Cairo University

Computer Vision (SBE3230)

Task 2

Name	Sec	BN
Dina Hussam Assem	1	28
Omar Ahmed Anwar	2	2
Omar Saad Mohamed	2	3
Mohamed Ahmed Ismail	2	16
Neveen Mohamed Ayman	2	49

Supervised By

Dr. Ahmed Badwy

TA: eng. Peter Emad & eng. Lila Abbas

Table of Contents

Introduction*:*

It’s an image processing implementation functions project implemented in C++ with a desktop application (Qt).

Algorithms Implemented and How They Work:

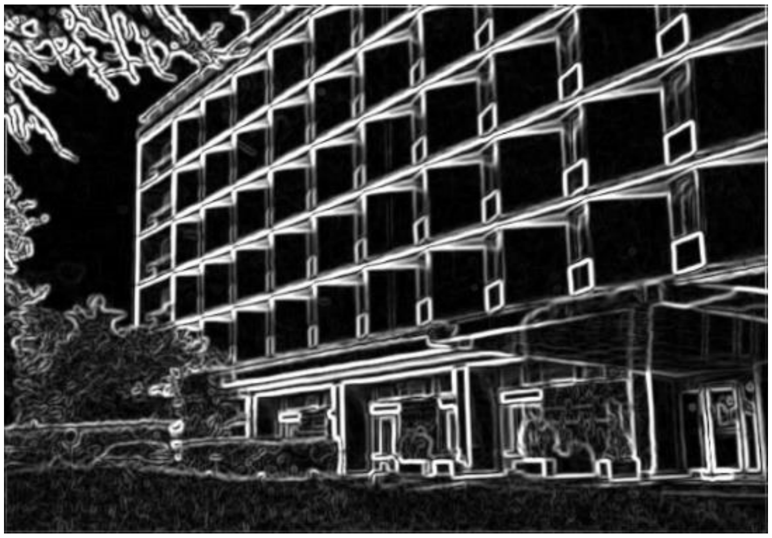
1) Edge Detection:

Canny detection: Canny edge detector is probably the most used and most effective Edge Detection Algorithms.

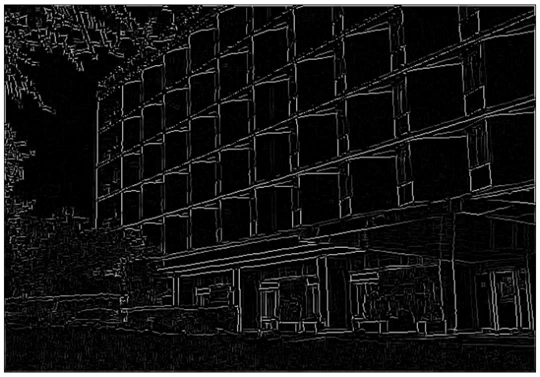
It is a multi-stage algorithm:

1. Gaussian blur: First, the input image is smoothed using a Gaussian filter to remove noise and small details from the image.

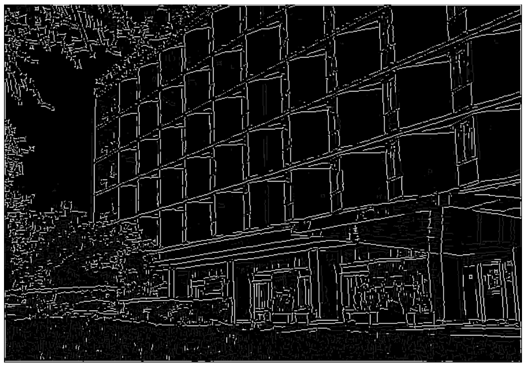
2. Gradient calculation: Next, the edges in the image are identified by calculating the gradient of the image. The gradient magnitude and gradient phase (direction) are calculated at each pixel using Sobel gradient operators.



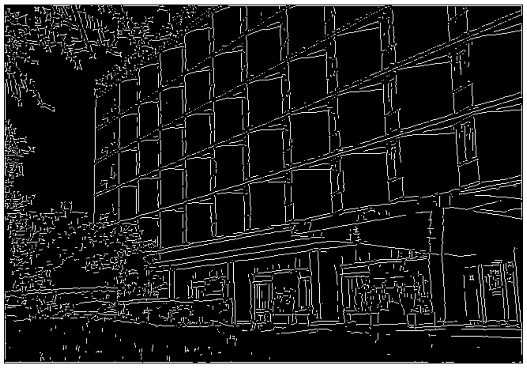
3. Non maximum suppression: In this stage, the algorithm goes through each pixel in the gradient magnitude image and suppresses non-maximum values in the direction of the gradient. The purpose of this step is to thin out the edges so that only the most prominent ones are preserved.



