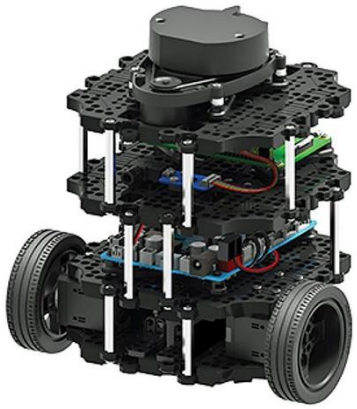
Project Innovation: Mid Report

NavigateT: TurtleBot



Students: Marc CHAMBRÉ / Loan MICHAUD / Dhia JENZRI / Reda TRICHA

Training: Electronics, Computing and Systems engineering

Teachers: Ionela PRODAN / Simon GAY

School Year: 2023 - 2024

Table des matières

1. Presentation of the project

In the realm of robotics, the main objective of mobile robots is to navigate through environments, whether these are known and mapped or entirely unknown. The main point to achieve this goal is to adopt smooth and optimal trajectories that can avoid static or movable obstacles.

The project's goal consists of introducing that notion by controlling a two-wheel robot named *TurtleBot*, equipped with a Lidar and a *Raspberry Pi* board. It's a differential drive mobile robot with open-source software. The project is intended to allow us to have an experience in exploring ROS and Gazebo.

Our project can be divided into three main phases:

1.1 Robot basic control

The first goal of the project will be to find a mathematical model for the robot and to tune a controller. This model will serve as the basis upon which we construct and fine-tune a controller, enabling the robot to follow predefined paths over time. Through this initial phase, we aim to set precise control over the robot's movements, thus setting the stage for more complex navigation challenges.

1.2 Navigation

Then the focus of our research will be shifted towards navigation algorithms and obstacle avoidance. This will allow us to generate trajectories going from point A to point B on a mapped area and define objectives through a room.

1.3 Multiple agents' navigation

The ultimate goal of our project will be to manage several Turtlebots with distinct objectives that may intersect or interfere with one another. Managing multiple agents adds layers of complexity and calls for the development of advanced navigation strategies. These strategies will cover a dynamic obstacle avoidance mechanism, where obstacles may assume the form of immobile barriers or mobile entities. The successful execution of this phase will allow our system to be more versatile and adaptable, allowing it to manoeuvre in environments teeming with dynamic elements and conflicting objectives.

In summary, our project is a journey for us to discover and innovate within the domain of robotics, with the aim of creating a robust, autonomous, and multi-agent capable system that can navigate efficiently through both known and unknown environments. All our work will have to be validated at the end on a TurtleBot at the Esisarium platform.

2. Project development plan

First of all, in order to understand the objectives of this project, it was presented to us by Ionela PRODAN and Simon GAY. We also took part in a demonstration of the software tools for communicating and using the *TurtleBot* by Guillaume SCHLOTTERBECK. During the first few weeks of the project, the aim has been to understand the various research and internship projects that have been carried out to date, and then to pursue these research projects. We were then able to set up our working environment and software. We installed all the software tools needed to develop, simulate and communicate with the *TurtleBot*: *ROS2 Humble* and *Gazebo*. Installation was not easy, as there were many different versions of these programs, and we had to ensure consistency between them. As we're working on a Linux environment with the software mentioned above, which requires a large number of commands, we've centralized these commands in a procedure file, so that we can find them quickly, and leave additional documentation for our project and future people working on it.

To explain briefly, the *Robot Operating System* (*ROS*) is a set of software libraries and tools that help you build robot applications. For that, we can create nodes that publish or listen to topics, transfer data from sensors to the controller, and control actuators. The working principle is robust because several nodes already exist, including *TurtleBot3* which is using, and many others. Then we can assemble nodes to make up our application. Gazebo is a simulator for ROS nodes that allows us to do 3D simulations, with obstacles and robot models. This is a convenient tool to first test our nodes virtually.

