Task2

Basssma alaa eid	**sec:1**	**BN:20**
Abdelrahman Ali farouq	**sec:1**	**BN:55**
Amr Kamal fathi	**sec:2**	**BN:6**
Mohammed sayed zaki	**sec:2**	**BN:17**

Part1:

Canny edge detection:

1. Smoothing for noise removal.

Since the mathematics involved behind the scene are mainly based on derivatives, edge detection results are highly sensitive to image noise. To get rid of the noise on the image, we apply Gaussian blur to smooth it. Image convolution technique is applied with a Gaussian Kernel. The kernel size depends on the expected blurring effect.

2. Finding Gradients.

The Gradient calculation step detects the edge intensity and direction by calculating the gradient of the image using edge detection operators.

3. None-maximum suppression.

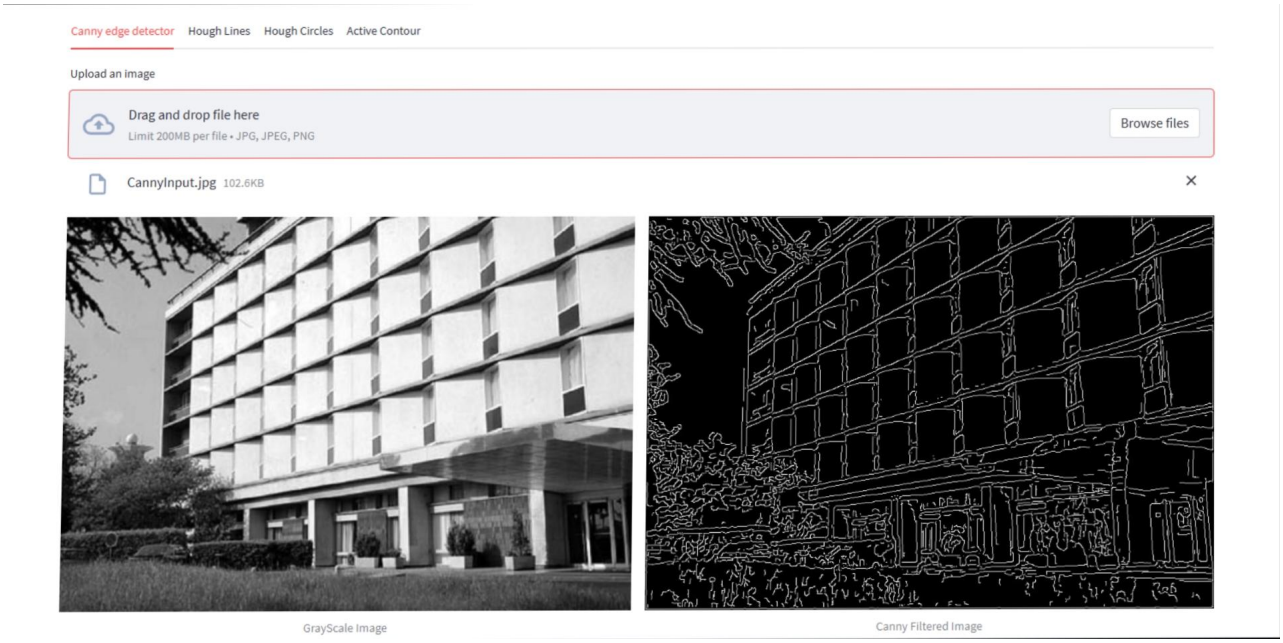
We must perform non-maximum suppression to thin out the edges. The principle is simple: the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions.

4. Double Thresholding.

The double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant: Strong pixels: are pixels that have an intensity so high that we are sure they contribute to the final edge. Weak pixels: are pixels that have an intensity value that is not enough to be considered as strong ones, but yet not small enough to be considered as non-relevant for the edge detection. Other pixels: are considered as non-relevant for the edge.

5. 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.



part2:

Hough lines:

We implement hough algorithm to detect lines in images.

Input parameters

image: a NumPy array representing the input image

edge_image: a NumPy array representing the edge-detected image

num_rhos: number of bins for the distance parameter of the Hough space

num_thetas: number of bins for the angle parameter of the Hough space

The function first calculates the dimensions of the edge image and calculates the maximum distance d between two points in the edge image. It then calculates the step sizes for the angle and distance parameters of the Hough space.

