Faculty of Engineering
Cairo University

Computer Vision
Hough Transform & Active Contours

Team 8

By:
Ahmed Mohamed,
Ahmed Mahmoud,
Eman Emad,
Rawan Shoaib

I. Introduction

IN this task we explore image analysis techniques for detecting edges and boundaries. We'll implement the Canny edge detector for capturing sharp image transitions. Additionally, the Hough transform will be used to identify specific shapes like lines, circles, and ellipses. Finally, the Active Contour Model (snake) algorithm will be employed to trace object outlines within the image. The following sections detail these algorithms and showcase the results obtained when applied to a set of images.

mds
March, 24, 2025

II. Tasks to implement

A. Line Detection:

This algorithm detects straight lines in an image using the Hough Transform. Given an input image I, the algorithm processes it through edge detection, maps edge points into Hough space, accumulates votes, extracts prominent lines, and finally overlays detected lines on the image.

- **Inputs**
  - image: Input image (NumPy array, shape (H,W,C) where H is height, W is width, and C is the number of channels).
  - num_rho: Number of discrete values for the distance parameter ρ.
  - num_theta: Number of discrete angles for θ (typically 180).
  - blur_ksize: Kernel size for Gaussian blurring.
  - low_threshold: Low threshold for Canny edge detection.
  - high_threshold: High threshold for Canny edge detection.
  - hough_threshold_ratio: Threshold ratio for selecting significant lines in the Hough accumulator.
  - min_angle: Minimum angle for filtering detected lines.
  - max_angle: Maximum angle for filtering detected lines.

- **Outputs**
  - output_image: the final image with detected lines.

- Algorithm Steps
  - **Initialization**
    Before execution, the algorithm initializes the required data structures and computes pre-defined values. First we convert image to grayscale, then apply gaussian blur and detect edges using canny edge detector. Finally compute maximum possible ρ and step sizes for parameter space dρ and create discretized ranges to initialize accumulator.
  - **Hough Transform Voting**
    Each edge pixel contributes votes in the accumulator. We find edge pixels and increment the accumulator.
  - **Peak Extraction from Accumulator**
    To extract the most prominent lines, we apply thresholding and non-maximum suppression. we determine voting threshold and find peaks in accumulator above threshold, then apply non-maximum suppression if multiple detected peaks are within a small distance we keep the strongest peak.
  - **Convert parameters to Line Coordinates**
    For each detected line in parametric form we convert it to a visualizable Cartesian line by computing unit vector components, base point and two endpoints to extend the line.
  - **Line Filtering**
    Compute line angles to ensure all angles remain in the valid range from minimum to maximum angle.
  - **Draw Detected Lines**
    For each valid line, draw it on the original image.

B. Circle Detection:

The algorithm detects circular shapes in an image using the Hough Transform for Circles. Instead of a 2D accumulator (as in line detection), it uses a 3D accumulator (x,y,r) where votes are cast for potential circle centers and radiance.

- **Inputs**
  - image: grayscale image 2D NumPy array of shape (H,W).
  - r_min: Minimum circle radius.
  - r_max: Maximum circle radius.

  - min_dist: Minimum distance to filter out overlapping circles.
  - threshold_ratio: Threshold ratio for peak detection in the accumulator.

- **Outputs**
  - output_image: Image with drawn circles.
  - detected_circles: List of detected circles (a,b,r).

- Algorithm Steps
  - **Initialization**
    sets up a 3D accumulator for storing votes. we extract image dimensions and initialize a 3D accumulator where each element represents the number of votes for a circle centered at (x,y) with radius.
  - **Hough Transform Voting**
    Compute Votes for Each Edge Pixel. Detect edge pixels and precompute trigonometric values For each edge pixel, iterate over all radiances. Compute possible circle centers and filter out invalid center points to update the accumulator.
  - **Peak Extraction from Accumulator**
    Extract local maxima. Compute the threshold based on max votes to find all peak locations where. Convert to list of circle candidates.
  - **Non-Maximum Suppression**
    To remove overlapping or redundant circles. we sort circles by vote strength (descending order). And iterate through circles For each candidate (a,b,r), check previously selected circles to compute euclidean distance if it is less than thereshold discard the weaker circle. Outputs final circle list.
  - **Draw Circles on Image**
    For each detected circle (a,b,r), draw it on the original image.

