



Faculty of Engineering
Cairo University



Cairo University

Computer Vision

Hough Transform & Active Contours

Team 19

Name	Section	B.N
Mohamed Alaa Ali	2	19
Mohamed Ibrahim Ismail	2	9
Mohamed El-sayed Ali	2	10
Mohamed El-sayed Eid	2	11

Introduction:	3
Tasks to be Implemented:	3
1. Hough Transform:	3
1.1 Line Detection:	4
1.2. Circle Detection:	6
1.3 Ellipse Detection:	7
1.4 Sample Results:	10
2. Active Contours:	11
2.1 Description and Algorithm:	11
2.2 Approach and Equations:	13
2.3 Observations and Comments	16
2.4 Results	17

Introduction:

This task explores 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.

Tasks to be Implemented:

- Use Hough transform to identify lines, circles, and ellipses
- Active Contours “Snakes”

1. Hough Transform:

The Hough Transform, is a pivotal technique in computer vision for detecting shapes like lines, circles, and ellipses in digital images. By converting the problem from spatial to parameter space, it identifies shape parameters through accumulation analysis. Enhanced over time, it finds application in diverse fields such as robotics and medical imaging. This report explores the principles and implementation of the Hough Transform, focusing on its effectiveness in detecting lines, circles, and ellipses in a specific context, demonstrating its significance in real-world image analysis tasks.

1.1 Line Detection:

Algorithm:

- **Input:**
 - **low_threshold:** The low threshold value for the Canny edge detector
 - **high_threshold:** The high threshold value for the Canny edge detector.
 - **smoothing_degree:** The degree of smoothing for the Canny edge detector.
 - **threshold:** The threshold value for line detection in the Hough space.
 - **rho:** The resolution of distance parameter in the Hough space.
 - **theta:** The resolution of angle parameter in the Hough space.
- **Initialization:**
 - Initialize image as a copy of the input image.
 - Apply Canny edge detection to the image using the provided parameters to obtain edges.
 - Compute the maximum possible length of the diagonal of the image.
 - Generate an array of **theta** angles spanning from $-\pi/2$ to $\pi/2$ and an array of **rho** values spanning from negative diagonal length to positive diagonal length.
- **Accumulator Initialization:**
 - Create an **accumulator** array of dimensions (**num_rhos** x **num_thetas**) filled with zeros.

- **Voting:**
 - Iterate over the edge pixels detected by the Canny edge detector.
 - For each edge pixel, iterate over **theta** angles.
 - Compute the corresponding **rho** value and increment the **accumulator** at the corresponding index.
- **Peak Detection:**
 - Find indices in the **accumulator** array where the values exceed the specified threshold.
 - Extract the corresponding **rho** and **theta** values from the accumulator.
- **Line Drawing:**
 - Convert the detected polar coordinates to Cartesian coordinates and draw lines on the input image.
- **Return Output:**
 - Return the input image with detected lines drawn on it.

1.2. Circle Detection:

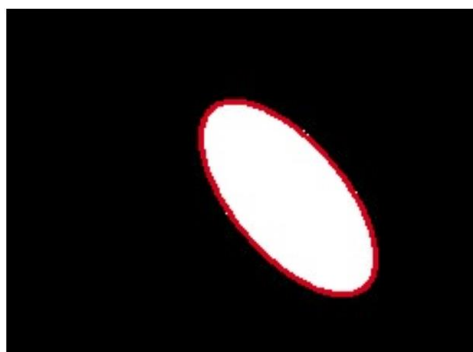
Algorithm:

- **Input:**
 - **low_threshold:** The low threshold value for the Canny edge detector
 - **high_threshold:** The high threshold value for the Canny edge detector.
 - **smoothing_degree:** The degree of smoothing for the Canny edge detector.
 - **threshold:** The threshold value for circle detection in the Hough space.
 - **min_radius:** The minimum radius of circles to detect.
 - **max_radius:** The maximum radius of circles to detect.
 - **min_dist:** The minimum distance between detected circles.
- **Initialization:**
 - Initialize image as a copy of the input image.
 - Apply Canny edge detection to the image using the provided parameters to obtain **edges**.
- **Circle Parameters Generation:**
 - Generate a set of points representing potential circle centers and radii based on the specified range of radii and number of theta angles.
- **Accumulator Initialization and Voting:**
 - Initialize an **accumulator** dictionary to accumulate votes for potential circles.
 - Iterate over **edge** points detected by the Canny edge detector.

