Faculty of Engineering
Cairo University

Second Year – 2nd Term

Systems & Biomedical Eng. Department

Cairo University

COMPUTER VISION TASK 2: FEATURE EXTRACTION WITH HOUGH AND SNAKE REPORT

Team 5

Salma Ashraf

Sarah Mohamed

Habiba Salama

Hager Samir

Aya Eyad

March 31, 2024

Setup & Installation

To set up the Filtering and Edge Detection Studio environment, follow these steps:

1. **Install the Requirements**
   Open your terminal or command prompt and enter the following command to install the required packages: `pip install -r requirements.txt`
   This command will install the necessary libraries and dependencies for the studio environment.

2. **Run the Main Script**
   After installing the required packages, run the `main.py` file to start the Filtering and Edge Detection Studio. Use the following command: `python main.py`
   This command will execute the main script and launch the studio application.

The following libraries are included in the `requirements.txt` file and will be installed during the setup process:

- PyQt5
- pyqtgraph
- scipy
- opencv-python
- matplotlib
- numpy

<u>Note:</u> these libraries are essential for the functionalities and features of the Filtering and Edge Detection Studio.

Canny and Lines

Those Functions collectively implement the steps of the Canny edge detection algorithm, providing a comprehensive solution for detecting edges in images.

1. canny_edge Function:

Parameters:

- img (np.ndarray): The input image.
- is_gray (bool, optional): Indicates whether the image is grayscale.

Main Points:

- ➢ This function applies the Canny edge detection algorithm to an image. It first optionally converts the image to grayscale if it's not already in grayscale.
- ➢ Then it applies a Gaussian filter, calculates gradient magnitude and direction, performs non-maximum suppression, double thresholding, and finally hysteresis thresholding to detect edges.
- ➢ The low and high thresholds for double thresholding are determined based on the maximum pixel intensity of the image. The low threshold is set to a percentage (0.02 by default) of the maximum intensity, while the high threshold is set to a percentage (0.5 by default) of the maximum intensity.

Observation:

The function provides flexibility by allowing users to input both grayscale and color images, simplifying the pre-processing step.

2. non_maximum_suppression Function:

Parameters:

- gradient_magnitude (np.ndarray): The gradient magnitude image.
- gradient_direction (np.ndarray): The gradient direction image (in radians).

Main Points:

This function performs non-maximum suppression on the gradient magnitude image. It suppresses all gradient values except the local maximum along the direction of the gradient.

Observation:

The function efficiently handles edge thinning by preserving only the strongest gradient values along the edge direction.

3. double_threshold Function:

Parameters:

- image (np.ndarray): The input image.
- LowThreshold (int): The low threshold value.
- high_threshold (int): The high threshold value.

Main Points:

This function applies double thresholding to the image, categorizing pixels as strong, weak, or non-edge pixels based on their intensity values.

Observation:

The function provides a tunable mechanism for distinguishing between strong and weak edges, allowing for flexibility in edge detection sensitivity.

4. hysteresis Function:

Parameters:

- image (np.ndarray): The input image.
- weak (int, optional): The weak threshold value. Defaults to 70.
- strong (int, optional): The strong threshold value. Defaults to 255.

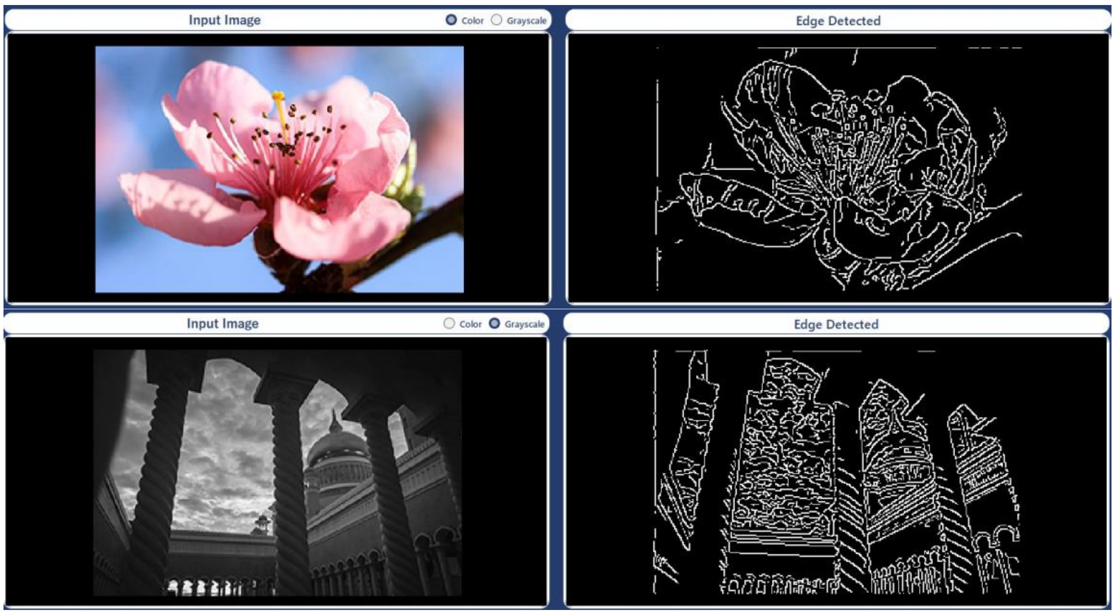
Main Points:

This function applies hysteresis thresholding to the image, determining the final edge pixels by connecting strong edge pixels with weak edge pixels.

