UNIVERSITATEA "POLITEHNICA" DIN TIMISOARA FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE DEPARTAMENTUL AUTOMATICĂ ŞI INFORMATICĂ

HIDE&SEEK

PROIECT SINCRETIC 1

AUTORI: BOTA IULIA-ALEXIS
CAUC ANA-MARIA ANDREEA

Coordonator: ing. Emil VOIŞAN

Anul III AIA – an universitar 2023/2024

CUPRINS

1.Introducere

Roboții sunt mașini concepute pentru a înlocui sau mima acțiunile umane, cu scopul de a ne face viața mai ușoară și mai bună. Construcția lor se bazează pe diferite cunoștințe de electronică, mecanică, programare, nanotehnologie și bioinginerie.

Roboți sunt construiți folosind trei tehnologii mari:

- Senzorii permit deplasarea și percepția detaliată a mediului înconjurător pentru a evita obstacolele (element critic în construcția mașinilor autonome, de exemplu).
- Actuatorii (motoare electrice folosite în sistemele automate pentru executarea comenzilor) stabilesc forța și puterea robotului și cât de natural și discret se mișcă.
- **I**nteligența artificială stabilește cât de inteligenți sunt.

Un robot mobil comandat de la distanță este un robot care poate fi controlat prin diverse medii de comunicare, accesând zone greu accesibile operatorului uman.

Robotica mobilă aduce o nouă dimensiune în tehnologie, permitând roboților să se deplaseze autonom și să execute sarcini diverse în medii variate. Cu roți, șenile sau picioare și echipați cu senzori avansați, aceștia se integrează în industrii precum producție, medicină și logistică.

Conducerea la distanta adauga o dimensiune de control,si permite operatorilor să îndrume roboții de la distanță. Tehnologii precum comunicația wireless și realitatea virtuală (VR) devin instrumente esentiale ,aducand beneficii semnificative in ceea ce priveste accesibilitatea, eficienta si securitatea operatiunilor

Se numeşte proces de conducere un proces dinamic de organizare şi coordonare de către cineva (ceva), într-o anumită perioadă de timp, a altor grupuri de membri ai sistemului, în scopul realizării unor sarcini sau scopuri specifice.

Beneficiile implementării roboților mobili și a conducării la distanță sunt evidente în creșterea eficienței operaționale, accesibilitatea în medii periculoase și capacitatea de a gestiona sarcini complexe. Cu toate acestea, nu putem ignora provocările, cum ar fi riscul de coliziuni sau gestionarea optimă a întârzierilor în transmiterea datelor.

2. Prezentarea temei

Tema echipei noastre ne propune programarea unui Turtlebot 3 Burger in asa fel incat acesta sa se poata juca Hide&Seek.Locul de joaca al robotului este o harta realizata in mediul de simulare Gazebo.

Obiectivul jocului este ca robotul mobil să găsească și să identifice locația ascunsă a unui obiect sau a unei alte entități într-un spațiu determinat de noi dar si de a se ascunde,gasind coordonatele unui punct mai putin vizibil.

Folosind datele de la camera robotului, implementăm un algoritm de procesare a imaginilor pentru a identifica zonele mai puțin vizibile.Turtlebot alege să se ascundă în aceste zone, evitând locurile evidente sau ușor accesibile.El reuseste sa se ascunda in momentul in care detecteaza coordonatele unui punct care nu este asa vizibil, si se deplaseaza in acel punct .

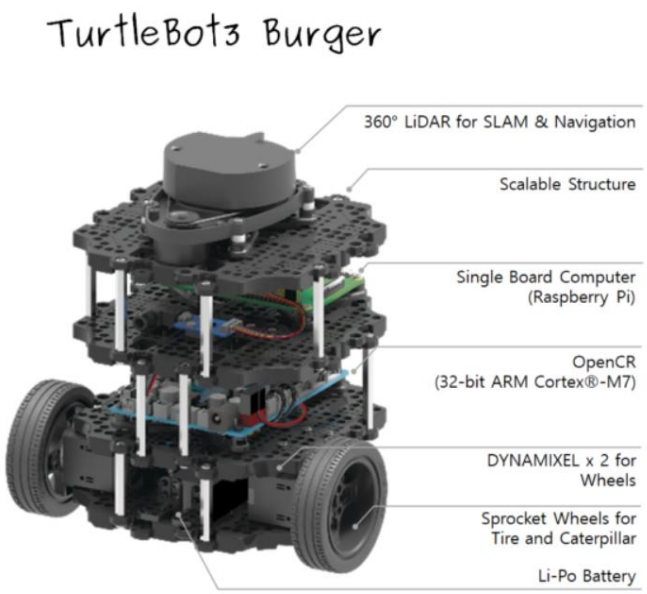
Pentru faza de căutare în jocul Hide&Seek, robotul utilizează un algoritm de procesare a imaginilor pentru a identifica un obiect de culoare roșie, plasat strategic în labirint. Datele provenite de la camera robotului sunt analizate pentru detectarea acestei culori specifice. Odată identificat obiectul roșu, Turtlebot calculează coordonatele acestuia și își planifică traseul pentru a se deplasa eficient către locația obiectului.

