UNIVERSITATEA “POLITEHNICA” TIMIȘOARA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL AUTOMATICĂ ȘI INFORMATICĂ

PROIECT SINCRETIC I

Conducerea roboților

Studenți:

BURLACU CAMELIA-ADELINA, GR. 1.2

MESTER ARANKA-NIKOLETTA, GR. 3.2

GUBICS MARCUS, GR. 3.1

GHEROGHIȚĂ-ADRIAN, GR. 3.1

MOISĂ GABRIEL, GR. 4.1

COORDINATORI: Conf.dr.ing. Florin DRĂGAN, As. Ing. Andrei PRODANIUC

THE TRANSFORMERS

CUPRINS

INTRODUCERE

Un robot este un operator mecanic sau virtual, artificial. Robotul este un sistem compus din mai multe elemente: mecanică, senzori și actuatori precum și un mecanism de direcționare. Mecanica stabilește înfățișarea robotului și mișcările posibile pe timp de funcționare. Senzorii și actuatorii sunt întrebuințați la interacțiunea cu mediul sistemului. Mecanismul de direcționare are grijă ca robotul să-și îndeplinească obiectivul cu succes, evaluând de exemplu informațiile senzorilor. Acest mecanism reglează motoarele și planifică mișcările care trebuiesc efectuate.

Construcţia unui robot cere cunoştinţe din domenii foarte diferite. Pentru a îndeplini chiar o misiune foarte simplă, este nevoie de sisteme complicate, care acoperă multe discipline.

În mare, robotica poate fi divizată în trei domenii: *percepţie*, *cogniţie* şi *acţiune*. Această diviziune e naturală: un robot trebuie în general să ``simtă", pentru a primi informaţii despre mediul înconjurător. Informaţiile în sine însă nu folosesc la nimic: robotul trebuie sa “inţeleagă"” ce se petrece, să construiască planuri, să evalueze situaţii, etc. Aceasta este partea de cogniţie. Un robot ar fi inutil dacă nu ar putea să *facă* ceva: să se deplaseze, să transforme în mod intenţionat mediul înconjurător, să exploreze, într-un cuvînt, să acţioneze.

În anumite cazuri putem elimina cogniţia, obţinînd fie ceea ce se numeşte *teleoperare* (operare de la distanţă), în care caz nu există decizie la nivelul robotului, fie celebrii roboţi lucrători la o linie de asamblare.

Scurt istoric

Bazele roboților de azi stau mult mai departe. Primele modele de mașini pot fi mai degrabă numite automate (provenind din grecescul *automatos*, care se mișcă singur). Acestea nu puteau executa decât câte un singur obiectiv, fiind constrânse de construcție.

Cu descoperirea ceasului mecanic din secolul XIV s-a deschis calea unor posibilități noi și complexe. Nu mult după aceea au apărut primele mașini, care semănau îndepărtat cu roboții de azi.

Dezvoltarea electrotehnicii din secolul XX a adus cu sine și o dezvoltare a roboticii. Printre primii roboți mobili se numără sistemul *Elmer şi Elsie* construit de William Grey Walter în anul 1948. Aceste triciclete se puteau îndrepta spre o sursă de lumină și puteau să recunoască coliziuni în împrejurimi. Anul 1956 este considerat ca anul nașterii a robotului industrial.

Clasficarea roboţilor

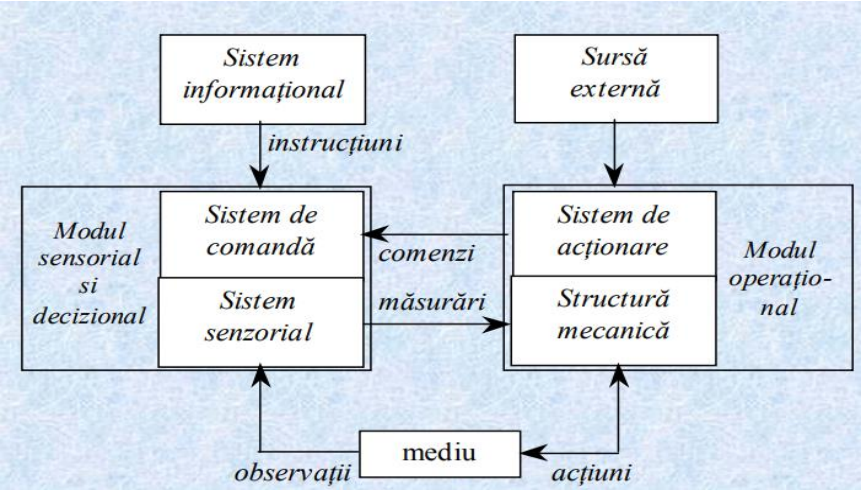
- din punctul de vedere al gradului de mobilitate se cunosc roboţi fixi si mobili;
- din punct de vedere al informaţiei de intrare şi a metodei de instruire există:
  - ❖ roboţi acţionaţi de om,
  - ❖ roboţi cu sistem de comandă cu relee (secvenţial);
  - ❖ roboţi cu sistem secvenţial cu program modificabil;
  - ❖ roboţi repetitori (cu programare prin instruire);
  - ❖ roboţi inteligenţi;
- din punct de vedere al sistemului de coordonate roboţii sunt în sistem de coordonate carteziene (18%), cilindrice (33%) şi sferice (40%);
- din punct de vedere al sistemului de comandă:
  - ❖ comandă punct cu punct (unde nu interesează traiectoria propriu zisă);
  - ❖ comandă pe contur (implică coordonarea mişcării axelor);
  - ❖ comandă pe întreaga traiectorie (implică toţi parametrii de mişcare);
- din punct de vedere al sarcinii manipulate;
- din punct de vedere al sistemului de acţionare: hidraulică (40%), electrică (30%), pneumatică (21%), mixtă;
- din punct de vedere al preciziei de poziţionare: sub 0,1mm, (0,1 ¸ 0,5)mm, (0,5 ¸ 1)mm, (1 ¸ 3)mm, peste 3mm;
- din punctul de vedere al tipului de programare:
  - ❖ cu programare rigidă (fără posibilităţi de corecţie);
  - ❖ cu programare flexibilă (există posibilitatea modificării programului);
  - ❖ cu programare adaptivă (există posibilitatea adaptării automate a programului în timpul funcţionării);

®Un robot este un sistem mecanic articulat, cu mai multe grade de libertate, ce asigură deplasarea pe verticală a obiectelor cu controlul deplasarii.

Arhitectura internă a unui robot conţine cinci sisteme importante, fiecare dintre acestea aparţinând unui domeniu al tehnicii clasice:

- sistemul mecanic de susţinere şi al articulaţiilor (cuplelor de rotaţie şi de translaţie);
- sistemul de acţionare (hidraulic, pneumatic electric sau mixt);
- sistemul de transmisie al mişcării;
- sistemul senzorial (intern şi extern);
- sistemul decizional.

Schema bloc a unui robot

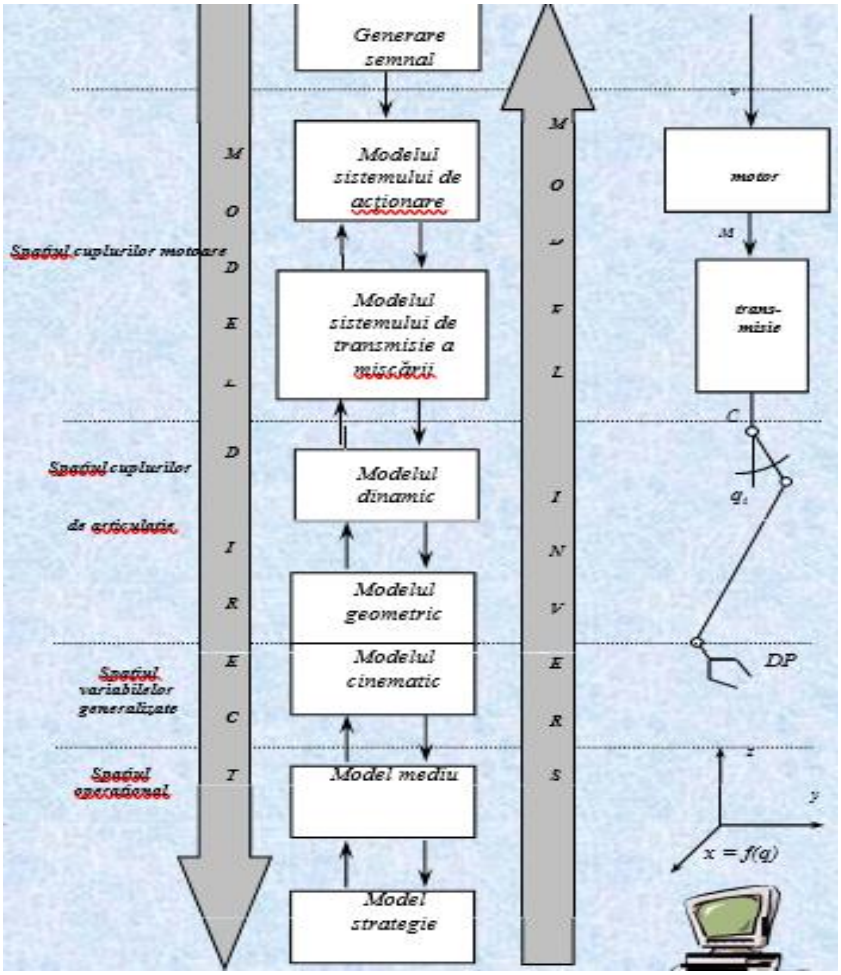


Metode de modelare a unui robot:

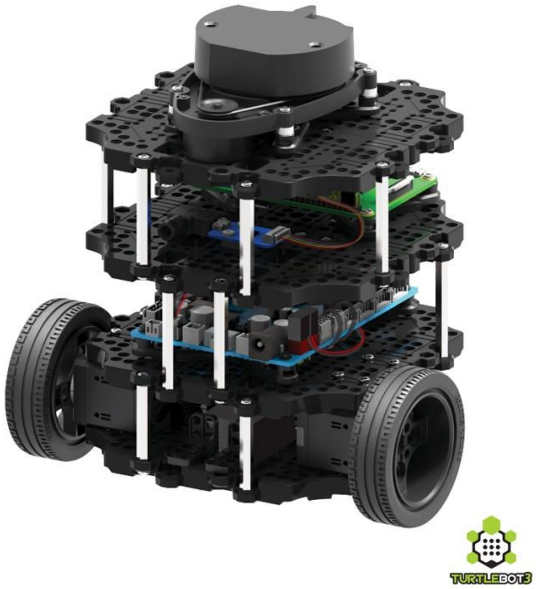
-model geometric;

