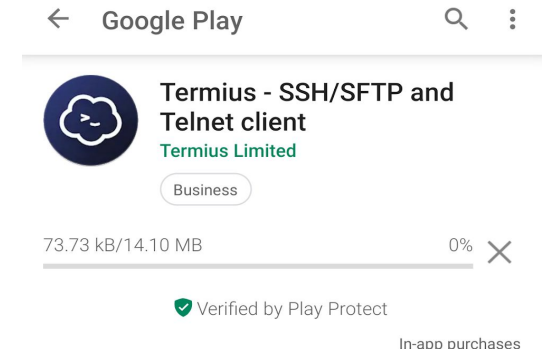
Integrating to Open-CV to Raspberry Pi

Robotics Club

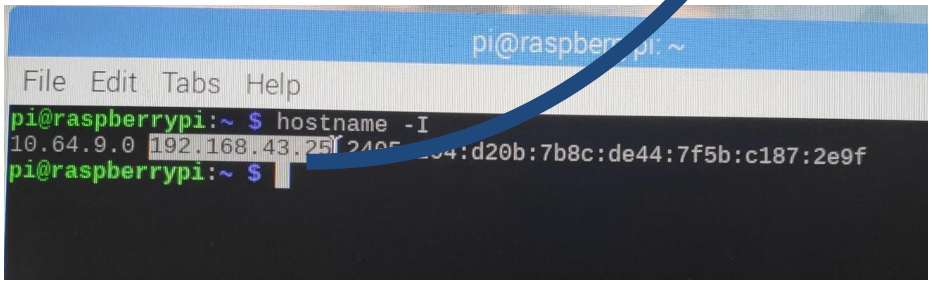
Connecting Termius to Command Line

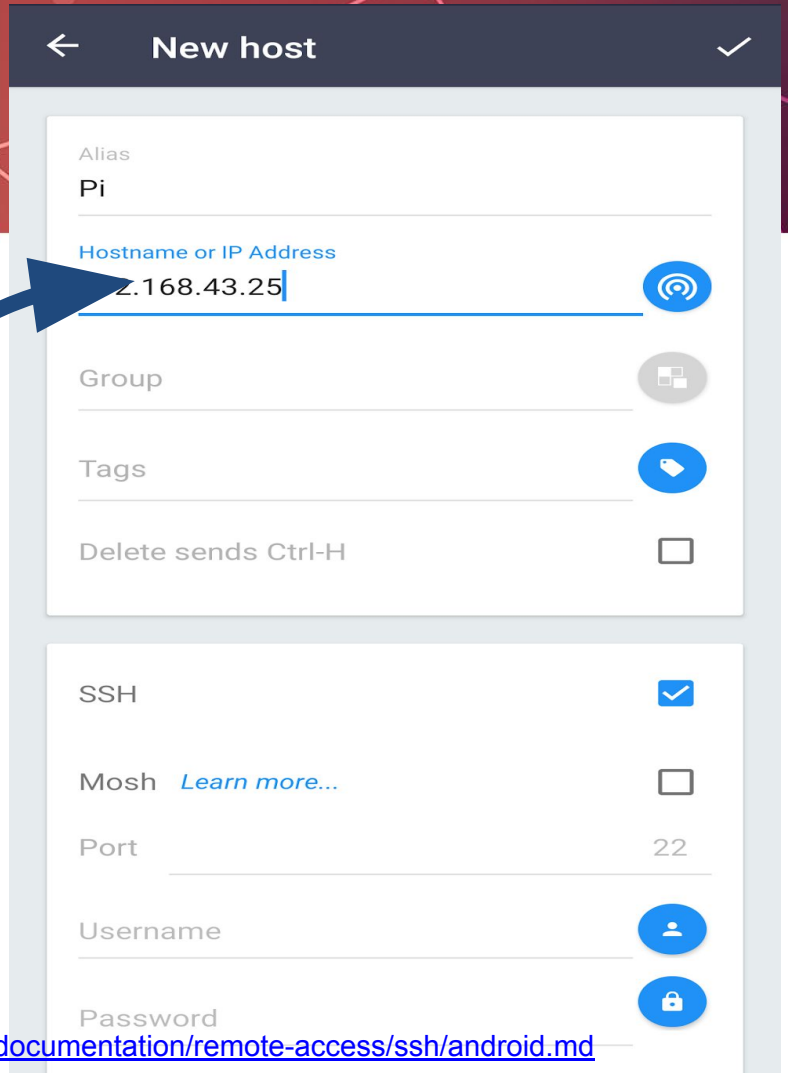
Step 1: Download Termius



Default Login
`Username`: pi
Password:
`raspberry`.