Cautarea se face cu un algoritm de procesare a imaginilor care identifica un obiect de culoare roșie, pe care l-am plasat strategic în labirint. Turtlebot utilizează informațiile de la camera sa pentru a localiza și se deplasează către obiectul roșu.

3. Tehnologii utilizate

- Prezentarea Robotului:

  Robotul utilizat de noi este un TURTLEBOT 3 Burger



Mediile de programare folosite:

- Mediul ROS(Robot Operating System)
- Turtlebot3
- Gazebo (instalat odată cu ROS) - pentru simularea robotului
- Rviz - folosit pentru crearea hărților, utilizând SLAM (Simultaneous Localization and Mapping)
- Visual Studio Code - utilizat pentru crearea codului

Specificatii

2. 1. 1. Hardware Specifications

Items	Burger	Waffle Pi
Maximum translational velocity	0.22 m/s	0.26 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)	1.82 rad/s (104.27 deg/s)
Maximum payload	15kg	30kg
Size (L x W x H)	138mm x 178mm x 192mm	281mm x 306mm x 141mm
Weight (+ SBC + Battery + Sensors)	1kg	1.8kg
Threshold of climbing	10 mm or lower	10 mm or lower
Expected operating time	2h 30m	2h
Expected charging time	2h 30m	2h 30m
SBC (Single Board Computers)	Raspberry Pi	Raspberry Pi
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Remote Controller	-	RC-100B + BT-410 Set (Bluetooth 4, BLE)
Actuator	XL430-W250	XM430-W210
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS-01 or LDS-02	360 Laser Distance Sensor LDS-01 or LDS-02
Camera	-	Raspberry Pi Camera Module v2.1
IMU	Gyroscope 3 Axis Accelerometer 3 Axis	Gyroscope 3 Axis Accelerometer 3 Axis
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A	3.3V / 800mA 5V / 4A 12V / 1A
Expansion pins	GPIO 18 pins Arduino 32 pin	GPIO 18 pins Arduino 32 pin
Peripheral	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
DYNAMIXEL ports	RS485 x 3, TTL x 3	RS485 x 3, TTL x 3
Audio	Several programmable beep sequences	Several programmable beep sequences
Programmable LEDs	User LED x 4	User LED x 4
Status LEDs	Board status LED x 1 Arduino LED x 1 Power LED x 1	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 x 2	Push buttons x 2, Reset button x 1, Dip switch x 2
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
PC connection	USB	USB
Firmware upgrade	via USB / via JTAG	via USB / via JTAG
Power adapter (SMPS)	Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A	Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A

	Part Name	Burger
Chassis Parts	Waffle Plate	8
.	Plate Support M3x35mm	4
.	Plate Support M3x45mm	10
.	PCB Support	12
.	Wheel	2
.	Tire	2
.	Ball Caster	1
.	Camera Bracket	0
Motors	DYNAMIXEL (XL430-W250-T)	2
.	DYNAMIXEL (XM430-W210-T)	0
Boards	OpenCR1.0	1
.	*Raspberry Pi	1
.	USB2LDS	1
Remote Controllers	BT-410 Set (Bluetooth 4, BLE)	0
.	RC-100B (Remote Controller)	0
Sensors	**LDS-01 or LDS-02	1
.	Raspberry Pi Camera v2.1	0
Memorys	MicroSD Card	1
Cables	Raspberry Pi Power Cable	1
.	Li-Po Battery Extension Cable	1
.	DYNAMIXEL to OpenCR Cable	2

Powers	SMPS 12V5A	1
.	A/C Cord	1
.	LIPO Battery 11.1V 1,800mAh	1
.	LIPO Battery Charger	1
Tools	Screw driver	1
.	Rivet tool	1
Miscellaneous	PH_M2x4mm_K	8
.	PH_T2x6mm_K	4
.	PH_M2x12mm_K	0
.	PH_M2.5x8mm_K	16
.	PH_M2.5x12mm_K	0
.	PH_T2.6x12mm_K	16
.	PH_M2.5x16mm_K	4
.	PH_M3x8mm_K	44
.	NUT_M2	0
.	NUT_M2.5	20
.	NUT_M3	16
.	Rivet_1	14
.	Rivet_2	2
.	Spacer	4
.	Silicone Spacer	0
.	Bracket	5
.	Adapter Plate	1

ROS (Sistemul de Operare pentru Roboți) furnizează biblioteci și instrumente pentru a ajuta dezvoltatorii de software să creeze aplicații pentru roboți. Acesta oferă abstractizare hardware, drivere pentru dispozitive, biblioteci, vizualizatoare, schimbul de mesaje, gestionarea pachetelor și multe altele. ROS are licență open source, sub licența BSD.

