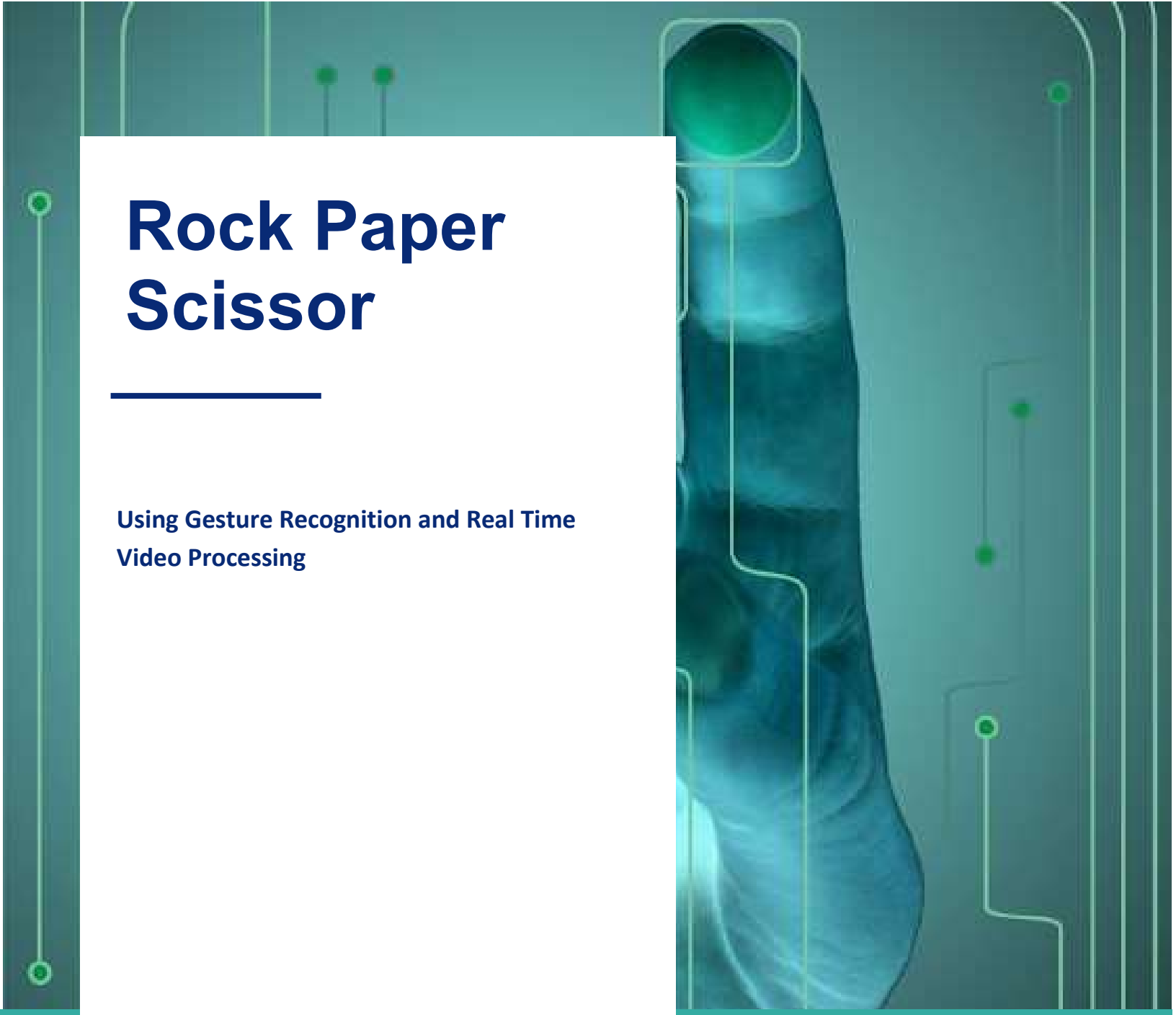


Rock Paper Scissor

Using Gesture Recognition and Real Time Video Processing

DECEMBER 20

Ashish Upadhyay (au628)
Yashvardhan Jain (yj1490)
Jaeki Shen (js9411)

NYU

About the Project

This project of gesture recognition provides an intriguing, interactive way to play the simple game of Rock Paper Scissor. When the program is executed the webcam is turned on and the user can see themselves on the screen. A small box with a green outline is visible at the top right corner which is the **region of interest**. Inside this box, the game is played. The program recognizes gestures in the form of hand symbols such as, a fist represents 'rock', two fingers represent 'scissor' whereas all five fingers/open palm represent 'paper'.