C. Ellipse Detection:

This algorithm detects ellipses using the Hough transform, based on focus points.

- **Inputs**
  - image: Input image (grayscale).
  - min_d: Minimum major axis length.
  - max_d: Maximum major axis length.
  - step_size: Step size for iterating over possible major axes.
  - threshold_factor: Fraction of max votes to consider as a threshold.
  - tolerance: Angle tolerance (in degrees) for perpendicularity check.

- **Outputs**
  - best_ellipses : A list of detected ellipses.
  - output_image: The image with ellipses overlaid.

- **Algorithm Steps**
  - **Edge Detection**
    We apply canny edge detection to identify contour pixels where the ellipse boundaries might exist.
  - **Initialize Hough Space**
    Since the accumulator will store votes for potential ellipse centers based on focus pairs, it is initialized as a default dictionary of lists. Each key corresponds to a potential center. Each value stores valid focus pairs that support this center.
  - **Voting in Hough Space**
    Cast votes for ellipse based on unique pairs of edge pixels. Loop through all unique pairs of edge points to compute center point and the euclidean distance between the two focus points. Check if the distance is within valid limits to store the focus pair in the accumulator under the computed center.
  - **Extracting Ellipse Candidates**
    Here we identify strong peaks in the accumulator. First, we find the maximum vote and compute a filtering threshold. Select candidate centers. Then, for each valid center compute all distances between stored focus pairs to find the major axis as the longest pair and the minor axis as the shortest. Compute the angle between axes to accept only ellipses where angle about ninety degree and store valid ellipses.
  - **Drawing Detected Ellipses**
    Draw each detected ellipse from the list. This results in an output image with detected ellipses.

D. Active Contour Mode:

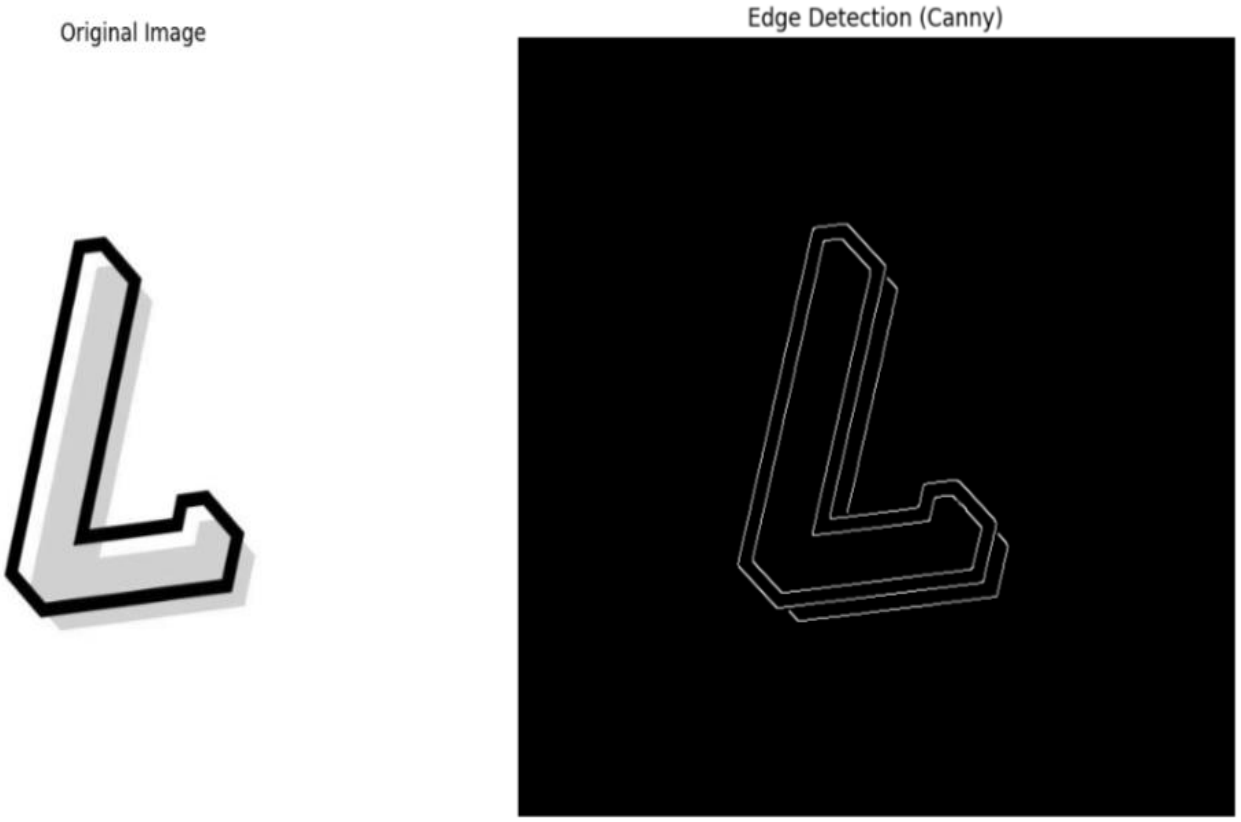
The Active Contour Model, also known as Snakes, is an edge-detection technique that iteratively deforms a contour toward object boundaries. The model balances internal forces (smoothness), external forces (image features), and constraint forces to evolve the contour over time.