Observation:

The function effectively removes isolated weak edges while retaining strong edges, enhancing the robustness of edge detection.

Experiments:



Conclusion for Canny:

- ✓ Canny edge detection excels due to its accurate edge localization, subpixel precision, and low error rate, minimizing false positives and negatives.
- ✓ Its ability to produce thin edges and robustness to noise makes it suitable for various image types.
- ✓ Overall, the Canny edge detection algorithm strikes a balance between accuracy, robustness, and computational efficiency, making it a preferred choice for various computer vision applications.

Hough lines Detection

Those functions collectively implement the Hough Transform-based line detection algorithm, enabling accurate detection and visualization of lines in images.

1. line_detection Function:

Parameters:

- image (numpy.ndarray): Input image in BGR format.
- th_low (int): Lower threshold for the Canny edge detector.
- th_high (int): Upper threshold for the Canny edge detector.

Main Points:

- ➢ This function detects lines in an image using the Hough Transform.
- ➢ It first converts the image to grayscale, applies Gaussian blur, and then performs Canny edge detection.
- ➢ Then, it constructs a Hough accumulator array and iterates through edge pixels to update the accumulator based on possible lines.

Observation:

The function efficiently detects lines in an image by leveraging the Hough Transform, providing a robust method for line detection.

2. hough_peaks Function:

Parameters:

- acc (numpy.ndarray): The Hough accumulator array.
- peaks (int): Number of peaks to find.
- neighborhood_size (int): Size of the neighborhood around each peak.

Main Points:

- ➢ This function identifies peaks in the Hough accumulator array, representing potential lines in the image.
- ➢ It iteratively finds the maximum value in the accumulator, marks it as a peak, and suppresses nearby values to find distinct peaks.

Observation:

The function efficiently extracts significant lines from the Hough accumulator, allowing for accurate line detection.

3. hough_lines_draw Function:

Parameters:

- img (numpy.ndarray): Input image.
- peaks_indices (list): List of peak indices.
- rhos (numpy.ndarray): Array of rho values.
- thetas (numpy.ndarray): Array of theta values.

Main Points:

- ➢ This function draws lines on the input image based on the detected peaks from the Hough Transform.

- It computes the line parameters (rho and theta) from the peak indices and draws corresponding lines on the image.

Observation:

The function visualizes detected lines on the image, providing a clear representation of the detected features.

4. hough_lines Function:

Parameters:

- img (numpy.ndarray): Input image.
- peaks (int): Number of peaks to find in the Hough accumulator array.
- neighborhood_size (int): Size of the neighborhood around each peak.
- th_low (int): Lower threshold for the Canny edge detector.
- th_high (int): Upper threshold for the Canny edge detector.

Main Points:

