Ball Detection and Tracking using Image Processing

Abstract- **This paper explains the methdology used for detection and tracking of ball (Rugby ball to be specific) using Image Processing(IP) and the communication between Raspberry Pi and Arduino for the required movement of bot**

I. INTRODUCTION

Image processing is a way of processing the raw data obtained from the Computer Vision (CV) algorithms so that it can be used according to the requirement. Raw data here refers to the multi-dimensional matrix obtained from CV, for example in case of a grayscale image, a 2-D matrix is obtained with pixel values ranging from 0 to 255, whereas in case of a RBG image, a 3-D matrix is obtained with each pixel containing R, G and B values, each ranging from 0 to 255. The matrices obtained are processed using various algorithms for obtaining the desired result.

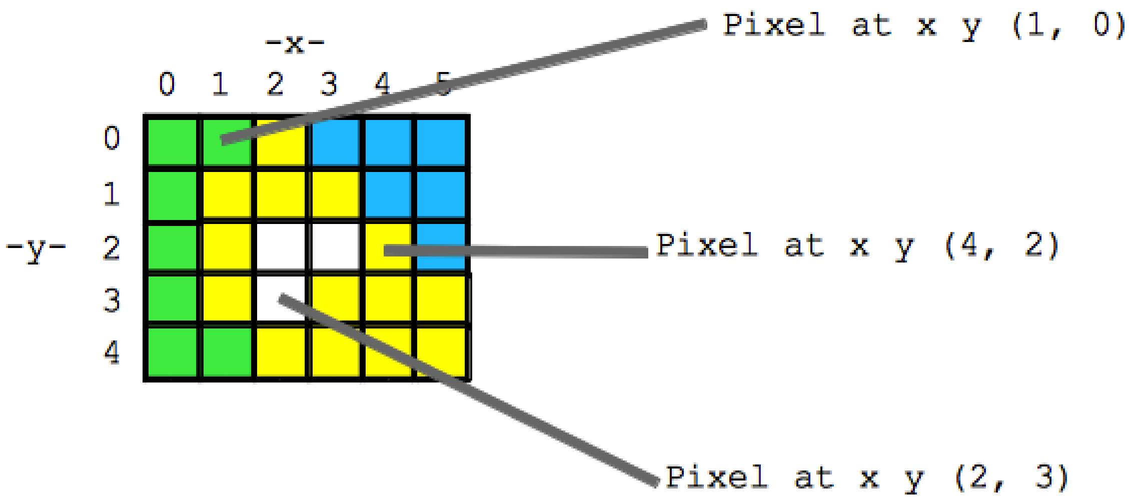
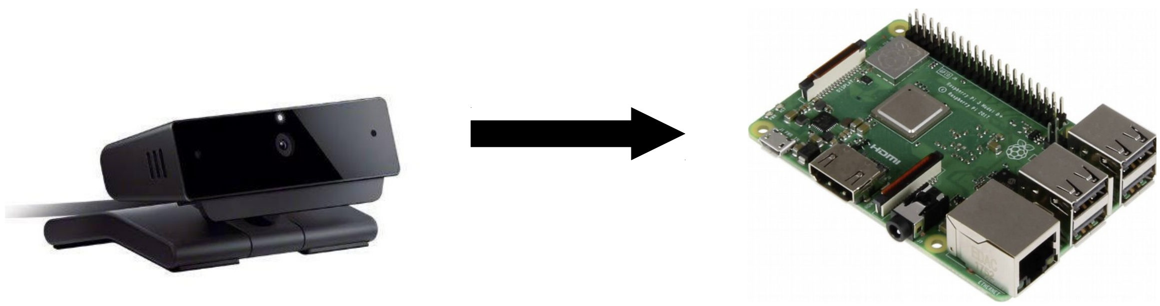


Fig. 1: Image as a matrix

II. ELECTRONICS CONFIGURATION

A. Image processing part:

The setup consists of a micro-processor (Raspberry Pi 3B+) which handles the image processing part and a USB webcam. A pair of array of LED lights were mounted on each side of the webcam to provide proper illumination on the object.



