

CSC 422/522

Computer Vision and

Pattern Recognition

Feature Detection (Edges and Lines)

2026 Spring



SOUTH DAKOTA
STATE UNIVERSITY

Today

- Features vs. Pixels
- How to detect edges?
- How to detect lines?
- How to detect other simple shapes?



SOUTH DAKOTA
STATE UNIVERSITY



SOUTH DAKOTA
STATE UNIVERSITY



Why Features Matter in Computer Vision?

Image source: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

Let's Play a Game: Jigsaw Puzzle!

- Link:
<https://onlinejigsawpuzzles.net/puzzle.php?image=images/puzzle/Spring-in-farmland-Sweden-old-bar--in-rural-surroundings.jpg#>
- Note: you can change the number of pieces
- Think: How do you solve the puzzle? What visual cues help you place pieces?



SOUTH DAKOTA
STATE UNIVERSITY

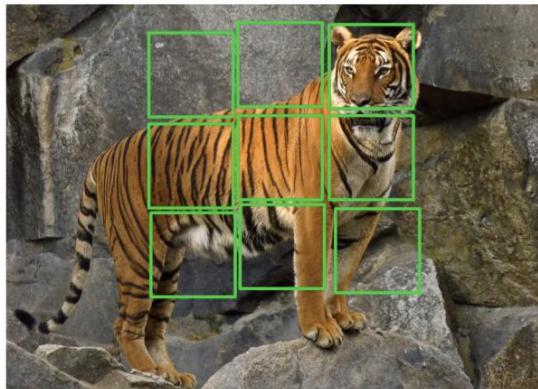
Jigsaw Puzzles in Computer Vision Research

- In a jigsaw puzzle, we reconstruct a global image from local pieces.
- During the process, we may consider edge continuity, texture similarity, color consistency, etc.
- Computer vision aims to automate the same reasoning process.
- Example research papers: Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles (ECCV 2016, paper link: <https://arxiv.org/abs/1603.09246>)



Screenshot for
<https://onlinejigsawpuzzles.net/puzzle.php?image=images/puzzle/Spring-in-farmland-Sweden-old-bar--in-rural-surroundings.jpg#>

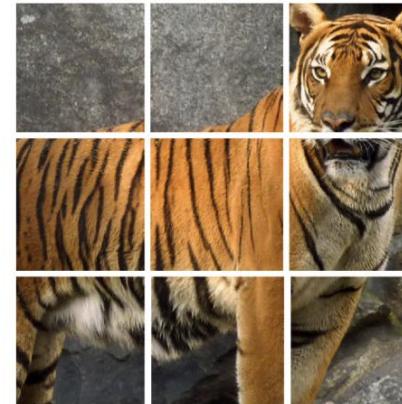
Jigsaw Puzzles in Computer Vision Research



(a)



(b)



(c)

Fig. 1: Learning image representations by solving Jigsaw puzzles. (a) The image from which the tiles (marked with green lines) are extracted. (b) A puzzle obtained by shuffling the tiles. Some tiles might be directly identifiable as object parts, but others are ambiguous (*e.g.*, have similar patterns) and their identification is much more reliable when all tiles are jointly evaluated. In contrast, with reference to (c), determining the relative position between the central tile and the top two tiles from the left can be very challenging [10].

Example research papers:
Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles (ECCV 2016, paper link: <https://arxiv.org/abs/1603.09246>)

What is a Feature in Computer Vision?

- A feature in computer vision is an identifiable part of an image or video that conveys meaningful information for tasks like object detection, tracking, or classification.
- Features are often distinct patterns, such as edges, corners, textures, or specific shapes, that algorithms use to understand and analyze visual data.

Texts adapted from <https://milvus.io/ai-quick-reference/what-is-a-feature-in-computer-vision>



Screenshot for
<https://onlinejigsawpuzzles.net/puzzle.php?image=images/puzzle/Spring-in-farmland-Sweden-old-bar-in-rural-surroundings.jpg#>

Why Not Just Use Pixels?

- Pixels are sensitive to noise, lighting variations, viewpoint changes, etc.
- Therefore, pixels are not stable representations for robust computer vision tasks.

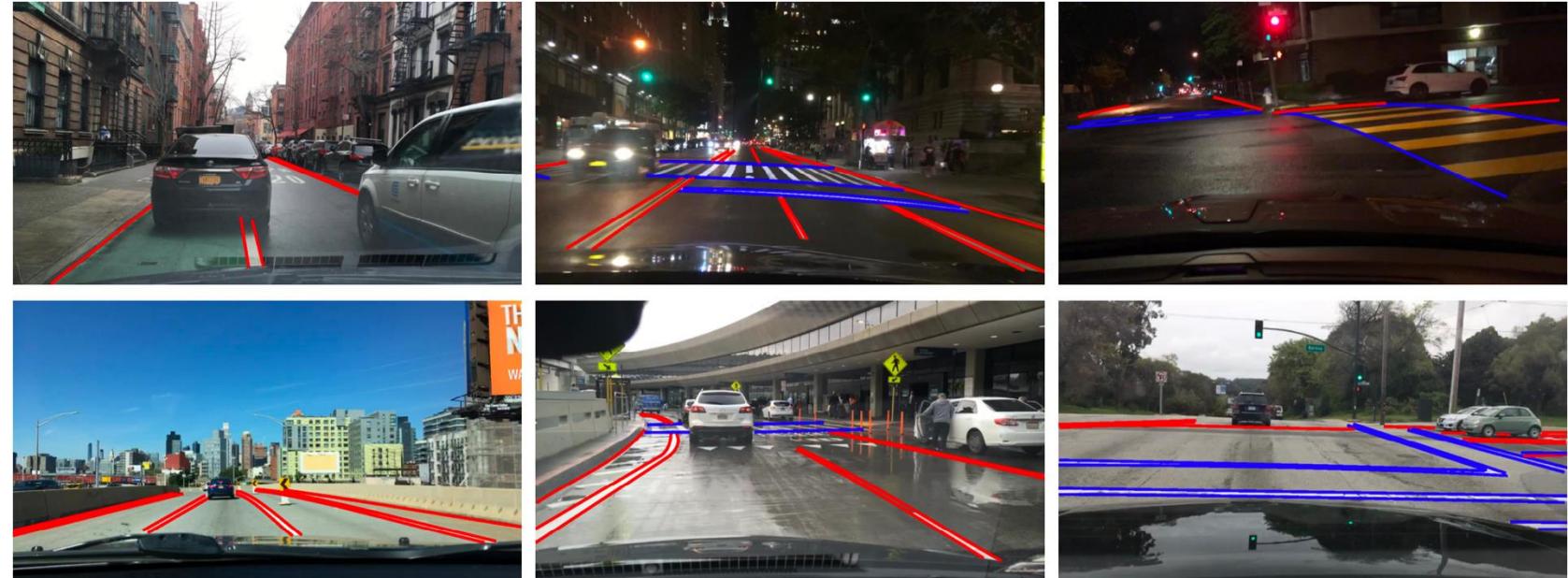


Image source: BDD100K: A Large-scale Diverse Driving Video Database
<https://bair.berkeley.edu/blog/2018/05/30/bdd/>



SOUTH DAKOTA
STATE UNIVERSITY

Classical (Hand-Crafted) Features

- Traditional feature extraction methods rely on mathematical techniques to identify and describe these key points.
- Algorithms like SIFT (Scale-Invariant Feature Transform) detect stable features by analyzing gradients, corners, or blobs in an image.

Texts adapted from <https://milvus.io/ai-quick-reference/what-is-a-feature-in-computer-vision>



Image found at
<https://ics.uci.edu/~majumder/VC/211HW3/vlfeat/doc/overview/sift.html>

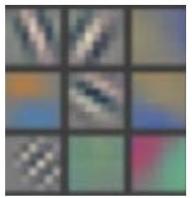
Learned Features

- Modern approaches, particularly deep learning, automate feature extraction using convolutional neural networks (CNNs).
- In CNNs, layers learn hierarchical features directly from data.
- Early layers detect simple patterns like edges or color gradients, while deeper layers combine these to recognize complex shapes or objects.

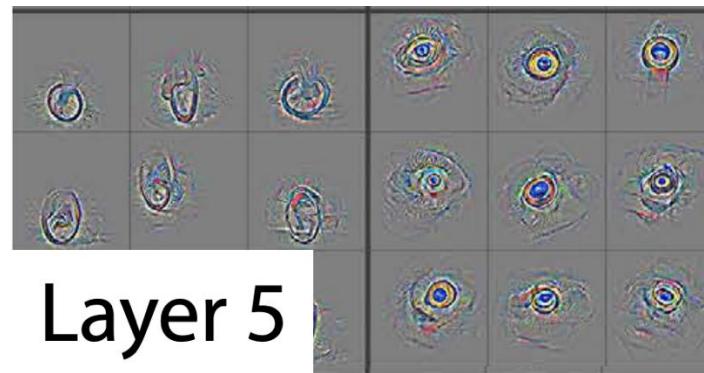
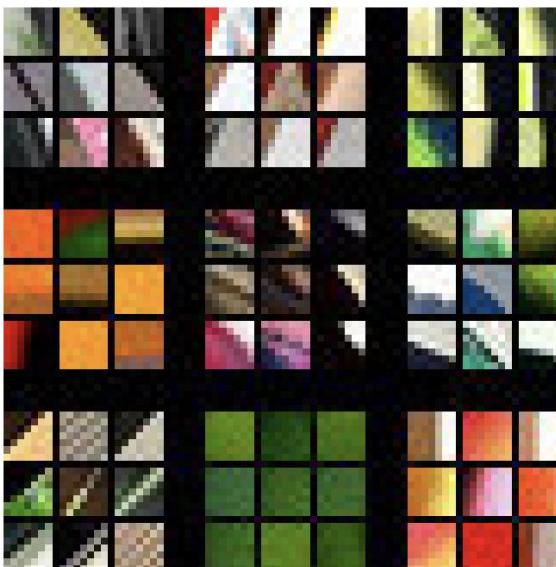


SOUTH DAKOTA
STATE UNIVERSITY

“Visualizing and Understanding Convolutional Networks” (ECCV 2014)



Layer 1



Layer 5



Check full Figure 2 in the original paper.
You can read the paper in this link:
<https://arxiv.org/pdf/1311.2901>



SOUTH DAKOTA
STATE UNIVERSITY



SOUTH DAKOTA
STATE UNIVERSITY

Edge Detection

Credit: slides of this section are adapted from CSC 492/592, Spring 2024

Image source: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

Original Image

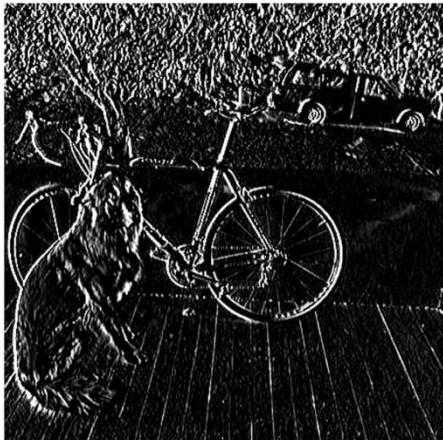


Edge Image



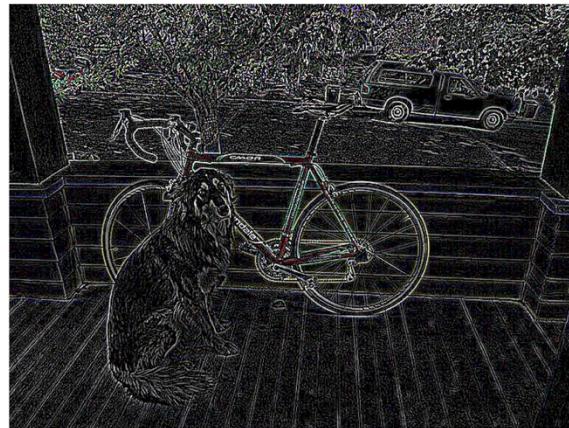
Recap: Edge Detection

Prewitt			Sobel		
$\frac{\partial I}{\partial x}$	-1 0 1 -1 0 1 -1 0 1		$\frac{\partial I}{\partial y}$	1 1 1 0 0 0 -1 -1 -1	



Laplacian of Gaussian

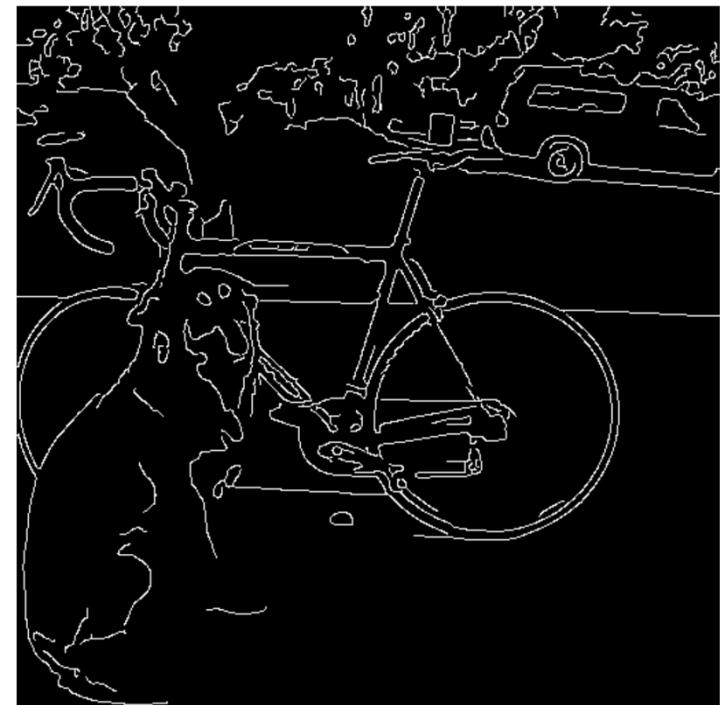
$$\nabla^2 \approx \frac{1}{6\epsilon^2} \begin{array}{|ccc|} \hline & 1 & 4 & 1 \\ \hline 4 & -20 & 4 \\ \hline & 1 & 1 & 1 \\ \hline \end{array}$$



Gradient	Laplacian
Provides location , magnitude , and direction of the edge	Provides only location of the edge
Detection using Maxima Thresholding	Detection based on Zero-crossing
<ul style="list-style-type: none">- Non-linear operation- Requires two convolutions	<ul style="list-style-type: none">- Linear operation- Requires only one convolution

Canny Edge Detection

- The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.
- It was developed by John F. Canny in 1986.
- Among the edge detection methods developed so far, Canny's algorithm is one of the most strictly defined methods that provides good and reliable detection.



