AugBot: indoor location system using ROS

João Paulo de Melo Carvalho
Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciências de Computadores
2022

Orientador
Eduardo Resende Brandão Marques
Professor Auxiliar
Faculdade de Ciências da Universidade do Porto

Coorientador
Sérgio Armindo Lopes Crisóstomo
Professor Auxiliar
Faculdade de Ciências da Universidade do Porto

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/_________

Declaração de Honra

Eu, João Paulo de Melo Carvalho, inscrito no Mestrado em Engenharia de Redes e Sistemas Informáticos da Faculdade de Ciências da Universidade do Porto declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U.Porto, que o conteúdo da presente dissertação reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta dissertação de estágio, declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente dissertação de estágio quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor.

Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

João Carvalho,

14 de Outubro de 2022

Abstract

The ability to obtain a precise location of a person or object has led to several so-called location-based services, mainly in logistics, but also in more consumer applications such as localized weather information and navigation. This works well when a clear view of the sky allows the use of a Global Navigation Satellite System (GNSS) to identify an asset's location with good accuracy. In an indoor environment, however, GNSS signal reception is severy compromised and consequently, the accuracy of position estimates is poor, and a custom sensor infrastructure is required within the environment to overcome the problem.

In the scope of the Augmanity project, we present a framework called AugBot for developing and testing indoor location algorithms using low-footprint embedded devices. AugBot is implemented using the Robot Operating System (ROS), allowing a modular separation of code for distinct concerns like sensor readings, communication, as well as the location algorithms themselves. AugBot has been deployed in a real-world physical environment for indoor location that includes the AlphaBot2 robot and Ultra-Wide Band (UWB) beacons, as well as in a simulation environment enabled using the Gazebo simulation engine. In both environments, we implemented and evaluated two different types of indoor location algorithms: multi-lateration feeding on UWB beacon ranges, and dead reckoning feeding on inertial sensor measurements.

AugBot is a basis for hardware/software integration in the Augmanity project tasks in collaboration with FCUP partners, and suitable for the development and testing of more complex algorithms for indoor location and different use cases in indoor location.

Resumo

A capacidade de obter a localização precisa de uma pessoa ou objecto levou à criação de diversos serviços baseados na localização do utilizador, para logísticas, mas também em aplicações para consumo, tais como informação meteorológica localizada e navegação. Isto funciona bem quando a vista para o céu não se encontra obstruída, permitindo a utilização de um Global Navigation Satellite System (GNSS) para identificar uma localização com boa precisão. No entanto, num ambiente *indoor*, a recepção do sinal GNSS está comprometida e, consequentemente, a precisão das estimativas de posição é fraca, sendo necessária uma infra-estrutura de sensores ajustada ao ambiente para ultrapassar o problema.

No âmbito do projecto Augmanity, apresentamos uma framework chamada AugBot para o desenvolvimento e teste de algoritmos de localização em ambientes *indoor* , utilizando dispositivos embutidos de pequenas dimensões. AugBot é implementado utilizando o Robot Operating System (ROS), permitindo uma separação modular do código para preocupações distintas como leituras de sensores, comunicação, bem como os próprios algoritmos de localização. AugBot foi implementado num ambiente físico real para localização *indoor* que inclui o robô AlphaBot2 e um sistema de *beacons* de Ultra-Wide Band (UWB), bem como num ambiente de simulação possibilitado através do motor de simulação Gazebo. Em ambos os ambientes, implementámos e avaliámos dois tipos diferentes de algoritmos de localização *indoor*: multi-lateração usando medições do sistema de *beacons* UWB, e dead reckoning baseado nas medições do sensor inercial.

O trabalho desenvolvido constitui uma base para a integração de hardware/software nas tarefas do projecto Augmanity em colaboração com os parceiros da FCUP, e é adequado ao desenvolvimento e teste de algoritmos mais complexos para localização *indoor* e diferentes casos de utilização neste tipo de cenários.

Acknowledgements

I would first like to thank my dissertation advisors, Professor Eduardo Marques and Professor Sérgio Crisóstomo for their assistance and dedicated involvement in every step, throughout the process.

Most importantly, I would like to thank my parents, my sister, and my girlfriend Sofia for supporting me and providing continuous encouragement to keep on giving my very best, especially throughout this last year. This accomplishment would not have been possible without their support.

The work of this dissertation has been funded by project Augmanity (POCI-01-0247-FEDER-046103). I would also like to thank Faculdade de Ciências da Universidade do Porto, Instituto de Telecomunicações (UIDB/50008/2020), and CRACS / INESC-TEC for hosting this work.

Contents

List of Tables

List of Figures

Acronyms

AGV	Automatic Guided Vehicles
AoA	Angle of Arrival
BLE	Bluetooth low energy
DCC	Departamento de Ciência de Computadores
DR	Dead Reckoning
ECC	Electronic Communications Committee
ERDF	European Regional Development Fund
FCUP	Faculdade de Ciências da Universidade do Porto
GNSS	Global Navigation Satellite Systems
IMU	Inertial Measurement Unit
IoT	Internet of Things
LoS	Line of Sight
NLoS	non-line of sight
PID	Proportional–Integral–Derivative
QoS	Quality of Service
ROS	Robot Operating System
RSS	Received Signal Strength
RSSI	Received Signal Strength Indication
RTLS	Real-Time Location System
RTT	Round Trip Time
TDoA	Time Difference of Arrival

ToA	Time of Arrival
ToF	Time of Flight
UWB	Ultra-Wide Band
VLC	Visual Light Communication
WLAN	wireless Local Area Network

Chapter 1

Introduction

This chapter introduces the concerned problem and the purpose of this thesis project.

1.1 Motivation

The importance of being able to accurately locate persons and objects has deeply increased over the last few years and has led to many so-called location-based services, mainly in logistics but also in more consumer applications such as localized weather information and navigation. This works well where a clear view of the sky enables the use of Global Navigation Satellite Systems (GNSS) for location estimates with good accuracy [6].

In an indoor environment, however, GNSS signal reception is severely compromised and consequently the accuracy of position estimates is poor, and a custom sensor infrastructure is required within the environment to overcome the problem. Several technologies can be used, in some cases simultaneously, e.g., the use of receive signal strength (RSS) from WiFi access points (APs) or Bluetooth beacons, range estimates from APs that are compliant with WiFi-RTT or Ultra-Wide Band (UWB) beacons, or inertial sensors that are present in devices such as smartphones or mobile robots that can be used to estimate their movement. Indoor location is challenged by the problems that, as in the case of GNSS, may lead to poor accuracy in some of these technologies, e.g., signal interference and multi-path propagation in WiFi or UWB or the accumulation of errors in position estimates that rely solely on inertial sensor measurements.

1.2 Problem statement and contributions

The Augmented Humanity (Augmanity) project [9] consortium aims to anticipate the future by developing user-friendly, immersive, and supportive technologies in industrial production environments. The project is expected to leverage its results in multiple sectors. The work of this thesis happens in the scope of Augmanity PPS3: Industrial Internet of Things and connectivity, a sector focused on putting I4.0 and edge technology at the service of people and corporations. In this overall context, there is the specific problem of developing an indoor location for tracking devices (e.g. robotic vehicles) or other physical assets (e.g. product packages) in a factory floor environment. FCUP and other project partners agreed upon an indoor location system that could make use of a UWB beacon infrastructure and inertial sensors when available and installed in devices.

In order to create a good indoor localization system, calibration has to be easily achievable, to quickly adapt to a new environment, and be capable of locating multiple targets. The system has to ensure that the estimated positions are accurate enough for the intended application and that the computation times for the estimations are fast enough to precisely estimate the position of a moving target. The software has to be small, portable and flexible, and have low power consumption, and since, the system may keep track of humans, and there is a risk of violating their integrity, safety is a must. The development and testing of indoor localization systems, starting from the prototype stage, must address all these needs.

This thesis addresses the concern of developing a framework for the development and testing of indoor localization algorithms in the scope of the Augmanity project. It proposes AugBot, implemented using the Robot Operating System (ROS) [25, 27] that enables a modular separation of code for distinct concerns like sensor readings, communication, as well as the localization algorithms themselves. AugBot has been deployed in a real-world physical environment for indoor location that includes the AlphaBot2 robot [32] and UWB beacons [8], as well as in a simulation environment enabled using the Gazebo simulation engine [26]. In both environments, we implemented and evaluated two different types of indoor location algorithms: multi-lateration feeding on UWB beacon ranges, and dead reckoning feeding on inertial sensor measurements. The third type of configuration for the system involves the replay of ROS logs containing sensor measurements, allowing the development of location algorithms fed by data obtained in previous executions in physical or simulation environments.

The thesis presents the design, implementation, and an evaluation of AugBot. The corresponding source code is available at https://github.com/JoaoCarvalho99/Augbot.

1.3 Thesis structure

Chapter 2 - **Background:** This chapter provides the necessary background to understand this thesis including: concepts and technologies in indoor location; information on the software and hardware components used in the dissertation; a discussion of related work in the state-of-the-art; and also relevant background information on the Augmanity project.

Chapter 3 - **State of Art:** This chapter describes some articles related to the work proposed. These articles were chosen due to similarities to this work regarding indoor location and robotics.

Chapter 4 - **Design and Implementation:** This chapter presents the design and implementation of AugBot. We identify the main starting requirements and the ROS-based design along with the relevant implementation details. Three types of deployment are presented regarding the use of AugBot: one involving the use of the AlphaBot2 robot and a UWB beacon infrastructure; the second involving a simulation framework for AlphaBot2 using the Gazebo simulation environment; and, finally, the use of ROS logs in replay mode.

Chapter 5 - **Experiments and Results:** This chapter presents an evaluation of AugBot and the associated results. Three case-study experiments are presented in correspondence to each of the types of deployment described in the previous chapter.

Chapter 6 - **Conclusion:** This chapter concludes the dissertation with a final discussion of the contributions and of future work.

Chapter 2

Background

This Chapter will provide an overview of the currently available and studied solutions to estimate position of target in indoor systems. It will also compare these solutions and how they fare in indoor location systems.

2.1 Indoor location

The ability to get an accurate location of a person or object has led to a number of so-called location-based services, mainly in logistics but also in more consumer applications such as localized weather information and navigation. These services work well when a clear view of the sky enables GNSS to pinpoint your location with sufficient accuracy. However, in an indoor environment the signals will be badly disrupted by the material between the user and the satellites, making it significantly harder to use the system to determine your location [6].

There are many promising technologies to estimate locations in indoor environments such as: light-based communication, computer vision, Ultra-Wide Band, WLAN, Bluetooth, inertial sensors, RFID-based solutions and WiFi-RTT.

Probably there will not be a one-technology-fits-all-solution, meaning that several technologies will coexist, each one with their own purpose and strengths. For these kind of systems, accuracy and response times are key requirements, but achieving the desired levels of performance may be challenging in an indoor environment due to the presence of refractive surfaces, interfering technologies and obstacles. When comparing different indoor positioning systems, several problems can be seen regarding the performance of said systems.

- **Accuracy:** critical to ensure that the data received is suitable for the intended application. Can be seen as the distance between the measured position of the object and the actual position.
- **Latency:** when considering a moving target, latency is critical because high latency results in high inaccuracy.

- **Power Consumption:** needs to be low to make the system small, portable and flexible.
- **Scalability:** the number of targets that the system can track at the same time and must easily adapt to different numbers of targets to be tracked.
- **Robustness:** affects how reliable the accuracy will be. Having low maximum errors and small disturbances are important factors, and being abe to handle non-line of sight (NLoS) is needed.
- **Complexity:** refers to how difficult and time-consuming it is to install or calibrate a system in a new environment.
- **Cost**
- **Portability:** describes how easy the hardware that is used can move around.
- **Security and integrity:** very important factor since the system keeps track of movements of human which could be seen as violating their integrity. Since the system keeps track of people or objects carried by people and may save the movements of persons it might be a security risk, making the security mandatory.

2.1.1 Important Concepts

This section explains what factors must be addressed in an indoor location system. To begin, the many aspects that cause difficulty while functioning in indoor locations are outlined, along with the limits they impose. The methods for estimating the target's location are then discussed, as the various ways for obtaining the distance to the target from the reference sites required for estimating the position.

2.1.1.1 Spectral interference

Any wireless system that uses electromagnetic waves for information transfer must follow specific criteria governing what frequencies can be utilized to prevent various devices from interfering with each other's transmissions. In Europe, these frequency bands are governed by the Electronic Communications Committee (ECC), which specifies which bands can be used for particular purposes in order to minimize equipment interference. Since bands are often more congested in interior contexts, spectral interference is a prevalent concern for most wireless transmission systems [6].

2.1.1.2 Line of Sight (LoS)

One of the most significant issues with indoor position estimation is that, unlike outdoors, where a clear view of the sky is almost always guaranteed for satellite communication, the space between the source and the target is frequently obstructed by people, walls, computers, and any other object present. This leads to two different situations when performing these types of distance estimations:

- **Line of Sight (LoS):** the path between a source and the target is free.
- **Non-Line of Sight (NLoS):** when the path is obstructed by some obstacle.

Because the LoS scenario is clearly characterized in terms of signal intensity fluctuations and electromagnetic wave propagation periods, all methods for location estimation operate well under these conditions, providing no spectrum interference is present. The NLoS example, on the other hand, is far more difficult to handle since it adds unknowns that are highly reliant on each environment and technology employed.

Electromagnetic waves may penetrate various things depending on the frequency of the wave and the substance of the blockage, although they are significantly attenuated and have a longer propagation time than by air. In rare situations, the signal in the direct channel is entirely obstructed, leading the target to be reached only by reflections or not at all [6].

NLoS can be described in Figure 2.1 where a tracking system can be seen trying to make contact with an object but there is an object between them blocking the signal.

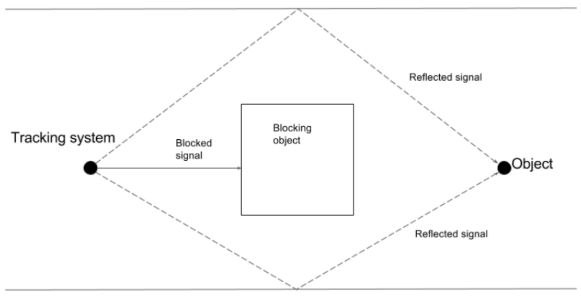


Figure 2.1: Communication in indoor location system with NLoS [6]

2.1.1.3 Multipath fading

Another issue brought on by complex indoor environments is a phenomenon known as multipath fading. The multipath effect is a phenomenon caused by receiving multiple replicas of the original signal as a result of signal reflections from objects in the environment or on surfaces between the transmitter and receiver [34]. These reflections arrive shortly one after another at the receiver, therefore signal's reflections from different surfaces may interfere with one another and since the reflected path is longer, determining the original signal from the reflected components can become difficult, potentially causing the time of flight or signal strength of the signal to become inaccurate [6].

Effects like these are difficult to predict and may result in inaccuracies for the majority of position estimation methods and if the pulse has large width (narrow bandwidth) the reflections will superimpose together, making the detection of the original signal almost impossible [34].

2.1.1.4 Distance estimation

In this section different types of distance measuring, methods are described. These methods estimate the distance between reference points and the target.

Time of Arrival (ToA) is the most straightforward of the time-based distance estimation methods. It estimates using one-way communications, which means that units are either dedicated transmitters or receivers, reducing their complexity. A ranging operation consists of only one transmission in which the transmitter sends the receiver a timestamp of the current time. The data is then compared to the time at the receiving instant, and the distance between the transmitter and the receiver can be calculated because the speed of light is a known constant. The accuracy of the ToA result, on the other hand, is highly dependent on the accuracy of the clock in both the transmitters and the receivers, which must be synchronized and have a very low drift, since a clock error of 1 ns results in an error of roughly 30 centimeters [6].

Time Difference of Arrival (TDoA) is very similar to ToA, but instead of calculating the time it takes for each message to travel from the transmitting point to the receiver, it uses the difference in arrival time from several known points to calculate the relative distance between them. TDoA necessitates strict synchronization between the reference points to ensure that the measuring signal is sent at the exact same time, but unlike ToA, the receiver is not required to share this synchronization because the relative difference in arrival time is measured rather than the absolute difference. Because the reference points are usually fixed in space, they can be linked with a wire, removing the need for more complex wireless clock synchronization algorithms [6].