TurtleBot este un robot platformă standard pentru ROS (Sistemul de Operare pentru Roboți). Numele "Turtle" este derivat de la robotul Turtle, care a fost utilizat în limbajul de programare educațional Logo în anul 1967. De asemenea, nodul turtlesim, prezent în primul tutorial de bază al ROS, reprezintă un program care imită sistemul de comandă al programului de turtle din limbajul Logo.

TurtleBot3 este un robot mobil de dimensiuni reduse, accesibil, programabil și bazat pe ROS (Sistemul de Operare pentru Roboți), destinat utilizării în educație, cercetare, hobby și prototipare de produse. Scopul TurtleBot3 este de a reduce semnificativ dimensiunile platformei și de a scădea prețul fără a sacrifica funcționalitatea și calitatea sa, oferind totodată posibilitatea de extindere.

Tehnologia de bază a TurtleBot3 este reprezentată de SLAM (Simultaneous Localization and Mapping), Navigație și Manipulare, făcându-l potrivit pentru roboți de serviciu la domiciliu.

TurtleBot poate executa algoritmi SLAM pentru a construi o hartă și poate circula prin încăperea dumneavoastră.

4. Ghidul programatorului

Programul pentru hide

```
#!/usr/bin/env python3

import rospy
import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
import cv2
import numpy as np


#funcția de găsire a coordonatelor unui punct mai puțin vizibil
def find_first_white_pixel(image_path, p1, p2):
#se citește harta salvată in urma utilizarii nodului line_of_sight si se extrag dimensiunile imaginii
    image = cv2.imread(image_path)
    height, width, channels = image.shape

#se caută punctul mai puțin vizibil
    for p in np.linspace(p1, p2, np.linalg.norm(p1 - p2)):
    pos = tuple(np.int32(p))
    if (image[pos[0], pos[1], 0] == 254 and
    image[pos[0], pos[1], 1] == 254 and
    image[pos[0], pos[1], 2] == 254):
    return pos

return None

#funcția de deplasare a robotului în punctul dorit
def movebase_client(x, y, orientation_z, orientation_w):
    client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
    client.wait_for_server()
```

```
    goal = MoveBaseGoal()
    goal.target_pose.header.frame_id = "map"
    goal.target_pose.header.stamp = rospy.Time.now()
    goal.target_pose.pose.position.x = x
    goal.target_pose.pose.position.y = y
    goal.target_pose.pose.orientation.z = orientation_z
    goal.target_pose.pose.orientation.w = orientation_w

    client.send_goal(goal)
    wait = client.wait_for_result()
    if not wait:
    rospy.logerr("Action server not available!")
    rospy.signal_shutdown("Action server not available!")
    else:
    return client.get_result()
```

#transformarea coordonatelor din pixeli în coordonate ROS

```
def convert_to_ros_coordinates(pixel_coordinates, image_width, image_height,
    ros_environment_width, ros_environment_height):
```

 # calcularea factorilor(valori?) de scalare

```
    x_scale = ros_environment_width / image_width
    y_scale = ros_environment_height / image_height
```

#calcularea valorilor de offset, ținând cont că originea hărții în pixeli diferă de originea hărții din Gazebo

```
    x_offset = -ros_environment_width / 2.0
    y_offset = -ros_environment_height / 2.0
```

 # Transformarea coordonatelor din pixeli în coordonate ROS

```
    ros_x = pixel_coordinates[1] * x_scale + x_offset
    ros_y = -pixel_coordinates[0] * y_scale + y_offset

    return ros_x, ros_y
```

#inițializarea nodului și valorilor necesare pentru locația imaginii și coordonatele inițiale ale robotului

```
rospy.init_node('goal_node')
image_path = 'Downloads/docs-main/docs-main/line_of_sight/result.png'
pt_a = np.array([0, 0])
pt_b = np.array([220, 210])
```

#găsirea primului punct mai puțin vizibil

```
first_white_pixel = find_first_white_pixel(image_path, pt_a, pt_b)
```

