Autonomous Vehicles

Assignment II: Feedback control of a differential drive robot

Tommaso Bocchietti 10740309

A.Y. 2024/25

POLITECNICO

MILANO 1863

Contents

1 Introduction **3**

2 System overview **3**
2.1 Request . . . 3
2.2 Waypoints . . . 3
2.3 Simulink model . . . 4

3 Proportional controller **5**
3.1 Implementation . . . 6
3.2 Results . . . 6

4 Lyapunov controller **8**
4.1 Lyapunov control law . . . 8
4.2 Implementation . . . 9
4.3 Results . . . 9

5 Conclusions **11**

List of Figures

1 Waypoints used for the simulation. . . . 4
2 Simulink model used for the implementation of the feedback control loop. . . . 4
3 Inside view of the controller block. . . . 5
4 Trajectory of the robot during the simulation with the proportional controller. . . . 7
5 Velocity profiles of the robot during the simulation with the proportional controller. . . . 7
6 Schematic representation of the unicycle-like system and the reference frame. . . . 8
7 Trajectory of the robot during the simulation with the Lyapunov controller. . . . 10
8 Velocity profiles of the robot during the simulation with the Lyapunov controller. . . . 11

1 Introduction

The aim of this work is to gain insight into the development of a feedback control loop for an autonomous vehicle, specifically a `Turtlebot3` robot of the model `burger`. Two different control strategies have been implemented, namely a simple proportional controller and a non-linear controller based on Lyapunov stability theory. As for the simulation environment, the `Turtlebot3` robot was simulated in a Gazebo world, specifically the `turtlebot3_world` provided by the `turtlebot3_gazebo` package.
The following sections provide a detailed description of the requests associated with this assignment, the approach taken to fulfill them, and the discussion of the results obtained.

Tools As for the tools used, `ROS1` (Robot Operating System) is employed as the main framework for communication between the different components of the system. `Simulink` is used as the main agent in the loop, sending control commands to the vehicles based on the telemetry data received from the vehicle itself. `MATLAB` instead is used to perform the analysis of the data collected during the simulation, and to visualize the results. Notice that with the current setup used by the author, `MATLAB 2024a` and `Simulink` are running in Windows 10, while `ROS1` is running in the `WSL2` (Windows Subsystem for Linux) environment, specifically in the `Ubuntu 22.04` distribution.

2 System overview

In this section, we provide a brief overview of the requests associated with this assignment, along with a description of the approach taken to fulfill them. In the successive two sections instead, we describe in detail the control strategies implemented, and the analysis of the results obtained.

2.1 Request

For this assignment, we are asked to implement a feedback control strategy that is able to drive a `Turtlebot3` robot of the model `burger` through some predefined waypoints in the environment.
In particular, the waypoints are specified as a list of 3D coordinates in the world frame, where the first two coordinates represent the position of the vehicle in the plane, and the third coordinate represents the orientation of the vehicle in the world frame.
Also, along with this core request, we are asked to implement two control strategy such as:

- A simple proportional controller, subjected to a set of constraint;
- A more advanced controller (no particular constraints on its choice).

2.2 Waypoints

Before proceeding with the actual implementation of the control system, we show in Figure 1 the waypoints used for this assignment. These waypoints are provided along with the text of the assignment.

