Project Report

Composed as part of the course : Robotics project S9

about

TURTLEBOT3

from

DUMAND Nathan

and

LABAUVIE–RAFFAELLI Eva

Study program : Robotics and Autonomous Systems
Year : 2024 - 2025
Course : Robotics Project
Supervised by : Mr Juan, Mr Lebreton, Mr Jacob, Ms Hait Hassou
Submission deadline : January 26th, 2025

January 26, 2025

Table of Content

1 Introduction **4**
1.1 General Idea 4
1.2 Team Assignments 4

2 Groundwork **4**
2.1 Installation 4

3 Topic **5**

4 Navigation **6**
4.1 Motion 6
4.2 Mapping 7
4.3 Semi-autonomous Navigation 8

5 Trajectory tracking **10**
5.1 Kalman filter 10

6 Conclusion **14**

7 Sources **14**

List of Figures

1 Node Publisher/Subscriber 6
2 Teleoperation node 7
3 Rviz Mapping screen 8
4 Rviz Navigation screen 9
5 Robot Gazebo model for the simulation 12
6 Simulation model for the robot 12
7 Test of the Kalman filter on simulation 13
8 Test of the Kalman filter on simulation 14

1 Introduction

1.1 General Idea

For our semester project, we will focus primarily on the programming aspect of the robot. To achieve this, we will use the Turtlebot3 platform as a foundation, where we will implement the features and test our code directly.
Our objective is to develop an autonomous system that enables the robot to navigate its environment independently.
The robot will use its built-in LiDAR sensor to avoid obstacles and simultaneously create a detailed map of its surroundings in real-time.

1.2 Team Assignments

- **Nathan**: Focuses on mapping and connection to the robot.
- **Eva**: Focuses on robot movement and tracking its path, which will be displayed on a user interface.

2 Groundwork

2.1 Installation

After mounting the robot, we decided to work with ROS2 Humble because of the documentation provided with the robot. Most of the programs were designed to be easily applied with the Turtlebot across different versions of ROS. To prepare and monitor the system, we installed Ubuntu 22.04 LTS on our computer.
To install ROS2 on the robot, we needed the corresponding Ubuntu version for it to work. We first tried to install it directly on the Raspberry Pi 4.

- At first, we attempted to install the desktop version. However, we had to stop as it took up too much memory, and the SD card became saturated. The process was also very slow due to the school's internet connection.
- Next, we tried the server version. But again, we faced issues connecting the robot to the school's network because of the portal login system. Without a desktop environment, we couldn't access the identification portal to input the necessary credentials.
- Finally, we used our phone's mobile network to avoid further issues.

Unfortunately, after updating and installing the Linux environment, the SD card became full again because of the ROS installation. We initially used a 16GB SD card, which was too small for the task. It took us 2 to 3 weeks to receive a new 64GB SD card.
With the new card's larger capacity, we reinstalled the desktop version and no longer had memory issues. This allowed us to successfully set up the ROS2 environment tailored for the Turtlebot3. However, we encountered a new problem: the Raspberry Pi 4's computational power wasn't sufficient to handle the builds. For example, the "colcon build" function, which is needed to compile the project and access the robot's features, was too demanding. We tried deleting unnecessary programs to free up resources for a test run,

but the programs were still too large for the Raspberry Pi to handle.
By this time, it was already December. After many attempts to troubleshoot the issues and optimize the installation, we decided to give up on ROS2. We considered writing and running our own program from scratch, but that would have meant losing all the advantages of using a Turtlebot and starting over. With so much time already lost, this wasn't a viable option.

Instead, we switched to ROS1 Noetic. Since this version is supported only on Ubuntu 20.04, we reinstalled Ubuntu 20.04 on our computer. We backed up all our important files before making the change. For the Raspberry Pi, we flashed the pre-configured ROS1 image onto the new SD card. This allowed us to start using the robot almost immediately after some minor updates.
As the pre-configured image was based on the server version of Ubuntu 20.04, we connected a screen to set up the WiFi. Then, we configured an SSH connection to control the robot remotely from our computer. Using the "ifconfig" command, we located the robot's IP address.
Once the SSH connection was established, programming the robot and running the necessary programs became much easier. With ROS1 Noetic installed and configured, we could use the robot's default movement parameters, such as translation and rotation. From there, we were finally able to begin working effectively.

3 Topic

Ros work through Node. Each node has a specific function and can be use to post a topic (publisher) or to read (subscriber) the data.
(Figure 1) All the data processed by the turtlebot are made available on different Topic. Each topic possess different information such as the battery level, position, movement of the robot and so on. This structure allows us to access all this data and process it ourselves in MATLAB.
For our simulation and to use the Kalman Filter we created on another class, we subscribe to the IMU Topic and Odometry Topic.
With a bit more time, we would have also been able to draw the map thanks to the cloud point on MATLAB.