#dacă am găsit punctul, se vor transforma coordonatele din pixeli în coordonate ROS și se va deplasa robotul în acea locație

```
if first_white_pixel is not None:
  #se inițializează dimensiunile mediului ros
        ros_environment_width= (192-first_white_pixel[1])*0.05
        ros_environment_height=(192-first_white_pixel[0])*0.05
        #se transformă coordonatele
        ros_coordinates = convert_to_ros_coordinates(first_white_pixel, 384, 384,
    ros_environment_width, ros_environment_height)
        print("ROS Coordinates of the first white pixel:", ros_coordinates)

        # se deplasează robotul în direcția dorită
        result=movebase_client(ros_coordinates[0], ros_coordinates[1], 0.0, 1.0)
else:
        print("No white pixel found in the specified region.")
if result:
rospy.loginfo("Goal execution done!"
```

Programul pentru procesarea de imagini

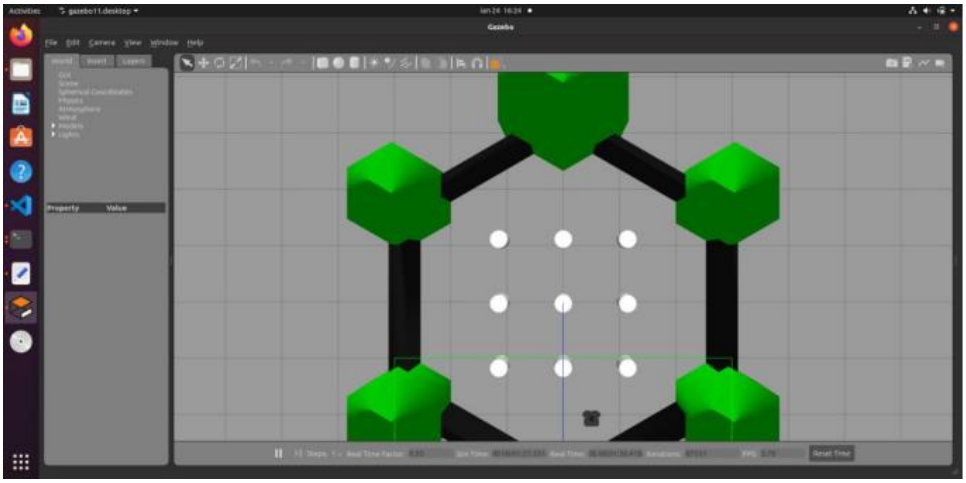
```
#!/usr/bin/env python3
import numpy as np
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import cv2
bridge_object = CvBridge() # create the cv_bridge object
image_received = 0 #Flag to indicate that we have already received an image
cv_image = 0 #This is just to create the global variable cv_image

def show_image():
```

```
        image_sub = rospy.Subscriber("/camera/rgb/image_raw",Image,camera_callback)
        r = rospy.Rate(10) #10Hz
        while not rospy.is_shutdown():
        if image_received:
        cv2.waitKey(1)
        r.sleep()
        cv2.destroyAllWindows()

def process_image(image):
        image = cv2.resize(image,(300,300))
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

        min_red = np.array([0,70,50])
        max_red = np.array([170,255,255])
        mask_r = cv2.inRange(hsv, min_red, max_red)

        res_r = cv2.bitwise_and(image,image, mask= mask_r)
        cv2.imshow('Red',res_r)
        cv2.imshow('Original',image)
        cv2.waitKey(1)
def camera_callback(data):
        global bridge_object
        global cv_image
        global image_received
        image_received=1
        try:
        print("received ROS image, I will convert it to opencv")
        # We select bgr8 because its the OpenCV encoding by default
        cv_image = bridge_object.imgmsg_to_cv2(data, desired_encoding="bgr8")
        #Add your code to save the image here:
        #Save the image "img" in the current path
        cv2.imwrite('robot_image.jpg', cv_image)
        ## Calling the processing function
        process_image(cv_image)
        cv2.imshow('Image from robot camera', cv_image)

        except CvBridgeError as e:
        print(e)

if __name__ == '__main__':
```

```
rospy.init_node('load_image', anonymous=True)
show_image()
```

5. Ghidul utilizatorului

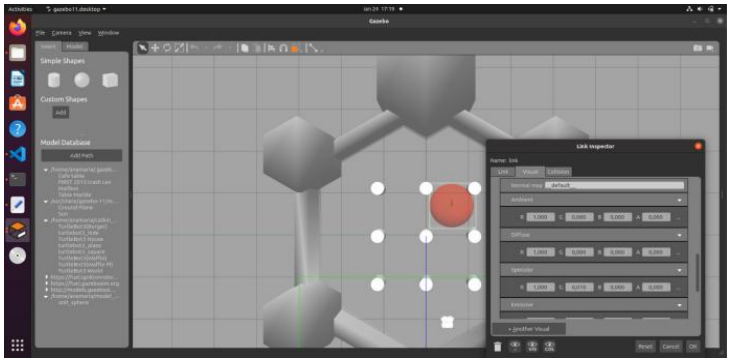
- Se va deschide mediul de simulare Gazebo, introducând în terminal comanda roslaunch turtlebot3_gazebo turtlebot3_world.launch