-model cinematic;

-model dinamic;



Prezentarea Robotului TURTLEBOT 3

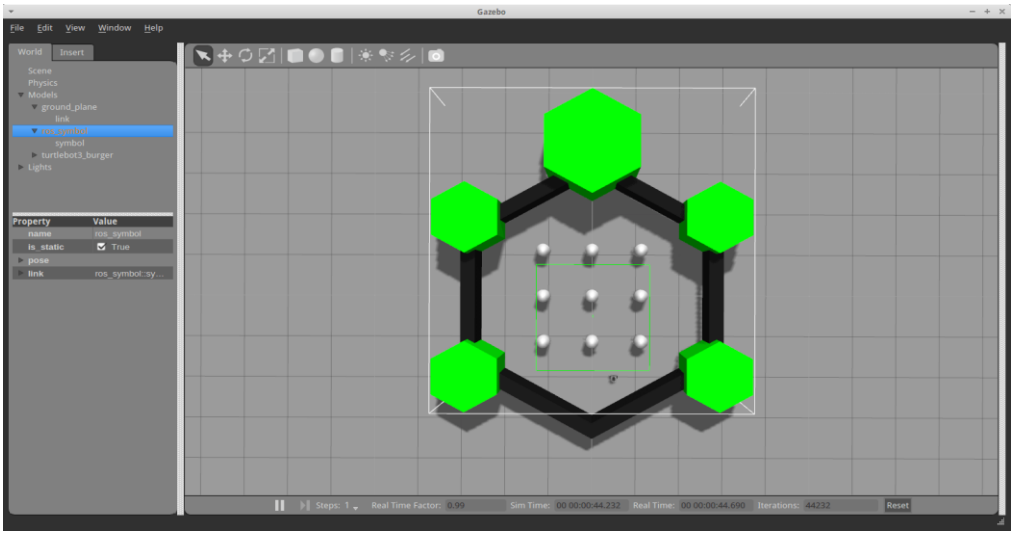


TurtleBot este un robot standard cu platformă ROS. Turtle este derivat din robotul Turtle, care a fost condus de către limbajul educațional de programare Logo în 1967.

TurtleBot3 este un robot mobil de nouă generație care este modular, compact și personalizabil. TurtleBot3 este un robot mobil mic, accesibil, programabil, bazat pe ROS, pentru utilizare în educație, cercetare, hobby și prototipuri de produse. Scopul TurtleBot3 este de a reduce dramatic dimensiunea platformei și de a scădea prețul fără a fi nevoie să sacrifici funcționalitatea și calitatea acesteia, oferind în același timp expansibilitate.

TurtleBot3 poate fi personalizat în diferite moduri, în funcție de modul în care reconstruiți piesele mecanice și utilizați piese opționale, cum ar fi computerul și senzorul. În plus, TurtleBot3 este evoluat cu un SBC rentabil și de dimensiuni mici, potrivit pentru sistem integrat robust, senzor la distanță de 360 de grade și tehnologie de imprimare 3D.

Simulare



PREZENTAREA TEMEI

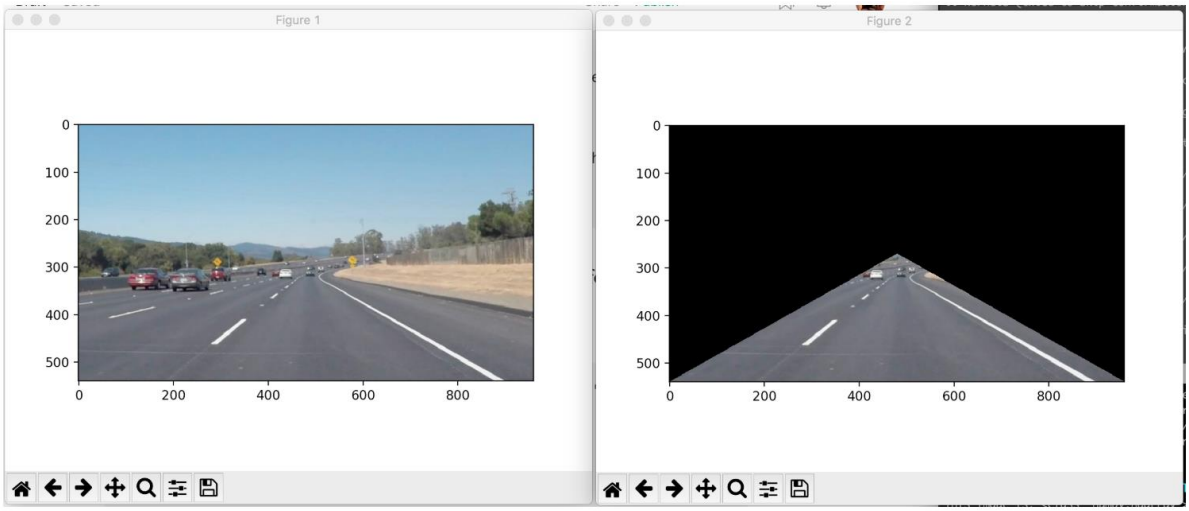
Tema proiectului “Autobot” consta in realizarea unei aplicatii care permite utilizatorului transmiterea unor comenzi catre robot. Autobotul programat de echipa noastra are urmatorul scenariu:

➢ Sa mearga intre doua linii rosi si sa nu depaseasca liniile

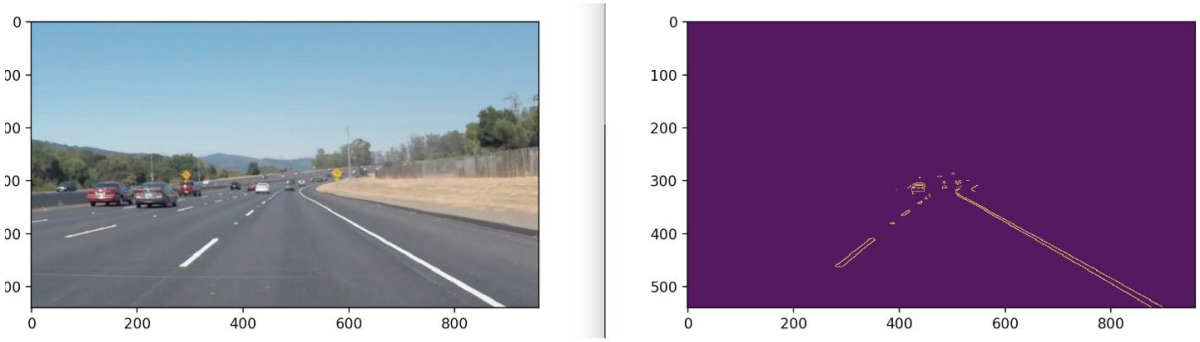
Pentru a merge intre doua linii trebuie sa folosim un algoritm pentru detectarea liniilor. Ca sa nu depaseasca liniile trebuie sa calibram rotile robotului.

Algoritmul pentru detectarea liniilor se realizeaza prin: folosirea tehnicii triunghiului, folosirea a doua filtre de imagini(**Grayscale** si **Canny Edge Detection**) si folosirea conceptului **Hough Transform** (luam toti pixelii noștri de margine si ii tranformam in linii apoi le separam in doua matricii: matricea pentru dreapta si matricea pentru stanga).

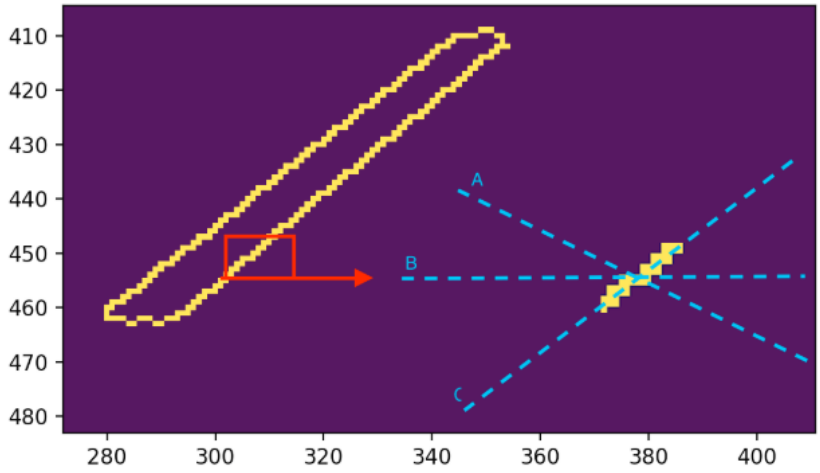
Tehnica triunghiului



Canny Edge Detection



Hough Transform

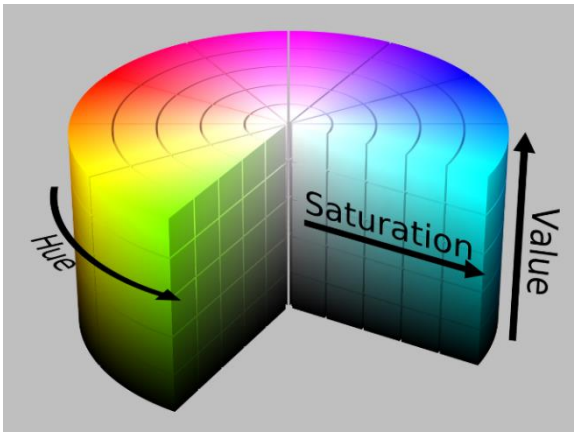


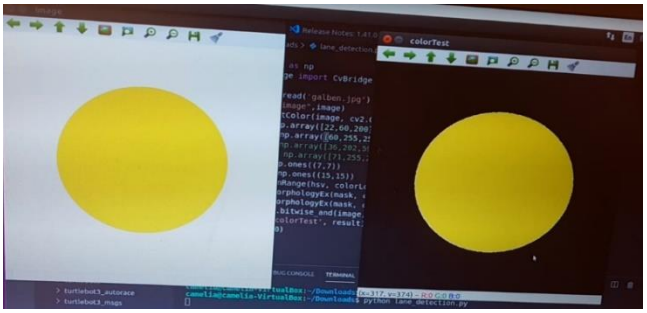
➢ Sa se opreasca la galben si sa porneasca la verde (ca un semafor)

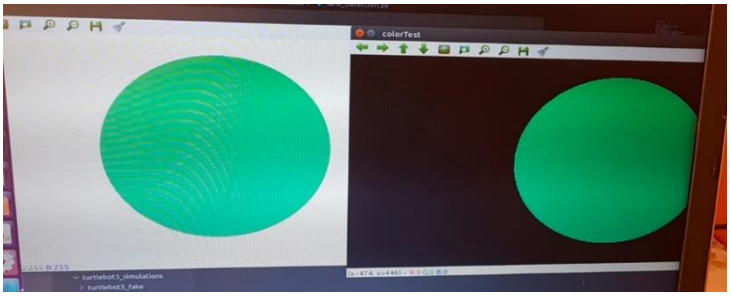
Robotul ca sa execute comanda de oprire sau de pornire trebuie sa recunoasca culoarea potrivita. Acest lucru se face cu detectare de culoare. Detectarea de culoare se realizeaza: cu **HSV** , **matriciile** cu culoarea respectiva si cu **o mask** aplicata imaginii.