Reference: https://en.wikipedia.org/wiki/Canny_edge_detector

Canny Edge Detection: Algorithm

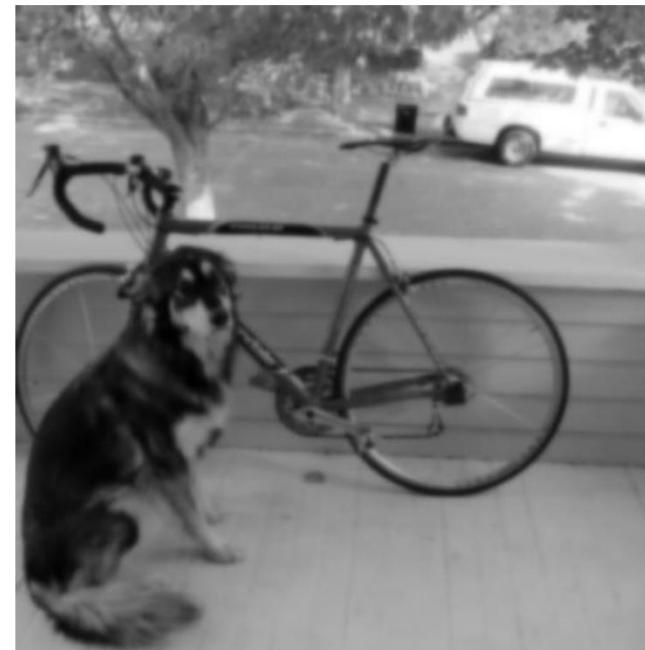
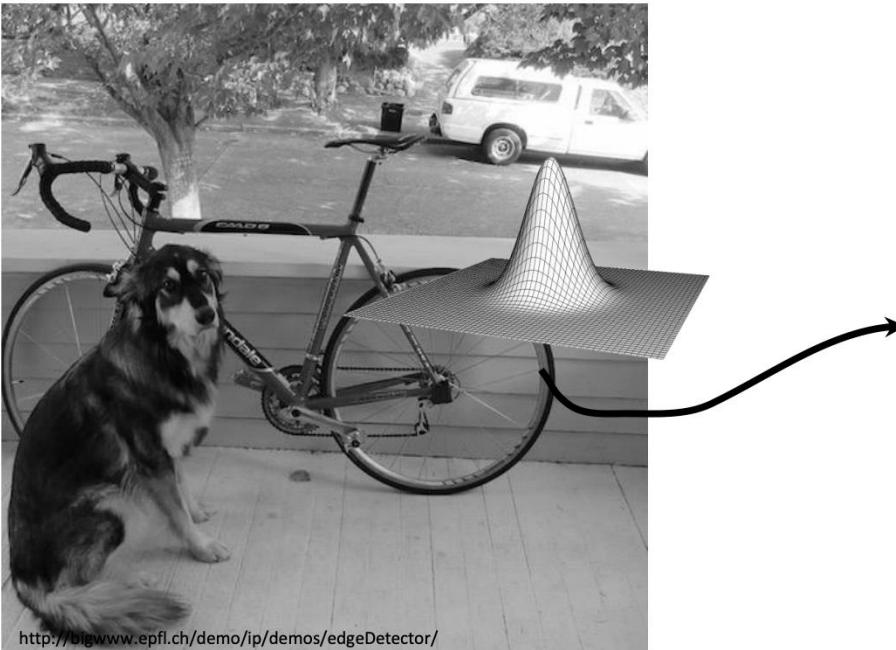
1. Smooth image
2. Calculate gradient direction and magnitude
3. Perform non-maximum suppression
4. Threshold the edges
5. Link the edges



SOUTH DAKOTA
STATE UNIVERSITY

Step 1: Smooth image

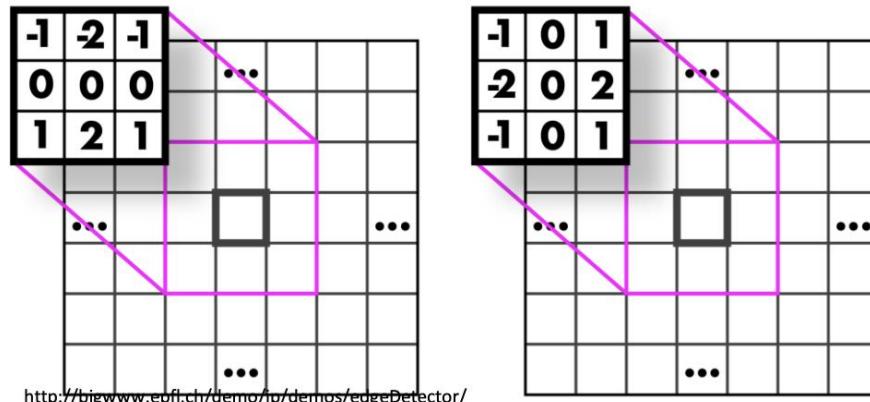
Apply Gaussian filter to smooth the image to remove the noise.



SOUTH DAKOTA
STATE UNIVERSITY

Step 2: Calculate gradient direction and magnitude

	Prewitt	Sobel																		
$\frac{\partial I}{\partial x}$	<table border="1"><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr></table>	-1	0	1	-1	0	1	-1	0	1	<table border="1"><tr><td>-1</td><td>0</td><td>1</td></tr><tr><td>-2</td><td>0</td><td>2</td></tr><tr><td>-1</td><td>0</td><td>1</td></tr></table>	-1	0	1	-2	0	2	-1	0	1
-1	0	1																		
-1	0	1																		
-1	0	1																		
-1	0	1																		
-2	0	2																		
-1	0	1																		
$\frac{\partial I}{\partial y}$	<table border="1"><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-1</td><td>-1</td></tr></table>	1	1	1	0	0	0	-1	-1	-1	<table border="1"><tr><td>1</td><td>2</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>-1</td><td>-2</td><td>-1</td></tr></table>	1	2	1	0	0	0	-1	-2	-1
1	1	1																		
0	0	0																		
-1	-1	-1																		
1	2	1																		
0	0	0																		
-1	-2	-1																		



SOUTH DAKOTA
STATE UNIVERSITY

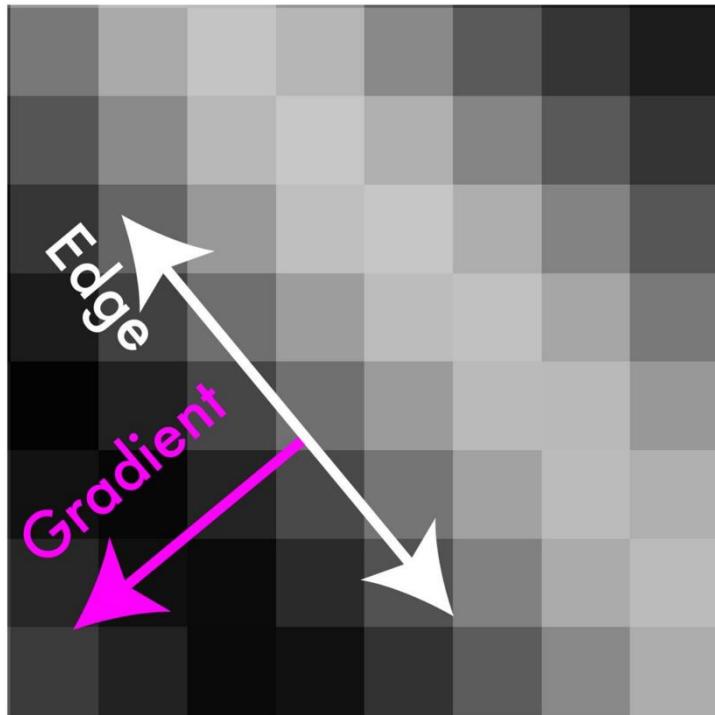
Step 3: Perform non-maximum suppression

- **Why?** Want single pixel edges, not thick blurry lines
- **Idea.** Check nearby pixels and keep the high response
- **How?** Go through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the **edge directions**



SOUTH DAKOTA
STATE UNIVERSITY

Step 3: Perform non-maximum suppression

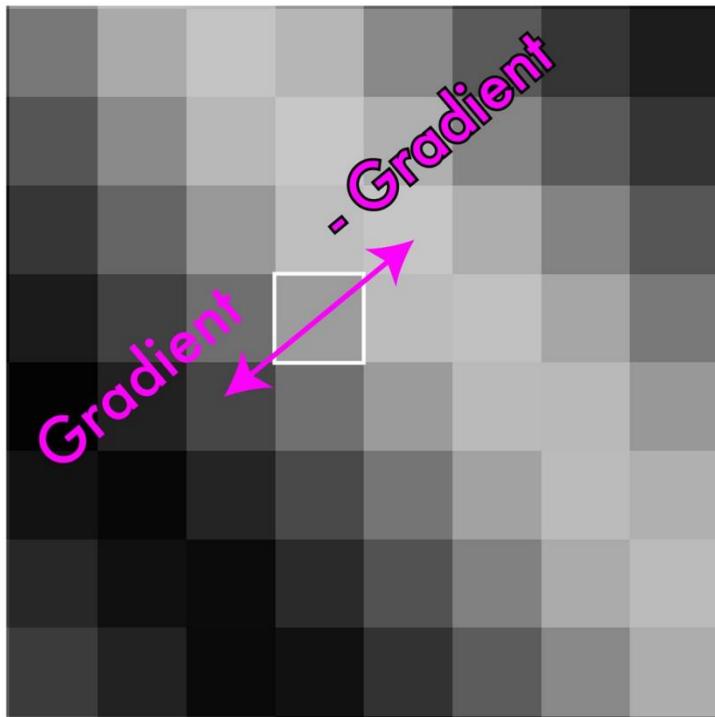


Go through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions.



SOUTH DAKOTA
STATE UNIVERSITY

Step 3: Perform non-maximum suppression



Example pixel 1:

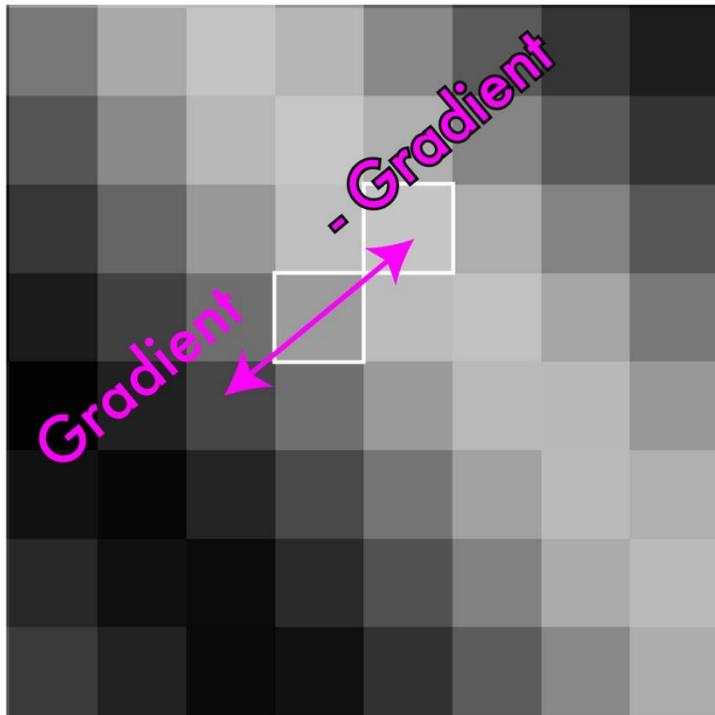
(highlighted in white box)

compare its gradient magnitude with
the magnitudes of its neighboring
pixels along the gradient direction



SOUTH DAKOTA
STATE UNIVERSITY

Step 3: Perform non-maximum suppression



Example pixel 1:

(highlighted in left-bottom white box)

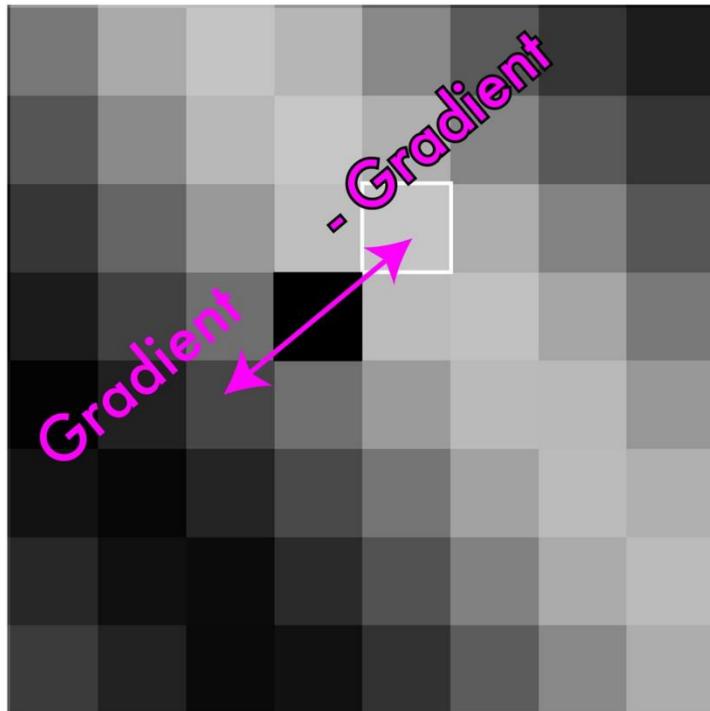
compare its gradient magnitude with
the magnitudes of its neighboring
pixels along the gradient direction

→ We find a neighboring pixel with
larger gradient magnitude



SOUTH DAKOTA
STATE UNIVERSITY

Step 3: Perform non-maximum suppression



Example pixel 1:

(highlighted in left-bottom white box)

