Autonomous Vehicles
Assignment I: Use of ROS bags

Tommaso Bocchietti 10740309

A.Y. 2024/25

Contents

List of Figures

1 Introduction

The aim of this work is to gain insight into the use of `rosbags` for the analysis of data collected by autonomous vehicles.
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 data collection, while their analysis is performed using `MATLAB`. Notice that with the current setup used by the author, `MATLAB 2024a` is running in Windows 10, while `ROS1` is running in the `WSL2` (Windows Subsystem for Linux) environment, specifically with the `Ubuntu 20.04` distribution.

2 Assignment Part A

In this section, we provide a brief overview of the requests associated with the first part of the assignment, along with a description of the approach taken to fulfill them. Discussion of the results obtained is also included.

2.1 Request

Starting from the data collected in the provided `rosbag` file, the goal of this first part of assignment is to evalute at each time instant, the following quantities:

- **Minimum distance** of the vehicle from the obstacles in the environment;
- **Estimate of `/cmd_vel` command** sent to the vehicle during the simulation.

We also recal that the `rosbag` file was obtained by running a `Turtlebot3` robot of the model `burger` in a simulated environment, specifically the `turtlebot3_world` provided by the `turtlebot3_gazebo` package.

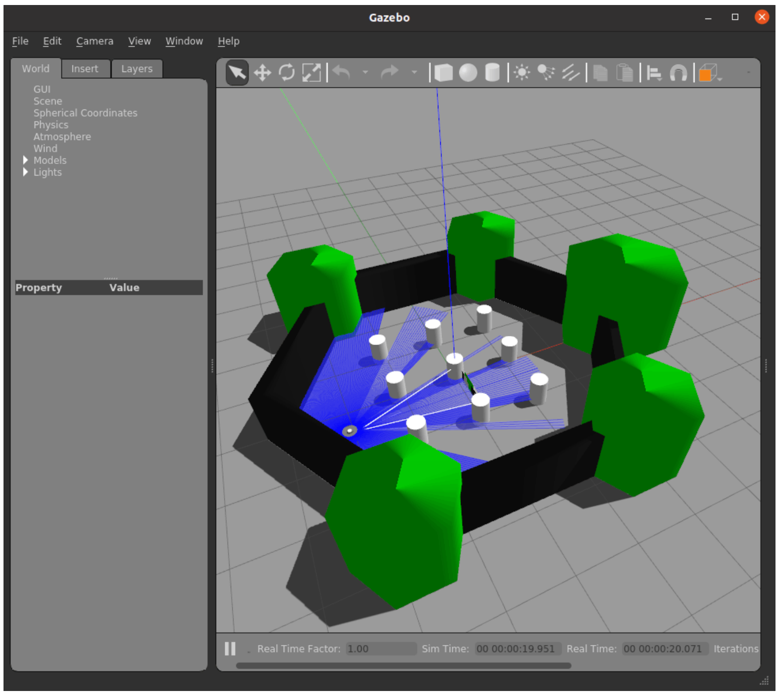

Figure 1: Gazebo world used for the simulation. Credit to `https://emanual.robotis.com/`

2.2 Analysis

At first, the `rosbag` file is loaded into `MATLAB` using the `rosbag` function, which allows to read and manipulate `rosbag` files in a convenient way. A preliminary analysis of the data contained in the `rosbag` file is performed showing the following topics:

	NumMessages	MessageType
/clock	110,710	rosgraph_msgs/Clock
/imu	110,240	sensor_msgs/Imu
/odom	3,316	nav_msgs/Odometry
/scan	552	sensor_msgs/LaserScan
/tf	3,316	tf2_msgs/TFMessage

Table 1: Topics contained in the `rosbag` file.

2.2.1 Minimum distance from obstacles

Given the presence of the `/scan` topic, which contains the laser scan data, we can use it to compute the minimum distance from obstacles in the environment. The `scan` message contains a field called `ranges`, which is an array of distances measured by the laser scanner at different angles. The minimum distance can be computed by taking the minimum value of this array, which represents the closest obstacle detected by the laser scanner at that time instant. The following code snippet shows how to extract the `ranges` field from the `/scan` topic and compute the minimum distance at each time instant:

```
scan = select(bag, 'Topic', '/scan');
scan_msgs = readMessages(scan, 'DataFormat', 'struct');
scan_time = scan.MessageList.Time - scan.StartTime;
scan_ranges = cell2mat(cellfun(@(msg) msg.Ranges(:)', scan_msgs, 'UniformOutput', false)
    );
min(scan_ranges, [], 2)
```

Listing 1: Extracting the `ranges` field from the `/scan` topic and computing the minimum distance at each time step.