- For each edge point, iterate over the generated circle parameters.
- Increment the accumulator for each potential circle that passes through the edge point.
- **Circle Detection:**
 - Iterate over the **accumulator** items, sorted by the number of votes in descending order.
 - For each potential circle, check if it meets the threshold and has minimum distance of (**min_dist**) with already detected circles.
 - If the conditions are met, add the circle to the list of detected circles and draw it on the image.
- **Return Output:**
 - Return the input image with detected circles drawn on it.

1.3 Ellipse Detection:

For the Ellipse transform there is the traditional algorithm that uses the center coordinates of the potential ellipse, the major & minor axis, and the angle of orientation. This algorithm uses 4 loops, so it's very slow and doesn't converge at the end.



This approach detect the ellipse perfectly. But it has one problem: it detects only one ellipse in the photo.

Algorithm:

draw_ellipse: This method takes the original image and the parameters of an ellipse (center coordinates, major and minor axes lengths, and rotation angle) and draws the ellipse on the image using OpenCV's `cv2.ellipse` function.

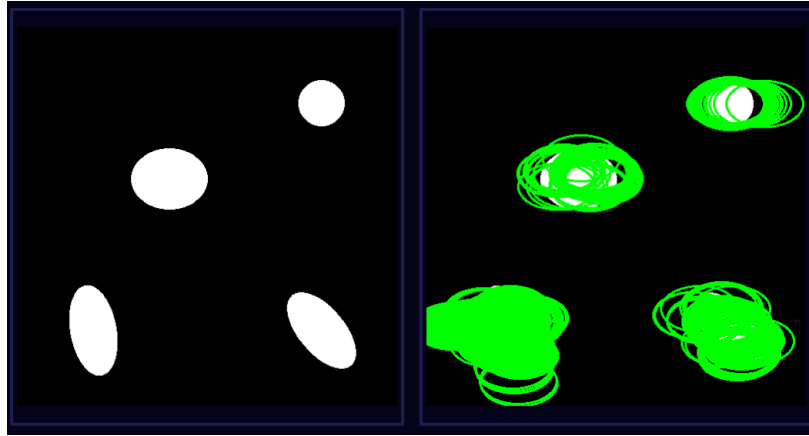
detect_ellipses: This method takes an image and parameters such as `min_major_axis` and `threshold`. It first applies the Canny edge detection algorithm to extract edges from the image. Then, it calls the

find_ellipse: This method takes the edge-detected image along with parameters such as `min_major_axis` and `threshold`. It iterates over pairs of edge points and computes the parameters of the ellipse passing through these points. It constructs an accumulator array to keep track of the number of votes for each possible minor axis length b . If a local maximum in the accumulator array exceeds the specified threshold, the parameters of the detected ellipse are returned. Otherwise, it returns `None` if no ellipses are detected.

Overall, this algorithm iterates through all pairs of edge points, constructing possible ellipses, and votes for the minor axis length based on the consistency of neighboring edge points. If a significant number of votes are accumulated for a particular minor axis length, an ellipse is detected.

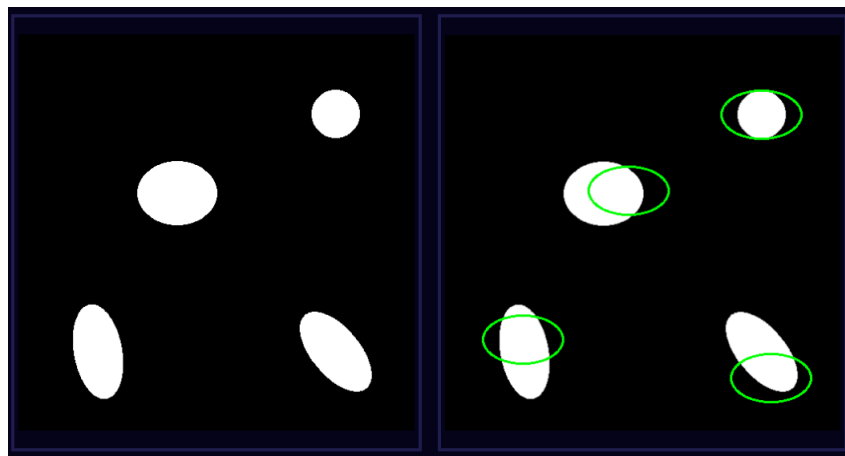
Trying to improve the algorithm used for ellipses, we found a way to reduce the parameter space for the Hough transform in a book called “Feature Extraction in Computer Vision and Image

Processing”, This method uses the geometric features of the ellipse to reduce the dimensionality of the accumulator matrix from 5d to 2d matrix, doing so reduces the computation time needed significantly (3 to 6 secs).