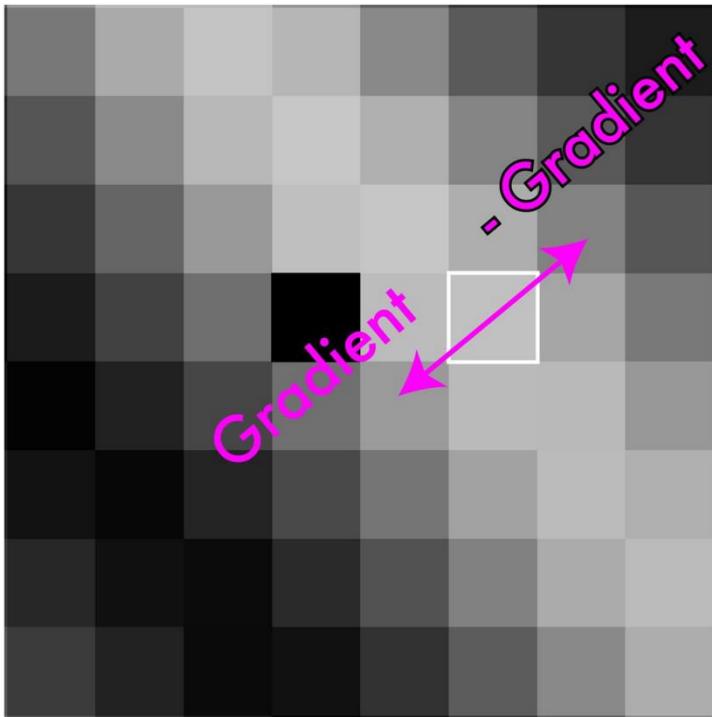
compare its gradient magnitude with
the magnitudes of its neighboring
pixels along the gradient direction

- We find a neighboring pixel with
larger gradient magnitude
- It is suppressed ☹



SOUTH DAKOTA
STATE UNIVERSITY

Step 3: Perform non-maximum suppression



Example pixel 2:

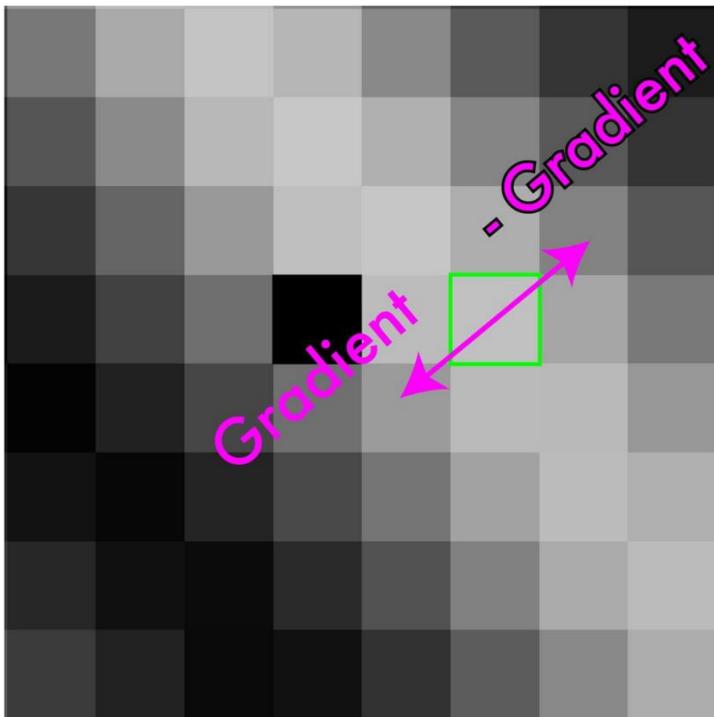
(highlighted in white box)

compare its gradient magnitude with
the magnitudes of its neighboring
pixels along the gradient direction



SOUTH DAKOTA
STATE UNIVERSITY

Step 3: Perform non-maximum suppression



Example pixel 2:

(highlighted in green box now)

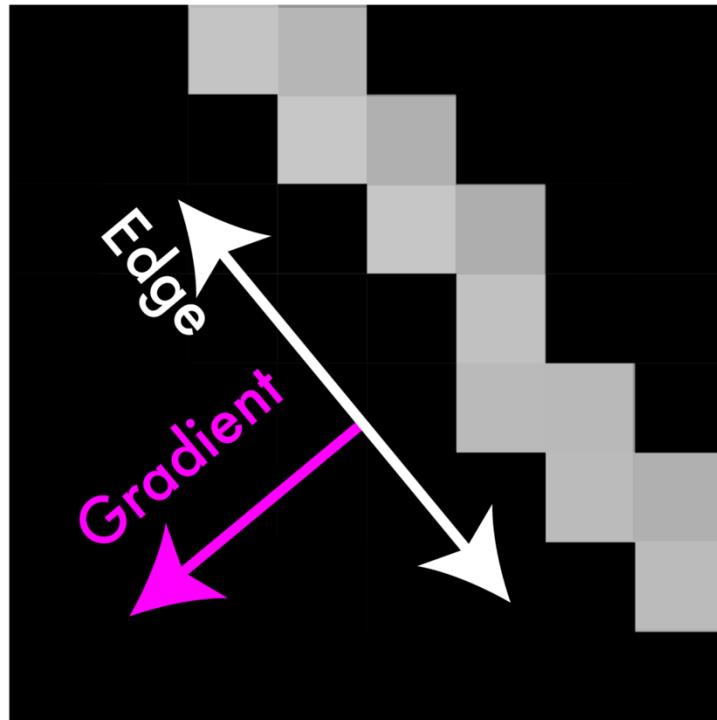
compare its gradient magnitude with
the magnitudes of its neighboring
pixels along the gradient direction

- Greater than that of both of its
neighboring pixels
- It is kept ☺



SOUTH DAKOTA
STATE UNIVERSITY

Step 3: Perform non-maximum suppression

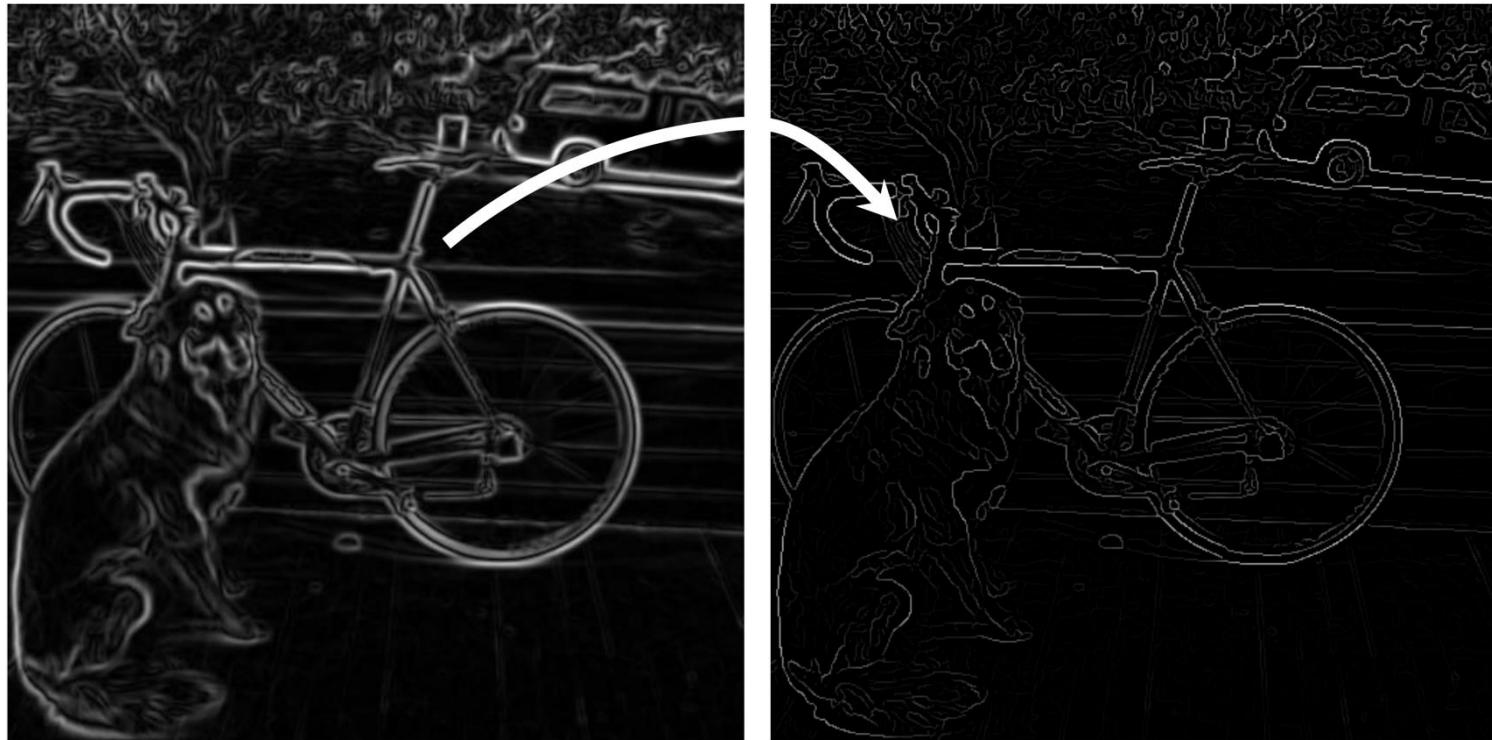


This process is performed for all pixels in the image, resulting in a thinned set of edge pixels where only local maxima **along the edges** remain.



SOUTH DAKOTA
STATE UNIVERSITY

Step 3: Perform non-maximum suppression



SOUTH DAKOTA
STATE UNIVERSITY

Step 4: Threshold the edges

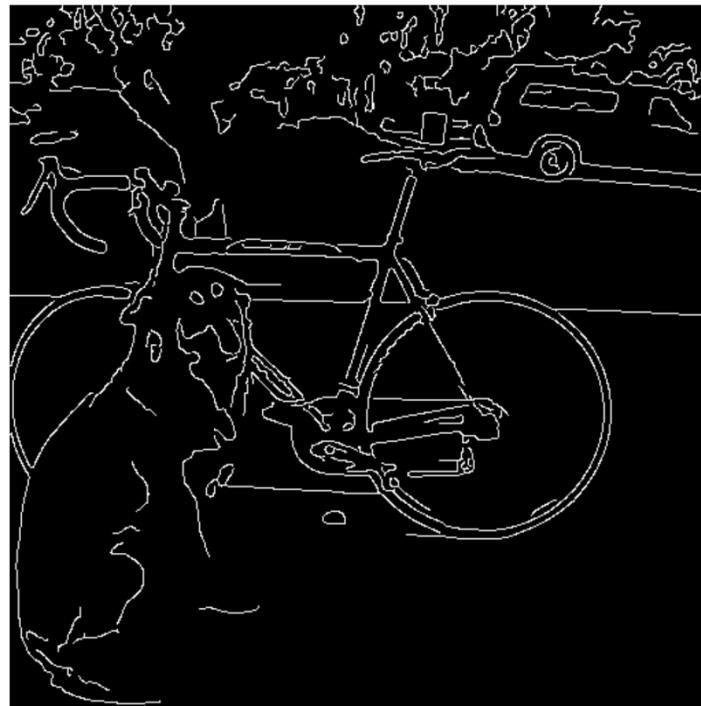
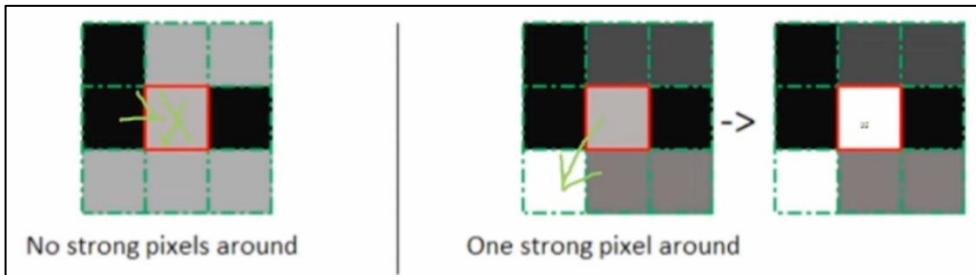
- **Why?** Still some noise
- **Idea.** Thresholding edges
- **How?**
 - Categorize edges into **strong**, **weak**, **non-relevant**
 - 2 thresholds T and t , 3 cases:
 - . $T <$: strong edge
 - . $t < \text{edge} < T$: weak edge
 - . $\text{edge} < t$: non-relevant



SOUTH DAKOTA
STATE UNIVERSITY

Step 5: Link the edges

- **Idea.**
 - Strong edges are edges!
 - Weak edges are edges if and only if they are connected to strong edges
- **How?**
 - Look in some neighborhood (usually 8 closet)



SOUTH DAKOTA
STATE UNIVERSITY

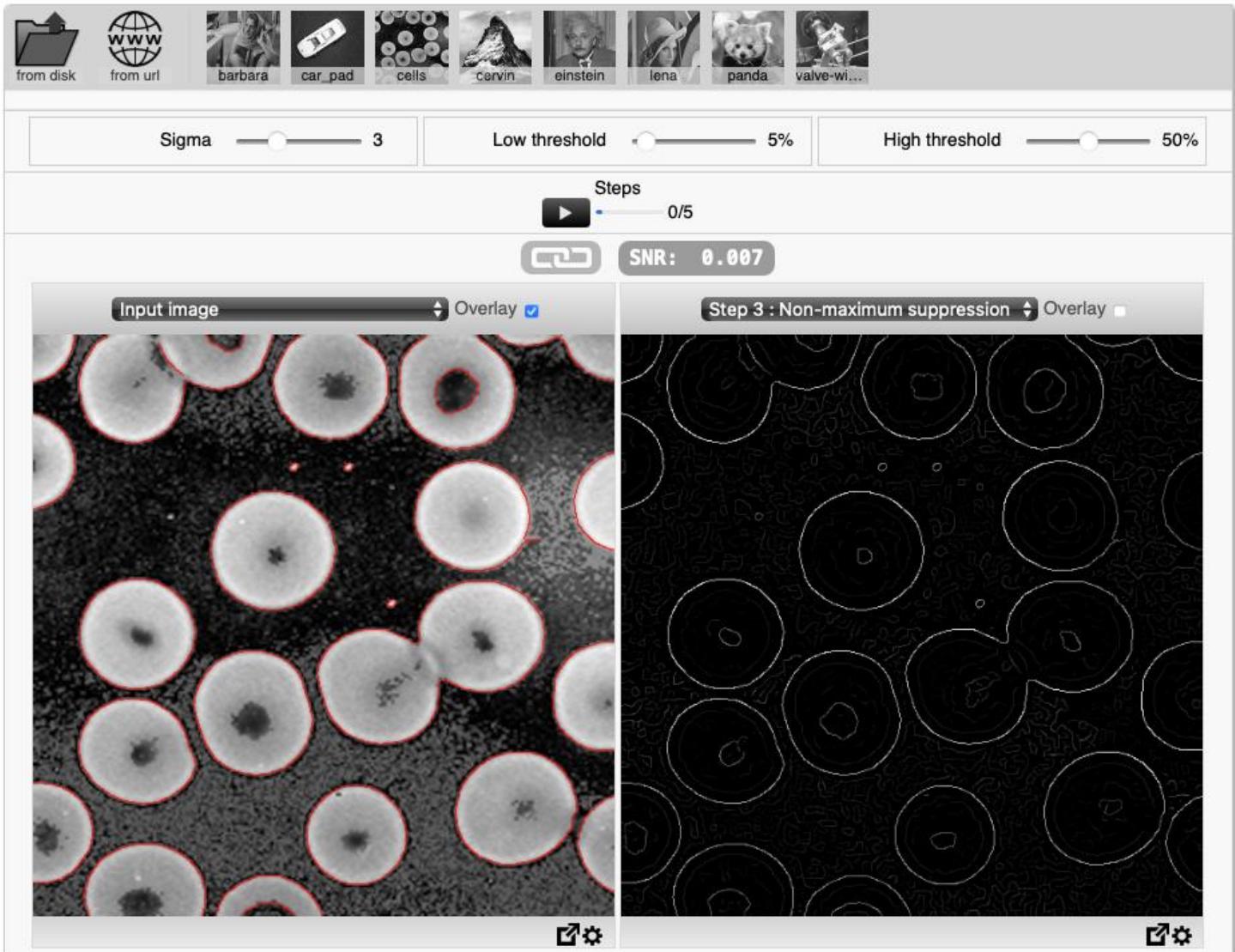
Demo

Let's try:

<https://bigwww.epfl.ch/demo/ip/demos/edgeDetector/>

Think: how the hyperparameters affect the results?

