Computer Vision: Task 2 Report

Detect edges in the image using Canny edge detection

In the previous report, we explained the 5 steps of Canny, but we did not fully implement them. Therefore, we used the built-in function from OpenCV. But now after we complete implementing them we will repeat the steps with detailed images for each step and how each step affects the final result.

The **Canny edge detector** is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.
The Canny edge detection algorithm is composed of 5 steps:

1. Noise reduction;
2. Gradient calculation;
3. Non-maximum suppression;
4. Double threshold;
5. Edge Tracking by Hysteresis.

This algorithm is based on grayscale pictures. Therefore, the prerequisite is to convert the image to grayscale before following the above-mentioned steps.

Noise Reduction:

One way to get rid of the noise on the image is by applying Gaussian blur to smooth it.

sigma = 1 and filter_size = 5 ***sigma = 5 and filter_size = 5***

we notice that when we increase ***sigma*** the final result of Canny was better.

Gradient Calculation

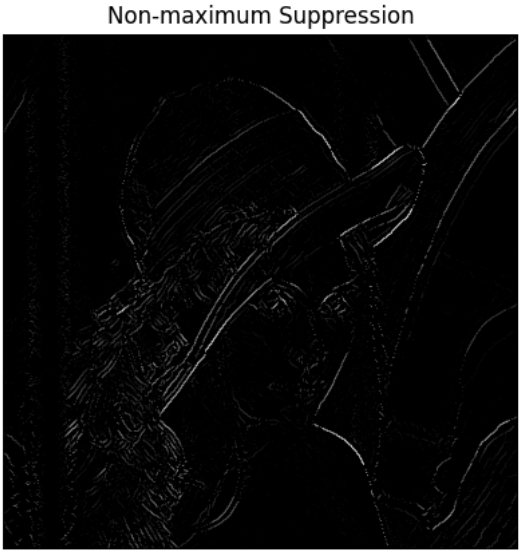
The Gradient calculation step detects the edge intensity and direction by calculating the gradient of the image using edge detection operators. Edges correspond to a change of pixels' intensity. To detect it, the easiest way is to apply filters that highlight this intensity change in both directions: horizontal (x) and vertical (y)

When the image is smoothed, the derivatives ***Gx*** and ***Gy*** w.r.t. ***x*** and ***y*** are calculated. This can be implemented by convolving ***I*** with ***Sobel*** kernels.

Non-Maximum Suppression

Ideally, the final image should have thin edges. Thus, we must perform non-maximum suppression to thin out the edges.

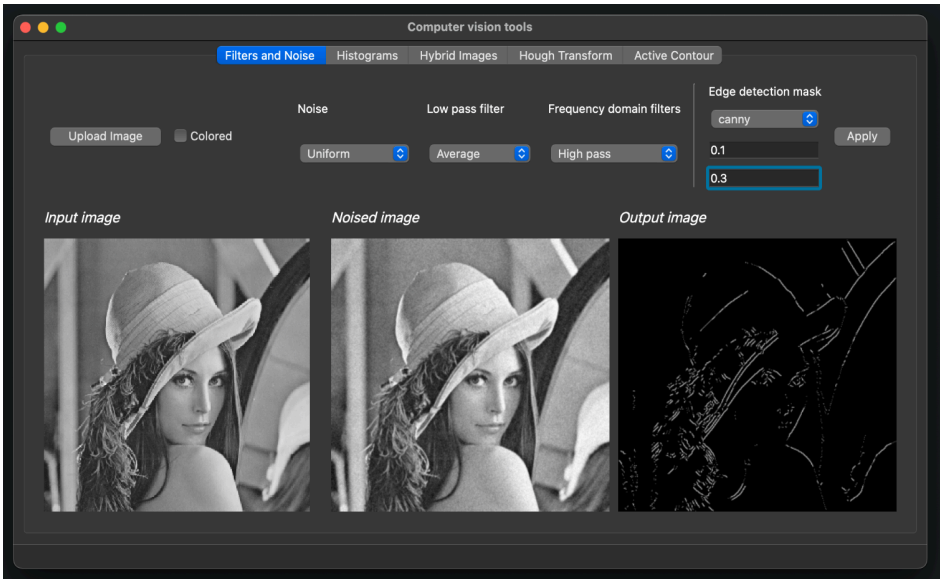
The process is that we loop over all the pixels and take two adjacent pixels of the current pixel and compare with them to find whether the intensity of the current pixel is greater than the two adjacent pixels, if so then continue, if not then set the current pixel intensity to **0**.



