

## PROIECT SINCRETIC II

### Proiect mecatronic *ROBOT MOBIL DE OCOLIRE*

Studenti:

BURLACU CAMELIA-ADELINA, GR. 1.2

MESTER ARANKA-NIKOLETTA, GR. 3.2

CAPĂȚĂ CRISTINA, GR. 1,2

CIUCURIȚĂ SAMUEL, GR. 4.1

COORDONATOR: Conf.dr.ing.Adrian KORODI

# CUPRINS

1. INTRODUCERE .....	2
1.1 Lucrări/documentări .....	2
1.1.1. Raspberry pi.....	2
1.1.2. Exemplu robot de urmărire.....	4
1.1.3. Rețele neuronale.....	5
1.1.4. OpenCV.....	8
1.1.5. YOLO.....	11
1.2 Tema proiectului .....	14
1.3 Descriere capitole .....	14
2. ARHITECTURA GENERALĂ A SISTEMULUI.....	15
2.1 Arhitectură generală.....	19
2.2 Regimuri de funcționare.....	19
3. DEZVOLTĂRI MECATRONICE .....	23
3.1 Coduri pentru testarea modulelor.....	23
3.2 Captura de imagine cu OpenCV.....	27
3.3 Real time cu OpenCV.....	28
3.4 Procesare de imagine.....	29
3.5 Detecția de forme- prima dată pe o imagine cu forme.....	33
4. BIBLIOGRAFIE .....	36

# 1. INTRODUCERE

## 1.1. LUCRĂRI / DOCUMENTĂRI

### 1.1.1. RASBERRY PI

Raspberry Pi pentru procesarea de imagine în domeniul educational:

Documentația tratată dorește în esență să răspundă următoarelor două întrebări:

1. Care este cea mai bună metodă pentru a învăța conceptele fundamentale în materie de micro-computere?
2. Care este cea mai bună aplicație pentru a stârni interesul în procesarea de imagine?

#### ▪ **Partea hardware:**

**Raspberry Pi**( = micro-computer) este la un cost mult redus față de un coputer clasic ( de 10 ori mai puțin), are caracteristici bune luând în considerare dimensiunile sale reduce.

#### **Caracteristici hardware :**

Modelul B : 512 MB (RAM) memorie internă, 700 MHz frecvență procesor.

Modelul 2 are 1 GB (RAM) și conține un procesor cu patru nuclee fiecare cu o frecvență de 900 MHz. Pe lângă acestea, dispozitivele mai pot avea două sau mai multe porturi USB, un port HDMI, un port pentru cipul de memorie Samsung, un port GPIO ș.a.. Camera cu care se face procesarea de imagine are un port prevăzut în această platformă. De menționat faptul că un cost estimativ total al acestui dispozitiv și perifericele necesare s-ar ridica undeva la 180 \$(USD).

#### ▪ **Partea software:**

##### 1) *Sistemul de operare.*

Prima sarcină care le revine studenților e să instaleze sistemul de operare pe cardul micro-SD al dispozitivului cu ajutorul unui PC și a unui soft precum Win32Disk Manager. Ei au nevoie de un fișier imagine pentru a realiza acest lucru. Scopul didactic este să-i determine pe studenți să reușească să-și dea seama că sistemul de operare al RPi-ului este plasat pe cardul micro-SD, nu în memoria internă a dispozitivului nici pe cipul de memorie Samsung conectat la dispozitiv.

##### 2) *Programarea dispozitivului.*

Se dorește utilizarea de către studenți a unor limbaje, medii de programare disponibile publicului larg ce nu implică nici un cost. Pentru a implementa algoritmi de procesare a imaginii studenții pot folosi limbaje precum Octave sau Python. Ei trebuie să aibe cunoștințe minime despre programare în limbaje de asamblare dar și în limbaje de nivel înalt cum ar fi C-ul. Pașii care trebuie urmați pentru a realiza un algoritm de procesare de imagine în Python sunt: importarea modulelor de

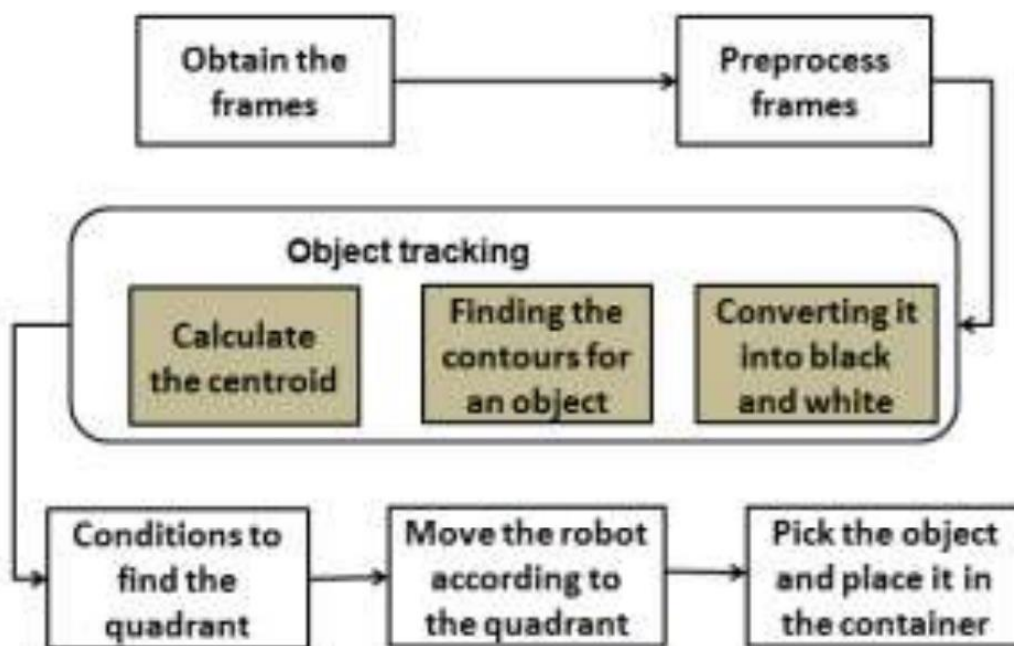
SimpleCV prevăzute, definirea mărimii imaginii pe care o poate da camera, definirea ferestrei de afișare, definirea matricei de imagine cu care se va lucra apoi realizarea binarizării, dilatării și eroziunii.

**CONCLUZII :** Această lucrare descrie o manieră nouă de a învăța noile tehnologii în micro-computere și procesare de semnal digital. Este centrată pe folosirea Rpi-ului și reprezintă un proiect ca o alternativă la cursurile orientate spre teorie.

### 1.1.2. EXEMPLU ROBOT DE URMĂRIRE

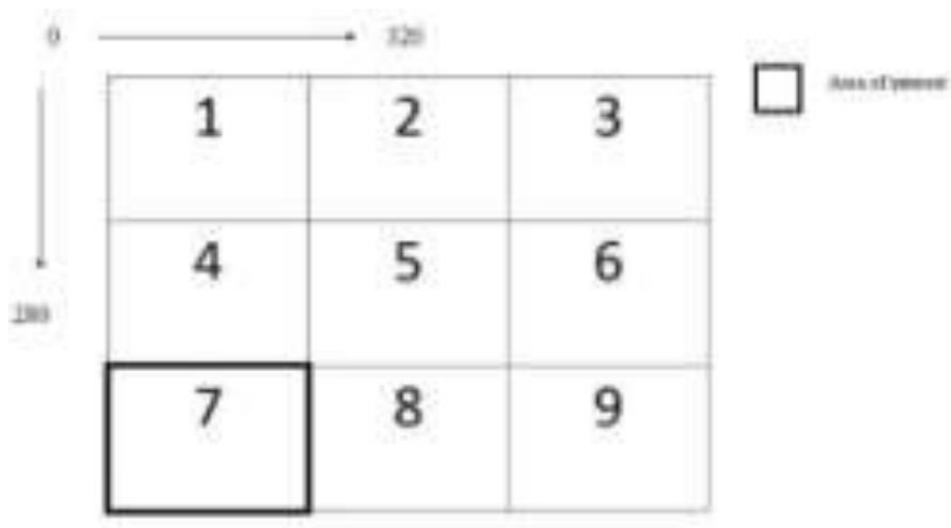
Robot care urmărește un obiect folosind Raspberry Pi și OpenCV:

- *OpenCV*  
Open Source Computer Vision este o bibliotecă cu funcții create special pentru procesarea în timp real a imaginilor.
- *Arhitectura sistemului*



Sistemul înregistrează în timp real un videoclip prin intermediul paginii web și le introduce în etapa de preprocesare. În această etapă, se reduce zgomotul imaginii și se transformă totul în grey-scale. Apoi, convertește cadrele obținute în cadre alb-negru și identifică obiectul folosindu-se de marginile acestuia.

Pentru a permite urmărirea, se calculează centru de masă al obiectului de urmărit, iar cadrele se împart virtual în chenare. Astfel, robotul detectează schimbarea poziției obiectului bazându-se pe centrul acestuia și chenarul în care se află.



În imaginea de mai sus este prezentat modul în care sunt împărțite cadrele în chenare virtuale.

Pentru a determina conturul se folosește următoarea linie de cod:

```
contours1,_=cv2.findContours(im_bw,cv2.RETR_TREE,cv2.CHAIN_APPROX_
NONE)
```

iar centrul obiectului este determinat folosind formula:

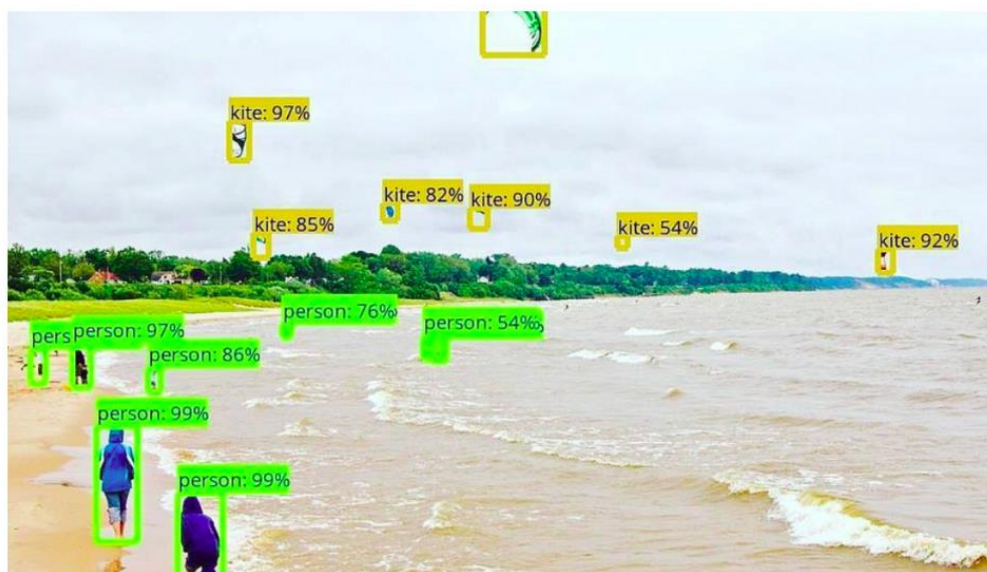
```
M=cv2.moments(im_bw)
centroid_x = int(M['m10']/M['m00'])
centroid_y = int(M['m01']/M['m00'])
```

### 1.1.3. REȚELE NEURONALE

O introducere pas cu pas în algoritmi de bază pentru detectia obiectelor(cu rețele neuronale):

Cât timp ai petrecut căutând cheile pierdute într-o de cameră dezordonată? Dar dacă un algoritm simplu ar putea localiza cheile dvs. în câteva milisecunde? Aceasta este puterea algoritmilor de detectare a obiectelor. Deși acesta a fost un exemplu simplu, aplicațiile de detectare a obiectelor acoperă mai multe și diverse industrii, de la supravegherea continuă până la detectarea vehiculelor în timp real în orașele inteligente.

