Simultaneous Localisation and Mapping with a Tracked Mobile Robot

Francisco Vieira Pinheiro Gonçalves

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisors: Prof. Alexandra Bento Moutinho
Prof. Mário António da Silva Neves Ramalho

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Supervisor: Prof. Alexandra Bento Moutinho
Member of the Committee: Prof. Rodrigo Martins de Matos Ventura

November 2022

Dedicated to my special people...

Acknowledgments

I express my gratitude to all involved in the development of this work. Mainly Prof.Alexandra Moutinho for her guidance during the key moments of the work and trust put in me; Prof.Mário Ramalho for his useful insights throughout the development of the work and his constant willingness to help; Mr.Luís Raposeiro for his seemingly never-ending patience and help with my frequent hardware problems and Eng.Camilo Christo for his practical advice and help with my electrical problems.

I also extend my gratitude to Bruno Cândido for his company at the Laboratory and his relatable problems during his work that frequently helped me improve my own work. A great thanks to the 2021/2022 FST Lisboa team for showing me what dedicated people working together can achieve, especially my Autonomous Systems department, the most dedicated and talented people I have met so far. You people showed me what real hunger for improvement is.

Thank you to my sister Leonor Gonçalves for her help with taking pictures of the robot and to Inês Cardoso for her help with recording videos of the robot.

A special thanks to my family for supporting me throughout this work.

Finally, a big thank you to my dear friends Afonso Gonçalves, Basílio Geada, Dinis Forte, Dinis Romão, Guilherme Dourado, Inês Sabino, Joana Ramos, Mariana Martins and Marta Marques. Without you all I would not have found the strength to do the things I was able to do.

Resumo

Este trabalho foca-se no desenvolvimento de um robô móvel de lagartas capaz de localização e mapeamento simultâneos (*SLAM*).

O modelo cinemático do robô foi aproximado a um de atuação diferencial equivalente e o modelo de odometria calibrado experimentalmente. Para uma estimativa de estado robusta antes do algoritmo de *SLAM*, fundem-se os dados da odometria e da unidade de medida inercial (*IMU*) num Filtro de Kalman Estendido (*EKF*). O desempenho da odometria calibrada e estimativa de estado do *EKF* são comparados com dados de um sistema de captura de movimento em trajectórias simples e complexas. A estimativa do *EKF* constitui uma melhoria considerável da estimativa da odometria.

A parte *SLAM* do trabalho concentra-se em dois algoritmos *open-source* baseados em grafos, um de mapeamento 2D e outro de mapeamento 3D. O desempenho de ambos os algoritmos de *SLAM* é analisado comparando a sua estimativa de mapa e trajectória com dados da captura de movimento num mapa personalizado. Ambos os algoritmos testados mostraram estimativas de localização melhoradas em comparação com a estimativa do *EKF* ao realizar trajectórias complexas. O mapa estimado, embora congruente com a geometria do mapa real, apresenta ruído significativo. O algoritmo de mapeamento 3D apresenta estimativas de mapa ruidosas devido a limitações de hardware.

Palavras-chave: Localização e Mapeamento Simultâneos, Odometria, Locomoção por Lagartas, Filtro de Kalman Estendido, Robótica

Abstract

This work focuses on the development of a tracked mobile robot capable of performing Simultaneous Localisation and Mapping (SLAM).

The robot's kinematics model was approximated by a differential-drive one, and its odometry estimate was experimentally calibrated. There is also focus on state estimation before SLAM by fusing odometry and inertial measurement unit (IMU) data with an Extended Kalman Filter (EKF). Both the calibrated odometry estimate and the EKF pose estimate are compared to motion-captured data in simple and complex trajectories. The EKF estimate is a considerable improvement on the odometry estimate.

The SLAM part of the work focuses on two existing open-source graph-based algorithms, one for 2D mapping and the other for 3D mapping. The performance of both SLAM algorithms is reviewed by comparing their map and trajectory estimates with motion-captured data in a custom-made map. Both algorithms tested show more accurate localisation estimates compared to the EKF pose estimate when performing complex trajectories. The map obtained, although congruent with the real map geometry, showed significant noise, especially the 3D mapping algorithm, due to hardware limitations.

Keywords: Simultaneous Localisation and Mapping, Odometry, Tracked-Drive, Extended Kalman Filter, Robotics

Contents

List of Tables

List of Figures

Nomenclature

Greek symbols

Δ	Discrete variable’s change from previous time step
$\dot{\theta}$	Motor output shaft rotational speed
ω	Wheel rotational speed
φ	Yaw
ξ	Robot Pose

Roman symbols

(k)	Discrete variable at current time step
(t)	Time dependent variable
B	Viscous friction factor
b	Wheelbase
E_b	Ratio between actual wheelbase and nominal wheelbase
E_d	Ratio between left and right wheel radius
E_{ccw}	Average ratio between actual counter-clockwise turn value and estimated one
E_{cw}	Average ratio between actual clockwise turn value and estimated one
E_{max}	Maximum distance from origin a cluster center is in UMBmark method
E_s	Ratio between actual distance travelled and estimated distance travelled
J	Motor moment of inertia
K	Current-developed torque conversion factor
L	Inductance
N	Encoder ticks per revolution
n	Encoder ticks

Q	Process noise covariance matrix
R	Resistor
r	Wheel radius
R_{imu}	IMU sensor measurement covariance matrix
R_{odom}	Odometry estimate covariance matrix
s	Distance travelled
T	Static friction factor
t	Time
u	Digital command
v	Linear velocity
V_{in}	Motor supply voltage
V_{max}	Maximum possible supply voltage, battery voltage level
x, y	2D Pose Cartesian components

Subscripts

g	In global-frame
i, j	Computational indexes
L	Left-wheel related
m	In moving-frame
R	Right-wheel related
t	Current time step

Superscripts

$\cdot$	Time Derivative
c	Command variable
T	Transpose

Glossary

2D - Two-dimensional

3D - Three-dimensional

AHRS - Attitude and Heading Reference System

DC - Direct Current

DoF - Degrees of Freedom

EKF - Extended Kalman Filter

IMU - Inertial Measurement Unit

KF - Kalman Filter

LiDAR - Laser Detection and Ranging

RGB-D - Red Green Blue - Depth

ROS - Robotic Operating System

RTAB-Map - Real-Time Appearance Based Mapping

SLAM - Simultaneous Localisation and Mapping

Chapter 1

Introduction

The term *robot* - derived from *robota* - first appeared in a 1920 Czech play by Karel Čapek called ”Rossum’s Universal Robots” (R.U.R.). In Czech the term translates directly to subordinate labour or, more colloquially, to hard work. The term *robotics* was coined in the 1940s by Isaac Asimov’s science fiction short-stories. Asimov also published the popular science-fiction series dealing with the ethics of human-robot interaction from which the famous ”three fundamental laws of robotics” originated. Throughout the decades, the meaning of these terms evolved along with the scientific development of the area. A more modern definition of a robot can be found in [1]: ”A robot is a complex mechatronic system enabled with electronics, sensors, actuators, and software, performing tasks with a certain degree of autonomy. It may be pre-programmed, teleoperated, or performing computations to make decisions”. Robotics is modernly defined as the science that studies the intelligent connection between perception and action [2].

Most of the profitability of the robotics area comes from industrial manufacturing. Equipped with end-effectors, robot arms can move with superhuman speed and precision to perform repetitive tasks [3]. However, these robots tend to be fixed to their environment. If the robot needs to move through its real-world environment to perform a task, then it must be *mobile*. This means that the system should be able to move using its own locomotion apparatus. Normally, the more hostile the environment, the more complex the apparatus must be. In dangerous or inhospitable environments, the robot must allow the human operator to be distant while being commanded. In these cases, the low-level complexity of the robot makes it nearly impossible for the operator to control its motion directly. The human can perform cognition activities such as path planning and action triggering, but relies on the robot’s localisation estimate and control scheme to provide at least low-level motion control.

The highest complexity tool for localisation estimation in robotics is SLAM (Simultaneous Localisation and Mapping). The SLAM problem asks whether it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a map of this environment while simultaneously determining its location within this map [4].

Mobile robotics and, more specifically, simultaneous localisation and mapping capable mobile robotics is a dynamic and ever-growing area of research steadily gaining popularity since the beginning of the millennium. The SLAM problem has been the subject of extensive technical and theoretical research. In its infancy, only the best hardware met the requirements for running SLAM algorithms, leaving SLAM applications limited to top-level academia. However, as time, industry, and technology progressed, the cost of components dropped. This enables SLAM algorithms to run on inexpensive hardware, which facilitates the research and development of SLAM applications in various fields. Furthermore, several publicly available data-sets[1] and a myriad of open source algorithms[2] contribute to the accessibility of the applicability of SLAM on any platform. There are also plug-and-play solutions in ROS/ROS2 package format[3] that are widely used in industry with promising results [5].

SLAM capabilities are one of the final steps before full robot autonomy. A SLAM capable robot can be truly exploratory of its environment without much human intervention. This enables a wide range of applications, from fully autonomous home vacuuming [6] to disaster site transversal and recognition [7] and even planetary exploration [8]. It is this freedom in applicability and the vastness of resources that makes robotics and SLAM an exciting area to work in.

1.1 Mobile Robotics Historical Overview

Robotics had its first jump-start due to World War II. The ever-increasing demand for better technology and the large investment in scientific research led to breakthroughs that made the area blossom. The first autonomous mobile robot came three years after the war. William Grey Walter built two autonomous robots named Elmer and Eslie [9]. Equipped with a rotating photocell and a contact sensor, they were capable of following a light source and detecting obstacles.

In the early 1960s *Beast* was created at John Hopkins University [10]. It wandered white hallways, self-centering by sonar, until it detected low battery power. Then it would identify the electrical outlets on the wall and then plug in to recharge its battery cells when a low battery was detected. Near the end of the decade and into the beginning of the 1970s, the Stanford Research Institute developed *Shakey*, a landmark robot known as the first mobile robot with the ability to perceive and reason about its surroundings. It was equipped with a camera, sonar range finders and collision detection sensors. Some of the notable results from the development of *Shakey* [11] were the development of the A* (pronounced "A-star") search algorithm, the Hough Transform deduction, and visibility graphs - all of which are still widely used today. At the same time, the end of the 1950s and into the 1960s was when robotics became extraterrestrial [12]. The Soviets successfully launched Sputnik 1, the first artificial Earth satellite, and two lunar probes, Luna 1 and Luna 2, before the halfway point of the decade. Later Ranger 9, Luna 9 and Surveyor 1 were all able to land safely and send back images of the Moon's surface. By the end of the decade, the US Lunar Orbiter series conducted five mapping missions that aided the Apollo missions in safely landing humans on the Moon from 1969 to 1972.

During the 1970s the Stanford Cart [13] and its various forms were developed. It was originally designed to test what it would be like to control a lunar rover from Earth, and it was eventually reconfig-

[1]https://sites.google.com/view/awesome-slam-datasets/
[2]https://openslam-org.github.io/
[3]https://index.ros.org/search/?term=slam

ured as an autonomous vehicle. In its final form, the Stanford Cart was able to navigate autonomously through an environment riddled with obstacles, although it did so very slowly. The cart was equipped with 3D vision capabilities. It was also reconfigured to stop after each metre of movement and take several minutes to reassess its surroundings and reevaluate its decided path.

From the mid 1980s to the early 1990s, the first autonomous car VaMoRs was developed by the team of Ernst Dieter Dickmans [14]. VaMoRs performed both road and object recognition and was able to drive on urban streets without traffic. This was also the beginning of affordable commercially available mobile robots, such as Pioneer[4], which was used mainly for education and research. Before the end of the millennium, NASA sent its first rover to Mars, Sojourner, onboard its Mars Pathfinder [15]. The Sojourner was equipped with three cameras, commanded from Earth, and was able to autonomously drive along predefined paths and avoid hazardous scenarios on the way.

The beginning of the new millennium brought the first model of the Roomba[5]. NASA landed two more rovers on Mars, Spirit and Opportunity [15]. The 2000s were also when the DARPA Grand Challenge had its origin [16]. In this event, several autonomous vehicle teams competed in the Mojave desert to determine who could finish the challenge. In its first edition, none of the cars were able to complete the route. In its second edition [17], Stanley from Stanford University was able to complete the route first. Of the 23 competing vehicles, only four were able to complete the challenge. The DARPA Urban Grand Challenge [18] was held in 2007, where six autonomous vehicles attempted to complete a course in an urban area that respected all traffic regulations, including autonomous merging into traffic. The original version of the Robot Operating System (ROS 0.4) was released in 2009.

The eventful nature of previous decades did not stop there. In the last decade, NASA sent the Curiosity rover to Mars. More recently, in 2020, NASA also sent the Perseverance rover to Mars to collect knowledge and demonstrate technologies that address the challenges of future human expeditions to Mars. The rover comes equipped with nineteen cameras and two microphones to record the martian atmosphere. Sent together with Perseverance was the Ingenuity helicopter, a robotic helicopter that demonstrated the technology for rotorcraft flight in the extremely thin atmosphere of Mars and has successfully flown several times to date [15].

The last decade also saw a significant increase in commercially available mobile robots such as Boston Dynamic's Spot[6]. A quadruped robot that navigates terrain with unprecedented mobility and robustness. It can carry loads up to 14kg, is equipped with several sensors for data acquisition, and is fully user-programmable. Spot has been deployed in disaster site recognition, industrial safety, and maintenance checks. Recently, NASA's Jet Propulsion Laboratory announced a Spot-based prototype[7] called "NeBula Spot" [19] for potential Mars and/or Moon missions.

Other recent notable robots are, for example, Boston Dynamic's Atlas[8], a humanoid robot defying the limits of how human-like a robot can move. Performing acrobatic stunts such as backflips, jumping from platform to platform, jumping over/around obstacles, and performing technical dance moves. Aquanaut

[4]`https://robots.ieee.org/robots/pioneer/`
[5]`https://robots.ieee.org/robots/roomba/`
[6]`https://robots.ieee.org/robots/spotmini/`
[7]`https://www.jpl.nasa.gov/robotics-at-jpl/nebula-spot`
[8]`https://robots.ieee.org/robots/atlas2016/`

is another notable robot[9], an unmanned underwater vehicle that can transform itself from a submarine designed for long-distance underwater travel to a humanoid robot with two arms for manipulation tasks. It has been used in inspection of subsea oil and gas infrastructure, underwater welding, and underwater valve operation. Another popular robot from the last decade is Sony's Aibo[10], a companion robot dog capable of performing tricks, adapting/learning with the user/owner's habits, and simulating a regular dog's emotions.

1.2 Simultaneous Localisation and Mapping Overview

The inception of the SLAM problem can be traced back to the 1986 IEEE Robotics and Automation Conference held in San Francisco, California. At the time, several robotics and Artificial Intelligence researchers were looking at applying estimation-theoretic methods to mapping and localisation problems. This conference made it clear that consistent probabilistic mapping is an important conceptual and computational problem of robotics that needed to be further investigated.

Over the next several years, several landmark papers on the topic were produced. Smith and Cheesman [20] and Durrant-Whyte [21] established that there must be a high correlation between estimates of the location of landmarks on a map to achieve successful map building. Smith, Self, and Cheeseman [22] showed that as a mobile robot moves through an unknown environment, taking relative observations of landmarks, the estimates of these landmarks are necessarily correlated with each other due to the common error in the estimated location of the vehicle [4]. This implied that a consistent full solution to the combined localisation and mapping problem would require a joint state composed of the vehicle pose and every landmark position to be updated following each landmark observation. In turn, this would require the estimator to employ a state vector of the order of the number of landmarks maintained on the map and with computation scaling as the square of the number of landmarks.

Leonard and Durrant-Whyte [23] developed a probabilistic SLAM approach based on an Extended Kalman Filter, a technique still used in modern SLAM algorithms. Most of the research at this time struggled with the computational cost of the SLAM problem and with loop closure, that is, recognising that the current scene or landmarks have already been processed and using that information to the benefit of the localisation and map building. These struggles led to a significant influx of research on the computational cost problem and the loop-closure efficiency problem at the beginning of the millennium.

Kalman Filter-based SLAM solutions have been the most popular ever since the SLAM problem became widely known; however, these approaches come with downsides that can make them invalid in certain applications. For one, since every received observation influences all parameters within the Gaussian, in feature-rich environments, the update step will have a significant computational cost. This also means that Kalman Filter-based approaches cannot be used in lifelong mapping applications where the environment is ever increasing and changing. Another significant disadvantage is the fact that Kalman Filters assume a Gaussian distribution. This means that, for example, these solutions are unable to

[9] `https://robots.ieee.org/robots/aquanaut/`
[10] `https://robots.ieee.org/robots/aibo2018/`

process negative information: not seeing a feature when one expects to see it. This carries relevant information that can trigger updates/corrections in localisation and/or mapping. The problem with negative information is that it induces non-Gaussian beliefs, which cannot be represented by mean and variance, and for this reason, Kalman Filter implementations simply ignore the issue of negative information and instead integrate only information from observed features [24]. These disadvantages are only noticeable in very specific conditions, which still make Kalman Filter-based solutions viable and applicable to most environments.

The information filter or the inverse covariance form of the Kalman Filter has been shown to improve the base Kalman Filter solutions in certain scenarios [25]; however, this inverse covariance form requires the recovery of the state estimate at each time step, which implies computationally costly matrix inversion. The same as the Kalman Filter, the Information Filter can be applied to nonlinear systems by performing a linearisation of the system at each time step. The nonlinear form of the Information Filter is regarded as the Extended Information Filter (EIF), and although some successful research was done on the topic, the costly matrix inversion required stopped it from being as widely applied as the Extended Kalman Filter. In 2004, the Sparse Extended Information Filter (SEIF) algorithm was introduced by Thrun et al. [26, 27] to directly combat the computational weight of EIF operations with a matrix sparsification step. Despite being computationally efficient, the quality of the results compared to the more popular EKF did not justify the added theoretical complexity.

Other popular SLAM algorithms are FastSLAM 1.0 and FastSLAM 2.0 presented in [28], [29] and [30]. These algorithms integrate particle filters and extended Kalman filters to exploit the fact that if the robot knew its true path, the errors in its feature estimates would be independent of each other. The advantage of FastSLAM-type algorithms over Kalman Filter-based ones is twofold: the fact that data association decisions can be made on a per-particle basis, similar to multihypothesis tracking algorithms, and as such the filter maintains posteriors over multiple data associations, not just the most likely one which makes FastSLAM significantly more robust to data association problems than algorithms based on maximum likelihood data association; the fact that particle filters can cope with non-linear robot motion models, whereas EKF-style solutions need to approximate such models with linear functions. FastSLAM 2.0 is an improvement on the robustness of FastSLAM 1.0 especially when it comes to noisy environments.

So far, only filter-based methods have been discussed; however, optimisation-based approaches have also been successfully applied to the SLAM problem. The most popular is the GraphSLAM algorithm [31] which is not bound by the key disadvantage of a filter technique in which pose data is processed and then discarded. This makes it impossible for Kalman Filter-based solutions to revisit all pose data at the time of map building. Also, unlike FastSLAM, which tracks multiple hypotheses to estimate the most likely pose of the robot, GraphSLAM works with all data at once to find the optimal (or near-optimal) solution. GraphSLAM algorithms are the first choice in noisy environments and large-scale mapping applications. The main drawback is the high computational cost that comes with running an optimisation algorithm over an ever-increasing map scale.

SLAM algorithm implementations tend to be separated into two major classes: approaches based on portable laser rangefinders named LiDAR-based SLAM and the approaches based on computer vision known as the vision-based SLAM (V-SLAM). The latter has developed rapidly over the past decade. This considerable interest in visual SLAM might be due to the fact that images contain significantly more information per observation than a normal LiDAR point cloud (without considering intensity). More information about the environment translates into more versatility to perform a larger variety of tasks. The main drawbacks of V-SLAM when compared to LiDAR SLAM is the fact that processing images is computationally expensive and the fact that the lighting conditions of the environment can significantly affect the quality of the results of visual SLAM. In addition to the two major classes, there is also room for less common SLAM applications, such as Sonar- and Radar-based SLAM.

One of the main V-SLAM algorithms is the Monocular SLAM method known as MonoSLAM [32] a method of estimating visual motion using monocular cameras. This feature-based technique can estimate the state of the features in the environment as well as the state of the camera simultaneously.

Another popular V-SLAM algorithm is the ORB-SLAM presented by Mur-Artal et al. [33]. Lv et al. [34] proposed a new and more efficient approach based on the ORB-SLAM algorithm that introduced a dense 3D real-time modelling technique. In this approach, the algorithm constructs an Octomap [35] according to the octotrees based on the estimated probabilistic occupancy. Later, Mur-Artal et al. [36] introduced an improved algorithm named ORB-SLAM2 that added support for stereo cameras and RGB-D cameras. Although computationally heavy, ORB-SLAM algorithms quickly became the most used type of visual SLAM algorithm due to their robustness and their ability to deal with low-level dynamic environments.

Dynamic environments are one of the most important challenges modern SLAM faces. Having a robot revisit a previously mapped location, it now having a different geometry, and the robot still recognising it correctly or at least not thinking it got lost is a major challenge in practical and mathematical terms. Having the robot ask the question ”Am I lost or has the environment changed?” and still be certain enough to localise itself and map the environment is a big challenge that has funnelled a significant amount of research into solving it. The current focus of the literature seems to be on deploying machine learning in conjunction with the typical SLAM algorithms for semantic object recognition. This means that more than just using the environment’s geometry and/or colour, the robot is also classifying objects and associating meaning to each one, a semantic spatial awareness. For example, if the robot recognises furniture and its training tells it that the chances of furniture moving between observations are very low, then it should confidently map the furniture as a fixed feature of the particular room. However, if at the same time the robot recognises a pet in the room, then it should know that the chance of the pet not being in the same place the next time the robot returns to the room is very high. This knowledge should help the robot discard the map features related to the pet in case it actually moves. This removal should also be efficient and not take a significant amount of time. Another challenge of semantic SLAM is to make the robot realise that an object is currently moving and that it should not be mapped until it stops or starts following a pattern. Notable works on the dynamic environment area are [37–40].

1.3 Objectives and Deliverables

This work focuses on the construction and development of a tracked mobile robot with SLAM capabilities using either a 2D laser rangefinder or a depth camera. Being the first ROS2-based tracked mobile robot with SLAM capabilities at the Laboratório de Controlo, Automação e Robótica of the Department of Mechanical Engineering of Instituto Superior Técnico, there is considerable focus on developing a modular platform that can easily be integrated in future work.

The main objectives of this work are threefold: i) the creation of a ROS2-based tracked mobile robot equipped with LiDAR and RGB-D sensors; ii) the overcoming of the irregular nature of tracked movement for state estimation before SLAM algorithms; and iii) performing localisation and mapping with 2D and 3D SLAM algorithms.

By the end of the work, the delivery of a fully functional ROS2-based SLAM capable robotic platform is expected.

1.4 Thesis Outline

After this introductory chapter, Chapter 2 provides the necessary theoretical background on the relevant theory and an overview of relevant software. The differential-drive's forward and inverse kinematics models are analysed, as well as the kinematics of a typical tracked-drive. The odometry model of a typical differential-drive is also discussed together with literature-based methods of odometry calibration. The Extended Kalman Filter is introduced and the SLAM problem is explained, in particular the relevant graph-based SLAM algorithm used in both implemented SLAM packages. For the software overview part, ROS2 is introduced, and its basic working principles are discussed. The ROS2 packages used for the implementation of the communication between low-level and main processing unit, the Extended Kalman Filter and both the SLAM algorithms are explained in the last sections of the chapter.

Chapter 3 focuses on the working principles of the hardware used in the developed robotic platform. First, the governing equations of DC motors are shown, the principles of hall-effect encoders, and the properties of regular DC motor controllers are explained. Then a brief overview of the capabilities of the processing components of the platform is provided, followed by an explanation of the working principles of the main sensors of the robot - its depth camera and 2D rangefinder. Finally, a brief rundown of the capabilities of the robot's IMU is provided.

Chapter 4 focuses on the developed robot itself. The robot's physical and electronics assembly is explained. The end of the chapter focuses on explaining the developed software's general idea and the functioning of each of its main components.

Chapter 5 contains the results of the experimental trials performed to validate the low level, the odometry model, the EKF pose estimate and finally both implemented SLAM algorithms' performance. It also provides an analysis of these results.

Chapter 6 concludes the work with a reflection on what was achieved and what can still be improved in future work.

Chapter 2

Theoretical Background and Software Overview

This chapter focuses on an overview of the theoretical concepts needed for understanding the work developed. First, in Section 2.1.1 the forward kinematics of a differential drive robot is analysed, followed by its inverse kinematics. The kinematics of tracked-drive vehicles is briefly analysed. The odometry model of differential drive vehicles and odometry tuning methods are discussed. Lastly, the relevant software is discussed, namely the ROS2 working principles and the relevant ROS2 packages used on the developed robot's codebase.

2.1 Motion Model

Motion models mathematically describe the motion of robots. A kinematics motion model describes the geometric relationships that are present in the system without taking into account forces or torques. Dynamics models include the physics of motion in which forces, energy, system mass, inertia, and velocity parameters are used. In wheeled mobile robotics, kinematics models are generally sufficient to design locomotion strategies. However, for robots moving in space, air, sea or with legged motion, a dynamics model is necessary for accurate motion prediction and control.

