Individual Project Report

Amogh Meenakshi Shadangi

LC3: Team 8

[Course number and name]

Prof. Brandon Harrison-Smith

10th December,2024

1. Project Introduction

My project focuses on real-time recognition of geometric shapes using a live camera feed and provides additional information about these shapes through an intuitive GUI. It addresses the challenge of automating shape recognition, with applications in education, design, and augmented reality (AR). It is one of the first initial steps toward my dream project of creating AR glasses that can help people with building and designing. By learning to recognize and understand shapes, this project sets the foundation for tools that can make tasks easier and more creative for users.

2. Project Overview of Inputs and Outputs

This project uses a live camera feed as its input, making each input unique and dynamic since it depends on the shape presented to the camera. The system processes video frames to detect shapes like circles, triangles, squares, rectangles, and pentagons. The camera continuously scans and identifies the contours of the shapes, showing the number of points (sides) it detects, which contributes to accurate shape recognition.
The expected output is displayed in real-time on the same screen as the live video feed, combining the detected shape's name, properties, and highlighted contours. This approach gives it an augmented reality (AR) glasses feel, as users see how the system scans the image and identifies shapes directly within the footage. For example, if a user shows a triangle, the screen will display the shape's highlighted contour, the number of points (3 sides), and label it as "Shape: Triangle." This interactive visualization bridges digital analysis with real-world perception.

3. User-defined Functions

1 **show_menu()**

- Displays a GUI menu asking the user whether to start shape recognition or exit the program.
- **Inputs:** None (user interaction through button clicks by pressing the "Z" on keyboard).
- **Outputs:** Displays a GUI screen which asks weather or not to start the shape recognising or not.
- It Serves as the entry point for the user to decide on how to proceed with the code. It will make him choose weather to open it or not just like an app or a software.

2 **display_output()**

- This function Opens a GUI window with buttons which display different shapes and they link to web pages of Wikipedia for that shape providing information about them.
- **Inputs:** None (predefined web links for shapes).
- **Outputs:** Opens the web page in a browser when a button is clicked with respect to whichever shape is chosen.
- This function completes the last step that I wanted to do in my project as I wanted to add a GUI which will be able to give instructions based on what the glasses show to them. This might not be automatic, however it is the first step in achieving that and it adds educational value by providing additional context about recognized shapes.

3 getcontours(img, imgcontour)

This function Detects contours in the provided live image and identifies the shapes based on the number of sides that it counts.

- **Inputs:**
  - **img:** The processed image (e.g., dilated or edge-detected) to find contours.
  - **imgcontour:** A copy of the original image for drawing and displaying contours.
- **Outputs:** finds the number of sides the shape has, highlights identified shapes with contours, displays the number of sides, and labels the shape name on the live feed**.**
  The core function for shape detection. It uses OpenCV functions like findContours and approxPolyDP to analyze contours and determine shape properties. It dynamically adjusts based on user-defined parameters (e.g., minimum area) and overlays the results on the live video, creating an AR-like experience.

4 stackImages(scale, imgArray)

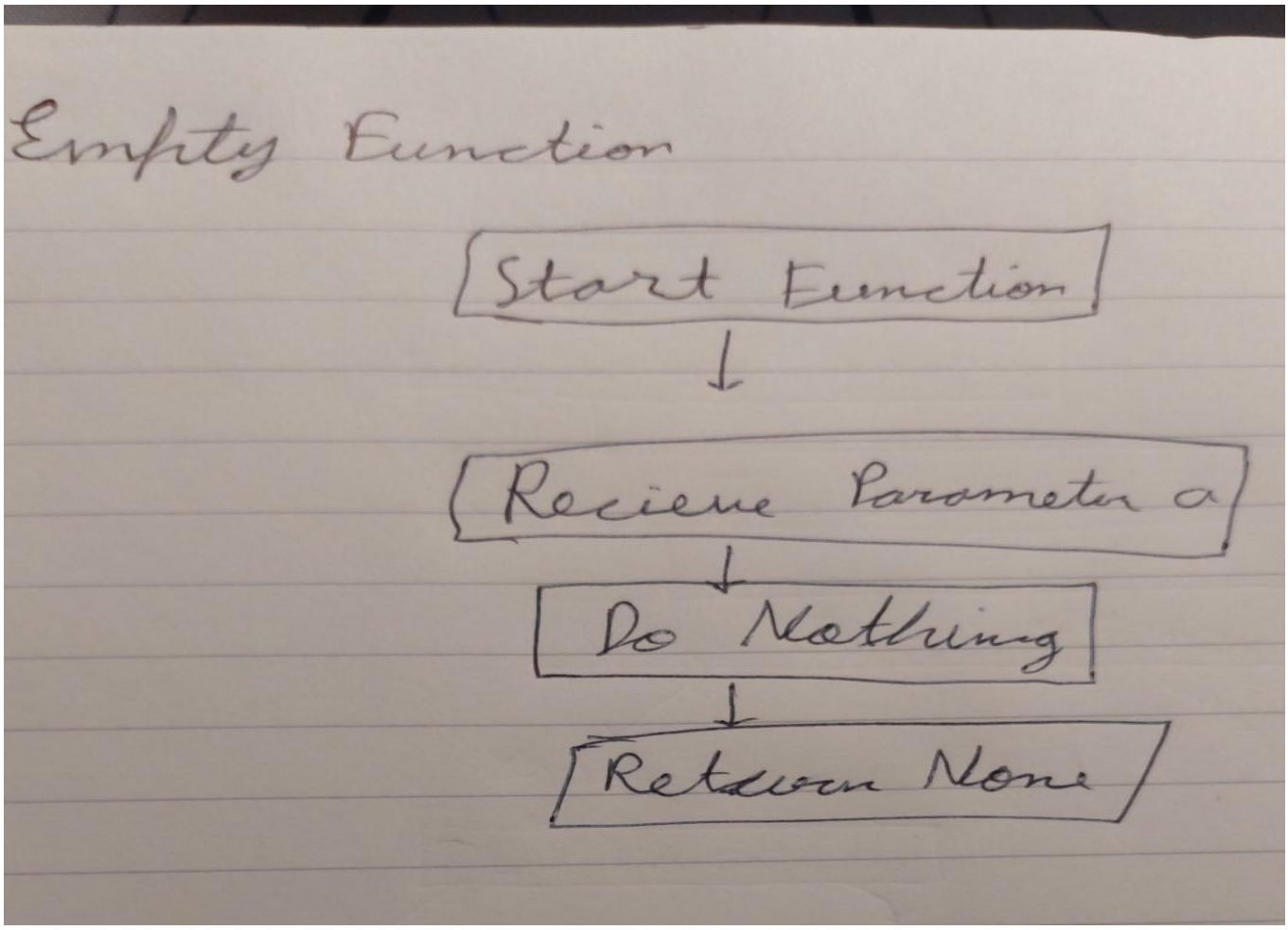
Combines multiple images into a single display for better visualization.