Când ni se arată o imagine, creierul nostru recunoaște instantaneu obiectele conținute de ea. Pe de altă parte, pentru o mașină necesită mult timp și date de instruire pentru a identifica aceste obiecte. Dar, cu progresele recente în materie de hardware și deep learning, acesta parte a devenit mult mai ușoară și mai intuitivă.



## Detecția obiectelor

Să presupunem că mașina dvs. surprinde o imagine ca cea de mai jos.

Imaginea înfățișează în esență că mașina noastră care este aproape de o mână de oameni traversează drumul din fața mașinii noastre. Sistemul de detectare a pietonilor ar trebui să identifice exact locul unde merg oamenii. Ca să facă acest lucru creează o caseta de delimitare în jurul acestor oameni, astfel încât sistemul să poată identifica unde se află oamenii în imagine.



Obiectivul nostru din spatele activității de detectare a obiectelor este :

- De a identifica ce obiecte sunt prezente în imagine și unde sunt amplasate
- De a alege doar obiecte care ne interesează

### Abordare 1: mod naiv (împărțim și cucerim)

Cea mai simplă abordare pe care o putem adopta este să împărțim imaginea în patru părți:

- Colțul din stânga sus

- Colțul din dreapta sus
- Colțul din stânga jos
- Colțul din dreapta jos

Acum, următorul pas este să puneti fiecare dintre aceste părți într-un clasificator de imagini (în rețeaua neuronală se construiește așa ceva). Acest lucru ne va da un rezultat dacă acea parte a imaginii are un pieton sau nu. Trebuie să identifice întregul obiect (sau o persoană în acest caz) deoarece numai localizarea părților unui obiect ar putea duce la rezultate catastrofale.



### Metoda 2: Efectuarea diviziilor structurate

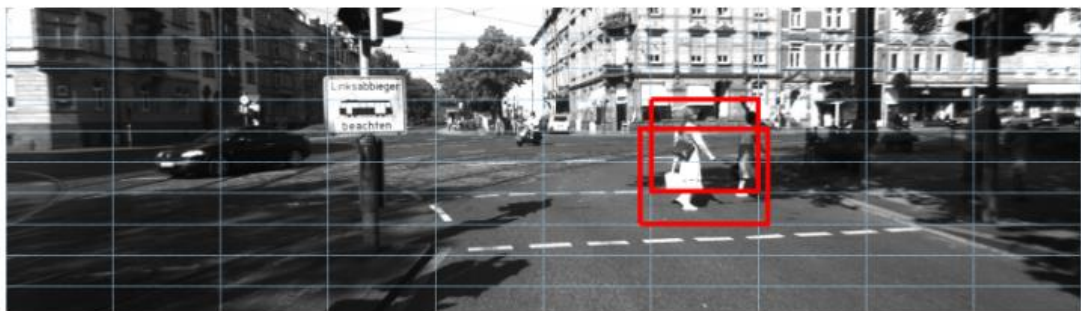
Pentru a ne construi sistemul de detectare a obiectelor într-un mod mai structurat, putem urma pașii de mai jos:

Pasul 1: Împarte imaginea într-o grilă de  $10 \times 10$  astfel:

Pasul 2: Definiți centru pentru fiecare bucată

Pasul 3: Pentru fiecare centru, luați trei bucati diferite de înălțimi diferite

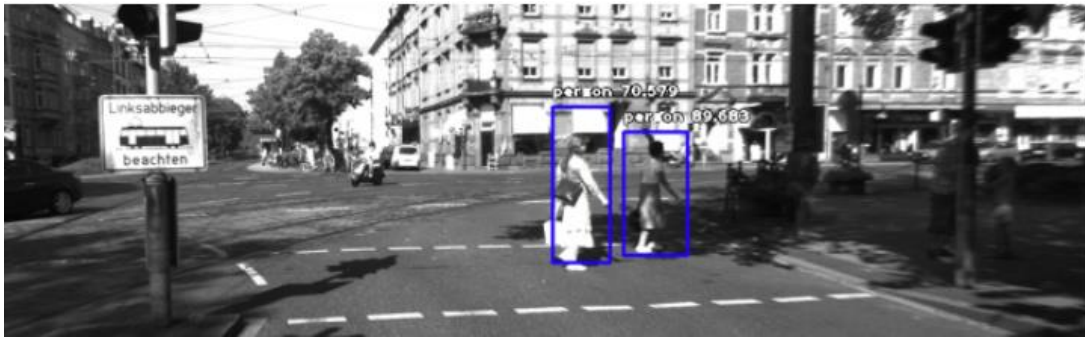
Pasul 4: Treceți toate bucatile create prin clasificatorul de imagini pentru a obține predicții.



### Metoda 3: Utilizarea deep learning

Deep learning are atât de mult potențial în spațiul de detectare a obiectelor. În loc să luăm bucatile din imaginea originală, putem trece imaginea originală printr-o rețea neuronală pentru a reduce dimensiunile. De asemenea, am putea folosi o rețea neurală pentru a sugera bucatile selective. Putem consolida un algoritm de deep learning pentru a oferi predicții cât mai aproape de caseta de delimitare inițială. Acum, în loc să antrenăm diferite rețele neuronale pentru rezolvarea fiecărei probleme individuale, putem lua un singur model de rețea neuronală profundă, care va încerca să rezolve

singur toate problemele.



#### 1.1.4. OPENCV

##### Detectia de forme

Primul pas în construirea detectorului nostru de forme este să scriem un cod care să încapsuleze logica de identificare a formei. Apoi vom defini clasa **ShapeDetector**, care are o metoda **detect**. În acest fel vom aproxima conturul(**cv2.approxPolyDP**). După cum sugerează și numele, aproximarea conturului este un algoritm pentru reducerea numărului de puncte dintr-o curbă cu un set redus de puncte.

```
1.  # import the necessary packages
2.  import cv2
3.
4.  class ShapeDetector:
5.      def __init__(self):
6.          pass
7.
8.      def detect(self, c):
9.          # initialize the shape name and approximate the contour
10.         shape = "unidentified"
11.         peri = cv2.arcLength(c, True)
12.         approx = cv2.approxPolyDP(c, 0.04 * peri, True)

```

OpenCV shape detection

```
14.         # if the shape is a triangle, it will have 3 vertices
15.         if len(approx) == 3:
16.             shape = "triangle"
17.
18.         # if the shape has 4 vertices, it is either a square or
19.         # a rectangle
20.         elif len(approx) == 4:
21.             # compute the bounding box of the contour and use the
22.             # bounding box to compute the aspect ratio
23.             (x, y, w, h) = cv2.boundingRect(approx)
24.             ar = w / float(h)
25.
26.             # a square will have an aspect ratio that is approximately
27.             # equal to one, otherwise, the shape is a rectangle
28.             shape = "square" if ar >= 0.95 and ar <= 1.05 else "rectangle"
29.
30.         # if the shape is a pentagon, it will have 5 vertices
31.         elif len(approx) == 5:
32.             shape = "pentagon"
33.
34.         # otherwise, we assume the shape is a circle
35.         else:
36.             shape = "circle"
37.
38.         # return the name of the shape
39.         return shape

```

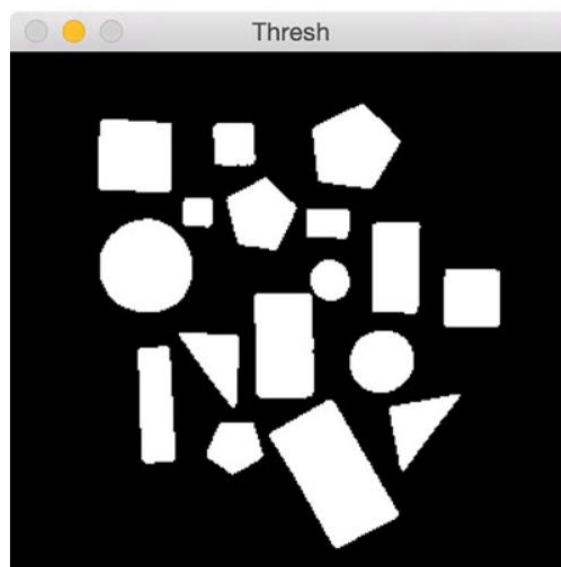


Este important să înțelegem că un contur este format dintr-o listă de vârfuri. Putem verifica numărul de intrări din această listă pentru a determina forma unui obiect. De exemplu, dacă conturul aproximativ are trei vârfuri, atunci acesta trebuie să fie un triunghi sau dacă un contur are patru vârfuri, îl putem eticheta drept pătrat.