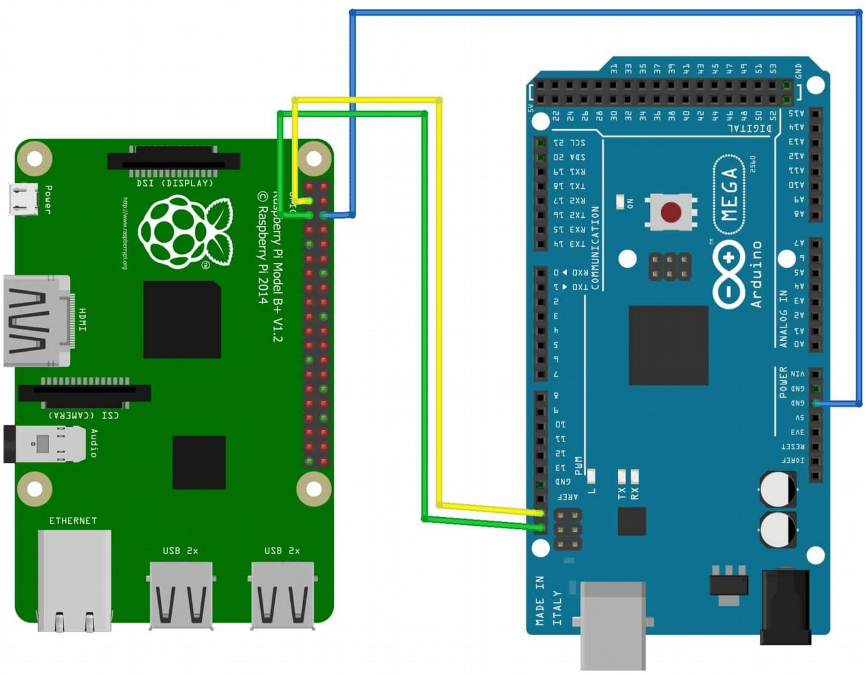
USB webcam

Raspberry Pi 3B+

Fig. 2: Connection for IP

B. Communication part:

Here, an Arduino Due is connected to Raspberry Pi via I2C communication protocol. The Raspberry Pi acts as the master and Arduino Due as the slave during the communication. This method of communication was used because the setup was fairly easy (only three connecting wires are required) and the requirement of high speed communication was fulfilled.



Yellow wire- SDA
Green wire – SCL
Blue Wire -GND

Fig. 3: Circuit diagram for I2C communication

III. OBJECT DETECTION AND TRACKING

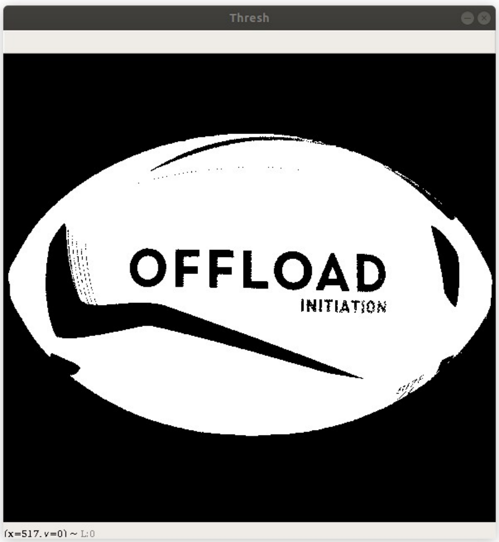
A. Algorithm:

The working of the algorithm is as follows-

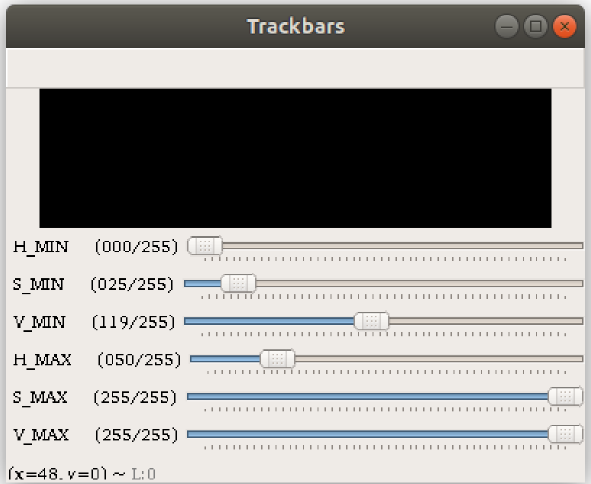
1. An image is captured from the webcam.
2. Gaussian blur is applied to the image to reduce high frequency noise.
3. The image is converted from RGB colour space to HSV colour space for better image segmentation.
4. Now, the localization of ball is performed by creating a mask based on the upper and lower limits of HSV colour values of the ball.
5. Further iterations to reduce noise are performed using erosions and dilations to remove any small blobs that may be left on the mask.
6. Contours of the objects in the masked image are computed.
7. If at least one contour was found, then we find the largest contour which is our ball.
8. A minimum enclosing ellipse is computed on the masked ball to get the required ellipse parameters which corresponds to the dimensions of our ball. The centre point of ellipse obtained from here is stored as *centre 1*.
9. Now, the centroid of the largest contour obtained is calculated and is stored as *centre 2*.
10. Finally, the average of *centre 1* and *centre 2* is taken to get the desired centre of the detected ball. The centre point obtained here is the parameter which is used to track the ball.

