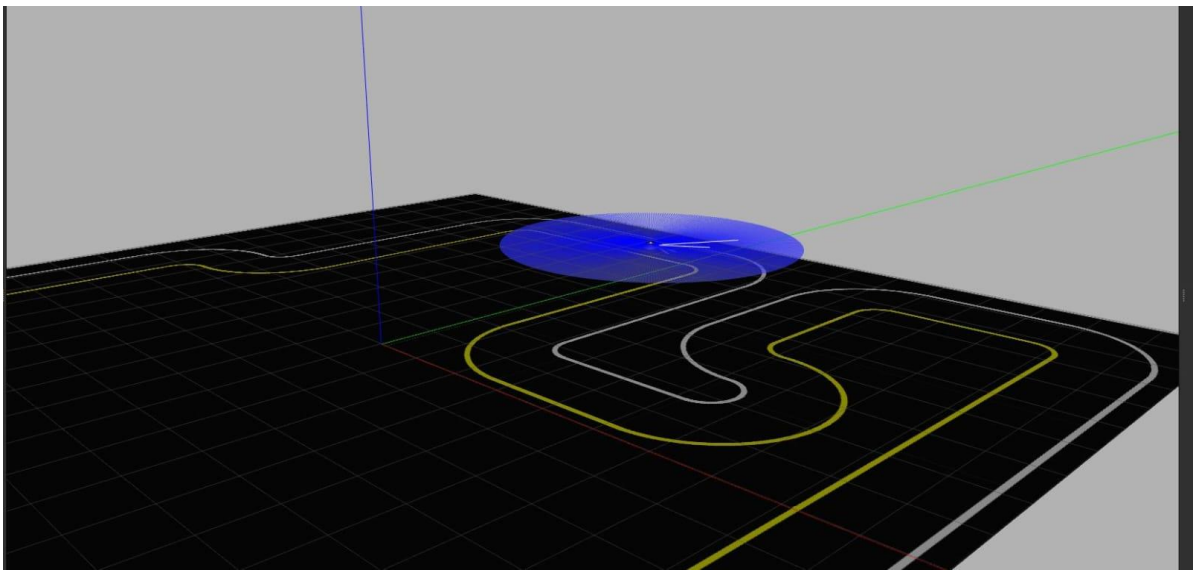
Documentație Proiect Sincretic Autobot

Facultatea de Automatica si Calculatoare Timisoara



Membrii echipa proiect

Ardelean Valentin

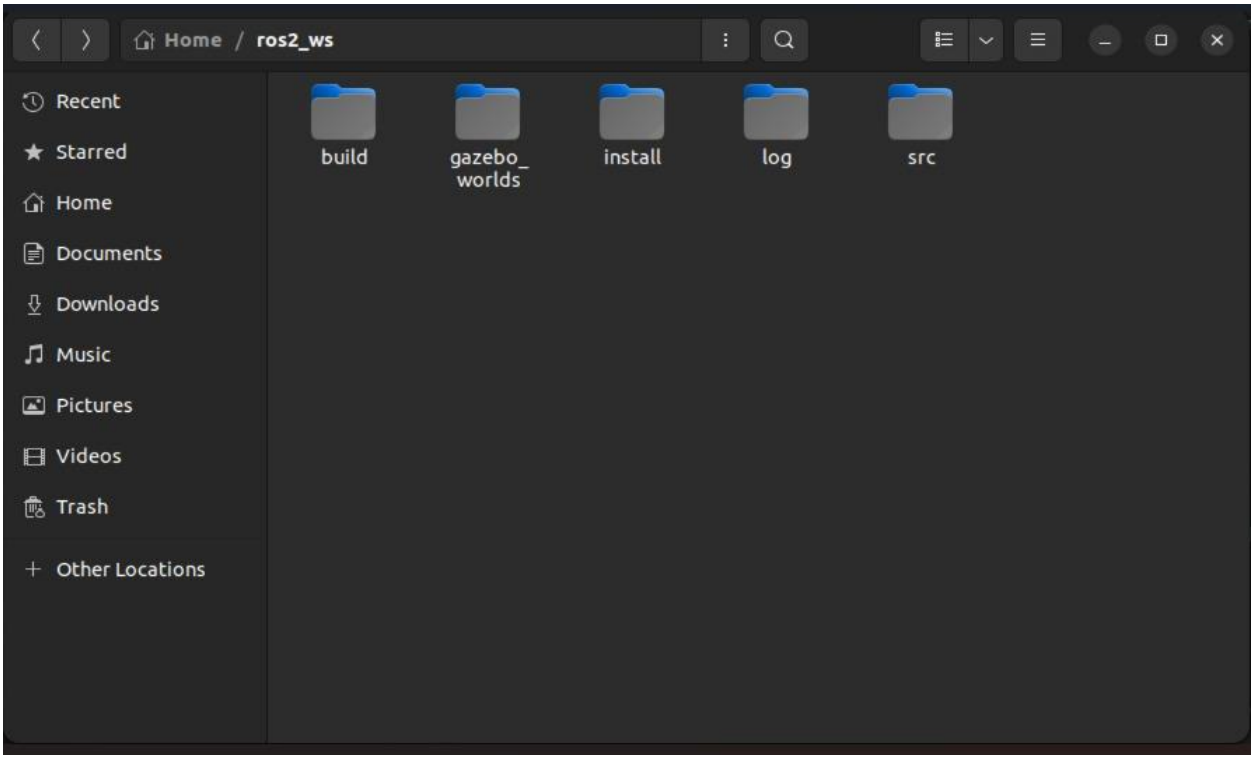
Cioveie David

Ciceu Claudiu

Cuprins

Introducere

Proiectul **Autobot** reprezintă o soluție completă de navigație autonomă pentru un robot mobil de tip Waffle Pi. Acesta folosește tehnologii avansate, precum **Ubuntu**, **ROS2** și **Gazebo**, pentru a simula și controla funcționarea robotului. Scopul proiectului este de a demonstra capabilitățile de navigație, procesare vizuală și control într-un mediu complex.



Tehnologii utilizate

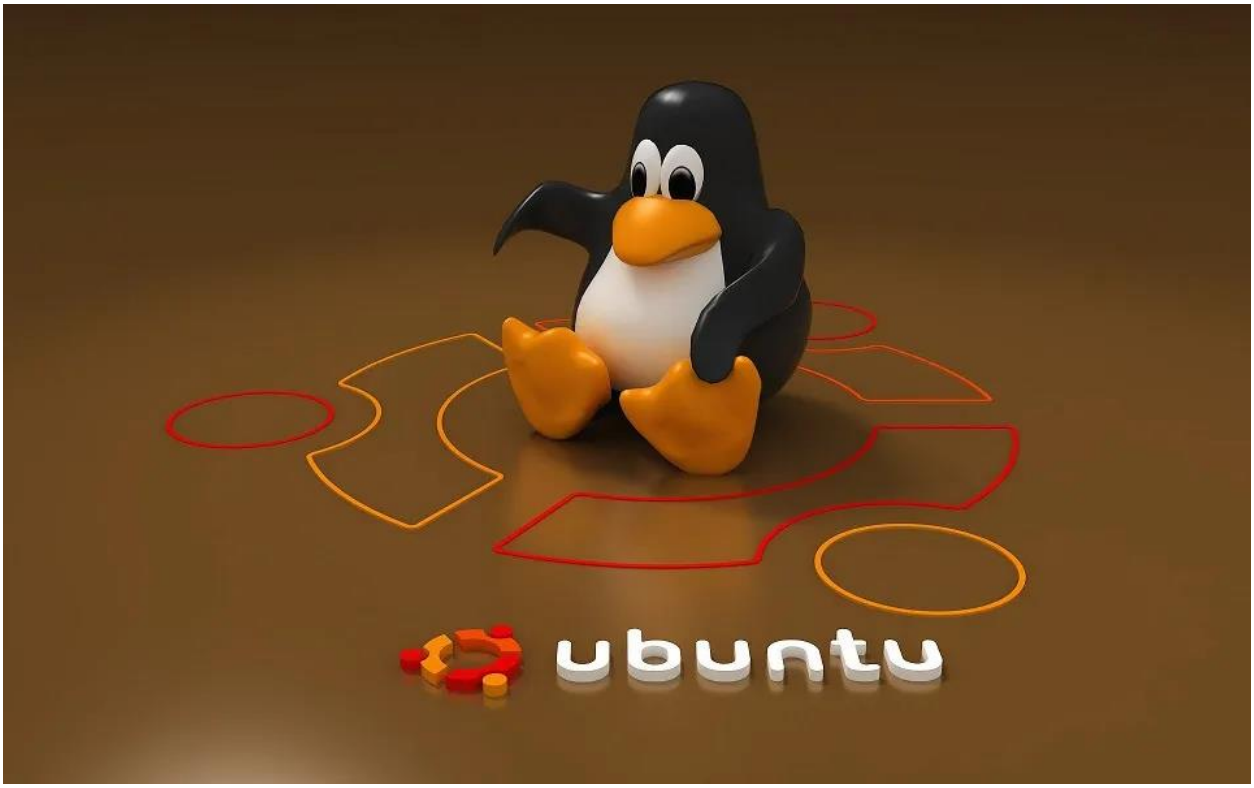
4.1. Ubuntu

Ubuntu este un sistem de operare bazat pe Linux, utilizat pentru dezvoltarea aplicațiilor robotice datorită stabilității, securității și compatibilității excelente cu instrumentele open-source.