HSV= hue, saturation,value

➔ Este o reprezentare alternativa a standardului RGB



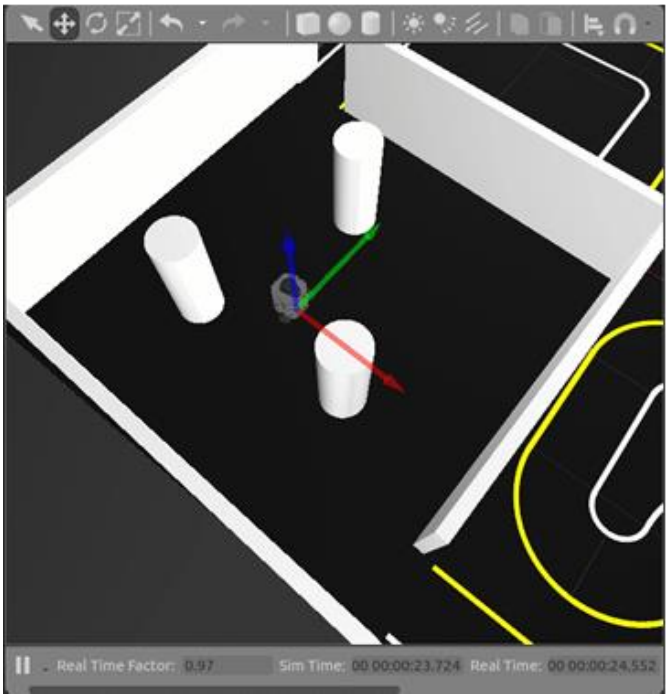




Detectare de culoare

➢ Daca are obstacol in fata sa se opreasca si sa-l ocoleasca

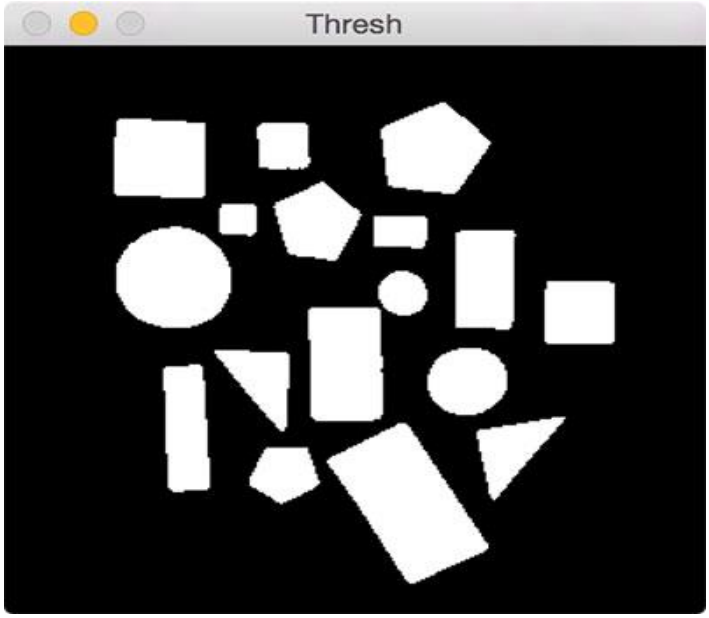
Pentru acest lucru am folosit senzorul de distanta a robotului si am modificat miscarea robotului cu ajutorul lui: **vel_msg.angular.z** si **vel_msg.linear.x**



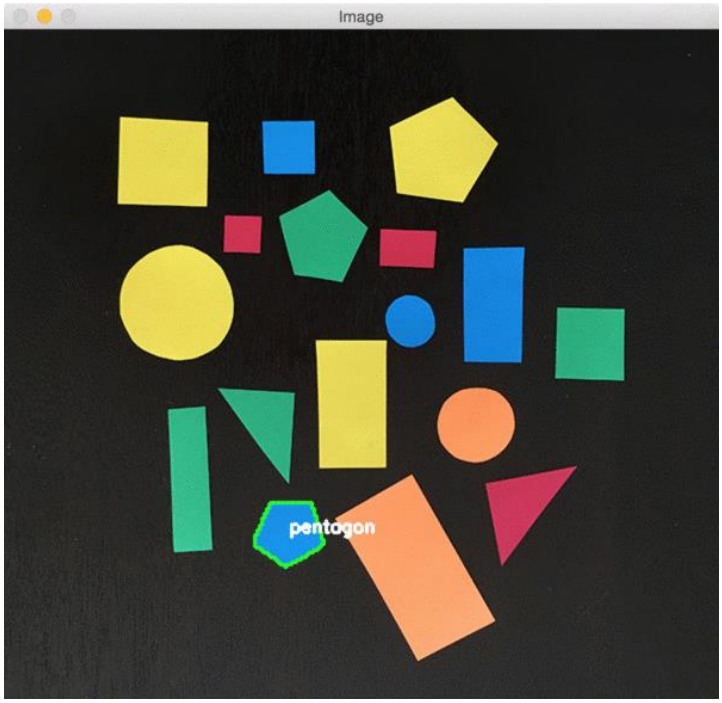
➢ Sa invete recunoasca semnele de circulatie

Pentru a realiza acesta sarcina vom folosi: ori algoritmi pentru a invata robotul sa recunoasca imaginile (algoritm de machine learning) ori algoritmi bazati pe detectarea formei si a culori. Formele tipice pentru semnele de circulatie sunt: cercuri, hexagoane, dreptunghiuri. Algoritmul de detectare a formei consta in:

- Folosirea a trei filtre de imagini (**Grayscale, GaussianBlur, Threshold)**

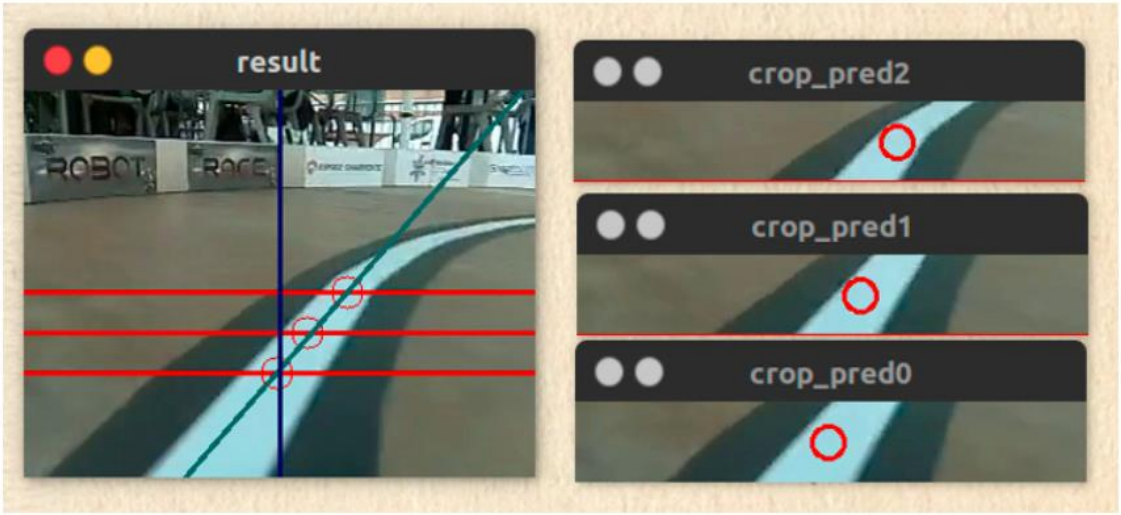


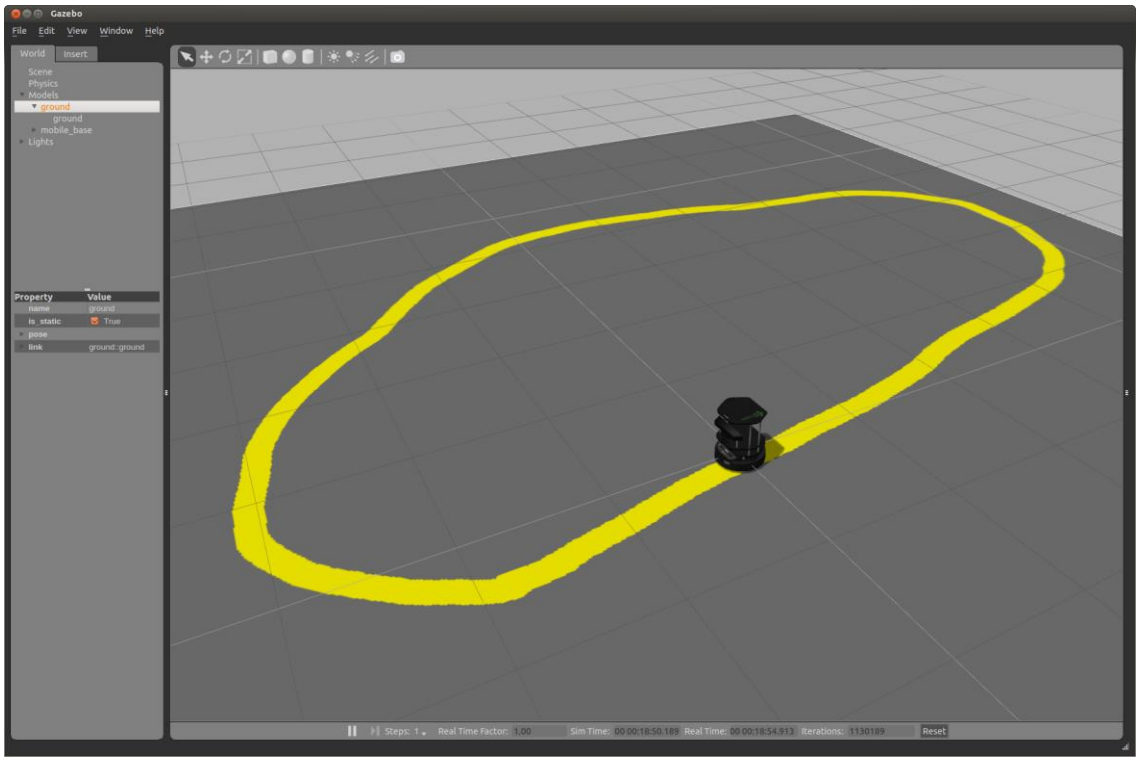
- Folosirea functiei **findContours** de la opencv care gaseste conturul obiectului
- Compararea conturului cu clasificarea de formelor pe care o avem si gasim ce forma



