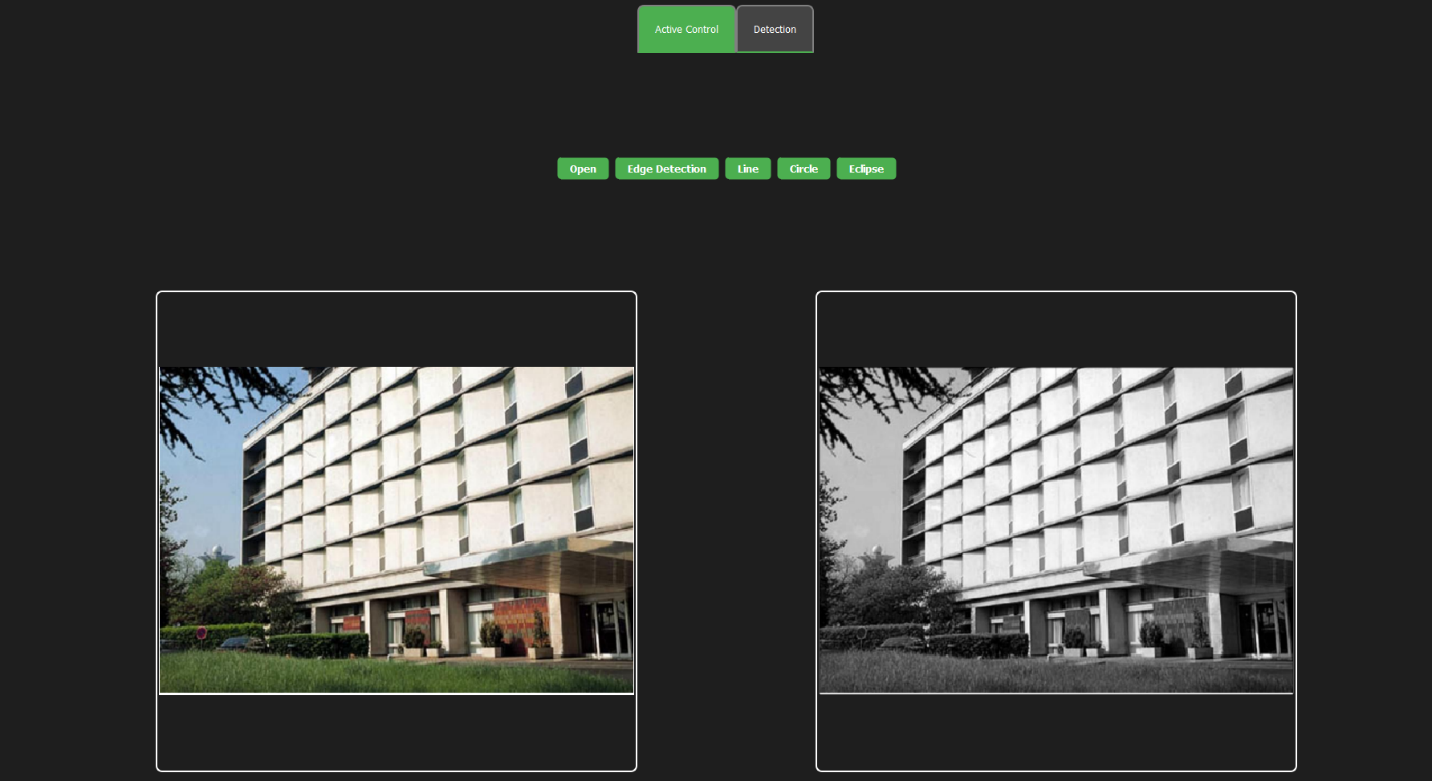
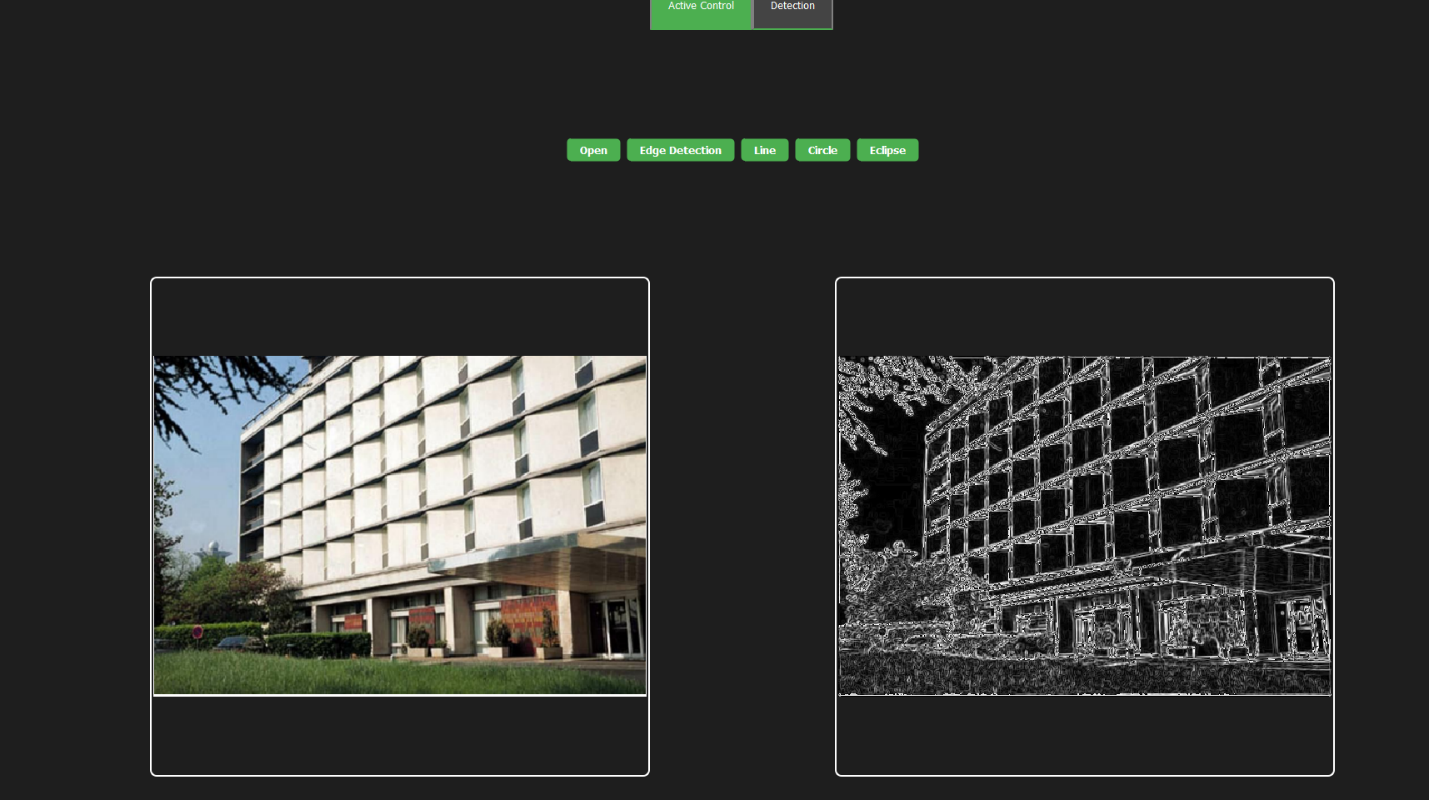