One can also plot the minimum distance over time, as shown in Figure 2.

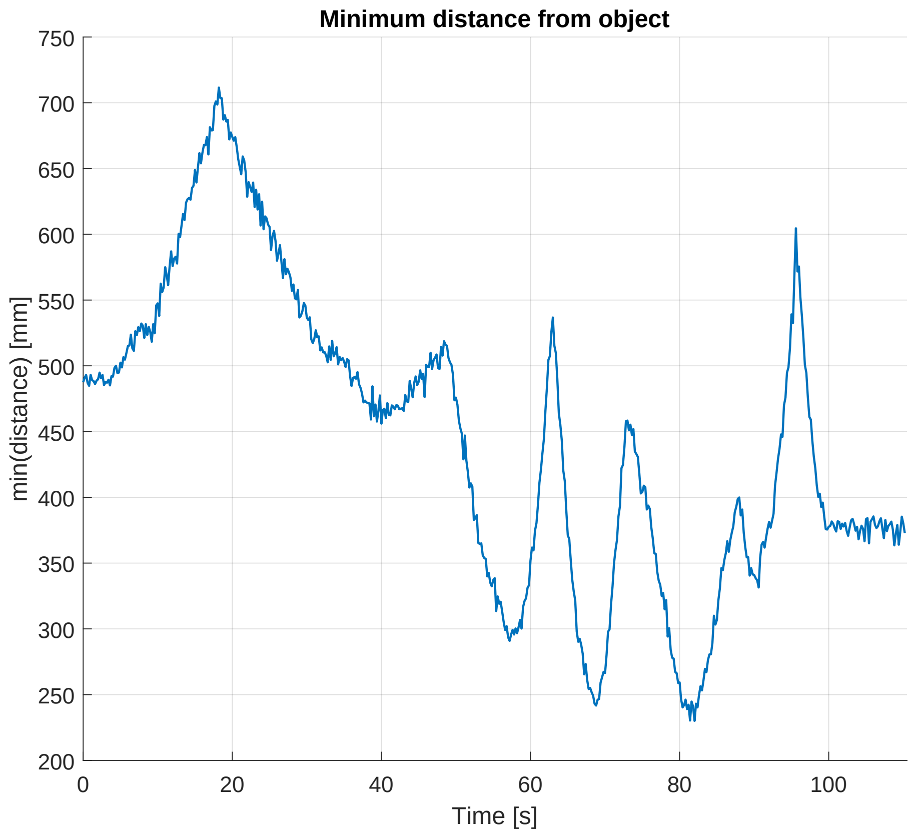

Figure 2: Minimum distance from obstacles over time.

It's important to notice that, being the data coming from a simulated environment, we can expect the accuracy of the laser scanner to be almost perfect (unless the simulation environment add noise on purpose), while on a real vehicle we could face issues like sensor noise or hit of minimum and/or maximum range of the laser scanner.

This could lead to a less accurate estimation of the minimum distance from obstacles, which could be a problem for the robot navigation and obstacle avoidance.

2.2.2 Estimate of `/cmd_vel` command

The `/cmd_vel` topic usually contains the velocity commands sent to the robot, which are typically represented as linear and angular velocities.
As an exercise, given that in the provided `rosbag` file the `/cmd_vel` topic has been removed, we can estimate the linear and angular velocities of the robot using the `/odom` topic, which contains the odometry data. The `odom` message contains the position and orientation of the robot in the world frame, as measured by the odometry system.
Again, with a similar faschion as done for the `/scan` topic, we can extract the information from the `/odom` topic, save them in a matrix form and plot them.