After that, we started to control the real *TurtleBot* using a hotspot and a *ssh* link. We also tried to map out our project room with RViz. One of our main issues was to understand how the connection setup between the hotspot and the *Raspberry Pi* card should work well, we had encountered some problems when trying to connect the Raspberry Pi to the hotspot (sometimes the connection crashes and we have to reboot everything in order for it worked again). We also tried to configure our hotspot on our personal computers to have better graphic performances on Rviz software.

We started to explore existing navigation algorithms among which we recognize things like 3rd order trajectory equations.

We started exploring the possibility of feedback control of a *ROS*-enabled robot using *Matlab* and *Simulink*. This will enable us to run a model that implements a simple closed-loop proportional controller for mobile robot trajectory tracking.

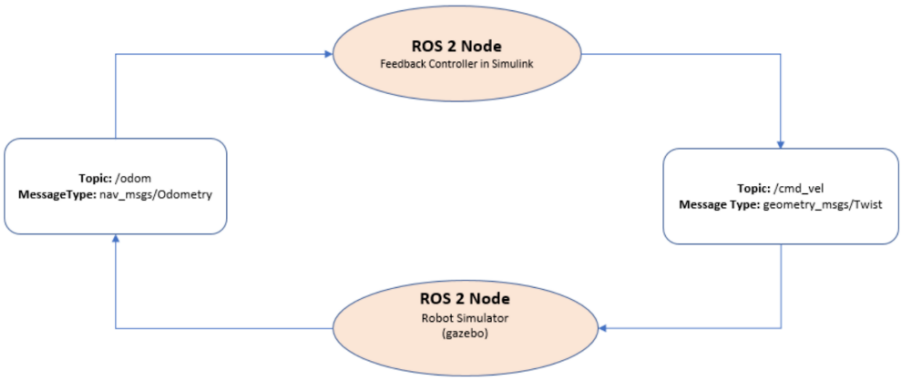


Figure 1 - ROS feedback controller

We started to research a mathematical model for the robot, to have a basis for a controller. We won't go into all the mathematical calculations in this report. They are included in the weekly reports we produced and will be included in the final report. The mathematical model we found helped us to represent the control loop as follows:

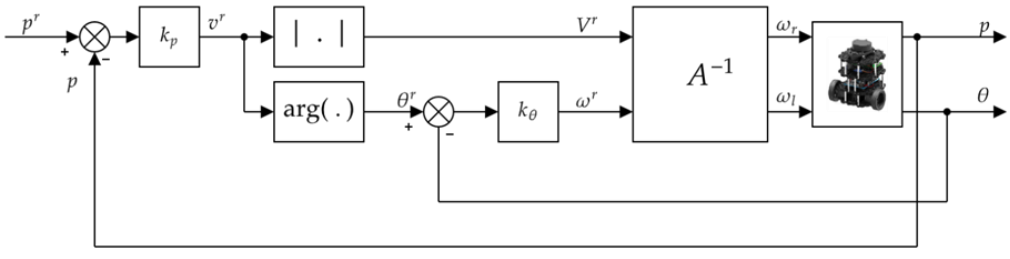


Figure 2 - Schematical control loop for the TurtleBot

As a reminder:

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \dot{\xi} = \begin{bmatrix} V\cos\theta \\ V\sin\theta \\ \omega \end{bmatrix} \quad \tilde{u} = \begin{bmatrix} V \\ \omega \end{bmatrix}$$

Where $\tilde{u}$ is the intermediate input. It is supported by Python functions but for the physical model, the real input is:

$$u = \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} \text{ such that } \tilde{u} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{L} & -\frac{r}{L} \end{bmatrix} u$$

Where r is the radius of the wheels, and L is the distance between the centre of the wheels.