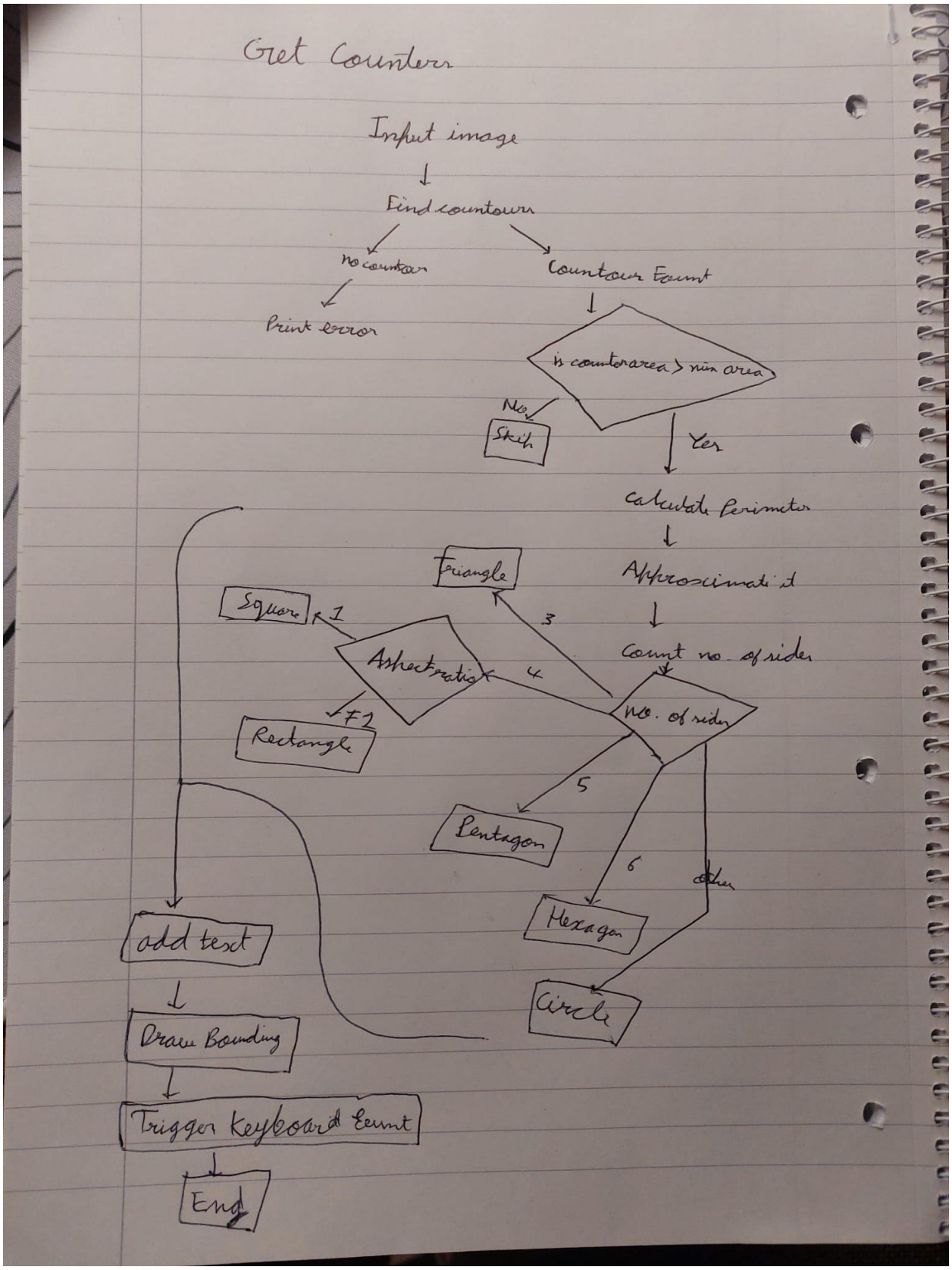
- **Inputs:**
  scale: The resizing scale for images.
  imgArray: A list of images to be stacked.
- **Outputs: A single stacked image combining all inputs.**
  It simplifies the visualization of multiple processing stages (e.g., original, edge-detected, and contour-highlighted images) on a single screen. It is so that the user can see how exactly the shape recognition works in real time.