➢ Sa urmareasca doar o linie rosie

Acest lucru se face cu detectia de culoare si apoi se calculeaza mijlocul liniei rosie. Si se traseaza o linie imagina exact pe mijloc ca sa poata urmarii robotul linia fara sa iasa din ea.





TEHNOLOGII UTILIZATE

- Ce este TurtleBot?

TurtleBot este un kit de robot persoanal cu costuri reduse, cu software open-source care utilizeaza platforma standard ROS.

- In ce consta setul TurtleBot?

Setul TurtleBot constă dintr-o bază mobilă, senzor la distanță 2D / 3D, computer laptop sau SBC (computer de bord unic) și kit hardware de montaj TurtleBot. Pe lângă kitul TurtleBot, utilizatorii pot descărca SDK TurtleBot de pe wiki-ul ROS. TurtleBot este conceput pentru a fi ușor de cumpărat, construit și asamblat, folosind produse de larg consum și piese care pot fi create cu ușurință din materiale standard. Ca platformă de robotică mobilă la nivel de intrare, TurtleBot are multe dintre aceleași capabilități ale platformelor de robotică mai mari ale companiei, precum PR2.

- Ce tehnologi utilizam?

Turtle este derivat de la robotul Turtle, care a fost condus de limbajul educational de programare Logo in 1967. TurtleBot este proiectat pentru ca oamenii care sunt noi in ROS sa invete cu usurinta prin intermediul TurtleBot, precum si pentru a preda limbaje de programare folosind Logo. De atunci TurtleBot a devenit platforma standard a ROS, care este cea mai populara platforma printre dezvoltatori si studenti .

Exista 3 versiuni ale seriei TurtleBot. TurtleBot1 a fost dezvoltat de Tully(Platform Manager la Open Robotics) si Melonee(CEO al Fetch Robotics) de la Willow Garage pe robotul de cercetare irobot's Roomba, creat pentru implementarea ROS. Ea a fost dezvoltata in 2010 si a fost de vanzare incepand cu 2011. In 2012, TurtleBot2 a fost dezvoltat de robotul Yujin bazat pe robotul de cercetare, iClebo Kobuki. In 2017, TurtleBot3 a fost dezvoltat cu functii care sa completeze functiile lipsa ale predecesorilor sai si cerintele utilizatorilor. TurtleBot3 adopta Dynamixel de actionare inteligenta ROBOTIS pentru conducere .

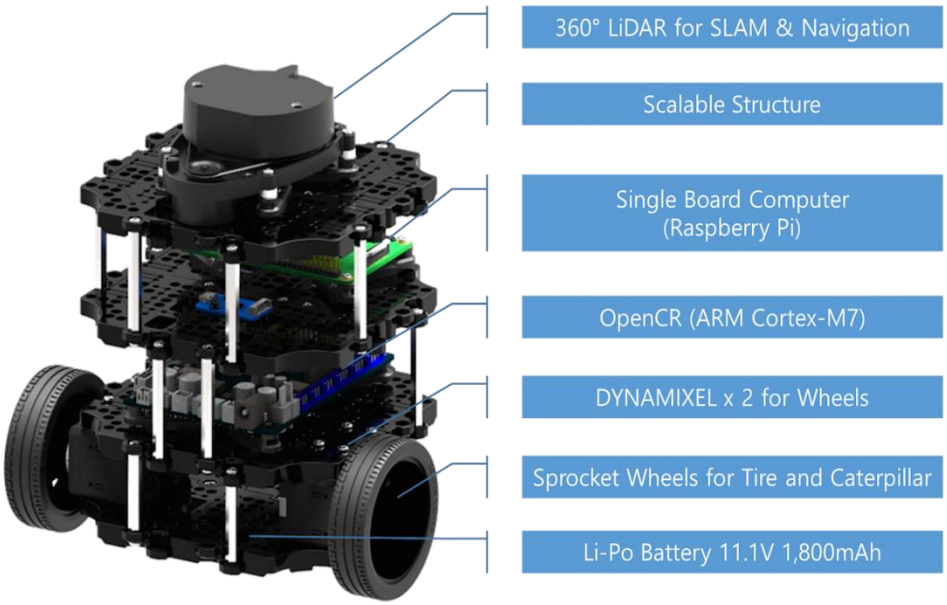
TurtleBot3 este un robot mic, accesibil, programabil, bazat pe ROS, folosit pentru educatie, cercetare, hobby si prototipuri de produse. Scopul programului TurtleBot3 este de a reduce in mod dramatic dimensiunea platformei si de a reduce pretul fara a trebui sa sacrifice functionalitatea si calitatea acestuia, oferind in acelasi timp si o extensibilitate. TurtleBot3 poate fi personalizat in diferite moduri, in functie de modul in care reconstruiti piesele mecanice si folositi componente optionale cu ar fi computerul si senzorul. In plus, TurtleBot3 este dezvoltat cu SBC de cost-eficienta si de marime mica, care este potrivit pentru un sistem robust incorporate, sensor de distanta de 360 de grade si tehnologie de imprimare 3D.

Tehnologia de baza pe care o utilizeaza TurtleBot3 este SLAM, Navigatie si Manipulare, ceea ce il face potrivit pentru robotii de serviciu la domiciliu. TurtleBot poate rula algoritmi SLAM(localizare simultana si cartografiere) pentru a construi o harta si poate conduce in jurul camerei. De asemenea acesta poate fi controlat de la distanta de la un laptop, un joypad sau un telefon intelligent bazat pe android. TurtleBot poate urmari, de asemenea, picioarele unei personae in timp ce se plimba intr-o camera. De asemenea , TurtleBot3 poate fi folosit ca un manipulator mobil capabil sa manipuleze un obiect atasand un manipulator ca OpenManipulator. OpenManipulatorul are avantajul de a fi compatibil cu TurtleBot3 Waffle and Waffle Pi. Prin aceasta compatibilitate poate compensa lipse de libertate si poate avea o completitudine mai mare ca robot de serviciu cu SLAM si capabilitatile de navigare pe care TurtleBot3 le are.

Pentru realizarea proiectului am avut la dispozitie robotul TurtleBot3 Burger:

Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s(162.72 deg.s)
Maximum payload	15kg
Size(L*W*H)	138mm*178mm*192mm
Weight(+SBC+Battery+Sensors)	1kg
Threshold of climbing	10 mm or lower
Expected operating time	2h 30m
Expected charging time	2h 30m
SBC(Single Board Computers)	Raspberry Pi 3 Model B
MCU	32-bit ARM Cortex=M7 with Fpu(216 MHz,462 DMIPS
Remote Controller	-
Actuator	Dynamixel XL430-W250
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS=01
Camera	-
IMU	Gyroscopes 2 Axis Accelerometer 3 Axis Magnetometer 3 Axis
Power connectors	3.3V/800mA 5V/4A 12V/1V
Expansion pins	GPIO 18 pins Arduino 32 pin
Peripheral	UARTx3, CANx1, SPIx1, l2Cx1, ADCx5, 5pin OLLOX4
Dynamixel ports	RS485x 3, TTL x 3
Audio	Several programmable beep sequences
Programmable LEDs	User LED x 4
Status LEDs	Board status LED x 1 Arduino LED x 1 Power LED x 1

Buttons and Switches	Push buttons x 2 , Reset button x 1 , Dip switch x2
Battery	Lithium polymer 11.1V 1800mAh/19.98 Wh 5C
PC connection	USB
Firmware upgrade	Via USB/ via JTAG
Power adapter (SMPS)	Input: 100-240V, AC 50/60Hz, 1.5A Output: 12V DC, 5A



LIMBAJUL DE PROGRAMARE

Pentru implementarea programului condus am folosit limbjul de programare python.

