



UNIVERSITATEA “POLITEHNICA” DIN TIMISOARA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL AUTOMATICĂ ȘI INFORMATICĂ APLICATĂ

TAMAGOTCHI

PROIECT SINCRETIC I

AUTORI: Andreea-Haideea RUS
Daria SLĂVOACĂ

Coordonatori: [Conf.dr.ing. Florin DRĂGAN, As. ing. Emil VOIȘAN]

CUPRINS

1. Introducere	1
2. Prezentarea temei	2
3. Tehnologii utilizate	3
4. Ghidul programatorului	5
5. Ghidul utilizatorului	12
6. Testare și punere în funcțiune	14
7. Prezentarea firmei	17
8. Concluzii	17
9. Bibliografie	18

1. INTRODUCERE

În contextul avansului continuu al tehnologiei, explorarea domeniului roboților mobili și a conducerii la distanță a acestora reprezintă un subiect de interes. Această documentație propune o analiză asupra interacțiunii dintre om și roboții mobili, focalizându-se pe modalitățile de control și comunicare la distanță.

În esență, funcționarea practică a roboților mobili într-un scenariu de conducere la distanță implică o interacțiune complexă între diferite componente tehnologice. Aceștia sunt echipați cu o serie de senzori avansați, precum camere video, LiDAR (Light Detection and Ranging), senzori de proximitate și alți senzori specializați. Aceste dispozitive permit robotului să colecteze informații detaliate despre mediul său înconjurător, cum ar fi obstacolele, suprafețele, sau alte elemente semnificative. Datele colectate de senzori sunt apoi transmise către sistemul central de control utilizând diverse mijloace de comunicație, cum ar fi rețelele wireless sau protocoalele de transmisie a datelor. Sistemul central, echipat cu algoritmi avansați de procesare a datelor și inteligență artificială, analizează aceste informații pentru a lua decizii în timp real și pentru a adapta comportamentul robotului în funcție de nevoile specifice ale mediului și sarcinilor în curs. Interfața om-mașină (HMI) servește ca punte de comunicare esențială între om și robot. Acest dispozitiv interactiv furnizează informații detaliate despre starea robotului, precum și date de la senzori.

În practică, roboții mobili sunt utilizați într-o varietate de domenii, iar controlul la distanță devine o soluție indispensabilă în situații în care intervenția umană directă poate fi dificilă sau periculoasă. Pentru a ilustra importanța și utilitatea roboților mobili, vom explora câteva aplicații ale conducerii la distanță a acestora:

Industrie: sunt folosiți pentru transportul mărfurilor în depozite, îmbunătățind eficiența.

Medicină: sunt folosiți în intervenții chirurgicale pentru precizie și acces mai bun.

Agricultură: sunt folosiți în diverse activități agricole precum recoltarea și administrarea de substanțe nutritive.

Aceste cazuri ilustrează versatilitatea și impactul semnificativ al acestei tehnologii în diferite domenii evidențiând totodată abilitatea de a controla roboții în mod eficient, oferind soluții inovatoare.



2. PREZENTAREA TEMEI

Tema echipei noastre, numită TechCompanions, este inspirată de conceptul clasic Tamagotchi și realizată cu ajutorul robotului mobil TurtleBot3 Burger.

Tamagotchi a fost creat de către compania japoneză Bandai și a fost lansat pentru prima dată pe piață în 1996. Jucăria a devenit rapid un fenomen global, captivând atenția copiilor și a adulților deopotrivă. Conceptul central este de a simula creșterea și îngrijirea unui animal virtual. Utilizatorii trebuie să se ocupe de nevoile acestuia, cum ar fi hrănirea și interacțiunea cu el, în timp ce observă cum acesta crește și evoluează.

În ceea ce privește legătura cu roboțul nostru, ne-am inspirat din spiritul jucăuș al Tamagotchi-ului original pentru a crea o experiență inovatoare, transpunând această interacțiune virtuală într-un mediu fizic prin tehnologia roboților mobili. Astfel, roboțul nostru reacționează la culorile din jur, oferind utilizatorilor o modalitate de a interacționa cu el și de a observa reacțiile sale distincte.

Robotul are trei comportamente:

1. **Comportament albastru – Comportament jucăuș :** când robotul percepe culoarea albastră, intră în modul de joacă, reflectând comportamentul activ asociat Tamagotchi-ului.

Acțiune: Roboțul va realiza mișcări de rotație și se va deplasa până când nu mai vede obiectul albastru.

2. **Comportament violet - Emisie de sunet:** când robotul percepe culoarea violet, interpretează o melodie, amintind de momentele muzicale din jocul original.

Acțiune: Prin intermediul unui difuzor integrat, roboțul interpretează o melodie prestabilită.

3. **Comportament roșu - Mișcări circulare:** când robotul percepe culoarea roșie, acesta exprimă bucuria prin mișcări circulare, asemănătoare Tamagotchi-ului în momentele de fericire.

Acțiune: Roboțul se rotește în jurul propriului ax.

Astfel, ne propunem să transpunem nostalgia și bucuria asociate jocului Tamagotchi într-un cadru modern, adoptând tehnologia roboților mobili.



3. TEHNOLOGII UTILIZATE

Robotul utilizat pentru tema propusă este TurtleBot 3 Burger, dezvoltat de către ROBOTIS. Acesta este un robot mobil echipat cu o serie de caracteristici care îl fac potrivit pentru diverse aplicații, inclusiv pentru scenariul propus de conducere la distanță în stilul unui Tamagotchi.