4. Double thresholding: The gradient magnitude image is then thresholded into two levels: a low threshold and a high threshold. Pixels with gradient magnitudes above the high threshold are considered strong edges, pixels below the low threshold are considered non-edges, and pixels with magnitudes between the two thresholds are considered weak edges.



5. Edge tracking by hysteresis: Finally, weak edges that are connected to strong edges are also considered as edges. This is done by tracing along the edges using a process called hysteresis. If a weak edge pixel is adjacent to a strong edge pixel, it is considered part of the edge, and if it is not adjacent to a strong edge pixel, it is considered noise and is discarded.



2) Hough Transformation: ---- Line, Circle, and Ellipse Detection ----

Introduction

The Hough Transform is a popular feature extraction technique used in image processing and computer vision to detect shapes within an image. It is particularly effective in detecting lines, circles, and ellipses even in the presence of noise or partial shapes.

Hough Transform

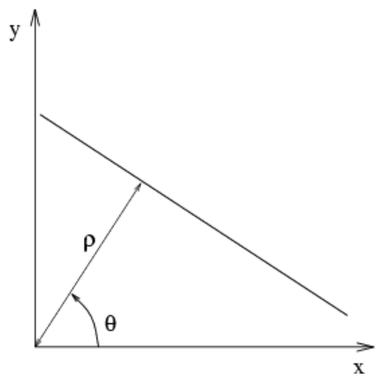
The Hough Transform is a voting-based method that maps points in the image space to corresponding points in the parameter space, also known as Hough Space. The key idea is to identify patterns or structures within an image by examining the parameter space rather than the image space, as the former often reveals simpler relationships.

Line Detection

The Hough Transform for line detection is based on the polar representation of a line, expressed as:

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

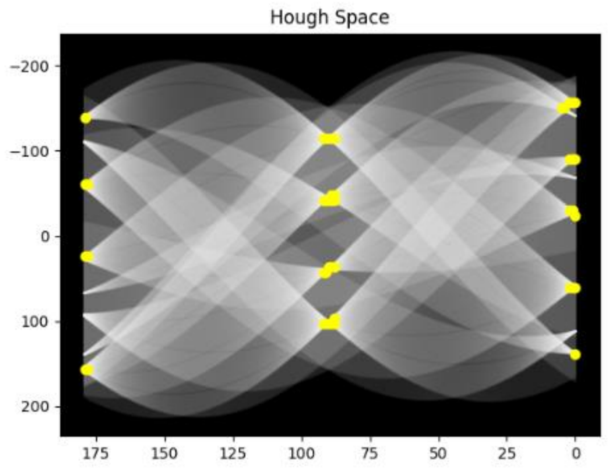
Where ρ is the distance from the origin to the closest point on the line, and θ is the angle between the positive x-axis and the line connecting the origin with that closest point.



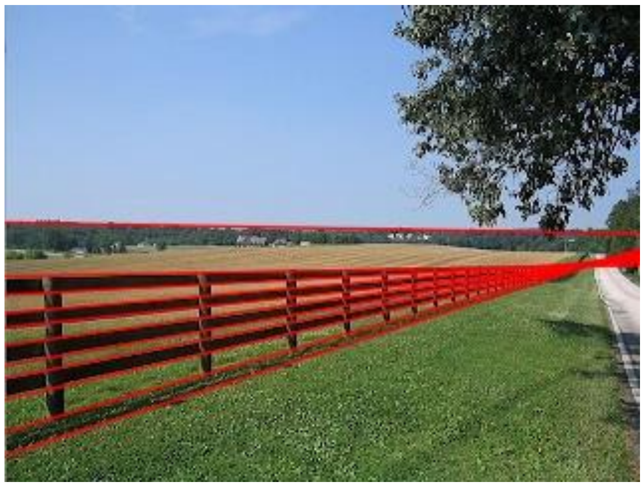
To apply the Hough Transform for line detection, the following steps are taken:

1. Edge detection: Apply an edge detection algorithm (Canny) to the input image to obtain a binary edge image.

2. Hough Space: Create an accumulator matrix representing the parameter space (ρ, θ).
3. Voting: For each edge pixel (x, y) in the binary image, compute the corresponding ρ values for a range of θ values and increment the accumulator matrix cells.
4. Peak detection: Loop over the accumulator and compare with the given threshold then Identify the local maxima in the accumulator matrix. Each local maximum corresponds to a line detected in the image.



