Simultaneous Localisation and Mapping with a Tracked Mobile Robot

Francisco Gonçalves
francisco.v.p.goncalves@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

November 2022

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: SLAM, Odometry, Tracked-Drive, Extended Kalman Filter, Robotics

1. Introduction

A 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]. 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 move using its own locomotion apparatus.

The highest complexity tool for localisation estimation in robotics is Simultaneous Localisation and Mapping (SLAM). 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 [3].

Mobile robotics and, more specifically, SLAM capable mobile robotics is a dynamic and ever-growing area of research steadily gaining popularity since the beginning of the millennium. Nowadays, SLAM algorithms can run on inexpensive hardware. 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 Robotic Operating System (ROS/ROS2) package format[3] that are widely used in industry with promising results [4].

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. 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.

2. Background

2.1. Differential-Drive Forward Kinematics

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

$$\xi_g(t) = \begin{bmatrix} x(t) & y(t) & \varphi(t) \end{bmatrix}^T \tag{1}$$

[1] https://sites.google.com/view/awesome-slam-datasets/

[2] https://openslam-org.github.io/

[3] https://index.ros.org/search/?term=slam

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. This information is schematised in Figure 1.

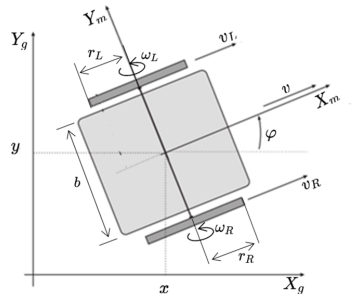


Figure 1: Schematic of differential-drive kinematics [5].

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

$$R\left(\varphi\right)=\begin{bmatrix}\cos\varphi & \sin\varphi & 0\\ -\sin\varphi & \cos\varphi & 0\\ 0 & 0 & 1\end{bmatrix} \tag{2}$$

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. There is also a need to assume that the motion speed and friction are low enough for the dynamics to not affect the accuracy of the kinematics model.

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. According to [5–7] a differential-drive robot's linear velocity is proportional to the sum of each of its drive-wheels' linear velocity:

$$v\left(t\right)=\frac{v_R\left(t\right)+v_L\left(t\right)}{2} \tag{3}$$

where v_R and v_L are the linear velocities of the right and left drive-wheels, respectively. Its rotational velocity is proportional to the difference between its drive-wheels' linear velocity:

$$\dot{\varphi}\left(t\right)=\frac{v_R\left(t\right)-v_L\left(t\right)}{b} \tag{4}$$

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

Since perfect friction is assumed and, therefore, no slippage between wheels and the ground, a drive-wheel's linear velocity will match its tangential velocity. This allows for the expression of the robot's linear and angular velocity of (3) and (4) in order of each drive-wheel's speed:

$$\dot{\xi}_g(t)=\begin{bmatrix}\cos\varphi\left(t\right) & 0\\ \sin\varphi\left(t\right) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}r_R/2 & r_L/2\\ r_R/b & -r_L/b\end{bmatrix}\begin{bmatrix}\omega_R\left(t\right)\\ \omega_L\left(t\right)\end{bmatrix} \tag{5}$$

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

2.2. 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:

$$\omega_L\left(t\right)=\frac{v^c\left(t\right)-\dot{\varphi}^c\left(t\right)\cdot\dfrac{b}{2}}{r_L} \tag{6}$$

$$\omega_R\left(t\right)=\frac{v^c\left(t\right)+\dot{\varphi}^c\left(t\right)\cdot\dfrac{b}{2}}{r_R} \tag{7}$$

2.3. Tracked-Drive Kinematics

The kinematics of the tracked-drive can be approximately described by the differential-drive's kinematics. However, the no slippage condition assumed for the 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. 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.

2.4. Odometry

Odometry 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, accumulation of orientation errors will cause the odometric position estimation to drift from the actual robot position proportionally to the distance travelled.