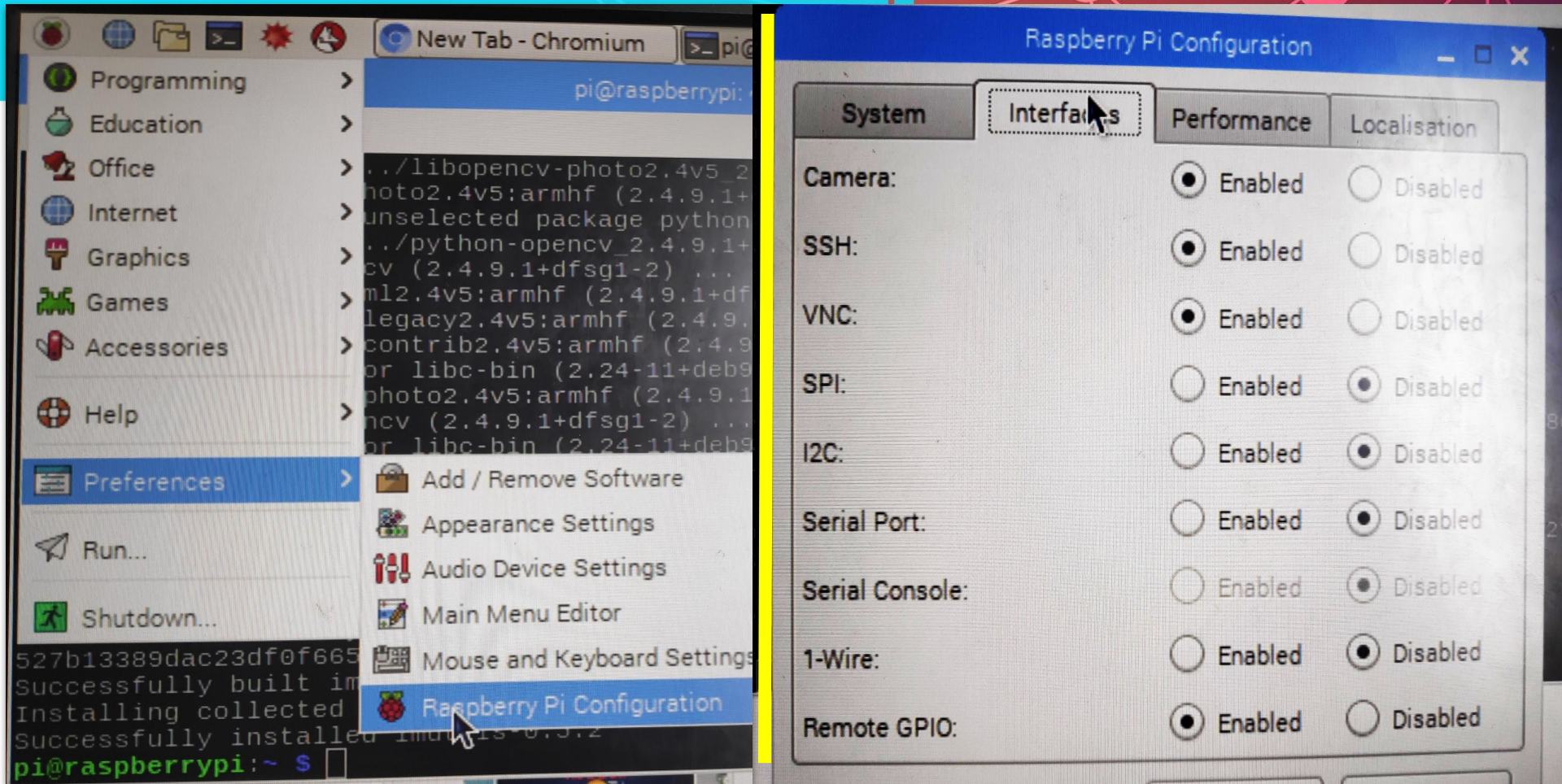
Step 2: Check IP Address of Raspberry Pi





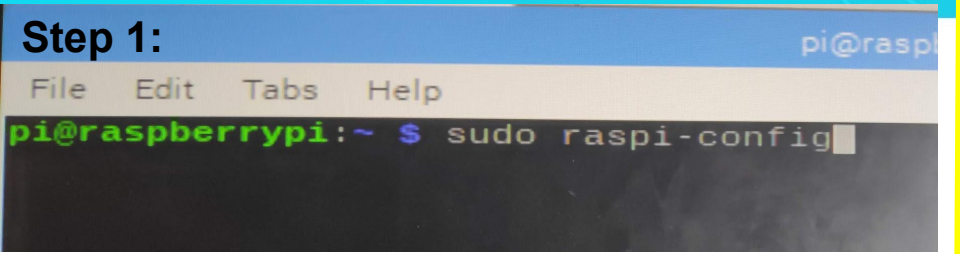
Reference: https://www.raspberrypi.org/documentation/remote-access/ssh/android.md

Enable SSH, VNC & Camera

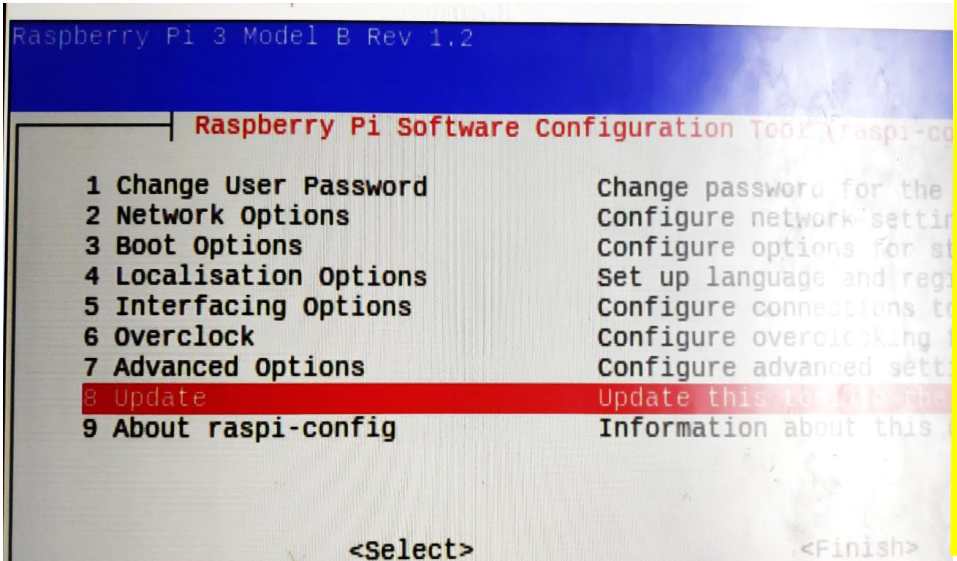


Installing Open-CV

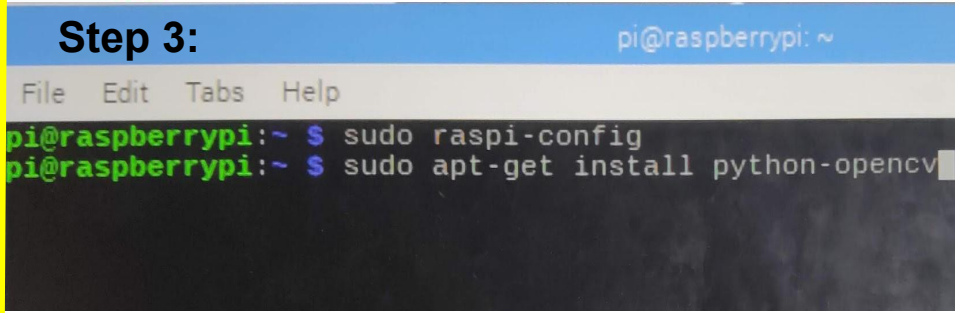
Step 1:



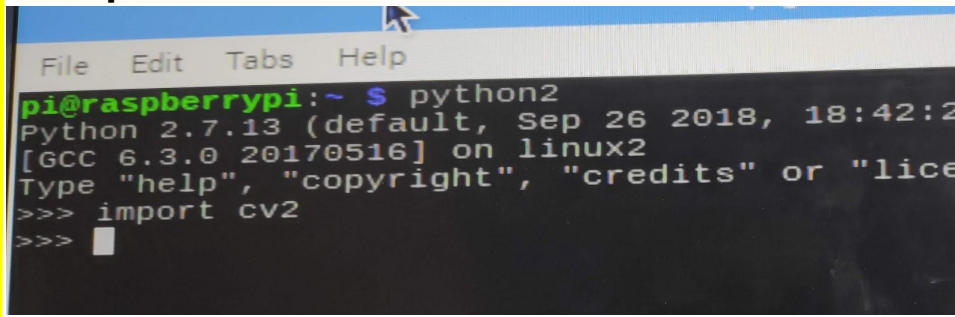
Step 2:



Step 3:



Step 4:

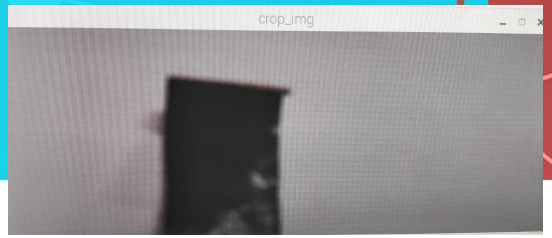


Reference:http://www.mindsensors.com/blog/how-to/how-to-install-opencv-on-raspberry-pi-and-do-face-tracking