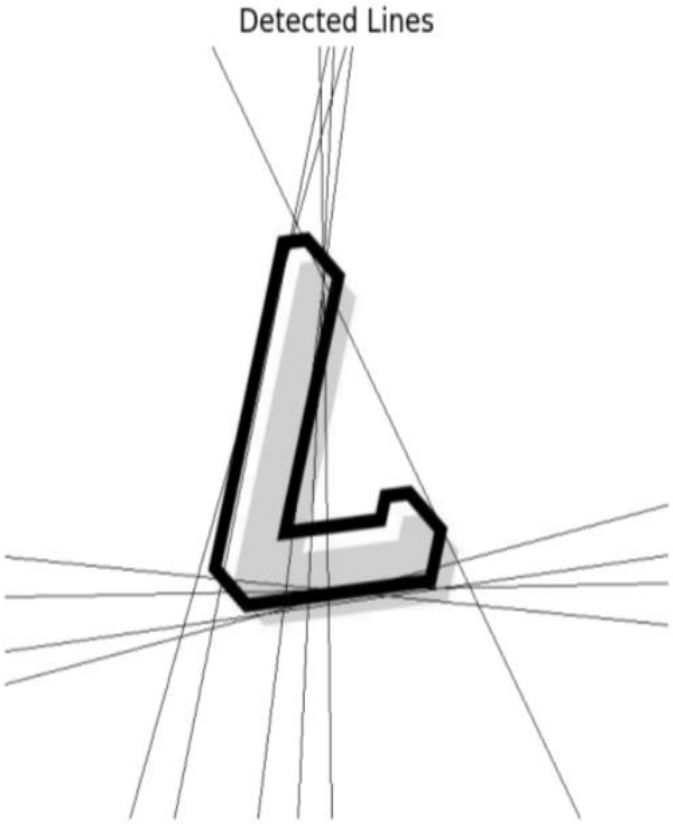
- **Inputs**
  - image: The grayscale image in which the contour will evolve.
  - center: Tuple (x, y) for the center of the snake.
  - radius: radius of the snake.
  - points: The number of discrete points that make up the contour.
  - sigma: Gaussian smoothing parameter (removes noise from the image).
  - beta: Curvature weight(prevents sharp bends).
  - gamma: Step size (controls update speed).
  - alpha: Elasticity weight (keeps the contour smooth).
  - w_edge: Weight for external force (controls how strongly the snake is pulled toward the edges).

- **Outputs**
  - new_snake: A refined set of points outlining the object boundary.