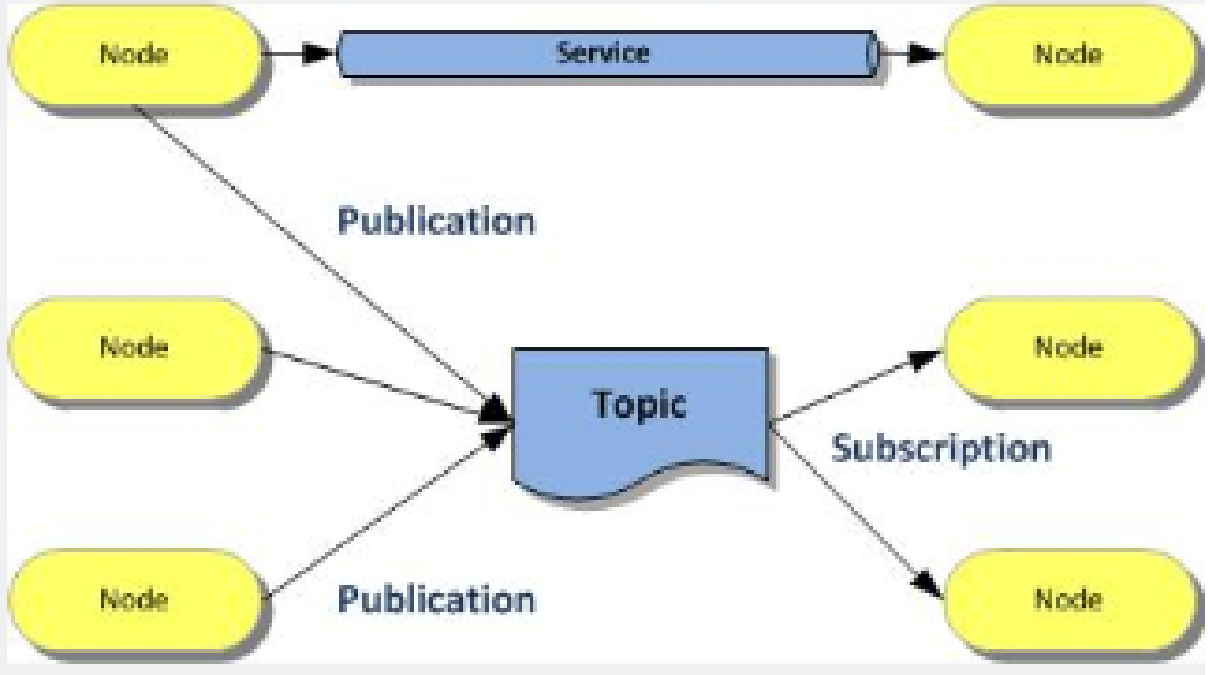

Figure 1: Node Publisher/Subscriber

4 Navigation

One of our goals was to give to the robot the ability to navigate alone. We were not able to make it explore alone but were able to chose a destination on an already map zone and it will go there with no more information from us.
To do so, we need to set the robot in movement and allow it to create a map.

4.1 Motion

One of the first tools we set up and tested was the motion control. We could try it using the test buttons on the OpenCR board. This board is similar to an Arduino and is used to manage the robot's power and motors. The two buttons allowed us to either move the robot forward by 30 centimeters or rotate it 180°.

Afterward, we started using the Teleoperation Node. This operation enabled us to monitor and control the robot's movement from a remote computer. We could adjust the robot's linear and angular speeds and receive feedback on its movements. These data were computed using the wheel encoder, gyroscope, and accelerometer.
We also tried controlling the robot's movement using a DualShock4 (PlayStation 4 joystick). However, we discovered that the robot didn't have a Bluetooth sensor, making it impossible to connect the controller. If the robot had been equipped with one, it would have been easier to manage its direction using the joystick.