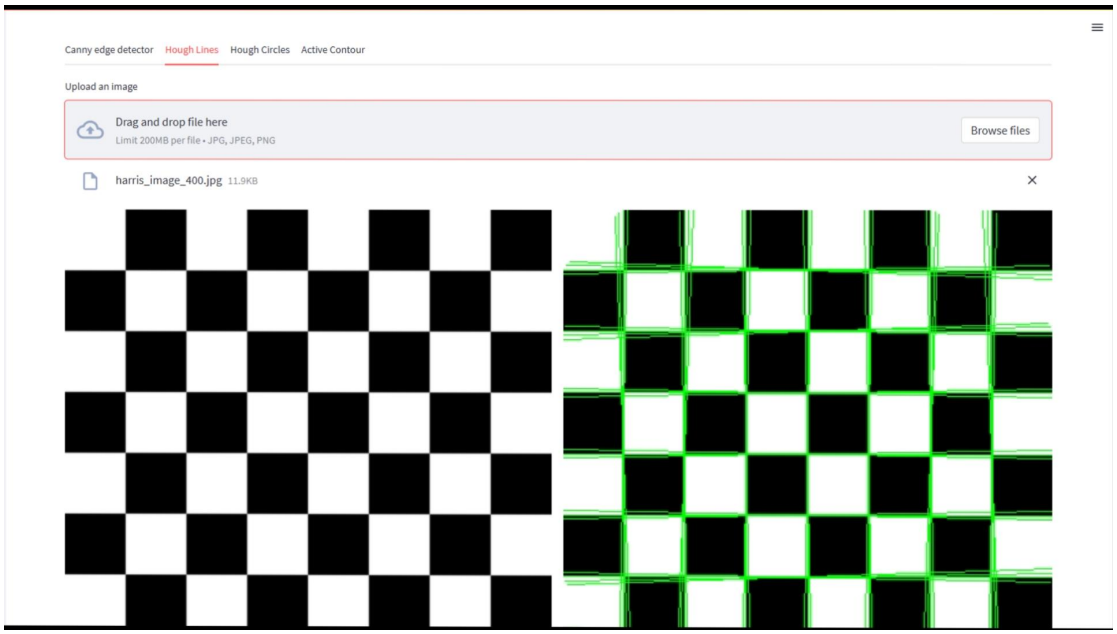
Next, it initializes a zero-filled accumulator array with dimensions (num_rhos, num_thetas) and initializes an output image array of the same size as the input image. It also initializes a figure with four subplots for visualizing the input image, edge image, accumulator array, and output image.

The function then iterates over all pixels in the edge image and for each edge pixel, it calculates the distance rho and angle theta parameters of the line passing through that pixel. It then finds the corresponding indices in the accumulator array and increments the corresponding accumulator bin by one. It also appends the rho and theta values to lists ys and xs.

After all edge pixels have been processed, the function calls the peak_votes function to find the top n values in the accumulator array. For each of these values, it calculates the corresponding rho and theta parameters and uses them to draw a line on the output image.

Finally, the output image is displayed using the st.image function, and the function returns the output image.

The peak_votes function takes in the accumulator array, the rho and theta arrays, and the number of top values to return n. It returns the indices, rho values, and theta values corresponding to the top n values in the accumulator array.



Part3:

Hough circles:

We implement hough algorithm to detect circles in images.

The input parameters to the function are:

image: a NumPy array representing the input image

edge_image: a NumPy array representing the edge-detected image

num_rhos: number of bins for the distance parameter of the Hough space

num_thetas: number of bins for the angle parameter of the Hough space

The function first calculates the dimensions of the edge image and calculates the maximum distance d between two points in the edge image. It then calculates the step sizes for the angle and distance parameters of the Hough space.

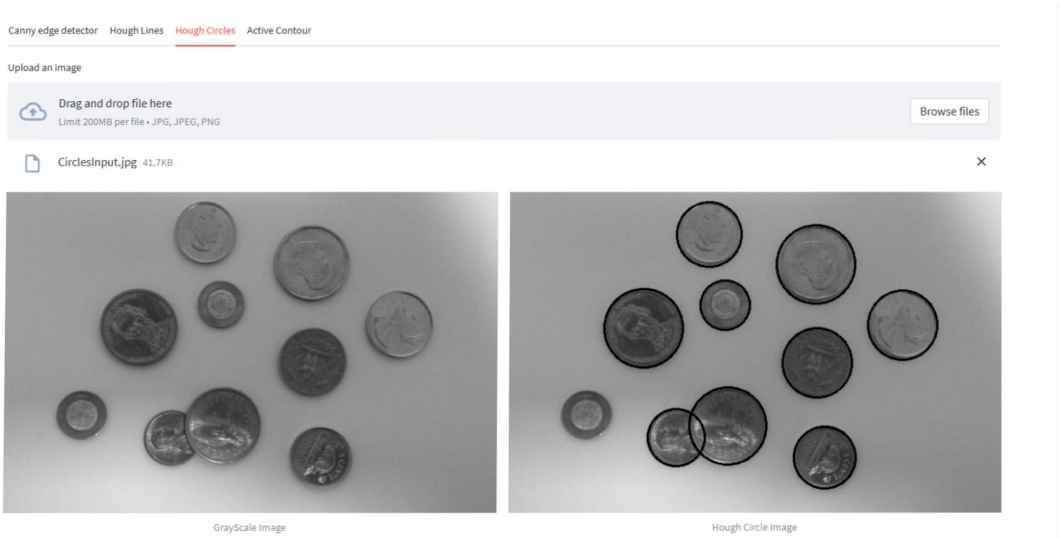
Next, it initializes a zero-filled accumulator array with dimensions (num_rhos, num_thetas) and initializes an output image array of the same size as the input image. It also initializes a figure with four subplots for visualizing the input image, edge image, accumulator array, and output image.

