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

Hide and Seek

PROIECT SINCRETIC I

AUTORI:
Bran Eduard-Denis,
Bratu Loredana-Mădălina,
Cocorăscu Andreea-Roxana

COORDONATORI:
Conf.dr.ing. Florin Drăgan
As.ing. Emil Voișan

Anul III AIA - an universitar 2024/2025

CUPRINS

1. INTRODUCERE

Roboții mobili reprezintă un domeniu dinamic și inovator în cadrul roboticii, având aplicabilitate într-o gamă variată de sectoare, de la industrie și logistică, până la explorare spațială și asistență personală. Aceștia sunt capabili să se deplaseze autonom sau semiautonom în medii diverse, utilizând tehnologii avansate precum senzori, algoritmi de navigație și inteligență artificială.

Un aspect esențial al utilizării roboților mobili este conducerea la distanță, care permite operatorilor să controleze sau să monitorizeze funcționarea acestora dintr-o locație remote. Această tehnologie este extrem de utilă în situații periculoase, în medii inaccesibile sau în cazul învățării automate a roboților prin intervenție umană minimă. Teleoperația roboților mobili implică utilizarea unor interfețe avansate și protocoale de comunicații eficiente pentru a asigura un control precis și fiabil. Dezvoltarea unor sisteme integrate pentru roboții mobili combină elemente de hardware, software și inginerie de sistem, ceea ce încearcă să rezolve provocări precum sincronizarea, rezistența la interferențe și reacția în timp real la factorii de mediu.
Proiectul nostru se concentrează pe aplicarea acestor principii pentru a crea un sistem de teleoperație a unui robot mobil, utilizând platforma TurtleBot3 și instrumentele ROS 2. Prin integrarea tehnologiilor de detecție a culorilor și de navigație autonomă, scopul acestui proiect este de a demonstra aplicabilitatea și beneficiile roboților mobili în scenarii practice.

2. PREZENTAREA TEMEI

Tema proiectului nostru este **"Hide and Seek"**, o aplicație care simulează un joc de ascunselea între un robot mobil și obiecte marcate cu culori specifice. Scopul proiectului este de a dezvolta un sistem care permite robotului TurtleBot3 să detecteze obiectele de culoare verde, și să se ascundă de ele după o cutie, cu coordonate fixe, în timp ce identifică și interacționează cu obiectele de culoare roșie, care reprezintă punctele de finalizare a sarcinii.

Proiectul implică:

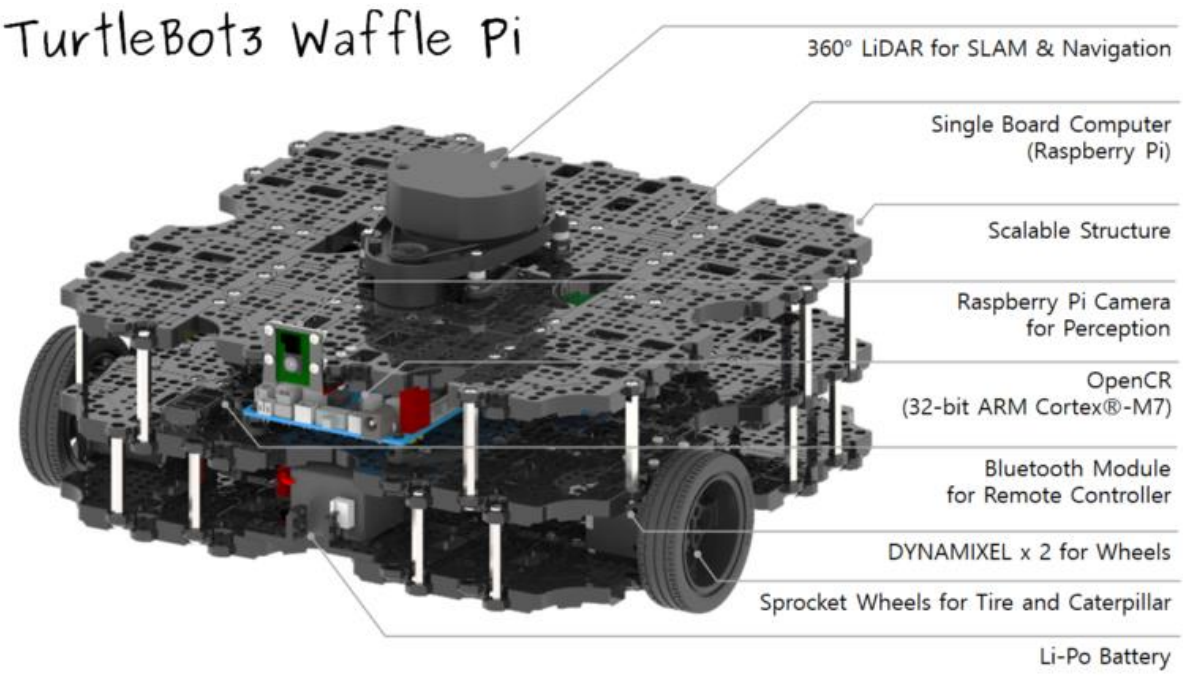
- Implementarea unui algoritm de detecție a culorilor folosind camere și senzori.
- Dezvoltarea unui mecanism de deplasare autonomă pentru a asigura navigația eficientă în zona de joc.
- Stabilirea unui comportament specific pentru robot în funcție de culorile detectate:
    - La detectarea unei cutii roșii, robotul este "găsit", urmând o rotire infinită.
    - La detectarea unei cutii verzi, robotul se deplasează aproximativ 6 cm într-o anumită direcție, pentru a evita obiectul și a încerca să se "ascundă".
- Integrarea teleoperației pentru a monitoriza și ajusta comportamentul robotului în timp real.

Această temă permite explorarea unor concepte cheie precum detecția și procesarea imaginilor, navigația autonomă, și colaborarea între modulele hardware și software. Prin implementarea sa, proiectul demonstrează utilitatea roboților mobili în medii dinamice și situații similare celor din lumea reală.

3. TEHNOLOGII UTILIZATE

Pentru realizarea proiectului "Hide and Seek", au fost utilizate următoarele tehnologii:

- Robotul TurtleBot3: TurtleBot3 este o platformă robotică mobilă flexibilă și accesibilă, dotată cu senzori, cameră și mecanisme de control avansate. Aceasta este utilizată pentru implementarea algoritmilor de navigație autonomă și detecție a culorilor.



