



**ROBOT LOCALIZATION WITH ADAPTIVE MONTE
CARLO LOCALIZATION AND REAL-TIME
LOCALIZATION SYSTEM**

SERCAN AĐDAŐ TEKKÖK

Thesis for the Master's Program in Electrical and Electronics Engineering

Graduate School
Izmir University of Economics

Izmir

2022

**ROBOT LOCALIZATION WITH ADAPTIVE MONTE
CARLO LOCALIZATION AND REAL-TIME
LOCALIZATION SYSTEM**

SERCAN AĐDAŐ TEKKÖK

THESIS ADVISOR: Assoc. Prof. Dr. Pınar OĐUZ EKİM

A Master's Thesis

Submitted to

the Graduate School of Izmir University of Economics
the Department of Electrical and Electronics Engineering

Izmir

2022

ETHICAL DECLARATION

I hereby declare that I am the sole author of this thesis and that I have conducted my work in accordance with academic rules and ethical behaviour at every stage from the planning of the thesis to its defence. I confirm that I have cited all ideas, information and findings that are not specific to my study, as required by the code of ethical behaviour, and that all statements not cited are my own.

Name, Surname: Sercan Çağdaş Tekkök

Date: 02.01.2023

Signature:

ABSTRACT

ROBOT LOCALIZATION WITH ADAPTIVE MONTE CARLO LOCALIZATION AND REAL-TIME LOCALIZATION SYSTEM

Tekkök, Sercan Çağdaş

Master's Program in Electrical and Electronics Engineering

Advisor: Assoc. Prof. Dr. Pınar OĞUZ EKİM

December, 2022

Autonomous systems are a trending topic due to their proven benefits to society and industry. Therefore, it is crucial to design and deploy reliable systems for safety and productivity. One of the most important parts of autonomous mobile robots is their ability to localize themselves and this problem becomes complicated depending on the use case. So, this thesis proposes a solution to localization problems for dynamic environments. Problems can be observed when there are dramatic changes in the area where the robot navigates. These changes become problematic when it starts to prevent LiDAR from taking matching measurements with the previously generated map. The proposed method is the fusion of different global localization sources together to have a more robust solution. These global localization sources are adaptive Monte Carlo localization and ultra-wideband-based real-time localization system. Both sources fused with an extended Kalman filter to create a more fault-tolerant system. Although the system requires sensors to be placed in the perimeter, results showed that the proposed method is able to eliminate occurring problems when the only active global

localization source is Adaptive Monte Carlo Localization.

Keywords: Localization, Sensor fusion. Real-time localization system, Autonomous robotics, Robot operating system.



ÖZET

ADAPTİF MONTE CARLO KONUMLANDIRMASI VE GERÇEK ZAMANLI KONUMLANDIRMA SİSTEMİ KULLANILARAK ROBOT KONUMLANDIRMA SİSTEMİ

Tekkök, Sercan Çağdaş

Elektrik Elektronik Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Doç. Dr. Pınar OĞUZ EKİM

Aralık, 2022

Otonom sistemler, topluma ve endüstriye kanıtlanmış faydaları nedeniyle popüler olan bir konudur. Bu nedenle, güvenlik ve üretkenlik için güvenilir sistemler tasarlamak ve devreye almak çok önemlidir. Otonom mobil robotların en önemli parçalarından biri kendi kendilerini konumlandırma yetenekleridir ve kullanım durumuna bağlı olarak bu problem karmaşık hale gelmektedir. Bu yüzden bu tez çalışması dinamik ortamlardaki konumlandırma problemlerine bir çözüm sunmaktadır. Problemler robotun çalıştığı alanda büyük değişiklikler olduğunda gözlemlenebilmektedir. Bu değişiklikler, LiDAR'ın önceden oluşturulmuş harita ile eşleşen ölçümler almasını engellemeye başladığında sorunlu hale gelir. Önerilen yöntem, daha sağlam bir çözüme sahip olmak için farklı küresel konumlandırma kaynaklarının harmanlamasıdır. Bu kaynaklar adaptif Monte Carla konumlandırması ve Ultra geniş bant temelli gerçek zamanlı konumlandırma sistemidir. Her iki kaynak genişletilmiş Kalman süzgeci ile harmanlanarak hataya dayanıklı bir sistem oluşturmaktadır. Sistem

evreyeye algılayıcıların yerleřtirilmesini gerektirmesine rađmen, sonular nerilen yntemin yalnızca adaptif Monte Carlo konumlandırması kullanıldıđında ortaya ıkan sorunları ortadan kaldırabildiđini gstermiřtir.

Anahtar Kelimeler: Konumlandırma, Sensr Harmanlama. Gerek zamanlı konumlandırma sistemi, Otonom robotik, Robot iřletim sistemi.



ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Assoc. Prof. Dr. Pınar Oğuz Ekim for her guidance and insight throughout the research.



TABLE OF CONTENTS

ACKNOWLEDGEMENTS	viii
TABLE OF CONTENTS	ix
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xiv
CHAPTER 1: INTRODUCTION.....	1
1.1. Problem Statement	2
CHAPTER 2: PRELIMINARIES.....	5
2.1. Exteroceptive sensors.....	5
2.2. Proprioceptive sensors	5
2.3. Active sensors	5
2.4. Passive sensors	6
CHAPTER 3: RELATED WORK.....	8
3.1. Wi-Fi	8
3.2. Bluetooth.....	9
3.3. RFID.....	9
3.4. Ultrasound	9
3.5. Ultra-wideband.....	9
3.6. LiDAR SLAM.....	10
3.6.1. Hector SLAM.....	11
3.6.2. Gmapping	12
3.6.3. Cartographer.....	13
3.7. Visual SLAM	14
3.7.1. ORB-SLAM	15
3.7.2. RTAB Map.....	16
CHAPTER 4: METHODOLOGY	18
4.1. Robot Operating System	18
4.1.1. Unified Robot Description Format	18
4.1.2. Gazebo.....	19
4.1.3. RViz	20
4.1.4. Transformation Package.....	22
4.1.5. Map Server	23

4.2. Real-Time Localization System	24
4.2.1. Signal Strength Based Ranging.....	25
4.2.2. Time-Based Ranging.....	25
4.2.2.1. Time of Flight	25
4.2.2.2. Time Difference of Arrival	27
4.2.2.3. Angle of Arrival	28
4.2.3. Decawave MDEK1001	28
4.2.3.1. Decawave DRTLS Manager	30
4.2.3.2. I/O Ports	32
4.2.3.3. Universal Serial Port	33
4.3. Adaptive Monte Carlo Localization.....	34
4.3.1 KLD - Sampling.....	35
4.3.2 ROS AMCL Package	36
4.4. Extended Kalman Filter	43
4.5. Robot Localization Package.....	47
CHAPTER 5: SIMULATION RESULTS.....	50
5.1. UWB Localization Tests	50
5.2. Adaptive Monte Carlo Localization Tests	52
5.3. Localization tests with the fusion of RTLS and AMCL	55
CHAPTER 6: CONCLUSION	58
6.1. Future work	58
REFERENCES	60
APPENDICES	64
Appendix A-Robot Localization Package Odom Parameters	64
Appendix B- Robot Localization Package Map Parameters	65

LIST OF TABLES

Table 1. List of Sensors.....	6
Table 2. AMCL Parameters.	38
Table 3. Kalman equations table.....	47
Table 4. Ultra-wideband measurement results.....	51



LIST OF FIGURES

Figure 1. Example warehouse with dynamic regions.	3
Figure 2. Decawave ultra-wideband device.	10
Figure 3. Hector SLAM example map.	12
Figure 4. Gmapping example map.	13
Figure 5. Cartographer example map.	14
Figure 6. ORB-SLAM2 map example with camera position estimations.	15
Figure 7. RTAB Map example.	16
Figure 8. Completed RTAB map example.	17
Figure 9. Visualized URDF model.	19
Figure 10. House environment created in Gazebo.	20
Figure 11. RViz display tab.	21
Figure 12. Default RViz buttons.	21
Figure 13. Visualization of transform frames.	22
Figure 14. ToF localization.	26
Figure 15. Possible locations of receiver relative to beacons.	27
Figure 16. Antenna Example Array.	28
Figure 17. RTLS network installation.	29
Figure 18. Decawave DWM1001-DEV module.	30
Figure 19. Example RTLS network.	30
Figure 20. Device configuration screen.	31
Figure 21. RTLS visualization screen (Triangles are anchor positions and circle is the position of the tag).	32
Figure 22. Terminal output.	33
Figure 23. Initialized AMCL (Red arrows represent particles and green dots are LiDAR scans).	42
Figure 24. Initial location of the robot in Gazebo.	42
Figure 25. AMCL Pose after a few iterations (Red arrows represent particles and green dots are LiDAR scans).	43
Figure 26. Ultra-wideband range measurements.	50
Figure 27. Histogram of measurements.	52
Figure 28. Simulation world in Gazebo.	53

Figure 29. Map of the simulated world..... 53
Figure 30. Localization results with AMCL (Green line is ground truth and red line is EKF output)..... 54
Figure 31. Particles when LiDAR does not measure any features in environment. .. 55
Figure 32. Localization results with UWB and EKF (Blue line is ground truth and red line is AMCL output)..... 57



LIST OF ABBREVIATIONS

UWB: Ultra Wide Band

AMCL: Adaptive Monte Carlo Localization

RTLS: Real-Time Localization System

LiDAR: Light Detection and Ranging

ROS: Robot Operating System

ToF: Time of Flight

GPS: Global Positioning System

GNSS: Global Navigation Satellite System

RTK: Real-Time Kinematics

Wi-Fi: Wireless Fidelity

RF: Radio Frequency

IMU: Inertial Measurement Unit

SPI: Serial Peripheral Interface

UART: Universal Asynchronous Receiver- Transmitter

USB: Universal Serial Bus

EKF: Extended Kalman Filter

CHAPTER 1: INTRODUCTION

Autonomous mobile robots are popular devices that are capable of increasing productivity and minimizing the errors caused by humans. Moreover, it is a promising technology for both industrial applications and end-users. These robotic applications provide automated solutions such as transporting goods inside the factories or household cleaning. However, these applications require fundamental components that allow them to move autonomously without external guidance or human intervention: mapping, localization, and path planning.

For a robot to localize itself, it must have some information about its environment. Different approaches have been proposed to solve localization problem, such as visual SLAM techniques like ORB-SLAM and Stereo SLAM (Mur-Artal et al., 2015)(Zhang et al., 2015). According to the ORB-SLAM paper, this solution is a real-time feature-based algorithm with a monocular camera. The accuracy of the system is at centimeter-level for small indoor areas. On the other hand, Stereo SLAM uses line-based features to map the environment, which appears to be a better solution compared to point-based algorithms. However, it suffers from low frame per second due to the line tracking process. SLAM techniques use different sensors, such as LiDARs (Light Detection and Ranging). One of the techniques is grid-based mapping in short G-mapping (Grisetti et al., 2007). This approach takes advantage of both odometry and LiDAR sensors to merge the sensory information with the help of a particle filter algorithm. Adaptive resampling is utilized in this research which achieved the goal with fewer particles, thus increasing the overall performance compared to the reference paper (Hahnel et al., 2003).

There are different approaches to localization problems, but the core idea is similar for all techniques, which is to fuse additional sensory information to better estimate the robot. One method to use for localization is Adaptive Monte Carlo Localization (AMCL) with a LiDAR and odometry setup (Fox et al., 2005). AMCL is a highly used and proven solution with relatively cheap sensors, but the algorithm's accuracy depends highly on the environment. For example, if the environment is highly

symmetric, odometry information has to be precise to maintain accurate localization estimation. An improved version of AMCL with UWB is also proposed in the earlier study (Wang et al., 2019). This technique is developed to solve the kidnapped robot problem and UWB sensors are used to fix the AMCL position when necessary. Another research shows that the replacement of odometry is also possible with the help of RFID sensors (Hahnel et al., 2004). The authors state that the system successfully localized itself and with the use of RFID technology, computational demand decreased. Practical experiments showed that the system could build accurate maps of RFID tags to localize moving objects without odometry information. Similar results can also be achieved by the fusion of ultra-wideband (UWB) ranging and odometry sensors via Kalman filtering (Oguz-Ekim et al., 2020). UWB sensors can be considered as an imitation of GPS systems for indoor areas. However, coverage and accuracy are still limited and highly depends on the quality and placement of the sensors. Various Kalman filters are applied in this area, such as Unscented and Adaptive Extended Kalman filters, as discussed in (Lasmadi et al., 2019)(Yuzhen et al., 2016). Filter selection mainly depends on the type of noise introduced by the sensors and their noise characteristics.

1.1. Problem Statement

AMCL is a widely used algorithm for robots among all localization systems that utilize LiDARs. These systems require pre-created maps in order to localize themselves. However, significant changes in highly dynamic environments can create considerable problems, such as warehouses or industrial facilities, where both human and other machinery activities are present such as forklifts. There are also cases where serious changes are made in these areas, which affect the performance of the algorithms that require static maps for localization. An example case where products are distributed on the ground, and there are no static rack systems at specific parts of the warehouse can be seen in Figure 1.



Figure 1. Example warehouse with dynamic regions (Source: Pierce ,2020).

If robots are mounted with 2D LiDAR Scanners with a limited range, this might prevent them from seeing static landmarks such as walls or columns. Eventually, they will lose track of their location during navigation since they are practically blind if LiDAR can not measure anything. So main objective of this work is to prevent AMRs from getting lost in highly dynamic environments with the help of alternative global localization systems.

The following sections of this thesis consist of five main chapters which are preliminaries, related work, methodology, simulation results, and conclusion. The second chapter will provide information about the common sensor types, their features, and how this data can be collected in one place so that it can be fused. Common sensors and algorithms to build a system that can achieve mapping and localization tasks are reviewed in the related work section. The fourth chapter has a detailed explanation of the theory behind localization algorithms and ROS tools that are used during the

research. Simulation results are presented in the fifth chapter. These results include the tests that are done in a Gazebo environment with localization algorithms and also the proposed solution. Finally, the proposed solution will be discussed in the last chapter with its benefits, drawbacks, and further improvements. This research contributes to the literature by examining algorithms and sensors that are proposed in earlier studies. Existing techniques are compared according to their accuracy, noise resilience, power consumption, and suitability for indoor applications. Subsequently, it extends the earlier works in this field and merges different systems together to create a more robust system design for indoor localization applications.



CHAPTER 2: PRELIMINARIES

Autonomous mobile robots require a localization system in order to accomplish their duties properly. Therefore, achieving full autonomy is crucial since the aim here is to eliminate human errors and increase productivity. There are many ways to solve the localization problems, and various sensors are utilized in order to make the system perceive its environment. Sensors are used to collect information both from the environment and the internal states of the robot. Common sensors can be listed under two categories which are Exteroceptive or Proprioceptive. Each sensor category has two main types, which are Passive or Active.

2.1. Exteroceptive sensors

These types of sensors are used to take measurements from the external world. Some examples of this type are Laser, LiDAR, Radar, Camera, Depth Cameras, Infrared, Sonar, Touch Sensors or Bumpers, GPS, and Proximity Sensors.

2.2. Proprioceptive sensors

These sensors extract information from the robot's internal states, such as its speed and acceleration

2.3. Active sensors

Active types of sensors work by emitting energy to its surrounding and taking measurements with the help of returning signals. Then, the sensor can use signal strength or return time to extract information. For example, LiDAR sensors emit light to the surroundings and wait for it to return to calculate the direct distance to an obstacle with the help of the time difference between the initial pulse and returned pulse.

2.4. Passive sensors

Cameras or temperature sensors can be counted as an example of this type. They mainly measure environmental energy.

Table 1. List of Sensors (Source: Lecture Notes 2002).