Teleoperation from the computer was done using the keyboard. However, I wasn't able to modify the key mapping, so we had to use the default QWERTY keys: W, X, A, and D for movement. (Figure 2)

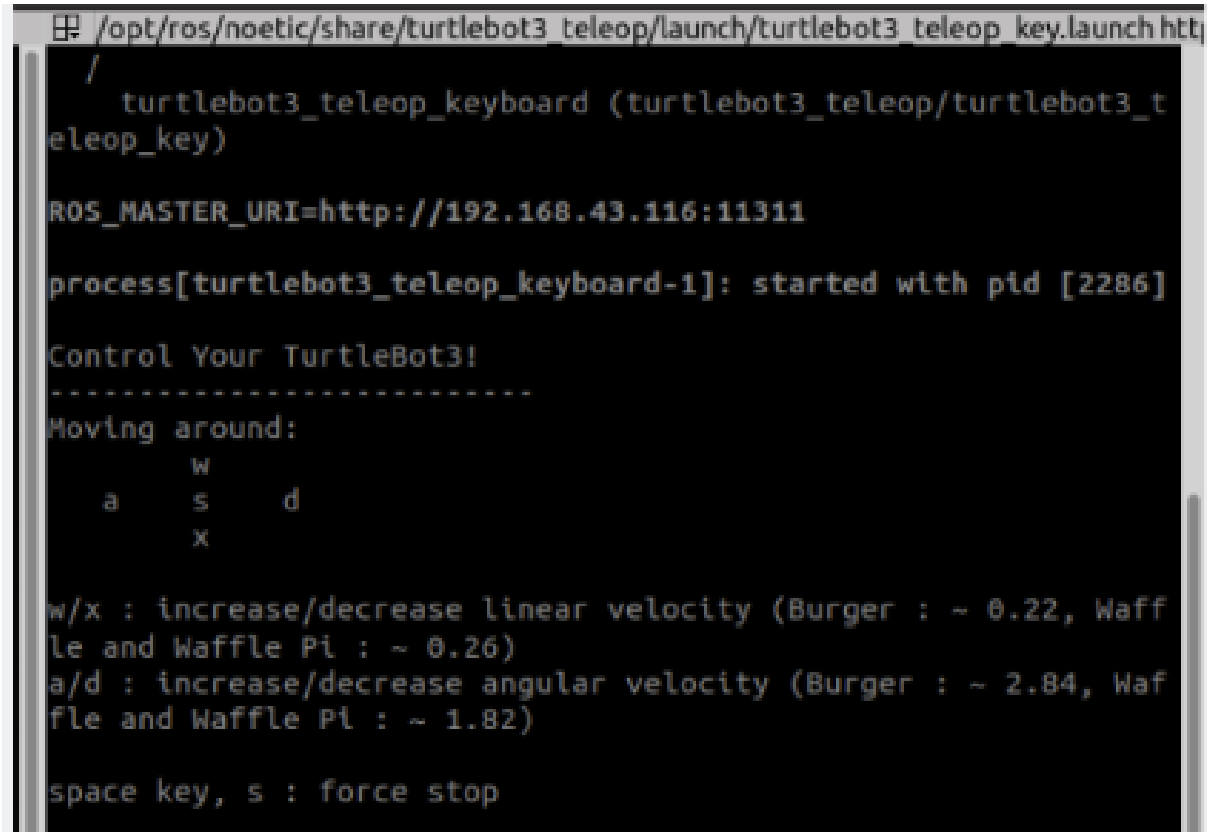

Figure 2: Teleoperation node

4.2 Mapping

Once we were able to move the robot, we start trying to use the SLAM (Simultaneous, Localization And Mapping). To do so, we execute the SLAM Node, numerous information are show on Rviz, a graphical interface that show the cloud point treat by the computer. The Lidar will detect obstacles with a precision of 0.5mm. It is really precise but have some dead angle. During this part, the robot will no avoid obstacle if we are not careful. What's interesting is the fact that the Lidar is able to make SLAM, so we can navigate and map at the same time.
(Figure 3) Fixed obstacles are created when the robot does not move for a set amount of time. During one of the tests, one of us was mistakenly detected as an irremovable obstacle.
One way to improve the mapping would be to add a camera that would allow to improve the quality of the image and put background information on the map.
The map can be save and use as a base to implement half-autonomous navigation on the robot.