- **Ce este Python de fapt?**

Python este ceva numit un **limbaj de programare**. Preia ceea ce scrii (în mod normal denumit **cod**), îl transformă în instrucțiuni pentru calculatorul tău și le execută

Python este un limbaj de programare dynamic multi-paradigma, creat in 1989 de programatorul olandez Guido Van Rossum este si in ziua de astazi un lider al comunitatii de dezvoltare de software care lucreaza la perfectionarea limbajului Python si implementarea de baza a acestuia, CPython, scrisa in C. Pyhton este un limbaj multifunctional folosit de exemplu de catre companii ca google sau yahoo! pentru programarea

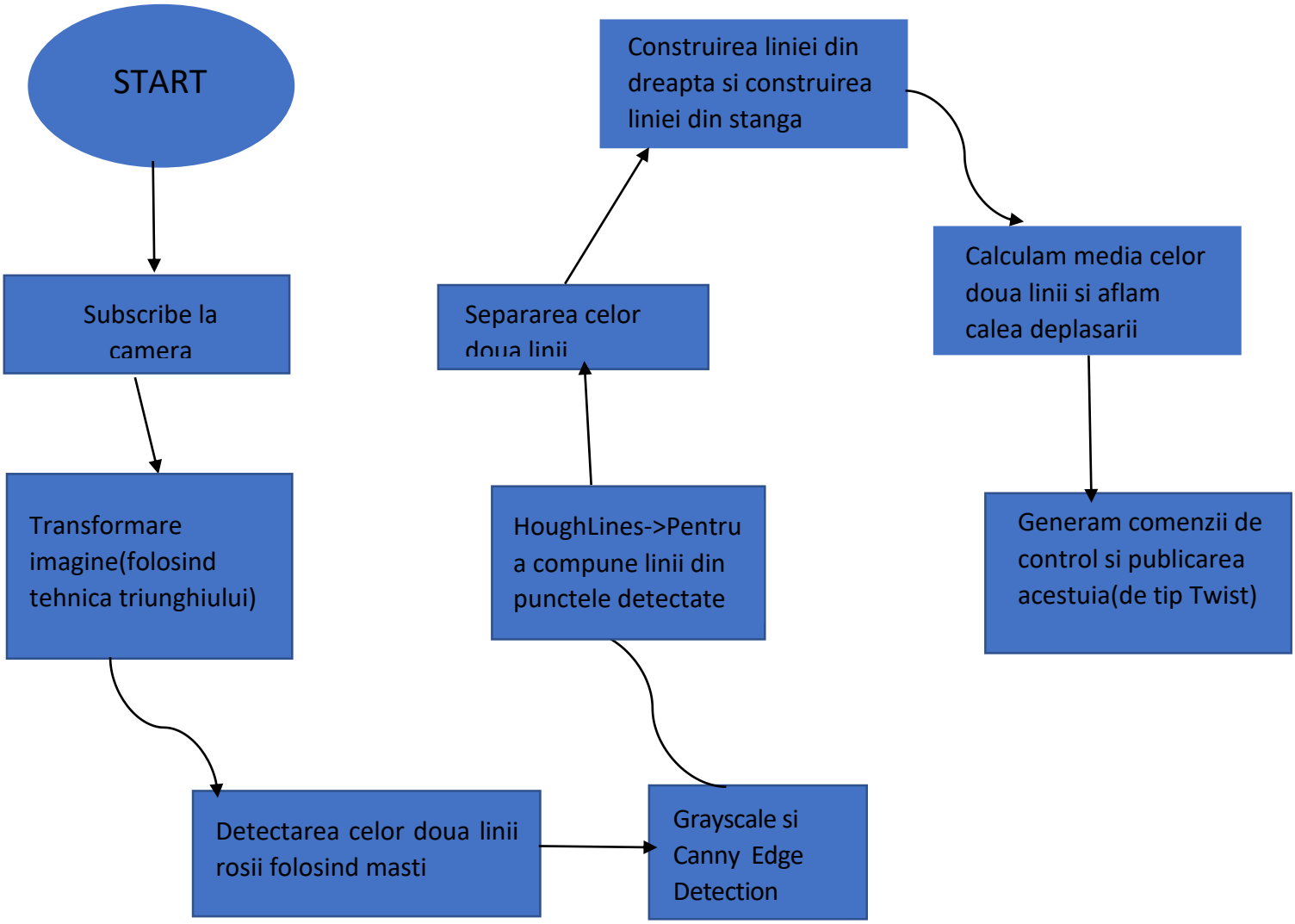
aplicatiilor web, insa exista si o serie de aplicatii stintifice sau de divertisment programate partial sau in intregime in Phyton.

Popularitatea in crestere, dar si puterea limbajului de programare Phyton au dus la adoptarea sa ca limbaj principal de dezvoltare de catre programatori specializati si chiar si la predarea limbajului in unele medii universitare. Din aceleasi motive, multe sisteme bazate pe Unix, inclusive Linux, BSD si Mac OS X include din start interpretatorul CPythton.

Python pune accentual pe curatenia si simplitatea codului,iar sintaxa sa le permita dezvoltatorilor sa exprime unele idei programatice intr-o maniera mai clara si mai concisa decat in alte limbaje de programare ca C.In ceea ce priveste paradigma de programare, Python poate servi ca limbaj pentru software de tipul object-oriented,dar permite si programarea imperativa,functionala sau procedurala.Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu”gunoier”(garbage collector).Alt avantaj al limbajului este existenta unui ample biblioteci standarde metode.

GHIDUL PROGRAMATORULUI

1. Sa mearga intre doua linii rosi si sa nu depaseasca liniile(Varianta 1->nu foloseste si detectie de culoare)




```
import cv2
import numpy as np
import math
import rospy
from sensor_msgs.msg import Image,CompressedImage
from geometry_msgs.msg import Twist
pub=rospy.Publisher('cmd_vel',Twist,queue_size=10)

class Burger:
 def_init_(self):
  #Subscribe la camera
```

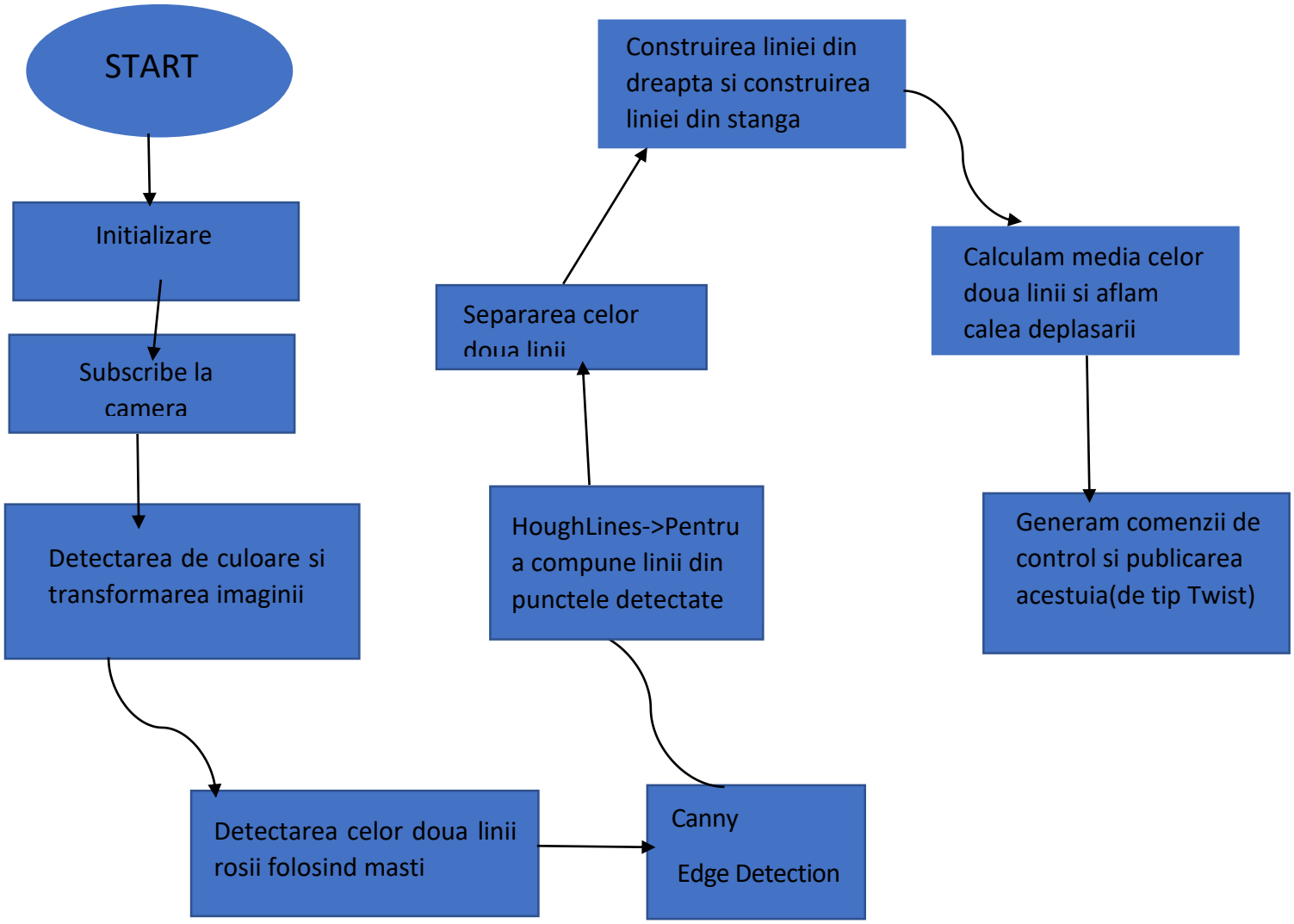
```
self.image=rospy.Subscriber("/raspicam_node/image/compressed",CompressedImage,self.cbFindLa
ne,queue_size=1)
def region_of_interest(img, vertices):
    # Definim o matrice goală care se potrivește cu înălțimea / lățimea imaginii.
    mask = np.zeros_like(img)
    match_mask_color = 255

    # Facem poligonul
    cv2.fillPoly(mask, vertices, match_mask_color)

    # Returnarea imaginii numai acolo unde se potrivesc pixelii de mască
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image

 def draw_lines(img, lines, color=[0, 0, 255], thickness=3):
    # Dacă nu există linii de desenat, ieșim.
    if lines is None:
        return
    #  Facem o copie a imaginii originale.
    img = np.copy(img)
    #Am creat o imagine goală care să se potrivească cu dimensiunea originală.
    line_img = np.zeros(
        (
            img.shape[0],
            img.shape[1],
            3
        ),
        dtype=np.uint8,
    )
    #Deseneaza liniile
    for line in lines:
        for x1, y1, x2, y2 in line:
            cv2.line(line_img, (x1, y1), (x2, y2), color, thickness)
    # Combinam imaginea cu linii cu originala
    img = cv2.addWeighted(img, 0.8, line_img, 1.0, 0.0)
    # Returnam imaginea modificata
    return img


#facem media intre coordonatele acestor doua linii si primim o singura linie care este mijlocul
unde se va deplasa robotelul
 def display_lines(self,image,lines):
   line_image=np.zeros_like(image)
   difference=1000
   if lines is not None:
     for line in lines:
       x1,y1,x2,y2=line.reshape(4)
       cv2.line(line_image,(x1,y1),(x2,y2),(0,0,255),10)
     x_1,y_1,x_2,y_2=lines[0].reshape(4)
```

```
      x_11,y_11,x_22,y_22=lines[1].reshape(4)
      x1=(x_1+x_11)/2
      y1=(y_1+y_11)/2
      x2=(x_2+x_22)/2
      y2=(y_2+y_22)/2
      cv2.line(line_image,(x1,y1),(x2,y2),(0,255,0),10)
      print(x1,y1,x2,y2)
      difference=x1-x2
    return line_image,difference

#Functia de control , care zice in ce directie sa o ia robotelul
 def_control(self,difference):
    x=0
    z=0
    if difference>999:
      x=0
    elif difference<70 and difference>-70:
      x=0.05
    elif difference>=70:
      x=0.01
      z=0.04
    elif difference<=-70:
      x=0.01
      z=-0.04
    return x,z

#Functia principal pentru detectarea liniilor
 def cbFindLane(self,image_msg):
     #Preia imaginea de la camera si o modifica ca sa putem lucre cu ea
    np_img= np.fromstring(image_msg.data,np.uint8)
    image=cv2.imdecode(np_img,cv2.IMREAD_COLOR)
    height = image.shape[0]
    width = image.shape[1]
     #Functia care foloseste tehnica triunghiului
    region_of_interest_vertices = [
     (0, height),
     (width / 2, height / 2),
     (width, height),
   ]
     # Folosim doua masti pentru imagine
    gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    cannyed_image = cv2.Canny(gray_image, 100, 200)

    cv2.imshow("real_image",img)
    cv2.imshow("cannyed_image", cannyed_image)

    cropped_image = region_of_interest(cannyed_image,np.array([region_of_interest_vertices],
np.int32),)
    cv2.imshow("croped_image", cropped_image)
```

```
 #Pentru a compune linii din punctele detectate
 lines = cv2.HoughLinesP(
  cropped_image,
  rho=6,
  theta=np.pi / 60,
  threshold=160,
  lines=np.array([]),
  minLineLength=40,
  maxLineGap=25
 )

   #Deseneaza cele doua linii
 line_image = draw_lines(image, lines)
 cv2.imshow("line_image", line_image)
 left_line_x = []
 left_line_y = []
 right_line_x = []
 right_line_y = []

 for line in lines:
    for x1, y1, x2, y2 in line:
       slope = (y2 - y1) / (x2 - x1) #Calculam panta
       if math.fabs(slope) < 0.5: #Consideram panta extram
          continue
       if slope <= 0: #Daca panta este negative, punem in grupul din stanga
          left_line_x.extend([x1, x2])
          left_line_y.extend([y1, y2])
       else: # Daca nu, punem in grupul din dreapta
          right_line_x.extend([x1, x2])
          right_line_y.extend([y1, y2])

 min_y = image.shape[0] * (3 / 5)#pentru partea de sus a imaginii
 max_y = image.shape[0] #pentru partea de jos a imaginii

#pentru a desena calumea linile, sa nu avem si altele care nu ne trebuie
 poly_left = np.poly1d(np.polyfit(
    left_line_y,
    left_line_x,
    deg=1
 ))
 left_x_start = int(poly_left(max_y))
 left_x_end = int(poly_left(min_y))
 poly_right = np.poly1d(np.polyfit(
    right_line_y,
    right_line_x,
    deg=1
 ))
 right_x_start = int(poly_right(max_y))
```

```
 right_x_end = int(poly_right(min_y))


 line_image = draw_lines(
     image,
     [[
         [left_x_start, int(max_y), left_x_end, int(min_y)],
         [right_x_start, int(max_y), right_x_end, int(min_y)]
     ]],
     thickness=10
 )
 cv2.imshow("line_image", line_image)
   #Desenam linia din mijloc dupa care se orienteaza robotelul
 line_image,difference=self.display_lines(img,lines)
 cv2.imshow("line_image", line_image)
   #Cum sa se miste robotelul ca sa ramana intre linii
 tx,tz=self.control(difference)
 if tx is None:
     tx,tz=0,0
 print(tx,tz)
 twist=Twist()
 twist.linear.x=tx
 twist.linear.y=0
 twist.linear.z=0
 twist.angular.x=0
 twist.angular.y=0
 twist.angular.z=tz
 pub.publish(twist)
 cv2.waitKey(5)

def main(self)
  rospy.spin()

 #aici se creeaza nodul care face actiuniile de mai sus
if __name__=='__main__':
   rospy.init_node('detect',anonymous=True)
   node=ControlBurger()
   node.main()
```

