TECHNICAL UNIVERSITY OF CRETE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
TELECOMMUNICATIONS DIVISION

Experimental Study of Simultaneous Localization and Mapping Algorithms on the Turtlebot Platform

by

Emmanouil Andrianakis

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DIPLOMA OF

ELECTRICAL AND COMPUTER ENGINEERING

February 2020

THESIS COMMITTEE

Professor Aggelos Bletsas, *Thesis Supervisor*
Professor Kostas Kalaitzakis
Associate Professor Michail G. Lagoudakis

Abstract

This work merges robotics and radio frequency identification (RFID) technologies, with the goal to enable precise RFID tag localization. A low-cost robotic platform was used to create a mobile robot equipped with multiple modules for RFID inventorying and enhanced perception of robot's environment. Various modifications were performed on the robot, so it could support all the additional modules. Open source software was exploited for the creation of accurate maps and estimation of the robot's pose. Prior art localization methods, based on optimization and particle filtering were exploited, using phase measurements of the RF signal. Implementation and experimental results showed mean location estimation error of 0.2 m and 0.04 m, for two dimensions and one dimension, respectively. Execution times suggest that for robot velocity of 10 cm/sec, the offered implementations were real time.

Thesis Supervisor: Professor Aggelos Bletsas

Acknowledgments

First of all, I would like to thank my supervisor Prof. Aggelos Bletsas for his guidance and support throughout this work as well as the Telecommunications Lab for providing all the necessary equipment for fulfilling this thesis.

My friends and colleagues from Telecommunications Lab, especially G. Vougioukas and K. Skyvalakis for the looting of knowledge we performed together.

My family and friends, especially E. Giannelos, A. Doko, A. Neli, A. Polychronakis, A. Zgourakis and E. J. Kenway for all the wonderful time we spent together.

Table of Contents

List of Figures

Chapter 1

Introduction

Nowadays, inventorying and asset tracking are very important problems in almost every organization. The most common example is that of a library and misplaced or missing books. In this case, someone has to find any kind of mistake, but this is time-consuming and humans are not perfect so more mistakes may occur. Another example could be at a warehouse, if something gets lost or if you have boxes with items and you need a specific item but you can't remember in which box it is placed.

In situations like these, where line of sight (LOS) methods are not available, RFID tags are a possible solution. They can be read in non-LOS scenarios, but the reading range is often some meters, so the exact location remains unknown. The goal of this thesis is to test whether the scenarios described above, can be resolved in an automated way in order to precisely estimate (within centimeters) the location of these assets using RFID tags.

Similar approaches, for example in [1], are trying to include reference tags in known locations and calculate the angle of arrival of unknown tags. Furthermore, in [2] the use of excessive bandwidth is exploited. Finally, approaches like [3], are taking advantage of optimization processes to estimate the tag's location but many consecutive measurements are required.

In this thesis, the latter approach is implemented as well as a particle filter approach found in [4] [5]. Chapter 2 offers a description of hardware and software used. Chapters 3 and 4 describe algorithms used for robot localization and RFID tag localization respectively. Finally Chapter 5 presents all the experimental results and Chapter 6 includes our conclusions and possible future improvements.

Chapter 2

Robot Architecture

2.1 Our Robot

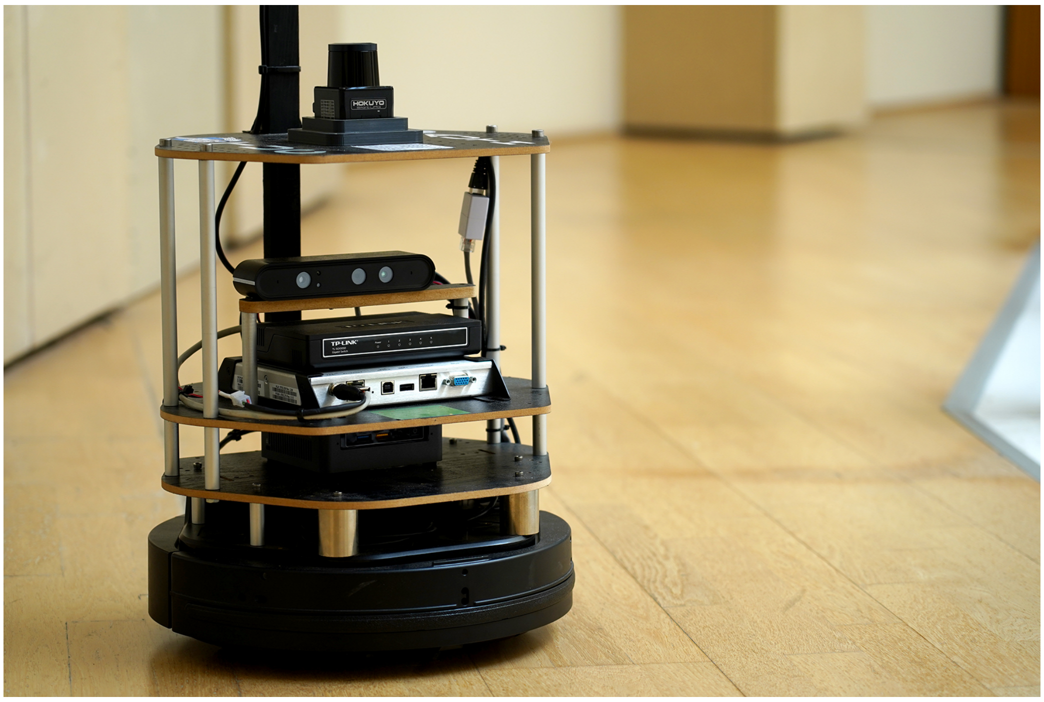


Figure 2.1: Our Robot Fully Assembled.