General Classification (typical use)	Sensor System	PC: Propriocep. EC: Exteroceptive	P: Passive A: Active
Tactile Sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers Optical barriers Non-contact proximity sensors	EC EC EC	P A A
Wheel/motor sensors (wheel/motor speed and position)	Brush Encoders Potentiometers Synchros, Resolvers Optical Encoders Magnetic Encoders Inductive Encoders Capacitive Encoders	PC PC PC PC PC PC PC	P P A A A A A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass Gyroscopes Inclinometers	EC PC EC	P P P/A
Ground-based beacons (localization in a fixed reference frame)	GPS Active optical or RF beacons Active ultrasonic beacons Reflective beacons	EC EC EC EC	A A A A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors Ultrasonic sensor Laser rangefinder Optical triangulation (1D) Structured light (2D)	EC EC EC EC EC	A A A A A

Table 1. (Cont'd) List of Sensors (Source: Lecture Notes 2002).

Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision-based sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s) Visual ranging packages Object tracking packages	EC	P

As listed in Table 1, there are many sensors for different use cases, so according to the requirements and limitations of the application, sensors must be chosen. Having many sensor options raises the need to merge all sensory information to use it. That is when the Robot Operating system (ROS) comes into play. It is not a standalone operating system but a sub-operating system that can run on Linux, Windows, or Mac. It is a powerful tool when one needs to gather all the data from the system to use it on tasks. For example, sensor fusion or navigation in a known space where control signals are sent to the motors according to the obstacle detection done with cameras simultaneously. ROS provides many tools and libraries for users to build and test their applications. Each process or node that runs on ROS uses TCP/IP protocol to communicate with each other. Publisher/Subscriber model is used in ROS, and data is exchanged under Topics and Services.

CHAPTER 3: RELATED WORK

Localization is an important topic where various techniques and sensors are used to solve the problem both for indoor and outdoor applications. For outdoor applications, Global Navigation Satellite Systems (GNSS) and Real-Time Kinematic (RTK) systems are the most common systems which are widely used in our everyday devices such as phones, cars, computers, or industrial and military-grade devices. Although GNSS provides sufficient accuracy of 3 to 10 meters (Wing et al., 2005) for outdoor applications, GNSS systems are insufficient for indoor applications because more precise positioning is required. RTK is a more advanced technology that utilizes GNSS data in order to correct it further to achieve better accuracy. The system includes a fixed station that transmits correction data and moving receivers that receives the data to reduce the position error. Commercial products such as U-blox Neo-M8P RTK claimed to have a centimeter-level accuracy. However, it is still not suitable for indoor applications since the GNSS signal is highly affected by buildings.

Despite the unavailability of GNSS in indoor environments, GPS-like positioning systems still exist. They are known as Real-Time Localization Systems (RTLS). Technologies that are used in such a system can be listed as follows:

Wi-Fi	UWB	RFID
Bluetooth	Ultrasound	

3.1. Wi-Fi

Wireless Fidelity is mainly used for communication and data transfers in the 2.4GHz and 5GHz bands. Although it is widely available, it is prone to noise, and processing requires complex algorithms. Therefore, suitable algorithms must be utilized in order to use them for localization purposes. These algorithms use a few principles like received signal strength, channel state information, time of flight, and angle of arrival. Previous studies claim that the accuracy level of decimeters can be achieved with Wi-Fi (Kotaru et al., 2015) (Kumar et al., 2014).

3.2. Bluetooth

Bluetooth is another standard wireless communication technology that uses the 2.4GHz band. Just like Wi-Fi, this technology is also prone to noise, but due to low power consumption, it might be more suitable for localization. Previous research showed that an accuracy of around 5 meters could be achieved with Bluetooth alone (Kriz et al. 2016). Moreover, in 2019 the newest version of Bluetooth 5.1 is introduced which improved its performance. Products from u-blox claimed to have a typical accuracy of 1 – 2 meters and they can even provide direction information since they use the angle of arrival technique and antenna arrays (u-blox, 2020).

3.3. RFID

This technology is originally used for storing and transferring data with electromagnetic waves and splits into two main types, which are Active and Passive RFID. Passive has a low power consumption and a lower range compared to the active type, making them unsuitable for localization. Even though the active type has a greater range, this technology can still not achieve a sub-meter level of localization accuracy.

3.4. Ultrasound

This sensor uses a sound signal over 20KHz frequency and uses the velocity of the sound to calculate the distance between its transmitter and receiver. This method is called the time of flight (ToF). On the other hand, environmental conditions have a high effect, so these conditions must be accounted for. According to the research, this technology can achieve centimeter-level accuracy (Hazas et al., 2006).

3.5. Ultra-wideband

The UWB technology uses large bandwidth of over 500MHz and a frequency range of 3.1GHz to 10.6GHz. Moreover, it has a low-duty cycle which decreases power

consumption. Another benefit of UWB is its resilience to interference and multipath effects. According to the datasheet of Decawave, the system can achieve localization accuracy of up to 10 centimeters with the time of flight (ToF) technique (Decawave, 2022).

In conclusion, UWB requires extra hardware and is costly compared to other sensors. However, it still might be the most fitting sensor for such an application due to its resilience to noise, accuracy, and low power consumption.

Although the RTLS system offers precise localization, greater accuracy is required to use it exclusively due to its limitations. Therefore it is more suitable to utilize different sources together. Aside from RTLS systems, another type of global localization system is Simultaneous Localization and Mapping (SLAM), which can be divided into two: LiDAR and Visual SLAM.



Figure 2. Decawave ultra-wideband device (Source: MDEK1001 User Manual, 2022).

3.6. LiDAR SLAM

LiDAR technologies date back to the 1960s and are mostly used in airplanes. Nonetheless, it has a wide range of applications today, including, such as geology, seismology, atmospheric physics, and autonomous vehicles. LiDARs use laser signals

to measure the distance, and range measurement is done by the ToF method. Distance is calculated with the following equation (1).

$$d = \frac{c \cdot t}{2} \tag{1}$$

where c refers to the speed of light, t refers to the time passed until the laser point returns, and d is the distance between the sensor and the object. Currently, there are two different models in the market: 2D and 3D. Essentially they work in a similar manner, but 3D LiDARs provide additional data in multiple layers ranging from 16 to 128, depending on the model. Therefore, it can increase the accuracy of SLAM and eliminate the need for extra sensors to detect obstacles since the field of view covers most of the area around the vehicle (Ouster, 2022). The placement of the sensor plays a vital role in maximizing the field of view. However, having more scan layers also increases calculation complexity since the computer must process more data quickly. They also increase the product cost compared to 2D LiDAR models.

There are many SLAM algorithms available as open-source packages. However, this paper will only discuss three of them: Hector SLAM, GMapping, Cartographer, and AMCL will be used to localize the vehicle in the frozen maps created by these SLAM techniques.

3.6.1. Hector SLAM

This algorithm solely relies on the scan data from LiDARs to create the map of the environment (Kohlbrecher et al., 2011). According to the paper, the system benefits from LiDAR, which has a high update rate and low measurement noise. Position estimation is done by scan matching using the actively created map until the last measurement, and the Gaussian-Newton equation is used to solve the scan matching problem. 3D State estimation is also possible with the help of an inertial measurement unit which enables the package to be used for Drone applications.

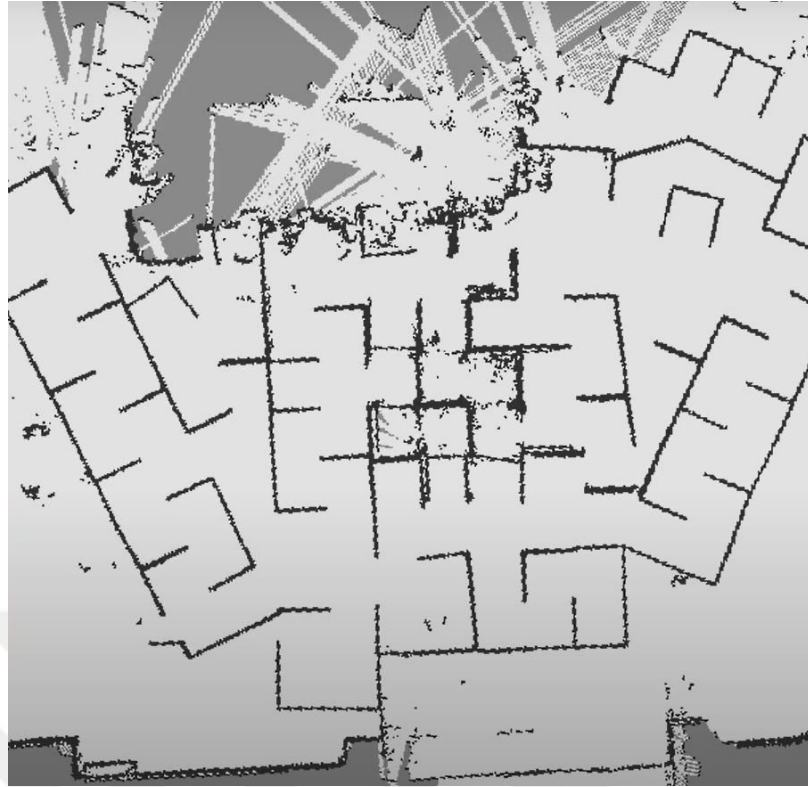


Figure 3. Hector SLAM example map (Source: Hector SLAM Documentation, 2014).

The map in Figure 3 was created in 2011 at RoboCup by a handheld system mounted with LiDAR. The map has solid measurements about obstacles and free space, which is why the map consists of three colors: black, white, and gray, which refer to filled, free space, and unknown, respectively.

3.6.2. Gmapping

This algorithm is one of the most popular SLAM packages used widely in robotic applications. It exploits the laser scan and odometry information to construct the map. The algorithm uses a particle filter-based approach which introduces computational complexity and particle depletion problem during resampling. The proposed solution to the depletion problem is an adaptive resampling technique that minimizes particle depletion (Grisetti et al., 2007). Another technique is also proposed to reduce the uncertainty in the prediction step of the filter with the help of the robot's most recent observations and movement, which lead to fewer particles.

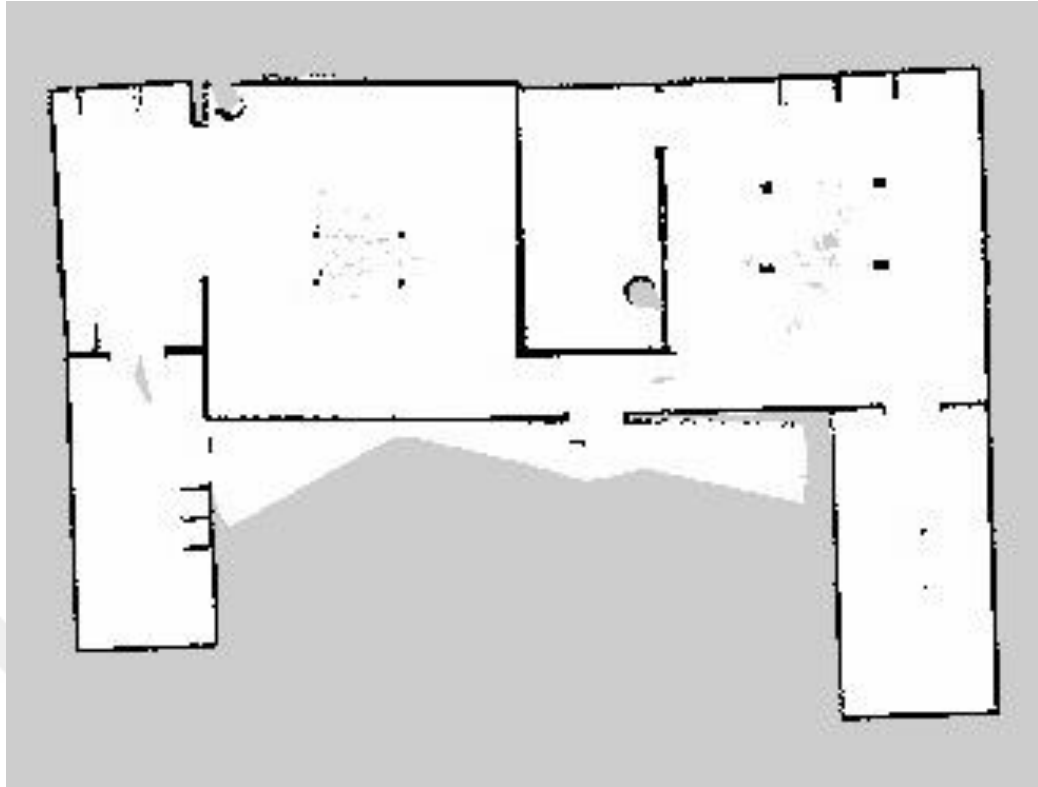


Figure 4. Gmapping example map.

As depicted in Figure 4, Gmapping also has similar output to Hector SLAM in terms of solid obstacle measurements since the map only consists of three main pixel values.

3.6.3. Cartographer

Google's Cartographer is an open-source grid-based SLAM algorithm. Although Cartographer is not directly available as a binary package for ROS, it is still supported and can be built for Kinetic, Melodic, and Noetic. According to the paper proposed system can achieve good performance without high-end hardware since it does not utilize any particle filter but pose optimizations instead. The algorithm uses a scan-to-submap matching method with a Ceres-based solver. In addition, it supports 3D SLAM, but an IMU is required to measure gravity.

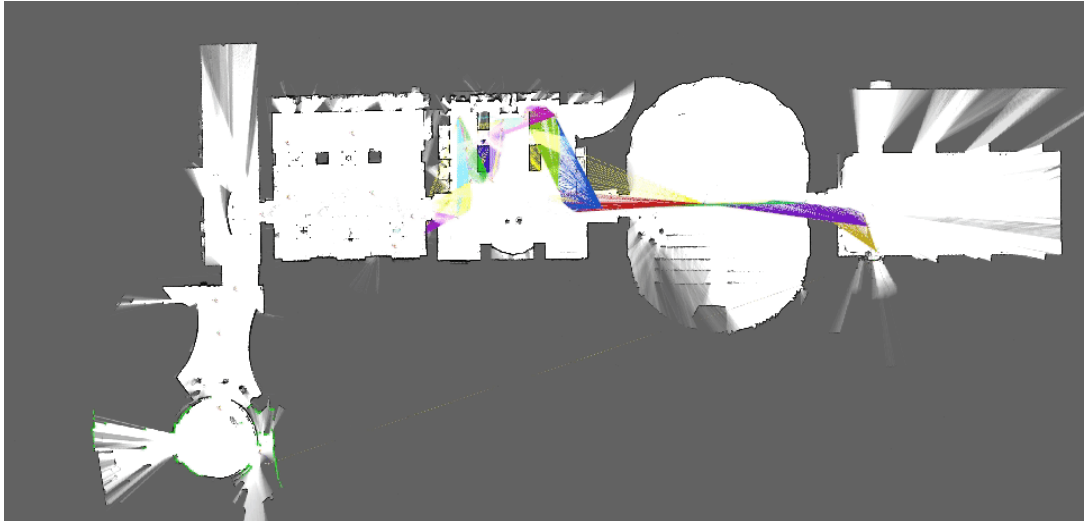


Figure 5. Cartographer example map (Source: Cartographer Documentation).

Unlike Gmapping and Hector SLAM, this algorithm leaves semi-known locations, which are shown with a color between gray and white. These pixels become whiter with each measurement and update during the mapping process. Semi-known locations do not cause any problems during navigation since they are not considered obstacles but free spaces.

3.7. Visual SLAM

Visual SLAM is a technique that uses visual inputs to construct a map, and it is often referred to as vSLAM in short. Unlike GMapping, the algorithm does not require additional sensors for mapping and localization. Nonetheless, it comes with a drawback which is the lack of field of view. The workflow of visual SLAM is composed of three basic modules, which are initialization, tracking, and mapping. The map is initialized at the first stage, where pose estimation of the camera can be done, and tracking is run over the initial features in the first key frames and new features. As more features are observed over time, the map expands.