#### OpenCV shape detection

```
13. # load the image and resize it to a smaller factor so that
14. # the shapes can be approximated better
15. image = cv2.imread(args["image"])
16. resized = imutils.resize(image, width=300)
17. ratio = image.shape[0] / float(resized.shape[0])
18.
19. # convert the resized image to grayscale, blur it slightly,
20. # and threshold it
21. gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
22. blurred = cv2.GaussianBlur(gray, (5, 5), 0)
23. thresh = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY)[1]
24.
25. # find contours in the thresholded image and initialize the
26. # shape detector
27. cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
28.                        cv2.CHAIN_APPROX_SIMPLE)
29. cnts = imutils.grab_contours(cnts)
30. sd = ShapeDetector()
```

Al doilea pas, ne încărcăm imaginea și o redimensionăm. Apoi facem conversie **grayscale** (adică facem imaginea alb-negru), o blurăm pentru a scăpa de impurități și aplicăm un **thresholding**. (înlocuiesc fiecare pixel dintr-o imagine cu un pixel negru dacă intensitatea imaginii este mai mică decât o constantă fixă T sau cu un pixel alb dacă intensitatea imaginii este mai mare decât această constantă). Așa va arata imaginea după aceste operații.



```

19. # find contours in the thresholded image
20. cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
21.     cv2.CHAIN_APPROX_SIMPLE)
22. cnts = imutils.grab_contours(cnts)

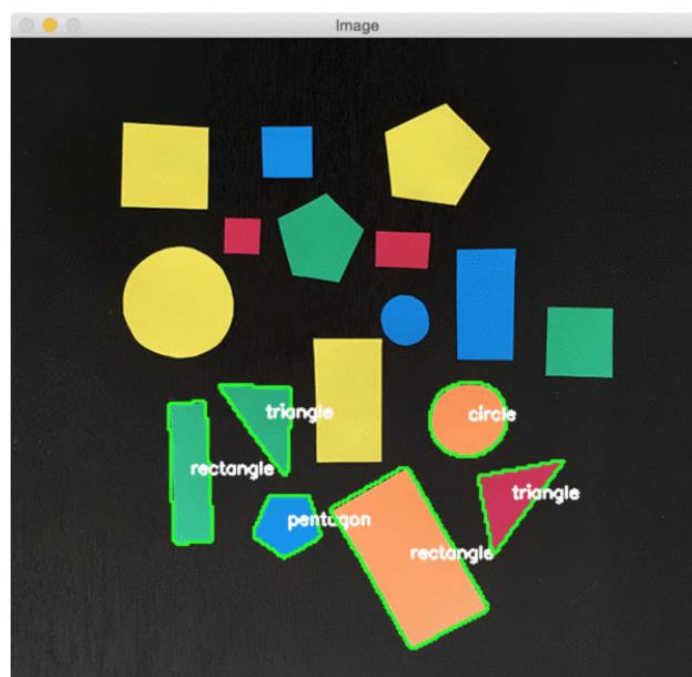
23. # loop over the contours
24. for c in cnts:
25.     # compute the center of the contour, then detect the name of the
26.     # shape using only the contour
27.     M = cv2.moments(c)
28.     cX = int((M["m10"] / M["m00"]) * ratio)
29.     cY = int((M["m01"] / M["m00"]) * ratio)
30.     shape = sd.detect(c)
31.
32.     # multiply the contour (x, y)-coordinates by the resize ratio,
33.     # then draw the contours and the name of the shape on the image
34.     c = c.astype("float")
35.     c *= ratio
36.     c = c.astype("int")
37.     cv2.drawContours(image, [c], -1, (0, 255, 0), 2)
38.     cv2.putText(image, shape, (cX, cY), cv2.FONT_HERSHEY_SIMPLEX,
39.         0.5, (255, 255, 255), 2)
40.
41. # show the output image
42. cv2.imshow("Image", image)
43. cv2.waitKey(0)

```

**Cv2.findContours** returnează setul de contururi care corespund fiecărei forme albe din imagine.

Începem să parcurgem lista de contururile individuale. Pentru fiecare dintre ele, calculăm centrul conturului, urmată de detectarea și etichetarea formei. În sfârșit, desenăm contururile și forma etichetată pe imaginea noastră.

**Cv2.drawContours** desenează conturul care înconjoară forma curentă. Apoi plasăm un cerc alb în centru (cX, cY) și scrierea centrului de text lângă cercul alb.



### 1.1.5.YOLO

#### YOLO – Sistem de Detectare a Obiectelor în timp Real

**YOLO – You Only Look Once** e un sistem de detectare a imaginilor în timp real de ultimă generație. Bazat pe rețele neuronale deja antrenate acesta aduce performanțe mari în ce privește tipul de procesare și acuratețea rezultatelor. O versiune mai recentă de YOLO o reprezintă YOLOv3 care în comparație cu alte sisteme de acest gen este mult superior.

- Cum funcționează YOLO?

Sistemele de detectare prealabilă reutilizează clasificatorii sau localizatorii pentru a efectua detectarea. Acestea aplică modelul unei imagini la mai multe locații și scale. Regiunile cu punctaj ridicat al imaginii sunt considerate detecții.

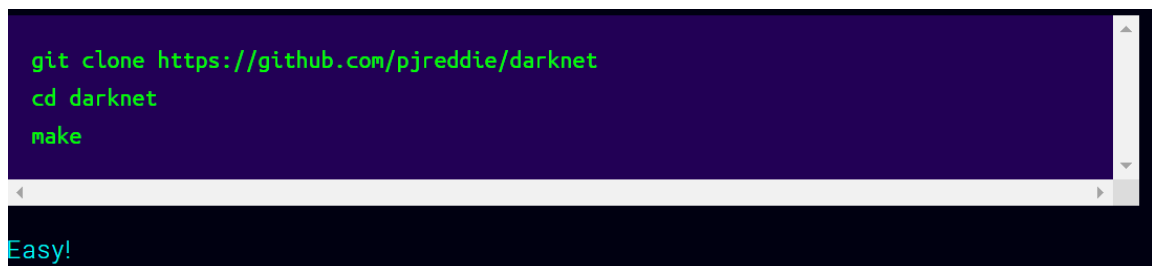
YOLO folosește o total altă abordare: aplică o singură rețea neuronală pentru imaginea completă. Această rețea împarte imaginea în regiuni și prezice căsuțele și probabilitățile de delimitare pentru fiecare regiune. Aceste căsuțe de delimitare sunt ponderate de probabilitățile prevăzute. Modelul nostru prezintă mai multe avantaje față de sistemele bazate pe clasificatori. Acesta privește întreaga imagine la momentul testării, astfel încât predicțiile sale sunt informate prin context global în imagine. De asemenea, face predicții cu o singură evaluare de rețea spre deosebire de sisteme precum R-CNN, care necesită mii pentru o singură imagine. Acest lucru îl face extrem de rapid, cu 1000x mai rapid decât R-CNN și 100x mai rapid decât R-CNN rapid. Consultați lucrarea noastră pentru mai multe detalii despre sistemul complet.

- Ce e nou în versiunea 3?

YOLOv3 folosește câteva trucuri pentru a îmbunătăți antrenamentul și a crește performanța, inclusiv: predicții pe mai multe scări, un clasificator mai bun al coloanei vertebrale și multe altele.

- Detecția folosind un model pre antrenat

Pentru a realiza acest lucru este nevoie să avem instalat Darknet. Se rulează comanda:



```
git clone https://github.com/pjreddie/darknet
cd darknet
make
```

Easy!

Acum trebuie configurat fișierul pentru YOLO în subdirectorul cfg/. Se descarcă modelul preantrenat cu numele „yolov3.weights” din sursa indicată la sfârșit, apoi se rulează comanda:

```
wget https://pjreddie.com/media/files/yolov3.weights
```

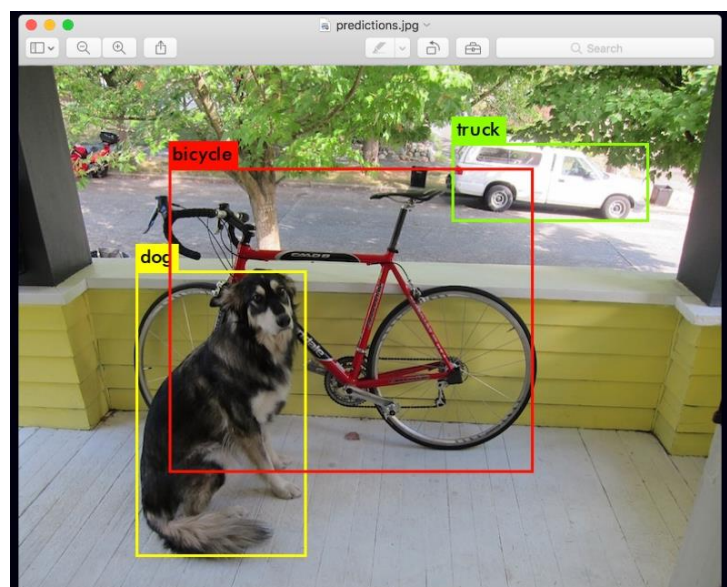
După care se rulează detectorul:

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

Ieșirea va arăta cam așa:

```
layer    filters  size      input              output
 0 conv    32  3 x 3 / 1  416 x 416 x 3  ->  416 x 416 x 32  0.299 BFLOPs
 1 conv    64  3 x 3 / 2  416 x 416 x 32  ->  208 x 208 x 64  1.595 BFLOPs
.....
105 conv   255  1 x 1 / 1   52 x  52 x 256  ->   52 x  52 x 255  0.353 BFLOPs
106 detection
truth_thresh: Using default '1.000000'
Loading weights from yolov3.weights...Done!
data/dog.jpg: Predicted in 0.029329 seconds.
dog: 99%
truck: 93%
bicycle: 99%
```

Darknet tipărește obiectele detectate, precizia cu care le-a detectat și cât a durat să o facă. Dacă nu compilăm Darknet cu OpenCV nu poate să afișeze detecțiile direct dar le salvează în fișierul „predictions.png”.



- Detecția în timp real folosind o cameră video

Pentru rularea următorului demo este necesar să rulăm Darknet cu CUDA sau OpenCV. Comanda necesară este:

```
./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights
```

YOLO va afișa FPS-ul curent și clasele previzionate, precum și imaginea cu casetele de delimitare desenate deasupra acestuia. Veți avea nevoie de o cameră web conectată la computer la care se poate conecta OpenCV. Dacă aveți mai multe webcam-uri conectate și doriți să selectați care să le utilizați, puteți trece steagul -c <num> pentru a alege (OpenCV folosește camera 0 în mod implicit).

Se poate utiliza de asemenea un videoclip dacă OpenCV îl poate citi. Comanda necesară pentru rulare cu videoclip este:

```
./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights <v
```

## 1.2 TEMA PROIECTULUI

### ❖ Scopul

Proiectul nostru urmărește o situație reală abstractizată astfel încât să se poată implementa. Situație care constă în ocolirea unor obiecte standardizate, pe un hol amenajat. Această situație se poate compara în viața reală cu un posibil autobot care va ocoli lucrurile/obiectele de pe marginea carosabilului.

### ❖ Strategie

- Folosim doar raspberry Pi (+avem senzor de distanță)
- Folosim obiecte de culori diferite(cuburi și piramide) pentru a folosi detecția de forme(pătrate sau triunghiuri)
- Detecția de forme constă în binarizare(grayscale,thresholding) și găsirea conturului(Cv2.findContours)
- Sau o să folosim YOLO, care are o rețea neuronală antrenată pe mii de clase de obiecte.
- O să mergem în paralel și cu YOLO și cu detecția de forme și vedem care va funcționa mai bine
- Detecție de culori (Roșu =oprire) . Dacă vede obiecte roșii să se oprească.

### ❖ Traseul

Traseul va consta într-un hol în care vor fi puse cutii de culori diferite și forme diferite(cuburi și piramide). În funcție de forma cutii, robotul va ocoli obstacolul prin partea stângă sau dreaptă. Pentru piramide(triunghiuri) se va ocoli pe partea dreaptă, iar pentru cuburi(pătrate) pe partea stângă.