2. Sa mearga intre doua linii rosi si sa nu depaseasca liniile(Varianta 2->se foloseste detectia de culoare)




```
import cv2
import numpy as np
import math
import rospy
from sensor_msgs.msg import Image,CompressedImage
from geometry_msgs.msg import Twist
pub=rospy.Publisher('cmd_vel',Twist,queue_size=10)

class ControlBurger:

   #functia de initializare , in care sunt determinate intervalele pentru culoarea rosie
 def_init_(self):
   self.lowerBound1=np.array([165,80,50])
   self.upperBound1=np.array([180,255,255])
   self.lowerBound2=np.array([0,80,50])
   self.upperBound2=np.array([10,255,255])
```

```
  self.kernelOpen=np.ones((5,5))
  self.kernelClose=np.ones((20,20))
```

#subscribe de la camera

```
self.img=rospy.Subscriber("/raspicam_node/image/compressed",CompressedImage,self.cbFindLane,queue_
size=1)
```

#ajuta la crearea celor doua linii;

```
 def make_coordinates(self,image,l_param):
  slp,inter=l_param
  y=image.shape[0]
  y1=int(y/1.8)
  x=int((y-inter)/slp)
  x1=int((y1-inter)/slp)
  return np.array([x,y,x1,y1])
```

#separam multimea de linii in doua array-uri,cea din stanga si cea din dreapta
#in final cu ajutorul mediilor celor doua array-uri putem compune liniile

```
 def average_slp(self,image,ls):
  fit_l=[]
  fit_r=[]
  for l in ls:
    x,y,x1,y1=l.reshape(4)
    prm=np.polyfit((x,x1),(y,y1),1)
    slp=prm[0]
    inter=prm[1]
    if slp<0.05:
      fit_l.append((slp,inter))
    else:
      fit_r.append((slp,inter))
  fit_l_average=np.average(fit_l,axis=0)
  fit_r_average=np.average(fit_r,axis=0)
  if not(np.isnan(fit_l_average).any() or np.isnan(fit_r_average).any()):
     left_l=self.make_coordinates(image,fit_l_average)
     right_l=self.make_coordinates(image,fit_r_average)
     return np.array([left_l,right_l])
```

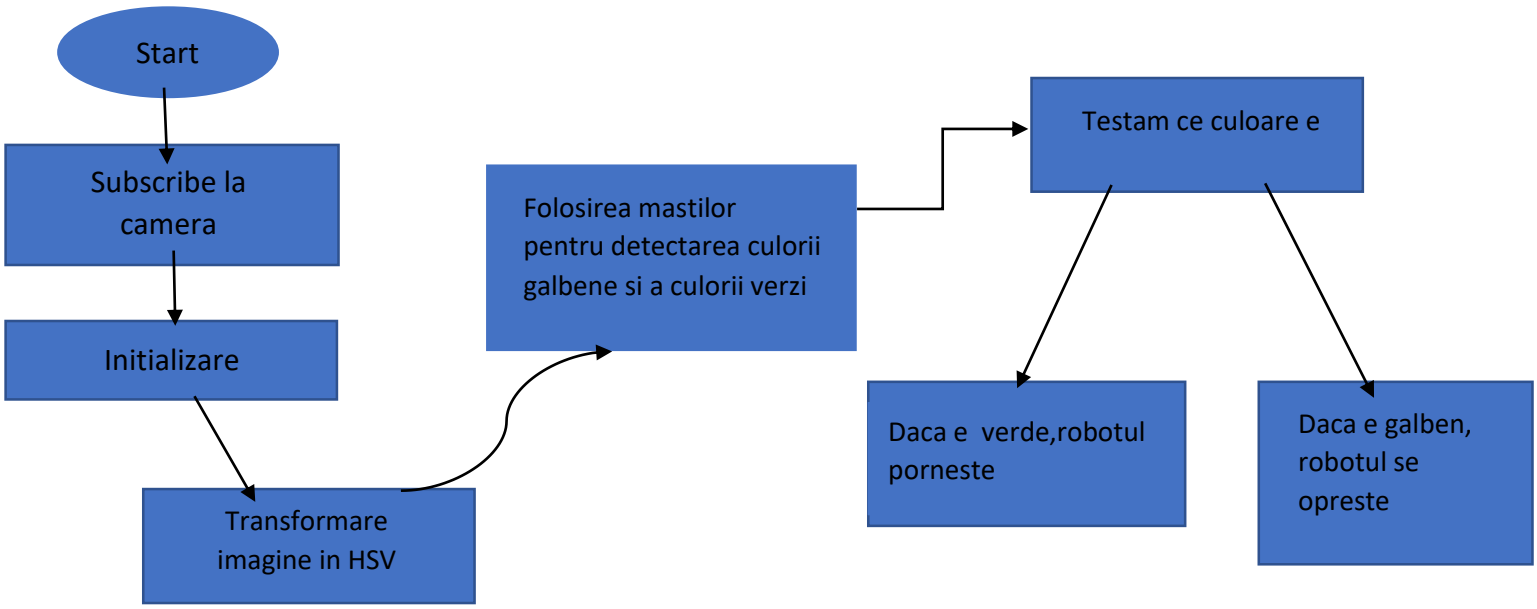
#facand media intre coordonatele acestor doua linii primim o singura linie care este mijlocul unde se va deplasa robotelul

```
 def display_ls(self,image,ls):
  l_image=np.zeros_like(image)
  diffr=1000
  if ls is not None:
     for l in ls:
       x,y,x1,y1=l.reshape(4)
       cv2.l(l_image,(x,y),(x1,y1),(0,0,255),10)
     x_1,y_1,x_2,y_2=ls[0].reshape(4)
     x_11,y_11,x_22,y_22=ls[1].reshape(4)
     x=(x_1+x_11)/2
```

```
    x=(y_1+y_11)/2
    x1=(x_2+x_22)/2
    x=(y_2+y_22)/2
    cv2.l(l_image,(x,y),(x1,y1),(0,255,0),10)
    print(x,y,x1,y1)
    diffr=x-x1
  return l_image,diffr

  #functia de control, care zice in ce directie sa o ia robotelul
 def_ctrl(self,diffr):
  x=0
  z=0
  if diffr>999:
    x=0
  elif diffr<70 and diffr>-70:
    x=0.05
  elif diffr>=70:
    x=0.01
    z=0.04
  elif diffr<=-70:
    x=0.01
    z=-0.04
  return x,z

   #Functia principal care transforma imaginea, pentru a putea lucre cu ea
 def cbFindLane(self,image_msg):
  np_img= np.fromstring(image_msg.data,np.uint8)
  img=cv2.imdecode(np_img,cv2.IMREAD_COLOR)
    #Transformam in hsv ca sa putem face detectia culorii rosii
  HSV=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

   #Folosim masti pentru detectia de culoare
  mk1=cv2.inRange(HSV,self.lowerBound1,self.upperBound1)
  mk2=cv2.inRange(HSV,self.lowerBound2,self.upperBound2)
  mk=mk1|mk2

  open_mk=cv2.morphologyEx(mk,cv2.MORPH_OPEN,self.kernelOpen)
  close_mk=cv2.morphologyEx(open_mk,cv2.MORPH_CLOSE,self.kernelClose)
  mk=close_mk

    #Canny-pentru margini
  edges=cv2.Canny(mk,50,150,apertureSize=3)

#Houghtls pentru a compune linii din punctele detectate
  ls=cv2.HoughlsP(edges,rho=2,theta=1*np.pi/180,threshold=100,minlLength=100,maxlGap=50)

# daca au fost gasite linii, din multime lor se fac 2 linii cea din stanga si cea din dreapta
  if ls in not None:
    avrg_ls=self.average_shope_inter(img,ls)
```

```
    ls=avrg_ls

  l_image,diffr=self.display_ls(img,ls)
  cv2.imshow("l_image",l_image)
  cv2.imshow("real_image",img)
  cv2.imshow("edges",edges)
 # cum sa se miste robotelul ca sa ramana intre linii
  x_t,z_t=self.control(diffr)
  if x_t is None:
     x_t,z_t=0,0
  print(x_t,z_t)
  twist=Twist()
  twist.lar.x=x_t
  twist.lar.y=0
  twist.lar.z=0
  twist.angular.x=0
  twist.angular.y=0
  twist.angular.z=z_t
  pub.publish(twist)
  cv2.waitKey(5)
 def main(self)
  rospy.spin()
#aici se creeaza nodul care face actiuniile de mai sus
 if __name__=='__main__':
    rospy.init_node('detect')
    node=ControlBurger()
    node.main()
```