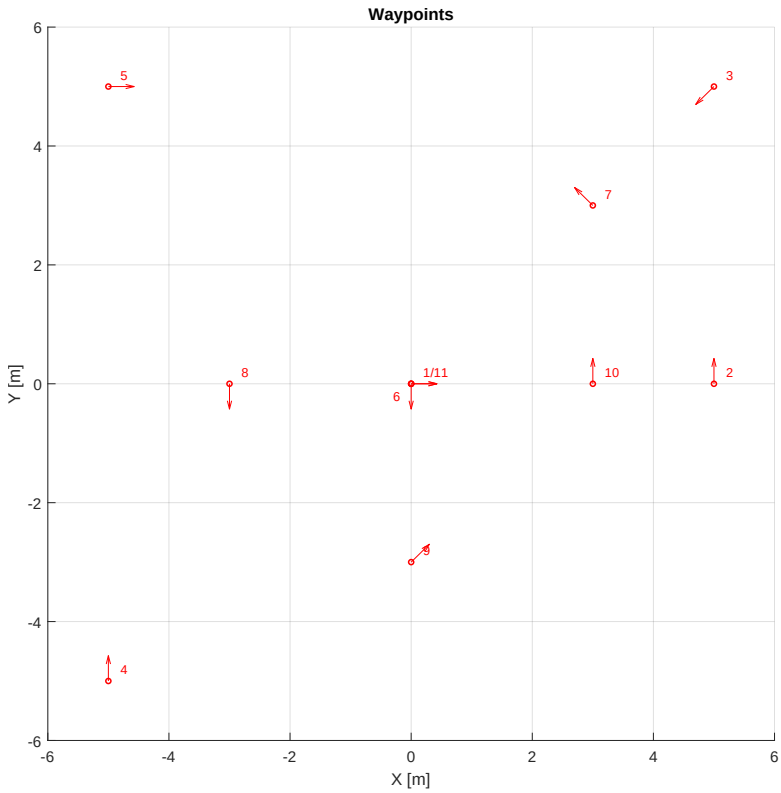

Figure 1: Waypoints used for the simulation.

In Figure 1, we can see the waypoints used for the simulation, which are represented as red dots in the world frame. The arrow associated to each waypoint represents the orientation of the vehicle at that waypoint, which is given by the third coordinate of the waypoint list.

2.3 Simulink model

As already mentioned in the previous section, the logic of the feedback control loop is implemented in `Simulink`, which is a graphical programming environment for modeling, simulating and analyzing dynamic systems. In Figure 2 we can see the Simulink model used for this assignment.

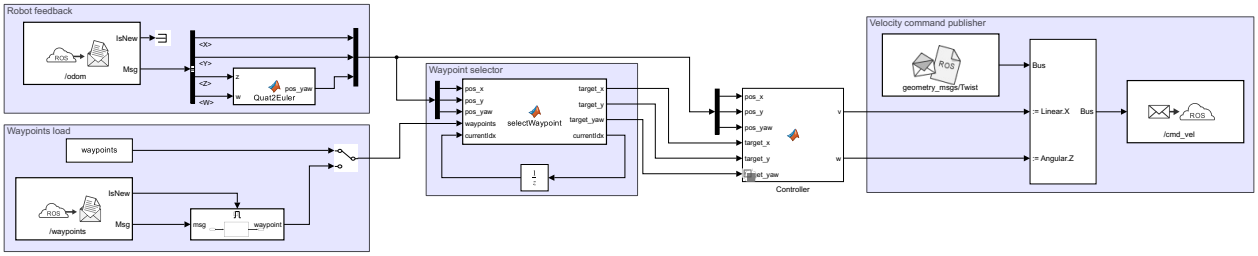

Figure 2: Simulink model used for the implementation of the feedback control loop.

One can easily recognize the different components of the system thanks to the colored areas used to group the different blocks. The main components of the system are:

- Sensors readings: these blocks are used to subscribe to the `/odom` topic and to extract the data needed for the control system;
- Waypoint loader: this block is used to load the waypoints from either a workspace variable or a custom topic `/waypoints` on which the list of waypoints is published;
- Waypoint selector: this block is used to select the current waypoint from the list of waypoints, based on the current position of the vehicle and its history;

- Controller: this block is responsible for the logic of the feedback control loop. It computes the control commands to be sent to the vehicle based on the current position and orientation of the vehicle, and the current waypoint to be reached. The controller is implemented as a MATLAB function block, which allows for a more flexible implementation of the control logic. More details on the implementation of the controller are provided in the next sections;
- Command publisher: this block is used to publish the control commands to the `/cmd_vel` topic, which is used by the vehicle to receive the control commands.

Notice that the commands sent to the vehicle are comprehensive of both linear and angular velocities. In fact, a differential drive model is used internally by the robot to compute the wheel velocities based on the linear and angular velocities provided by the control system.
Inside the controller block, we can find the two different control strategies implemented for this assignment, namely a simple proportional controller and a more advanced controller based on Lyapunov analysis. See Figure 3 for an inside view of the controller block.