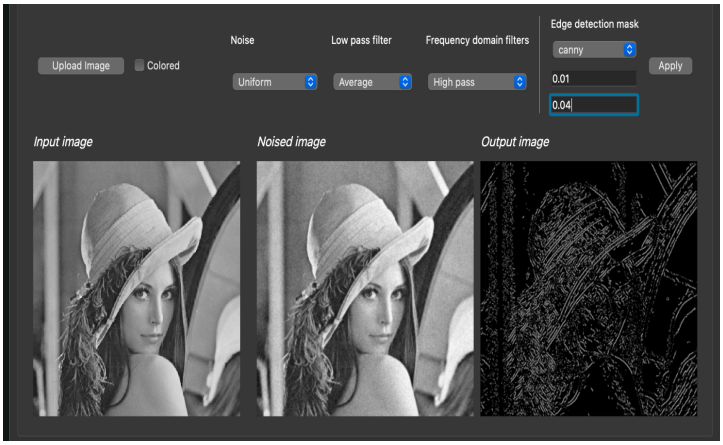
Double threshold

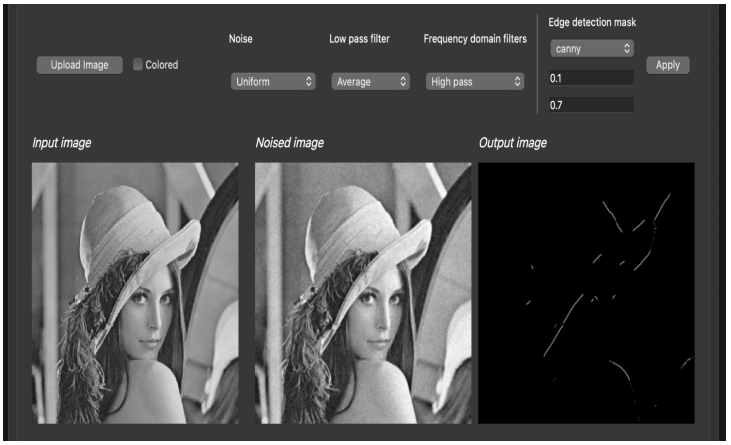
- A high threshold is used to identify the strong pixels (intensity higher than the high threshold)
- A low threshold is used to identify the non-relevant pixels (intensity lower than the low threshold)
- All pixels having intensity between both thresholds are flagged as weak and the Hysteresis mechanism (next step) will help us identify the ones that could be considered as strong and the ones that are considered as non-relevant.

the user can set the low and high threshold from the ui :



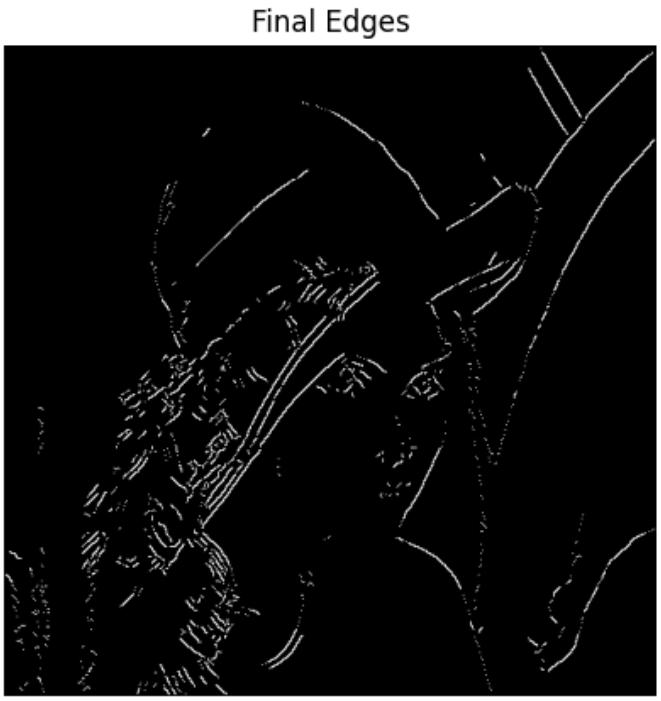
low threshold = 0.1 and high threshold =0.3 give the best result.





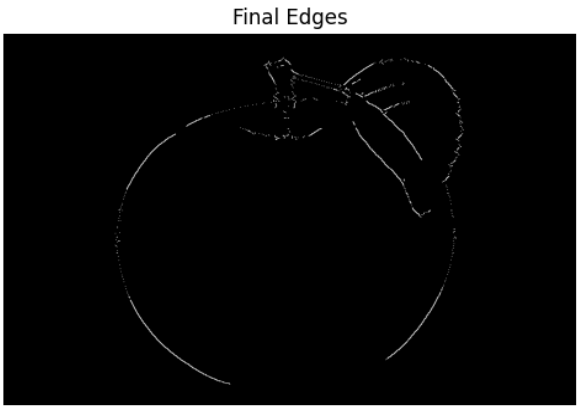
Edge Tracking by Hysteresis

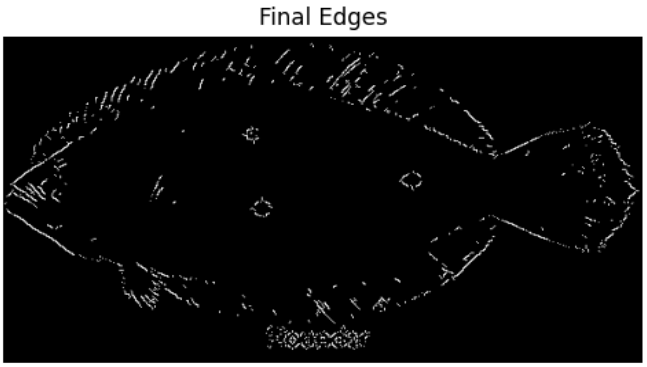
Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one