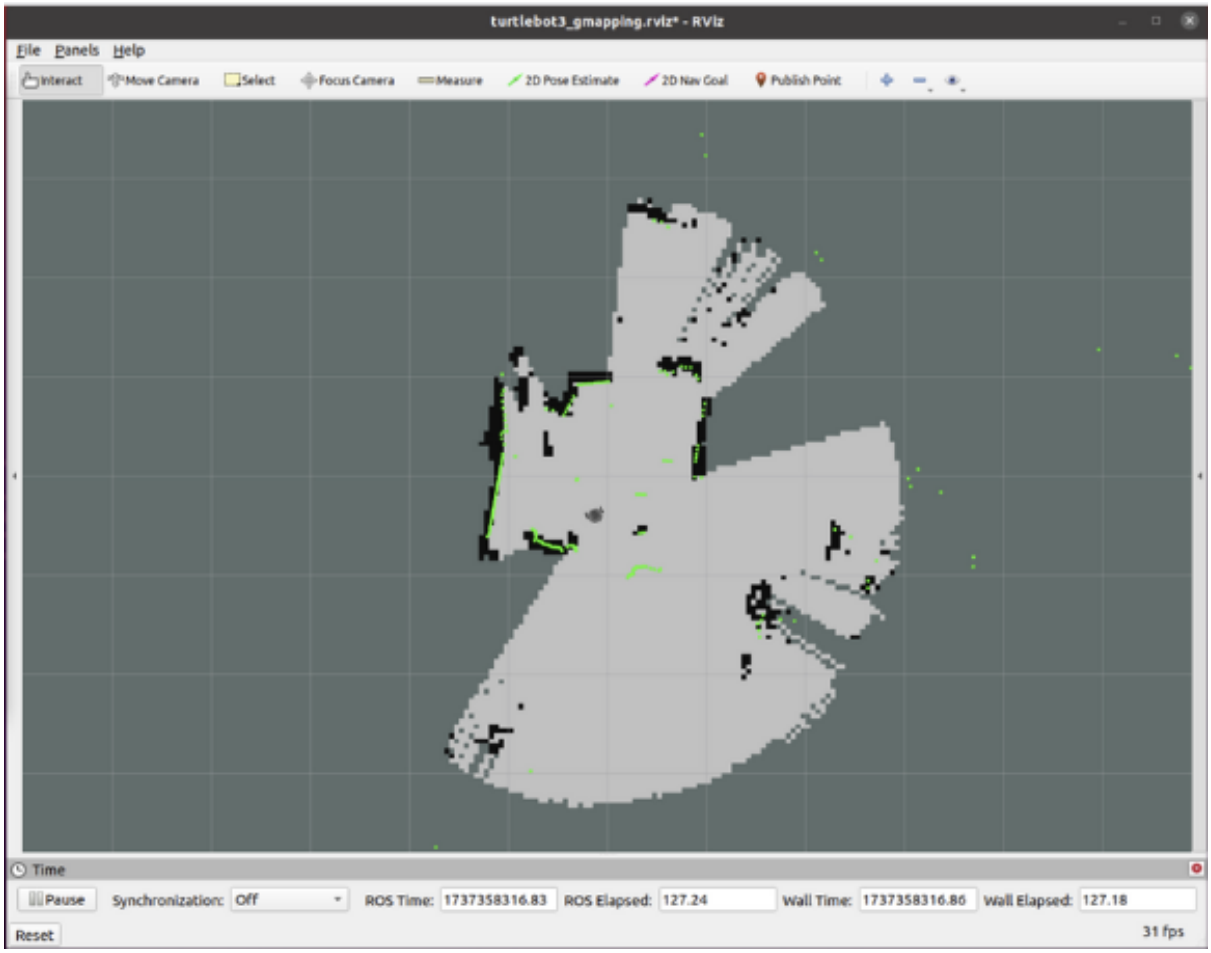

Figure 3: Rviz Mapping screen

4.3 Semi-autonomous Navigation

Once the map is drawn, we can start using the navigation tool. This one will allow the Turtlebot to take the information he previously map, or from a download map and he will be able to go from point A (starting point) to point B chosen on an already discover area.
He will avoid any obstacle, even if it's one that was not on the map. He will bypass any obstacle detect from a minimal distance thanks to the Lidar.
The depending of the parameter, it can go through undiscovered area, and also go closer or further away from obstacle.