5 empty(a)

It is placeholder function for the trackbars to prevent errors while adjusting parameters.

- **Inputs:** A single parameter (a) passed automatically by the trackbar.
- **Outputs:** None.
  Allows smooth operation of dynamic parameter adjustment (e.g., thresholds for Canny edge detection) without interrupting the program**.**


Empty Function
Start Function
Recieve Parameter a
Do Nothing
Return None


Get Counters
Input image
Find countours
no countour
Countour Count
Print error
is counterarea > min area
No
Skip
Yes
Calculate Perimeter
Approximate it
Count no. of sides
Triangle
3
Square
1
Aspectratio
4
No. of sides
2
Rectangle
5
Pentagon
6
other
Hexagon
Circle
add text
Draw Bounding
Trigger Keyboard Event
End

4. References

Murtaza's Workshop - Robotics and AI. YouTube, www.youtube.com/@murtazasworkshop.

Shape Detection. GitHub, github.com/jrieke/shape-detection.

Fabio Musanni. YouTube, www.youtube.com/@FabioMusanni.

AI correction and suggestions

5. Appendix

1. **User manual**

1. Install Necessary Dependencies
Ensure that you have the required libraries installed. Run the following commands in your terminal:

pip install opencv-python

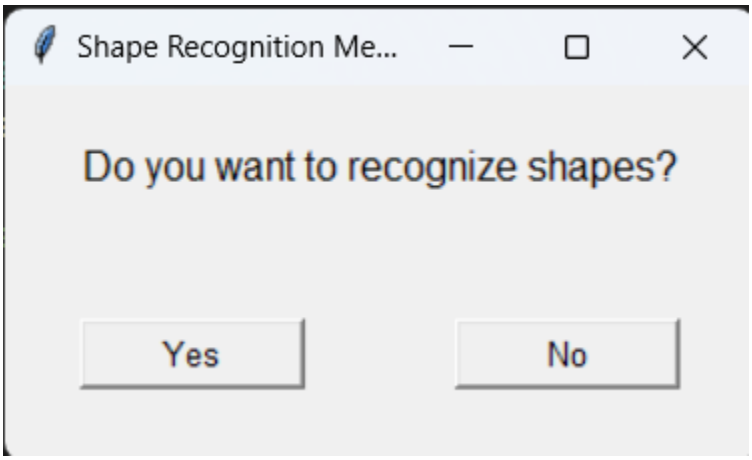
pip install numpy

pip install tkinter

pip install keyboard

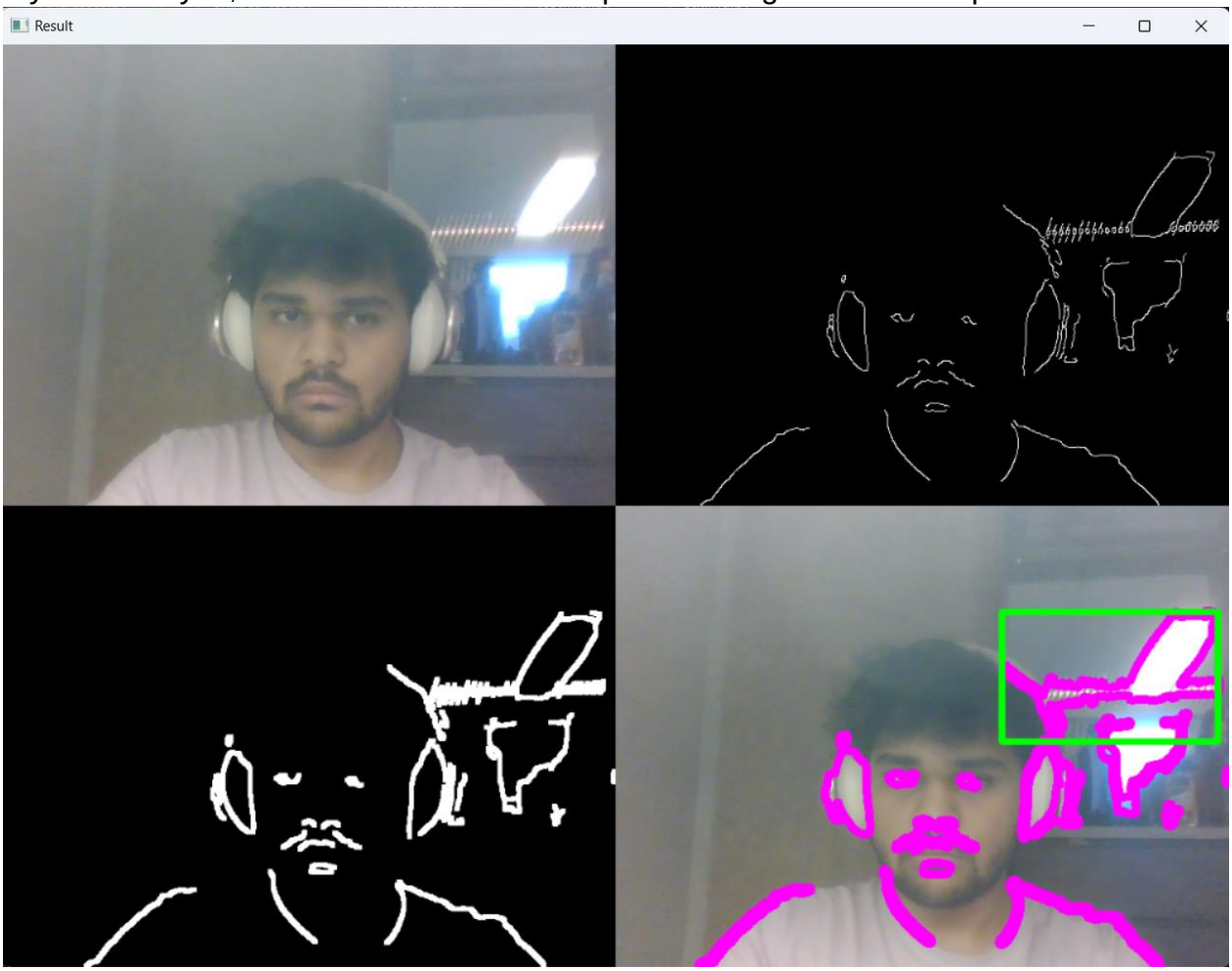
2. Run the Code

   When we run the code, we will see a small GUI screen which will ask the user weather or not to start shape recognition.



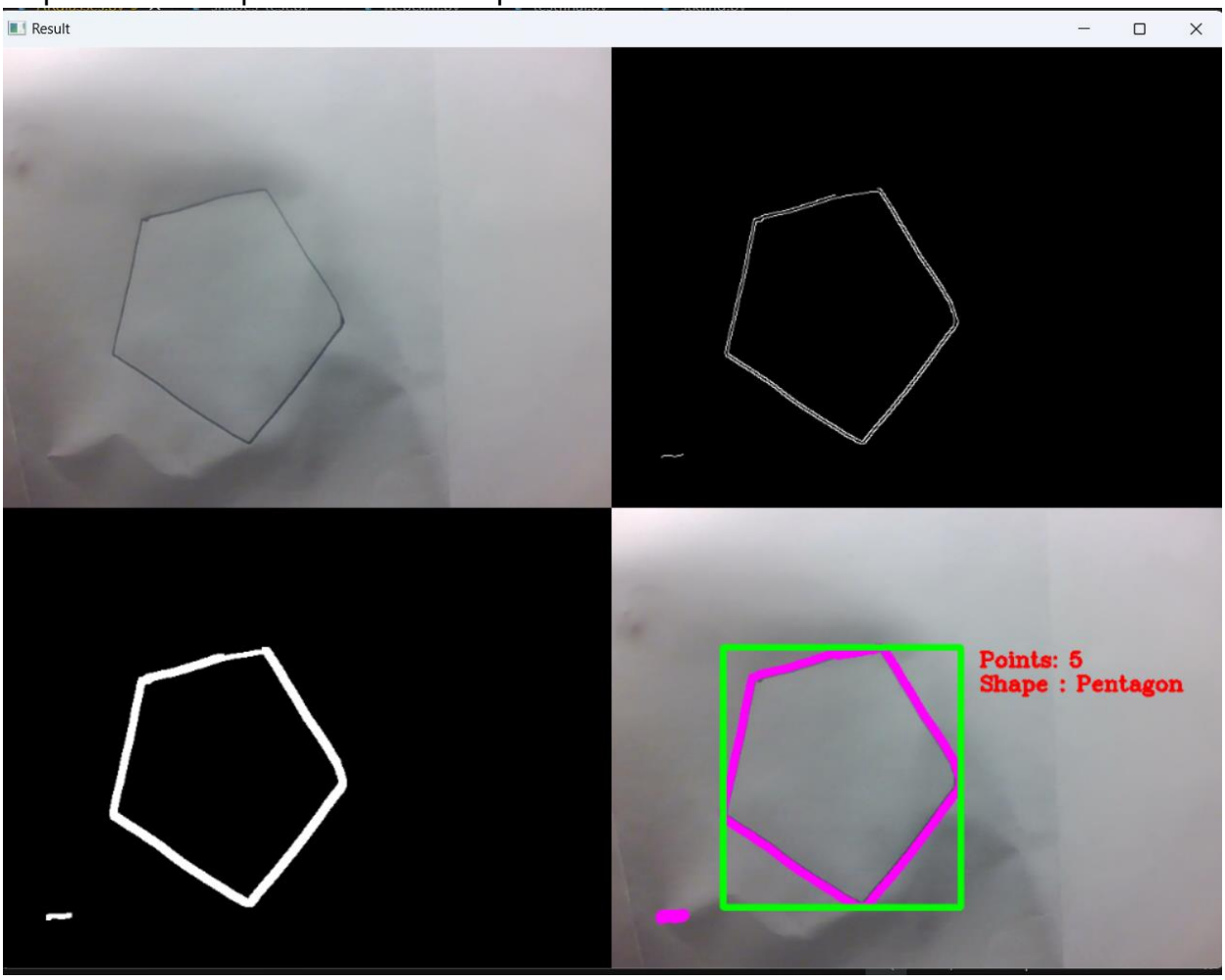
Select yes if you want to run it or select no and the code will stop.