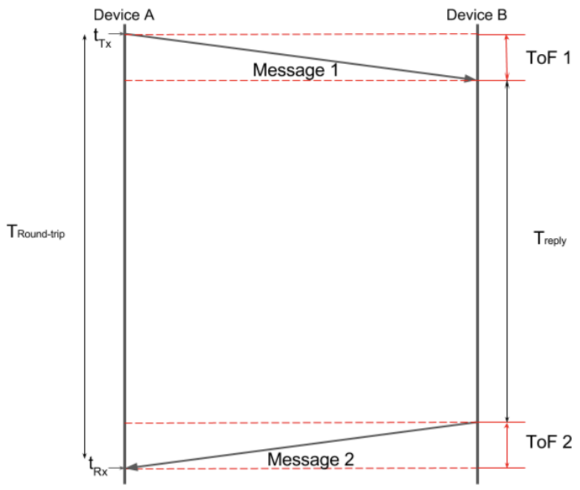


Figure 2.2: Message structure for ToF measurement [6]

Time of Flight (ToF) is a further extension of TDoA to remove the need for synchronization between points in the system. this is achieved by sending the measuring signal from a reference point to the target, which then responds after a known delay, Figure 2.2. This allows the reference point to calculate the total time of flight, therefore calculating the distance by:

$$Distance = C.\frac{(t_{Rx} - t_{Tx}) - t_{reply}}{2}$$

Where C is the speed of light at 3x10^8 (299792458) m/s, t_{Tx} and t_{Rx} are the transmission time of the first message and the reception time of the second message respectively, and the difference between these two becomes the total round-trip time. The distance can be calculated without any kind of influence of clock offsets between the involved nodes. Using this method, the clock error can be reduced to the drift that happens between t_{Tx} and t_{Rx} [6].

Angle of Arrival (AoA) uses the angles of two incoming signals to the receiver to determine its position relative to the two fixed reference points. This allows a position to be performed using one less fixed point than using one of the time-based methods, significantly reducing system hardware required, and also suffering from fewer error sources. However, determining the angle of an incoming signal with the required precision is significantly more complex than time stamping, and reflected signals have a significant impact on the accuracy in NLoS situations, so this method is much less commonly used than its time-based counterparts[6].

Received Signal Strength (RSS) uses reference points or searched objects as transmitters and the other side as receivers. This allows the receivers to obtain the signal strength in decibels that was transmitted from the transmitters. Many transceivers using various RF-techniques for communication are enabled to produce something called Received Signal Strength Indication (RSSI), which is a normalised value that uses a reference value at a distance of one meter from the transmitter as a baseline to ensure that the RSS can be easily read and used by a system. RSS is simple to use because it relies on simple measurements of signal strength, but there is a problem if the reference points and searched object are NLoS of each other because the signal can be absorbed by a variety of materials. RSS is usually used in low cost applications with a lower demand in accuracy since the results are notoriously inconsistent, lacking in either accuracy or range, and difficulties in handling mixed conditions. To obtain better estimations, extra algorithms or hardware are needed [6].

2.1.2 Technologies

There are several technologies available to solve the problem of indoor positioning, each with its own set of advantages and disadvantages. Many of them necessitate that the tracking target be equipped with some kind of hardware that communicates with the rest of the system in order to relay the information used to determine the target's position. This section will provide an overview of some of the technologies and highlight their key features.

2.1.2.1 GNSS

The term Global Navigation Satellite System (GNSS) refers to any satellite constellation that provides positioning, navigation, and timing (PNT) services on a global or regional scale [11]. GNSS can achieve high-precision positioning in the outdoor environment, allowing outdoor users to receive location-based services such as user positioning and navigation. However, GNSS signals cannot always arrive in an indoor environment and achieve positioning, or the signals that do arrive are too weak to achieve high precision positioning [4].

Because of the proliferation of smartphones, positioning technologies are now available to a wide range of users. GNSS are the most well-known and widely used technologies for outdoor localization. GNSS provide sufficient position accuracy in open sky conditions for most mass market applications. However, GNSS positioning accuracy may be significantly reduced inside buildings or in urban canyons. In such cases, GNSS signals may be affected by multipath effects, received at low power, or even blocked [10].

2.1.2.2 Bluetooth

Bluetooth can be used to estimate a target's location by measuring the RSSI. It enables phone tracking without the need for additional hardware on the tracking target, though a system of reference points must be installed. The addition of Bluetooth low energy (BLE) enables the development of small and energy-efficient hardware [6].

2.1.2.3 WiFi-RTT

WiFi Round Trip Time (WiFi-RTT) is a two-way ranging method. One of the most significant advantages of using WiFi-RTT is that it does not necessitate clock synchronization [4]. It enables computing devices to determine their indoor location and measure the distance to nearby WiFi access points with a precision of 1 meter using round-trip delay, by other words, the technology's operation principle is based on signal reception and transmission time delays. Basically, WiFi-RTT protocol allows us to estimate the distance between two WiFi devices [10].

2.1.2.4 UWB

Ultra-Wide Band technology is notable for its precision and robustness. This technology employs a wide range of frequency bands to allow the transmission of high-energy pulses while minimizing interference with other RF equipment operating at the same frequencies. To determine the target position, UWB-based systems typically use time-based methods.

UWB has good accuracy assuming high precision when measuring travel time, and it supports NLoS conditions because the wide frequency band is resistant to interference caused by reflected signals, and the signal's high energy content can penetrate many softer materials. Because obstruction is common in normal indoor environments, UWB is an excellent choice for general-purpose systems rather than specialized systems where LoS can be guaranteed. The high-energy signal's long range implies excellent scalability, and the technology reduces the need for complex processing of the obtained results. This precision is due primarily to the fact that signal energy is dispersed over a wide frequency range, allowing for very accurate time-stamping of incoming messages. The main disadvantage of UWB is that the technology is not as mature as many of the others, which raises the cost of specific hardware and limits the amount of support available [6].

The Federal Communications Commission (FCC) has allocated 7.5 GHz of spectrum for unlicensed use of Ultra-Wide Band devices (UWB) in the 3.1 to 10.6 GHz frequency band. UWB is emerging as a solution for the IEEE 802.15.3a (TG3a) standard. The purpose of this standard is to provide a specification for a low-complexity, low-cost, low-power consumption, and high-data-rate wireless connectivity among devices within or entering the personal operating space. The data rate must be high enough (greater than 110 Mb/s) to satisfy a set of consumer multimedia industry needs for wireless personal-area networks (WPAN) communications. The standard also addresses the quality of service (QoS) capabilities required to support multimedia data types [2]. Since the signal period of UWB is less than a nanosecond, these short impulses allow multiple transmitters to operate in parallel while mitigating the multipath effect [34].

2.1.2.5 Visual Light Communication

Visual light communication (VLC) uses different types of light emitters and detectors to determine the targets position within the area. Indoor positioning systems based on VLC can be partitioned into two different types. The first type uses a grid of rather basic photodiodes to receive light from the tracking target, and based on which detectors can see the emitted light the position can be computed. This allows for a system consisting of cheap hardware, but does not allow the target to know its position without extra communication, and does not support more complex setups with multiple targets. The second type uses a camera as a detector on the target together with stationary emitters which each send a unique ID by blinking a binary sequence, allowing the target to compute its position based on the angle to each individual emitter. This method is very exact and allows an almost unlimited amount of targets, however the cameras are rather expensive and demands LoS, making it unsuitable for dynamic environments [6].

2.1.2.6 Inertial Sensors

Inertial sensors provide information about the movement of the target.

Accelerometer provides the speed of the moving object and **Gyroscope** gives the angular velocity to determine the direction of the movement. These units together are called an Inertial Measurement Unit (IMU), with the possibility of adding:

- **Magnetometer** to improve gyroscope's measurements.

- **Barometer** to get altitude if a position in three dimensions is needed by measuring pressure.

2.1.3 Algorithms

2.1.3.1 Multi-lateration

Tri-lateration

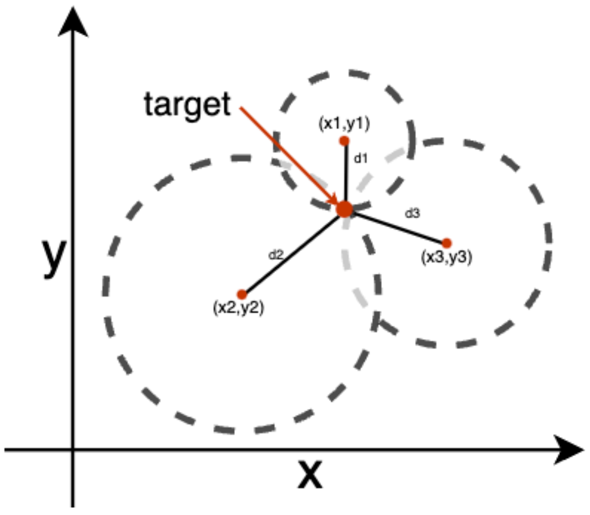


Figure 2.3: Tri-lateration example

The tri-lateration method uses geometry to estimate the target position of the mobile targets. For estimating the 2D position using multi-lateration, at least three base stations/anchors are required as represented in Figure 2.3 [16].

Least Squares

Least Squares is a classic positioning algorithm for multi-lateration [4]. When measuring distances between multiple transmitters and the target are gathered, the equating can be established as follows:

$$\begin{cases} (X_1 - x)^2 + (Y_1 - y)^2 + (Z_1 - z)^2 = d_1^2 \\ (X_2 - x)^2 + (Y_2 - y)^2 + (Z_2 - z)^2 = d_2^2 \\ \ldots \\ (X_n - x)^2 + (Y_n - y)^2 + (Z_n - z)^2 = d_n^2 \end{cases} \tag{2.1}$$

Where (X_i , Y_i, Z_i) i = 1,2,...,n represent the position of the reference points, (x,y,z) the position of the target and d_i i = 1,2,...,n is the measured distance between the reference points and the target.

2.1.3.2 Dead Reckoning

Dead Reckoning (DR) uses the last known positioning of the target combined with data collect by Inertial Sensors about its movement to estimate the new position after a short delay. DR systems are prone to error propagation since every new position is based on the previous estimation. Therefore, frequent calibrations are necessary to ensure that the position is reliable, this makes DR systems impractical by itself, and is often used to improve the accuracy of other technologies [6].

2.1.3.3 Sensor Fusion

In order to improve systems and overcome each technology's drawbacks, combinations of technologies may be a good option. **WLAN together with BLE** manages to get a better accuracy while still maintaining a better range in comparison to then individually, **UWB with DR** would give increased accuracy but requires more hardware, as well as more processing of the data to estimate the position, **UWB and WLAN** opens the possibility to utilize the advantages of WLAN and UWB while canceling most of their respective weaknesses. This allows building apps and communicating over WLAN while still maintaining the high precision of UWB [6].

2.2 Software components

2.2.1 ROS

Robot Operating System (ROS) is an open-source software development kit for robotics applications. ROS offers a standard software platform to developers across industries that will carry them from research and prototyping through to deployment and production [27]. The fundamental concepts of the ROS implementation are nodes, messages, topics, and services.

Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale: a system is typically comprised of many nodes. This term arises from visualizations of ROS-based systems at runtime: when many nodes are running, it is convenient to render the peer-to-peer communications as a graph, with processes as graph nodes and the peer-to-peer links as arcs [25].

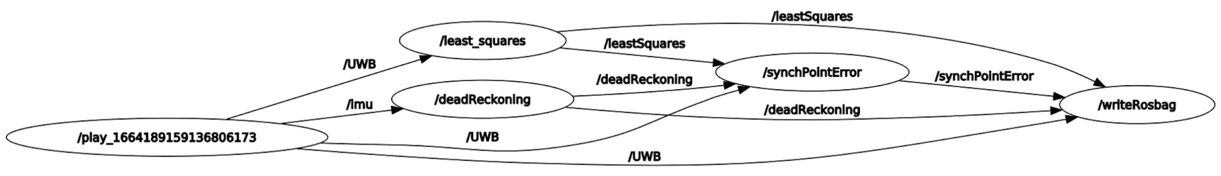


Figure 2.4: Representation of the structure of a ROS-based system. Nodes are oval shaped and arrows represent ROS topics

Messages are used to communicate between nodes. A message is a data structure that is precisely typed. Standard primitive types (integer, floating point, boolean, and so on) as well as arrays of primitive types and constants are supported. Messages can be made up of other messages, and arrays of other messages can be layered indefinitely deep. A node transmits a message by publishing it to certain buses known as topics. A node that is interested in a certain type of data will subscribe to the appropriate topic. A single topic may have several concurrent publishers and subscribers, and a single node may publish and/or subscribe to numerous topics as long as they have the appropriate message type. In general, neither publishers nor subscribers are aware of the presence of the other [25]. This structure can be observed in Figure 2.4, where the structure of the execution of a ROS-based system is depicted. The nodes are represented in an oval shape and they communicate through topics, represented as arrows.

Although the topic-based publish-subscribe model is a versatile communication paradigm, its broadcast routing technique is incompatible with synchronous transactions, which might simplify the design of some nodes. This is referred to as a service in ROS, and it is defined by a string name and a pair of strictly typed messages: one for the request and one for the answer.

2.2.1.1 Use Cases

In this section, we will go through some of the features that ROS's open design offers for the development of a wide range of tools. In addition to discussing the ROS approach to various use cases, we will provide a variety of tools intended to be utilized with ROS [25].

Logging and Playback:

Robotic perception research is frequently conducted with recorded sensor data to allow controlled comparisons of various algorithms and to simplify the experimental approach. ROS facilitates this technique by offering generic logging and playback capabilities. Any ROS communication stream can be saved to disk and replayed later. Importantly, all of this can be done from the command line, with no changes to the source code of any of the applications in the graph. As previously said, node instantiation may be accomplished simply by initiating a process and it can be done at the command line, in a debugger, from a script, and so on [25].

A bag file in ROS is used to store ROS message data from topics and services. A bag file is represented by the extension.bag. Bag files are generated with the rosbag command, which subscribes to one or more topics and stores the message data in a file as it arrives. This file can play the same subjects that were recorded, or it can remap the existing topics [15].

Packaged subsystems:

Although each node can be run from the command line, repeatedly typing the commands to launch the processes could get tedious. To allow for *packaged* functionality such as a navigation system, ROS provides a tool called roslaunch, which reads an XML description of a graph and instantiates the graph on the cluster, optionally on specific hosts. The end-user experience of launching the navigation system then boils down to roslaunch *file_name.launch* and a single Ctrl-C will gracefully close all five processes. This functionality can also significantly aid sharing and reuse of large demonstrations of integrative robotics research, as the set-up and tear-down of large distributed systems can be easily replicated.

Visualization and Monitoring:

It is frequently important to examine some condition while the system is running while building and debugging robotics software. Although printf is a well-known approach for debugging programs on a single computer, it might be challenging to adapt this technique to large-scale distributed systems. Instead, ROS may use the connection graph's dynamic structure to tap into any message stream on the system. Furthermore, the separation of publishers and subscribers enables the development of general-purpose visualizers.

ROS has a variety of tools for troubleshooting, visualizing, and running simulations, such as tools like rqt_gui, RViz, and Gazebo. ROS supports high-end sensors and actuators since it has device drivers and interface packages for numerous sensors and actuators used in robotics [15].

Simple programs can subscribe to a particular topic name and plot a particular type of data, such as laser scans or images. However, visualization software that employs a plugin architecture is allowed by rviz program, which is supplied with ROS. Rviz allows to display robot positions and trajectories, and other data types may by dynamically building visualization panels. Similarly, a tool called rxplot provides the functionality of a virtual oscilloscope, plotting any variable in real-time as a time series [25].

Transformations:

Robotic systems often need to track spatial relationships for a variety of reasons: between a mobile robot and some fixed frame of reference for localization, between the various sensor frames and manipulator frames, or to place frames on target objects for control purposes [25].

To simplify and unify the treatment of spatial frames, a transformation system has been written for ROS, called tf. The tf system constructs a dynamic transformation tree that relates all frames of reference in the system. As information streams in from the various subsystems of the robot (joint encoders, localization algorithms, etc.), the tf system can produce streams of transformations between nodes on the tree by constructing a path between the desired nodes and performing the necessary calculations.

For example, the tf system can be used to easily generate point clouds in a stationary "map" frame from laser scans received by a tilting laser scanner on a moving robot. As another example, consider a two-armed robot: the tf system can stream the transformation from a wrist camera on one robotic arm to the moving tooltip of the second arm of the robot. These types of computations can be tedious, error-prone, and difficult to debug when coded by hand, but the tf implementation, combined with the dynamic messaging infrastructure of ROS, allows for an automated, systematic approach [25].

ROS Master:

ROS Master functions like a DNS server, associating unique names and IDs to ROS elements active in our system. When any node starts in the ROS system, it will start looking for the ROS Master and register the name of the node in it. So, the ROS Master has the details of all the nodes currently running on the ROS system. When any details of the nodes change, it will generate a callback and update with the latest delays. These node details are useful for connecting with each node. When a node starts publishing a topic, the node will give the details of the topic, such as name and data type, to the ROS Master. The ROS Master will check whether any other nodes are subscribed to the same topic. If any nodes are subscribed to the same topic, the ROS Master will share the node details of the publisher with the subscriber node. After getting the node details, these two nodes will interconnect using the TCPROS protocol, which is based on TCP/IP sockets. After connecting the two nodes, the ROS Master has no role in controlling them. If one of the nodes stops, the others will check with the ROS_Master once again. The same method is used for ROS Services. The ROS_MASTER_URI environment variable contains the IP and port of the ROS Master. Using this variable, ROS nodes can locate the ROS Master. In a distributed network, in which computation is on different physical computers, we should define ROS_MASTER_URI properly, only then the remote nodes will be able to find each other and communicate with each other. ROS Master should run on a computer in which all other computers can ping properly to ensure that remote ROS nodes can access the MASTER [15].