[1]

TurtleBot3 dispune de următoarele componente:

a) Unitatea centrală de procesare

- Raspberry Pi 3/4: Placă de dezvoltare utilizată ca unitate centrală de procesare (CPU), rulează ROS2 și gestionează comunicațiile și procesele software.

b) Senzori:

- Lidar (Light Detection and Ranging): Senzor LDS-01 pentru mapare și evitarea obstacolelor. Acesta permite TurtleBot3 să detecteze mediul înconjurător în 360 de grade.
- IMU (Inertial Measurement Unit): Modul cu accelerometru și giroscop, utilizat pentru măsurarea orientării și stabilității.
- Senzor de culoare: Poate fi adăugat pentru detectarea culorilor.

c) Actuatori:

- Motoare DC cu codificatoare (encoders): Permite controlul precis al vitezei și poziției roților. Motoarele sunt integrate în modulul motor Dynamixel XM430.
- Roți omnidirecționale: Două roți principale pentru mișcare și o roată sferică pasivă pentru echilibru.

d) Sursa de alimentare:

- Baterie Li-ion (11.1V, 1800mAh): Oferă energie pentru toate componentele robotului, inclusiv unitatea centrală, senzori și actuatori.

e) Placa de control:

- OpenCR (Open Control Robotics): Placă de control dedicată care gestionează comunicațiile dintre senzori, motoare și unitatea centrală. Este compatibilă cu Arduino IDE pentru personalizare avansată.

f) Camera:

- Raspberry Pi Camera sau RealSense Camera: Utilizată pentru procesare de imagini, detectarea obiectelor și navigație vizuală.

g) Conectivitate:

- Wi-Fi și Bluetooth: Permite controlul la distanță și comunicarea între robot și stațiile de lucru.
- USB și GPIO (General Purpose Input/Output): Pentru extinderea funcționalităților cu module suplimentare.

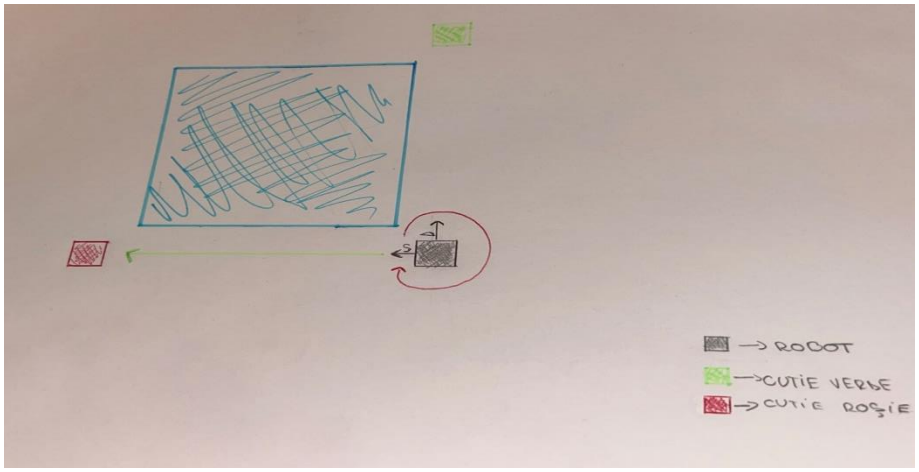
- ROS2 (Robot Operating System 2): ROS 2 este un sistem de operare dedicat roboticii, care oferă un set extins de instrumente pentru dezvoltarea, simularea și implementarea aplicațiilor robotice. Este utilizat pentru a crea noduri care gestionează funcții specifice, cum ar fi detecția culorilor, controlul mișcării și comunicarea între module.

- Gazebo: Gazebo este un mediu de simulare avansat, folosit pentru a testa și valida comportamentul robotului într-un mediu virtual controlat, replicând scenariile din lumea reală.
- Visual Studio Code este un editor de cod sursă dezvoltat de Microsoft, cunoscut pentru flexibilitatea, performanța și suportul extins pentru diferite limbaje de programare. Este utilizat pe scară largă de dezvoltatori, atât pentru proiecte personale, cât și pentru aplicații complexe.
- Python**:** Limbajul de programare Python a fost utilizat pentru implementarea algoritmilor de procesare a imaginilor, control al mișcării robotului și integrare a modulelor software.

Prin utilizarea acestor tehnologii, proiectul propune o soluție interesantă și eficientă pentru crearea unui sistem robotic capabil să se adapteze sarcinilor dinamice.

4. GHIDUL PROGRAMATORULUI

Proiectul ”Hide and Seek” a fost implementat folosind diferite moduri de lucru, care includ următoarele etape:



- În urma unei scheme bloc, concepută pe o foaie A4, am decis cum dorim să arate și să funcționeze proiectul.
- După instalarea tuturor celor necesare, precum sistemul de operare Ubuntu, Gazebo, Visual Studio Code și pachetele necesare unei bune utilizări, am început

să implementăm o logică pentru dezvoltarea codului. Limbajul de programare utilizat este Python, l-am ales deoarece are o sintaxă simplă și ușor de învățat, în plus este un limbaj de programare ușor de citit atât pentru cunoscători, cât și pentru persoanele mai puțin familiarizate cu acest domeniu. În plus, pe lângă cele menționate, python prezintă o multitudine de biblioteci, cu funcționalități avansate de procesare a datelor.