We implant it in Simulink & Matlab:

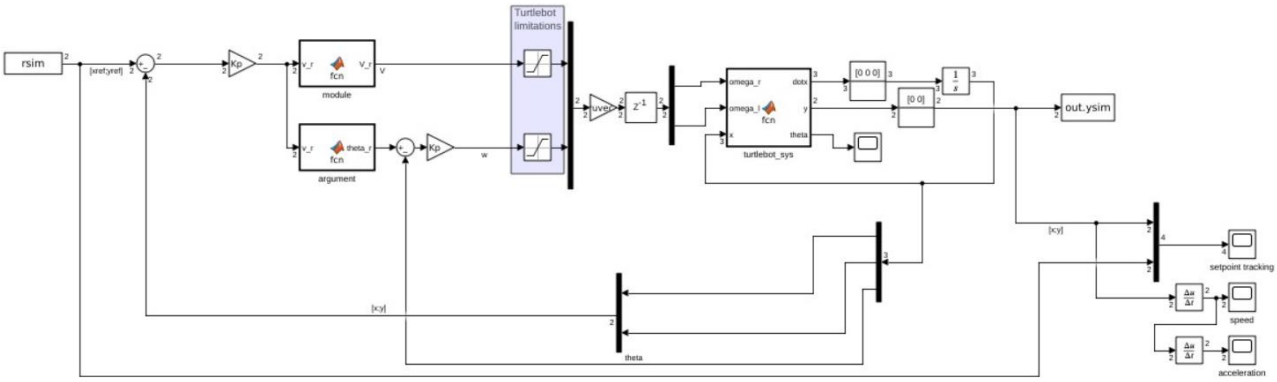


Figure 3 - Simulink model of our P-controller

At the same time, we started creating nodes in Python to navigate the *TurtleBot* remotely. Thus, we have created our workspace in Visual Studio to code nodes. The first node we have created gives the *TurtleBot* a command to go straight ahead during 10s along the x-axis. It worked both in simulation and with the real *TurtleBot*. The Python code is available in the appendix (**Appendix 1**).

Thanks to the work of the automation team, we now have a control model that needs to be coded into a Python node so that the *TurtleBot* can implement it. We've done some initial code tests in Python, but for the moment we haven't had any results because our code isn't compatible with the different versions we use between the school computers and our personal installations.

We also visited the platform at Esynov and we will try to set up the link between the system and the *TurtleBot* next week, using existing ROS nodes. Thanks to the cameras in the Esynov room, we can accurately estimate the *TurtleBot's* coordinates in the space in which it is located. Actually, the *TurtleBot* embeds coordinates computing. We did some experiments with these computed coordinates, and we understood that they are only calculated thanks to the wheel speed. Therefore, it doesn't consider wheel grip (what we could expect), but it also doesn't remark whenever the *TurtleBot* is stuck against a wall or whenever we lift it. This system will nevertheless be useful to do some preliminary tests in our room before testing at Esynov.

After that, we tried to code an obstacle-avoidance algorithm in Python. When an obstacle is detected, it changes direction and continues to navigate in a space. The code for this algorithm is in the appendix (**Appendix 2**). We were able to test it in simulation, and it was quite effective. However, we haven't yet managed to test it on the real *TurtleBot* due to the version conflicts we still have.

Following the *MPC* course, the automation team tested to control the *TurtleBot* with *MPC*. The discretization of the model is the following:

$$\dot{\xi}(k) = \begin{bmatrix} \dot{x}(k) \\ \dot{y}(k) \\ \dot{\theta}(k) \end{bmatrix} \approx \frac{1}{T_s} \begin{bmatrix} x(k+1) - x(k) \\ y(k+1) - y(k) \\ \theta(k+1) - \theta(k) \end{bmatrix} = \begin{bmatrix} V(k)\cos(\theta(k)) \\ V(k)\sin(\theta(k)) \\ \omega(k) \end{bmatrix}$$

$$\xi(k+1) = \begin{bmatrix} x(k+1) \\ y(k+1) \\ \theta(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + T_s \begin{bmatrix} V(k)\cos(\theta(k)) \\ V(k)\sin(\theta(k)) \\ \omega(k) \end{bmatrix}$$

$$u_{\max} = \begin{bmatrix} V_{\max} \\ \omega_{\max} \end{bmatrix} = \begin{bmatrix} 0.22\ \mathrm{m.s^{-1}} \\ 2.84\ \mathrm{rad.s^{-1}} \end{bmatrix}$$

We can then implement this to get *MPC* from *casadi* optimization solver.