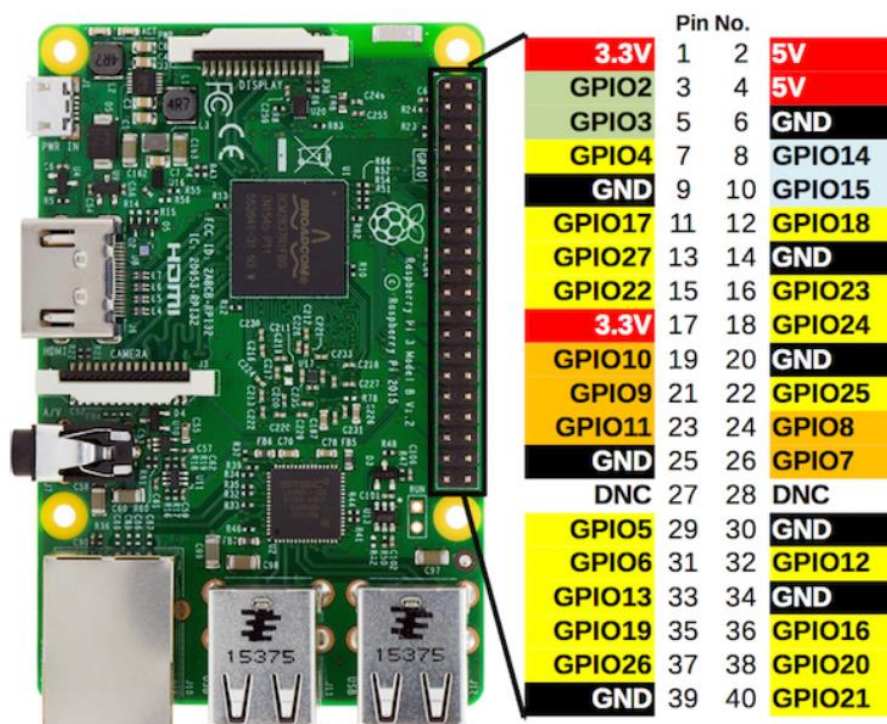
## 1.3 DESCRIERE CAPITOLE

1. **Introducere** – conține cele 4 lucrări științifice realizate de fiecare persoană din echipă (documentare), o mică descriere a temei, scopului și a strategiei aplicate în proiect.
2. **Arhitectura generală a sistemului** – conține arhitectura hard a robotului, elementele necesare pentru realizarea conectării acestuia, diagramele de stare și organigramele axate pe mișcarea robotului
3. **Dezvoltări mecatronice** – conține procesarea de imagine, câteva exemple, poze din timpul încercării acesteia.

## 2. ARHITECTURA GENERALĂ A SISTEMULUI

### 2.1. ARHITECTURA GENERALĂ

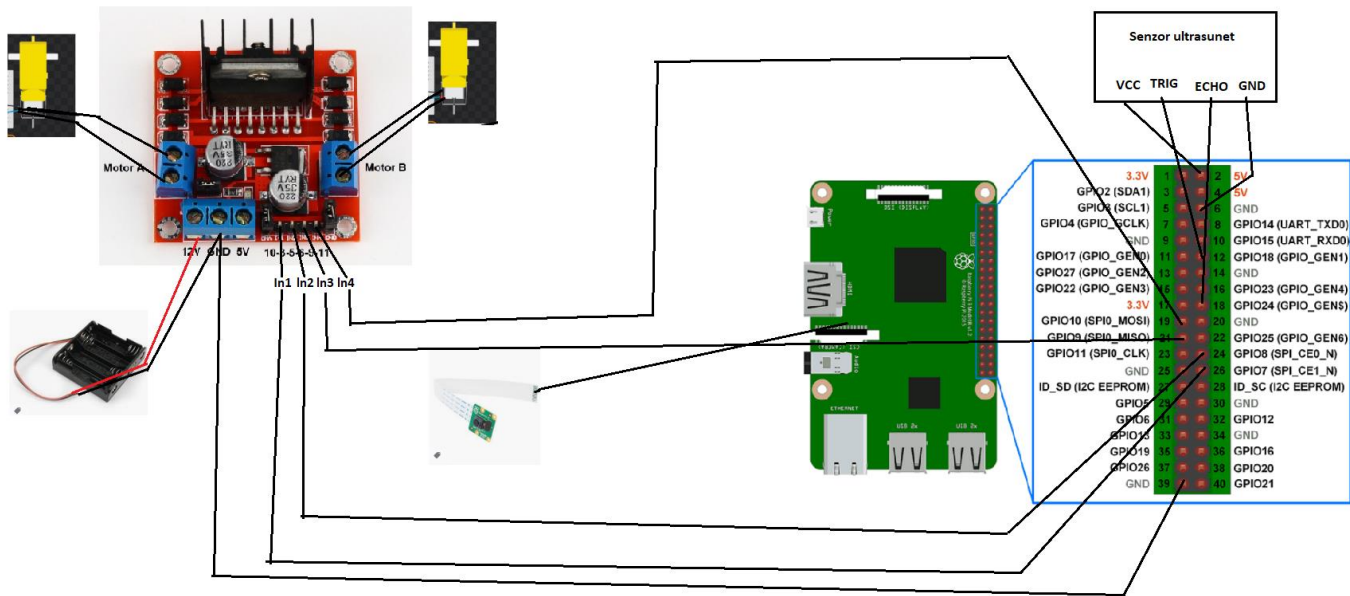
Roboțelul are următoarele componente : Camera Raspberry Pi , Senzor ultrasunet, Driver motor L298 care se leagă la motoarele de la roți , tot de el se leagă și suportul de bateri . Toate componentele se conectează la o placuța Raspberry Pi



- Fiecare motor se leaga cu cate doua fire de driver
- Suportul de baterii se conecteaza cu doua fire la driver (GND-ul suportului la GND-ul driverului , VCC-ul suportului la 12 V al driverului)
- Driverul se conecteaza cu cinci fire la placuța:
  - GND cu GND
  - Ln 1 cu pinul 26(GPIO7)
  - Ln 2 cu pinul 24 (GPIO 8)
  - Ln 3 cu pinul 21(GPIO 9)
  - Ln 4 cu pinul 19(GPIO 10)
- Camera Pi se conectează ca în imaginea de mai jos
- Senzorul cu ultrasunet se conectează cu patru fire la placuța:
  - Vcc cu pinul 2(5V)
  - TRIG cu pinul 12 (GPIO18)
  - ECHO cu pinul 18 (GPIO24)



- GND cu pinul 6 (GND)



## Lista de componente

- roți și motoarele lor



- Șasiu mare pentru roboțel



- Driver motor L298





- Fire :mamă-tată, mamă-mamă, tată-tată



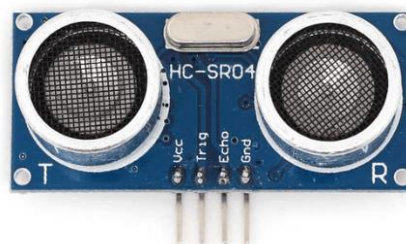
- Rotila 25 mm
- Suport de baterie



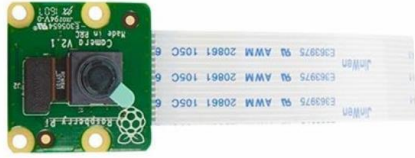
- Baterii
- Baterie externă pentru Raspberry
- Raspberry Pi 3 v1.2



- Senzor ultrasunet(este optional)



➤ **Camera Raspberry Pi**



➤ **Piulițe**

➤ **Intrerupator roșu**



➤ **Șuruburi (diverse mărimi)**

➤ **4 suport motor robot acril**



➤ **Suport rotita robot din placaj**



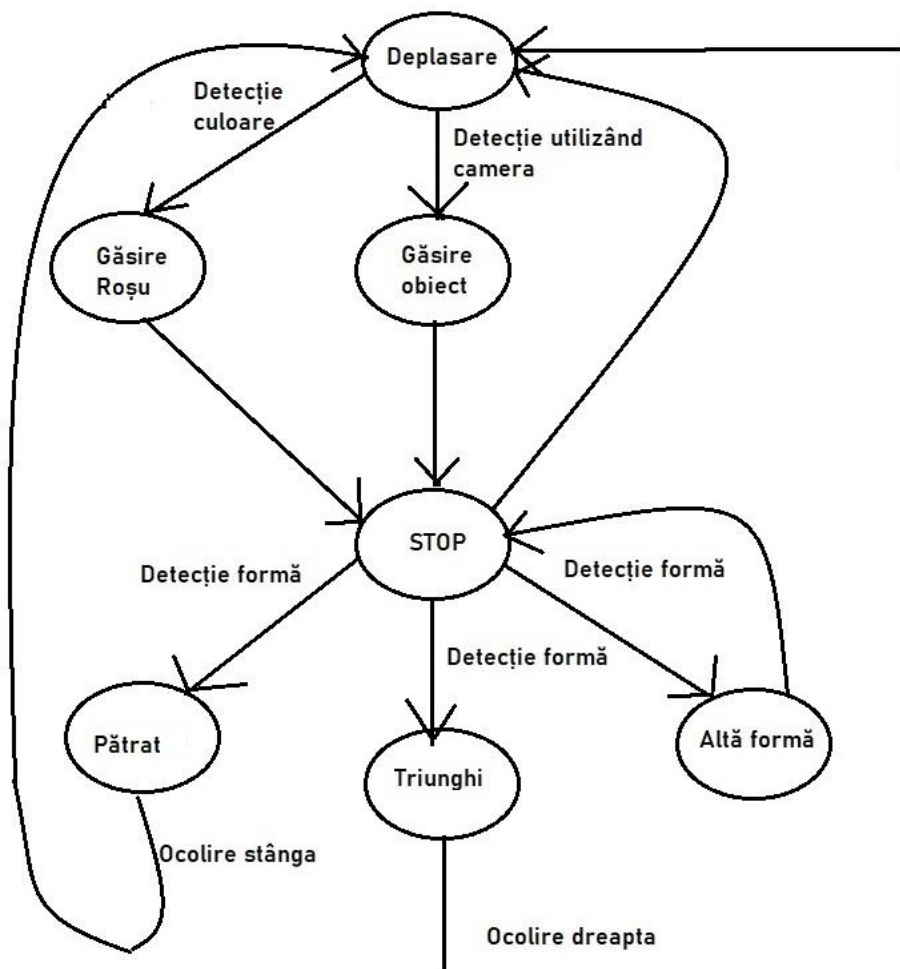
## 2.2. REGIMURI DE FUNCȚIONARE

Diagrame de stare:

Specificații:

- Se deplasează înainte pe un hol
- Oprește la culoarea roșie
- Oprește la detectarea unui obiect
- Detectarea formei obiectului
- Execuție ocolire în funcție de formă (triunghi, pătrat, orice altceva).