In OpenCV: simply call “cv2.Canny” function
(reference:
https://docs.opencv.org/4.x/d22/tutorial_py_canny.html)



SOUTH DAKOTA
STATE UNIVERSITY

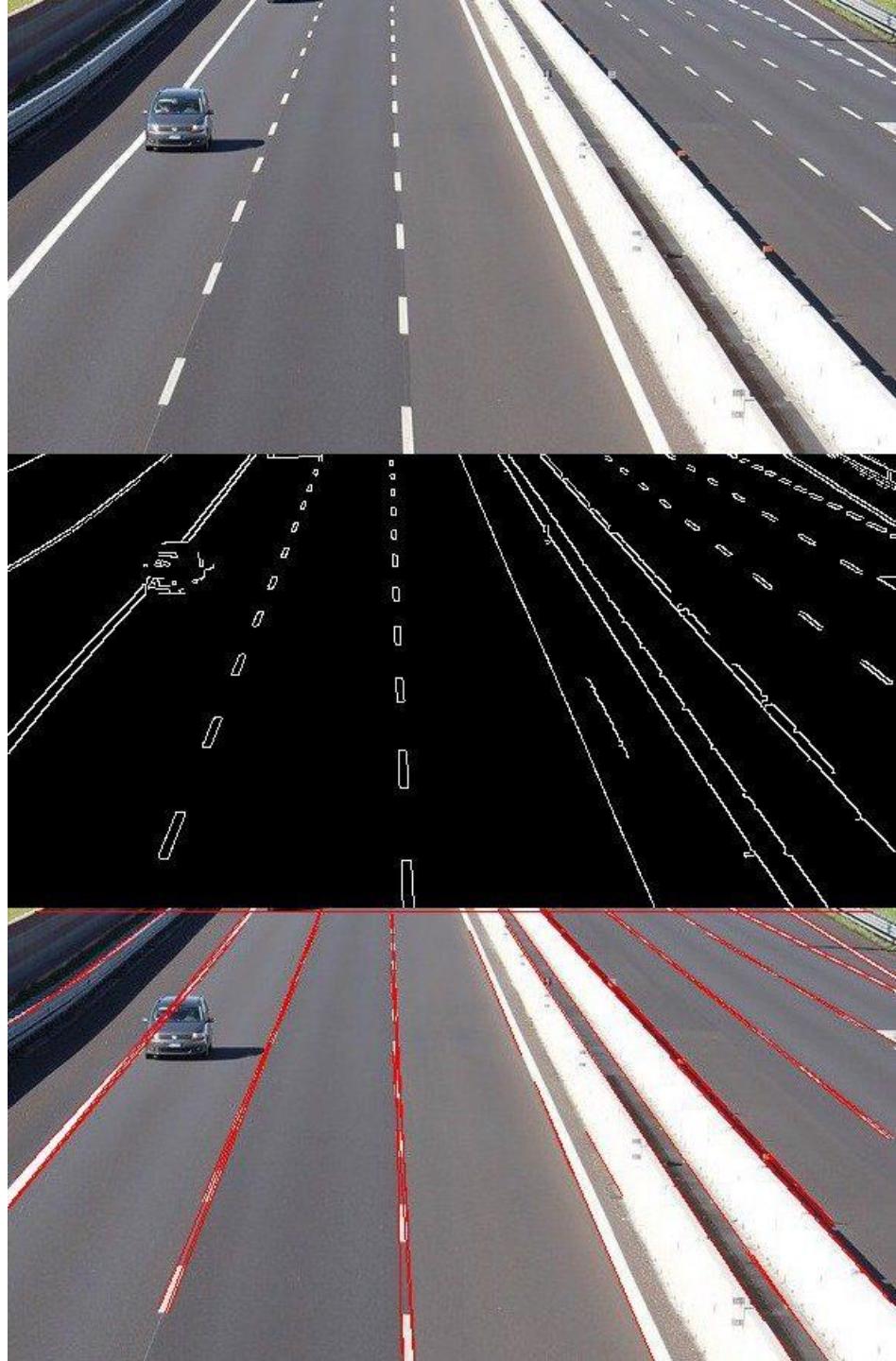


SOUTH DAKOTA
STATE UNIVERSITY

Finding Lines via Hough Transform

Credit: slides of this section are adapted from CSC 492/592, Spring 2024

Image source:
<https://learnopencv.com/hough-transform-with-opencv-c-python/>



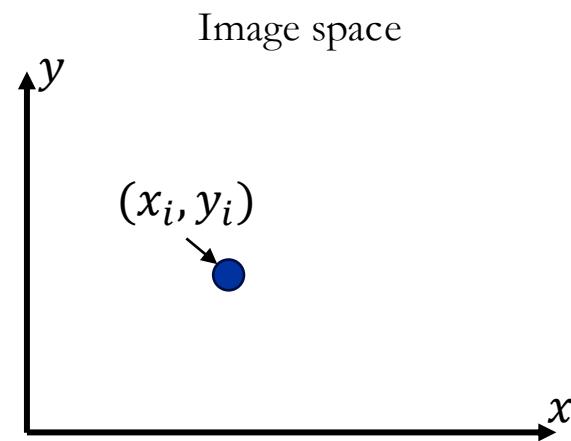
Finding lines: Hough Transform

- ***Problems.***
 - Extraneous Data: Which points to fit to
 - Incomplete Data: Only part of the model is visible
 - Noise
- ***Solution.*** Hough Transform

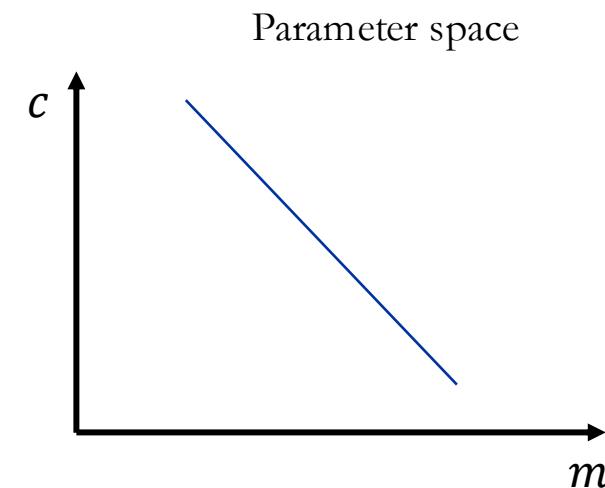


SOUTH DAKOTA
STATE UNIVERSITY

Example (Part 1)



$$y_i = m x_i + c$$

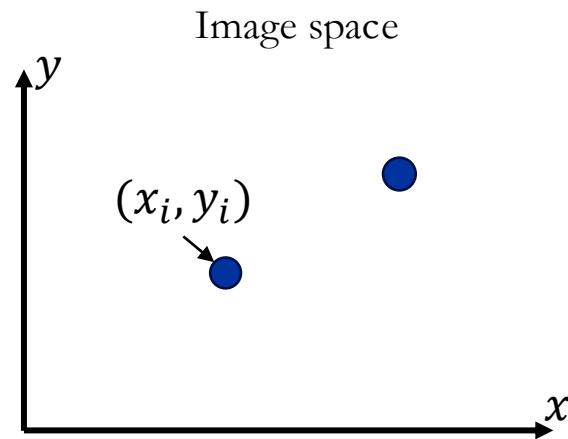


$$c = -m x_i + y_i$$

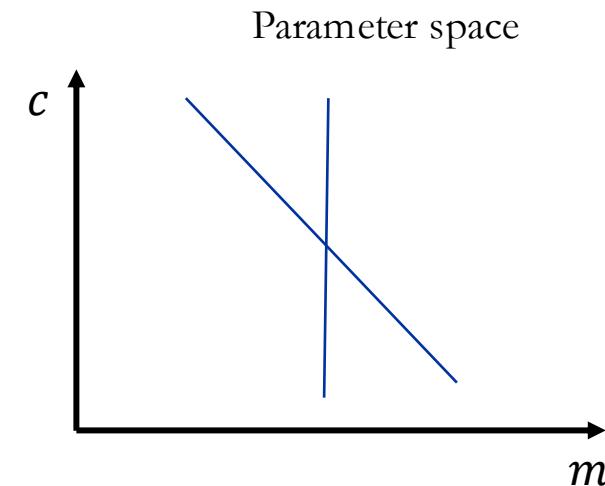


SOUTH DAKOTA
STATE UNIVERSITY

Example (Part 2)



$$y_i = m x_i + c$$

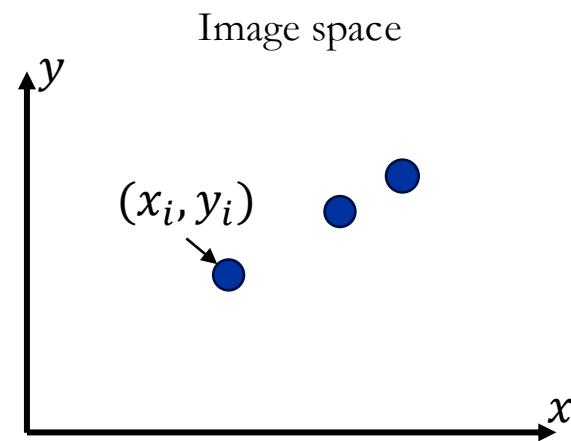


$$c = -m x_i + y_i$$

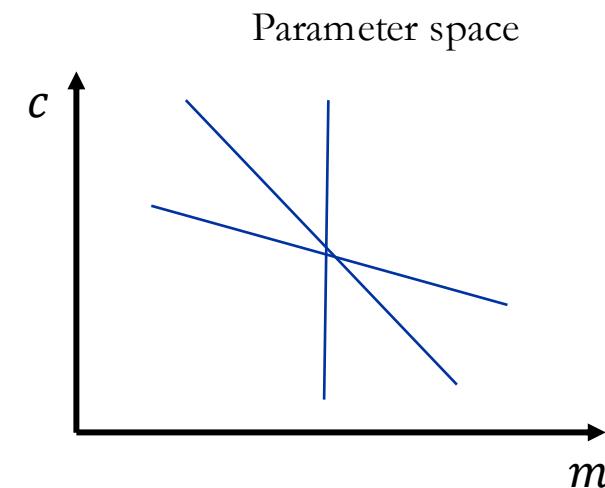


SOUTH DAKOTA
STATE UNIVERSITY

Example (Part 3)



$$y_i = mx_i + c$$

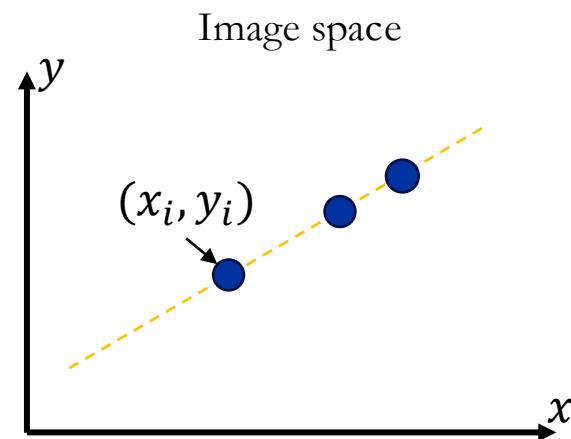


$$c = -mx_i + y_i$$

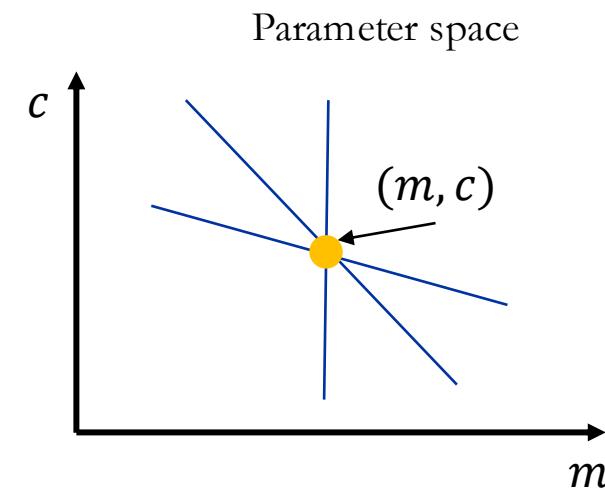


SOUTH DAKOTA
STATE UNIVERSITY

Example (Part 4)