2.1.1 The Turtlebot Platform

Turtlebot is a low-cost and easy-to-use robotic platform designed for education and research on state-of-the-art robotics. It was created at Willow Garage by Melonee Wise and Tully Foote in November 2010. The kit includes a mobile base, a 3D sensor, a computer and a stack-able structure

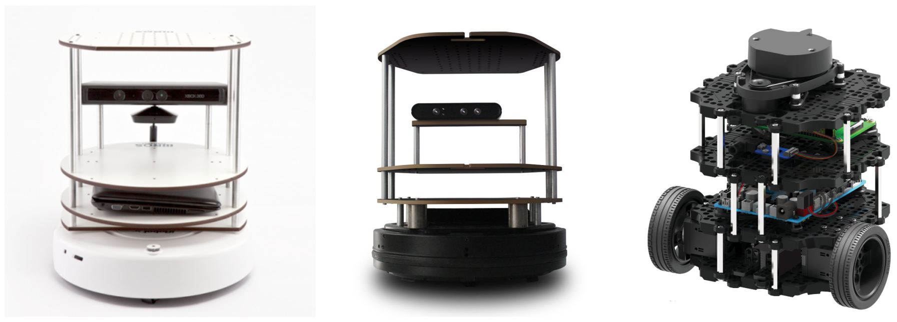


Figure 2.2: Turtlebot Family.

where additional modules can be mounted. Its software is open source and there are plenty ready-to-use ROS packages available.

The original Turtlebot (Fig. 2.2a) was based on the iRobot Create which was an upgraded version of the well-known robotic vacuum cleaner Roomba. In 2012, Turtlebot's mobile base was upgraded to a Kobuki Base, manufactured by Yujin Robot (Turtlebot 2 Fig. 2.2b). The new base offered more features like analog inputs, more power connectors, touch buttons, indication LEDs and a better battery. Finally in 2017, Turtlebot 3 (Fig. 2.2c) was announced. It was developed by ROBOTIS and Open Source Robotics Foundation. This was a major upgrade offering a lower overall cost, a smaller footprint, better motors, a Single Board Computer and a laser distance sensor.

2.1.2 Kobuki Base

The Base of our robot is a Kobuki Mobile Base made by Yujin Robot. It is responsible for motion control as well as providing power to the additional modules installed on the robot. The base is powered by a 4S2P Lithium-Ion battery pack with a nominal voltage of 14.8 V and a capacity of 4400 mA h. The battery can power the system for up to 7 hours according to manufacturer specifications. This proved to be quite accurate from our experience operating the robot. The base provides the following connectors for supplying power to the additional modules:

- 19 V/2 A: Laptop power supply.
- 12 V/5 A: Arm power supply.
- 12 V/1.5 A: Microsoft Kinect power supply.
- 5 V/1 A: General power supply.

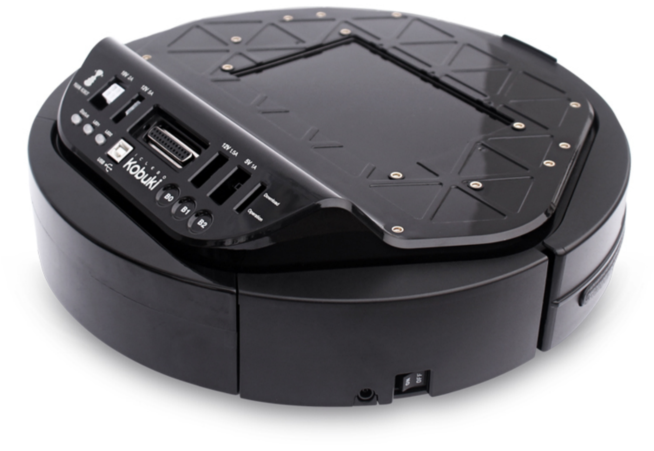


Figure 2.3: Kobuki Base.

The 19 V/2 A provides power when the base is charging so it can't be used during operation. The 2nd port is designed to provide power to high power accessories like a robotic arm. We use this port to provide power to all of our 12 V modules through a custom made distribution board further described in Sec. 2.1.7. The next port is provided for supplying power to a Microsoft Kinect but in our case, we don't use a Kinect as a 3D sensor so this spare port is used for DC-to-DC converters, so we can provide various voltage levels depending on our needs.

The base is configured as a differential drive base, with one wheel on each side. Wheels are powered through a geared motor and each wheel contains a built-in encoder with a resolution of 2578.33 $\frac{\text{ticks}}{\text{wheel rev}}$ or 11.7 $\frac{\text{ticks}}{\text{mm}}$. The base can achieve a maximum translational velocity of 70 cm/s and a maximum rotational velocity of 180 deg/s. The base supports payloads up to 5 kg.

In respect to sensors, Kobuki base contains a factory calibrated single-axis Gyroscope, 3 bumpers for collision detection, 3 cliff sensors and one wheel drop sensor in each wheel. The base also provides some basic programmable input-output interfaces: 4 analog inputs, 4 digital inputs, 4 digital outputs, 2 bi-color LEDs, 3 touch buttons and a serial port. PC connectivity is achieved via a USB port. Finally, all the above are controlled through an STM 32-bit microcontroller.

2.1.3 Hokuyo UST-20LX

Figure 2.4: Hokuyo UST-20LX mounted on Turtlebot.

As a range finder, a Hokuyo UST-20LX is deployed. This device is based on Light Detection and Ranging (LiDAR) technology. Its working principle is to send rapid light pulses and measure the reflection time. Then it calculates the distance from the point that the pulse got reflected from the equation,

$$\text{Distance} = \frac{c \cdot \Delta\tau}{2}, \tag{2.1}$$

where c is the speed of light and $\Delta\tau$ is the duration of the pulse's round trip.

Hokuyo UST-20LX is a 2D laser scanner which means that this measurement is repeated for different angles on a 270° field of view with a resolution of 0.25°. The detection distance ranges from 0.06 m to 20 m with an accuracy of ±40mm. The device is rated for 10 V to 30 V, it draws 450 mA at start up and 150 mA during operation, so it can be safely powered from the Kobuki Base. PC connectivity is achieved via an Ethernet port.