It worked well for circular reference tracking:

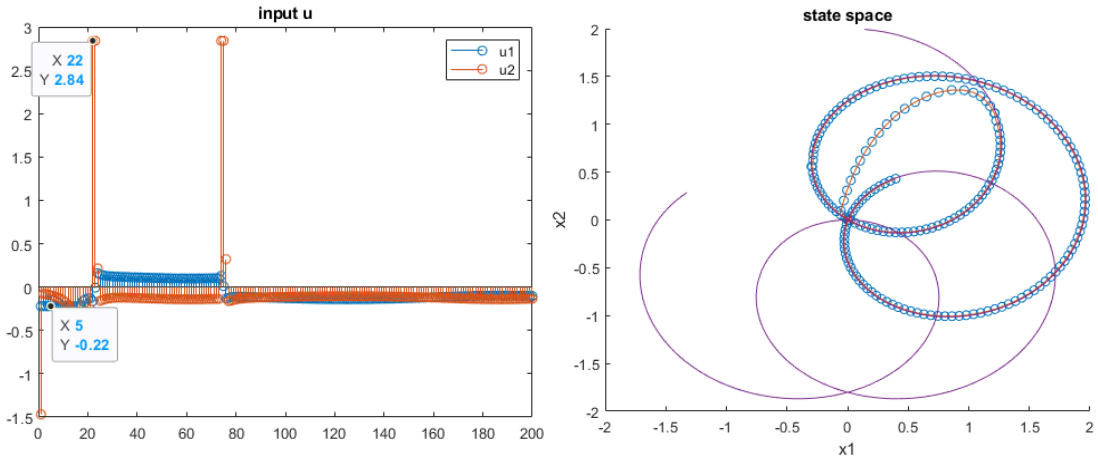


Figure 4 - Reference tracking using MPC

On the left, we can see that *TurtleBot's* constraints are fulfilled. On the right, we can see the reference in full line, and the dotted robot's trajectory. You can find the Matlab code in the appendix (**Appendix 3**)

After all the research we've carried out since the start of the project, we now know what our objectives are. Over the next few weeks, the development team will have to find ways of implementing the automatic team's mathematical models in Python, so that we can test them on the real *TurtleBot*.

3. Planification and project schedule

To organize our project and keep it on schedule, we implemented a different set of tools. Firstly, to meet deadlines and keep track of our project's progress, we decided to use Gantt software. In particular, it will enable us to associate several hours for each stage of our project and adjust our schedule according to the technical constraints we face.

The second tool we've decided to use is the Trello project management tool. It will help us on a few levels: organizing sessions using checklists, sharing documents, tasks in progress or completed, distributing work and visualizing work progress. To access our Trello account, please click on the following link:

https://trello.com/b/4TAB7QMB

To be productive, we put several tools in place for communication and document backup tools. We've set up various channels such as Discord and Messenger to keep in touch and communicate about the project. To save documents, we've set up a shared Google Drive. We also drew up a Gantt chart to give us objectives for the duration of the project, but above all to structure the different parts of the project and move forward productively (**Appendix 4**).

To share the progress of our project and motivate ourselves every week, we produce a weekly report on what we've done or tried to do. This also enables us to keep a written record of our work

and see how it's progressing. Each week, we share this report with Ionela PRODAN, who monitors our progress and motivates us in our project.

With these different tools, we hope to be as efficient as possible in our work. The final keyword we'll be using throughout the project is communication. We'll need to communicate within our group to share ideas and progress on each sub-task carried out by a group member.

4. Distribution of individual tasks

As far, we decided to split into two teams. After experimenting for the 3 first weeks with everyone working on the same topic, we needed to speed up our project and we decided to distribute tasks: Control theory ones, and software development.