Motion kinematics can be separated into forward and inverse kinematics. Forward kinematics describes robot states as a function of its input (wheel speeds, joint motion, wheel steering, etc.). Inverse kinematics describes the necessary inputs for a desired robot state. However, the inverse kinematics problem deals directly with motion constraints. Not every configuration might be reachable by the model. For example, a car imposes what are called non-holonomic constraints on establishing its position. It can only turn its front wheel axle. It cannot move directly sideways.

2.1.1 Differential-Drive Forward Kinematics

The pose of the robot in a plane is defined by its state vector

$$\xi_g(t) = \begin{bmatrix} x(t) & y(t) & \varphi(t) \end{bmatrix}^T \tag{2.1}$$

where x and y represent the robot coordinates in a global coordinate frame $(X_g\,,\,Y_g)$. A moving frame $(X_m\,,\,Y_m)$ is attached to the robot's body centred on the drive-wheels' common axis. All this information is schematised in Figure 2.1.

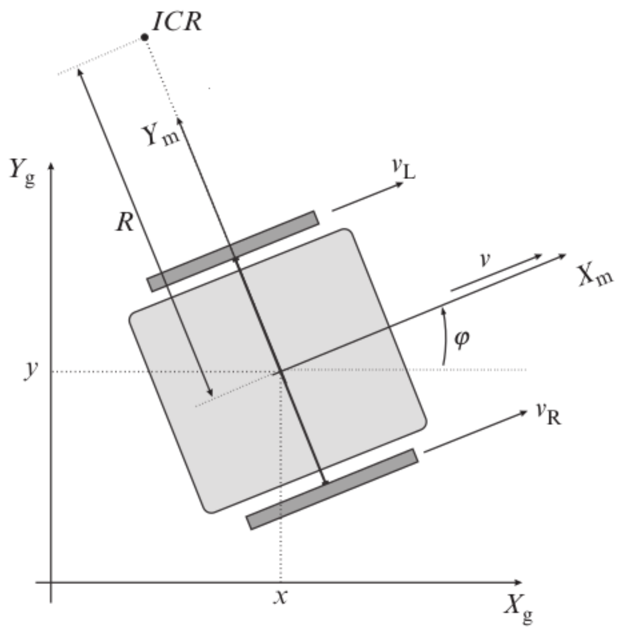


Figure 2.1: Schematic of differential-drive kinematics [41].

The relation between the moving and the global frame is defined by the translation vector $[x\,,\,y]^T$ and a rotation about φ:

$$R(\varphi) = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

For the kinematics model of a wheeled mobile robot to be applicable, one has to make the following assumptions: the plane of the wheels always remains vertical, and that in all cases there is one single point of contact between each wheel and the ground plate; there is no sliding at this single point of contact. This last assumption is equivalent to assuming that the wheel undergoes motion only under conditions of pure rolling and rotation about the vertical axis through the contact point. Evidently, there is also need to assume that motion speed and friction are low enough for dynamics to not affect the accuracy of the kinematics model.

In a wheeled mobile robot, each wheel can rotate freely around its own axis; therefore, there exists a common point that lies in the intersection of all wheel axes. This point is called an instantaneous centre of rotation (ICR) or an instantaneous centre of curvature, and it defines a point around which all the wheels follow their circular motion with the same angular velocity with respect to the ICR.

The differential-drive mechanism consists of two drive wheels mounted on a common axis. Each of these wheels can be independently driven forward or backward. Usually, there is a third wheel, a castor-wheel, or a spherical wheel, to support and balance the chassis. This enables the differential-drive robot to move forward or backward if both wheels spin in accordance with the same speed, and to rotate if both wheels spin in opposite directions or in accordance with different speeds. Differential-drive robots are non-holonomic, meaning that they are subject to a non-holonomic motion constraint: they cannot directly move laterally, having instead to make a series of manoeuvres.

According to [3, 41, 42] a differential robot's linear velocity is proportional to the sum of each of its drive-wheel's linear velocity:

$$v\left(t\right) = \frac{v_R\left(t\right) + v_L\left(t\right)}{2} \tag{2.3}$$

where v_R and v_L are the linear velocities of the right and left drive-wheel, respectively.

Its rotational velocity is proportional to the difference between its drive-wheel's linear velocity:

$$\dot{\varphi}\left(t\right) = \frac{v_R\left(t\right) - v_L\left(t\right)}{b} \tag{2.4}$$

where b is the robot's wheelbase, the distance between both drive-wheels' centre.

Since perfect friction is assumed and, therefore, no slipping between wheels and the ground, drive-wheel i's linear velocity v_i will match its tangential velocity:

$$v_i = \omega_i \cdot r_i \tag{2.5}$$

where ω_i is the wheel i's rotational speed and r_i its radius.

(2.5) allows for the expression of the robot's linear and angular velocity of (2.3) and (2.4) in order of each drive-wheel's speed:

$$v\left(t\right) = \frac{\omega_R\left(t\right) \cdot r_R + \omega_L\left(t\right) \cdot r_L}{2} \tag{2.6}$$

$$\dot{\varphi}\left(t\right) = \frac{\omega_R\left(t\right) \cdot r_R - \omega_L\left(t\right) \cdot r_L}{b} \tag{2.7}$$

Transforming these quantities to the global frame one gets:

$$\dot{\xi}_g(t) = R\left(\varphi\right) \cdot \dot{\xi}_m(t) = \begin{bmatrix} \cos\varphi\left(t\right) & 0 \\ \sin\varphi\left(t\right) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v\left(t\right) \\ \dot{\varphi}\left(t\right) \end{bmatrix} \tag{2.8}$$

2.1.2 Differential-Drive Inverse Kinematics

If there are desired moving frame linear velocity $\dot{v}^c$ and yaw velocity $\dot{\varphi}^c$, the necessary wheel speed inputs to reach them can be calculated by: (2.6) and (2.7):

$$\omega_L(t) = \frac{v^c(t) - \dot{\varphi}^c(t) \cdot \frac{b}{2}}{r_L} \tag{2.9}$$

$$\omega_R(t) = \frac{v^c(t) + \dot{\varphi}^c(t) \cdot \frac{b}{2}}{r_R} \tag{2.10}$$

2.1.3 Tracked-Drive Kinematics

The kinematics of the tracked drive - Figure 2.2 - can be approximately described by the differential-drive's kinematics. However, the no-slippage condition assumed for differential-drive is significantly weaker for tracked motion. The tracked drive has a larger contact surface between its tracks and the ground and requires slippage to turn [41]. The amount of slippage between the track and the ground is also not constant, it depends on ground contact. Odometry on a tracked vehicle is much less reliable than that of a differential-drive one.

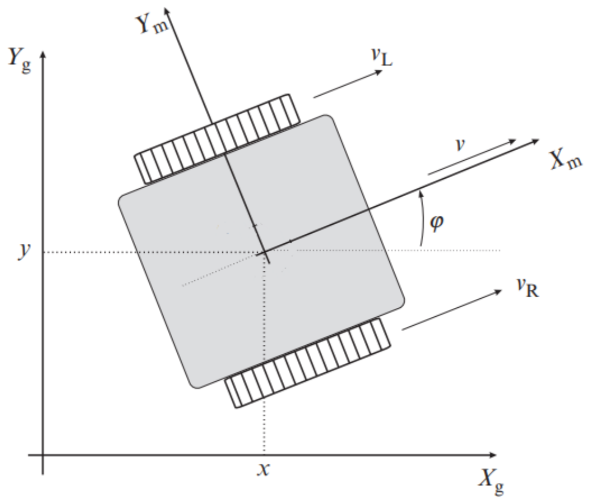


Figure 2.2: Schematic of tracked-drive kinematics [41].

2.1.4 Odometry

Odometry (*odos* meaning ”route” and *metron* meaning ”measure”) is the use of data from motion sensors to estimate the change in position over time. In the case of wheeled mobile robotics, odometry consists of estimating the distance travelled by each wheel and translating it into a robot position estimate. Odometry provides good short-term accuracy, is inexpensive, and allows for very high sampling rates. However, the basic idea of integrating incremental motion information over time leads to error accumulation. In particular, the accumulation of orientation errors will cause the odometric position estimation to drift from the actual robot position proportionally to the distance travelled. Nevertheless, odometry is still an invaluable state estimate that can be fused with absolute position measurements to provide better and more reliable position estimates. The kinematics model of (2.6), (2.7) and (2.8) can be integrated at some time t to obtain the robot pose:

$$\begin{aligned} x(t) &= \int_0^t v(t)\cos(\varphi(t))dt \\ y(t) &= \int_0^t v(t)\sin(\varphi(t))dt \\ \varphi(t) &= \int_0^t \dot{\varphi}(t)dt \end{aligned} \tag{2.11}$$

which can be written in discrete form using Euler's numerical integration approximation and evaluated at discrete time instants $t = kT_s$, $k \in \mathbb{Z}^+$ where T_s is the sampling interval between the two consecutive time-steps:

$$\begin{aligned} x(k) &= x(k-1) + v(k)T_\mathrm{s}\cos(\varphi(k)) \\ y(k) &= y(k-1) + v(k)T_\mathrm{s}\sin(\varphi(k)) \\ \varphi(k) &= \varphi(k-1) + \dot{\varphi}(k)T_\mathrm{s} \end{aligned} \tag{2.12}$$

If trapezoidal numerical integration is performed, a less noisy approximation could be obtained:

$$\begin{aligned} x(k) &= x(k-1) + v(k)T_\mathrm{s}\cos\left(\varphi(k-1) + \frac{\dot{\varphi}(k)T_\mathrm{s}}{2}\right) \\ y(k) &= y(k-1) + v(k)T_\mathrm{s}\sin\left(\varphi(k-1) + \frac{\dot{\varphi}(k)T_\mathrm{s}}{2}\right) \\ \varphi(k) &= \varphi(k-1) + \dot{\varphi}(k)T_\mathrm{s} \end{aligned} \tag{2.13}$$

The practical problem that arises from integrating the kinematics model to obtain the robot pose estimation is its dependence on accurate wheel speed values. Most applications do not include a sensor that directly measures the speed of the wheels. Most estimate wheel speeds from encoder ticks by performing numerical differentiation. This introduces noisy wheel speed values into the kinematics model, which in turn will be numerically integrated to obtain a pose estimate. One way around the problem is to take advantage of the idealised geometric properties of the differential-drive robot and derive an incremental model only reliant on the distance travelled by each wheel between each time-step, as illustrated in Figure 2.3. This way, there is no numerical differentiation step, only an inherent numerical integration step.

Since slippage between the wheel and the floor is not considered, all wheel rotation is converted into the distance travelled by the same wheel. Therefore, given the increase/decrease in encoder ticks for each wheel Δn_i, the formula to calculate the distance travelled by each wheel Δs_i during the time-step is as follows:

$$\Delta s_i = \frac{2\pi}{N}\left(\Delta n_i \cdot r_i\right) \tag{2.14}$$

where N is the number of encoder ticks per wheel revolution. Figure 2.3 depicts the distance travelled by each wheel between two consecutive time-steps as arcs for simplicity and ease of understanding. In reality, the system will be running at smaller time-steps, and the arcs will be much closer to straight lines.

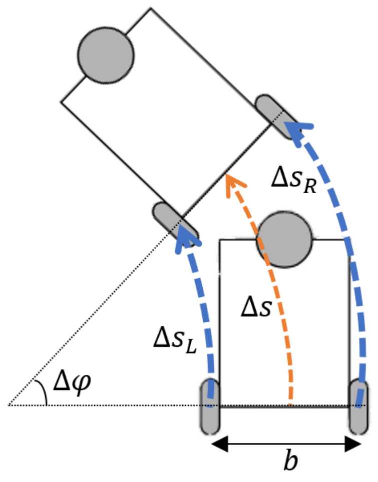


Figure 2.3: Illustration of the geometric properties of differential-drive motion.

However, the arc representation makes the inference simpler. The distance travelled by each wheel is an arc length; finding the corresponding arc angle and centre arc length is a trivial geometric matter:

$$\Delta s = \frac{(\Delta s_R + \Delta s_L)}{2} = \frac{\pi}{N}\left(\Delta n_R \cdot r_R + \Delta n_L \cdot r_L\right) \tag{2.15}$$

$$\Delta\varphi = \frac{(\Delta s_R - \Delta s_L)}{b} = \frac{2\pi}{N \cdot b}\left(\Delta n_R \cdot r_R - \Delta n_L \cdot r_L\right) \tag{2.16}$$

The global coordinate frame pose estimate is then calculated incrementally:

$$\begin{aligned} x(k) &= x(k-1) + \Delta s\cos(\varphi(k)) \\ y(k) &= y(k-1) + \Delta s\sin(\varphi(k)) \\ \varphi(k) &= \varphi(k-1) + \Delta\varphi \end{aligned} \tag{2.17}$$

Odometry Calibration

The odometry model is ever as reliable as the assumptions behind it. Assumptions such as no slippage between wheels and floor do not hold in real applications. In reality, there will always be some slippage. This means that some amount of wheel revolution will not result in robot movement. The previously determined odometry model will not be able to discern between wheel revolutions that result in movement and those that do not. Therefore, it will overestimate the distance travelled by each wheel. Due to the incremental nature of the model, these errors will accumulate and cause the odometric estimate to diverge significantly from the true robot pose. Borenstein et al. [43] divided odometry errors into systematic and non-systematic errors and focused most of their work on generalised methods to reduce the effect of both types of errors on odometric pose estimation [44]. Systematic errors can be due to unequal drive wheel diameters, actual wheel diameters being different from the nominal/measured ones, actual wheelbase being different from the nominal/measured one, misalignment of wheels, and even encoder limitations. Non-systematic errors can be due to travel over uneven floor, travel over unexpected obstacles, and wheel slippage.

Borenstein and Feng [44], on the calibration of the odometry of differential-drive robots, focused on two types of systematic errors: errors due to unequal wheel diameters and errors due to uncertainty about the wheelbase. The first type of error would make the odometry estimate a straight line when in reality the robot would swerve to the side of the lesser-diameter wheel. The second type makes the odometry overestimate or underestimate the amount of rotation the robot performed (the amount of rotation is inversely proportional to the wheelbase). To quantify these errors, Borenstein and Feng defined two experimental ratios. The experimental ratio between the radius of both drive-wheels and the experimental ratio between the actual wheelbase and the assumed one:

$$E_d = \frac{r_R}{r_L} \tag{2.18}$$

$$E_b = \frac{b_{actual}}{b} \tag{2.19}$$

With these ratios, (2.15) and (2.16) can be rewritten as:

$$\Delta s = \frac{\pi \cdot r_L}{N} \left(\Delta n_R \cdot E_d + \Delta n_L\right) \tag{2.20}$$

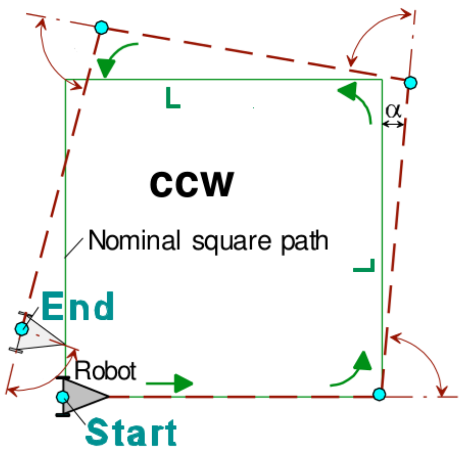
$$\Delta \varphi = \frac{2\pi \cdot r_L}{N \cdot E_b \cdot b} \left(\Delta n_R \cdot E_d - \Delta n_L\right) \tag{2.21}$$

The values of E_b and E_d could then be experimentally determined following the University of Michigan benchmark (UMBmark), also known as the bidirectional square path method [43, 44]. Figure 2.4 illustrates the effect of each of these types of errors following the UMBmark square trajectory. The method consists of making the robot perform a square of an arbitrary edge length L_{Bor} both clockwise and counterclockwise and measuring where the robot ends its trajectory. If the odometry model was perfectly in accordance with the actual robot hardware, then both the odometric estimation and the actual trajectory followed would end at the starting point. However, in real-world applications, this will not be the case. Borenstein and Feng took advantage of the geometric properties of the intended trajectory - a square - and the kinematics properties of the differential-drive robot to ultimately find an experimental methodology to determine values for E_b and E_d as a balance between the clockwise and counter-clockwise performance to improve the odometry estimation.

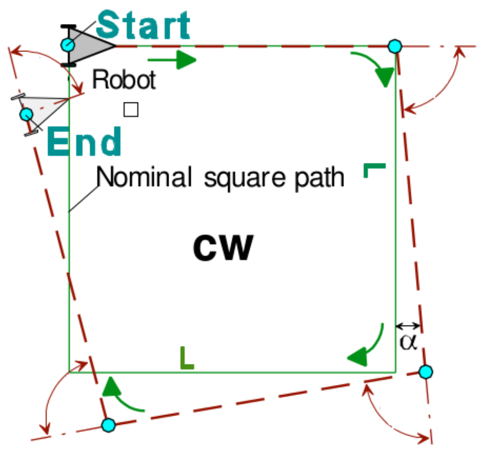
After performing multiple clockwise and counter-clockwise runs and measuring the end position of the robot, one would start to see the development of two clusters, one for each run direction. Borenstein and Feng used the centroid of these clusters $\left(x_{c.g.cw/ccw}\, ,\, y_{c.g.cw/ccw}\right)$ to compare quantitative odometric errors between different robots. The centroid of the cluster that is farthest from the origin E_{max} should be the UMBmark result used for comparison.

If the robot consistently overestimates or underestimates the amount of rotation it performs, there will be a constant angle α between the respective sides of the ideal square trajectory and the actual trajectory performed at each turn, as exemplified in Figure 2.4. By creating a balance between the performance of the clockwise and counter-clockwise runs, this angle can be approximated by:

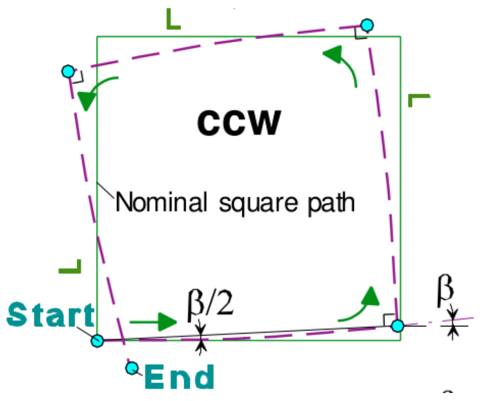
$$\alpha = \frac{x_{c.g.cw} + x_{c.g,ccw}}{-4\, L_{Bor}} \frac{180^{\circ}}{\pi} \tag{2.22}$$



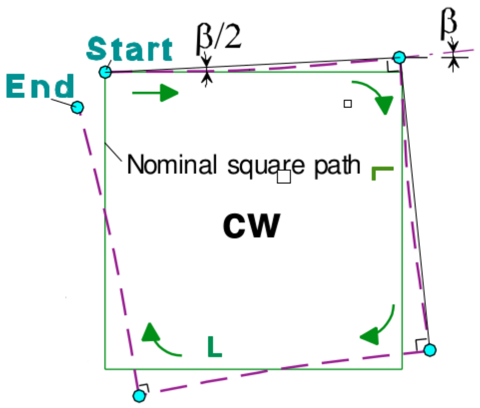
(a) Counter-Clockwise + Effect of E_b error



(b) Clockwise + Effect of E_b error



(c) Counter-Clockwise + Effect of E_d error



(d) Clockwise + Effect of E_d error

Figure 2.4: Effects of E_b and E_d errors on the UMBmark trajectory [43].

If the robot follows a curved trajectory instead of a straight one, as also illustrated in Figure 2.4's bottom row, the angle of curvature β can also be approximated, allowing for the balance between clockwise and counter-clockwise performance:

$$\beta = \frac{x_{\text{c.g.,cw}} - x_{\text{c.g., ccw}}}{-4\, L_{Bor}} \frac{180^\circ}{\pi} \tag{2.23}$$

The radius of curvature R_{Bor} can be calculated:

$$R_{Bor} = \frac{L_{Bor}/2}{\sin \beta/2} \tag{2.24}$$

With these geometric properties calculated, the determination of the ratios of (2.18) and (2.19) is based on taking advantage of the kinematics properties of the differential-drive robot [44]:

$$E_d = \frac{R_{Bor} + b/2}{R_{Bor} - b/2} \tag{2.25}$$

$$E_b = \frac{90^\circ}{90^\circ - \alpha} \tag{2.26}$$

Borenstein and Feng claim to have seen a 10- to 20-fold reduction in systematic errors by calibrating odometry with the UMBmark method [44].

2.2 Extended Kalman Filter

The nomenclature shown in this section is self-contained, meaning that the variable representation here presented is explained here and not used anywhere else in the document. Only Q_t and R_t, the process noise covariance and the measurement noise covariance matrices nomenclature, are adapted and used in other sections of the document.

The Kalman Filter [45] is an algorithm or optimal estimator that assumes that the process it is estimating is subject to perturbations and that the sensor measurements on the process are corrupted by noise. It also assumes that both the process noise and the measurement noise can be modelled as stochastic white noise, are independent, and that they have a Gaussian distribution at each time step. The ”filter” part refers to the process of filtering out this noise and optimally estimating the state at each time step.

Kalman filters admit a system of type [46]:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{\Phi}\mathbf{x}_{t-1} + \mathbf{w}_x \\ \mathbf{z}_t &= \mathbf{H}\mathbf{x}_t + \mathbf{w}_z \end{aligned} \tag{2.27}$$

where

- $\mathbf{x}_t \in \mathbb{R}^n$ is state of the system at time t.
- $\mathbf{z}_t \in \mathbb{R}^m$ are the measured variables of the system state at time t.
- $\mathbf{w}_x \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$ is the process noise at time t.
- $\mathbf{w}_z \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$ is the measurement noise at time t.
- $\mathbf{\Phi} \in \mathbb{R}^{n\times n}$ is the state transition matrix.
- $\mathbf{H}$ is the matrix of the measurement model.

The filter has two steps: the prediction and the correction steps. The prediction step estimates/predicts the next state and the covariance of this estimate, given the previous state and current input:

$$\begin{aligned} \hat{\mathbf{x}}_t &= \mathbf{\Phi}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t \\ \hat{\mathbf{P}}_t &= \mathbf{\Phi}\mathbf{P}_{t-1}\mathbf{\Phi}^T + \mathbf{Q}_t \end{aligned} \tag{2.28}$$

The correction step rectifies the prediction given the current measurements:

$$\begin{aligned}
\mathbf{v}_t &= \mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t \\
\mathbf{S}_t &= \mathbf{H}\hat{\mathbf{P}}_t\mathbf{H}^T + \mathbf{R_t} \\
\mathbf{K}_t &= \hat{\mathbf{P}}_t\mathbf{H}^T\mathbf{S}_t^{-1} \\
\mathbf{x}_t &= \hat{\mathbf{x}}_t + \mathbf{K}_t\mathbf{v}_t \\
\mathbf{P}_t &= \hat{\mathbf{P}}_t - \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^T
\end{aligned} \tag{2.29}$$

where

- $\hat{\mathbf{P}}_t \in \mathbb{R}^{n\times n}$ is the predicted covariance of the state before correction with the measurements $\mathbf{z}_t$.
- $\mathbf{P}_t \in \mathbb{R}^{n\times n}$ is the corrected estimated covariance of the state after acknowledging the measurements $\mathbf{z}_t$.
- $\mathbf{u}_t \in \mathbb{R}^k$ is the current control input vector to the system.
- $\mathbf{B} \in \mathbb{R}^{n\times k}$ is the control input matrix model.
- $\mathbf{H} \in \mathbb{R}^{m\times n}$ is the observation matrix model that links the predicted state and the measurements.
- $\mathbf{v}_t \in \mathbb{R}^m$ is the innovation or measurement residual.
- $\mathbf{S}_t \in \mathbb{R}^{m\times m}$ is the measurement prediction covariance.
- $\mathbf{K}_t \in \mathbb{R}^{n\times m}$ is the Kalman gain, directly proportional to the weight that the correction should have on the prediction.

Since the Kalman Filter is a recursive algorithm, it needs the initial state $\mathbf{x_0}$ and its respective initial covariance $\mathbf{P_0}$.

The previous equations assume the system to be linear; however, most real-world systems are not. The Extended Kalman Filter is the modification of the original Kalman Filter to work with nonlinear systems. It does this by forming a Gaussian approximation of the joint distribution of state $\mathbf{x}$ and measurements $\mathbf{z}$ using Taylor series-based transformations [46]. Instead of working with linear dynamics and measurement model functions, the EKF deals with nonlinear model functions by linearising them at each time-step and working with their jacobians. The equations of the first- and second-order Taylor series approximation for the EKF are deduced and shown in [46].

An EKF applied to attitude estimation as a quaternion with observations of triaxial gyroscopes, accelerometers, and magnetometers - similar to what the BNO055 main computing module performs - can be studied in [47]. An example of an open-source application of an EKF in ROS package format that estimates a mobile robot's state can be studied in [48].