$$y_i = m x_i + c$$

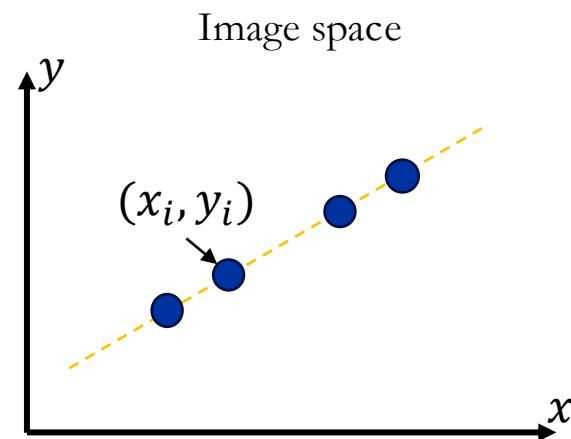


$$c = -m x_i + y_i$$

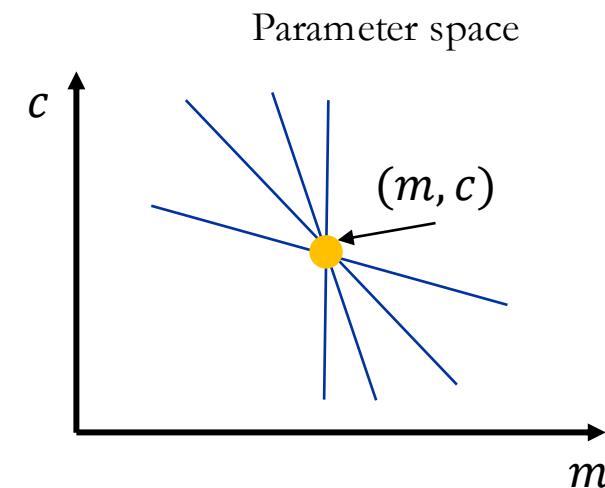


SOUTH DAKOTA
STATE UNIVERSITY

Example (Part 5)



$$y_i = m x_i + c$$

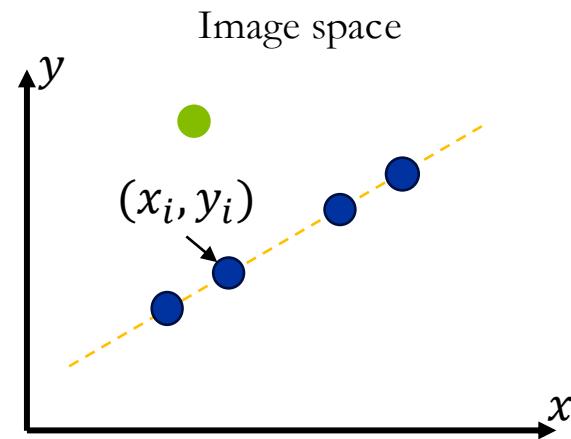


$$c = -m x_i + y_i$$

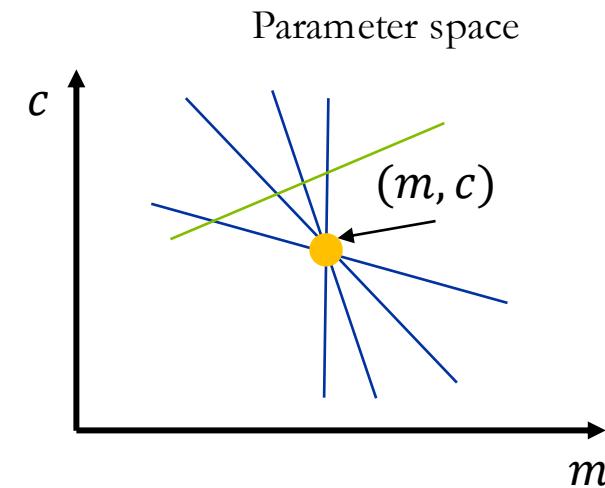


SOUTH DAKOTA
STATE UNIVERSITY

Example (Part 6)



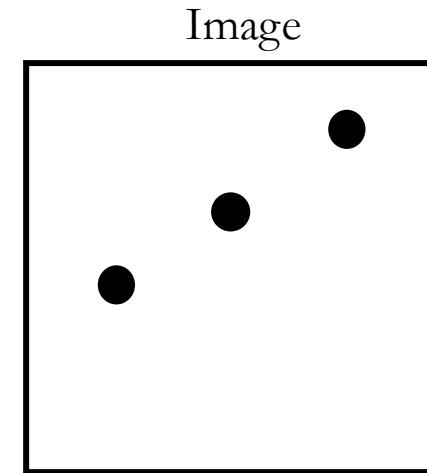
$$y_i = m x_i + c$$



$$c = -m x_i + y_i$$



Now summarize the line detection algorithm!

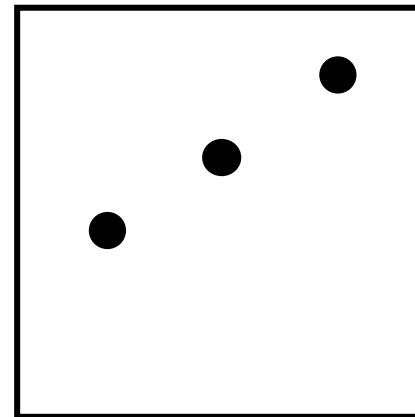


SOUTH DAKOTA
STATE UNIVERSITY

Line Detection Algorithm

- Step 1. Quantize parameter space (m, c)
- Step 2. Create **accumulator array** $A(m, c)$

Image

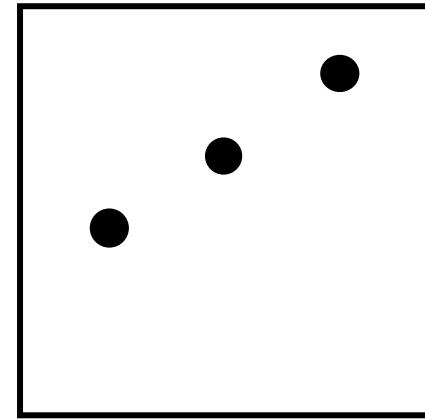


$A(m, c)$



SOUTH DAKOTA
STATE UNIVERSITY

Image



Line Detection Algorithm

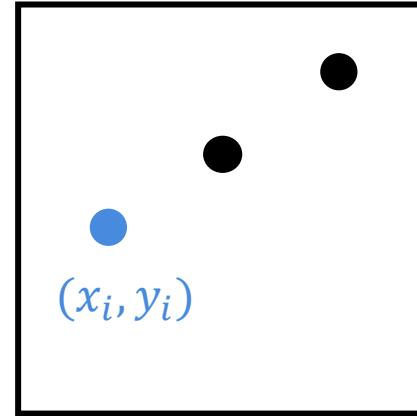
- Step 1. Quantize parameter space (m, c)
- Step 2. Create **accumulator array** $A(m, c)$
- Step 3. Set $A(m, c) = 0$ for all (m, c)

$A(m, c)$				
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0



SOUTH DAKOTA
STATE UNIVERSITY

Image



Line Detection Algorithm

- Step 1. Quantize parameter space (m, c)
- Step 2. Create **accumulator array** $A(m, c)$
- Step 3. Set $A(m, c) = 0$ for all (m, c)
- Step 4. For each edge point (x_i, y_i) ,

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$

$A(m, c)$				
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

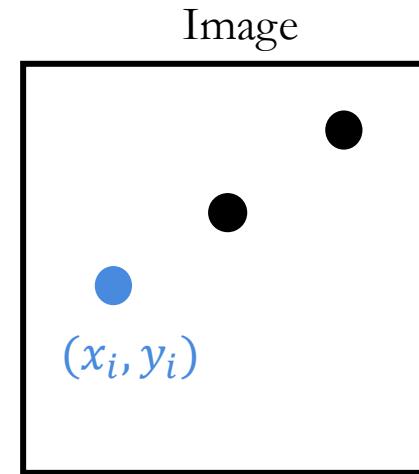


Line Detection Algorithm

- Step 1. Quantize parameter space (m, c)
- Step 2. Create **accumulator array** $A(m, c)$
- Step 3. Set $A(m, c) = 0$ for all (m, c)
- Step 4. For each edge point (x_i, y_i) ,

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$



$A(m, c)$				
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

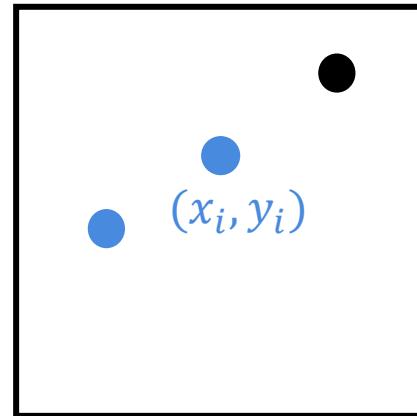
Line Detection Algorithm

- Step 1. Quantize parameter space (m, c)
- Step 2. Create **accumulator array** $A(m, c)$
- Step 3. Set $A(m, c) = 0$ for all (m, c)
- Step 4. For each edge point (x_i, y_i) ,

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$

Image



$A(m, c)$

1	0	0	0	1
0	1	0	1	0
0	0	2	0	0
0	1	0	1	0
1	0	0	0	1



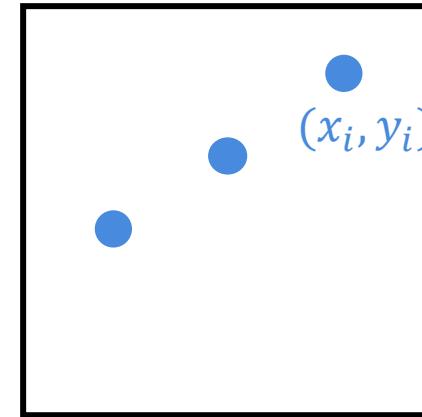
Line Detection Algorithm

- Step 1. Quantize parameter space (m, c)
- Step 2. Create **accumulator array** $A(m, c)$
- Step 3. Set $A(m, c) = 0$ for all (m, c)
- Step 4. For each edge point (x_i, y_i) ,

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$

Image



$A(m, c)$

1	0	0	0	1
0	1	0	1	0
1	1	3	1	1
0	1	0	1	0
1	0	0	0	1



SOUTH DAKOTA
STATE UNIVERSITY

Line Detection Algorithm

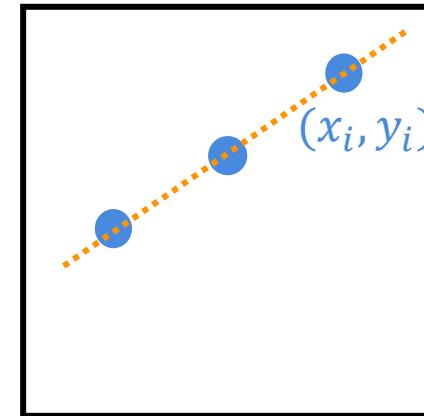
- Step 1. Quantize parameter space (m, c)
- Step 2. Create **accumulator array** $A(m, c)$
- Step 3. Set $A(m, c) = 0$ for all (m, c)
- Step 4. For each edge point (x_i, y_i) ,

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$

- Step 5. Find local maxima in $A(m, c)$

Image

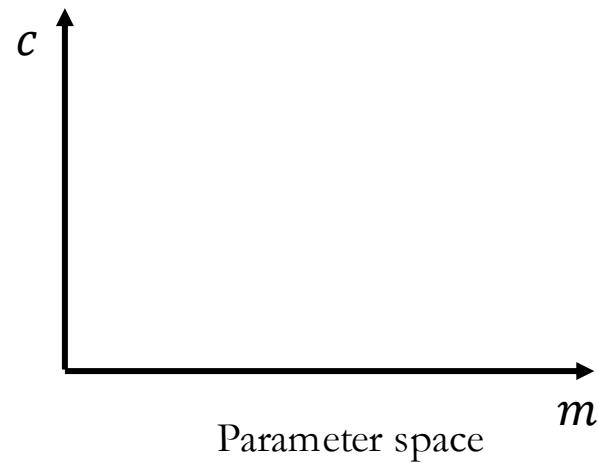
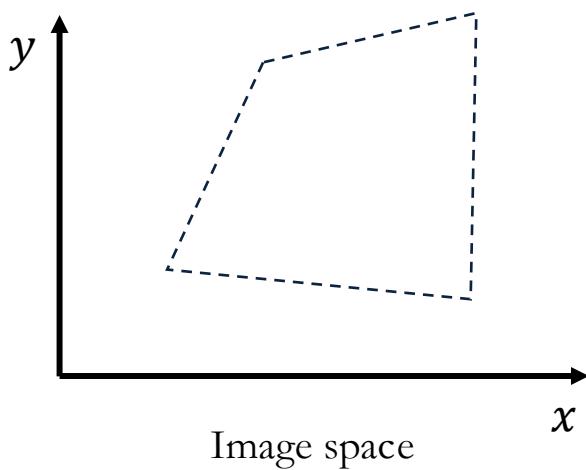


$A(m, c)$

1	0	0	0	1
0	1	0	1	0
1	1	3	1	1
0	1	0	1	0
1	0	0	0	1

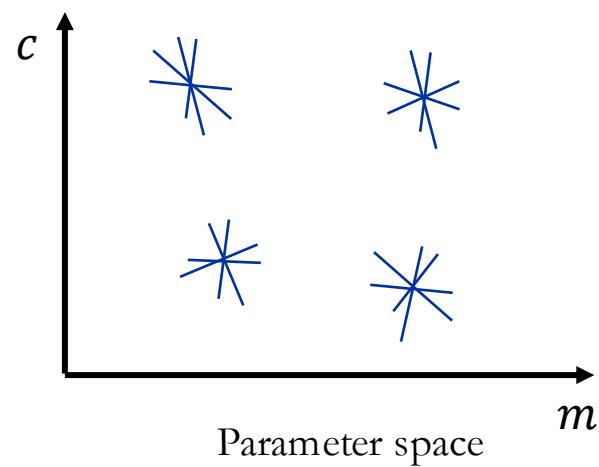
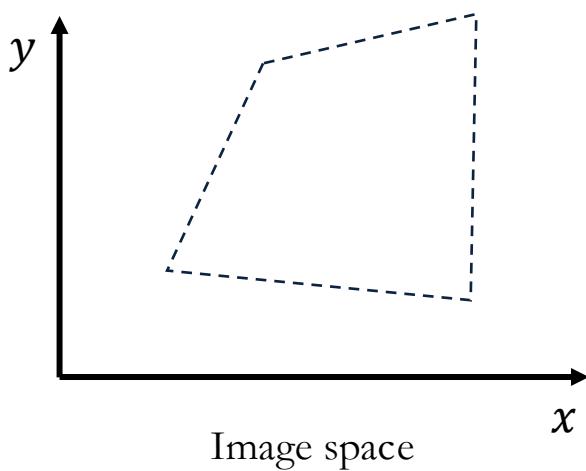


Multiple Lines?