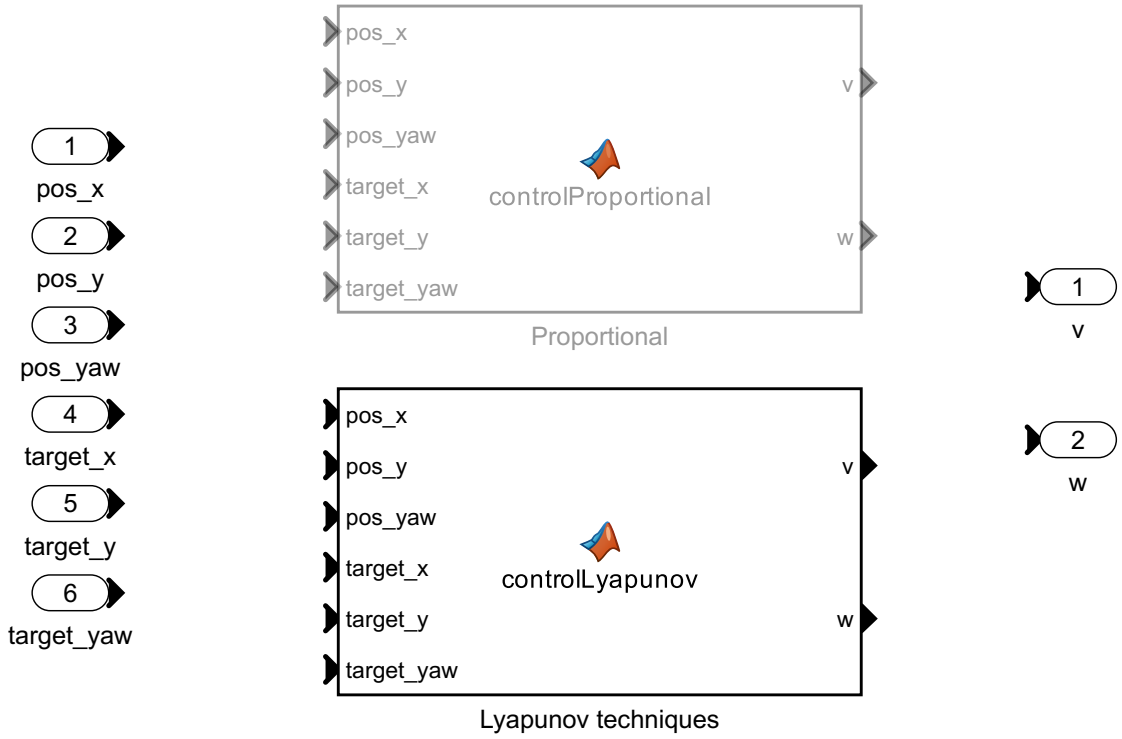

Figure 3: Inside view of the controller block.

3 Proportional controller

As explained in the previous section, one of the requests associated with this assignment is to implement a simple proportional controller for the `Turtlebot3` robot.
For this task, we have opted for a simple two-stage proportional controller. In particular, based on the distance of the robot from the current target waypoint, we apply a control input so that:

- If the robot is far from the target waypoint, we force its direction to be aligned with the target position, and we apply a linear velocity proportional to the distance from the target;
- Instead, when the robot is close to the target waypoint, we apply a control input that forces the robot to rotate in place, so that it can align its direction with the direction required by the target waypoint.

Some control constraints are also applied. In particular, we limit the maximum linear velocity to 0.2 [m/s] and the maximum angular velocity to 0.4 [rad/s]. Moreover, we consider the robot to be close to the target waypoint when the Euclidean distance from the target waypoint is less than 0.05 [m].

3.1 Implementation

The code for the implementation of the controller is provided in the Listing 1.

```
function [v, w] = controlProportional(pos_x, pos_y, pos_yaw, target_x, target_y,
    target_yaw)
% This controller align the robot with the target waypoint and guide it to
% reach it. Once within the threshold zone, correct the yaw of the robot.
% It uses simple Proportional controller for both actions.

bound = @(value, limit) max(min(value, limit), -limit);

% Gains
K_v = 1.2;
K_w = 1.5;
max_linear_speed = 0.2;
max_angular_speed = 0.4;

dx = target_x - pos_x;
dy = target_y - pos_y;
dist_to_target = hypot(dx, dy);
angle_to_target = atan2(dy, dx);

% Control logic
if dist_to_target > 0.05
    v = K_v * dist_to_target;
    w = K_w * wrapToPi(angle_to_target - pos_yaw);
else
    v = 0;
    w = K_w * wrapToPi(target_yaw - pos_yaw);
end

v = bound(v, max_linear_speed);
w = bound(w, max_angular_speed);

end
```

Listing 1: Code for the implementation of the proportional controller.