The most important part of the algorithm is to properly mask the ball for which the upper and lower limits of the HSV colour of the ball needs to calculated precisely. This range was calculated manually using the range-detector script and the desired range limits were noted and fed into our algorithm.

Original Image



Thresholded Image



Trackbars with adjusted values

Fig. 4: Range-detector output screens

In our case, the range-detector script was used on the live video stream from webcam for adjusting the HSV values and not from a particular image of the ball.

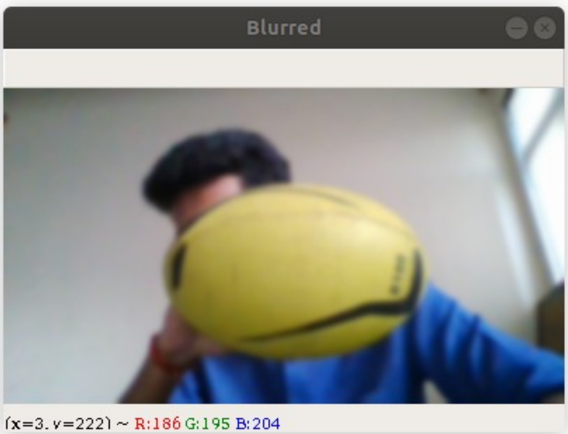
B. Implementation of Algorithm:

For implementing our algorithm, OpenCV in Python 3 was used for ease of coding. OpenCV provides a huge number of Image Processing methods and the methods used in our code in proper order are as follows-

- Gaussian Blur:

  ```
  blurred = cv2.GaussianBlur(frame, (11, 11), 0)
  ```

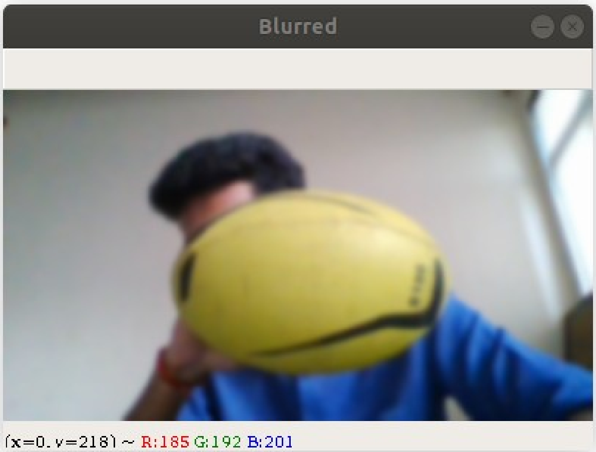
  The above method applies a Gaussian Blur for reducing the high frequency noise so that we can focus more on the structural objects in our image.



- <u>RGB to HSV conversion</u>:

  ```
  hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
  ```

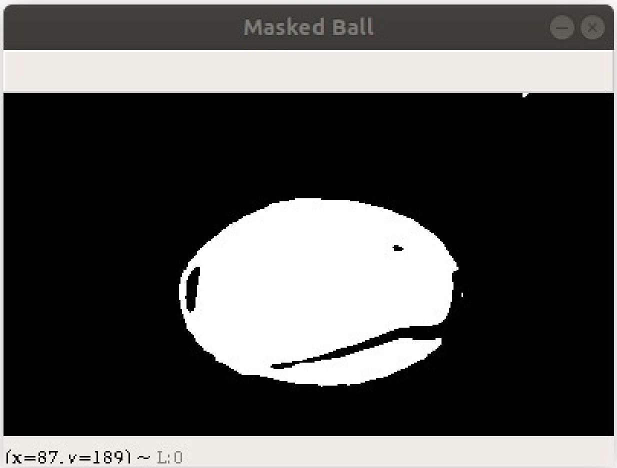
  This method is used to convert the RGB colour space to HSV colour space. HSV colour space is much more suitable for image segmentation purpose as it does show a huge change with a change in lightning conditions.