The kinematics model of (5) can be integrated at some time t to obtain the robot pose which can be written in discrete form using Euler's numerical integration approximation or Trapezoidal numerical 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. However, 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. 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 to robot motion. Therefore, given wheel i's increase/decrease in encoder ticks between time-steps Δn_i, the distance travelled can be calculated as:

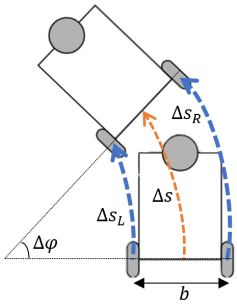


Figure 2: Illustration of the geometric properties of differential-drive motion.

$$\Delta s_i = \frac{2\pi}{N}\left(\Delta n_i \cdot r_i\right) \tag{8}$$

where N is the number of encoder ticks per wheel revolution.

Figure 2 shows 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. Nevertheless, following Figure 2's depiction, the distance travelled by each wheel is an arc length; finding the corresponding arc angle and centre arc length is a trivial geometric matter:

$$\Delta s = \frac{(\Delta s_R + \Delta s_L)}{2} = \frac{\pi}{N}\left(\Delta n_R \cdot r_R + \Delta n_L \cdot r_L\right) \tag{9}$$

$$\Delta\varphi = \frac{(\Delta s_R - \Delta s_L)}{b} = \frac{2\pi}{N \cdot b}\left(\Delta n_R \cdot r_R - \Delta n_L \cdot r_L\right) \tag{10}$$

The global coordinate frame pose estimate is then calculated incrementally:

$$\begin{aligned} x(k) &= x(k-1) + \Delta s\cos(\varphi(k)) \\ y(k) &= y(k-1) + \Delta s\sin(\varphi(k)) \\ \varphi(k) &= \varphi(k-1) + \Delta\varphi \end{aligned} \tag{11}$$

2.5. 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. Borenstein et al. [8] 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 [9]. Their work in differential-drive odometry calibration focused on two types of systematic errors: errors due to unequal wheel diameters and errors due to uncertainty about the wheelbase. To quantify these errors, Borenstein and Feng [9] defined two ratios. The experimental ratio between the radius of both drive-wheels (12) and the experimental ratio between the actual wheelbase and the assumed one (13):

$$E_d = \frac{r_R}{r_L} \tag{12}$$

$$E_b = \frac{b_{actual}}{b} \tag{13}$$

With these ratios, (9) and (10) can be rewritten as:

$$\Delta s = \frac{\pi \cdot r_L}{N}\left(\Delta n_R \cdot E_d + \Delta n_L\right) \tag{14}$$

$$\Delta\varphi = \frac{2\pi \cdot r_L}{N \cdot E_b \cdot b}\left(\Delta n_R \cdot E_d - \Delta n_L\right) \tag{15}$$

The values of E_b and E_d can be experimentally determined following the University of Michigan benchmark (UMBmark), also known as the bidirectional square path method [8, 9]. 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.

2.6. 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. The governing equations of a DC motor can be derived from Kirchhoff's voltage law and Newton's second law. Assuming that the voltage drop due to the inductance is negligible, the governing differential equation of the DC motor can be solved in order of the rotational velocity of the motor's output-shaft $\dot{\theta}$:

$$\dot{\theta}(t) = \frac{a_1}{a_2}\left(1 - \exp\left(-a_2 \cdot t\right)\right) \tag{16}$$

$$\text{where} \quad a_1 = \frac{K \cdot V(t) - R \cdot T}{R \cdot J}$$

$$a_2 = \frac{K^2 + R \cdot B}{R \cdot J}$$

where J is the motor moment of inertia, K the current-developed torque conversion factor, R the resistor, T the static friction factor, B the viscous friction factor and V the voltage supplied to the motor.

2.7. Extended Kalman Filter

The Kalman Filter 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. 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. The correction step rectifies the prediction given the current measurements. The algorithm assumes the system to be linear; however, most real-world systems are not. The Extended Kalman Filter (EKF) is the modification of the original Kalman Filter to work with non-linear systems. Instead of working with linear dynamics and measurement model functions, the EKF deals with non-linear 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 [10].

2.8. 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 [6].

The localisation part of SLAM is the problem of estimating the robot position and even its entire previous trajectory given a map of the environment. The mapping part is the construction of the map of the environment knowing the robot's trajectory. 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 [6]. The most commonly used exteroceptive sensors for SLAM are laser rangefinders, ultrasonic sensors, and cameras. 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.