3.2 Results

Given the above implementation of the proportional controller, we can now analyze the results obtained during the simulation.
The results of the obtained trajectory are shown in Figure 4, where we report the waypoints highlighted in red and the trajectory of the robot drawn in black.

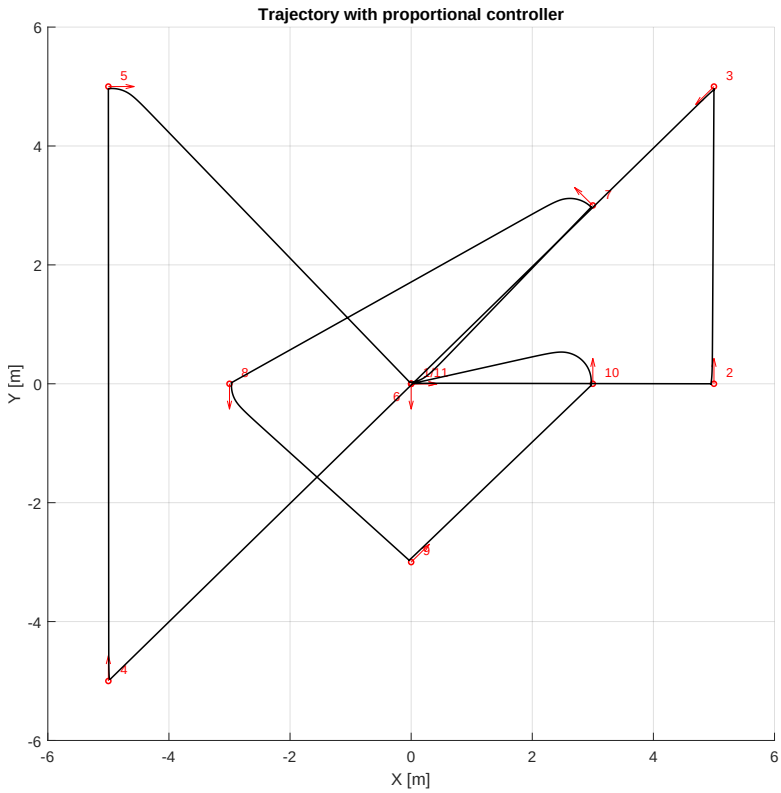

Figure 4: Trajectory of the robot during the simulation with the proportional controller.

One can easily observe that all the waypoints have been reached successfully.
When it comes to the velocity profiles, both in linear and angular dimension, it's easy to visualize the constraints applied to the controller. The linear velocity is limited to 0.2 [m/s], while the angular velocity is limited to 0.4 [rad/s]. This results in a trapezoidal profile for the velocities, as shown in Figure 5.