Caracteristici cheie:

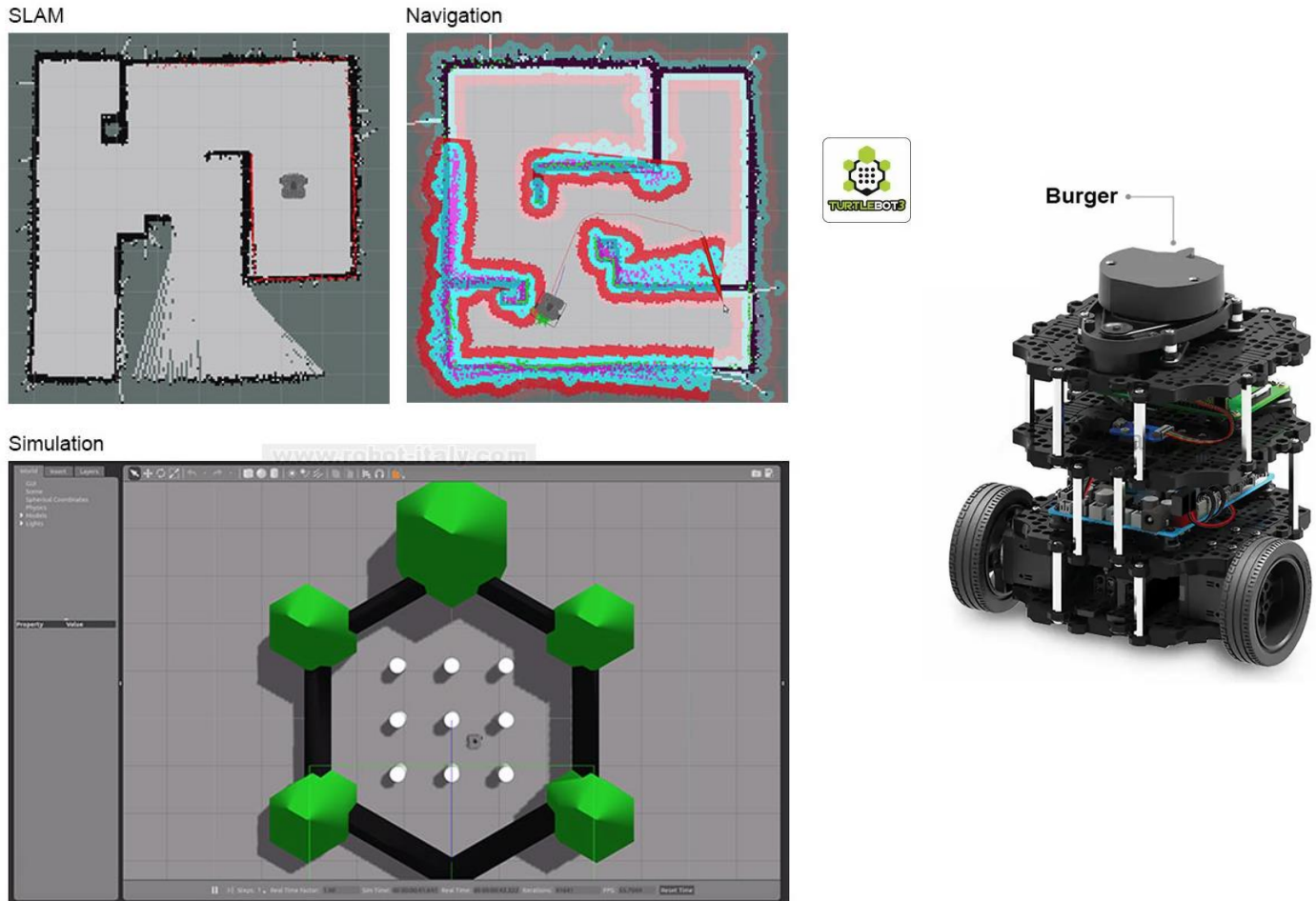
- **Senzori avansați:** Echipat cu senzori de distanță, senzori de înclinare și, opțional, un senzor LiDAR, TurtleBot 3 Burger poate percepe mediul înconjurător și colecta date relevante în timp real.
- **Placă de dezvoltare Raspberry Pi:** Această placă servește drept creier central al robotului, gestionând software-ul de control, luând decizii bazate pe datele colectate și facilitând comunicarea eficientă între diversele componente ale sistemului.
- **Motoare și șasiu:** Cu motoare puternice și un șasiu mobil, TurtleBot 3 Burger se deplasează agil și poate executa manevre precise.
- **Tehnologia SLAM (Simultaneous Localization and Mapping):** este o tehnologie esențială pentru navigarea autonomă a roboților. Utilizând senzori avansați precum LiDAR și camere, robotul poate determina poziția și poate construi o hartă detaliată a mediului înconjurător în timp real. Datele sunt procesate cu algoritmi specializați și integrate în sistemul de operare ROS, permitând robotului să se deplaseze autonom, să evite obstacole și să ajusteze traseul în funcție de schimbările din mediu.