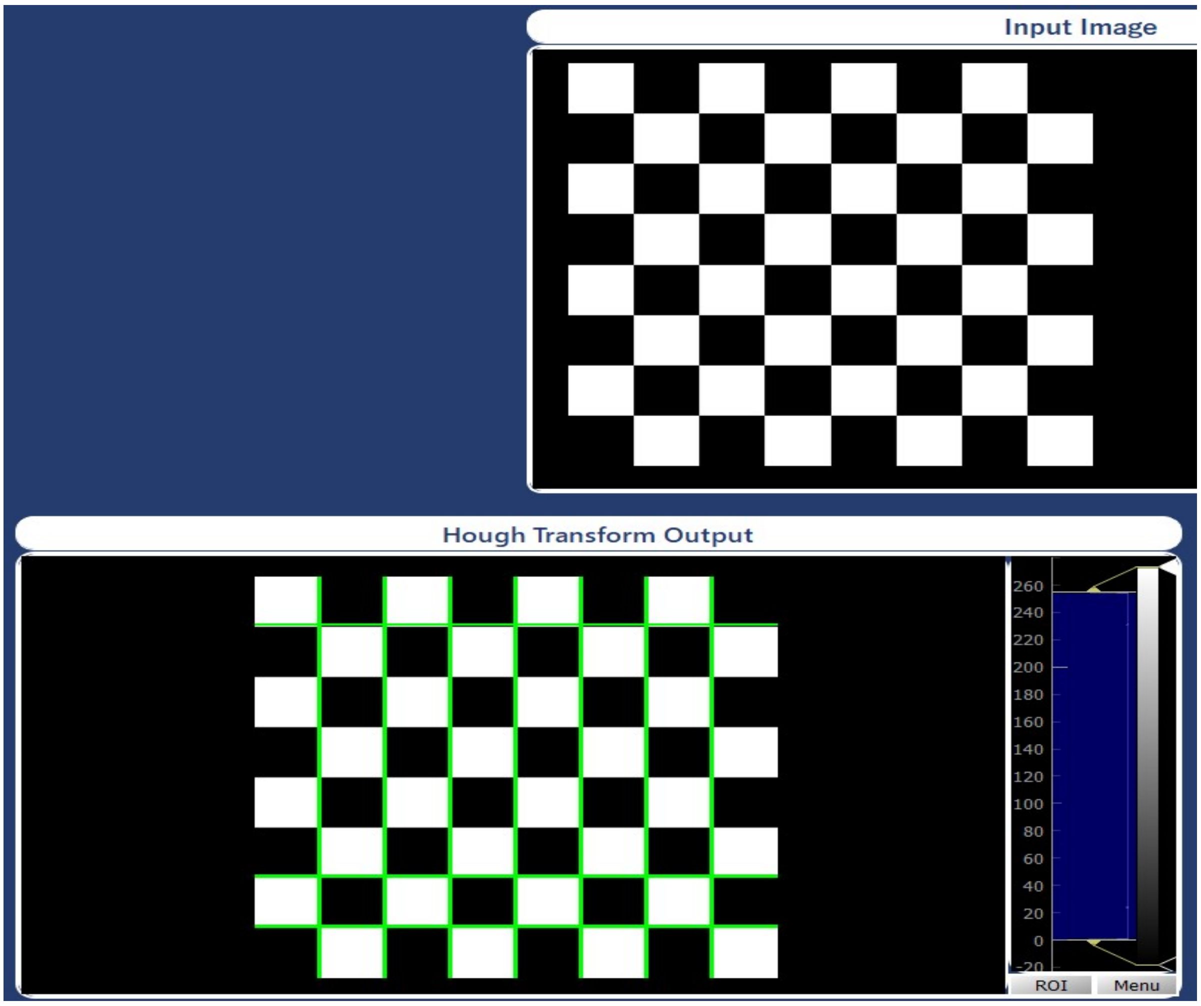
This function orchestrates the entire line detection process by calling other functions and returning the image with detected lines drawn on it.

Observation:

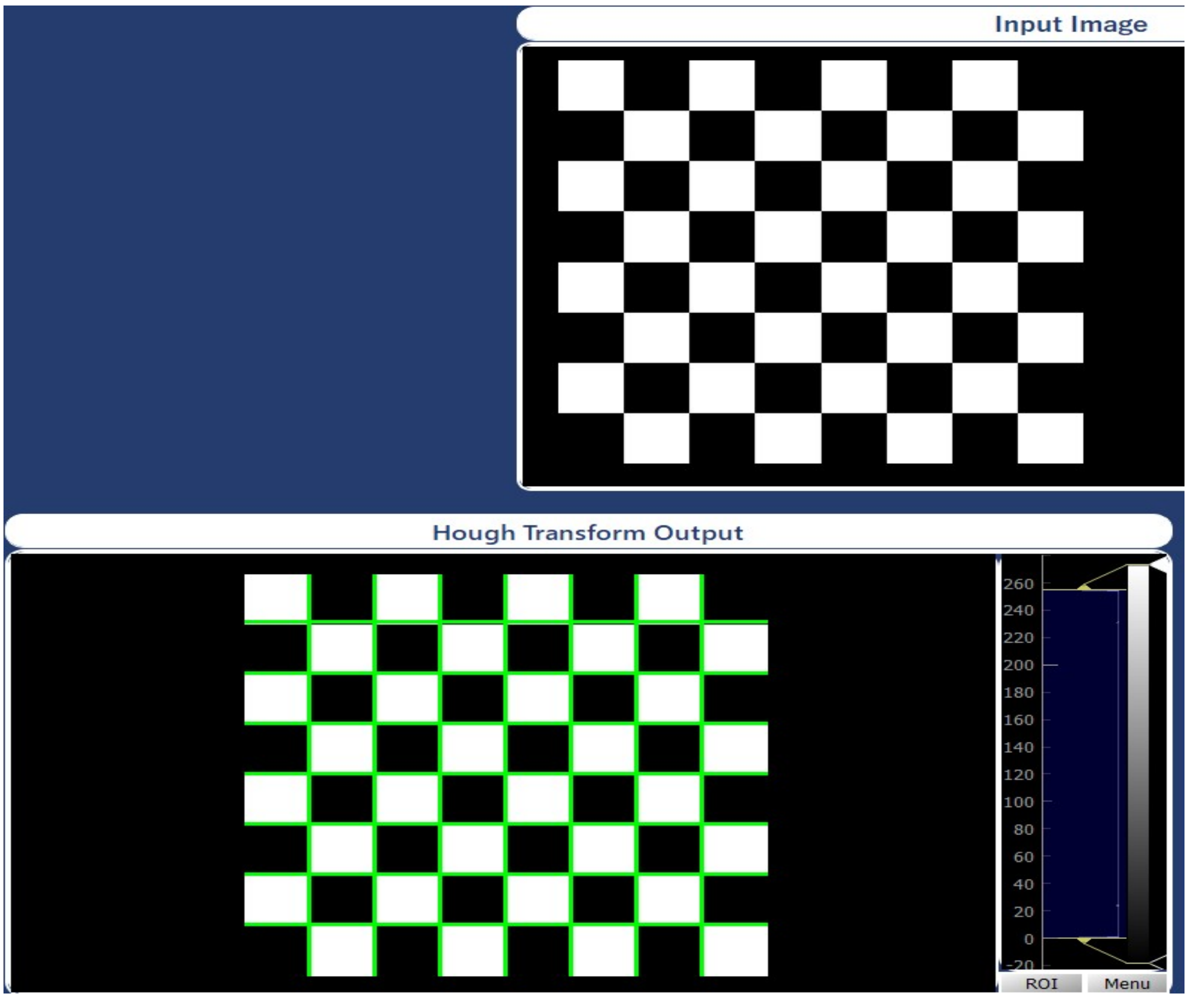
The function provides a convenient interface for line detection, encapsulating the entire process in a single function call.

