



UNIVERSIDADE D  
COIMBRA

Diogo Gomes Rodrigues

**TRACKED AUTONOMOUS VEHICLES MOVING  
ON ROUGH TERRAIN  
DYNAMIC DRIVE ADJUSTMENT**

Dissertação no âmbito do Mestrado Integrado em Engenharia Mecânica na área de Produção e Projeto, orientada pelo Professor Doutor Carlos Xavier Pais Viegas, coorientada pelo Professor Doutor Pedro Mariano Simões Neto e apresentada ao Departamento de Engenharia Mecânica da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

September 2022



1 2



9 0

FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE  
COIMBRA

# **Tracked Autonomous Vehicles Moving on Rough Terrain - Dynamic Drive Adjustment**

Submitted in Partial Fulfilment of the Requirements for the Degree of Master  
in Mechanical Engineering in the speciality of Production and Project

## **Veículos Autónomos com Lagartas a Operar em Terrenos Acidentados - Ajuste Dinâmico do Perfil de Tração e Locomoção**

Author

**Diogo Gomes Rodrigues**

Advisor[s]

**Carlos Xavier Pais Viegas**

**Pedro Mariano Simões Neto**

Jury

President	<b>Professor Doctor Miguel Rosa Oliveira Panão</b> <b>Professor Auxiliar da Universidade de Coimbra</b> <b>Professor Doctor Nuno Miguel Fonseca Ferreira</b> <b>Professor Coordenador com Agregação do Politécnico de Coimbra</b>
Vowels	<b>Professor Doctor Ricardo Nuno Madeira Soares Branco</b> <b>Professor Auxiliar da Universidade de Coimbra</b>
Advisor	<b>Professor Doctor Carlos Xavier Pais Viegas</b> <b>Professor Auxiliar Convidado da Universidade de Coimbra</b>

**Coimbra, September 2022**



To my parents, Manuel and Margarida.

To my brother, Fábio.

To my family.

To my friends.



## **ACKNOWLEDGEMENTS**

The completion of this dissertation would not have been possible without the support of all those who have followed me over the years. To all of them, the greatest gratitude.

I would like to thank my advisors, Professor Doctor Carlos Viegas and Professor Doctor Pedro Neto, for the support given, the availability to help and their teachings.

I would also like to thank Tiago Gameiro and Tiago Pereira who contributed to the increase of my knowledge and the realization of the dissertation and also for the availability they showed from the beginning of the project to the end in helping. Thanks also to Diogo Valério who showed willingness to help whenever necessary.

I would like to thank my parents and my brother for all the support given to me over these five years and for always believing in me and encouraging me to move forward, thus achieving my goals, for all the advice they gave me and for always believing in me.

To conclude, I would also like to thank all my friends who have always supported me along the journey, with a special thanks to my friend Rita Pinto who helped me a lot to overcome less good moments and has always supported me throughout the years.





## Abstract

Forest cleaning is of great importance in fighting forest fires, because in addition to removing fuel for fires, it makes access to land easier for fighting possible fires. Being uncontrolled fires of great danger to humans, it is necessary to find ways to prevent them from happening. For this there are fuel management strips along roads, high voltage lines, around housing clusters and other infrastructure at the urban forestry interface. A major problem in carrying out forest cleanings is the lack of manpower to allow the cleaning of all land and places defined as places of high danger for society in the event of fire before the start of the fire season. With technological developments, the problem of lack of manpower can be compensated, by using autonomous vehicles, capable of carrying out forest cleanings and adapting to the terrain.

This work consists of the development of a control algorithm to implement in an autonomous robot moving in rough terrains. Its objective is the development of an algorithm capable of detecting deviations in the robot's trajectory and correcting them, and an algorithm that allows real-time knowledge of the robot's yaw value.

This work was developed in virtual environments, using Robot Operating Systems (ROS) and Python programming language.

Initially, the work consisted of determining the robot's yaw value and reading this same value to know the robot's orientation at each instant. The next step consisted of the development of an algorithm that, by reading the robot's orientation data, would correct the robot's trajectory so that its movement would be rectilinear, suffering only small deviations, immediately compensated. Finally, tests were carried out in virtual environments to validate the work developed. The tests allowed to conclude the best angular velocity of the robot whenever it was necessary to correct its trajectory, and to obtain the best approximation of the robot's orientation data due to the noise caused by the sensors. It also allowed to verify the trajectory performed by the robot in different virtual environments, comparing the behavior of the robot with the use of the developed control and the non-use, verifying its operation.

**Keywords:** Forest Fires, Tracked Autonomous Vehicles, Robot Operating Systems (ROS), Traction Control, Rough Terrains.

## Resumo

A limpeza florestal é muito importante no combate a incêndios florestais, além de remover combustível para os incêndios, torna o acesso aos terrenos mais fácil para o combate de eventuais incêndios. Sendo os incêndios não controlados de grande perigo para o ser humano, é necessário arranjar formas de prevenir que estes aconteçam. Para isso existem faixas de gestão de combustível ao longo das estradas, linhas de alta tensão, em torno de aglomerados habitacionais e outras infraestruturas na interface urbano florestal. Um grande problema na realização das limpezas florestais é a falta de mão de obra, não permitindo a limpeza de todos os terrenos e locais definidos como locais de alto perigo para a sociedade em caso de incêndio, antes do início da época de incêndios. Com a evolução tecnológica, pode compensar-se o problema da falta de mão de obra, recorrendo a veículos autónomos capazes de realizar limpezas florestais e de se adaptarem ao terreno.

Este trabalho consiste no desenvolvimento de um algoritmo de controlo para implementar num robô autónomo a mover-se em terrenos acidentados. Tem como objetivo o desenvolvimento de um algoritmo capaz de detetar desvios na trajetória do robô e fazer a sua correção e de um algoritmo que permite o saber em tempo real o valor do yaw do robô.

Este trabalho foi desenvolvido em ambiente virtuais, utilizando Robot Operating System (ROS), a linguagem de programação Python.

Inicialmente, o trabalho consistiu na determinação do valor de yaw do robô e leitura desse mesmo valor de modo a conhecer a orientação do robô em cada instante. O passo seguinte consistiu no desenvolvimento de um algoritmo que através da leitura dos dados da orientação do robô, corrigiria a trajetória do robô de modo que o movimento deste fosse retilíneo, sofrendo apenas pequenos desvios, imediatamente compensados. Por fim foram realizados testes em ambientes virtuais de modo a validar o trabalho desenvolvido. Os testes permitiram concluir a melhor velocidade angular do robô sempre que era necessário corrigir a sua trajetória, obter a melhor aproximação dos dados da orientação do robô devido ao ruído dos sensores. Permitiram também verificar a trajetória realizada pelo robô em diferentes ambientes virtuais, comparando o comportamento do robô com a utilização do controlo e a não utilização do controlo desenvolvido, verificando o seu funcionamento.

**Palavras-chave:** Incêndios Florestais, Veículos Autônomos de Lagartas, Robot Operating Systems (ROS), Controle de Tração, Terrenos Acidentados.

---

## Contents

LIST OF FIGURES .....	ix
LIST OF TABLES .....	xi
LIST OF SYMBOLS AND ACRONYMS/ ABBREVIATIONS .....	xiii
List of Symbols.....	xiii
Acronyms/Abbreviations.....	xiii
1. INTRODUCTION .....	1
1.1. Motivation.....	1
1.2. Objectives .....	2
1.3. Outline .....	2
2. STATE OF THE ART .....	5
2.1. Tracked vehicles .....	5
2.1.1. Howe & Howe .....	5
2.1.2. Vallfirest .....	5
2.1.3. McConnel .....	6
2.1.4. Bermarthor.....	7
2.1.5. Green Climber .....	8
2.1.6. Comparison between robots .....	9
2.2. Sensors .....	9
2.2.1. RTK GPS.....	9
2.2.2. LiDAR .....	10
2.2.3. Traversability maps .....	11
2.3. Autonomous dynamic drive in rough terrain .....	12
3. METHODOLOGY .....	19
3.1. Ubuntu .....	19
3.1.1. Robot Operating System (ROS) .....	19
3.2. Robot Orientation .....	23
3.2.1. Quaternions.....	23
3.2.2. Euler angles .....	24
3.3. Swift Navigation .....	25
3.3.1. Skylark™.....	25
3.3.2. Starling® .....	26
3.3.3. Duro Inertial .....	27
4. WORK DEVELOPMENT .....	31
4.1. ROS Topics.....	31
4.2. Operating Mode of the Control Code .....	34
4.3. Calculation of expected final position .....	37
4.3.1. Line equation .....	37
4.3.2. Calculate the final coordinates .....	38
5. VIRTUAL SIMULATIONS AND RESULTS .....	41
5.1. Realized tests .....	41
5.1.1. Angular velocity .....	41
5.1.2. Yaw approximation .....	42
5.1.3. Empty world .....	42

5.1.4. Agriculture world .....	42
5.1.5. Inspection world .....	43
5.2. Results .....	45
5.2.1. Angular velocity .....	45
5.2.2. Yaw approximation .....	46
5.2.3. Empty world .....	48
5.2.4. Agriculture world .....	49
5.2.5. Inspection world .....	50
6. CONCLUSIONS .....	53
6.1. Future Work.....	54
7. BIBLIOGRAPHY .....	55
ANNEX A .....	57
ANNEX B .....	59
ANNEX C .....	63
ANNEX D .....	65

---

## LIST OF FIGURES

Figure 2.1 - Thermite RS1. Adapted from [2].....	5
Figure 2.2 - VF Dronster work gradient. Adapted from [3].....	6
Figure 2.3 - VF Dronster. Adapted from [3].....	6
Figure 2.4 - ROBOCUT2 RC40. Adapted from [4].....	7
Figure 2.5 - E-TRAIL. Adapted from [5].....	7
Figure 2.6 - Green Climber LV400. Adapted from [6].....	8
Figure 2.7 - RTK GPS. Adapted from [8].....	10
Figure 2.8 – LiDAR. Adapted from [10].....	11
Figure 2.9 - Bayesian fusion. Adapted from [11].....	12
Figure 2.10 - Path tracking results on grass and gravel. Adapted from [12].....	13
Figure 2.11 - Vehicle following a desired circular path. Adapted from [13].....	14
Figure 2.12 - Scheme of the robotic tracked vehicle uncontrolled movement prevention system. Adapted from [14].....	15
Figure 2.13 - Simulation results. Adapted from [14].....	15
Figure 2.14 - Block diagram representation of the proposed planner. Adapted from [15].	16
Figure 2.15 - Flow chart showing the working of the proposed planning algorithm. Adapted from [15].....	17
Figure 2.16 - Work of the system. Adapted from [15].....	18
Figure 2.17 – Terrain topography map showing the 3D path followed by the robot under both the planners over the experimental setup. Adapted from [15].....	18
Figure 3.1 - a) Inspection world; b) Agriculture world; c) Empty world.....	21
Figure 3.2 - a) Turtlebot3, model “waffle_pi”; b) Husky robot.....	21
Figure 3.3 - ROS topics.....	22
Figure 3.4 – Some of Rviz’s available sensors.....	22
Figure 3.5 - Data demonstration on map.....	23
Figure 3.6 - Quaternions representation. Adapted from [28].	24
Figure 3.7 - Roll, pitch, and yaw axes. Adapted from [29].	25
Figure 3.8 - Swift Navigation ecosystem. Adapted from [19].....	25
Figure 3.9 - Skylark™ network. Adapted from [20].....	26
Figure 3.10 - Flow chart of Starling®. Adapted from [22].....	27
Figure 3.11 - Duro Inertial. Adapted from [23].....	27

Figure 3.12 - Duro Inertial system architecture. Adapted from[24] .....	28
Figure 3.13 - Swift Console Magnetometer window. Adapted from[25] .....	29
Figure 3.14 - Swift Console Velocity window. Adapted from[25].....	29
Figure 4.1 - ROS topic " <i>imu/data</i> " data. The presented values serve as a mere example. .	32
Figure 4.2 – ROS topic " <i>cmd_vel</i> " data. The presented values serve as mere examples....	32
Figure 4.3 - Importing library, data, and functions to code. ....	33
Figure 4.4 – Approximate small variations to the initial yaw.....	33
Figure 4.5 - Deviation (x) calculation .....	35
Figure 4.6 – Expected robot movement while the code is running, and a deviation is detected. ....	36
Figure 4.7 - Robot frame.....	37
Figure 5.1 - Approximately initial position of the robot in each test of location 2.....	43
Figure 5.2 - Approximately initial position of the robot in each test of location 3.....	44
Figure 5.3 - Approximately initial position of the robot in each test of location 4.....	44
Figure 5.4 - Robot's trajectory considering the variation of the angular velocity.....	46
Figure 5.5 - Robot's trajectory considering the approximation of the yaw values.....	47
Figure 5.6 - Deviation of the robot after approximately 50 meters.....	49
Figure 5.7 - Deviation of the robot after the movement, at location 2.....	50
Figure 5.8 - Zoom of the final position of the robot, at location 2.....	50
Figure 5.9 - Deviation of the robot after the movement, at location 3.....	52
Figure 5.10 - Deviation of the robot after the movement, at location 4.....	52
Figure 5.11 - Zoom of the final position of the robot, at location 3.....	52
Figure 5.12 - Zoom of the final position of the robot, at location 4.....	52



## LIST OF TABLES

Table 1 - Comparison between robots .....	9
Table 2 - Average initial robot position for each realized test, at location 1.....	42
Table 3 - Medium initial position of the robot for each realized test in agriculture world, at location 2. ....	43
Table 4 - Medium initial position of the robot for each realized test, in the inspection world, at location 3. ....	45
Table 5 - Medium initial position of the robot for each realized test in the inspection world, at location 4. ....	45
Table 6 – Average final robot position for each realized test, at location 1.....	48
Table 7 - Average final robot position for each realized test, at location 2.....	50
Table 8 - Average final robot position for each realized test, at location 3.....	51
Table 9 - Average final robot position for each realized test, at location 4.....	51



## LIST OF SYMBOLS AND ACRONYMS/ ABBREVIATIONS

### List of Symbols

$\phi$  – Roll

$\Theta$  – Pitch

$\Psi$  – Yaw

### Acronyms/Abbreviations

ADRC – Active Disturbance Rejection Controller

ANP – Associação Natureza Portugal

CAN – Controller Area Network

FoV – Field of View

GNSS – Global Navigation Satellite System

GPS – Global Positioning System

IMU – Inertial Measurement Unit

INS – Inertial Navigation Solution

LiDAR – Light Detection and Ranging

PID – Proportional Integral Derivative

RTK – Real-Time Kinematic

ROS – Robot Operating System

WWF – World Wide Fund for Nature



# 1. INTRODUCTION

The use and development of tools are part of the evolution of the human being, that always worked to use more and better tools for every type of work, making it easier to do. Fire always threatened human lives, and because of that it was necessary to have some control over it, but wildfires still represent a huge threat to humans, animals, and every kind of life because it is impossible to have full control over it, so it is important to have something that helps avoid wildfires or, in case that happens, that allows better control over those. One way to have some type of control over it starts with forest cleaning. In the beginning human beings created tools to facilitate that job that had evolved to be easier to use and more effective. The use of vehicles such as trucks, airplanes and helicopters increased the effectiveness of firefighting, but the loss of lives is inevitable and continues to happen, to combat this problem it is necessary to increase the use of unmanned vehicles.

Unmanned vehicles allow more effective cleanings of forests and firefighting, safer and without danger to human lives. These vehicles can move autonomously and adapt to the type of terrain, allowing a more effective forest cleaning that reduce which risk of forest fires.

## 1.1. Motivation

The evolution of technology makes it possible to build or improve tools to make tasks easier and more effective. The use of autonomous vehicles is growing and brings benefits to society.

Forest cleaning is very important not only in extinguishing, but also in controlling forest fires because in addition to removing fire fuel, facilitates firefighting by improving the access to the forest zones, but the lack of manpower makes it impossible to realize the cleaning in useful time before the beginning of fire season. There are several cleaning lanes along roads, high voltage lines, around housing clusters, and other infrastructures at the forest urban interface but, once again, it is not possible to accomplish this goal because of the lack of manpower.

According to a report of 2020 from ANP (Associação Natureza Portugal) in association with WWF (World Wide Fund for Nature) [1] in the last 30 years, Portugal is the country with more hectares burned and is the first country in Europe and the fourth in the world to have lost the largest forest mass since the beginning of XXI century, largely due to the wildfires that ravage the country every summer. “If we continue with this dynamic, fires, especially those that affect the rural-urban interface area, will increasingly put people’s lives at serious risk”. In Europe, Portugal and Spain could become vulnerable to super fires.

That being said, it is very important and beneficial to develop and use autonomous vehicles, without the need for human monitoring, thus compensating for the lack of manpower, but to make this possible, autonomous vehicles must be able to automatically adapt to the terrain they travel on. It is necessary to develop a system of automatic dynamic drive adjustment to allow the vehicle to overcome the difficulties imposed by rough terrains and move autonomously with the capacity to maintain the course, regardless of the slope and type of terrain.

## **1.2. Objectives**

The goal of this thesis work is to develop a generic traction control algorithm for tracked autonomous vehicles moving on rough terrain. For this, it will be used a tracked vehicle equipped with several sensors and controllers. The algorithm should be able to:

- Be implemented in any type of tracked or differential drive vehicle;
- Adjust in real time the traction control of the vehicle to improve its movement as a function of the type and configuration of the terrain;
- Keep a straight line movement, even in rough terrain conditions;

## **1.3. Outline**

This dissertation is divided into seven chapters.

The first chapter is the introduction to the theme of this dissertation and the exposition of the existing problem that inspired this dissertation.

The second chapter is the state of the art that consists of the research about the existing robots and articles information about research already done by other authors. This chapter is

divided into three sub-chapters. The first one is a comparison of the tracked vehicles existing on the market, the second one is about the sensors used and the third one is about the research on the autonomous dynamic drive in rough terrain.

In the third chapter the methodology is presented. This chapter is divided in three sub-chapters, the first one about Ubuntu and its functionalities, the second one about Swift Navigation and its services and the third one is about the robot's orientation.

The fourth chapter is the work development and is divided into three sub-chapters, the first one is about the ROS topics and the data provided by them. The second one is about the operating mode of the control and explains how the control works and the last one is about the location of the robot, and how it can be determined.

The fifth chapter is divided into two sub-chapters, the first one is about the realized tests and the environments where the tests were realized and the conditions of each test, and the second sub-chapter is about the results obtained in each test.

The sixth chapter is the conclusion and relates if the main objectives were accomplished or not and gives a summary of the approach used during all the work. The sub-chapter is about the future work that can be made.





## 2. STATE OF THE ART

Tracked vehicles are special vehicles designed to move on rough terrain. The use of tracks augments the contact area with the ground, thus providing an increased, amount of traction, while reducing the pressure on the ground surface, making it easier for heavy vehicles to move on soft grounds.

Since the appearance of the first tracked vehicle for agricultural purposes, these have constantly evolved and today there are tracked vehicles for civilian use or even tracked autonomous vehicles for forest cleaning or cargo transportation.

This chapter will make an overview of existing unmanned tracked vehicles for forestry applications. It will then dwell on the sensors used in autonomous vehicles and the mathematic models used for vehicle control.