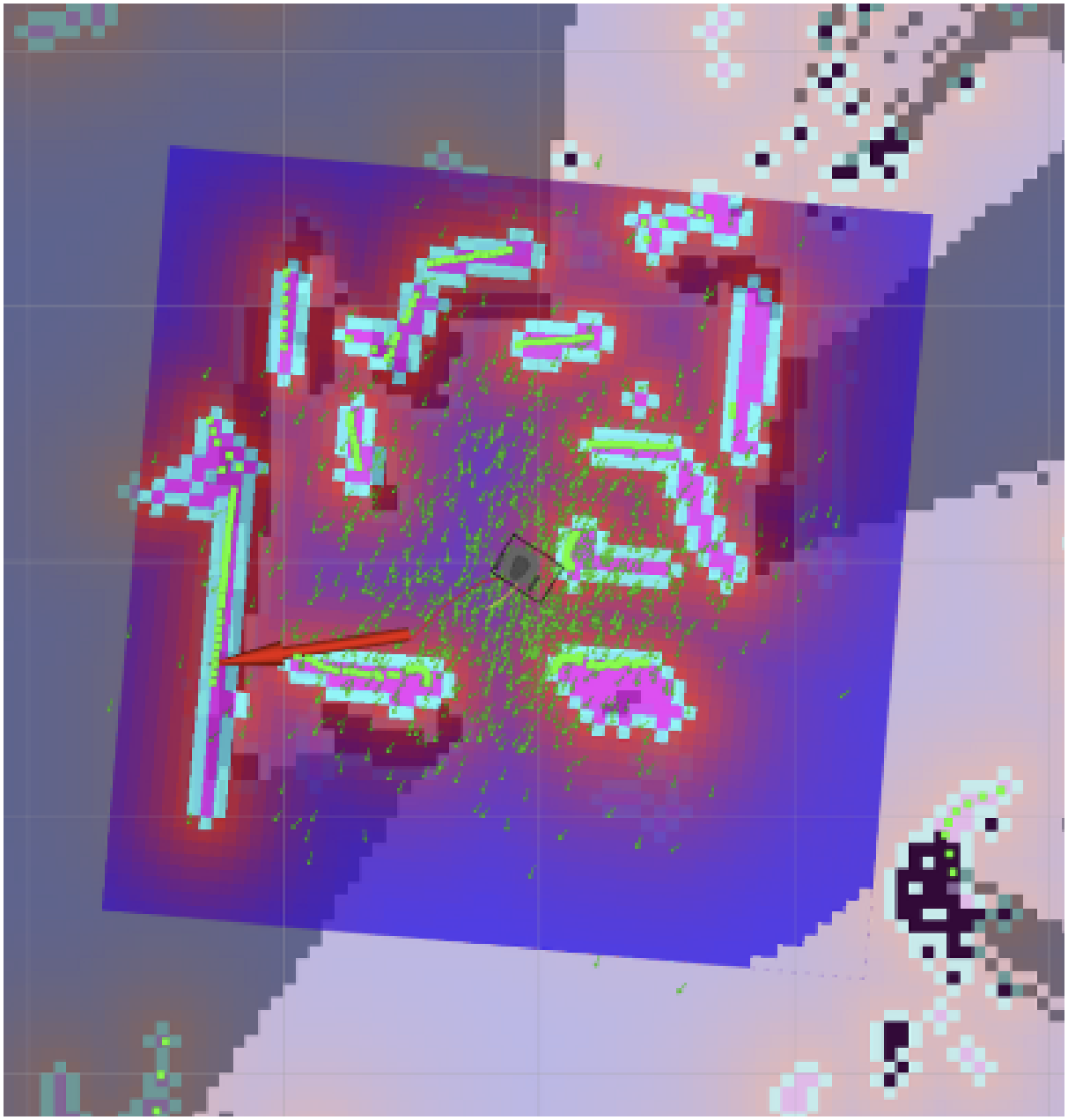

Figure 4: Rviz Navigation screen

On Figure 4, the arrow correspond to the orientation of the robot at the finishing point. The red line is the optimal path and the yellow one is the chosen one.
If an obstacle is detected, a new path will be calculated and the yellow path will change.

5 Trajectory tracking

5.1 Kalman filter

Part 1 : Theory

In the case of our TurtleBot3, the state vector $\mathbf{X}$ is defined as:

$$\mathbf{X} = \begin{bmatrix} x & y & \dot{x} & \dot{y} & \ddot{x} & \ddot{y} & \psi \end{bmatrix}^\top,$$

where x, y represent the robot's position, $\dot{x}, \dot{y}$ the velocities, $\ddot{x}, \ddot{y}$ the accelerations, and ψ the yaw (orientation).

The Kalman Filter relies on four key matrices:

- The measurement matrix $\mathbf{H}$, which maps the state vector to the measurable outputs, such as x, y, ψ:
$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$
- The process noise covariance matrix $\mathbf{Q}$, which accounts for uncertainties in the robot's dynamics:
$$\mathbf{Q} = \begin{bmatrix} 0.001 & 0 & \cdots & 0 \\ 0 & 0.001 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 10{,}000 \end{bmatrix}.$$
- The measurement noise covariance matrix $\mathbf{R}$, which models sensor noise:
$$\mathbf{R} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.0025 \end{bmatrix}.$$
- The state covariance matrix $\mathbf{P}$, initialized with large values to reflect high initial uncertainty:
$$\mathbf{P} = \begin{bmatrix} 10^6 & 0 & \cdots & 0 \\ 0 & 10^6 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 10^6 \end{bmatrix}.$$

Prediction Step

The prediction step updates the state estimate $\dot{\mathbf{X}}$ based on the system's motion model:

$$\dot{\mathbf{X}} = \mathbf{F}\mathbf{X} + \mathbf{G}\mathbf{u},$$

where $\mathbf{F}$ is the state transition matrix and $\mathbf{G}$ represents control inputs (e.g., wheel velocities V_{rg} and V_{rd}):

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\tau} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{\tau} & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The yaw rate ψ is updated based on the differential wheel velocities:

$$\psi = \frac{V_{rd} - V_{rg}}{L},$$

where L is the wheelbase.

