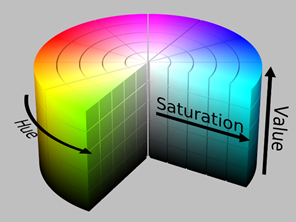
**HSV= hue, saturation,value**

* Este o reprezentare alternativă a standardului RGB



**Detecția de forme plus detectie de culoare- real time**

Am făcut o detecție de forma și o detecție de culoare . Vede doar lucrurile de culoare albastra si detecteaza ce forma au. Este un algoritm reușit , am reușit sa scoatem și zgomotele , doar dacă scenariul este : sa vadă ca obiectele doar lucrurile albastre

**# import pentru pachetele necesare**

from picamera.array import PiRGBArray

from picamera import PiCamera

import time

import cv2

import numpy as np

**#alegem un font de scriere**

font = cv2.FONT\_HERSHEY\_COMPLEX\_SMALL

**# initializam camera, schimbam rezolutia si capturam o referinta a camerei**

camera = PiCamera()

camera.resolution = (640, 480)

camera.framerate = 32

rawCapture = PiRGBArray(camera, size=(640, 480))

**# lăsăm 0.1 secunde ca sa isi faca camera inițializarea**

time.sleep(0.1)

def nothing(x):

pass

**# capturam încontinuu cadre de la camere**

for frame in camera.capture\_continuous(rawCapture,format="bgr",use\_video\_port=True):

**#imaginea primește cadrele capturate**

image = frame.array

**#creem trackbaruri pentru matricile de culori pentru a găsi nuanța cea mai potrivită de albastru**

cv2.namedWindow("Trackbars")

cv2.createTrackbar("L-H","Trackbars",87,180,nothing)

cv2.createTrackbar("L-S","Trackbars",115,255,nothing)

cv2.createTrackbar("L-V","Trackbars",0,255,nothing)

cv2.createTrackbar("U-H","Trackbars",180,180,nothing)

cv2.createTrackbar("U-S","Trackbars",255,255,nothing)

cv2.createTrackbar("U-V","Trackbars",255,255,nothing)

**#punem valorile finale în matricii**

l\_h=cv2.getTrackbarPos("L-H","Trackbars")

l\_s=cv2.getTrackbarPos("L-S","Trackbars")

l\_v=cv2.getTrackbarPos("L-V","Trackbars")

u\_h=cv2.getTrackbarPos("U-H","Trackbars")

u\_s=cv2.getTrackbarPos("U-S","Trackbars")

u\_v=cv2.getTrackbarPos("U-V","Trackbars")

lower\_blue=np.array([l\_h,l\_s,l\_v])

upper\_blue=np.array([u\_h,u\_s,u\_v])

**#transformăm în hsv și apoi folosim o masca pentru detecția de culoare**

hsv=cv2.cvtColor(image,cv2.COLOR\_BGR2HSV)

mask=cv2.inRange(hsv,lower\_blue,upper\_blue)

**#pentru a elimina din zgomotele imaginii**

kernel=np.ones((5,5),np.uint8)

mask=cv2.erode(mask,kernel)

**#căutăm contururile**

\_, contours, \_ = cv2.findContours(mask, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

for cnt in contours:

**#pentru elimina zgomotele ca sa nu le ia ca forme**

area=cv2.contourArea(cnt)

approx = cv2.approxPolyDP(cnt, 0.02\*cv2.arcLength(cnt, True), True)

**#calculeaza coordonatele x și y**

x=approx.ravel()[0]

y=approx.ravel()[1]

**# tot ce este mai mic decat 400 sunt zgomote**

if area>400:

**#trasează contururile**

cv2.drawContours(image,[approx],0,(255),5)

**# vedem cate laturi are conturul și știm atunci ce forma este**

if len(approx) == 3:

cv2.putText(image, "Triangle", (x, y), font, 1, (255))

elif len(approx) == 4:

cv2.putText(image, "Rectangle", (x, y), font, 1, (255))

elif len(approx) == 5:

cv2.putText(image, "Pentagon", (x, y), font, 1, (255))

else:

cv2.putText(image, "Circle", (x, y), font, 1, (255))