3. Sa se opreasca la galben si sa porneasca la verde (ca un semafor)

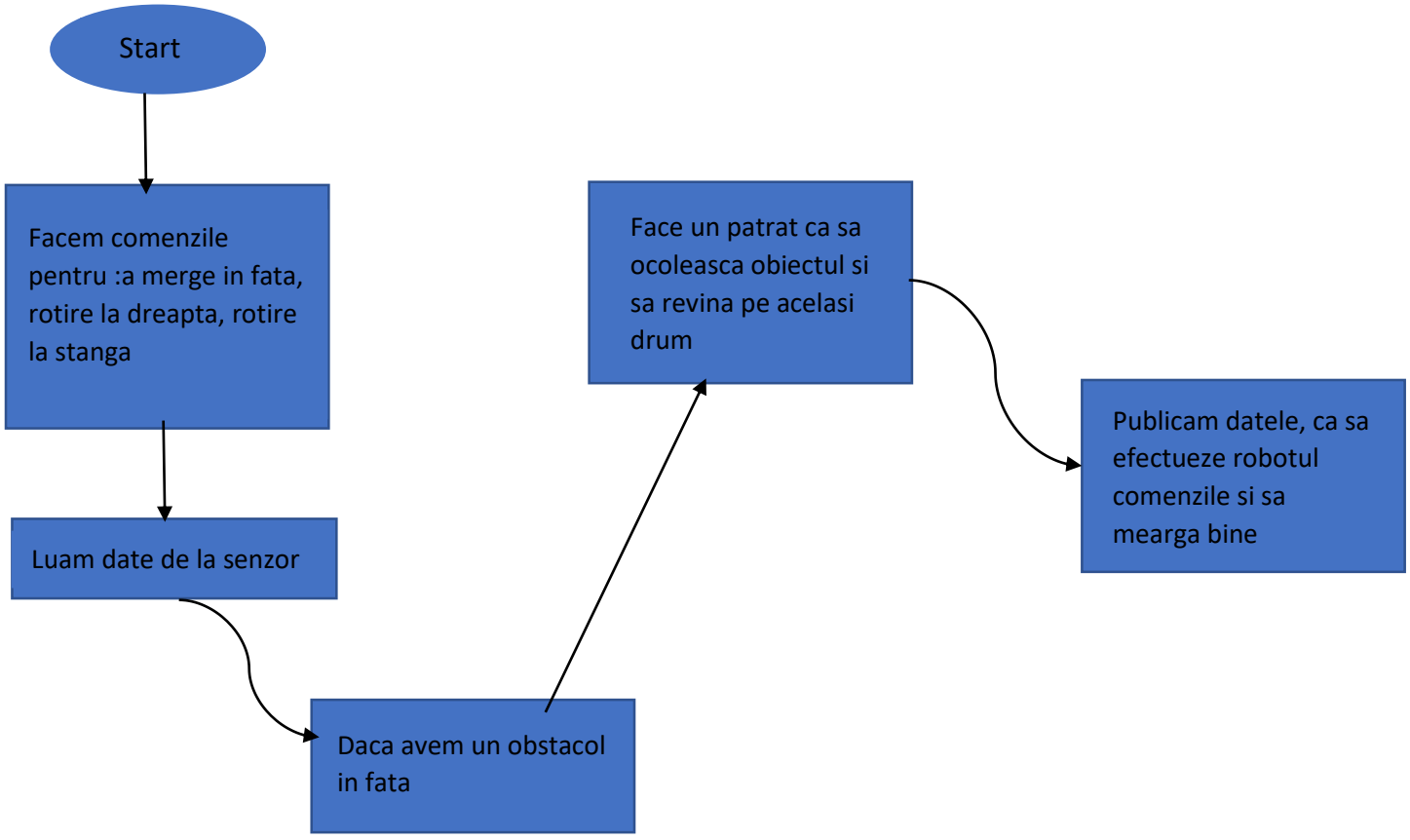



```
import rospy
import numpy as np
import cv2
```

```
from sensor_msgs.msg import CompressedImage
from geometry_msgs.msg import Twist

def callback(ros_data,):
    #Preia datele de la camera si le transforma ca sa putem lucra cu ele
    np_arr = np.fromstring(ros_data.data, np.uint8)
    image=cv2.imdecode(np_arr,1)
    cv2.imshow('cv_img', image)

    #Transformam in hsv ca sa putem face detectia culorii galbene si  a culorii verzi
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Initializare , determinam intervalele pentru culoarea galbena si pentru culoarea verde
    colorLow1 = np.array([22,60,200])
    colorHigh1 = np.array([60,255,255])

    colorLow = np.array([36,202,59])
    colorHigh = np.array([71,255,255])
    kernalOpen=np.ones((7,7))
    kernalClose=np.ones((15,15))
    #galben(masti pentru detectarea culorii)
    mask1 = cv2.inRange(hsv, colorLow1, colorHigh1)
    mask1 = cv2.morphologyEx(mask1, cv2.MORPH_CLOSE,kernalClose)
    mask1 = cv2.morphologyEx(mask1, cv2.MORPH_OPEN, kernalOpen)

    #verde(masti pentru detectarea culorii)
    mask2 = cv2.inRange(hsv, colorLow, colorHigh)
    mask2 = cv2.morphologyEx(mask2, cv2.MORPH_CLOSE,kernalClose)
    mask2 = cv2.morphologyEx(mask2, cv2.MORPH_OPEN, kernalOpen)
    #rezultatele dupa modificarile imaginilor(dupa ce am pus si masca)
    result1 = cv2.bitwise_and(image, image, mask = mask1)
    result2 =cv2.bitwise_and(image,image,mask=mask2)
    #partea pentru miscarea robotelului
    msg = Twist()
    velocity_publisher = rospy.Publisher('cmd_vel', Twist, queue_size=10)
    #daca detecteaza culoarea verde sa porneasca robotul cu viteza de 0.1 pe secunda
    if(np.any(mask1)):
        print("True")
        msg.linear.x =0.1
    else:
        print ("False")
     #daca detecteaza culoarea galbena sa se opreasca robotul
    if(np.any(mask2)):
        print("True")
        msg.linear.x =0
    else:
        print ("False")
    velocity_publisher.publish(msg)
    cv2.imshow('colorTest', result1)
```

```
    cv2.imshow('colorTest1',result2)
    cv2.waitKey(2)

# aici se creeaza nodul care realizeaza actiuniile de mai sus si face subscribe la camera
def main():
    rospy.init_node('image_detect',anonymous=True)
    rospy.Subscriber("/raspicam_node/image/compressed",
        CompressedImage,callback,queue_size=1)
    print "started"
    try:
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down ROS Image feature detector module"
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

4. Daca are obstacol in fata sa se opreasca si sa-l ocoleasca




```
#!/usr/bin/env python
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry
import math
 #Publica comenzi pentru miscarea robotelului
```

```
pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
command = Twist()

#Functia pentru rotire la stanga
def rotate_left():
    target = 90
    #Transformam din radiani in grade
    target_rad = target*math.pi/180
    command.angular.z = 0.3
    r = rospy.Rate(10)
    current = 0
    # Se intoarce pana face 90 de grade
    while current < target_rad:
        pub.publish(command)
        current = current + 0.03
        r.sleep()
    print("target={}", target)
    print("target_rad={}", target_rad)
    #Se intoarce si se opreste
    command.linear.x = 0
    command.angular.z = 0
    pub.publish(command)

# Functia pentru rotire la dreapta
def rotate_right():
    target = 90
    #Transformam din radiani in grade
    target_rad = target*math.pi/180
    command.angular.z = -0.3
    r = rospy.Rate(10)
    current = 0
    # Se intoarce pana face 90 de grade
    while current < target_rad:
        pub.publish(command)
        current = current + 0.03
        r.sleep()
    print("target={}", target)
    print("target_rad={}", target_rad)
    #Se intoarce si se opreste
    command.linear.x = 0
    command.angular.z = 0
    pub.publish(command)