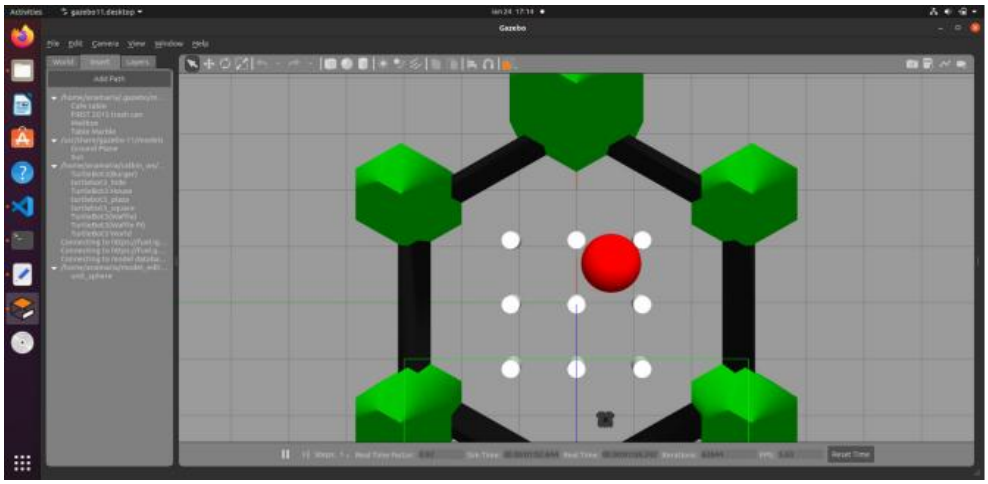


Pentru detecția de culoare:

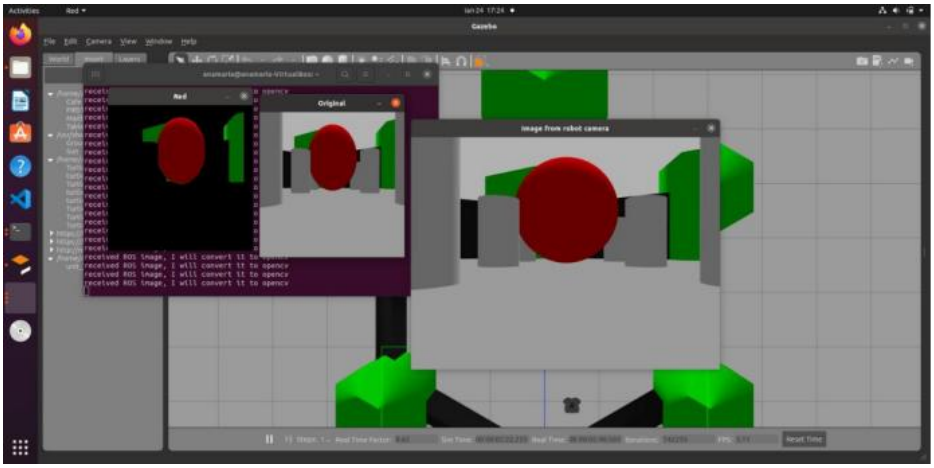
Se va insera un obiect de culoare roșie în orice loc dorit. Se poate utiliza și un obiect prestabilit din mediul de simulare, schimbându-i culoarea prin Edit Model, urmând click dreapta pe model și se deschide apoi Link Inspector și se modifică în Visual componentele Ambient, Diffuse și Specular la culoarea de roșu(R), restul fiind lăsate pe 0.

Atenție: se va salva modelul creat și va închide simularea, urmând să fie redeschisă.