Medii de programare utilizate:

- **ROS (Robot Operating System):** Un sistem de operare pentru roboți caracterizat de flexibilitatea pe care o oferă, construit pentru a facilita programarea și controlul roboților. Prin intermediul unui sistem modular, ROS permite dezvoltatorilor să creeze aplicații robotice prin împărțirea funcționalităților în pachete independente. Este special proiectat pentru a gestiona variatele dispozitive hardware folosite de roboți și pentru a facilita comunicația între diferitele componente ale sistemului.
- **Limbaje de programare:** Pentru dezvoltarea aplicației software am utilizat limbajul Python, integrat în mediul ROS. Alegerea acestui limbaj facilitează dezvoltarea de cod eficient și ușor de înțeles, esențial pentru o comunicare fluidă între om și robot.



- Simulator Gazebo: În procesul de dezvoltare, folosim Gazebo, un simulator 3D, pentru a testa și simula comportamentul robotului în medii virtuale. Această abordare ne permite să validăm și să optimizăm codul software într-un mediu controlat înainte de a-l implementa pe robotul real.



4. GHIDUL PROGRAMATORULUI

Pentru instalarea ROS se realizează următorii pași:

- Configurarea fișierului source.list prin instrucțiunea

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >  
/etc/apt/sources.list.d/ros-latest.list'
```

- Setarea cheilor

```
sudo apt install curl
```

```
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key  
add -
```

- Instalarea propriu zisă

```
sudo apt update
```

Se alege varianta de ROS1 cu care vom lucra: Noetic

```
sudo apt install ros-noetic-desktop-full
```

- Setarea mediului de lucru

```
source /opt/ros/noetic/setup.bash
```

Pentru configurarea mediului de lucru

- Se creează un director

```
mkdir catkin_ws
```

```
cd catkin_ws
```

```
mkdir src
```

Apoi se compilează codul sursă

```
catkin_make
```

Se configurează variabilele de mediu

```
source ~/catkin_ws/devel/setup.bash
```

Se creează un pachet în catkin_ws/src

```
catkin_create_pkg numele_pachetului rospy (sau roscpp) turtlesim
```



Utilizând comanda `code .` se deschide IDE-ul ales, Visual Studio

Apoi se compilează din nou codul cu `catkin_make`

Pentru crearea unui nod se folosește calea

```
cd catkin_ws/src/my_robot_controler/
```

Unde se creează un director `scripts`

```
mkdir scripts
```

În care se adaugă un fișier de tip `py` sau `cpp` în funcție de tehnologia utilizată

```
touch cod_lab.py
```

Căruia i se acordă privilegii prin comanda

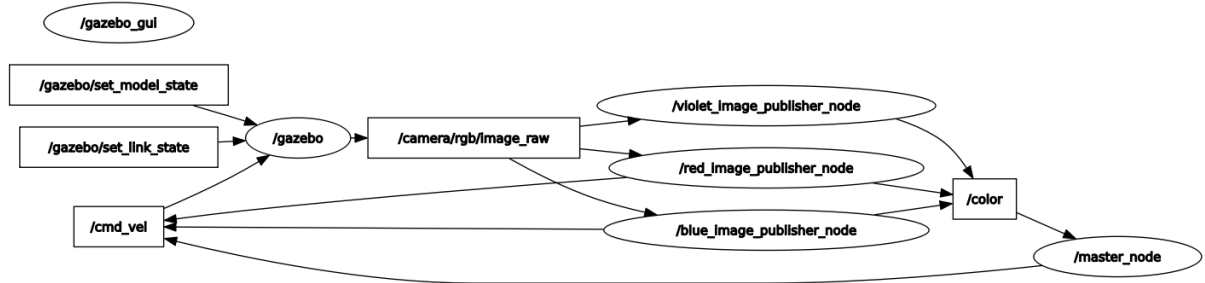
```
chmod +x cod_lab.py
```

Fișier în care se va adăuga codul corespunzător .



- Pentru acest proiect, am creat 4 noduri:

Aceste noduri comunică între ele prin intermediul topicurilor. În continuare vom descrie fiecare nod și cum sunt conectate între ele:



1. Nodul Principal (`master_node`):

- Publică comenzi de mișcare pe topicul "cmd_vel" în funcție de starea culorilor detectate.
- Se abonează la topicurile de culoare ("color") pentru a primi informații despre culorile detectate de nodurile specializate.
- Rulează un obiect de tip `TamagotchiBehavior` care gestionează starea și comenzile de mișcare.

```

1  #!/usr/bin/env python3
2  import rospy
3  from geometry_msgs.msg import Twist
4  from std_msgs.msg import String
5
6  class TamagotchiBehavior:
7      def __init__(self):
8          self.state = "STOP"
9          self.color_sub = rospy.Subscriber("color", String, self.color_callback)
10         self.cmd_vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
11
12     def color_callback(self, msg):
13         self.state = msg.data
14
15     def run(self):
16         rate = rospy.Rate(2)
17         while not rospy.is_shutdown():
18             if self.state == "BLUE":
19                 self.publish_move(0.5, 1.0)
20                 rospy.sleep(3)
21                 self.state = "STOP"
22             elif self.state == "VIOLET":
23                 self.publish_move(0.0, 0.0)
24                 rospy.sleep(3)
25                 self.state = "STOP"
26             elif self.state == "RED":
27                 self.publish_move(0.0, 1.0)
28                 rospy.sleep(3)
29                 self.state = "STOP"
30             else:
31                 self.publish_move(0.5, 0.0)
32                 rate.sleep()
33
34     def publish_move(self, linear_x, angular_z):
35         msg = Twist()
36         msg.linear.x = linear_x
37         msg.angular.z = angular_z
38         self.cmd_vel_pub.publish(msg)
39
40 if __name__ == '__main__':
41     rospy.init_node("master_node")
42     rospy.loginfo("Master node is running")
43     state_machine = TamagotchiBehavior()
44     state_machine.run()
45