2.2.2 Gazebo

Gazebo allows the simulation of robotic and sensor applications in 3D indoor and outdoor environments. It has a Client/Server architecture and has a topic-based Publish/Subscribe model. The Gazebo clients can access its data through shared memory. Each simulation object in Gazebo can be associated with one or more controllers that process commands for controlling the object and generating the state of that object. The Client sends control data and simulated objects' coordinates to the Server which performs the real-time control of the simulated robot. It is possible to realize a distributed simulation by placing the Client and the Server on different machines. Deploying ROS Plugin for Gazebo helps to implement a direct communication interface to ROS, thus controlling the simulated and the real robots using the same software. This provides an effective simulation tool for testing and development of real robotic systems [28].

2.2.3 MQTT

MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT allows for messaging between device to cloud and cloud to device. This makes for easy broadcasting messages to groups of things, can scale to connect with millions of IoT devices, guarantees reliability, and allows easy encryption by using TLS and authenticating clients using modern authentication protocols. Messages are published to a broker on a topic, then the MQTT broker filters messages based on the topic, and then distributes through the subscribers and does not store messages. A client can receive these messages by subscribing to that topic on the same broker, therefore there is no direct connection between a publisher and subscriber. All clients can publish (broadcast) and subscribe (receive) [21].

2.3 Hardware components

2.3.1 AlphaBot2-Pi

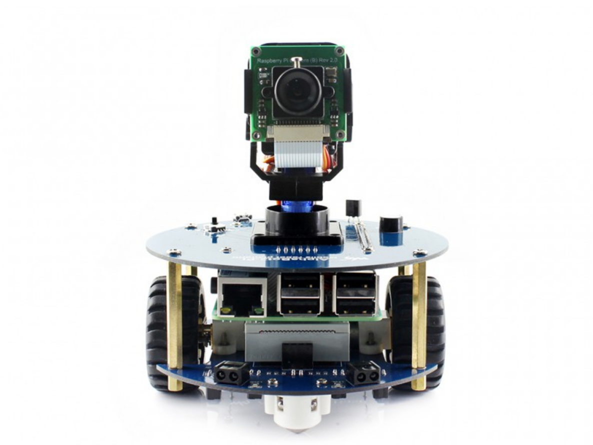


Figure 2.5: AlphaBot2-pi [32]

AlphaBot2-Pi, Figure 2.5, is a compact two-wheeled robot with a Alphabot2-Base chassis and an AlphaBot2-Pi adapter board. It features rich common robot functions including line tracking (due to 5 Infrared sensors in the lower part of the chassis), obstacle avoiding (due to 2 Infrared sensors in the upper part of the chassis), Bluetooth/Infrared/WiFi remote control, video monitoring and more. The highly integrated modular design and available source demo code, makes this robot easy to get started with [32].

2.3.2 Micro:bit

The BBC micro:bit, Figure 2.6, is a pocket-sized computer. It has an LED light display, buttons, sensors and many input/output features, such as: [18]

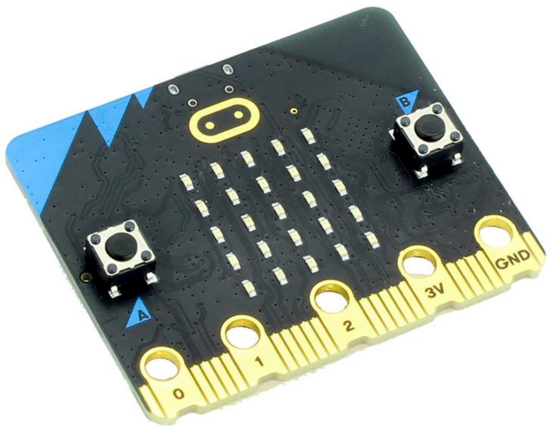


Figure 2.6: micro:bit [18]

- Temperature sensor
- Compass: It can measure magnetic fields in three dimensions
- Accelerometer: measures forces in 3 dimensions, including gravity

2.3.3 Pi Pico

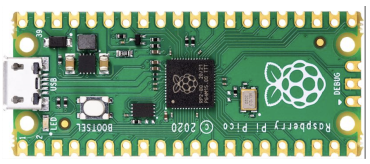


Figure 2.7: Raspberry Pi Pico [22]

Raspberry Pi Pico, Figure 2.7, is a low-cost, high-performance micro controller board with flexible digital interfaces. It has the RP2040 micro controller chip, which was designed by Raspberry Pi and features a dual-core Arm Cortex-M0+ processor with 264kB internal RAM and support for up to 16MB of off-chip flash. A wide range of flexible I/O options includes I2C, SPI, and Programmable I/O (PIO) and is programmable in C and MicroPython [22].

2.3.3.1 Pico 10DOF IMU

The Pico-10DOF-IMU, Figure 2.8, is an IMU sensor expansion module specialized for Raspberry Pi Pico. It incorporates sensors including a gyroscope, accelerometer, magnetometer, and baroceptor, and uses an I2C bus for communication. Combined with the Raspberry Pi Pico,

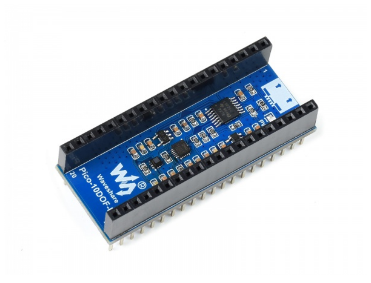


Figure 2.8: Pico 10DOF IMU [31]

it can be used to collect environment sensing data like temperature and barometric pressure or to detect motion gestures and orientations [31].

2.3.4 DWM1001

Figure 2.9: DWM1001 [8]

The DWM1001 Ultra-Wide Band (UWB) transceiver module, shown in Figure 2.9, allows users of this development board to easily assemble a fully wireless real time location system without designing any hardware or writing a single line of code and quickly progress into developing an application. In this thesis we used the MDEK1001 Ultra-Wide Band (UWB) Transceiver Development Kit, which brings 12 DWM1001-DEV development boards with embedded firmware binaries, with a UWB cable for flashing and APIs to configure and control the module via UART/SPI/Bluetooth, through a gateway or a tablet/smartphone.

The Qorvo DWM1001 module, Figure 2.10, is based on the DW1000 Ultra-Wide Band (UWB) transceiver IC which is IEEE 802.15.4-2011 UWB compliant. The module integrates a UWB and BLUETOOTH® antenna, all RF circuitry, Nordic Semiconductor nRF52832, and a motion sensor. The DW1000 uses a 38.4MHz reference crystal to reduce the initial frequency error to approximately 3ppm [8].

The DWM1001 module can be configured to behave as an *anchor*, a fixed node in the system, or as a *tag*, the mobile module to be located in the system. There are also 2 other modes, a *listener*, which collects the position of every *tag* in the system, and *gateway*, which supposedly can get every information possible from the system. Since the main goal of using this device was to collect the estimated distance from the *anchors* to the *tags*, the *listener* configuration was discarded, since it only provided the estimated location.

The minimal configuration needed in order to get estimations is with 4 devices, 3 of them configured as *anchors* and 1 as a *tag* and the configuration of the module may be achieved either via Bluetooth using the Decawave DRTLS Manager Android App, via SPI or UART connection from an external host or via USB serial port connection, and these modules estimate the distance between two devices with the Time-Of-Flight method.

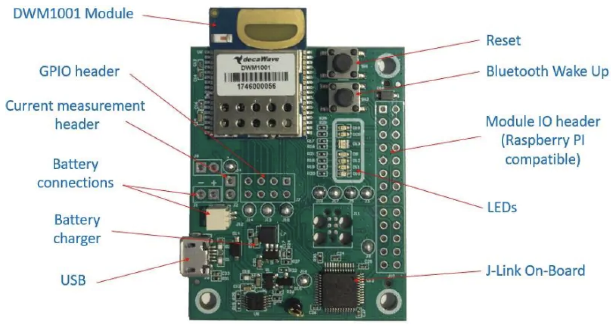


Figure 2.10: DWM1001 module [8]

2.4 The Augmanity project

The Augmanity (Augmented Humanity) Project (reference POCI-01-0247-FEDER-046103) is a project led by Bosch Termotecnologia, S.A, with the participation of many partners/co-promoters/participating institutions including FCUP and FEUP, but also for instance University of Aveiro, Franhaufer Portugal, Altice Portugal, among several others. It is funded by Portugal 2020 under the POCI and by European Regional Development Fund (ERDF).

The project is centered on the development of user-friendly, immersive, and supportive technologies in industrial production environments in different realms [7, 9, 24], e.g., industrial IoT, artificial vision and augmented reality, big data and predictive analysis of industrial processes, or worker health.

FCUP is involved in a work package identified as PPS3 and with the title “Industrial Internet of Things and connectivity” [1]. Within PPS3, FCUP is responsible for tasks concerning the development of low latency, high availability, and high accuracy solutions for indoor location. Indoor location is required for tracking moving assets (e.g., packages, AGVs, workers) in the factory floor.

In particular, FCUP is responsible for a task identified as T17.3 and titled “Development of geolocation modules” [1] that relates to the development of a physical IoT geo-location environment suitable for development and prototyping, comprising:

- physical IoT devices amenable to lab and factory floor testing, including small-size, low-cost AGVs and wearables that require location estimates with high precision;
- end-device and infra-structural sensors for geo-location, including UWB-based hardware;

- and a simulation environment for test and development purposes, able to mimic AGV movement.

These requirements are in direct correspondence to the developments in this thesis.

In addition to these general requirements, our choice of UWB hardware is motivated by the development of a special hardware device known as the “5G Tag” [23], to be used for a number of purposes in the project including indoor location. The 5G Tag, of which a prototype is shown in Figure 2.11, is being developed by Globaltronic, one of the partners in project. The UWB hardware incorporated in the 5G Tag is based on the DWM1000 UWB transceiver from Decawave described earlier in this chapter. Accounting for future hardware-software integration tasks in the project, our developments employs the same type of UWB hardware. The 5G tag is also planned to include inertial measurement sensors that can be used by indoor location algorithms.

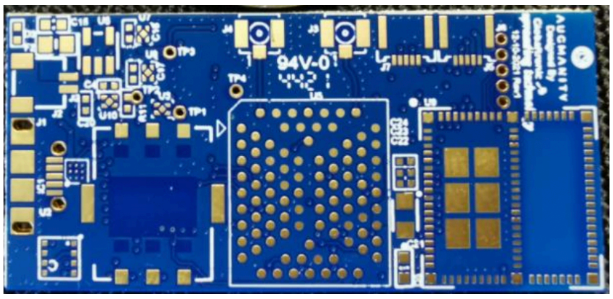


Figure 2.11: 5G Tag prototype [23]

Chapter 3

State of Art

System	Sensors	Algorithms	Robot(s)	ROS-based	Simulation Support	Open-source
AugBot	UWB, inertial sensors	Multi-lateration (least squares) and dead reckoning	AlphaBot2	Yes	Yes	Yes
Atlas [29]	UWB and optical reference system	TDoA for UWB measurements	Dr. Robot Jaguar V2	No	No	Yes
Atlas FaSt [30]	UWB and optical reference system	TDoA for UWB measurements	Dr. Robot Jaguar V2	Yes	No	Yes
Bostanci et al. [3]	UWB and LiDAR	Multi-lateration (least squares) and light detection initialization algorithm	TurtleBot 3	Yes	Yes	Yes
Guan et al. [12]	VLC	Kalman Filter Tracking Algorithm based on improved Camshift Algorithm	TurtleBot3	Yes	No	No
Okumus et al. [20]	Odometry	Algorithms based on linear and angular accelerations	TurtleBot2	Yes	Yes	No
Mishra et al. [19]	Microsoft Kinect XBOX 360	Adaptive Monte Carlo Localization	TurtleBot	Yes	Yes	No
Cheng et al. [5]	ZigBee	Least squares and Cramer-Rao bound to hybrid RSSI and TDOA measurements	?	No	Yes	No

Table 3.1: Comparision of AugBot framework with some related systems

In this Chapter, some state-of-the-art articles are discussed in regard to similarities to the framework developed in this thesis concerning indoor location and robotics, and particularly systems that use UWB for indoor location and are implemented using ROS. Table 3.1 provides a comparative overview between systems. We discuss each individual work in a separate section.

3.1 ATLAS

ATLAS [29] is an Open-Source TDOA-based Ultra-Wide Band Localization System that proposes and demonstrates a unique technique for a multi-user TDOA-based localization system using wireless clock synchronization. A sophisticated experiment involving robotic movement and an optical reference system as ground truth is used to test system accuracy. The Dr. Robot Jaguar V2 robot was the robot employed for the studies. The UWB firmware utilized in the ATLAS experiment, like in this thesis, was a Decawave module, in this instance the DWM1000, and the robot's velocity is similarly controlled by PID. This study demonstrates that the accuracies achieved by TDOA positioning with wireless clock synchronization are equivalent to similar TWR-based techniques, and all raw samples, reference data, and processed positions are provided with the software for comparability.

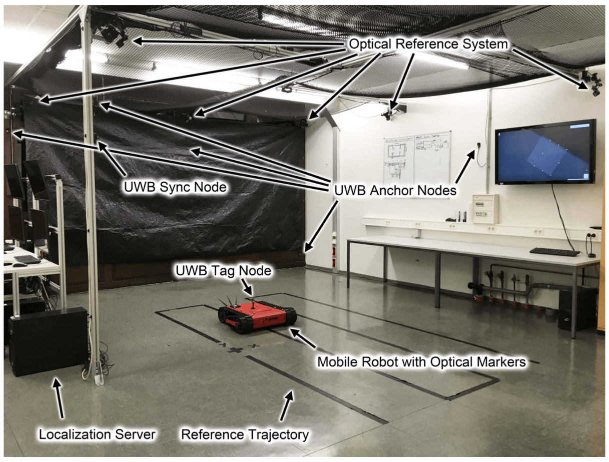


Figure 3.1: Top-down illustration of the scenario used for experimental evaluation of the system accuracy.

The ATLAS localization system is composed of modular components. A collection of static nodes (anchors) is dispersed along the localization environment, which can be observed in Figure 3.1. These nodes are linked by a wired backbone, which is controlled by a single micro-controller unit (MCU) and allows the nodes to connect to the localization server via serial connection.

Clock synchronization is required at the reception nodes due to the distinct clock drift of each anchor node and to comply with the accuracy required for reliable TOA measurements. The synchronization node broadcasts precisely timed periodic synchronization frames at a frequency of 10 Hz and due to the periodicity of those frames, a reference clock is can be generated to estimate the clock drift of the receiving anchors.

As depicted by Figure 3.2, the ATLAS localization server is the central application used for positioning and is in charge of managing and configuring synchronization nodes and anchors, receiving, matching, and assembling samples, clock correction, and positioning, and ultimately logging and reporting samples and results.

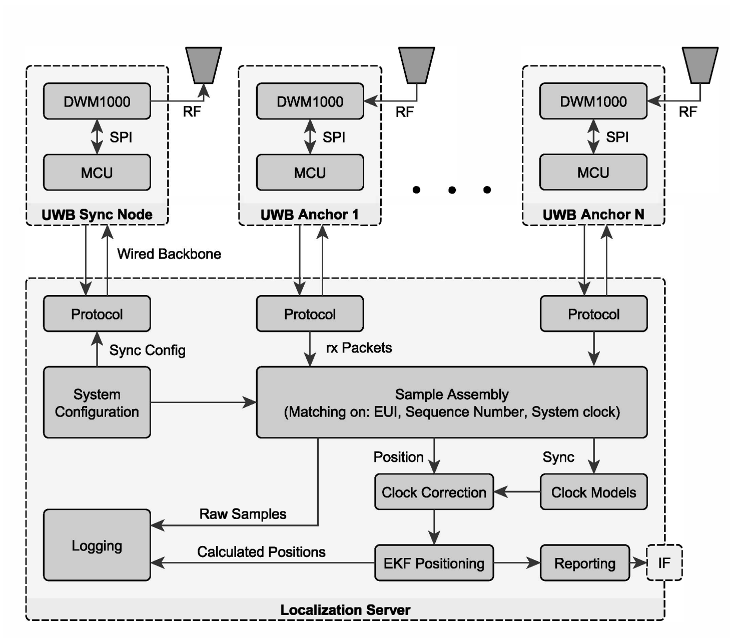


Figure 3.2: Schematic illustration of the system architecture for the ATLAS localization system.

The anchors, after configuration, start reporting received frames with precise TOA timestamps and a unique identifier to the sample assembly engine, while the server-side ensures that only tags beloging to the system are processed. Each anchor has its own clock model, which is updated whenever a synchronization frame is received. If a positioning frame is received, that model is used to calculate the corrected TOA with respect to the synchronization node clock.