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

$$X_T = \{x_0\,,\, x_1\,,\, \ldots\,,\, x_T\} \tag{17}$$

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 motion of the robot can then be written as follows:

$$U_T = \{u_0\,,\, u_1\,,\, \ldots\,,\, u_T\} \tag{18}$$

Let M denote the true map of the environment:

$$M = \{m_0\,,\, m_1\,,\, \ldots\,,\, m_{n-1}\} \tag{19}$$

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:

$$Z_T = \{z_0\,,\, z_1\,,\, ...\,,\, z_T\} \tag{20}$$

The online SLAM problem derives only the latest robot pose, while the full SLAM problem derives the entire sequence of robot poses [11] . 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 [6]:

$$p\,(X_T, M \mid Z_T, U_T) \tag{21}$$

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 [11].

2.9. 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 [6]. 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 [6]. These constraints can be motion constraints between two successive robot poses and measurement constraints between a robot pose and a feature in the environment. 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 [11]. 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 [6].

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. [11].

2.10. ROS2 and micro-ROS

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. ROS2 [12] is the improved version of ROS. It 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.

micro-ROS[4] 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 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.

2.11. Robot Localisation Package

The Robot Localisation ROS package [13] 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 EKF and an Unscented Kalman Filter. Both 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.

2.12. SLAM Toolbox Package

SLAM Toolbox [4] 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 kinematic map merging. 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.

2.13. RTAB-Map 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. 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 detailed breakdown of the RTAB-Map algorithm is presented in the work of Labbé and Michau [14].

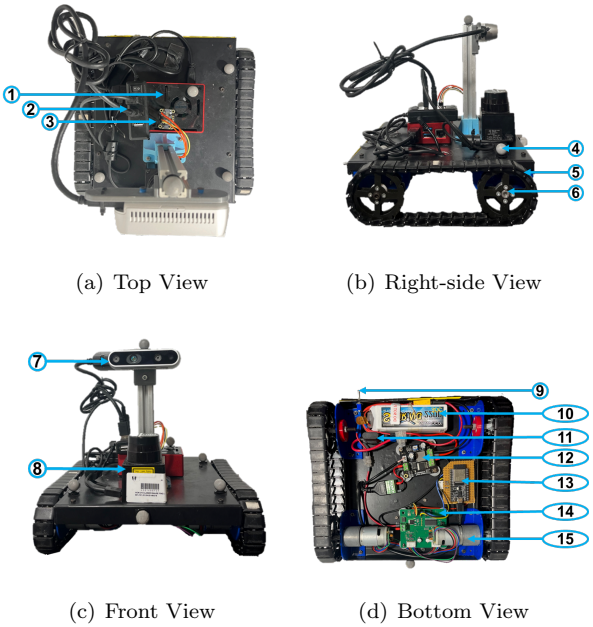
[4] `https://micro.ros.org/`

3. TIR-ANT

The developed robot was named TIR-ANT 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 [15].

3.1. Robot Overview

The final assembled robot can be seen in Figure 3 and its components breakdown can be consulted in the accompanying Table 1.



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

(c) Front View (d) Bottom View

Figure 3: The TIR-ANT. Identifier breakdown in Table 1.

Table 1: Figure 3's accompanying table with the components breakdown.

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

3.2. Software Integration

Figure 4 contains the schematic of the entire ROS2 framework that runs on the TIR-ANT. The entire software codebase was developed for ROS2 Humble Hawksbill. The operating system running on the 8GB Raspberry Pi 4B equipped with a 32GB SD-card is Ubuntu 22.04 Jammy Jellyfish. Starting at the low-level, the algorithm running on the ESP32 microcontroller 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 and IMU information at a 50Hz rate. This mapping between wheel speed command and motor command is deduced from combining (16) and the work of Gonçalves et al [16], who modelled the EMG30 motors for simulation. Considering the motors fast enough so that their transient response is dispensable, leaving only its stationary response $t \longrightarrow +\infty$, and forcing a 1rad/s deadzone, the estimated mapping between wheel speed and motor command u is:

$$\begin{cases} u\left(t\right) = -\frac{66.814\left(\left|\omega^{c}\left(t\right)\right|+1.0691\right)}{V_{\max}\left(t\right)} & \omega^{c}\left(t\right) < -1\text{rad/s} \\ u\left(t\right) = \frac{66.293\left(\left|\omega^{c}\left(t\right)\right|+1.0691\right)}{V_{\max}\left(t\right)} & \omega^{c}\left(t\right) > 1\text{rad/s} \\ u\left(t\right) = 0 & \text{otherwise} \end{cases} \tag{22}$$

where V_{max} is the available battery voltage.