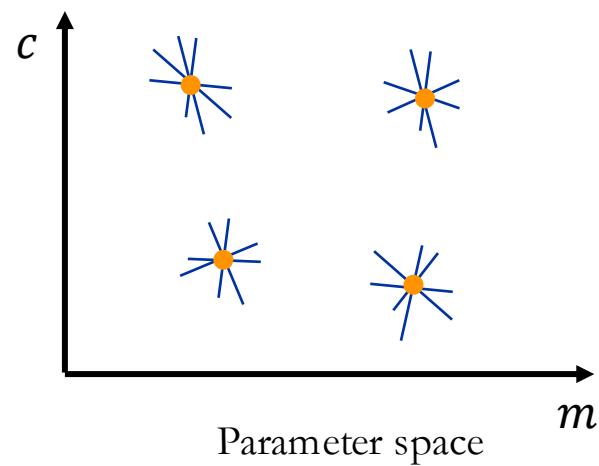
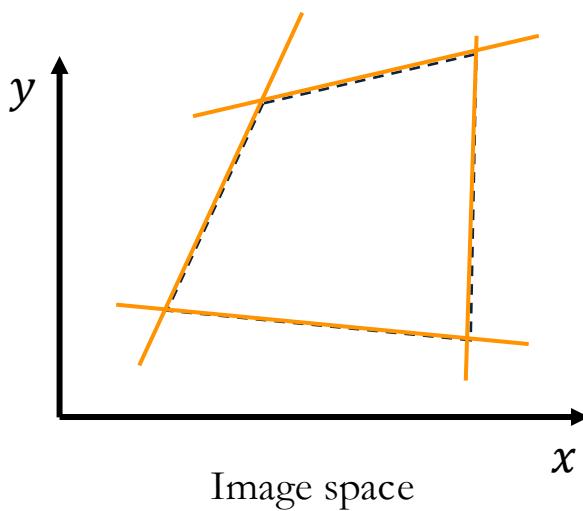
SOUTH DAKOTA
STATE UNIVERSITY

Multiple Lines?



SOUTH DAKOTA
STATE UNIVERSITY

Multiple Lines?



SOUTH DAKOTA
STATE UNIVERSITY

Better Parameterization?

- **Issue:** slope of the line $-\infty \leq m \leq \infty$
 - Large accumulator
 - More memory and computation
- **Solution:** use $x\sin\theta - y\cos\theta + \rho = 0$
 - Orientation θ is finite: $0 \leq \theta < \pi$
 - Distance ρ is finite



SOUTH DAKOTA
STATE UNIVERSITY

Better Parameterization?

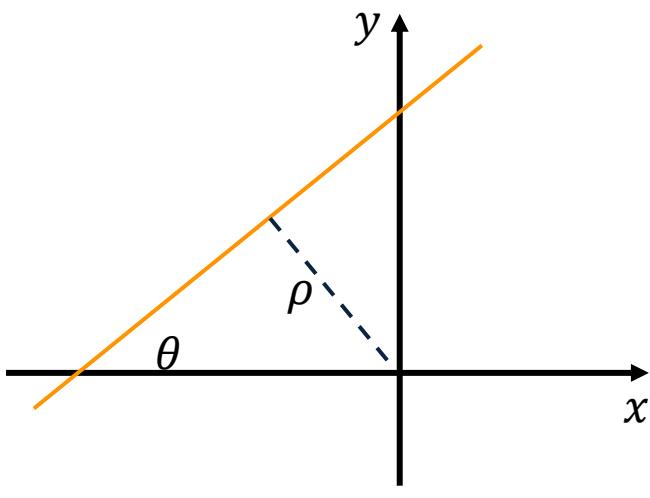
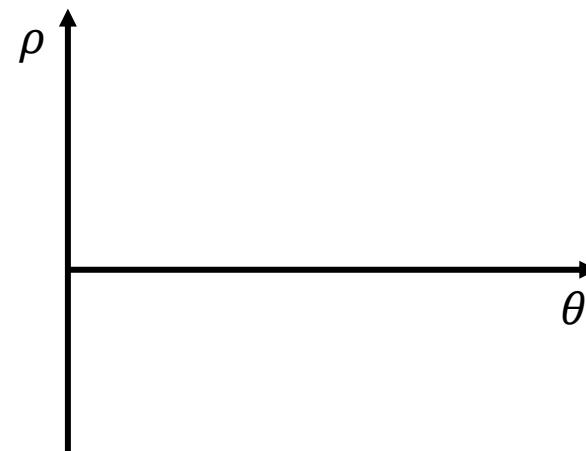


Image space

$$x \sin \theta - y \cos \theta + \rho = 0$$



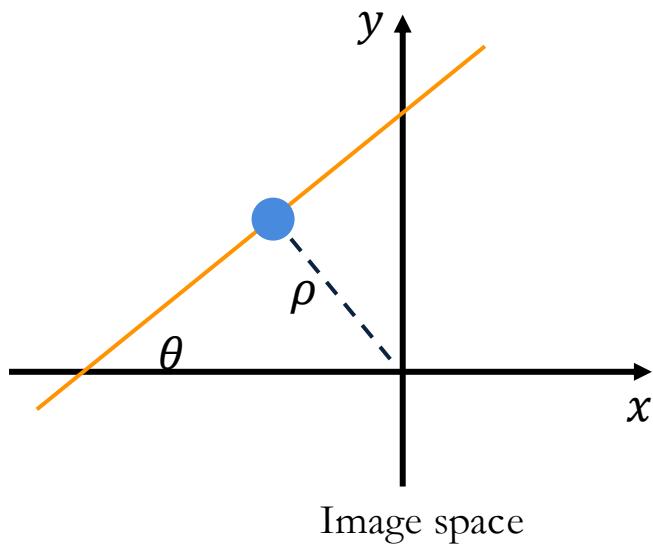
Parameter space

$$x \sin \theta - y \cos \theta + \rho = 0$$

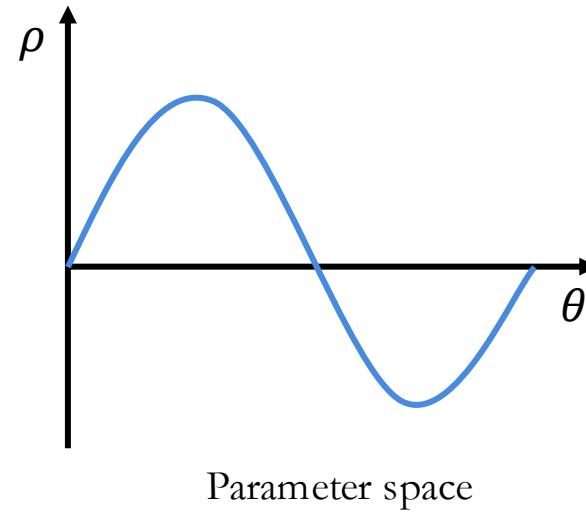


SOUTH DAKOTA
STATE UNIVERSITY

Better Parameterization?



$$x \sin \theta - y \cos \theta + \rho = 0$$

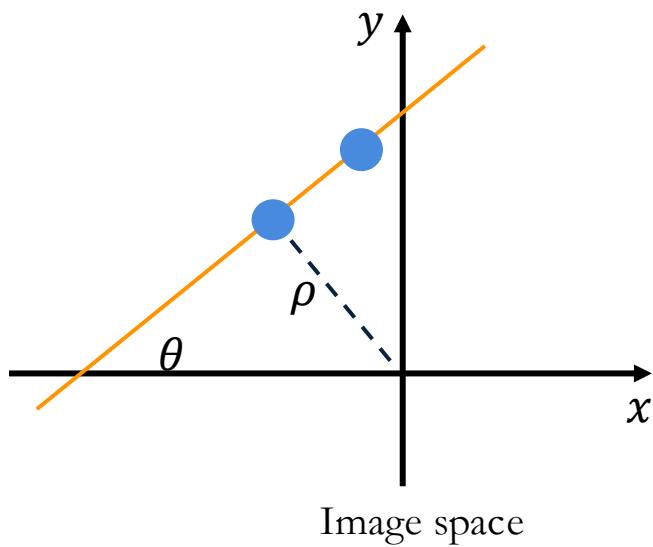


$$x \sin \theta - y \cos \theta + \rho = 0$$

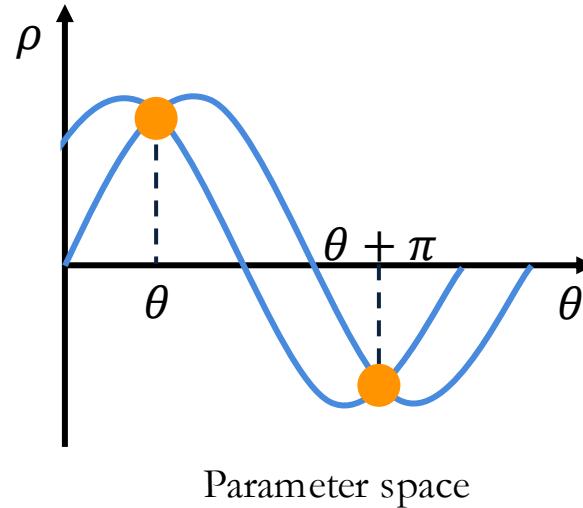


SOUTH DAKOTA
STATE UNIVERSITY

Better Parameterization?