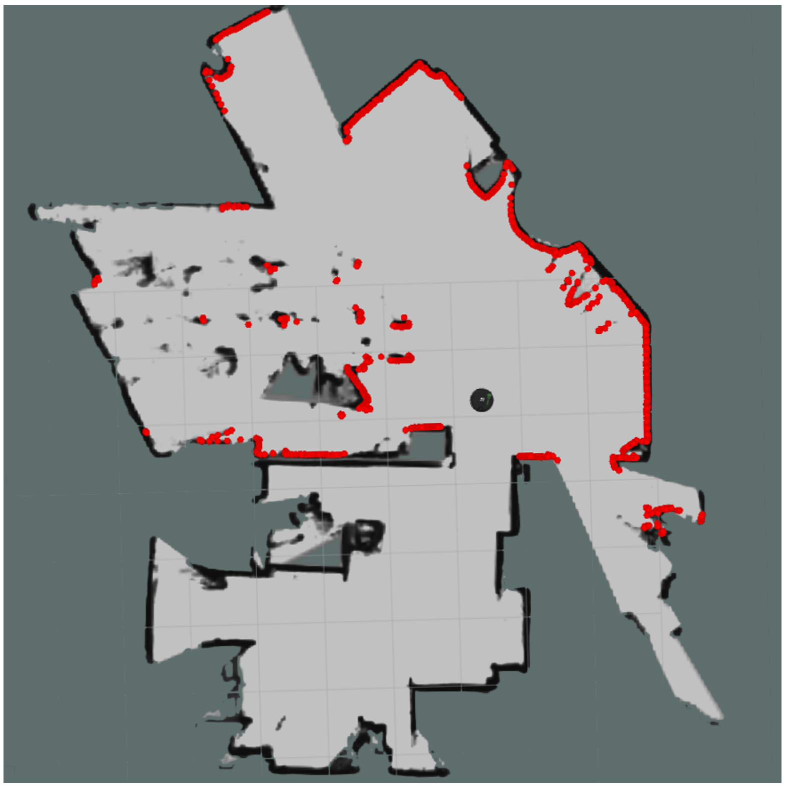
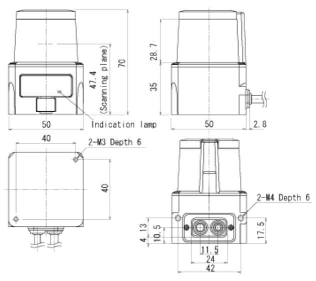


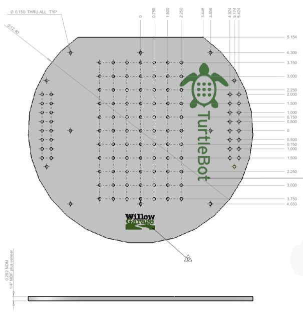
Figure 2.5: Laser Scan Example. Red Bots represent distance measurements.

This device is installed on the top mounting surface of the Turtlebot Platform. The hole grid pattern on the mounting surface (Fig. 2.6b) was incompatible with the mounting holes of the LiDAR (Fig. 2.6a). A special base had to be designed so that the LiDAR could be mounted safely and reliably without any modification to either device. Taking into account the dimensions of the device and the hole grid pattern a base was designed (Fig. 2.6c)[1]. Then it was manufactured using 3D printing technology.

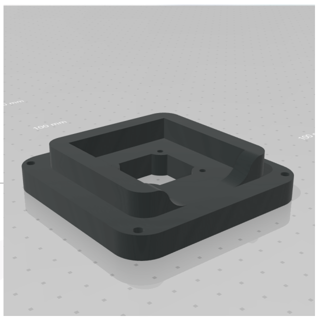
[1]The base was designed by Evaggelos Giannelos (Thanks).



Lidar's Dimensions.



Top Plate Blueprint.



3D Design.

Figure 2.6: Lidar's Base.

2.1.4 Orbbec Astra Camera

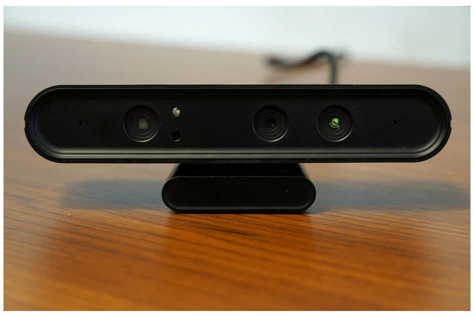
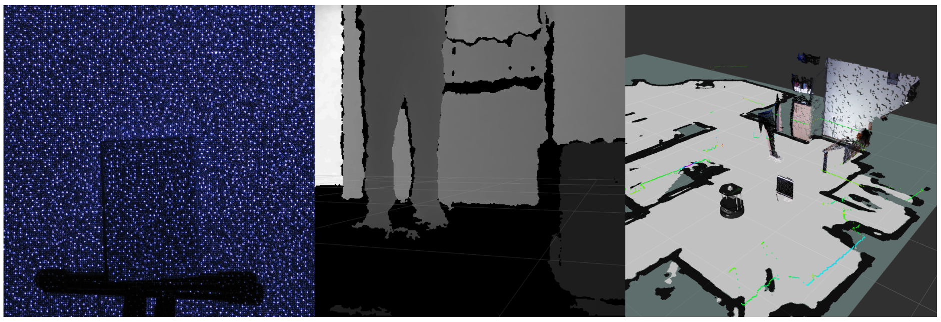


Figure 2.7: Front View of the Depth Camera.

For depth perception, our robot utilized an Orbbec Astra Camera. Depth map is calculated by projecting a unique IR dot pattern not visible by the human eye. Figure 2.8a demonstrates such a pattern projected on a wall. Then a CMOS sensor captures a frame of the environment and with the help of epipolar geometry, depth information can be extracted. To view that information, a depth image can be used where darker areas represent objects closer from the camera while lighter areas represent objects far from the camera (see Fig. 2.8b). 3D representation is also available using Point Clouds (see Fig. 2.8c). Also, Astra camera contains an RGB Image Sensor. Both images (depth and RGB) come with a resolution of 640 x 480 pixels and a Field of View of 60°H x 49.5°V x 73°D. Objects can be detected from 0.6 m up to 8 m. Accuracy is not listed in the manufacturer's specifications.