The function then iterates over all pixels in the edge image and for each edge pixel, it calculates the distance rho and angle theta parameters of the line passing through that pixel. It then finds the corresponding indices in the accumulator array and increments the corresponding accumulator bin by one. It also appends the rho and theta values to lists ys and xs.

After all edge pixels have been processed, the function calls the peak_votes function to find the top n values in the accumulator array. For each of these values, it calculates the corresponding rho and theta parameters and uses them to draw a line on the output image.

Finally, the output image is displayed using the st.image function, and the function returns the output image.

The peak_votes function takes in the accumulator array, the rho and theta arrays, and the number of top values to return n. It returns the indices, rho values, and theta values corresponding to the top n values in the accumulator array.



Part4:

Hough ellipse:

The hough_ellipse_detection function takes an input image and several parameters, including the detection threshold, minimum and maximum major and minor axis lengths, and the major axis step size. The function first preprocesses the image by converting it to grayscale and detecting edges using the Canny edge detector. It then downsamples the image if its dimensions are greater than 500 pixels in either
direction.

```
for y in range(height):

for x in range(width):

    if edges[y, x]:

        for a_idx, a in enumerate(range(min_major_axis, max_major_axis, major_axis_step)):

            for theta in range(180):
```

```
                b = int(a * np.sin(np.deg2rad(theta)))
                xc = int(x - a * np.cos(np.deg2rad(theta)))
                yc = int(y - b * np.sin(np.deg2rad(theta)))

                if 0 <= xc < width and 0 <= yc < height and min_minor_axis <= b < max_minor_axis:
                    accumulator[yc, xc, a_idx, theta] += 1
for y in range(height):
    for x in range(width):
        if edges[y, x]:
            for a_idx, a in enumerate(range(min_major_axis, max_major_axis, major_axis_step)):
                for theta in range(180):
                    b = int(a * np.sin(np.deg2rad(theta)))
                    xc = int(x - a * np.cos(np.deg2rad(theta)))
                    yc = int(y - b * np.sin(np.deg2rad(theta)))

                    if 0 <= xc < width and 0 <= yc < height and min_minor_axis <= b < max_minor_axis:
                        accumulator[yc, xc, a_idx, theta] += 1
```

For each pixel in the image, the loop iterates over the range of major axis lengths again and computes the corresponding minor axis length using the formula for an ellipse. If the count in the corresponding bin of the accumulator array is greater than or equal to the detection threshold, an ellipse is added to the ellipses list.

The draw_ellipses function takes the input image and the list of detected ellipses and draws each ellipse on the image using the cv2.ellipse function. The resulting image with the detected ellipses is then returned.

Part5:

Active contour:

Active contour, also known as snakes, is a technique in computer vision used to detect and track the boundaries of objects in an image. The Active Contour Model (ACM) is a type of deformable model that evolves over time to fit the boundaries of an object in an image.

Some additional details on Active Contour Models include:

Initialization: The initialization of the curve is an important step in the Active Contour Model algorithm. The initial curve can be a simple shape, such as a circle or square,

or it can be generated using other techniques, such as edge detection or thresholding. The accuracy of the initial curve can have a significant impact on the performance of the algorithm.

Energy functions: The energy functions used in Active Contour Models can vary depending on the specific application. The internal energy term is often based on the curvature of the curve, while the external energy term is based on the image gradients or other image features that correspond to the object boundaries. The weights given to each energy term can be adjusted to control the influence of each term on the curve evolution.

Gradient descent: The ACM algorithm typically uses gradient descent to minimize the energy functional. In each iteration, the gradient of the energy functional with respect to the curve is computed, and the curve is updated by moving it in the direction of the negative gradient.

Snake visualization: The Active Contour Model algorithm can be visualized by plotting the curve at each iteration over the image. This allows the user to see how the curve evolves over time and to monitor the convergence of the algorithm.

Limitations: Active Contour Models are sensitive to the initialization of the curve and the choice of energy functions. They can also be computationally expensive, particularly for large images or complex curves. Additionally, they can struggle with objects that have multiple boundaries or irregular shapes.

In summary, Active Contour Models are a powerful technique for image segmentation and object tracking, but they require careful parameter tuning and initialization to achieve good performance. They are widely used in computer vision research and applications, and many libraries and frameworks provide implementations of the algorithm.