If you select yes, a command window will open showing 4 camera outputs.

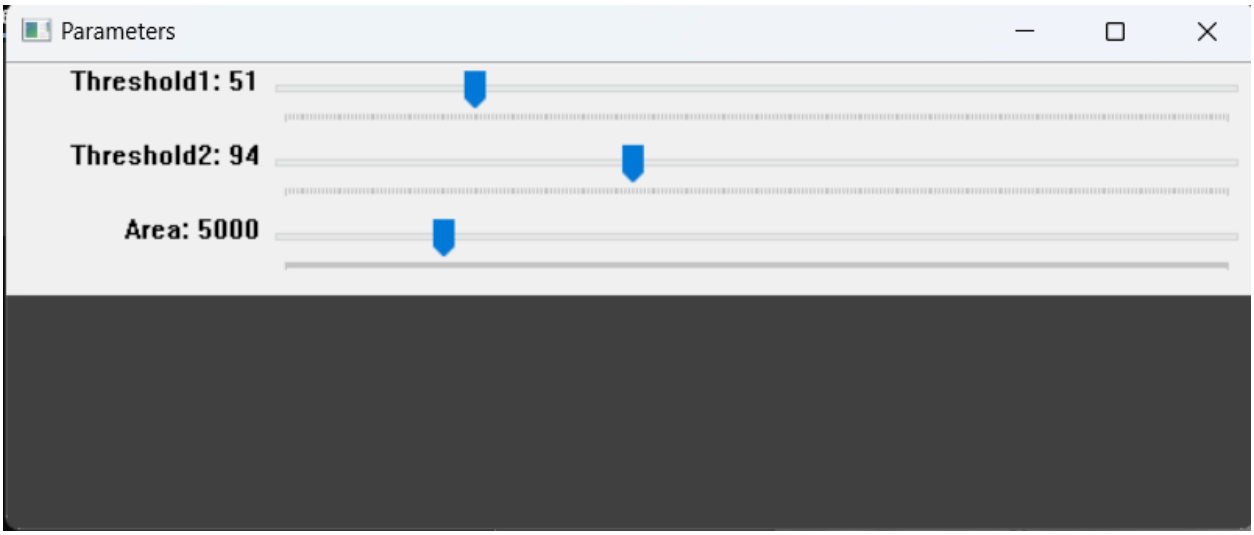


However, our main display will only be shown in the bottom right corner. The other screens are just to show the user how the shape recognision works in real time.

Test the code: “ put a picture of any shape from circle , triangle up to a Pentagon. And I the auto should be automatically be able to recognise the shape and displat the no. of points and the shape name on the 4th screen



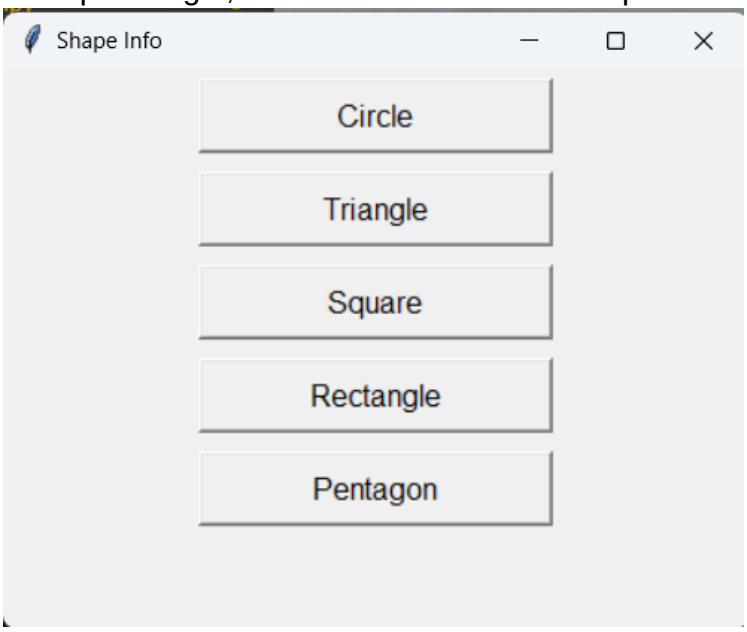
If the code can’t recognise the shape then it means that either there is no proper shape that’s being shown or the area is to small. If that’s the case then you can mess around with the settings bar which will be displayed right from the beginning.



