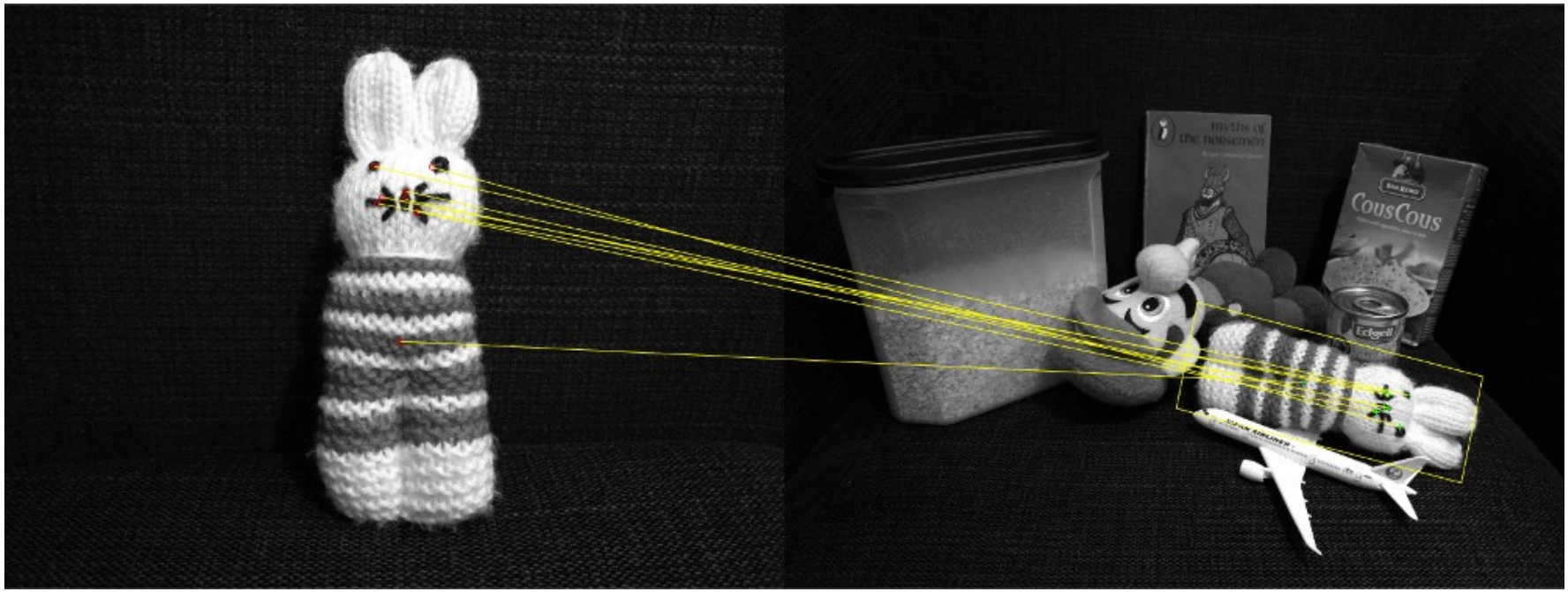
- For large numbers of features, some associations are incorrect (outliers)
- One effective means of detecting and rejecting outliers is via a RANSAC model fitting strategy that attempts to use corresponding pairs to compute the relative alignment of the object between images
- We will explore how RANSAC can be used in this way after we begin to discuss projective geometry (and how sets of corresponding points in images behave owing to geometry)



SIFT Feature Correspondances

Cleaning up incorrect associations

- For large numbers of features, some associations are incorrect (outliers)
- One effective means of detecting and rejecting outliers is via a RANSAC model fitting strategy that attempts to use corresponding pairs to compute the relative alignment of the object between images
- We will explore how RANSAC can be used in this way after we begin to discuss projective geometry (and how sets of corresponding points in images behave owing to geometry)



Objected detected using RANSAC computed associations

Further Reading and Next Week

- References:
  - D. A. Forsyth and J. Ponce, “Computer Vision - A Modern Approach”, Prentice Hall, 2002
  - R. Szeliski, “Computer Vision: Algorithms and Applications”, Springer, 2010

- Next Week:
  - Introduction to projective geometry and stereo vision
  - More on interest points and their use in stereo vision when combined with model fitting strategies (i.e. RANSAC)