4.1 Automation Team

This team is composed of Marc CHAMBRÉ and Reda TRICHA. Their role is to model the robot and design/tune a controller for this system on Matlab/Simulink. The main objective is to perform several simulations to find optimal tunings, and finally to explain their work to the development team.

Their main tasks are the ones below:

- Describe the mathematical model
- Implement a controller in *Matlab*/*Simulink*
- Improve this controller
- Try feedback linearization
- Try MPC
- Work on obstacle avoidance

4.2 Embedded software programming Team

This team is composed of Loan MICHAUD and Dhia JENZERI. They will focus on all that is related to *TurtleBot3* hardware setup, *ROS2* configuration and software development that will include programming and implementing the automation team trajectory planning and path optimization model and of course the test and troubleshooting of the whole system.

The precision of their work is primordial because they must optimize and debug the code before testing at the platform in Esynov. The access to this platform is rigorously planned, it is important to be the most efficient once in tests.

Their tasks are described below:

- Hardware setup
- ROS2 configuration and setup
- Familiarizing and understanding of ROS2
- Writing and testing a first node
- Writing more nodes and doing simulation
- Implement controllers in Python
- Testing on the Turtlebot

5. Milestones and risk analysis

Before starting up a project, it is important to be aware of risks that are involved in the project, and that could put him in danger. Also, to guarantee the advancement of the project, we have to put some Milestones in the project, defining the big steps.

5.1 Milestones

We set four Milestones, being the end of the project's initialization, the mid-report and the results of the tests at Esynov. These Milestones will be the occasion for us to take a step back on the project a readjust the trajectory of the project.

We planned two sessions at Esynov as Milestones, but it is not excluded that we plan some other sessions to do preliminary or complementary tests. As we have a solution to test our algorithms in our room, we may not need to add lots of sessions. Of course, these sessions will be announced in advance.

5.1.1 End of initialization: end of week 39

We set this Milestone because we saw that we took too much time to set up setting up the project. We defined a deadline at which we were going to effectively start the project (code, modelize, simulate etc.). Indeed, it helped us to really work on the main objectives of the project.

5.1.2 Mid-report: end of Week 43

This present report allows us to see what we already achieved and what it remains to do. While doing the Gantt diagram, we saw that we need to speed up a bit to have time to achieve advanced objectives.

5.1.3 First test: end of Week 47

We planned the first tests at Esynov for the end of week 47. This Milestone will allow us to validate all the algorithms we did until there, being setpoint and reference tracking. If the tests are conclusive, we will pass to the obstacle avoidance and the multiple-*TurtleBot* management.

5.1.4 Second test: Week 2

The second test will allow us to do a great retrospective of what we achieved until there. It will be the occasion to test everything we had time to do, to finalize the project.

5.2 Risk analysis

5.2.1 Technical risks

Technical risks can compromise the quality of the project and conceal problems such as software bugs due to the management of different ROS versions on computers and in the Raspberry PI software. Technical risks also include design and data backup errors. Indeed, during this project, we'll be using software tools that we're not familiar with. The risk with unfamiliar software tools is that we spend a lot more time configuring them, but also using them. A certain amount of rigour will be required in the communication and use of these various tools, as well as very regular data backup in the event of design errors or data loss. In a corporate project, we could have talked about the financial costs incurred due to the technical risks mentioned above. In the case of our project, we won't mention the budgetary cost, as the hardware is provided by the school. There is also a risk of hardware failure, but it has a low likelihood to happen as the hardware provided by the school is in a perfect condition.

5.2.2 Resources

Some risks are linked to the fact that the link between our computer and the school network has been cut. Access to *Matlab* is provided by temporary licenses or *Matlab* online, which is not ideal for working. Even if we can access the TP computers equipped with *Matlab* licenses, we can't make direct communication between the school PCs and the *TurtleBot*. Internet access is also limited and

relies solely on our personal resources. The same applies to software installation, which requires a lot of graphics resources, something we don't have on our personal computers or those currently supplied to us. Simulation is very important to our project, especially for testing navigation controls in Python. Lastly, we have access to the Esynov test platform, which is not permanently accessible, so we'll have to plan sessions in advance, while not being sure that the room will be available when we plan to use it.