Caracteristici cheie:

- Suport pentru instrumente precum ROS2 și Gazebo.
- Management facil al pachetelor prin APT Package Manager.
- Platformă stabilă pentru dezvoltarea aplicațiilor distribuite.

În cadrul proiectului, Ubuntu 20.04 a fost utilizat pentru instalarea și gestionarea ROS2 și a altor pachete necesare.



4.2. ROS2

Robot Operating System 2 (ROS2) este un cadru software care facilitează comunicarea între componentele unui robot. În proiect, ROS2 a fost utilizat pentru controlul senzorilor și motoarelor robotului Waffle Pi.

Caracteristici cheie:

- Comunicare în timp real bazată pe DDS.
- Suport pentru mai multe platforme (Linux, Windows).
- Instrumente pentru simulare și vizualizare.

Prin ROS2, am realizat integrarea senzorilor LIDAR și a camerei RGB-D pentru detectarea și evitarea obstacolelor.

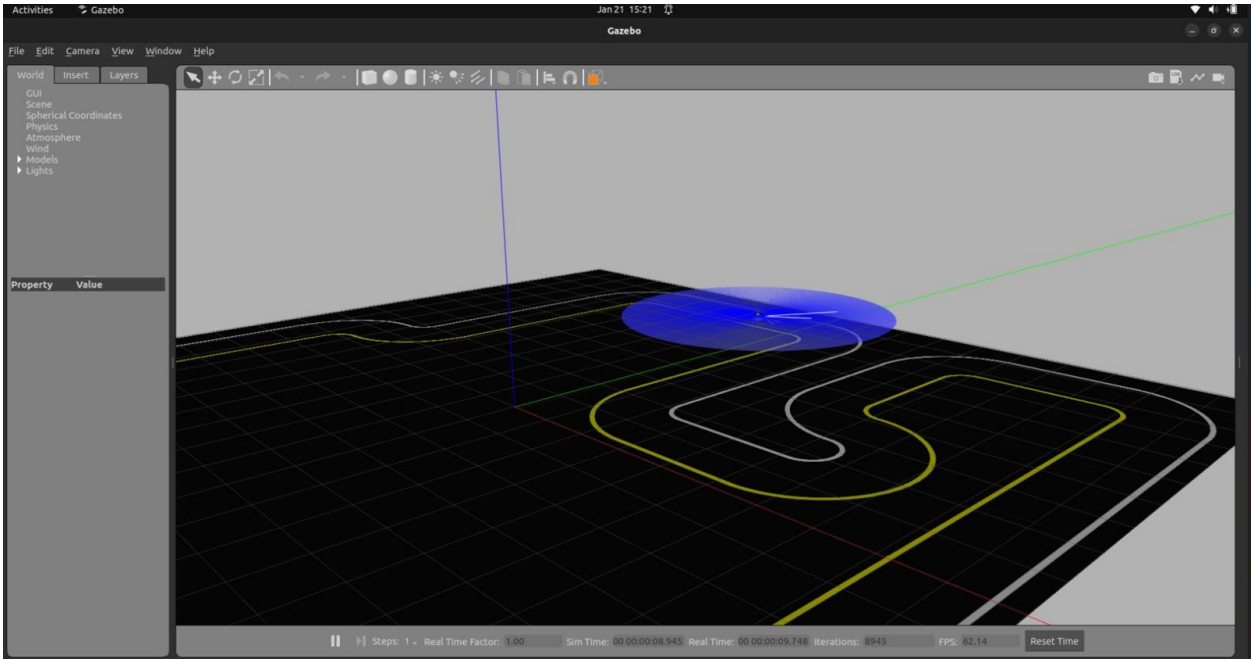
4.3. Gazebo

Gazebo este un simulator puternic utilizat pentru testarea roboților în medii virtuale complexe. Acesta a permis simularea completă a robotului Waffle Pi, inclusiv dinamica senzorilor și interacțiunea cu mediul.

Caracteristici cheie:

- Fizică realistă (gravitație, coliziuni).
- Integrare completă cu ROS2.
- Posibilitatea de a crea medii personalizate.

În proiect, Gazebo a fost utilizat pentru a testa algoritmi de navigație autonomă, folosind mediile virtuale prezentate în imaginile atașate.