A **threshold window** also opens up along with the webcam window. This threshold window is used for the purpose of calibration of the system. Once calibrated it appears completely black as it is functioning on the greyscale concept converting the background to a dark color. After which when we make our gesture inside the region of interest box on the top right corner. How the gesture is detected is discussed later in details.

The concept used to map the gesture made is based on peaks and troughs concept which are used to detect the number of fingers held up., that further determine whether the program detects rock/paper/scissor. Real time recognized values such as for one, three and four fingers are also input in the program in order for the program to round of the calculated threshold values and then compare them to the nearest possible options. For example, if the program computes the threshold value to be somewhere between 2.1-3.5 (buffer of 10 previous values) then it inputs the gesture to be a scissor. Anything above 3.5 is recognized as paper. Similarly, when the program finds the computed value to be somewhere between 0-1.4 then it inputs as a rock else it inputs the gesture as scissor.

Once the program recognizes the gesture of the user, the program itself generates a random value which corresponds to a particular gesture using random function generator. Finally, by comparing the random program gesture to the user the gesture the program outputs the result.

If the user wins, then the win counter is incremented, if the user loses then the loss counter is incremented else the draw counter is incremented. The three counter values are reflected on the scoreboard in the form of wins/loss/draw. This counter value is updated in real time to give a score after every round. We have also programmed some random quirky remarks that the program displays along with the result of every round. There are 6 different remarks programmed for all three conditions and these remarks are also selected using a random function at the end of each round.

```
# messages to disp
Win_msg=["Looks like I Win Again", "Do You Even Know The Rules", "Today is Not Your Day",
        "Stop Crying, Man Up!", "You Came To NYU for This?", "Machine Will rise and Take Over Soon!!" ]

Draw_msg=["You Are a Worthy Opponent", "Ditto, Its A Match", "We Think Alike",
          "Stop Copying ME", "Humans Are Match For AI After All", "I Know Your Next Move" ]

Lose_msg=["GHz In Processing Speed, For Nothing", "You Won the Battle, Not War", "Rub Off that Stupid Smile",
         "Next Move And You Are Dead", "WOW!! Even the Worst Human can Defeat Me", "Sure Add This To Your CV"  ]
```

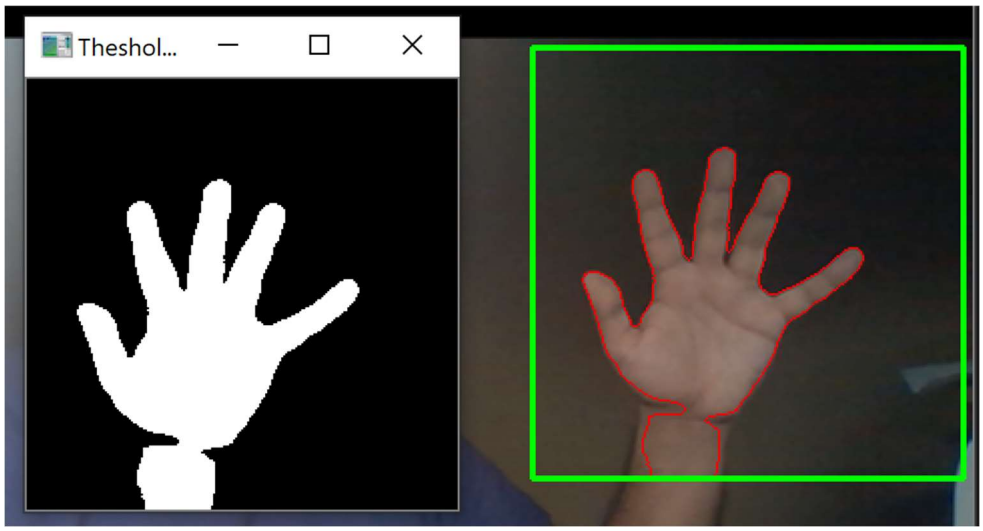
Implementation

To begin the project implementation in Python, the **OpenCV and imutils** library must be imported. As Our Project is based on it.

The OpenCV library is responsible for real-time computer vision functions needed to perform gesture recognition, while the imutils library provides the function needed to resize the frame of interest.