Identificarea tuturor stărilor semnificative:

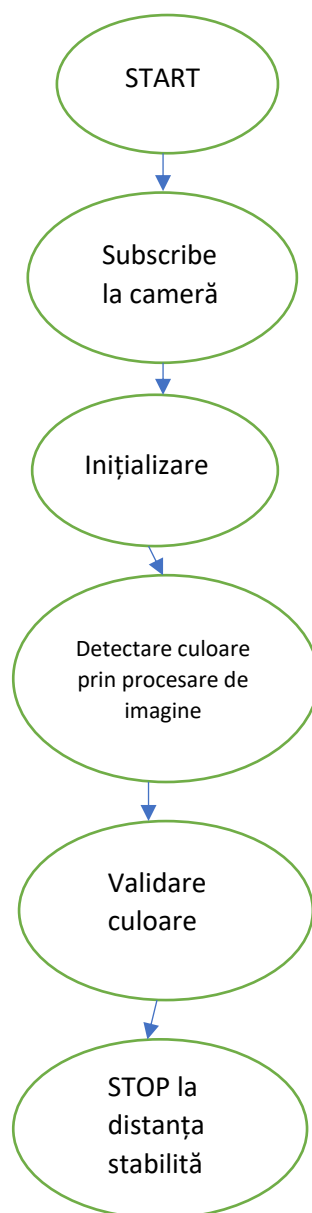


Organigrame axate pe mișcarea robotului:

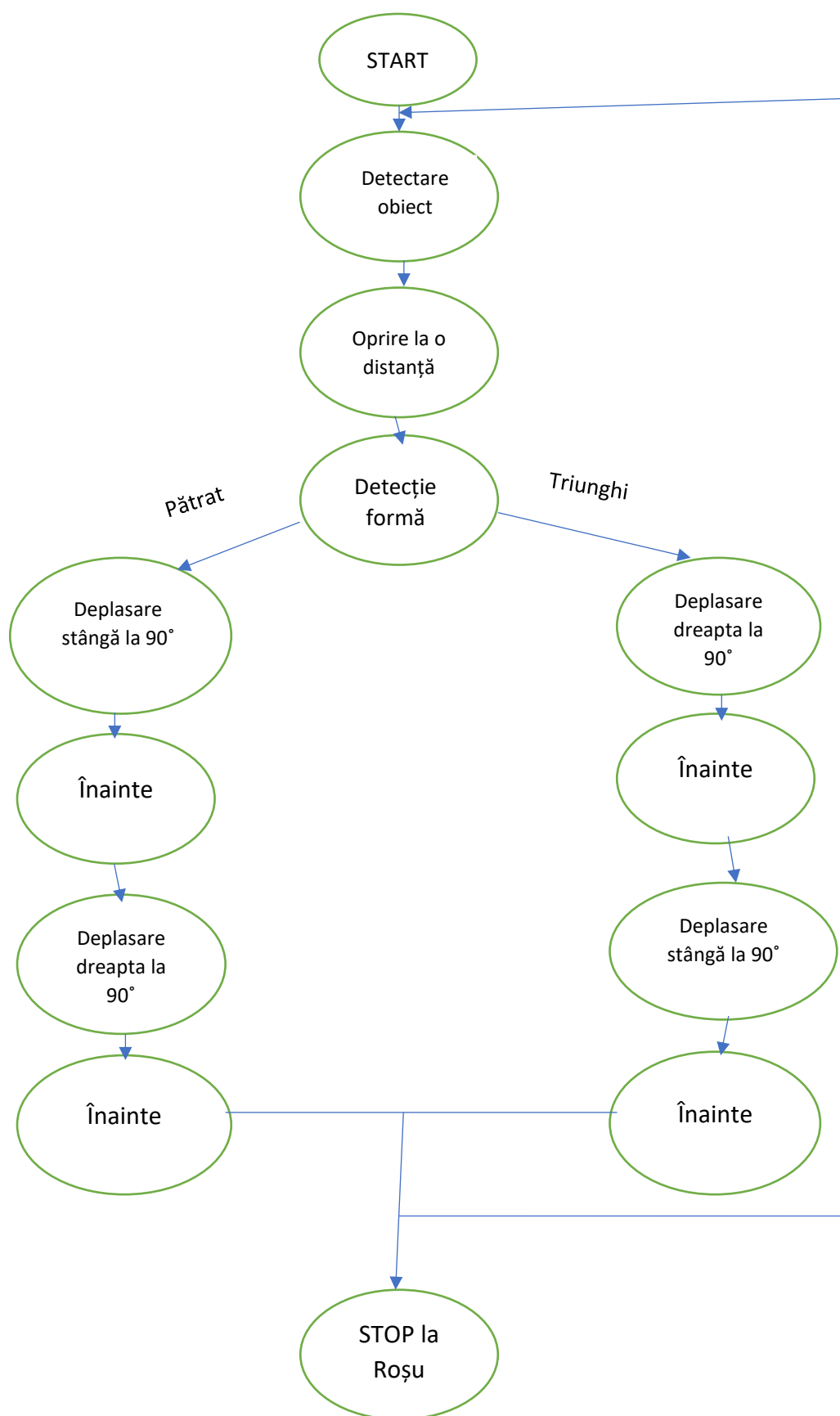
Toată mișcarea robotului e permisă doar în cadrul holului, cu condiția să se deplaseze înainte fără a fi nevoie să se deplaseze înapoi.

- Mișcarea înainte și oprirea la culoarea roșie
- Detectarea obiectului și executarea ocolirii
- Reglarea robotului spre mijlocul holului (cu senzor de distanță)  
-Distanța marginii holului fiind stabilită de la început

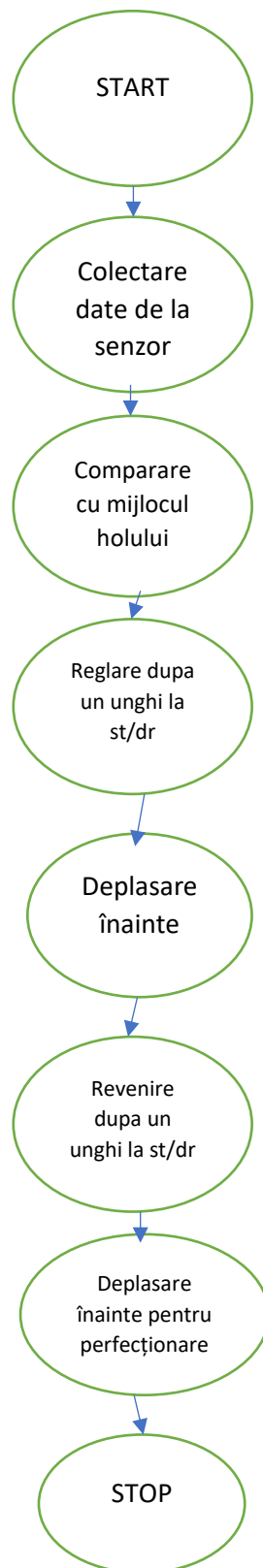
*1. Mișcarea înainte și oprirea la culoarea roșie*



## 2. Detectarea obiectului și executarea ocolirii



3. *Reglarea robotului spre mijlocul holului (cu senzor de distanță)*



### 3. DEZVOLTĂRI MECATRONICE

#### 3.1 CODURI PENTRU TESTAREA COMPONENTELOR:

##### 3.1.1. Driver

Pentru a testa driverul și motoarele roților parcurgem următorii pași:

- Deschidem un editor Python 3, cum ar fi Thonny Python IDE
- Scriem codul:

```
from gpiozero import Robot
robby = Robot(left=(7,8), right=(9,10))
```

- Salvăm fișierul și îl numim **robby.py**. Putem rula apoi făcând click pe **RUN**.
- Deschidem un shell python făcând click pe pictograma terminalului din bara de activități din partea de sus a ecranului, apoi tastăm „python” și apăsăm Enter. Apoi testăm driverul cu următoarele două funcții: **robby.forward()** => pentru a merge și **robot.stop()** => pentru a opri
- Funcțiile pentru deplasarea robotului sunt:
  - robot.forward() => pentru a merge în fata
  - robot.backward() => pentru a merge în spate
  - robot.right() => pentru a merge în dreapta
  - robot.left() => pentru a merge în stanga
  - robot.stop() => pentru a se opri
  -

##### 3.1.2. Senzor ultrasunet

Pentru a testa senzor parcurgem următorii pași:

- Instalată biblioteca Python GPIO
- Creare fișier nou :

```
sudo nano ultrasonic_distance.py
```

- Scriem următorul cod în fișier:

```

1 #Libraries
2 import RPi.GPIO as GPIO
3 import time
4
5 #GPIO Mode (BOARD / BCM)
6 GPIO.setmode(GPIO.BCM)
7
8 #set GPIO Pins
9 GPIO_TRIGGER = 18
10 GPIO_ECHO = 24
11
12 #set GPIO direction (IN / OUT)
13 GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
14 GPIO.setup(GPIO_ECHO, GPIO.IN)
15
16 def distance():
17     # set Trigger to HIGH
18     GPIO.output(GPIO_TRIGGER, True)
19
20     # set Trigger after 0.01ms to LOW
21     time.sleep(0.00001)
22     GPIO.output(GPIO_TRIGGER, False)
23
24     StartTime = time.time()
25     StopTime = time.time()
26
27     # save StartTime
28     while GPIO.input(GPIO_ECHO) == 0:
29         StartTime = time.time()
30
31     # save time of arrival
32     while GPIO.input(GPIO_ECHO) == 1:
33         StopTime = time.time()
34
35     # time difference between start and arrival
36     TimeElapsed = StopTime - StartTime
37     # multiply with the sonic speed (34300 cm/s)
38     # and divide by 2, because there and back
39     distance = (TimeElapsed * 34300) / 2
40
41     return distance
42
43 if __name__ == '__main__':
44     try:
45         while True:
46             dist = distance()
47             print ("Measured Distance = %.1f cm" % dist)
48             time.sleep(1)
49
50     # Reset by pressing CTRL + C
51     except KeyboardInterrupt:
52         print("Measurement stopped by User")
53         GPIO.cleanup()

```

- Vom rula cu comanda :

```
sudo python ultrasonic_distance.py
```

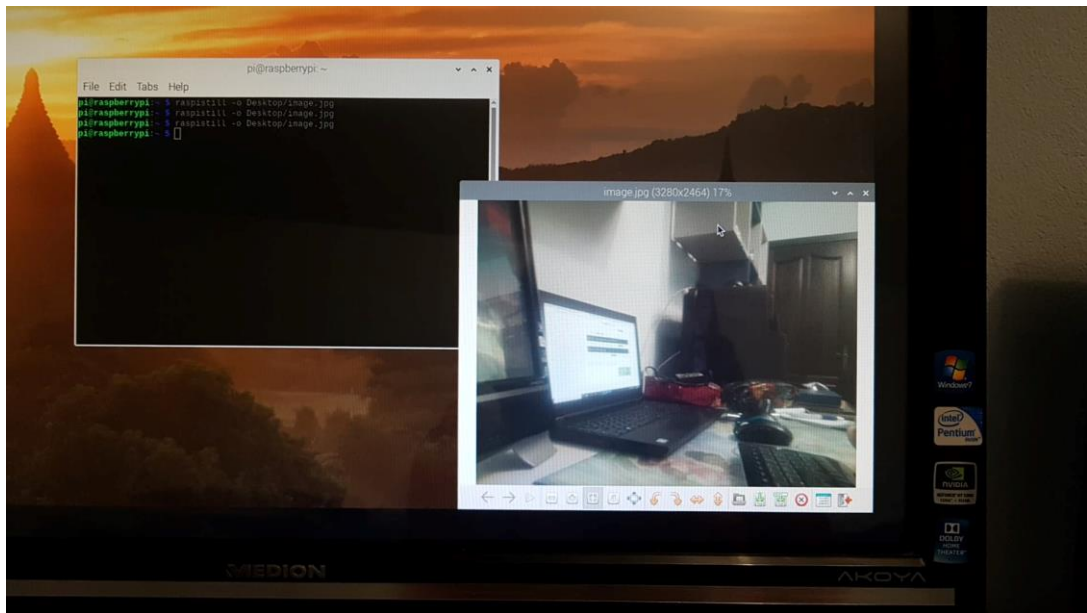
Așadar, în fiecare secundă, distanța va fi măsurată până când fișierul este anulat prin apăsarea CTRL + C.