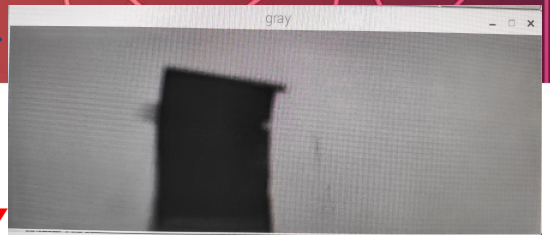
Line following Robot with OpenCV

Video Link

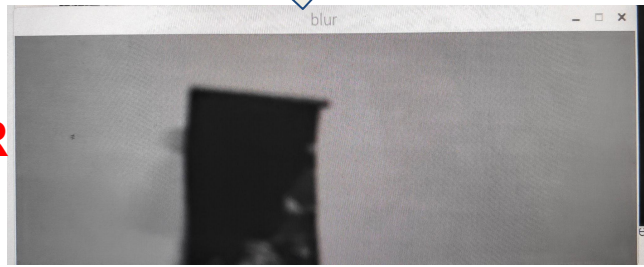
Actual



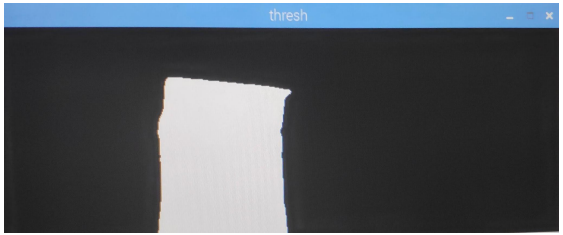
CROP



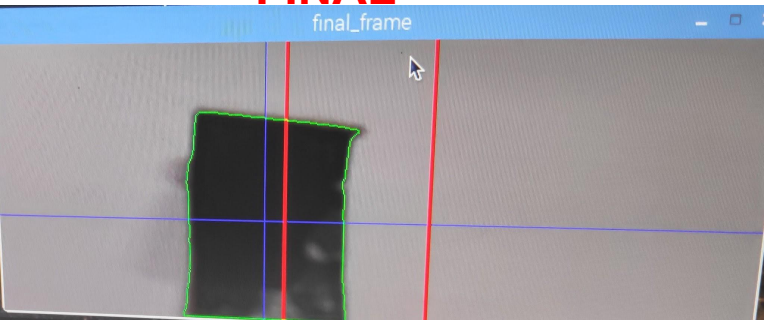
GRAY



BLUR



THRESH

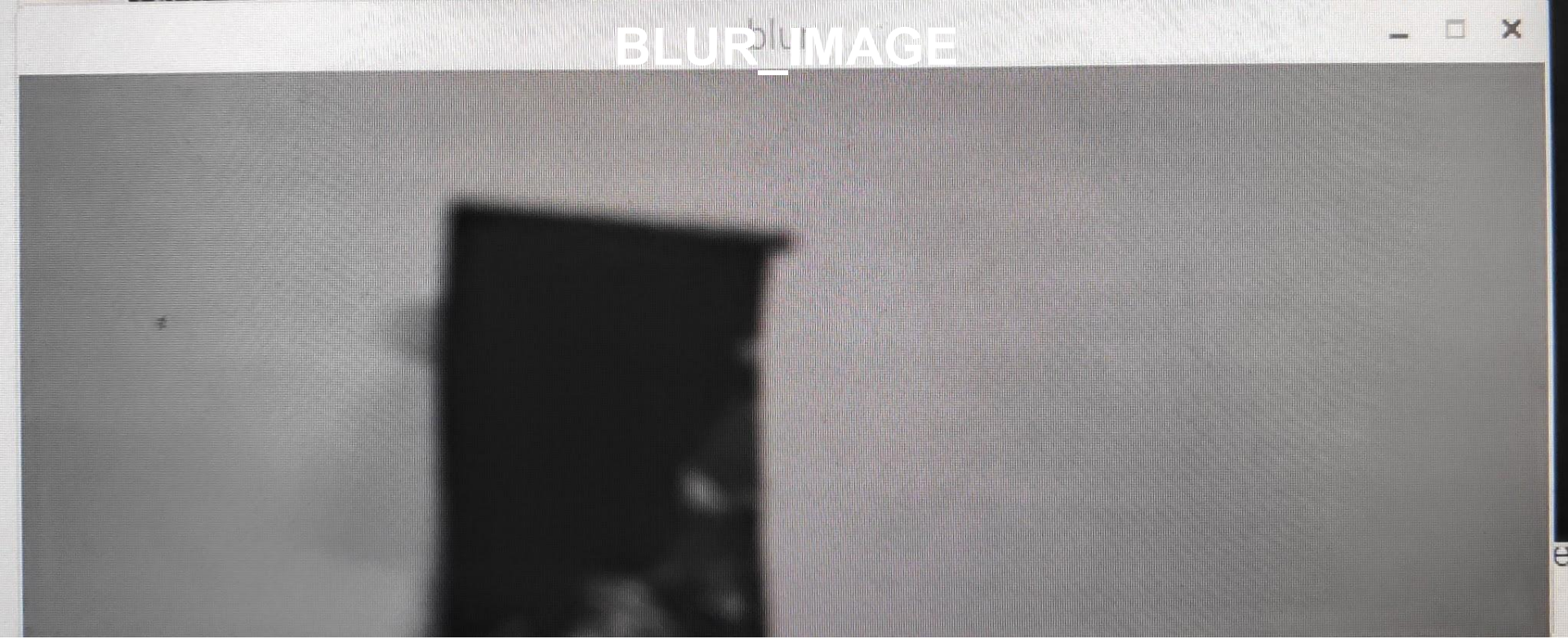


FINAL

ACTUAL_IMAGE

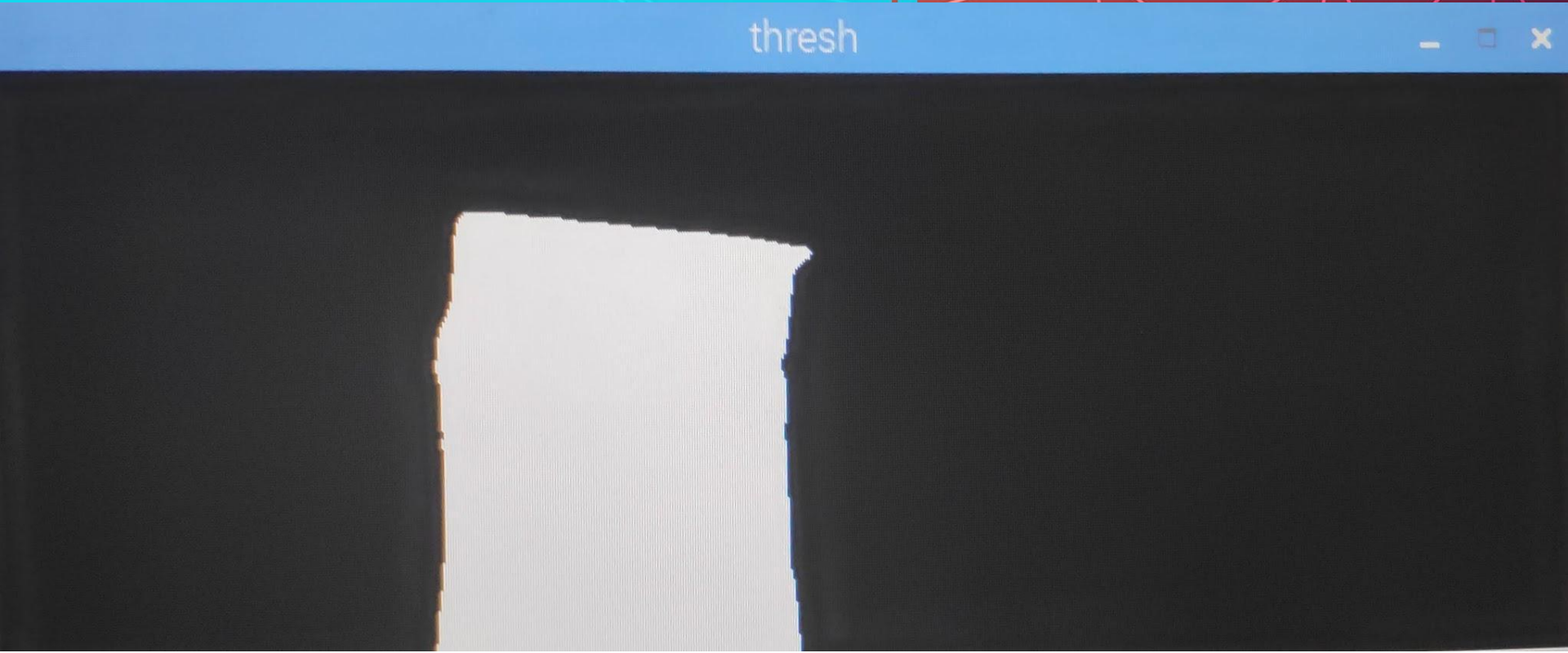
CROP_IMAGE

GRAY_IMAGE

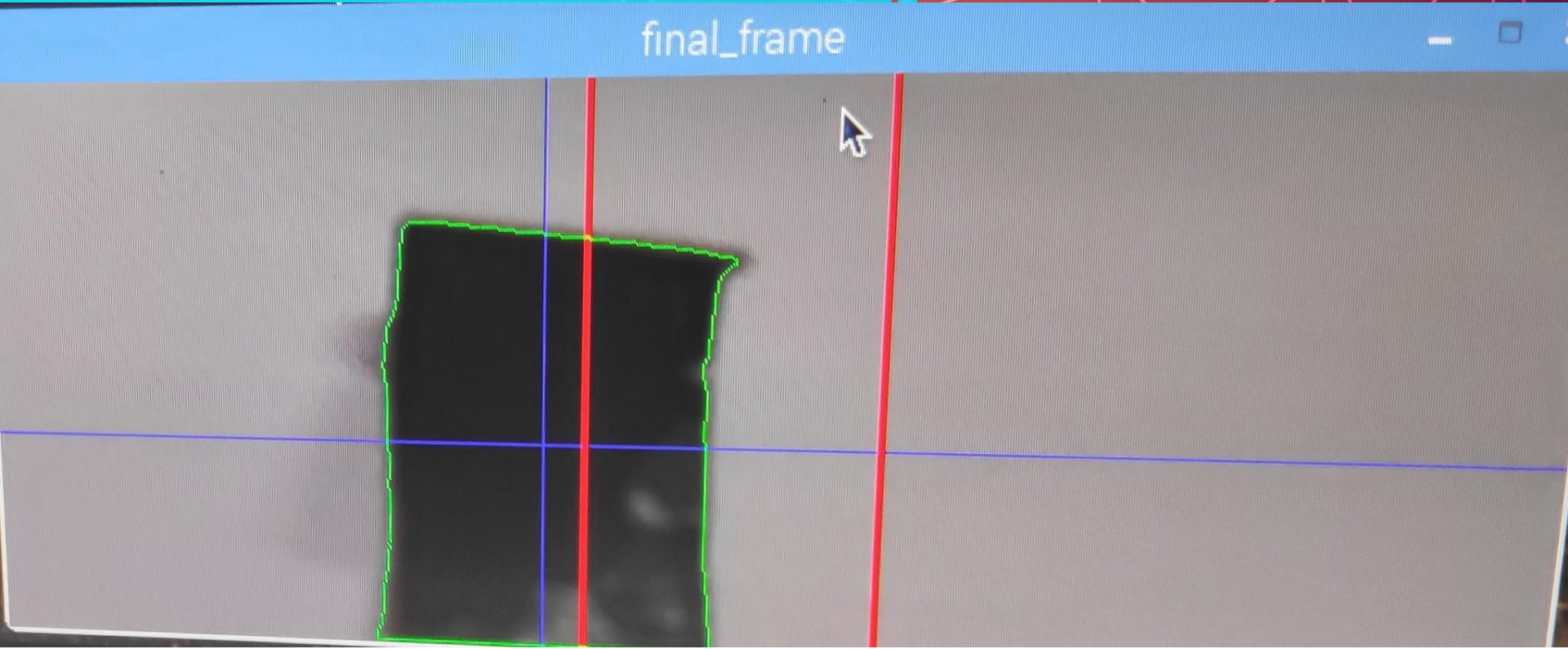


When **we blur** an **image**, **we** make the color transition from one side of an edge in the **image** to another smooth rather than sudden. The effect is to average out rapid changes in pixel intensity. The **blur**, or smoothing, of an **image** removes “outlier” pixels that may be noise in the **image**.

THRESH_IMAGE



FINAL_IMAGE



```
#========================== IMPORTS ========================#
import numpy as np
import cv2
import time
import imutils

from picamera import PiCamera          #PICAMERA
from picamera.array import PiRGBArray

import RPi.GPIO as GPIO                #GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

#========================== MOTOR PINS  AND FUNCTIONS (Only run on Raspi)=======================#
lm1= 31
lm2 = 33
rm1 = 35
rm2 = 37

setpin1 = [lm1,lm2,rm1,rm2]
for pin in setpin1:
  GPIO.setup(pin,GPIO.OUT)
```

```
24    def forward():
25        GPIO.output(lm1,1)
26        GPIO.output(lm2,0)
27        GPIO.output(rm1,1)
28        GPIO.output(rm2,0)
29
30    def left():
31        GPIO.output(lm1,0)
32        GPIO.output(lm2,1)
33        GPIO.output(rm1,1)
34        GPIO.output(rm2,0)
35
36    def right():
37        GPIO.output(lm1,1)
38        GPIO.output(lm2,0)
39        GPIO.output(rm1,0)
40        GPIO.output(rm2,1)
41
42    def sleft():
43        GPIO.output(lm1,0)
44        GPIO.output(lm2,0)
45        GPIO.output(rm1,1)
46        GPIO.output(rm2,0)
```

```
48    def sright():
49        GPIO.output(lm1,1)
50        GPIO.output(lm2,0)
51        GPIO.output(rm1,0)
52        GPIO.output(rm2,0)
53
54    def backward():
55        GPIO.output(lm1,0)
56        GPIO.output(lm2,1)
57        GPIO.output(rm1,0)
58        GPIO.output(rm2,1)
59
60    def Stop():
61        GPIO.output(lm1,0)
62        GPIO.output(lm2,0)
63        GPIO.output(rm1,0)
64        GPIO.output(rm2,0)
```

```
#========================== VARIABLE ====================
fx = 640
fy = 480
delta = 60
tl= fx/2 - delta
tr= fx/2 + delta

fright=0 #FLAG
fleft=0
```

```
'''
#******************** USING WEB CAM====================#
vc = cv2.VideoCapture(-1)