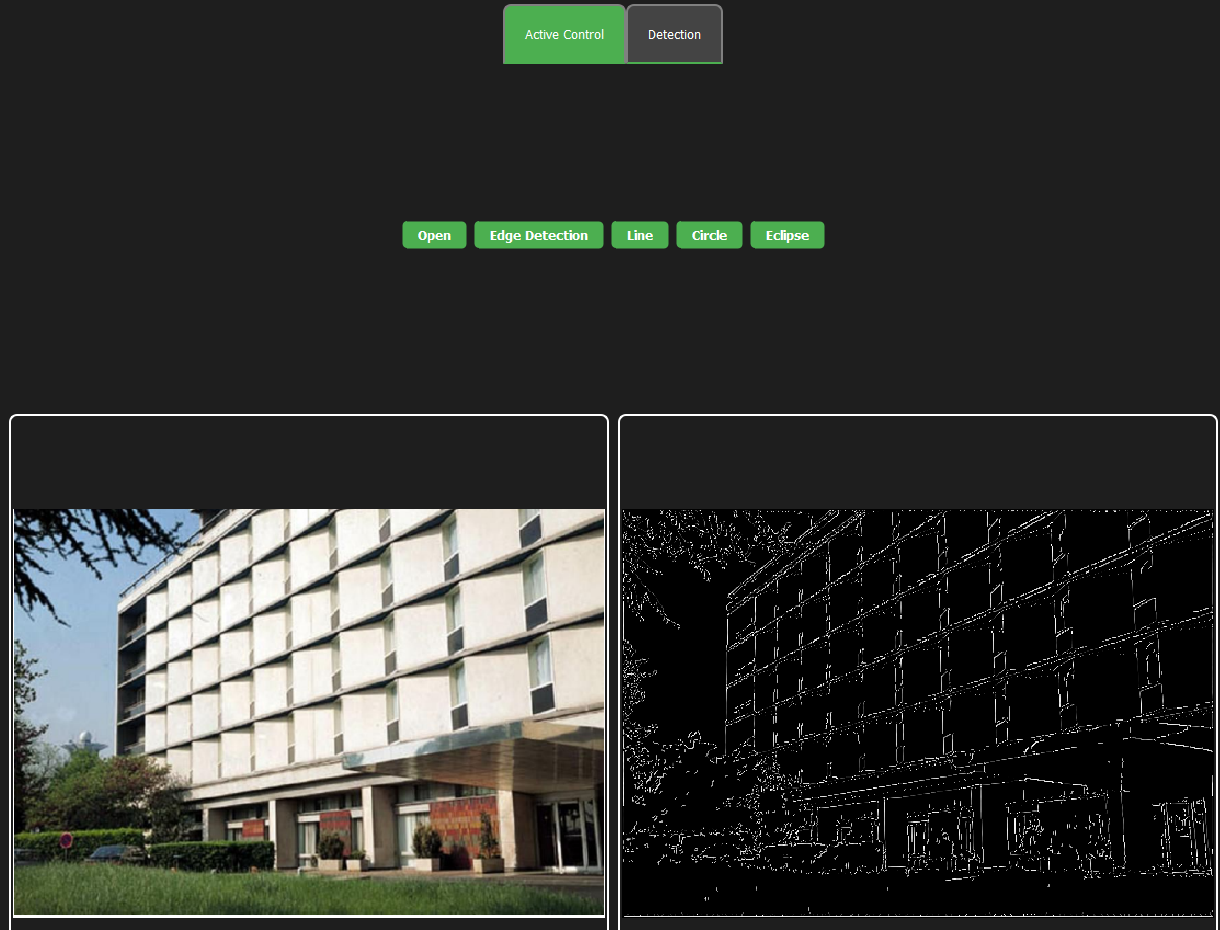