5. Robotul Waffle Pi și Camera

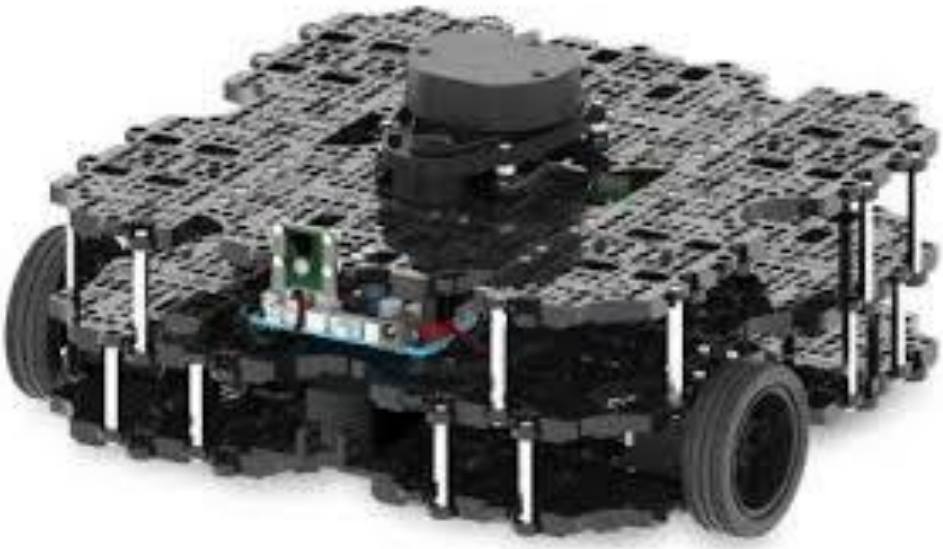
5.1. Descrierea robotului

Robotul Waffle Pi este o platformă mobilă autonomă din seria TurtleBot3. Acesta este echipat cu senzori avansați, o cameră RGB-D și un sistem de control bazat pe Raspberry Pi.

Caracteristici principale:

- Procesor: Raspberry Pi 3 Model B.
- Controler: OpenCR 1.0.
- Senzori: LIDAR, IMU, Cameră RGB-D.
- Motoare DC cu encodere pentru mișcare precisă.

Robotul este proiectat pentru a realiza navigație autonomă și detectare a obstacolelor, utilizând tehnologiile ROS2 și Gazebo.



Pornire camera

```
david@david-Nitro:~$ ros2 launch my_robot_controller free_space_follower.launch.
py
```

Explicații:

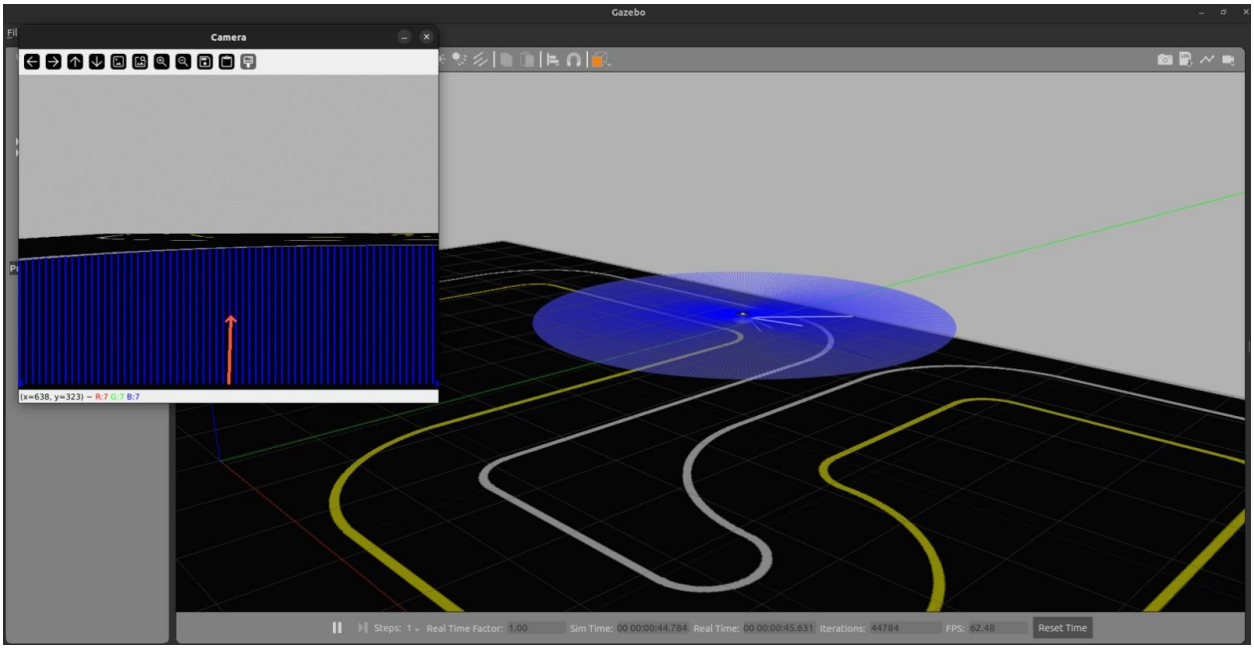
1. **ros2**
    - Este prefixul utilizat pentru a apela comenzile din ROS2 (Robot Operating System 2). Acesta este punctul de pornire pentru toate operațiunile ROS2, cum ar fi lansarea nodurilor, gestionarea topicurilor sau serviciilor.
2. **launch**
    - Este o subcomandă care lansează un fișier de configurare launch. Acest fișier definește unul sau mai multe noduri care trebuie rulate împreună. În cazul de față, free_space_follower.launch.py este fișierul care conține specificațiile nodului ce controlează camera și mișcările robotului.
3. **my_robot_controller**
    - Este numele pachetului ROS2 care conține logica și fișierele necesare pentru funcționarea robotului. Acest pachet include codul sursă pentru camera și alte componente precum senzori sau motoare.
4. **free_space_follower.launch.py**
    - Este fișierul specific de tip launch scris în Python. Acesta configurează nodurile ROS2 necesare pentru a activa funcționalitatea camerei și algoritmii care:
        - Detectează spațiile libere din jurul robotului.
        - Planifică traseele optime pe baza informațiilor primite de la cameră și alte senzori.