### 3.1.3. Camera raspberry Pi

Pentru a testa si a configura camera parcurgem următorii pași:

- Accesam meniul principal și deschidem Raspberry Pi Configuration
- Selectam Interfaces și ne asigurăm că aparatul foto este activat
- Tastăm comanda **raspistill -o Desktop/image.jpg** în terminal si obținem prima fotografie care se salvează pe desktop



- Dacă lucrăm în Python, avem biblioteca picamera care permite să ne controlăm modulul de cameră
- Deschidem un editor Python 3, cum ar fi Thonny Python IDE
- Deschidem un nou fișier și salvăm cu numele de camera.py.
- Scriem următorul cod în fișier :

```
from picamera import PiCamera
from time import sleep
camera = PiCamera()
camera.start_preview()
sleep(5)
camera.stop_preview()
```

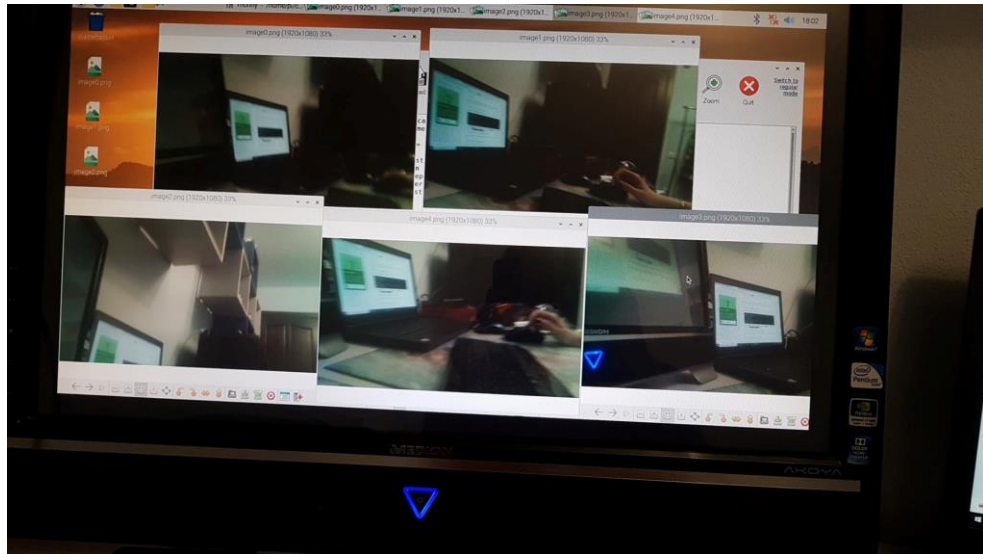
- Salvăm și rulăm programul. Programul afișată timp de cinci secunde imaginea pe care ne vede camera și apoi din nou o închidem.
- Utilizăm acum modulul camerei și Python pentru a face cinci fotografii. Iar codul îl găsim mai jos:

```
camera.start_preview()
for i in range(5):
```

```

sleep(5)
camera.capture('/home/pi/Desktop/image%s.jpg' % i)
camera.stop_preview()

```



- Acum o sa înregistram 5 secunde cu camera
- Codul nostru este acesta:

```

camera.start_preview()
camera.start_recording('/home/pi/Desktop/video.h264')
sleep(5)
camera.stop_recording()
camera.stop_preview()

```

- Pentru a scrie un text și a-i schimba culoarea pe o imagine folosim următoarea bibliotecă :

```

from picamera import PiCamera, Color

```

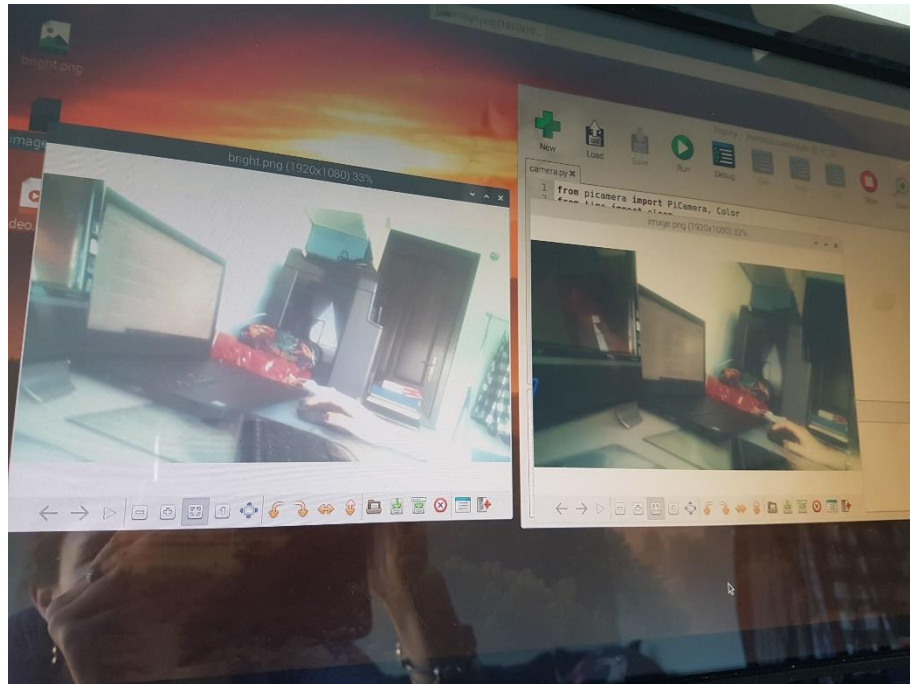
și următoarele funcții:

```

camera.annotate_background = Color('blue'),
camera.annotate_text = " Hello world "

```

- Putem sa schimbam luminozitatea pozei cu ajutorul funcției **camera.brightness**. Luminozitatea implicită este 50 și o putem seta la orice valoare cuprinsă între 0 și 100.



### 3.2. Captura de imagine cu OpenCV

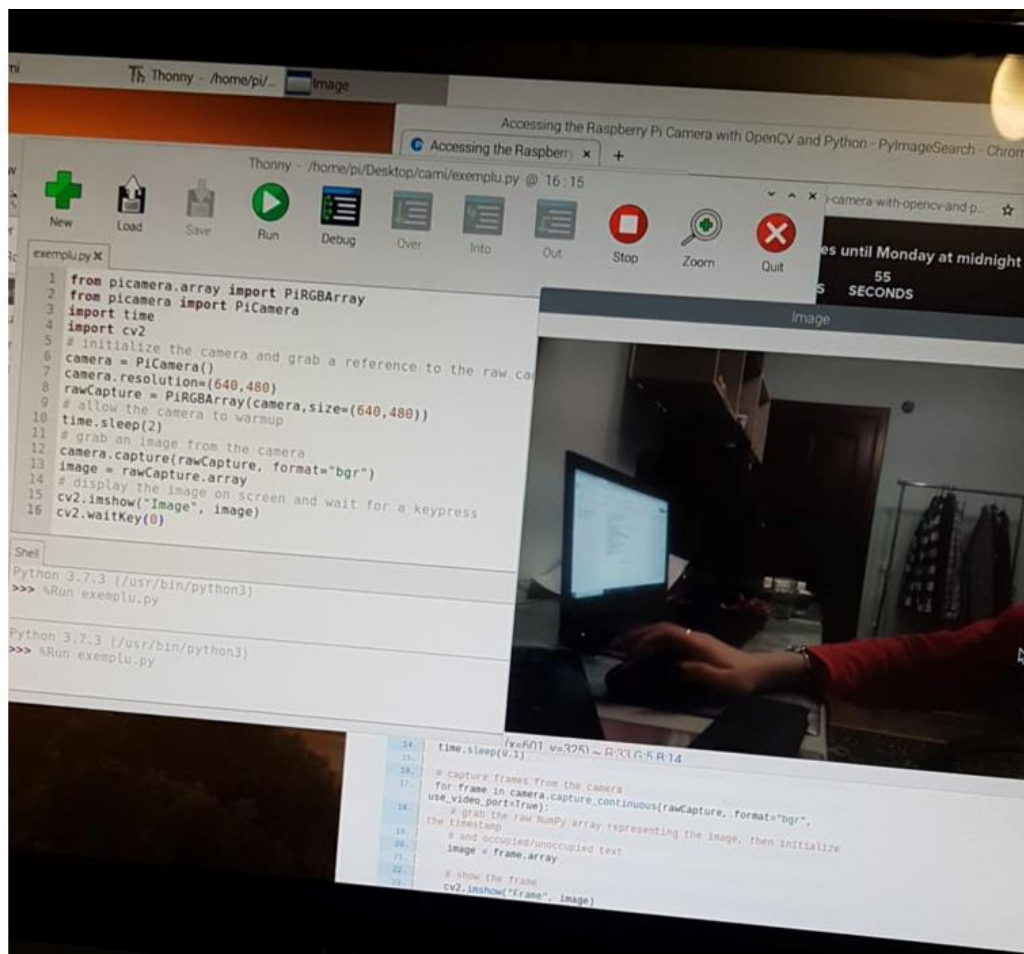
```
# import pentru pachetele necesare
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2

# initializam camera, schimbam rezolutia si capturam o
referinta a camerei
camera = PiCamera()
camera.resolution = (640, 480)
rawCapture = PiRGBArray(camera, size=(640, 480))

# lăsăm 2 secunde ca sa isi faca camera inițializarea
time.sleep(2)

# reusim sa prindem o imagine de la camera
camera.capture(rawCapture, format="bgr")
image = rawCapture.array

# afisam imaginea
cv2.imshow("Image", image)
cv2.waitKey(0)
```



### 3.3.Real time cu OpenCV

```
# import pentru pachetele necesare
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
```

```
# 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))
```

```
# lăsaș 2 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

# afișează cadrele
    cv2.imshow("Frame", image)
    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