### 2.1. Tracked vehicles

#### 2.1.1. Howe & Howe

Howe & Howe it's the company that builds tracked vehicles for defense, that requires a pilot and for civilian use that can be driven by people, as in the case of the Ripsaw F4, or remote controlled, like the Thermit series of the company.

The Thermite series robots are firefighters' robots operated by remote belly-pack controllers and are one of the most capable, durable firefighting robots on the market. The pilot has access to a real-time video feed, so he can control the robot at a large distance, enabling its function in extreme conditions without endangering human lives.



Figure 2.1 - Thermite RS1. Adapted from [2]

#### 2.1.2. Vallfirest

Vallfirest is a reference company in the manufacturing of equipment, tools, and solutions to fight and prevent forest fires. One of the pieces of equipment manufactured by

them is the VF Dronster, a multipurpose remote-controlled robot, capable of being controlled up to a distance of 25 meters.

The VF Dronster operates with a 3-cylinder motor with 24,5kW of power and can work on rough terrains with slopes up until 40 degrees, going up or down, and 30 degrees going sideways, as shown in figure 2.4.



**Figure 2.2** - VF Dronster work gradient. Adapted from [3]

Due to its narrow structure, the VF Dronster can move easily on the terrain, managing to pass through spaces of 90cm between trees. It can also be equipped with several attachments that can be used by the operated in case of need.



**Figure 2.3** - VF Dronster. Adapted from [3]

### 2.1.3. McConnel

This United Kingdom company builds a huge amount of types of equipment, including power arms, remote control technology, rotary and flail mowers, arable machinery, sprayers and spreaders, pasture and livestock. The remote control technology encompasses several

cut robots, with different dimensions, capacities and sizes, and all can be operated remotely from a distance up to 150 meters.

One of their robots is the ROBOCUT2 RC40 which operates with a 3 cylinder motor and 27,5kW of power, and can work on rough terrains with slopes up until 40 degrees, with the standard tracks, but can be added special tracks with spikes that allow the robot to work on slopes up until 55 degrees. The ROBOCUT2 RC40 can be seen in figure 2.6.



Figure 2.4 - ROBOCUT2 RC40. Adapted from [4]

#### 2.1.4. Bermarthor

Bermarthor is a company that manufactures, commercializes and repairs all kind of agricultural and forestry cleaning machines.

This company manufactures the E-TRAIL, which is a 2-speed diesel engine with approximately 29,8kW that can be operated by remote control up to 150m and can work on slopes with 55 degrees in every direction. The E-TRAIL by Bermarthor can be seen in figure 2.7.

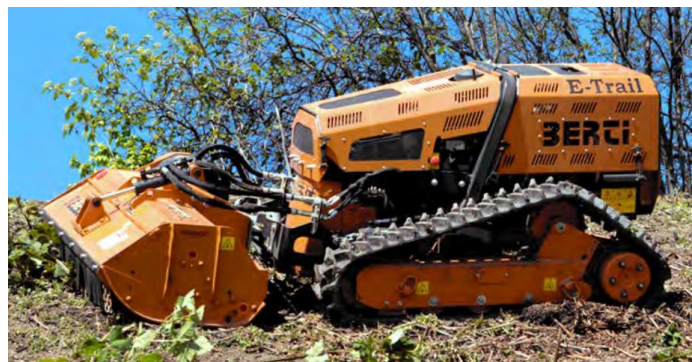


Figure 2.5 - E-TRAIL. Adapted from[5]

### 2.1.5. Green Climber

The company Green Climber builds remoted control slope mowers that can handle the most difficult tasks. The slope mowers can work on rough terrains with slopes up to 56 degrees and can be operated by remote control from 150m. In this thesis, will be used the robot LV400 Pro from Green Climber.

The LV400 Pro operates with a 3 cylinders motor with 26,9kW of power and has a work gradient of 56 degrees at the maximum width and has the dimensions shown in figure 2.6.

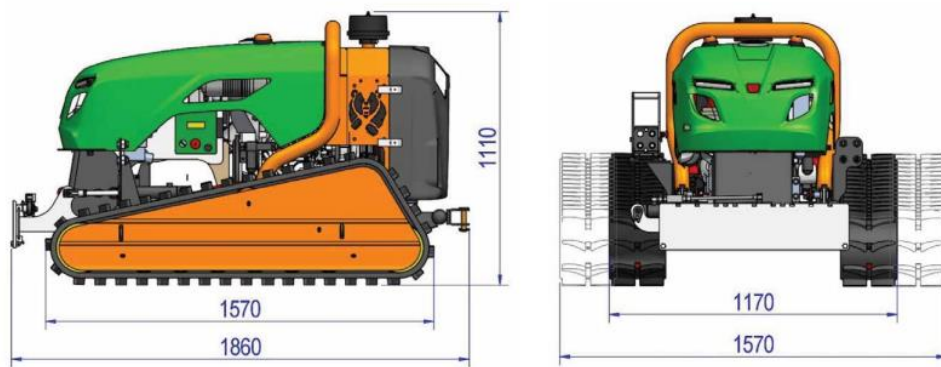


Figure 2.6 - Green Climber LV400. Adapted from [6]

### 2.1.6. Comparison between robots

**Table 1** - Comparison between robots

Company	Model	Length [mm]	Width [mm]	Height [mm]	Motor	Power [kW]	Remote control Range [m]	Weight [kg]	Work Gradient [°]
Howe&Howe	Thermite RS1	1962	1118	1625	3 cylinder Diesel	18	300-500	725	-
Vallfirest	VF Dronster	1710	887	1023	3 cylinder Diesel	24,5	25	850	Up to 30/40*
McConnel	ROBOCUT2 RC40	1960	1310	1120	3 cylinder Diesel	27,5	150	1150	Up to 40/55**
Bermarthor	E-TRAIL	1580	1430	1060	4 cylinder Diesel	29	150	1100	Up to 55
Green Climber	LV400 Pro	1860	1570	1110	3 cylinder Diesel	26,9	150	875	Up to 56

\*Front/lateral gradient

\*\*Depending on tracks

## 2.2. Sensors

There are several types of sensors and cameras on the market, that can be installed on robots to allow their localization and recognition of the terrain, being able to detect the type of terrain and its characteristics in the way to adapt is locomotion.

### 2.2.1. RTK GPS

The RTK (Real Time Kinematic) allows error reduction to just a few centimeters, reducing the GPS (Global Positioning System) error.

As mentioned by Feng & Wang (2008) at [7], “an RTK system consists of a continuous operating reference station network and data links between a network server and reference stations and between the server and user-terminal”.

The RTK accuracy is defined as the degree of conformance of an estimated RTK position at a given time to a defined reference coordinate value, which is obtained from an independent approach. The availability, in terms of accuracy, is the percentage of the time

during which the RTK solutions are available at a certain accuracy using the ambiguity-fixed and ambiguity-float phase measurements. The availability, in terms of ambiguity resolution, is the percentage of the time, in which position estimation is based on all the phase measurements whose integers have been correctly fixed at each epoch, assuming all the ambiguity-fixed solutions will give required accuracy. The RTK integrity relates to the confidence level that can be placed in the information provided by the RTK system. The continuity of the RTK is the availability over a certain operational period and conditions. [7]

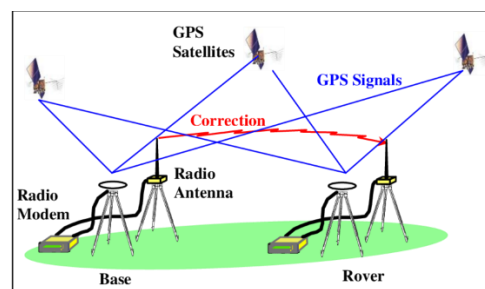


Figure 2.7 - RTK GPS. Adapted from [8]

### 2.2.2. LiDAR

LiDAR (Light Detection and Ranging) sensors are very important and are on the rise due to the need to make vehicles fully autonomous. As described by Roriz et al. (2021) at [9] “it can measure distances by simply calculating the round-trip time of a laser pulse travelled to the target and back”.

The detection range values may change due to sunlight interferences and the target’s surface reflectivity. The transmitted power of the laser is limited by eye safety regulations and various factors change the calculated value as the laser’s wavelength, beam diameter, motion, pulse width and repetition rate for pulsed operations. Currently, LiDAR uses two wavelengths, 905 nm, and 1550 nm, making possible lights spatial resolution on the order of 0.1 degrees, which allows for extremely high-resolution 3D representation of objects around the vehicle. The Field of View (FoV) is the angle at which LiDAR signals are emitted and it must provide both horizontal and vertical FoV to allow a 3D representation of the vehicle’s surroundings.[9]



Figure 2.8 – LiDAR. Adapted from [10]

### 2.2.3. Traversability maps

Traversability maps are very useful when autonomous vehicles are moving on rough terrains. These maps make it possible to distinguish traversable paths from non-traversable paths. To create a traversability map can be used several sensors and algorithms.

Sock et al. (2016) [11] used 3D-LiDAR and a camera to generate probabilistic traversability maps. Their approach estimates traversability of the terrain and build a 2D probabilistic grid map online using 3D-LiDAR and camera. The detection results by LiDAR and camera needed to be represented in a form which was both simple and intuitive. It is used the Bayesian fusion to create the maps produced by LiDAR and camera. The traversability map with camera follows some steps to map probabilistic pixel value to a dense grid map using spatially sparse LiDAR. Initially it extracts LiDAR points projected to an image and transform their coordinates to GPS coordinate. The next step is to triangulate the points with Delaunay triangulation. After that, it assumes each triangle has a single probability value and then project the triangle's centroid to the image and map the corresponding pixel value to a grid map. The traversability map with LiDAR needs to convert range data to  $n \times n$  elevation map, interpolate the elevation map to fill the missing cells and convert the slope feature probabilistic value. Then it is used Bayesian fusion to produce the complete traversability map, as shown in figure 2.9. [11]

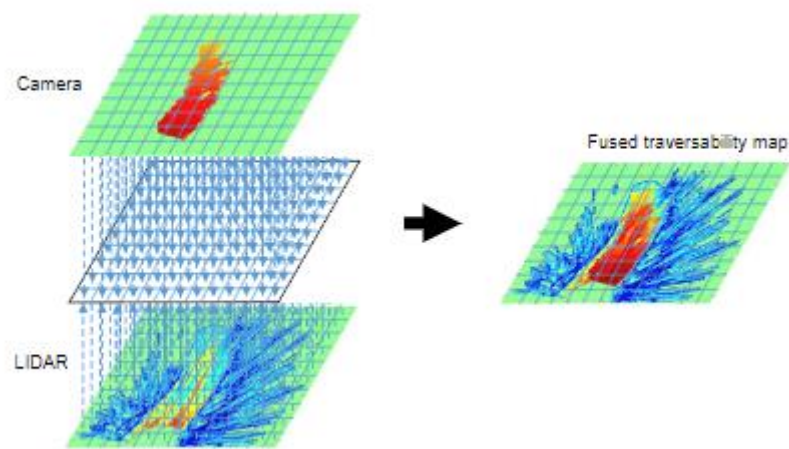


Figure 2.9 - Bayesian fusion. Adapted from [11]

### 2.3. Autonomous dynamic drive in rough terrain

There are several autonomous dynamic drive models with different goals that can be used on autonomous tracked vehicles.

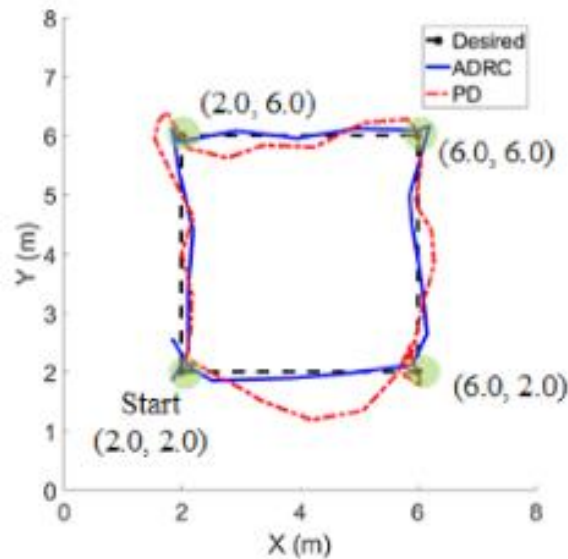
Those models allow autonomous navigation, enabling problem-solving such as avoiding obstacles and correcting the trajectory so that the robot can reach its goal, without requiring human intervention.

Bijo & Pinhas (2019) [12] described the use of an active disturbance rejection controller (ADRC) to estimate and compensate for the effect of slip in an online manner to improve the path tracking the performance of autonomous ground vehicles. It is used as a generalized model, of the many proposed over the years to account for the effect of slip in autonomous ground vehicles. The generic model proposed by them considers the scaling and shift produced in the robot states as result of a slip. Their model uses an observer EKF (extended Kalman filter). The low-level controller uses a simple behavior “go-to-goal”, that guides the robot to the target point from an initial position and orientation.

The experimental validation considered different types of terrain, such as vinyl flooring, asphalt, artificial turf, and grass and gravel. For the ADRC implementation encoders were used on both tracks to obtain the forward and angular velocities. It used a POZYX positioning system, an ultra-wide band positioning system that uses four anchors placed on the perimeter of the experimental area along with a tag placed on the robot. Path following trials were conducted to compare the use of ADRC which provides smooth



corrections and the low-level controller alone (PD) that sub-corrects which results in a jerky motion. Figure 2.10 shows the improvement obtained by using the ADRC over the PD. Notably, the path done has more accuracy when is used ADRC architecture instead of the low-level controller alone.



**Figure 2.10** - Path tracking results on grass and gravel. Adapted from [12]

Zou et al. (2018) [13] proposed a novel approach to the dynamic modeling and motion control of tracked vehicles undergoing skid-steering on horizontal, hard terrain, under nonholonomic constraints. They proposed a motion control methodology, using the backstepping method, based on a modified Proportional-Integral-Derivative (PID) computed-torque controller. To verify the proposed approach were made simulations that resulted in high accuracy of the motion-control performance.

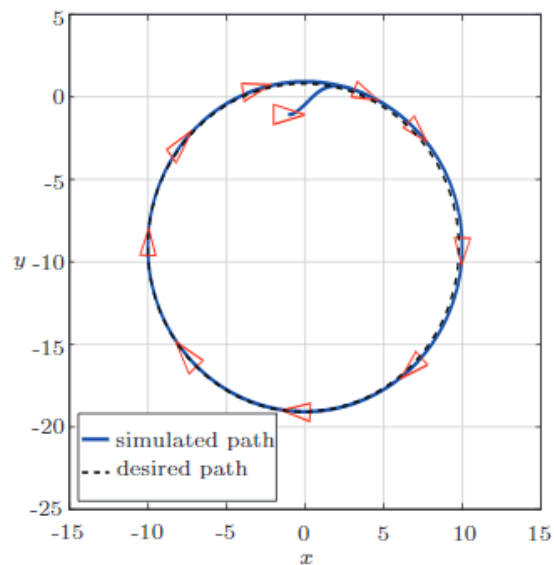
The authors did the modeling of tracked vehicles regarding the kinematics, the nonholonomic constraints of tracked vehicles and the mathematic model.

The kinematics modeling considers two coordinate frames, the vehicle-fixed frame, and the inertial frame. Also consider the slip angle that is caused by the skid-steering turning maneuver of the vehicle. The mathematic model has into account the tractive force, the longitudinal and lateral resistance forces and the turning moment and moment of turning resistance.

Several systems for localization and navigation were proposed, with the authors opting to use three bi-axial accelerometers that form an IMU (inertial measurement unit), which enable the estimation of the pose and the twist of the tracked vehicle. Instead of the regular

IMU composed of accelerometers and gyroscopes, an accelerometer strap-down was employed. A modified PID computed-torque controller was designed for this nonlinear system.

The authors devised an experimental test where the vehicle should follow a circular path with a 10m radius, at a constant angular velocity of 0,8rad/s. During the test was possible to verify that due to the significant slip at the initial stage of motion the trajectory of the tracked vehicle does not converge to the desired path but can follow the desired path quite closely after a significantly short period, as shown in figure 2.12.



**Figure 2.11** - Vehicle following a desired circular path. Adapted from [13]

In manned controlled vehicles the driver decides the optimal turning speed, but in autonomous vehicles, the speed is decided by the autonomous control system. For tracked autonomous vehicles, the velocity is an important factor while turning to maintain the desired path.

Naumov et al. (2019) [14] studied tracked autonomous vehicles' effectiveness estimation while turning. Their study considers the relation power-weight on dry ground roads. The algorithm is represented by them as follows: "when the robotic vehicle is moving, the angular velocities of the right and left boards are determined. The angular velocities enter the valuator unit for calculating the theoretical angular velocity of the chassis steering. Next, go to the divider unit. The actual angular velocity from the sensors is fed into the divider

unit. Next, the comparison with the reference signal and the command to the actuator” and the structural diagram of the system can be seen in figure 2.13.

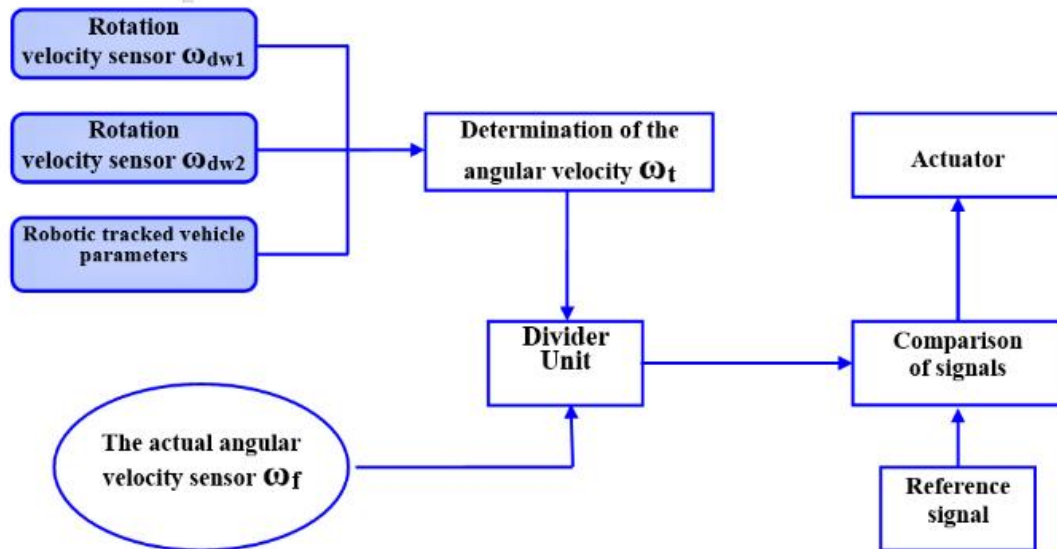


Figure 2.12 - Scheme of the robotic tracked vehicle uncontrolled movement prevention system. Adapted from [14]

After that, they realized some simulations at different speeds with a steering radius of 25 meters. Figure 2.14 shows the path followed by the vehicle during the simulations at different speeds.