Because early experiments revealed static offsets in the receiver's TOA measurements, the option to initially calibrate the system using a calibration node at a known point was added in the system. Since the node's location is known, the predicted TDOAs can be determined. By comparing the predicted and the measured TDOAs, a calibration offset for each anchor may be calculated. This offset is then deducted from the tags' measured TDOAs.

For error analysis, the localization system had to be matched against a reference system, therefore an OptiTrack motion capture system was implemented. Since the localization system and the optical tracking were executed on different machines, the clocks had to be synchronized, and because the motion capture system's frame rate is much higher than the UWB rate, the temporally closest subset of motion capture frames was chosen for comparison to the estimated position by UWB.

ATLAS FaSt: [30] Fast and Simple Scheduled TDOA for Reliable Ultra-Wide Band Localization, is an extension for the previously introduced ATLAS localization system, implementing the previously developed system in ROS to increase the capabilities of the system in terms of scalability and real-time capabilities, without degrading the performance of the localization results, and as observable in Figure 3.3, since the system is built upon ROS, the modules have no inter-dependencies between localization and scheduling. The software is once again open-source and the raw data and system configurations of the experimental analysis is also provided as a ROSbag to replayability. Another functionality introduced was the possibility of obtaining motion vectors, by integrating an IMU to the mobile node.

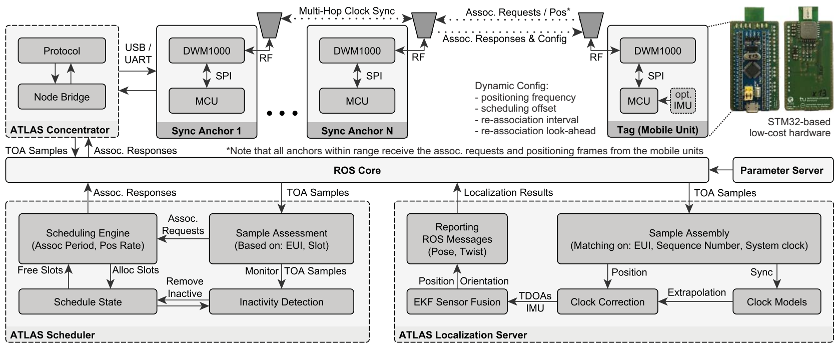


Figure 3.3: Schematic illustration of the system ROS-based architecture for the ATLAS localization system using FaST scheduling.

The current implementation only supports a static number of devices, however, FaST easily supports more than 1000 mobile units at 1 Hz. Energy consumption was also one of the main goals of this project, therefore a study was conducted to compare the energy consumption to other channel access schemes. By calculating the power per transmitted and received UWB frames, the maximal battery lifetime can be calculated, and the results were close to those of random-acces, which are the baseline for low power consumption. Therefore, FaST is capable of tracking many low-power devices without interfering with real-time requirements.

To evaluate the system performance, an experiment in an industrial environment was conducted. The experimental setup comprises a set of three mobile robots (based on Dr. Jaguar V2), 38 cameras for the optical reference system, and the proposed wireless localization system with one synchronization anchor and 8 passive anchors nodes (DWM1000 modules from Decawave).

3.2 Bostanci et al.

Bostanci et al. [3] proposes LiDAR and UWB-based source localization and initialization algorithms for autonomous robotic systems. This work discusses the source localization algorithm based on least squares approaches and the squared range measurements derived from UWB sensors for multi-lateration to locate the robot in an indoor environment, as well as the initialization algorithms based on LiDAR scans. The software is open-source and developed in ROS, with ROS modules for both real and simulated environments. The robot utilized in this experiment was the TurtleBot3.

The procedure starts in an initial starting pose and the goal destination is set manually on

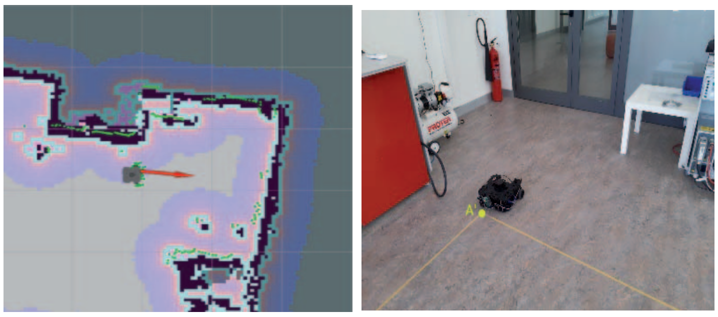


Figure 3.4: The left picture represents the map obtained by LiDAR in the graphical interface Rviz. Both pictures present the final position of the TurtleBot3.

rviz, which is a ROS graphical interface. To successfully reach the goal location as in Figure 3.4, the initialization, which uses both UWB and LiDAR data, has to be accurate. The variation of iterative closest point algorithms is used to math to the previously obtained map data and the current LiDAR scans, however, if the map data is quite big and has similar points, the matching process can be time-consuming and erroneous.

The UWB sensors are used to determine the robot's current position, and LiDAR scans are used to determine the direction the robot is facing. This is why proper LiDAR initialization is critical for the robot to arrive at the objective location, since if the robot does not know its right beginning position and where it is aimed, major mistakes will occur. Reference points are established throughout the scenario to corroborate the map data scanned by LiDAR and the estimated locations by UWB.

There were chairs and tables in the real experimental test setup, and whenever the robot got close to them, the localization performance declined dramatically, exactly as it did in this thesis.

The UWB simulation node in the ROS simulation environment subscribes to the gazebo/model states topic to obtain the robot's real position on the simulation. The range between the anchor and the robot's true location is then determined, and gaussian noise is added to give realism. The range is then published, allowing another ROS node to estimate the robot's position via multi-lateration. This approach is pretty similar to the UWB simulation used in this thesis.

3.3 Guan et al.

Guan et al. [12] presents an indoor robot localization system based on ROS and Visible Light Communication (VLC). To obtain dynamic positioning, this work provides a Kalman Filter Tracking Algorithm based on an improved Camshift Algorithm and an algorithm based on video target tracking. The purpose of this experiment is to address the low positioning accuracy and poor real-time capability of standard VLC positioning systems that rely on the AOA of visible light or the RSS of the light to determine the distance to each target.

Visible light wireless communication, also known as Light Fidelity (LiFi) technology is a wireless transmission technology that uses visible light spectrum for data transmission. It uses electrical signals to control the high-speed flashing LEDs to transmit information. Visible light positioning is then a positioning technology based on VLC. In this experiment, VLC was exploited for indoor robot positioning based on a double-lamp experiment platform and a TurtleBot3.

To acquire precise three-dimensional coordination of the object, the dynamic tracking detection of the Camshift algorithm, which is a high-precision VLC image positioning algorithm, is proposed. The TurtleBot3 receives the LED signal, converts the images to ROS image messages, and then publishes the message to a topic, which is listened by a node responsible for obtaining the LED coordinates and ID. The picture is then processed by another node, which computes the robot's location. Because the TurtleBot3 single board computer's CPU capability is insufficient for image processing and positioning computations, the camera node runs on the TurtleBot and the locator node on a remote controller. This is simple since the ROS platform simplifies a distributed design.

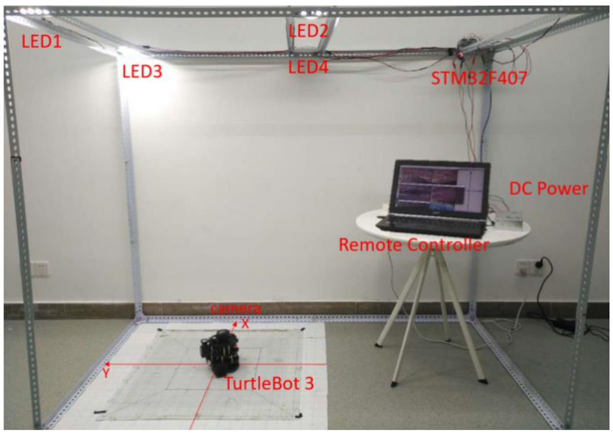


Figure 3.5: Top-down illustration of the scenario used for experimental evaluation of the system.

The experiment, represented in Figure 3.5, was carried out on an experimental platform with four LEDs, and since the robot cannot detect light on the far side at the edge, three or four LED luminaires were employed so that the robot could observe at least two LED luminaires at various positions. The information was shown in real-time using the 3D visualization tool rviz. The experiment begins with the robot at coordinate (0,0), and then the results of the positioned coordinate are obtained. When the robot is static, the dispersion radius of the estimations is utilized to characterize the positioning accuracy. As a result, the smaller the dispersion circle, the denser the data distribution, the higher the accuracy. The positioning error may be represented as the Euclidean distance between the data and the center of the dispersion circle if the center of the dispersion circle corresponds with the coordinates of the origin after adjustment. If the dispersion radius is narrow enough, after proper calibration of the camera, accurate data can be achieved.

3.4 Okumus et al.

Okumus et al. [20] presents a Cloud Based Indoor Navigation and Communication for ROS-enabled Automated Guided Vehicles. ROS is used to send and manage commands to TurtleBot 2, which has been the AGV used in the study. In this study, an application to manage multiple robots over a cloud-based system has been conducted. The technique used to estimate the position of the robot was based on the robot's odometry.

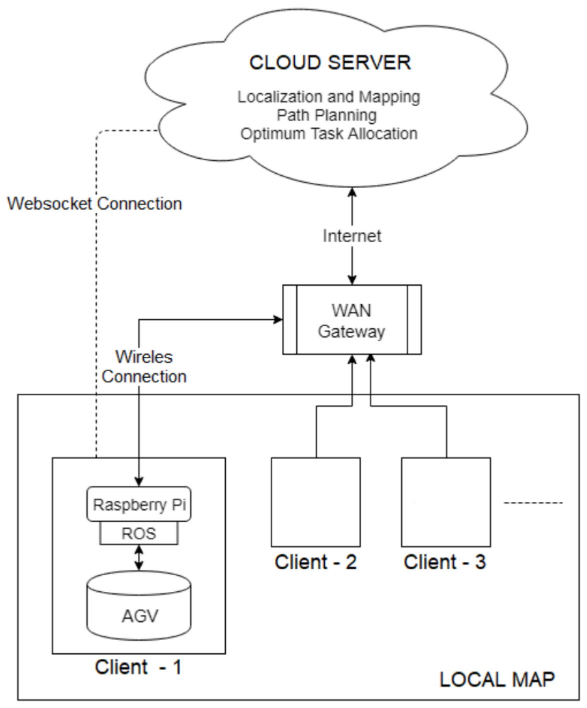


Figure 3.6: Architecture of System presented

To ensure that the AGV navigates toward the goal, the environment in which the robot is operating is mapped in the cloud system, where an optimum path is found by using path planning algorithms, since localization, mapping the environment, and detecting obstacles are required to control and navigate AGVs. When the map changes or obstacles move, the information is uploaded to the cloud system, updating the map. As described in Figure 3.6 the AGV communicates with cloud cloud via ROS installed on the Raspberry Pi.

To address localization issues, the odometry of the robot is picked up with the information obtained from the wheel encoders of the robot. This is one of the basic approaches for robot navigation that allows one to estimate the distance traveled by the robot by counting the number of wheel turns. Tests were done in both a simulated and a real-world environment to evaluate this strategy, and they appeared to be effective.

3.5 Mishra et al.

Mishra et al. [19] proposes a ROS-based service robot platform for mapping, localizing, and navigating in an indoor environment that aims to simulate a low-cost service robot platform that is capable of mapping an unknown environment using a RGB-D camera, the Microsoft Kinect XBOX 360. The approach used for mapping was based on Real Time Appearance Based Mapping and a TurtleBot robot model was used for visualization.

Microsoft Kinect XBOX 360 was the sensor chosen because it is a low-cost 3D RGB-D sensor that can effectively be used for Simultaneous Localization and Mapping (SLAM). It provides both the depth data with the color image. It can be used to capture 3D point cloud data from the target environment and can be further used for indoor navigation.

The first step is mapping a 2D projection of the indoor environment with the sensor Microsoft Kinect, then, in Gazebo, the autonomous navigation of the robot model to the desired location is performed to find the best path possible. In order to navigate successfully, the robot is located with the Adaptive Monte Carlo Localization algorithm, a Bayesian probabilistic based algorithm that uses pre-existing 2D maps.

3.6 Cheng et al.

Cheng et al. [5] describe an indoor robot localization based on wireless sensor networks and proposes a hybrid RSSI and TDOA approach to mitigate the substantial errors observed during indoor RSSI localization and the high cost associated with TDOA localization.

This research demonstrates how combining RSSI and TDOA data may result in reliable location calculations. This research utilizes two different node types: the ZigBee node, which employs a 2.4 GHz low-power radio chip and measures RSSI values, and a TDOA node, which measures TDOA values as despicted in Figure 3.7.

When the beacon node gets 50 packages, it filters RSSI values using an iterative recursive weighted average filter. Then, a polynomial fitting algorithm is used to convert RSSI to distance, and finally, polynomial parameters are defined using maximum likelihood estimation and statistical test methods. To obtain hybrid RSSI/TDOA localization, least squares and Cramer-Rao bound are calculated.

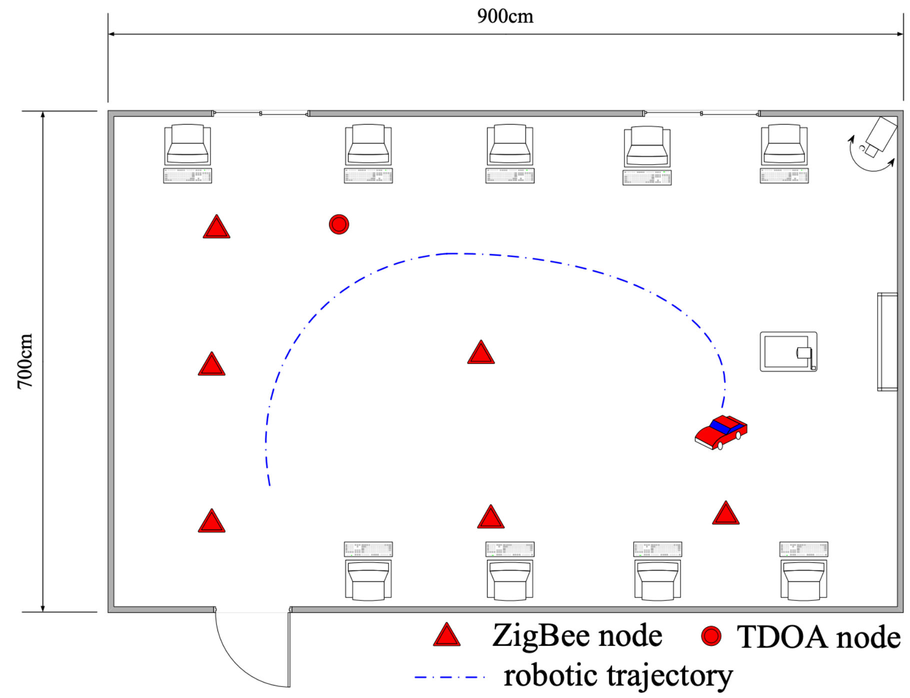


Figure 3.7: Floor plan of experimental site

Chapter 4

Design and Implementation

This chapter presents the design and implementation of the AugBot framework. Section 4.1 begins by listing the primary requirements for the AugBot framework. Section 4.2 provides a quick summary of the framework's primary parts as well as the three existing configurations assembled in the framework. The modules that are shared by all three configurations will be discussed first, followed by the modules that are used in the Robot configuration. The modules used in the Simulation configuration are next, followed by the modules used in the Replay configuration. The organization of the AugBot package will be explained in the final Section 4.3, with a brief description of each folder and the communication topology employed.

4.1 Framework Requirements

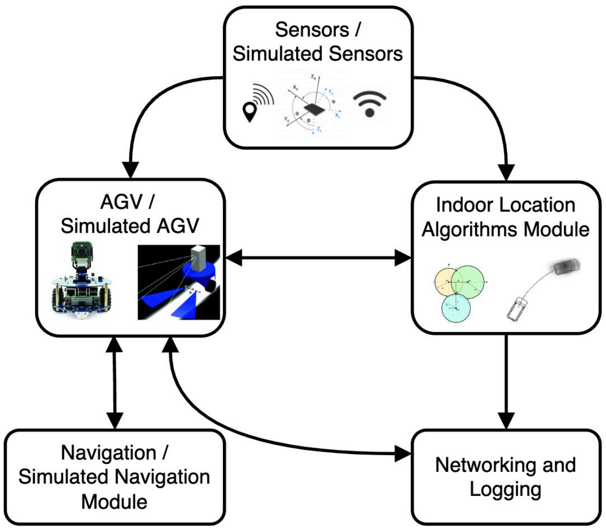

Sensors / Simulated Sensors
AGV / Simulated AGV
Indoor Location Algorithms Module
Navigation / Simulated Navigation Module
Networking and Logging

Figure 4.1: Framework requirements

The Augmented Humanity (Augmanity) project [9] consortium seeks to foresee the future by creating user-friendly, immersive, and helpful technology in industrial manufacturing environments.