5.2.3 Human risks

Human risks can be many and varied. The success of this project can only be achieved within a framework of benevolent communication and working group cohesion between the members of our team. Indeed, the group is made up of one student, two apprentices and one MISTRE, who don't know each other very well because of our status at school. That's why we'll need to meet regularly as the various stages of the project progress. What's more, given that the project involves a wide range of skills, it's possible that some team members won't have the skills required to carry out their tasks. To mitigate this risk, it will be important to help each other with complex tasks, and not to leave a member in difficulty. Finally, if one or more members are ill or absent for various reasons, we'll need to rebalance tasks between the other members. In addition, it will be important for each member to document his or her part in such a way that we can not only understand it but also take it over if necessary.

5.2.4 Timing risks

For this project, we have dedicated sessions on Tuesday afternoons. However, due to other courses scheduled in the afternoon, notably Jean-Pierre CEYSSON's innovation courses, as well as events such as the "Forum des entreprises", it is not possible to follow the objectives we set ourselves each week. To make up for this shortfall, we're trying to schedule an extra session every Thursday afternoon. This project requires a considerable amount of personal time, in addition to the specialist courses we take every week.

5.2.5 Continuity after our work

Since the beginning of the project, we have tried to make everything clear so that the one who will take over the project after us will understand our work clearly. The risk may be the compatibility of ROS versions. Indeed, the version of ROS implemented in the TurtleBot is not the LTS one. It may be more convenient to update the version, but we don’t have time to do it. That is why the future workers on the project may have trouble translating our nodes from ROS Foxy to Humble (LTS).

6. Insights on the societal challenges of innovation

This innovative project in the field of robotic navigation in unfamiliar environments has significant implications for several key societal challenges. Firstly, it can make a substantial contribution to the autonomous mobility revolution. The advanced navigation systems developed in this project could find applications in autonomous public transport, shared electric vehicles, and even automated delivery services, thereby reducing traffic jams and greenhouse gas emissions, and improving mobility in congested urban areas.

What's more, this innovation can have a significant impact on environmental mapping and monitoring, helping to manage natural disasters and monitor high-risk areas. Autonomous robots equipped with advanced navigation systems could be deployed to explore dangerous or hard-to-reach areas, providing crucial information for disaster prevention, rescue operations and environmental preservation.

In this context, some of today's existing solutions, such as autonomous vehicles from Waymo and Tesla, autonomous car-sharing services, drones for environmental monitoring, and autonomous

robots in industry, are already being developed and applied. However, it is essential to note that these current solutions are still in the development and optimization phase, and challenges remain, particularly in terms of safety, regulation and public acceptance.

The ongoing project on robotic navigation in unfamiliar environments can help to address these challenges by developing more advanced and robust approaches, thereby reinforcing existing solutions to societal challenges more efficiently and ethically. All in all, it offers a promising perspective for a future of smarter mobility, safer environmental management and enhanced industrial transformation.

Conclusion

To conclude these first few weeks of the project, we're pretty satisfied with our progress. As far, we managed to understand quite well *ROS2* and the implementation on Python, we coded some nodes that work well, we succeeded to retrieve the position of the TurtleBot and we did some simulations about obstacle avoidance. Currently, we are implementing and testing the controller on the real TurtleBot. Despite the various problems we've encountered, we've managed to get a good grasp of the project's objectives, and our research has so far been fruitful. The fact that we've split the group in 2, with some people working on the mathematical models and the rest on the implementation of the *TurtleBot*, has enabled us to make efficient progress. We look forward to promising results in the future, and to controlling several *TurtleBots* at the same time.