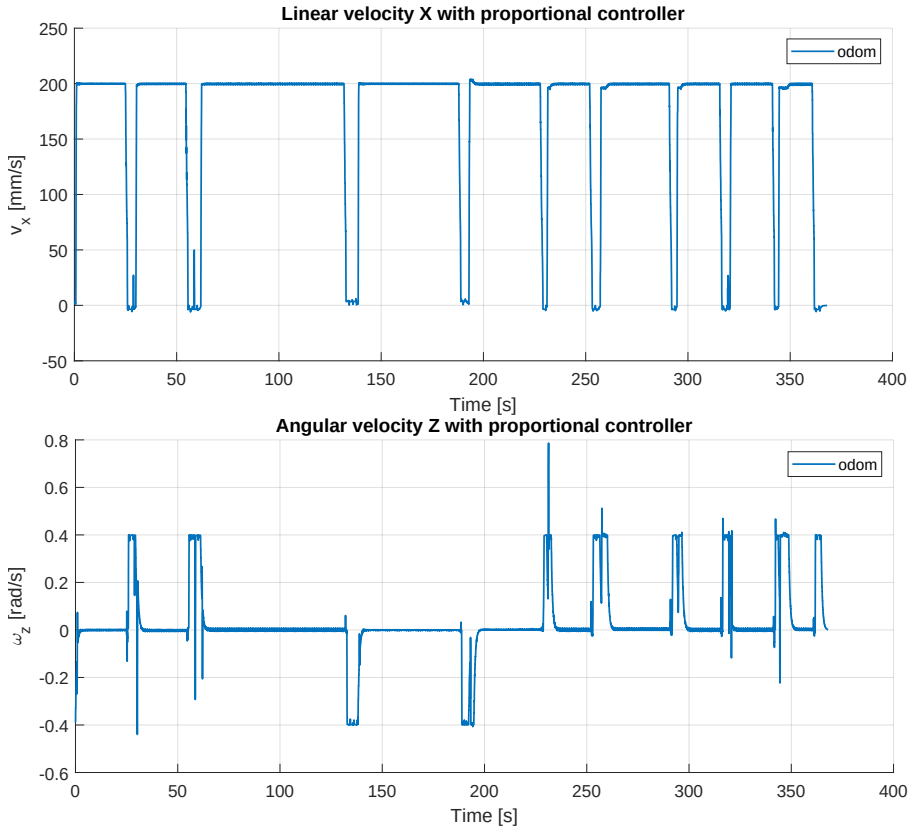

Figure 5: Velocity profiles of the robot during the simulation with the proportional controller.

One can also observe some severe (but luckily not too frequent) peaks in the velocities profiles that violate the constraints imposed. Similarly to the behavior observer in previous assignment, this fact find its explanation in poor performances of the machine on which the simulation is run. The simulation is run on a virtual machine with limited computational power, which results in some momentary oscillations in the robot's dynamics which are well captured by its telemetries. We think that a switch to a native environment of `Ubuntu 20.04` on a `Linux` kernel machine would solve this issue.

4 Lyapunov controller

In this section, we proceed with the implementation of a Lyapunov-based controller for the already defined system.
Lyapunov control theory is a powerful tool for designing controllers that proofs the stability of non-linear systems. The main idea is to find a Lyapunov function associated with the system and prove its convergence to the desired equilibrium point. Notice that global asymptotic stability is guaranteed only if the Lyapunov function is positive definite and radially unbounded.
Once the Lyapunov function is defined and its properties are verified, we can derive a control law that ensures the system's stability. Again, many methods from non-linear control theory exists such as the backstepping method, the feedback linearization method, or again passivation-based methods.
A final important remark here is that the control law derived from Lyapunov theory is not unique and generally speaking is also much more complex than the one derived from traditional linear control theory. Even if stability is guaranteed, the performance of the system may not be optimal and a proper tuning of the controller is not always straightforward.

4.1 Lyapunov control law

Based on the work of [1], we can derive a Lyapunov control law for a generic unicycle-like system. Given that the low level firmware of the `Turtlebot3` robot allows us to send control inputs in the form of linear and angular velocities, we can consider the turtlebot as a unicycle-like system and apply the Lyapunov control law to it.

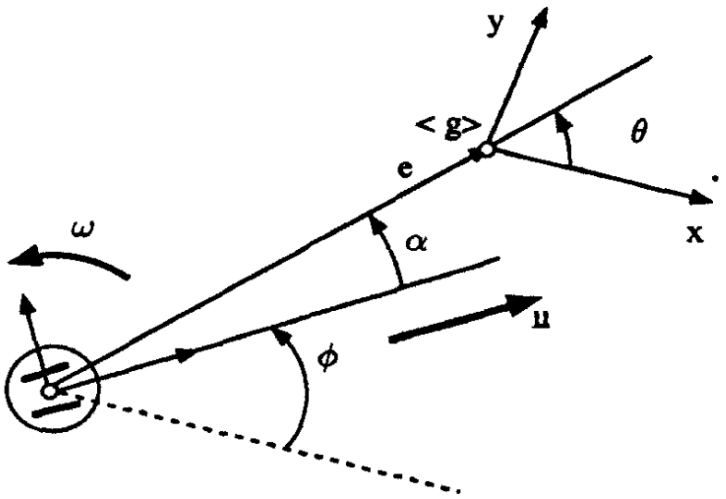

Figure 6: Schematic representation of the unicycle-like system and the reference frame.

Based on the notation used in Figure 6, we can define the following Lyapunov function:

$$v = k_1 \cos(\alpha) e \tag{1}$$

$$w = k_2 \alpha + k_1 \frac{\sin(\alpha)\cos(\alpha)}{\alpha} (\alpha + k_3 \theta) \tag{2}$$

Where k_1, k_2, and k_3 are positive constants, e is the Euclidean distance from the target waypoint, α is the angle between the robot heading and the target waypoint, and θ is the angle between the vector connecting the robot to the target waypoint and the waypoint direction (see Figure 6).