5. Convert the detected points from accumulator in Hough space (ρ, θ) back to image space and draw the lines.



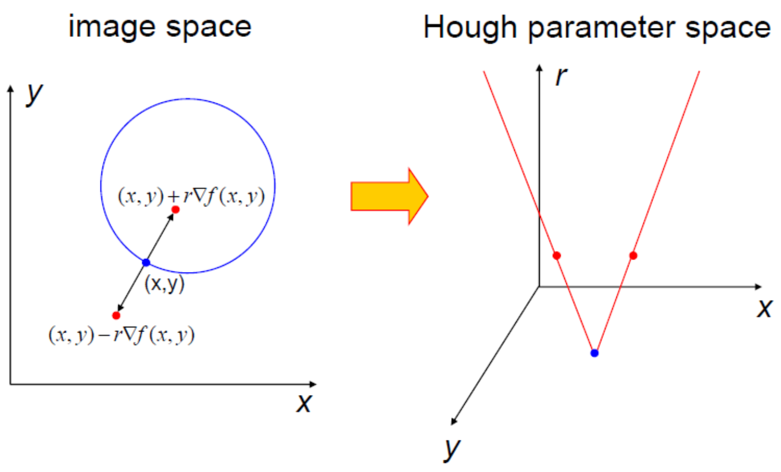
Circle Detection

The Hough Transform for circle detection uses the circle equation in the Cartesian coordinate system: $(x-a)^2 + (y-b)^2 = r^2$
Where (a, b) are the coordinates of the circle's center and r is the radius of the circle.

The Hough Transform for circle detection involves the following steps:

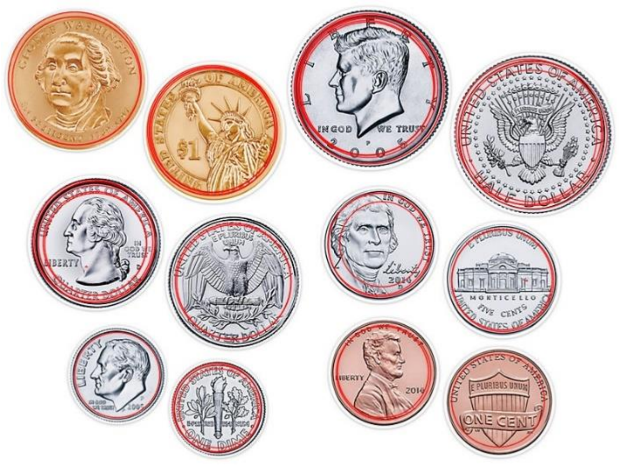
1. Edge detection: Apply an edge detection algorithm (e.g., Canny) to the input image to obtain a binary edge image.

2. Hough Space: Create a 3D accumulator matrix representing the parameter space (a, b, r).



3. Voting: For each edge pixel (x, y) in the binary image, compute the corresponding (a, b) values for a range of r and θ (1:360) values and increment the accumulator matrix cells.
4. Peak detection: Loop over the accumulator and compare with the given threshold then Identify the local maximum points which corresponds to a circle detected in the image.

5. From the detected points from accumulator in Hough space (a, b, r) draw circles with center (a, b) and radius r in image space.



Ellipse Detection

The Hough Transform can be used to detect ellipses in an image by extending the classic Hough Transform to a 5-dimensional parameter space, where the parameters are the center coordinates **(x,y)**, the major and minor axis lengths **(a,b)**, and the orientation angle (theta) of the ellipse.

The Hough Transform for circle detection involves the following steps:

1. We first convert input image to gray scale.
2. Then detect edges in the input image using edge detection techniques such as Canny edge detector or Sobel operator.
3. Create an accumulator array in the 5-D Hough parameter space for detecting ellipses. This accumulator array is a 5D matrix that stores the number of votes for each possible ellipse parameter.

4. For each edge point in the binary edge map, calculate the set of possible ellipse parameters that pass through that point using the following equation

$$(x - xc)^2/a^2 + (y - yc)^2/b^2 = 1$$

where **(x,y)** is the edge point coordinates, **(xc, yc)** is the center of the ellipse, and **(a,b)** are the major and minor axis lengths. The orientation angle theta can be computed from the gradient direction at the edge point.