List of Figures

APPENDIX

TABLE OF APPENDIX

Appendix 1: Python node to control the *TurtleBot* straight ahead

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

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
import time
class  move(Node):
    def  __init__(self):
         super().__init__('test_node')
         self.publisher=self.create_publisher(Twist,'cmd_vel',10)
         self.timer = self.create_timer(0.1,self.timer_callback)
         self.cmd_msg=Twist()
         self.start_time=time.time()

    def timer_callback(self):
         current_time=time.time()
         if current_time - self.start_time < 10.0:
            self.cmd_msg.linear.x=0.2
            self.publisher.publish(self.cmd_msg)
         else:
             self.cmd_msg.linear.x=0.0
             self.publisher.publish(self.cmd_msg)

def main(args=None):
         rclpy.init(args=args)
         node=move()
         try :
             rclpy.spin(node)
         except KeyboardInterrupt:
             pass
         node.destroy_node()
         rclpy.shutdown()

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

Appendix 2: Python node to avoid obstacles

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist

class LaserScanSubscriber(Node):
    def __init__(self):
        super().__init__('reading_laser')
        self.publisher = self.create_publisher(Twist, '/cmd_vel', 1)
        self.subscription = self.create_subscription(LaserScan,
'/robot/laser/scan', self.laser_callback, 10)
        self.threshold_dist = 1.5
        self.linear_speed = 0.6
        self.angular_speed = 1

    def laser_callback(self, msg):
        regions = {
            'right':  min(min(msg.ranges[0:2]), 10),
            'front':  min(min(msg.ranges[3:5]), 10),
            'left':   min(min(msg.ranges[6:9]), 10),
        }

        self.take_action(regions)

    def take_action(self, regions):
        msg = Twist()
        linear_x = 0
        angular_z = 0

        state_description = ''

        if regions['front'] > self.threshold_dist and regions['left'] >
self.threshold_dist and regions['right'] > self.threshold_dist:
            state_description = 'case 1 - no obstacle'
            linear_x = self.linear_speed
            angular_z = 0
        elif regions['front'] < self.threshold_dist and regions['left'] <
self.threshold_dist and regions['right'] < self.threshold_dist:
            state_description = 'case 7 - front and left and right'
            linear_x = -self.linear_speed
            angular_z = self.angular_speed  # Increase this angular speed
for avoiding obstacle faster
        elif regions['front'] < self.threshold_dist and regions['left'] >
self.threshold_dist and regions['right'] > self.threshold_dist:
            state_description = 'case 2 - front'
            linear_x = 0
            angular_z = self.angular_speed
        elif regions['front'] > self.threshold_dist and regions['left'] >
self.threshold_dist and regions['right'] < self.threshold_dist:
            state_description = 'case 3 - right'
            linear_x = 0
            angular_z = -self.angular_speed
        elif regions['front'] > self.threshold_dist and regions['left'] <
self.threshold_dist and regions['right'] > self.threshold_dist:
            state_description = 'case 4 - left'
            linear_x = 0
            angular_z = self.angular_speed
```

```
        elif regions['front'] < self.threshold_dist and regions['left'] >
self.threshold_dist and regions['right'] < self.threshold_dist:
            state_description = 'case 5 - front and right'
            linear_x = 0
            angular_z = -self.angular_speed
        elif regions['front'] < self.threshold_dist and regions['left'] <
self.threshold_dist and regions['right'] > self.threshold_dist:
            state_description = 'case 6 - front and left'
            linear_x = 0
            angular_z = self.angular_speed
        elif regions['front'] > self.threshold_dist and regions['left'] <
self.threshold_dist and regions['right'] < self.threshold_dist:
            state_description = 'case 8 - left and right'
            linear_x = self.linear_speed
            angular_z = 0
        else:
            state_description = 'unknown case'
            self.get_logger().info(str(regions))

        self.get_logger().info(state_description)
        msg.linear.x = linear_x
        msg.angular.z = angular_z
        self.publisher.publish(msg)


def main(args=None):
    rclpy.init(args=args)
    laser_scan_subscriber = LaserScanSubscriber()
    rclpy.spin(laser_scan_subscriber)
    laser_scan_subscriber.destroy_node()
    rclpy.shutdown()