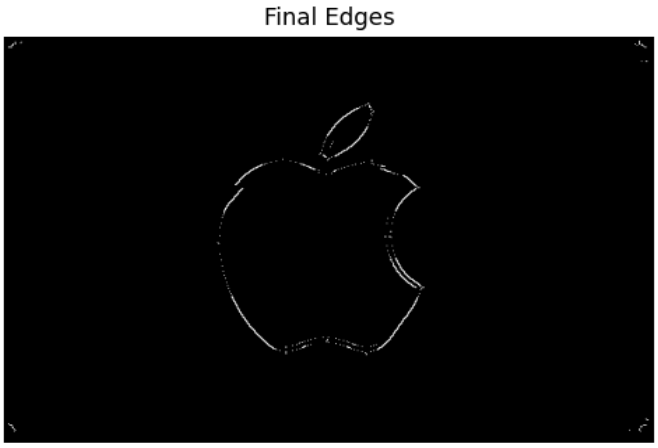


compare with built-in function : `cv2.Canny(image,100,200)`

Canny applied on another images :







Hough Transform

Line Hough Transform

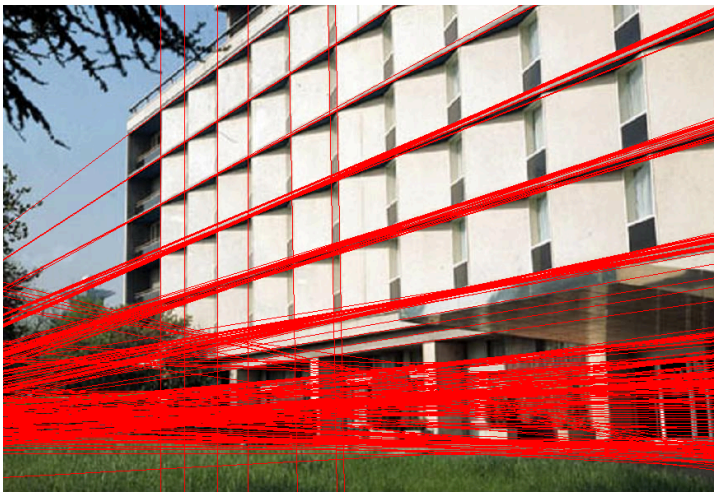
The Hough Line Transform is a transform to detect straight lines in an image. It works by converting the image from the Cartesian coordinate system to the Hough parameter space, which consists of two parameters: the distance from the origin (r) and the angle with the horizontal axis (θ).

Steps of the Line Hough Transform:

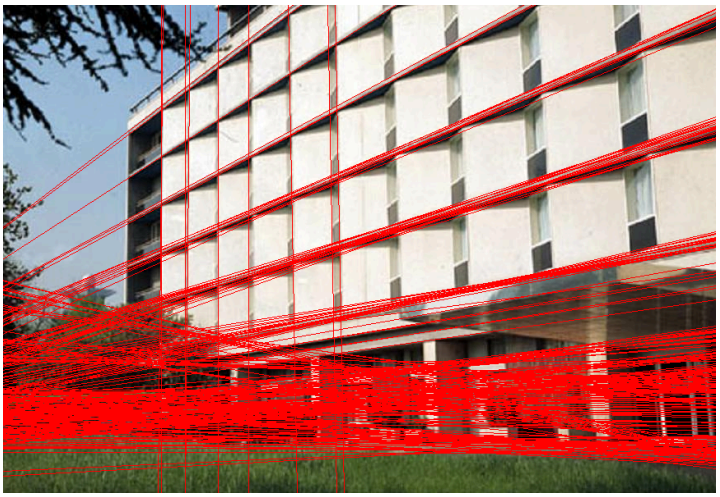
1. Edge Detection: First, an edge detection algorithm, that is Canny edge detector, is applied to the input image to identify the edges.
2. Initializing the Parameter Space: The Hough parameter space is defined by the range of possible values for r and θ. Typically, r represents the distance from the origin to the line, and θ represents the angle between the line and the horizontal axis. The parameter space is discretized into bins to accumulate votes for potential lines.
3. Determine The Range: To apply the Hough Line Transform, it is essential to define the range for the parameters r and θ. The range of r depends on the size of the image and the maximum expected distance between the origin and a line. Typically, the range covers the diagonal length of the image. The range for the parameter θ typically spans from -90 degrees to 90 degrees.
4. Voting: For each edge pixel in the image, the Hough Line Transform algorithm calculates the corresponding r value for all possible θ values. It increments the accumulator array at the corresponding (r, θ) bin, indicating that a line with those parameters has received a vote.
5. Thresholding: After the voting process, the accumulator array is examined to identify the lines that received a sufficient number of votes. A threshold value, that is taken from the user, is set to determine the minimum number of votes required for a line to be considered valid. The lines that exceed the threshold are extracted by analyzing the accumulator array.