5. Voting: The algorithm then goes through each edge point and votes for the parameters that could represent an ellipse. For each edge point, the algorithm calculates the parameters **(a,b)** for all possible ellipses passing through that point. The corresponding point in the Hough space is then incremented.
6. Thresholding: Once voting is complete, the Hough space is thresholded to remove false positives. This is done by setting a threshold on the number of votes required for a point in the Hough space to be considered as a potential ellipse.
7. Finally, the remaining points in the Hough space that exceed the threshold are considered as potential ellipses. The algorithm then extracts the parameters **(a,b)** for each potential ellipse and draws the ellipse on the original image.

3) Active Contour Model:

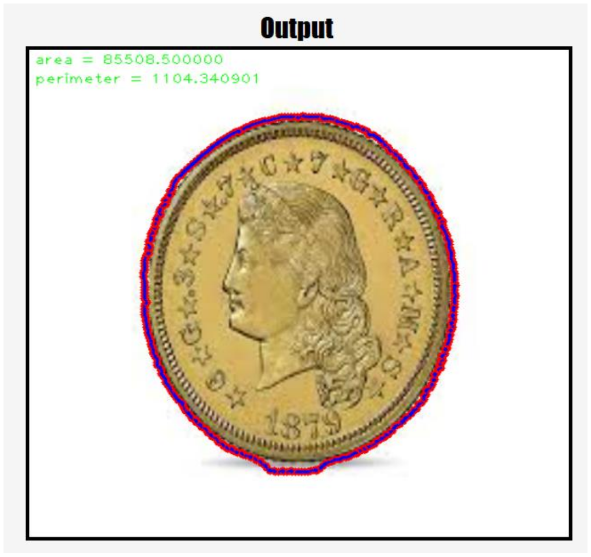
Snake algorithm is for image segmentation. The algorithm works by iteratively moving a curve to fit the object boundary in the image.
The Active Contour Model algorithm is a powerful method for segmenting objects in an image, particularly when the object boundary is complex and irregular.

Algorithm

- The main function, "active Contour_Model", takes an input image, a center point, a radius, and several parameters for the algorithm. It first initializes a curve using the "initial_contour" function, which creates a circular curve around the center point. The "snake" function is then called repeatedly for a fixed number of iterations, which moves the curve to minimize an energy function based on the internal, external, and balloon energies. The resulting curve is then drawn on the output image and the area and perimeter of the segmented object are displayed.
- The internal energy is calculated using the "calcInternalEnergy" function, which computes the curvature of the curve at a point. The external energy is calculated using the "calcExternalEnergy" function, which measures the difference between the intensity of the image at the point and a constant value. The balloon energy is calculated using the "calcBalloonEnergy" function, which penalizes deviations from the initial curve shape.

$$E_{internal}\big(v(s)\big) = \propto \left|\frac{dv}{ds}\right|^2 + \beta \left|\frac{d^2v}{d^2s}\right|^2 \qquad E_{total} = E_{internal} + \gamma E_{image}$$

- The "moveCurve" function is used to move the curve in different directions and choose the one with the minimum energy. The "initial_contour" function creates a circular curve around the center point.

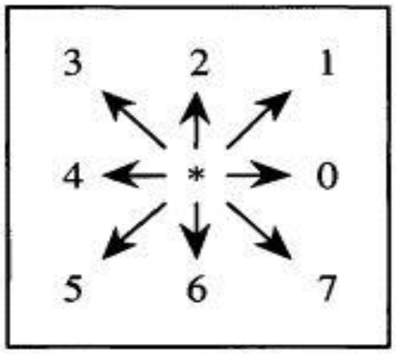


4) The Boundary Chain Code:

Introduction

The boundary chain code (BCC) is another well-known technique often used to encode the results of image segmentation. Compared with the object label map, it is a more compact way to store the image segmentation information. With such an approach, only the boundary is required to define an object, and therefore it is not necessary to store the location of interior pixels so Chain code is useful in cutting down storage relative to the number of bits required to hold a list of the coordinates of all the boundary pixels.

The chain code devised in 1961 by Herbert Freeman. This method recodes the boundary of a region into a sequence of octal digits, each one representing a boundary step in one of the eight basic directions relative to the current pixel position.