The first step in the implementation is to be able to identify and segment the hand from the observed frame and count the number of fingers. First, we define the background by initializing it through the cv2.accumulateweighted() function which returns computed value of the running average of the background model and current frame.

The function for segmenting the hand in the frame is then performed through the cv2.threshold() function, where the threshold value determines the level at which certain objects are detected as the foreground. Once this foreground image is determined, cv2.findContours() function is used to take the contour of the hand. Cv2.cvtcolor() as well as cv2.GaussianBlur() are used to convert the region of interest into a blurred grayscale image such that high-frequency components in the image are attenuated.

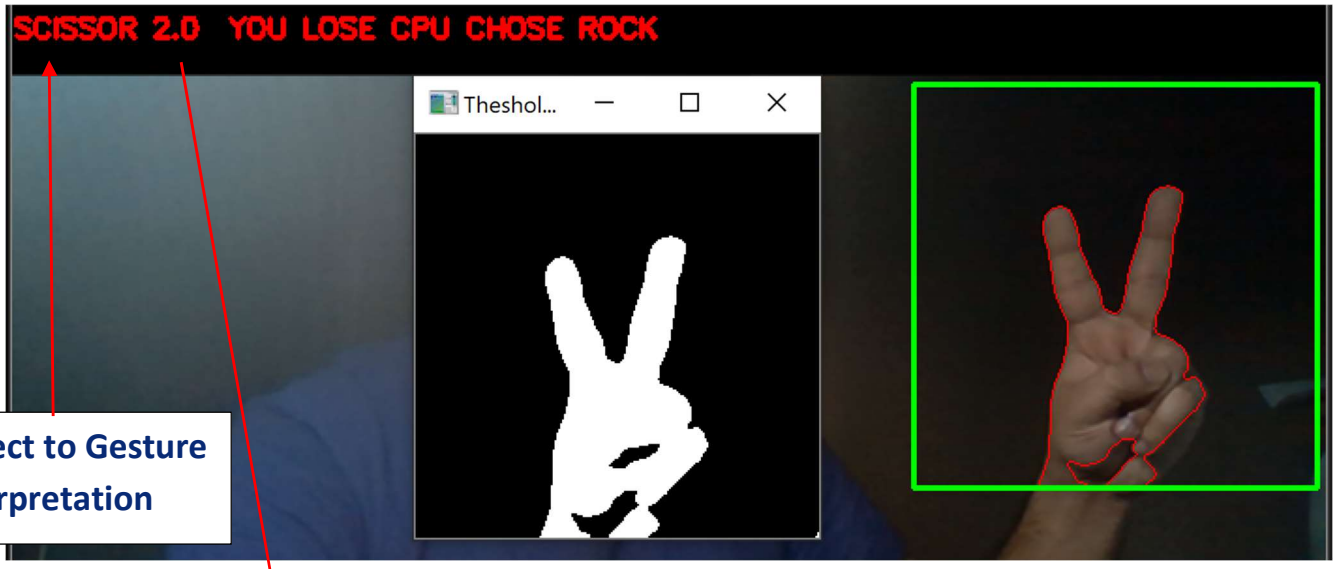


Object Detection in Greyscale **Contours around Object**

In the main loop, these functions are run in an infinite loop, and set up such that if the current detected frame exceeds the threshold value specified in the code, it will activate

the contour drawing to display the binary image of the hand through the cv2.drawContours and cv2.imshow functions. To count the fingers in the image, the cv2.convexHull function is used to find the extreme points in the contour of the segmented hand.

Once these points are known, the palm is identified using this information and a circle is drawn around the palm. Using the cv2.bitwise_and () function between the circle and thresholder hand image, the number of contours in the circle is found.
To determine the number of fingers more accurately, the condition is set such that the finger count is implemented only if the distance between the circle and the contour point is greater than 25% of its circumference.