Experiments:

- **Apply on image with 10 votes (No. of Peaks):**



- **Apply on image with 14 votes (No. of Peaks):**



Hough Circle Detection

1. Pre-processing:

Before passing the image to the find_hough_circles function, it undergoes pre-processing steps, which typically involve filtering and edge detection. These steps aim to enhance the image's features relevant to circle detection and reduce noise, thereby improving the accuracy of the Hough Circle detection algorithm.

1. **Filtering:**
   Filtering techniques such as **Gaussian Blur** are commonly applied to smooth the image and reduce noise. Gaussian Blur helps in removing high-frequency components from the image, which can interfere with edge detection and circle detection algorithms.

2. **Edge Detection:**
   Edge detection algorithms, such as the **Canny edge detector**, are employed to highlight regions in the image where sharp intensity changes occur. By detecting edges, the algorithm identifies potential boundaries of objects, including the edges of circles. This step is crucial for accurately detecting circles using the Hough Transform method.

Overall, the pre-processing steps play a vital role in preparing the image for circle detection, ensuring that the subsequent algorithm operates on a cleaner and more informative representation of the image content.

2. hough_lines Function:

Parameters:

- image (numpy.ndarray): Input image.
- edge_image (numpy.ndarray): Edge-detected image.
- r_min (int): Minimum radius of the circles to be detected.
- r_max (int): Maximum radius of the circles to be detected.
- delta_r (int): Step size for radius values between r_min and r_max.
- num_thetas (int): Number of steps for theta (angle).
- bin_threshold (float): Threshold for the percentage of votes required to consider a circle.
- post_process (bool, optional): Flag to enable post-processing of detected circles. Defaults to True.

Description:

The function find_hough_circles detects circles in an image using the Hough Transform method. It takes an input image and its edge-detected version, along with various parameters defining the search space for circles and the threshold for circle detection.

Main Points:

- ➢ Circle Candidate Generation: The function generates a list of circle candidates by iterating over possible radius and theta values. Each candidate is represented as a tuple containing the radius r and its (x, y) coordinates calculated using the parametric equations of a circle.
- ➢ Voting: For each edge pixel in the edge-detected image, the function votes for potential circle candidates that pass through that pixel. It updates an accumulator with the count of votes for each circle candidate.
- ➢ Thresholding: After voting, the function selects circles with a vote percentage above the specified threshold (bin_threshold). These circles are considered as detected circles.

- Post-processing: Optionally, the function performs post-processing to remove duplicate or overlapping circles based on a pixel threshold.
- Output: The function returns the output image with detected circles drawn and a list of detected circles, each represented by a tuple containing the (x, y) coordinates of the circle center, its radius, and the vote percentage.