This device is powered from a USB 2.0 connection.



Dot Pattern. Depth Image. Point Cloud.

Figure 2.8

2.1.5 RFID Tag Readers

An RFID Tag Reader was mandatory in our experiments. Two commercial readers were tested to find out which one better satisfies our requirements. The first one was a Thingmagic Sargas (Fig. 2.9a) and the second an Impinj Speedway R420 (Fig. 2.9b). Both of them are small factor UHF Gen2 RFID Tag Readers with a maximum transmit power of 30 dbm and support of multiple antennas. Each one had its dedicated API. In the early stages of the robot's development, Thingmagic Sargas was used because it requires 5 V to operate. This ensured that it could be powered directly from Kobuki's Base General power supply port, in contrast to 24 V required by Impinj Speedway R420. Furthermore, the API supplied by Thingmagic was much simpler to use, in combination with ROS API. But after further review, it was found that phase measurements were unstable and had half of the expected range. We decided to continue our experiments with Impinj Speedway R420. That was a challenge because of three main reasons:

- It required a different voltage level from that supplied by Kobuki Base.
- Problems with the provided API further described in Sec. 2.2.3.
- Requirement for an additional Ethernet port in the system.

To solve the last problem, an Ethernet 5-Port Switch was installed on our robot.

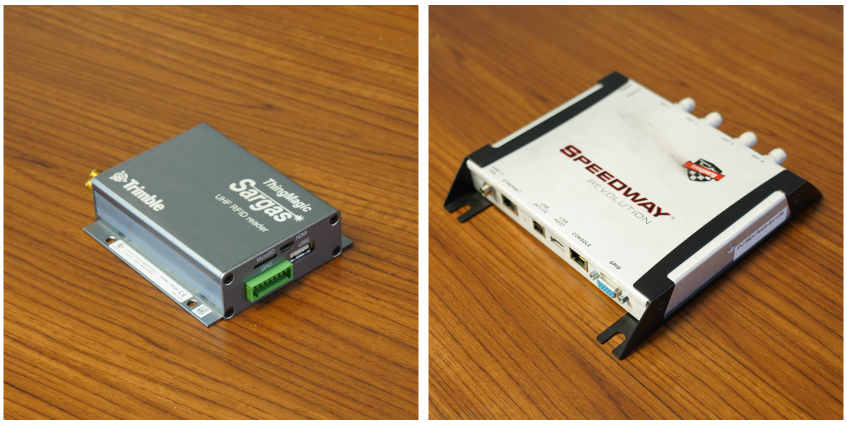


Figure 2.9: RFID Tag Readers.

2.1.6 Ethernet Switch

As described in Sec. 2.1.5 an additional Ethernet port was required. The simplest solution was to add an Ethernet Switch and assign static IP addresses from the same subnet to each device. That created a local area network on our robot. The only challenge was that the Ethernet Switch required a 9 V power supply. To cope with this issue, the Ethernet Switch was disassembled, to find out if the internal IC chips could handle a 12 V power supply. That proved to be true according to the manufacturer's data-sheets. So this device could be powered directly from Kobuki's Base 12 V port.

Figure 2.10: Utilized Ethernet Switch (TP-Link TL-SG1005D).

2.1.7 Power Distribution

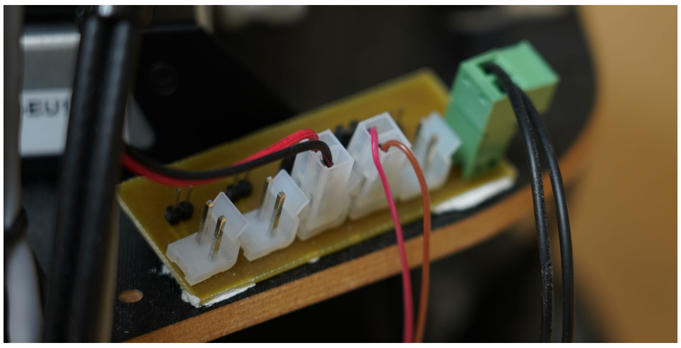
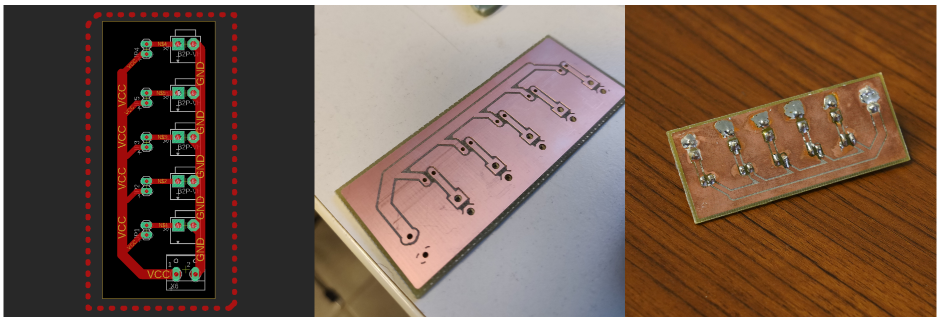


Figure 2.11: Power Distribution Board in action.

Turtlebot platform is designed to include a variety of additional devices and modules. Even from the early stages of development, it was known that we were going to install 3-4 devices on the robot. Almost all of them require some kind of power supply which in most cases is 12 V. It is necessary for all those devices to get power safely, with the proper voltage level and not get damaged from some kind of human error, for example, reverse polarity or over-voltage. In addition, it must be simple to add or remove devices. Furthermore, the capability of disabling a device without disconnecting it must be included. There might be some cases where a device is not required for a particular experiment. This could increase the battery life of the robot as well as the overall life span of the device.



Eagle Design. Bare PCB. PCB Solder Joints.