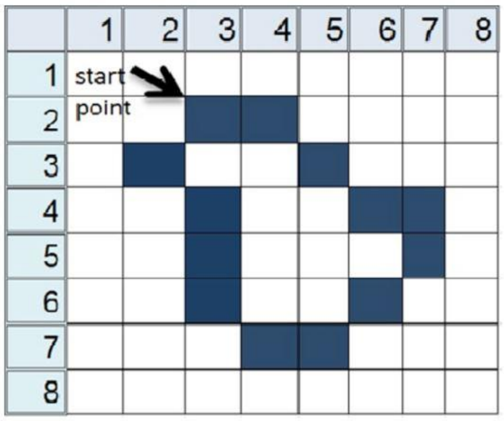
Purpose of the code

The code computes the normalized chain code of a given boundary, represented by a list of 2D points. The chain code is a compact representation of the contour of the boundary that is suitable for storage and processing.

Code structure

The code is written in C++ and is organized into several functions, each responsible for a specific task. The functions are:

1. getNeighbor: Takes a point and a direction, and returns the neighbor of the point in the given direction.
2. pixelValue: Takes an image and a point, and returns the value of the pixel at that point in the image.
3. findNextDirection: Takes a current direction and returns the next direction in the clockwise order.
4. chainCode: Takes an image and a starting point, and returns the chain code of the contour starting from that point. It uses the getNeighbor, pixelValue, and findNextDirection functions to traverse the contour in a clockwise direction, and adds each direction to a vector of chain codes.
5. findStartingPoint: Takes an image and returns a suitable starting point for the chain code algorithm. It simply returns the first non-zero pixel found.
6. normalizeContour: Takes a chain code and rotates it to a canonical representation. It finds the minimum chain code value in the contour, and rotates the contour so that the minimum value is at the beginning.
7. createImageFromBoundary: Takes a list of boundary points, creates a binary image from them, and sets the pixel values at the boundary points to 1.
8. main: The main function that puts everything together. It defines the input boundary points, creates a binary image from them, finds the starting point, calculates the chain code, normalizes it, and prints it to the console.



5) Contour Area:

Algorithm:

Substitution for the Snake Contour points in the shoelace formula.

Shoelace formula:

It’s a mathematical algorithm to determine the area of a simple polygon whose vertices are described by their Cartesian coordinates in the plane. It is called the shoelace formula because of the constant cross-multiplying for the coordinates making up the polygon, like threading shoelaces. It has applications in surveying and forestry, among other areas.

$$A = \sum_{i=1}^{n-1} \frac{1}{2}(x_i y_{i+1} - x_{i+1} y_i) + \frac{1}{2}(x_n y_1 - x_1 y_n)$$

$$= \tfrac{1}{2}[(x_1 y_2 - x_2 y_1) + (x_2 y_3 - x_3 y_2) + \cdots + (x_{n-1} y_n - x_n y_{n-1}) + (x_n y_1 - x_1 y_n)]$$

6) Contour Perimeter:

Algorithm:

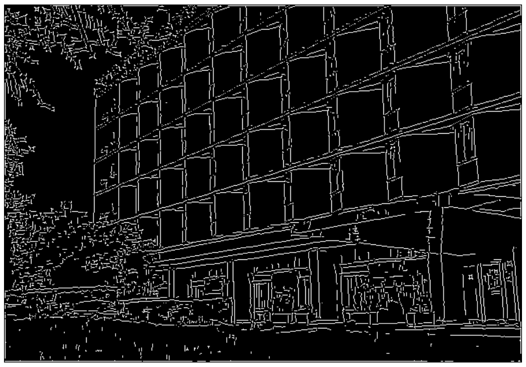
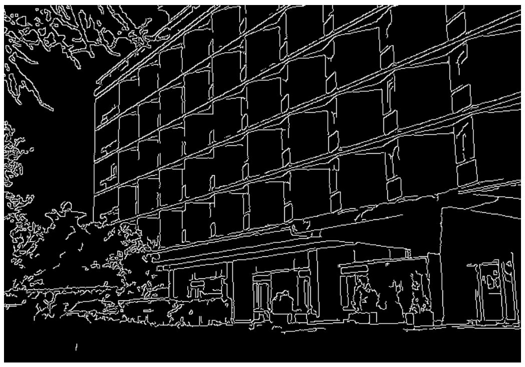
To calculate the perimeter of the shape, we calculate the sum of the distances between each two adjacent points by substituting in the distance between two points law.

$$Distance\ btween\ two\ points = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