**#afisam imagine și mask**

cv2.imshow("Frame", image)

cv2.imshow("mask", mask)

key = cv2.waitKey(1) & 0xFF

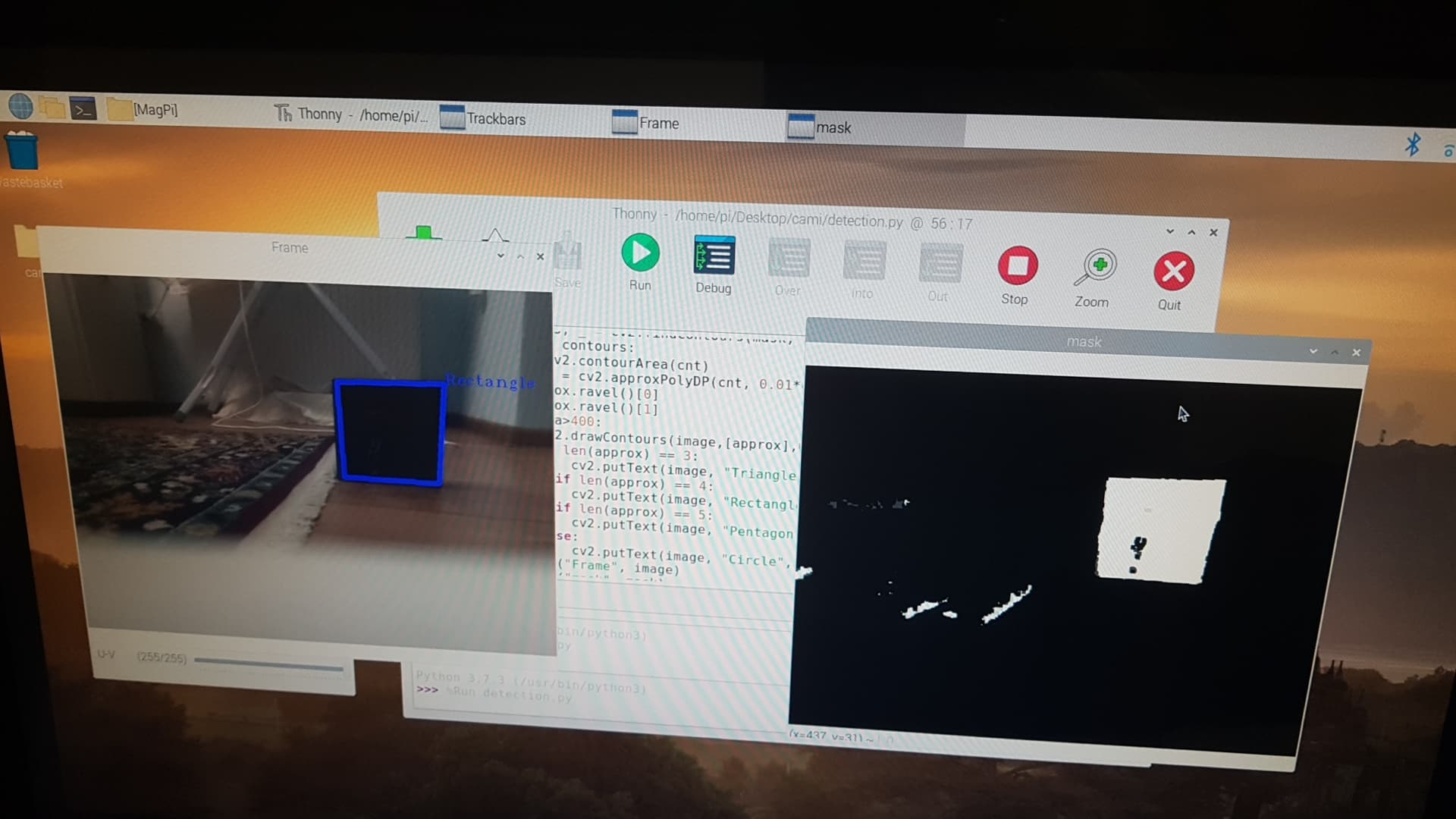
**# ștergem ce avem pentru pregătirea următorului cadrul**

rawCapture.truncate(0)