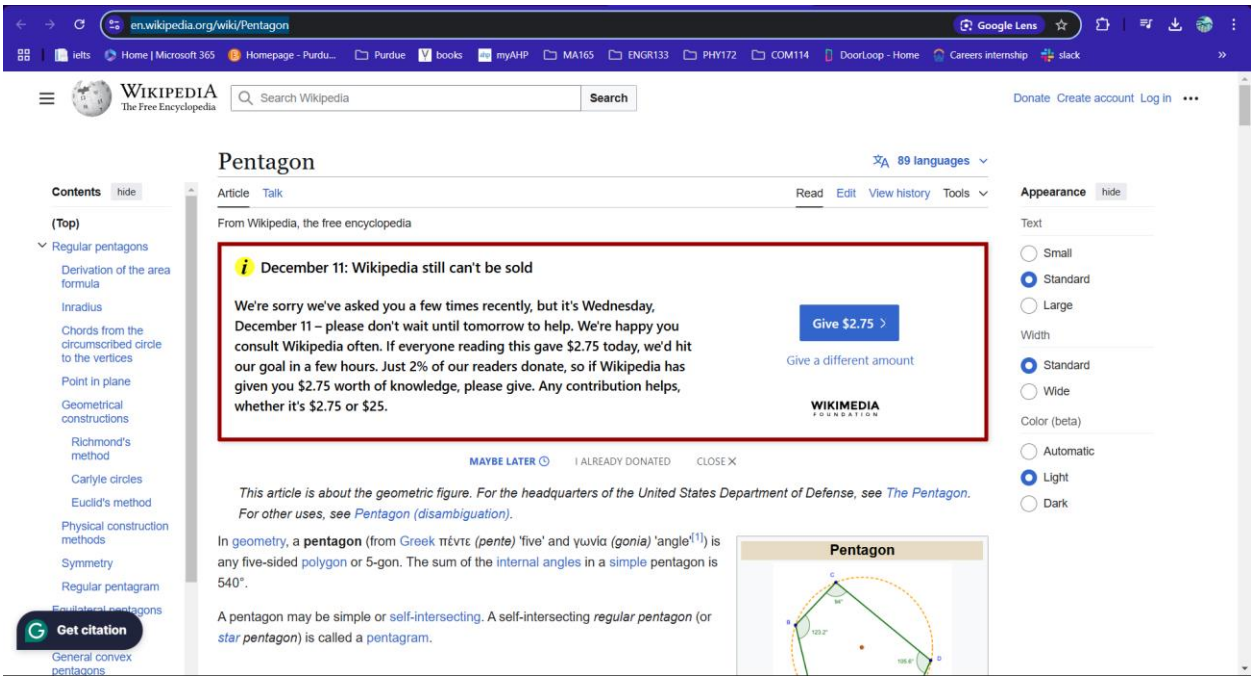
This will give more details and help with better recognizing the shape of the image. When you make changes in the slider, you'll see the change in real time on the black screens on how it effects the scanning.

Now that we know our shape, press the key"Z" on your computer.

After pressing it, the menu window should open.

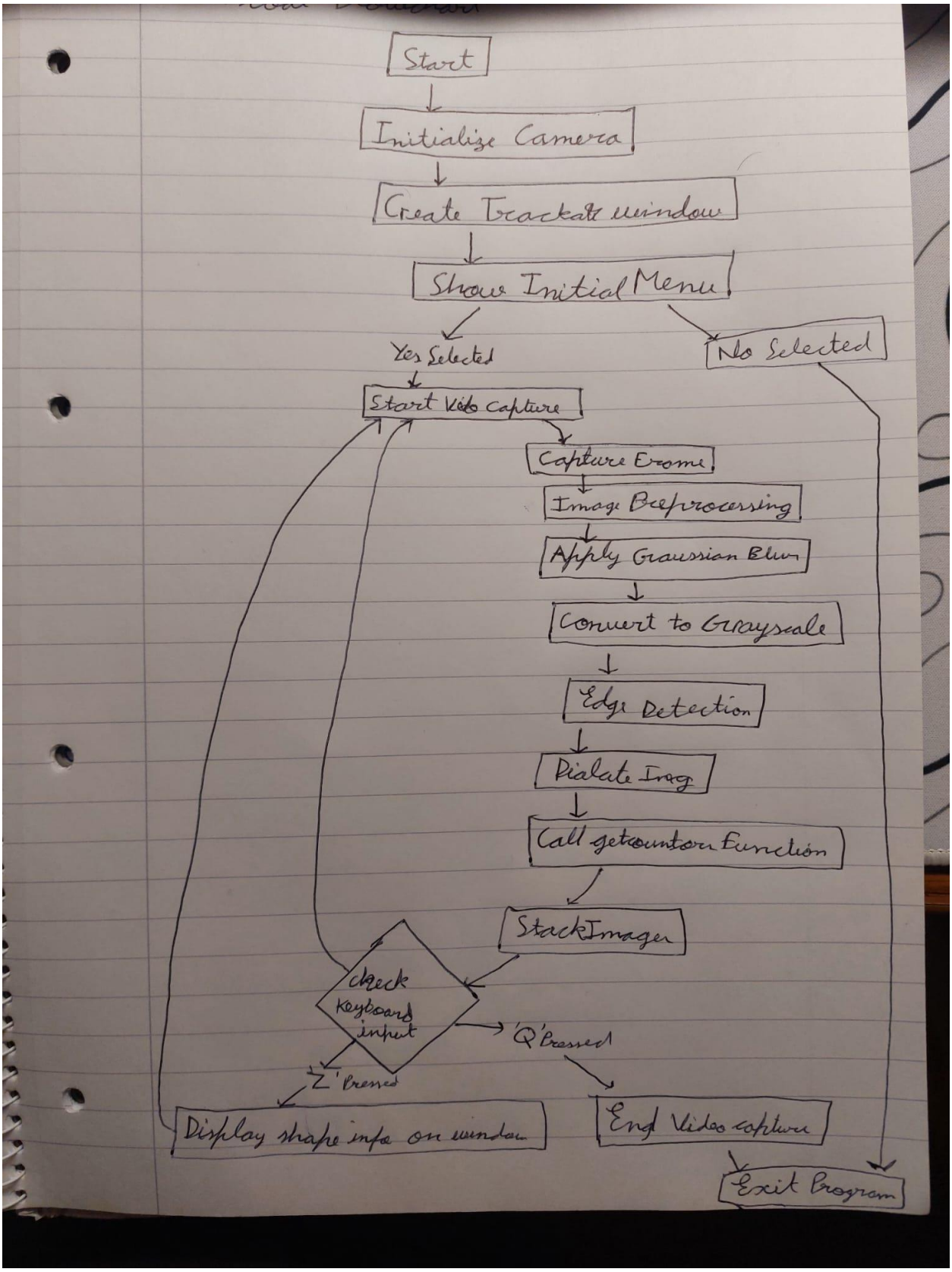


As you can see, you have options to choose from the menu and you click on the shape that you saw and it should lead you to the wikipedia page for the respective shape.