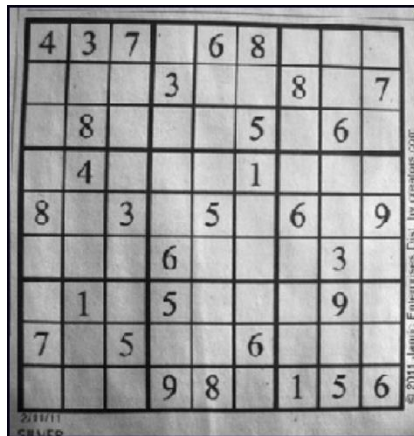
This Approach detected the centers of the potential Ellipses as indicated in the above example. It's obvious that it detects a lot of overlapping centers for the same ellipse.

To solve this issue we used non max suppression to get only one ellipse. And this is the result after using it.

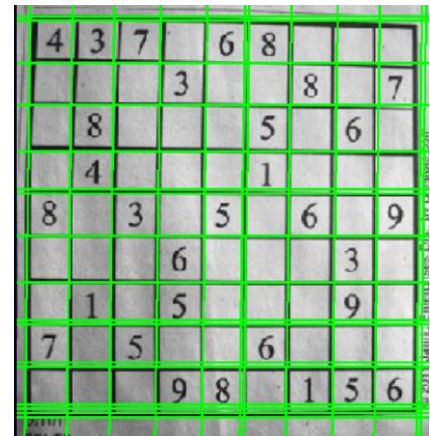


This Approach has one problem that it only gives the center coordinates of the potential ellipse, so it only detects the center and with these coordinates it's not possible to represent the detected images on the original photo.

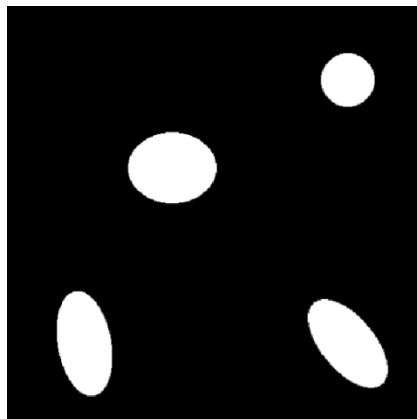
1.4 Sample Results:



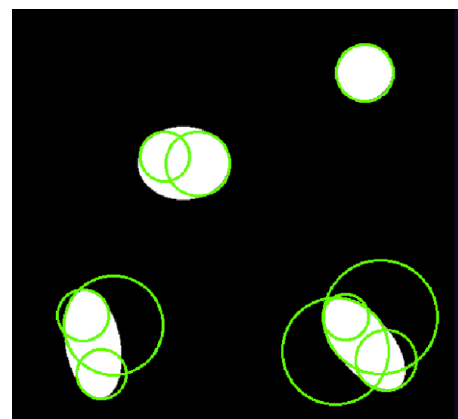
Original Image



After line detection with Hough Transform



Original Image



After Circle detection with Hough Transform

2. Active Contours:

Active contours, also called snakes, are a technique used in image processing to automatically identify the boundaries of objects in an image. Imagine a flexible rope that can deform to fit around the object. The "rope" is influenced by forces in the image that pull it towards edges and by internal forces that make it resist stretching or bending too much. By minimizing these energies, the active contour can effectively trace the object's outline. This is useful for tasks like object segmentation, where you want to separate the object of interest from the background.

2.1 Description and Algorithm:

- The objective is to discover a contour that most accurately represents the boundary of an object.
- This method is employed on the gradient magnitude of the image rather than on individual edge points, as seen in techniques such as the Hough transform.

Algorithm

- **Inputs:**
 - **Grayscale Image:** The input image on which the contour detection will be performed.
 - **Circle Center and Radius:** Parameters defining the initial contour shape (e.g., a circular region).
 - **Alpha:** Weight parameter for internal energy calculation.
 - **Beta:** Weight parameter for external energy calculation.
 - **Gamma:** Weight parameter for gradient energy calculation.