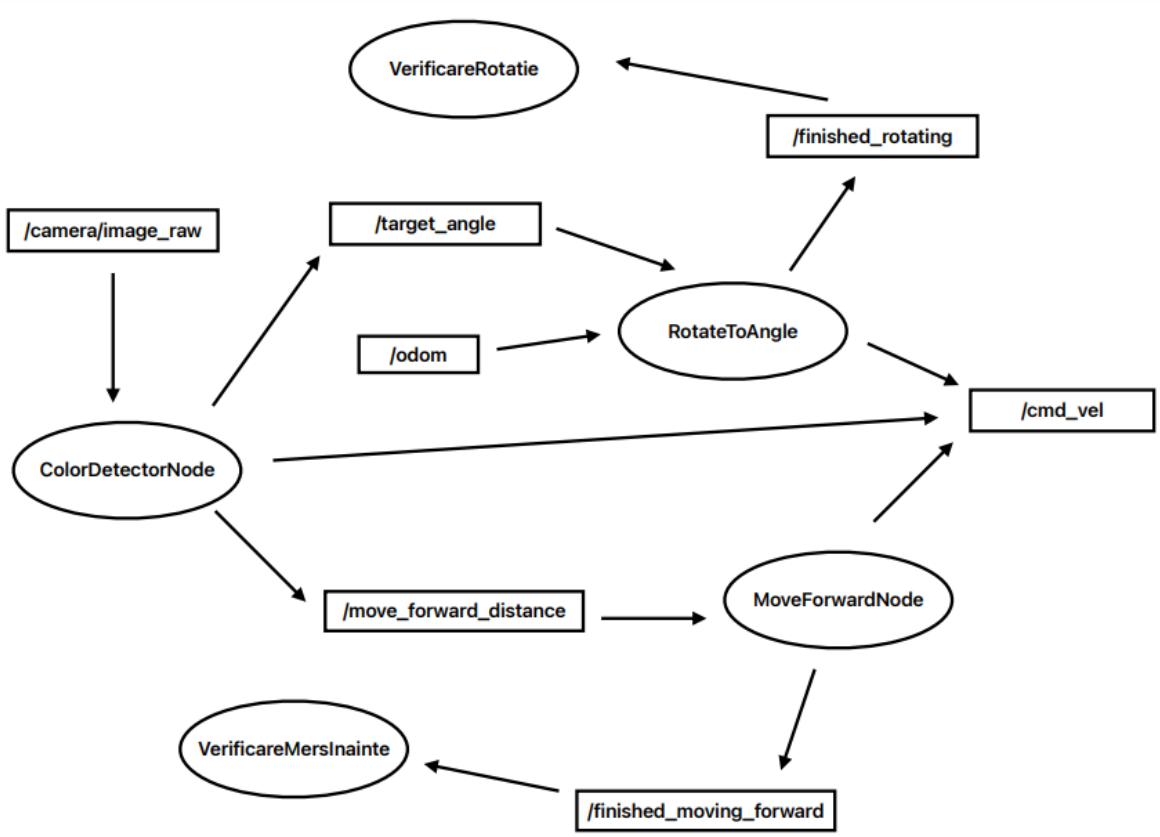
- În urma mai multor discuții și planuri strategice, am ajuns la a crea trei noduri: MoveForward, Rotație și Culori.

Pentru deplasarea în jurul cutiei robotul trebuie să știe în permanență în care colț se află pentru a cunoaște la ce unghiuri trebuie să se uite și în ce direcții să meargă. De asemenea, trebuie să știm pe care latură a cubului se uită în fiecare moment pentru ca atunci când detectează culoarea verde să știe pe care din cele 2 posibile rute să o aleagă. Pentru facilitarea acestui lucru sunt utilizate 2 enumerații:

```
#Enum pentru pozitiile fiecarui colt al cubului
class Pozitie(Enum):
    DREAPTA_JOS=1
    DREAPTA_SUS=2
    STANGA_SUS=3
    STANGA_JOS=4
```

```
#Enum pentru directia in care se uita robot la un anumit moment
class Directie(Enum):
    STANGA=1
    DREAPTA=2
```

Pentru implementarea acestui proiect au fost proiectate 5 noduri:



1) ColorDetectorNode:

- Utilizat pentru detecția culorilor verzi sau roșii
- Are rolul de nod de comanda
- In funcție de culorile detectate ia anumite decizii și publică mesaje către alte noduri subordonate

Acesta este subscriber la următoarele topicuri:

```
# Subscriber la topicul care contine imaginea captata de robot
self.image_sub = self.create_subscription(
    Image,
    '/camera/image_raw',
    self.image_callback,
    10
)
```

Acesta este publisher la următoarele topicuri:

```
# Publisher pentru a trimite nodului de rotire unghiul la care sa o faca
self.angle_pub = self.create_publisher(Float32, '/target_angle', 10)

#Publisher pentru a trimite nodului de mers inainte distanta dorita
self.move_forward_pub=self.create_publisher(Float32,'/move_forward_distance',10)

#Publisher pentru a trimite topicului de control al robotului pentru al roti infinit
self.rotate_infinitely_pub = self.create_publisher(Twist, '/cmd_vel', 10)
```

Atunci când robotul publică o imagine se apelează funcția de callback de la subscriber și se începe detecția culorii din imaginea primită apelând funcția detect_colors.

```
def image_callback(self, msg):
    # conversie din imagine ROS in imagine de tip OpenCV
    cv_image = self.bridge.imgmsg_to_cv2(msg, "bgr8")

    self.detect_colors(cv_image)
```

În interiorul funcției detect_colors se realizează detectarea culorii, iar pe baza răspunsului, în cazul în care e verde, roșu sau niciuna dintre acestea se iau anumite decizii.

Pentru detectarea culorii:

```
# Conversia imaginii in spatiul de culori HSV
hsv_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2HSV)

# Definirea limitelor de culori in spatiul HSV (Verde si Rosu)
green_lower = np.array([40, 40, 40])  # Limita inferioara pentru verde
green_upper = np.array([80, 255, 255])  # Limita superioara pentru verde
red_lower1 = np.array([0, 120, 70])  # Limita inferioara pentru rosu (primul interval)
red_upper1 = np.array([10, 255, 255])  # ULimita superiora pentru rosu (primul interval)
red_lower2 = np.array([170, 120, 70])  # Limita inferioara pentru rosu (al doilea interval)
red_upper2 = np.array([180, 255, 255])  # Limita superioara pentru rosu (al doilea interval)

#Crearea mastilor pentru detectia culorii verzi si rosii
green_mask = cv2.inRange(hsv_image, green_lower, green_upper)
red_mask1 = cv2.inRange(hsv_image, red_lower1, red_upper1)
red_mask2 = cv2.inRange(hsv_image, red_lower2, red_upper2)

# Combinarea celor 2 masti pentru detectia rosului
red_mask = cv2.bitwise_or(red_mask1, red_mask2)
```

În funcție de culoarea detectată se realizează diferite operații:

- Verde:

  În funcție de colțul în care se află și direcția în care se uită atunci când a detectat verdele se alege unghiul la care să se deplaseze robotul. Pentru rotirea și deplasarea robotului se construiesc și trimit mesaje către nodurile MoveForwardNode și RotateToAngleNode pentru fiecare caz particular. Se asigură îndeplinirea sarcinilor prin blocarea programului în a aștepta confirmarea reușitei acestora prin nodurile VerificareRotatie și VerificareMersInainte.

```
# Diferitele moduri de operare in functie de culoarea detectata
if np.any(green_mask):
    self.get_logger().info("verde")

    if self.pozitie==Pozitie.DREAPTA_JOS:
        if self.directie==Directie.DREAPTA:

            rotatie_msg=Float32()
            rotatie_msg.data=90.

            global ok_rotatie
            ok_rotatie=0
            self.angle_pub.publish(rotatie_msg)
            while ok_rotatie==0:
                self.get_logger().info("Se roteste!")


            move_msg=Float32()
            move_msg.data=6.0477

            global ok_mers
            ok_mers=0
            self.move_forward_pub.publish(move_msg)
            while ok_mers==0:
                self.get_logger().info("Merge inainte!")
            self.pozitie=Pozitie.STANGA_JOS

        else:
            rotatie_msg=Float32()
            rotatie_msg.data= 0.

            ok_rotatie=0
            self.angle_pub.publish(rotatie_msg)
            while ok_rotatie==0:
                self.get_logger().info("Se roteste!")
            self.directie=Directie.DREAPTA


            move_msg=Float32()
            move_msg.data=6.0477

            ok_mers=0
            self.move_forward_pub.publish(move_msg)
            while ok_mers==0:
                self.get_logger().info("Merge inainte!")
            self.pozitie=Pozitie.DREAPTA_SUS
```

- Roșu:

  Robotul se rotește rapid într-un punct în mod infinit

```
elif np.any(red_mask):
    self.get_logger().info("rosu")
    while 1:
        self.rotatie = Twist()
        self.rotatie.linear.x = 0.0
        self.rotatie.angular.z = 2.  # Ajustare viteza de rotire(rad/s)
        self.rotate_infinitely_pub.publish(self.rotatie)
```

- Niciuna dintre acestea:

Robotul se uită în mod constant de la stânga la dreapta, unghiurile la care se uita fiind diferite în funcție de colțul la care se află robotul.

```
else:
    self.get_logger().info("Nu s-a detectat verde sau rosu")

    if self.pozitie==Pozitie.DREAPTA_JOS:
        if self.directie==Directie.DREAPTA:

            rotatie_msg=Float32()
            rotatie_msg.data=90.

            ok_rotatie=0
            self.angle_pub.publish(rotatie_msg)
            while ok_rotatie==0:
                self.get_logger().info("Se roteste!")
            self.directie=Directie.STANGA

        else:
            rotatie_msg=Float32()
            rotatie_msg.data=0.

            ok_rotatie=0
            self.angle_pub.publish(rotatie_msg)
            while ok_rotatie==0:
                self.get_logger().info("Se roteste!")
            self.directie=Directie.DREAPTA
```

2) MoveForwardNode:

- Utilizat pentru comanda robotului de a merge înainte pentru o anumită distanță
- Distanța este preluată din topicul „/move_forward_distance”, fiind publicată de ColorDetectorNode

- Anunță nodul VerificareMersInainte atunci când sarcina a fost finalizată publicând în topicul „/finished_moving_forward”

Acesta este subscriber la următoarele topicuri:

```
# Subscriber la topicul care contine distanta care trebuie strabatuta
self.subscription = self.create_subscription(
    Float32,
    '/move_forward_distance',
    self.distance_callback,
    10
)
```

Acesta este publisher la următoarele topicuri:

```
# Publisher la topicul care controleaza miscarea robotului
self.velocity_publisher = self.create_publisher(Twist, '/cmd_vel', 10)

#Publisher pentru anuntarea finalizarii deplasarii robotului
self.finished_publisher=self.create_publisher(String,'/finished_moving_forward',10)
```

Atunci când nodul ColorDetectorNode publică distanța pe topicul „/move_forward_distance” se apelează funcția de callback „distance_callback” și se începe acțiunea de mers înainte a robotului apelând funcția „move_forward”

```
def distance_callback(self, msg):
    distance = msg.data
    self.get_logger().info(f'Distanta primita: {distance}')

    self.move_forward(distance)
```

```
def move_forward(self, distance):

    velocity_msg = Twist()
    # Setare viteza liniara
    velocity_msg.linear.x = 0.5  # Ajustare viteza
    velocity_msg.angular.z = 0.0  # Sa nu existe rotatie

    # Calcularea duratei necesare de mers inainte cu o anumita viteza
    duration = distance / velocity_msg.linear.x

    self.get_logger().info(f'Merge inainte pentru {duration:.2f} secunde.')

    # Publicare comenzi de viteza pentru timpul calculat
    start_time = self.get_clock().now()
    while (self.get_clock().now() - start_time).nanoseconds < (duration * 1e9):
        self.velocity_publisher.publish(velocity_msg)


    # Oprirea robotului
    velocity_msg.linear.x = 0.0

    for _ in range(10):  # Trimitem mesajul de stop de mai multe ori pentru a asigura oprirea
        self.velocity_publisher.publish(velocity_msg)
        self.get_logger().info('Oprire robot.')

    self.get_logger().info('Robot oprit.')
    finished_msg=String()
    finished_msg.data='gata'
    self.finished_publisher.publish(finished_msg)
```

3) VerificareMersInainte:

- Atunci când nodul MoveForwardNode publică faptul că a terminat acțiunea, nodul VerificareMersInainte modifică o variabilă la care are acces și nodul ColorDetectorNode, acesta netrecând mai departe la executarea următoarelor acțiuni până la modificarea ei.

```python
#Nod folosit pentru semnalarea nodului principal atunci cand
# deplasarea inainte pentru o anumita distanta a fost finalizata
class VerificareMersInainte(Node):
    def __init__(self):
        super().__init__('mers_inainte')
        my_callback_group = ReentrantCallbackGroup()
        #Subscriber pentru semnalarea finalizarii mersului inainte
        self.finished_moving_sub=self.create_subscription(
            String,
            '/finished_moving_forward',
            self.finished_moving_callback,
            10,
            callback_group=my_callback_group
        )
    def finished_moving_callback(self,msg):
        finalizare_mers=msg.data
        if(finalizare_mers=="gata"):
            global ok_mers
            ok_mers=1
            self.get_logger().info("mers inainte din if")
        self.get_logger().info("call de la mers inainte")
```