vc.set(3, fx)
vc.set(4, fy)

if vc.isOpened(): # try to get the first frame
    rval, frame = vc.read()
else:
    rval = False
while rval:
        rval, frame = vc.read()
        #*******************************#
        '''

#********************** USING PICAM ******************#
camera = PiCamera() # initialize the camera and grab a reference to the raw camera cap
camera.resolution = (fx, fy)
camera.framerate = 64
rawCapture = PiRGBArray(camera, size=(fx, fy))

time.sleep(0.1)# allow the camera to warmup

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
        frame = frame.array
        #*********************************************#
        #'''
```

```
103         #cv2.imshow('Actual_frame',frame)                                   #<<--
104
105         # Crop the image
106         crop_img = frame[fy/2:fy, 0:fx]
107         #cv2.imshow('crop_img',crop_img)                                    #<<--
108
109         # Convert to grayscale
110         gray = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)
111         #cv2.imshow('gray',gray)                                            #<<--
112
113         # Gaussian blur
114         blur = cv2.GaussianBlur(gray,(5,5),0)
115         #cv2.imshow('blur',blur)                                            #<<--
116
117         # Color thresholding
118         ret,thresh = cv2.threshold(blur,60,255,cv2.THRESH_BINARY_INV)
119         #cv2.imshow('thresh',thresh)                                        #<<--
120
121         # Find the contours of the frame
122         contours = cv2.findContours(thresh.copy(), 1, cv2.CHAIN_APPROX_NONE)
123         contours = imutils.grab_contours(contours)
```

```
125         if len(contours) > 0:
126                 fright=0
127                 fleft=0
128
129                 c = max(contours, key=cv2.contourArea) #CONTOUR WITH MAX AREA
130
131                 M = cv2.moments(c)
132
133
134
135                 cx = int(M['m10']/M['m00'])
136                 cy = int(M['m01']/M['m00'])
137
138
139                 # DISPLAYING CX AND CY
140                 cv2.line(crop_img,(cx,0),(cx,720),(255,0,0),1)
141                 cv2.line(crop_img,(0,cy),(1280,cy),(255,0,0),1)
142
143                 # DISPLAYING LEFT, RIGHT, FORWARD
144                 cv2.line(crop_img,(tl,0),(tl,fy/2),(0,0,255),2)
145                 cv2.line(crop_img,(tr,0),(tr,fy/2),(0,0,255),2)
146
147                 # DRAWING OUTLING AROUND CONTOUR
148                 cv2.drawContours(crop_img, contours, -1, (0,255,0), 1)
149
```