- **Window Size:** Size of the neighborhood window for energy calculation.
- **Number of Points:** Number of points to initialize the contour.
- **Number of Iterations:** The number of iterations to perform for contour refinement.

- **Outputs:**
 - **Final Contour:** The list of (x, y) tuples representing the refined contour points.
 - **Output Image:** The input image with the detected contour drawn.

- **Initialization:**
 - Initialize the contour around the specified shape (e.g., a circle) with the given number of points.
 - Energy Calculation:

- **Internal Energy:**
 - Calculate the curvature-based internal energy for each point based on its neighbors.

- **External Energy:**
 - Calculate the intensity-based external energy for each point in the image.

- **Gradients:**
 - Calculate the gradients of the contour at each point based on the change in coordinates.

- **Point Energy Calculation:**
 - Combine the internal energy, external energy, and gradients to compute the total energy of each point in the contour.
- **Snake Operation (Contour Evolution):**

For each point in the current contour:

 - Determine a new position for the point by minimizing the total energy within a local neighborhood.
 - Update the contour with the new positions of the points.
- **Iteration:**
 - Repeat the snake operation for the specified number of iterations to refine the contour.
- **Output:**
 - Draw the final contour on the original image.
 - Save the output image with the detected contour drawn.

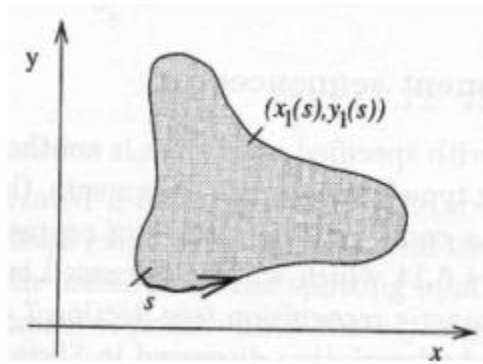
2.2 Approach and Equations:

The approach involves formulating an energy functional that evaluates the suitability of the contour.

- Optimal outcomes are associated with minimizing this functional.
- The objective is to minimize the functional concerning the contour parameters.

2.2.1 Contour parameterization

The snake is a contour represented parametrically as $c(s) = (x(s), y(s))$ where $x(s)$ and $y(s)$ are the coordinates along the contour and $s \in [0,1]$.



2.2.2 The energy functional

- The energy functional used is a sum of several terms, each corresponding to some force acting on the contour.
- A suitable energy functional is the sum the following three terms:

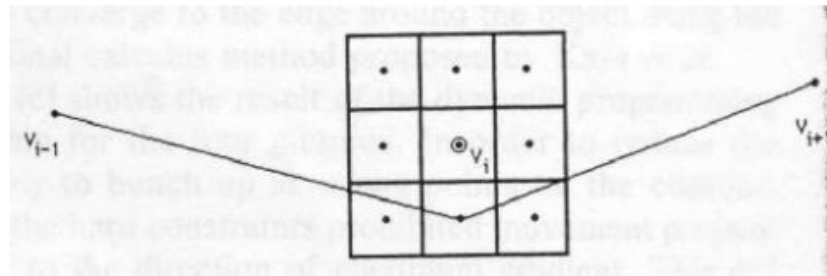
$$E = \int (\alpha(s)E_{cont} + \beta(s)E_{curv} + \gamma(s)E_{image})ds$$

- The parameters α , β , and γ control the relative influence of the corresponding energy terms and can vary along C
- Each energy term serves a different purpose:
 - E_{image} : it attracts the contour toward the closest image edge.
 - E_{cont} : it forces the contour to be continuous.
 - E_{curv} : it forces the contour to be smooth.
- E_{cont} and E_{curv} are called internal energy terms.
- E_{image} is called external energy term

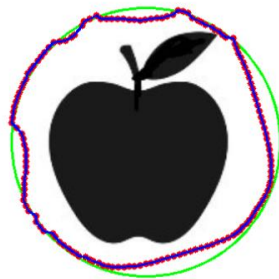
2.2.3 A greedy algorithm

- A greedy algorithm makes locally optimal choices, hoping that the final solution will be globally optimum.