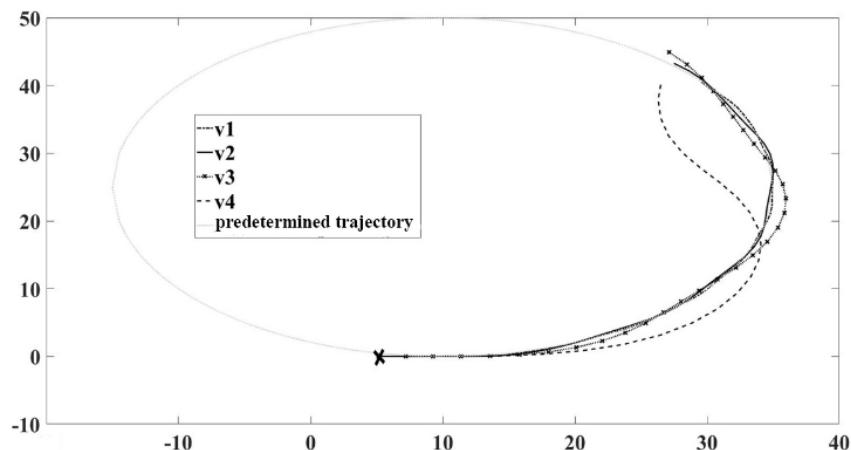


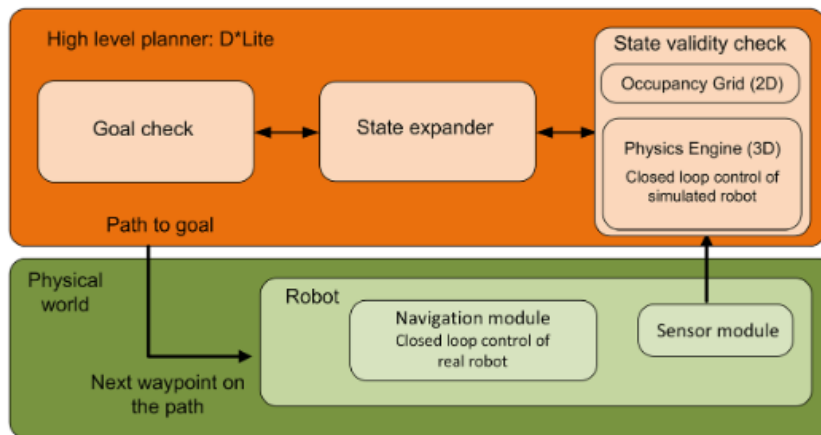
Figure 2.13 - Simulation results. Adapted from [14]

When a goal is defined for the autonomous robot to reach, it is necessary to understand if the robot can avoid all the obstacles that can be found. In that case, the robot needs to select the best path on its own, using systems that allow the robot to define what direction it should take, avoiding obstacles and terrain conditions that it cannot overcome, for that Bijo

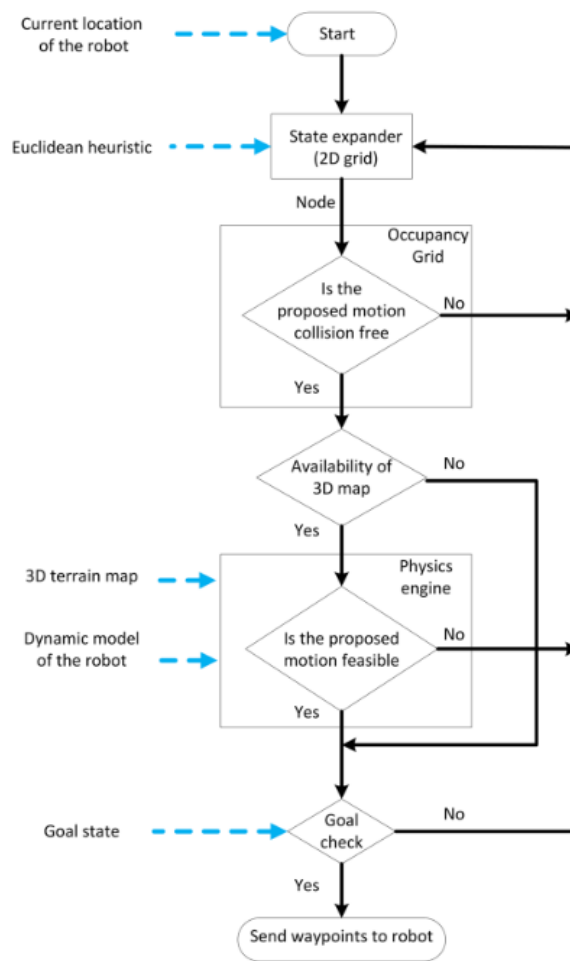
& Pinhas (2019) [15] proposed a method that allows the autonomous system to recognize the terrain conditions and obstacles and defines the path to follow until reaching the goal.

The proposed architecture uses the D\* Lite algorithm working on a 2D grid representation of the terrain as the high-level planner. The proposed approach for the path planning algorithm needs to consider the dynamic interactions between the robot and the terrain by simulating the closed-loop motion of the robot with a low-level controller on a realistic terrain model inside a physics engine.

To obtain the terrain topology of the robot's current position and the close cells, the high-level planner starts with a 2D grid map of the region in which the terrain topology is initially flat, and then with the help of sensors, like LiDAR it actualizes the grid to include close obstacles. After knowing the neighboring cells, the robot tries to reach the next cell, if after a certain time the robot doesn't move, the next cell is defined as unreachable, and the robot tries to reach another neighboured cell. Figure 2.15 shows a block diagram and figure 2.16 a flow chart explaining the work of the proposed method.



**Figure 2.14** - Block diagram representation of the proposed planner. Adapted from [15]



**Figure 2.15** - Flow chart showing the working of the proposed planning algorithm. Adapted from [15]

The low-level plane used continuously monitoring the state of the robot and the environment through sensors to generate control inputs to ensure stable navigation from the current state to the next waypoint.

The system uses two different behaviors, the go-to-goal (GTG), and then avoid obstacle (AO) behaviors. It starts at go-to-goal and stays in it until an obstacle point is detected at a close distance, when it chances to avoid-obstacle-and-go-to-goal, as shown in figure 2.17.

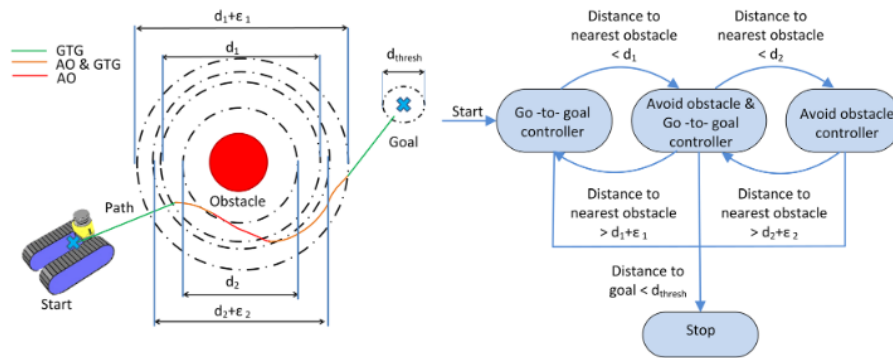


Figure 2.16 - Work of the system. Adapted from [15]

At the experimental validation, the goal was given to the robot, and it is possible to see in figure 2.18 that were proposed two different paths to the robot. The kinematic planner proposed the shortest path that failed because the robot was unable to cross the ridge on the map. While proposed planner, provided with the complete map of the terrain within the physics engine proposed a longer but feasible path that successfully guided the robot to the goal. Concluding that the proposed method works and can guide a robot to the desired point avoiding obstacles and terrain hard conditions.

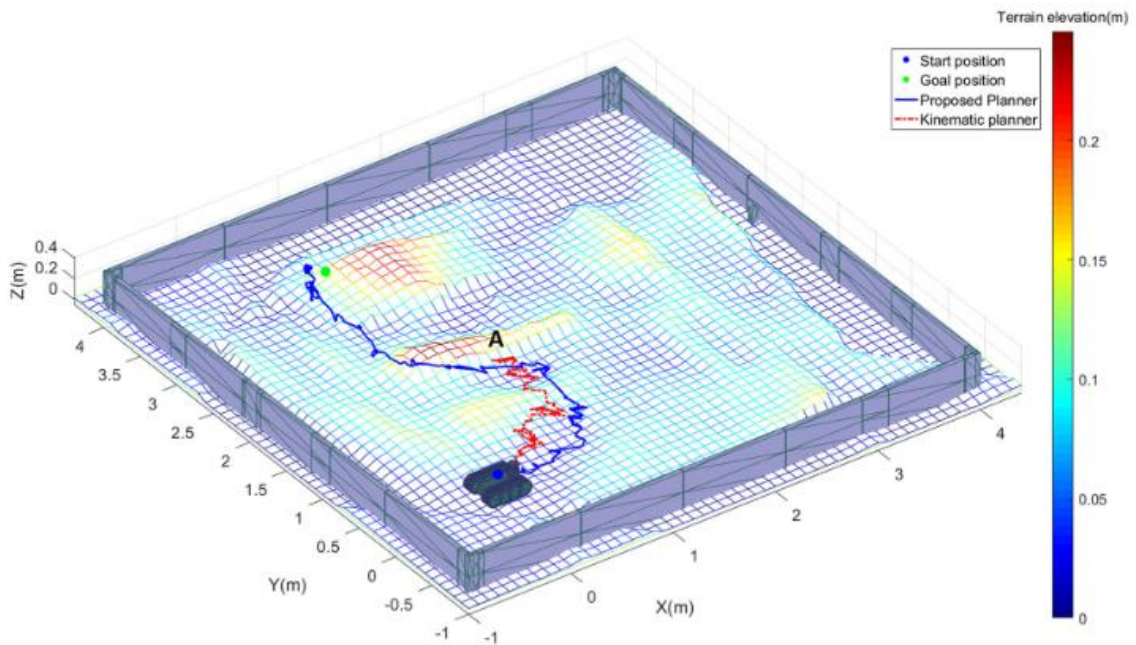


Figure 2.17 – Terrain topography map showing the 3D path followed by the robot under both the planners over the experimental setup. Adapted from [15]

### **3. METHODOLOGY**

This chapter introduces the methodology used in this work, such as the operating system (Ubuntu) of the virtual environment, for performing simulations (ROS), the sensors applied (Duro Inertial) in the real robot (Green Climber LV 400 Pro) and the concepts used for the evaluation of the robot's position (quaternions and Euler angles).

The Ubuntu operating system was used to use ROS and perform simulations that allowed the developing and the test of algorithms for the vehicle's traction control. To test the developed algorithms were installed two different robots and three different virtual environments where the tests were performed.

After developing the algorithm, a validation phase with the real machine followed, where a high-precision GNSS (global navigation satellite system) system was used (Duro Inertial).

#### **3.1. Ubuntu**

Ubuntu is a Linux open-source operating system available for free with professional and community support. It has a built-in firewall and virus protection software making it one of the most secure operating systems, is fully translated into over 50 languages and includes essential assistive technologies.

It provides the fastest way from development to deployment on desktop, mobile, server or cloud. It offers the best development tools and libraries and has the most popular productivity tools such as Zoom, Microsoft Teams, Telegram, and Discord. It also provides easier game and artificial intelligence development with NVIDIA GPU supported out the box hassle-free.[16]

##### **3.1.1. Robot Operating System (ROS)**

ROS is an open-source framework that helps researchers and developers build and reuse code between robotic applications, are a global open-source community of engineers, developers, and hobbyists who contribute to making robots better, more accessible, and available to everyone. ROS is powering the future of robotics in industry, in the enterprise, and for developers.

ROS is used by some of the biggest names in robotics. It is used across numerous industries from agriculture to medical devices and even vacuum cleaners and is spreading to include all kinds of automation and software-defined dynamic use-cases.[17]

#### **3.1.1.1. Why use ROS?**

ROS allows developers to easily simulate their robot environment before deployment in the real world. Tools like Gazebo allow the creation of simulations with countless robotic platforms. The base code and knowledge can be applied across all robotic platforms, like drones, robotic arms, mobile bases, etc.

The robots from ROS can speak any language as Python and C++ and it is even possible to get libraries to allow the use of most other languages or install rosbridge and use any languages that can speak JSON. There are ROS packages for everything, whether to compute a trajectory, conduct SLAM algorithms or implement remote control.[17]

#### **3.1.1.2. ROS Gazebo**

ROS gazebo is an open-source 3D robotics simulator. Gazebo simulates real-world physics in a high-fidelity simulation. It helps developers rapidly test algorithms and design robots using digital environments.

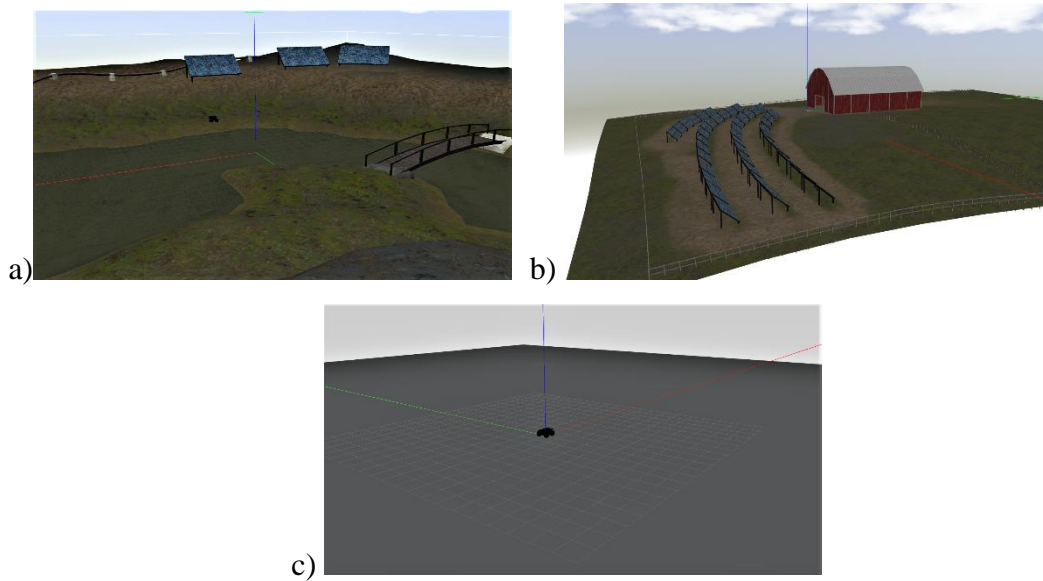
Robotic simulations save a lot of time and money because it allows engineers to test how robots work without having to deploy or risk such robots in a real environment. It helps to replicate gravity, friction, torques, and any other real-life conditions that could affect the robot's behavior and performance.

Gazebo helps to integrate a multitude of sensors, and it gives the tools to test these sensors and develop algorithms to best use them. In situations where it is not possible to access robotic hardware or it is necessary to test hundreds of robots simultaneously, Gazebo allows such simulations, seemingly and hassle-free.[18]

#### **3.1.1.3. Gazebo simulation**

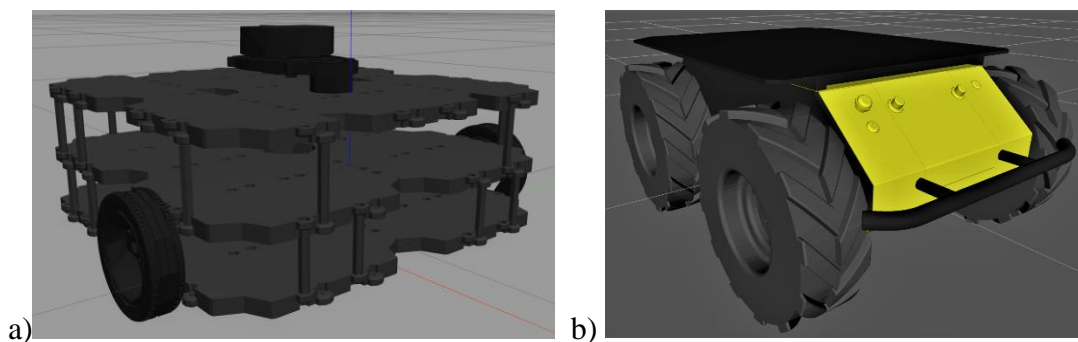
The gazebo has several robots and worlds where the simulations can be performed, from worlds with a completely flat surface to more complex ones, as shown in figure 3.1.





**Figure 3.1 - a)** Inspection world; **b)** Agriculture world; **c)** Empty world

Several robots can be used on gazebo simulations with several configurations that allow multiple different simulations. It is possible to use robotic arms to manipulate objects, aerial robots to simulate flights, and different types of ground robots, as shown in figure 3.2.



**Figure 3.2 - a)** Turtlebot3, model “waffle\_pi”; **b)** Husky robot

#### 3.1.1.4. ROS topics

While performing a simulation in Gazebo, there are several data topics, given by the implemented sensors on the robots which are constantly updated, and accessible, providing critical data about the robot. The topics available depend on which sensors are implemented on each robot. For the Husky robot, the available topics are shown in figure 3.3.

```

a) *top@ttg:~$ rostopic list
/clock
/cmd_vel
/diagnostics
/e_stop
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/husky_velocity_controller/cmd_vel
/husky_velocity_controller/odom
/husky_velocity_controller/parameter_descriptions
/husky_velocity_controller/parameter_updates
/imu/data
/imu/data/accel/parameter_descriptions
/imu/data/accel/parameter_updates
/imu/data/bias
/imu/data/rate/parameter_descriptions
/imu/data/rate/parameter_updates
/imu/data/yaw/parameter_descriptions

b) /imu/data/yaw/parameter_updates
/joint_states
/joy_teleop/cmd_vel
/joy_teleop/joy
/joy_teleop/joy/set_feedback
/navsat/fix
/navsat/fix/position/parameter_descriptions
/navsat/fix/position/parameter_updates
/navsat/fix/status/parameter_descriptions
/navsat/fix/status/parameter_updates
/navsat/fix/velocity/parameter_descriptions
/navsat/fix/velocity/parameter_updates
/navsat/vel
/odometry/filtered
/rosout
/rosout_agg
/set_pose
/tf
/tf_static
/twist_marker_server/cmd_vel
/twist_marker_server/feedback
/twist_marker_server/update
/twist_marker_server/update_full
    
```

Figure 3.3 - ROS topics

### 3.1.1.5. Rviz

Rviz is a ROS tool that allows the visualization of the robot and the data from the sensor implemented in it in real-time. It is a highly configurable environment, being possible to add or remove data information according to needs or availability. The tool is used simultaneously with the simulations in the Gazebo and provides all the information about the world of Gazebo simulation. Figure 3.4 it is possible to see some of the sensors that can be applied to the robot and figure 3.5 shows the tool in action, with some data information being provided and shown on the map.