3.7.1. ORB-SLAM

ORB-SLAM is a system that works in real-time and both for indoor and outdoor environments. According to the paper and conducted experiments, the system does not require a GPU's high processing power and manages to work in a computer with Intel Core i7-4700MQ (four cores @2.40Ghz) and 8GB RAM. This system was developed from scratch with some novel ideas and algorithms along with some old works from the loop detection of (Galvez et al., 2012), the loop closing procedure and covisibility graph of (Strasdat et al. 2010) also the optimization framework g2o by (Kuemmerle et al., 2011) and ORB features by (Rublee et al., 2011) Experiments shows that the accuracy of the system is below centimeter-level for small indoor environments. Code is also publicly available along with its ROS packages for Kinetic and Melodic. The package supports monocular, stereo, and RGB-D cameras.



Figure 6. ORB-SLAM2 map example with camera position estimations.

The authors claim that the algorithm outperforms its alternative LSD-SLAM in different datasets and sequences. In some sequences, ORB-SLAM has half as absolute translation root mean square error for the KITTI dataset. This dataset is acquired from

a stereo camera mounted on a car and has different sequences from urban and highway environments.

3.7.2. RTAB Map

Another popular visual SLAM algorithm is Real-Time Appearance-Based Mapping (Labbe et al., 2014). The algorithm uses point clouds to create a dense map which is a graph with nodes and links. Nodes have odometry poses and visualization information from sensors such as depth images and range measurements from LiDAR or images. Loop closures are done with this visualization information. Graph optimization is also used to correct the map by propagating odometry errors to all links. Tree-based network optimizer (TORO) is selected as an optimization tool.

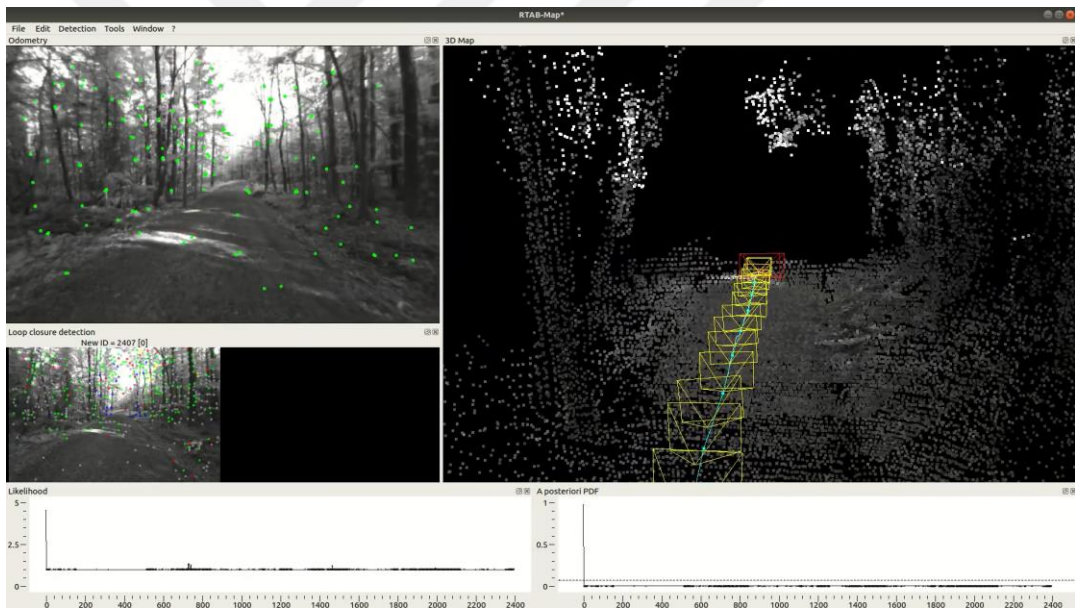


Figure 7. RTAB Map example.

Live RViz screens during the SLAM process can be seen in Figure 7. On the right window constructed map of the environment is shown as a 3D point cloud. Additionally, the estimated positions of the camera are shown as yellow rectangles. The top left corner shows odometry measurements which are extracted from visual features. Additionally, loop closure detections are shown under odometry. This live

demonstration of the algorithm is done in a forest, and the full-length recording is available online at (YouTube, 2020).

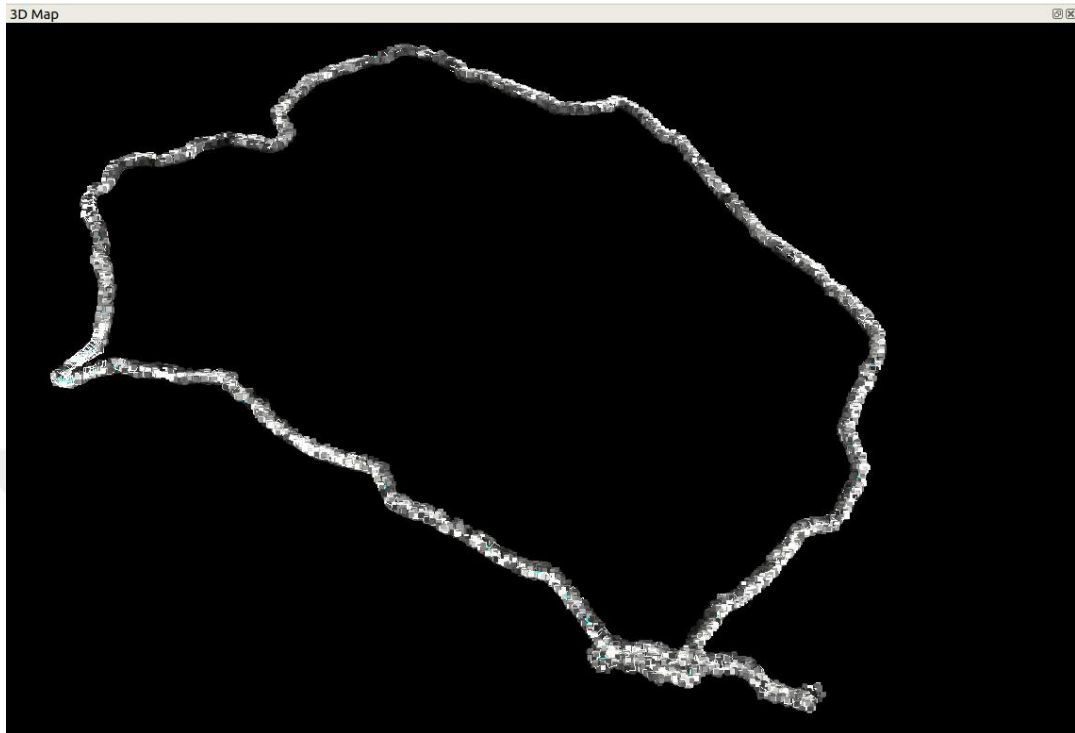


Figure 8. Completed RTAB map example.

The final form of the 3D map is shown in Figure 8. According to the demo total distance is around four kilometers.

As a result, there are many sensors and algorithms to solve localization problems, and each method has different benefits and drawbacks. Therefore, the most convenient way is to fuse multiple sensors to achieve more accurate results. Different sensors can be added and fused with the Kalman filter applications to improve the location estimation accuracy, such as inertial measurement units or GPS-like real-time localization system measurements (Moore et al., 2015).

CHAPTER 4: METHODOLOGY

This chapter will have a detailed explanation of the techniques and technologies that are used.

4.1. Robot Operating System

Robot operating system is an open-source system that offers hardware abstraction, low-level device control, and communication so that developers can focus on their area of interest without dealing with software-related problems. It also offers many open-source libraries and drivers for different types of sensors developed by the community. They are publicly available for the use of developers. ROS is used in many commercial products, from drones to robotic arms or mobile robotic platforms. ROS offers an exchange of data between processes through its publishers and subscribers. This data may be sensor data, a control message, a state, or a user-defined custom message for different use. Due to its valuable tools and packages, ROS is selected as the development and testing environment for this work since it is the most suitable system.

4.1.1. Unified Robot Description Format

This file format defines all links and joints of a robot in an XML format. Robots are described in xacro files in order for joint_state_publisher and robot_state_publisher packages. These packages provide state information about the system links and joints with the help of URDF description. Provided data mainly contains forward axes, limits, positions, and orientations of actuators and sensors. Furthermore, it is possible to visualize this model using a widget called RViz.

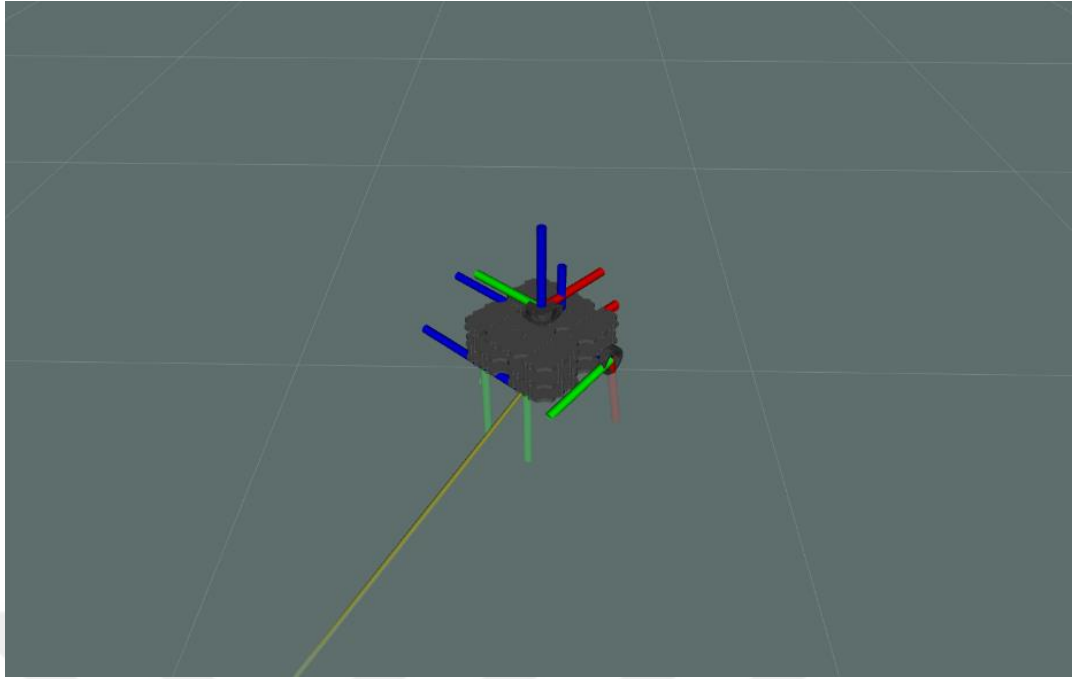


Figure 9. Visualized URDF model.

Transformation frames and the model of Turtlebot3 Waffle are visualized in Figure 9. The screenshot is taken from the RViz tool.

4.1.2. Gazebo

Gazebo is a simulation environment for robotics. It offers many plugins to simulate sensors, such as cameras, LiDARs, and depth cams like Kinect. Besides, Gazebo allows developers to simulate the physical world with high-performance physics engines like ODE and Bullet. The OGRE engine renders the graphics, offering high-quality lighting, textures, and 3D rendering. The simulation environment is also used in competitions such as Toyota Prius Challenge, DARPA Challenges, and NASA Space Robotics Challenge. Therefore, Gazebo environment will be used as the primary simulation tool for this work's development and test environment.

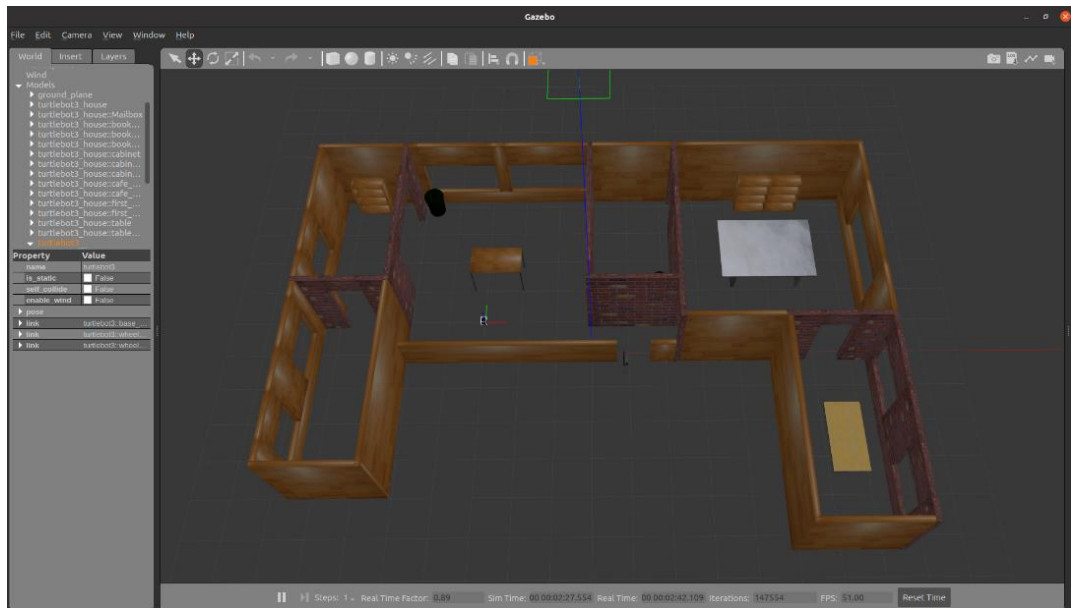


Figure 10. House environment created in Gazebo.

Many different environments and applications can be simulated through Gazebo. One use case is shown in Figure 10, where a house environment is created for simulation. For instance, one can test mapping, localization, and navigation algorithms in this simulation world. The mapping output of this environment is given in the previous chapter in *Figure 4*. This environment was initially created to simulate Turtlebot3, and models are shared on the ROBOTIS GitHub page (ROBOTIS GitHub, 2022).

4.1.3. RViz

RViz is a 3D visualization tool that allows developers to visualize any data related to the robot and its surroundings, such as sensory information or the robot's state. It also allows developers to create their plugins in order to visualize custom data types or control certain states of the robot by publishing messages. The display tab in RViz allows adding any type of message and topic for visualization by clicking the add button at the bottom left. A selection window will pop up to add anything by topic or type. Figure 11 shows the currently active displays, and the boxes next to each display allow the user to hide and show them individually.

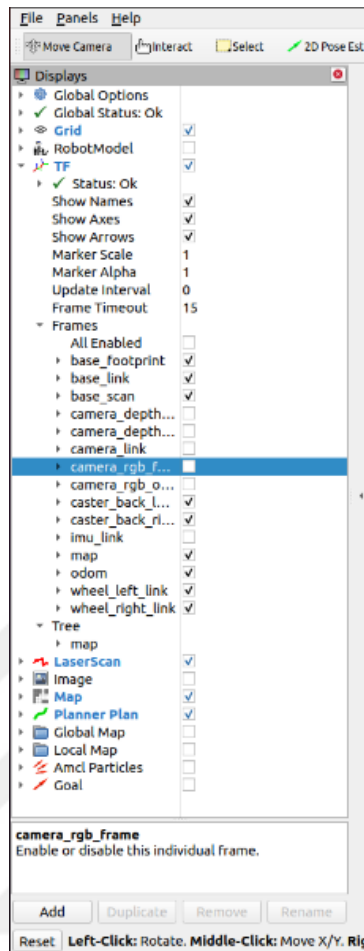


Figure 11. RViz display tab.

RViz also comes with a few default buttons that can publish the initial position to the localization package and goal point for navigation purposes. It also allows measuring distance with the measure button. Figure 12 shows all default buttons.

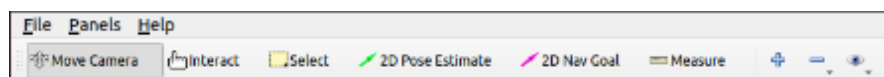


Figure 12. Default RViz buttons.

4.1.4. Transformation Package

Transform package offers a straightforward way to deal with all static and dynamic transformations between coordinate frames. For instance, calculating an object's position relative to the robot by using the sensor's known location and measurement (Foote, 2003). TF library will allow this information to be transformed so that its relative position to the robot can be calculated for obstacle avoidance or any other application. The library consists of two main modules Broadcaster and Listener. The broadcaster module publishes messages each time an update occurs for a particular transform while Listener collects this information to interpolate with the last known sample. Interpolation is done by the spherical linear interpolation (SLERP) technique. This technique makes the system resilient to package losses and allows unsynchronized communication between publisher and subscriber. Nevertheless, publish frequency of the transforms must be high enough for accurate results. Actively published transformation frames can be visualized with RViz.