Figure 2.12

Taking all the above into account, a Power Distribution Board was designed and created. It included all the features for safety, expand-ability and ease-of-use. Eagle CAD was used to create a PCB which was then CNC milled, using an LPKF ProtoMat S103[2]. Then all the required components were soldered. The board offers a single power inlet, which is connected to the Kobuki's Base 12 V/5 A port using a custom made connector (Fig. 2.14a) and 5 power outlets individually controllable through jumper blocks. Then custom cables were made for each device. Some of our devices are rated for a different voltage level than 12 V. In these cases DC-to-DC converters were attached to their cables, so they can be powered properly. This is shown in Figures 2.14b and 2.14c.

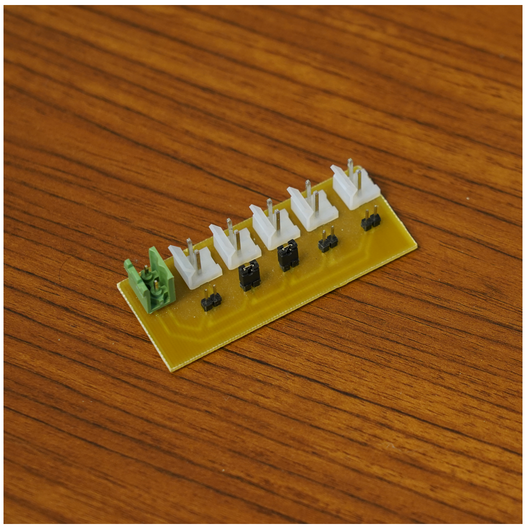


Figure 2.13 : Power Distribution Board Assembled.

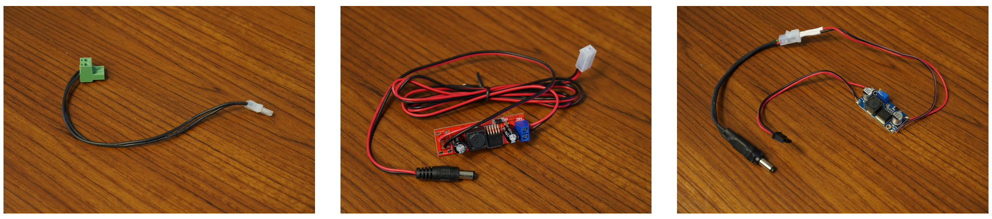


Figure 2.14 : Cable Examples

[2]with the help of George Vougioukas.

2.1.8 Computing Unit

All devices described in Sec. 2.1 are slave devices and need some kind of master device which communicates with each of them, collects all necessary data (e.g. laser scans or depth maps), processes them and then calculates all the required actions. Such a device could be a Single Board Computer, a mini PC, a laptop or any kind of device with sufficient computing power and the ability to run Linux OS. Our Turtlebot offered an Intel NUC with a 2.2 GHz Intel i5 processor, 8 GB of RAM and a 120 GB Solid State Disk for storage. In our experiments, we did not use this Mini PC because of power supply issues. Instead, we used a Dell Latitude laptop with similar specifications. All the required software was installed and the laptop was placed on the dedicated position on the Turtlebot Platform. The laptop was operated via a Remote Desktop Application.

2.1.9 Other

Additional modifications and addons are listed below. These modifications were made to increase our system's stability and functionality.

2.1.9.1 Joystick Controller

When it was necessary to manually move the robot, we wanted to have the ability to control it remotely. The only way to achieve this out of the box was with the help of computer keyboard arrows. This proved to be difficult because it was necessary to use a second PC as well due to the lack of analog control. Taking all of these into account, we built a remote controller using a 2 axis analog joystick and an ESP32 microcontroller power by a 5 V USB power bank. Micro-controller included WiFi capabilities so it was easy to send UDP packages and then received them on the Computing Unit. A picture of the controller described above is shown in Figure 2.15.

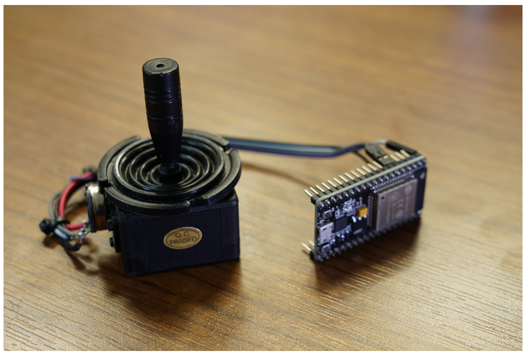


Figure 2.15 : ESP32 based Turtlebot Controller.

2.1.9.2 Antenna Mount

A FlexiRay SF-2110 5 $\mathrm{dB_i}$ antenna had to be mounted on the robot for the RFID tag reader to work. A wooden stick was mounted on the robot and the FlexiRay antenna was placed on top of it (see Fig. 2.16). Wood was chosen to reduce possible reflections of RF signals transmitted and received by the antenna.

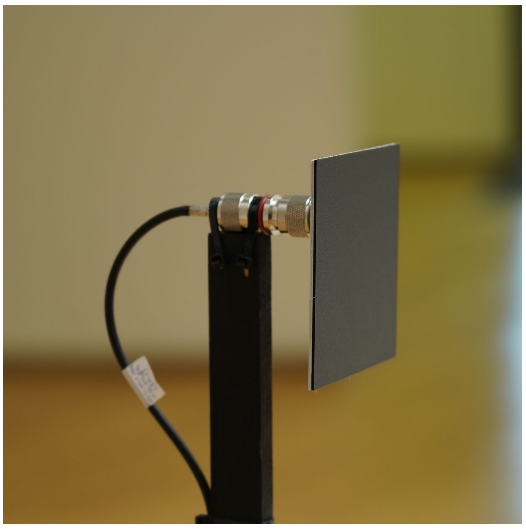


Figure 2.16 : FlexiRay SF-2110.