2.3 Simultaneous Localisation and Mapping

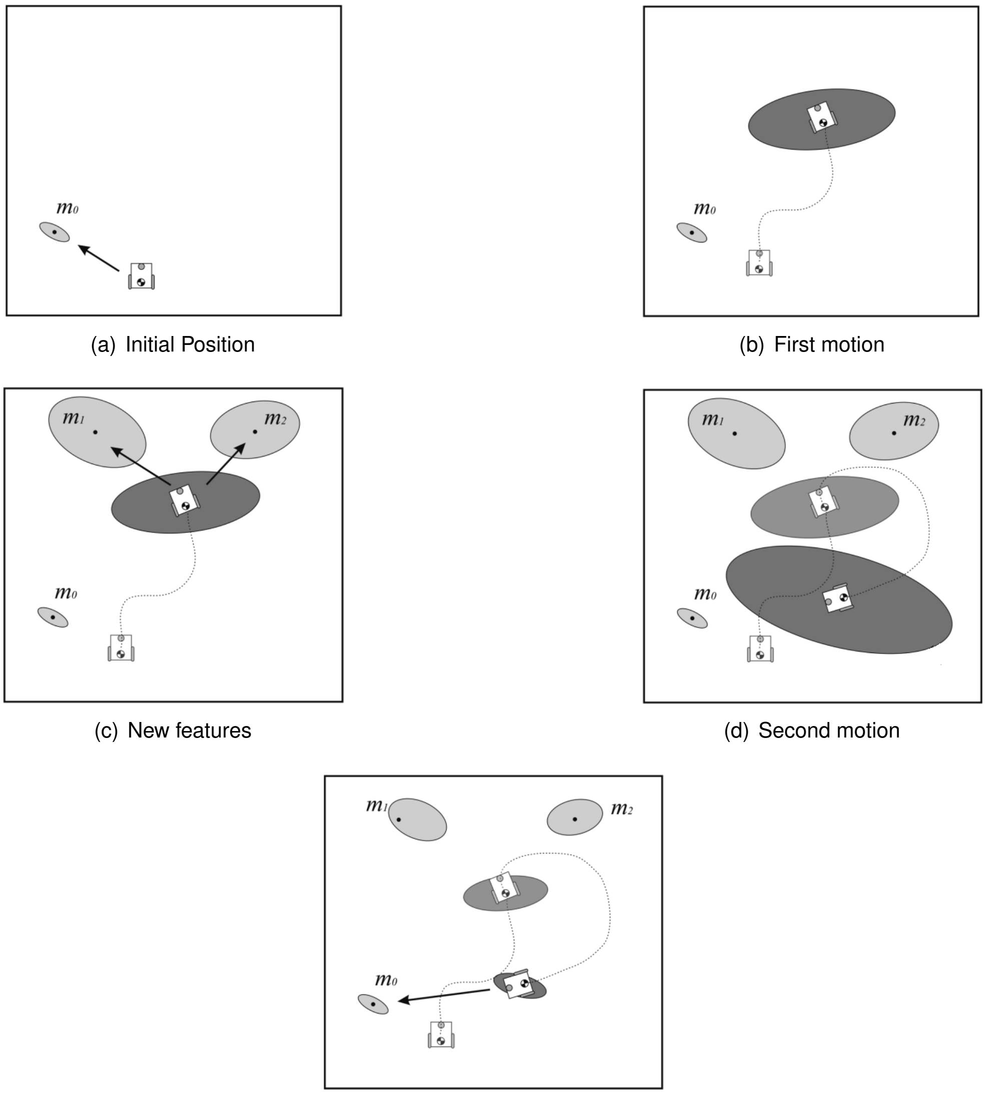
The main motivation behind the SLAM problem is the following: starting from an arbitrary initial point, a mobile robot should be able to explore autonomously the environment with its on-board sensors, gain knowledge about it, interpret the scene, build an appropriate map, and localise itself relative to this map [3].

The localisation problem consists of estimating the robot position and even its entire previous trajectory given a map of the environment. The act of mapping is the construction of a map of the environment knowing the robot's trajectory. This justifies the origin of the popular SLAM expression: *Simultaneous localisation and mapping is a chicken-and-egg problem*. For localisation, the robot needs to know where the features are, while for mapping, the robot needs to know where it is on the map [3].

From the sensor data collected, the SLAM algorithms should recover both the robot path and the environment map. Sensors that acquire information about the environment are called exteroceptive sensors, and sensors that collect information about the internal state of the robot are proprioceptive sensors [3]. The most commonly used exteroceptive sensors for SLAM are laser rangefinders, ultrasonic sensors, and cameras with or without depth recognition that can extract features from the environment. SLAM algorithms should balance the odometric estimation calculated from proprioceptive sensors and the flow of features extracted from the environment from the exteroceptive data to converge into a more precise pose estimate and a feasible environment map. However, the problem is the fact that the sensors are subject to noise. This means that a measurement's information cannot be taken as truth but as the most likely (not taking into account outliers). This makes the problem significantly harder to solve and requires the adoption of a probabilistic framework for the estimation process.

Normally, the uncertainty at the robot's initial location is assumed to be null. Figure 2.5 illustrates the typical sequence of events in a SLAM application. From its initial position, the robot recognises a feature of the environment that is mapped with an uncertainty related to the exteroceptive sensor error model. Next, the robot starts to move, which means that its pose uncertainty will increase due to the error in odometry estimation; at a new position, the robot recognises two new features; now the mapping of these features needs to take into account that the robot is uncertain about its pose and also the uncertainty related to the exteroceptive sensor error model, meaning that the map is now correlated with the robot's pose estimate. It is also possible for the robot to update its position based on the recognition of a known feature in the environment or a feature whose location is relatively well known, like previously mapped features. If the robot recognises that it is seeing a previously mapped feature, it can recursively update the uncertainty of each previous pose and consequently each feature in the map, reducing it. The re-observation of environment features is called loop closure. The robot successfully detecting loop closures is the key to solving the SLAM problem. If there are no loop closures, there is no uncertainty update, and therefore it will grow unbounded.

The SLAM problem has a continuous component and a discrete component. The continuous estimation problem pertains to the location of the features of the map and the robot's own pose variables and path. The discrete estimation problem is related to correspondence: either the current detected feature



(a) Initial Position (b) First motion

(c) New features (d) Second motion

(e) Loop closure

Figure 2.5: Typical SLAM application sequence of events [3].

is the same as a previously detected one, or it is not [24].

From a probabilistic perspective, there are two main forms of the SLAM problem. The online SLAM problem derives only the latest robot pose, while the full SLAM problem derives the entire sequence of robot poses [24] .

If the pose of the robot at time t is defined by x_t, the path of the robot is given as:

$$X_T = \{x_0\,, x_1\,, \ldots, x_T\} \tag{2.30}$$

In most SLAM algorithms, it is assumed that the initial location of the robot x_0 is known.

Let u_t denote the robot motion between the previous time step $t-1$ and the current t. This can be proprioceptive sensor data or control inputs given to the system. The sequence of relative motions of the robot can then be written as follows.

$$U_T = \{u_0\,,\, u_1\,,\, \ldots\,,\, u_T\} \tag{2.31}$$

Let M denote the true map of the environment:

$$M = \{m_0\,,\, m_1\,,\, \ldots\,,\, m_{n-1}\} \tag{2.32}$$

where $m_i\,,\, i = 0 \ldots n-1$ are vectors composed of the true positions of landmarks or characteristics extracted from the environment.

Finally, assuming that the robot takes one measurement at each time step, the sequence of landmarks or extracted environment features in the sensor's reference frame can be denoted as:

$$Z_T = \{z_0\,,\, z_1\,,\, \ldots\,,\, z_T\} \tag{2.33}$$

With this notation, the full SLAM problem can now be described as the estimation of the joint posterior probability over X_T and M from the data [3]:

$$p\,(X_T, M \mid Z_T, U_T) \tag{2.34}$$

A graphical illustration of the full SLAM problem can be seen in Figure 2.6.

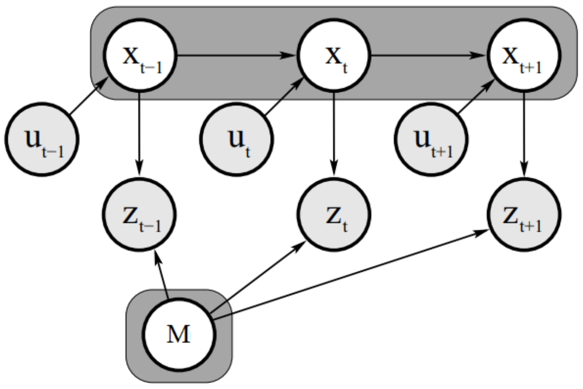


Figure 2.6: Illustration of the full SLAM problem [24].

In real applications, calculating the full posterior is usually infeasible. This is due to the high dimensionality of the continuous parameter space and the large number of discrete correspondence variables. Thus, practical SLAM algorithms rely on approximations to deal with the correspondence problem [24].

2.3.1 GraphSLAM

GraphSLAM, as its name implies, is a graph-based SLAM algorithm. These types of algorithms attempt to solve the full SLAM problem. Graph-based SLAM is born from the intuition that the SLAM problem can be interpreted as a sparse graph of nodes and constraints between nodes [3]. A node represents a robot pose. Any two nodes are connected by a soft spatial constraint, an edge. ”Soft” because it is by relaxing these constraints in a strategic/optimal way that the most likely robot path and the environment map can be estimated [3]. These constraints can be motion constraints between two successive robot poses and measurement constraints between a robot pose and a feature in the environment - illustrated in Figure 2.7. More precisely, each link in the graph is a nonlinear quadratic constraint. The target function of the GraphSLAM is the sum of these constraints. Minimising it produces the most likely map and the most likely robot path [24]. Another more physical way of interpreting the algorithm is that graph-based SLAM represents robot poses and map features as the nodes of an elastic net, the stiffness of each spring being related with the degree of uncertainty of the motion and measurement models for that observation. The complete SLAM solution can then be found by computing the minimal energy state of this net [3].

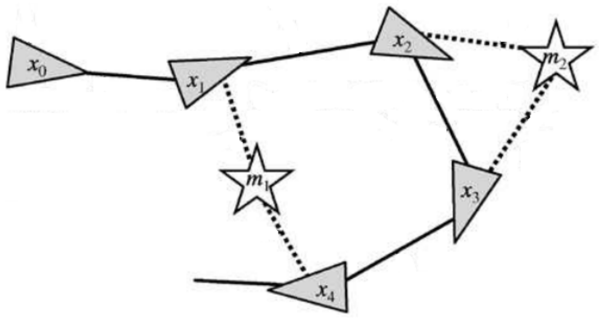


Figure 2.7: Illustration of a typical GraphSLAM graph [24].

There are two components in Graph-based SLAM: the front-end, or graph construction, where the spatial constraints are added and manipulated into the graph; the back-end, where the graph is optimised to find the best estimate of the robot path and the map. Adding constraints to the graph involves no significant computation, whereas optimising the graph can be a computationally intensive task depending on the size of the graph and topology. To compute a map posterior, GraphSLAM linearises the set of constraints into an information matrix. If the robot moves through the environment, never returning to a previously visited location, then the graph will have a linear topology. This graph will produce a nearly sparse matrix or at least a matrix that can be strategically reordered to move the non-zero elements near its diagonal. Inverting this matrix, a necessary step in the GraphSLAM algorithm, can be performed in linear time. However, if the robot returns to previously visited locations and successfully recognises this, the graph will have a cyclical topology. The information matrix resulting from these graphs is not sparse and cannot be easily reordered to facilitate inversion. A variable elimination algorithm must be applied to the graph and matrix to allow for a time-tractable inversion of the matrix. Variable elimination can be applied iteratively to remove all cyclical constraints. Going back to the elastic net analogy, this elimination is the same as removing a node and connecting adjacent ones with an equivalent stiffness spring or adjusting the tension in existing springs to compensate for the loss of the node.

The mathematical deduction, algorithm breakdown, and practical considerations on the GraphSLAM algorithm and many other SLAM algorithms can be studied in depth in the work of Thrun et al. [24].

2.3.2 Map Representation

The problem of representing the environment in which the robot moves is a dual of the problem of representing the possible position or positions of the robot [3]. The accuracy of the position representation is tied with the map's representation properties. Therefore, the precision of the map must be in accordance with the precision with which the robot is tasked with achieving its commands. The precision and possibility of features to be represented on the map must be directly tied to the capability of the exteroceptive sensors of the robot exploring the map. The more complex the map, the higher the computational cost of populating it accordingly.

Grid Map Representation

Occupancy grid map representation is the most popular map representation technique in mobile robotics [3]. In classical occupancy grids, the environment is represented by a discrete grid where each cell is either filled, representing an obstacle, or free, representing free space. In some variations of the occupancy grid, there is a third state that means that there is no information on that particular cell. The occupancy grid method works particularly well with range-based sensors. These sensor's distance measurements and the robot's own absolute position can be combined to directly fill the cells on the grid map.

In an occupancy grid, each cell can have a counter, where the value 0 indicates that the cell has not been "hit" by any range measurements and therefore it is likely to represent free space. As the number of ranging strikes increases, the cell value increases and, above a certain threshold, the cell is considered to represent an obstacle. Cell values are collectively discounted when a ranging strike travels through the cell, striking a further cell. By also discounting the values of cells over time, both hysteresis and the possibility of transient obstacles can be represented using this occupancy grid approach [3]. In probabilistic grid maps, each cell has a fill probability associated with it. This means that at every ”hit", instead of incrementing a counter, the probability of that cell being filled increases. Normally, this probability is assigned to cell darkness. In this way, the darker the cell, the higher the probability that it is part of an obstacle. Normally, a 50% cell, usually mapped to grey, means that there is no information about that particular cell, its state is unknown, and it can equally be an obstacle or represent free space.

The main disadvantage of the occupancy grid method is the fact that the size of the memory requirement increases with the size of the map. If a small cell size is selected, then the necessary memory to hold all map information will eventually explode.

2.4 Robot Operating System

The Robot Operating System (ROS) is a set of software libraries and tools to build robotic applications. ROS2 [49] is the improved version of ROS. ROS/ROS2 is a communication system, a framework for building and managing dependencies, a tool for visualisation and simulation, recording, and playing data streams, and an ecosystem with hardware drivers and libraries. It includes several open-source libraries for navigation, control, motion planning, vision, simulation, and any other robotics-related area.

ROS2 has a distributed real-time system architecture[1]. A robot's sensors, motion controllers, detection algorithms, artificial intelligence algorithms, navigation algorithms, and more are all components of this distributed architecture. The DDS (Data Distribution System) middleware selected with ROS2 for data exchange enables these components to communicate with each other in a distributed environment. ROS2 provides its own abstraction layer, leaving the details of the DDS middleware interface abstracted from the user. It is this abstraction that makes ROS/ROS2 so popular in the robotics community. The user only needs to adapt the algorithms for the ROS2 client library in either C++ or Python language, and the ROS2 middleware takes care of the communication and synchronisation between each component.

The nodes (components) communicate with each other via Topic, Service Invocation, and Actions. The topic-based communication has publish-subscribe architecture where the data produced and published by a node can be subscribed by more than one node. Service calls use a remote method invocation approach. Unlike publish-subscribe, in service communication, there is message relay only when a client node signals that it needs it. Actions are an approach used for long-service calls. The client requests an action with a goal, the action server starts the long-running process, publishes intermediate results, and reports the final result when the goal is achieved.

2.4.1 micro-ROS

micro-ROS[2] is the framework that bridges the gap between larger processing units and resource-constrained robotic applications. micro-ROS allows for the integration of microcontrollers such as the ESP32 and many others[3] into ROS2-based applications. Compatible microcontrollers with their intended algorithms adapted for the micro-ROS library can communicate directly with ROS/ROS2 applications either through serial or remote communication by running a micro-ROS agent in the main processing unit, which interprets the information being relayed by the microcontroller and publishes it in ROS-message form.

2.4.2 Robot Localisation Package

The Robot Localisation ROS package [48] is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. It contains two state estimation nodes, an Extended Kalman Filter and an Unscented Kalman Filter. It also contains a node for the integration of GPS data.

The Robot Localisation package allows for the fusion of an arbitrary number and type of sensors, as long as they are published in supported ROS message types. It offers per-sensor customisation of what data to include and other customisation options for filter behaviour.

All package state estimation nodes track the 15-dimensional state of the robot: its linear position, velocity, and acceleration, and its angular attitude (roll, pitch, and yaw) and velocity.

[1]https://medium.com/software-architecture-foundations/robot-operating-system-2-ros-2-architecture-731ef1867776
[2]https://micro.ros.org/
[3]https://micro.ros.org/docs/overview/hardware/

2.4.3 SLAM Toolbox Package

SLAM Toolbox [5] is a set of open-source tools and capabilities for 2D SLAM from laser scans for ROS-based systems. Its main features are its lifelong mapping capability, which enables the user to load a previously saved pose-graph and continue mapping. It offers synchronous and asynchronous modes of mapping and even kinematic map merging. The SLAM toolbox package is an improvement to an existing ROS SLAM package, the Karto-SLAM package[4]. Both of these SLAM algorithms are graphSLAM based. The SLAM Toolbox algorithm focuses on improving Karto's difficulty with large map scales and a more optimised integration with the ROS2 framework. SLAM Toolbox also makes several quality-of-life improvements to the Karto SLAM algorithm for graph manipulation, visualisation, and data storage.

The SLAM Toolbox package has been deployed in several robotics applications, including applications where scenes of up to 30000ft were mapped. It does this by what it calls lifelong mapping. Lifelong mapping is the concept of being able to map a space, completely or partially, and, over time, refine and update that map with continuous interaction with the space by strategically serialising and deserialising map information. This allows the user to create and update existing maps, then serialise the data for use in other mapping sessions, or even serialise it and reload strategic faraway parts of the map in the same mapping session. This serialisation also facilitates multiple robot mapping of the same environment. The scan matching component of the algorithm is the same as in the Karto-SLAM package.

2.4.4 RTAB-Map SLAM Package

RTAB-Map (real-time appearance-based mapping) is a RGB-D, stereo and LiDAR graph-based SLAM approach based on an incremental appearance-based loop closure detector[5]. The loop closure detector uses a bag-of-words approach to determinate how likely a new image represents a new, and therefore unseen, scene or not. A memory management approach is used to limit the number of locations used for loop closure detection and graph optimisation, so that real-time constraints on large-scale environments are respected. RTAB-Map can be used with a stereo camera for 3DoF mapping. It can also synchronise the information from both a stereo camera and a laser rangefinder and perform a more robust 3DoF or 2DoF map and pose estimate.

The RTAB-Map algorithm needs to know the location of the sensors in relation to the main robot body, has to have access to an odometry source, be it from proprioceptive sensors or exteroceptive sensors, and access to a stream of RGB-D images or stereo images. Optional inputs are either a laser scan from a 2D LiDAR or a point cloud from a 3D LiDAR. All messages from these inputs are then synchronised and passed to the graphSLAM algorithm. The outputs are map data containing the latest added node with compressed sensor data and the graph, corrected odometry, an optional 2D occupancy grid, and an optional map point cloud. A detailed breakdown, and practical considerations on the RTAB-Map algorithm can be consulted in Labbé and Michau's work [50].

[4]`https://github.com/ros-perception/slam_karto`
[5]`http://introlab.github.io/rtabmap/`

Chapter 3

Hardware Overview

This chapter focuses on providing brief explanations of the hardware used on the developed robot. First, the functioning of DC motors and their governing equations are explained in Section 3.1.1, next Hall-effect encoders are introduced in Section 3.1.2 and DC motor controllers are explained in Section 3.1.3. The following section focuses on the processing units, namely the Raspberry Pi in Section 3.2.1 and the ESP32 in Section 3.2.2. The operation of depth cameras is reviewed in Section 3.3 and the IntelRealsense D435 in particular in Section 3.3.1. Next are laser rangefinders in Section 3.4 and the Hokuyo URG-04X-UG01 in particular in Section 3.4.1. Finally, there is a brief explanation on AHRS in Section 3.5 and Adafruit's BNO055 in particular in Section 3.5.1.

3.1 Direct Current Motors, Accompanying Sensors and Controllers

3.1.1 Direct Current Motors

The term Direct Current motor (DC motor) is used to refer to any rotary electrical machine that converts direct current electrical energy into mechanical energy. DC motors range from light weight motors weighing less than 100g, capable of powering mini-electronics, to weighing several tons and being capable of powering large wind tunnels, for example.

The DC motor is made up of two main parts: the stator, which remains stationary, and the armature, the moving part. In a DC motor, the stator provides a rotating magnetic field that forces the armature to rotate.

All electric motors develop torque by alternating the polarity of rotating magnets attached to the rotor and stationary magnets on the surrounding stator. At least one of these sets of magnets is an electromagnet made from a coil of wire around an iron core. In a DC motor, the direct current flowing through the wire winding creates a magnetic field. Each time the armature rotates $180°$, the positions of the north and south poles are reversed. If the magnetic field of the poles remained the same, the rotor would not turn. To create torque in one direction in a DC motor, the direction of the electric current must be reversed with every $180°$ turn of the armature. In a traditional brushed motor, this would be done by a commutator, but in a brushless DC motor, an electronic sensor instead detects the angle of the rotor, with controlled semiconductor switches either reversing the direction of the current or turning it off at the correct time in the rotation to create torque in one direction.

The governing equations of a DC motor can be derived from Kirchhoff's voltage law and Newton's second law :

$$\ddot{\theta}(t) = \frac{K \cdot i(t) - T - B \cdot \dot{\theta}(t)}{J} \tag{3.1}$$

$$V(t) = K \cdot \dot{\theta}(t) + R \cdot i(t) + L \cdot \frac{\partial i(t)}{\partial t} \tag{3.2}$$

where J is the motor moment of inertia, K the current-developed torque conversion factor, L the inductance, R the resistor, T the static friction factor, B the viscous friction factor, V the supplied voltage and $\dot{\theta}$ the motor's output-shaft rotational speed.

Assuming that the voltage drop due to L is negligible, (3.1) and (3.2) can be reworked as follows:

$$\ddot{\theta}(t) = \frac{1}{J}\left(\frac{K}{R}\left(V(t) - K \cdot \dot{\theta}(t)\right) - B \cdot \dot{\theta}(t) - T\right) \tag{3.3}$$

The differential equation (3.3) can be solved in order of the motor spin value:

$$\dot{\theta}(t) = \frac{a_1}{a_2}\left(1 - \exp\left(-a_2 \cdot t\right)\right) \tag{3.4}$$
$$\text{where} \quad a_1 = \frac{K \cdot V(t) - R \cdot T}{R \cdot J} \qquad a_2 = \frac{K^2 + R \cdot B}{R \cdot J}$$

From (3.4) one can infer that the a_2 term is always null or positive and therefore for the a_1 term to be positive:

$$\begin{gathered} \{\forall t \geq 0\,,\, a_2 \geq 0\} \implies (1 - \exp(-a_2 \cdot t)) \geq 0 \\ \therefore \\ \left\{\forall t \geq 0\,,\, \dot{\theta}(t) \geq 0\right\} \implies a_1 \geq 0 \iff V(t) \geq \frac{R \cdot T}{K} \end{gathered} \tag{3.5}$$

This is relevant because, when only considering the stationary response of the motor $t \longrightarrow \infty$, this is the value from which the signal of $\dot{\theta}$ changes.

3.1.2 Hall Effect Encoders

Encoders are sensors that encode rotation angle (rotary encoder) or linear displacement (linear encoder). They can be of mechanical, optical, magnetic or electromagnetic induction type depending on what physical property is sensed to detect rotation or linear motion.

The magnetic encoder detects the rotational position information as the magnetic field changes and converts this information into an electrical signal. A simple magnetic encoder consists of a permanent magnet and a magnetic sensor. The permanent magnet is attached to the tip of a rotating body, such as a motor shaft, and the magnetic sensor is fixed in a state where it is mounted on a PCB board at a position where it receives the magnetic field generated by the permanent magnet. When the permanent magnet attached to the motor shaft rotates, the direction of the magnetic field detected by the magnetic sensor changes, and the encoder detects the rotational position and speed of the motor shaft [51].

EMG30 Motors

The EMG30[1] (encoder, motor, gearbox) is a $12\mathrm{V}$ motor fully equipped with hall effect encoders and a $30 : 1$ reduction gearbox can be seen in Figure 3.1. It also includes a standard noise suppression capacitor in all motor windings. Under nominal conditions its speed can range from 1.5 to 200 rotations per minute. The main characteristics of the motor are found in Table 3.1.

Figure 3.1: EMG30 motor.

Table 3.1: Main characteristics of EMG30 DC motors.

Rated Voltage	$12\mathrm{V}$
Rated Torque	$150\mathrm{kg/m}$
Rated Current	$530\mathrm{mA}$
No Load Current	$150\mathrm{mA}$
Stall Current	$2.5\mathrm{A}$
Rated Output	$4.22\mathrm{W}$
Output Shaft Diameter	$10\mathrm{mm}$
Encoder Ticks per Output-Shaft Turn N	360

3.1.3 DC Motor Controllers

DC motor controllers are devices that can control the position, speed, or torque of DC motors. DC motor controllers can exploit the fact that these types of motor can easily reverse their spin direction by simply switching their leads. DC motor controllers can also alter motor speeds by simulating a supply voltage decrease/increase. Adjustable speed drive controllers send periodic pulses to the motor, which, when combined with the smoothing effect caused by coil inductance, makes the motor act as if it is being powered by a lower/higher voltage. Another way to affect the DC motor speed is by varying the current fed through either the field coil or the armature. The speed of the output shaft will change when the current through these coils changes, as its speed is proportional to the strength of the magnetic field of the armature (dictated by the current). Variable resistors or rheostats in series with these coils can be used to alter the current and therefore the speed of the motor [52].