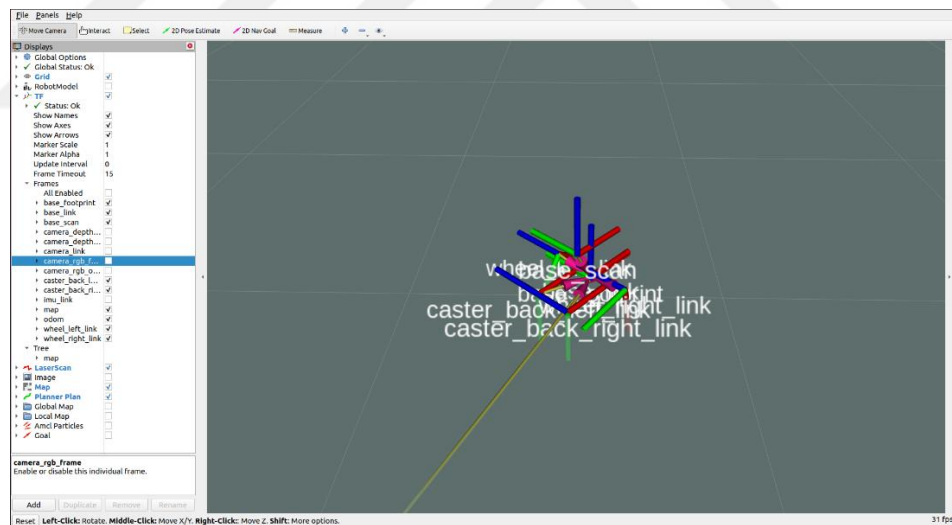


Figure 13. Visualization of transform frames.

Each component of the robot is described in the URDF file, and the transform package generates the related frames, as depicted in Figure 13. Every axis is shown with a different color.

4.1.5. Map Server

Map server is a sub-package from the ROS navigation stack to provide all necessary services to generate and use the maps. One can use this package to generate map files when mapping is finished. The package generates two files in PNG and YAML formats. The PNG file holds the map as a picture, and YAML has the map description. This description defines the map's features, such as resolution, origin, occupied threshold, free threshold, and name of the map image. A sample YAML file is as follows.

```
image: ExampleMap.png
resolution: 0.5
origin: [0.0, 0.0, 0.0]
occupied_thresh: 0.7
free_thresh: 0.2
negate: 0
```

Each field refers to the following information about the map.

- `image`: Path of the image. Both absolute and relative path is accepted
- `resolution`: Resolution of the map in meters/pixel
- `origin`: Position of the lower-left pixel in the map in `[x, y, yaw]` format
- `occupied_thresh`: Threshold value for a pixel to be considered as occupied. If the pixel value is above the threshold, it is considered occupied.
- `free_thresh`: Threshold value for a pixel to be considered free space. If the pixel value is below the threshold.
- `negate`: Reverses the calculation of occupancy probability of pixel values. If `negate` is set to true, pixel values of 255 will be considered occupied, and 0 will be free space.

The package provides two essential utilities, which are `map_server` and `map_saver`. The map server is used to publish the map under the `/map` topic and related metadata under `/map_metadata` to other ROS nodes that require this information, such as AMCL. This node can be launched by running the following in the console.

```
roslaunch map_server map_server <map.yaml>
```

As the name implies, `map_saver` saves the maps generated during SLAM, but the following process must be run through the console to save the map.

```
roslaunch map_server map_saver [--occ <threshold_occupied>] [--  
free <threshold_free>] [-f <mapname>]  
map:=/your/costmap/topic
```

However, this node requires the map data under the `/map` topic in the form of `nav_msgs/OccupancyGrid` in order to successfully save the map. If parameters for the map are not given to the process, it will automatically generate a YAML file with default parameters.

4.2. Real-Time Localization System

Real-Time Localization System is a composition of hardware and software where location data of the desired object is available with a relatively low latency depending on the system limitations. The most common example of a real-time localization system is the global positioning system (GPS) which is available to us through our everyday devices such as mobile phones, modern cars, smart watches, or computers. These systems can be used for different cases apart from being an absolute location measurement system. For example, it can work as a proximity sensor or relative location measurement device. However, these two applications are not much of use for this work since finding the absolute location is the main objective in order to achieve more precise global localization. 2D localization is sufficient for this work since our mobile robotic platform does not move in 3D, and our map is created from 2D LiDAR measurements. The system has two degrees of freedom: x and y , hence at least three sensors in known locations must be in the line of sight in order to estimate the position of an object in two dimensions. Nevertheless, this requires all devices to be placed on the same plane, which may only be feasible for some applications.

Two methods are frequently employed to measure line of sight distance:

- Signal Strength Based Ranging
- Time-Based Ranging

4.2.1. Signal Strength Based Ranging

Signal strength ranging exploits received radio signal strength to calculate the distance between sensors. It is possible to calculate the approximate distance between the transmitter and the receiver because emitted signal strength and propagation characteristics in the target medium are known.

4.2.2. Time-Based Ranging

This method uses time information that the signal takes until it reaches the receiver after it is emitted from the transmitter. Time-based ranging methods consist of four different approaches:

- Time of Flight
- Time Difference of Arrival
- Angle of Arrival

All four methods use the propagation time of the signal and the known propagation speed of the signal.

4.2.2.1. Time of Flight

This method, also known as the Time of Arrival approach, uses the propagation time between the transmitter and receiver to calculate the distance between sensors. In this technique, distance is calculated by multiplying the speed of the signal by propagation time. Measurement of this time requires synchronization between transmitter and receiver so the receiver can calculate the time of flight with the timestamp sent from the transmitter. The accuracy of this system is highly affected by the bandwidth and sampling rate. Frequency domain super-resolution techniques are used to overcome

the low sampling rate problem, and increasing the bandwidth solves the multipath problem. However, errors still exist due to missing line of sight path between transmitter and receiver. This causes wrong distance calculations since the signal propagates at different speeds in different mediums. As a consequence, the error is introduced to the results because the distance is calculated with the help of travel time and obstructions cause delays.

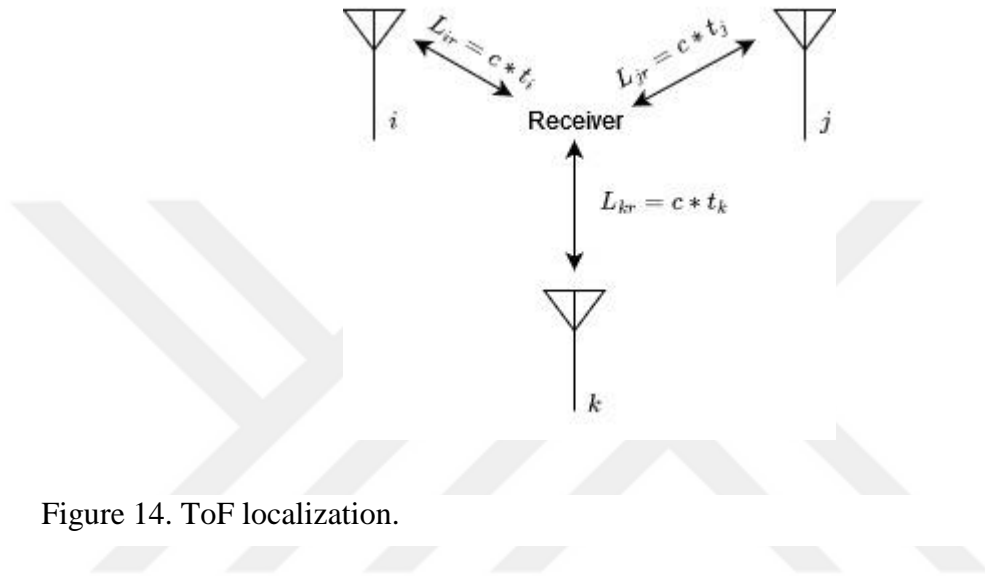


Figure 14. ToF localization.

Let L_{ij} be the distance between the transmitter and receiver, and t_1, t_2 are the timestamps of sent and received messages, respectively. So distance is calculated as follows

$$L_{ij} = (t_2 - t_1) \times v \tag{2}$$

where v is the speed of the signal.

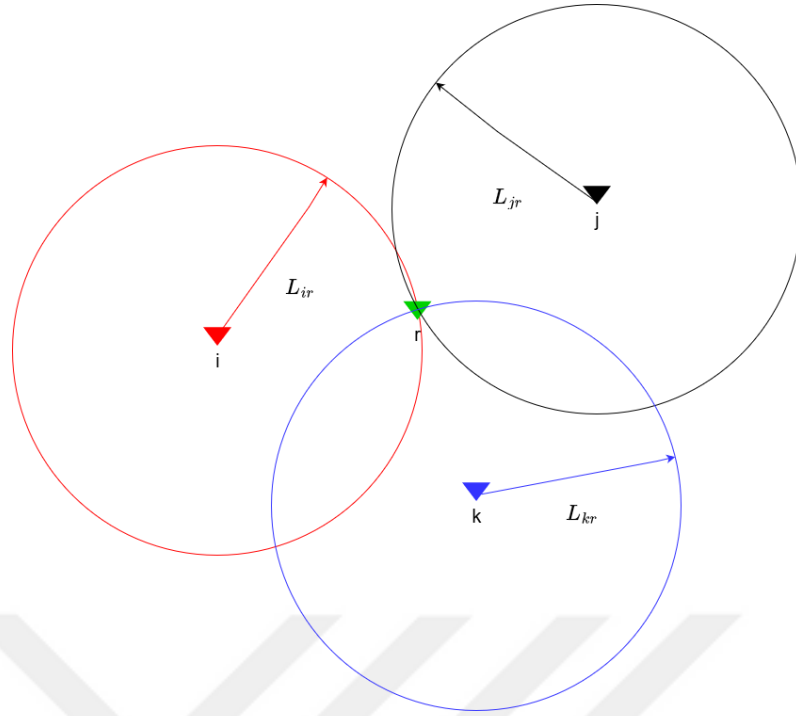


Figure 15. Possible locations of receiver relative to beacons.

After the distance for each beacon is measured, circle equations (3) will represent all the possible locations for the target receiver. Solving all three circle equations simultaneously will yield the exact location of the receiver.

$$L = \sqrt{(x_{beacon} - x)^2 + (y_{beacon} - y)^2} \quad (3)$$

4.2.2.2. Time Difference of Arrival

Similar to the time of flight, the distance between the receiver and beacon is calculated from the time difference. However, in this approach, the transmitter side does not send the time stamp; instead, the time difference between received signals is used. After each anchor receives the message, the time of arrival is stamped, and the time difference between each anchor is used to estimate the tag's position. Four anchors are required to estimate the location of the tag, and the clocks of each anchor must be accurately synchronized to estimate the tag location correctly.

4.2.2.3. Angle of Arrival

Antenna arrays are used on the receiver side to estimate the angle of arrival by calculating the time difference of arrival at each antenna. This method can localize the receiver with two transmitters in 2D and three in 3D. However, significant problems occur even with minor errors in angle calculation. The system also suffers from the multipath problem since the line of sight is not easy to achieve, and more complex hardware is required for this method (Kumar et al., 2014).

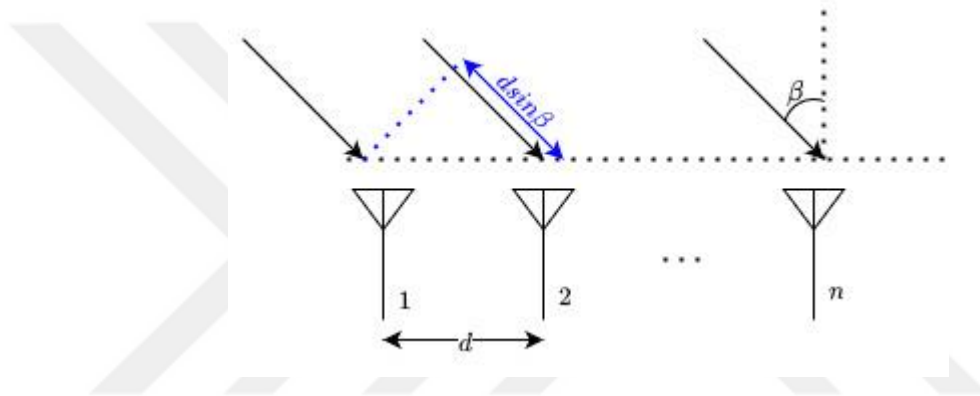


Figure 16. Antenna Example Array.

4.2.3. Decawave MDEK1001

Decawave MDEK1001 kit is based on Dacewave's UWB integrated circuit DWM1001C, which offers a Real-Time Localization system ready for development and evaluation. This kit has twelve development boards, and each unit can be used as an anchor or tag. The official DWM1001 documentation contains some guidelines to follow in order to get the best performance out of the setup. According to the documentation, DWM1001 antennas support ranges up to 60 meters under ideal conditions and have a line of sight between antennas. However, it is unlikely that the system will work with this range since the tag's orientation with respect to anchors is not always ideal and varies a lot. Thus, the manufacturer suggests reducing the range between anchors. For the best performance, 20 – 25 meters as the maximum range

between anchors is advised, and sensors should be placed to create a square grid, as shown in Figure 17. Depending on the environment, this distance should be adjusted since not being in a line of sight has an impact on system performance and localization quality.

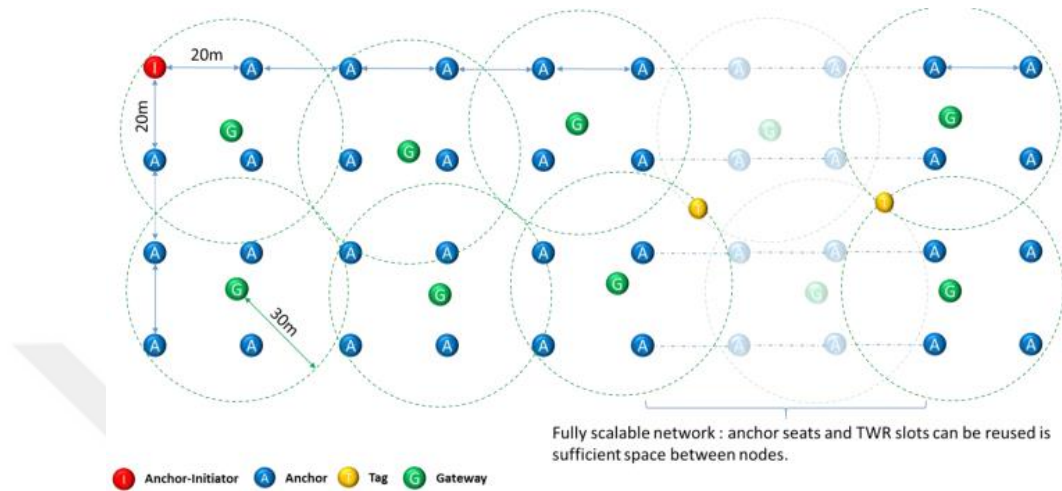


Figure 17. RTLS network installation (Source: DWM1001 System Overview, 2022).