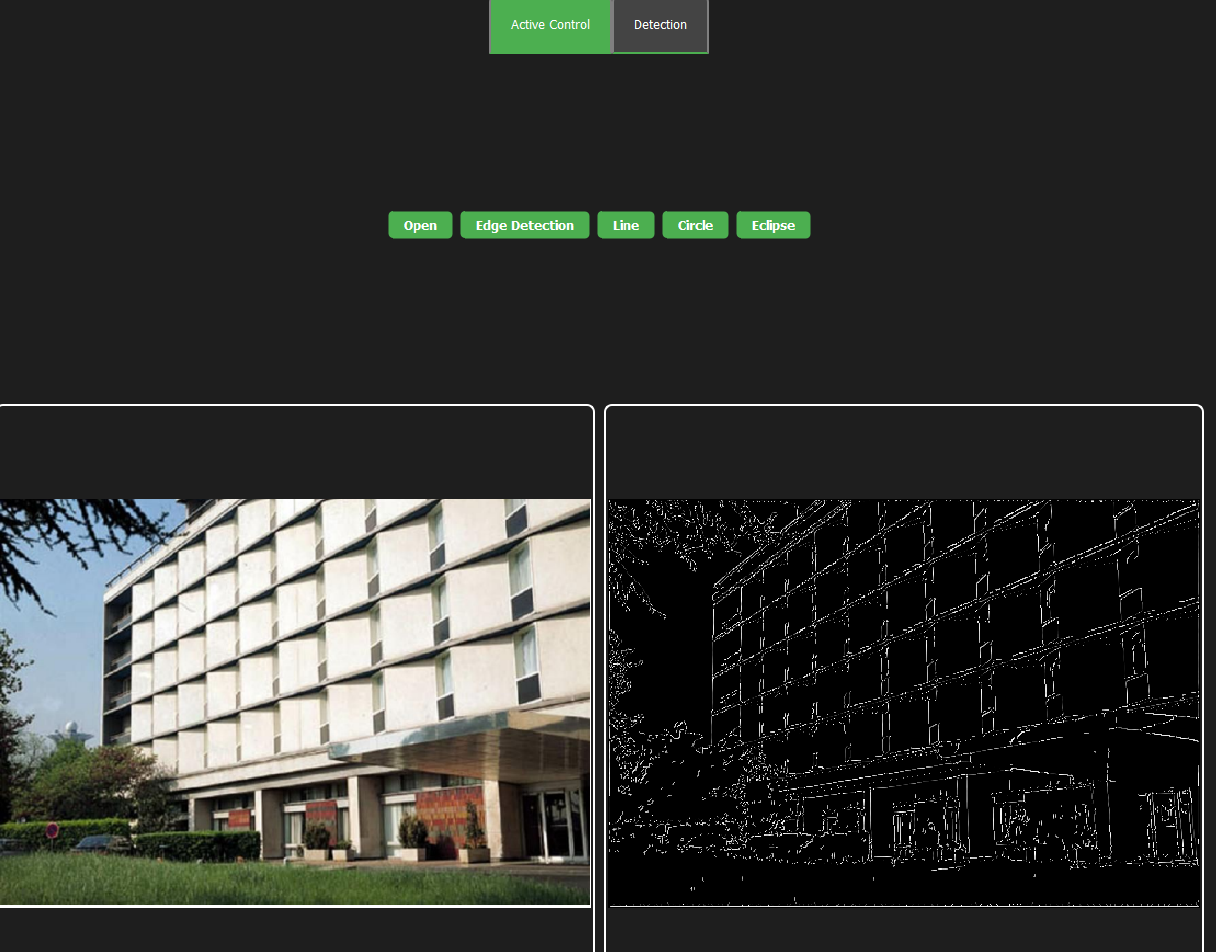
**Edge Detection Process:**