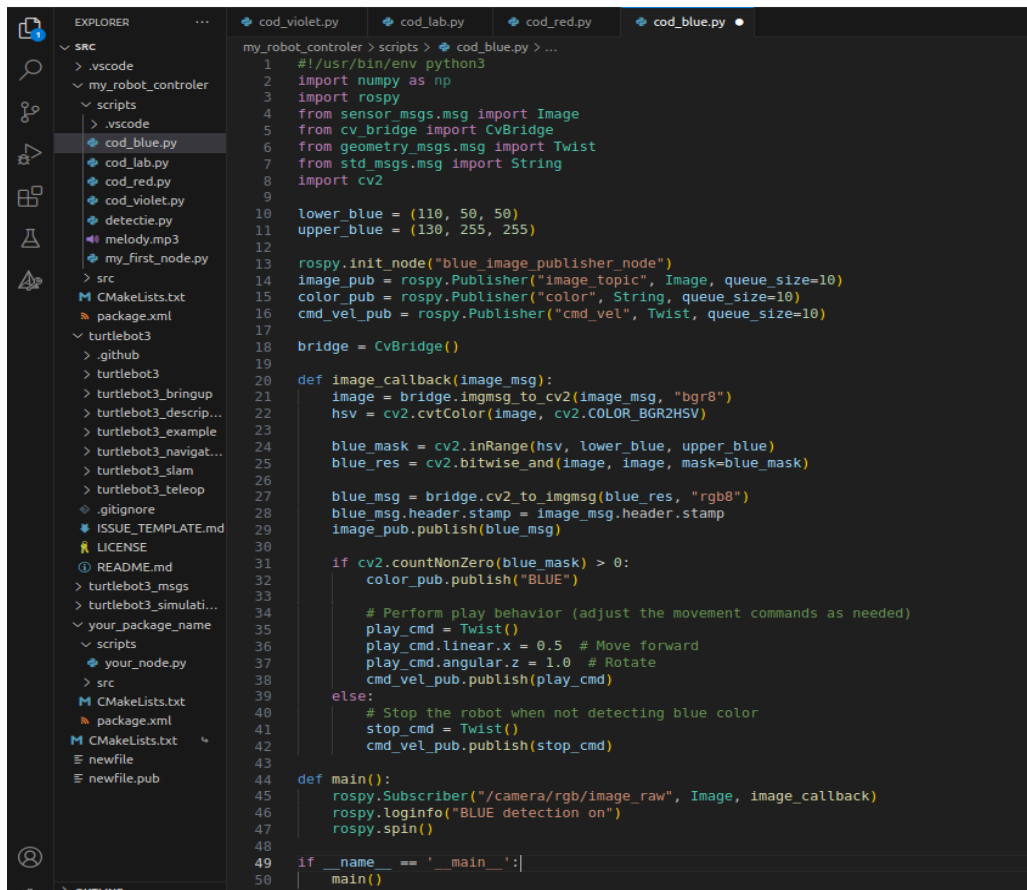
```

Fig. Nod principal



2. Nodul Albastru ('blue_image_publisher_node'):

- Publică imagini prelucrate pentru culoarea albastră pe topicul "image_topic".
- Publică informații despre culoarea detectată ("color") pe topicul "color".
- Publică comenzi de mișcare pe topicul "cmd_vel" în funcție de starea culorii.
- Se abonează la imaginile de la camera ("camera/rgb/image_raw") pentru a le prelucra și pentru a detecta culoarea albastră.



```

1  #!/usr/bin/env python3
2  import numpy as np
3  import rospy
4  from sensor_msgs.msg import Image
5  from cv_bridge import CvBridge
6  from geometry_msgs.msg import Twist
7  from std_msgs.msg import String
8  import cv2
9
10 lower_blue = (110, 50, 50)
11 upper_blue = (130, 255, 255)
12
13 rospy.init_node("blue_image_publisher_node")
14 image_pub = rospy.Publisher("image_topic", Image, queue_size=10)
15 color_pub = rospy.Publisher("color", String, queue_size=10)
16 cmd_vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
17
18 bridge = CvBridge()
19
20 def image_callback(image_msg):
21     image = bridge.imgmsg_to_cv2(image_msg, "bgr8")
22     hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
23
24     blue_mask = cv2.inRange(hsv, lower_blue, upper_blue)
25     blue_res = cv2.bitwise_and(image, image, mask=blue_mask)
26
27     blue_msg = bridge.cv2_to_imgmsg(blue_res, "rgb8")
28     blue_msg.header.stamp = image_msg.header.stamp
29     image_pub.publish(blue_msg)
30
31     if cv2.countNonZero(blue_mask) > 0:
32         color_pub.publish("BLUE")
33
34         # Perform play behavior (adjust the movement commands as needed)
35         play_cmd = Twist()
36         play_cmd.linear.x = 0.5 # Move forward
37         play_cmd.angular.z = 1.0 # Rotate
38         cmd_vel_pub.publish(play_cmd)
39     else:
40         # Stop the robot when not detecting blue color
41         stop_cmd = Twist()
42         cmd_vel_pub.publish(stop_cmd)
43
44 def main():
45     rospy.Subscriber("/camera/rgb/image_raw", Image, image_callback)
46     rospy.loginfo("BLUE detection on")
47     rospy.spin()
48
49 if __name__ == '__main__':
50     main()

```

Fig. Nod albastru



3. Nodul Roșu (`red_image_publisher_node`):

- Publică imagini prelucrate pentru culoarea roșie pe topicul "image_topic".
- Publică informații despre culoarea detectată ("color") pe topicul "color".
- Publică comenzi de mișcare pe topicul "cmd_vel" în funcție de starea culorii.
- Se abonează la imaginile de la camera ("camera/rgb/image_raw") pentru a le prelucra și pentru a detecta culoarea roșie.