Features and visual of the development board is shown in Figure 18. The device has three ways of streaming data to the user. Two of these are wired connections, while one is wireless:

- Decawave DRTLs Manager
- I/O Ports
- Universal Serial Bus

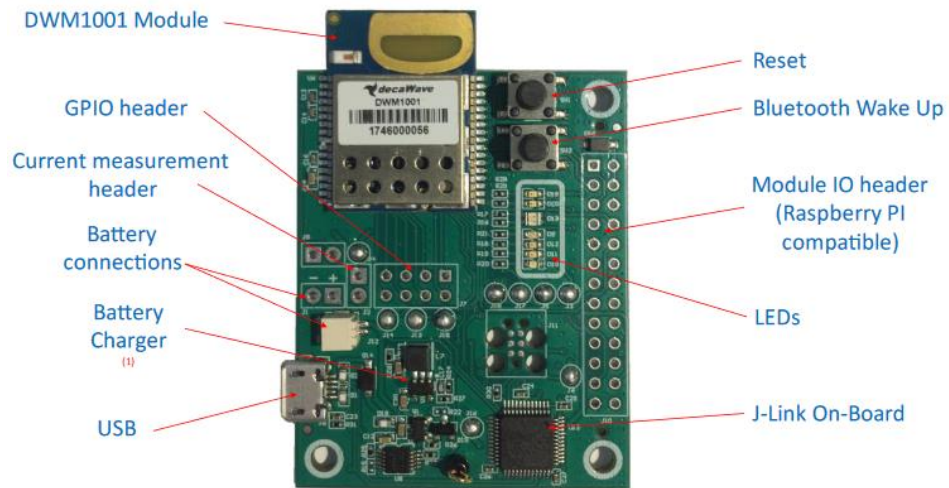


Figure 18. Decawave DWM1001-DEV module (Source: MDEK1001 User Manual, 2022)

4.2.3.1. Decawave DRTLManager

Boards come with pre-flashed firmware that supports wireless data exchange through Bluetooth and android application, which is downloadable at the Decawave’s website. This application allows users to create, configure and visualize their RTLS networks for evaluation purposes. An example of a test network created for evaluation can be seen in Figure 19

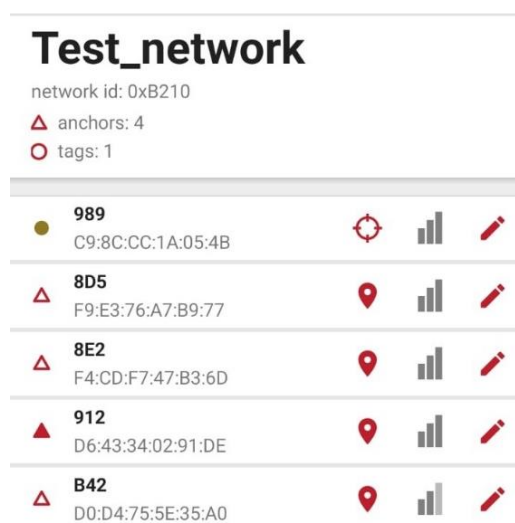


Figure 19. Example RTLS network.

Triangles represent configured units as anchors in the network, and circles represent tags. Moreover, the system provides further configuration for tags and anchors where users can change localization properties, such as updating the localization frequency or enabling motion sensors on the device for more accurate results. All available configurations for the tags can be seen in Figure 20.

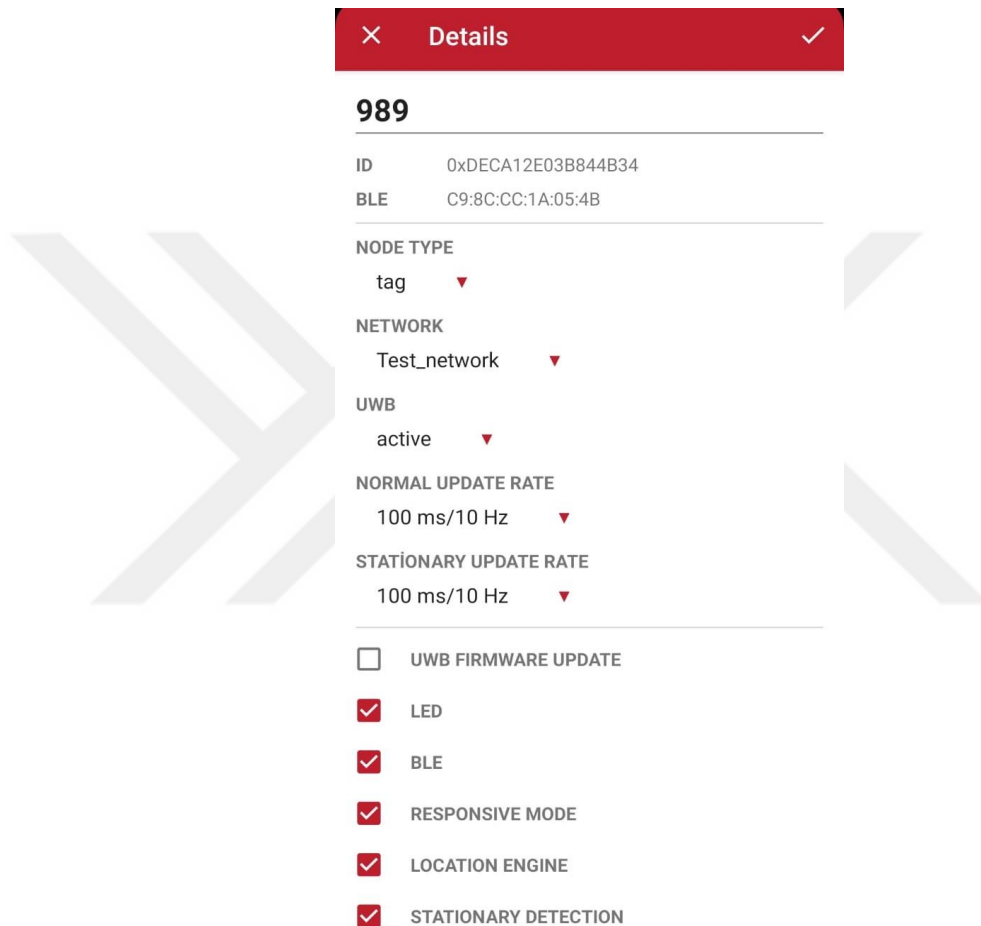


Figure 20. Device configuration screen.

After the network is configured, the app can visualize localization results. The position of the tag is shown in Figure 21, along with its estimated (x, y, z) coordinates which is relative to the anchors in the test network.

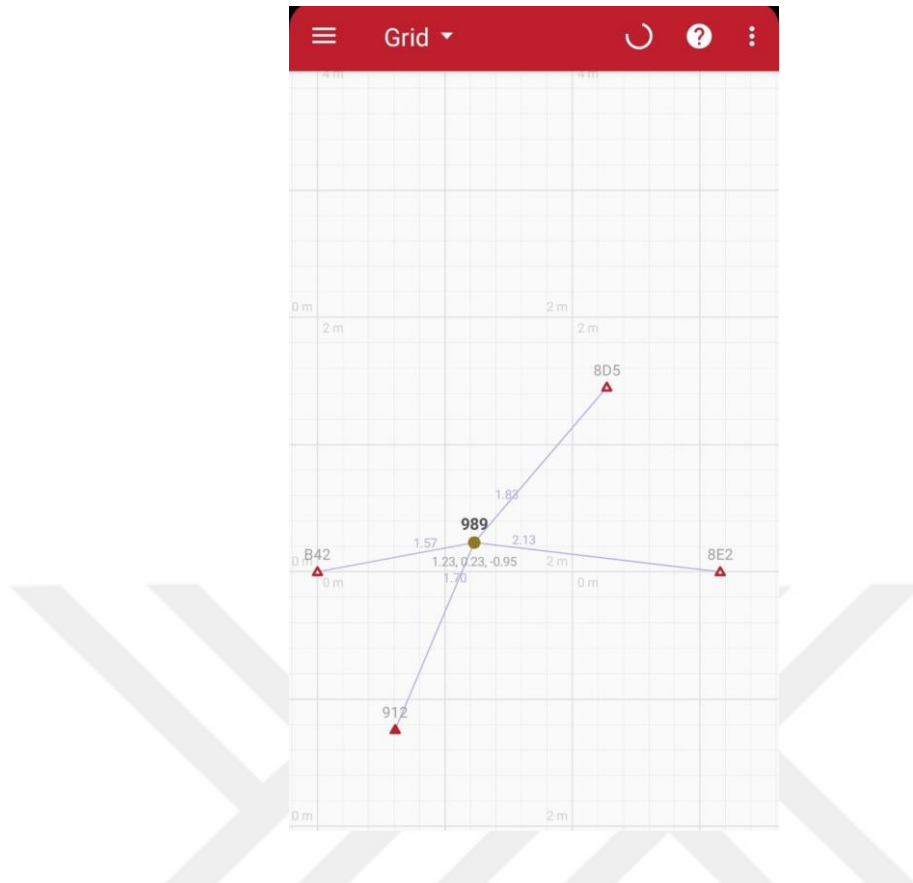


Figure 21. RTLS visualization screen (Triangles are anchor positions and circle is the position of the tag).

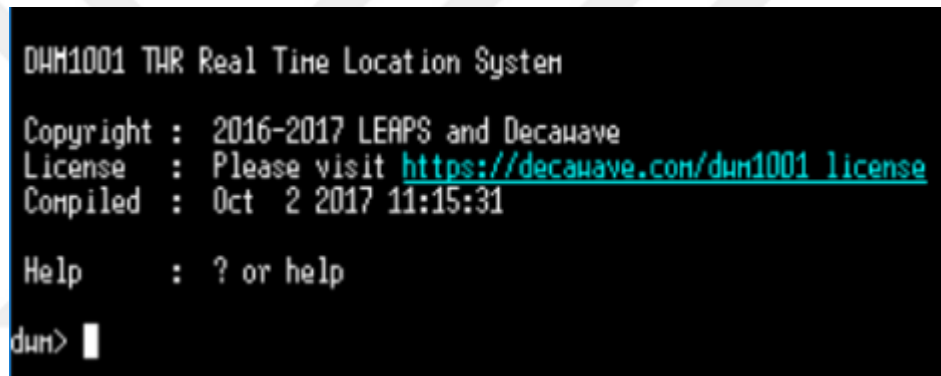
Although the app offers easy deployment features, the system is not usable by a robotic platform through an android app. Therefore, exchanging data with a wired connection is more suitable for such an application to programmatically read and publish data to ROS and its relative sub-processes.

4.2.3.2. I/O Ports

DWM1001-Dev module offers compatible pinouts with Raspberry Pi to directly connect SPI and UART channels. Either type of connection is sufficient to work with the device, so it is entirely up to the user to decide which one to use. Powering the device through these ports is also possible, and recommended voltage level is +5V, and the current level is 500mA which is a standard for USB as well.

4.2.3.3. Universal Serial Port

USB connection provides the same features as I/O ports since firmware offers commands to configure and exchange data with the board. Furthermore, it is more straightforward compared to the I/O connection and suitable for a PC connection. Data coming from the board can be visualized and logged with third-party software such as PuTTY or Tera Term. Although they are initially used as SSH and Telnet clients, they can connect to a serial port to send and receive data. After connecting to the tag, specific commands must be sent to activate the tag and start the data stream. First, double enter must be sent through the terminal. Once this command is sent following message in Figure 22 will be printed on the terminal.



```
DHM1001 THR Real Time Location System
Copyright : 2016-2017 LEAPS and Decauave
License   : Please visit https://decauave.com/dun1001\_license
Compiled  : Oct  2 2017 11:15:31
Help      : ? or help
dun> █
```

Figure 22. Terminal output.

This message indicates that the connection was successfully established with the tag. Secondly, the tag must be activated for localization with the related command, which requires “nmt” and a double return to be sent. This command will put the tag in active mode. Finally, data can be read in two formats: string or CSV. In order to trigger the data stream in CSV format “lec” command is used, and once it is sent board will start sending position information at the designated rate in *Figure 20*. A sample message is as follows

```
DIST,4,AN0,1237,0.61,1.24,0.00,0.66,AN1,D980,0.00,0.00,0.00,2.38,AN2,D81B,3.16,0.00,0.00,2.11,AN3,5529,2.27,1.45,0.00,2.61,POS,1.27,-1.07,0.29,55
```

Messages come with three main headers: DIST, AN, and POS. The first header, DIST,

has information about how many anchors are included in the position estimation; the given sample has four. The second header, AN, indicates the unique anchors and their IDs. The exact location of the anchors is also sent next to their IDs. The final header is POS which is the estimated position of the tag in the x, y, z, and quality factor, respectively. Since this work will not cover development around the DWM1001 chip, only the USB port of the module is essential to establish communication between the computer and the UWB network. More information about the rest of the commands and a detailed explanation of the usage of the device can be found at (MDEK1001 User Manual, 2020). Evaluation of the localization results will be given in the results section.

4.3. Adaptive Monte Carlo Localization

Adaptive Monte Carlo Localization is a probabilistic approach for 2D localization problems, and it is also known as particle filter localization. Initially Monte Carlo localization algorithm utilized recursive Bayesian estimation for resampling, which introduces computational complexity in the long run. However, the problem occurs when the number of particles is maintained even if particles converge to the correct location. This shortcoming is addressed and solved in an adaptive manner in the earlier work (Fox, 2001). The proposed method uses Kullback–Leibler divergence (KLD) sampling to adaptively change the sample size to decrease the number of particles to compute depending on the error. A particle filter is used to estimate the posterior distribution of the states. Posteriors can be calculated recursively with new inputs because the Markov condition assumes that the Bayesian network is memoryless, so present states do not depend on the past. Input data consists of observations denoted as z_t and control input denoted as u_t where t is the time index.

$$S_t = \{ (x_t^{(i)}, w_t^{(i)}) \mid i = 1, \dots, n \} \quad (4)$$

States are denoted as x_t and importance weights of the particles, which sums up to one is w_t . The particle filter continuously updates the beliefs at each step by sampling and resampling.

Resampling is done by selecting random samples from sample set S_{t-1} with the use of importance weights w_{t-1} . The next step is sampling which uses states x_{t-1} and control information u_{t-1} to sample x_t from $p(x_t | x_{t-1}, u_{t-1})$. Density $p(x_t | x_{t-1}, u_{t-1})Bel(x_{t-1})$ is represented by x_t , and will be used in the next step. The final step of the filter is called importance sampling, which weights the sample x_t with the likelihood of x_t given the measurement, and it is denoted as $p(z_t | x_t)$.

4.3.1 KLD - Sampling

This method is introduced to the particle filter to bound the error due to the sample-based representation. It is achieved by calculating the minimum number of particles at each sampling to maintain approximation accuracy. According to the paper (Fox, 2001) number of particles required is calculated as follows. Likelihood ratio statistic for λ_n for testing p

$$\log \lambda_n = \sum_{j=1}^k X_j \log \frac{\hat{p}_j}{p_j} = n \sum_{j=1}^k \hat{p}_j \log \frac{\hat{p}_j}{p_j} \quad (5)$$

Where n is the number of samples and k is the number of different bins. X represents the number of samples drawn from each bin, and p refers to the probability of each bin. Lastly, the maximum likelihood estimate of \hat{p} is given as $\hat{p} = n^{-1}X$. This equity reduces to the following

$$n = \frac{1}{2\varepsilon} x_{k-1,1-\delta}^2 \quad (6)$$

With the probability $1 - \delta$, Kullback–Leibler between Maximum likelihood estimate and distribution is less than error bounds. The number of samples is then calculated by approximating equation (6) by Wilson-Hilfert transformation, which results as

$$n = \frac{1}{2\varepsilon} x_{k-1, 1-\delta}^2 = \frac{k-1}{2\varepsilon} \left\{ 1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)} z_{1-\delta}} \right\}^3 \quad (7)$$

where z is the standard normal $N(0,1)$ distribution. Therefore, the sample size that is required can be calculated with an upper bound ε on K-L distance. According to the paper (Fox, 2001), a bin of multinomial distribution has support as long as the probability is above the threshold. Furthermore, k will decrease depending on the certainty of the state estimation.

4.3.2 ROS AMCL Package

There exists a package that is implemented using the theory explained in the previous section. It is offered as an open source to all users and is part of the ROS navigation stack. Package offers global localization in a given map with the help of four topics subscribed by default.

- Scan topic

This topic has the 2D point cloud information of the surroundings and is published as a `sensor_msgs/LaserScan` message type. Message mainly includes range measurements and other helpful information such as max scan range, intensities, scan time, etc. Data is published under `/scan` topic.