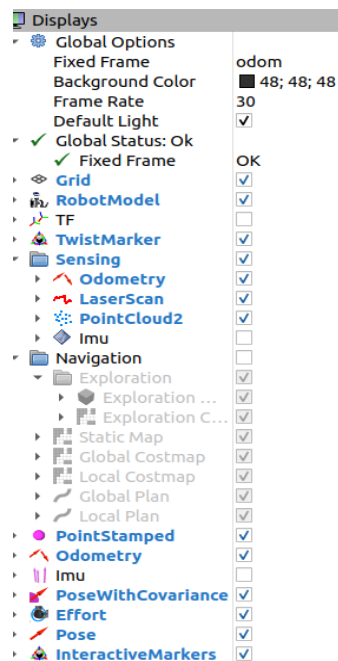


Figure 3.4 – Some of Rviz’s available sensors.

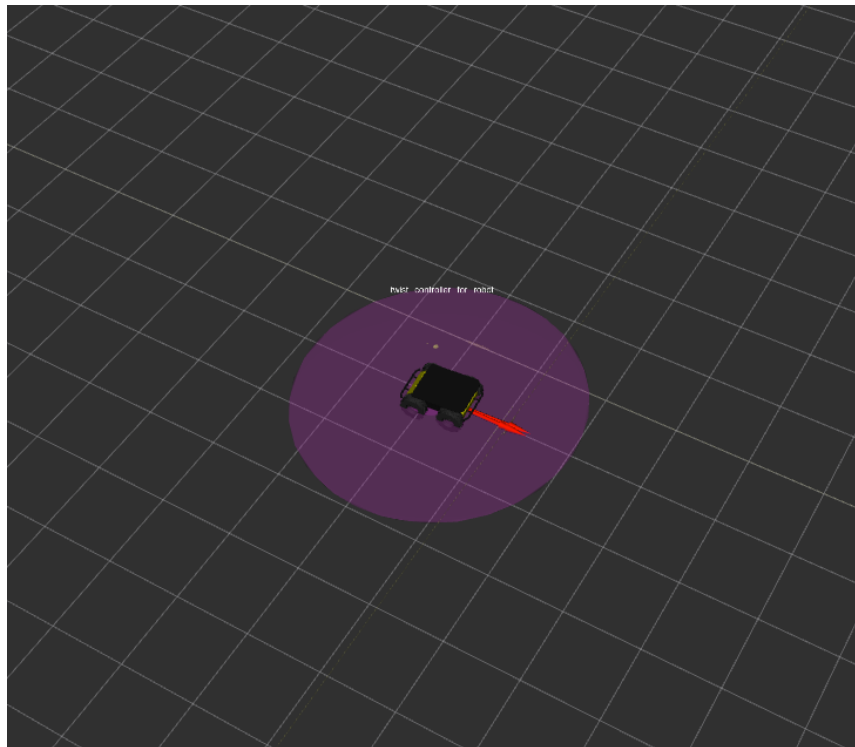


Figure 3.5 - Data demonstration on map.

## 3.2. Robot Orientation

### 3.2.1. Quaternions

The quaternions were discovered by Sir William Rowan Hamilton and are an extension of the complex number, a hyper-complex number, and are written as a scalar plus a vector, as shown in equation 3.1. Its representation can be seen in figure 3.13. [27]

And are denoted as shown in equation 3.2.

$$\dot{q} = s + v = s + v_1i + v_2j + v_3k \quad (3.1)$$

$$\dot{q} = s \langle v_1, v_2, v_3 \rangle \quad (3.2)$$

Where the orthogonal complex numbers are defined as in equation 3.3

$$i^2 = j^2 = k^2 = ijk = -1 \quad (3.3)$$

The matrix of rotations is given by equation 3.4.

$$R(s, v) = \begin{bmatrix} 2(s^2 + v_1^2) - 1 & 2(v_1v_2 - sv_3) & 2(v_1v_3 + sv_2) \\ 2(v_1v_2 + sv_3) & 2(s^2 + v_2^2) - 1 & 2(v_2v_3 - sv_1) \\ 2(v_1v_3 - sv_2) & 2(v_2v_3 + sv_1) & 2(s^2 + v_3^2) - 1 \end{bmatrix} \quad (3.4)$$

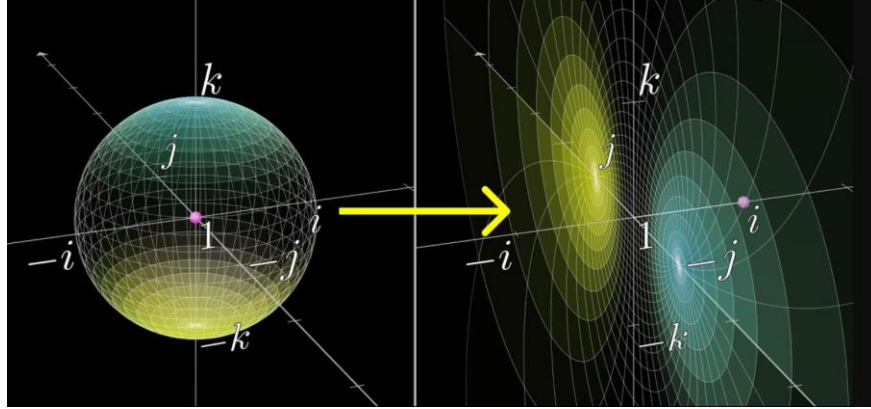


Figure 3.6 - Quaternions representation. Adapted from [28].

### 3.2.2. Euler angles

The Euler angles, roll, pitch, and yaw are angles sequences, relative to a fixed frame reference, widely used in the description of vehicles behavior. The roll, pitch, and yaw rotation matrix can be expressed as shown in equation 3.5.

$$\text{RPY}(\phi, \theta, \psi) = \begin{bmatrix} C(\phi)C(\theta) & -S(\phi)C(\psi) + C(\phi)S(\theta)S(\psi) & S(\phi)S(\psi) + C(\phi)S(\theta)C(\psi) \\ S(\phi)C(\theta) & C(\phi)C(\psi) + S(\phi)S(\theta)S(\psi) & -C(\phi)S(\psi) + S(\phi)S(\theta)C(\psi) \\ -S(\theta) & C(\theta)S(\psi) & C(\theta)C(\psi) \end{bmatrix} \quad (3.5)$$

Where the letters “C” and “S” mean cos and sin, respectively.

The roll corresponds to the rotation on the x axe, the pitch to the y axe, and the yaw to the z axe, as shown in figure 3.7.

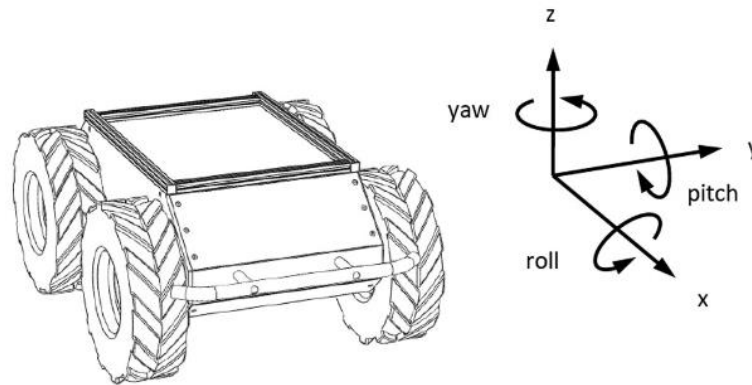


Figure 3.7 - Roll, pitch, and yaw axes. Adapted from [29].

### 3.3. Swift Navigation

Swift Navigation offers services (Skylark™), software (Starling®), and hardware (SwiftPath™, Duro®, Duro Inertial, Piksi® Multi, Piksi Multi Inertial) for several kinds of mechanisms, as shown in figure 3.8, allowing precise positioning solutions to deliver proven performance and high accuracy.





								
	Automotive	Rail	Delivery	Mobile	Industrial	Robotics	UAS	Micromobility
SERVICE								
SOFTWARE	Skylark™	✓	✓	✓	✓	✓	✓	✓
Starling®	✓	✓	✓	✓	✓	✓	✓	✓
SwiftPath™			✓		✓	✓	✓	✓
HARDWARE								
Duro®		✓						
Duro Inertial		✓				✓		
Piksi® Multi		✓				✓	✓	
Piksi Multi Inertial		✓				✓	✓	

Figure 3.8 - Swift Navigation ecosystem. Adapted from [19]

#### 3.3.1. Skylark™

Skylark™ is a wide area, cloud-based GNSS correction service that provides real-time high-precision positioning to autonomous vehicles, automotive, mobile, and mass-market applications. It delivers seamless corrections to continents across the globe. It was built from the ground-up for autonomy at scale, allowing lane-level positioning, fast convergence times, and high integrity and availability required by mass-market automotive and

autonomous applications. It uses observations of hundreds of GNSS reference stations around the globe to provide real-time atmospheric and other errors affecting GNSS. The corrected data is available on the internet for the user and can be accessed anywhere within the Skylark network, the connected users only need to turn on their devices to access to get the correction stream they need. The Skylark™ network is shown in figure 3.9.[20]

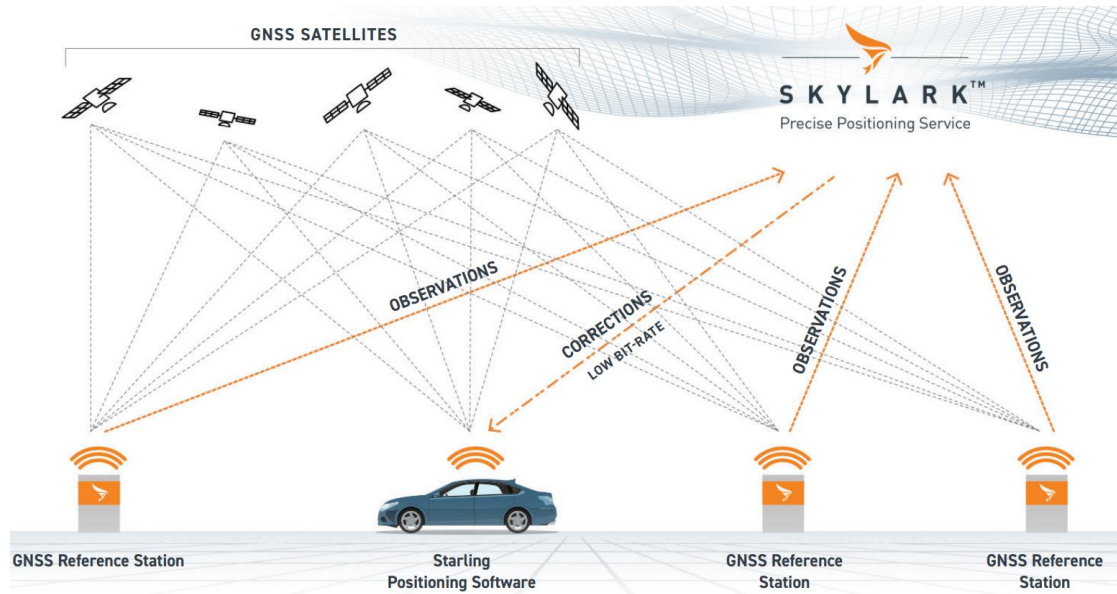


Figure 3.9 - Skylark™ network. Adapted from [20]

### 3.3.2. Starling®

Starling® is an advanced, high-precision positioning engine designed for automotive, industrial, and consumer applications that require centimeter or decimeter accuracy using GNSS and dead reckoning sensor fusion. A flow chart of Starling® is shown in figure 3.10.

It is ideal for mass-market autonomous applications because it works with a variety of GNSS chipsets and inertial sensors. Its software is written in C++ which makes it possible to run on a variety of computing platforms. It uses commercially available GNSS receivers to provide centimeter accuracy and high integrity. It supports the calculation of precise position, velocity, and time (PVT), and when combined with inertial sensor measurements, wheel odometry, and other sensors inputs, it can assist with localization, decision, and control.

Starling® works with multi-frequency, multi-constellation and commercial grade GNSS measurements engines that when combined with the wide-area Skylark™ cloud-

based GNSS precise positioning service, significantly reduce the cost of high-accuracy positioning for autonomous applications.[21], [22]

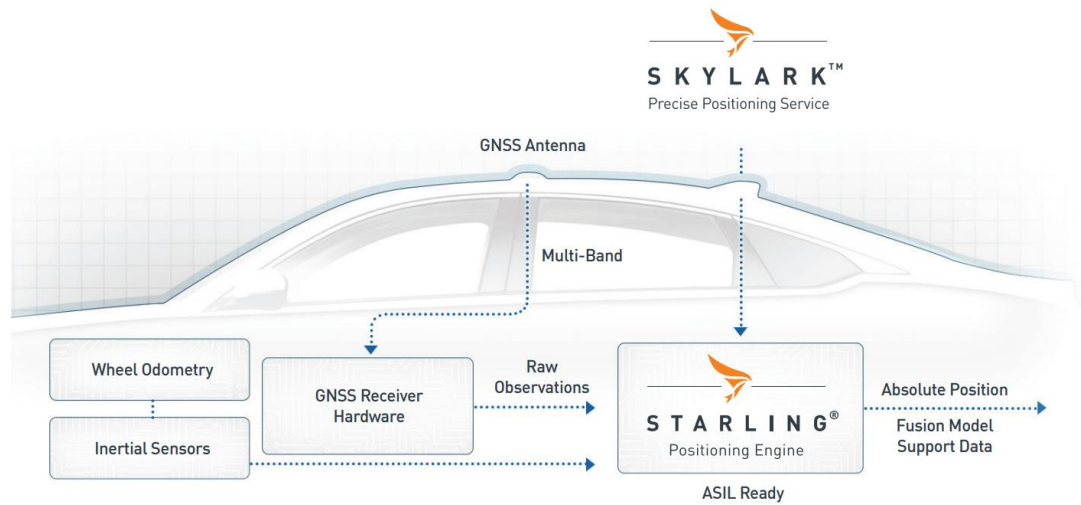


Figure 3.10 - Flow chart of Starling®. Adapted from[22]

### 3.3.3. Duro Inertial

The Duro Inertial combines the position, velocity and time solution of the Starling® position engine with the on-board IMU to deliver a continuous and precise positioning solution, even when the visibility of GNSS is low or none.

It is easy to integrate and supports several interfaces, including RS232, CAN (controller area network), and Ethernet. The Starter Kit has everything necessary for easy deployment and is available in an RTK Pack for highly accurate assessment with an IMU.

It has military-grade ruggedness, is packed in an IP67-rated ruggedized enclosure, and is designed for deployment in rough terrains. The Duro Inertial will be used on the real robot to provide data about it as can be seen in figure 3.11.[23]



Figure 3.11 - Duro Inertial. Adapted from [23]

The typical architecture of the Duro Inertial system is shown in figure 3.12.

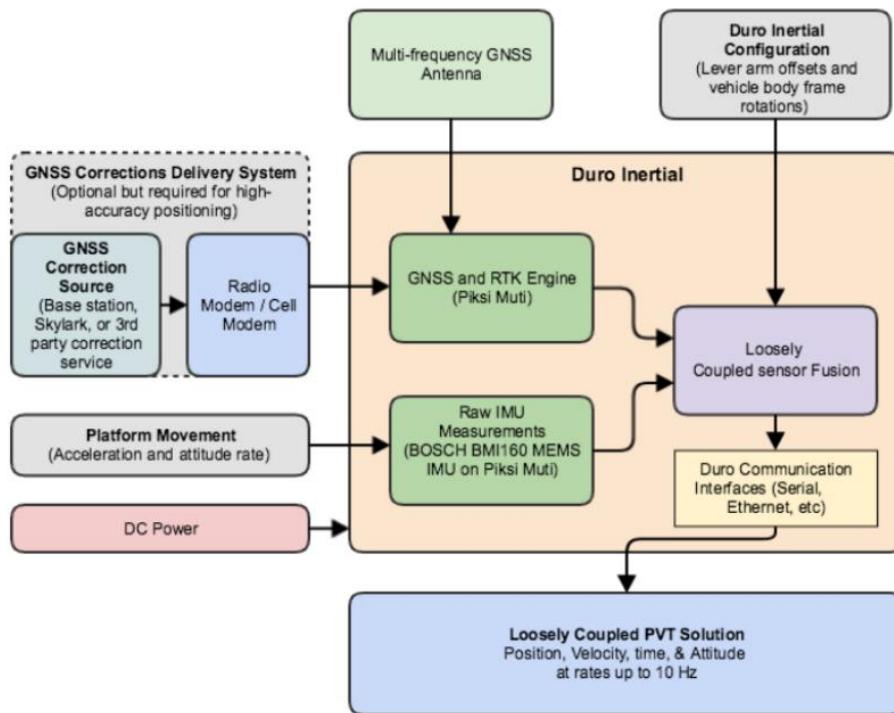


Figure 3.12 - Duro Inertial system architecture. Adapted from[24]

The Duro Inertial data can be consulted on the Swift Console, which provides data from the IMU, the Magnetometer, the INS and also the speed and velocity, as shown in figures 3.13 and 3.14.