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 messages going through the serial communication appropriately.

The data streams from the RealSense camera, the Hokuyo rangefinder and the PS3 controller enter the ROS2 framework directly through the *realsense_ros*[5], *urg_node*[6] and *teleop_twist_joy*[7] libraries, respectively.

The *imu_interpreter* node receives the BNO055's attitude estimate, angular velocity and acceleration data and publishes it in the appropriate ROS-

[5] https://github.com/IntelRealSense/realsense-ros/tree/ros2

[6] https://github.com/ros-drivers/urg_node/tree/ros2-devel

[7] https://github.com/ros2/teleop_twist_joy/tree/humble

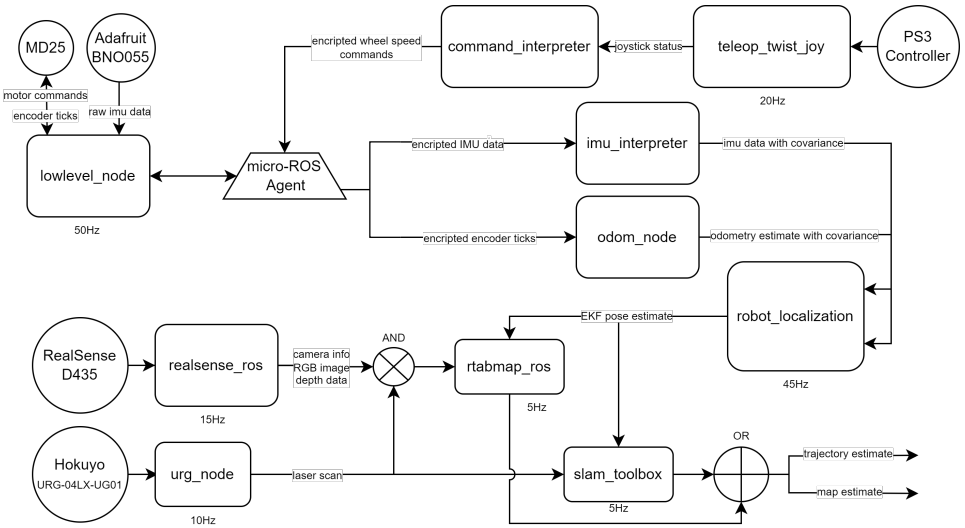


Figure 4: Schematic of developed software pipeline.

message type together with the covariance associated with the measurement.

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. The main approximation done in this work is to apply a differential-drive model to the TIR-ANT for odometry and pose estimation. This model is then calibrated using the UMBmark method proposed by Borenstein et al. [9]. 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. The main kinematics properties measured for the nominal differential-drive model of the TIR-ANT can be seen in Table 2.

Table 2: 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

odom_node takes the encoder tick data and converts it into an odometric pose estimate applying the model from (14) and (15) with the kinematics properties of Table 2 and E_b, E_d calculated experimentally. Before calibration, these values are considered unity. 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*[8] package then fuses the information from *odom_node* and *imu_interpreter* and publishes an improved pose estimate at a 45Hz rate.

The *command_interpreter* node takes the desired body frame velocity and rotation from the PS3 controller, applies (6) and (7) and sends the wheel speed commands to the low-level.

For the 2D mapping SLAM algorithm *slam_toolbox* package[9] only the laser scan data from the Hokuyo rangefinder is necessary. These scans are published at 10Hz. The 3D mapping algorithm is the one from *rtabmap_ros* package[10], requiring both the laser scan and the RealSense camera data. Both SLAM algorithms will take the EKF pose estimate and the exteroceptive sensors' data to perform map building and localisation and publish results at 5Hz.

4. Experimental Results

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, which is equipped with a 3D infrared tracking system. 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 5mm) 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.