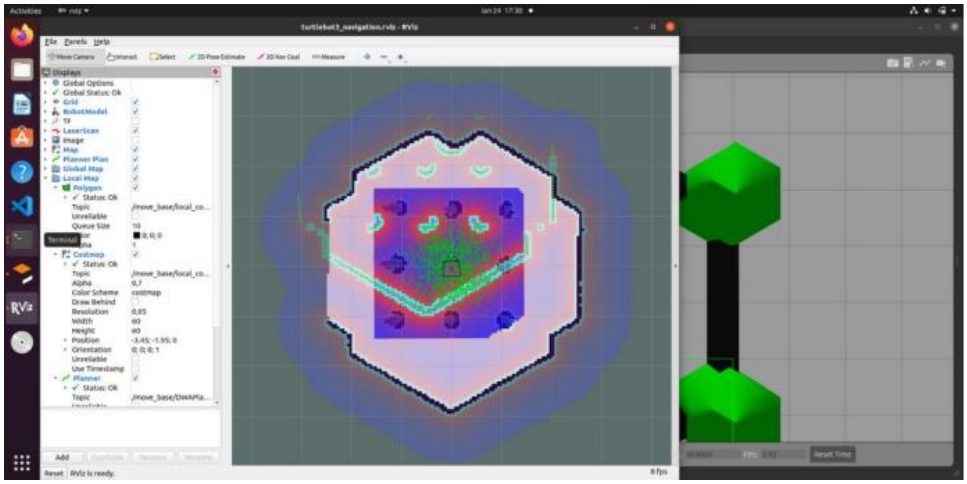


- Pentru detecția de culoare se va utiliza nodul image_color prin comanda: rosrun image_proc image_color.py

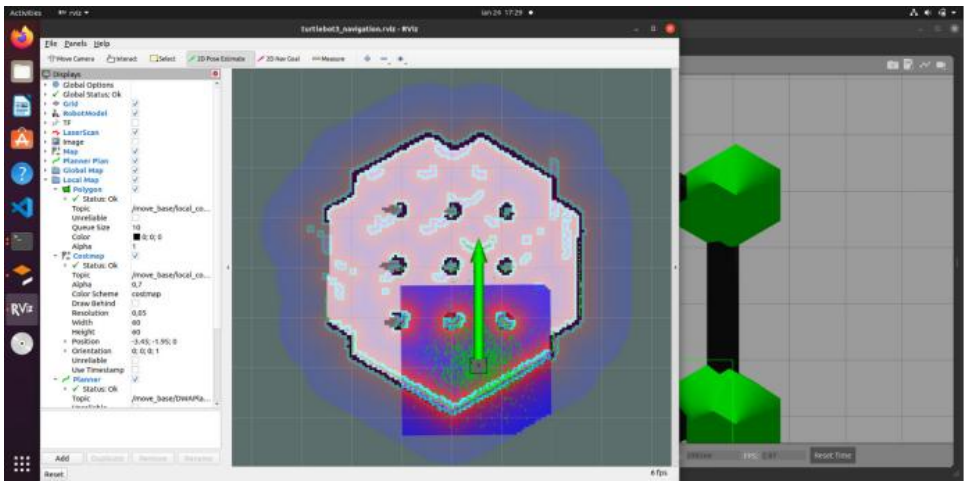


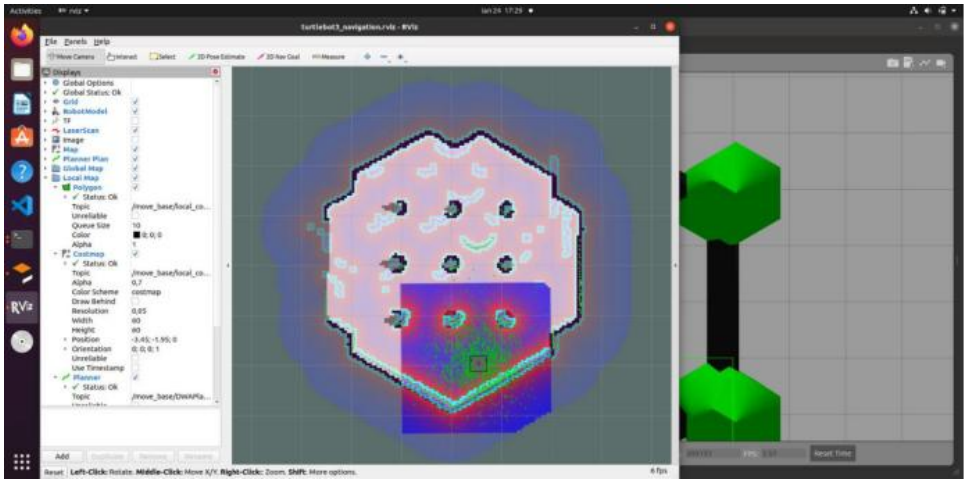
Pentru deplasarea robotului în locul mai puțin vizibil

- După deschiderea mediului de simulare Gazebo, se va deschide hărții salvate în fișierul map.yaml în mediul Rviz prin comanda:
  roslaunch turtlebot3_navigation turtlebot3_navigation
  .launch map_file:=$HOME/map.yaml