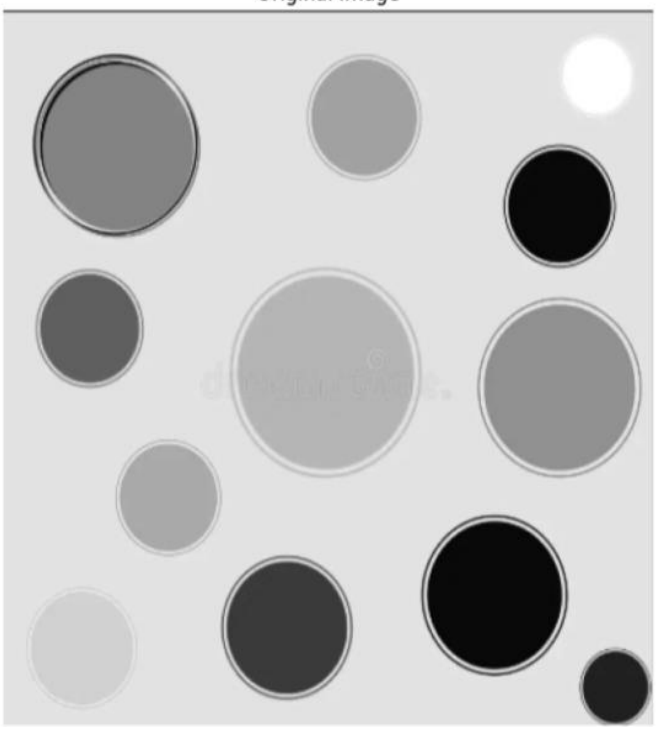
- **Algorithm Steps**
  - **Initialize the Snake**
    Create an initial contour in a circular shape. Generate points evenly spaced around a circle using parametric equations
  - **Compute External Energy**
    Compute the image energy that attracts the snake to object edges. Smooth the image using a Gaussian filter to remove noise, then compute the image gradient and the edge energy map.
  - **Iterative Snake Evolution**
    Adjust the contour points by minimizing energy function. For each contour point, we compute internal and external forces (edge force) which pull the contour towards high-gradient areas. Update the Contour Point by moving the snake point in the direction of force. Finally, to ensure snake stays in bounds we clip values to keep points inside image dimensions.