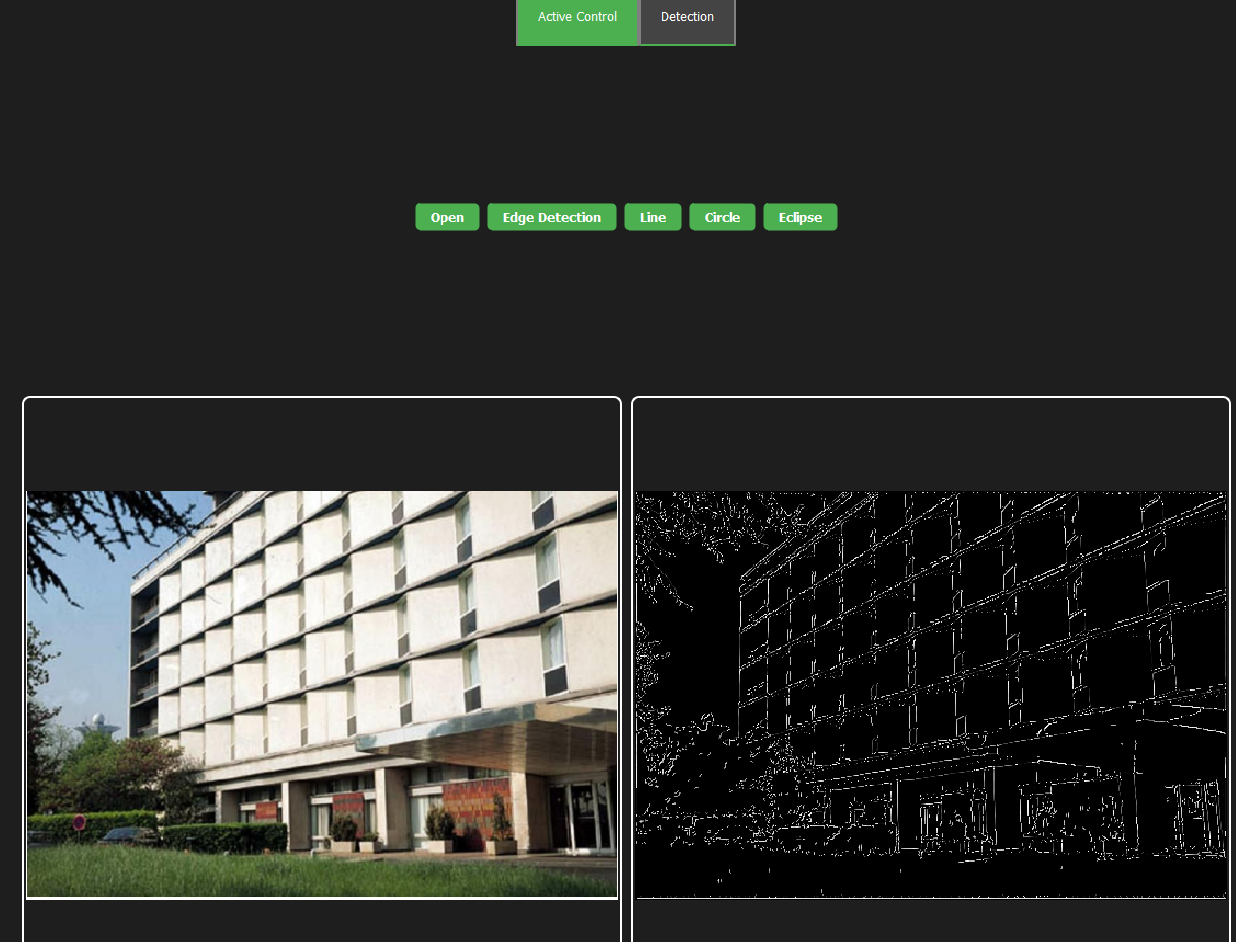
1. **Smoothing for Noise Removal:** The first step in edge detection involves smoothing the image to remove noise. This is typically achieved using techniques such as Gaussian blurring or median filtering. Smoothing helps in reducing high-frequency noise, which can interfere with edge detection algorithms.