- Transformations

This message has the transformations of the platform and is published from the transformation package. Transformations are generated according to the description in the URDF of the platform. It is published under the `/tf` topic and has the `tf/tfMessage` format.

- Initial Position

This message is used to initialize the particle filter and triggers an action to re-distribute the particles with a given mean and covariance. The message is published under `/initialpose` topic. The message format is

geometry_msgs/PoseWithCovarianceStamped, which has pose and covariance.

- Map

This message contains the map data as an occupancy grid and is published under /map topic. The message format is defined as nav_msgs/OccupancyGrid and published by the map_server package.

These messages are required to estimate the robot's position. Once they are received, the filter outputs the following information.

- Estimated Position

This message holds the estimated position of the robot relative to the map and with covariance. The message is published under /amcl_pose, and the same format is used as initial pose geometry_msgs/PoseWithCovarianceStamped.

- Particle Cloud

This message holds the particles that are estimated with the filter. The message is published under /particlecloud, and uses the geometry_msgs/PoseArray format, which has the positions of the particles used by the filter.

- Transformation

The filter publishes the transformation between odometry and map.

The adaptive Monte Carlo Localization package is highly configurable with the exposed parameters. Both omnidirectional and differential drive is supported by default.

Table 2. AMCL Parameters.

Parameter Name	Description	Value
min_particles	Minimum number of particles	(int) 100
max_particles	Maximum number of particles	(int) 5000
kld_err	The maximum error between the actual distribution and the estimated distribution.	(double) 0.01
kld_z	Upper standard normal quantile for (1 - p), where p is the probability that the error on the estimated distribution will be less than kld_err.	(double) 0.99
update_min_d	Minimum translational movement to trigger filter update	(double) 0.2 meters
update_min_a	Minimum rotational movement to trigger filter update	(double) $\pi/6.0$ radians
resample_interval	Number of filter updates before resampling	(int) 2
transform_tolerance	Maximum tolerable transformation delay	(double) 0.1 seconds
recovery_alpha_slow	Parameter to trigger recovery behavior	(double) 0.0 (disabled)
recovery_alpha_fast	Parameter to trigger recovery behavior	(double) 0.0 (disabled)
initial_pose_x	Initial x position that particles distributed around	(double) 0.0 meters
initial_pose_y	Initial y position that particles distributed around	(double) 0.0 meters

Table 2. (Cont'd) AMCL Parameters.

initial_pose_a	Initial yaw position that particles distributed around	(double) 0.0 radians
initial_cov_xx	Covariance of the initial x position distribution	(double) 0.5*0.5 meters
initial_cov_yy	Covariance of the initial y position distribution	(double) 0.5*0.5 meters
initial_cov_aa	Covariance of the initial yaw position distribution	(double) $(\pi/12)*(\pi/12)$ radian
gui_publish_rate	Maximum rate for publishing visualization data	(double) 1.0 Hz
save_pose_rate	Maximum rate to save last pose and covariance	(double) 0.5 Hz
use_map_topic	This parameter allows AMCL to subscribe to map topic	(bool) false
first_map_only	This parameter prevents AMCL from receiving a new map.	(bool) false
selective_resampling	Reduces resampling rate and avoids particle deprivation	(bool) false
laser_min_range	Minimum scan range for a measurement to be considered in calculations	(double) -1.0
laser_max_range	Maximum scan range for a measurement to be considered in calculations	(double) -1.0
laser_max_beams	Maximum number of evenly-spaced beams used.	(int) 30
laser_z_hit	Mixture weight for the z_hit part of the model.	(double) 0.95
laser_z_short	Mixture weight for the z_short part of the model.	(double) 0.1

Table 2. (Cont'd) AMCL Parameters.

laser_z_max	Mixture weight for the z_max part of the model.	(double) 0.05
laser_z_rand	Mixture weight for the z_rand part of the model.	(double) 0.05
laser_sigma_hit	Standard deviation for Gaussian model used in z_hit part of the model.	(double) 0.2 meters
laser_lambda_short	Exponential decay parameter for z_short part of the model.	(double) 0.1
laser_likelihood_max_dist	Maximum distance to do obstacle inflation on a map, for use in the likelihood_field model.	(double) 2.0 meters
laser_model_type	Parameter to choose which model will be used likelihood_field, or likelihood_field_prob	(string) "likelihood_field_prob"
odom_model_type	Driving model of the platform, such as differential or omnidirectional	(string) "diff"
odom_alpha1	Expected noise in odometry's rotation estimate on the rotational component	(double) 0.2
odom_alpha2	Expected noise in odometry's rotation estimate on the translational component	(double) 0.2
odom_alpha3	Expected noise in odometry's translation estimate on the translational component	(double) 0.2
odom_alpha4	Expected noise in odometry's translation estimate on the rotational component	(double) 0.2

Table 2. (Cont'd) AMCL Parameters.

odom_alpha5	Translational noise only valid for omnidirectional drive	(double) 0.2
odom_frame_id	Frame id for odometry	(string) "odom"
base_frame_id	Name of the base frame	(string) "base_link"
global_frame_id	Name of the global localization frame	(string) "map"
tf_broadcast	If set to false, AMCL will no longer publish transform between odometry and global frame	(bool) True

The algorithm will be tested with the parameters in Table 2. and their respective values. As explained earlier, initialization is required for AMCL to work correctly, and Figure 23 shows an example of initialization. Particles, which are shown with red arrows, are distributed around the possible locations of the robot. Green dots depict the LiDAR measurements, which are aligned with the map, so one can assume that initialization is very close to the robot's actual location. Since the algorithm runs with the likelihood_field_prob model amount of particles is less and assembled around the same point. If the algorithm could not match the LiDAR points with the map or there were no available measurements initial distribution of the particles would be more scattered.

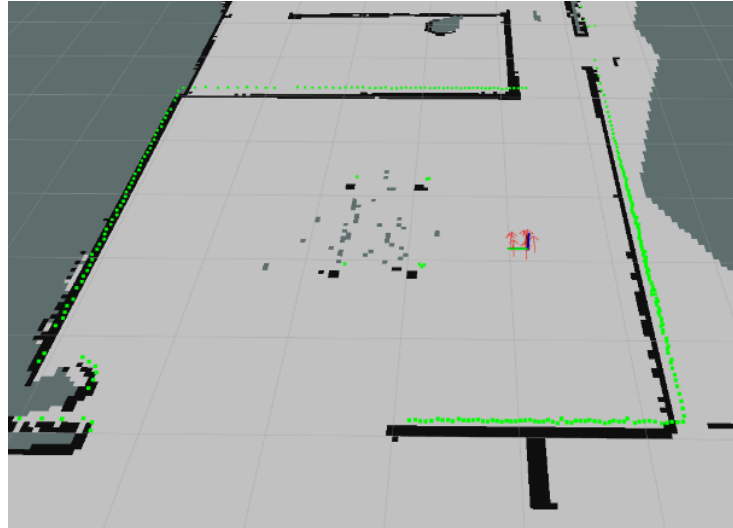


Figure 23. Initialized AMCL (Red arrows represent particles and green dots are LiDAR scans).

The actual position of the robot can be seen from Gazebo. Figure 24 shows the exact location of the robot in the simulation environment.



Figure 24. Initial location of the robot in Gazebo.

Once the robot starts moving, the algorithm will iterate with the new measurements, eventually eliminating the particles that do not match with measurements and keeping

the ones that match it. After some time, particles will gather around the point closer to the robot's actual position, as shown in Figure 25.

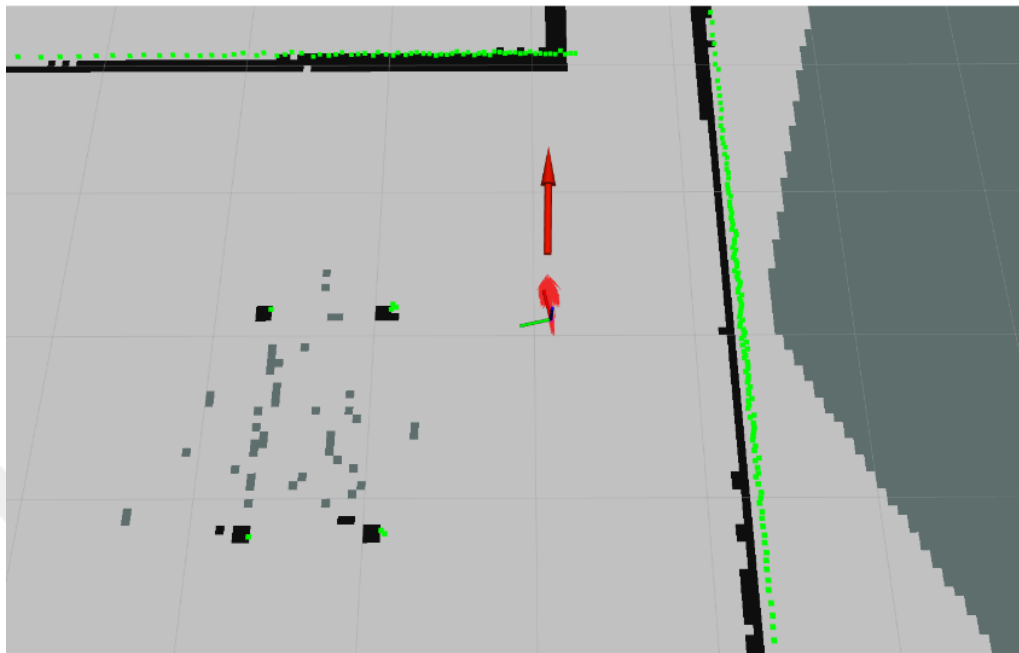


Figure 25. AMCL Pose after a few iterations (Red arrows represent particles and green dots are LiDAR scans).

Localization will successfully continue to estimate the location correctly unless measurements are not aligned with the estimated position of the robot. If the algorithm starts to receive measurements that are not matched with the map or there is nothing in the sensor's measurement range, particles will start to scatter around. This scattering will lead to wrong position estimation, and the robot will get lost in the long run. In this case, re-initialization is required. Otherwise, the robot will navigate to the wrong place or get stuck.

4.4. Extended Kalman Filter

Kalman filter is proposed as a solution to prediction problems in linear systems (Kalman, 1960). However, due to the non-linear nature of many systems and problems we face, localization issues require for specific filter modifications. Consequently,

variations of the filter are eventually developed and have been utilized to solve problems in many different fields, from economics to object tracking (Pasricha, 2006)(Tekkok et al., 2021). Therefore, it is a well-known and used filter for fusing information from different measurement sources to estimate the desired parameters of the system

Firstly, the linear Kalman filter will be described since most equations and idea behind is similar to the extended version. Kalman filter can be reduced to a single equation, but it is usually defined in two main steps: prediction and update. The prediction step is the estimate of the system with the information from the previous update or initial state. Additionally, observation information is not used in this step.

$$x_k^- = Fx_{k-1} + Bu_{k-1} \quad (8)$$

The first equation of the prediction step looks very similar to the state-space representation of dynamic systems. However, Kalman filter have a set of linear equations to transition x one step further in time instead of a state matrix. F matrix here is called the state transition matrix, which is found by solving the dynamic system model. This model is often described with a set of linear differential equations. As one can recall from linear time-invariant systems theory, it is solved by the matrix exponential method or with the Laplace transform. After the next state of the system is predicted with the mathematical model, covariance is estimated step further as follows.

$$P_k^- = FP_{k-1}F^T + Q \quad (9)$$

In equation (9), covariance estimate P is predicted with the help of process noise Q . This matrix is selected depending on the noise model of the system and must be calculated accordingly. Briefly, a small Q will make the filter more confident in its prediction model. Moreover, if values are too large, the prediction step will be introduced with unnecessary noise, leading to sub-optimal results. Therefore, modeling the noise is essential for producing accurate filter outputs, but it is usually difficult to represent the noise perfectly. Because of this, tuning is frequently necessary

to modify the filter quality.

The second step is to run the update. It can be broken down into three main equations: Kalman gain calculation, state update, and covariance update. Kalman gain decides how much weight will be put on prediction or measurement. It is calculated as follows.

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \quad (10)$$

In the univariate Kalman case, Kalman gain results in a real number between 0 and 1. For example, if the gain is calculated as 0.7, measurement will contribute to the result as 70%, and prediction will contribute as 30%. Similar logic applies to the multivariate case as well. However, in multivariate cases, gain results in a vector rather than a single number. There are new terms introduced in this step which are measurement function H and measurement noise covariance matrix R .

The measurement function converts the states to the same type of measurement. For instance, if the state is temperature and the sensor measures it as voltage, the error cannot be calculated between the two. Hence, the state must be converted such that the error can be calculated as shown in equation (11). Measurements are notated as z and $H\hat{x}_k^-$ refers to the converted state.

$$y_k = z_k - Hx_k^- \quad (11)$$

The second term R is measurement noise covariance which models the noise in the sensors. The matrix expands according to the number of sensors, so if there are n number of sensors in the system R matrix will have a dimension of $n \times n$. Once Kalman gain is calculated, as shown in equation (10), state update can be performed.

$$x_k = x_k^- + K_k(z_k - Hx_k^-) \quad (12)$$

Equation (12) is the state update equation where Kalman gain acts as a weight and sums both measurement and prediction state to better estimate the system. Lastly, covariance update is achieved with equation (13), and the update step is concluded.

$$P_k = (I - K_k H) P_k^- \quad (13)$$

The extended version of the filter has a few different operations while calculating the state transition matrix F and measurement function H . These changes are necessary since the original filter is unsuitable for non-linear cases. Therefore, linearization around the current state is required for the filter to work. Linearization is done by taking the partial derivative of the matrixes mentioned earlier, called the Jacobian matrix.

$$F = \frac{\partial f(x_t, u_t)}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad (14)$$

$$H = \frac{\partial h(x_t^-)}{\partial x^-} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \dots \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad (15)$$

Furthermore, state estimation calculation takes a different form since linearization causes inaccuracies. Hence, it is computed directly from the non-linear function f , as shown in equation (16).

$$x^- = f(x, u) \quad (16)$$

For the same reason, states are converted by computing $h(x^-)$ directly as follows.

$$y = z - h(x^-) \quad (17)$$

To summarize, the extended Kalman filter following equations are used to calculate prediction and update steps.

Table 3. Kalman equations table.

Prediction Step	Update Step
(1) Predict the state $x^- = f(x, u)$	(3) Compute Kalman $K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$
(2) Calculate covariance $P_k^- = F P_{k-1} F^T + Q$	(4) Update the estimate $x_k = x_k^- + K_k (z_k - H x_k^-)$
	(5) Update error covariance $P_k = (I - K_k H) P_k^-$

In conclusion, this is how the extended Kalman filter is described. The following section will investigate the generic application of EKF as a ROS package.

4.5. Robot Localization Package

Robot localization is an open-source ROS package for sensor fusion applications with two supported filters: the extended Kalman filter and the unscented Kalman filter. The package supports an unlimited number of inputs from different types of sensors, such as GPS, IMU, and Odometry. The theory behind the package is described in detail in the research paper (Moore et al., 2015). The system is described in equation (18) with a non-linear state transition matrix f and process noise w which is assumed to be normally distributed.

$$x_k = f(x_{k-1}) + w_{k-1} \quad (18)$$

State vector x is 12 - dimensional vector that includes position and orientation in three dimensions along with their respective velocities. Additionally, the measurement model of the system is described in equation (19), where h is a non-linear sensor

model to convert states to measurement space and normally distributed measurement noise v_k .

$$z_k = h(x_k) + v_k \quad (19)$$

Filter equations are the same as described in the previous section except for the covariance update (20), which is in Joseph form to keep P_k as a positive semi-definite.

$$P_k = (I - KH)\bar{P}_k(I - KH)^T + KRK^T \quad (20)$$

The authors also propose to take H matrix as an identity matrix to accommodate different sensors. Therefore, if there are n number of variables being measured, H matrix will have the shape of $n \times 12$, and nonzero values will be related columns of the measured variables. Process noise covariance is exposed, so it becomes a configurable parameter depending on the application.