4) RotateToAngleNode:

- Utilizat pentru comanda robotului de a se roti la un anumit unghi
- Unghiul este preluat din topicul „/target_angle”, fiind publicat de ColorDetectorNode
- Anunță nodul VerificareRotatie atunci când sarcina a fost finalizată publicând în topicul „/finished_rotating”

Acesta este subscriber la următoarele topicuri:

```python
#Subscriptie la topicul de odom a robotului pentru a citi dinamica lui
self.odom_subscriber = self.create_subscription(Odometry, '/odom', self.odom_callback, 10)

#Subscriptie la topicul care contine unghiul la care trebuie sa fie rotit robotul
self.rotation_subscriber = self.create_subscription(Float32, '/target_angle', self.target_angle_callback, 10)
```

Acesta este publisher la următoarele topicuri:

```
# Publisher pentru trimiterea comenzilor de miscare catre robot
self.cmd_vel_publisher = self.create_publisher(Twist, '/cmd_vel', 10)

#Publisher pentru a anunta finalizarea rotirii
self.finished_publisher=self.create_publisher(String,'/finished_rotating',10)
```

Atunci când nodul ColorDetectorNode publică unghiul (în grade) pe topicul „/target_angle” se apelează funcția de callback „target_angle_callback” care la rândul ei apelează funcția „rotate_to_angle” care convertește unghiul din grade în radiani.

```
def target_angle_callback(self, msg):
    """Callback pentru obtinerea unghiului de la subscriber"""
    target_angle_deg = msg.data
    self.rotate_to_angle(target_angle_deg)
```

```
def rotate_to_angle(self, target_angle_deg):
    """Setarea unghiului de rotatie (in grade) la care sa se roteasca robotul"""
    self.target_yaw = math.radians(target_angle_deg)
    self.get_logger().info(f"Rotirea spre {target_angle_deg} de grade...")
```

Pentru obținerea poziției robotului se utilizează callbackul topicului „/odom”, iar unghiul este calculat prin conversia coordonatelor din quaternion în euler.

```
def odom_callback(self, msg):
    """Callback pentru actualizarea axei de rotire din odometru"""
    orientation_q = msg.pose.pose.orientation
    quaternion = (orientation_q.x, orientation_q.y, orientation_q.z, orientation_q.w)
    _, _, self.current_yaw = euler_from_quaternion(quaternion)
```

```python
def euler_from_quaternion(quaternion):
    """
    Convert quaternion (x, y, z, w) to Euler angles (roll, pitch, yaw).
    """
    x, y, z, w = quaternion
    t0 = +2.0 * (w * x + y * z)
    t1 = +1.0 - 2.0 * (x * x + y * y)
    roll = math.atan2(t0, t1)

    t2 = +2.0 * (w * y - z * x)
    t2 = +1.0 if t2 > +1.0 else t2
    t2 = -1.0 if t2 < -1.0 else t2
    pitch = math.asin(t2)

    t3 = +2.0 * (w * z + x * y)
    t4 = +1.0 - 2.0 * (y * y + z * z)
    yaw = math.atan2(t3, t4)

    return roll, pitch, yaw
```

Nodul execută funcția lui principală de rotire a robotului prin apelul repetat al funcției „control_loop” datorită timerului setat.

```python
# Timer pentru apelul regulat al loopului de control
self.timer = self.create_timer(0.1, self.control_loop)
```

```
def control_loop(self):
    """Loopul de control pentru rotirea robotului"""
    if self.target_yaw is None:
        # Fa nimic daca nu e setat niciun unghi de rotire
        return

    #Calculeaza cea mai mica diferenta de unghi pana la unghiul dorit
    angle_diff = self.normalize_angle(self.target_yaw - self.current_yaw)

    # Verifica daca ne aflam in limitele de toleranta pentru a finaliza rotatia
    if abs(angle_diff) <= self.angle_tolerance:

        self.stop_robot()
        self.duration=2
        start_time = self.get_clock().now()
        while (self.get_clock().now() - start_time).nanoseconds < (self.duration * 1e9):
            self.get_logger().info("Rotire robot pentru a atinge unghiul")
        self.get_logger().info("Unghiul dorit atins!")
        finished_msg=String()
        finished_msg.data='gata'
        self.finished_publisher.publish(finished_msg)
        self.target_yaw = None  # reseteaza unghiul pentru un nou apel
        return

    # Determinare directie de rotire
    twist = Twist()
    twist.angular.z = self.angular_speed if angle_diff > 0 else -self.angular_speed
    self.cmd_vel_publisher.publish(twist)
```

5) VerificareRotatie

- Atunci când nodul RotateToAngleNode publică faptul că a terminat acțiunea, acest nod modifică o variabilă la care are acces și nodul ColorDetectorNode, acesta netrecând mai departe la executarea următoarelor acțiuni până la modificarea ei.

```python
#Nod folosit pentru semnalarea nodului principal atunci cand o rotire la un anumit unghi a fost finalizata
class VerificareRotatie(Node):
    def __init__(self):
        super().__init__('rotatie')

        my_callback_group = ReentrantCallbackGroup()

        #Subscriber pentru semnalarea finalizarii rotirii la un anumit unghi
        self.finished_rotating_sub=self.create_subscription(
            String,
            '/finished_rotating',
            self.finished_rotating_callback,
            10,
            callback_group=my_callback_group
        )

    def finished_rotating_callback(self,msg):
        finalizare_rotatie=msg.data
        if(finalizare_rotatie=="gata"):
            global ok_rotatie
            ok_rotatie=1
            self.get_logger().info("rotatia din if")
        self.get_logger().info("call de la rotatie")
```