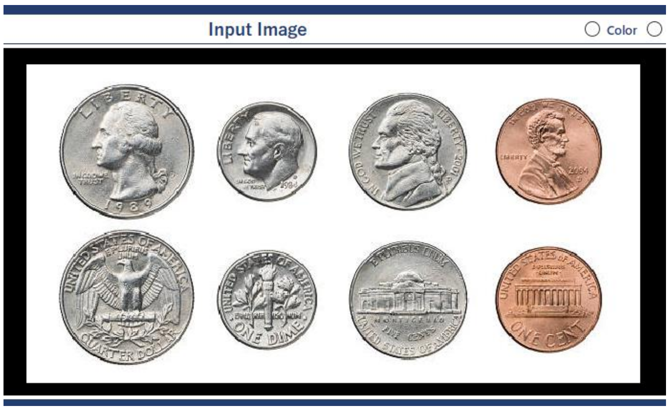
Observations:

- The Hough Circle Detection method implemented in the function provides a robust approach to detect circles of varying radii in an image.
- The function offers flexibility through parameters such as r_min, r_max, and bin_threshold, allowing users to fine-tune the detection process based on specific requirements.
- Post-processing helps in refining the detected circles, ensuring accurate detection even in noisy images or cases with overlapping circles.
- Performance may vary based on the size of the input image, the complexity of the circle patterns, and the parameter settings. Adjustments may be necessary for optimal performance in different scenarios.

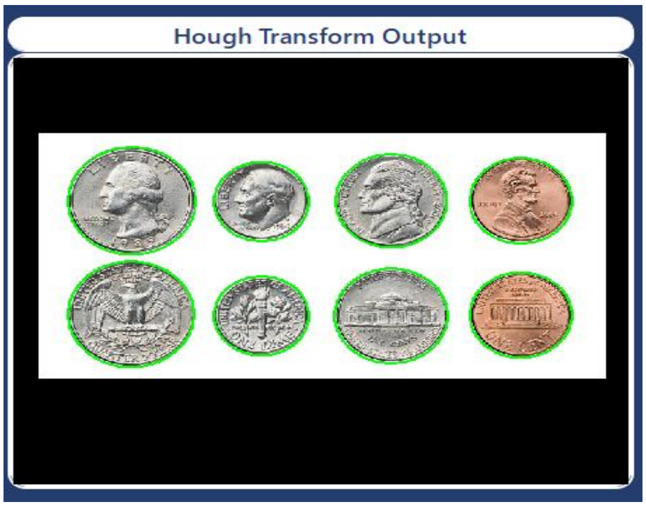
Experiment:

- Image with size (612, 317)
- Applying:

  r_min = 50, r_max = 90
  delta_r = 1
  num_thetas = 100
  bin_threshold = 60%
  kernel_size = (17, 17)



$\longrightarrow$



Conclusion:

In conclusion, the “find_hough_circles” function provides an effective method for detecting circles in images using the Hough Transform technique. By iterating over potential circle candidates and voting for those that align with edge pixels in the edge-detected image, the function identifies circles with varying radii.

Hough Ellipse Detection

1. find_hough_ellipses

Parameters:

- image (numpy.ndarray): Input image.
- edge_image (numpy.ndarray): Edge-detected image.
- a_min (int): Minimum value of semi-major axis.
- a_max (int): Maximum value of semi-major axis.
- b_min (int): Minimum value of semi-minor axis.
- b_max (int): Maximum value of semi-minor axis.
- delta_a (int): Step size for semi-major axis.
- delta_b (int): Step size for semi-minor axis.
- num_thetas (int): Number of steps for theta.
- bin_threshold (float): Threshold for voting percentage.
- post_process (bool, optional): Perform post-processing. Defaults to True.

Main Points:

- ➢ The function "find_hough_ellipses" detects ellipses in an image using the Hough Transform technique.
- ➢ It takes an input image and an edge-detected image as mandatory inputs, along with various parameters to control the detection process.
- ➢ Ellipses are parameterized by their semi-major and semi-minor axes (a and b), which are sampled within specified ranges (a_min, a_max, b_min, b_max) with step sizes (delta_a, delta_b).
- ➢ The Hough Transform algorithm iterates over potential ellipse candidates and votes for those that align with edge pixels in the edge-detected image.
- ➢ A threshold (bin_threshold) is applied to select detected ellipses based on the percentage of votes relative to the total number of theta steps (num_thetas).
- ➢ Post-processing steps can be optionally performed to refine the detected ellipses, such as removing duplicates or overlapping detections.

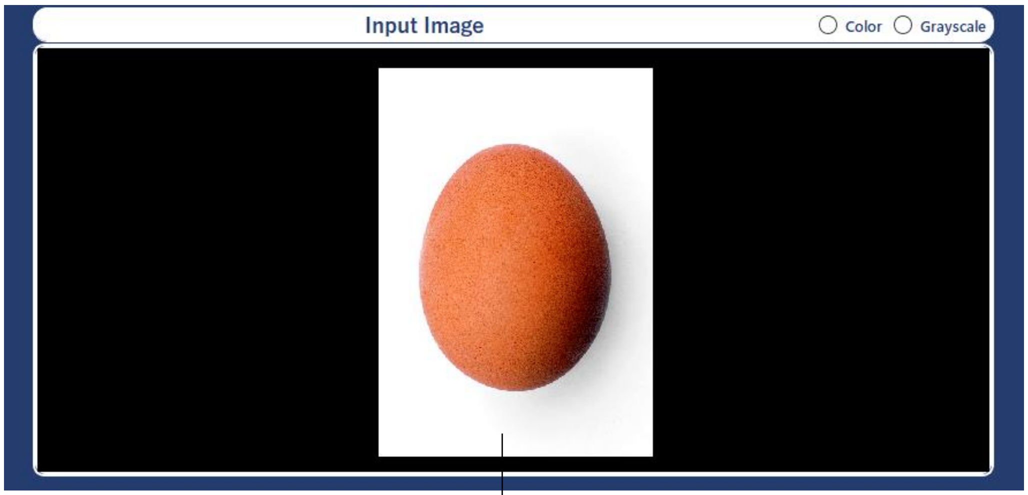
Observations:

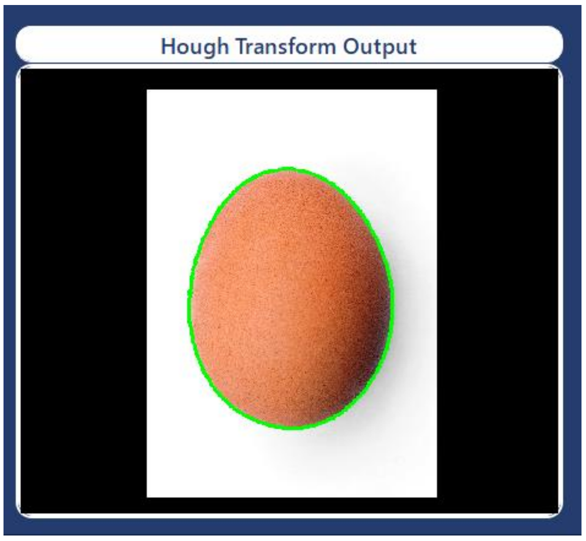
- The function efficiently processes the input image and identifies ellipses with varying sizes and orientations.
- Adjusting the parameters such as a_min, a_max, b_min, b_max, and bin_threshold allows fine-tuning the detection process based on specific image characteristics and requirements.
- Post-processing helps improve the accuracy of detected ellipses by removing redundant detections and refining the results.

Experiment:

- ➢ Image with size (500, 500)
- ➢ Applying:

  a_min = 1, a_max = 5, delta_a = 1
  b_min = 1, b_max = 5, delta_b = 1
  num_thetas = 100
  kernel_size = (3, 3)
  bin_threshold = 98%





Conclusion:

In conclusion, the find_hough_ellipses function provides a robust and flexible method for detecting ellipses in images. By leveraging the Hough Transform technique and incorporating adjustable parameters and post-processing steps, it offers accurate and reliable ellipse detection capabilities. This function is particularly useful in various applications, including object recognition, medical imaging, and computer vision tasks.

Active Contour Model (Snake Algorithm)

1. create_initial_contour function

Parameters:

- source: Input image as a numpy array.

Description:

Generates an initial contour for active contour models.
Uses a circular contour with a specified number of points.
The center of the circle is the center of the image, and the radius is 40% of the minimum dimension of the image.

2. internal_energy function

Parameters:

- contour_points: Array of shape (N, 2) containing the coordinates of N points defining the initial contour.
- alpha: Weight parameter controlling the contribution of elasticity to the internal energy.
- beta: Weight parameter controlling the contribution of stiffness (curvature) to the internal energy.

Description:

Calculates the internal energy of a contour defined by a set of points using the following formula:

$$E_{internal} = \alpha * \sum_{i=0}^{N-1} \left(\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} - \overline{d} \right)^2 + \beta * \sum_{i=0}^{N-1} (x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2$$

$where\ \overline{d}\ is\ the\ mean\ of\ the\ euclidean\ distance\ between\ consecutive\ points$

- Consists of elasticity and stiffness (curvature) components.
- Elasticity (first term) is based on the difference between distances of consecutive points from the mean distance.
- Stiffness (second term) is based on the curvature of the contour.

Observations:

Balancing alpha and beta is crucial for proper contour convergence. Higher values of alpha prioritize elasticity, while higher values of beta prioritize contour smoothness.

3. prepare_external_energy function

Parameters:

image: Input image as a numpy array.

Description:

Prepares external energy components for active contours (intensity and edges).
Applies Gaussian blur to the input image to reduce noise.

Computes edges/edge map using the Sobel filter to highlight object boundaries.

Observations:

- Blurring helps in smoothing the image, which can improve the accuracy of edge detection.
- Sobel edge detection emphasizes gradient changes, which are indicative of object boundaries.

4. external_energy function

Parameters:

- point: Coordinates of the point (x, y).
- intensity: Intensity values of the image.
- gradient: Gradient magnitude of the image.
- w_line: Weight for intensity term.
- w_edge: Weight for gradient term.
- gamma: Scaling factor.

Description:

- Gets external energy at a given point based on intensity and gradient information.
- Combines intensity and gradient energies with specified weights.