```

1  #!/usr/bin/env python3
2  import numpy as np
3  import rospy
4  from sensor_msgs.msg import Image
5  from cv_bridge import CvBridge
6  from geometry_msgs.msg import Twist
7  from std_msgs.msg import String
8  import cv2
9
10 lower_red = (0, 50, 50)
11 upper_red = (10, 255, 255)
12
13 rospy.init_node("red_image_publisher_node")
14 image_pub = rospy.Publisher("image_topic", Image, queue_size=10)
15 color_pub = rospy.Publisher("color", String, queue_size=10)
16 cmd_vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
17
18 bridge = CvBridge()
19
20 def image_callback(image_msg):
21     image = bridge.imgmsg_to_cv2(image_msg, "bgr8")
22     hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
23
24     red_mask = cv2.inRange(hsv, lower_red, upper_red)
25     red_res = cv2.bitwise_and(image, image, mask=red_mask)
26
27     red_msg = bridge.cv2_to_imgmsg(red_res, "rgb8")
28     red_msg.header.stamp = image_msg.header.stamp
29     image_pub.publish(red_msg)
30
31     if cv2.countNonZero(red_mask) > 0:
32         color_pub.publish("RED")
33
34         rotate_cmd = Twist()
35         rotate_cmd.angular.z = 1.0 # Set angular velocity for rotation
36         cmd_vel_pub.publish(rotate_cmd)
37     else:
38         stop_cmd = Twist()
39         cmd_vel_pub.publish(stop_cmd)
40
41 def main():
42     rospy.Subscriber("/camera/rgb/image_raw", Image, image_callback)
43     rospy.loginfo("Red detection on")
44     rospy.spin()
45
46 if __name__ == '__main__':
47     main()
48

```

Fig. Nod roșu

4. Nodul Violet (`violet_image_publisher_node`):

- Publică imagini prelucrate pentru culoarea violet pe topicul "image_topic".
- Publică informații despre culoarea detectată ("color") pe topicul "color".
- Publică comenzi de mișcare pe topicul "cmd_vel" în funcție de starea culorii.
- Se abonează la imaginile de la camera ("camera/rgb/image_raw") pentru a le prelucra și pentru a detecta culoarea violet.



```

1  #!/usr/bin/env python3
2  import numpy as np
3  import rospy
4  from sensor_msgs.msg import Image
5  from cv_bridge import CvBridge
6  from std_msgs.msg import String
7  from geometry_msgs.msg import Twist
8  import cv2
9  import pygame
10
11  lower_violet = (130, 50, 50)
12  upper_violet = (170, 255, 255)
13
14  rospy.init_node("violet_mage_publisher_node")
15  image_pub = rospy.Publisher("image_topic", Image, queue_size=10)
16  color_pub = rospy.Publisher("color", String, queue_size=10)
17
18  bridge = CvBridge()
19
20  # Initialize Pygame
21  pygame.init()
22  melody_file = "/home/andreea/catkin_ws/src/my_robot_controller/scripts/melody.mp3"
23  is_color_detected = False
24
25  def play_melody():
26      pygame.mixer.music.load(melody_file)
27      pygame.mixer.music.play()
28
29  def stop_melody():
30      pygame.mixer.music.stop()
31
32  def image_callback(image_msg):
33      global is_color_detected
34
35      image = bridge.imgmsg_to_cv2(image_msg, "bgr8")
36      hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
37
38      violet_mask = cv2.inRange(hsv, lower_violet, upper_violet)
39      violet_res = cv2.bitwise_and(image, image, mask=violet_mask)
40
41      violet_msg = bridge.cv2_to_imgmsg(violet_res, "rgb8")
42      violet_msg.header.stamp = image_msg.header.stamp
43      image_pub.publish(violet_msg)
44
45      if cv2.countNonZero(violet_mask) > 0:
46          if not is_color_detected:
47              is_color_detected = True
48              color_pub.publish("VIOLET")
49              play_melody()
50          else:
51              if is_color_detected:
52                  is_color_detected = False
53                  stop_melody()
54
55  def main():
56      rospy.Subscriber("/camera/rgb/image_raw", Image, image_callback)
57      rospy.loginfo("Violet detection on")
58      rospy.spin()
59
60  if __name__ == '__main__':
61      main()

```

Fig. Nod violet

De asemenea cu ajutorul topicurilor se realizează următoarele:

1. Topicul "cmd_vel" (Comenzi de Mișcare):

- Publicat de nodul principal ('master_node').
- Se abonează la acest topic pentru a primi comenzi de mișcare.
- Folosit pentru a controla mișcarea robotului în funcție de culorile detectate.
- Nodurile speciale (albastru, roșu, violet) publică comenzi pe acest topic pentru a influența mișcarea robotului.

2. Topicul "color" (Informații despre Culori):

- Publicat de nodurile speciale pentru fiecare culoare (albastru, roșu, violet).
- Se abonează la acest topic pentru a primi informații despre culorile detectate.
- Folosit de nodul principal ('master_node') pentru a cunoaște starea culorilor detectate și a decide acțiunile robotului.



3. Topicul "image_topic" (Imagini Prelucrate):

- Publicat de nodurile speciale pentru fiecare culoare (albastru, roșu, violet).
- Se abonează la acest topic pentru a primi imagini prelucrate ale culorilor detectate.
- Folosit de nodul principal (`master_node`) pentru a afișa imagini prelucrate în timpul rulării.

De asemenea, folosind următoarea serie de comenzi am reușit să scanăm harta în care robotul își desfășoară acțiunile:

- Deschiderea lumii și adăugarea robotului