III. Results

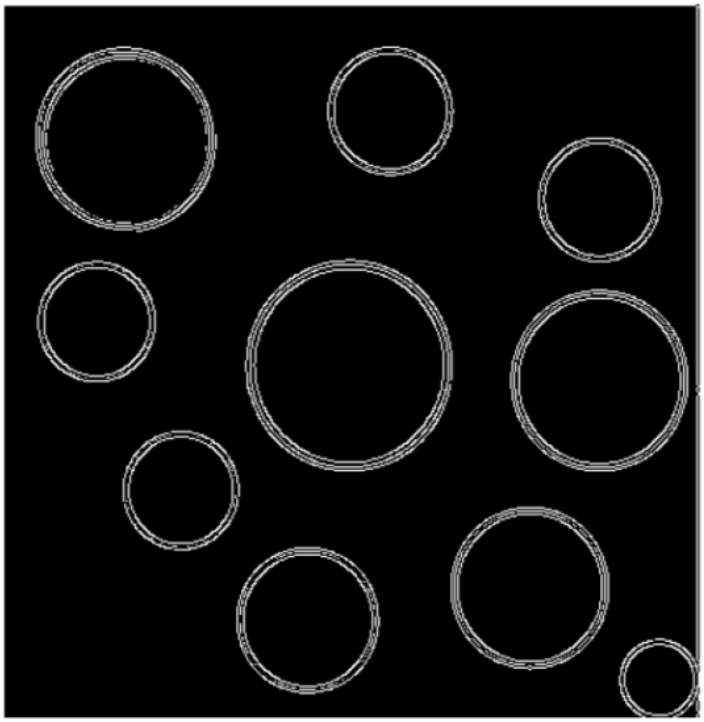




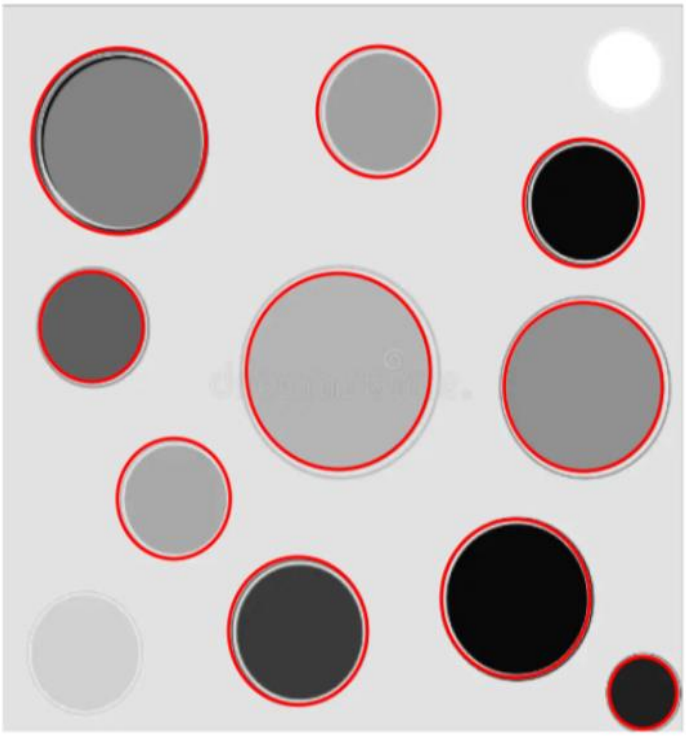
Original Image



Edge Detection

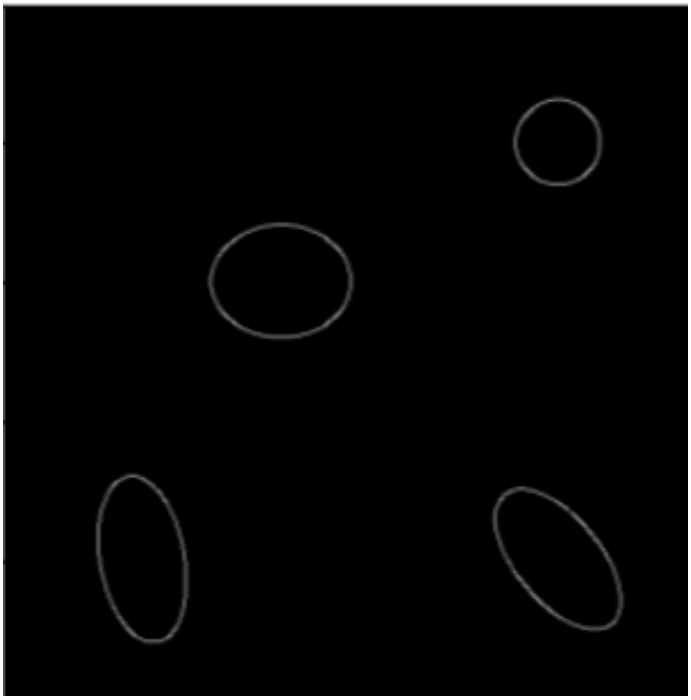


Detected Circles