Next is the exiting of the code which is pretty simple. Click the cross button on the GUI screen and once the GUI screen is gone, just press the key" Q" and the code should automatically stop and exit everything.

3. Flowchart

4. 
Start
Initialize Camera
Create Trackbar window
Show Initial Menu
Yes Selected
No Selected
Start Video capture
Capture Frame
Image Preprocessing
Apply Gaussian Blur
Convert to Grayscale
Edge Detection
Dialate Img
Call getcountors Function
StackImages
check keyboard input
'Q' Pressed
'Z' Pressed
Display shape info on window
End Video capture
Exit Program

5. **Code**

UDF- imported – (main function after this)

```
import cv2
import numpy as np

def stkimgs(scale, imgArray):
        # Get the number of rows in the input array
        rows = len(imgArray)
        cols = len(imgArray[0])
        rowsAvailable = isinstance(imgArray[0], list)
        width = imgArray[0][0].shape[1]
        height = imgArray[0][0].shape[0]

        # If the input is a 2D array (grid of images)
        if rowsAvailable:
            # Loop through all rows and columns
            for x in range(rows):
                for y in range(cols):
                    # If the current image dimensions match the first image, scale it
                    if imgArray[x][y].shape[:2] == imgArray[0][0].shape[:2]:
                        imgArray[x][y] = cv2.resize(imgArray[x][y], (0, 0), None, scale, scale)
                    else:
                        # Resize the current image to match the dimensions of the first image
                        imgArray[x][y] = cv2.resize(imgArray[x][y],
                                                    (imgArray[0][0].shape[1], imgArray[0][0].shape[0]),
                                                    None, scale, scale)
                    # Convert grayscale images to BGR format
                    if len(imgArray[x][y].shape) == 2:
```

```
                        imgArray[x][y] =
cv2.cvtColor(imgArray[x][y], cv2.COLOR_GRAY2BGR)

            # Create a blank image with the same dimensions as
the first image
            imageBlank = np.zeros((height, width, 3), np.uint8)
            hor = [imageBlank] * rows
            for x in range(rows):
                hor[x] = np.hstack(imgArray[x])
            ver = np.vstack(hor)
        else:
            for x in range(rows):
                if imgArray[x].shape[:2] ==
imgArray[0].shape[:2]:
                    imgArray[x] = cv2.resize(imgArray[x], (0,
0), None, scale, scale)
                else:
                    # Resize the current image to match the
dimensions of the first image
                    imgArray[x] = cv2.resize(imgArray[x],
                                             (imgArray[0].shape[
1], imgArray[0].shape[0]),
                                             None, scale, scale)
                if len(imgArray[x].shape) == 2:
                    imgArray[x] = cv2.cvtColor(imgArray[x],
cv2.COLOR_GRAY2BGR)

            # Horizontally stack all the images in the 1D array
            hor = np.hstack(imgArray)
            ver = hor

        return ver

```

main-

"""

Course: ENGR 13300, Fall 2024

Program Description: This program captures video from a webcam to detect and identify geometric shapes using OpenCV. It

also includes a Tkinter GUI for starting shape recognition and displaying shape info with web links.

Assignment: 7.3.1 Py3 ind1

Team ID: LC3 - 08

Author: Amogh Meenakshi Shadangi, ashadang@purdue.edu

Date: 09/25/2024

Contributors:

Academic Integrity:

I have not used source code obtained from any other unauthorized

source, either modified or unmodified. Neither have I provided

access to my code to another. The project I am submitting

is my own original work.