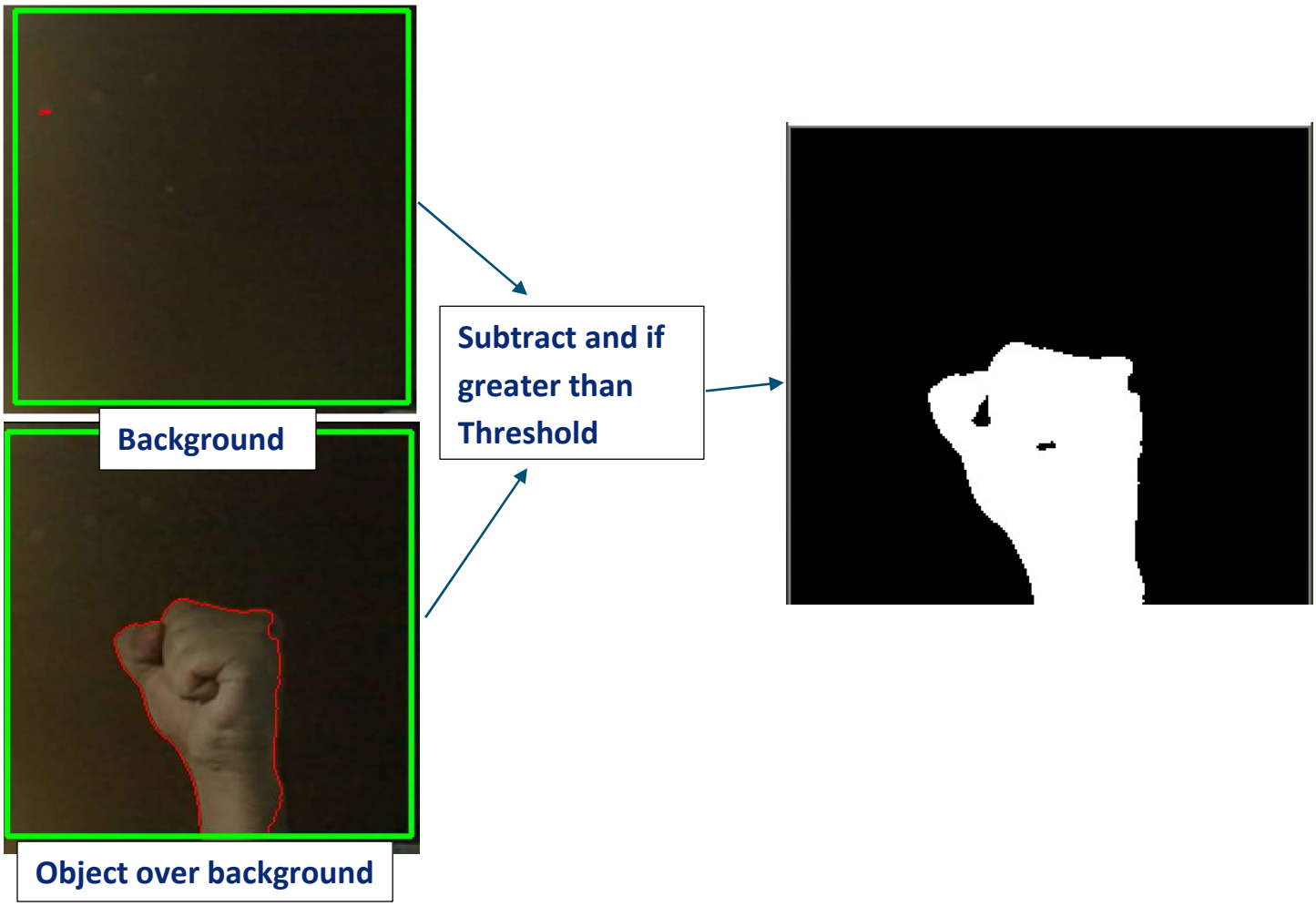


Background Subtraction

In the calibration stage the background is captured for 50 frames these frames are put into a running average function which computed the running average over the current frame and the previous frames