- <u>Thresholding the ball</u>:

  ```
  greenLower = (18, 60, 82)
  greenUpper = (56, 255, 255)
  mask = cv2.inRange(hsv, greenLower, greenUpper)
  ```

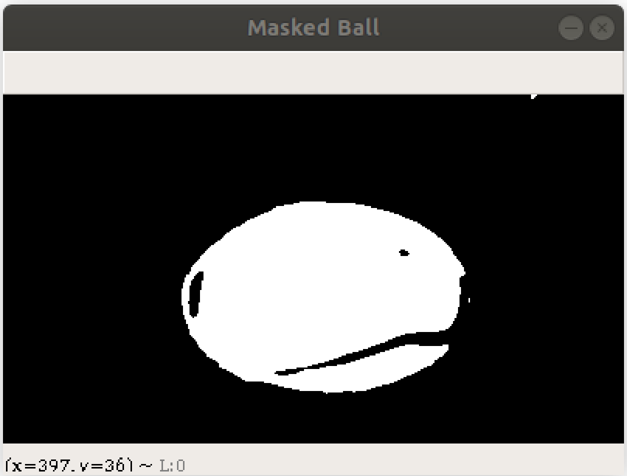
  The HSV colour values of the ball obtained from the *range-detector* script are stored here as `greenLower` and `greenUpper`. Then the method `cv2.inRange()` is used to mask the ball.

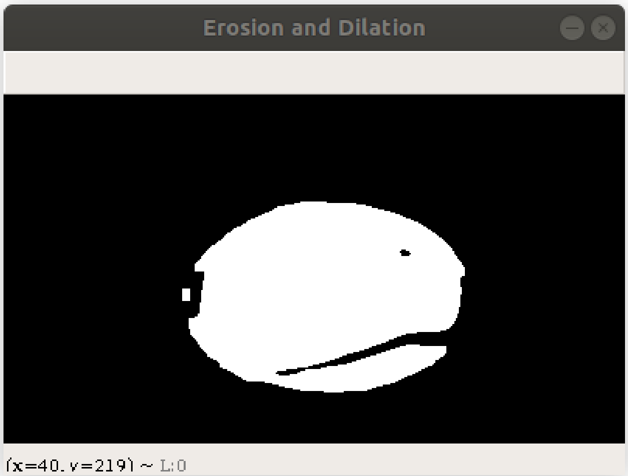


- Erosion and Dilation-

    mask = cv2.erode(mask, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)

    After masking the ball, there maybe still some small blobs left in the thresholded image which are now removed by applying iterations of erosion and dilation on the masked image.



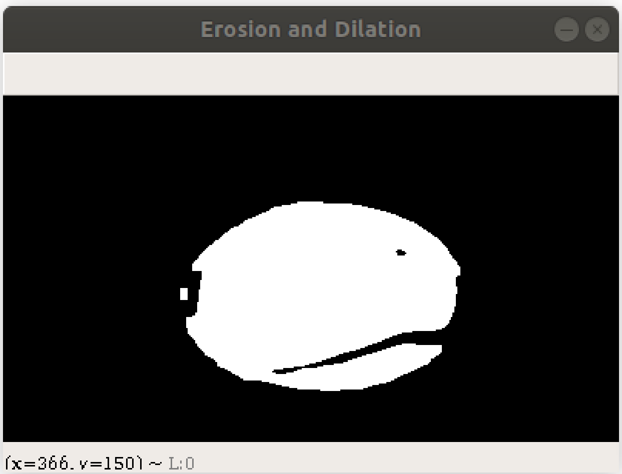


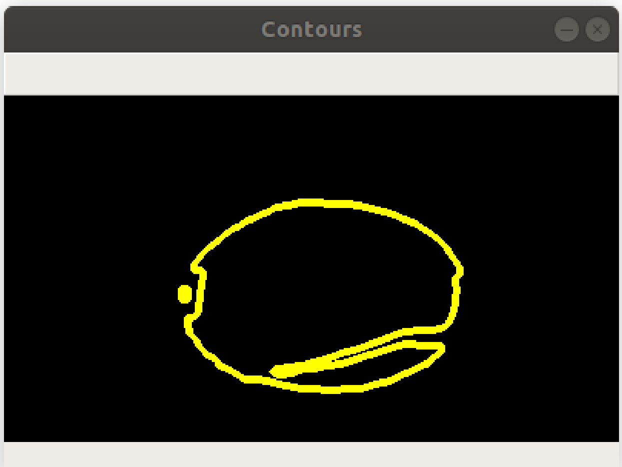
Here you can see that a small white blob on the right upper part of the masked image is removed by applying these methods.