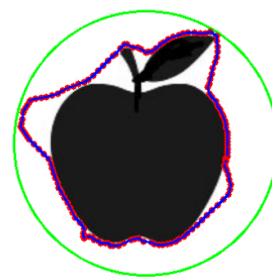
- Step1 (greedy minimization): each point of the snake is moved within a small neighborhood (e.g., $M \times M$) to the point which minimizes the energy functional



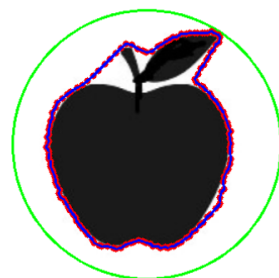
- Step 2 (corner elimination): search for corners (curvature extrema) along the contour; if a corner is found at point p_j , set β_j to zero.



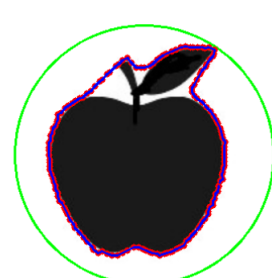
Contour After 10 Iterations



Contour After 50 Iterations



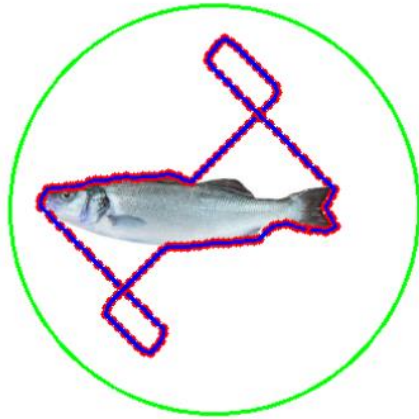
Contour After 100 Iterations



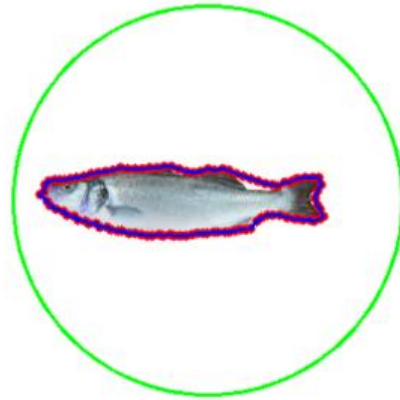
Contour After 200 Iterations

2.3 Observations and Comments

- The effectiveness of the suggested approach is heavily influenced by the configuration of model parameters.




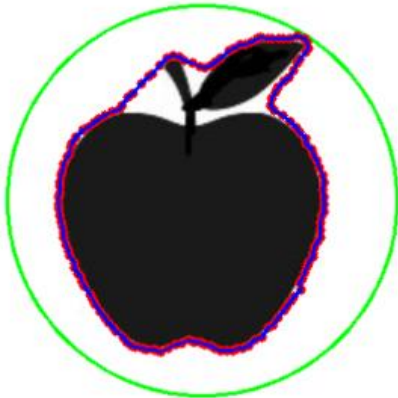

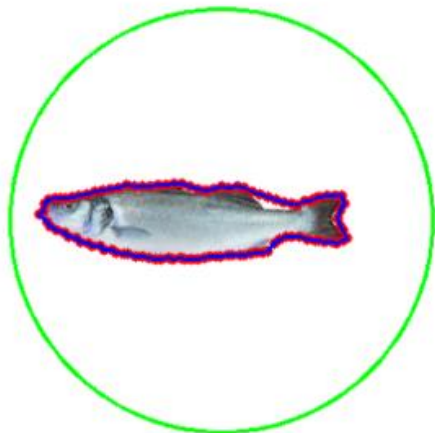
Contour With 150-point



Contour With 100 points

- This approach is simple and has low computational requirements ($O(MN)$). Here, 'N' represents the number of points on the contour, while 'M' denotes the possible directions at each point on the contour.
- Despite its computational efficiency, this point-wise method does not guarantee convergence.
- Works well as far as the initial snake is not too far from the desired solution.

2.4 Results

Parameters	Original image	Image with contour
Window Size: 8 #Iterations: 300 #Points: 150 Alpha: 10 Beta: 3 Gamma: 1		
Window Size: 8 #Iterations: 400 # Points: 100 Alpha: 10 Beta: 3 Gamma: 1		
Window Size: 8 #Iterations: 300 #Points: 120 Alpha: 10 Beta: 3 Gamma: 15	