Correction Step

During the correction step, the Kalman Gain $\mathbf{K}$ is calculated to update the state estimate:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top \left(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\top + \mathbf{R}_k\right)^{-1}.$$

The updated state $\mathbf{X}_k$ and covariance $\mathbf{P}_k$ are given by:

$$\mathbf{X}_k = \mathbf{X}_{k|k-1} + \mathbf{K}_k\left(\mathbf{Z}_k - \mathbf{H}_k\mathbf{X}_{k|k-1}\right),$$

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_{k|k-1}.$$

Our Kalman Filter provides a robust way to estimate the TurtleBot3's position, velocity, and orientation, even in the presence of noise. By iteratively applying the prediction and correction steps, it ensures accurate localization and motion tracking.

Part 2 : Simulation

We started by focusing on implementing a Kalman filter to track the robot's position, velocity and acceleration.
To begin with, we started using the Turtlebot3 simulation on Gazebo.
We created a ROS2 environment on our virtual machines to simulate the Turtlebot3 and be able to test it [Figure 5].

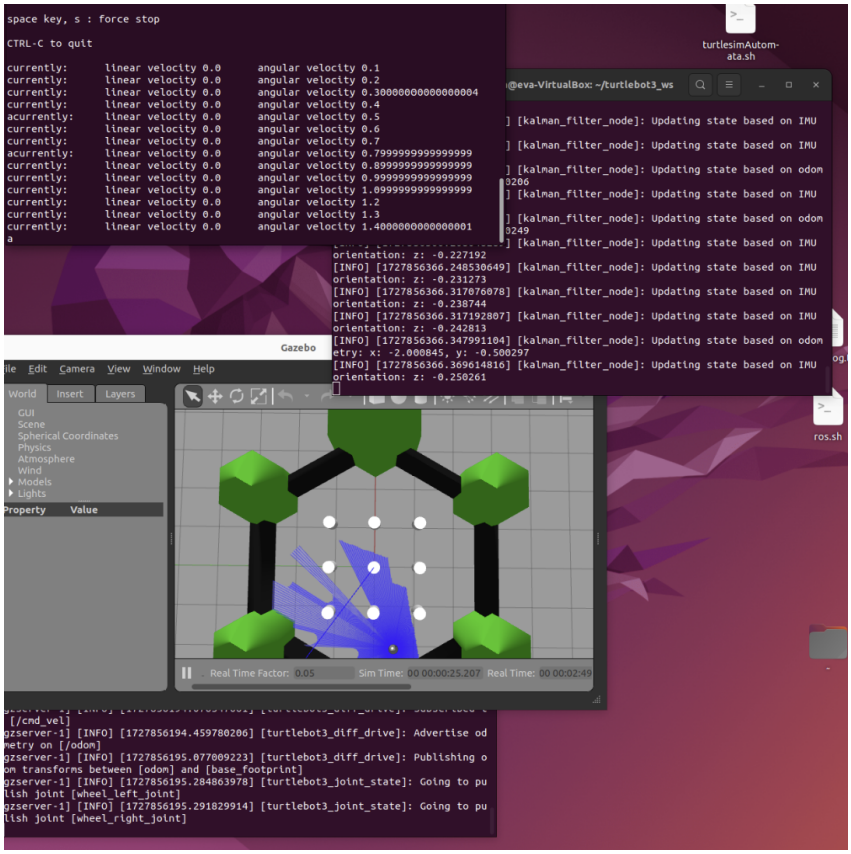

Figure 5: Robot Gazebo model for the simulation

This was not ideal since we had issues with Gazebo simulation that was not behaving correctly and had a lot of crasher. To avoid this and allow the testing, we changed the simulation method and went back on Matlab.
To be able to make a Matlab simulation, we used a pre-made model on simulink [Figure 6] to simulate the behaviour of the robot and get data.