```

### 3.4. Procesare de imagine

#### 1. Transformă o imagine obișnuită într-o imagine alb-negru(grayscale)

- folosim funcția:

```

cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

# import pentru pachetele necesare
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2

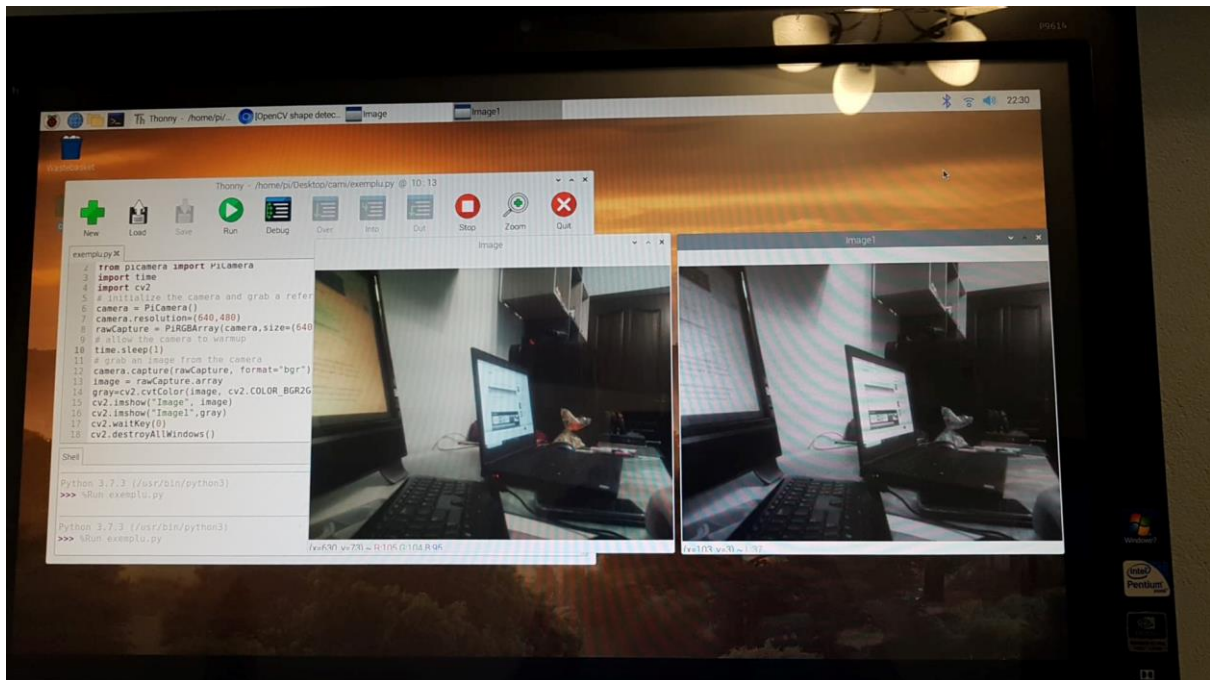
# initializam camera, schimbam rezolutia si capturam o
referinta a camerei
camera = PiCamera()
camera.resolution = (640, 480)
rawCapture = PiRGBArray(camera,size=(640, 480))

# lăsăm 2 secunde ca sa isi faca camera inițializarea
time.sleep(2)

# reusim sa prinde o imagine de la camera
camera.capture(rawCapture, format="bgr")
image = rawCapture.array

```

```
# transforma o imagine obișnuită într-o imagine alb-negru
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# afisam imaginile
cv2.imshow("Image", image)
cv2.imshow("Image1", gray)
cv2.waitKey(0)
```



## 2. Bluram imaginea alb-negru

- folosim funcția:

```
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```

```
# import pentru pachetele necesare
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
```

```
# initializam camera, schimbam rezolutia si capturam o
referinta a camerei
camera = PiCamera()
camera.resolution = (640, 480)
rawCapture = PiRGBArray(camera, size=(640, 480))
```

```
# lăsăm 2 secunde ca sa isi faca camera inițializarea
time.sleep(2)
```



```

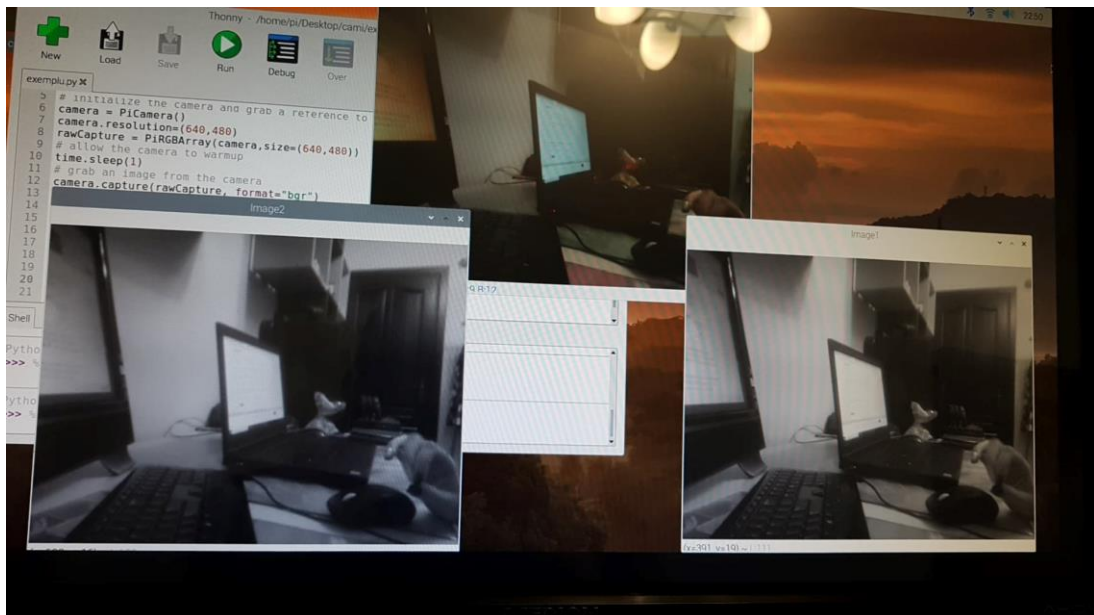
# reusim sa prinde o imagine de la camera
camera.capture(rawCapture, format="bgr")
image = rawCapture.array

# transforma o imagine obisnuita intr-o imagine alb-negru
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# bluram imaginea
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# afisam imaginile
cv2.imshow("Image", image)
cv2.imshow("Image1", gray)
cv2.imshow("Image2", blurred)
cv2.waitKey(0)

```



### 3. Threshold(binărizare)

- folosim functia:

```
thresh = cv2.threshold(blurred, 80, 255, cv2.THRESH_BINARY)
```

```

# import pentru pachetele necesare
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2

```

```

# initializam camera, schimbam rezolutia si capturam o
referinta a camerei
camera = PiCamera()
camera.resolution = (640, 480)
rawCapture = PiRGBArray(camera,size=(640, 480))

# lăsăm 2 secunde ca sa isi faca camera inițializarea
time.sleep(2)

# reusim sa prinde o imagine de la camera
camera.capture(rawCapture, format="bgr")
image = rawCapture.array

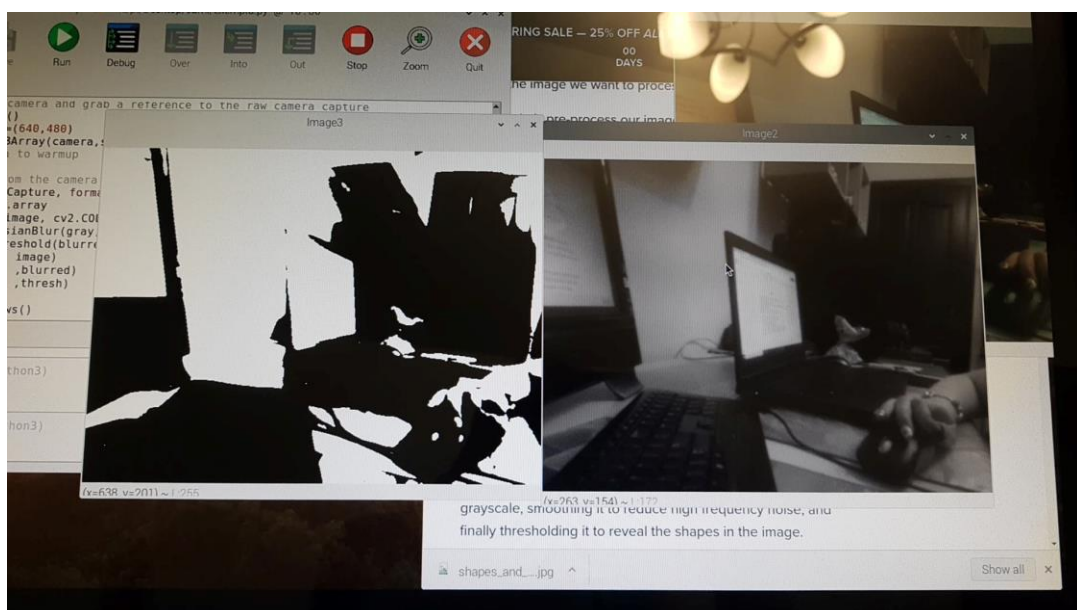
# transforma o imagine obișnuită într-o imagine alb-negru
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# bluram imaginea
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# tot ce are culoare inchisa face negru ,restul il face alb
thresh = cv2.threshold(blurred, 80, 255,
cv2.THRESH_BINARY)

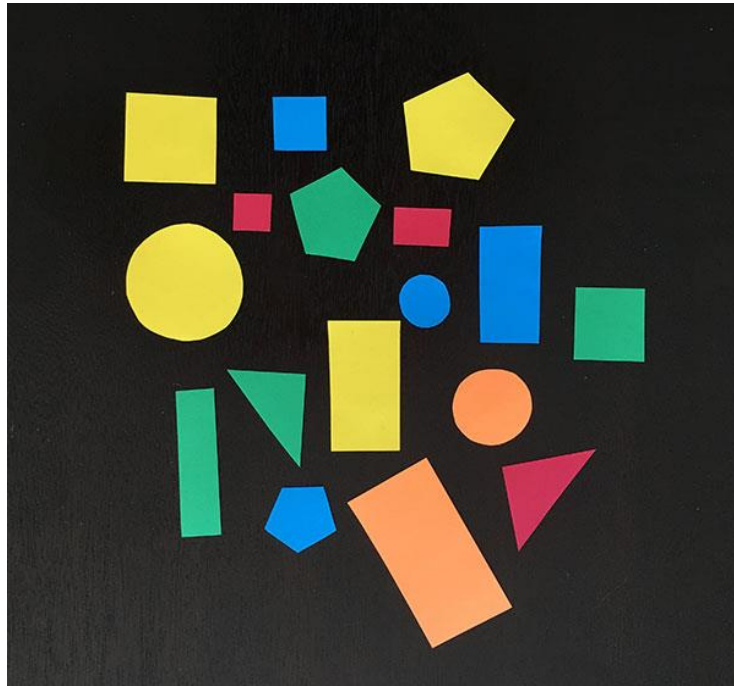
# afisam imaginile
cv2.imshow("Image", image)
cv2.imshow("Image2", blurred)
cv2.imshow("Image3", blurred)
cv2.waitKey(0)

```





### 3.5. Detectia de forme- prima data pe o imagine cu forme

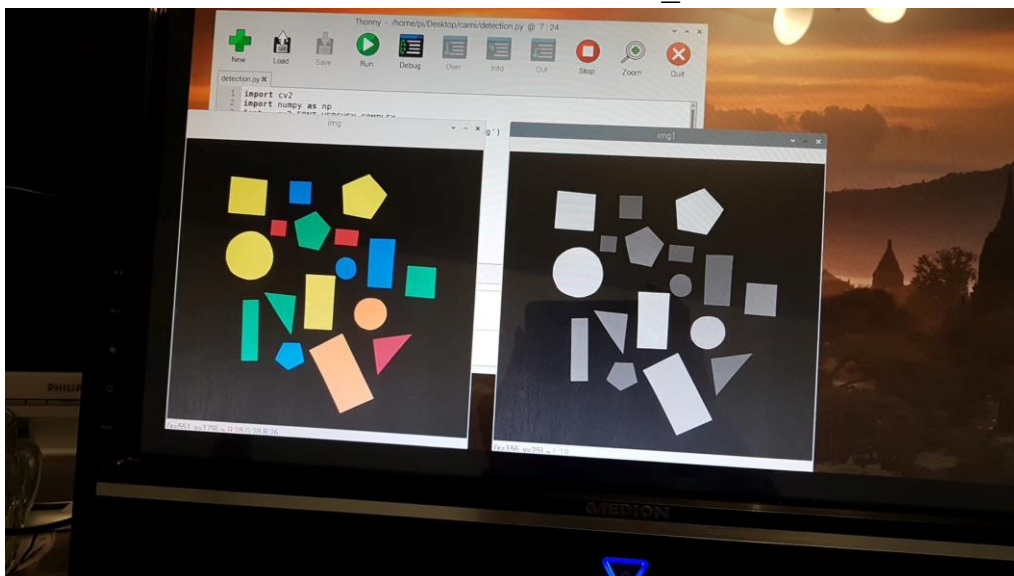