**#dacă apasam tasta q ieșim din for**

if key == ord("q"):

break



**Detecția de culoare - pe poza**

**Algoritmul este pentru a detectata culoarea roșie , dacă nu e roșu vedem o imagine neagră**

import cv2

import numpy as np

**#citim imaginea**

image=cv2.imread('red.png')

**#afisam imaginea**

cv2.imshow("image",image)

**#transformăm în hsv**

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

**#matricile de culori(parte inferioara si superioasa a culorii )**

colorLow = np.array([0,120,70])

colorHigh = np.array([10,255,255])

kernalOpen=np.ones((5,5))

kernalClose=np.ones((5,5))

**#măști pentru a detecta culoarea roșie**

mask = cv2.inRange(hsv, colorLow, colorHigh)

mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE,kernalClose)

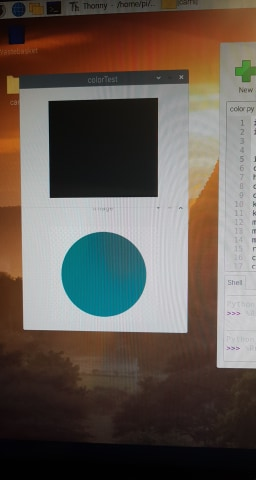
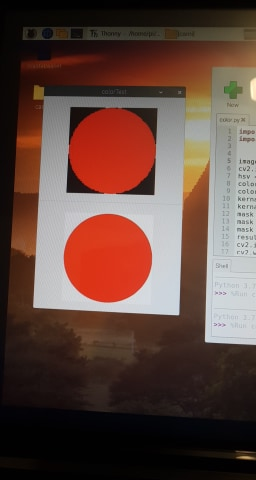
mask = cv2.morphologyEx(mask, cv2.MORPH\_OPEN, kernalOpen)

**#pentru afișa culoarea roșie dacă este detectata**

result = cv2.bitwise\_and(image, image, mask = mask)

cv2.imshow('colorTest', result)

cv2.waitKey(0)



**Detecția de culoare - real time**

**Algoritmul este pentru a detectata doar culoarea roșie , merge perfect . Acest algoritm se va folosi ca sa se opreasca robotelul cand vede culoarea roșie.**

from picamera.array import PiRGBArray

from picamera import PiCamera

import time

import cv2

import numpy as np

**# initializam camera, schimbam rezolutia si capturam o referinta a camerei**

camera = PiCamera()

camera.resolution = (640, 480)

camera.framerate = 32

rawCapture = PiRGBArray(camera, size=(640, 480))

**# lăsăm 0.1 secunde ca sa isi faca camera inițializarea**

time.sleep(0.1)

**# capturam încontinuu cadre de la camere**

for frame in camera.capture\_continuous(rawCapture,format="bgr",use\_video\_port=True):

**#imaginea primește cadrele capturate**

image = frame.array

**#transformăm în hsv**

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

**#matricile de culori(parte inferioara si superioasa a culorii**

colorLow=np.array([131,94,0])

colorHigh=np.array([180,255,255])

kernalOpen=np.ones((5,5))

kernalClose=np.ones((5,5))

**#măști pentru a detecta culoarea roșie**

mask = cv2.inRange(hsv, colorLow, colorHigh)

mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE,kernalClose)

mask = cv2.morphologyEx(mask, cv2.MORPH\_OPEN, kernalOpen)

**#pentru afișa culoarea roșie dacă este detectata**

result = cv2.bitwise\_and(image, image, mask = mask)

**#pentru a afișa imaginea originală și cea in care folosim masca pentru detectia culorii**

cv2.imshow("image",image)

cv2.imshow('colorTest', result)

key = cv2.waitKey(1) & 0xFF

**# ștergem ce avem pentru pregătirea următorului cadrul**

rawCapture.truncate(0)

**#dacă apasam tasta q ieșim din for**

if key == ord("q"):

break