Funcționalități oferite de acest nod:

1. **Activarea camerei RGB-D:**

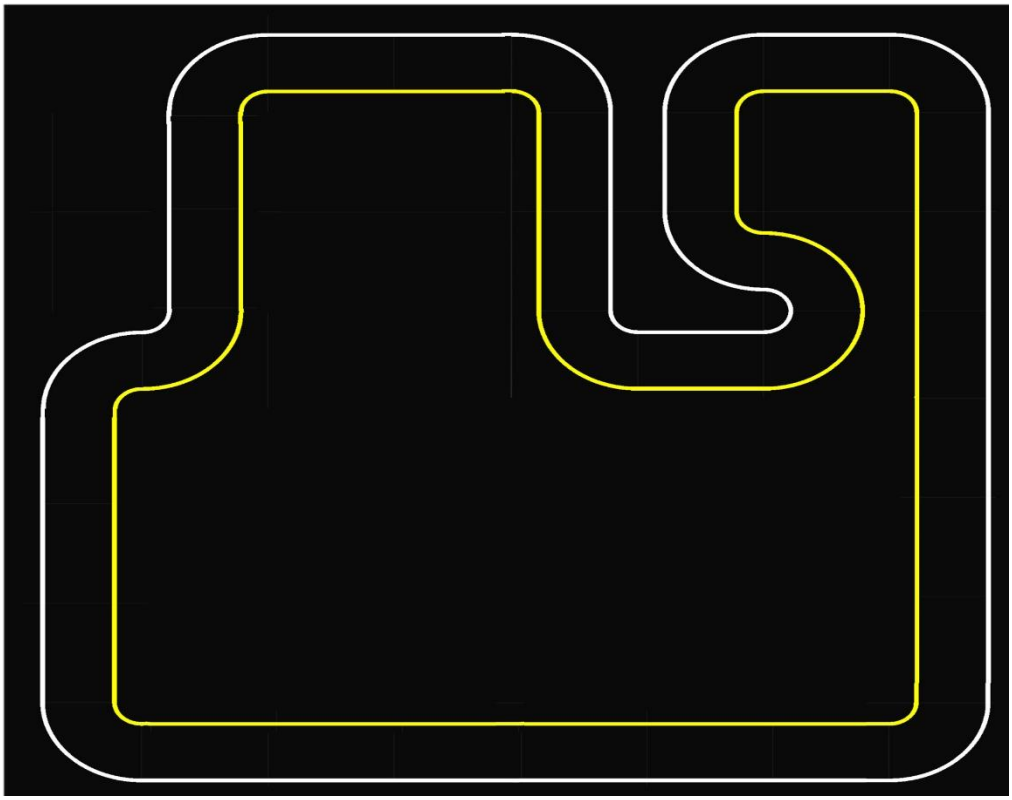
- Camera este pornită și începe să transmită imagini în timp real către nodurile de procesare.
- Datele RGB sunt utilizate pentru analiză vizuală, iar informațiile de adâncime (D) sunt folosite pentru detectarea obstacolelor.

2. **Procesare imagini:**
   - Algoritmul din nodul free_space_follower identifică traseele libere în mediul înconjurător pe baza datelor oferite de cameră.
   - Acest lucru este realizat folosind metode precum detectarea de contururi sau calcularea distanței până la obstacole.
3. **Control al mișcării robotului:**
   - În funcție de datele de la cameră, nodul trimite comenzi motoarelor pentru a direcționa robotul pe traseul liber.