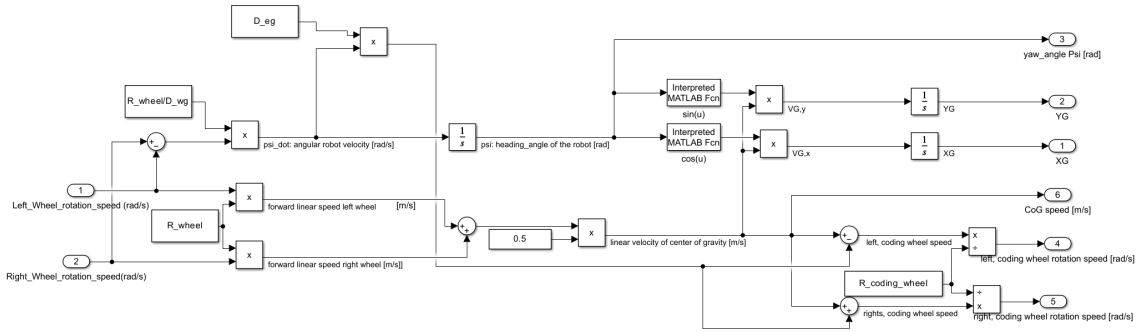

Figure 6: Simulation model for the robot

Then we applied our Kalman filter theory to the data gathered thanks to the model, which allowed us to see that the Kalman tracking was working [Figure 7].

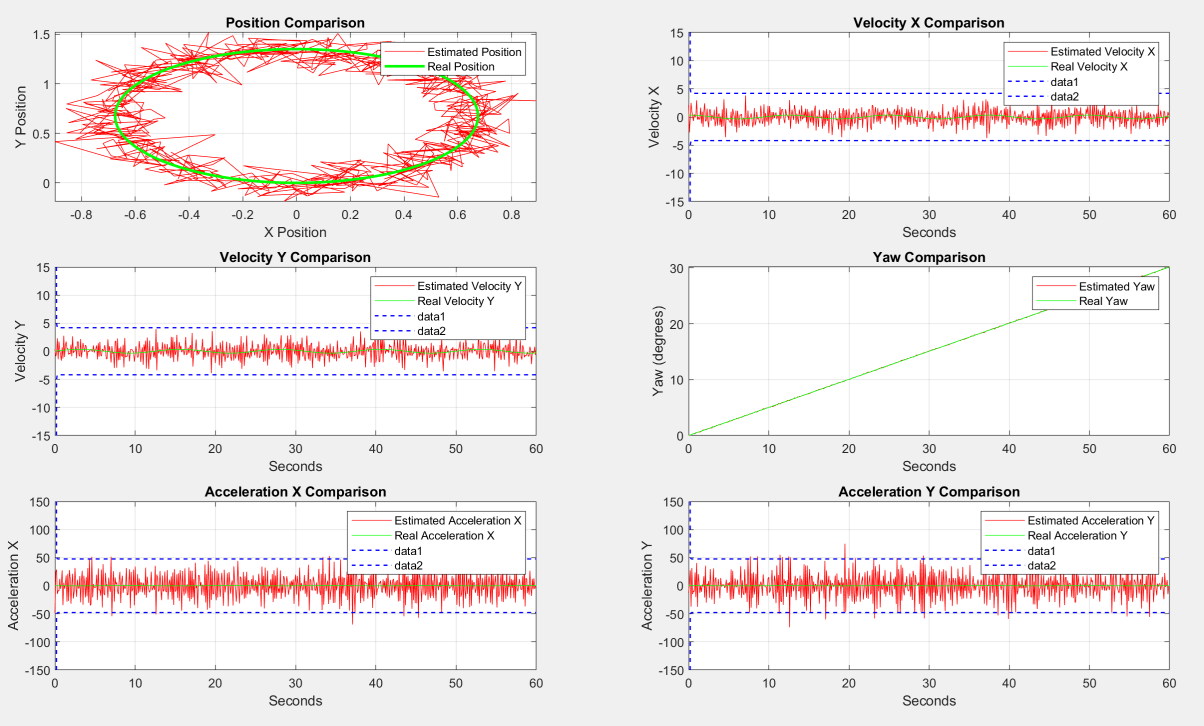

Figure 7: Test of the Kalman filter on simulation

The next step is connecting the robot to Matlab to test the filter on the real robot.

Part 3 : Integrating on the Turtlebot3

To be able to integrate the Extended Kalman Filter on the real Turtlebot, we need to respect some prequisites.

- The Turtlebot needs to be connected on the same wifi source then the computer hosting Matlab.
- Matlab needs to have ** packages installed
- The connection between Matlab and the ros master needs to function
- We need to make sure we are subscribed to the sensors we need on the robot.

Once we are sure this is active, we are able to execute the Kalman code. It was adapted to the real robot by adding a section with a "get data" function whose role is to get data from the robot's sensor during a fixed amount of seconds.