**But I used** Gaussian blurring with with kernel size =3 and standered\_deviaation =1

1. **Finding Gradients:** After noise removal, the next step is to compute the gradients of the smoothed image. Gradients represent the rate of change of pixel intensities and are essential for detecting edges. Common methods for gradient calculation include Sobel, Prewitt, or Roberts operators and I used Sobel with kernel size=5.
2. The output can be change with change the kernel size.

**3. Non-Maximum Suppression:** Once gradients are computed, non-maximum suppression is applied to suppress non-edge pixels. This step involves comparing the gradient magnitude of each pixel with its neighbors along the direction of the gradient. Only pixels that represent local maxima in gradient magnitude are retained as potential edge pixels.

**4. Double Thresholding:** After non-maximum suppression, double thresholding is used to classify pixels as either strong, weak, or non-edges. This involves setting two thresholds: a high threshold and a low threshold. Pixels with gradient magnitudes above the high threshold are classified as strong edges, while those below the low threshold are classified as non-edges. Pixels with gradient magnitudes between the two thresholds are classified as weak edges and I use high threshold as 0.6 from max value of strong edges and low threshold as 0.1 from max value of strong edges and then delete pixel which is lower than low threshold and the output is :

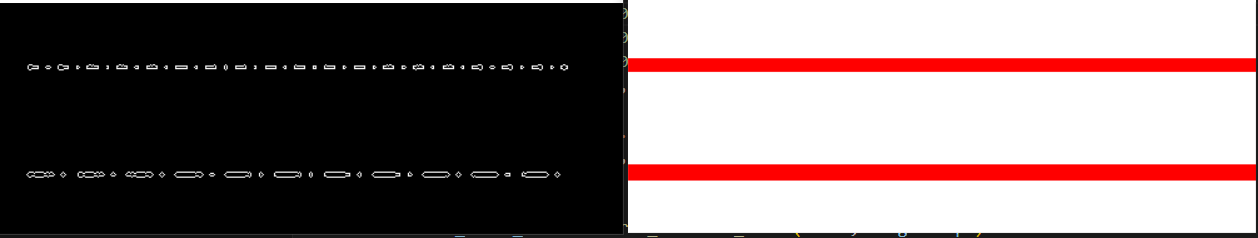