Observations:

- gamma adjusts the overall influence of external energy on the contour evolution.
- Proper selection of w_line and w_edge balances the contribution of intensity and gradient information to the contour evolution.

5. perform_active_contour function (main process)

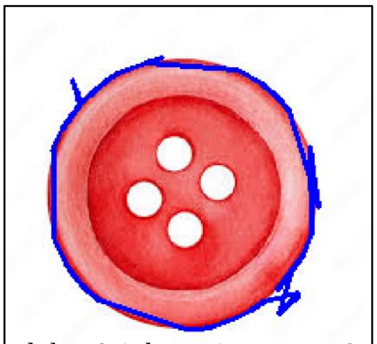
Description and Steps:

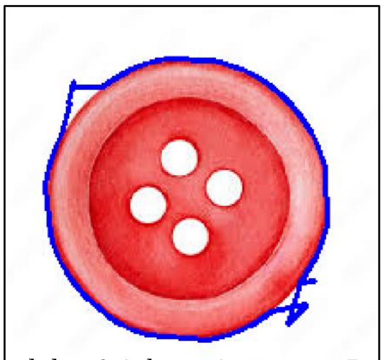
- It iterates over a specified number of iterations (termination variable), updating the contour at each iteration.
- Within each iteration, it iterates over each point in the contour and explores neighboring locations to minimize the total energy, which is a combination of internal and external energies.
- It updates the contour based on the point with minimum energy.
- Drawing the contour on the original image for visualization of the segmentation process in real-time.
- It repeats the process for the specified number of iterations.

Observations:

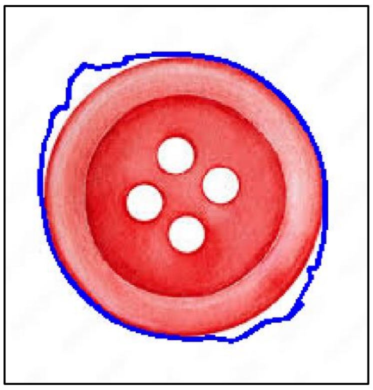
- ➢ The weights w_line and w_edge determine the influence of intensity and gradient information on the contour evolution, respectively.
- ➢ The performance of the algorithm depends mainly on the proper selection of parameters such as alpha, beta, gamma, and the number of iterations.

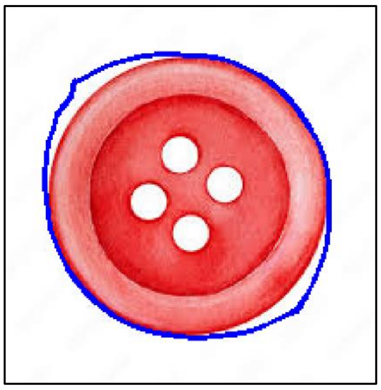
Experiments:


alpha=0.1, beta=1, gamma=2

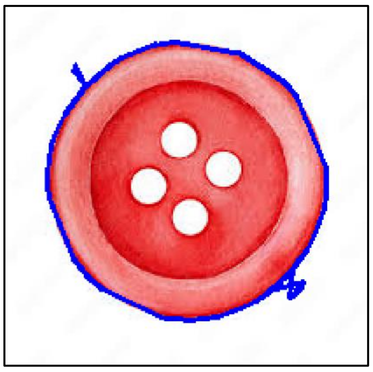

alpha=0.1, beta=1, gamma=5

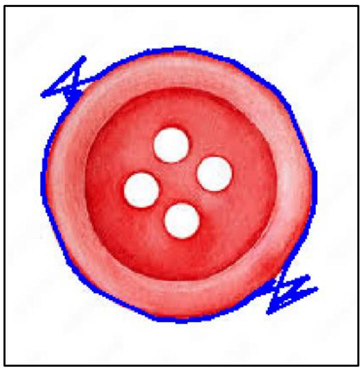
- ➢ For the same alpha and beta values decreasing gamma prioritizes contour shape over image features hence the contour misses the object’s boundaries.


alpha=0.1, beta=1, gamma=10

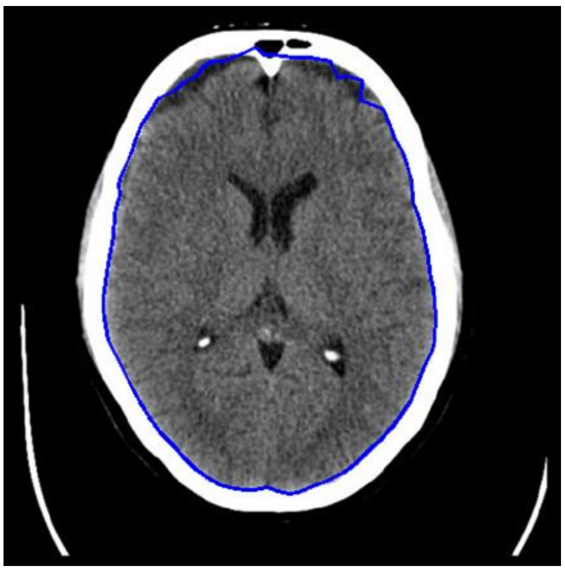

alpha=0.1, beta=4, gamma=10

- ➢ For the same alpha and gamma values increasing beta makes the contour smoother.