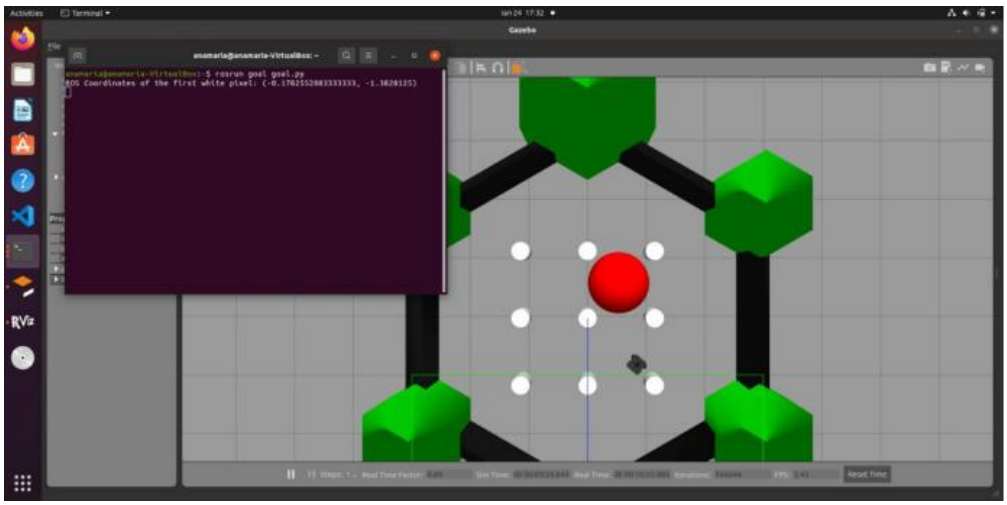


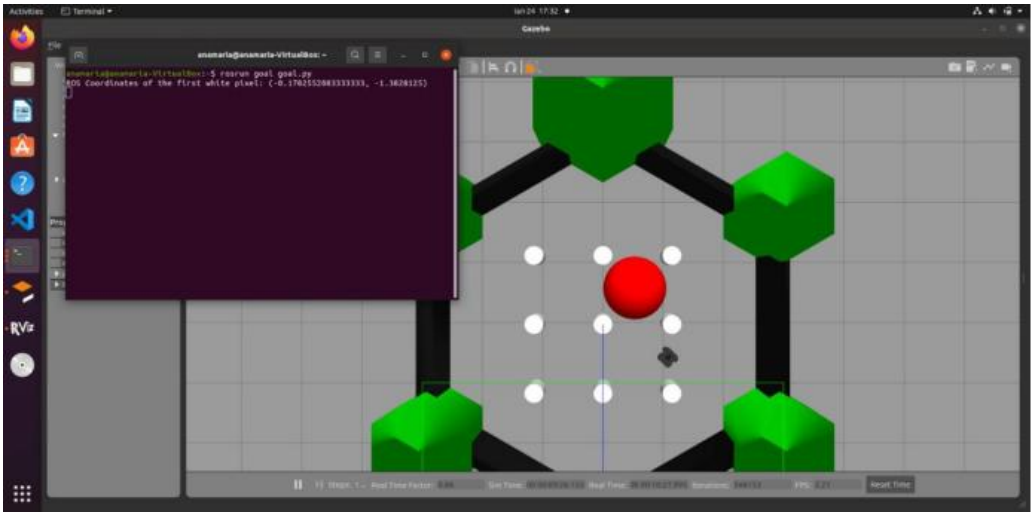
- Prin utilizarea 2d Pose Estimate se va estima poziția inițială a robotului

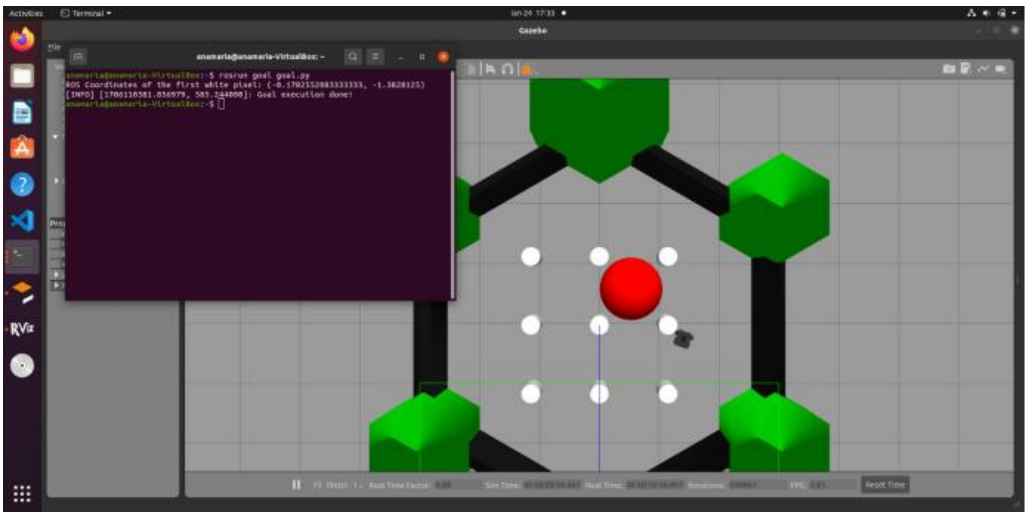




- Pentru deplasarea robotului în cel mai puțin vizibil punct se va folosi nodul goal prin rularea comenzii: rosrun goal goal.py







6. Testare şi punere în funcţiune

Modul de Testare al Aplicației Hide&Seek pentru TurtleBot3

- **Configurarea Mediului de Testare:**

Asigurați-vă că aveți un simulator ROS (Gazebo) instalat și funcțional, împreună cu dependențele necesare pentru TurtleBot3.