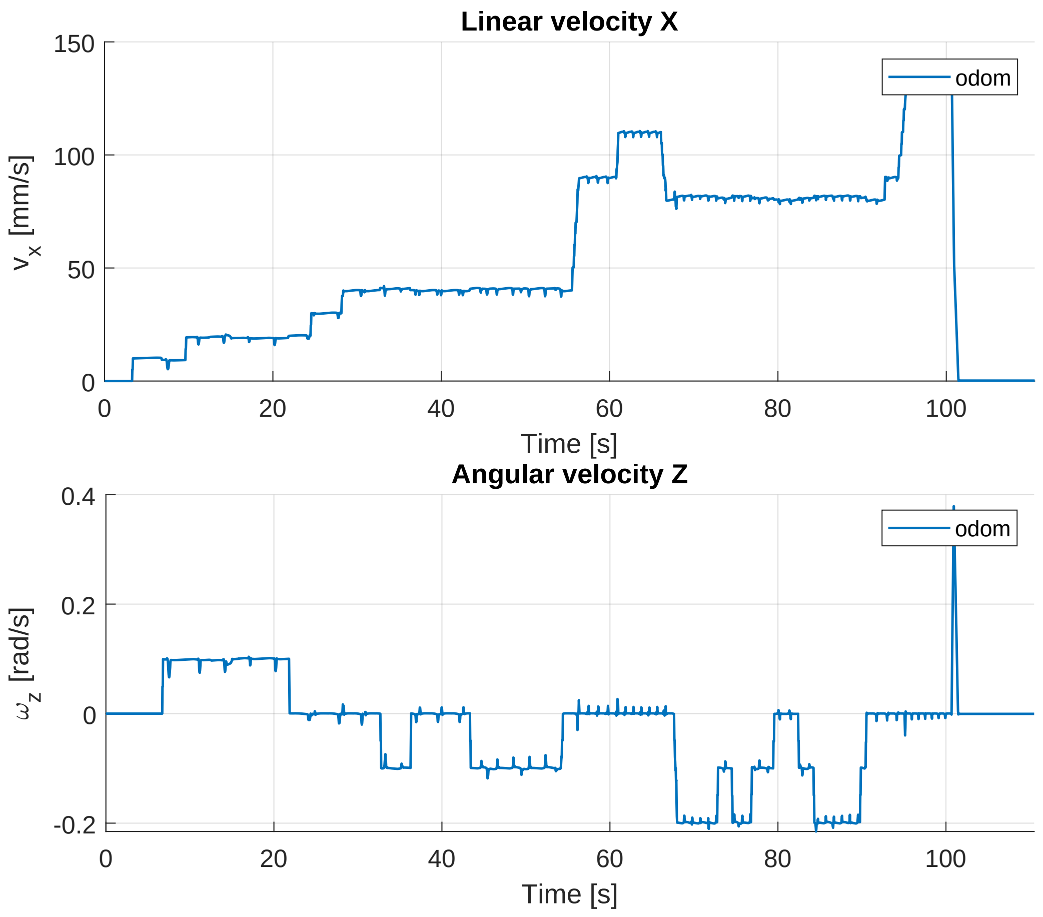

Figure 3: Velocities of the robot over time, purely based on the odometry data.

Notice that one could have also possibly used the `/imu` topic to estimate both position and velocities of the robot (via single and double integration of the acceleration data). However, due to the presence of noise in the IMU data and cumulative errors due to integration, this approach is definitely not recommended as it easily results in drifted output estimation.
On the other hand, a sensor fusion approach based for example on the Kalman filter could have been used to combine the two sources of data (IMU and odometry) ultimately increasing the overall accuracy.

3 Assignment Part B

In this section, we provide a brief overview of the requests associated with the second part of the assignment, along with a description of the approach taken to fulfill them. Discussion of the results obtained is also included in the following sections.

3.1 Request

Given that in the previous section we have estimated the control commands sent to the robot during the original simulation, in this second part we are asked to evaluate the accuracy of such commands.

3.2 Analysis

In order to evaluate the accuracy of the control commands sent to the robot during the simulation, we need to rerun a simulation by our own, using the same world and the same robot model, and then compare both the trajectory and the velocities with the ones saved in the original `rosbag` file.
We recall that our working environment is setup so that `ROS1` is running in the `WSL2` (Windows Subsystem for Linux) environment, specifically in `Ubuntu 20.04`, while `MATLAB 2024a` is running in Windows 10. Based on these considerations, we can run the simulation in at least three different ways:

- Create a publisher script in `MATLAB` that sends the commands to `ROS1` running in `WSL2`;
- Create a publisher in `Simulink` that sends the commands to `ROS1` running in `WSL2`;
- Perform the analysis directly in `WSL2` using the native commands of `rosbag` to automatically replay the estimated commands.