```
export TURTLEBOT3_MODEL=waffle  
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

- Într-un alt terminal se rulează nodul pentru localizare și simulare a hărții

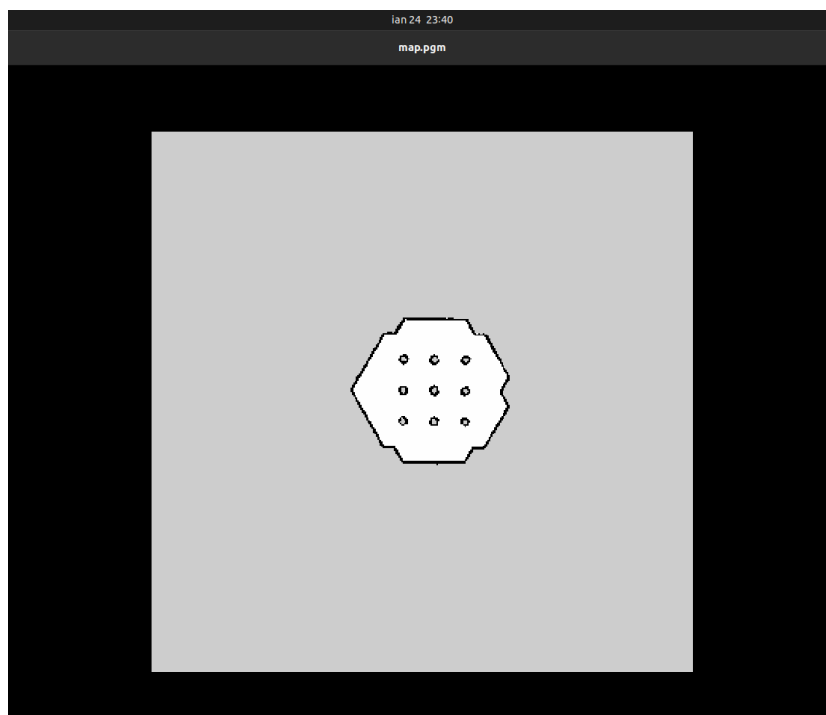
```
export TURTLEBOT3_MODEL=waffle  
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

Într-un alt terminal se rulează nodul responsabil pentru deplasarea robotului

```
export TURTLEBOT3_MODEL=burger  
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

După parcurgerea hărții cu robotul, vom obține o imagine pe care o vom salva astfel

```
roslaunch map_server map_saver -f ~/map
```



5. GHIDUL UTILIZATORULUI

Pentru a rula aplicația, e nevoie de instalarea Ubuntu 20.04, ROS Noetic, Python3 și OpenCV2, urmând pașii de mai jos:

- Instalare Ubuntu 20.04:

Utilizatorul trebuie să descarce și să instaleze Ubuntu 20.04 pe computerul său. Se pot urma pașii descriși în link-ul următor

<https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview>

- Instalare ROS Noetic:

Utilizatorul trebuie să urmeze pașii de instalare pentru ROS Noetic, conform instrucțiunilor oficiale disponibile pe site-ul ROS:

<https://wiki.ros.org/noetic/Installation/Ubuntu>

- Instalare Python3 și OpenCV2:

Pentru instalare Python3 se va folosi comanda:

```
sudo apt-get install python3
```

Pentru OpenCV2, poate utiliza comanda:

```
sudo apt-get install python3-opencv
```

Apoi se vor urma pașii de mai jos:

- Se deschide un terminal unde se va rula comanda:

```
ssh.ros@[turtlebot3IP]
```

- Pornirea nodului principal (roscore):

Utilizatorul trebuie să deschidă un terminal și să ruleze comanda:

```
roscore
```

- În primul terminal se execută:

```
roslaunch turtlebot3_bringup turtlebot3_remote.launch
```

Comanda realizează conexiunea cu robotul



- RVIZ:

Pentru a vizualiza camera robotului se va executa:

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:gmapping
```

Iar pentru a lansa camera frontală:

```
Roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch
```

- Rularea scripturilor:

Utilizatorul trebuie să ruleze scripturile necesare pentru controlul roboțelului și pentru detectarea cuburilor.