**5. Edge Tracking by Hysteresis:** The final step in edge detection is edge tracking by hysteresis. Weak edge pixels that are adjacent to strong edge pixels are promoted to strong edges, while weak edge pixels not connected to strong edges are suppressed. This helps in connecting fragmented edge segments and producing continuous edge contours and the final output is 

**Line Detection:**

1. **Detect Line Method (detect\_line):**
   * Description:
     + Explanation of the **detect\_line** method that identifies lines in an input image using the Hough transform.
   * Steps:
     + Convert the input image to grayscale.
     + Apply Canny edge detection to find edges in the grayscale image.
     + implement the Hough transform to detect lines in the edge-detected image.
     + Extract lines based on a specified threshold.
   * Output:
     + Returns a QPixmap with detected lines drawn on the input image.
2. **Hough Lines Method (hough\_lines):**

Steps:

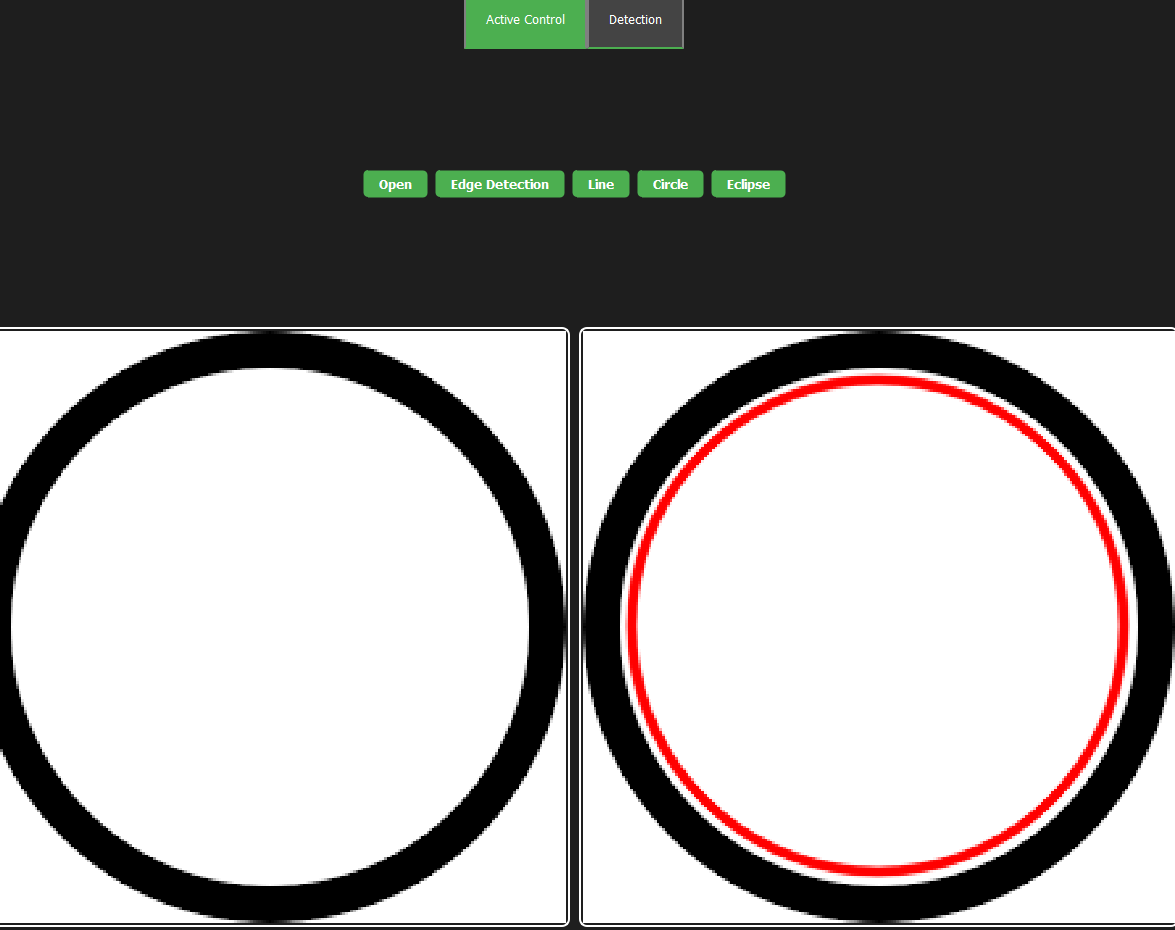
* + - Define the Hough space accumulator.
    - Find edge pixels using Canny edge detection.
    - Perform the voting process in the Hough space.
    - Extract lines from the accumulator based on a threshold.
  + Output:
    - Returns a list of detected lines represented by rho (distance from the origin) and theta (angle).