Pregătiți o hartă simulată în Gazebo care să reprezinte labirintul pentru jocul Hide&Seek.

- **Instalarea Aplicației:**

Descărcați codul sursă al aplicației Hide&Seek de pe repository-ul dedicat.

Deschideți terminalul în directorul proiectului și executați comanda de instalare pentru a gestiona dependențele și pentru a asigura integritatea pachetelor ROS.

- **Configurarea Parametrilor de Joc:**

Modificați setările pentru a specifica dimensiunile și caracteristicile labirintului, precum și proprietățile robotului în fișierul de configurare al aplicației.

- **Rularea Aplicației în Mod de Testare:**

Porniți simulatorul Gazebo.

În terminal, executați comanda pentru a lansa nodul principal al aplicației Hide&Seek.

Vizualizați simularea în Gazebo și monitorizați comportamentul robotului în timp real.

- **Testarea Diferitelor Scenarii:**

Testați capacitatea robotului de a se ascunde în zone mai puțin vizibile și de a găsi obiectul roșu în diferite locații ale labirintului.

Monitorizați traseul robotului, interacțiunea cu mediul și reacțiile sale la comenzi.

Modul de Instalare al Aplicației Hide&Seek:

- **Descărcarea Surselor:**

Accesați repository-ul dedicat pe platforma de versionare (GitHub) și descărcați sursa aplicației Hide&Seek.

- **Instalarea Dependințelor ROS:**

Asigurați-vă că aveți instalat ROS pe sistemul dumneavoastră și că toate pachetele necesare pentru TurtleBot3 și simularea Gazebo sunt instalate.

- **Configurarea Mediului de Dezvoltare:**

Utilizați un mediu de dezvoltare integrat (IDE) pentru a edita și adapta codul sursă în funcție de specificațiile mediului vostru.

- **Compilarea și Construirea Pachetului:**

Executați comenzile de compilare și construire a pachetului pentru a asigura că aplicația este pregătită pentru rulare.

- **Lansarea Aplicației:**

În terminal, executați comanda pentru a porni aplicația Hide&Seek, specificând parametrii necesari, cum ar fi mediul de simulare și caracteristicile jocului.

7.Prezentarea echipei

Numele echipei: Cameleon

Număr membrii: 2

Membri:

Bota Iulia-Alexis

-Instalare ROS

-Pregătirea simulării

-Funcția de găsire a unui punct mai puțin vizibil

-Scrierea documentării și a ghidului

Cauc Ana-Maria Andreea

-Funcția de transformare a coordonatelor din pixeli în coordonate ROS

-Găsirea coordonatelor originii în mediul Gazebo

-Nodul pentru procesarea imaginii

-Testare

8.Concluzie

Roboții mobili, echipați cu tehnologii avansate, au o gamă largă de aplicații practice, de la industrie la asistență medicală și logistică. Conducerea la distanță adaugă eficiență și accesibilitate, permițând operatorilor să coordoneze operațiuni în timp real. Cu toate acestea, apar și provocări, cum ar fi securitatea sistemelor și întârzierile în comunicare. În ansamblu, această evoluție tehnologică redefinește modul în care interacționăm cu mediul înconjurător, aducând soluții inovatoare și eficiență sporită în diverse domenii.

În concluzie, aplicația Hide&Seek pentru TurtleBot3 reprezintă o implementare inovatoare și captivantă a conceptului clasic de ascundere și căutare într-un mediu simulat cu ajutorul ROS. Integrarea tehnologiilor precum SLAM, navigație autonomă și procesare a imaginilor adaugă complexitate și provocări, oferind un cadru stimulativ pentru dezvoltarea și testarea capacităților robotului mobil. Aplicația nu doar demonstrează abilitățile tehnologice ale TurtleBot3, ci și ilustrează potențialul său de a fi utilizat în contexte educaționale, de cercetare sau în domeniul hobby-ului.

9.Bibliografie

[www 01] https://emanual.robotis.com/
[www 02] https://stackoverflow.com/
[www 03] https://prabhjotkaurgosal.com/
[www 04] https://answers.ros.org/
[www 05] http://wiki.ros.org/
[www 06] http://logiscool.com/