```
roslaunch my_robot_controler my_launch_file.launch
```

- Deschiderea Lumii Alese:

Utilizatorul trebuie să deschidă o nouă fereastră de terminal și să ruleze comanda pentru a deschide lumea în care robotelul va explora. De exemplu, dacă utilizează Gazebo, poate utiliza o comandă de genul:

```
roslaunch my_robot_controler my_launch_file.launch
```

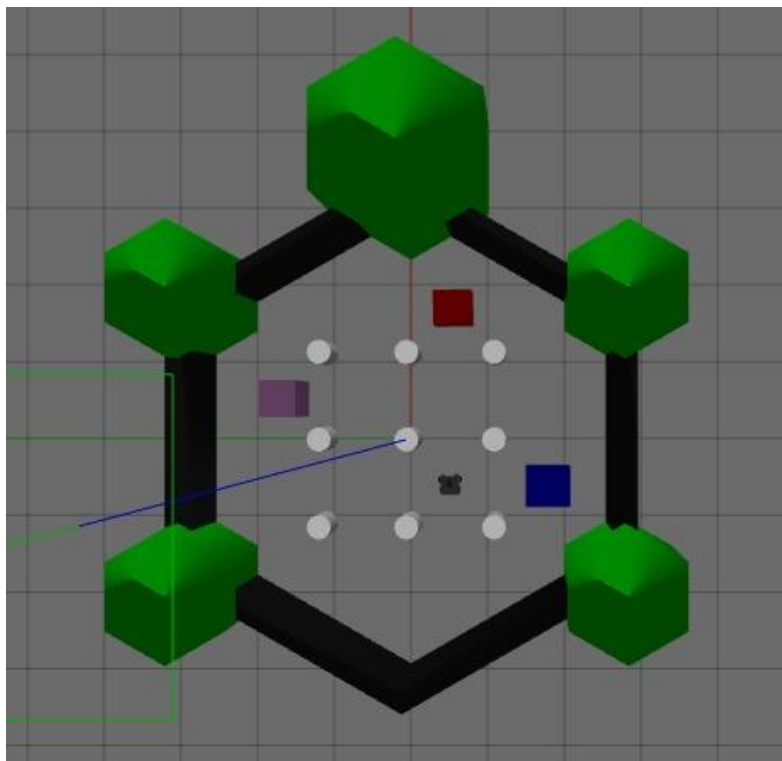


Fig. Gazebo



Funcționalitatea roboțelului:

- Explorarea hărții de către roboțel:

După ce scripturile sunt rulate și lumea este deschisă, roboțelul va începe să exploreze harta până când găsește unul dintre cele trei cuburi.

- Reacții la găsirea cuburilor:

- La găsirea cubului violet, roboțelul va porni fișierul melody.mp3 ce conține melodia aleasă
- La găsirea cubului roșu, roboțelul se va învârti de bucurie în jurul propriului ax
- La găsirea cubului albastru, roboțelul va deveni energic, va realiza mișcări de rotație și se va deplasa până când nu mai vede cubul albastru.

Acești pași oferă o imagine generală a modului în care utilizatorul ar trebui să ruleze aplicația în mediul ROS. Utilizatorul trebuie să se asigure că scripturile sunt corecte și integrate cu mediul său de simulare, iar lumea aleasă este deschisă corespunzător.

6. TESTARE ȘI PUNERE ÎN FUNCȚIUNE

Asigurarea Instalării și Configurării ROS:

1. Instalarea ROS:

- Utilizatorul trebuie să instaleze și să configureze ROS pe sistemul său. Acest lucru poate fi realizat urmând ghidurile de instalare oficiale disponibile pe site-ul ROS.

2. Crearea spațiului de lucru ROS:

- Utilizatorul trebuie să creeze un spațiu de lucru ROS utilizând comanda:

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws/
```

```
catkin_make
```

```
source devel/setup.bash
```

3. Instalarea bibliotecilor necesare:

- Bibliotecile necesare (cum ar fi OpenCV, Pygame, etc.) trebuie să fie instalate în spațiul de lucru ROS. Acest pas a fost descris în capitolul Ghidul utilizatorului.



4. Lansarea nodurilor:

- Utilizatorul trebuie să creeze un fișier de lansare configurat corespunzător. Acest fișier de lansare ar trebui să includă toate nodurile necesare pentru funcționalitatea proiectului și să fie lansat cu comanda:

`roslaunch my_robot_controller my_launch_file.launch`

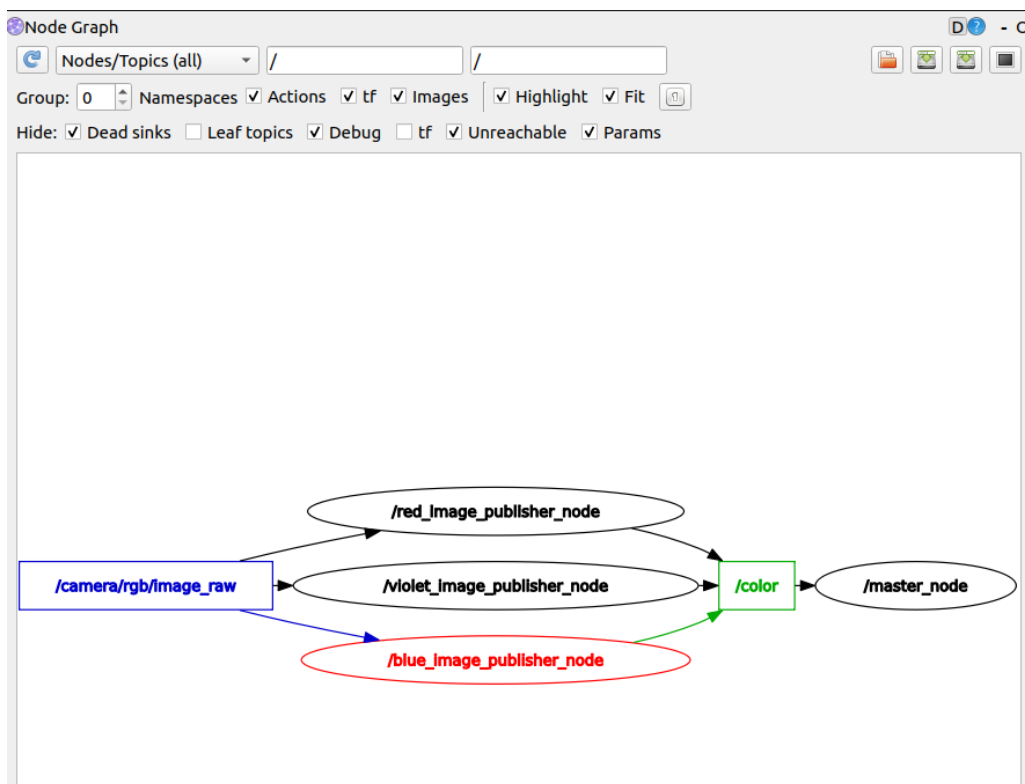
5. Vizualizarea grafului de comunicație:

- Utilizatorul poate utiliza comanda `rqt_graph` pentru a vizualiza legăturile dintre noduri și a verifica dacă comunicarea între ele este corectă.

```

1 <!-- my_launch_file.launch -->
2 <launch>
3   <!-- Launch the master node -->
4   <node name="master_node" pkg="my_robot_controller" type="cod_lab.py" output="screen" />
5
6   <!-- Launch the blue node -->
7   <node name="blue_image_publisher_node" pkg="my_robot_controller" type="cod_blue.py" output="screen" />
8
9   <!-- Launch the red node -->
10  <node name="red_image_publisher_node" pkg="my_robot_controller" type="cod_red.py" output="screen" />
11
12  <!-- Launch the violet node -->
13  <node name="violet_image_publisher_node" pkg="my_robot_controller" type="cod_violet.py" output="screen" />
14 </launch>
15

```



6. Testarea scenariilor:

- Utilizatorul trebuie să experimenteze cu diferite scenarii, cum ar fi aducerea robotului în prezența uneia dintre cele 3 culori și observarea comportamentului său.

Prin urmare, pentru a testa și a experimenta cu proiectul, utilizatorul ar trebui să urmeze acești pași, asigurându-se că mediul ROS este corect instalat și configurat, iar nodurile proiectului sunt lansate și comunică corespunzător.

7. Controlul robotului cu Teleop:

Utilizatorul deschide o nouă fereastră de terminal pe sistemul său.

Apoi, utilizatorul lansează simularea Gazebo cu TurtleBot3.

```
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

În terminalul deschis, utilizatorul rulează comanda pentru teleop:

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Va apărea o interfață de tastatură, oferind utilizatorului controlul asupra robotului.

- **Controlul robotului:**

Se utilizează tastele W, A, S, D pentru mișcare înainte, la stânga, înapoi și la dreapta, respectiv.

Reglarea vitezei se face cu tastele săgeți sus/jos.

- **Oprirea Teleop-ului:**

După ce utilizatorul a terminat de controlat robotul, poate opri controlul tastând **Ctrl+C** în fereastra terminalului.

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

Control Your TurtleBot3!
-----
Moving around:
      w
a      s      d
      x

w/x : increase/decrease linear velocity
a/d : increase/decrease angular velocity
space key, s : force stop
```



7. PREZENTAREA FIRMEI

Acest capitol prezintă membrii echipei TechCompanions, evidențiind rolurile lor în dezvoltarea proiectului cu TurtleBot3 și simularea comportamentului Tamagotchi.

Andreea-Haideea RUS

- Implementarea codului
- Realizarea documentației

Daria SLĂVOACĂ

- Implementarea codului
- Realizarea documentației

8. CONCLUZII

În urma dezvoltării proiectului, am reușit să integram cu succes comportamentul Tamagotchi în cadrul robotului TurtleBot3.

În funcție de culoarea detectată, robotul inițiază reacții specifice, oferind o gamă variată de experiențe utilizatorului. Atunci când identifică culoarea albastră, robotul se angajează într-un comportament jucăuș, interacționând cu o minge (sferă). Culoarea verde determină robotul să interpreteze o melodie, iar culoarea roșie, declanșează un comportament în care robotul se rotește în cerc, indicând o stare de bucurie.

Robotul TurtleBot3 dispune de facilități esențiale care au contribuit semnificativ la realizarea proiectului nostru. Acestea includ senzori avansați pentru percepție, integrare eficientă cu sistemul de operare ROS pentru control și tehnologia SLAM pentru navigare autonomă în medii necunoscute. Aceste caracteristici au jucat un rol important în implementarea comportamentelor inspirate din celebrul Tamagotchi.



9. BIBLIOGRAFIE

- [www 01] <https://youtube.com/playlist?list=PLLSegLrePWgIbIrA4iehUQ-impvIXdd9Q&si=1LYzdYIT6B8QI6oo>
- [www 02] <http://wiki.ros.org/noetic/Installation/Ubuntu>
- [www 03] <http://wiki.ros.org/turtlebot3>
- [www 04] <https://ro.robotic-tech.com/news/what-is-a-mobile-robot-and-what-are-the-classi-58488195.html>
- [www 05] <https://www.scribd.com/document/370071519/Robot-Mobil-Autonom>
- [www 06] <https://en.wikipedia.org/wiki/Tamagotchi>
- [www 07] <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>