alpha=0.1, beta=1, gamma=4


alpha=1, beta=1, gamma=4

- ➢ For the same beta and gamma values, higher values of alpha result in a stiffer contour, it tends to maintain its shape and resist deformations caused by external forces (image features). Decreasing alpha makes the contour more flexible and allows it to deform more easily.


alpha=0.01, beta=1, gamma=10
initial contour is an ellipse

- ➢ An initial contour that has a similar shape to the object boundary can help speed up convergence.
- ➢ If the initial contour is placed far from the object, it might not capture the entire boundary. Also, if it's too close, it may miss fine details.

6. get_perimeter function

Description:

Calculate the perimeter of a polygon given its vertices using Euclidean distance.

Parameters:

pts (ndarray): A 2D ndarray of shape (no of points, 2) representing the x and y coordinates of the vertices of the polygon.

Returns:

perimeter (float): The perimeter of the polygon.

7. get_area function

Parameters:

- pts (ndarray): A 2D ndarray of shape (no of points, 2) representing the x and y coordinates of the vertices of the polygon.

Returns:

- area (float): The area of the polygon.

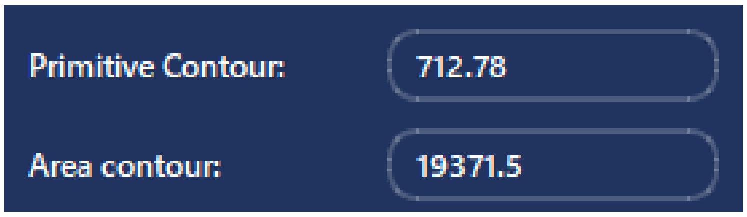
Description

Calculates the area of a polygon given its vertice.

Steps:

- Multiplying each x coordinate by the y coordinate of the next point.
- Multiplying each y coordinate by the x coordinate of the next point
- Summing these 2 products
- Subtracting the sums
- Dividing the result by 2.

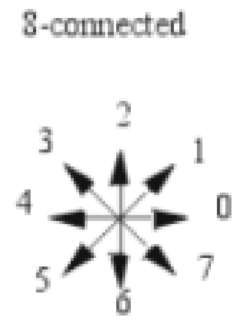
Sample Output:



```
Primitive Contour:   712.78
Area contour:        19371.5
```

Chain Code

Those functions collectively produce the chain code for the active contour produced previously, using the 8-connected chain code convention.



1. get_direction

Parameters:

- curr_pt (ndarray): a numpy array containing the x and y coords of the second pt
- prev_pt (ndarray): a numpy array containing the x and y coords of the first pt

Returns:

orient (string): carrying the orientation in one of 8 directions (N, S, E, W, NE, NW, SE, SW)

Main Points:

Compares the coordinates of 2 consecutive contour points to decide the orientation (ex: N, SE, NW) of the line segment joining them. Subtracting the x and y coordinates and checking for their signs.

2. directions_to_chain_code

Parameters:

- directions (ndarray): a 2D numpy array produced by the snake algorithm, its shape is (no of points, 2) to produce its chain code

Returns:

chain (list): A list the same size as directions carrying the corrsponding chain codes

Description:

Maps every direction (ex: E, W, SN) to its corresponding number in chain code using a list ['E', 'NE', 'N', 'NW', 'W', 'SW', 'S', 'SE'], where each direction has the index of its corresponding number in chain code.

3. print_chain_code

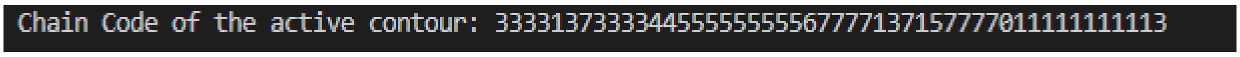
Parameters:

- pts_2d (ndarray): a numpy array containing directions (char)

Steps:

1. Loops on the whole array, comparing 2 successive points to get the direction of the line segment joining them using the get_direction function (1), appending all directions to the chain_directions list.
2. Then converting this list of directions (ex: “N”, “SW”) to their corresponding numbers in chain code using the directions_to_chain_code function.
3. Turning this list of integers to a string using the list_to_string functions which iterates on the list and appends to an empty string.
4. Printing into the console with the option to export to a txt file using the export_chain_code function.

Experiment:



```
Chain Code of the active contour: 3333137333344555555555677771371577770111111111113
```

- This is the chain code for a circular active contour of a coin, we can see that the abundance of 3, 5, 7, 1 pattern specifies north west then south west then south east and north east making a full circle.