2.1.9.3 Pole Modification

Turtlebot's Platform Plate structure is held together with 12 mm aluminum rods. Those rods are threaded together with M4 bolts, which quickly failed. The robot was disassembled and those rods were modified to support M6 bolts. This modification was done in Micro-machining and Manufacturing Modeling Lab, School of Production Engineering And Management, Technical University of Crete.

2.1.10 Robot Overview

After installing all necessary devices, cable management was done to ensure that no cable was loose and all connections were made reliably. The final robot structure is presented in the following figures.

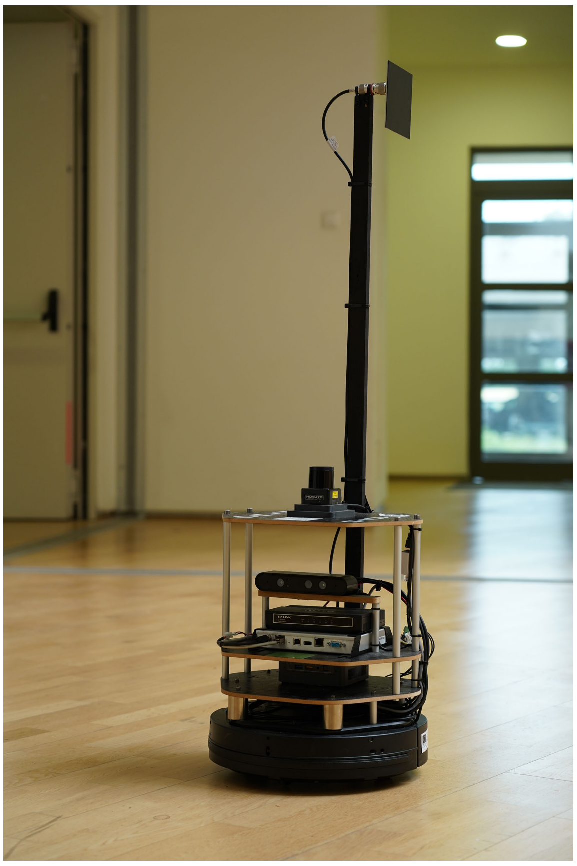


Figure 2.17 : Final robot structure used in our experiments.

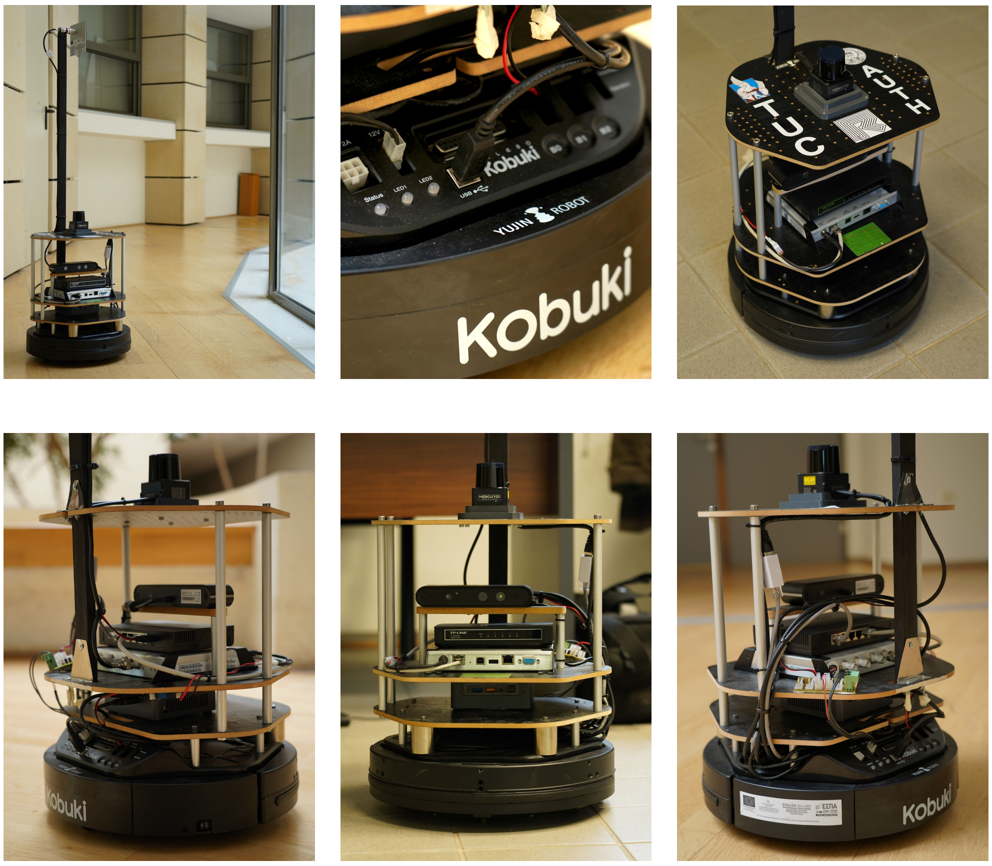


Figure 2.18 : Some photographs around the robot.

2.2 ROS

Robots are usually complex systems containing many subsystems. To accomplice the desired tasks, all these systems have to communicate so they can exchange information and calculate the next action. Furthermore, it must be simple and fast to install new modules in the system. All those problems are solved with the help of the Robot Operating System (ROS) [6].

The code in ROS is organized into Nodes. Nodes run as individual processes so a system can include many Nodes. Typically a Node performs a single task, for example, a Node is launched to communicate with Hokuyo LiDAR and publish laser scan data to a dedicated topic. Nodes communicate through topics and services. Topics are used for message exchange and a node can publish or subscribe to many topics. The same applies to topics, many nodes can publish or subscribe to the same topic. Messages contain data in a structured way. Services provide a way to call functions from other nodes. Manufacturers of almost every device designed for robotic use, provide the appropriate ROS package. This is very important because we don't have to write our drivers for each device and we only have to learn to use a single API.

In addition, ROS provides many useful tools for data visualization, data logging and playback, robot simulators and many more. ROS was initially developed by Stanford University and got released in 2007. All of its features made it the standard middleware for writing robot software.