At first, we tried to create a publisher script in `MATLAB` that was sending the commands over the bridged network, pacing the sender based on the original timing of the `/odom` topic. However, we encountered some major issues with the timing of the commands, which were sent at almost unpredictable intervals, leading to a completely different behavior of the robot in the simulation.
In order to avoid or at least reduce the issues related to the timing of the commands, we decided to directly move to the third option, which is to perform the analysis directly in `WSL2` using the native commands of `rosbag` to automatically replay the estimated commands. At first, the `/odom` topic has been ported to a new `rosbag` file using a simple `python` script, which is able to read the original `rosbag` file and write the `/odom` topic to a new `rosbag` file while also reshaping the data to match the type of the `/cmd_vel` topic. In particular, the following relationship table has been implemented:

/odom	**/cmd_vel**
msg.Twist.Twist.Linear.X	msg.Linear.X
msg.Twist.Twist.Angular.Z	msg.Angular.Z

Table 2: Relationship between the `/odom` and `/cmd_vel` topics.

Once the content of the `/odom` topic has been saved as `/cmd_vel` topic in a new `rosbag` file, we can use the `rosbag play` command to replay the commands in the simulation.
In Figure 4 we can see the comparison of the trajectories performed by the original simulation and the one performed by the replayed commands.

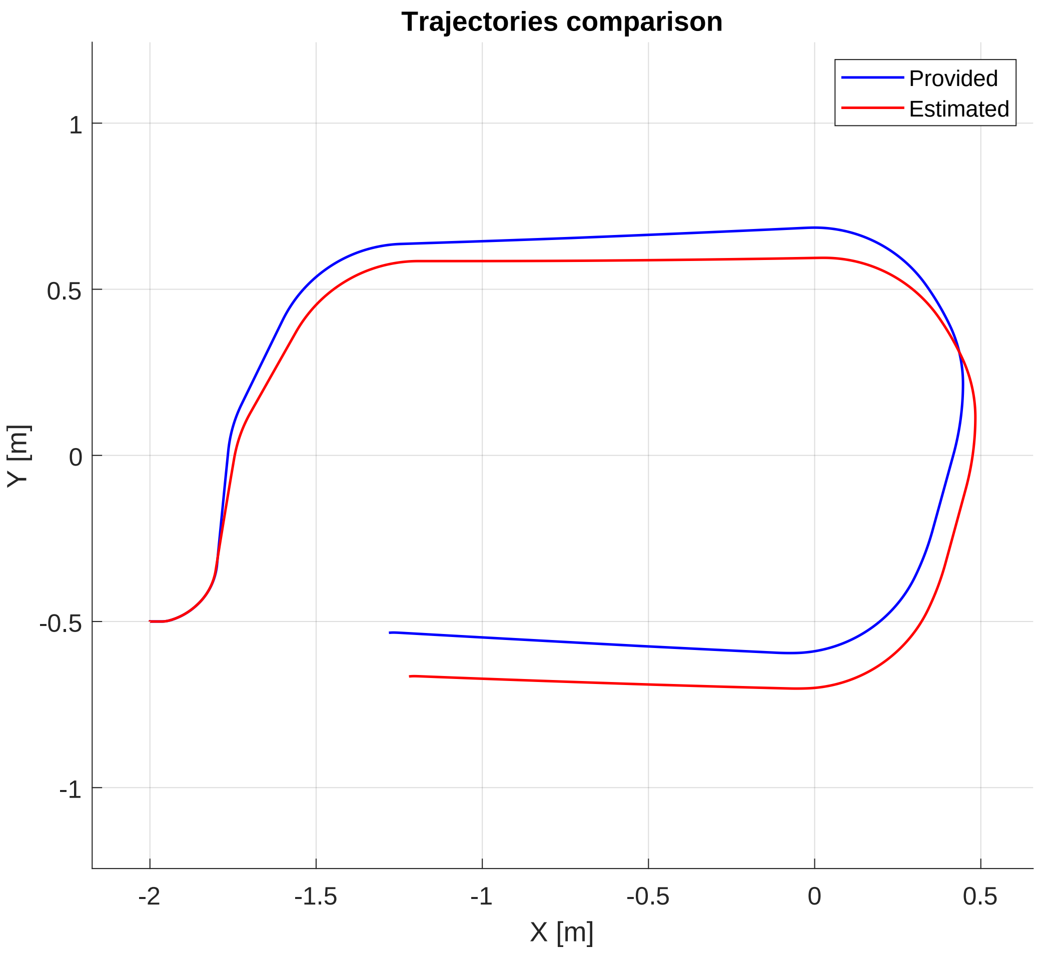

Figure 4: Comparison of the trajectories performed by the original simulation and the one performed by the replayed commands.