[1]`https://www.robot-electronics.co.uk/htm/emg30.htm`

MD25 Motor Controller

The MD25[2] is a dual DC motor controller, designed for use with EMG30 motors - Figure 3.2. The MD25 can communicate over I2C or Serial protocols. It can read the EMG30's hall-effect encoders and provide the encoder ticks directly; it can drive two motors with independent or combined control; it can estimate the amount of current the motor/s are drawing and the power-source voltage level; it offers speed regulation; it has got variable acceleration and power regulation user-selected modes. The MD25 is designed to work with a $12\mathrm{V}$ battery and accepts voltages of $9\mathrm{V}$ to $14\mathrm{V}$.

The board regulates the speed of the motors with supply voltage decrease/increase and a closed-loop PID-like circuit. This means that by using feedback from the encoders the MD25 is able to dynamically increase power as required. If the required speed is not reached, MD25 will increase the motor power until it reaches the desired rate or the motors reach maximum output.

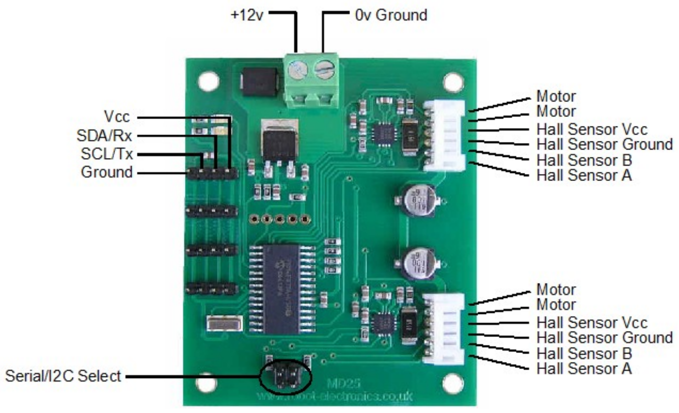


Figure 3.2: MD25 motor controller schematic.

3.2 Main Processing Components

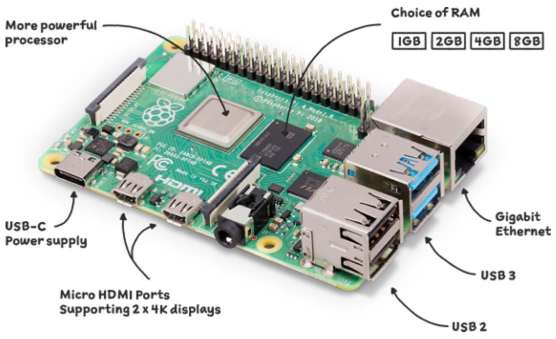
3.2.1 Raspberry Pi 4 Model B

The Raspberry Pi is a series of small single-board computers (SBCs) developed by the Raspberry Pi Foundation. An SBC is a complete, functioning computer in which the microprocessor, input/output functions, memory, and other features are all built on a single circuit board, with RAM built in at a predetermined (typically nonexpandable) amount. Furthermore, the Raspberry Pi series of computers comes with a set of GPIO (general purpose input/output) pins, allowing the control of electronic components for physical computing and Internet of Things (IoT) applications.

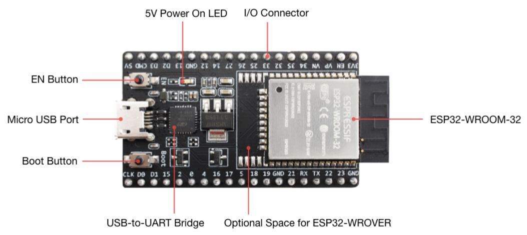
The Raspberry Pi 4 Model B[3] - Figure 3.3 - is the latest product (2018) in the Raspberry Pi series. It's key features include a high-performance 64-bit quad-core processor, dual-display support at resolutions up to 4K via a pair of micro-HDMI ports, hardware video decode at up to 4Kp60, up to 8GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, and USB 3.0 ports.

[2]https://www.robot-electronics.co.uk/htm/md25tech.htm
[3]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

There are several different Linux-based operating systems available for use on the Raspberry Pi. Linux is great for critical applications because it focuses on security and stability rather than the mainstream operating systems that focus on ease of use. Linux is also lightweight and less resource intensive than commercial operating systems. Most Linux based operating systems compatible with the Raspberry Pi come in two different versions: the desktop version, a version with a desktop and command line interface, or the server version, a lightweight version with just a command line interface.



(a) Raspberry Pi 4B



(b) ESP32 DevKit C

Figure 3.3: Processing units used.

3.2.2 ESP32 DevKit C

The ESP32 series is a low-cost, low-power microcontroller series with integrated Wi-Fi and Bluetooth capabilities and 4 MB flash memory. DevKit C models - Figure 3.3 - in particular, offer 520kB of sRAM, 34 GPIOs, 28 of which are pulse width modulation (PWM) capable, several pins compatible with the I2C, SPI and UART communication protocols, and considerable open-source community support[4].

3.3 Depth Cameras

Standard digital cameras output images as a 2D grid of pixels. Each pixel has values associated with it – generally a group of numbers from 0 to 255. Using the red-green-blue system (RGB), for example, the group (255,0,0) would mean a bright red pixel. On the other hand, a depth camera has pixels that have a different numerical value associated with them, the distance from the camera, or 'depth'. Some depth cameras have both an RGB and a depth system, which can give pixels with all four values (RGB-D). The most popular methods to calculate depth are the Structured Light, Coded Light, and Stereo Depth methods.

Structured light and coded light depth cameras both rely on projecting light (infrared) from an emitter onto the environment. The projected light is patterned. Because the projected pattern is known, how the camera sensor sees the pattern in the scene provides the depth information. The reliance of these methods on accurately seeing a projected pattern of light makes coded and structured light cameras perform best indoors at close range. However, the noise in the environment from other cameras or devices that emit infrared light can affect the performance of these methods.

[4]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html

Stereo depth cameras also often project infrared light onto a scene to improve the accuracy of the data, but unlike coded or structured light cameras, stereo cameras can use any light to measure depth. Stereo depth cameras have two sensors, spaced a small distance apart. A stereo camera takes the two images from these two sensors and compares them. Since the distance between the sensors is known, depth information can be calculated from this comparison. Because stereo cameras use any visual features to measure depth, they will work well in most lighting conditions, including outdoors. The addition of an infrared projector means that, under low lighting conditions, the camera can still perceive depth details [53].

3.3.1 Intel RealSense D435

The Intel RealSense D435 stereo depth camera - Figure 3.4 - offers a wide field of view and a range of up to 10m, in a small form factor. It is a widely supported piece of hardware in the robotics and open-source community, mainly due to Intel's release and frequent update of its RealSense SDK (Software Development Kit) 2.0 . Its depth sensor has a field of view of $87^\circ \times 58^\circ$, an output resolution of up to 1280×720 and a frame rate of up to 90fps. The minimum depth acknowledged is about 28cm and its reported depth accuracy is < 2% at 2m [5]. Its RGB sensor has a $69^\circ \times 42^\circ$ field of view, a 2MP resolution, up to 30fps frame-rate, and up to 1920×1080 frame resolution.

Figure 3.4: Intel Realsense D435 camera.

3.4 Laser Rangefinders

Laser rangefinders estimate distances by calculating phase dispersion. They emit a light wave at a particular frequency, which moves in a straight line, rebounds from obstacles, and returns to the sensor. The rangefinder compares the transmitted and received waves and calculates the phase difference between the two. This phase difference is, in fact, proportional to the time taken by the laser to go from the sensor to the obstacle and back. With the knowledge of the time taken and the speed of light, one can calculate the distance travelled. Lasers are very effective because their light wave reflects off all solid surfaces with limited divergence, regardless of obstacle nature. To take measurements over a plane, the sensor incorporates a small rotating mirror that shifts an angular amount between each measurement, redirecting the light wave and performing a sweeping movement. 3D laser scanners use the same technique but with a mirror rotating on two axes.

[5] https://www.intelrealsense.com/depth-camera-d435/

3.4.1 Hokuyo URG-04LX-UG01

The URG-04LX-UG01[6] - Figure 3.5 - is a 2D laser rangefinder for measuring distances. It has a scan range of $240°$ and a depth detection interval between 3cm and 5.6m . The reported accuracy for obstacles farther away than 1m is $\pm 3\text{cm}$ and for objects farther away is $\pm 3\%$. The sensor has an angular resolution of $0.352°$ and a 10Hz rate. Its main properties are shown in Table 3.2.

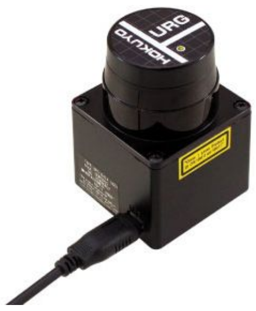


Figure 3.5: Hokuyo URG-04LX-UG01 2D laser rangefinder.

Table 3.2: Hokuyo URG-04LX-UG01 2D rangefinder main specifications.

Scan Area	$240°$
Maximum Range	5.6m
Wavelength	785nm
Angular Resolution	$0.352°$
Radial Resolution	1mm
Measurement Accuracy	$\pm 3\%$
Power Supply	5VDC
Connection Interface	USB2.0

3.5 Attitude and Heading Reference Systems

An Inertial Measurement Unit (IMU) is an electronic device that measures and reports acceleration, orientation, angular rates, and other gravitational forces. A 9DoF IMU is made up of three accelerometers, three gyroscopes, and, depending on the heading requirement, three magnetometers. An AHRS - like the BNO055 in Figure 3.6 - adds a central processing unit (CPU) to the IMU. This processing unit enables the onboard execution of attitude and heading estimation algorithms from all sensor data. The inclusion of a processing unit also facilitates more robust calibration procedures to avoid/correct sensor biases, gyro drift, cross-axis, and misalignment effects.

3.5.1 Adafruit's BNO055 AHRS

The BNO055[7] is a System in Package (SiP) solution that integrates a 14-bit triaxial accelerometer, a 16-bit close-loop triaxial gyroscope, a 32-bit geomagnetic sensor, and a 32-bit microcontroller running the BSX3.0 FusionLib software. The package can output its absolute orientation in Euler vector or

[6]https://www.hokuyo-aut.jp/search/single.php?serial=166
[7]https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor

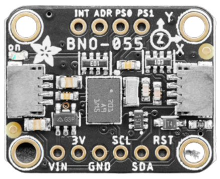


Figure 3.6: Adafruit's BNO055.

quaternion form at $100\mathrm{Hz}$, its angular velocity and acceleration vector at $100\mathrm{Hz}$, its temperature at $1\mathrm{Hz}$, the magnetic field strength vector at $20\mathrm{Hz}$ and finally the gravity and linear acceleration vectors at $100\mathrm{Hz}$. All this can be easily integrated into any project taking advantage of Adafruit's open source libraries made specifically for this package[8] and sensors [9].

[8] https://github.com/adafruit/Adafruit_BNO055
[9] https://github.com/adafruit/Adafruit_Sensor

Chapter 4

TIR-ANT: Tracked Inspired Robot - Autonomous Navigation Tool

This chapter focuses on the mechanical (Section 4.1.1) and electronics (Section 4.1.2) assembly of the robot. The robot built was named *TIR-ANT* and will be referred to as so from this chapter on. After the section on the TIR-ANT's assembly, the developed software is explained in Section 4.2.

TIR-ANT comes from **T**racked **I**nspired **R**obot - **A**utonomous **N**avigation **T**ool. The letter ”T” might also double for **T**raxter, the name of the robot that donated its tracks to TIR-ANT. The ”ANT” part of the name is commonplace to mobile robots developed in the Laboratório de Controlo, Automação e Robótica of the Department of Mechanical Engineering of Instituto Superior Técnico.

Access to all the developed software, how-to-guides about TIR-ANT and media of the robot, including videos of it functioning, are made available in the robot's Github repository [54].

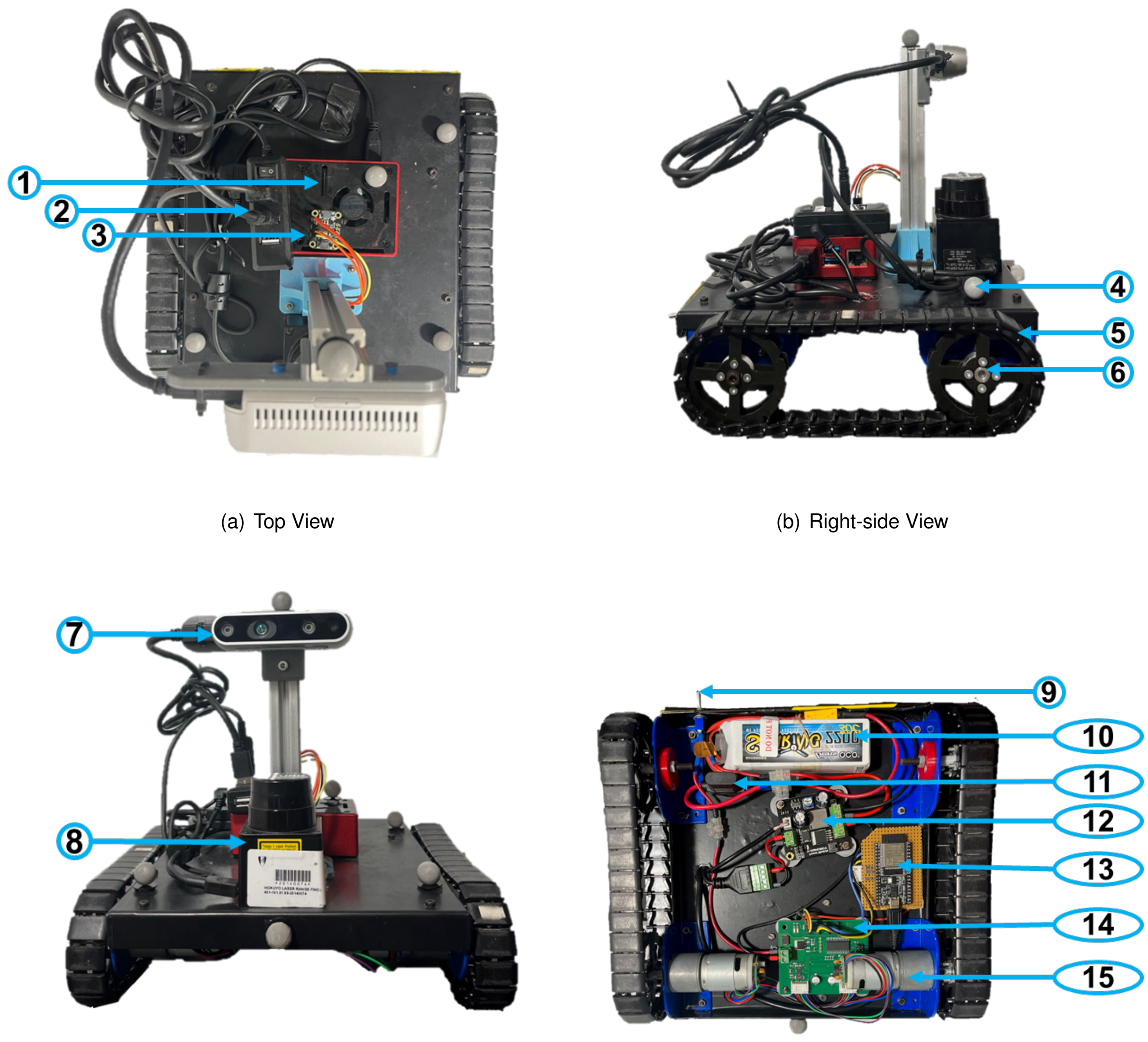
4.1 Robot Assembly

The final assembled robot can be seen in Figure 4.1 and its components breakdown can be consulted in the accompanying Table 4.1.

4.1.1 Mechanical Assembly

The main robot chassis was made from a $230 \times 200 \times 1.5\mathrm{mm}$ aluminium plate which was bent $20\mathrm{mm}$ front and back and $10\mathrm{mm}$ laterally by $90°$. The tracks and track mounts of the robot were reused from an old non-functional robot named *Traxter*. The EMG30 mounting brackets and track mounting pieces were placed in the chassis in a way that ensured that the tracks were fully tense. The EMG30 motors are placed in the front of the robot.

A Hokuyo URG-04LX-UG01 2D rangefinder was screwed to the front edge of the chassis. Right behind the rangefinder, a $20 \times 20\mathrm{mm}$ V-slot aluminium extrusion profile was screwed onto the chassis using 3D printed mounting. This aluminium extrusion profile extends $15\mathrm{cm}$ vertically from the chassis,



(a) Top View (b) Right-side View

(c) Front View (d) Bottom View

Figure 4.1: The TIR-ANT. Identifier breakdown in Table 4.1.

and an Intel Realsense D435 camera was attached to it via another 3D printed mount.

A Raspberry Pi 4B with 8GB of RAM equipped with a 32GB micro-SD card was attached to the chassis using acrylic tape for ease of removal. The board is protected by an aluminium case and cooled by an included small electric fan. The board was placed with its USB and Ethernet ports facing the right-hand side of the robot and its power and video output ports facing backwards. A hole was made in the right side of the chassis to allow cables to go from the underside of the robot to the Raspberry Pi.

The Adafruit's BNO055 was placed on top of the Raspberry Pi case, carefully centred on the chassis and in a way to ensure that its x-axis faces the front of the robot, the y-axis is to the left, and the z -axis upward. An externally powered USB-hub was also attached with acrylic tape on top of the Raspberry Pi case.

Seven reflective markers were placed throughout the robot chassis to ensure that a motion capture system can track its movements.

Table 4.1: Figure 4.1's accompanying table with the components breakdown.

Identifier	**Component Name**
1	Raspberry Pi 4B
2	USB Hub
3	Adafruit's BNO055 AHRS
4	Reflective Marker
5	Right Track
6	Right Track Mount
7	Intel RealSense D435 Camera
8	Hokuyo URG-04LX-UG01 Laser Rangefinder
9	Low-level On/Off Switch
10	Li Po Battery 11.1V/2200mAh
11	10A Fuse
12	25W Buck Converter
13	ESP32 DevKit C microcontroller
14	MD25 DC Motor Controller
15	EMG30 DC Motor

The robot is equipped with a $2200\mathrm{mAh}$ $11.1\mathrm{V}$ LiPo battey. A $25\mathrm{W}$ buck converter, an ESP32 DevKit C and a MD25. Most of these components are screwed or mounted to a 3D printed part, which in turn is attached to the underside of the chassis using acrylic tape.

4.1.2 Electronics Assembly

As stated above, the robot's power source is a $11.1\mathrm{V}$ Li-Po battery. This voltage level is compatible with MD25 and EMG30 motors, but not with the Raspberry Pi, Hokuyo rangefinder, RealSense camera, and ESP32 which accept only $5\mathrm{V}$ input voltage. Therefore, the robot power supply circuit is divided into high and low voltage circuits. The $25\mathrm{W}$ buck converter takes power from the Li-Po battery and outputs it at $5\mathrm{V}$ and is capable of providing at most $5\mathrm{A}$ of current. This way, the MD25 and the motors are supplied directly by the Li-Po battery, and the Raspberry Pi and other components are supplied with the $5\mathrm{V}$ converted by the buck converter. The power supply circuit is schematised in Figure 4.2.

For safety of the circuit, a $10\mathrm{A}$ fuse is installed in series with the battery. This means that if at any time the components draw more than $10\mathrm{A}$ current, the fuse will break and the circuit will open. Components reaching a $10\mathrm{A}$ current draw directly from the battery would mean something is wrong, since the current draw in the expected worst-case scenario is around $8.5\mathrm{A}$. There is also an On/Off switch connected directly to the MD25 supply. This allows for the emergency shutdown of the low-level of the robot without shutting down the Raspberry Pi, the RealSense camera, and the Hokuyo rangefinder.

The maximum current that the Raspberry Pi 4 can pass through its USB ports is $1.2\mathrm{A}$. This means that the total current that the board can supply from all its combined USB ports cannot exceed $1.2\mathrm{A}$. The Raspberry Pi cannot power both the RealSense camera and the Hokuyo rangefinder. To avoid this issue, an externally powered USB hub was added to the circuit. The RealSense camera and the Hokuyo rangefinder can be powered through this USB hub directly from the same power source as the Raspberry Pi. Not having to power the components directly, the Raspberry Pi is connected to the USB hub only receiving sensor data (Figure 4.3). The ESP32 draws a small amount of current and can be

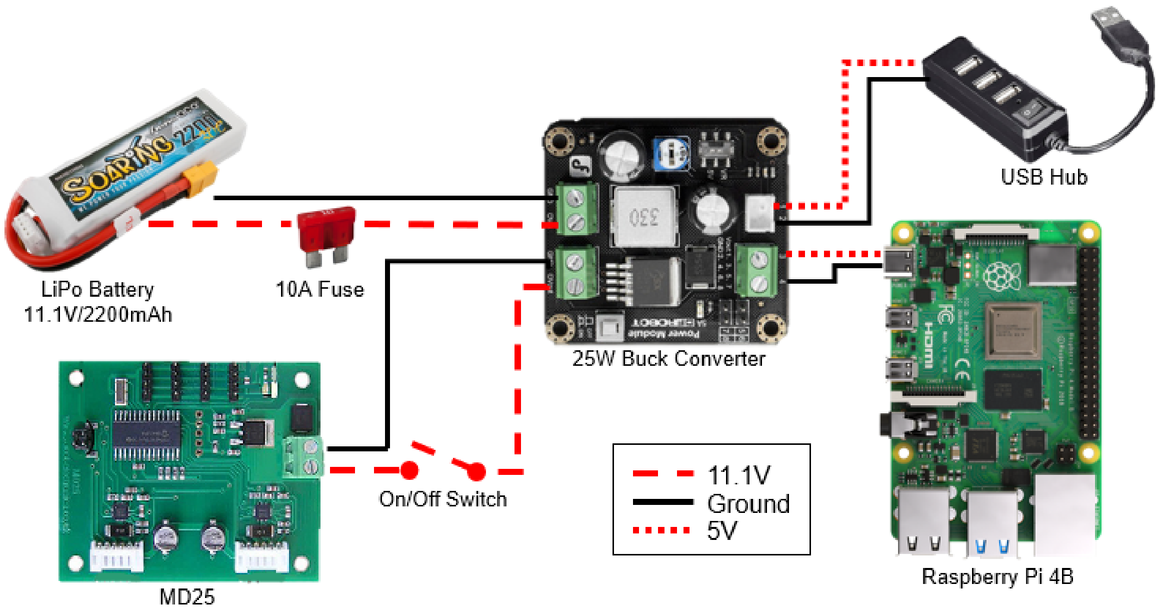


Figure 4.2: Power supply circuit.

directly connected to the Raspberry Pi. It is also worth noting that if only one of the sensors is needed and not both the RealSense camera and the Hokuyo rangefinder at the same time, then the USB hub is not necessary and the sensor can be directly connected to the Raspberry Pi.

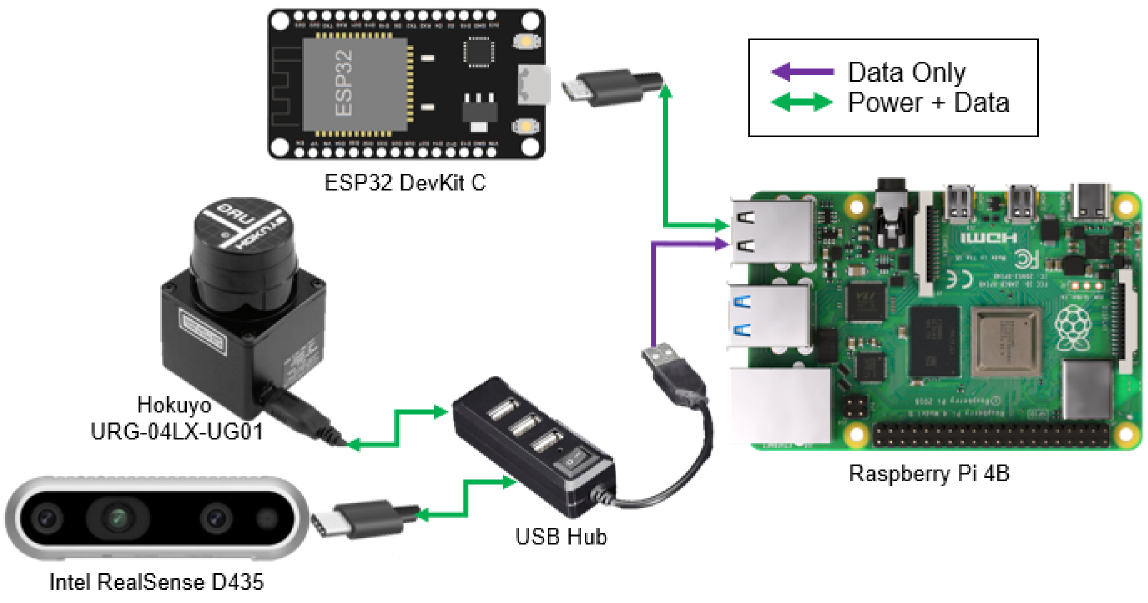


Figure 4.3: Sensor power and data flow circuit.

