Assignment II

Mobile Robotics (MECH-B-4-MRV-MRO-ILV)

Mechatronics, Design and Innovation

4. Semester

Lecturer: Daniel McGuiness, PhD

Author: Lang Andreas

June 30, 2025

1 Introduction

Part of the lecture "Mobile robotics" revolved around learning the basics of ROS2, a popular framework for creating software for robots. The task which is the topic of this report, is to use the integrated simulation environment "turtlesim" to create a simple program. The goal of this program is to make the turtlebot move around the area it is constrained to. The type of movement pattern is described in the document containing the details of this assignment:

- Face a random direction
- Start moving forward
- If an edge is detected, print a message to the terminal and turn 90° clockwise
- Continue moving forward until an edge is detected again

2 Turtlesim

Turtlesim is a simple simulation environment, included by default in ROS2. There are several nodes included in this environment, th most important one being "turtlesim_node"2.1.

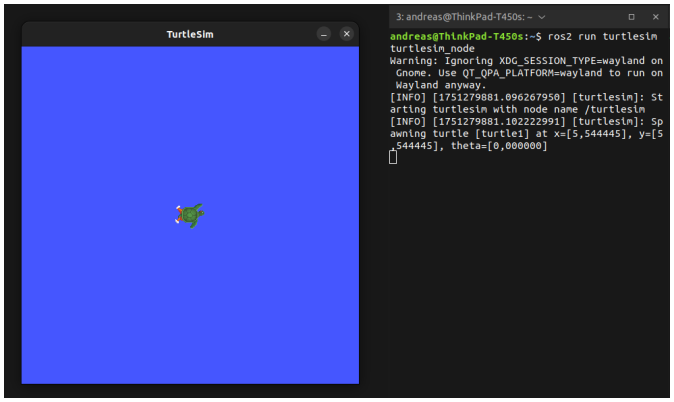

Figure 2.1: turtlesim_node

This node creates a graphical representation of the robot while also providing the interfaces necessary to interact with it. These interfaces are implemented using topics2.2.

Figure 2.2: turtlesim topics

3 Software implementation

As we can see in the previous section, there are several topics we can use to interact with the robot. The ones used in this implementation will be "/turtle1/cmd_vel" and "/turtle1/pose". "/turtle1/cmd_vel" enables us to move the robot in a direction at a certain speed and "/turtle1/pose" provides the current position.

The node implementing the required logic is called "moveNode.py". In general this node can be divided into 3 main components:

- Publisher
- Subscriber
- State machine

We can see part of the previously mentioned components being initialized inside the "__init__" function3.1.

```
class moveNode(Node):
    def __init__(self):
        super().__init__("moveNode")
        self.state = 1
        self.randomNumber = random.uniform(0,5)

        #Creating publisher and subscriber
        self.cmd_vel_pub = self.create_publisher(Twist, "/turtle1/cmd_vel",10)
        self.pose_sub = self.create_subscription(Pose,"/turtle1/pose",self.poseCallback,9)

        #Checking the state machine every 0.1 seconds
        self.timer = self.create_timer(0.1, self.stateMachine)
        self.get_logger().info("Node started")
```

Figure 3.1: init function

Publisher and subscriber each represent a method inside the node 3.2. This ensures simple handling of the values within the following state machine.

```
def sendMoveCommand(self,x,z):
    msg = Twist()
    msg.linear.x = x
    msg.angular.z = z
    self.cmd_vel_pub.publish(msg)

def poseCallback(self,msg:Pose):
    self.x = msg.x
    self.y = msg.y
    self.theta = msg.theta
```

Figure 3.2: Publisher and subscriber

The main component of this node is the state machine 3.3. it handles subscriber/publisher data and defines the order of events/actions required to implement the movement of the turtlebot. As we can see in the init functio of the node the state machine is being called every 100 ms

```python
def stateMachine(self):
    match self.state:
        #Turn for a random duration
        case 1:
            self.sendMoveCommand(0.0,1.0)
            self.randomNumber-=0.1
            if(self.randomNumber < 0):
                self.state = 2
        #Start moving forward, stop turning
        case 2:
            self.sendMoveCommand(1.0,0.0)
            self.state = 3
        #Check boundaries
        case 3:
            if((X_MIN < self.x < X_MAX) and (Y_MIN < self.y < Y_MAX)):
                self.state = 2
            else:
                self.state = 4
                self.oldTheta = self.theta
        #Out of bounds -> start turning
        case 4:
            self.get_logger().info("edge detected!")
            self.sendMoveCommand(0.0,-2.0)

            angle = self.theta - self.oldTheta
            angle = math.fmod(angle, 2 * math.pi)

            if angle > math.pi:
                angle -= 2 * math.pi
            elif angle < -math.pi:
                angle += 2 * math.pi

            angle = abs(angle)

            #If 90° have been reached, move with increased speed to "escape" the wall
            if(angle > math.pi/2):
                self.sendMoveCommand(2.0,0.0)
                self.state = 2
```

Figure 3.3: State machine

The resulting movement reflects the one mentioned in the description of this assignment 3.4.

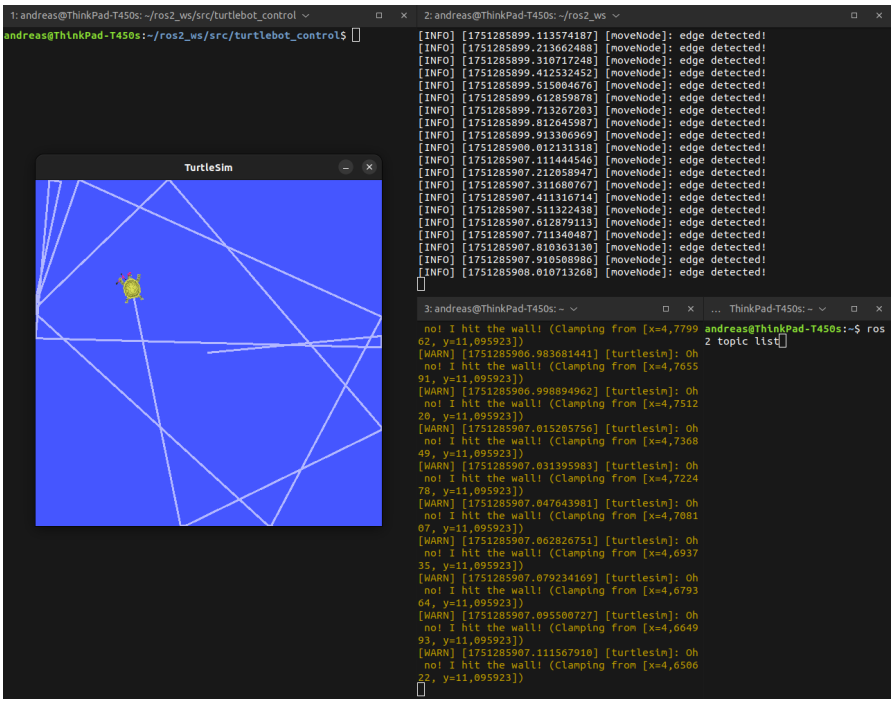

Figure 3.4: Resulting movement

4 Conclusion

The concepts of ROS2 were quickly understood in the lecture itself but using it in software has proven to be a challenge at times. Being used to more of a sequential style of programming meant that understanding nodes in an asynchronous context was key to using the framework effectively. This assignment represents a solid first step in getting familiar with ROS2.

List of Figures

2.1 turtlesim_node 1
2.2 turtlesim topics 1
3.1 init function 2
3.2 Publisher and subscriber 2
3.3 State machine 3
3.4 Resulting movement 3