It turns out that with a proper remapping of the control inputs, we can obtain a Lyapunov function that is positive definite and radially unbounded, hence ensuring the global asymptotic stability of the system.

4.2 Implementation

The code for the implementation of the Lyapunov controller is provided in the Listing 2.

```
function [v, w] = controlLyapunov(pos_x, pos_y, pos_yaw, target_x, target_y, target_yaw)
% This controller implements a Lyapunov-based control law for the unicycle-like
% system. It uses a Lyapunov function to derive the control inputs for the robot.
% The controller is designed to ensure global asymptotic stability of the system.
% Inspired by: Aicardi et al. (1995)

bound = @(value, limit) max(min(value, limit), -limit);

% Limits
max_linear_speed = 0.8;
max_angular_speed = 1.6;

% Gains
K_rho   = 1.0;
K_alpha = 1.5;
K_delta = 1.0;

dx = target_x - pos_x;
dy = target_y - pos_y;
rho = hypot(dx, dy);
alpha = wrapToPi(atan2(dy, dx) - pos_yaw);
theta = wrapToPi(atan2(dy, dx) - target_yaw);

% Control law (from Aicardi paper)
v = K_rho * rho * cos(alpha);
w = K_alpha * alpha + K_rho * (sin(alpha) * cos(alpha)) + K_rho * (sin(alpha) * cos(
    alpha)) * K_delta * theta / max(abs(alpha), 1e-3) * sign(alpha);

v = bound(v, max_linear_speed);
w = bound(w, max_angular_speed);

end
```

Listing 2: Code for the implementation of the Lyapunov controller.

4.3 Results

The results of the obtained trajectory are shown in Figure 7, where we can see the waypoints highlighted in red and the trajectory of the robot drawn in black.

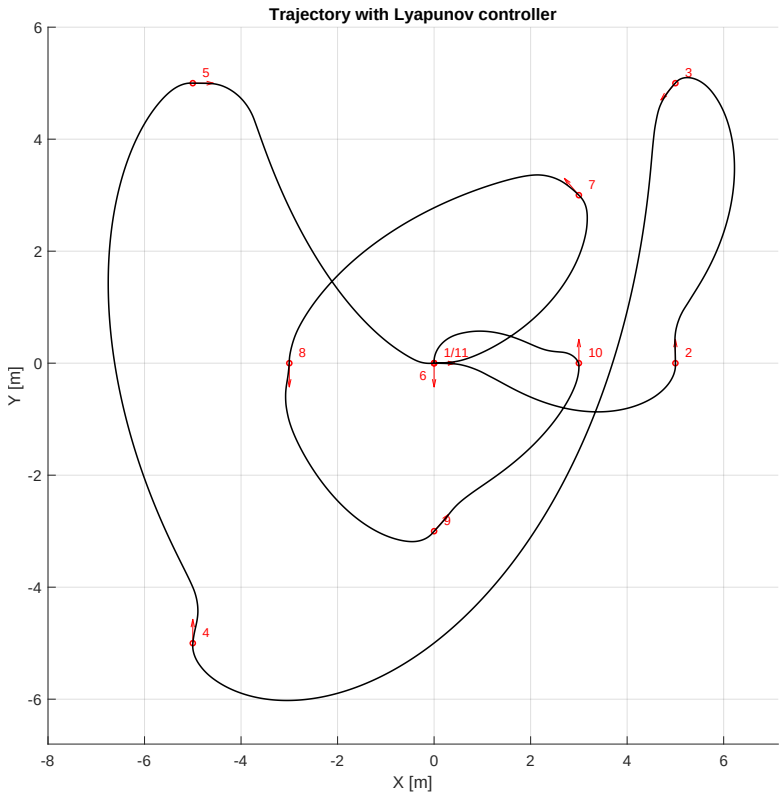

Figure 7: Trajectory of the robot during the simulation with the Lyapunov controller.

With respect to the proportional controller, we can see that the Lyapunov controller is able to smoothly go through the waypoints and to reach them avoiding stop-and-go behavior. The if-else logic is not needed any more given that the control law is able to smoothly connect the waypoints weighting the timing and amplitude for steering and velocity commands based on the 3 parameters k_1, k_2, and k_3. One could also modify the parameters to obtain a more aggressive or conservative behavior of the robot, with consequent changes in the trajectory.
When it comes to the velocities profile, we observe that linearity is lost and non-linear behavior is present.