```
import cv2
import numpy as np

font = cv2.FONT_HERSHEY_COMPLEX_SMALL

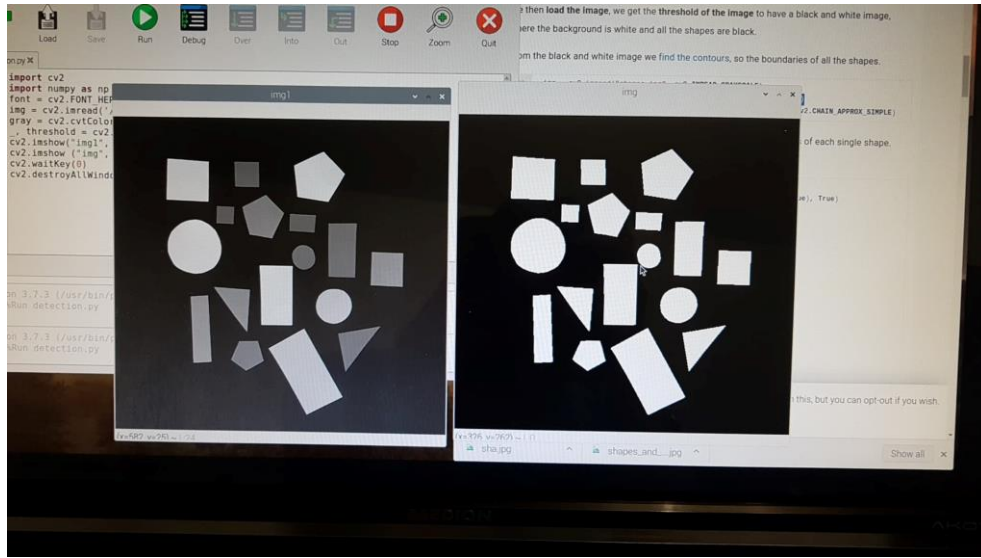
#luăm imaginea
img = cv2.imread("shapes.jpg")

# transforma o imagine obișnuită într-o imagine alb-negru
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```



```
# bluram imaginea ca sa scapam de impuritati
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# tot ce are culoare inchisa face negru ,restul il face
alb(binarizare)
_, threshold = cv2.threshold(blurred, 80, 255,
cv2.THRESH_BINARY)
```



```
# luam contururile pentru figurile albe
_, contours, _ = cv2.findContours(threshold,
cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

#Începem să parcurgem lista de contururile individuale. Pentru fiecare dintre ele, desenăm contururile, vedem cate varfuri are ca sa stim ce forma avem și etichetarea formei.

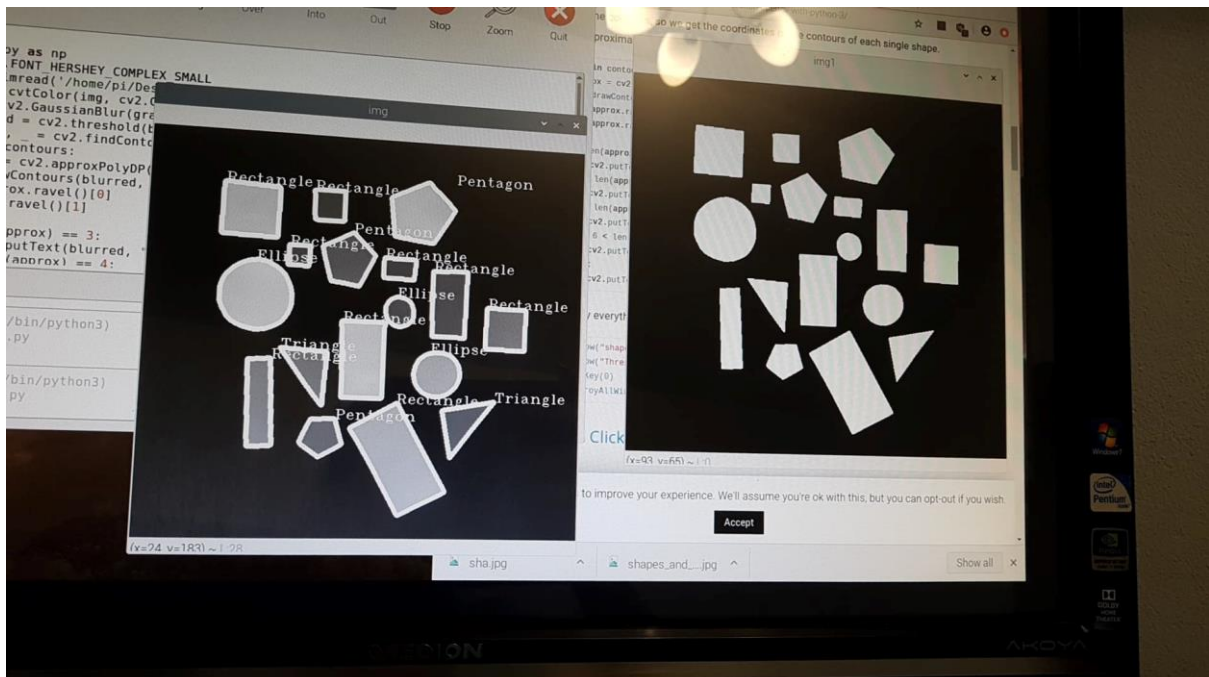
```
for cnt in contours:
    approx = cv2.approxPolyDP(cnt,
0.01*cv2.arcLength(cnt, True), True)
    cv2.drawContours(img, [approx], 0, (255), 5)
    x = approx.ravel()[0]
    y = approx.ravel()[1]

    if len(approx) == 3:
        cv2.putText(blurred, "Triangle", (x, y), font, 1,
(255))
    elif len(approx) == 4:
        cv2.putText(blurred, "Rectangle", (x, y), font, 1,
(255))
```

```

elif len(approx) == 5:
    cv2.putText(blurred, "Pentagon", (x, y), font, 1,
        (255))
elif 6 < len(approx) < 15:
    cv2.putText(blurred, "Ellipse", (x, y), font, 1, (255))
else:
    cv2.putText(blurred, "Circle", (x, y), font, 1, (255))
cv2.imshow("shapes", blurred)
cv2.imshow("Threshold", threshold)
cv2.waitKey(0)
cv2.destroyAllWindows()

```



#### 4. BIBLIOGRAFIE

1. <https://www.eurasip.org/Proceedings/Eusipco/Eusipco2017/papers/1570343986.pdf?fbclid=IwAR1jQIdCu2rkuEZAd598v3xF68rTl3PHRJkOj6KzxEVX6uR1yOVuLmJUR0U>
2. <https://www.youtube.com/watch?v=QV1a1G4IL3U&fbclid=IwAR3vyUoVGW1quuh4IpVgeXOEr7tITAJDySEZCALVHJM9vKxDvdedGNOz4dg>
3. <https://www.youtube.com/watch?v=QV1a1G4IL3U&fbclid=IwAR1jQIdCu2rkuEZAd598v3xF68rTl3PHRJkOj6KzxEVX6uR1yOVuLmJUR0U>
4. <https://www.youtube.com/watch?v=iMZnJou5Umc&fbclid=IwAR1-TAiuJVwDih6SXq4p9zFnb5GNdNxtpyL7jQvYYBBpnS4nt59Fj7LUjlg>
5. [https://www.analyticsvidhya.com/blog/2018/06/understanding-building-object-detection-model-python/?fbclid=IwAR3h7iXe\\_bRJnACIetQFyhXaQHPzwbq1uUAAtY-94JLN1S15WbbVkazMBEtg](https://www.analyticsvidhya.com/blog/2018/06/understanding-building-object-detection-model-python/?fbclid=IwAR3h7iXe_bRJnACIetQFyhXaQHPzwbq1uUAAtY-94JLN1S15WbbVkazMBEtg)
6. <https://towardsdatascience.com/all-the-steps-to-build-your-first-image-classifier-with-code-cf244b015799>
7. [https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/?fbclid=IwAR0qp3ovKv4lp-tYed6H-Mli4IZU\\_OQPnKuxCLO4gfTUvgVXfIRapeWPDT4](https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/?fbclid=IwAR0qp3ovKv4lp-tYed6H-Mli4IZU_OQPnKuxCLO4gfTUvgVXfIRapeWPDT4)
8. [https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/?fbclid=IwAR1zlKNAN2a6XTPVnq7J4X7OODTHPwUVuEcTOnHlVdq\\_q9a-5xXKPEkO8uw](https://www.pyimagesearch.com/2017/08/21/deep-learning-with-opencv/?fbclid=IwAR1zlKNAN2a6XTPVnq7J4X7OODTHPwUVuEcTOnHlVdq_q9a-5xXKPEkO8uw)
9. <https://www.pyimagesearch.com/2016/02/01/opencv-center-of-contour/>
10. <https://www.pyimagesearch.com/2016/02/08/opencv-shape-detection/>
11. <https://projects.raspberrypi.org/en/projects/build-a-buggy/>
12. <https://projects.raspberrypi.org/en/projects/getting-started-with-picamera>
13. <https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/>
14. <https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/>
15. <https://pysource.com/2018/09/25/simple-shape-detection-opencv-with-python-3/>
16. <https://pysource.com/2018/12/29/real-time-shape-detection-opencv-with-python-3/>