- Finding Contours:

    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
            cv2.CHAIN_APPROX_SIMPLE)

    Here, the contours in the masked image are calculated and are collected in the variable cnts. Contours are the boundaries of the objects in the masked image.





- Finding largest contour and fitting an ellipse:

```
c = max(cnts, key=cv2.contourArea)

e = cv2.fitEllipse(c)
((x1,y1),(ma,MA),angle) = e
```

Now we find the largest contour by taking the max of all the coutours found. After that we fit a minimum area ellipse on the largest coutour (which is the ball) and then extract the required ellispe parameters.
Note that the centre obatined from here is stored as (x1, y1) which is *centre 1*.

- Finding the centre of the ball:

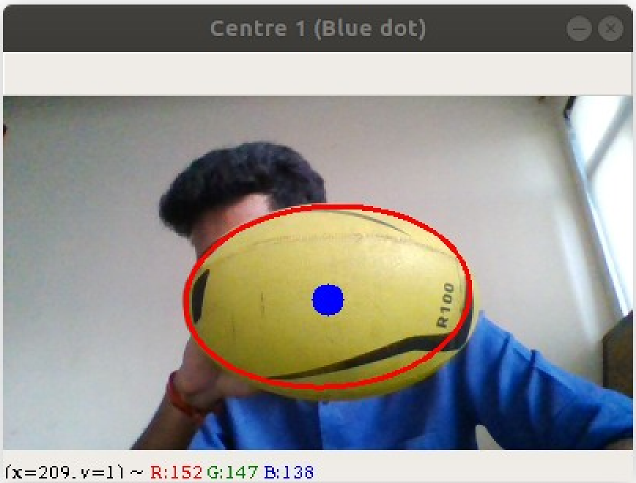
```
M = cv2.moments(c)
(x2,y2) = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

(x,y)=((x1+x2)/2,(y1+y2)/2)
```

Here, for calculating the centroid, first the Image Moments are calculated and then the *centre 2* is calculated using the formula-

$$C_x = \frac{M_{10}}{M_{00}}$$

$$C_y = \frac{M_{01}}{M_{00}}$$

Finally we take the average of *centre 1* and *centre 2* which gives us our final centre which is stored here as (x,y).



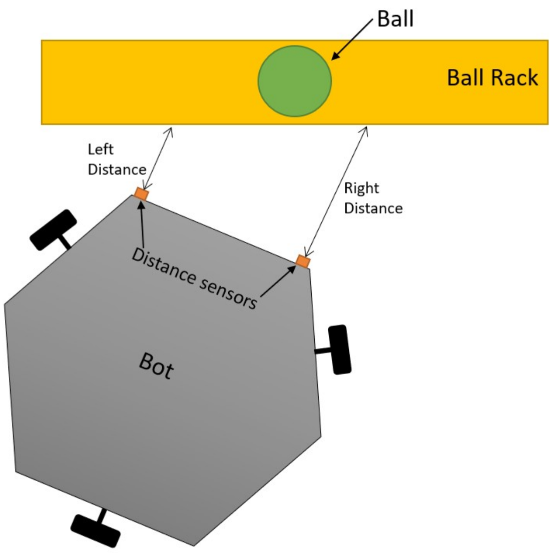
With this, the ball is successfully detected and visualized along with the centre point for tracking the ball.
With this algorithm, we achieved frame rates between **20-30 FPS** with a frame width of 400 pixels on a Raspberry Pi model 3B+