Customization of the filter can be done with the YAML file. The package supports different messages as input to the filter, but they must be defined in the YAML file according to their related message types. For instance, data acquired from wheel encoders must be converted to a `nav_msgs/Odometry` message and published under any desired topic name. Once measurement messages are published, each sensor can be added to the configuration file with its name and index, so for the wheel encoder case, it must be defined as `odom<N>: /name_of_the_odometry_topic`. Each input is sequentially indexed starting from zero, therefore `<N>` must be replaced with a related index, and the topic's name must be inserted as its value. Supported message types that can be used as described are:

- `nav_msgs/Odometry`
This message holds position and velocity estimates in `PoseWithCovariance` and `TwistWithCovariance` format.
- `sensor_msgs/Imu`
This message holds measurement data from the inertial measurement unit under

orientation, angular velocity, linear acceleration, and their relative covariance as a 3×3 matrix.

- `geometry_msgs/PoseWithCovarianceStamped`
This message holds a position estimate with a reference frame and time stamp.
- `geometry_msgs/TwistWithCovarianceStamped`
This message holds velocity estimates with reference frame and timestamp.

Filter estimates fifteen states (21) of the vehicle using the message types given above, and sensors should provide data related to the states.

$$(X, Y, Z, roll, pitch, yaw, \dot{X}, \dot{Y}, \dot{Z}, \ddot{X}, \ddot{Y}, \ddot{Z}) \quad (21)$$

After the measurement source is defined in YAML as `odom0` new variable must be defined as `odom0_config`, which is a boolean vector where each element refers to the given states at (21) respectively. Considering a 2D localization case of an autonomous robot, one can set the vector as follows.

```
odom0_config: [false, false, false,
               false, false, false,
               true, true, false,
               false, false, true,
               false, false, false]
```

Another critical parameter is `two_d_mode` which zeros out the 3D pose variables if set to true. Since our robot operates on an even surface and does not leave the ground, this parameter will be true. The parameters used in tests and simulations are listed in Appendix A and B.

CHAPTER 5: SIMULATION RESULTS

This chapter will explain how sensors and algorithms are tested and simulated, along with the results.

5.1. UWB Localization Tests

The measurement errors of ultra-wideband sensors will be evaluated in order to emulate their noise in a simulation environment. Sensors are tested both for obstructed and unobstructed setups. Figure 26 shows the results from both test setups.

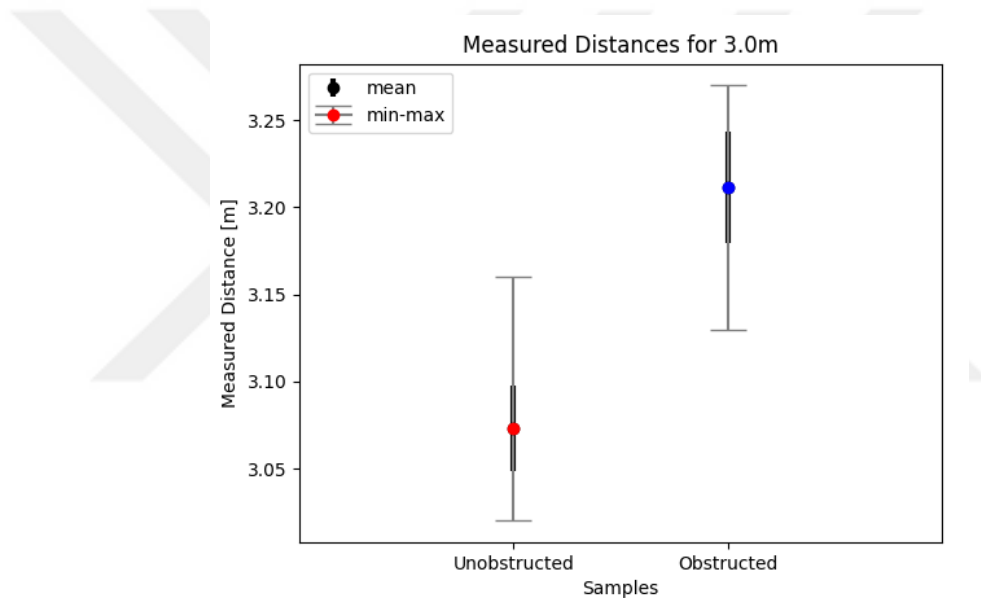


Figure 26. Ultra-wideband range measurements.

The left-hand side depicts the minimum maximum and average measurement values when sensors are not obstructed, and the right side shows the measurements when a wall obstructs the sensor. Around a hundred measurements were taken at the same place. The following results have been gathered from the tests and are shown in Table 4.

Table 4. Ultra-wideband measurement results.

	Unobstructed			Obstructed
True distance (m)	0.7	1.6	3.0	3.0
Mean (m)	0.637	1.752	3.073	3.211
Standard deviation	0.0259	0.0309	0.0244	0.0322
Minimum (m)	0.57	1.69	3.02	3.13
Maximum (m)	0.7	1.84	3.16	3.27

Histograms of the above measurements are shown in Table 4. Based on the tests, Gaussian noise with a standard deviation of 0.0285 will be used to imitate the natural behavior of the sensor in the simulation as noise. This value is selected since it is the mean of obstructed and unobstructed test cases.

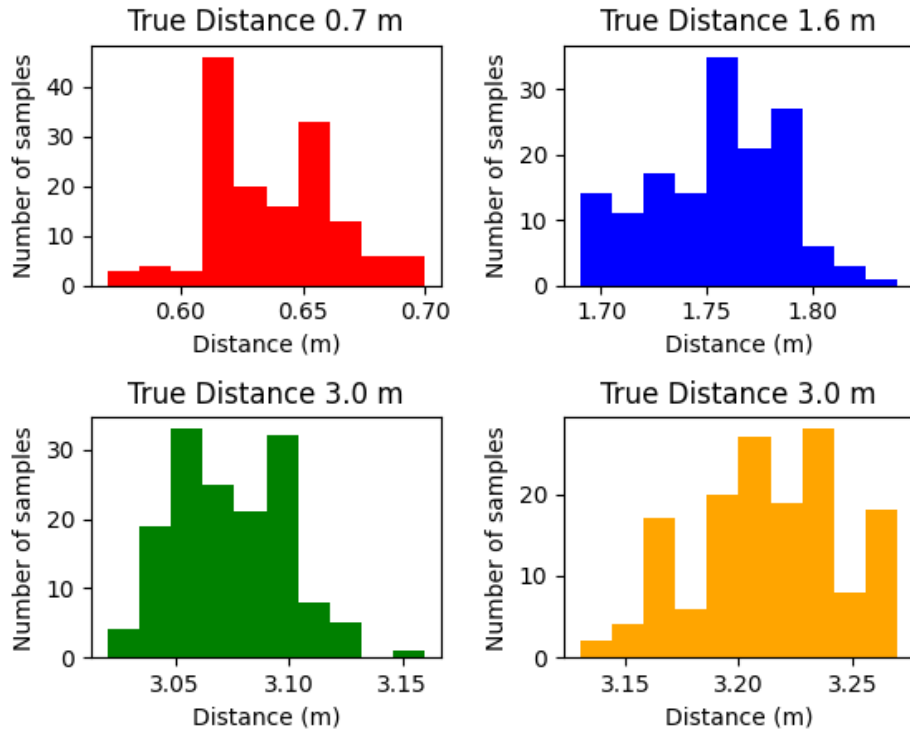


Figure 27. Histogram of measurements.

5.2. Adaptive Monte Carlo Localization Tests

This section will cover the tests with the Monte Carlo localization, and problems described in the problem statement will be shown in the simulation environment. The first custom environment is created for tests consisting of 6 x 3 evenly separated blocks with dimensions of 15 x 1.2 meters. Blocks are separated with 5 meters gaps to create corridors. The simulation world in Gazebo is shown in Figure 28, where red blocks represent outer walls and yellow blocks separate corridors.

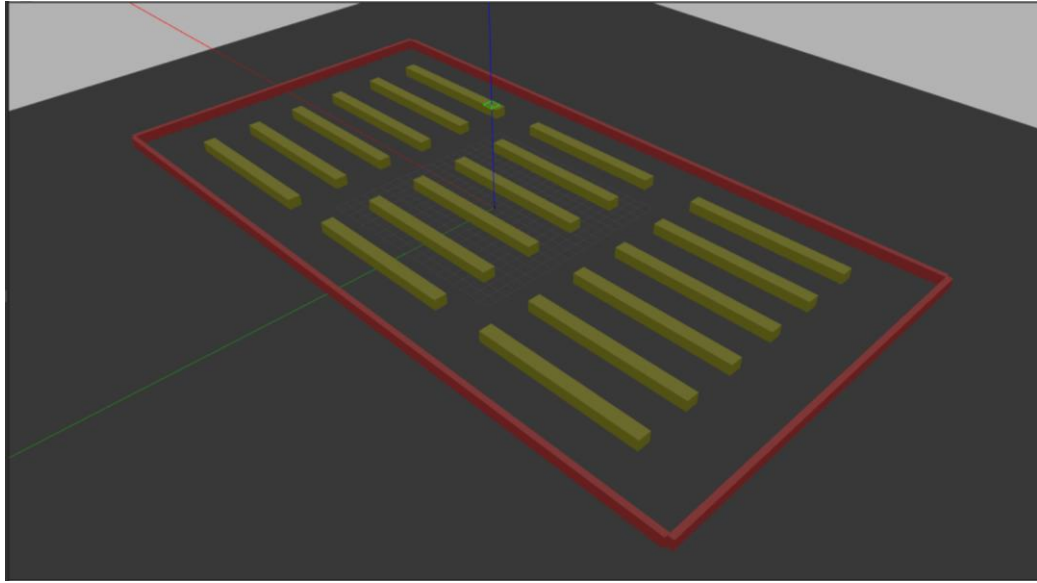


Figure 28. Simulation world in Gazebo.

Once the simulation is up and running, the robot can start mapping the environment. In order to generate the map, the Gmapping algorithm is used. The final map is depicted in Figure 29, where dark gray areas in the inner part refer to the yellow blocks in Figure 28. Generated map is 4000 x 4000 pixels, and the resolution is 0.05

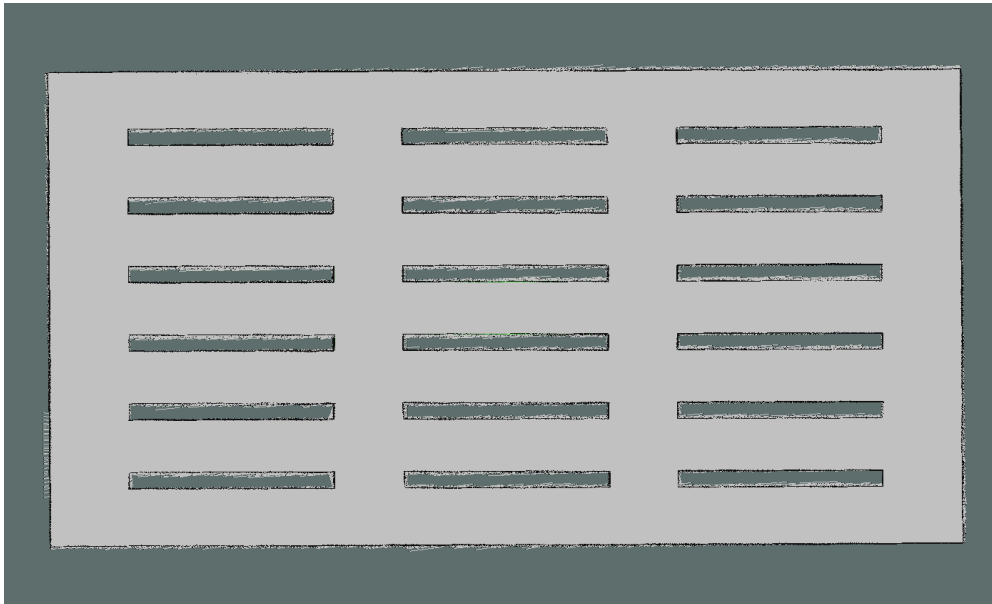


Figure 29. Map of the simulated world.

After the map is generated, localization can be tested. The robot will start moving from the center of the map and will continue to navigate until it reaches $(-5, 0)$. Both ground truth and AMCL pose will be saved and printed on the map during navigation. In these tests, all blocks will remain in the world so that LiDAR will have accurate measurements of the environment.

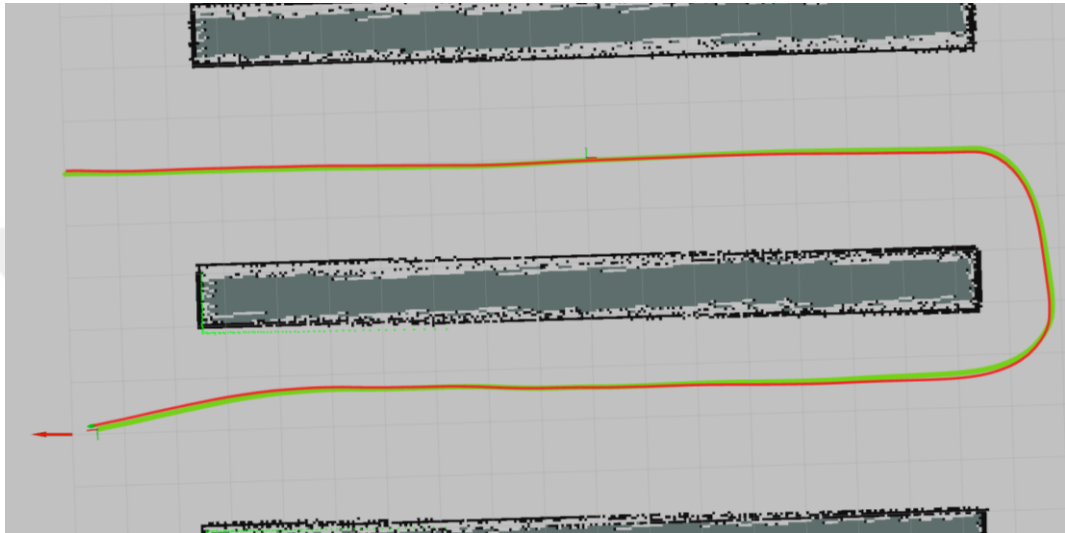


Figure 30. Localization results with AMCL (Green line is ground truth and red line is EKF output).

The resulting paths are shown in Figure 30, where the green line refers to the ground truth, and the red is the AMCL position output. Since the environment is unchanged, AMCL accurately estimated the robot's position. AMCL is tested in the whole map, and the algorithm can maintain accurate estimation if the environment is static and unchanged. However, the issue arises in highly dynamic environments when LiDAR measurements do not match the map. An example of this problem can be seen in Figure 31. Red arrows represent AMCL particles. At the initial state, particles are unified at the starting location, but once the robot moves, they spread around. When all particles are considered, the output position estimation of the filter is still close to the actual position. However, once a certain amount of distance is traveled position estimation starts to jump. These jumps eventually lead to the robot getting stuck in a static obstacle or getting lost entirely such that it will not recover on its own. The following

section will cover an example of this situation, along with the proposed solution and its results.

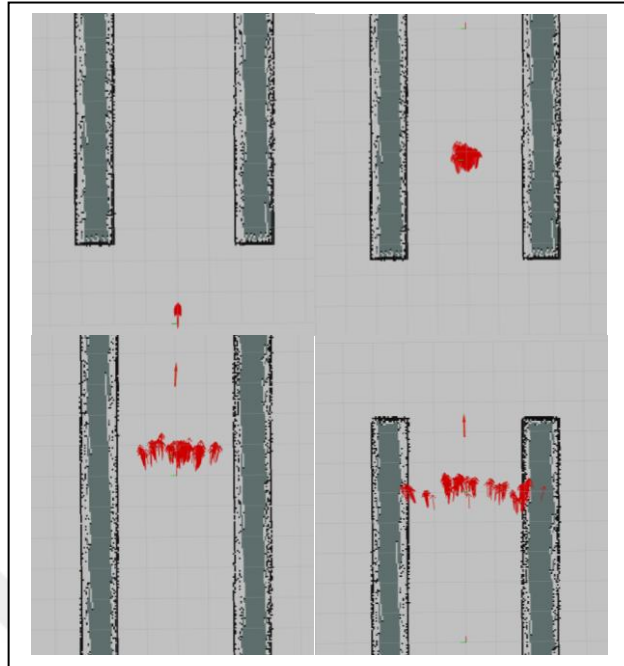


Figure 31. Particles when LiDAR does not measure any features in environment.