3.3Harta

Mai jos se regaseste o imagine cu harta propriu zisa, harta care este parcursa de robot, acesta o traverseaza urmarind cele doua linii paralele.



Mai jos o sa prezint algoritmul hartii. Acest cod reprezintă un fișier de configurare în format XML pentru o lume virtuală utilizată în simulatorul Gazebo. Acesta definește un plan de fundal (background plane) și elementele vizuale și de coliziune asociate cu acesta. Fișierul este utilizat pentru a crea o hartă goală pe care robotul poate naviga.

Structura codului

1. Versiuni SDF

- Specifică versiunea fișierului XML (1.0) și versiunea SDF (Simulation Description Format, 1.6) utilizată de Gazebo.

2. Definirea lumii

```
xml
CopyEdit
<world name="default">
```

- Creează o lume numită `default`. Aceasta este containerul principal care include toate elementele (lumini, modele, roboți, etc.).

3. Adăugarea unei surse de lumină

<include>

<uri>model://sun</uri>

</include>

- Adaugă o lumină de tip „soare” în lume. Luminile sunt necesare pentru a ilumina scena și a vizualiza obiectele.

4. Crearea planului de fundal (background_plane)

4.1. Definirea modelului

<model name="background_plane">

<static>true</static>

- Creează un model static numit background_plane.
- **<static>true</static>** indică faptul că modelul este fix și nu poate fi mișcat.

4.2. Link-ul pentru plan

<link name="background_link">

- **Definirea unui link numit background_link, care conține informații despre geometria și comportamentul fizic al planului.**

4.3. Coliziunea pentru plan

<collision name="background_collision">

<geometry>

<plane>

<normal>0 0 1</normal>

<size>20 20</size>

</plane>

</geometry>

</collision>

- **Coliziune: Este folosită pentru a defini interacțiunea fizică a planului cu alte obiecte.**
  - **<normal>0 0 1</normal>: Specifică orientarea normală a planului (perpendicular pe axa Z).**
  - **<size>20 20</size>: Dimensiunea planului în metri (20m x 20m).**

4.4. Vizualizarea planului

<visual name="background_visual">

<geometry>

<plane>

<normal>0 0 1</normal>

<size>20 20</size>

</plane>

</geometry>

<material>

<script>

<uri>file://media/materials/scripts</uri>

<name>custom_material</name>

</script>

</material>

</visual>

- **Vizualizare: Definește cum arată planul în simulare. Acesta este diferit de coliziune, care se ocupă doar de partea fizică.**

- **<geometry>: Specifică dimensiunea și orientarea planului.**
- **<material>: Definește aspectul vizual al planului.**
  - **<uri>: Locația fișierelor de materiale utilizate (ex.: texturi, culori).**
  - **<name>: Materialul personalizat care va fi aplicat planului (numit custom_material).**

5. Închiderea modelului și lumii

</link>

</model>

</world>

</sdf>

- **Închide definițiile pentru link, model și lume.**

Rezumat

- **Scopul: Codul creează un plan de fundal static pentru simulare. Acesta are atât componente fizice (coliziuni), cât și componente vizuale (textură).**
- **Dimensiune plan: 20m x 20m.**
- **Utilizare: Robotul va naviga pe acest plan în timpul simulării.**

gazebo_worlds > empty_world.world

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <world name="default">
    <!-- Adaugă lumina și un plan pentru imagine -->
    <include>
      <uri>model://sun</uri>
    </include>

    <!-- Planul de fundal cu coliziune -->
    <model name="background_plane">
      <static>true</static>
      <link name="background_link">
        <!-- Coliziune pentru plan -->
        <collision name="background_collision">
          <geometry>
            <plane>
              <normal>0 0 1</normal>
              <size>20 20</size> <!-- Dimensiunea planului -->
            </plane>
          </geometry>
        </collision>

        <!-- Vizual pentru fundal -->
        <visual name="background_visual">
          <geometry>
            <plane>
              <normal>0 0 1</normal>
              <size>20 20</size> <!-- Dimensiunea planului -->
            </plane>
          </geometry>
          <material>
            <script>
              <uri>file://media/materials/scripts</uri>
              <name>custom_material</name>
            </script>
          </material>
        </visual>
      </link>
    </model>
  </world>
</sdf>
```

3.4Miscare Robot

Acest cod reprezintă o implementare a unui nod ROS2 care controlează mișcarea unui robot autonom pe baza imaginilor capturate de cameră. Codul este împărțit în mai multe funcții care procesează imaginile și iau decizii pentru controlul robotului. Voi explica pe scurt componentele principale:

1. Importuri și inițializare