$$x \sin \theta - y \cos \theta + \rho = 0$$



$$x \sin \theta - y \cos \theta + \rho = 0$$



SOUTH DAKOTA
STATE UNIVERSITY

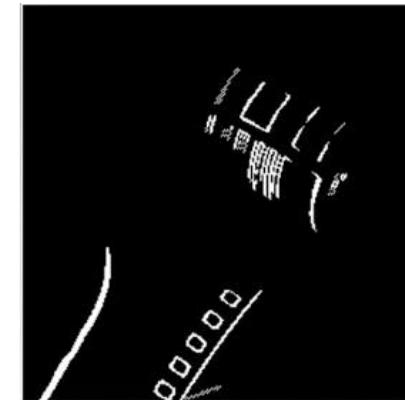
Detection Results



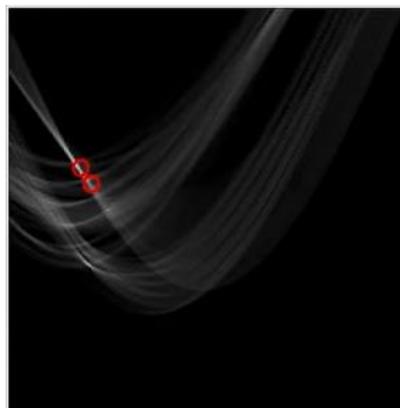
Original



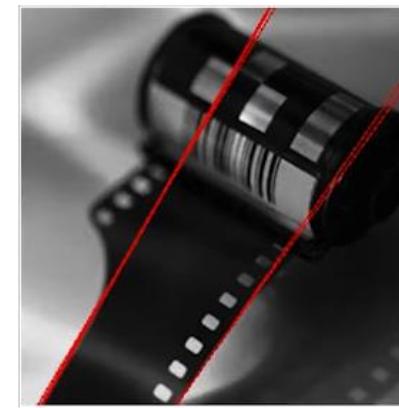
Gradient



Edge



Hough
Transform
 $A(\rho, \theta)$



Detected Lines

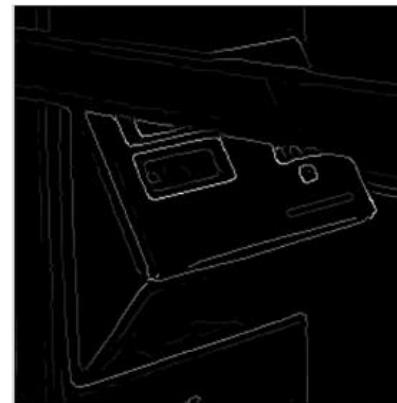


SOUTH DAKOTA
STATE UNIVERSITY

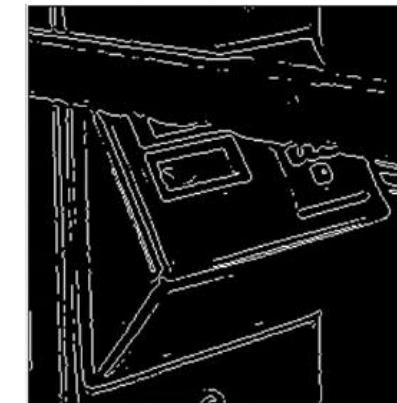
Detection Results



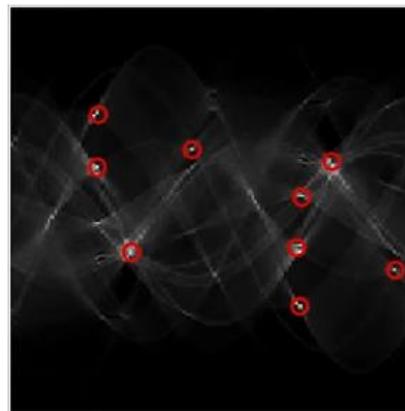
Original



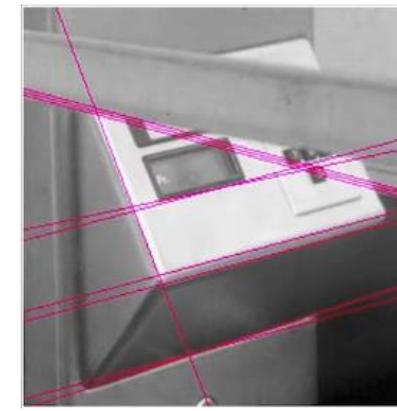
Gradient



Edge



Hough
Transform
 $A(\rho, \theta)$



Detected Lines



SOUTH DAKOTA
STATE UNIVERSITY

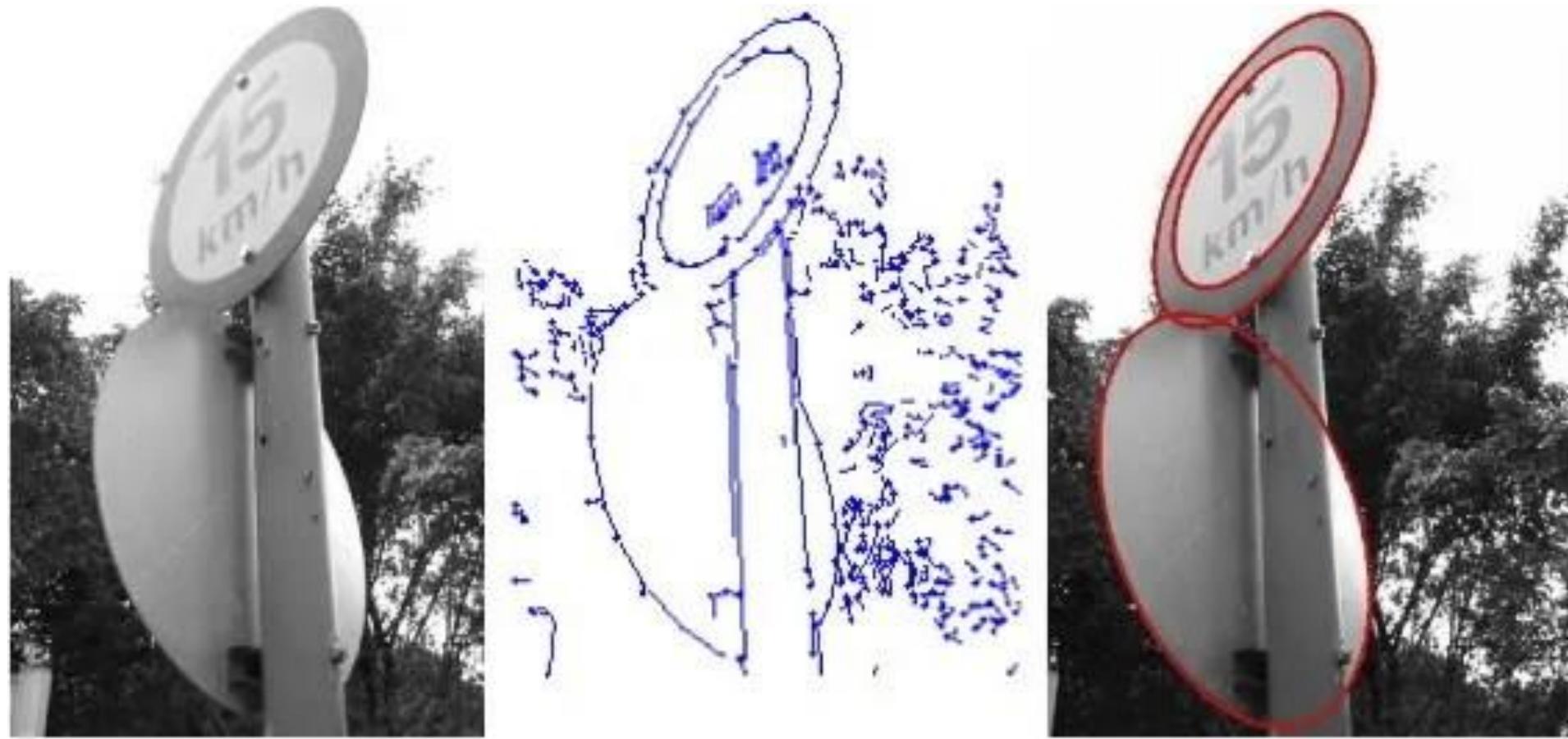


SOUTH DAKOTA
STATE UNIVERSITY

Hough Transform: Beyond Line Detection

Credit: slides of this section are adapted from CSC 492/592, Spring 2024

Image source: <https://laid.delanover.com/hough-transform-ellipse-detection-and-space-reduction/>



Hough Transform

- The Hough Transform is a feature extraction technique.
- Purpose: to find **imperfect** instances of objects within a certain class of shapes by a **voting scheme**.
- This voting procedure is carried out in a **parameter space**, from which object candidates are obtained as **local maxima** in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

Reference: https://en.wikipedia.org/wiki/Hough_transform



SOUTH DAKOTA
STATE UNIVERSITY

One Example: Finding Circle

- If radius r is known: Accumulator Array: $A(a, b)$

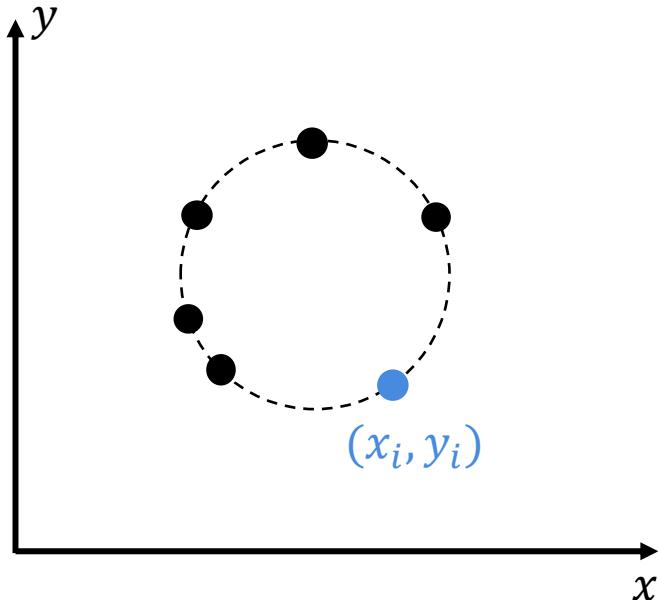
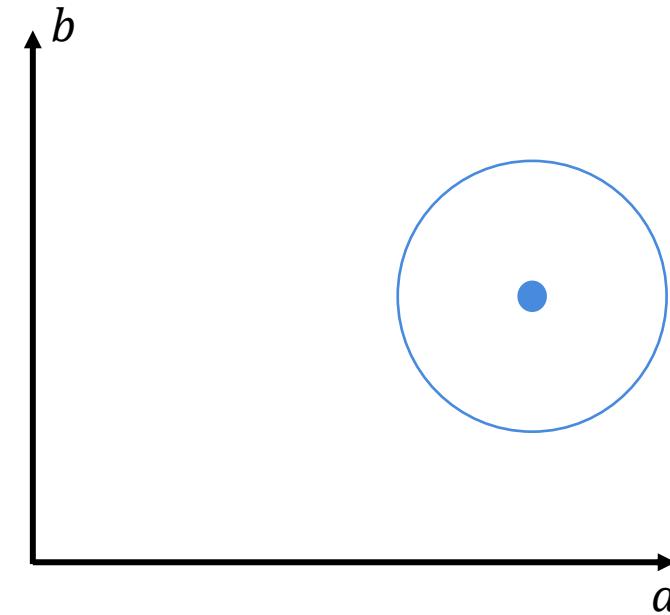


Image space

$$(\textcolor{orange}{x}_i - a)^2 + (\textcolor{orange}{y}_i - b)^2 = r^2$$



Parameter space

$$(\textcolor{orange}{a} - x_i)^2 + (\textcolor{orange}{b} - y_i)^2 = r^2$$

Finding Circle

- If radius r is known: Accumulator Array: $A(a, b)$

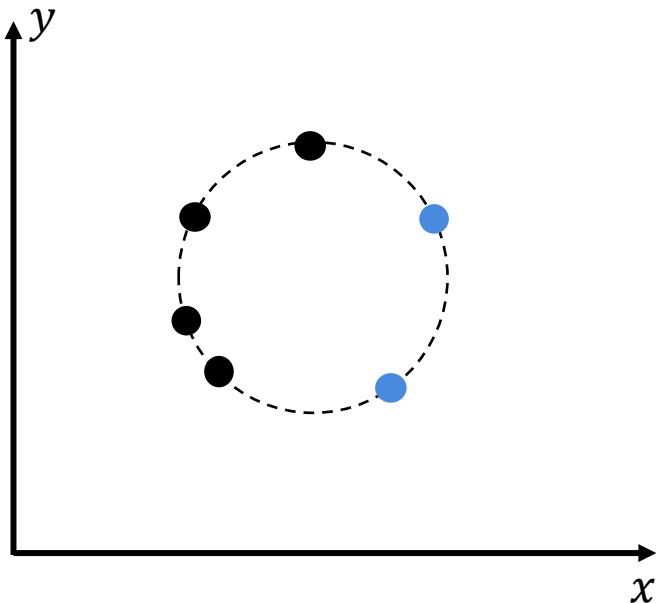
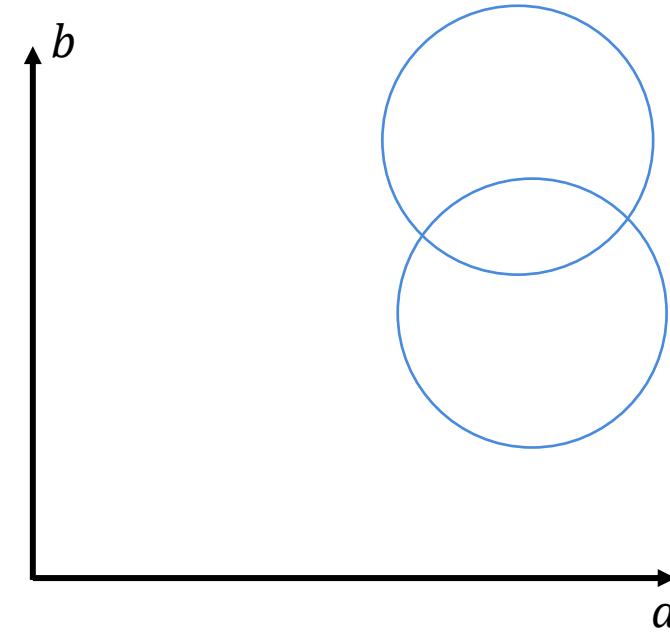


Image space

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$



Parameter space

$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

Finding Circle

- If radius r is known: Accumulator Array: $A(a, b)$

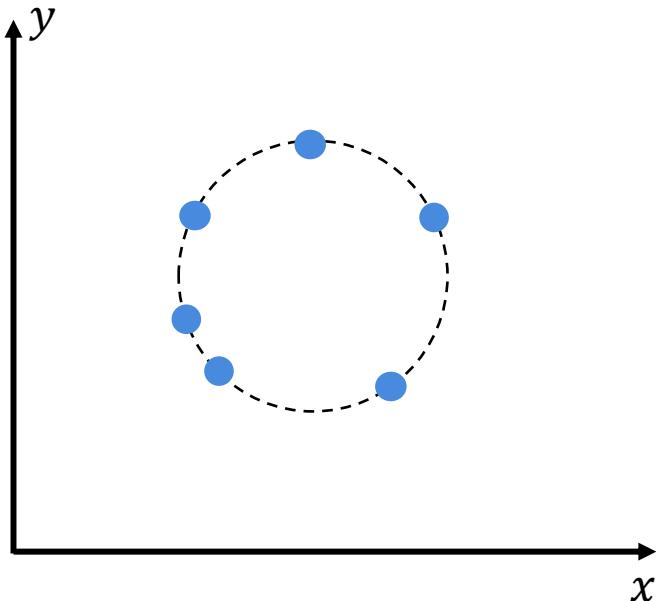
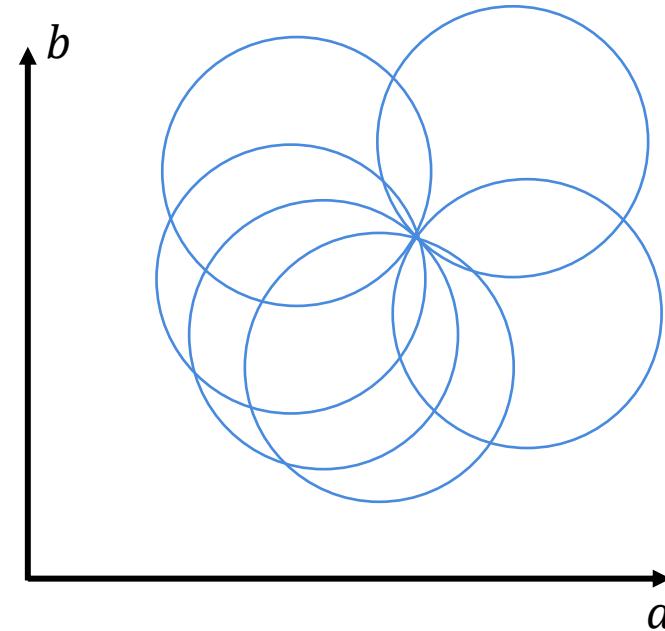


Image space

$$(\textcolor{orange}{x}_i - a)^2 + (\textcolor{orange}{y}_i - b)^2 = r^2$$



Parameter space

$$(\textcolor{orange}{a} - x_i)^2 + (\textcolor{orange}{b} - y_i)^2 = r^2$$



SOUTH DAKOTA
STATE UNIVERSITY

Finding Circle

- If radius r is known: Accumulator Array: $A(a, b)$

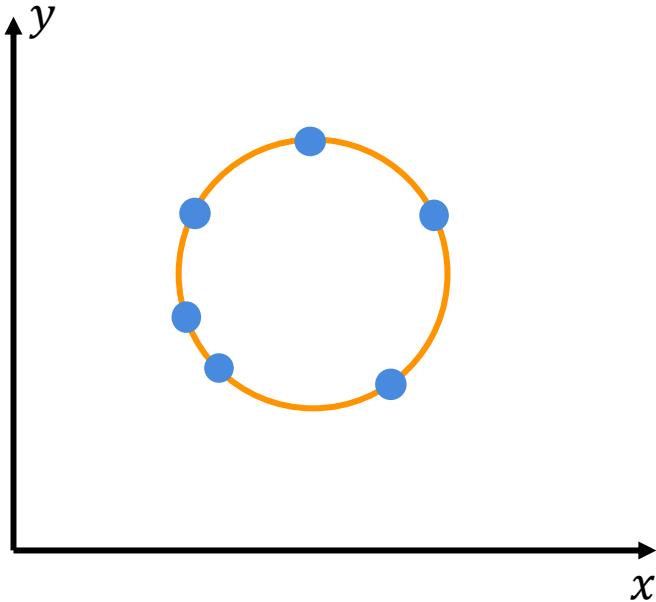
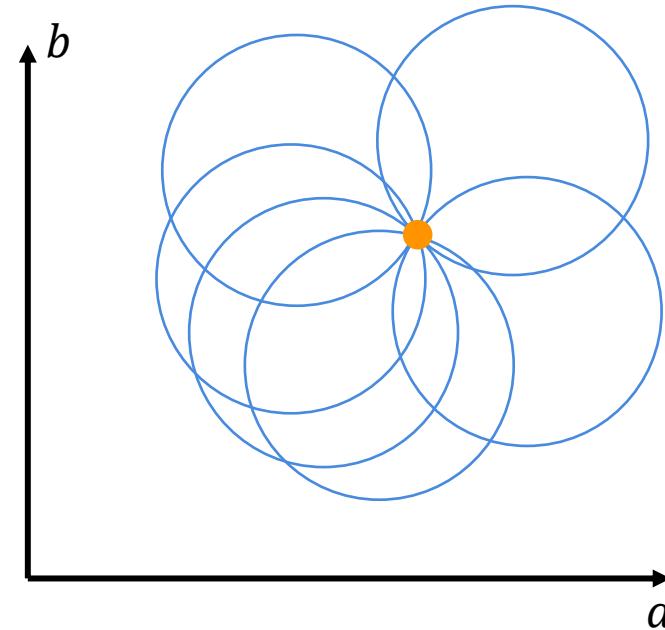