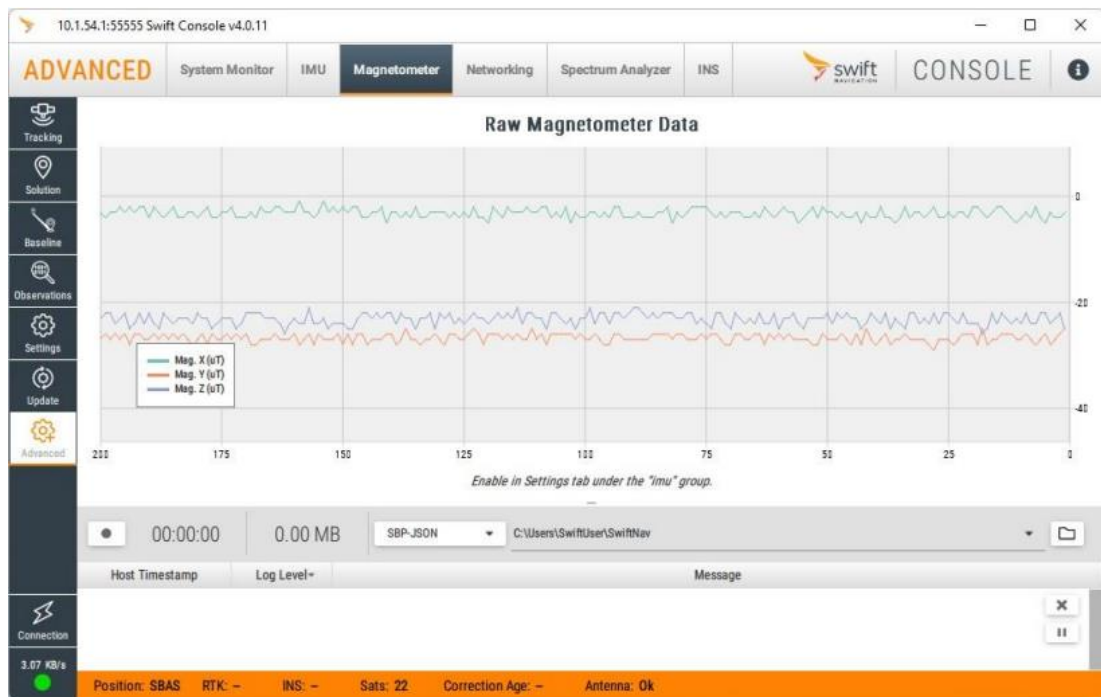


Figure 3.13 - Swift Console Magnetometer window. Adapted from[25]

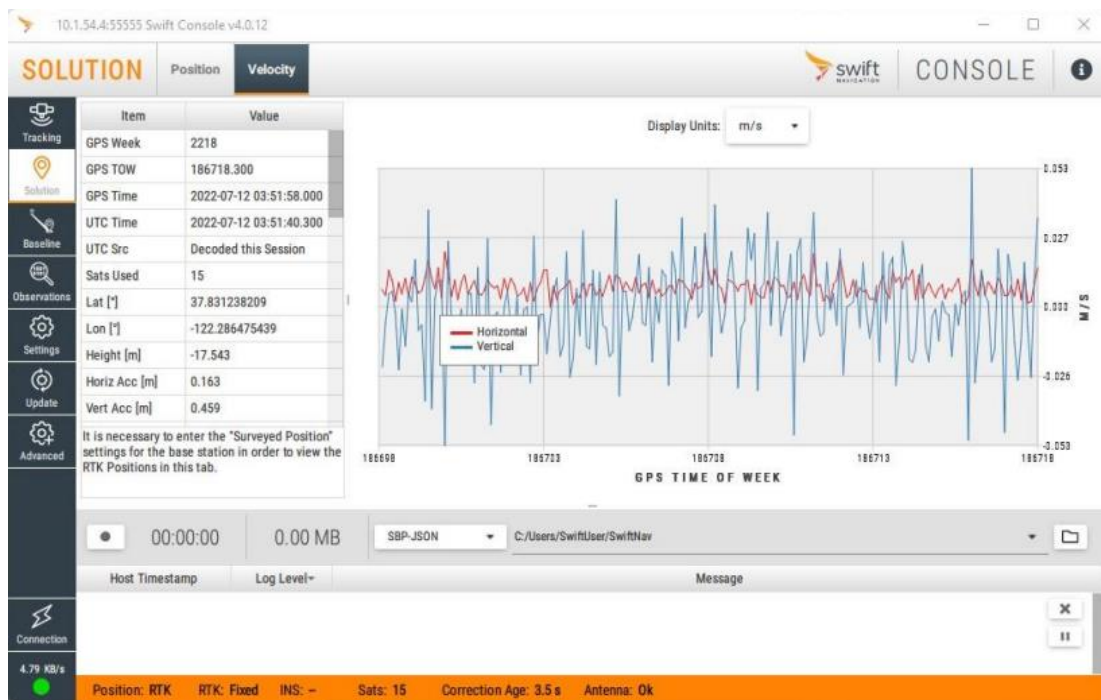


Figure 3.14 - Swift Console Velocity window. Adapted from[25]

The Duro Inertial combines the GNSS, RTK and IMU technologies and has continuous position outputs even when GNSS is not available, and increased robustness to challenging GNSS environments.

The Hardware is future-proof with in-field software upgrades and has intuitive LEDs for status and diagnostic and flexible and electrically protected ports.

The provided solutions are 100 times more accurate when utilizing RTK technology in conjunction with GNSS than standard GNSS-only solutions. It can operate in temperatures that go from  $-40^{\circ}\text{C}$  to  $75^{\circ}\text{C}$  and can provide position update rate (GNSS + INS (inertial navigation solution)), measurement (raw data), standard position outputs, and RTK position outputs up to 10 Hz and has a maximum operating velocity of 515 m/s.[26]

The Duro Inertial can provide the Euler angles that allow knowing the vehicle frame orientation and it is also possible to use the Euler angles to configure the device frame orientation to the vehicle frame orientation. For example, a rotation matrix that can be used to rotate a vector from the device frame to the vehicle frame can be represented mathematically as follows in equation 3.6.[24]

$$\begin{bmatrix} v_{x-vehicle} \\ v_{y-vehicle} \\ v_{z-vehicle} \end{bmatrix} = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(pitch) & 0 & \sin(pitch) \\ 0 & 1 & 0 \\ -\sin(pitch) & 0 & \cos(yaw) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) \\ 0 & \sin(roll) & \cos(roll) \end{bmatrix} * \begin{bmatrix} v_{x-device} \\ v_{y-device} \\ v_{z-device} \end{bmatrix} \quad (3.6)$$

## 4. WORK DEVELOPMENT

This chapter describes the work developed to create the robot control.

For the work development, it was necessary to become familiar with the Python programming language and the ROS. The Turtlebot3 was installed in ROS to be used in simulations and code development for the robot control. One of the biggest difficulties in Gazebo's simulations is the instability of the robot coordinates, which are constantly changing even with the robot stopped. Initially, Turtlebot3 was used in a planar world, in which the control of the robot was developed because although the world has no slopes, the robot tends not to perform a movement in a straight line.

The control was developed in phases, starting initially with the conversion of the quaternions into Euler angles, followed by the development of the code that keeps the robot in a straight line. Once the robot control was developed, it was necessary to perform the simulation in virtual worlds with real characteristics. For this, it was necessary to install the Husky robot and the worlds so that any necessary adjustments could be made in the control code.

The work procedures are described in more detail throughout this chapter.

### 4.1. ROS Topics

The most important topics, in this case, are the "*imu/data*" that provide the orientation of the robot in quaternions  $(x, y, z, w)$ , the angular velocity  $(x, y, z)$  and the linear acceleration  $(x, y, z)$  of the robot, shown at figure 4.1, and the "*cmd\_vel*" that provides the linear velocity  $(x, y, z)$  and the angular velocity  $(x, y, z)$  of the robot, shown at figure 4.2.

```
^Cttgp@ttgp:~$ rostopic echo /imu/data
header:
  seq: 36758
  stamp:
    secs: 735
    nsecs: 52000000
  frame_id: "base_link"
orientation:
  x: -0.07600428214777054
  y: 0.013128372055544518
  z: 0.05189899877927153
  w: 0.9956693672440076
orientation_covariance: [2.6030820491461885e-07, 0.0, 0.0, 0.0, 2.6030820491461885e-07, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: -0.0012352645002861053
  y: 0.010050342204558494
  z: -0.0036990155201906878
angular_velocity_covariance: [2.5e-05, 0.0, 0.0, 0.0, 2.5e-05, 0.0, 0.0, 0.0, 2.5e-05]
linear_acceleration:
  x: -0.330144309881917
  y: -1.4743782956413747
  z: 9.691596924159828
linear_acceleration_covariance: [2.5e-05, 0.0, 0.0, 0.0, 2.5e-05, 0.0, 0.0, 0.0, 2.5e-05]
---
```

Figure 4.1 - ROS topic `"imu/data"` data. The presented values serve as a mere example.

```
^Cttgp@ttgp:~$ rostopic echo /cmd_vel
linear:
  x: -0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: -0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: -0.5
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

Figure 4.2 – ROS topic `"cmd_vel"` data. The presented values serve as mere examples.

From the data provided by these two topics, the most relevant is the orientation of the robot and the linear velocity.

The orientation of the robot in quaternions can be transformed in Euler angles (roll, pitch, and yaw) allowing a much easier understanding of the robot orientation. To achieve that objective, it was created a python code that can read, in real-time, the values of the quaternions given by the IMU and transforms, through equations of the library, the x, y, z, and w values into Euler angles. A range of values was constructed for the yaw angle. This means that whenever the yaw value is in an interval of  $+0.00001$  and  $-0.00001$  concerning the first read value, this yaw value will be considered to be equal to the initial yaw value.

This strategy was employed to filter the signal noise on the magnetic compass and IMU. The data provided by the IMU that allows the conversion of the quaternions to Euler angles are constantly changing and it has 17 decimal places which means that to keep a constant yaw, without a defined range of values, there is the possibility that once this value changes, the data provided will never match that value again and so, even if the value is very close to the initial one, without this approximation the program would assume that it never returned to its initial value. The defined range of values was chosen through tests, for the initial development of the code it started with a different range of values from the one applied at the end of the code, because once the control was developed it was possible to verify the influence of this range on the success of the control.

To write the code that allows transforming quaternions into Euler angles is necessary to have access to the robot's IMU, so, while writing it is necessary to import that data to use it. In figure 4.3 is possible to see the command line that does the importation of the data.

```
import rospy #Pure Python client library for ROS
from sensor_msgs.msg import Imu #Import Imu data
from tf.transformations import euler_from_quaternion #Import transformation function
```

Figure 4.3 - Importing library, data, and functions to code.

Once the data from IMU was available to use, it was necessary to define a function that read, transform and filter the data. Initially, the quaternions values and the path to get them were defined, after that, through the python function “*euler\_from\_quaternion*”, imported to the code, as shown in figure 4.3, it was possible to transform the quaternions (x, y, z, w) into Euler angles (roll, pitch, yaw). As soon as the transformation was done, an “if” function defined the limit to the values of the yaw angle, as shown in figure 4.4.