The ESP32 microcontroller is tasked with receiving commands from the Raspberry Pi and sending sensor data through a USB serial connection. The microcontroller is connected to the MD25 and BNO055 (Figure 4.4). It communicates with both these components through an I2C bus. The BNO055 is a $3.3\mathrm{V}$ component and therefore is powered directly by the ESP32.

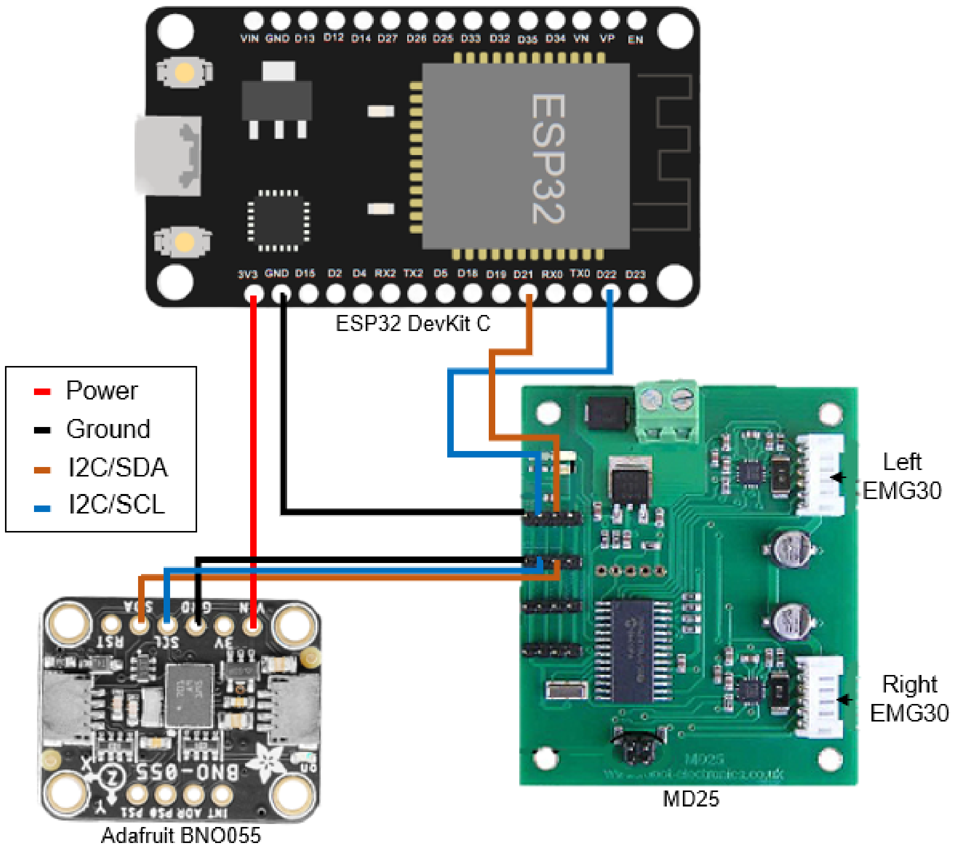


Figure 4.4: Low-level circuit.

4.2 Software Integration

The entire software codebase was developed for ROS2 Humble Hawksbill, which was released in May 2022. This version of ROS2 was chosen because it is the latest long-term service release of the ROS2 framework, which guarantees support until 2027. The operating system running on the 8GB Raspberry Pi 4B equipped with a 32GB SD-card is Ubuntu 22.04 Jammy. This OS was released in April 2022. The reasons behind the selection of this Ubuntu version were twofold: it is the latest long-term service release of Ubuntu that guarantees support until 2027; ROS/ROS2 software works best on Ubuntu-based devices.

The selection of the latest operating system and the newest ROS2 version adds future proofing to the work and allows the robot to remain operational at least until 2027.

Figure 4.5 contains the schematic of the entire ROS2 framework that runs on the TIR-ANT. Starting at the low-level, the algorithm running on the ESP32 is recognised as *lowlevel_node*. This algorithm is responsible for receiving wheel speed commands and mapping them to appropriate motor commands. It is also tasked with publishing encoder tick information and BNO055 information at a 50Hz rate. More detailed information on the low-level algorithm is available in Section 4.2.1.

Communication between the low-level and the main computing Raspberry Pi is made through USB-serial. The micro-ROS library takes care of the transport and synchronisation tasks. The interpreter between the Raspberry and the low-level is the micro-ROS agent, which codifies and decodifies the

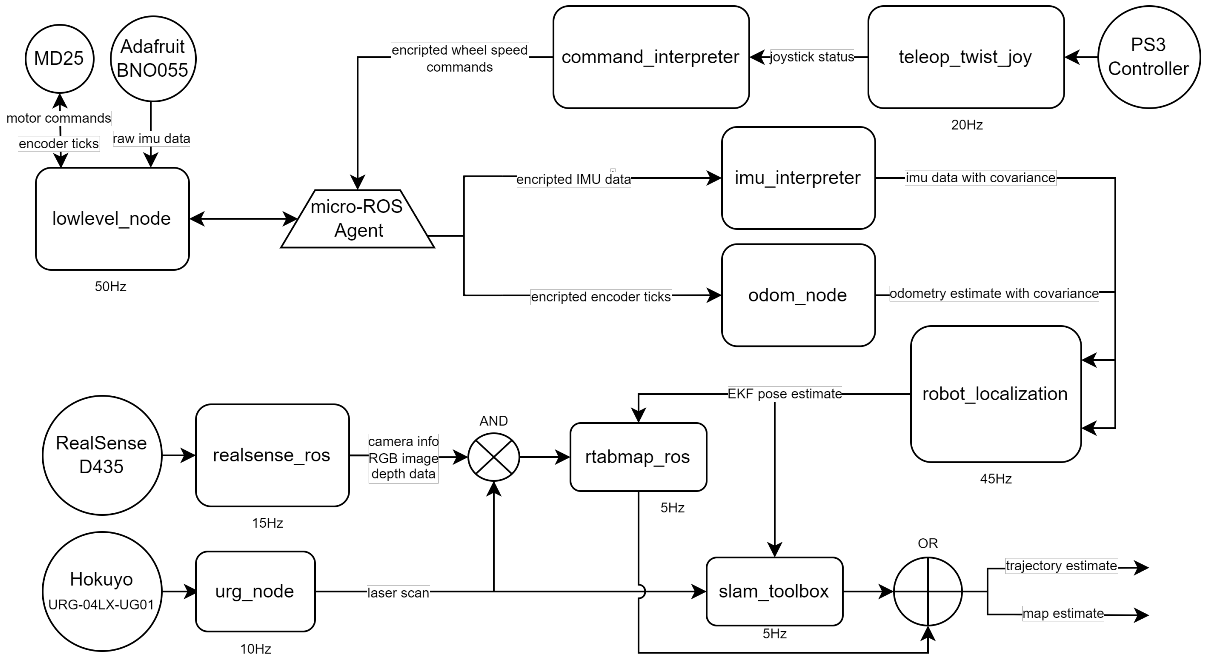


Figure 4.5: Schematic of entire developed software pipeline.

messages going through serial communication appropriately. Every message that goes in or out of the micro-ROS agent is ”encripted”. This only means that the numerical values are multiplied by a factor in order to avoid decimal points. Serial communication is better at dealing with integers than float types. Consequently, nodes that receive information that went through the micro-ROS agent need to be prepared to remove the multiplicative factor. More information on the workings of the micro-ROS framework is available in Section 2.4.1.

Data streams from the RealSense camera, the Hokuyo rangefinder, and the PS3 controller can be entered directly into the ROS2 framework through the *realsense_ros*, *urg_node* and *teleop_twist_joy* libraries, respectively. More information in Section 4.2.3.

The $imu\_interpreter$ node removes the multiplicative factor from the BNO055 orientation, angular velocity, and acceleration data and publishes it into the appropriate ROS-message type together with the covariance associated with the measurement. $odom\_node$ takes the encoder tick data and converts it into an odometric pose estimate. It publishes this pose estimate, together with a body frame velocity estimate in an appropriate ROS-message type together with the covariance associated with these estimates. The EKF of the *robot_localization* package then fuses the information by $odom\_node$ and $imu\_interpreter$ and publishes an improved pose estimate at a $45\mathrm{Hz}$ rate.

The $command\_interpreter$ algorithm takes the desired linear and angular moving frame velocity from the PS3 controller, applies (2.9) and (2.10) and sends the wheel speed commands to the low-level.

For the 2D mapping SLAM algorithm $slam\_toolbox$ package[1] only the laser scan data from the Hokuyo rangefinder is necessary. These scans are published at $10\mathrm{Hz}$. The 3D mapping algorithm is the one from $rtabmap\_ros$ package[2], requiring both the laser scan and the RealSense camera data.

[1]https://github.com/SteveMacenski/slam_toolbox
[2]https://github.com/introlab/rtabmap_ros

Both SLAM algorithms will take the EKF pose estimate and the exteroceptive sensors' data to perform map building and localisation and publish results at $5\mathrm{Hz}$.

4.2.1 Low-Level

Desired Speed to Motor Command

Gonçalves et al [55] modelled the EMG30 motors for simulation. The identified motor parameters can be seen in Table 4.2. With these values, (3.4) can be further developed as the following:

$$\dot{\theta}(t) = -3.7638\left(\mathrm{e}^{-6.599\,t} - 1.0\right)(0.509\,V_{in} - 0.28404) \tag{4.1}$$

Table 4.2: EMG30 motor parameters from [55].

Parameter	Value
K	0.509
R	7.101Ω
L	$3.4 \times 10^{-3}\mathrm{H}$
B	9.31×10^{-4}
J	$3.8 \times 10^{-3}\mathrm{kg}\cdot\mathrm{m}^2$
T	0.04

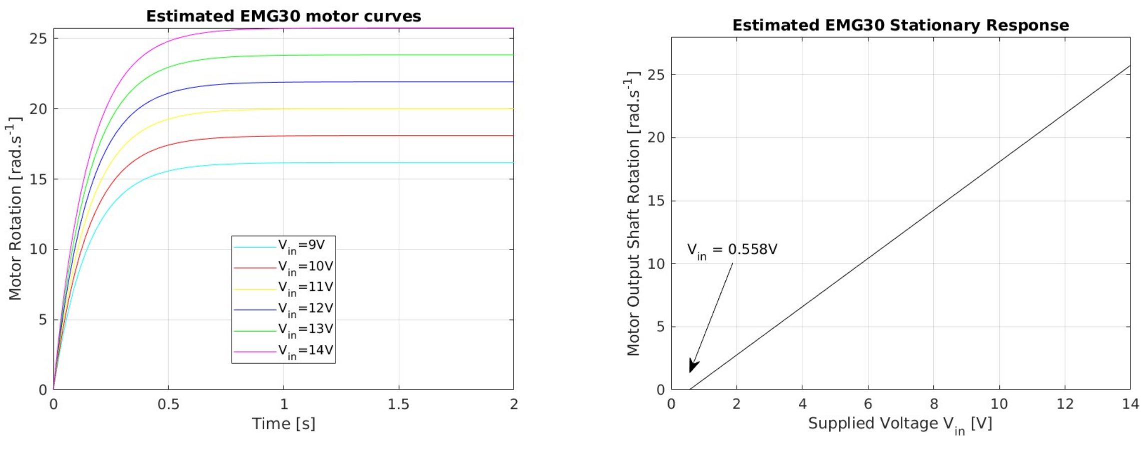
When the motor is considered fast enough so that its transient response is dispensable, leaving only its stationary response $t \longrightarrow +\infty$, (4.1) becomes:

$$\dot{\theta}(t) = 1.9158\,V_{in}(t) - 1.0691 \tag{4.2}$$

As (3.5) denounces, the interval in which the output rotation is positive is the following:

$$V_{in}(t) \geq 0.558\mathrm{V} \tag{4.3}$$

The complete estimated motor curves and the stationary response curve are shown in Figure 4.6.



(a) Motor curves (b) Stationary response

Figure 4.6: EMG30 modeling based on parameters identified in [55].

The motors will be controlled by an MD25 board, which is an 8bit controller. This means that it accepts 256 different integer digital command values, which, in turn, are mapped to converter output values to the motor/s. Running the MD25 in its default mode means that it accepts digital commands u in the following interval:

$$\{u \in \mathbb{Z}\,, u \in [-128, 127]\} \tag{4.4}$$

This means that $u = -128$ commands full reverse throttle, $u = 0$ commands the motors to be stationary, and $u = 127$ commands full forward throttle. The max throttle corresponds to feeding the motors the maximum available voltage for the MD25 controller. According to the MD25's specifications, the input voltage operational range is $V_{max} \in [9\,,\,14]\,\mathrm{V}$. This behaviour can be assigned to a nonlinear function of the form:

$$V_{in} = \begin{cases} \dfrac{V_{\max}}{127} u & u \geq 0 \\ -\dfrac{V_{\max}}{128} u & u < 0 \end{cases} \tag{4.5}$$

This information can be substituted into (4.1):

$$\begin{cases} \dot{\theta}\,(t) = 3.85\,(0.0039766\,V_{\max}\,u + 0.28404)\left(\mathrm{e}^{-0.64512\,t} - 1.0\right) & u < 0 \\ \dot{\theta}\,(t) = -3.85\,(0.0040079\,V_{\max}\,u - 0.28404)\left(\mathrm{e}^{-0.64512\,t} - 1.0\right) & u \geq 0 \\ \dot{\theta}\,(t) = 0.0 & \text{otherwise} \end{cases} \tag{4.6}$$

One of the objectives of the low-level algorithm is to receive wheel speed commands and successfully map them to digital commands that the MD25 can send to the motors. Assuming that the motor-output shaft is perfectly attached to the track mount of the robot, that is, there is no wasted rotation in the connection:

$$\omega\,(t) = \dot{\theta}\,(t) \tag{4.7}$$

The mapping between the desired wheel (or track) speed and the digital motor command can be extracted from (4.6) by rearranging it in the order of the digital command. However, this rearrangement leads to ambiguous branch limits, since the motor equation from (4.1) only considers the absolute value of motor output-shaft rotation and (4.5) considers directional rotation but only outputs positive values. To aid in the rearrangement, $\omega^c\,(t) < 0$ was assigned to the $u < 0$ branch in (4.5). To consider the directional rotation in (4.6), the signal of the $u < 0$ branch was inverted and a minimum wheel/track speed command value of $1\mathrm{rad.s}^{-1}$ was admitted to not deal with the dead-zone of the motors:

$$\begin{cases} u\,(t) = \frac{66.814\left(|\omega^c(t)| - 1.0691\,\mathrm{e}^{-6.599\,t} + 1.0691\right)}{V_{\max}(t)\,(\mathrm{e}^{-6.599\,t} - 1.0)} & \omega^c\,(t) < -1\mathrm{rad/s} \\ u\,(t) = -\frac{66.293\left(|\omega^c(t)| - 1.0691\,\mathrm{e}^{-6.599\,t} + 1.0691\right)}{V_{\max}(t)\,(\mathrm{e}^{-6.599\,t} - 1.0)} & \omega^c\,(t) > 1\mathrm{rad/s} \\ u\,(t) = 0 & \text{otherwise} \end{cases} \tag{4.8}$$

The final assumption made about the mapping between the desired speed and motor command was the dismissal of the transitory response of the motors ($t \longrightarrow +\infty$). This will introduce error; however, it significantly cheapens the calculations the ESP32 microcontroller has to perform at every command received:

$$\begin{cases} u(t) = -\dfrac{66.814\left(\left|\omega^c(t)\right| + 1.0691\right)}{V_{\max}(t)} & \omega^c(t) < -1\mathrm{rad/s} \\ u(t) = \dfrac{66.293\left(\left|\omega^c(t)\right| + 1.0691\right)}{V_{\max}(t)} & \omega^c(t) > 1\mathrm{rad/s} \\ u(t) = 0 & \text{otherwise} \end{cases} \tag{4.9}$$

After sending this digital command to the MD25, its built-in PID-like controller will ensure that the reference is followed. This means that MD25 is the component that actually closes the low-level control loop. Furthermore, the quality of the reference following will be strongly tied with the quality of this PID-like controller.

Low-Level Algorithm

The ESP32 microcontroller functions as the second brain of the TIR-ANT. It is tasked with communicating with the MD25 to get the encoder ticks and send motor commands, and it is also responsible for communication with the BNO055 and relaying its data to the Raspberry Pi. Communication between the ESP32 and the Raspberry is done through USB serial with micro-ROS encoding to allow the messages to be sent, delivered, and immediately interpreted as ROS messages.

A high level breakdown of the low-level algorithm is available in Algorithm 1. The first $0.5\mathrm{ms}$ of each loop the low-level algorithm running on the ESP32 checks for new desired wheel speed messages published by the Raspberry. If there are new messages, it immediately accepts and processes them to motor commands. If there are none or if it has already processed the new messages, then it will wait at most another $0.5\mathrm{ms}$ for the encoder ticks and the BNO055 publishing timer to end. This is a $20\mathrm{ms}$ timer. If the timer is not finished before $0.5\mathrm{ms}$ are over, the algorithm repeats the loop from the beginning. If the timer is completed within the waiting period, the algorithm can ask the MD25 for encoder data and BNO055 for its attitude estimate, angular velocity, and acceleration data and publish it immediately. This approach enables the publication rate of encoder ticks and BNO055 data to be around $50\mathrm{Hz}$ and does not enforce a particular rate for the desired wheel speed commands.

The algorithm is robust to sudden communication failure between the Raspberry Pi and the ESP32. If more than 300 loops are performed without receiving any new wheel speed commands, the algorithm will automatically order the motors to stop spinning and wait for new commands. The messages exchanged between the ESP32 and the main ROS2 system are custom messages that include status flags. If, for example, MD25 is disconnected from the I2C bus, the algorithm will publish what is essentially an error message on the encoder tick data topic. The same happens with the BNO055. The desired wheel speed message also contains a flag that can be used to ask for resetting of the encoder values.

The BNO055 message contains flags for the sensor calibration status. This sensor calibration status is available from the BNO055 library. On every startup, the device needs to be calibrated. For the calibration of the gyroscope, it is enough to leave the robot still for a few seconds. For magnetometer calibration, moving the robot at random in 6DoF is enough. The calibration of the accelerometer is what normally consumes more time. The advised method is to rotate the robot according to the 6 faces of a cube. This means making the robot point upwards/downwards, putting it on both sides, flipping it upside down, and letting it on its normal horizontal position. The calibration status of each sensor is determined using heuristics from the BNO055 library. It measures calibration at levels 0 to 3. Level 3 means fully calibrated. The BNO055 library advises not to trust the sensor when its calibration level is equal to or lower than 1.

Algorithm 1 Pseudocode for Low-level algorithm

```
Ensure: micro-ROS communication established                          ▷ Performed on startup
Ensure: MD25 communication established
Ensure: BNO055 communication established
Ensure: publishing_timer is finished every $20\mathrm{ms}$
  function LOOP                                                      ▷ Loop until powered down
    for at most $0.5\mathrm{ms}$ do
      if new_wheel_speed_commands then                               ▷ Check for new message commands
        $commands \leftarrow$ Equation 4.9
        send $commands$ to MD25
        if encoder_reset_asked then
          reset encoders
        end if
        return
      end if
    end for
    for at most $0.5\mathrm{ms}$ do
      if publishing_timer_finished then                              ▷ Check if it's time to publish data
        get BNO055 data
        get encoder data
        publish BNO055 data
        publish encoder data
        return
      end if
    end for
    if cycles_no_commands $> 300$ then                              ▷ Stop robot in case of communication failure
      stop motors
    end if
    if MD25_disconnected or bno055_disconnected then                 ▷ Warn that something is wrong
      publish error message
    end if
  end function
```

4.2.2 Odometry Node

The TIR-ANT is a tracked-mobile robot. It is not a differential-drive robot. It has much more ground contact than a normal wheeled differential-drive. Due to the discontinuous nature of its tracks, the motion is not as smooth as a wheeled one. Tracks are very prone to slippage. However, there are still similarities in kinematics. The TIR-ANT's velocity is directly proportional to the sum of each track's spin, and the robot is able to turn/rotate by having its tracks spin in opposite directions or in the same direction at different speeds. Applying a differential-drive kinematicS model to the TIR-ANT is applicable; however, bad results are to be expected if no further effort is made to improve the model.

The main approximation done in this work is to apply a differential-drive model to the TIR-ANT for odometry and pose estimation purposes. The model is then calibrated using the UMBmark method proposed by Borenstein et al. [44]. Essentially, this calibration will find the closest differential-drive properties that model the TIR-ANT.

For the TIR-ANT's wheelbase, the perpendicular distance between the centre of each side's tracks is considered. For the supposed radius of the drive wheels, the distance between the centre of the track mount and the ground was considered. From now on, when mentioning the "wheels" of the TIR-ANT, what is meant is virtual wheels coincident with the robot's two front track mounts with the previously explained radius. The main kinematics properties measured for the nominal differential-drive model of the TIR-ANT can be seen in Table 4.3.

Table 4.3: Measured kinematics properties for the TIR-ANT.

Property	**Value**
Wheelbase b	0.225m
Approximate Wheel Radius r_R , r_L	0.0355m
Same Side Wheel Distance	0.15m
Track Width	0.03m

The high-level breakdown of the odometry algorithm is made available in Algorithm 2. The odometry algorithm will apply the model (2.20) - (2.21) with the kinematics properties of Table 4.3 and E_b and E_d calculated experimentally. Before calibration, these values are considered unity.

The odometry algorithm waits to receive encoder tick data from the low-level node, at every new message, it processes, and publishes the results immediately. Calculate the increment/decrement in each encoder and use that to calculate the pose estimate at that time step. Every message published by the odometry node includes the pose estimate and the covariance associated with these estimates. These covariance values were experimentally tuned to obtain the better EKF estimation results.

Algorithm 2 Pseudocode for the odometry node

```
Ensure: ROS communication established                 ▷ Performed on startup
Ensure: all parameters loaded        ▷ Including the covariance associated with odometry
  function LOOP                                        ▷ Loop until stopped
      if encoder_ticks_received then                   ▷ New data available
          compute $\Delta n_R$ and $\Delta n_L$
          compute $\Delta s_R$ and $\Delta s_L$ from (2.14)
          compute $\Delta s$ and $\Delta \varphi$ from (2.20) and (2.21)
          compute odometry pose estimate from (2.17)
          publish odometry pose estimate + associated covariance $\mathbf{R}^{\mathbf{odom}}$
      end if
  end function
```

The odometry algorithm was programmed to take advantage of ROS2's parameter server. This means that the user can change most of the variable values without having to compile the code for every change. When changing parameter values, the only required step is to source the respective launch file before running the algorithm.

4.2.3 Sensor Drivers

For the Hokuyo rangefinder to work directly with the ROS2 framework, a wrapper is needed. A wrapper is the translation of driver software to ROS/ROS2 compatible code. The wrapper used is the *urg_node* library[3] and its dependencies. The wrapper and the hardware itself worked as intended with the default parameters. This wrapper enables the Hokuyo rangefinder to publish the laser scan directly as an appropriate ROS message that can be used by SLAM algorithms without extra steps.

The Intel Realsense camera wrapper used was the *realsense_ros* library[4]. Unlike the rangefinder, several parameter tuning iterations were needed for the camera to work as intended. The main problem faced was frame droppage. The solution was to drop both the stereo module and RGB image resolution and to lower both sensor frame rates. To further simplify the data so that the limited computing power of the Raspberry Pi could handle image processing while also performing all the other necessary calculations for the robot to work, filters were applied. The summary of the depth camera configuration adopted can be seen in Table 4.4.

Table 4.4: Intel Realsense extra configuration to avoid frame drops by the Raspberry Pi.

	RGB Sensor	**Stereo Module**
Frame-rate	15fps	
Resolution	640×480	
Applied Filters	Disparity Filter	Disparity Filter Hole-Filling Filter

The TIR-ANT is commanded through body-frame velocity commands, which is standard practice in ROS/ROS2 applications. The only requirement to control the TIR-ANT is to be able to publish messages on the necessary topic. For most of the work, a PlayStation 3 controller was used to send these commands. However, the robot can also be commanded with a regular keyboard or any other controller, provided that the necessary wrapper is installed.

The wrapper installed for the PlayStation 3 controller is from the *teleop_twist_joy* library[5]. This algorithm publishes the state of each of the joysticks at 20Hz. The state of a joystick is the scale in the $[-1\,,\,1]$ interval of how much the joystick is away from its rest position. This means that a 1 represents the joystick being completely forward and -1 completely backwards.

The state of the left joystick controls the desired body frame linear velocity from (2.6). The state of the right joystick controls the desired body-frame yaw velocity from (2.7). The $command\_interpreter$ node performs these calculations after receiving the state of the joysticks.