1. **Draw Lines Method (draw\_lines):**
   * Steps:
     + Convert polar coordinates to Cartesian coordinates for line endpoints.
     + Draw it on original image .

Detect circule:

the purpose of the **Hough\_Circle\_Transform** function is to identify circles in a given image based on edge detection using the Canny edge detector and subsequent analysis of edge points in the Hough parameter space

#### **Parameters:**

* **image**: The input image in which circles are to be detected.
* **threshold**: The minimum accumulator threshold for detecting circles. Higher values result in fewer detected circles.
* **min\_radius**: The minimum radius of the circles to be detected (default value: 10).
* **max\_radius**: The maximum radius of the circles to be detected (default value: 200).
* **canny\_min\_threshold**: The lower threshold value for the Canny edge detector (default value: 100).
* **canny\_max\_threshold**: The upper threshold value for the Canny edge detector (default value: 200).
* **Steps**
* **Convert Image to Grayscale**: The input image is converted to grayscale using my function convert RGB into gray
* **Edge Detection**: Edge detection is performed on the grayscale image using the Canny edge detector using my function cany\_edg\_detection .
* **Initialize Accumulator**: A 3D accumulator array is initialized to store the votes for circle parameters (center coordinates and radius).
* **Iterate Over Edge Points**: For each edge point detected in the image, a loop iterates over possible circle radii and theta values.
* **Calculate Circle Parameters**: For each combination of edge point, radius, and theta, the center coordinates of the circle are calculated using the Hough circle equation.
* **Increment Accumulator**: The accumulator array is updated to record the votes for each circle parameter combination.
* **Identify Circles**: Once all edge points have been processed, the accumulator is analyzed to identify circles with votes above the specified threshold.
* **Draw Detected Circles**: Detected circles are drawn on a copy of the original image using the **cv2.circle** function.
* **Return Result**: The modified image with detected circles is returned as the output of the function.
* **And here is the final result but it takelong time due its computation complexty**.

**Active contour**

The Active Contour Segmentation Algorithm, also known as Snake algorithm, is a powerful technique used in image processing and computer vision for contour detection and segmentation. It is widely used in various applications such as medical imaging, object tracking, and boundary detection.

**Implementation Details**

1. **Image Loading**: The load\_image() method is responsible for loading an image from the specified file path and displaying it in the graphics view.
2. **Contour Drawing**: The draw\_contour() method is used to draw a contour on the loaded image. It takes parameters such as center coordinates, radius, and number of points to generate the contour and I allow user to make initialization to contour by entering center , radius , number of points

**And this is the intiall contour**:

1. **Contour Manipulation**: The start() method starts the active contour segmentation algorithm. It iterates over a specified number of iterations, moving the contour to minimize the energy based on internal and external forces , I also make user to enter alpha ,beta, gama and number of iteration and I notice during implementation that the accuracy depend on the previous parameter and computation process will increase by increase the number of iteration ,
2. **Energy Calculation**: Internal energy and external energy of the contour are computed using the \_compute\_internal\_energy() and \_compute\_external\_energy() methods, respectively.

**Experimentation and Results**

The implementation of the Active Contour Segmentation Algorithm was tested using sample images. The algorithm successfully detected and segmented contours in the images, demonstrating its effectiveness in contour detection