```
if yaw < yaw_initial + 0.00001 and yaw > yaw_initial - 0.00001: #Rounding small values of Yaw to 0
    yaw = yaw_initial
```

Figure 4.4 – Approximate small variations to the initial yaw.

Then it was necessary to define a new function that read the values of the Euler angles and publish them into a new topic, created for this purpose. The robot data of roll, pitch, and yaw is now available to understand more about the robot's orientation. The Python conversion code can be seen in annex A.

## 4.2. Operating Mode of the Control Code

Once the code to convert quaternions into Euler angles is done, it is possible to read and manipulate the values of the angles and know more about the robot orientation. The code to control the robot is based, essentially, on the control of the yaw values given by the sensors of the robot and its angular velocity, whenever the yaw values suffer alterations.

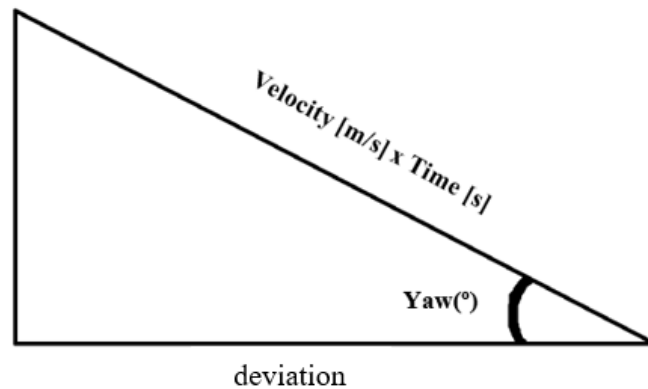
To make sure that the robot is moving in a straight line, ideally, the yaw values should remain constant, meaning that the robot is not suffering any deviation from its intended path. If the yaw value suffers any alteration, means that the robot is starting to move in a different direction than the one intended.

To correct the trajectory, it is necessary to know the distance traveled since the variation of the yaw was detected. To calculate the traveled distance, it is only necessary to know the velocity of the robot which is given in the ROS topic “*cmd\_vel*”, the new yaw value in radians which must be converted to degrees using the equation 4.1 and the time in seconds since the deviation was detected. The deviation can now be calculated through equation 4.2. The time calculation is done by the control code, which starts to count at the moment that a deviation is detected.

$$\text{yaw}[^{\circ}] = \text{yaw}[\text{rad}] \times \frac{180}{\pi} \quad (4.1)$$

$$\text{deviation} = \cos(\text{yaw}[^{\circ}]) \times \text{velocity} [\text{m/s}] \times \text{time} [\text{s}] \quad (4.2)$$

Now it is possible to know the deviation value, represented as “deviation” in the figure 4.5.

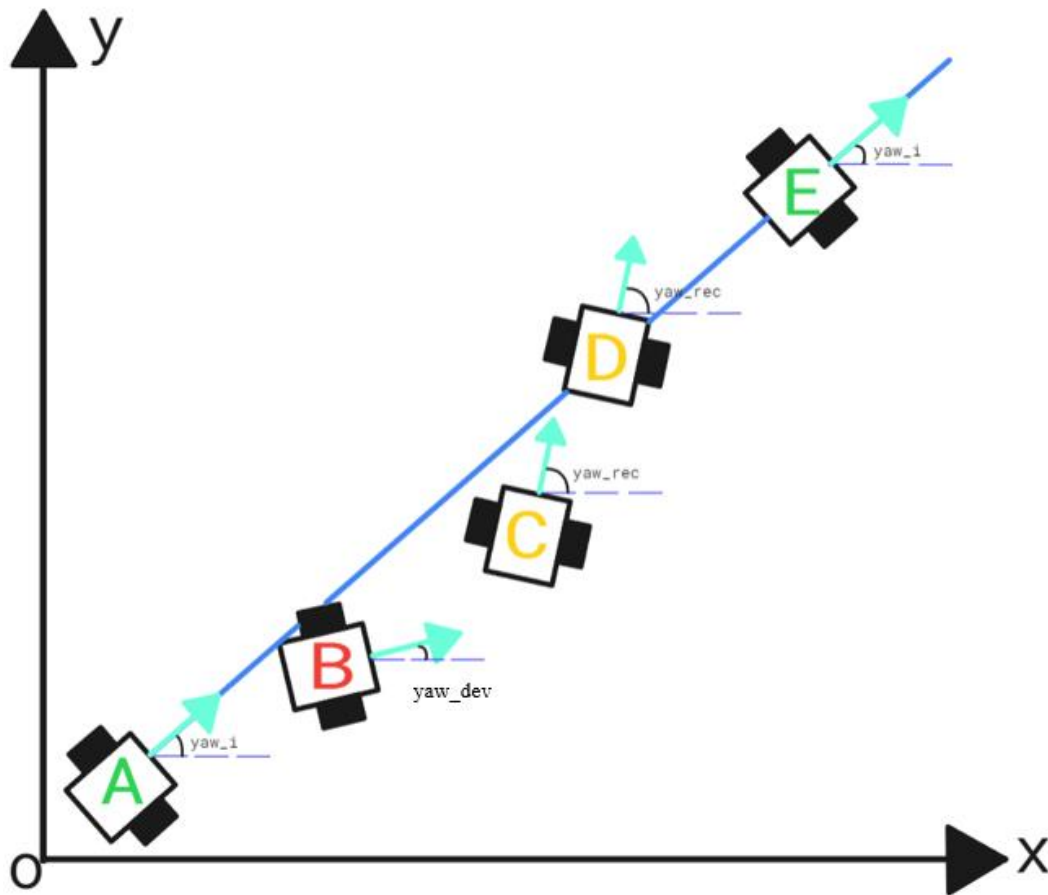


**Figure 4.5** - Deviation (x) calculation

The calculation of the deviation is also done by the python code and once is done it is necessary to recover the deviation already suffered. To start the recovery the first step is to transform the yaw value into negative if it was positive, or into positive if it was negative, and give the robot an angular velocity, contrary to the deviation movement, making it turn to the other way and start to move in the opposite direction. Regardless of the yaw of deviation being positive or negative it starts a new time counting and the yaw value is once again converted to degrees using equation 4.1, and the recover distance can be calculated by equation 4.3 until it is equal to the deviation distance.

$$\text{recover} = \cos(\text{yaw}[^{\circ}]) \times \text{velocity [m/s]} \times \text{time [s]} \quad (4.3)$$

Once the distance of recovery is equal to the distance of deviation, the yaw value should now be the initial value again and the angular velocity returns to the value of zero, causing the robot to return to its initial direction and trajectory.



**Figure 4.6** – Expected robot movement while the code is running, and a deviation is detected.

On an enlarged scale, while the code is working, it is supposed to see the robot being controlled, as shown in figure 4.6, where the initial position is represented by the letter A and de initial yaw value is represented by “*yaw\_i*”. At this moment the robot is moving in a straight line with constant yaw value and velocity. At position B, it was detected a deviation and at that moment, the yaw value, is represented by “*yaw\_dev*”, and the code already started calculating the time and converting the yaw from radians into degrees, to get the value of the deviation. In position C, an angular velocity was given to the robot and a new time is running to calculate the robot’s recovery. It is also necessary to know the new yaw value, that is given by the multiplication of the “*yaw\_dev*” by -1. At position D, the yaw value is still the same as it was at C, but it detected the total recover of the trajectory, so a new yaw value applies to the robot, being the same as the beginning. The robot keeps moving in a straight line until it suffers a new deviation, repeating the process while the user doesn’t shut it down.



The linear velocity of the robot is constant during the entire process and its frame is shown in figure 4.7. The Python code can be seen in annex B.

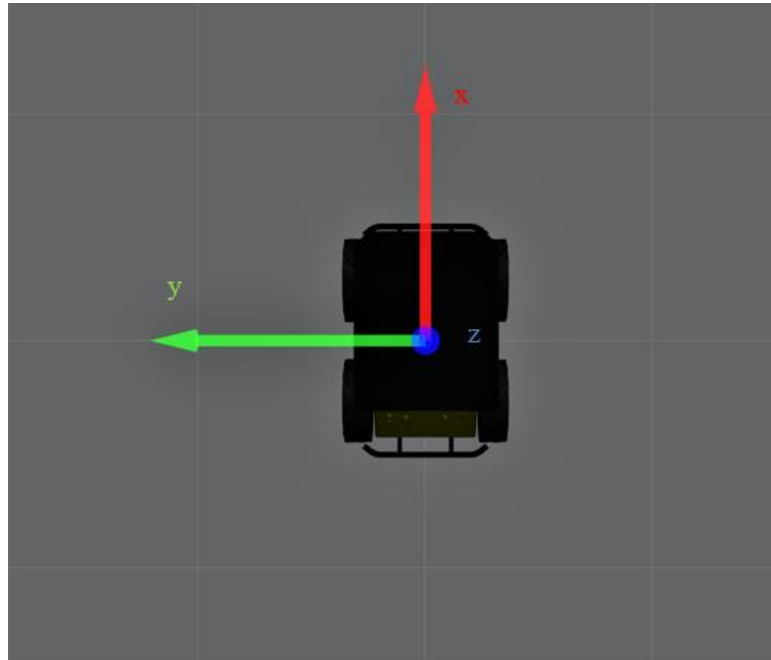


Figure 4.7 - Robot frame.

### 4.3. Calculation of expected final position

The final position of the robot can be estimated in at least two different ways. The first one is calculating the line equation, for that is only necessary to know the value of the initial yaw and the robot coordinates  $(x, y)$  at the beginning, and if the robot is moving in a straight line, it should keep its movement through that line, so the final position, will be a point on that line. To estimate the final position of the robot in a second way, it is necessary to know the velocity of the robot, the time it is running, and the initial value of the yaw. After that, it is possible to know the expected final position.

In both cases, it is possible to do a comparison between the estimated values and the actual values obtained for both movements, with the control code activated to keep the robot moving in a straight line and to the uncontrolled movement.

#### 4.3.1. Line equation

The line equation can be obtained by knowing the  $x$  and  $y$  coordinates, the slope of the line ( $m$ ), and one constant value ( $b$ ), as shown in equation 4.4.

$$y = mx + b \quad (4.4)$$

The calculation of the constant slope of the line,  $m$ , is done by knowing the initial value of yaw in degrees and can be done by equation 4.5.

$$m = \tan(\text{yaw}[^{\circ}]) \quad (4.5)$$

After that, and by knowing the initial coordinates of the robot ( $x$ ,  $y$ ) the value of the constant  $b$  can now be calculated, replacing in equation 4.4 the values of  $x$  and  $y$  by the initial coordinates of the robot, so the  $b$  value is given by the equation 4.6.

$$b = y - mx \quad (4.6)$$

When the initial value of yaw is zero, that means that the slope value will be null and the constant  $b$  will be equal to the  $y$  coordinate of the robot meaning that the robot is only moving in the  $x$  and  $z$  axes, keeping the  $y$  value constant from the beginning of the movement until it stops.

If the robot movement was done in a straight line, at the end of his movement  $x$  and  $y$  coordinates should belong to the line, if they don't belong to it, it means that the robot has suffered a deviation and ended up moving on a different direction than the initial.

#### 4.3.2. Calculate the final coordinates

To calculate the final coordinates a time count has to be done, that together with the initial value of yaw and the movement velocity of the robot, can provide the final coordinates of the robot. The traveled distance on the  $x$  and  $y$  axes are given by equations 4.7 and 4.8, respectively.

$$x_{final} = x_{initial} + \cos(\text{yaw}[^{\circ}]) \times \text{velocity [m/s]} \times \text{time [s]} \quad (4.7)$$

$$y_{final} = y_{initial} + \sin(\text{yaw}[^{\circ}]) \times \text{velocity [m/s]} \times \text{time [s]} \quad (4.8)$$

Then, it is possible to get the expectable values of  $x$  and  $y$  of the robot at the end of its trajectory, assuming that the robot was always moving in a straight line. The detection of a deviation is done by calculating the supposed final coordinates and comparing the obtained values with the real coordinates of the robot.



## 5. VIRTUAL SIMULATIONS AND RESULTS

This chapter will show the tests performed to validate the created control and the results obtained in each test, making a comparison between the use and non-use of the control in the robot's movement. The tests were carried out on three different maps, with two different robots. For each position of the robot and velocity, two tests were done, one using the control and the other one using the “*Teleop*” (velocity command name, to remotely control the mobile robot via keyboard, it allows to drive the robot through the virtual environment without the need to implement code that gives the robot information about what to do). Tests were also carried out to determine the angular velocity that would be used whenever the robot needed to recover its trajectory and also to determine the best approximation to the yaw values.

A completely flat map was used with the Turtlebot3 waffle\_pi robot moving at three different speeds in each test. The Husky robot was also used to carry out tests on the “*agriculture world*”, a map that contains some slight slopes, and “*inspection world*”, a map that contains very steep slopes. In these two maps, the tests were also carried out using different speeds and, in addition, with different initial positions of the robot.

### 5.1. Realized tests

#### 5.1.1. Angular velocity

Whenever the robot suffers a deviation, to enable recovery, it needs a certain value of angular velocity, so that it is possible to return to the initial trajectory. As the linear speed of the robot can change, it was decided to test the angular velocity as a percentage of the linear velocity of the robot. To determine the best angular velocity, several tests were performed with different values of angular velocity and with some parameters to consider, such as:

- Recovery should be smooth;
- Recovery should be relatively fast to avoid large deviations from the desired trajectory;
- Recovery should not induce a new deviation.

### 5.1.2. Yaw approximation

Tests were carried out to select the best approximation of the yaw angle value to its initial value. The need for this approximation stems from the fact that the yaw values contain 17 decimal places and are the transformation of the IMU quaternions, which contains noise and can cause the robot to move in a circular path after the first deviation is detected, as it is difficult to obtain the exactly initial yaw value again and thus closing the recovery cycle and starting the straight line trajectory again. To determine the best approximation, several tests were performed, increasing one decimal place at each test and it was verified in which of the situations the robot's trajectory was more linear.

### 5.1.3. Empty world

To carry out the tests the robot was placed at the origin of the map, due to the constant change of values of the robot coordinates, its position is just approximated to the origin (location 1). The tests were carried out with three different linear speeds, a test with a speed of 0.11 [m/s], a test with 0.15 [m/s] and a test with the maximum speed of the robot of 0.25 [m/s], being verified by initial coordinates of the robot for each test, to predict the final position of the robot. For each velocity, the tests were repeated three times, and the average coordinates, the average initial value of yaw, and the respective velocity can be seen in table 2.

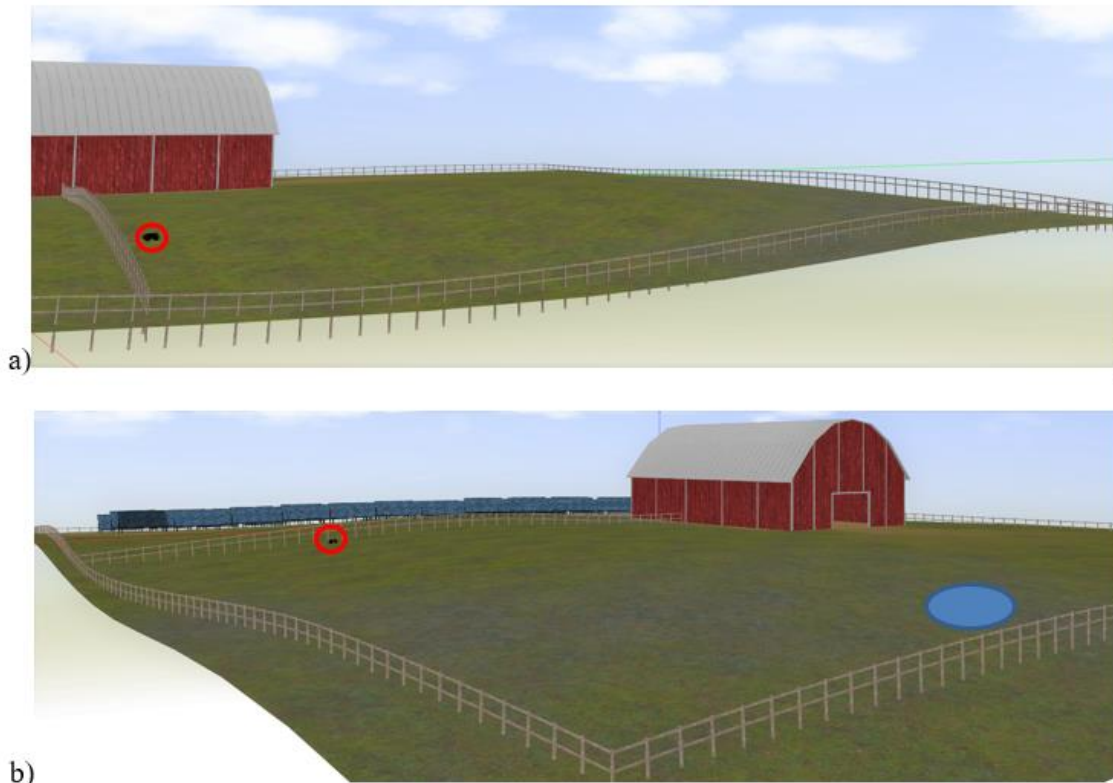
**Table 2** - Average initial robot position for each realized test, at location 1.

Location 1	Vx = 0.11 [m/s]		Vx = 0.15 [m/s]		Vx = 0.25 [m/s]	
	Code	Teleop	Code	Teleop	Code	Teleop
<b>x initial</b>	-0.000005	-0.000005	-0.000005	0.000005	-0.000005	-0.000005
<b>y initial</b>	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001
<b>yaw initial</b>	0.000004	0.000004	0.000004	0.000008	0.000004	0.000004

### 5.1.4. Agriculture world

In order to carry out the tests in the agriculture world, the robot was positioned with a certain yaw value, approximated in each test due to the instability of the simulations, so that the trajectory it should travel passed through the areas of the map with the greatest slopes (location 2). Twelve tests were performed, six of them without the use of the control and the other six with the use of it and were also used at three different speeds, approximately, 0.8053 [m/s], 1.0718 [m/s] and 1.5692 [m/s], for each velocity, the test was repeated twice.

The initial position of the robot can be seen in figure 5.1, and the blue circle represents the zone where the robot should stop.



**Figure 5.1** - Approximately initial position of the robot in each test of location 2.

The average position of the robot at the beginning of each test can be seen in table 3.

**Table 3** - Medium initial position of the robot for each realized test in agriculture world, at location 2.

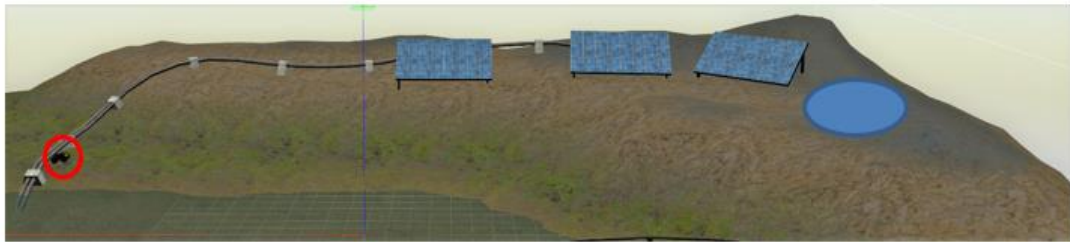
Location 2	Vx = 0.8053 [m/s]		Vx = 1.0718 [m/s]		Vx = 1.5692 [m/s]	
	Code	Teleop	Code	Teleop	Code	Teleop
x initial	57.0039	57.0039	57.0039	57.0039	57.0039	57.0039
y initial	12.9984	12.9984	12.9986	12.9983	12.9985	12.9985
yaw initial	1.578652	1.578622	1.578632	1.578634	1.578628	1.578647

### 5.1.5. Inspection world

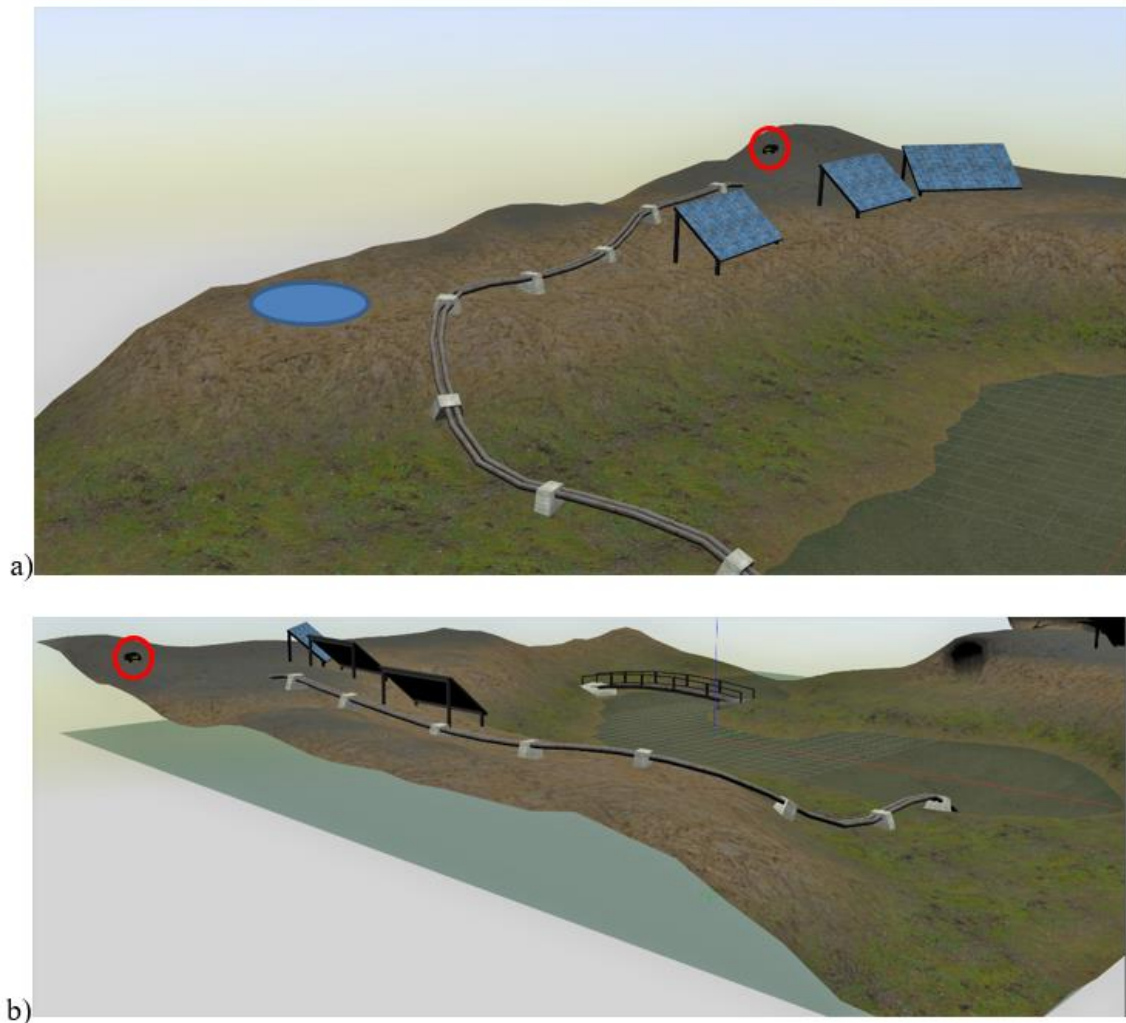
In the inspection world, were performed test in two different locations on the map. The first one, with a flattering start and slight slopes and an end with a great slope (location 3). In this location, twelve tests were performed, six with the use of the control and the rest without the use of it, being used, as in the agriculture world, at the same three different speeds, repeating each one twice. The second location (location 4) on the map was an area with average slopes during the entire route taken by the robot. The testing method was the

same as in the previous location, twelve tests were carried out under the same conditions. In both locations, the movement of the robot is, approximately, 40 meters for each test.

The test locations can be seen in figures 5.2 and 5.3, and the blue circle represents the zone where the robot should stop.



**Figure 5.2** - Approximately initial position of the robot in each test of location 3.



**Figure 5.3** - Approximately initial position of the robot in each test of location 4.



The following tables, 4 and 5 show the average initial position of the robot for each realized test at locations 3 and 4, respectively.

**Table 4** - Medium initial position of the robot for each realized test, in the inspection world, at location 3.

Location 3	Vx = 0.8053 [m/s]		Vx = 1.0718 [m/s]		Vx = 1.5692 [m/s]	
	Code	Teleop	Code	Teleop	Code	Teleop
<b>x initial</b>	15.9836	15.9740	15.9762	15.9803	15.9832	15.9728
<b>y initial</b>	-10.4949	-10.4594	-10.4630	-10.4619	-10.4630	-10.4158
<b>yaw initial</b>	-0.059591	0.008033	-0.072311	0.004628	-0.068076	0.005785

**Table 5** - Medium initial position of the robot for each realized test in the inspection world, at location 4.

Location 4	Vx = 0.8053 [m/s]		Vx = 1.0718 [m/s]		Vx = 1.5692 [m/s]	
	Code	Teleop	Code	Teleop	Code	Teleop
<b>x initial</b>	-23.0718	-23.0549	-23.0142	-23.0724	-23.1077	-23.0670
<b>y initial</b>	-27.4949	-27.5011	-27.5092	-27.4949	-27.5038	-27.5017
<b>yaw initial</b>	0.088506	0.014949	0.042036	0.029483	0.039488	0.030777

## 5.2. Results

### 5.2.1. Angular velocity

The obtained results in the tests to determine the angular velocity of the robot can be seen in figure 5.4, where the trajectory of the robot can be seen depending on the increase of angular velocity.

It can be noted that for an angular velocity corresponding to 10% of the linear velocity, the robot recoveries are slightly slow, which may induce large deviations from the desired trajectory, thus not fulfilling the necessary parameters. For angular velocities of 25% and 30% of the linear velocity, it is possible to verify that some of the recoveries induced new deviations, which causes a constant change in the trajectory of the robot and does not fulfill the necessary parameters. From 40% to 70% of the linear velocity, the recoveries are too fast, inducing new deviations and causing a constant change of direction, which does not meet the pre-defined parameters. The angular velocity selected to use in the control was 20% of the linear velocity, which can be seen that the recoveries are relatively fast, not causing circular trajectories, and are also smooth and without inducing new deviations.



Figure 5.4 - Robot's trajectory considering the variation of the angular velocity.

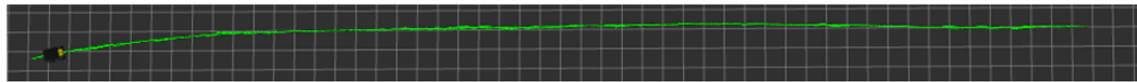
### 5.2.2. Yaw approximation

The obtained results in the approximation of the yaw values to the initial yaw can be seen in figure 5.5. Each test shows the trajectory made by the robot is possible to distinguish the most rectilinear ones.

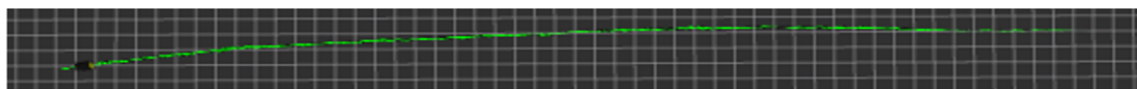
For each performed test, one decimal place was increased to the yaw value to approximate it to the initial yaw. The objective of these tests was to detect an approximation

that guarantees a straight line trajectory and that does not make an approximation too large so that the read value is constantly considered equal to the initial value.

Some good results were obtained that kept the robot's trajectory in a straight line, one of them was selected for use in the control, being, the letter "d" represented in figure 5.5, because it was enough to fulfill the requirements.



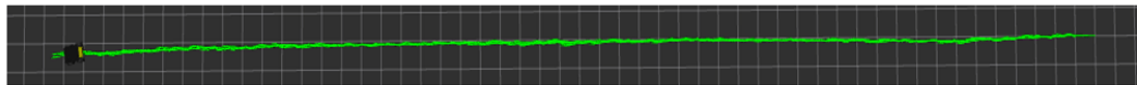
$$\text{a) } \text{yaw } \pm (1 \times 10^{-2}) = \text{yaw initial}$$



$$\text{b) } \text{yaw } \pm (1 \times 10^{-3}) = \text{yaw initial}$$



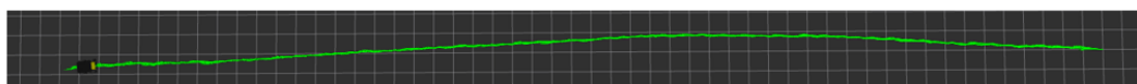
$$\text{c) } \text{yaw } \pm (1 \times 10^{-4}) = \text{yaw initial}$$



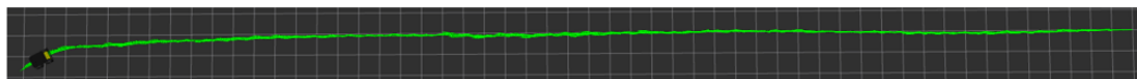
$$\text{d) } \text{yaw } \pm (1 \times 10^{-5}) = \text{yaw initial}$$



$$\text{e) } \text{yaw } \pm (1 \times 10^{-6}) = \text{yaw initial}$$



$$\text{f) } \text{yaw } \pm (1 \times 10^{-7}) = \text{yaw initial}$$



$$\text{g) } \text{yaw } \pm (1 \times 10^{-8}) = \text{yaw initial}$$



$$\text{h) } \text{yaw } \pm (1 \times 10^{-9}) = \text{yaw initial}$$

**Figure 5.5** - Robot's trajectory considering the approximation of the yaw values.

### 5.2.3. Empty world

The obtained results in the tests showed a great improvement regarding the use of the control while the robot was moving.

With the initial coordinates and yaw value, it was possible to determine the expected position of the robot at the end of its trajectory. After the robot's movement, the final coordinates of the robot on the map were read allowing the calculation of the robot's deviation. The expected and the real final coordinates can be seen in table 6. At each test, the robot did a trajectory of approximately 50 meters. At figure 5.6 is possible to see the average deviation of the robot for each test after the 50 meters trajectory for the use and non-use (Teleop) of the control. It allows to understand that the robot's deviation decreases with the increase of its linear velocity with the non-use of the control. It is also possible to notice that even with the maximum linear velocity of the robot, the final deviation, after a movement of 50 meters, is in the order of 4 meters. It can be compared with the movement performed by the robot with the use of the control, verifying that in this case, regardless of the robot's linear velocity, the final deviation is minimal.

**Table 6** – Average final robot position for each realized test, at location 1.

Location 1	Vx = 0.11 [m/s]		Vx = 0.15 [m/s]		Vx = 0.25 [m/s]	
	Code	Teleop	Code	Teleop	Code	Teleop
y final	-0.0106694	10.07538	0.047632	7.013503	0.1571106	3.650052
y expected	0.0114602	0.0114602	0.011663	0.022980	0.011527	0.011484

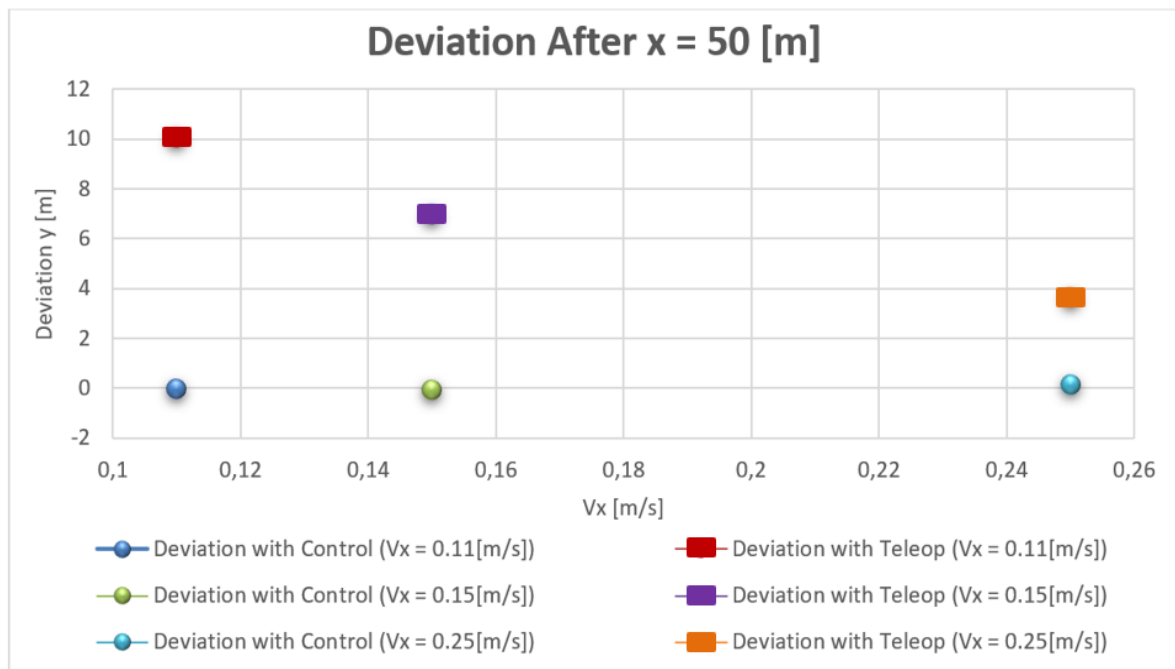


Figure 5.6 - Deviation of the robot after approximately 50 meters.

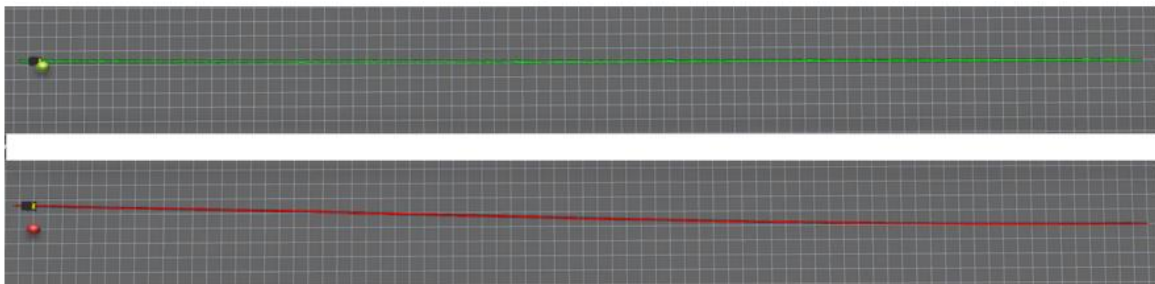
#### 5.2.4. Agriculture world

The average final position of each test is shown in table 7. It is possible to verify that the robots move approximately 77 meters in each test with the y-axis, with the smallest variation being read on the x-axis, allowing us to verify if the robot suffered deviation or if it managed to control it using the control. It is not possible to analyze the results in the same way as in the empty world since there are slopes, which do not allow a correct approximation of the expected final coordinates of the robot through the equation of the line. To circumvent this problem, Rviz was used, which allows tracing the trajectory of the robot, making it possible to verify the movement made by the robot and analyze whether or not the trajectory was in a straight line. The results obtained through Rviz for the 0.8053 [m/s] velocity test can be seen in figure 5.7, which shows the complete robot's trajectory, and figure 5.8, which shows in zoom the final zone of the trajectory and the final position of the robot, where, in both figures, the green and red dots correspond to the position where the robot should stop at the end of the trajectory, and the green line corresponds to the trajectory of the robot with the use of the control and the red one to the non-use of the control. The remaining results are presented in annex C. Through the analysis of the results, it is possible to verify that there is an improvement in the trajectory of the robot using the control. The improvement obtained

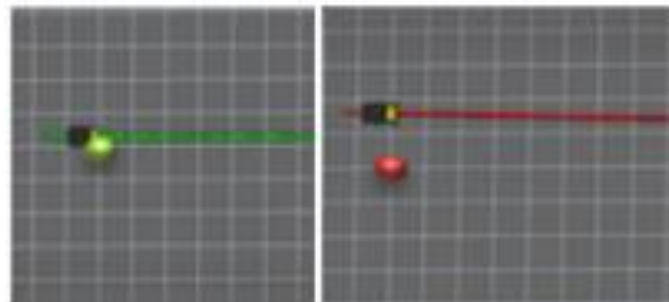
makes the movement of the robot more rectilinear and guarantees a final position much closer to the expected.

**Table 7** - Average final robot position for each realized test, at location 2.

Location 2	Vx = 0.8053 [m/s]		Vx = 1.0718 [m/s]		Vx = 1.5692 [m/s]	
	Code	Teleop	Code	Teleop	Code	Teleop
x final	58.2942	57.8994	58.2614	57.3758	56.7408	57.3790
y final	90.5890	90.1814	90.4875	90.3627	90.5192	90.1819



**Figure 5.7** - Deviation of the robot after the movement, at location 2.



**Figure 5.8** - Zoom of the final position of the robot, at location 2.

### 5.2.5. Inspection world

The obtained results for the two locations used to realize tests in the inspection world were analyzed in the same way as the results in the agriculture world results.

The final average position of the robot in each of the locations can be seen in tables 8 and 9, respectively the locations 3 and 4 on the map.

At location 3, it can be noted, in figure 5.9, that the robot maintains a straight line movement, but at some point, it suffers a deviation, that is not compensated immediately, due to the large final slope causing a slight deviation in its movement. Although the deviation is not fully compensated, it is possible to verify, in figure 5.11, that the final position of the

robot at the end of the approximately 40 meters of movement, is quite close to the expected position.

In location 4, the deviations suffered during the movement of the robot using the control are smaller compared to location 3, due to the smaller slopes of the path performed by the robot. The control worked during all the trajectories to keep the robot's movement in a straight line, as is possible to see in figure 5.10, and the robot's final position is close to the expected final position, as shown in figure 5.12.

The results obtained shown in figures 5.9, 5.10, 5.11, and 5.12 correspond to the average of the tests done with a linear velocity of 1.5692 [m/s]. Figures 5.11 and 5.12 show in zoom the final zone of the trajectory and the final position of the robot, where, in all figures, the green and red dots correspond to the position where the robot should stop at the end of the trajectory, and the green line corresponds to the trajectory of the robot with the use of the control and the red one to the non-use of the control. The remaining tests for each location can be seen in Annex D.

It is possible to notice that the use of the control increases the stability of the robot's trajectory, keeping it with a more rectilinear movement and avoiding large deviations from the pretended trajectory.

**Table 8** - Average final robot position for each realized test, at location 3.

Location 3	$V_x = 0.8053$ [m/s]		$V_x = 1.0718$ [m/s]		$V_x = 1.5692$ [m/s]	
	Code	Teleop	Code	Teleop	Code	Teleop
<b>x final</b>	-24.6379	-26.1948	-25.8931	-25.4987	-25.2574	-21.9918
<b>y final</b>	-6.8446	-10.2306	-6.8523	-10.2604	-6.8496	-12.3971

**Table 9** - Average final robot position for each realized test, at location 4.

Location 4	$V_x = 0.8053$ [m/s]		$V_x = 1.0718$ [m/s]		$V_x = 1.5692$ [m/s]	
	Code	Teleop	Code	Teleop	Code	Teleop
<b>x final</b>	16.0029	15.0731	16.7241	15.9400	16.5101	16.8976
<b>y final</b>	-23.3563	-28.1254	-25.2528	-27.8872	-25.3098	-28.0113

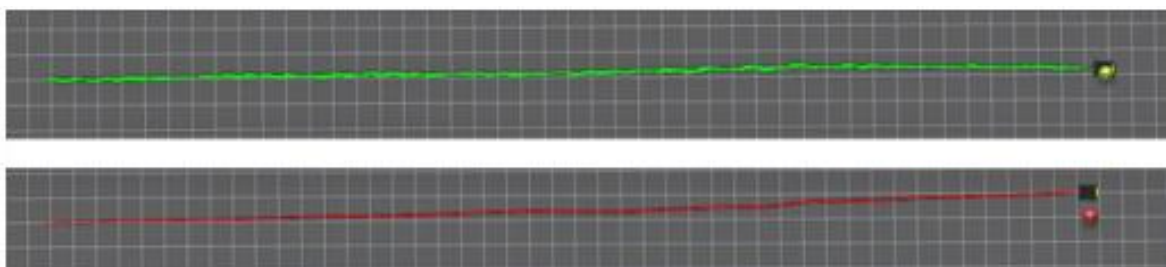


Figure 5.9 - Deviation of the robot after the movement, at location 3.

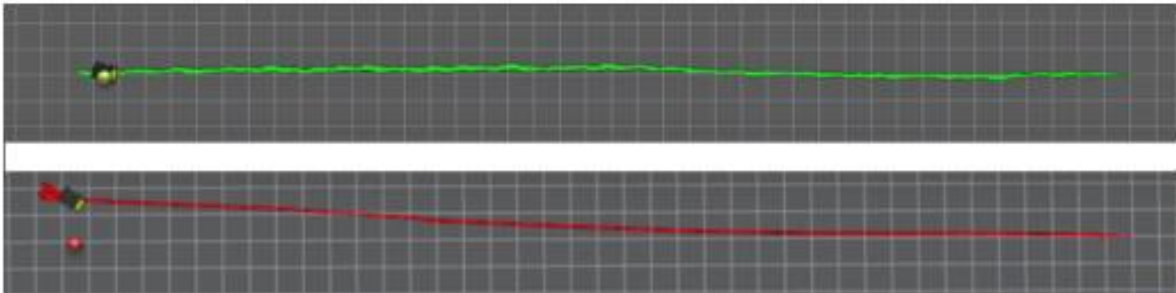


Figure 5.10 - Deviation of the robot after the movement, at location 4.

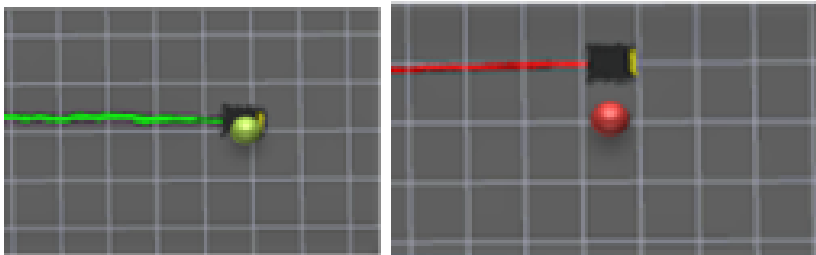


Figure 5.11 - Zoom of the final position of the robot, at location 3.

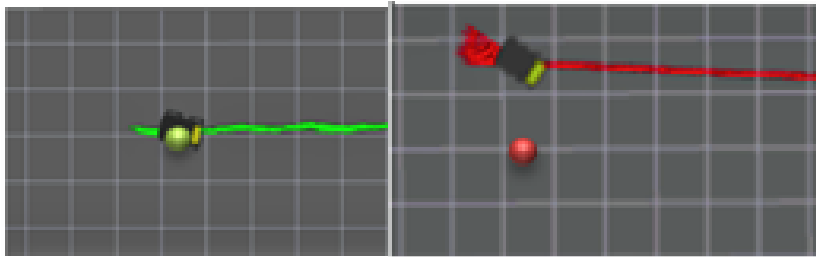


Figure 5.12 - Zoom of the final position of the robot, at location 4.



## 6. CONCLUSIONS

The main objective of this dissertation was the development of an algorithm for the traction control of tracked autonomous vehicles moving on rough terrain.

For the development of the algorithm, it was necessary to understand more about this type of vehicle and research studies by other authors to know and avoid problems in the development of the algorithm. It was also necessary to study the sensors used to understand their characteristics and the data that each one of them could provide. The initial approach started by thinking of a way to control the velocity of each caterpillar of the robot independently to vary the velocities of each one to keep the trajectory in a straight line, realizing later that, with the sensors available on the real robot it would not be possible to do. It was necessary to move on to a different approach which consisted of analyzing the IMU data.

The next process consisted of understanding the robot's orientation, for this it was necessary to study quaternions to realize that they could be converted into Euler angles, which were also later studied, allow a better understanding of the robot's orientation.

After converting the quaternions into Euler angles, it was then possible to start developing the control. The approach taken varied throughout its development, with several errors and inefficient approaches being detected, which had to be circumvented and carried out differently. Initially, the control started by just reading and modifying the Euler angles, more specifically the yaw of the robot, but it was possible to perceive that this modification alone could not be as effective as an approach that considers the yaw reading and the modification of the velocity of the angular velocity of the robot. This approach allowed us to develop robot control efficiently and to achieve the main objective of this dissertation.

During the development of this dissertation, some problems did not allow the testing of the control in the real robot (Green Climber LV 400 Pro) as initially planned, but tests were carried out in virtual environments that allowed to verify the operation of the control, with virtual robots that despite not being tracked, their operation mode is the same as tracked vehicles.

The development of this dissertation was quite challenging, mainly because the topics treated with the development of control, such as python programming and the use of ROS, are not studied in a mechanical engineering course, thus forcing the learning of these topics from the beginning. However, the theme of this dissertation is quite interesting and allowed

for learning new areas, allowing us to notice the development made in the industry and to realize that some areas of work can be complemented with the use of robots, thus allowing us to compensate for the lack of manpower.

In short, even not having tested the control in the real robot, the solution obtained seems to solve the problems of traction control. Considering all the tests carried out in a virtual environment, it can be concluded that the developed algorithm works, and some improvements can still be made, but it is already sufficiently capable.

## **6.1. Future Work**

In future work, new functions can be implemented in the robot algorithm, such as:

- Be able to carry out trajectories autonomously;
- Detection and recognition of objects, stopping or circumventing to resume the trajectory;
- Side slip detection;
- Changing the robot's speed depends on the type of terrain.

---

## 7. BIBLIOGRAPHY

- [1] “UM PLANETA EM CHAMAS PROPOSTA IBÉRICA DA WWF PARA A PREVENÇÃO DE INCÊNDIOS RURAIS,” 2020, Accessed: Apr. 26, 2022. [Online]. Available: [www.natureza-portugal.org](http://www.natureza-portugal.org)
- [2] “Thermite® | Howe & Howe Technologies.” <https://www.howeandhowe.com/civil/thermite> (accessed Apr. 06, 2022).
- [3] “Escavadora forestal para manobras de ataque indireto no combate de incêndios florestais | Dronster.” <https://www.vallfirest.com/pt/escavadora-forestal-incendios-florestais> (accessed Apr. 06, 2022).
- [4] “Remote control technology.” [https://www.mcconnel.com/remote-control-technology/\\_product/20/robocut2-rc40/](https://www.mcconnel.com/remote-control-technology/_product/20/robocut2-rc40/) (accessed Apr. 06, 2022).
- [5] P. Attrezzi Radiocomandato, “E-TRAIL.”
- [6] “REMOTE CONTROLLED SLOPE MOWER PRODUCT BROCHURE WANT MORE?” [Online]. Available: [www.greenclimber.com.au](http://www.greenclimber.com.au)
- [7] Y. Feng and J. Wang, “GPS RTK Performance Characteristics and Analysis,” 2008.
- [8] “O RTK e suas aplicações » GeoSensori.” <https://www.geosensori.com.br/2019/05/27/o-rtk-e-suas-aplicacoes/> (accessed Apr. 06, 2022).
- [9] R. Roriz, J. Cabral, and T. Gomes, “Automotive LiDAR Technology: A Survey,” *IEEE Transactions on Intelligent Transportation Systems*, 2021, doi: 10.1109/TITS.2021.3086804.
- [10] “Velodyne Lidar anuncia contrato de vendas triannual com a Baidu | Business Wire.” <https://www.businesswire.com/news/home/20201012005638/pt/> (accessed Apr. 06, 2022).
- [11] J. Sock, J. Kim, J. Min, and K. Kwak, “Probabilistic traversability map generation using 3D-LIDAR and camera,” in *Proceedings - IEEE International Conference on Robotics and Automation*, Jun. 2016, vol. 2016-June, pp. 5631–5637. doi: 10.1109/ICRA.2016.7487782.
- [12] B. Sebastian and P. Ben-Tzvi, “Active disturbance rejection control for handling slip in tracked vehicle locomotion,” *J Mech Robot*, vol. 11, no. 2, Apr. 2019, doi: 10.1115/1.4042347.
- [13] T. Zou, J. Angeles, and F. Hassani, “Dynamic Modelling and Trajectory Tracking Control of Unmanned Tracked Vehicles,” 2018.
- [14] V. N. Naumov, K. Y. Mashkov, and K. E. Byakov, “Autonomous tracked vehicles effectiveness estimation,” in *IOP Conference Series: Materials Science and Engineering*, Jun. 2019, vol. 534, no. 1. doi: 10.1088/1757-899X/534/1/012006.
- [15] B. Sebastian and P. Ben-Tzvi, “Physics-Based Path Planning for Autonomous Tracked Vehicle in Challenging Terrain,” *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 95, no. 2, pp. 511–526, Aug. 2019, doi: 10.1007/s10846-018-0851-3.
- [16] “Desktop for developers | Ubuntu.” <https://ubuntu.com/desktop/developers> (accessed Aug. 24, 2022).
- [17] “What is ROS? | Ubuntu.” <https://ubuntu.com/robotics/what-is-ros> (accessed Aug. 24, 2022).

- [18] “ROS Gazebo: Everything You Need To Know - Robotic Simulation Services.” <https://roboticsimulationservices.com/ros-gazebo-everything-you-need-to-know/> (accessed Aug. 25, 2022).
- [19] “Get Started | Swiftnav.” <https://www.swiftnav.com/get-started> (accessed Aug. 27, 2022).
- [20] “BENEFITS • Designed for Autonomy • Skylark”, Accessed: Aug. 30, 2022. [Online]. Available: [www.swiftnav.com](http://www.swiftnav.com)
- [21] “Starling by Swift is a Receiver-Agnostic Precise Positioning Engine.” <https://www.swiftnav.com/starling-positioning-engine> (accessed Aug. 30, 2022).
- [22] “ABSOLUTE POSITION, VELOCITY AND TIME”, Accessed: Aug. 30, 2022. [Online]. Available: [www.swiftnav.com](http://www.swiftnav.com)
- [23] “Duro Inertial Ruggedized GNSS Receiver Ideal for Outdoor Deployments.” <https://www.swiftnav.com/duro-inertial> (accessed Aug. 30, 2022).
- [24] “Duro Inertial User Manual,” 2022, Accessed: Aug. 30, 2022. [Online]. Available: <https://www.swiftnav.com/latest/duro-user-manual>
- [25] “Swift Console 4.0 User Manual,” 2022.
- [26] “CONTINUOUS AND ROBUST INERTIAL NAVIGATION SYSTEM (INS) POSITIONING”, Accessed: Aug. 30, 2022. [Online]. Available: [www.swiftnav.com](http://www.swiftnav.com)
- [27] P. C. Robotics, “Vision and Control 123 FUNDAMENTAL ALGORITHMS IN MATLAB®.” [Online]. Available: [www.petercorke.com/RVC](http://www.petercorke.com/RVC)
- [28] “Visualizing quaternions (4d numbers) with stereographic projection - YouTube.” <https://www.youtube.com/watch?v=d4EgbgTm0Bg> (accessed Sep. 01, 2022).
- [29] “How to mount Razor IMU on Clearpath Husky Robot - ROS Answers: Open Source Q&A Forum.” <https://answers.ros.org/question/256370/how-to-mount-razor-imu-on-clearpath-husky-robot/> (accessed Sep. 01, 2022).

## ANNEX A

Python code to convert quaternions into Euler angles.