Performance of Line Hough Transform at threshold 200:

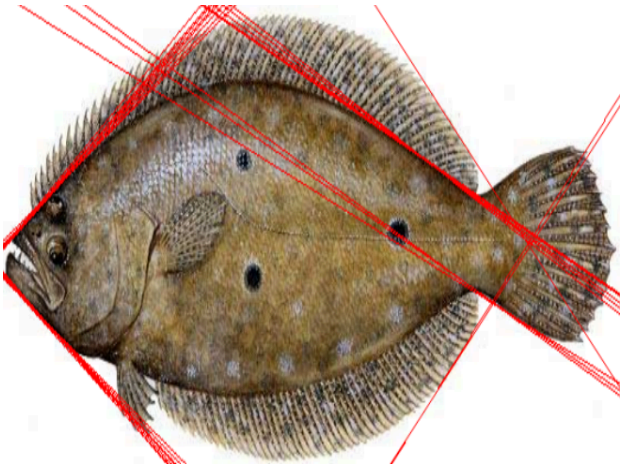
The output of our function

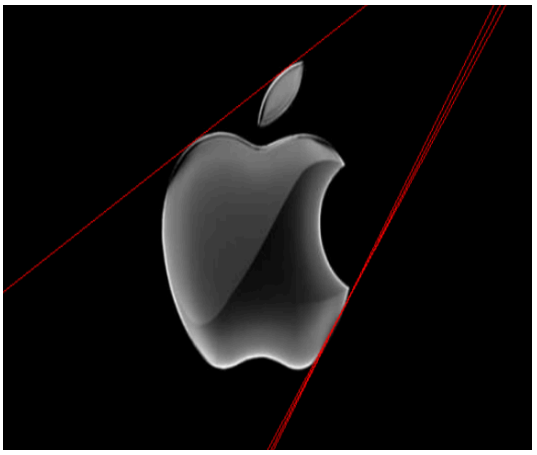


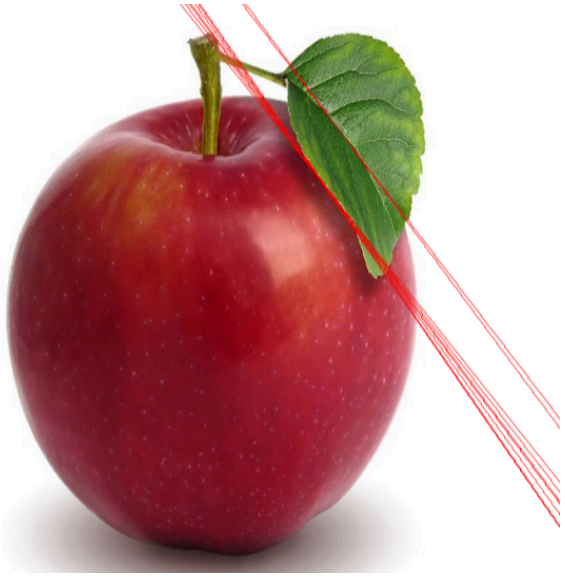
The output by CV built-in HoughLines function

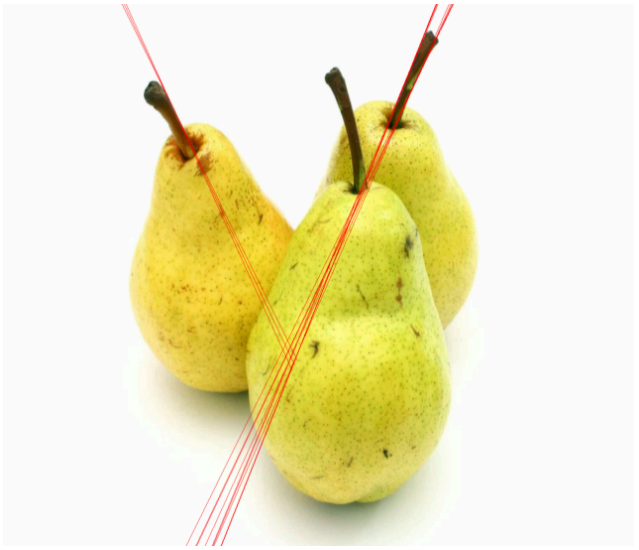


Line Hough Transform of images:









Circle Hough Transform

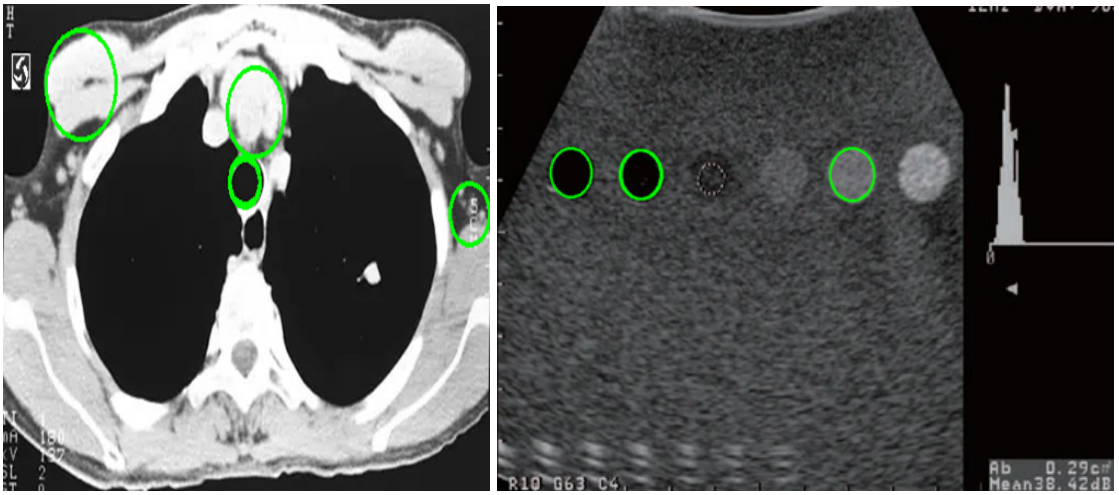
The Circle Hough Transform is a transform to detect circles in an image. It builds upon the concept of the Hough Transform, where geometric shapes are represented in a parameter space. In the case of the Circle Hough Transform, the parameter space consists of three parameters: the x-coordinate of the center of the circle (a), the y-coordinate of the center of the circle (b), and the radius of the circle (r).

Steps of the Circle Hough Transform:

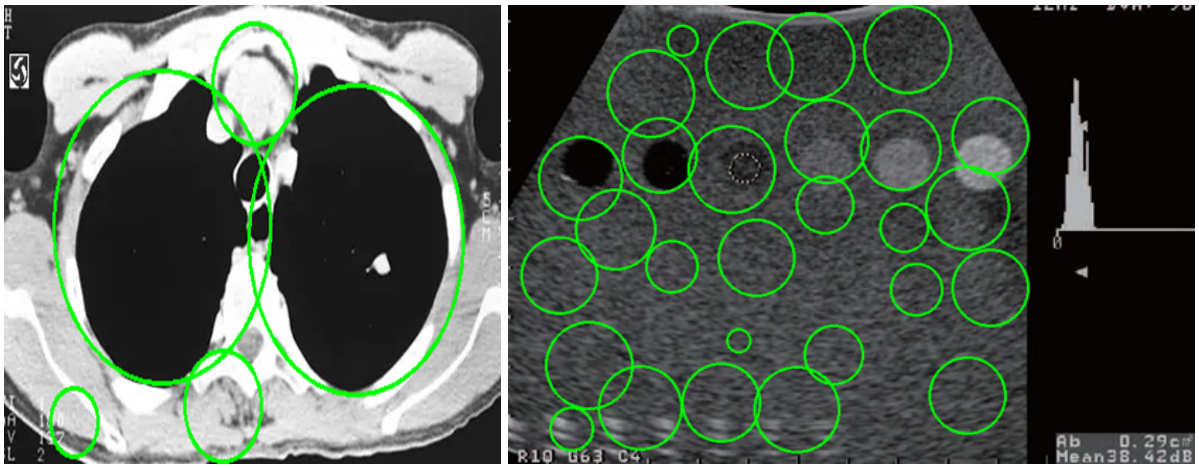
1. Edge Detection: First, an edge detection algorithm, that is Canny edge detector, is applied to the input image to identify the edges.
2. Initializing the Parameter Space: The parameter space is defined by the range of possible values for a, b, and r. The parameter space is discretized into bins to accumulate votes for potential circles.
3. Voting: For each edge pixel in the image, the Circle Hough Transform algorithm calculates the corresponding (a, b, r) values for all possible circles. It increments the accumulator array at the corresponding (a, b, r) bin, indicating that a circle with those parameters has received a vote.
4. Thresholding: After the voting process, the accumulator array is examined to identify the circles that received a sufficient number of votes. A threshold value is set to determine the minimum number of votes required for a circle to be considered valid. The circles that exceed the threshold are extracted by analyzing the accumulator array. The circles that meet the threshold criteria are extracted from the accumulator array, and their corresponding center coordinates (a, b) and radius values (r) are obtained.