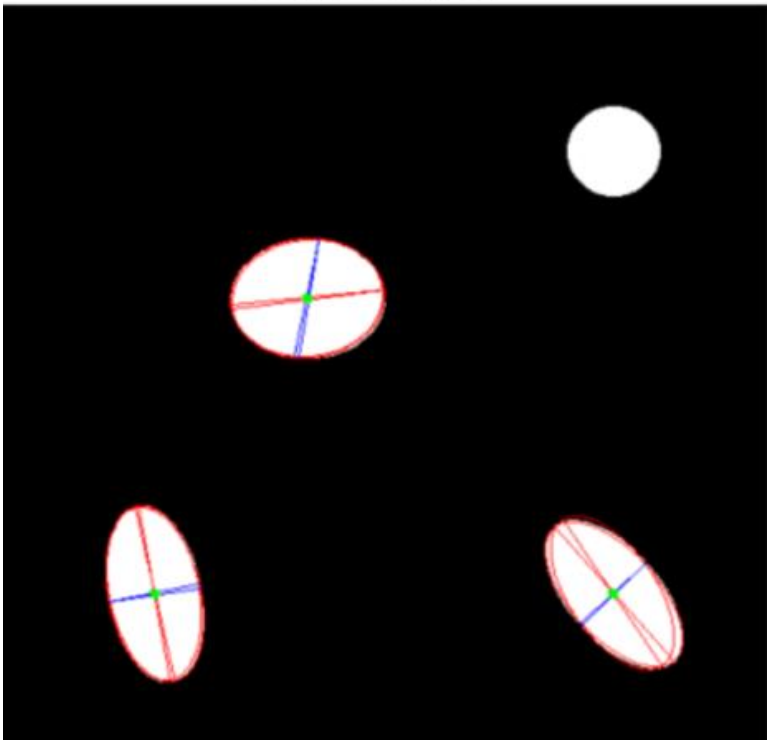


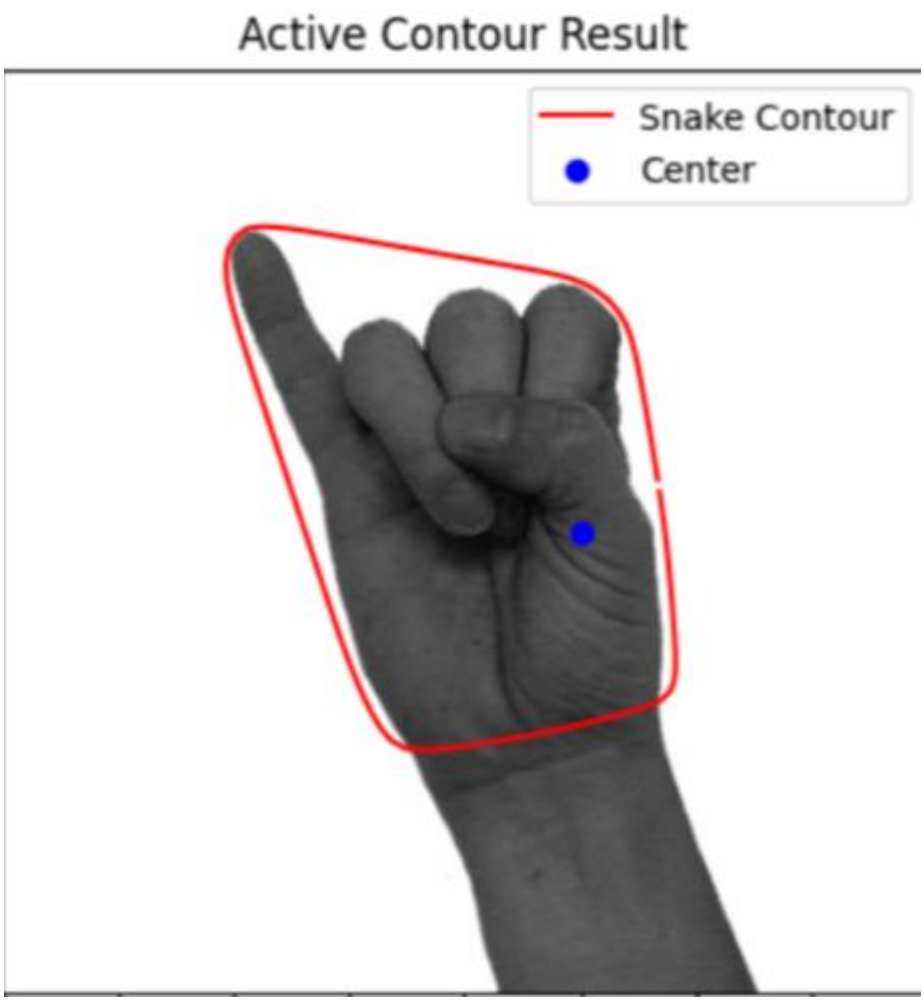
1. Original Image

2. Edge Detection



5. Final Ellipses




Active Contour Result
Snake Contour
Center