Image space

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$



Parameter space

$$(a - x_i)^2 + (b - y_i)^2 = r^2$$



SOUTH DAKOTA
STATE UNIVERSITY

Circle Detection Results

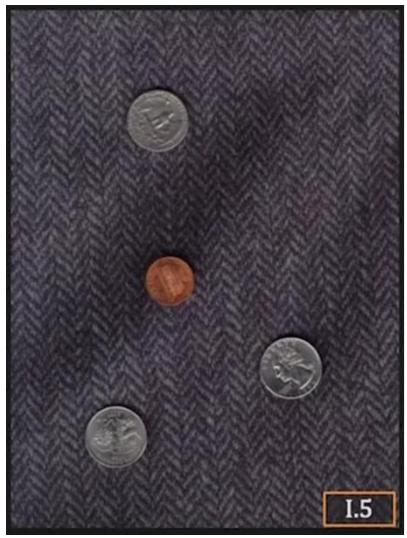


Original

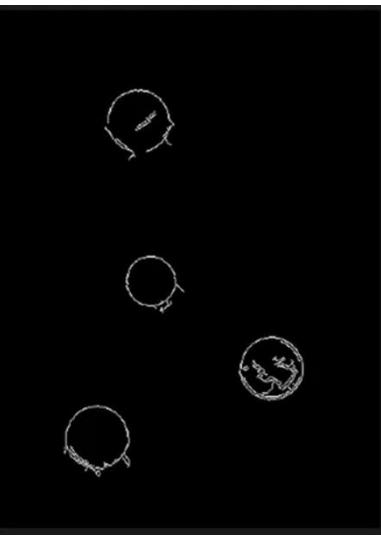


SOUTH DAKOTA
STATE UNIVERSITY

Circle Detection Results



Original

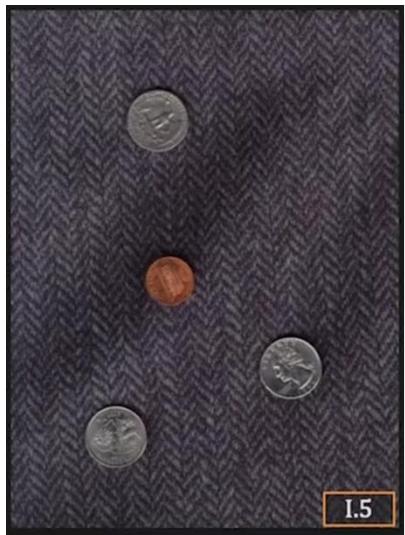


Edge

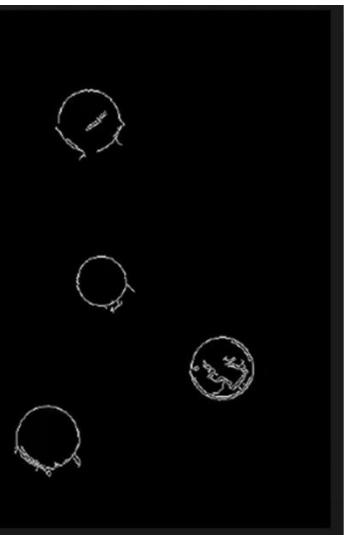


SOUTH DAKOTA
STATE UNIVERSITY

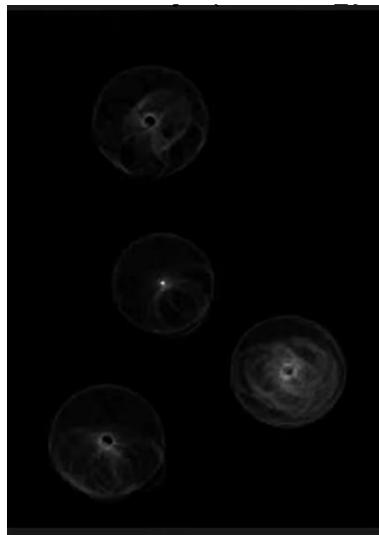
Circle Detection Results



Original



Edge



Hough Transform
 $A_1(a, b)$

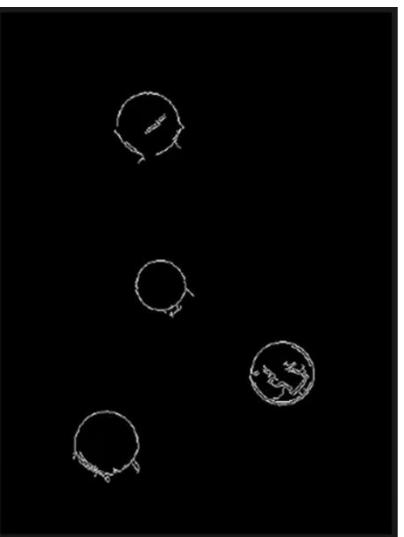


SOUTH DAKOTA
STATE UNIVERSITY

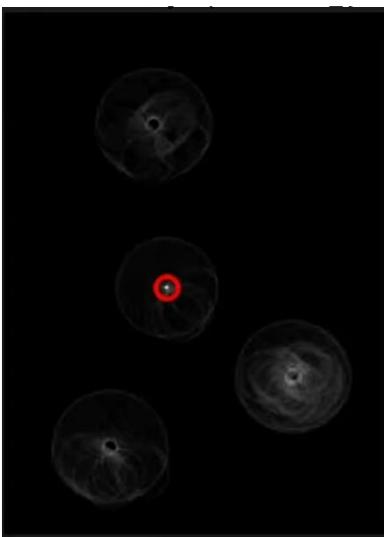
Circle Detection Results



Original



Edge



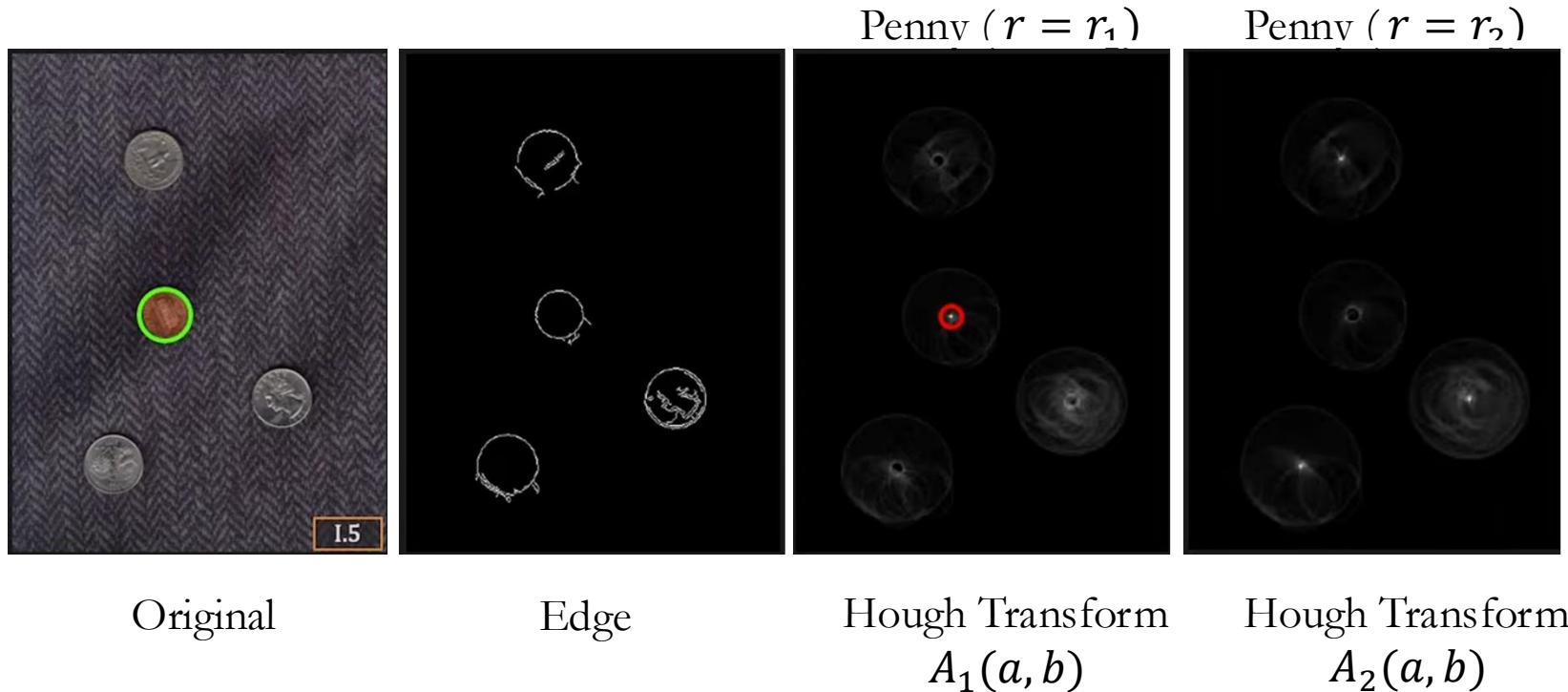
Hough Transform
 $A_1(a, b)$

Pennv ($r = r_1$)

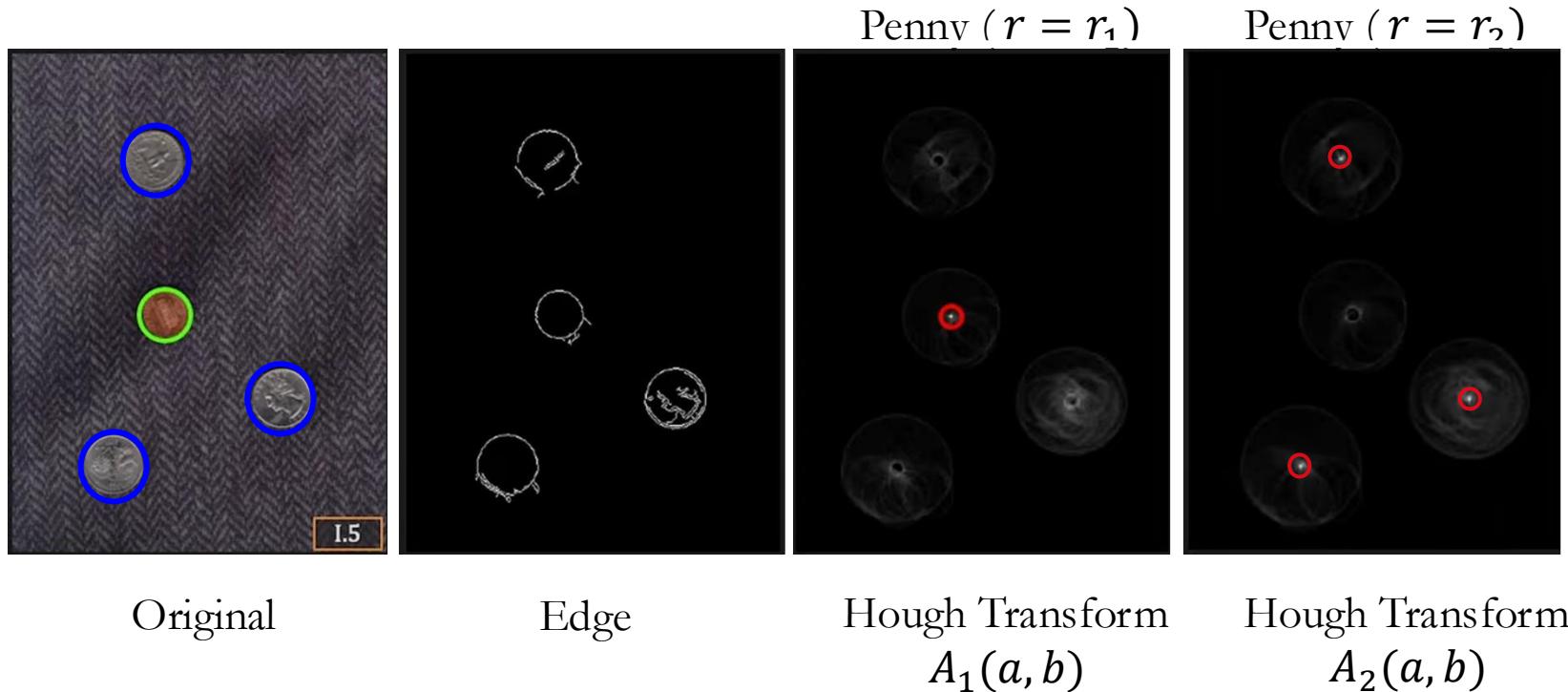


SOUTH DAKOTA
STATE UNIVERSITY

Circle Detection Results



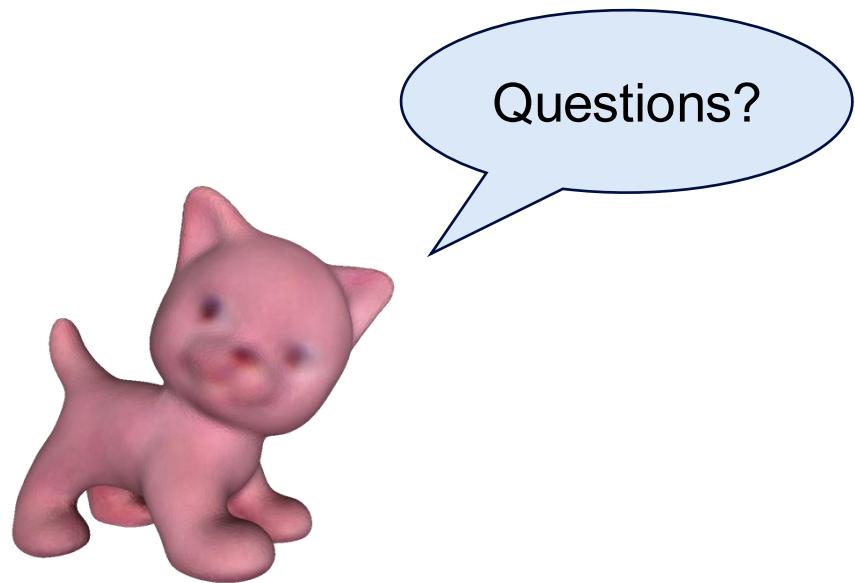
Circle Detection Results



SOUTH DAKOTA
STATE UNIVERSITY

Takeaway

- Why Features matter in computer vision?
- Different ways to detect edges
 - . Prewitt & Sobel
 - . Laplacian of Gaussian
 - . Canny Edge Detector
- Hough Transform
 - . The voting scheme
 - . How to detect lines
 - . How to detect circles



SOUTH DAKOTA
STATE UNIVERSITY