The mapping between the state of the joystick and the desired velocities is as follows: the maximum displacement of the left joystick equates to a desired linear velocity of 0.5m/s; the maximum displacement of the right joystick equates to a desired yaw velocity of 1.5rad/s. These values are user-configurable.

The user is also free to launch a node that enables force feedback from the controller. That is, if the Hokuyo rangefinder detects an object closer to the user-defined limit, the controller vibrates.

[3]https://github.com/ros-drivers/urg_node/tree/ros2-devel
[4]https://github.com/IntelRealSense/realsense-ros/tree/ros2
[5]https://github.com/ros2/teleop_twist_joy/tree/humble

4.2.4 Robot Localisation Package

The *robot_localization*[6] package's EKF comes with default values that should work in most applications without much tuning effort.

The EKF was made to publish messages at $45\mathrm{Hz}$ with a queue limit of 10 messages for both the odometry and IMU messages. This means that the algorithm will keep at most 10 messages of each type waiting to be processed if there is a processing held-up.

The option for 2D mode was enabled. This means that the EKF will only keep track of the following state vector:

$$\mathbf{X} = [x \; y \; \varphi \; \dot{x} \; \dot{y} \; \dot{\varphi} \;]^T \tag{4.10}$$

The EKF keeps track of the $[x\,,\, y\,,\, \varphi]^T$ estimates of *odom_node* and its derivative. From the IMU data, it keeps track of the yaw φ estimate, the angular velocity on the z -axis $\dot{\varphi}$ and the acceleration on the x -axis $\ddot{x}$. The acceleration measurement is internally integrated for use as an extra data source for the $\dot{x}$ state estimate.

The *robot_localization* package's EKF offers the option for a dynamic process noise covariance matrix. It is a boolean flag that in the case of this application was changed to true. This essentially means that the algorithm will not increase the pose covariance values when the vehicle is not moving.

A noteworthy aspect of this algorithm is that its gravitational acceleration value can be changed. This is useful when dealing with acceleration data from faulty accelerometers or when operating at extreme altitudes. The BNO055 tends to not measure exactly the expected $9.81\mathrm{m/s^2}$. It varies from start-up to start-up. During testing procedures, the measured gravitational acceleration was as low as $9.67\mathrm{m/s^2}$. For optimal EKF performance, it is advised to change this value before launching the mission according to what the BNO055 is currently reading.

The final process noise covariance matrix $\mathbf{Q}$ for the state vector in (4.10) implemented that seemed to work best and provide better results is:

$$\mathbf{Q} = \mathrm{diag}\left(\begin{bmatrix}0.075\mathrm{m^2} & 0.075\mathrm{m^2} & 0.09\mathrm{rad^2} & 0.025\mathrm{m^2/s^2} & 0.025\mathrm{m^2/s^2} & 0.02\mathrm{rad^2/s^2}\end{bmatrix}\right) \tag{4.11}$$

where the diag operator means that these are the values of the diagonal of the matrix, all other values are null.

The covariance matrix associated with the IMU measurements was determined by leaving the BNO055 still in its normal configuration on top of the robot for some time and capturing its readings. Since the expected true value should be null in every sensor reading except the accelerometer readings that should be reading the gravitational acceleration the sensor is subject to, the measurement covariance matrix for the sensor readings can be determined from the recorded data. The implemented EKF only tracks the IMU measurements for $[\varphi\,,\, \dot{\varphi}\,,\, \ddot{x}]^T$ states and the covariance matrix of the IMU measurements determined for this state vector is:

$$\mathbf{R^{bno}} = \mathrm{diag}\left(\begin{bmatrix}0.001507\mathrm{rad^2} & 0.007454\mathrm{rad^2/s^2} & 0.01916\mathrm{m^2/s^4}\end{bmatrix}\right) \tag{4.12}$$

[6]`https://github.com/cra-ros-pkg/robot_localization`

The BNO055 sensor measurement covariance matrix was approximated to a diagonal matrix; therefore, only the variance of each of the relevant state's readings was taken into account. The *imu_interpreter* node that publishes the IMU data in the correct ROS-message format with its associated covariance matrix $\mathbf{R^{bno}}$ is made aware of the calibration level heuristics of the sensor. When the calibration level of the BNO055 lowers in the middle of a run, this node will double the values of $\mathbf{R^{bno}}$ to signal to the EKF that this sensor is not be as trusted as before.

For the covariance matrix associated with the odometry estimate, a tuning process was necessary. The EKF keeps track of the odometry's $[x\,,\,y\,,\,\varphi]$ state estimates and its derivatives. However, the covariance of the derivative states are determined internally based on the normal state's associated covariance matrix, therefore, the covariance matrix associated with the odometry estimate $\mathbf{R^{odom}}$ only needs to account for the $[x\,,\,y\,,\,\varphi]$ state estimates:

$$\mathbf{R^{odom}} = \mathrm{diag}\left(\begin{bmatrix}0.04\mathrm{m}^2 & 0.09\mathrm{m}^2 & 0.007615\mathrm{rad}^2\end{bmatrix}\right) \tag{4.13}$$

Again, the odometry estimate covariance matrix was assumed to be a diagonal matrix, and therefore only the variance of the relevant state estimates is taken into account.

A more detailed explanation and access to the EKF's launch files are made available in the TIR-ANT's Github repository [54].

4.2.5 SLAM Toolbox Package

As with the *robot_localisation* algorithm, the *slam_toolbox* package comes with default values for its parameters that were optimised for use in most applications.

The values not left as default were the following: the algorithm frequency was made $5\mathrm{Hz}$; the resolution of the 2D occupancy map to generate was made $6\mathrm{cm}$, this means that the cells on the generated grid maps will be $6\mathrm{cm}$ sided squares; the algorithm will only consider scan data at most $4.5\mathrm{m}$ away to avoid the noisiest section of the Hokuyo rangefinder's measurements; the minimum distance of travel before processing a new scan was changed to $2\mathrm{cm}$; the minimum change in heading to justify an update was changed to $0.015\mathrm{rad}$.

The algorithm was also made to keep in buffer - fastest access memory - the latest 3 laser scans for immediate localisation purposes and map modifications. These scans in buffer are the candidates for the package's scan matching algorithm to refine the odometric pose estimate. If a matched scan differs from the odometric pose estimate, a penalty of 0.5 for the distance variance and 0.9 for the angular variance will be applied to it.

A more detailed overview of the *slam_toolbox*'s implementation on the developed software and pratical considerations on the effects of changing certain parameters are made available in TIR-ANT's Github repository [54].

4.2.6 RTAB-Map Package

The *rtabmap_ros* package offers more than 200 configurable parameters to the user, although the default values are usually acceptable for most applications. This algorithm is considerably more computationally intensive than the *slam_toolbox* one. The main focus in tuning the *rtabmap_ros* algorithms was to make them run at a steady rate on the Raspberry Pi 4. Therefore, most of the tuning was done by limiting confidence levels and choosing the least computationally demanding option.

The Hokuyo rangefinder scans and the data from the Intel Realsense camera are used by the algorithm. The main source of data is actually the depth camera, since the RTAB-Map is a visual SLAM type. However, the laser scans are used as an extra source of data from the environment to refine the map.

A frame-to-frame visual odometry strategy that integrates optical flow to refine the odometric pose estimate was adopted. The number of maximum optimisation iterations was reduced to 20, the limit for the number of features extracted from a single image was reduced to 500. The minimum travel to justify an update is $0.1\mathrm{m}$ or $0.1\mathrm{rad}$ rotation. The SLAM algorithm was made to publish its calculations at $5\mathrm{Hz}$.

Most of the other parameters were left as default. An in-depth explanation of the changed parameters and practical considerations of the effect of changing certain parameters is available at the TIR-ANT's Github repository [54].

4.2.7 Simulator

A Gazebo-based physics simulator was developed to test the functionality of the algorithms developed. It simulates a simplified 3D model of the TIR-ANT robot without tracks but, instead, four wheels placed where the track mounts should be and the kinematic properties of Table 4.3 as illustrated in Figure 4.7. This 3D model can be coded into the simulator using URDF and XACRO commands [56]. The simulator is also capable of simulating sensors. It simulates depth camera data, 2D rangefinder data, and IMU readings. The user can adjust how noisy these data should be. Encoder data are simulated by knowing exactly the state of each of the wheels in simulation and converting that wheel angular position into what an encoder should read. Again, the user can easily add noise or faulty readings to this data stream. Another useful feature of Gazebo-based simulators is the fact that world files are very easily loaded without much concern for compatibility. This means that as long as a world can be modelled using any 3D CAD software that can output `.sdf` files, then it can be loaded into the simulator for the simulated TIR-ANT to explore. There are also several open source world files available on the Internet[7]. An example of the simulator running on one of these worlds can be seen in Figure 4.8. On the left of this figure is the simulated robot interacting with the virtual world. On the right are the simulated sensor data. The point cloud from the simulated depth camera is coloured with regard to the point's distance in the longitudinal axis of the robot; the thicker red points are the simulated laser scan data.

No work was done to validate the simulator or bring it closer to reality. For the simulator to be accurate, work needs to be done to model the friction between the tracks/wheels and the ground and also the dynamics properties of the robot, like each component's mass and inertial properties for collision

[7] `https://app.gazebosim.org/`

simulation. The simulator was mainly used to demonstrate the functionality of the code developed before applying it in real hardware.

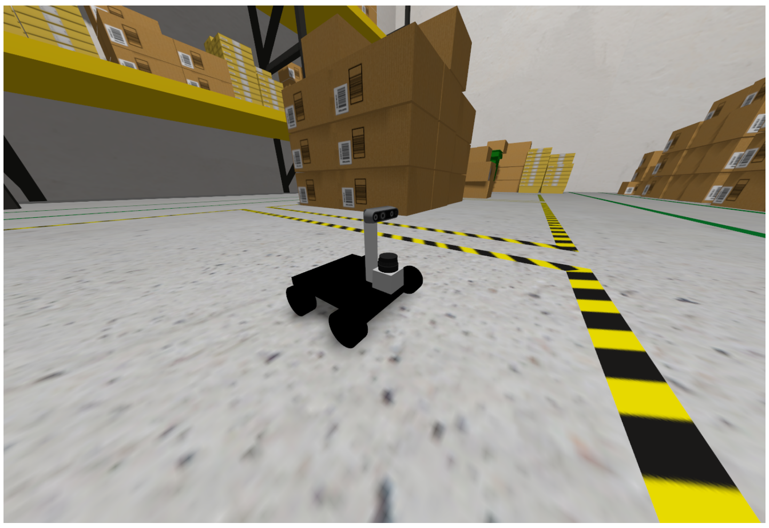
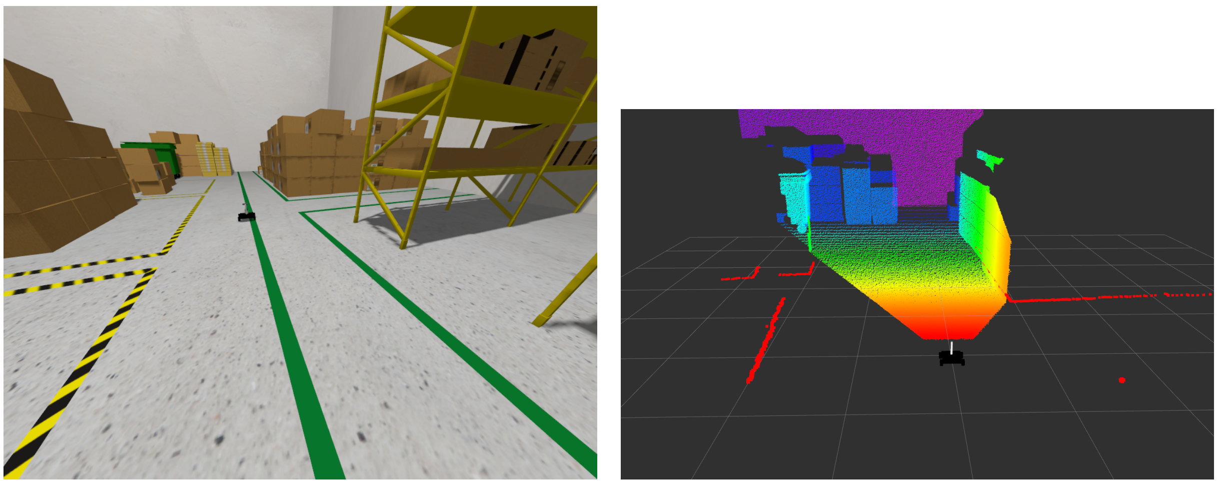


Figure 4.7: Simulated TIR-ANT robot model in the Gazebo-based simulator.


(a) Gazebo-based simulator

(b) RVIZ2 visualisation tool

Figure 4.8: A scene from the developed simulator.

Chapter 5

Experimental Results

This chapter focuses on showing the results obtained for the validation of the low-level control (Section 5.1), the odometry calibration (Section 5.2) and the validation of the EKF (Section 5.3). The results of both SLAM algorithms deployed are also shown in Section 5.4.

For the validation or performance measurement of the developed and implemented algorithms, a ground truth reference is necessary. For this reason, most of the experimental trials were conducted in the Robotics Arena of the Laboratório de Controlo, Automação e Robótica of the Department of Mechanical Engineering of Instituto Superior Técnico, Figure 5.1, which is equipped with a 3D infrared tracking system. The arena's motion capture system tracks the motion of reflective markers in real time within its $12\,(L) \times 4\,(W) \times 4\,(H)\,\mathrm{m}^3$ volume. The motion of a body can be tracked by associating a unique configuration of markers with it. This motion capture system is accurate enough (the maximum expected absolute error is $5\mathrm{mm}$) to be used as reference. The arena's motion capture system was used to track the robot motion and to determine the placement of objects when testing mapping capabilities of the SLAM algorithms.

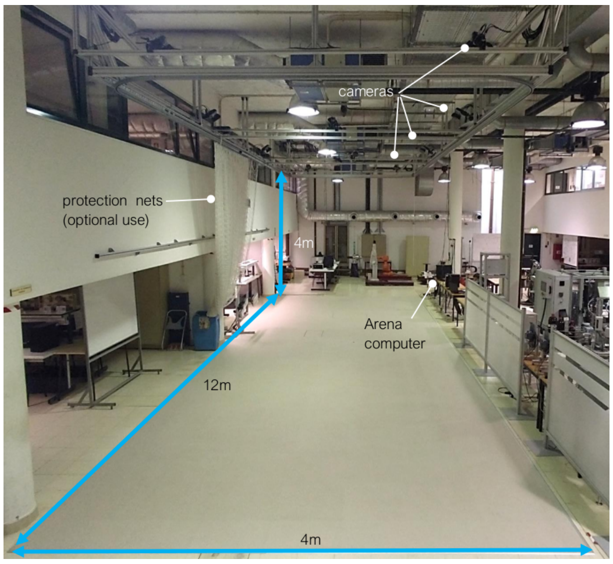
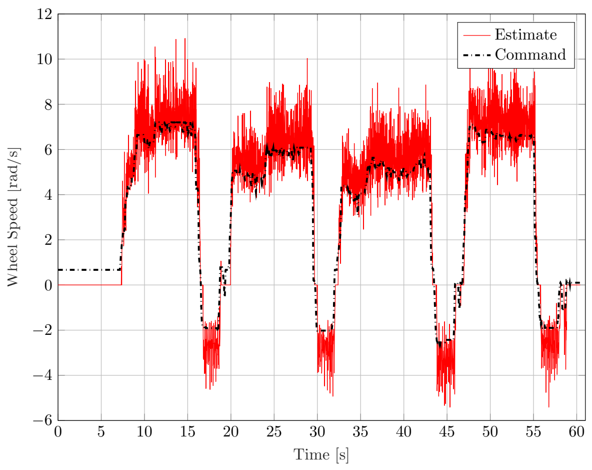


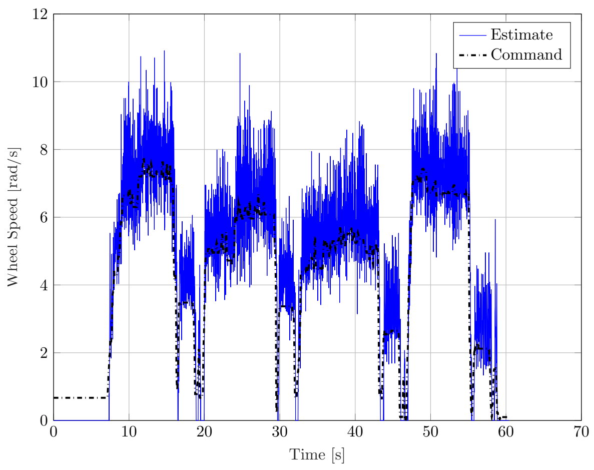
Figure 5.1: Robotics Arena of the Laboratório de Controlo, Automação e Robótica of the Department of Mechanical Engineering of Instituto Superior Técnico.

5.1 Low-Level Validation

The validation of the low-level control consists of comparing the wheel speed commands sent and the resulting wheel speed estimate using the encoder ticks. The typical results of this comparison when performing the square trajectory of the UMBmark can be seen in Figure 5.2. The wheel speed estimation was done through numerical differentiation, which introduces noise.



(a) Left Motor Response



(b) Right Motor response

Figure 5.2: Comparison between wheel speed command and estimated wheel speed obtained.

Some key takeaways:

- Despite the estimation of the wheels' speeds introducing noise, it is clear that the low-level control works as intended;
- The minimum velocity requirement is working as intended as made clear from the beginning of the command signal;
- The approximation made by not taking into account the motor's transient response indicates reasonable. The estimated wheel speeds follow the trend in the command with very little delay.

5.2 Odometry Calibration

For the odometry calibration the described UMBmark method (Section 2.1.4) was applied. For this, the robot was made to follow a square trajectory as close as possible with a $1.9\mathrm{m}$ side in both clockwise and counter-clockwise directions. To ensure repeatability, the intended path was marked on the arena's floor - Figure 5.3. The robot's trajectory was motion-captured.

Performing the UMBmark trajectory and using the uncalibrated odometry model from (2.17), the typical trajectory estimates obtained compared to the actual trajectory performed can be seen in Figure 5.4.

The main points that can be drawn from this comparison are:

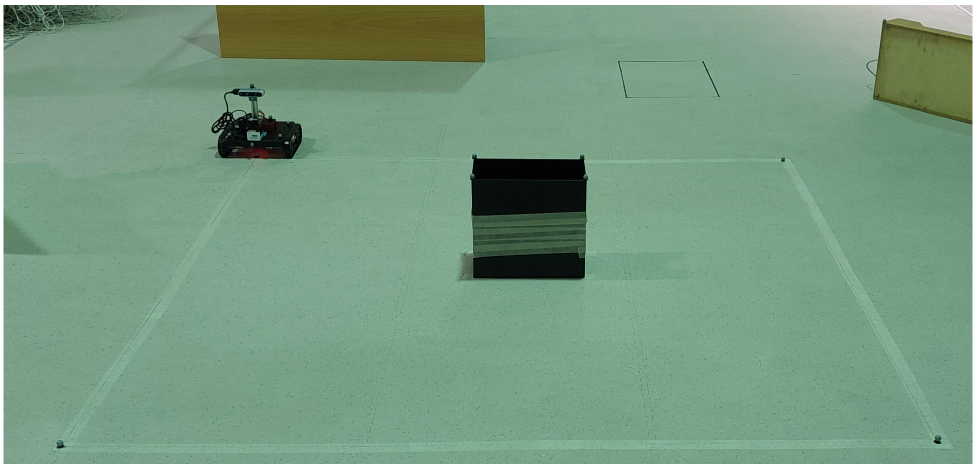
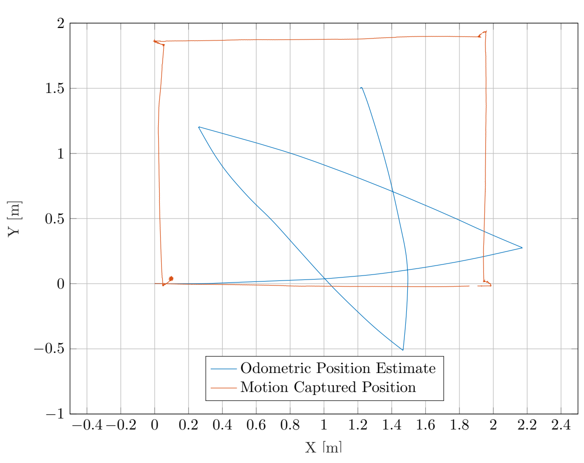
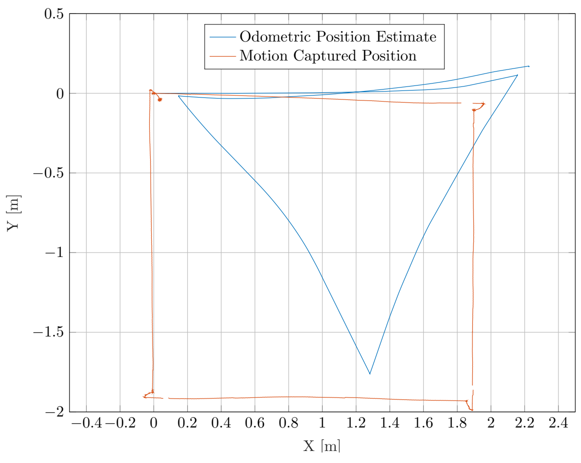


Figure 5.3: UMBmark's square trajectory marked on the floor of the arena..



(a) Counter-Clockwise



(b) Clockwise

Figure 5.4: Comparison between actual trajectory followed and uncalibrated odometry estimation performing the UMBmark square.

- The odometry model overestimates the actual amount of rotation performed. This means that there is significant slippage. The amount of encoder ticks necessary for the TIR-ANT to perform a $90°$ turn would make an approximate differential-drive robot with the kinematics properties of Table 4.3 turn between $115°$ and $150°$;
- The counterclockwise and clockwise turning behaviours vary significantly.

After performing several counter-clockwise and clockwise runs of the UMBmark trajectory and recording the uncalibrated odometry end position estimate - in Figure 5.5. Equations (2.22)-(2.26) can be applied to determine the relevant calibration coefficients - available in Table 5.1. Having calculated these coefficients, the encoder tick data recorded from these runs can be replayed and the odometry recalculated now using the new calibrated model - results also in Figure 5.5.

Table 5.1: UMBmark results.

$\alpha\,[°]$	$\beta\,[°]$	E_d	E_b	E_{max}
27.997	-8.028	0.984	1.452	2.227

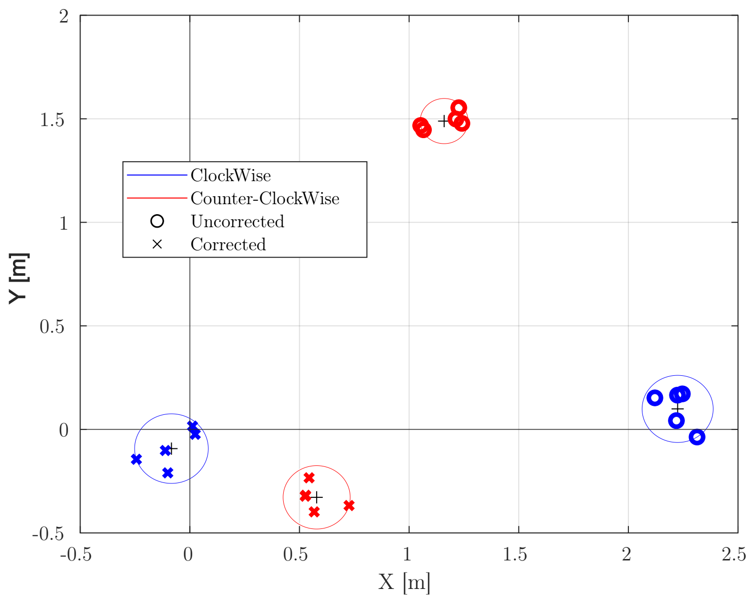
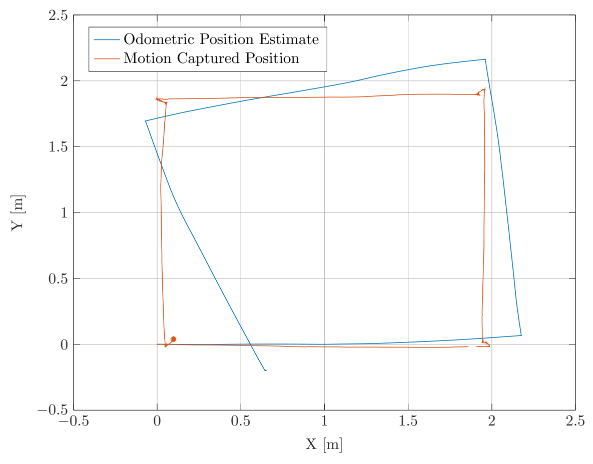


Figure 5.5: UMBmark result comparison between corrected and uncorrected odometry.