Performance of Circle Hough Transform:

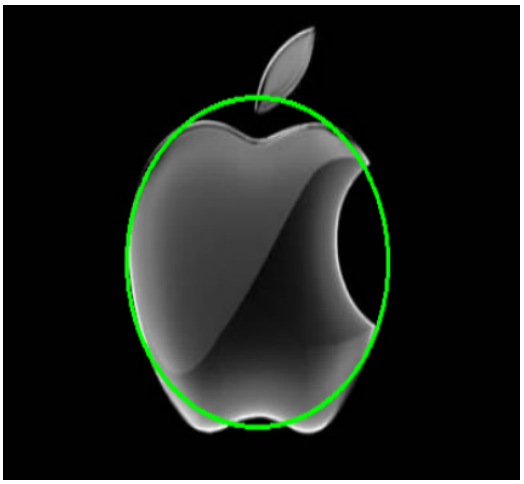
The output of our function

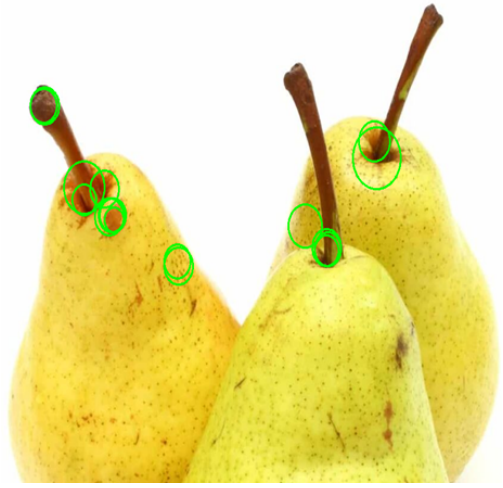


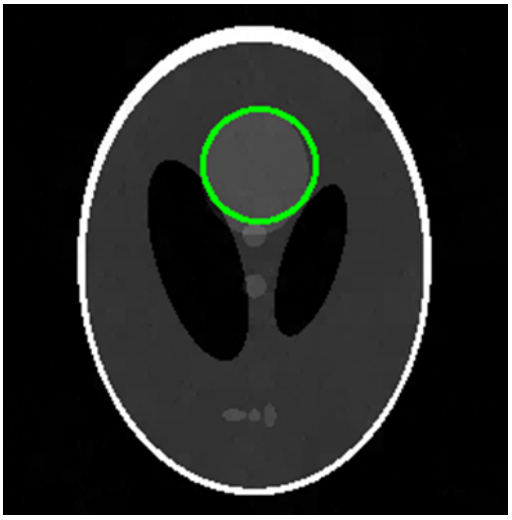
The output by CV built-in HoughCircle function

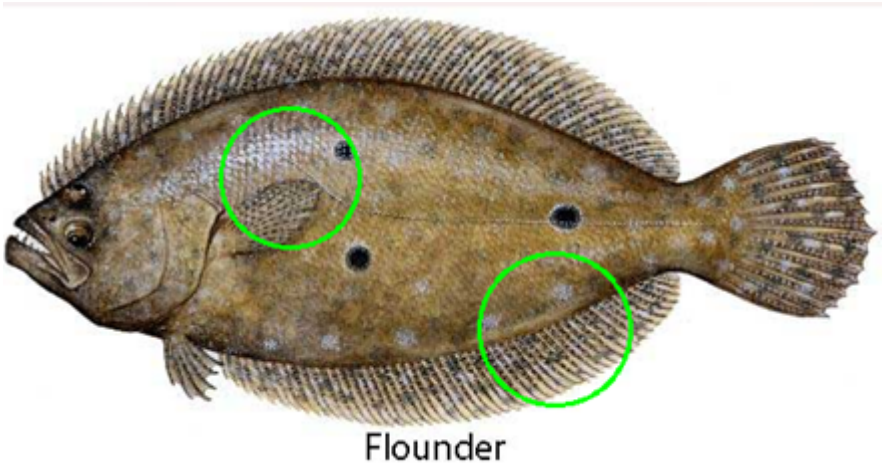


Circle Hough Transform of images:









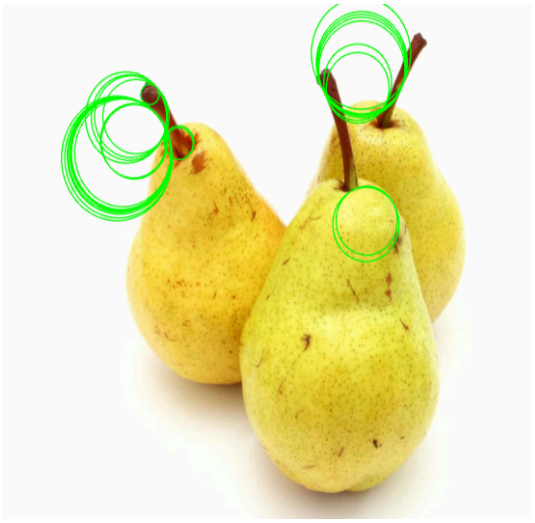
Ellipse Hough Transform

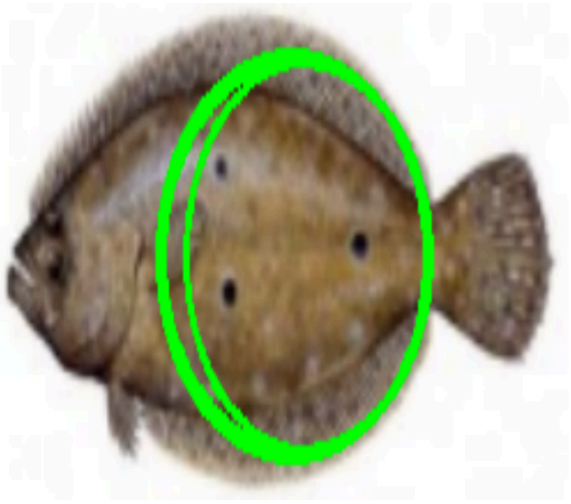
The Ellipse Hough Transform is an extension of the Hough Transform used to detect ellipses in an image. The Ellipse Hough Transform can identify elliptical shapes present in the image.

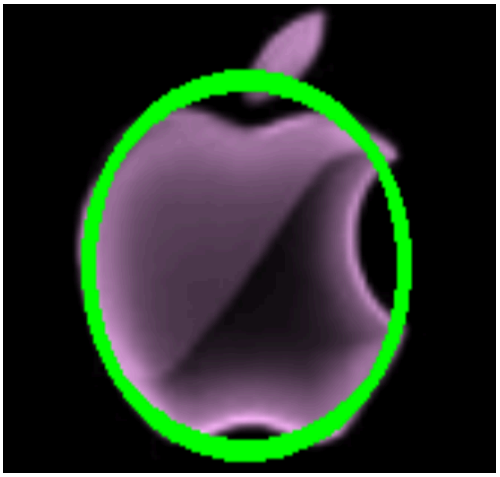
Steps of the Ellipse Hough Transform:

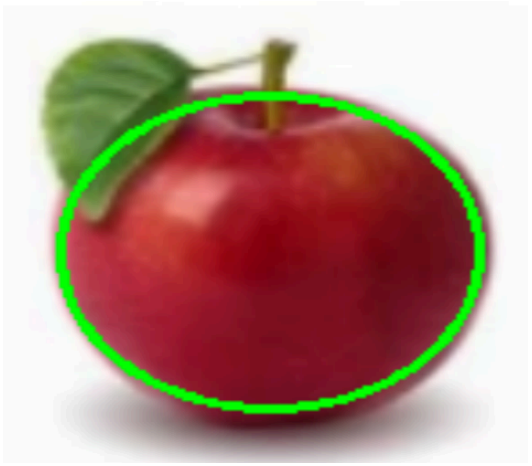
1. Edge Detection: First, an edge detection algorithm, that is Canny edge detector, is applied to the input image to identify the edges.
2. Initializing the Parameter Space: The Hough parameter space for ellipses is defined by the center coordinates (x, y) and the major axis length (a). The parameter space is discretized into bins to accumulate votes for potential ellipses.
3. Determine the Range: It is important to define the range for the parameters (x, y) and a to apply the Ellipse Hough Transform successfully. The range for (x, y) depends on the size of the image, while the range for a depends on the expected size of the ellipses in the image.
4. Voting: For each edge pixel in the image, the Ellipse Hough Transform algorithm calculates the corresponding (x, y) values for all possible values. It increments the accumulator array at the corresponding (x, y, a) bin, indicating that an ellipse with those parameters has received a vote.
5. Thresholding: After the voting process, the accumulator array is examined to identify the ellipses that received a sufficient number of votes. A threshold value is set to determine the minimum number of votes required for an ellipse to be considered valid. The ellipses that exceed the threshold are extracted by analyzing the accumulator array.

Ellipse Hough Transform of images:









Active Contour Model (snake):

Concept:

Active contours, also known as snakes, are curve-evolving algorithms used in image processing to extract object boundaries from digital images. The concept behind active contours involves iteratively deforming an initial curve or contour to align with the desired object boundary while minimizing an energy function. This energy function consists of two main components: internal energy and external energy.

Equations of Energies:

1. Internal Energy(E_internal):

Internal energy ensures the smoothness and regularity of the contour. It consists of two components:

- **Elastic Energy**: Encourages the contour to maintain its elasticity and resist deformation.
- **Curvature Energy**: Penalizes high curvature, promoting smoothness of the contour.