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

Appendix 3: MPC control in *Matlab*, using *casadi*

```
clear all; close all; clc;

%% Definition of constants
% System dimension
dx=3;   %[x y theta]
dy=2;   %[x y]
du=2;   %[V omega]

% Initial conditions
x0=[0;0;5*pi/4]; %Point 0;0 with theta=5*pi/4
u0=zeros(du, 1); % No initial input value

% Constraints
umax=[0.22;2.84]; %Turtlebot constraints
umin= [0; -2.84]; %Vmin = 0 if the robot is not allowed to go backwards.
In some cases, when Vmin = -0.22 it travels the whole path backward,
which may not be acceptable

delta_u_min = -10*0.1;
delta_u_max = 10*0.1;
posmax= 10;
posmin=-posmax;
delta_v_max = 15;
delta_v_min = -delta_v_max;

%% Definition of parameters
% Define control parameters
Q = 1*eye(dx); %cost for the state x
R = 1; %cost for the input u
Qy = 1*eye(dy); %cost for the output y
P = 1*Q; %cost for the terminal cost
Py = 75*Qy;

%number of predictions and simulations
Npred = 4;
Nsim = 200;
Ts = 0.5;

%Reference path
N=Nsim+Npred;
t=1:Ts:N;
r1=1; r2=1; f1=0.02; f2=0.12;
yref = [r1*sin(f1*t)+r1*sin(f2*t); r2*cos(f1*t)+r2*cos(f2*t) ]; %
Ellipsoid reference

%% Optimization problem using CasADi
import casadi.*

solver = casadi.Opti(); %usinf Opti class

% Define variables
x = solver.variable(3, Npred+1);
pos = solver.variable(2, Npred);
u = solver.variable(du, Npred);

xinit=solver.parameter(dx, 1);
uinit=solver.parameter(du, 1);
ypath = solver.parameter(dy, Npred);
```

```
% Initialize constraints
solver.subject_to(x(:,1) == xinit)
for k=1:Npred
    solver.subject_to(x(:,k+1) ==
x(:,k)+[u(1,k)*cos(x(3,k));u(1,k)*sin(x(3,k));u(2,k)]*Ts) %Dynamics of
the turtlebot
    solver.subject_to(pos(:,k) == x(1:2,k));

    solver.subject_to(umin <= u(:,k) <= umax) %Input magnitude
constraints
    solver.subject_to(posmin <= x(1:2,k) <= posmax) %Position magnitude
constraints
    if k==1
        solver.subject_to(delta_v_min <= u(1,k) - uinit(1,k-1) <=
delta_v_max) %Speed variations constraints
    else
        solver.subject_to(delta_v_min <= u(1,k) - u(1,k-1) <=
delta_v_max)
    end;
end

%% Objective
% Initialize objective
objective = 0;
for k=1:Npred
    objective = objective + (pos(:,k)-ypath(:,k))'*Qy*(pos(:,k)-
ypath(:,k)) + u(1,k)'*R*u(1,k); %Quadratic cost function
end
solver.minimize(objective);

% Define the solver
options.ipopt.print_level = 0;
options.ipopt.sb = 'yes';
options.print_time = 0;
solver.solver('ipopt', options);

%% Simulation
%simulation loop
usim=zeros(du, Nsim);
possim=zeros(dy, Nsim);
xsim=zeros(dx, Nsim+1);
xsim(:,1)=x0;
usim_init = u0;

timer = tic;
for i=1:Nsim
    solver.set_value(xinit, xsim(:,i));
    solver.set_value(uinit, usim_init);
    solver.set_value(ypath, yref(:,i:i+Npred-1));

    sol=solver.solve();
    usol = sol.value(u);
    usim_init = usol(:,1);
    usim(:,i)=usol(:,1);

    xsim(:,i+1) =
xsim(:,i)+[usim(1,i)*cos(xsim(3,i));usim(1,i)*sin(xsim(3,i));usim(2,i)]*T
s; %Dynamics
    possim(:,i) = xsim(1:2,i);
end
```

Appendix 4: Gantt chart of the project