"""

```
import cv2  # OpenCV for image processing
import numpy as np  # Numpy for mathematical operations
import webbrowser  # To open web pages for shapes
import tkinter as tk
```

```
import keyboard
from tkinter import messagebox
from stkimg import stkimgs  # Function to stack images

# Initialize camera dimensions
framewidth = 640
frameheight = 480

cap = cv2.VideoCapture(0)
cap.set(3, framewidth)
cap.set(4, frameheight)

# Empty function for trackbars
def empty(a):
    pass

# Create a trackbar window for dynamic parameter adjustment
cv2.namedWindow("Parameters")
cv2.resizeWindow("Parameters", 640, 240)
cv2.createTrackbar("Threshold1", "Parameters", 51, 255, empty)
cv2.createTrackbar("Threshold2", "Parameters", 94, 255, empty)
cv2.createTrackbar("Area", "Parameters", 5000, 30000, empty)
```

```
# Function for displaying the initial menu
def show_menu():
    root = tk.Tk()
    root.geometry("300x150")
    root.title("Shape Recognition Menu")

    def start_recognition():
        root.destroy()  # Close the menu

    # Callback function for "No" button
    def exit_program():
        root.destroy()  # Close the menu
        cap.release()  # Release the camera
        cv2.destroyAllWindows()  # Close all OpenCV windows
        exit()  # Exit the script

    label = tk.Label(root, text="Do you want to recognize shapes?", font=("Arial", 12))
    label.pack(pady=20)

    yes_button = tk.Button(root, text="Yes", command=start_recognition, font=("Arial", 10), width=10)
    yes_button.pack(side="left", padx=30)

    no_button = tk.Button(root, text="No", command=exit_program, font=("Arial", 10), width=10)
```

```
    no_button.pack(side="right", padx=30)


    root.mainloop()


# GUI function for displaying shape links
def display_output():
    if hasattr(display_output, 'window') and display_output.window.winfo_exists():
        return


    display_output.window = tk.Tk()
    display_output.window.geometry("400x300")
    display_output.window.title("Shape Info")
    websites = {
        "Circle": "https://en.wikipedia.org/wiki/Circle",
        "Triangle": "https://en.wikipedia.org/wiki/Triangle",
        "Square": "https://en.wikipedia.org/wiki/Square",
        "Rectangle": "https://en.wikipedia.org/wiki/Rectangle",
        "Pentagon": "https://en.wikipedia.org/wiki/Pentagon",
    }


    # Add buttons dynamically based on shapes
    for shape_name, url in websites.items():
        btn = tk.Button(
```

```
            display_output.window,
            text=shape_name,
            command=lambda url=url: webbrowser.open(url),
            font=("Arial", 12),
            width=20,
            pady=5
        )
        btn.pack(pady=5)
    display_output.window.mainloop()
if not cap.isOpened():
    print("Error: Could not open webcam.")
    exit()
# Function to detect contours and identify shapes
def getcontours(img, imgcontour):
    # Find contours in the dilated image
    contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    # Error check: Skip processing if no contours are found
    if not contours:
        print("Warning: No contours found.")
        return

    cv2.drawContours(imgcontour, contours, -1, (255, 0, 255), 7)
```

```
    for cnt in contours:
        area = cv2.contourArea(cnt)
        min_area = cv2.getTrackbarPos("Area", "Parameters")  # Adjustable minimum area
        if area > min_area:
            cv2.drawContours(imgcontour, cnt, -1, (255, 0, 255), 7)
            perimeter = cv2.arcLength(cnt, True)
            approx = cv2.approxPolyDP(cnt, 0.02 * perimeter, True)
            sides = len(approx)  # Number of sides
            x, y, w, h = cv2.boundingRect(approx)
            cv2.rectangle(imgcontour, (x, y), (x + w, y + h), (0, 255, 0), 5)

            # Identifing the shape based on the number of sides
            if sides==3:
                cv2.putText(imgcontour, "Points: " + str(sides), (x+w+20, y+20),
cv2.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)
                cv2.putText(imgcontour, "Shape : Triangle ", (x+w+20, y+45), cv2.FONT_HERSHEY_COMPLEX,
0.7, (0,0,255), 2)
            elif sides==4:
                cv2.putText(imgcontour, "Points: " + str(sides), (x+w+20, y+20),
cv2.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)
                asp_ratio = w/float(h)
                if asp_ratio>0.95 and asp_ratio<1.05:
```

```
            cv2.putText(imgcontour, "Shape : Square ", (x+w+20, y+45), cv2.FONT_HERSHEY_COMPLEX,
0.7, (0,0,255), 2)
        else:
            cv2.putText(imgcontour, "Shape : Rectangle ", (x+w+20, y+45),
cv2.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)
    elif sides==5:
        cv2.putText(imgcontour, "Points: " + str(sides), (x+w+20, y+20),
cv2.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)
        cv2.putText(imgcontour, "Shape : Pentagon ", (x+w+20, y+45),
cv2.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)
    elif sides==6:
        cv2.putText(imgcontour, "Points: " + str(sides), (x+w+20, y+20),
cv2.FONT_HERSHEY_COMPLEX, 0.7, (0,0,255), 2)
        cv2.putText(imgcontour, "Shape : Hexagon ", (x+w+20, y+45), cv2.FONT_HERSHEY_COMPLEX,
0.7, (0,0,255), 2)
    else:
        cv2.putText(imgcontour, "Points: 0", (x+w+20, y+20), cv2.FONT_HERSHEY_COMPLEX, 0.7,
(0,0,255), 2)
        cv2.putText(imgcontour, "Shape : Circle ", (x+w+20, y+45), cv2.FONT_HERSHEY_COMPLEX,
0.7, (0,0,255), 2)
    keyboard.on_press_key('z', lambda _: display_output())
show_menu()
```

```
# Main loop for video capture and processing
while True:
    success, img = cap.read()
    imgcontour = img.copy()
    blur = cv2.GaussianBlur(img, (7, 7), 1)
    grayimg = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)
    threshold1 = cv2.getTrackbarPos("Threshold1", "Parameters")
    threshold2 = cv2.getTrackbarPos("Threshold2", "Parameters")
    cannyimg = cv2.Canny(grayimg, threshold1, threshold2)
    kernel = np.ones((5, 5))
    imgdil = cv2.dilate(cannyimg, kernel, iterations=1)
    getcontours(imgdil, imgcontour)
    imgstack = stkimgs(0.8, ([img, cannyimg],
                             [imgdil, imgcontour]))
    cv2.imshow("Result", imgstack)  # Display stacked images
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```