```
1  #!/usr/bin/env python3
2
3  import rospy
4  from sensor_msgs.msg import Imu
5  from geometry_msgs.msg import Twist
6  from tf.transformations import euler_from_quaternion
7  from std_msgs.msg import Float64
8
9  yaw = 0.0
10 vel = 0.0
11 i = 1
12 yaw_initial = 0.0
13 yaw_pub = 0.0
14 yaw_pub_ini = 0.0
15
16 def get_vel(msg):
17     global vel
18
19     vel = msg.linear.x
20
21 def get_rotation(msg):
22     global roll
23     global pitch
24     global yaw
25     global yaw_initial
26     global i
27     global yaw_pub
28     global yaw_pub_ini
29
30     orientation_q = msg.orientation
31
32     orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z, orientation_q.w] # Define list of the quaternions
33
34     (roll, pitch, yaw) = euler_from_quaternion (orientation_list) #Conversion of quaternions to euler using python functions
35     if i == 1:
36         yaw_initial = yaw
37         yaw_pub = rospy.Publisher("/yaw_initial", Float64, queue_size=10)
38         yaw_pub_ini = Float64()
39         yaw_pub_ini.data = yaw_initial
40         i = 0
41
42     if yaw < yaw_initial + 0.00001 and yaw > yaw_initial -0.00001: #Rounding small values of Yaw to 0
43         yaw = yaw_initial
44
45     if vel == 0.0: #Keep the yaw value constant with the robot stopped
46         yaw = yaw_initial
47         print("Robot Stopped")
48
49     print ([roll,pitch ,yaw])
50
51 sub = rospy.Subscriber("/imu/data", Imu, get_rotation)
52
```