One can clearly see that the two trajectories are not completely overlapping.
Looking at the values of the actual velocities of the robot during the simulation, we can see a minor discrepancy between the original simulation and the one performed by the replayed commands, similarly to the behaviour observer for the trajectories. Figure 5 shows the comparison of the velocities performed by the original simulation and the one performed by the replayed commands.

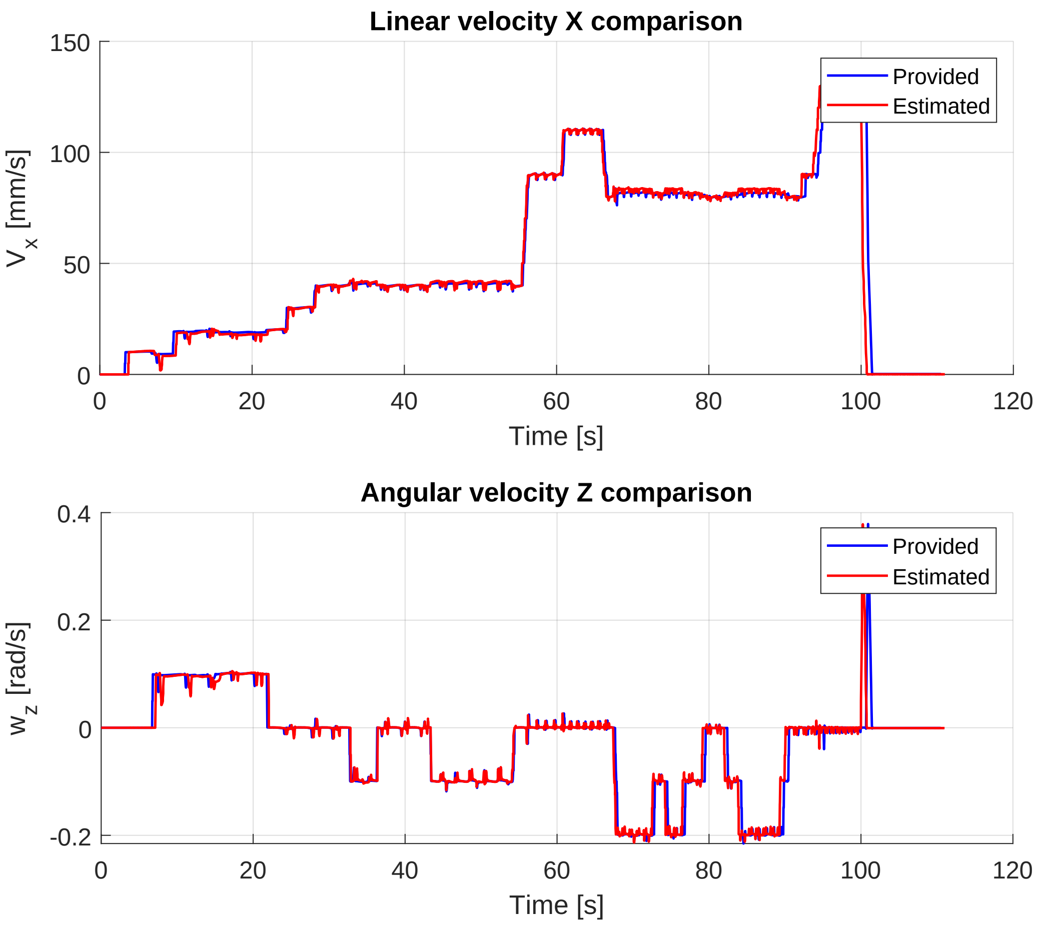

Figure 5: Comparison of the velocities performed by the original simulation and the one performed by the replayed commands.

Again, some minor issues related to timing can be observed. For some reason that the author is not able to determine, the `rosbag play` command is not able to replay the commands at the exact same pace as the original simulation, leading to a slight shift in the velocities.
What is also interesting to notice, are the juggling of the velocities along the whole simulation. These are, in fact, not desired behaviors and most probably caused by the simulated environment itself.

4 Conclusions

In this short report, we have presented the results of the first assignment of the course on Autonomous Vehicles. We have shown how to analyze data collected by a simulated robot in a Gazebo environment, using the `rosbag` command to store the data. We have also demonstrated how to estimate the control commands sent to the robot during the simulation, and how to evaluate their accuracy by comparing trajectories and odometer readings.
Results obtained show that the estimated commands are representative of the original ones, but not completely overlapping due to poor timing performance caused (probably) by the `WSL2` environment that can't fully afford the real-time performance required by the simulation.
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] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.