```
import rclpy
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist
from cv_bridge import CvBridge
import cv2
import numpy as np
```

- Importă module ROS2 și alte biblioteci necesare:
  - **rclpy** pentru funcționalitatea ROS2.
  - **Image** și **Twist** pentru gestionarea datelor de imagine și control al mișcării.
  - **cv2** pentru procesare de imagini folosind OpenCV.
  - **numpy** pentru operații matematice.

2. Clasa FreeSpaceFollower

Această clasă definește nodul ROS2 care procesează imaginile și controlează mișcarea robotului.

Inițializare (__init__)

```
self.image_sub = self.create_subscription(
    Image,
```

'/camera/image_raw',

self.image_callback,

10

)

- **Abonare la topicul /camera/image_raw:** Primește imagini brute de la cameră.
- **Callback:** self.image_callback este apelată pentru fiecare imagine primită.

self.cmd_pub = self.create_publisher(Twist, '/cmd_vel', 10)

- Creează un publisher pentru topicul /cmd_vel, folosit pentru a trimite comenzi motoarelor robotului.

---

3. Funcția image_callback

Această funcție procesează fiecare cadru primit de la cameră:

frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")

- Conversie din mesaj ROS în imagine OpenCV.

red_mask = self.detect_red(resized_frame)

- Detectează culoarea roșie (de exemplu, semne de oprire) în imagine.

stop_detected = self.detect_stop_sign_shape(red_mask, resized_frame)

- Verifică dacă un semn STOP este prezent. Dacă da, robotul se oprește.

if centroid_x is not None:

error = centroid_x - width // 2

twist.linear.x = 0.2

twist.angular.z = -error / 100

- Urmărește o linie sau o cale liberă, ajustând rotația robotului pe baza erorii calculate (diferența între poziția centrului și mijlocul imaginii).

4. Funcția freespace

Această funcție detectează spațiile libere în imagine:

mask = np.zeros((height, width), dtype=np.uint8)

- Creează o mască pentru marcarea spațiilor libere.

contour.append((StangaLim, height - 10))

- Detectează limitele stânga și dreapta ale traseului.

cv2.drawContours(mask, contours, 0, (255), cv2.FILLED)

- Desenează contururile traseului liber.

5. Funcția detect_red

Această funcție identifică regiuni de culoare roșie în imagine:

lower_red1 = np.array([0, 50, 50])

upper_red1 = np.array([10, 255, 255])

- Definește intervalele de culoare pentru detectarea roșului (HSV).

mask1 = cv2.inRange(hsv, lower_red1, upper_red1)

- Creează o mască pentru părțile imaginii care se încadrează în intervalele de culoare roșie.

6. Funcția detect_stop_sign_shape

Această funcție verifică dacă există un semn STOP în imagine:

approx = cv2.approxPolyDP(contour, 0.01 * cv2.arcLength(contour, True), True)

if len(approx) == 8:

- Detectează contururi și verifică dacă sunt octogonale (specific pentru semnul STOP).

7. Funcția main

```
def main(args=None):
    rclpy.init(args=args)
    node = FreeSpaceFollower()
    rclpy.spin(node)
```

- Inițializează nodul ROS2 și rulează procesarea imaginilor.

Rezumat

1. Primește imagini de la cameră prin ROS2.
2. Procesează imaginile folosind OpenCV:
    - Detectează culoarea roșie și semnele STOP.
    - Identifică spațiile libere și traseul de urmat.
3. Controlează robotul, publicând comenzi de mișcare pe topicul /cmd_vel.

src > my_robot_controller > my_robot_controller > free_space_follower.py > FreeSpaceFollower > image_callback

```python
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist
from cv_bridge import CvBridge
import cv2
import numpy as np

class FreeSpaceFollower(Node):
    def __init__(self):
        super().__init__('free_space_follower')
        self.bridge = CvBridge()
        self.image_sub = self.create_subscription(
            Image,
            '/camera/image_raw',  # Topic actualizat pentru Waffle Pi
            self.image_callback,
            10
        )

        self.cmd_pub = self.create_publisher(Twist, '/cmd_vel', 10)  # Topic pentru controlul robotului

    def image_callback(self, msg):
        self.get_logger().info("Imagine primită de la cameră.")
        frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
        resized_frame = cv2.resize(frame, (640, 480))
        blur = cv2.blur(resized_frame, (3, 3))
        edge = cv2.Canny(blur, 100, 200)  # Ajustat pragurile pentru detecția marginilor

        # Detectăm culoarea roșie
        red_mask = self.detect_red(resized_frame)
        cv2.imshow("Red Mask", red_mask)  # Afișăm masca roșie pentru debugging