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

[9] `https://github.com/SteveMacenski/slam_toolbox`

[10] `https://github.com/introlab/rtabmap_ros`

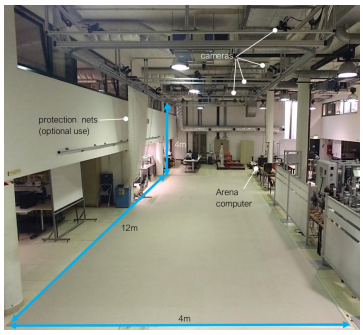
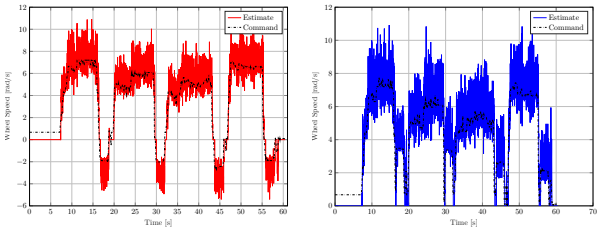


Figure 5: Robotics Arena of the Laboratório de Controlo, Automação e Robótica of the Department of Mechanical Engineering of Instituto Superior Técnico.

4.1. 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 6. The wheel speed estimation was done through numerical differentiation, which introduces noise. Despite this fact, it is clear that the low-level control works as intended. The approximation made by not taking into account the motor's transient response indicates reasonable.



(a) Left Motor (b) Right Motor

Figure 6: Comparison between wheel speed command and estimated wheel speed obtained.

4.2. Odometry Calibration

For the calibration of the odometry, the described UMBmark method (Section 2.4) was applied. The robot was made to follow a square trajectory with a 1.9m side in both clockwise and counter-clockwise directions. To ensure repeatability, the intended path was marked on the floor of the arena - Figure 7. The robot's trajectory was motion-captured. Performing the UMBmark trajectory and using the uncalibrated odometry model from (11), typical trajectory estimates obtained compared to the actual trajectory performed can be seen in Figure 8. After several trials of the UMBmark, the experimentally found calibration coefficients can be found in Table 3. The uncalibrated clockwise runs showed the most

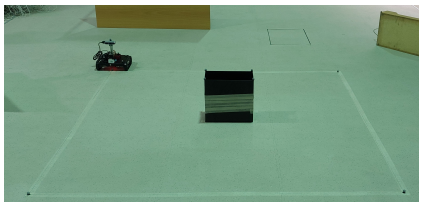
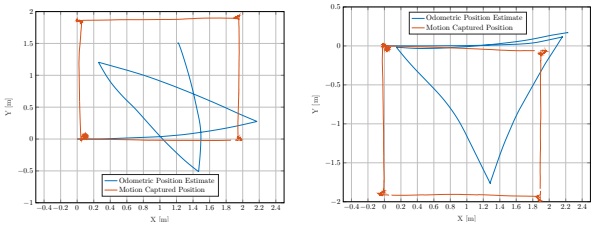


Figure 7: UMBmark's square trajectory marked on the floor of the arena.

variance and the worst overall error when compared to the uncalibrated counter-clockwise runs. 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.



(a) Counter-Clockwise (b) Clockwise

Figure 8: Comparison between actual trajectory followed and uncalibrated odometry estimation.

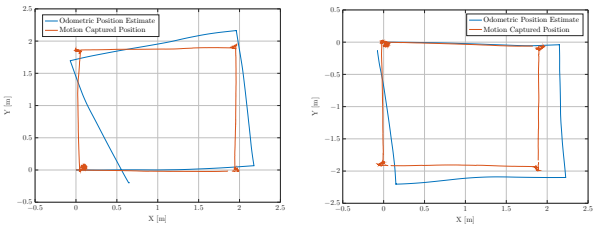
Table 3: UMBmark results for uncalibrated odometry.

E_d	E_b	E_{max}
0.984	1.452	2.227

After applying the calibration coefficients from Table 3 to the odometry model, the odometric pose estimate performing the UMBmark trajectory is now much closer to reality - Figure 9. 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 wheel rotation that does not result in motion. 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.

4.3. Extended Kalman Filter Valdiation

By fusing the calibrated odometry estimate and the robot's IMU data, the EKF is able to improve on the odometry estimation and maintain acceptable accuracy in small- to medium-scale trajectories. An example of the performance of the EKF compared



(a) Counter-Clockwise (b) Clockwise

Figure 9: Comparison between actual trajectory followed and calibrated odometry estimation.

to the calibrated odometry on a complex trajectory is shown in Figure 10. 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.