2.2.1 Rviz

Rviz is a powerful visualization tool included in ROS. Its Graphical User Interface is straightforward and easy-to-use. It can visualize laser scans (Fig. 2.5), point clouds (Fig. 2.8c), maps, robot models and any other data type which can be visualized. Its main window provides a 3D viewer, a 2D viewer is available in a separate window and is used to visualize images like a depth map (Fig. 2.8b). Furthermore, it provides some basic robot navigation functionality like sending navigation goals. An example of a navigation viewer is shown in Figure 2.19.

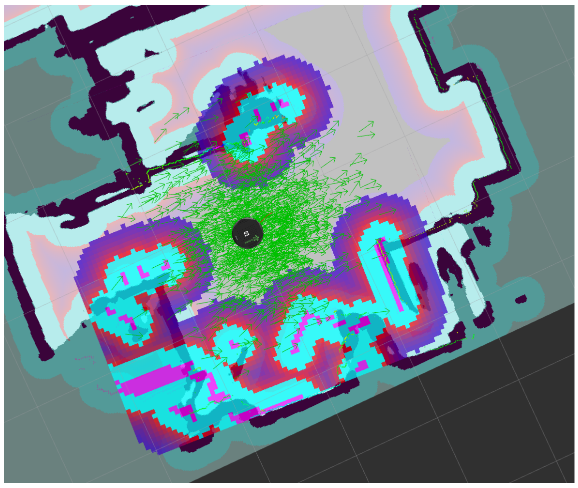


Figure 2.19 : Rviz Navigation.

2.2.2 TF

Another useful tool provided by ROS is the TF library. This library organizes the coordinate frames of a system in a tree like structure and provides transformations from one coordinate frame to another. Every device on our robot, which performs some kind of measurement, has its coordinate frame. We adjusted those frames position in the dedicated files. For example, the depth camera's default position is on the rear position of the middle plate and we mounted it on the middle position. Also, we installed an antenna that had to be included in the TF tree. So we defined a static transformation from our robot frame to the antenna frame. An example of a TF tree can be seen in Figure 2.20.

2.2.3 Log Synchronization

There were some compiler issues interfering with APIs of ROS and Impinj RFID Reader. The problem was that the free version of Impinj API could only work with GCC v4.8, while ROS required v5. As a result, we couldn't compile a node that could publish ROS messages to a topic containing information about the RFID Tags read by the Reader. This was an issue because the requirement was to log the tag's information as well as the exact location

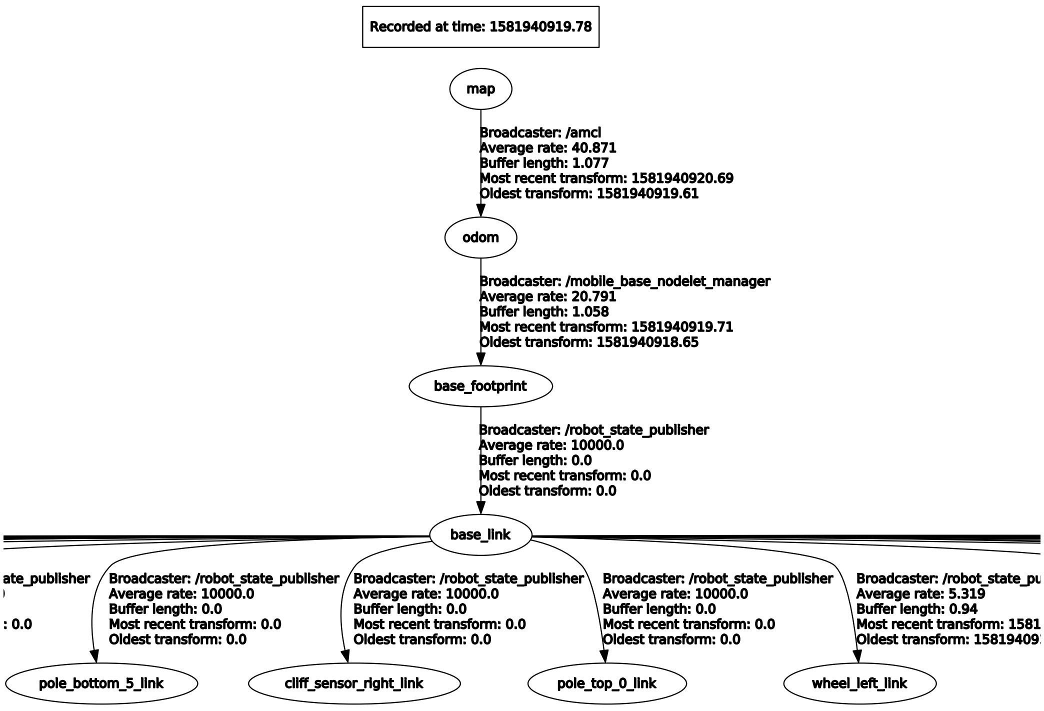

Recorded at time: 1581940919.78
map
Broadcaster: /amcl
Average rate: 40.871
Buffer length: 1.077
Most recent transform: 1581940920.69
Oldest transform: 1581940919.61
odom
Broadcaster: /mobile_base_nodelet_manager
Average rate: 20.791
Buffer length: 1.058
Most recent transform: 1581940919.71
Oldest transform: 1581940918.65
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
base_link
ate_publisher
: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
pole_bottom_5_link
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
cliff_sensor_right_link
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
pole_top_0_link
Broadcaster: /robot_state_pu
Average rate: 5.319
Buffer length: 0.94
Most recent transform: 1581
Oldest transform: 15819409
wheel_left_link

Figure 2.20 : TF tree.

of the robot calculated by AMCL, when the tag was read. Named FIFOs and sub-processes were tested as a solution but they introduce a huge synchronization error because of the Operating System's delays. Finally, the solution was found with the help of time-stamps. Two executables were created, the first one logged the robot's pose measurements and the second one the Tags read measurements. Each measurement contained a time-stamp and logs were saved into separate files. After the execution of the loggers, a simple script can open these files, merge them according to time-stamps and then save the merged data which can then be used as an input to RFID tag Localization Algorithms.