# Functia pentru mersul inainte
def move_forward():
    current = 0
   #Merge in fata cu viteza 0.1 pe secunde
    command.linear.x = 0.1
    r = rospy.Rate(10)
```

```
    #Merge cativa cm si se opreste
    while current < 30:
        pub.publish(command)
        current = current + 1
        r.sleep()
    command.linear.x = 0
    command.angular.z = 0
    pub.publish(command)

# Functia pentru mersul inainte , dar merge inainte mult mai ca functia de mai sus
def move_forward1():
    current = 0
    #Merge in fata cu viteza 0.1 pe secunde
    command.linear.x = 0.1
    r = rospy.Rate(10)
    #Merge cativa cm si se opreste
    while current < 70:
        pub.publish(command)
        current = current + 1
        r.sleep()
    command.linear.x = 0
    command.angular.z = 0
    pub.publish(command)

def callback(data):
   #Primeste date de la senzor
    rospy.loginfo(data.ranges[0])
    r = rospy.Rate(10)
   #Merge cu viteza de 0.1 pe secunda
    command.linear.x =0.1
   #Daca este un obiect aproape de robot
   # Ocoleste obiectul facand un patrat(se roteste la stanga ,merge umpic in fata se roteste din nou iar merge)
    if data.ranges[0] < 0.7:
        rotate_left()
        move_forward()
        rotate_right()
        move_forward1()
        rotate_right()
        command.angular.z = 0

    pub.publish(command)

# Aici se creeaza nodul care face actiuniile de mai sus

def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("scan", LaserScan, callback)
    rospy.spin()
```

```
if __name__ == '__main__':
    listener()
```

5. Sa urmareasca doar o linie rosie

```
import rospy
import numpy as np
import cv2
from cv_bridge import CvBridge
from sensor_msgs.msg import Image, CompressedImage
from geometry_msgs.msg import Twist

class Follower:
 #Functia de initializare in care facem Subscribe la camera si Publicam date pentru comenzile rotilor
  def __init__(self):
    self.bridge = cv_bridge.CvBridge()
    self.image_sub=rospy.Subscriber("/rospicam_node/Image/compressed",compressedImage,
self.image_callback, queue_size=1)
    self.cmd_vel_pub = rospy.Publisher('cmd_vel',Twist, queue_size=1)
    self.twist = Twist()
  def image_callback(self, msg):
    #Preluam imagini de la camera si transformam imagini ca sa putem lucra cu ea
    np_arr=np.fromstring(msg.data,np.uint8)
    image=cv2.imdecode(np_arr,cv2.IMREAD_COLOR)
    #Trasformam in HSV pentru a putea face detectia de culoare
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    #Initializare , pentru determinarea de intervale pentru culoarea rosie
    colorLow=np.array([165,80,50])
    colorHigh=np.array([180,255,255])
    kernalOpen=np.ones((7,7))
    kernalClose=np.ones((15,15))
    # Folosim masti pentru detectia de culoare
    mask1 = cv2.inRange(hsv, colorLow, colorHigh)
    mask = cv2.morphologyEx(mask1, cv2.MORPH_CLOSE,kernalClose)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernalOpen)
    result = cv2.bitwise_and(image, image, mask = mask)
    cv2.imshow('colorTest', result)
    # Se calculeaza mijlocul liniei rosie
    h, w, d = image.shape
    search_top = 3*h/4
    search_bot = 3*h/4 + 20
    mask[0:search_top, 0:w] = 0
    mask[search_bot:h, 0:w] = 0
    M = cv2.moments(mask)
    if M['m00'] > 0:
      cx = int(M['m10']/M['m00'])
      cy = int(M['m01']/M['m00'])
   # Se deseneaza cerculete pe mijloc ca sa stie robotul pe unde sa mearga , el are ca punct de reper
cerculetele
      cv2.circle(image, (cx, cy), 20, (0,0,255), -1)
```

#Se calculeaza eroarea in functie de linia de mijloc si o sa urmareasca linia rosie fara sa iasa de pe ea

```
err = cx - w/2
self.twist.linear.x = 0.2
self.twist.angular.z = -float(err) / 100
self.cmd_vel_pub.publish(self.twist)
```

GHIDUL UTILIZATORULUI

Primul pas pentru utilizarea robotului este conectarea lui cu calculatorul , se porneste un terminal prin care o sa ne conectam la acesta. Conectarea se face folosind comanda **ssh ros@192.168.0.6** . In acest moment ne aflam in robot si dam comanda **nano ~/.bashrc** pentru a alege modul robotului(burger) si pentru a face conexiunea acestuia cu gazda punand **ROS_HOSTNAME,** iar apoi dam comenzile: **source~/.bashrc** si **roslaunch turtlebot3_bringup turtlebot3_robot.launch.** Mai deschidem un terminal pornim nodul master cu **roscore.** Apoi in alt terminal bagam **roslaunch turtlebot3_bringup turtlebot3_remote.launch.** Dupa acesti pasi putem sa rulam codurile scrise in python(codurile de mai sus) pe robot. Conceptele si semnificatile liniilor de cod sunt in capitolul 2 si capitolul 4.

TESTARE SI PUNEREA IN FUNCTIUNE

Testarea robotului a inceput inca de la primul laborator , un prim pas a constat in citirea documentatiei cu ajutorul careia ne-am familiarizat cu liniile de comanda , modul de operare dar si cu clasa din care face parte robotul si a limbajului folosit pentru a-l controla si programa.

Dupa ce ne-am familiarizat si inteles rolul fiecarei functii , urmatorul pas a fost testarea robotului in simulator iar dupa aceasta etapa l-am testat practic. O prima incercare practica de testare a fost un mic esec din cauza versiunii de python, iar dupa instalarea python 3 am reusit testarea practica .

Prima testare a fost cea a componentelor robotului, cum ar fii camera, senzorul de distanta, aceasta testare s-a dovedit a fi un success, urmatorul pas a fost scrierea codului pentru functiile de detectie a liniilor, detectarea culorilor verde si rosu , ocolierea obstacolelor.

O prima testare importanta a fost functia de detectie de linii, insa aceasta functie nu am putut-o testa practic din cauza versiunii de python mai veche, insa am reusit testarea acesteia pe simulator si a functionat.

Testarea detectiei de culoare a fost un success atat practic cat si pe simulator, insa am avut mult de munca si cu acest cod deoarece nu citea bine culorile , dar am reusit sa

functioneze. Tot aici am aflat ce se intampla cand ti se descarca bateria la robot si trebuie bagat la incarcat , si anume acesta se opreste brusc si nu mai ruleaza comenzile primite de la noi, chiar daca acesta e pus la incarcat.

Ocolierea unui obstacol a fost partea cea mai grea de realizat deoarece nu intelegeam anumite lucruri cum ar trebui scrisa functia astfel incat acesta sa ocoleasca un obiect, primele testari au avut loc in simulator iar cand am fost siguri ca functioneaza codul si pe robotel l-am testat punandu-l sa ocoleasca o sticla, dar surpriza s-a terminat bateria robotului si nu am mai putut testa.

PREZENTAREA ECHIPEI

Echipa noastra e formata din urmatorii studenti:

- BURLACU CAMELIA-ADELINA
- MESTER ARANKA-NIKOLETTA
- GUBICS MARCUS
- GHEROGHIȚĂ-ADRIAN
- MOISĂ GABRIEL

BURLACU CAMELIA-ADELINA

- ✓ Am fost coordonatorul echipei
- ✓ Am facut codul pentru detectia de culoare
- ✓ Am ajutat la codul de detective a liniilor
- ✓ Am ajutat la documentatie(prezentarea temei,ghidul programatorului,ghidul utilizatorului)
- ✓ Am contribuit la testarea codului

MESTER ARANKA-NIKOLETTA

- ✓ Am facut codul pentru ocolirea obiectului
- ✓ Am ajutat la realizarea PowerPoint
- ✓ Am ajutat la documentatie(introducerea)
- ✓ Am propus idei de realizare a temei de proiect

GUBICS MARCUS

- ✓ Am ajutat la documentatie(concluzie,introducere)
- ✓ Am contribuit la testarea codului
- ✓ Am ajutat la codul de detective a liniilor
- ✓ Am propus idei de dezvoltare a proiectului

GHEROGHIȚĂ-ADRIAN

- ✓ Am ajutat la codul de a merge robotul pe o linie
- ✓ Am ajutat la documentatie(testare, tehnologii utilizate)
- ✓ Am contribuit la testarea codului
- ✓ Am propus idei de realizare a temei de proiect

MOISĂ GABRIEL

- ✓ Am ajutat la realizarea PowerPoint

- ✓ Am ajutat la documentatie(limbajul de programare)
- ✓ Am contribuit la testarea codului
- ✓ Am propus idei de realizare a temei de proiect

CONCLUZII

In urma parcurgerii orelor de laborator, avand ca tema “Conducerea la distanta a robotului mobil TurtleBot3 Burger”, pentru realizarea completa a proiectului am putut observa in functie de dificultatea cerintelor propuse si a scenariului cateva dintre punctele tari si punctele slabe ale aplicatiei.

Rezolvarea problemelor a necesitat multa munca din partea intregii echipe pentru aducerea la forma finala a algoritmului de functionare al robotului.

In urma proiectului am dobandit cunostinte tehnice precum dezvoltarea unei aplicatii prin limbajiul de programare Python, demonstrand fiecare abilitati de lucru in echipa si de rezolvare rapida a problemelor.

BIBLIOGRAFIE

- https://www.cs.cmu.edu/~mihaib/articole/roboti/roboti-html.html
- https://ro.wikipedia.org/wiki/Robot
- http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
- https://medium.com/@mrhwick/simple-lane-detection-with-opencv-bfeb6ae54ec0
- https://www.hackster.io/WolfxPac/colour-detection-using-opencv-python-8cbbe0
- https://en.wikipedia.org/wiki/HSL_and_HSV
- https://www.pyimagesearch.com/2016/02/08/opencv-shape-detection/
- https://becominghuman.ai/autonomous-racing-robot-with-an-arduino-a-raspberry-pi-and-a-pi-camera-3e72819e1e63
- https://www.turtlebot.com/
- http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/#turtlebot3
- https://www.link-academy.com/cursul-python-limbaj-de-programare
- https://ro.wikipedia.org/wiki/Python
- https://opentechschool.github.io/python-beginners/ro/getting_started.html#what-is-python-exactly