        # Verificăm dacă există semnul STOP
        stop_detected = self.detect_stop_sign_shape(red_mask, resized_frame)
        if stop_detected:
            self.get_logger().info("Semn STOP detectat! Robotul se oprește.")
            twist = Twist()
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            self.cmd_pub.publish(twist)
            cv2.imshow("Camera Feed", resized_frame)
            cv2.waitKey(1)
            return  # Nu mai procesăm alte comenzi
```

```
        # Continuăm logica pentru urmărirea liniei dacă nu detectăm semnul STOP
        centroid_x, centroid_y, width, height, img_with_arrow = self.freespace(edge, resized_frame)

        if centroid_x is not None:
            error = centroid_x - width // 2
            twist = Twist()
            twist.linear.x = 0.2  # Viteză liniară constantă
            twist.angular.z = -error / 100  # Rotație bazată pe eroare
            self.cmd_pub.publish(twist)
            self.get_logger().info(f"Robotul urmărește linia. Eroare: {error}")
        else:
            twist = Twist()
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            self.cmd_pub.publish(twist)
            self.get_logger().info("Linie pierdută. Robotul s-a oprit.")

        cv2.imshow("Camera Feed", img_with_arrow)
        cv2.waitKey(1)

    def freespace(self, canny_frame, img):
        height, width = canny_frame.shape
        DreaptaLim = width // 2
        StangaLim = width // 2
        mask = np.zeros((height, width), dtype=np.uint8)
        contour = []

        for i in range(width // 2, width - 1):
            if canny_frame[height - 10, i]:
                DreaptaLim = i
                break
        for i in range(width // 2):
            if canny_frame[height - 10, width // 2 - i]:
                StangaLim = width // 2 - i
                break
        if StangaLim == width // 2:
            StangaLim = 1
        if DreaptaLim == width // 2:
            DreaptaLim = width
        contour.append((StangaLim, height - 10))
        cv2.circle(img, (StangaLim, height - 10), 5, (255), -1)
        cv2.circle(img, (DreaptaLim, height - 10), 5, (255), -1)
```

```
87
88          for j in range(StangaLim, DreaptaLim - 1, 10):
89              for i in range(height - 10, 9, -1):
90                  if canny_frame[i, j]:
91                      cv2.line(img, (j, height - 10), (j, i), (255), 2)
92                      contour.append((j, i))
93                      break
94                  if i == 10:
95                      contour.append((j, i))
96                      cv2.line(img, (j, height - 10), (j, i), (255), 2)
97          contour.append((DreaptaLim, height - 10))
98          contours = [np.array(contour)]
99          cv2.drawContours(mask, contours, 0, (255), cv2.FILLED)
100
101         M = cv2.moments(contours[0])
102         if M["m00"] != 0:
103             centroid_x = int(M["m10"] / M["m00"])
104             centroid_y = int(M["m01"] / M["m00"])
105             cv2.arrowedLine(img, (width // 2, height - 10), (centroid_x, centroid_y), (60, 90, 255), 4)
106             return centroid_x, centroid_y, width, height, img
107         else:
108             return None, None, width, height, img
109
110     def detect_red(self, frame):
111         hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
112
113         # Interval pentru culoarea roșie
114         lower_red1 = np.array([0, 50, 50])     # Prag ajustat pentru roșu deschis
115         upper_red1 = np.array([10, 255, 255])
116
117         lower_red2 = np.array([170, 50, 50])   # Prag ajustat pentru roșu închis
118         upper_red2 = np.array([180, 255, 255])
119
120         mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
121         mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
122
123         return mask1 + mask2
```

```
125     def detect_stop_sign_shape(self, mask, frame):
126         contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
127         for contour in contours:
128             approx = cv2.approxPolyDP(contour, 0.01 * cv2.arcLength(contour, True), True)
129             if len(approx) == 8:  # Forma octogonală
130                 x, y, w, h = cv2.boundingRect(approx)
131                 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)  # Evidențiază semnul STOP
132                 self.get_logger().info("Semn STOP detectat în imagine!")
133                 return True
134         return False
135
136
137 def main(args=None):
138     rclpy.init(args=args)
139     node = FreeSpaceFollower()
140     rclpy.spin(node)
141     node.destroy_node()
142     rclpy.shutdown()
143
144
145 if __name__ == '__main__':
146     main()
```

Mai jos o sa atasez o fotografie cu comanda care porneste robotul si harta in simulatorul gazebo.

```
david@david-Nitro:~$ export TURTLEBOT3_MODEL=waffle_pi
david@david-Nitro:~$ ros2 launch turtlebot3_gazebo empty_world.launch.py
```

4.Bibliografie

https://chatgpt.com/

https://www.youtube.com/

https://github.com/