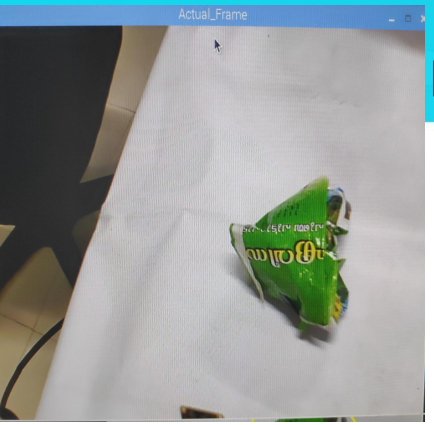
```
150
151             if cx >= tr:
152                 sright()
153                 fright =1
154                 print("Turn Right")
155             elif cx < tr and cx > tl:
156                 forward()
157                 print("On Track!")
158             elif cx <= tl:
159                 sleft()
160                 fleft =1
161                 print("Turn Left!")
162         else: #IF no countor
163             '''
164             if fright==1:
165                 sleft()
166             elif fleft==1:
167                 sright()
168             else:'''
169             Stop()
170             print("I don't see the line")
171
172
173         #==== Show Image ======#
174         cv2.imshow('final_frame',crop_img)                    #<<--
175
```

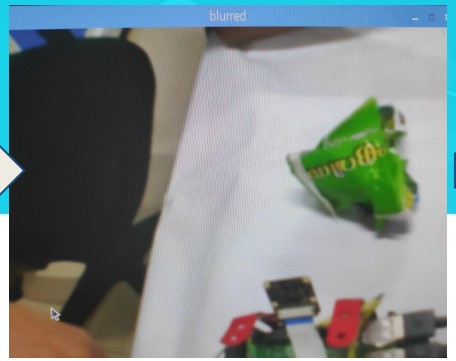
```
176
177         rawCapture.truncate(0)# picamera only # clear the stream in preparation for the next frame
178
179         #Breaking condition
180         key = cv2.waitKey(1) & 0xFF
181         if key == ord("q"):
182             Stop()
183             break
184
185
186 # close all windows
187 cv2.destroyAllWindows()
188 Stop()
189
190
```