The project's outcomes are planned to be leveraged across numerous industries. This thesis is part of Augmanity PPS3: Industrial Internet of Things and Connectivity, a sector devoted to putting I4.0 and edge technologies to work for individuals and corporations. In this context, there is the specific issue of providing an indoor location for tracking devices (e.g., robotic vehicles) or other physical assets (e.g., product bundles) on a manufacturing floor. FCUP and the other project partners decided on an indoor positioning system that would employ a UWB beacon infrastructure and inertial sensors as they were available and installed in devices.

We considered the following requirements for the AugBot framework, illustrated in Figure 4.1:

- study and experiment algorithms capable of performing indoor localization, using UWB is mandatory to comply with the work performed by other partners of the Augmanity project;
- develop a system with low latency, high availability and highly accurate that is capable of locating embedded devices, especially AGVs, in real-time;
- have the possibility in the system to send information about the tracking devices to the network;
- keep software with low execution footprint in terms of resource consumption, highly portable and easy to install and calibrate to be easily adapted to a new environment;
- develop a simulation environment for testing and development, while ensuring that the movement from simulated AGVs approximates to real ones.

4.2 Approach

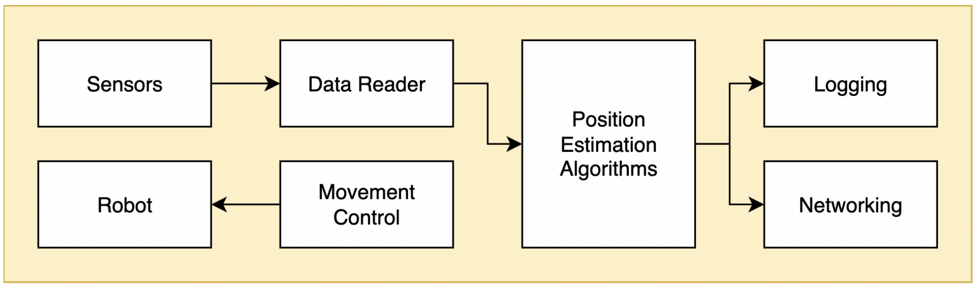


Figure 4.2: General representation of framework functionalities

To comply with the needs of the Augmanity Project, this thesis developed AugBot, a ROS-based framework capable of locating a robot in both real and simulated environments. The localization system is based on UWB measurements for multi-lateration plus inertial sensors to obtain motion vectors of the robot's movement for dead reckoning. In Figure 4.2 we can observe the approach of the framework, which consists in feeding location estimation algorithms with data, such as UWB ranges and motion vectors from sensors. Then, the positions estimated are both broadcasted to the network and stored in log files for further analysis. Finally, to move the robot, a robot controller is also defined.

The least squares algorithm estimates the robot's position doing multi-lateration with the positions of the UWB anchors and the respective estimated ranges to the robot. Dead reckoning estimates a new robot position, applying the motion received by the IMU to the previously estimated position. In the simulation, the actual position of the robot is always known. As in the real case, this doesn't occur, for ground truth 4 cardboard boxes were employed as control points. These control points have the function of detecting the instant of the passage of the robot in certain positions for subsequent analysis and comparison of position estimation algorithm outcomes.

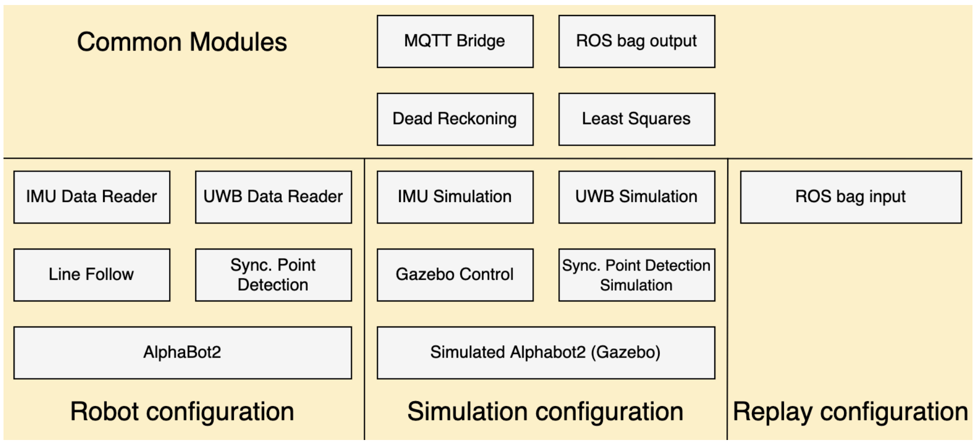


Figure 4.3: General representation of each configuration

To achieve these requirements, three different configurations are defined as shown in Figure 4.3. The first configuration allows the execution of the localization system in a real scenario with Decawave DWM1001 Development Kit to obtain UWB range measurements and Micro:bit to obtain motion vector data about the robot. The second configuration is a simulated AlphaBot2 moving in a Gazebo world while being tracked by simulated UWB ranges and simulated inertial sensors. This configuration allows us to experiment and test algorithms before implementing them in real scenarios. Finally, the third configuration allows replaying logs from previous experiments to replicate tests and compare results from different algorithms under the same circumstances.

Associating the modules in Figure 4.3 with the general operation shown in Figure 4.2, we have MQTT bridge to provide Networking capabilities, ROS bag output as the node responsible for Logging data, and the Least Squares and Dead Reckoning as position estimation algorithms in the common nodes. Then, in the Robot configuration, we have an AlphaBot2, and in the Simulation configuration, we have a simulated AlphaBot2. In these two setups, we have a module for controlling the robot, lineFollow in the Robot configuration, and Gazebo Control in the Simulation configuration, as well as a module to detect control points. The IMU Data Reader and UWB Data Reader read data from sensors in the Robot configuration, while in the Simulation the sensors are simulated. Since the Replay configuration replays logs from the prior experiments from the other two setups, it only requires one extra module.

4.2.1 Common modules

The common modules for every configuration are MQTT bridge, ROS bag output, Dead Reckoning, and Least Squares.

MQTT bridge is responsible to broadcast information from and to the outside of the system by MQTT. The main goal is to broadcast every estimated position from every algorithm by MQTT to the cloud.

ROS bag output listens to a variety of ROS topics during the experiments and writes them into two different ROSbags. One of them has the full data from topics during the experiment: data from IMU, every variant of Dead Reckoning, every variant of Least Squares, data from UWB, and timestamps of detection of the control points. The other ROSbag has the purpose of saving the sensor data of the experiment to use for a future experiment, detailed in Subsection 4.2.4, therefore it only contains the information from UWB, IMU, and control points.

Dead Reckoning listens to `/imu` to obtain the movement's orientation and to `/synchPoint`. Every time deadReckoning receives a message from `imu` it estimates 3 positions of the robot: one with constant speed to `/deadReckoning` (0.15 m/s), other to `/deadReckoningSP` with constant speed (0.15 m/s) as well, but listens from `/synchPoint` and every time that the synchronization point (control point A) is detected, the position is adjusted, and the last one uses the accelerations from `/imu` to estimate the speed of the robot to `/deadReckoningACC`.

leastSquares is the node responsible to estimate the position of the robot using the UWB's ranges with multi-lateration. Every time a message is listened from `/UWB`, one Least Squares estimation is sent to `/leastSquares` and another with moving average (the average of the last 10 leastSquares estimations) is sent to `/leastSquaresMA`.

4.2.2 Robot Configuration

The robot configuration is illustrated in Figure 4.4.

AlphaBot2 is controlled by a Raspberry Pi, therefore the most logical Operative System to flash would be Raspbian. The problem was that ROS Noetic was not compatible with Debian armhf, so Ubuntu was the chosen Operative System. AlphaBot2 comes with a demo code API from waveshare which was used to develop the code to control the robot's movement and detect the control points.

Micro:bit and Pi Pico were the available sensors to work as inertial sensors. Micro:bit was the first sensor used to obtain information about the AlphaBot2 movement. The objective was to collect both the accelerations and the orientation of the movement, but the accelerations were not precise enough, since the interval of the measurements was 1 millig which is about 0.15

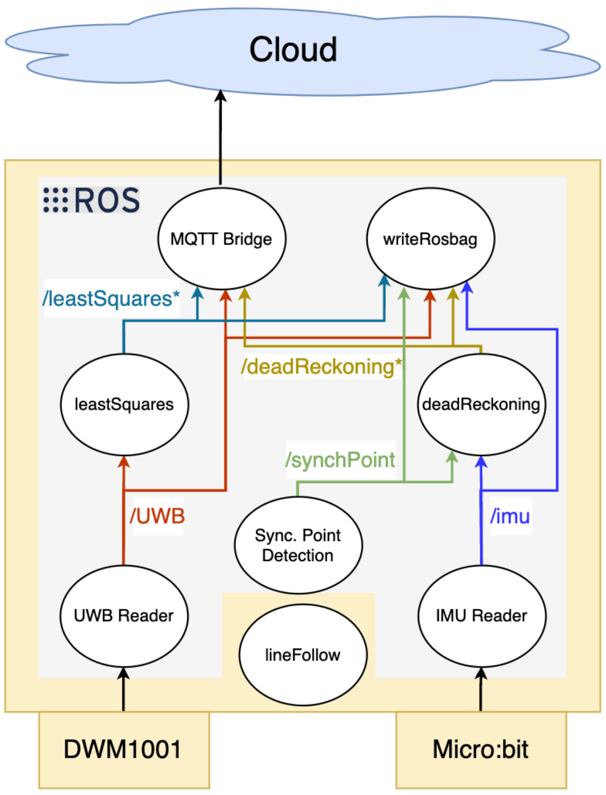


Figure 4.4: Diagram of Robot configuration

m/s^2, so only the orientations were used. A good calibration of the micro:bit was a must to obtain a precise orientation of the robot. This calibration was quite tricky due to the existence of metallic tables and chairs scattered across the room and the magnetic field created by the robot's batteries. Pi pico was not used in the latter stages of the thesis, due to difficulty in achieving both accurate accelerations and orientations. This was due to the lack of the option to correctly calibrate the sensor.

The configuration illustrated in Figure 4.4 runs on AlphaBot2, although it is possible to separate it in order to reduce the power consumption and computing requirements of the AGV. For example, in the experiments performed, we separated the computation as follows: in the server, we have `ROS_MASTER` running and `writeRosbag`. **writeRosbag** is the node responsible for the ROS bag output and in the second device (AlphaBot2), we have the nodes that are responsible to collect and process data.

From the bottom to the top, we have **UWB Reader** that reads and parses the data collected from DWM1001 configured as a tag to obtain the estimation of position by the UWB system and the ranges between the robot and the anchors. After parsing the data, it writes a `tagFull` message to the `/UWB` topic.

IMU Reader has similar functions to UWB Reader. It reads and parses data collected from micro:bit, to be more precise, accelerations and the orientation of the robot's movement. Converts the heading estimated by Micro:bit to a quaternion and converts the acceleration from millig to m/s^2 and sends both in a `sensor_msgs/Imu` message to `/imu`.

lineFollow moves the robot, determining the force to apply to each wheel according to the data received by AlphaBot2's ITR20001/T reflective infrared photoelectric sensor. The Line Follow program consisted in a proportional–integral–derivative controller (PID), which is a control loop mechanism that continuously calculates an error value and applies a correction based on proportional, integral, and derivative terms, hence the name. PID automatically applies an accurate and responsive correction to a control function. This algorithm with the position gathered by the Infrared Sensors of the robot manages the force to apply to each wheel, therefore forcing the robot to move forward when both wheels were submitted to the same force, and turn when different speeds were set. The value used for proportional was 0.6, integral was set to 0.00005, and derivative to 0. If the sensor estimates a position less than 2000, the black line is on the left side of the robot, and greater than 2000 is on the right.

Synch. Point Detection detects objects along the path. These objects are control points, and there are 4 along the course. When a control point is met, a synchPoint message is sent to `/synchPoint` with the timestamp of detection and the differential time between this control point and the previous one. The obstacles are detected with the Infrared photoelectric sensor (ST188) for obstacle avoidance and they can detect obstacles up to 10 centimeters away from them.

4.2.3 Simulation Configuration

The simulation configuration is illustrated in Figure 4.5. From bottom to top, we have the **Gazebo Simulation**, including a GUI where we can observe the Simulated AlphaBot2 moving. The simulated AlphaBot2, represented in Figure 4.6, moves according to `geometry_msgs/Twist` messages sent to the topic `/alphabot2/control`. These messages contain linear and angular velocities, in m/s, to move the robot. These messages are sent by **AlphaBot2 control** which is responsible to move the AlphaBot2 in the simulation and to simulate synchronization points. The robot, after completing each turn simulates a synchronization point and sends a message to `/synchPoint`. The robot's movement consists in turning 90 degrees after moving for 10 meters, adjusting the linear and angular velocities as necessary to keep a fluent movement.

The simulation is based on Alphabot2 ROS Package and Simulator [26]. However, since the track previously used in the simulation was too short for the intended, a script to move the robot was developed, executing loops in the shape of squares with 10 meters length.

IMU simulation simulates an imu with configurable orientation error. Just as the IMU node from the robot configuration in Section 4.2.2, it sends to `/imu`, the information about the robot's orientation, in quaternion form and the accelerations in m/s^2, using a `sensor_msgs/IMU` message. The IMU plugin used in the simulation is from Hector Gazebo Plugins [17].

UWB simulation, as the same suggests, simulates UWB. From the real position of the simulated robot obtained from the ROS topic `/tf` and the positions of the anchors set in the

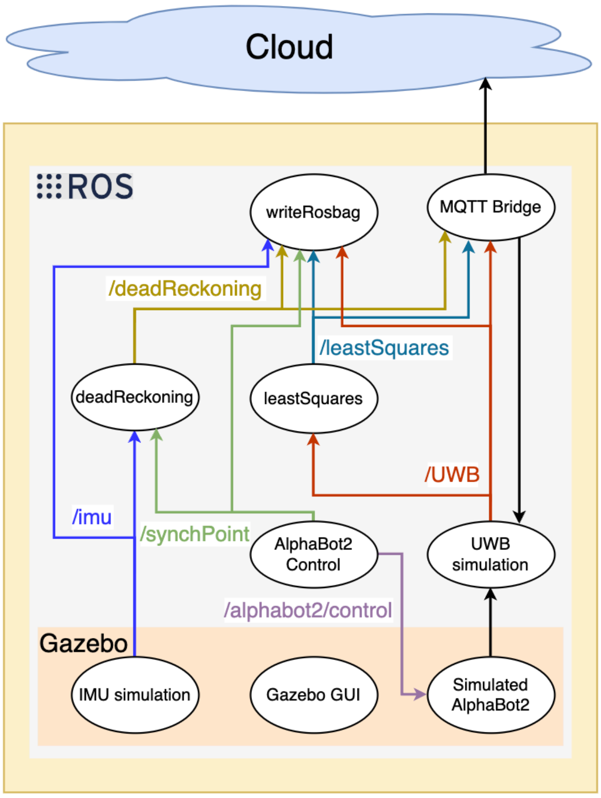


Figure 4.5: Diagram of Simulation

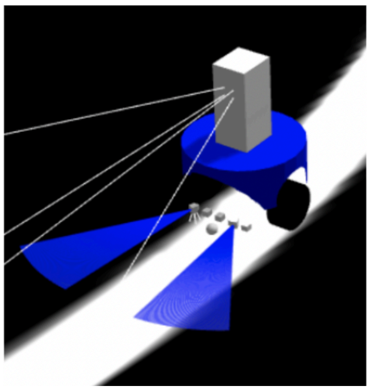


Figure 4.6: Simulated AlphaBot2 on Gazebo

configuration file (`config/uwb_simulation.yaml`), ranges are calculated between the robot and the anchors. To simulate noise in the simulation, in the same configuration file, there are two parameters to define the values for the mean and variance of the Gaussian Noise. According to these values, noise is added to every range calculated. The position from `/tf` and the ranges calculated are sent to `/UWB` in a `tagFull` message.

Then we have the common modules previously detailed in Section 4.2.1. **MQTT bridge**, in this case, has another function, which is to allow to change the positions of the simulated anchors through MQTT in real-time, by sending a ROS message to `/anchor_config` topic.