5.3. Localization tests with the fusion of RTLS and AMCL

This section will describe the proposed solution to solve unreliable position estimation problems when LiDAR can not receive correct measurements. Furthermore, the test setup and results will also be presented.

Firstly, an imitation of the RTLS system must be simulated in the ROS environment to provide position measurement data to the robot localization package. Since the simulation world is created in Gazebo, it is possible to retrieve the exact location of the robot. Location data will be published from a Gazebo plugin `libgazebo_ros_p3d` defined in the `xacro` file as follows.

```
<plugin name="p3d_base_controller"  
filename="libgazebo_ros_p3d.so">  
  <alwaysOn>true</alwaysOn>  
  <updateRate>50.0</updateRate>  
  <bodyName>base_link</bodyName>  
  <topicName>ground_truth/state</topicName>  
  <gaussianNoise>0.00</gaussianNoise>  
  <frameName>world</frameName>  
  <xyzOffsets>0 0 0</xyzOffsets>  
  <rpyOffsets>0 0 0</rpyOffsets>  
</plugin>
```

This plugin will publish the robot's exact position in the desired topic as `nav_msgs/Odometry` message. Once these messages are received from a python script, Gaussian noise with $\mu = 0$ and $\sigma = 0.0285$ will be added to (x, y) positions, and they will be republished with 10 Hz under `/uwb_pose` topic.

For `robot_localization` to work, transform publishers must also be re-arranged. When AMCL works as the only global estimator, the `tf_broadcast` option must be set to true to let AMCL handle the transformation between the global and odometry frame. However, after an extended Kalman filter is introduced, this transformation will be published from the `robot_localization` package, and the `tf_broadcast` option must be changed to false. A new frame for RTLS messages must also be defined according to the sensor's location with respect to the robot. However, since the exact position of the robot is taken from the simulation world, this frame is the same as the robot's base link, which is the center of the robot.

Tests are conducted in a simulation environment with an extended Kalman filter and AMCL. The problem is reproduced and shown in Figure 32. The red line depicts the AMCL position output, and as the robot continues to move, it starts to jump around the true position of the robot. Green dots also show the Monte Carlo particles, which are spread around since blocks in the middle are removed, and LiDAR is unable to measure enough features. Unlike AMCL, the extended Kalman filter outputs more stable and continuous position estimation. The blue line shows the extended Kalman filter output, and sudden jumps do not appear when two sources are fused.

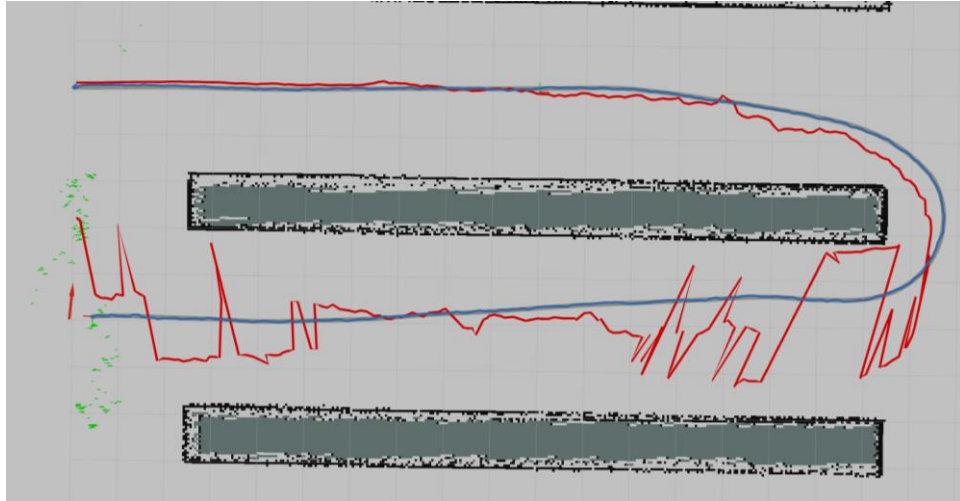


Figure 32. Localization results with UWB and EKF (Blue line is ground truth and red line is AMCL output).

Although sensor fusion results in a more robust positioning and does not prevent navigation from working safely, AMCL still needs to be manually or programmatically corrected once LiDAR starts getting measurements from the environment. Otherwise, AMCL will keep estimating the position wrong, making the extended Kalman filter ignore the jumps and estimations from AMCL since pose rejection is set in the configuration file.

CHAPTER 6: CONCLUSION

This paper showed that global localization from a single source might cause problems. This research tested AMCL as a global localization source, and tests showed that AMCL fails in dynamic environments where LiDAR is unable to capture enough features from the environment. The problem is reproduced, and the solution is tested in a simulation environment.

The proposed solution to the problem is fusing two global localization sources to have a more accurate and stable estimation. The second source is a real-time localization system with ultra-wideband sensors that provides absolute position information. These sensors are configured as tags or anchors and placed in the robot's operation area to cover the mapped region. Furthermore, line of sight is crucial since obstructions cause measurement errors, as shown in tests. Additionally, max range of sensors must be considered during placement of the sensors. The sample setup for the system is given in Figure 17, and a re-arrangement of sensors might be required depending on the application area.

In conclusion, fusing global localization estimations offers sufficient and stable results compared to using a single source only. Therefore, supporting the AMCL from a different source solves the localization problem in highly dynamic environments for robots with limited LiDAR measurement range. Another way to eliminate the problem could be using high-end LiDARs with increased ranges up to a few tens of meters long. However, this will introduce higher setup costs, and as the LiDAR range increases, even minor changes in ground slope could create high errors while measuring objects far from the sensor. So, one should consider all these while designing such a system.

6.1. Future work

Although the proposed solution is a good way to eliminate localization issues in the described environments, a few problems still exist if AMCL goes far away from the actual position. It is observed in the tests that AMCL is not able to recover from such

high errors if measured features match with someplace else. In such an event, position estimations of AMCL will be useless, and particles must be reinitialized. Therefore, a mechanism to reinitialize the particles at the correct location must be designed and implemented to have a more robust system.



REFERENCES

- [matlabbe]. (1 June 2020). Four Kilometers Walk in Forest (an uncut real-time visual SLAM demo) [Video File]. Available at: <https://www.youtube.com/watch?v=G-5jesjNfLc>
- Decawave. (2022) REAL-TIME LOCATION SYSTEMS. [Online] Available at: https://www.decawave.com/sites/default/files/resources/aps003_dw1000_rtls_introduction.pdf. (Accessed: 16 May 2022).
- DWM1001 System Overview and Performance. (2022), DWM1001 Documents [Online], Available at: <https://www.qorvo.com/products/p/DWM1001C#documents> (Accessed: 20 October 2022).
- Foote, T. (2013) *tf: The transform library*, IEEE Conference on Technologies for Practical Robot Applications (TePRA), pp. 1-6.
- Fox D., Burgard W., Thrun S. (eds.)(2005). *Probabilistic Robotics*. MIT Press. Available at Google Books. (Accessed: 06 April 2022).
- Fox, D. (2001) *KLD-sampling: Adaptive particle filters and mobile robot localization*. Advances in Neural Information Processing Systems, vol. 14, no. 1.
- Galvez-López, D., and Tardos, J. D. (2012). *Bags of Binary Words for Fast Place Recognition in Image Sequences*. IEEE Transactions on Robotics, vol. 28, no. 5, pp. 1188–1197.
- Grisetti, G., Stachniss, C., and Burgard, W. (2007). *Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters*. IEEE Transactions on Robotics, vol. 23, no. 1, pp. 34-46.
- Hahnel, D., Burgard, W., Fox, D., and Thrun, S. (2003). *An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements*. IEEE/RSJ International Conference on Intelligent Robots and Systems, vol.1, pp. 206-211.
- Hahnel, D., Burgard, W., Fox, D., Fishkin, K., and Philipose, M. (2004, April). *Mapping and localization with RFID technology*. IEEE International Conference on Robotics and Automation. Proceedings. ICRA '04. Available at: <https://ieeexplore.ieee.org/document/1307283>
- Hazas, M., and Hopper, A. (2006). *Broadband ultrasonic location systems for*

improved indoor positioning. IEEE Transactions on Mobile Computing , vol. 5 no. 5, pp. 536–547.

Hector SLAM Documentation, (2014). *Hector SLAM* [Online], Available at: http://wiki.ros.org/hector_slam (Accessed: 06 April 2022).

Kalman, R. E. (1960). *A New Approach to Linear Filtering and Prediction Problems*. In Journal of Basic Engineering. vol. 82, no. 1, pp. 35–45.

Kohlbrecher, S., von Stryk, O., Meyer, J., and Klingauf, U. (2011). *A flexible and scalable SLAM system with full 3D motion estimation*. 9th IEEE International Symposium on Safety, Security, and Rescue Robotics, , pp. 155-160.

Kotaru, M., Joshi, K., Bharadia, D., and Katti, S. (2015). *SpotFi*. ACM SIGCOMM Computer Communication Review, vol. 45, no. 4, pp. 269–282.

Kriz, P., Maly, F., and Kozel, T. (2016). *Improving Indoor Localization Using Bluetooth Low Energy Beacons*. Mobile Information Systems vol. 2016, pp. 1–11.

Kumar, S., Gil, S., Katabi, D., and Rus, D. (2014). *Accurate indoor localization with zero start-up cost*. Proceedings of the 20th annual international conference on Mobile computing and networking. MobiCom'14: The 20th Annual International Conference on Mobile Computing and Networking, pp. 483–494.

Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). *G2o: A general framework for graph optimization*. 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 3607-3613.

Labbe, M., and Michaud, F. (2014, September). *Online global loop closure detection for large-scale multi-session graph-based SLAM*. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2661-2666. Available at: <https://ieeexplore.ieee.org/document/6942926>

Lasmadi, L., Kurniawan, F., Dermawan, D., and Pratama, G. N. P. (2019, December). *Mobile Robot Localization via Unscented Kalman Filter*. 2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), pp. 129-132. Available at: <https://ieeexplore.ieee.org/document/9034570>

Lecture Notes. (2002). *Carnegie Mellon University lecture notes* [Online]. Available at: <https://www.cs.cmu.edu/~rasc/Download/AMRobots1.pdf> (Accessed: 06 April 2022).

MDEK1001 User Manual. (2022), MDEK1001 Documents [Online], Available at:

<https://www.qorvo.com/products/p/MDEK1001> (Accessed: 20 October 2022).

Moore, T., and Stouch, D. (2015, September). *A Generalized Extended Kalman Filter Implementation for the Robot Operating System*. Intelligent Autonomous Systems 13, pp. 335–348.

Mur-Artal, R., and Tardos, J. D. (2016). *ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras*. IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262.

Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015, October). *ORB-SLAM: A Versatile and Accurate Monocular SLAM System*. IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163.

Oguz-Ekim, P., Bostanci, B., Tekkok, S. C., and Soyunmez, E. (2020, October). *The EKF based Localization and Initialization Algorithms with UWB and Odometry for Indoor Applications and ROS Ecosystem*. 2020 28th Signal Processing and Communications Applications Conference (SIU), pp. 1-4. Available at: <https://ieeexplore.ieee.org/document/9302137>

Ouster. (2022) Ouster OS1 LiDAR Scanners [Online]. Available at: <https://ouster.com/products/scanning-lidar/os1-sensor/> (Accessed: 21 May 2022)

Pasricha, G. K. (2006). *Kalman filter and its economic applications*. MPRA [Online]. Available at: https://mpra.ub.uni-muenchen.de/22734/1/MPRA_paper_22734.pdf (Accessed: 09 April 2022).

Pierce F. (2020). *IKEA switches from timber to cardboard pallets* [Online] Available at: <https://supplychaindigital.com/supply-chain-risk-management/ikea-switches-timber-cardboard-pallets> (Accessed: 8 December 2020)

ROBOTIS GitHub. (2022). Simulations for TurtleBot3 [Online]. Available at: https://github.com/ROBOTIS-GIT/turtlebot3_simulations

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011, November). *ORB: An efficient alternative to SIFT or SURF*. 2011 International Conference on Computer Vision, pp. 2564-2571. Available at: <https://ieeexplore.ieee.org/document/6126544>

Strasdat, H., Davison, A. J., Montiel, J. M. M., and Konolige, K. (2011, November). *Double window optimisation for constant time visual SLAM*. 2011 International Conference on Computer Vision, pp. 2352-2359. Available at: <https://ieeexplore.ieee.org/document/6126517>

- Strasdat, H., M. M. Montiel, J., and Davison, A. (2011). *Scale Drift-Aware Large Scale Monocular SLAM*. Robotics: Science and Systems 2011 [Online]. Available at: <http://www.roboticsproceedings.org/rss06/p10.pdf> (Accessed: 13 April 2022).
- Tekkok, S. C., Soyunmez, M. E., Bostanci, B., and Ekim, P. O. (2021, June). *Face Detection, Tracking and Recognition with Artificial Intelligence*. 2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), pp. 1-5. Available at <https://ieeexplore.ieee.org/document/9461356>
- u-blox. (2020) Bluetooth Indoor Positioning [Online]. Available at: <https://www.u-blox.com/en/technologies/bluetooth-indoor-positioning> (Accessed: 26 December 2022).
- Wang, Y., Zhang, W., Li, F., Shi, Y., Chen, Z., Nie, F., Zhu, C., and Huang, Q. (2019, October). *An improved Adaptive Monte Carlo Localization Algorithm Fused with Ultra Wideband Sensor*. 2019 IEEE International Conference on Advanced Robotics and its Social Impacts (ARSO), pp. 421-426. Available at: <https://ieeexplore.ieee.org/document/8948809>
- Wing M. G., Eklund A., and Kellogg L.D., *Consumer-Grade Global Positioning System (GPS) Accuracy and Reliability*, Journal of Forestry, vol. 103, no. 4, pp. 169–173.
- Yuzhen, P., Quande, Y., and Benfa, Z. (2016, May). *The application of adaptive extended Kalman filter in mobile robot localization*. 2016 Chinese Control and Decision Conference (CCDC), pp. 5337-5342.
- Zhang, G., Lee, J. H., Lim, J., and Suh, I. H. (2015, December). *Building a 3-D Line-Based Map Using Stereo SLAM*. IEEE Transactions on Robotics, vol. 31, no. 6, pp. 1364-1377.

APPENDICES

Appendix A-Robot Localization Package Odom Parameters

```
frequency: 60
two_d_mode: true
use_control: false
control_config: [true, false, false, false, false, true]

base_link_frame: base_footprint
odom_frame: odom
world_frame: odom
map_frame: map
publish_tf: true

odom0: /odom
odom0_config: [false, false, false,
               false, false, false,
               true, true, false,
               false, false, true,
               false, false, false]
odom0_differential: false
```

Appendix B- Robot Localization Package Map Parameters

```
frequency: 10
two_d_mode: true

map_frame: map
odom_frame: odom
base_link_frame: base_footprint
world_frame: map

publish_tf: true
transform_time_offset: 0.1

smooth_lagged_data: false
dynamic_process_noise_covariance: false

odom0: /odom
odom0_config: [false, false, false,
               false, false, false,
               true, true, false,
               false, false, true,
               false, false, false]

odom0_differential: false

pose1: uwb_pose
pose1_config: [true, true, false,
               false, false, false,
               false, false, false,
               false, false, false,
               false, false, false]

pose1_rejection_threshold: 3

pose0: amcl_pose
pose0_config: [true, true, false,
               false, false, true,
               false, false, false,
               false, false, false,
               false, false, false]

pose0_rejection_threshold: 4
```