Ball Tracking Robot using OpenCV

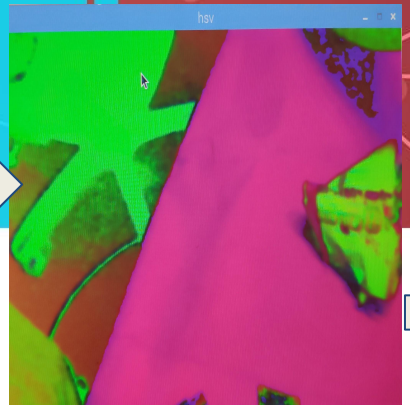
Video Link

Actual Image

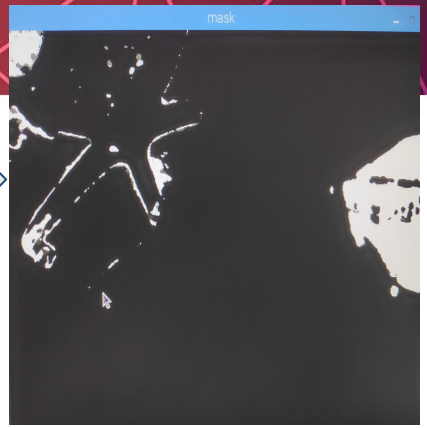




Blurred

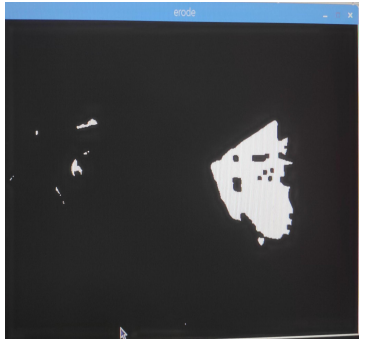


HSV



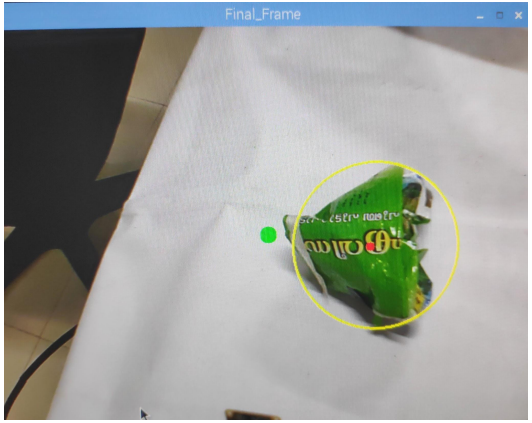
Mask

Erode



Dilate

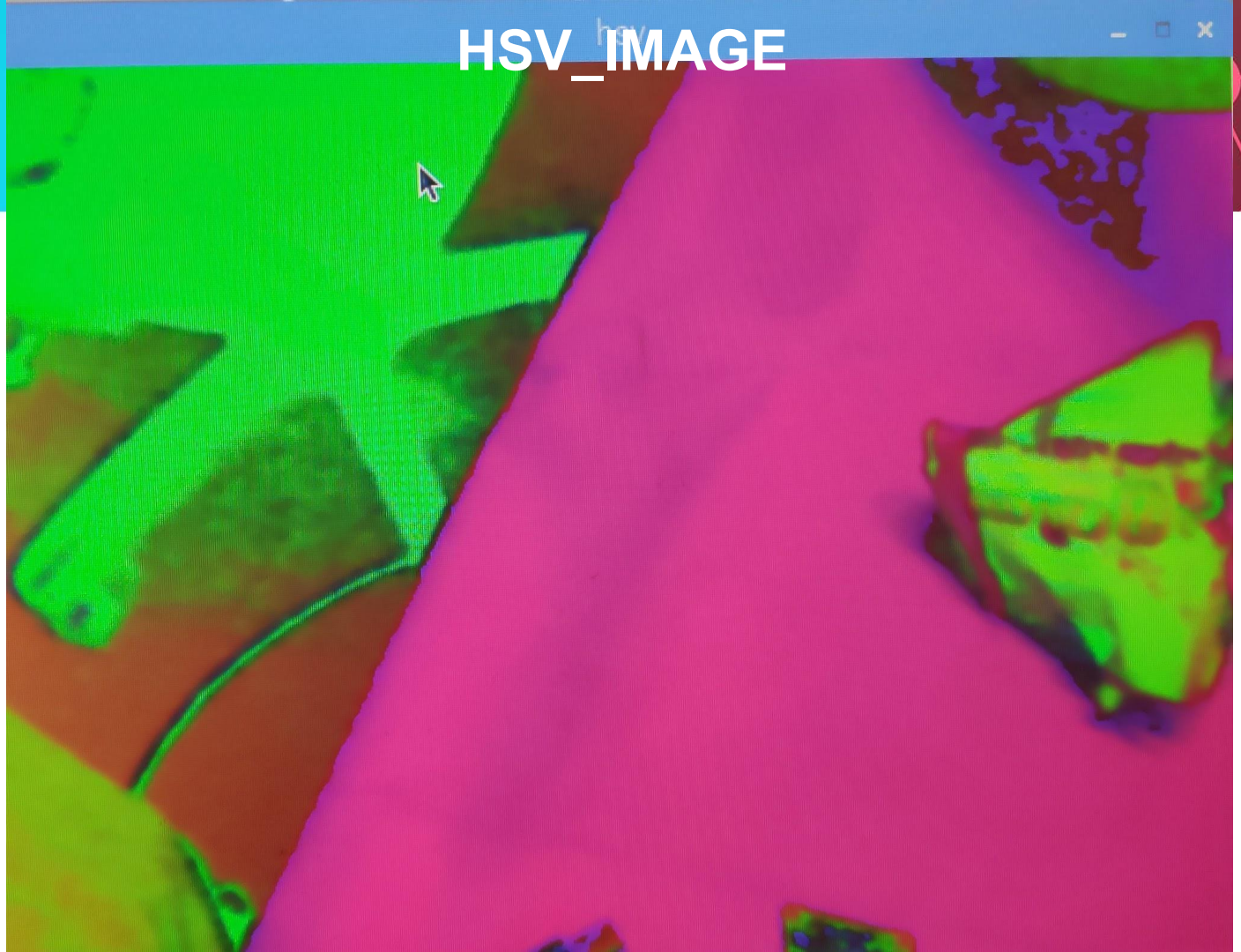
FINAL



ACTUAL_IMAGE

BLURRED_IMAGE

HSV_IMAGE

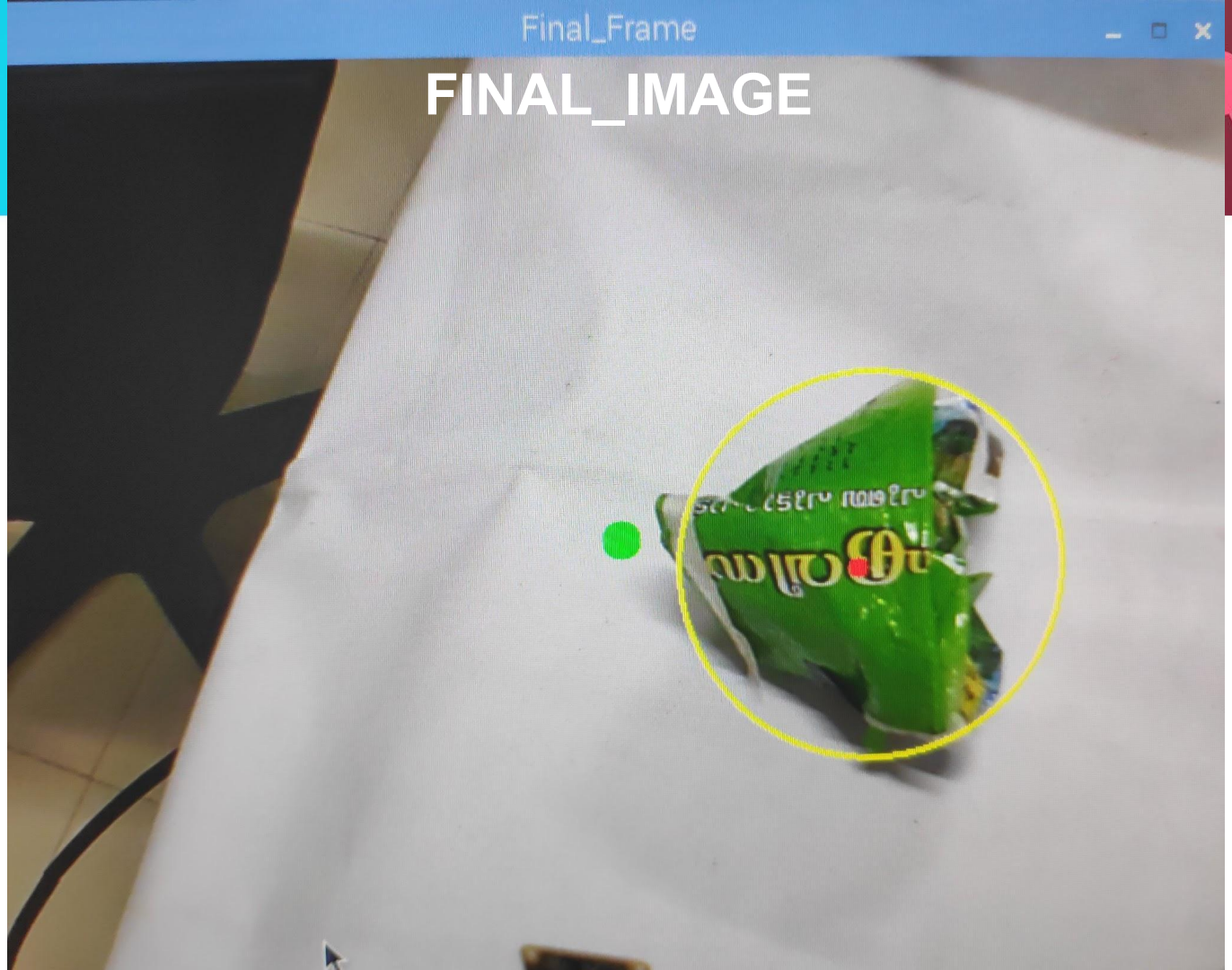


MASK_IMAGE

ERODE_IMAGE

DILATE_IMAGE

FINAL_IMAGE



```
#========================== IMPORTS ========================#
from collections import deque
import numpy as np
import cv2
import imutils
import time

from picamera import PiCamera          #PICAMERA
from picamera.array import PiRGBArray

import RPi.GPIO as GPIO #GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

#========================== MOTOR PINS  AND FUNCTIONS (Only run on Raspi)======================#
lm1= 31
lm2 = 33
rm1 = 35
rm2 = 37
b = 7

setpin1 = [b,lm1,lm2,rm1,rm2]
for pin in setpin1:
  GPIO.setup(pin,GPIO.OUT)