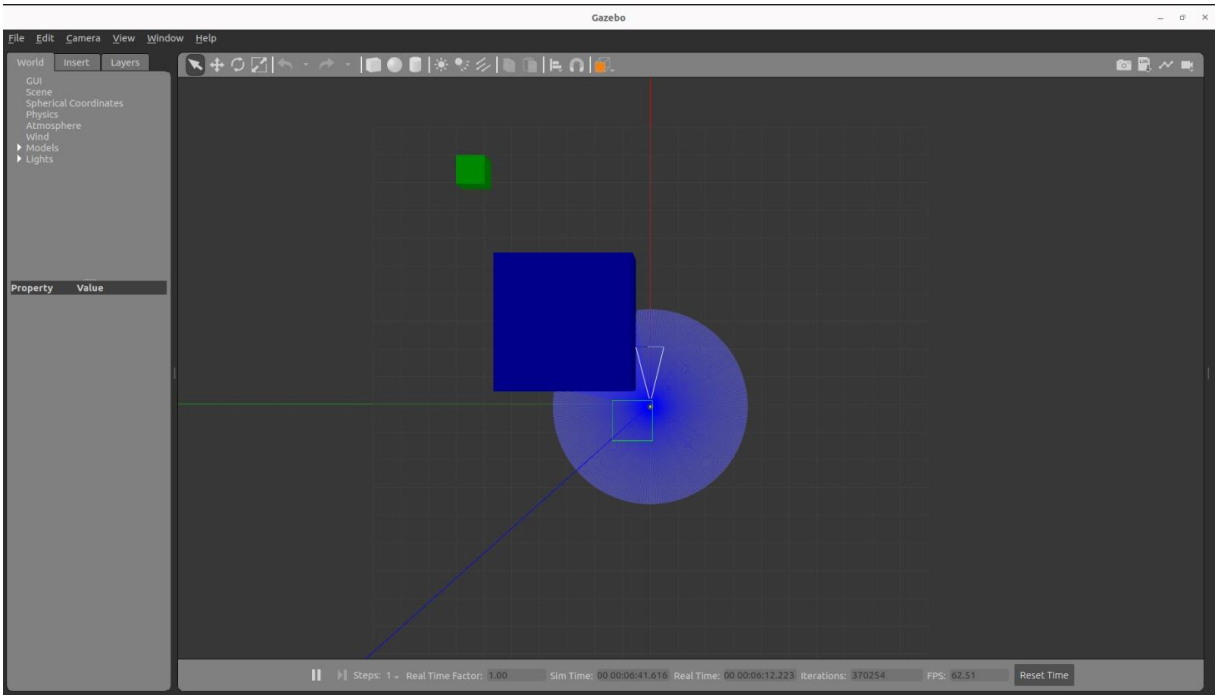
Nodurile VerificareRotatie, VerificareMersInainte si ColorDetectorNode se află în același fișier și rulează în paralel datorită configurării unui executor multi-threaded.

```python
def main(args=None):
    rclpy.init(args=args)

    node = ColorDetectorNode()
    node2=VerificareMersInainte()
    node3=VerificareRotatie()

    # activarea rularii in paralel a celor 3 noduri
    executor=MultiThreadedExecutor()
    executor.add_node(node)
    executor.add_node(node2)
    executor.add_node(node3)

    try:
        executor.spin()

    except KeyboardInterrupt:
        pass

    finally:
        node.destroy_node()
        rclpy.shutdown()

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

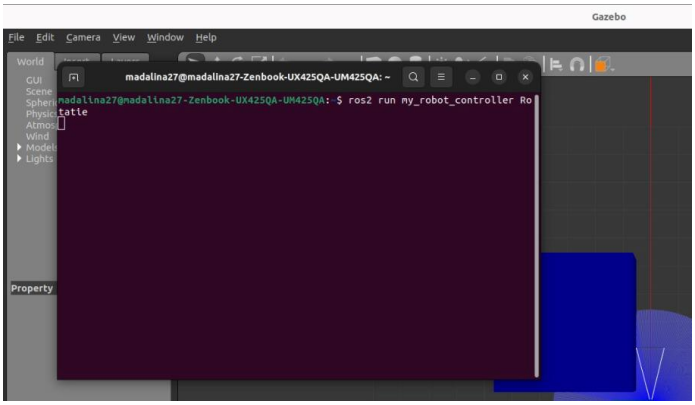
5. GHIDUL UTILIZATORULUI

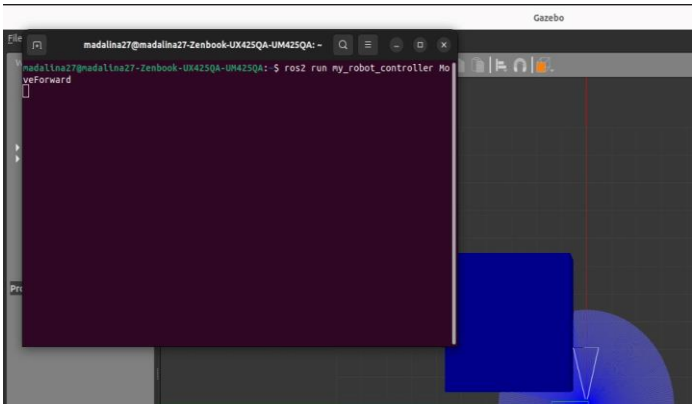
Pentru a putea utiliza acest proiect, trebuie realizați următorii pași:

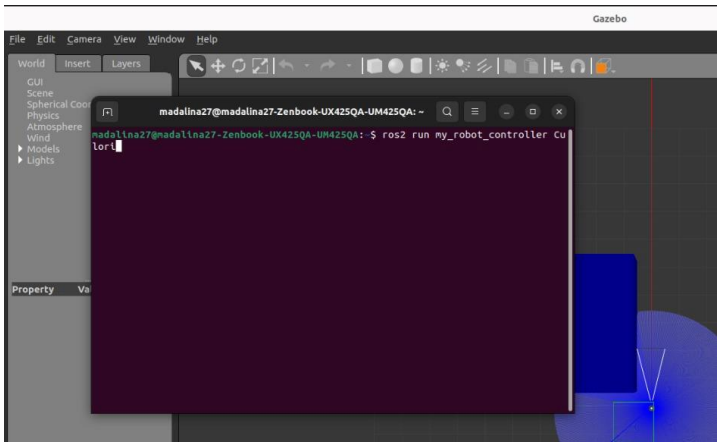
- ros2 launch turtlebot3_gazebo Proiect_PS.launch.py-----această comandă trebuie introdusă într-un terminal nou, pentru a deschide lumea creată in Gazebo.



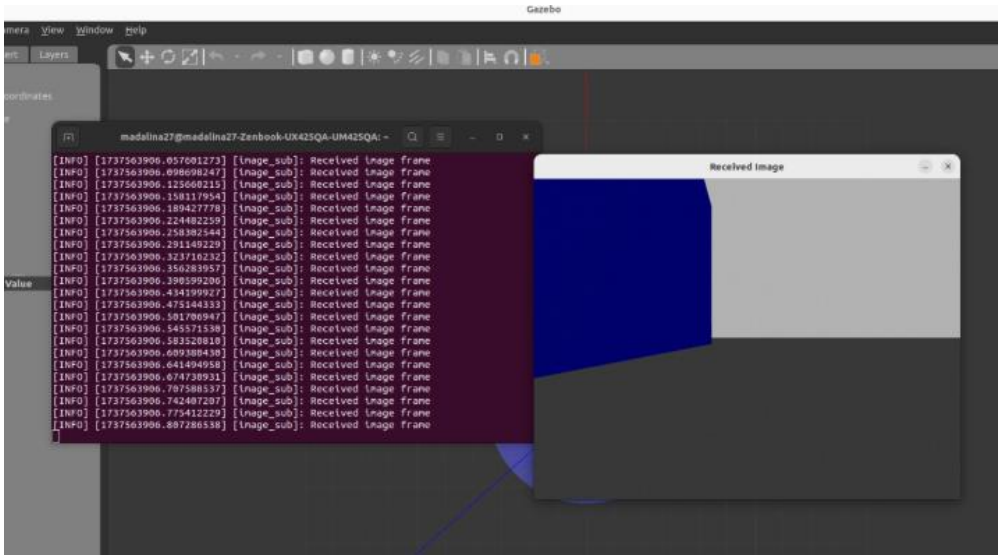
- ros2 run my_robot_controller nume_nod-----această comandă este utilizată pentru a executa nodurile necesare.







- colcon build --symlink-install-----această comandă este folosită pentru a compila nodurile create în Visual Studio Code.
- export TURTLEBOT3_MODEL=waffle_pi-----această comandă poate să fie utilizată în condițiile în care, pornită fiind simularea, robotul nu apare.