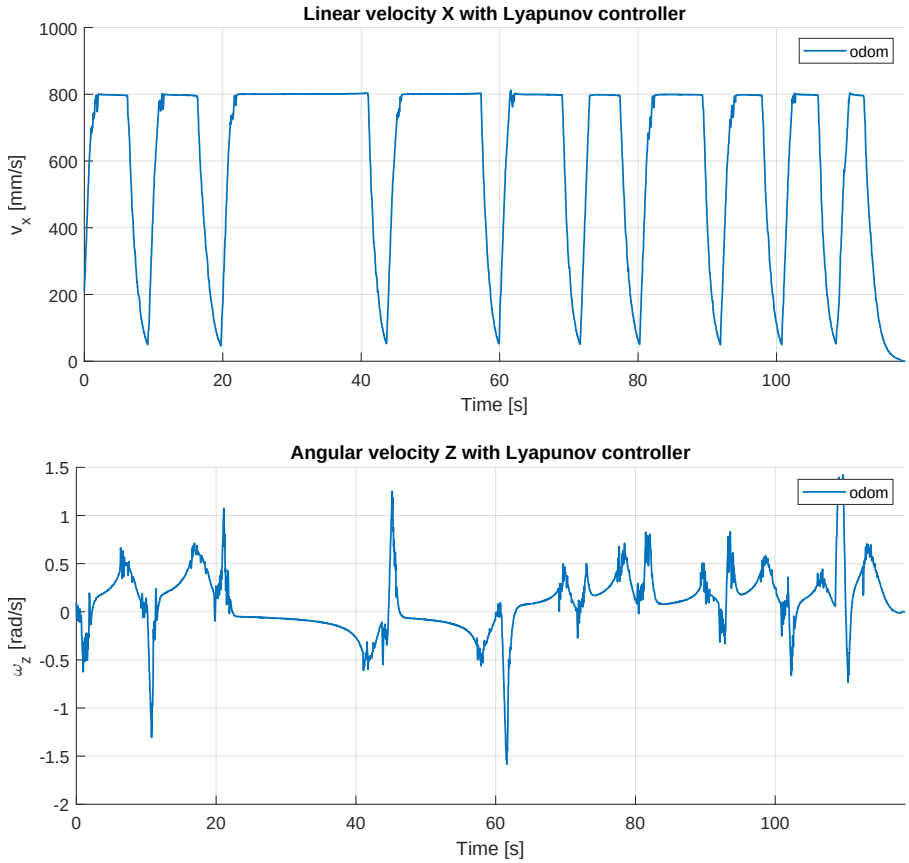

Figure 8: Velocity profiles of the robot during the simulation with the Lyapunov controller.

Notice that with respect to the simulation run with the proportional controller, the linear velocity is now not limited to 0.2[m/s], but it can reach up to 0.8[m/s], and similarly the angular velocity can reach up to 1.6[rad/s]. The author has decided to increase the limits of the velocities in order to allow the robot to reach the waypoints in a shorter time. Nevertheless, the no-slip condition generally required for the unicycle-like system has been checked and verified even with the presence of stronger inertial forces due to the higher velocities.

5 Conclusions

In this short report, we have presented the results of the second assignment of the course on Autonomous Vehicles. We have shown how to implement a feedback control strategy for a mobile robot, specifically the `Turtlebot3` robot, in a simulated environment using `ROS1` and `Gazebo`.
Even if not extensively discussed in this report, the `Simulink` system has allowed to easily load waypoints from different sources (i.e. workspace or ROS topic) and no differences in the results have been observed.
Most importantly, we have shown how to implement a simple proportional controller and a Lyapunov-based controller for the robot. While both have shown to be effective in guiding the robot to the target waypoints, the Lyapunov controller has demonstrated a native smoothness in the trajectory, avoiding stop-and-go behavior.
Some unwanted spikes in the telemetry of the robot have been observed during the simulation, which could be due to the fact that the simulation was run on a `Windows` environment connected to a `WSL` (Windows Subsystem for Linux). Further investigations and testing could be done by switching to a native `Linux` environment, which could provide better performance and more accurate results.

References

[1] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino. Closed loop steering of unicycle like vehicles via lyapunov techniques. *IEEE Robotics & Automation Magazine*, 2(1):27–35, 1995.

[2] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.