4.2.4 Replay Configuration

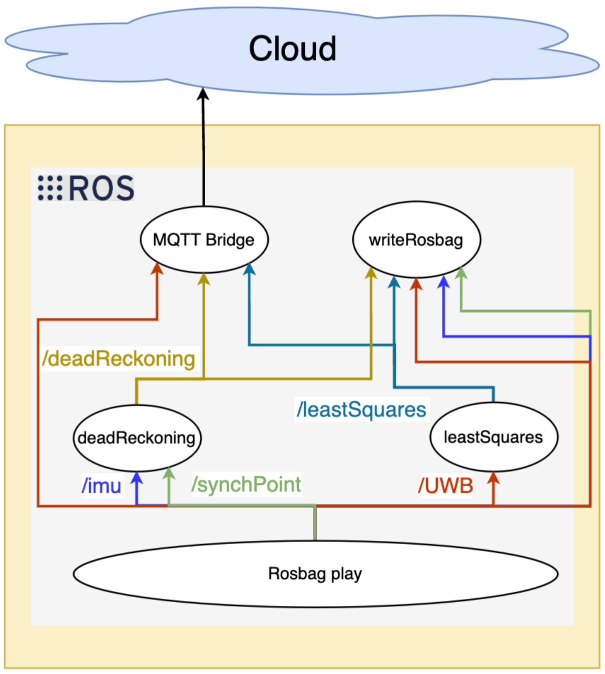


Figure 4.7: Diagram of Replay

In this last scenario, illustrated in Figure 4.7, the only new node is Rosbag play, that replays ROSbag containing `/imu`, `/UWB`, and `/synchPoint` data. By replaying these three topics in the same temporization as they were collected, we can redo the experiment and try new localization algorithms under the exact same conditions. Then we have the common modules described in Section 4.2.1.

4.3 Framework Structure

AugBot is developed using the ROS Noetic toolkit. It is organized as a ROS package, with the C++ source code in the `src` folder and the executable Python scripts in the `script` folder. In the AugBot package, we have the usual `CMakeLists.txt` and `package.xml`, along with a `launch/` folder, a `msg/` folder, and a `config/` folder. In `CMakeList.txt` we have the compilation directives for the package, the system's dependencies, and the message files created in the package.

The system depends on:

- **roscpp** and **rospy**, C++ and Python packages respectively;
- **std_msgs** to use ROS standard messages;
- **roslaunch** to launch `.launch` files to execute one or more ROS nodes with the necessary configuration file (`.yaml` file);

- the package **ROS Serial** for C++ [33], serial is a cross-platform library for using serial ports on computers;
- **ROS MQTT bridge** [13] that provides the functionality to bridge between ROS and MQTT bidirectionally. MQTT bridge uses ROS message as its protocol and the messages from ROS are serialized by JSON for MQTT, and messages from MQTT are deserialized for ROS topic, therefore MQTT messages should be ROS compatible.

4.3.1 src/ folder

In the `src/` folder we have the C++ programs described in Table 4.1.

File name	**Description**
`UWB_Reader.cpp`	source code for reading and parsing UWB data from Decawave firmware
`IMU_Reader.cpp`	source code for reading and parsing IMU data
`deadReckoning.cpp`	source code to estimate the new position of the robot with the dead reckoning algorithm

Table 4.1: C++ files in `src/` folder

4.3.2 script/ folder

File name	**Description**
`uwb_simulation.py`	simulates UWB data (substitutes UWB_Reader.cpp in the Simulation configuration)
`least_squares.py`	calculates multi-laterations with UWB range measurements
`rosmqtt_node.py`	initiates MQTT connection
`mqttBridge.py`	broadcasts information bidirectionally from and to the system and the cloud
`writeRosbag.py`	stores log from the experiments into ROSbags
`rosbag2scv.py`	converts ROSbags into CSV files
`AlphaBot2/python/lineFollow.py`	to control the robot's movement in the real robot configuration
`AlphaBot2/python/obstacleDetection.py`	detect the control points
`AlphaBot2_Control.py`	control the simulated AlphaBot2 in the Gazebo World, it defines the robot's movements, and simulates the detection of the simulated control points

Table 4.2: Python executables in `scripts/` folder

The `script/` folder contains the Python executables developed in the thesis and a description can be found in Table 4.2.

4.3.3 launch/ and config/ folders

In folder `launch/` we have the launch files to start each configuration of the framework (`real.launch`, `simulation.launch`, `replay.launch`), `writeRosbag.launch`, and `ROSmqtt.launch`, and in the `config/` folder the parameters for the initialization of some nodes.

4.3.4 msg/ folder and communication

To achieve the communication between the framework modules, 6 ROS message were created and all of them have the correspondent ROS MQTTbridge message to broadcast information from and to the cloud. These messages are defined in the `msg/` folder, each one in their respective file.

The **position** message is defined in `position.msg` and contains the coordinates x,y,z of the position of either the robot or of an anchor. This message type is used inside of other messages or simply to send the estimated position by one of the location estimation algorithms at that instant.

Type	**Variable**	**Description**
`float64`	`timestamp`	timestamp
`string`	`ID`	ID of the anchor which is unique for each anchor
`position`	`position`	position of the anchor
`float64`	`range`	range to the tag

Table 4.3: Structure of `anchor` message

The **anchor** message defined in `anchor.msg` was created to contain the information about each anchor, and its structure is described in Table 4.3.

The **estimate** message, defined in `estimate.msg`, contains the information about the estimation of the position of the robot by Decawave firmware, described in Table 4.4. This message is solely used inside of a `tagFull` message. **tagFull** is the message designed to contain data read from the Decawave firmware configured as tag and its structure is depicted in Table 4.5. This message is used in the ROS topic `/UWB`, to broadcast to the system the estimated position by Decawave firmware with the UWB range measurements used to calculate the position.

The next message is only used in the simulation context. **anchorConfig**, defined in `anchorConfig.msg`, is a message that is used to change the anchors setting in the simulation in real-time. It contains the number of anchors and the array of anchors, including their IDs and

Type	**Variable**	**Description**
`float64`	`timestamp`	timestamp of the UWB estimation
`int32`	`le_us`	time of computation that was needed to calculate the estimation of the position in microseconds
`int32`	`accuracy`	quality factor considered by the firmware
`bool`	`valid`	valid is TRUE whether an estimation was achieved and FALSE if not
`position`	`position`	contains the robot's estimated position

Table 4.4: Structure of the `estimate` message

Type	**Variable**	**Description**
`float64`	`timestamp`	timestamp of the UWB estimation
`anchor[]`	`anchors`	array of anchor message
`uint8`	`nAnchors`	amount of anchors in the array
`estimate`	`estimate`	estimate message

Table 4.5: Structure of message type `tagFull`

their positions. This message type is used in the ROS topic `/anchor_config`.

Type	**Variable**	**Description**
`float64`	`timestamp`	timestamp of the instant that the synchronization point is detected
`float64`	`diff`	time that passed from the previous synchronization point to the current one

Table 4.6: Structure of message type `synchPoint`

Whenever a synchronization Point is detected, **synchPoint** is sent and has the following structure depicted in Table 4.6. This message is used in the ROS topic `/synchPoint` and is defined in `synchPoint.mnsg`.

In addition to these implemented messages, four ROS standard messages were utilized:

- **sensor_msgs/Imu** includes IMU data rather than Decawave data and contains the linear acceleration, angular velocity, and a quaternion indicating the robot's orientation. Accelerations should be expressed in m/s^2, and rotational velocity in rad/s.
- **geometry_msgs/Twist** to control the simulated robot in Gazebo in the topic `/alphabot2/control`. This message expresses velocity in space broken into its linear and angular components.
- **tf2_msgs/TFMessage** to retrieve the position of the actual position of the robot in the simulation from the topic `/tf`.

- **gazebo_msgs/ModelState** message is sent to `/gazebo/set_model_state` when the simulation begins to reset the robot's location and orientation.

Chapter 5

Experiments and Results

This chapter presents an evaluation of the framework developed. We begin with results for a case-study experiment in Section 5.1 with a real AlphaBot2, then a simulation experiment in Section 5.2, where a simulated AlphaBot2 moves in a Gazebo World. This last experiment has two variants, one without any noise on the sensor data and the other with simulated noise. Finally, a case study where the sensor logs from a previous experiment are played, to perform the same experiment to validate whether the results remain the same in Section 5.3.

For each experimental case, results of the position estimated by various algorithms will be presented:

- **Decawave**, which is the robot position estimated by the Decawave firmware.
- **leastSquares**, the estimation of the robot's position using multi-lateration with the positions of the UWB anchors and the respective UWB ranges to the robot, determined by Decawave firmware.
- **leastSquaresMA**, a variant of leastSquares that contains moving average. It uses the average of the last 10 leastSquares estimations to estimate the current position.
- **deadReckoning**, estimates the position of the robot using a constant velocity (0.15 m/s) applied to the orientation estimated by the IMU.
- **deadReckoningSP**, a variant of deadReckoning. Like deadReckoning, it uses a constant speed (0.15 m/s) applied to the orientation received from the IMU, but whenever the robot detects the synchronization point (control point A), the position is adjusted.
- **deadReckoningACC**, the original variant of deadReckoning. It no longer uses a constant velocity, calculating the motion vector from the accelerations and orientations debited by the IMU.

The position estimates will be analyzed and compared to determine if they are indeed effective in determining the robot's position and in case of incorrect estimations, possible reasons to justify the errors will be mentioned.

5.1 Real AlphaBot2 case-study

5.1.1 Setup

For this experimental case, a track was elaborated in the Embedded Systems room of the Computer Science Department of the Faculty of Sciences of the University of Porto. To make the path, multiple white sheets of paper were placed on the floor of the room. Then, insulating tape was glued to the white sheets, to outline the robot's path. To calculate the error of the estimations, it was needed to obtain information about the real position of the robot during the experiment, in order to do that, 4 pieces of cardboard box were placed along the path, which work as control points. We used 6 UWB devices in this experiment, 5 of them were scattered around the room configured as anchors (DW0AAA as the initiator) and another one configured as tag, which moves along with the AlphaBot2. These configurations can be observed in detail in Figure 5.1, together with their respective positions.

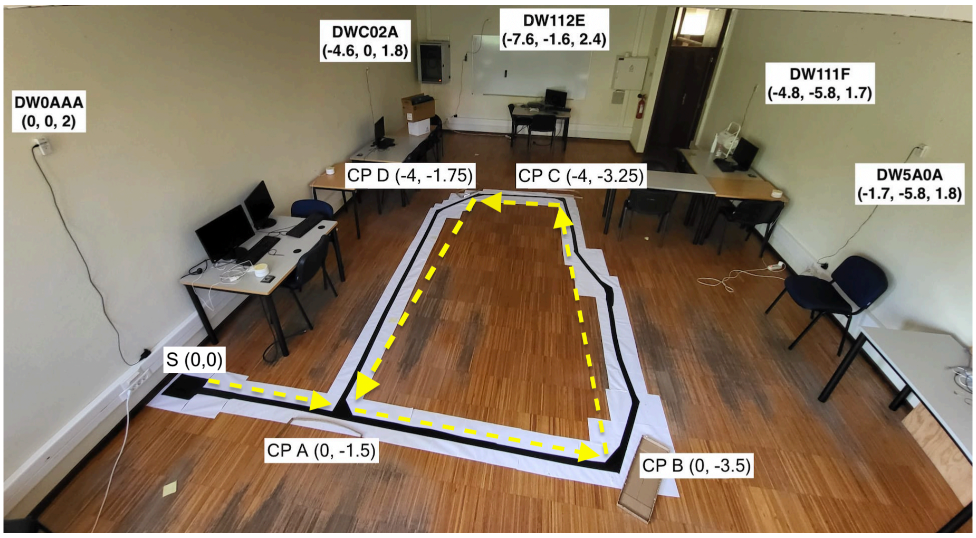


Figure 5.1: AlphaBot2 setup, with UWB anchors (DW...) and control points (CP) with their respective positions. It also includes the AlphaBot2 route that starts at the S point and moves according to the yellow arrows

AlphaBot2 follows the path outlined with the insulating tape, using the reflective infrared photoelectric ITR20001/T sensores for line tracking. The robot starts at (0, 0), then moves towards control point A, then to control point B, C, D, and back to A, B, C, D, and A until the end of the test. To obtain information about the UWB ranges as well as the positions estimated by the Decawave firmware, the UWB device configured as tag was placed on top of the robot adapter board. In addition to the UWB device, a micro:bit was also placed on top of the robot adapter board to collect data about the orientation and acceleration of the robot during the experiment. The sensor placements, can be observed in Figure 5.2.

Each time the robot detects a control point, with the Infrared photoelectric sensor (ST188) for obstacle avoidance, it beeps and sends a ROS message with the timestamp and how much

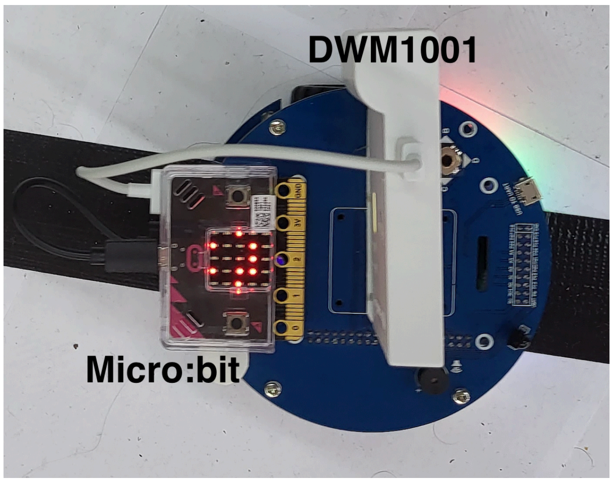


Figure 5.2: AlphaBot2 setup

time passed since the last control point. It is important to mention that the control points are 30 centimeters length boxes and that the robot can detect obstacles up to 10 centimeters away. Since the control point's job is to allow to estimate the error of each algorithm, the error is the distance between the robot's estimated position (at the instant that the control point is detected) to the center of the box, therefore the error calculated may be different than the real one. The framework received data from Decawave at about 10Hz and IMU data at about 50Hz.

5.1.2 Results

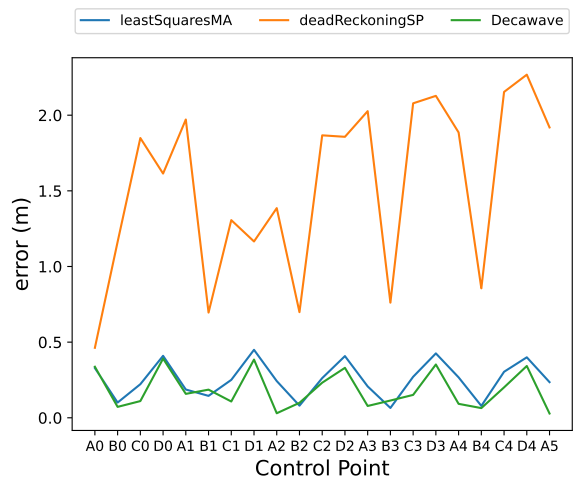
The results for the real-case experiment can be observed in Figures 5.3. From there, we can observe the error at each control point in Figure 5.3a and the estimations of the positions of the robot by each algorithm:

- Decawave firmware estimation, Figure 5.3b
- Least Squares with moving average, Figure 5.3c
- Dead Reckoning with constant speed (0.15 m/s) and adjusting position on synchronization point - (control point A) (deadReckoningSP), Figure 5.3d

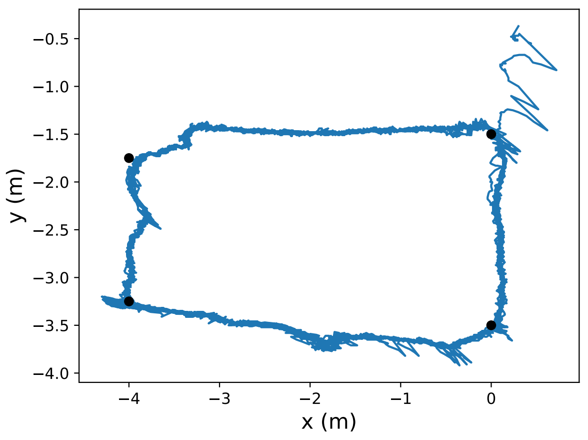
Overall, we can observe that every algorithm that is based on UWB range measurements is very proximate to the real position of the robot during the experiment, while Dead Reckoning's estimations are a bit off, due to incorrect orientations.

Decawave firmware estimations, Figure 5.3b, were highly accurate over the entire experiment, the only time the estimations were a bit off, was at the beginning, from (0, 0) to the first control point (0, -1.5). This was probably due to the proximity of metallic tables, and because at the start of the course, the robot was right below an anchor.