```

```
26  def forward():
27      GPIO.output(lm1,1)
28      GPIO.output(lm2,0)
29      GPIO.output(rm1,1)
30      GPIO.output(rm2,0)
31
32  def left():
33      GPIO.output(lm1,0)
34      GPIO.output(lm2,1)
35      GPIO.output(rm1,1)
36      GPIO.output(rm2,0)
37
38  def right():
39      GPIO.output(lm1,1)
40      GPIO.output(lm2,0)
41      GPIO.output(rm1,0)
42      GPIO.output(rm2,1)
43  def sleft():
44      GPIO.output(lm1,0)
45      GPIO.output(lm2,0)
46      GPIO.output(rm1,1)
47      GPIO.output(rm2,0)
48
```

```
48
49  def sright():
50      GPIO.output(lm1,1)
51      GPIO.output(lm2,0)
52      GPIO.output(rm1,0)
53      GPIO.output(rm2,0)
54
55  def backward():
56      GPIO.output(lm1,0)
57      GPIO.output(lm2,1)
58      GPIO.output(rm1,0)
59      GPIO.output(rm2,1)
60
61  def Stop():
62      GPIO.output(lm1,0)
63      GPIO.output(lm2,0)
64      GPIO.output(rm1,0)
65      GPIO.output(rm2,0)
66
67  def buzzon():
68      GPIO.output(b,1)
69
70  def buzzoff():
71      GPIO.output(b,0)
72  #----------------------------------------
```

```
74  #========================= VARIABLE ======================#
75  fx = 640
76  fy = 480
77  delta = 60
78  tl= fx/2 - delta
79  tr= fx/2 + delta
80
81  # define the lower and upper boundaries of the "green"
82  #ball in the HSV color space, then initialize the
83  #list of tracked points
84  greenLower = (29, 86, 6)
85  greenUpper = (64, 255, 255)
86  #---------------------------------------------------------#
87
88
```

```
89
90  '''
91  #******************* USING WEB CAM====================#
92  vc = cv2.VideoCapture(-1)
93  time.sleep(2.0)# allow the camera or video file to warm up
94  vc.set(3, fx)
95  vc.set(4, fy)
96
97  if vc.isOpened(): # try to get the first frame
98      rval, frame = vc.read()
99  else:
100     rval = False
101 while rval:
102     rval, frame = vc.read()
103     #******************************#
104     '''
105
106 #******************** USING PICAM ******************#
107 camera = PiCamera() # initialize the camera and grab a reference to the raw camera capture
108 camera.resolution = (fx, fy)
109 camera.framerate = 64
110 rawCapture = PiRGBArray(camera, size=(fx, fy))
111
112 time.sleep(0.1)# allow the camera to warmup
113
114 for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):# grab
115     frame = frame.array
116     #*********************************************#
117     #'''
```

talker.cpp | ballrobo_picam_simple.py | linerobo_picam.py | talker.py

```
117     #"""
118
119     #cv2.imshow("Actual_Frame", frame)      #<<--
120
121     blurred = cv2.GaussianBlur(frame, (11, 11), 0)
122     #cv2.imshow("blurred", blurred)      #<<--
123
124     hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
125     #cv2.imshow("hsv", hsv)      #<<--
126
127     # construct a mask for the color "green", then perform
128     # a series of dilations and erosions to remove any small
129     # blobs left in the mask
130     mask = cv2.inRange(hsv, greenLower, greenUpper)
131     #cv2.imshow("mask", mask)      #<<--
132     mask = cv2.erode(mask, None, iterations=2)
133     #cv2.imshow("erode", mask)      #<<--
134     mask = cv2.dilate(mask, None, iterations=2)
135     #cv2.imshow("dilate", mask)      #<<--
136
137
138     # find contours in the mask and initialize the current
139     # (x, y) center of the ball
140     cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
141     cnts = imutils.grab_contours(cnts)
142     center = None
```

```
143
144     xflag = 0
145     # only proceed if at least one contour was found
146     if len(cnts) > 0:
147
148         # find the largest contour in the mask, then use
149         # it to compute the minimum enclosing circle and
150         # centroid
151         c = max(cnts, key=cv2.contourArea)
152         ((x, y), radius) = cv2.minEnclosingCircle(c)
153
154         M = cv2.moments(c)
155         center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
156
157         # only proceed if the radius meets the constrains
158         if (radius > 30) and (radius < 100):
159             xflag = 1
160             # draw the circle and centroid on the frame,
161             # then update the list of tracked points
162             cv2.circle(frame, (int(x), int(y)), int(radius),(0, 255, 255), 2)
163             cv2.circle(frame, center, 5, (0, 0, 255), -1)
164
165     fcx = np.shape(frame)[0]/2
166     fcy = np.shape(frame)[1]/2
167     cv2.circle(frame,(fcy,fcx),10,(0,255,0),-1)
```

```
    if xflag == 1:
        delta = radius
        if x < fcy - delta:
            print("left")
            sleft()
        elif x  > fcy + delta:
            print("right")
            sright()
        else:
            print("stop")
            Stop()
    else:
        print("stop")
        Stop()

    # show the frame to our screen
    cv2.imshow("Final_Frame", frame) #<<---
    key = cv2.waitKey(1) & 0xFF

    rawCapture.truncate(0)# picamera only # clear the stream in preparation for the next frame

    # if the 'q' key is pressed, stop the loop
    if key == ord("q"):
        break

# close all windows
cv2.destroyAllWindows()
Stop()
```

Thank You!