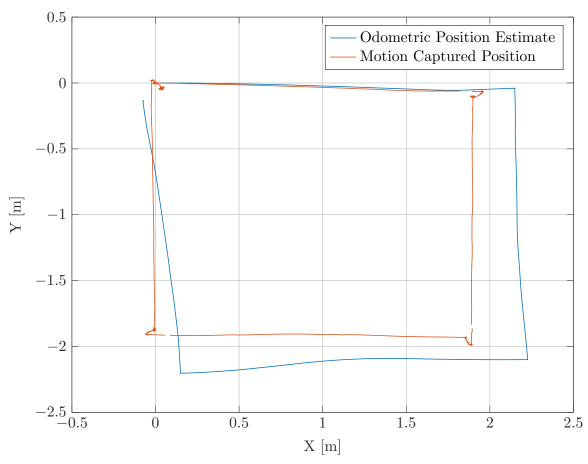
The main takeaways from this analysis:

- The uncalibrated clockwise runs showed the most variance between themselves and the worst overall error when compared with the uncalibrated counter-clockwise runs;
- The fact that $E_b = 1.452$ means that the uncalibrated model overestimated the amount of rotation by as much as 50%;
- $E_d < 1$ means that radius of the left wheel must be greater than the radius of the right wheel on the model used in software to get accurate trajectory estimation;
- the calibration worked. Calibration brought both clusters closer to the origin. The worst performing direction after calibration is now the counter-clockwise one. Again, the different nature of the odometry error between clockwise and counter-clockwise runs remains noticeable;
- The maximum error E_{max} reduced from 2.227m to 0.665m, which is a 3.349-fold improvement.

Typical clockwise and counter-clockwise calibrated trajectory estimates compared with the actual trajectory can be seen in Figure 5.6. The estimated trajectories are much closer to reality. However, there seems to be an overestimation of the actual distance travelled. This is again due to slippage of the tracks on the floor. There is wasted rotation that does not result in movement and the odometry model does not expect this. The counter-clockwise calibrated estimation is significantly worse than in clockwise runs. This might be due to the tracks not being equally stretched on assembly, which enables different ground contact behaviour from each track. An attempt was made to further calibrate the odometry to correct this discrepancy in the run direction and to reduce the overestimation of the distance travelled - discussed in Appendix A. Ultimately, the results of the further calibration attempt did not improve the base calibration reliably and its results were not implemented.



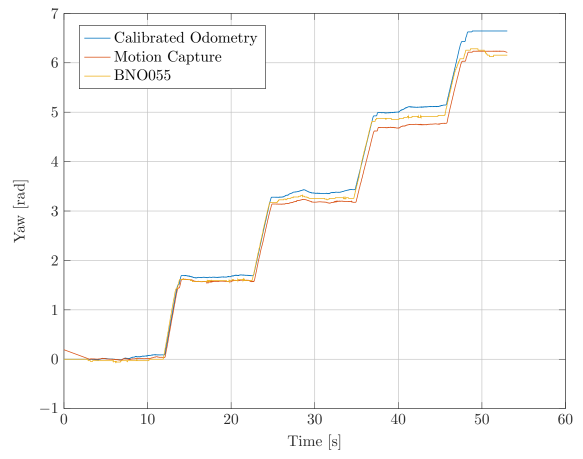
(a) Counter-Clockwise



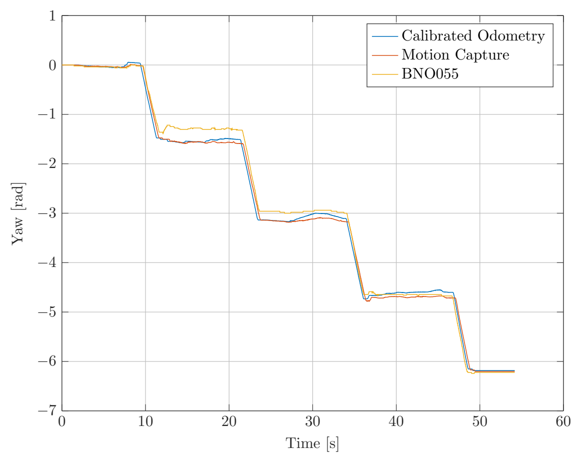
(b) Clockwise

Figure 5.6: Comparison between actual trajectory followed and calibrated odometry estimation performing the UMBmark square.

The comparison between the yaw estimation from calibrated odometry, the BNO055 and the motion-captured yaw is available in Figure 5.7.



(a) Counter-Clockwise



(b) Clockwise

Figure 5.7: Comparison between actual and estimated yaw from both the calibrated odometry and the BNO055.

The main points that can be drawn from the comparison are:

- The odometry yaw estimation gets worse incrementally, as expected;
- The calibrated odometry yaw estimation is excelent in clockwise runs;
- The BNO055 estimates the orientation accurately without the incremental worsening the odometry is subject to.

5.3 EKF Validation

Having the odometry calibrated and the BNO055 yaw estimation validated, means that the EKF can confidently fuse them into a more robust pose estimate. The validation procedure followed was the same as for the odometry calibration. Several UMBmark clockwise and counter-clockwise runs were performed and the EKF end-point estimate was recorded. The comparison between the endpoint estimate between the EKF and the calibrated odometry is available in Figure 5.8. Typical clockwise and counter-clockwise run trajectory EKF estimates are available in Figure 5.9. The comparison between the EKF estimate and the motion captured trajectory of a more complex trajectory is available in Figure 5.10.

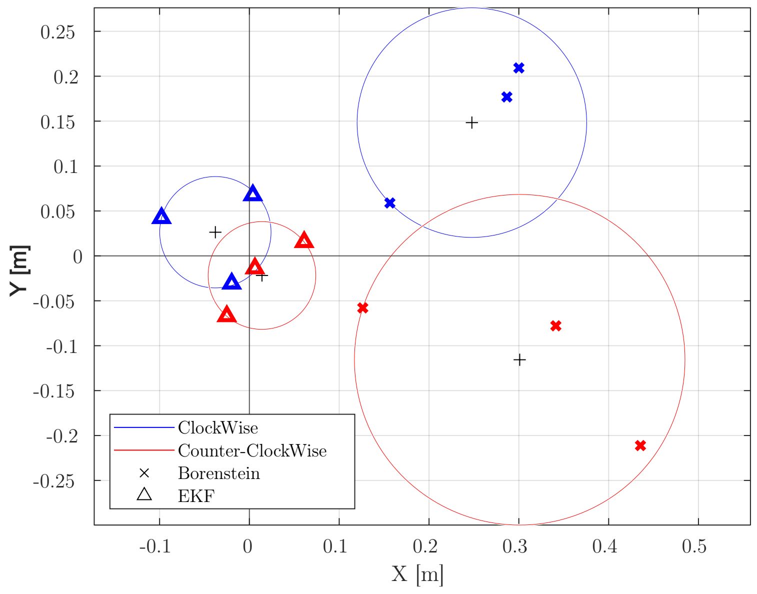
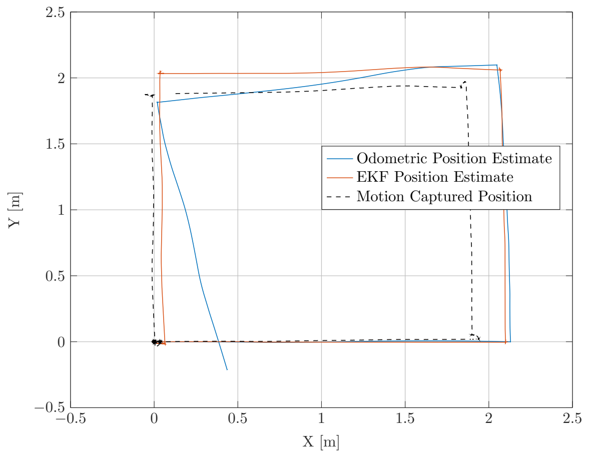
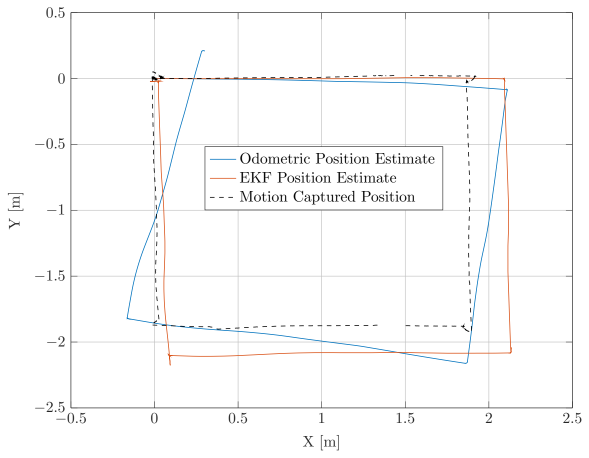


Figure 5.8: UMBmark result comparison between corrected odometry and the EKF pose estimate.



(a) Counter-Clockwise



(b) Clockwise

Figure 5.9: Comparison between actual trajectory followed, calibrated odometry and EKF estimate performing the UMBmark square.

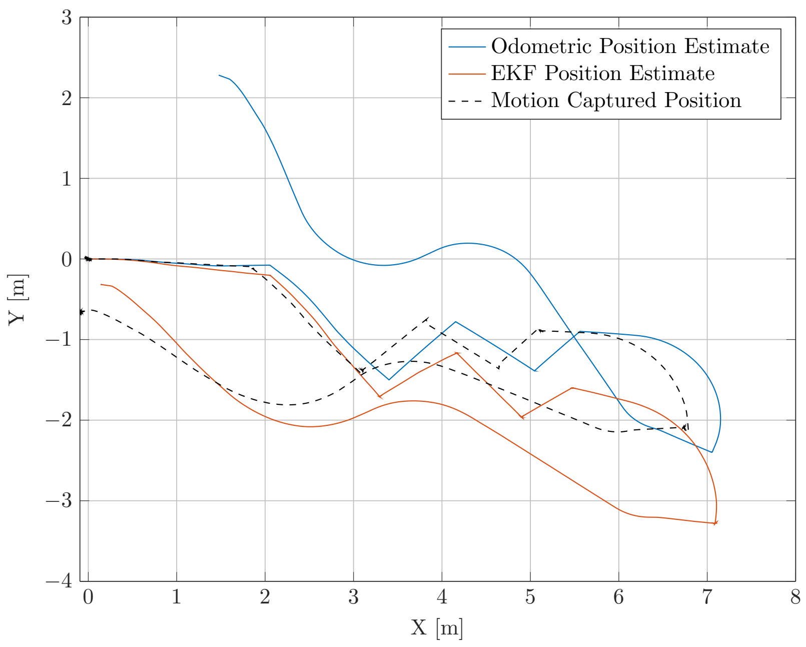


Figure 5.10: Comparison between actual trajectory followed, calibrated odometry and EKF estimate performing a complex trajecotry.

The main takeaways from the comparison are the following:

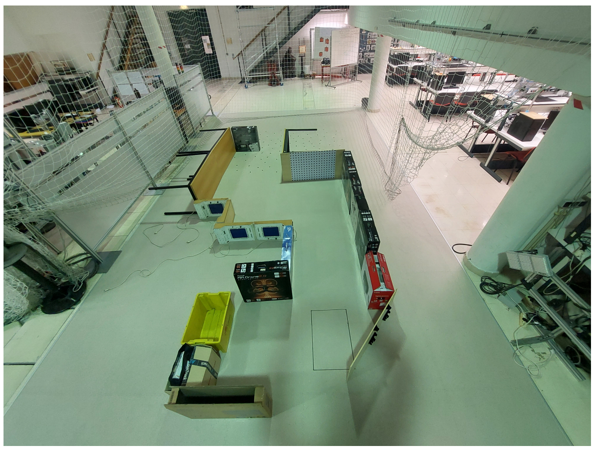
- The EKF is working and improving on odometry estimation;
- E_{max} went from 0.184m from odometry estimation to 0.062m, reflecting a significant improvement;
- Regarding the square trajectory, the EKF's estimate of rotation is a major improvement on the odometry's. However, there is still some overestimation of the actual distance travelled;
- Regarding the complex trajectory, which spans roughly 20m in length, the EKF estimated the end point of the trajectory about 0.4m away from the actual end point, which is a 2% deviation in a trajectory that spans roughly 20m in length.

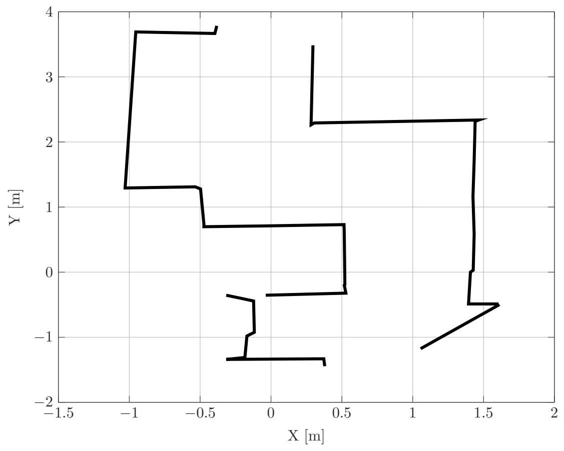
5.4 Experimental SLAM Validation

For the validation of the SLAM algorithm, several obstacles were placed in the arena and the robot was made to explore it. The motion capture camera system was used to measure the obstacle location - Figure 5.11. The cameras were also used to capture the robot's motion for localisation performance comparison. Both the *slam_toolbox* algorithm and the *rtabmap_ros* algorithms were tested on this map.

5.4.1 SLAM Toolbox Algorithm

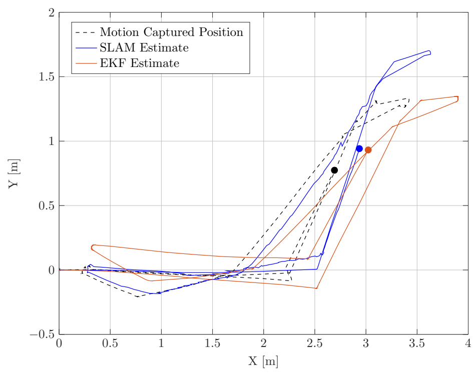
The comparison between the obtained estimate of the followed trajectory and the actual followed trajectory and the comparison between the estimated map and the actual expected map is made in Figure 5.12. For a better idea of the motion captured data and the full *slam_toolbox* estimate, Figure 5.13 puts them side by side.


(a) Map.

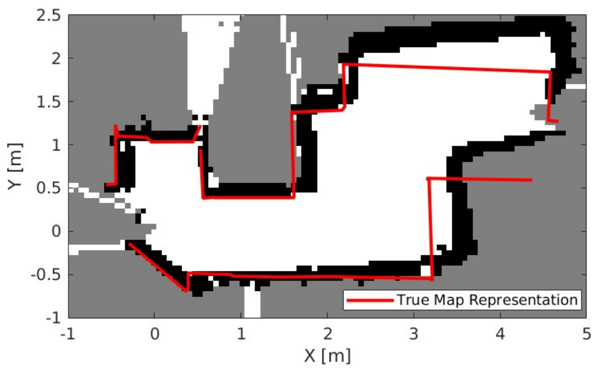


(b) 2D Representation.

Figure 5.11: Map built and its 2D representation.. Obstacle thickness not taken into account unless it was necessary for map's skeleton geometry.

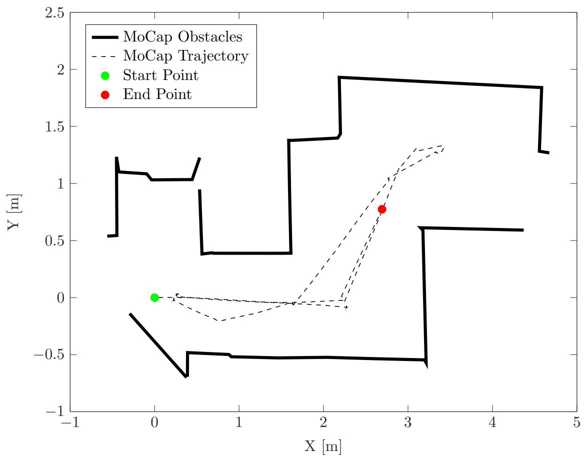


(a) Trajectory

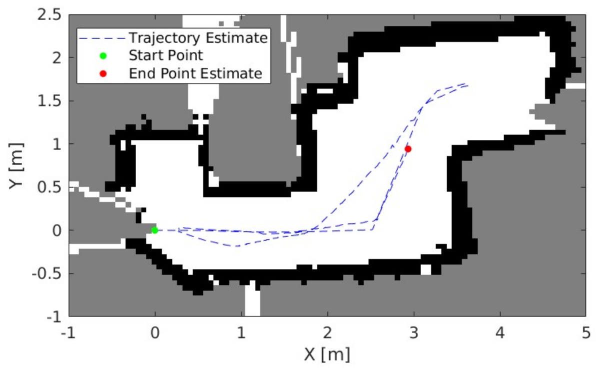


(b) Map

Figure 5.12: Comparison between *slam_toolbox*'s estimation and motion-captured data.



(a) Motion Capture



(b) *slam_toolbox* estimate

Figure 5.13: Comparison between motion-captured data and the *slam_toolbox* estimate.

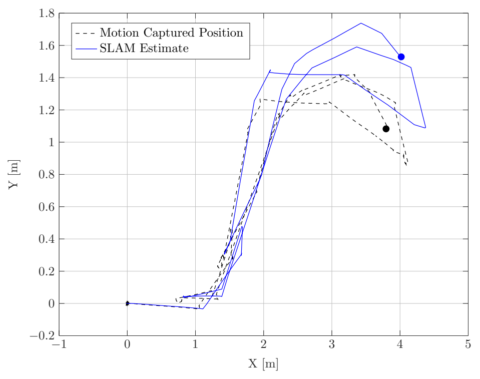
The conclusions that can be drawn are the following:

- The $slam\_toolbox$ algorithm works as intended;
- The estimated map's geometry is congruent with the real geometry;
- Due to the noisy nature of the Hokuyo rangefinder and the significant shakiness of the entire robot when moving, the SLAM algorithm is uncertain of where to fill cells that represent a simple line. This is the reason why the ”walls” of the estimated map are so thick. The noisier the scan data, the thicker the lines in the estimated map;
- Despite the noise, the SLAM algorithm was able to successfully identify the openings in the map;
- With regard to the overlay of the true and estimated maps, it is clear that the map optimisation performed misplaced some of the boundaries. The map estimate on its $x \longrightarrow +\infty$ side is stretched and distorted from reality. This is also noticeable in the trajectory estimate when the robot reaches the map edge and turns back. The system correctly detected loop-closure, however, the optimisation of the map and trajectory performed resulted in some discrepancy with the truth. This could be due to the fact that the EKF estimate at the time of this particular loop-closure significantly overestimated the X position of the robot;
- Regarding the pose estimate, it is clear that the SLAM trajectory estimation overinflated the actual distance travelled. This makes sense with the analysis of the obtained map. However, it still performed better overall than the EKF estimate. Especially on the left side of the map after looping around. This is also seen in the fact that the left side of the map estimate is very similar to the actual left side of the real obstacle layout.

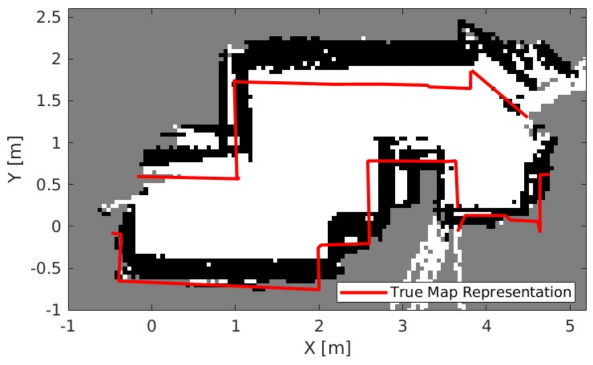
5.4.2 RTAB-Map Algorithm

The comparison between the obtained estimate of the followed trajectory and the actual followed trajectory and the comparison between the estimated map and the actual expected map is made in Figure 5.14. For a better idea of the motion captured data and the full *rtabmap_ros* estimate, Figure 5.15 puts them side by side. This 2D map estimate is the lamination of the estimated 3D map at the level of the Intel RealSense camera. The top view of the estimated 3D map is also available in Figure 5.15. The estimated 3D map is shown in Figure 5.16. The main conclusions that can be drawn are:

- The $rtabmap\_ros$ algorithm works. However, not as well as the $slam\_toolbox$ one;
- The generated maps are considerably noisy. Despite the main geometric properties of the true map still being present, there is wrong geometry on the estimated map;
- The high amount of shaking during the tracked motion seems to damage camera performance. Sensor data had already been significantly downgraded to allow fast processing (Table 4.4). The image resolution being so low works together with the shaking to disturb the loop closure and data association of the algorithm, which leads to false loop closure rejections. False loop closure

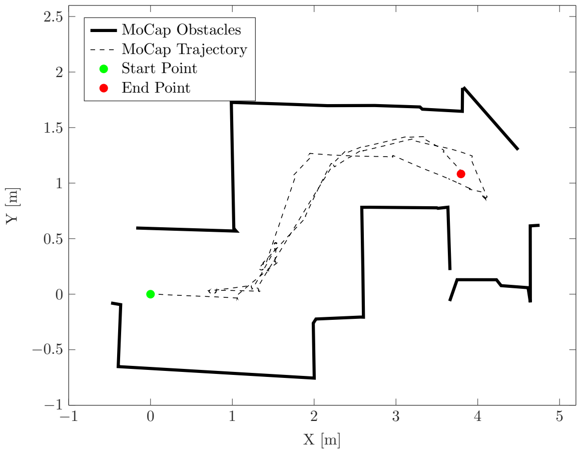


(a) Trajectory

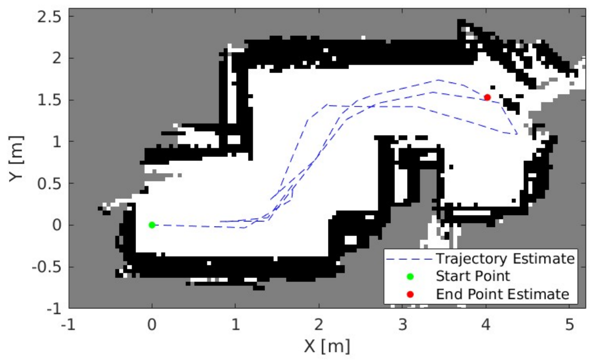


(b) Map

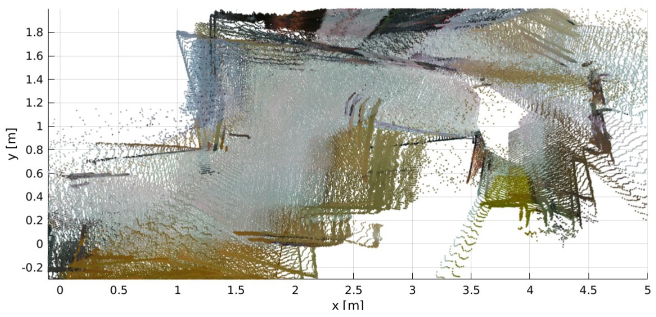
Figure 5.14: Comparison between *rtabmap_ros*'s estimation and motion-captured data



(a) Motion Capture



(b) *rtabmap_ros* trajectory and grid map estimate



(c) Top view of *rtabmap_ros* 3D map estimate

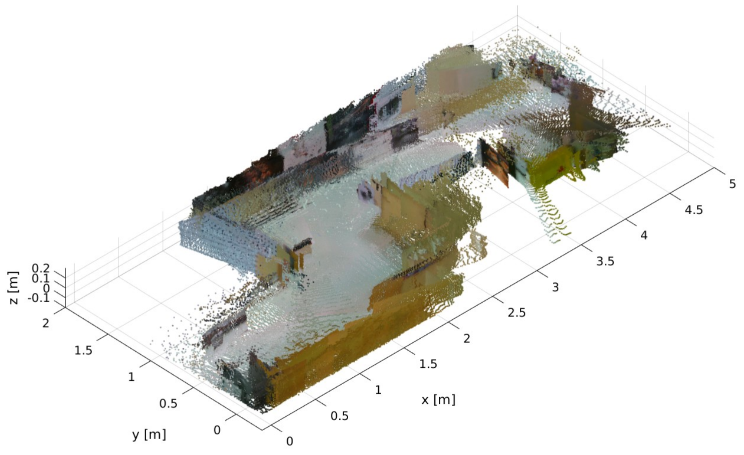
Figure 5.15: Comparison between motion-captured data, the *rtabmap_ros*'s grid map and trajectory estimation and the top view of the 3D map estimate.

rejections lead to what can be seen in Figure 5.15's top view of the 3D map: there are two of the same wall. If a stronger map optimisation was performed, these wall observations would be associated to the same one;

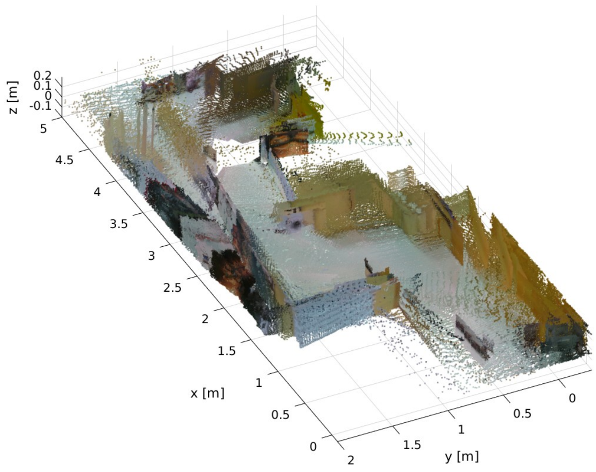
- The trajectory estimate significantly overestimated the distance travelled, especially in the upper part of the map;

- Work is to be done to filter out the vibration effect on the exteroceptive sensors. Work also is to be done on optimising the software codebase to allow a higher resolution or higher frame rate for the depth camera data stream;

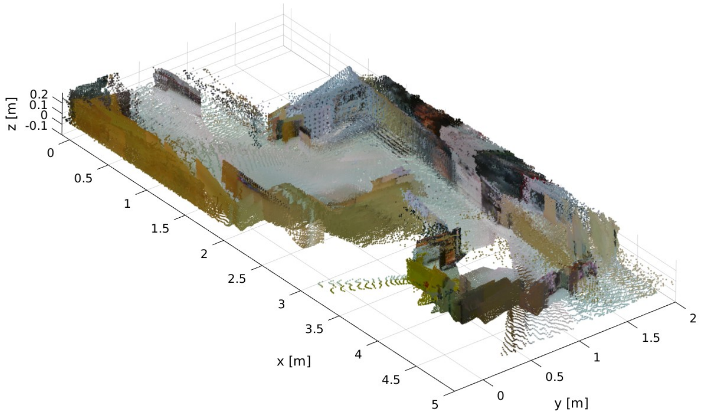
- With regard to the 3D map, it is clear that the map's main geometry is present in the estimate. However, the noise and wrong geometry makes the map estimate not usable for anything else other than the robot's immediate localisation.