This data will then be used in the Kalman filter and the results would be written in a file to exploit them right after. At first we intended to make the plot of all the data live, but the computer showed difficulties in making that work, so we decided to simulate the trajectory, acceleration and yaw in a second time.

After exploiting the data of our Extended Kalman Filtering we have our final results. These figures [Figure 8] are appearing little by little live in real life, and here we display their final form.

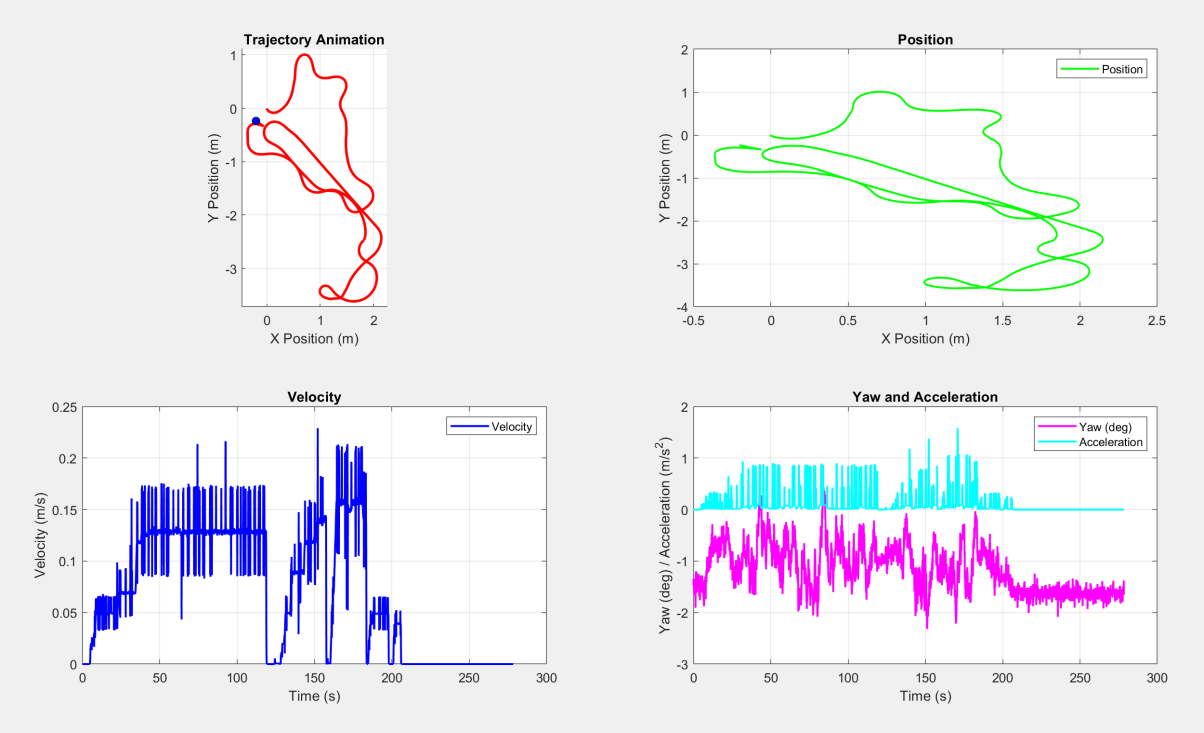

Figure 8: Test of the Kalman filter on simulation

We couldn't plot them live because the computer didn't handle the graphics as well as handling the robot's moves, so we wrote the data in a file to be able to simulate it again afterward, and that's what we got in the figure.

6 Conclusion

In this project, we have laid the foundation for an autonomous robotic system capable of efficient navigation and exploration. Through the implementation of key features such as obstacle avoidance, mapping, and trajectory tracking, the system demonstrates significant potential for real-world applications in diverse environments.

Looking ahead, there is room for improving the system's capabilities. Possible additions include a "return to base" feature, which would optimize exploration time by ensuring efficient resource management, and a user-friendly interface that allows users to select specific locations for the robot to navigate autonomously.

With these refinements, the system could achieve even greater levels of autonomy, efficiency, and usability, solidifying its position as a robust solution for dynamic and complex navigation tasks.

7 Sources

- Principal source of information for the installations: TurtleBot3 Documentation

- MATLAB guide for building a map using LiDAR and SLAM with ROS: Build a Map Using LiDAR and SLAM in MATLAB
- Comprehensive ROS tutorial videos: ROS Tutorial Playlist on YouTube
- Guide for tuning navigation parameters in ROS: ROS Navigation Tuning Guide
- Online course for mastering TurtleBot3 with ROS: Mastering TurtleBot3 with ROS - The Construct