The internal energy is defined as:

$$E_{\text{internal}} = E_{\text{elastic}} + \alpha \cdot E_{\text{curvature}}$$

where α is a user-defined parameter controlling the influence of curvature energy.

2. External Energy (E_external):

External energy attracts the contour towards image features of interest, such as object boundaries. Common external energies include gradient-based energies, edge-based energies, and region-based energies.

The external energy is defined as:

$$E_{\text{external}} = -\beta \cdot \text{Image\_Gradient}$$

- where β is a user-defined parameter controlling the influence of external energy, and

Image_Gradient Image_Gradient represents the image gradient at the contour points.

3. **Total Energy (E_total):**

   The total energy is the combination of internal and external energies.

   The goal is to minimize the total energy to deform the contour towards the desired object boundary.

   The total energy is defined as:

$$E_{\text{total}} = E_{\text{internal}} + E_{\text{external}}$$

Algorithm for Energy Minimization:

1. **Initialization:**
   - Initialize the contour with an initial shape (e.g., circle, rectangle) close to the object boundary.

2. **Iterative Deformation:**

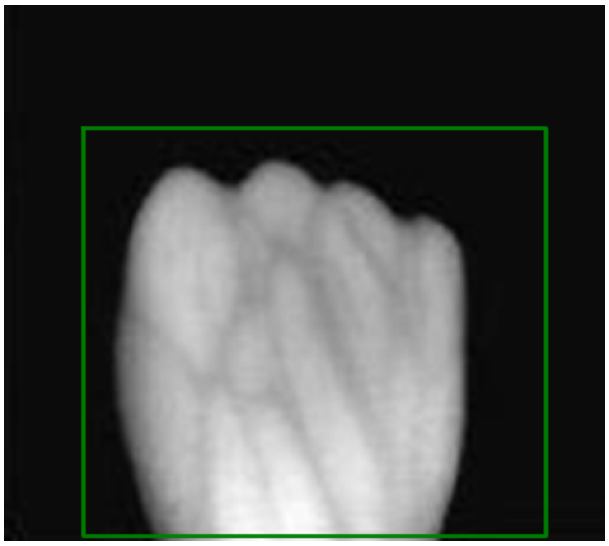
   Repeat the following steps until convergence:

   - Compute the internal energy based on the current contour shape.
   - Compute the external energy based on image features.
   - Compute the total energy
   - Update the contour by minimizing the total energy using a greedy approach.
   - Adjust the contour points to reduce internal and external energies iteratively.

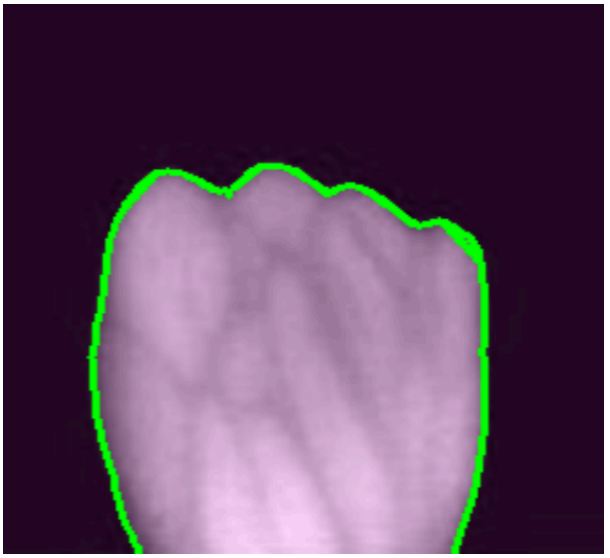
3. **Convergence:**
   - Terminate the iteration when the contour converges, or after a fixed number of iterations.

Output Example:

- Here's an output Example of our implemented Active Contour Using Greedy algorithm:



Initial contour (Square)



Final Contour after convergence

Contour representation using chain code:

- The chain code represents the direction of contour edges relative to a starting point.
- Each element in the chain code corresponds to a specific movement direction (0-7).
- The chain code is sequentially interpreted to reconstruct the contour shape.
- Starting from a reference point, movements specified by the chain code are applied to connect contour points.
- Example of our output chain code:

```
[2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2,
2, 2, 2, 2, 1, 2, 2, 2, 0, 4, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6
, 6, 5, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 6, 6, 6, 6, 6, 6]
```

Perimeter Calculations:

- Perimeter calculation involves summing up the lengths of contour segments.
- Horizontal and vertical movements contribute 1 unit each to the perimeter.
- Diagonal movements contribute √2 units to the perimeter.

Area Calculations:

- Area calculation utilizes Green's theorem, relating the area of a closed curve to the integral of its coordinates.
- The signed area between consecutive contour points is accumulated, accounting for the orientation of the contour.
- Trapezoidal approximations are used to calculate the area enclosed by the contour.