```
53 def listener():
54
55     rospy.init_node("Listener", anonymous=True)
56     rospy.Subscriber("/cmd_vel", Twist, get_vel)
57     rospy.Subscriber("/imu/data", Imu, get_rotation)
58     pub = rospy.Publisher("/euler_angles", Imu, queue_size=100) #Publish Roll, Pitch and Yaw values
59     data_euler = Imu()
60
61     rate = rospy.Rate(10)
62
63     while not rospy.is_shutdown():
64         data_euler.angular_velocity.x = roll #Define where to publish Roll
65         data_euler.angular_velocity.y = pitch #Define where to publish Pitch
66         data_euler.angular_velocity.z = yaw #Define where to publish Yaw
67
68         pub.publish(data_euler)
69         yaw_pub.publish(yaw_pub_ini)
70
71         rate.sleep()
72
73 if __name__ == '__main__':
74     try:
75         listener()
76     except rospy.ROSInterruptException:
77         pass
78
```

## ANNEX B

Python code of the control.

```
1  #!/usr/bin/env python3
2
3  import rospy
4  import math
5  from sensor_msgs.msg import Imu
6  from geometry_msgs.msg import Twist
7  from std_msgs.msg import Float64
8
9  yaw = 0.0
10 yaw_initial = 0.0
11
12 def callback_EULER(msg):                                #Get Euler Angles
13     global roll
14     global pitch
15     global yaw
16
17     roll = msg.angular_velocity.x
18     pitch = msg.angular_velocity.y
19     yaw = msg.angular_velocity.z
20
21 def callback_IMU(msg):
22     global imu_data_var_x
23     global imu_data_var_y
24     global imu_data_var_z
25
26     imu_data_var_x = msg.linear_acceleration.x
27     imu_data_var_y = msg.linear_acceleration.y
28     imu_data_var_z = msg.linear_acceleration.z
29
30
31 def callback_Yaw_Initial(msg):
32     global yaw_initial
33
34     yaw_initial = msg.data
35
36 def control2():
37
38     rospy.init_node('control2', anonymous=True)
39
40     rospy.Subscriber("/euler_angles", Imu, callback_EULER)
41     rospy.Subscriber("/imu_2", Imu, callback_IMU)
42     rospy.Subscriber("/yaw_initial", Float64, callback_Yaw_Initial)
43
44     velocity_publisher = rospy.Publisher("/cmd_vel", Twist, queue_size=10)
45
46     velocity = Twist()
```

```

47 dist_recover = 0.0
48 velocity.angular.z = 0.0
49 speed = float(input("Input your speed: ")) #The user inputs the pretended speed of the robot
50
51
52 while not rospy.is_shutdown():
53
54     dist_deviation = 0.0
55     dist_recover = 0.0
56     velocity_publisher.publish(velocity)
57
58     if yaw == yaw_initial:           # Keep the robot in straight line
59
60         velocity.linear.x = speed
61         velocity.angular.z = 0.0
62         print("Robot moving in straight line")
63
64     elif yaw > yaw_initial or yaw < yaw_initial and velocity.linear.x > 0.0: #Detection of deviation
65
66         time_deviation = rospy.Time.now().to_nsec()           #Calculating time of deviation
67
68         yaw_g = math.cos(yaw * (180 / math.pi))               #Converting yaw into degrees
69
70         dist_deviation = float(yaw_g) * velocity.linear.x * (time_deviation * 0.000000001) #Calculate the deviation
71
72         rospy.loginfo('dist_detour %f', dist_deviation)
73
74     while((abs(dist_deviation) - abs(dist_recover)) > 0.1 or (abs(dist_deviation) - abs(dist_recover)) < -0.1 ): #Keep recovering deviation
75
76         if yaw < yaw_initial:
77
78             rospy.loginfo('dist_recover %f', dist_recover)
79
80             time_recover = rospy.Time.now().to_nsec() #Calculating time of recovery
81
82             yaw_recover = yaw * (-1)

```

```

83
84     rospy.loginfo('yaw_rec %0.8f', yaw_recover)
85
86     yaw_g_recover = math.cos(yaw_recover * (180 / math.pi)) #Converting yaw into degrees
87
88     dist_recover = float(yaw_g_recover) * velocity.linear.x * ((time_recover - time_deviation) * 0.000000001) #Calculating recovery
89
90     if speed > 0.0:
91         velocity.angular.z = velocity.linear.x * 0.60 #Give angular velocity to the robot
92
93     else:
94         velocity.angular.z = velocity.linear.x * (-0.60)
95
96     elif yaw > yaw_initial:
97
98         rospy.loginfo('dist_recover %f', dist_recover)
99
100        time_recover = rospy.Time.now().to_nsec() #Calculating time of recovery
101
102        yaw_recover = yaw * (-1)
103
104        rospy.loginfo('yaw_rec %0.8f', yaw_recover)
105
106        yaw_g_recover = math.cos(yaw_recover * (180 / math.pi)) #Converting yaw into degrees
107
108        dist_recover = float(yaw_g_recover) * velocity.linear.x * ((time_recover - time_deviation) * 0.000000001) #Calculating recovery
109
110        if speed > 0.0:
111            velocity.angular.z = velocity.linear.x * (-0.60) #Give angular velocity to the robot
112
113        else:
114            velocity.angular.z = velocity.linear.x * 0.60
115
116    elif ((abs(dist_deviation) - abs(dist_recover)) < 0.1 or (abs(dist_deviation) - abs(dist_recover)) > -0.1) :
117
118        dist_deviation = 0.0 #Assuming the same initial parameters
119        dist_recover = 0.0
120        velocity.angular.z = 0.0
121        print("Robot direito")
122        break
123
124
125    velocity_publisher.publish(velocity) #Publish velocity of the robot
126

```



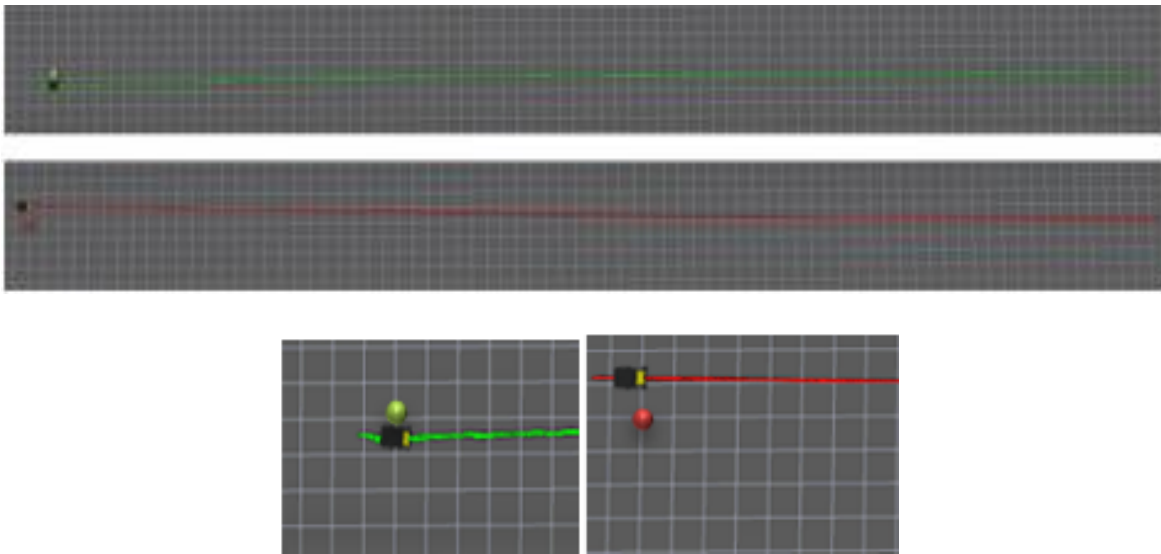
```
127
128
129 if __name__ == '__main__':
130     try:
131         control2()
132     except rospy.ROSInterruptException:
133         pass
134
```



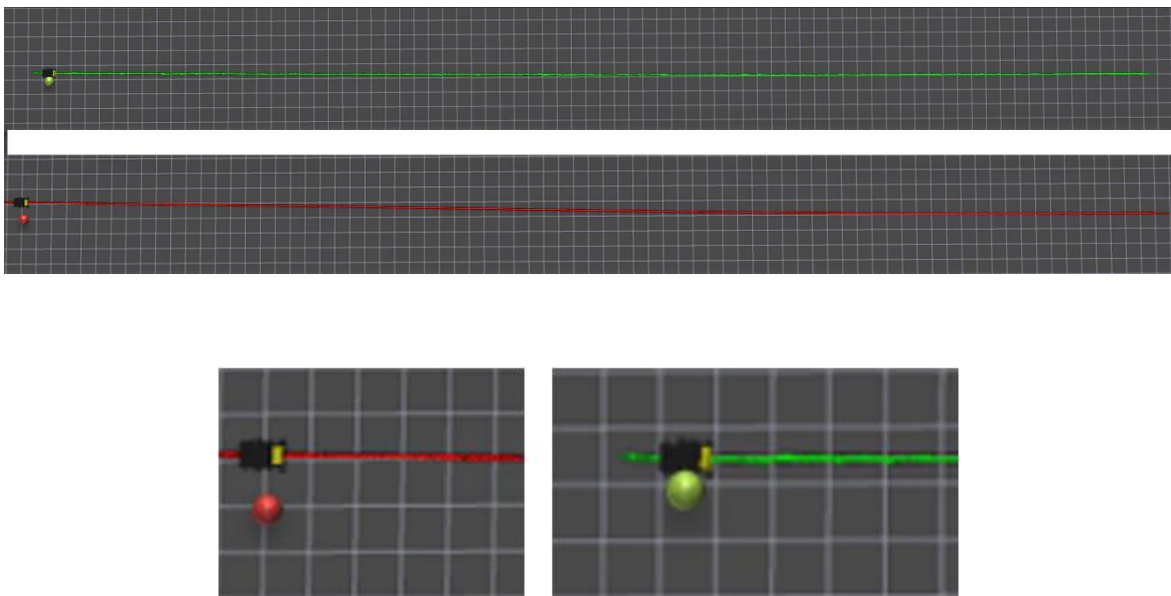
## ANNEX C

Deviation of the robot at location 2 with the velocity of 1.5692 [m/s] and 1.0718 [m/s], where the red dot and red line correspond to the robot's final expected position and trajectory without the control, respectively. And the green dot and green line correspond to the robot's final expected position and trajectory without the control, respectively. The zoom of the final position of each case is also shown.

- 1.5692 [m/s]



- 1.0718 [m/s]

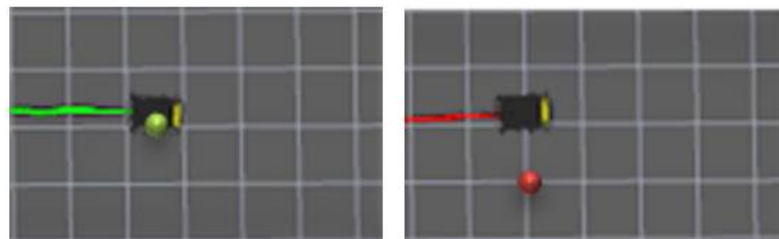
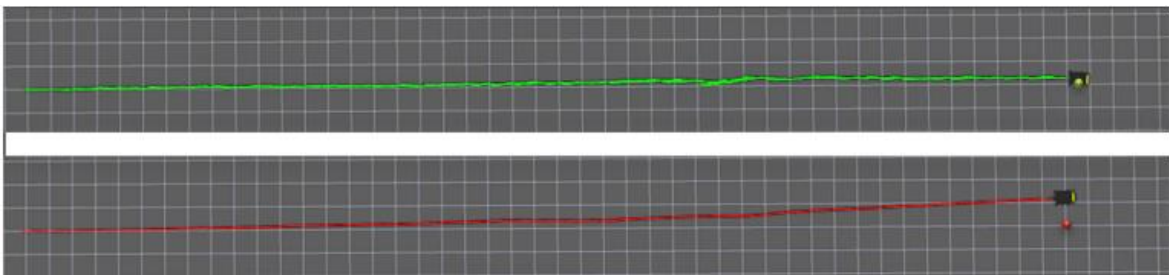




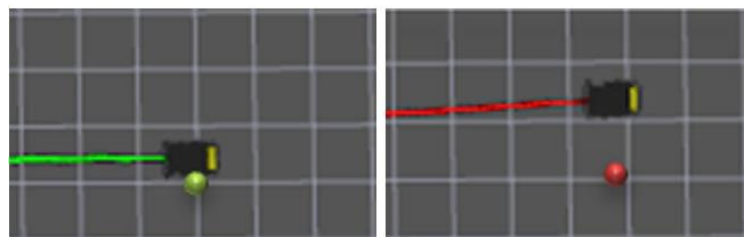
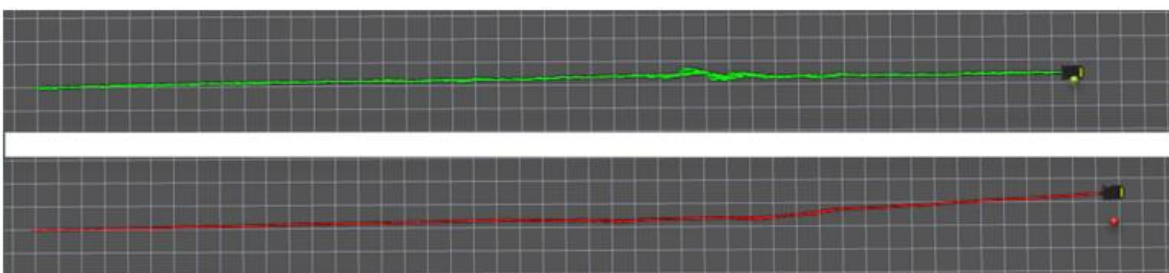
## ANNEX D

Deviation of the robot at location 3 with the velocity of 1.0718 [m/s] and 0.8053 [m/s], where the red dot and red line correspond to the robot's final expected position and trajectory without the control, respectively. And the green dot and green line correspond to the robot's final expected position and trajectory without the control, respectively. The zoom of the final position of each case is also shown.

- 1.0718 [m/s]



- 0.8053 [m/s]



Deviation of the robot at location 4 with the velocity of 1.0718 [m/s] where the red dot and red line correspond to the robot's final expected position and trajectory without the control, respectively. And the green dot and green line correspond to the robot's final expected position and trajectory without the control, respectively. The zoom of the final position of each case is also shown.

- 1.0718 [m/s]