(a) Angled Front Right View



(b) Angled Front Left View



(c) Angled Back Right View

Figure 5.16: Angled views of 3D map estimated by *rtabmap_ros*.

Chapter 6

Conclusions

This chapter focuses on the general work done, the results obtained, and possible improvements that could be implemented in future work.

6.1 Achievements

All the objectives exposed in Section 1.3 were achieved. A functional robot equipped with LiDAR and depth camera sensors was built and developed. This robot is capable of localising itself from its proprioceptive sensors alone with acceptable accuracy given its locomotion method, tracked motion. The robot is also SLAM capable and ROS2-based. This means that it is not only capable of improving its localisation estimate from the proprioceptive sensors, but also capable of generating a 2D or 3D map estimate - depending on the chosen algorithm - of its unknown environment.

The robot was built from the ground up, only reusing the tracks of an older robot. This construction process can be easily reproduced and used as a framework for the building of more TIR-ANT-like robots for the Laboratório de Controlo, Automação e Robótica of the Department of Mechanical Engineering of Instituto Superior Técnico using different locomotion apparatuses but maintaining the electronics basis.

The developed robotic platform is proven robust, having travelled roughly $1.5\mathrm{km}$ in recorded experimental trials and having recorded more than $100\mathrm{GB}$ of data during these trials.

The ROS2-based software created was done in a way that followed all ROS-REPS. This means that the pipeline is compatible with most ROS-based packages and that little modification to existing software has to be made to add new features. The developed robotic platform is therefore easily upgraded and available for future work, which was one of its main goals being the first ROS2-based SLAM capable tracked robot of the Laboratório de Controlo, Automação e Robótica of the Department of Mechanical Engineering of Instituto Superior Técnico. The robot is also already equipped and working with two types of exteroceptive sensors, the depth camera and the laser rangefinder. This further increases future work possibilities.

The work resulted in not only a physical robot but also a Gazebo-based simulator for the robot and its capabilities. Being a Gazebo-based simulator and programmed following the previously mentioned

ROS-REPS, it can be very easily modified and tuned according to the needs of the work being done.

A systematic approach to the testing and validation of the algorithms developed was adopted that enabled the acquisition of results from which confident conclusions on performance could be drawn.

The challenge of the tracked motion was overcome, and a satisfactory approximation of the kinematics model of the robot was found using experimental calibration techniques intended for differential drive robots.

An EKF was successfully applied to the platform which enables the robot to estimate its position accurately in small to medium sized trajectories further overcoming the challenge of the tracked motion.

The implemented 2D SLAM algorithm works robustly and estimates 2D maps very close to the truth. The 3D SLAM algorithm implemented does not work as well as the 2D algorithm and estimates noisy maps. However, the environment's main geometry is still present in these maps and, as such, for localisation purposes, both algorithms perform acceptably.

6.2 Future Work

The fastest improvement to the robot can be to further tune the *rtabmap_ros* algorithm to better deal with shaky tracked motion. The next improvement would be to optimise the code and framework in a way that enables higher frame rates and higher-resolution data streams from the depth camera without the Raspberry PI losing frames.

The 3D SLAM algorithm was tested only using the depth camera and the 2D rangefinder. Testing the algorithm with only the depth camera might reduce the computational load and allow improvements in SLAM performance.

The robot was only tested in the arena. Mapping larger-scale environments and running on different terrain types should be the next big step after both SLAM algorithms are considered in an acceptable state.

The tracks mounted on the robot can be easily disassembled. Testing various locomotion apparatuses or even different types of tracks and their influence on localisation and map generation would be an interesting study.

Changing the main computing unit for a more powerful one would enable the robot to perform more complex tasks and run heavier algorithms. For example, for the dynamic environment problem of SLAM, where the state of the art is applying artificial intelligence, a more powerful computing unit should be necessary.

An idea that was discussed during the development of the robot, but was not pursued, was the integration of a controller with more than one vibrating motor available for force feedback. This would enable for a more complex operator interaction. Having the controller vibrate in a directional way with the placement of the obstacle and the proximity of the environment to the robot is an interesting idea that can easily be implemented.

The robot is capable of SLAM. This means that it is capable of robustly localising itself in an unknown environment. Making it truly autonomous would only require adding any type of high-level control to the software pipeline, and making the operator guidance not necessary.

Only previously available SLAM algorithms were tested. Further testing with SLAM algorithms and developing a more custom SLAM approach for the particular robot would in theory improve the results obtained and would at least be interesting work.

A simulator was also developed together with the robot. However, little attention was paid to it in terms of making it as close to reality as possible. The improvement of the simulator would enable more work to be done on the robot and on the development of algorithms without the need to access the actual hardware.

Bibliography

[1] T. Haidegger. Taxonomy and Standards in Robotics. In M. H. Ang, O. Khatib, and B. Siciliano, editors, *Encyclopedia of Robotics*. Springer Berlin Heidelberg, 2021. doi: 10.1007/978-3-642-41610-1_190-1.

[2] B. Siciliano and O. Khatib, editors. *Springer Handbook of Robotics*. Springer Handbooks. Springer International Publishing, Cham, 2016. doi: 10.1007/978-3-319-32552-1.

[3] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to autonomous mobile robots*. MIT Press, 2nd ed edition, 2011. ISBN 978-0-262-01535-6.

[4] H. Durrant-Whyte and T. Bailey. Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. 2006.

[5] S. Macenski and I. Jambrecic. SLAM Toolbox: SLAM for the dynamic world. *Journal of Open Source Software*, 6(61), May 2021. ISSN 2475-9066.

[6] E. Ackerman and E. Guizzo. iRobot Brings Visual Mapping and Navigation to the Roomba 980 - IEEE Spectrum. URL `https://spectrum.ieee.org/irobot-brings-visual-mapping-and-navigation-to-the-roomba-980`. Acessed: 2022-10-05.

[7] H. Wang, C. Zhang, Y. Song, B. Pang, and G. Zhang. Three-Dimensional Reconstruction Based on Visual SLAM of Mobile Robot in Search and Rescue Disaster Scenarios. *Robotica*, 38, Feb. 2020. doi: 10.1017/S0263574719000675.

[8] D. Geromichalos, M. Azkarate, E. Tsardoulias, L. Gerdes, L. Petrou, and C. Perez Del Pulgar. SLAM for autonomous planetary rovers with global localization. *Journal of Field Robotics*, 37, Aug. 2020. doi: 10.1002/rob.21943.

[9] Meet the Roomba's Ancestor: The Cybernetic Tortoise - IEEE Spectrum, . URL `https://spectrum.ieee.org/meet-roombas-ancestor-cybernetic-tortoise`. Acessed: 2022-10-06.

[10] D. P. Watson and D. H. Scheidt. Autonomous Systems. *Johns Hopkins APL Technical Digest*, 26 (4), 2005.

[11] SHAKEY: The World's First Mobile Intelligent Robot, 1972 - Engineering and Technology History Wiki, . URL `https://ethw.org/Milestones:SHAKEY:_The_World%E2%80%99s_First_Mobile_Intelligent_Robot,_1972`. Acessed: 2022-10-06.

[12] History | Exploration – Moon: NASA Science, . URL `https://moon.nasa.gov/exploration/history/`. Acessed: 2022-10-06.

[13] Stanford's robotics legacy | Stanford News, . URL `https://news.stanford.edu/2019/01/16/stanfords-robotics-legacy/`. Acessed: 2022-10-06.

[14] E. D. Dickmanns, R. Behringer, D. Dickmanns, T. Hildebrandt, M. Maurer, F. Thomanek, and J. Schiehlen. The seeing passenger car 'VaMoRs-P'.

[15] Missions | Mars Exploration Section – NASA Mars Exploration, . URL `https://mars.nasa.gov/mars-exploration/missions/?page=0&per_page=99&order=date+desc&search=`. Acessed: 2022-10-06.

[16] R. Behringer. The DARPA grand challenge - autonomous ground vehicles in the desert. *IFAC Proceedings Volumes*, 37(8), July 2004. doi: 10.1016/S1474-6670(17)32095-5.

[17] B. Siciliano, O. Khatib, and F. Groen. Springer Tracts in Advanced Robotics: The 2005 DARPA Grand Challenge. 36, 2005.

[18] Autonomous Vehicles Complete DARPA Urban Challenge - IEEE Spectrum, . URL `https://spectrum.ieee.org/autonomous-vehicles-complete-darpa-urban-challenge`. Acessed: 2022-10-11.

[19] A. Bouman, M. F. Ginting, N. Alatur, M. Palieri, D. D. Fan, T. Touma, T. Pailevanian, S.-K. Kim, K. Otsu, J. Burdick, and A.-a. Agha-mohammadi. Autonomous Spot: Long-Range Autonomous Exploration of Extreme Environments with Legged Locomotion, Nov. 2020.

[20] R. C. Smith and P. Cheeseman. On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5, Dec. 1986. doi: 10.1177/027836498600500404.

[21] H. F. Durrant-Whyte. Uncertain Geometry in Robotics. 1988. doi: 10.1109/56.768.

[22] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, volume 4, Raleigh, NC, USA, 1987. Institute of Electrical and Electronics Engineers. doi: 10.1109/ROBOT.1987.1087846.

[23] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7, June 1991. doi: 10.1109/70.88147.

[24] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, 2005. ISBN 978-0-262-20162-9.

[25] S. Thrun, C. Martin, Y. Liu, D. Hahnel, R. Emery-Montemerlo, D. Chakrabarti, and W. Burgard. A Real-Time Expectation-Maximization Algorithm for Acquiring Multiplanar Maps of Indoor Environments With Mobile Robots. *IEEE Transactions on Robotics and Automation*, 20, June 2004. doi: 10.1109/TRA.2004.825520.

[26] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A. Y. Ng. Simultaneous Mapping and Localization with Sparse Extended Information Filters: Theory and Initial Results. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, volume 7. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-540-45058-0_22.

[27] S. Thrun and Y. Liu. Multi-robot SLAM with Sparse Extended Information Filers. In B. Siciliano, O. Khatib, P. Dario, and R. Chatila, editors, *Robotics Research. The Eleventh International Symposium*, volume 15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. doi: 10.1007/11008941_27. Series Title: Springer Tracts in Advanced Robotics.

[28] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. 2002.

[29] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Probably Converges. 2003.

[30] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. FastSLAM: An Efficient Solution to the Simultaneous Localization And Mapping Problem with Unknown Data Association. 2003.

[31] S. Thrun and M. Montemerlo. The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *The International Journal of Robotics Research*, 25, May 2006. doi: 10.1177/0278364906065387.

[32] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29, June 2007. doi: 10.1109/TPAMI.2007.1049.

[33] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31, Oct. 2015. doi: 10.1109/TRO.2015.2463671.

[34] Q. Lv, H. Lin, G. Wang, H. Wei, and Y. Wang. ORB-SLAM-based tracing and 3D reconstruction for robot using Kinect 2.0. In *2017 29th Chinese Control And Decision Conference (CCDC)*, Chongqing, China, May 2017. IEEE. doi: 10.1109/CCDC.2017.7979079.

[35] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34, Apr. 2013. doi: 10.1007/s10514-012-9321-0.

[36] R. Mur-Artal and J. D. Tardos. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics*, 33, Oct. 2017. doi: 10.1109/TRO.2017.2705103.

[37] X. Zhao, T. Zuo, and X. Hu. OFM-SLAM: A Visual Semantic SLAM for Dynamic Indoor Environments. *Mathematical Problems in Engineering*, 2021, Apr. 2021. doi: 10.1155/2021/5538840.

[38] M. S. Bahraini, A. B. Rad, and M. Bozorg. SLAM in Dynamic Environments: A Deep Learning Approach for Moving Object Tracking Using ML-RANSAC Algorithm. *Sensors*, 19, Aug. 2019. doi: 10.3390/s19173699.

[39] L. Cui and C. Ma. SOF-SLAM: A Semantic Visual SLAM for Dynamic Environments. *IEEE Access*, 7, 2019. doi: 10.1109/ACCESS.2019.2952161.

[40] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei. DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Oct. 2018. IEEE. doi: 10.1109/IROS.2018.8593691.

[41] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, editors. *Wheeled mobile robotics: from fundamentals towards autonomous systems*. Butterworth-Heinemann, an imprint of Elsevier, 2017. ISBN 978-0-12-804204-5.

[42] P. Corke. *Robotics and Control: Fundamental Algorithms in MATLAB®*, volume 141 of *Springer Tracts in Advanced Robotics*. Springer International Publishing, Cham, 2022. doi: 10.1007/978-3-030-79179-7.

[43] J. Borenstein, H. Everett R., and L. Feng. *"Where am I?" sensors and methods for mobile robot positioning*. 1996.

[44] J. Borenstein and L. Feng. UMBmark: a benchmark test for measuring odometry errors in mobile robots. Philadelphia, PA, Dec. 1995. doi: 10.1117/12.228968.

[45] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems, 1960.

[46] S. Arno and S. Särkkä. Optimal Filtering with Kalman Filters and Smoothers - a Manual for Matlab toolbox EKF/UKF, 2011.

[47] A. M. Sabatini. Kalman-Filter-Based Orientation Determination Using Inertial/Magnetic Sensors: Observability Analysis and Performance Evaluation. *Sensors*, 11, Sept. 2011. doi: 10.3390/s111009182.

[48] T. Moore and D. Stouch. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. volume 302. Springer International Publishing, 2016. doi: 10.1007/978-3-319-08338-4_25.

[49] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7, May 2022.

[50] M. Labbé and F. Michaud. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36, Mar. 2019. ISSN 15564959.

[51] Principle and advantages of magnetic encoder | Tutorials | Rotation Angle Sensors | Products | Asahi Kasei Microdevices (AKM), . URL https://www.akm.com/eu/en/products/rotation-angle-sensor/tutorial/magnetic-encoder/. Acessed: 2022-10-16.

[52] C. Cavallo. All About DC Motor Controllers - What They Are and How They Work. URL https://www.thomasnet.com/articles/instruments-controls/dc-motor-controllers/. Acessed: 2022-10-16.

[53] Beginner's guide to depth (Updated) – Intel® RealSense™ Depth and Tracking Cameras, . URL https://www.intelrealsense.com/beginners-guide-to-depth/. Acessed: 2022-10-16.

[54] TIR-ANT's Github Repository. URL https://github.com/PlatHum/Traxter-Software.

[55] J. Gonçalves, J. Lima, P. J. Costa, and A. P. Moreira. Modeling and Simulation of the EMG30 Geared Motor with Encoder Resorting to SimTwo: The Official RobotFactory Simulator. Springer International Publishing, Heidelberg, 2013. ISBN 978-3-319-00556-0 978-3-319-00557-7.

[56] URDF — ROS 2 Documentation: Humble documentation, . URL https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/URDF-Main.html. Acessed: 2022-10-26.

Appendix A

Further Odometry Calibration Attempt

The odometry calibration results shown in Table 5.1 enable the application of the odometry estimate model of Equations (2.20), (2.21) and (2.17) which leads to the trajectory estimates shown in Figure 5.6. It is clear that this is a major improvement over the uncalibrated odometry of the robot. However, there is still significant overestimation of distance travelled, and the estimate's behaviour on counter-clockwise runs differs significantly from clockwise ones. This chapter shows an attempt to further calibrate the obtained odometry model to reduce these effects.

For the distance travelled overestimation problem, a scaling factor E_s was determined from the data recorded in the UMBmark trials in Figure 5.5. At each trial's square vertex, the motion captured distance travelled from the previous vertex to the current one was recorded and compared with the calibrated odometry estimated one. Essentially, the actual distance travelled and the estimated distance travelled in each leg of the square trajectory was compared. The average ratio between the estimated distance travelled and the actual distance travelled is:

$$E_s = 1.0798 \tag{A.1}$$

With a standard deviation of $0.0215\mathrm{m}$. This means that on average, the calibrated odometry overestimates the distance travelled by roughly 8%.

Using the UMBmark trajectory from the trials of Figure 5.5 and the motion captured data, the ratio between the calibrated odometry estimated turn angle performed at each corner of the square and the actual motion captured one for clockwise and counter-clockwise runs can be determined. For the counter-clockwise trials, the average value of this ratio E_{cwc} calculated is:

$$E_{ccw} = 1.07 \tag{A.2}$$

With a standard deviation of $0.017\mathrm{rad}$. This means that in counter-clockwise turns, the calibrated odometry overestimates the actual turn performed by 7% on average. For clockwise trials, the average value of its respective ratio E_{cw} is:

$$E_{cw} = 0.991 \tag{A.3}$$

With a standard deviation of $0.0299\mathrm{rad}$. This means that on average the calibrated odometry estimate deviates less than 1% from the actual angular value of the turn in clockwise rotation. This supports the conclusion that the robot performed better in clockwise runs than in counter-clockwise ones after calibration.

The odometry model tested for further calibration aimed to take advantage of these ratios and incorporate them into the estimation. Therefore, the model in Equations (2.20) and (2.21) would be adapted to:

$$\Delta s = \frac{\pi \cdot r_L}{N \cdot E_s} \left(\Delta n_R \cdot E_d + \Delta n_L \right) \tag{A.4}$$

$$\Delta \varphi = \begin{cases} \dfrac{2\pi \cdot r_L}{N \cdot E_b \cdot b \cdot E_{cw}} \left(\Delta n_R \cdot E_d - \Delta n_L \right) & \left(\Delta n_R \cdot E_d - \Delta n_L \right) > 0 \\ \dfrac{2\pi \cdot r_L}{N \cdot E_b \cdot b \cdot E_{ccw}} \left(\Delta n_R \cdot E_d - \Delta n_L \right) & \left(\Delta n_R \cdot E_d - \Delta n_L \right) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{A.5}$$

Equation (A.4) simply reduces the estimated distance travelled by the calculated scaling factor. However, Equation A.5 introduces nonlinearity in the odometry model. Essentially, depending on the rotation direction, either the angular clockwise or counter-clockwise factor is applied.

This calibration approach has two major potential flaws. First, the experimental data recorded for the determination of these scaling factors were taken in only one type of floor, the Robotics Arena floor. Due to the highly dynamic nature of the tracks on the robot, the behaviour on different floors may be radically different, and the calculated factors may not be applicable. Second, this switching of the angular scaling factor depending on the rotation direction might not perform well in trajectories that rotation in both directions occurs. These angular scalar factors were determined from data in which rotation mainly occurred in one direction. The interaction between both scaling factors in a complex trajectory might result in a poor estimation.

This further calibrated model, named the ”Ratio” method, was implemented in the odometry calculation and the results were compared with the UMBmark calibrated model in the UMBmark squared trajectory. The results can be seen in Figure A.1 and the typical trajectories followed in Figure A.2.

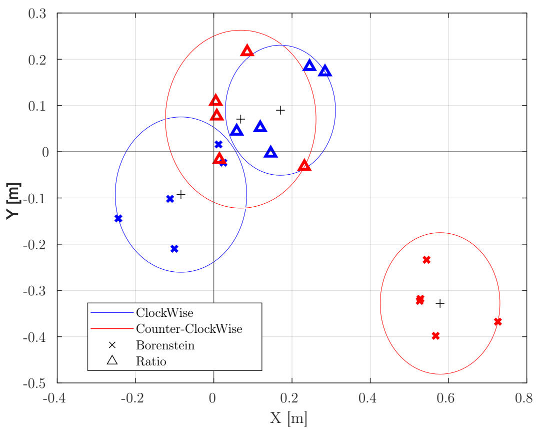
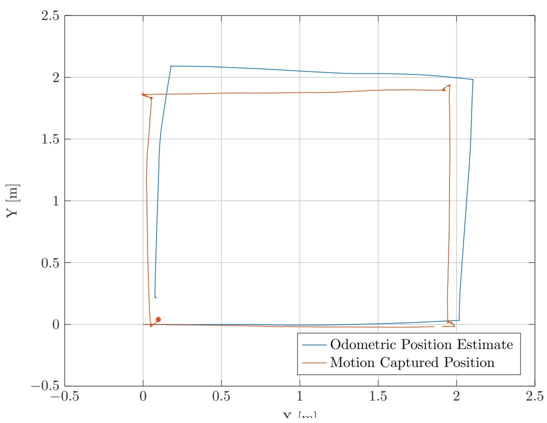
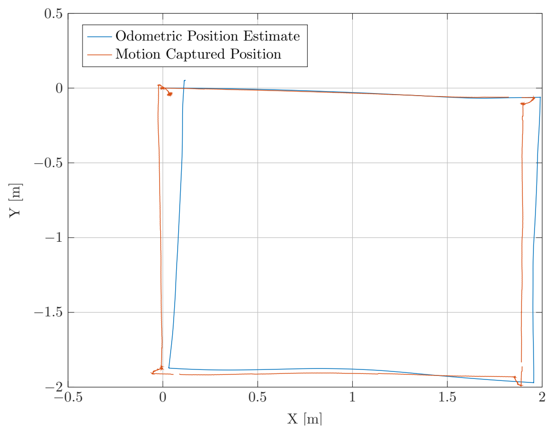


Figure A.1: UMBmark result comparison between Borenstein corrected odometry and Ratio further calibrated odometry.



(a) Counter-Clockwise

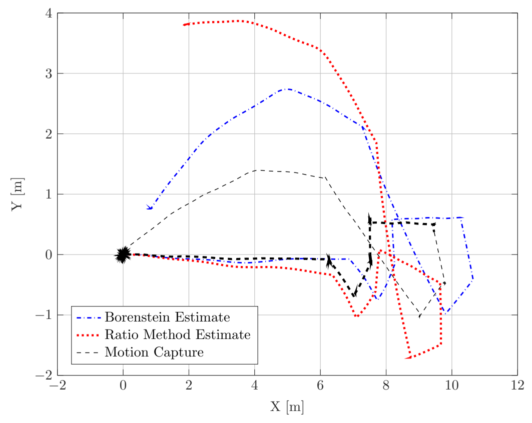


(b) Clockwise

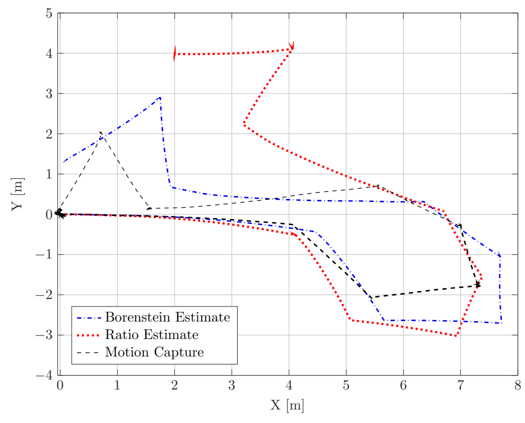
Figure A.2: Comparison between actual trajectory followed and further calibrated odometry using the Ratio method.

As expected the further calibrated odometry improved the Borenstein calibrated one. This is because the trajectory followed is a simple square, performed in a single direction on the arena floor. Comparing Figures 5.6 and A.2 it is clear that distance overestimation is not a problem for the further calibration model. Also, the significantly different nature of the results in both directions is now toned down. The further calibration normalised the estimate in both run directions. However, to test the potential flaws of this odometry model, its performance in complex trajectories needs to be tested.

A series of complex trajectories were performed, and the motion captured data compared with both the Borenstein calibrated estimate and the Ratio further calibrated estimate. The results are available in Figure A.3. It seems that the hypothesis of the interaction between the angular scaling factors ruining the estimate is valid. Switching between the two factor depending on the direction of rotation must ruin the estimation for both. The Ratio method calibrated odometry performed consistently worse than the Borenstein calibrated odometry in complex trajectories.



(a)



(b)

Figure A.3: Comparison between Borenstein calibrated odometry, Ratio method further calibrated odometry and actual trajectory followed in two complex trajectories.

Based on these results, this further calibration failed and should not be implemented.