C. Communication and Arduino algorithms:

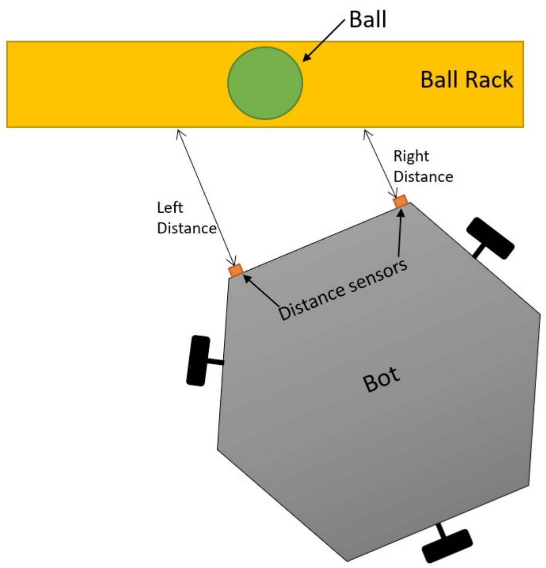
For communication between Raspberry Pi and Arduino, I2C communication was used. For that, the smbus library was used in the Python 3 code to send data from the Raspberry Pi. On the Arduino side, Wire library was used.

The data communicated here was the centre co-ordinates of the ball, sending one byte at a time to the Arduino.

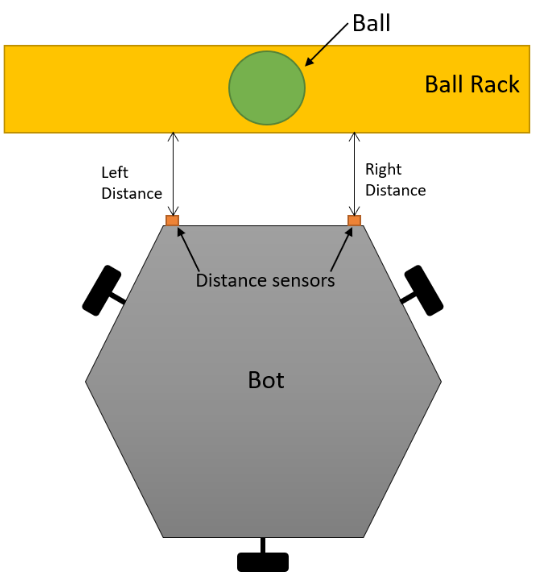
On the Arduino side, a PID controller was implemented which took the centre co-ordinate of the ball as the feedback to position the bot in front of the ball. A pair of VL53L0X Laser-based distance sensor was mounted on each side in front of the bot and was used to calulate accurate distance and rotation of the bot with respect to the ball. Various orientations of the bot are discussed below:



In this orientation, the bot is alligned in front of the ball but it is on the left side with respect to the ball rack and therefore the distance recorded by the right sensor will be greater than the left one. Thus, the bot is required to move to the right side so that its front surface becomes parallel to the ball rack



In this orientation also, the bot is alligned in front of the ball but now it is on the right side with respect to the ball rack and therefore the distance recorded by the left sensor will be greater than the right one. Thus, the bot is now required to move to the left side so that its front surface becomes parallel to the ball rack.



In this orientation, the ball is properly alligned in front of the ball as well as it is perfectly parallel to the ball rack. So the required position of the bot is achieved and the further operations can be now be performed.

IV. DRAWBACKS AND CHALLENGES FACED

A few drawbacks of the codes and challenges faced during development are as follows-

- The HSV range values of the ball are subject to change according to the colour of the ball used. Therefore this range needs to be calculated manually everytime a new coloured ball is used for detection purpose.
- The Image Processing algorithm discussed here works only if a single ball is present in the frame. Therefore the algorithm needs to be worked on and modified accordingly so that it works well in presence of multiple balls in the same frame.
- Currently there is no PID controller implemented for the part of distance and rotation of the bot with respect to the ball. Adding a PID controller for that with distance from the two laser-based sensors as feedback will increase the paramaters to be tuned but can be done.
- During development, Ultrasonic sensors were first used for providing distance measurement but these sensors were not capable of providing accurate and constant measurements in short range (0 to 15 cm). Therefore, we switched to VL53L0X infrared laser-based sensor which provided the desired measurements.
- As the bot used to close in towards the ball, a shadow of the picking mechanism mounted in the front of the bot used to fall on the ball, which made it difficult for the Image Processing algorithm to detect the ball. To overcome this, a pair of an array of white LED lights were mounted on each side of the webcam to provide proper illumination on the ball.

V. REFERENCES

[1]. Ball Tracking with OpenCV- pyimagesearch

[2]. Creating Bounding rotated boxes and ellipses for contours

[3]. Controlling an Arduino from a Pi3 using I2C