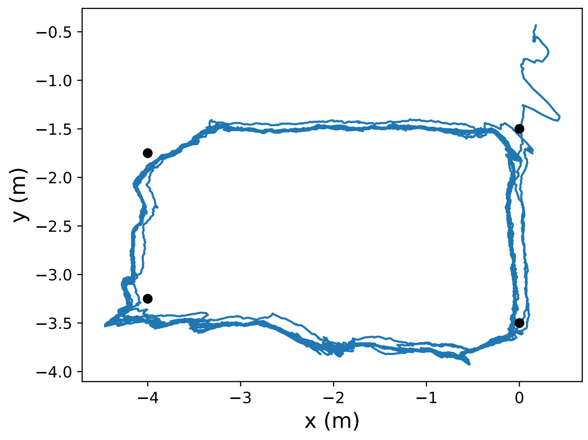
Since **Least Squares with moving average**, Figure 5.3c, is based on Decawave UWB



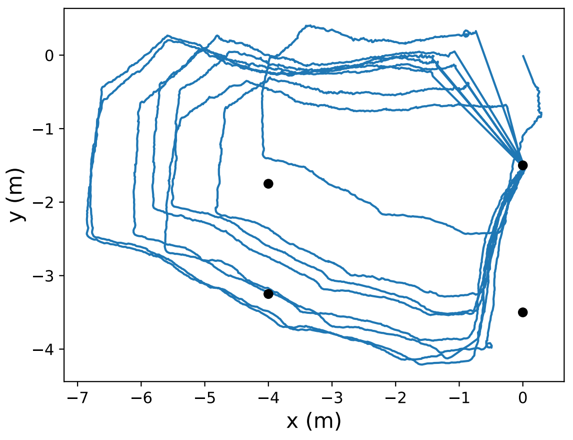
(a) Error of each algorithm estimation at each control point over the first 5 laps



(b) Estimated positions by Decawave over 8 laps



(c) Estimated positions by Least Squares with moving average over 8 laps



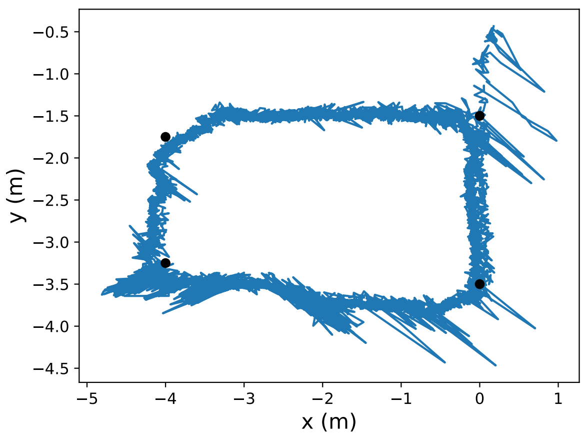
(d) Estimated positions by Dead Reckoning algorithm with constant speed and position adjustment on synchronization point over 8 laps

Figure 5.3: Real case-study estimations

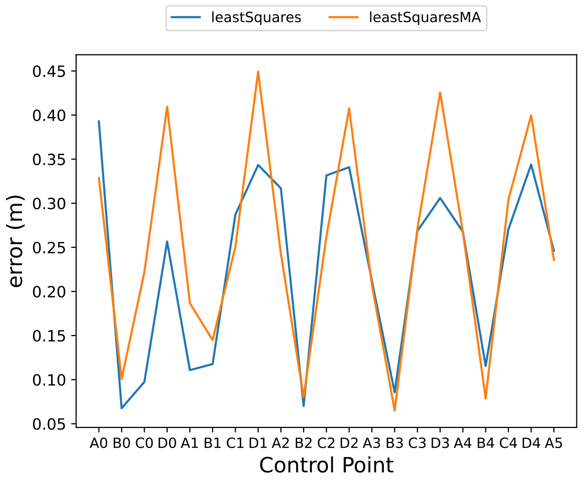
ranges, the path is very similar, though since it makes the estimation based on the average of the last 10 estimations, the errors are a lot smoother than Decawave.

Least Squares estimations, Figure 5.4a, have very big variations in a short period of time, therefore the graph generated is trickier to analyze than with moving average, which shows a much cleaner movement path, generating a plot much easier to analyze, but as shown in Figure 5.4b, errors are propagated to the next estimations, while without moving average it does not happen.

Due to the variations observed in Figure 5.5a, Dead Reckoning with position adjustment on synchronization point (control point A), **Dead ReckoningSP**, was developed, which can be observed in Figure 5.3d. These variations could be caused due to many reasons, such as battery



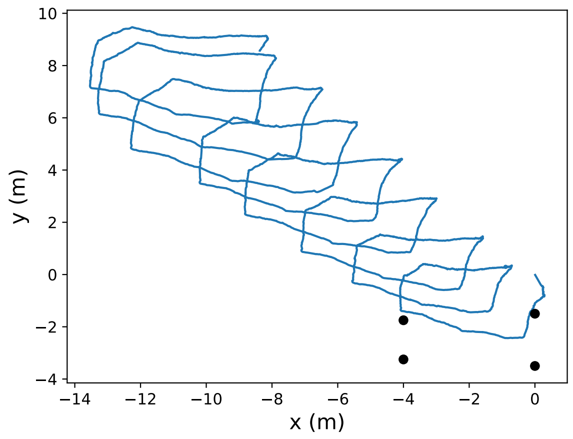
(a) Estimated positions by Least Squares over 8 laps



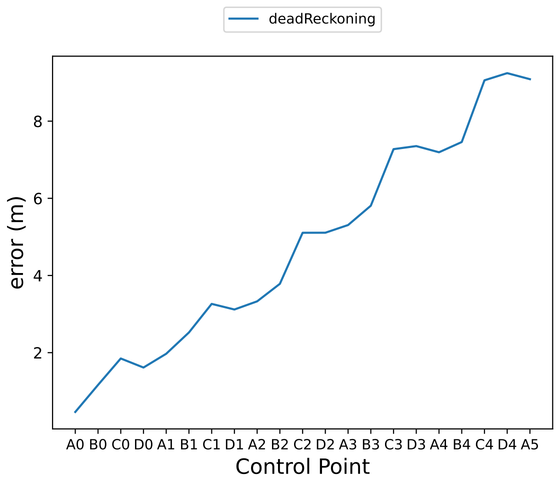
(b) Comparison of Least Squares error with and without moving average at each control point over the first 5 laps

Figure 5.4: Comparison of Least Squares algorithms

drainage, which reduces the velocity of the robot over time, which can be observed in the almost linear increase of the error in Figure 5.5b. Another reason is the existence of errors and drift in the orientations estimated by micro:bit. As observed in Figure 5.5, Dead Reckoning algorithms provide much less accurate estimations than UWB-based algorithms from Figure 5.4.



(a) Estimated positions by Dead Reckoning with constant speed over 8 laps



(b) Error of Dead Reckoning with constant speed over the first 5 laps

Figure 5.5: Dead Reckoning results

deadReckoningACC, the Dead Reckoning with the actual accelerations instead of using constant speed, was the original Dead Reckoning algorithm used. Without any errors from the sensors, it worked perfectly with minimal error, but since, the accelerations and orientations from micro:bit weren't 100% accurate, the estimations of Dead Reckoning using the accelerations given were very bad, therefore a constant speed had to be determined for the robot.

5.2 Simulation case-study

5.2.1 Setup

For the simulation case-study experiment, a simulated AlphaBot2 moves, in a Gazebo world, forward for 10 meters, then rotates 90º to the right and moves for another 10 meters, making square alike movements as observed in Figure 5.6, starting at (0, 0), then moving towards A, then to B, C, D and then to A again to finish 1 lap. In Subsection 5.2.2 5 laps without introducing any kind of noise is presented, and in the Subsection 5.2.3 for another 5 laps with noise in both orientation and in ranges measurements.

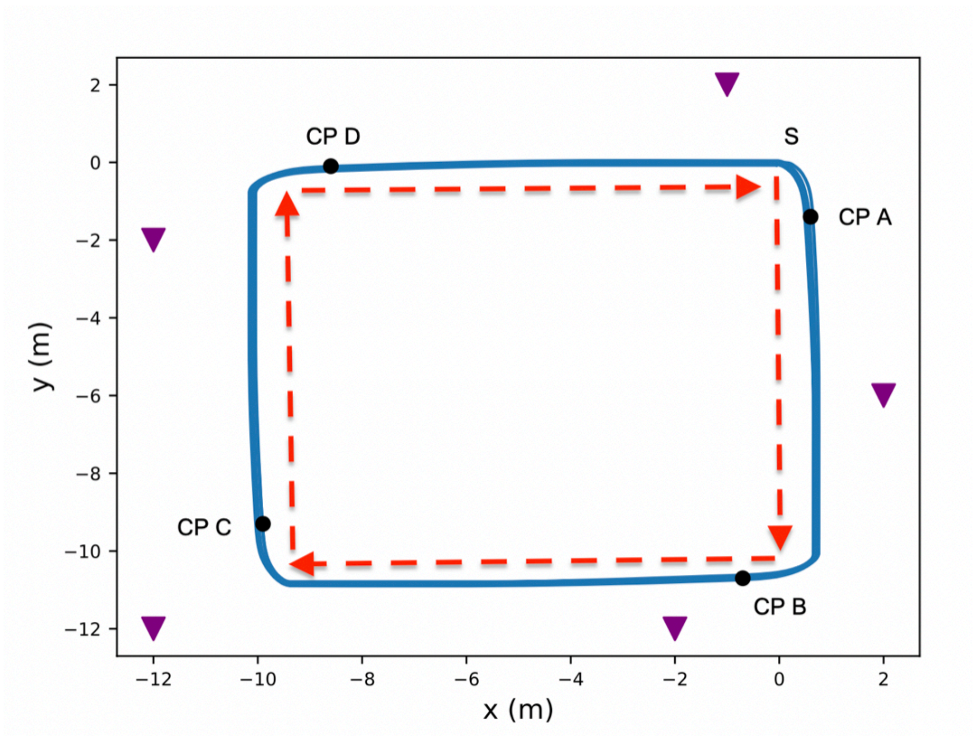


Figure 5.6: Simulation course with simulated control points (black dots) and anchors (purple triangles). Robot starts at point S (0,0) and moves according to the red arrows.

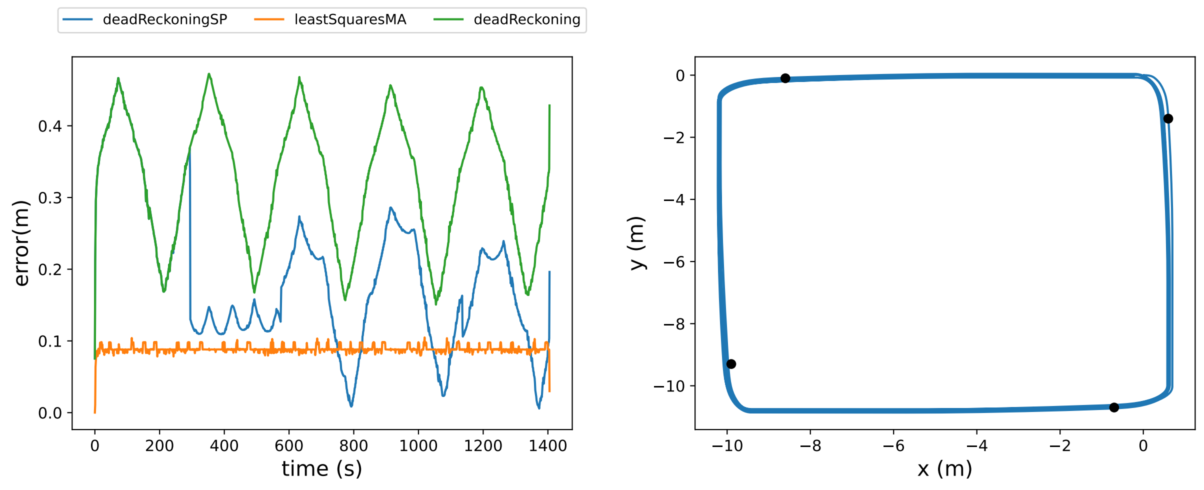
In order to obtain similar results to the previous experimental case, the same conditions were applied to this one. The simulation will also have 6 UWB devices, 5 of them configured as anchors scattered around the scenario and 1 as a tag placed on the robot. Besides the simulated anchors, we determined 4 control points A,B,C and D, positioned at the end of each turn. In Figure 5.6 we can visualize the positions of the anchors represented as purple triangles and the control points represented as black dots. The simulated UWB and IMU data will keep the rates of the real experimental case, so 10Hz and 50Hz respectively.

In the simulation, a UWB range is the calculation of the distance between the anchor and the real position of the robot. For the results with Noise in the Subsection 5.2.3, Gaussian Noise of (0.5m, 0.5m) is added to each range. The IMU data is calculated by a Gazebo plugin and for

Subsection 5.2.3 (0, 4º) Gaussian Noise was added to the orientations estimated.

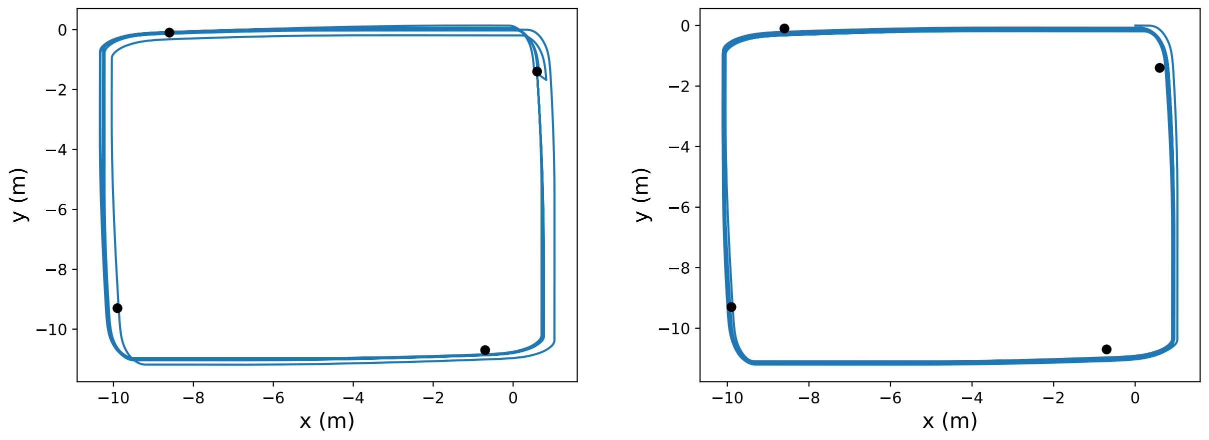
Since in the simulation, the actual position of the robot is known at every instant during the entire duration of the experiment, the error calculated in this section is the actual error during the experiment, rather than the relative error at the instant the robot detects the control points.

5.2.2 Results without noise



(a) Error of each algorithm estimation

(b) Estimated positions by Least Squares with moving average over 5 laps



(c) Estimated positions by Dead Reckoning algorithm with constant speed and position adjustment on synchronization point over 5 laps

(d) Estimated positions by Dead Reckoning algorithm with constant speed over 5 laps

Figure 5.7: Simulated estimations without noise

In order to obtain similar results to the previous experimental case, the In this experiment, in Figure 5.6 we can observe the actual positions of the robot during the experiment, and in Figure 5.7 the estimations of the positions of the robot by each algorithm for the full experiment (5 laps):

- Least Squares with moving average, Figure 5.7b
- Dead Reckoning with constant speed (0.15 m/s) adjusting the position in the synchronization point - (control point A) (deadReckoningSP), Figure 5.7c
- Dead Reckoning with constant speed (0.15 m/s) (deadReckoning), Figure 5.7d

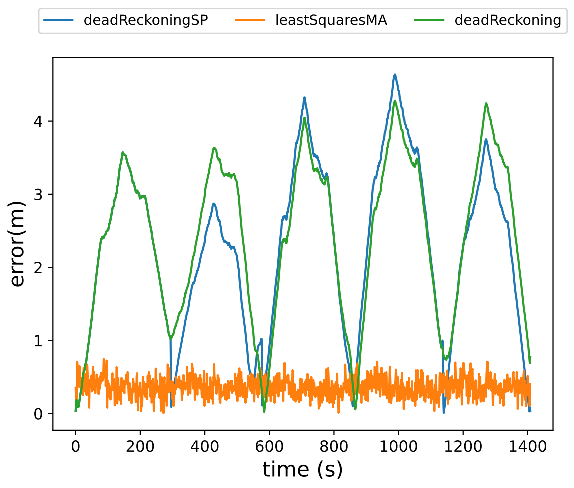
By observing Figure 5.7a it is possible to conclude that every algorithm is working as intended and giving good estimations of the robot's location, with **deadReckoning** being the one with worse results. We can also observe that adjusting the position of the robot is beneficial to the estimation since **deadReckoningSP** had less error over time than **deadReckoning**. Once again **leastSquaresMA** 5.7b gives the most proximate estimations.