- ros2 run my_robot_controller image-----folosim comanda pentru a putea utiliza senzorul de cameră al robotului

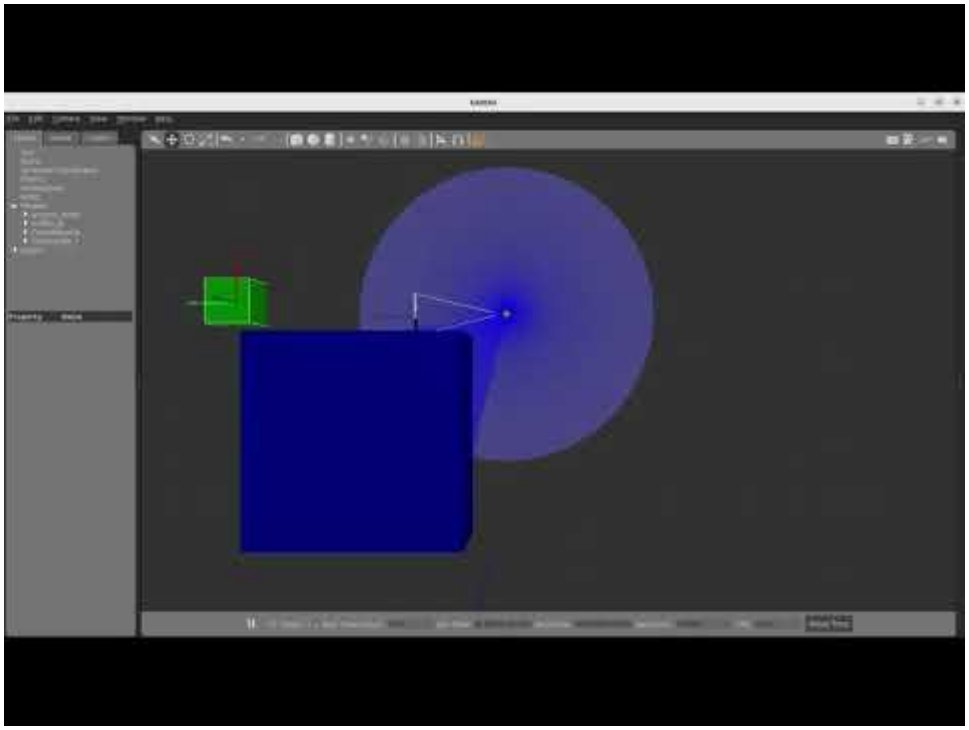


În videoclipul următor este prezentat modul în care funcționează proiectul, fiind încercate toate scenariile posibile:

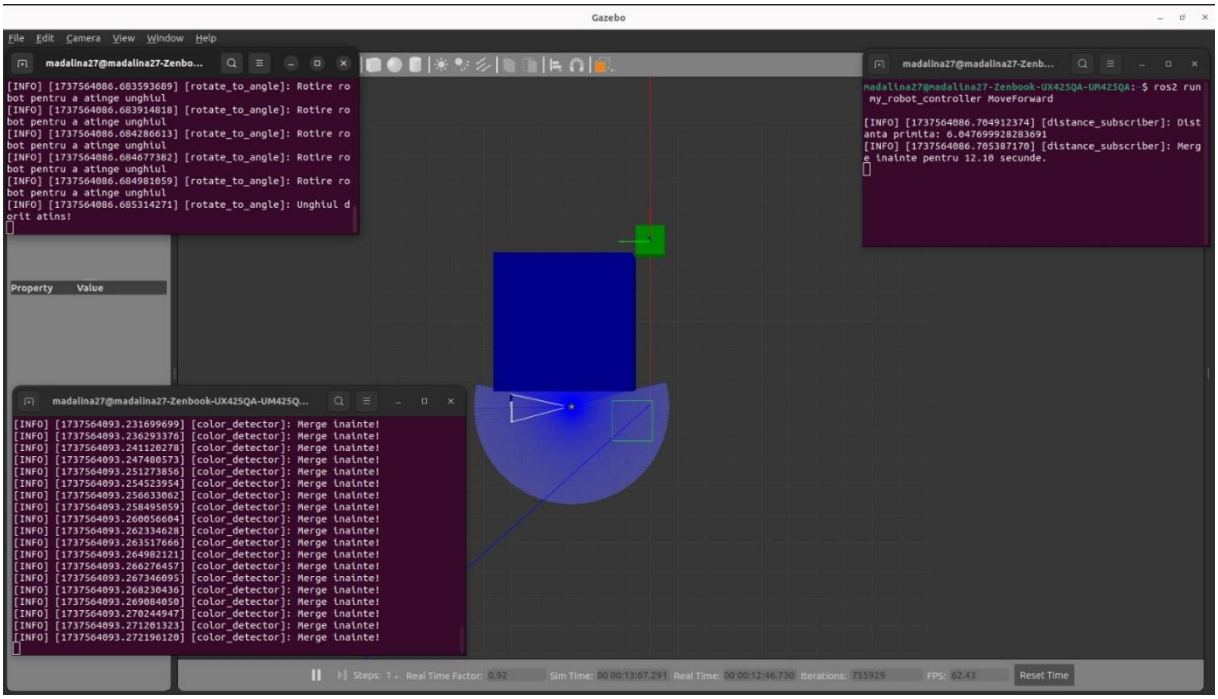
- robotul pornește din colțul din dreapta-jos, verificând atât în dreapta cât și în stânga, dacă există vreun bloc de culoare verde, iar în

momentul în care îl vede, merge într-un colț al cutiei, unde se poate ascunde, pentru a nu mai vedea blocul.

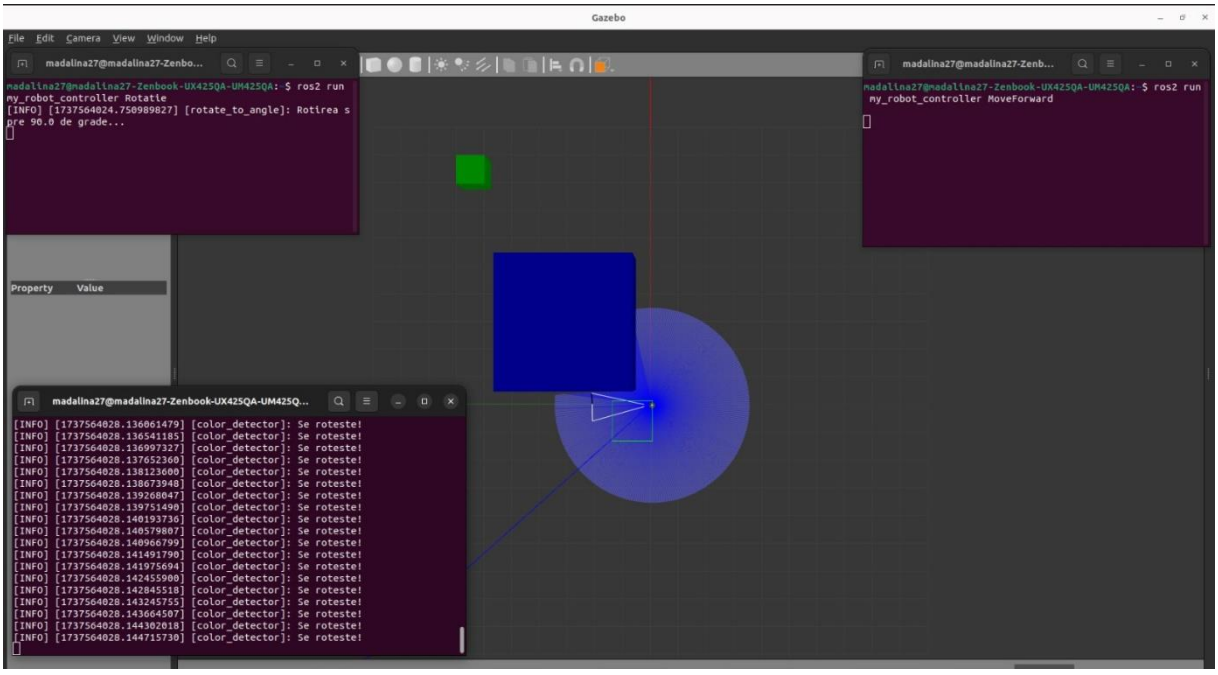
- în momentul în care detectează blocul de culoare roșie, se presupune că a fost "găsit", prin urmare, acesta se învârte la infinit, în punctul fix, în care se află.



6. TESTARE ȘI PUNERE ÎN FUNCȚIUNE



Cu ajutorul acestor terminale și datorită mesajelor sugestive implementate în cod, în timpul simulării reușim să înțelegem și să vizualizam mult mai ușor comportamentul robotului și comunicarea dintre noduri.



7. CONCLUZII

Proiectul **"Hide and Seek"** a reprezentat o oportunitate excelentă de a explora și aplica concepte esențiale din domeniul roboticii mobile, utilizând platforma TurtleBot3 și ROS 2. Prin integrarea tehnologiilor avansate de detecție a culorilor, navigație autonomă și teleoperație, am demonstrat cum roboții pot interacționa eficient cu mediul înconjurător și pot îndeplini sarcini specifice într-un mod adaptabil și precis.