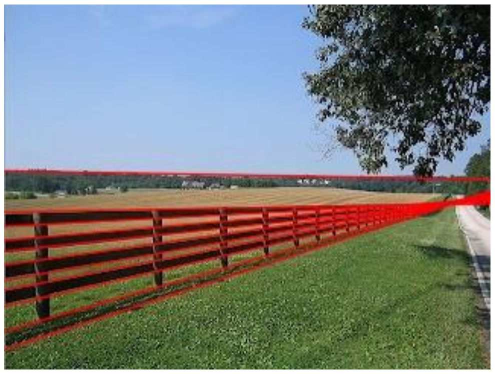
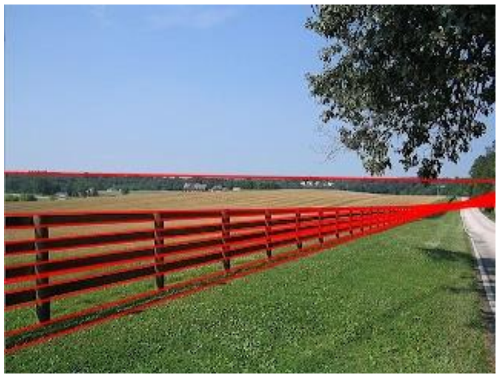
Note: the result is considered an approximation of the perimeter and the area because of the quantization error and discrete integration.

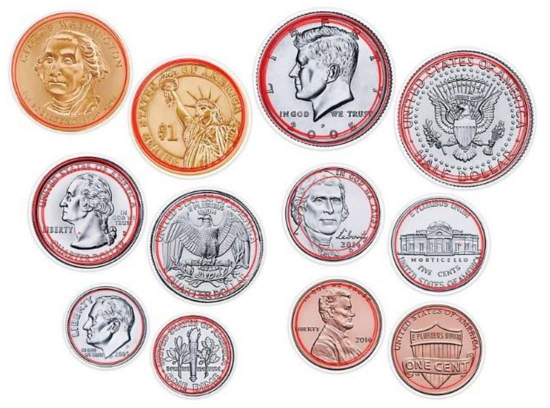
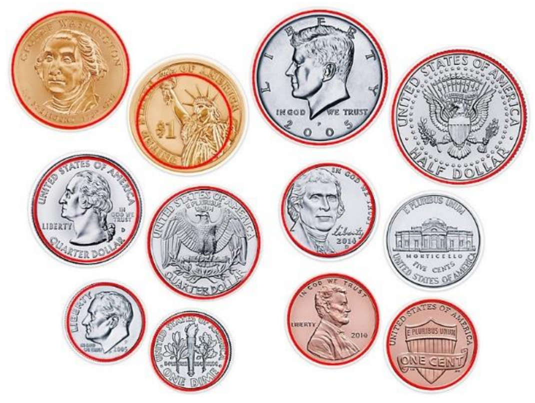
Compare to Library:

1) Canny Edge Detection:

	C++ Implementation	Library
Canny Detection		

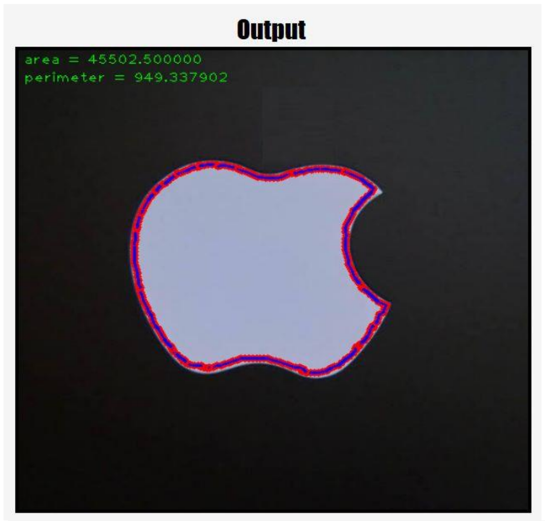
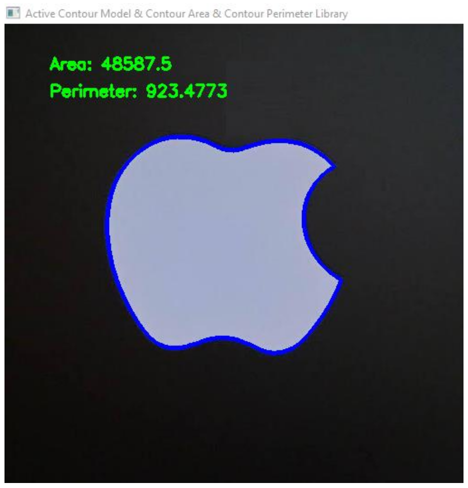
2) Hough Transformation:

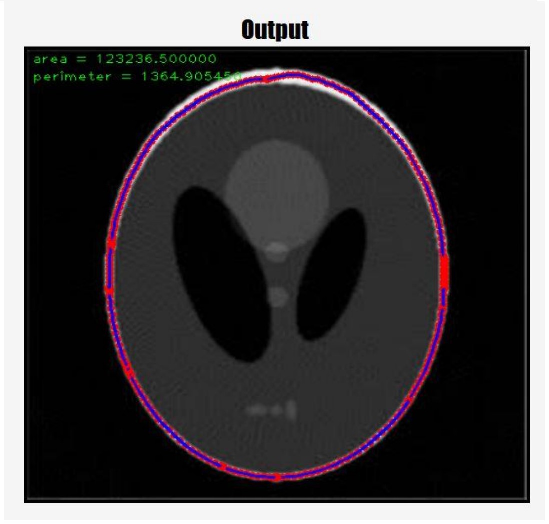
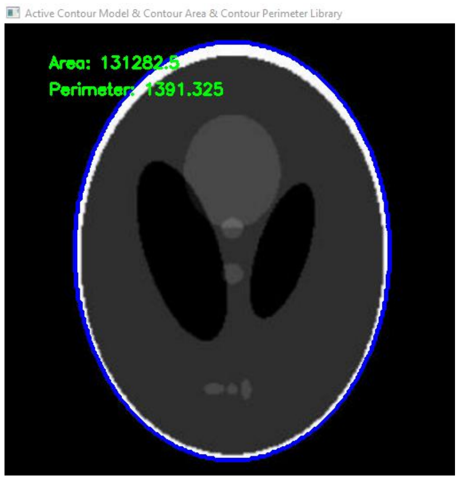
	C++ Implementation	Library
Line Detection		

Circle Detection		

3) Active Contour Model & Contour Area & Contour Perimeter:

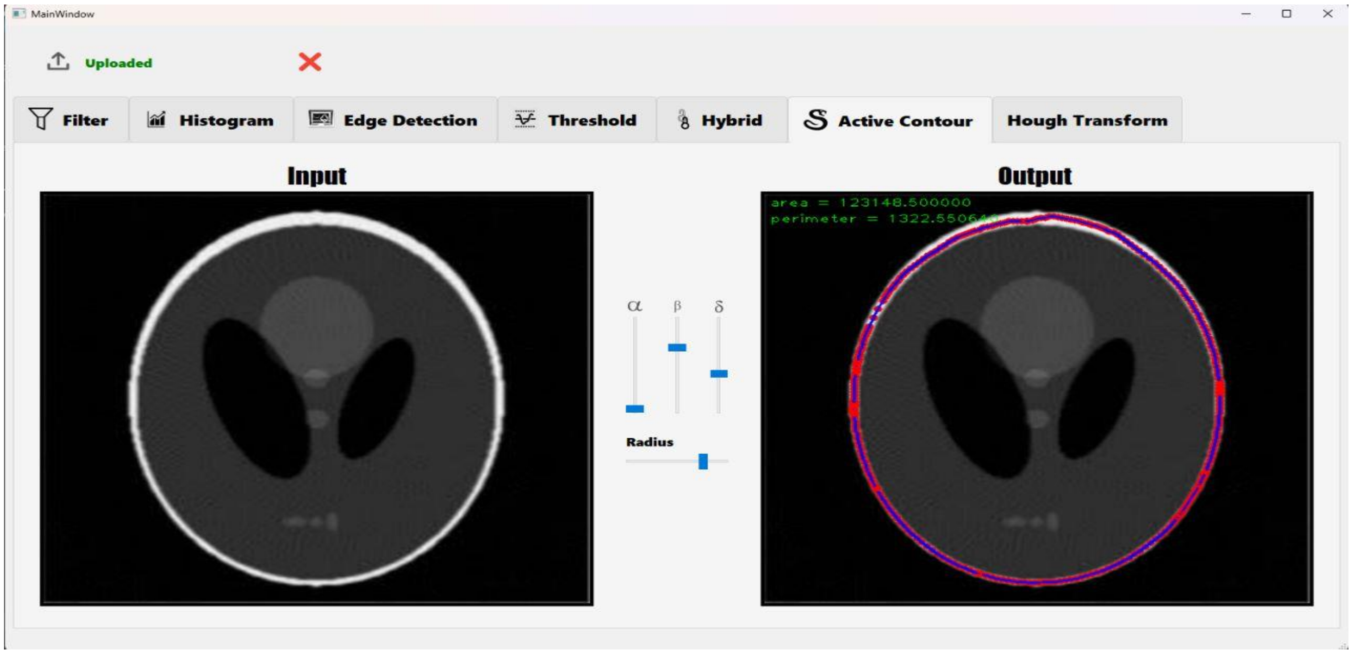
	C++ Implementation	**Library**

**Active Contour Model
&
Contour Area
&
Contour Perimeter**
	 Output
	area = 45502.500000
	perimeter = 949.337902
		 Active Contour Model & Contour Area & Contour Perimeter Library
		Area: 48587.5
		Perimeter: 923.4773

	 Output
	area = 123236.500000
	perimeter = 1364.905450
		 Active Contour Model & Contour Area & Contour Perimeter Library
		Area: 131282.5
		Perimeter: 1391.325

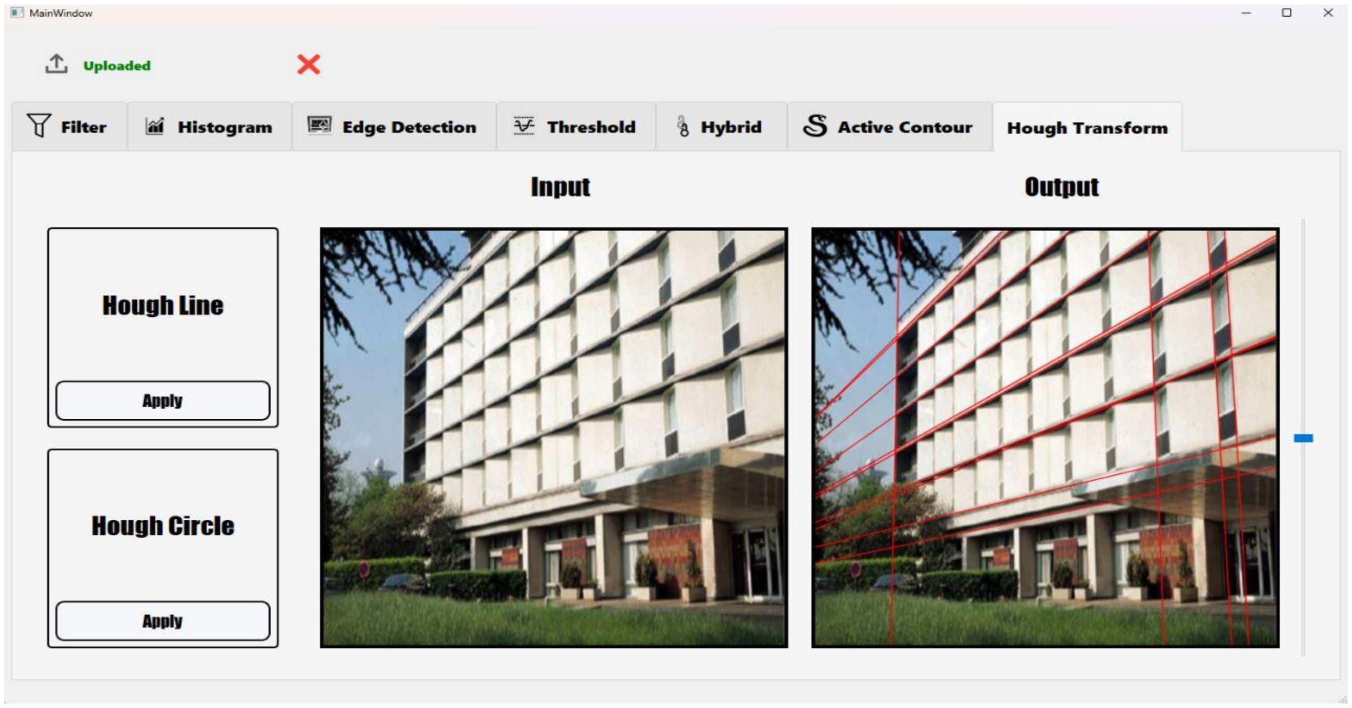
Application Screenshots:

Active Contour Tab



```
area = 123148.500000
perimeter = 1322.550640
```

Hough Tab


MainWindow
Uploaded
Filter
Histogram
Edge Detection
Threshold
Hybrid
Active Contour
Hough Transform
Input
Output
Hough Line
Apply
Hough Circle
Apply