5.2.3 Results with noise

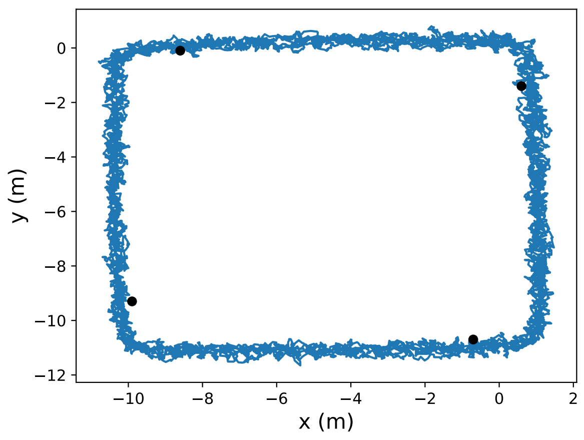
In this experiment, (0.5m, 0.5m) Gaussian noise was added to UWB Simulation ranges and to IMU orientation (0, 4º). From Figure 5.8, we can observe the estimations of the positions of the robot by each algorithm over the 5 laps:

- Dead Reckoning with constant speed (0.15 m/s) (deadReckoning), Figure 5.8d
- Least Squares with moving average, Figure 5.8b
- Dead Reckoning with constant speed (0.15 m/s) adjusting position in the synchronization point - (control point A) (deadReckoningSP), Figure 5.8c

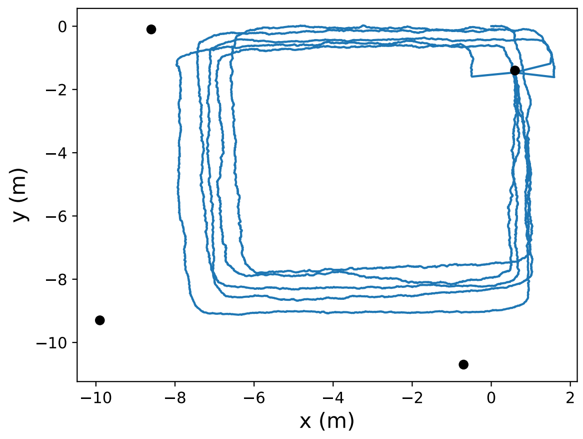
By observing 5.8 it is possible to once again validate **Least Squares**, with minimal error, and to conclude that **Dead Reckoning** algorithms suffer a lot from errors, even if small ones. In Figure 5.8a we can observe the error of these 3 algorithms during the experiment.



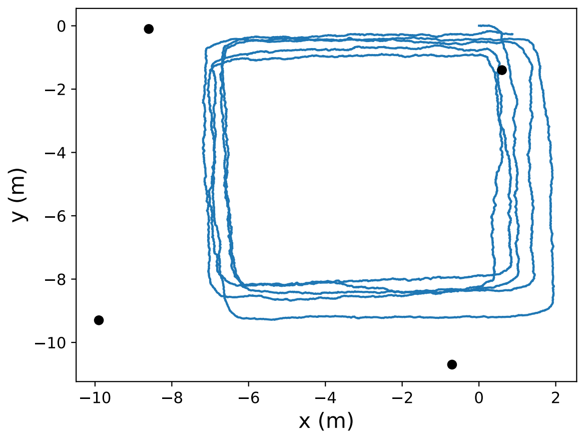
(a) Error of each algorithm estimation



(b) Estimated positions by Least Squares with moving average over 5 laps



(c) Estimated positions by Dead Reckoning algorithm with constant speed and position adjustment on synchronization point over 5 laps



(d) Estimated positions by Dead Reckoning algorithm with constant speed over 5 laps

Figure 5.8: Simulated estimations with noise

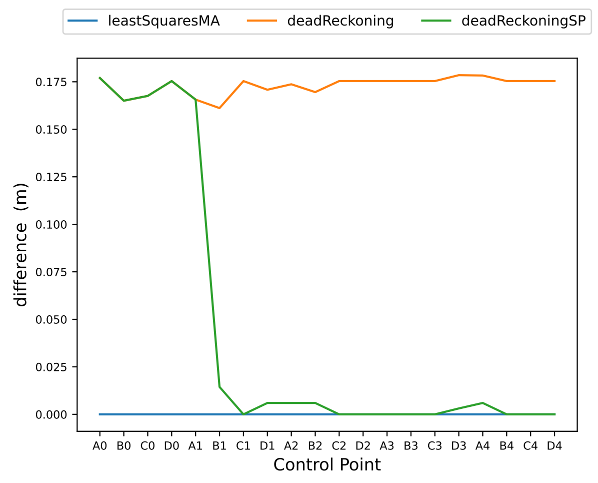
5.3 Replay case-study

In this last case study, an experiment was performed by replaying the logs of the sensors of the simulated case study from Section 5.2.3 in order to validate if replaying the logs produces the same results, in other words, to check whether the replay has the same temporization as the original experiment.

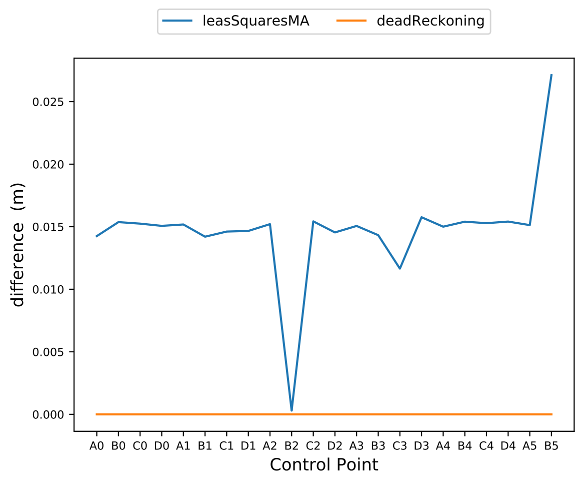
5.3.1 Setup

Finally, in the replay case study, the logs from the sensors (UWB, IMU, and synchronization points) of previous experiments are replayed, to perform the same experiment to validate if replaying the logs produces the same results.

5.3.2 Results



(a) Comparison between estimations at each control point from the experiment that generated the sensor logs (Simulation with noise 5.2.3) and the estimations from replaying the logs



(b) Comparison between estimations at each control point from a experiment with the Robot Configuration and the estimations from replaying the sensor logs (ajustes necessários (todo)

Figure 5.9: Plots that validate the Replay Configuration

We can see a tiny discrepancy between the estimated positions acquired in the real result and by replaying the logs in Figure 5.9. The estimates using **leastSquaresMA** are quite near to each other in both trials, with a difference very close to 0m. **Dead Reckoning** estimates, on the other hand, contain a substantial inaccuracy. This gap might occur when a new location is determined by Dead Reckoning at 50Hz but only published at around 10Hz, implying that there might be a difference in the publication timer between them. Since leastSquares estimates a fresh position for each UWB log, the methods give 0 errors during the trial. The difference in scale between replaying logs from a simulation experiment and replaying from the robot configuration might be due to the greater distance traveled by the robot in the simulation compared to the real route.

If each algorithm, for each log of data received by the sensors, estimates and publishes a new position, it should be possible to obtain a near-perfect replication of the experiment. With the current operation of the framework, even with a small error in the deadReckoning estimations, the replay configuration appears to be feasible to replicate experiments, although it can still be improved.

5.4 Summary

In this chapter, we present an evaluation of the implemented algorithms in both simulation and reality. The aim was to allow an analysis that can prove that simulation can indeed represent real scenarios and to analyze the behavior of each algorithm in each case.

UWB measurements are highly accurate and both least squares algorithms can estimate very approximately the target position. Least Squares with moving average accumulate errors from previous estimates, which are very difficult to recover. On the other hand, Least Squares without moving average can give very wrong estimates from time to time, but in general, has a smaller average error.

Dead Reckoning algorithms, suffer a lot from noise, while Dead Reckoning acceleration is the most accurate in a perfect scenario, with errors however small, it starts giving completely wrong estimates. Dead Reckoning with constant speed adjusting the position occasionally seems a very good option to reduce the errors of this algorithm.

During the real experiment, there were some limitations. When it came to the robot, there was a large amount of time devoted to getting a more regular movement. Another problem was to determine a distance between the robot track and the control points that would cause the robot to get stuck, to avoid that, the solution was to cut the boxes at a height that the cables connecting the sensors to the AlphaBot2's raspberry could pass over. Another great difficulty was calibrating the orientation of the IMUs. With the micro:bit, a calibration that allowed the collection of enough data to estimate the Dead Reckoning was achieved, although with a small drift and some irregularities, while with the pi pico, this calibration was not achieved, not allowing its use in the tests. With these sensors, it was also not possible to collect accelerations that would allow their use. The accelerations estimated by the micro:bit had a minimum error of 16 millig (0.1569064 m/s^2) and even with the robot stopped, returned somewhat high accelerations. The accelerations given by pi pico had no congruence at all, as did the orientations.

Chapter 6

Conclusion

In this chapter we conclude with a final discussion of the contributions of the dissertation and of a few directions for future work.

In this dissertation, we presented AugBot, a ROS-based framework for developing and testing indoor location algorithms that can be deployed on embedded software systems, robots in particular as illustrated for the AlphaBot2 robot, or in combination with physical simulation engines as we did with Gazebo. AugBot's architecture is modular, allowing for a clear separation of logic regarding aspects such as sensor readings, algorithms for indoor position, and communication. Thanks to this, the code for an indoor location algorithm can run without changes across different deployment environments (physical, simulated, or replay-based). This has been achieved for UWB-based multi-lateration using the Least Squares algorithm and also dead reckoning algorithms that feed on inertial sensor readings.

AugBot addresses key requirements of some use cases in the Augmanity project regarding indoor location such as the use of AGVs and UWB-based indoor location. We expect AugBot to be a basis for hardware/software integration in the project tasks in collaboration with FCUP partners, in particular regarding the use of the 5G tag devices described in Chapter 2.

Overall, we believe the work of this thesis may motivate several interesting directions for future work. AugBot is suitable for the development and testing of more complex algorithms for indoor location and different use cases in indoor location. Regarding algorithms, sensor fusion algorithms that can integrate different types of sensor readings are of particular interest. Regarding use-cases, this work considered on-device location tracking, but not the more complex use-case of external tracking of devices in a physical environment at the network level, this is in principle doable for instance using the same UWB infrastructure by configuring UWB anchors as tags and vice-versa so that UWB ranges are received at static node locations. Overall, we believe these goals can be attained without architectural changes and reusing all the current code in place for other concerns.

The integration of the framework into more AGVs is another challenge to be taken into account for future work. The Jetbot [14] is an open-source robot based on NVIDIA Jetson Nano, and it is a portable, battery-powered AI computer with a camera. This makes it the perfect platform for trying new ideas related to AI, image processing, and robotics. JetBot includes a set of Jupyter notebooks that cover basic robotics concepts like programmatic motor control, to more advanced topics like training a custom AI for avoiding collisions. To provide more information and a bigger variety of data for more accurate estimations, more technologies should be included in the system. The first technology that should be included is WiFi-RTT, which would be another technology to provide ranges from the targeted AGV to nearby RTT-capable WiFi access points. That way, the multi-lateration-based algorithms could estimate better positions.

One of the main limitations of the real case study environment was the control point scheme. For better detection and better ground truth data, my suggestion is to use it in upcoming experiences, QR codes as a more robust and portable alternative as long as the robot has a camera. The primary issue of the currently used control points (30 centimeter length boxes) is that the detection does not always occur in the same places, therefore the ground truth is not always optimal, with this alternative, it is believable that the detection would be more precise and more controlled.

Bibliography

[1] AUGMENTED HUMANITY - APPLICATION PROPOSAL PART B (TECHNICAL ANNEX). CALL 14/SI/2019 - Portugal 2020, 2019.

[2] G.R. Aiello and G.D. Rogerson. Ultra-wideband wireless systems. *IEEE Microwave Magazine*, 4(2):36–47, 2003. doi:10.1109/MMW.2003.1201597.

[3] Bekir Bostanci, Sercan Tekkok, Emre Soyunmez, Pinar Oguz-Ekim, and Faezeh Yeganli. The LiDAR and UWB based Source Localization and Initialization Algorithms for Autonomous Robotic Systems. In *2019 11th International Conference on Electrical and Electronics Engineering (ELECO)*, pages 900–904, 2019. doi:10.23919/ELECO47770.2019.8990648.

[4] Hongji Cao, Yunjia Wang, Jingxue Bi, Shenglei Xu, Minghao Si, and Hongxia Qi. Indoor positioning method using WiFi RTT based on LOS identification and range calibration. *ISPRS International Journal of Geo-Information*, 9(11):627, 2020.

[5] Long Cheng, Cheng-dong Wu, and Yun-zhou Zhang. Indoor robot localization based on wireless sensor networks. *IEEE Transactions on Consumer Electronics*, 57(3):1099–1104, 2011. doi:10.1109/TCE.2011.6018861.

[6] Sebastian Dädeby and Joakim Hesselgren. A system for indoor positioning using ultra-wideband technology. Master's thesis, 2017.

[7] Universidade de Aveiro. Bosch and UA present first results of the project for Industry 4.0. https://www.ua.pt/en/news/9/76116. Last access: July 2022.

[8] Decawave. MDEK1001 Kit User Manual Module Development Evaluation Kit for the DWM1001. https://eu.mouser.com/datasheet/2/412/MDEK1001_System_User_Manual-1.1-1878639.pdf. Last access: July 2022.

[9] Aicos Fraunhofer. Augmanity. https://www.aicos.fraunhofer.pt/en/our_work/projects/augmanity.html. Last access: February 2022.

[10] Christian Gentner, Markus Ulmschneider, Isabel Kuehner, and Armin Dammann. WiFi-RTT Indoor Positioning. In *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 1029–1035, 2020. doi:10.1109/PLANS46316.2020.9110232.

[11] GPS gov. Other Global Navigation Satellite Systems (GNSS). https://www.gps.gov/systems/gnss/#:~:text=Global%20navigation%20satellite%20system%20(GNSS,a%20global%20or%20regional%20basis.. Last access: July 2022.

[12] Weipeng Guan, Shihuan Chen, Shangsheng Wen, Wenyuan Hou, Zequn Tan, and Ruihong Cen. Indoor Localization System of ROS mobile robot based on Visible Light Communication, 2020. doi:10.48550/ARXIV.2001.01888.

[13] Junya Hayashi. MQTT bridge. https://github.com/groove-x/mqtt_bridge, 2021.

[14] NVIDIA AI IOT. JetBot. https://jetbot.org/master/index.html. Last access: September 2022.

[15] Lentin Joseph and Jonathan Cacace. *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. Packt Publishing Ltd, 2018.

[16] Ghazaleh Kia, Jukka Talvitie, and Laura Ruotsalainen. RSS-based fusion of UWB and WiFi-based ranging for indoor positioning. In *11th International Conference on Indoor Positioning and Indoor Navigation-Work-in Progress Papers, IPIN-WiP 2021*. CEUR-WS, 2021.

[17] Stefan Kohlbrecher and Johannes Meyer. Hector Gazebo Plugins. https://github.com/tu-darmstadt-ros-pkg/hector_gazebo, 2022.

[18] Micro:bit. Bit educational foundation. https://microbit.org/. Last access: July 2022.

[19] Ruchik Mishra and Arshad Javed. ROS based service robot platform. In *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pages 55–59, 2018. doi:10.1109/ICCAR.2018.8384644.

[20] Fatih Okumuş and Adnan Fatih Kocamaz. Cloud Based Indoor Navigation for ROS-enabled Automated Guided Vehicles. In *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pages 1–4, 2019. doi:10.1109/IDAP.2019.8875993.

[21] MQTT org. The standard for IOT messaging. https://mqtt.org/. Last access: July 2022.

[22] Raspberry Pi. Raspberry pi documentation. https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html. Last access: July 2022.

[23] Augmanity Project. 5G Tag - D17.2 Base Hardware platform with 5G communication. https://www.augmanity.pt/sites/default/files/inline-files/D17.2%20Base%20Hardware%20platform%20with%205G%20%20communication.pdf, . Last access: July 2022.

[24] Augmanity Project. Augmanity - About Us. https://www.augmanity.pt/about, . Last access: July 2022.

[25] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[26] Ana Rafael, Cássio Santos, Diogo Duque, and Sara Fernandes. Alphabot2 ROS Package and Simulator. https://github.com/ssscassio/alphabot2-simulator, 2019.

[27] Robot Operating System. Robot operating system. https://www.ros.org/. Last access: July 2022.

[28] Kenta Takaya, Toshinori Asai, Valeri Kroumov, and Florentin Smarandache. Simulation environment for mobile robots testing using ROS and Gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 96–101. IEEE, 2016.

[29] Janis Tiemann, Fabian Eckermann, and Christian Wietfeld. Atlas-an open-source tdoa-based ultra-wideband localization system. In *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–6. IEEE, 2016.

[30] Janis Tiemann, Yehya Elmasry, Lucas Koring, and Christian Wietfeld. ATLAS FaST: Fast and Simple Scheduled TDOA for Reliable Ultra-Wideband Localization. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2554–2560, 2019. doi:10.1109/ICRA.2019.8793737.

[31] Waveshare. Pico 10DOF IMU. https://www.waveshare.com/wiki/Pico-10DOF-IMU. Last access: September 2022.

[32] Waveshare. AlphaBot 2 User Manual - Mouser Electronics. https://www.mouser.com/pdfdocs/Alphabot2-user-manual-en.pdf, Aug 2017. Last access: July 2022.

[33] William Woodall. Serial Communication Library. https://github.com/wjwwood/serial, 2022.

[34] Reza Zandian. *Ultra-wideband based indoor localization of mobile nodes in ToA and TDoA configurations*. PhD thesis, Dissertation, Bielefeld, Universität Bielefeld, 2018, 2019.