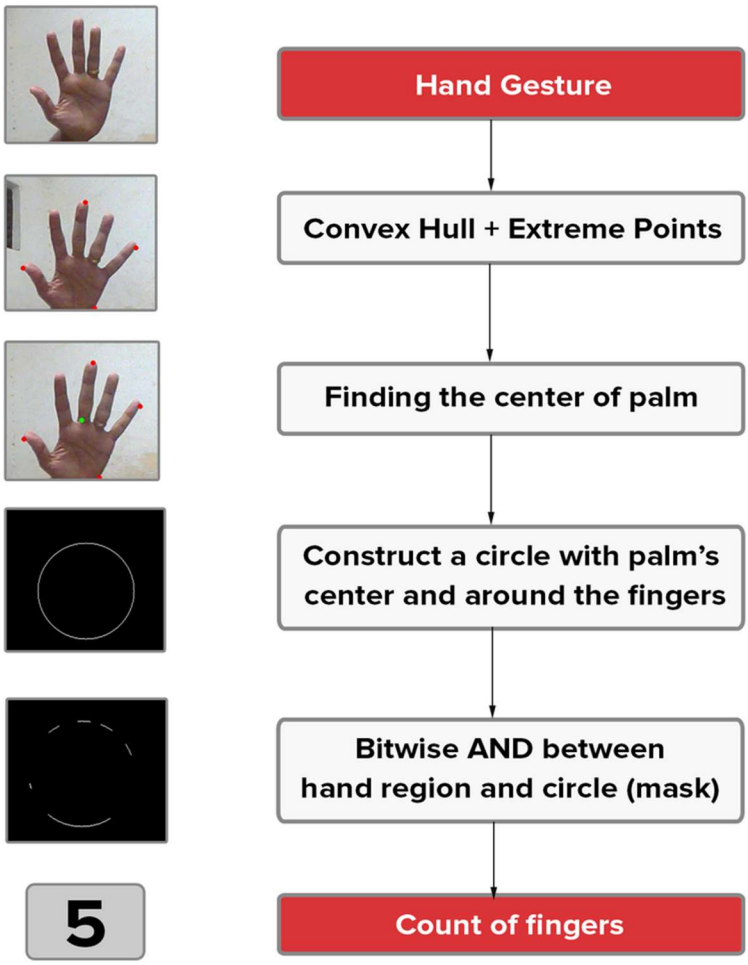
Basically, training our system that those inputs (Average of 50 frames) are our Background.

After training we bring our hand to the ROI box and system understands that hand is the new entry to the background or it's a foreground.



Counting the Fingers

There are various methods for counting the fingers. However, this is a faster approach to perform hand gesture recognition as proposed by Malima et.al. The methodology to count the fingers (as proposed by Malima et.al) is shown in the figure below.



The Steps of implementations are as follows

- We find the convex hull of the segmented hand region (which is a contour) and compute the most extreme points in the convex hull (Extreme Top, Extreme Bottom, Extreme Left, Extreme Right).
- Find the center of palm using these extremes points in the convex hull.
- Using the palm's center, construct a circle with the maximum Euclidean distance (between the palm's center and the extreme points) as radius.

- Perform bitwise AND operation between the thresholder hand image (frame) and the circular ROI (mask). This reveals the finger slices, which could further be used to calculate the number of fingers shown.

Contours

The outline or the boundary of the object of interest. This contour could easily be found using OpenCV's cv2.findContours() function.

Bitwise-AND

Performs bit-wise logical AND between two objects. You could visually think of this as using a mask and extracting the regions in an image that lie under this mask alone. OpenCV provides cv2.bitwise_and () function to perform this operation.

Euclidean Distance

This is the distance between two points given by the equation

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Scikit-learn provides a function called pairwise.euclidean_distances() to calculate the Euclidean distance from one point to multiple points in a single line of code - Pairwise Euclidean Distance. After that, we take the maximum of all these distances using NumPy's argmax() function.

Convex Hull

Convex Hull will look similar to contour approximation, but it is not (Both may provide same results in some cases). Here, cv.convexHull() function checks a curve for convexity defects and corrects it. Convex curves are the curves which are always bulged out, or at-least flat. And if it is bulged inside, it is called convexity defects. For example, check the below image of hand. Red line shows the convex hull of hand. The double-sided arrow

marks show the convexity defects, which are the local maximum deviations of hull from contours.

image

hull = cv.convexHull(points[, hull[, clockwise[, returnPoints]]

Conclusion

This was our attempt to introduce the concept of gesture recognition to our peers. This concept can be widely used in gesture recognition, motion detection and real time video processing. This program can easily be scaled to collision avoidance in self driven cars however the live update of the background image and foreground calculation poses to be challenge.

Our system is highly sensitive to ambient surrounding such as change in background lighting can introduce noise in the computed contours. This problem can be averted by increasing the threshold values, but this reduces the object detection capability. Real time multiple calibration could help, and the calibration time can be reduced by decreasing the frame rate from 50 to 10. However, this also effects the precision of object detection.

References

[1] Creators of open CV library.
[2] Creators of numpy library.
[3] Creators of imutils library.
[4] Creators of Tkinter library.
[5] A FAST ALGORITHM FOR VISION-BASED HAND GESTURE RECOGNITION FOR ROBOT CONTROL Asanterabi Malima, Erol Özgür, and Müjdat Çetin Faculty of Engineering and Natural Sciences, Sabancı University, Tuzla, İstanbul, Turkey