În cadrul proiectului, am reușit să:

- Dezvoltăm un algoritm funcțional pentru detecția culorilor (roșu și verde), care ghidează comportamentul robotului în funcție de obiectivele stabilite.
- Implementăm un sistem de navigație autonomă care permite robotului să se deplaseze în siguranță în zona de testare.
- Validăm toate funcționalitățile prin simulări în Gazebo, asigurându-ne că soluția este robustă și eficientă.

De asemenea, ne-a permis să aprofundăm cunoștințele despre ROS 2 și să ne dezvoltăm abilitățile practice în domeniul roboticii.

Acest proiect poate servi drept bază pentru extinderea funcționalităților viitoare, cum ar fi utilizarea algoritmilor de inteligență artificială pentru luarea deciziilor mai complexe sau integrarea unor senzori suplimentari pentru o interacțiune mai bogată cu mediul.

Prin urmare, considerăm că **"Hide and Seek"** este un exemplu concludent al potențialului pe care roboții mobili îl oferă în rezolvarea unor sarcini dinamice și diverse.

8. BIBLIOGRAFIE

[1] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/

[2] https://www.intelrealsense.com/

[3] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications

[4] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/

[5] https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation

[6] https://docs.ros.org/

[7] https://emanual.robotis.com/docs/en/dxl/

[8] https://www.raspberrypi.com/documentation/

[9] https://emanual.robotis.com/docs/en/parts/controller/opencr10/