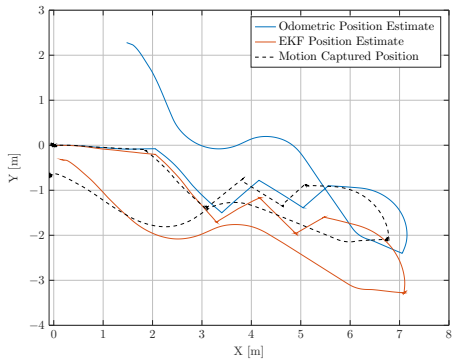
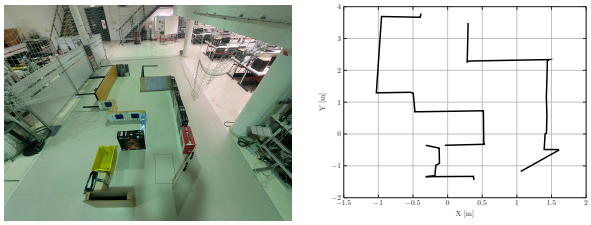


Figure 10: Comparison between actual trajectory followed, calibrated odometry and EKF estimate.

4.4. SLAM Validation

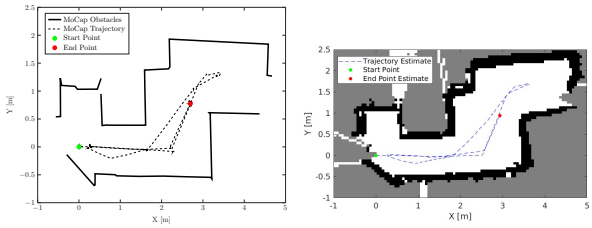
For the validation of the SLAM algorithms, several obstacles were placed in the arena and the robot was made to explore it. The motion capture camera system was used to measure the obstacle location - Figure 11. The cameras were also used to capture the robot's motion for localisation performance comparison. Both the *slam_toolbox* algorithm and the *rtabmap_ros* algorithms were tested on this map. The results for the *slam_toolbox* can be seen in Figure 12, the *rtabmap_ros* results in Figures 13 and 14.

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 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, drawing "thick" walls; The *rtabmap_ros* algorithm works 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; The



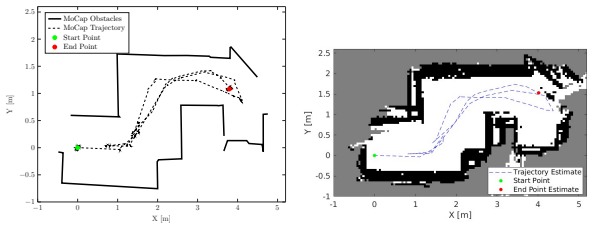
(a) Map (b) 2D Representation

Figure 11: Map built and its 2D representation.



(a) Motion Capture (b) *slam_toolbox* estimate

Figure 12: Comparison between motion-captured data and the *slam_toolbox* estimate.



(a) Motion Capture (b) *rtabmap_ros* estimate

Figure 13: Comparison between motion-captured data and the *rtabmap_ros* estimate.

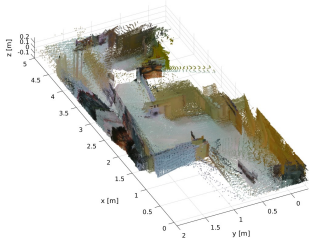


Figure 14: *rtabmap_ros* map estimate.

shaking during the tracked motion seems to damage camera performance. Sensor data had already been significantly downgraded to allow fast processing which works together with the shaking motion to disturb the loop closure and data association of the algorithm.

5. Conclusions

All the objectives described in Section 1 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. An EKF was successfully applied to the platform, which enables the robot to estimate its position accurately in small- to medium-sized trajectories. The robot is also SLAM capable. This means that it is capable of generating 2D or 3D map estimates - depending on the chosen algorithm - of its unknown environment. The implemented 2D SLAM algorithm works robustly and estimates 2D maps accurately. 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.

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

Acknowledgements
I express my gratitude to all involved in the development of this work. Mainly Prof.Alexandra Moutinho and Prof.Mário Ramalho for their guidance and Mr.Luís Raposeiro and Eng.Camilo Christo for their help with hardware problems.

References

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

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

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

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

[12] 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(66), May 2022. ISSN 2470-9476.

[13] 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.

[14] 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(2), Mar. 2019. ISSN 15564959.

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

[16] 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.