Chapter 3

Robot Localization Algorithms

One of the fundamental problems for a robot is to determine its pose in an unknown environment as well as simultaneously map that environment. This problem is referred as simultaneous localization and mapping (or SLAM). The only data available are those of action control u_t and measurement data z_t. Examples of control actions are robot movement and object manipulation. Measurement data include sensor measurements (e.g. laser scans) at a specific point in time. The 2D pose x_t of a robot is modeled as,

$$x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}, \tag{3.1}$$

where x, y are 2D coordinates and θ represents robot's heading. Robot's translation from one pose to another can be modeled as,

$$bel(x_t) = p(x_t \mid x_{t-1}, u_t), \tag{3.2}$$

where x_t represents the current pose and x_{t-1} the previous. Further analysis of the aforementioned model can tell that when the robot takes an action u_t the result is a transition from x_{t-1} to x_t. Similarly, measurement data take the form:

$$bel(z_t) = p(z_t \mid x_t). \tag{3.3}$$

Probability distributions are used because action control and sensor measurements are governed by probabilistic laws. Simply put, sensor measurements introduce noise and action control can't be precise. The probabilistic model (3.2) can be reffered as *motion model* while (3.3) as *measurement model.*

A map of the environment m is a representation of objects in the environment. Maps are very useful for path planning as well as obstacle avoidance. Thus, the robot must estimate a map so that the robot can revisit locations if that is required or avoid them. There are many map representations but the most common is the occupancy grid map, where space is divided into cells with a specific resolution (e.g. 0.05 m/cell edge) and each cell represents the probability of occupancy. An example of an occupancy grid map can be seen in Figure 3.1, where white color represents free space, black cells are obstacles and grey are unknown areas.

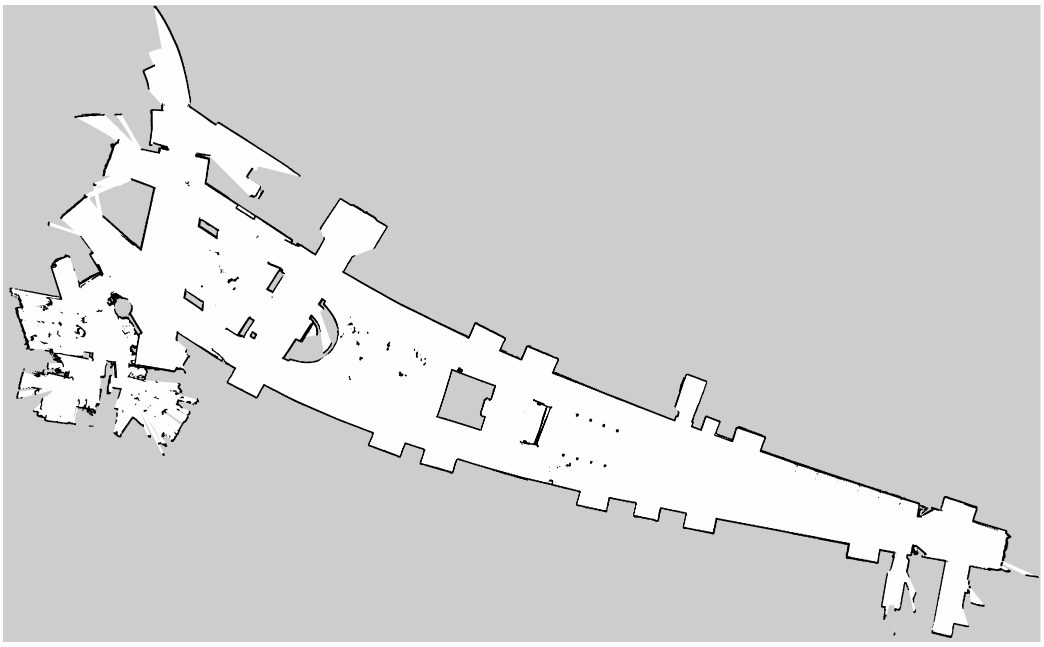


Figure 3.1 : Level 0, East Wing, Science Building, TUC.

The SLAM can be solved with two main approaches. One is known as Full-SLAM in which the complete path ($x_{1:t}$) of the robot as well as the map m are estimated:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}). \tag{3.4}$$

The second approach is called Online-SLAM where just the current pose is estimated:

$$p(x_t, m \mid z_{1:t}, u_{1:t}). \tag{3.5}$$

Both approaches are equally important and there is extensive literature that addresses them. In our experiments we used Google Cartographer SLAM [7] approach because it is fairly new and it produces acceptable results.

3.1 Google Cartographer

Google Cartographer was released in 2016 as an open source project [7]. It provides a real-time simultaneous localization and mapping algorithm in 2D and 3D space. Cartographer comes with Robot Operation System (see Chap. 2.2) support and ready to use packages for popular robotic platforms like the Turtlebot. Among its more important features is the capability to close loops over large trajectories. Loop closing is defined as the problem of revisiting a location. In this case, the robot must be able to identify that location in the estimated map and not insert it as a new. This proved to be a legit claim and the Loop Closure capabilities can be seen in a real example from our experiments in Figure 5.3. In many cases, due to measurement error accumulation, this is not possible and the other approaches would fail to produce an accurate map.

Cartographer divides the SLAM problem into a Local SLAM and a Global SLAM. In their local SLAM approach, laser scans $\{h_k\}$ are matched against a small portion of the map. Those portions are called submaps $\{s_i\}$. Scan points are modeled in the following form:

$$H = \{h_k\}_{k=1,2...K}, h_k \in \mathbb{R}^2. \tag{3.6}$$

Then a transformation $T_x h_k$ of the scan points from the scan's frame to the submap frame is define as:

$$T_x h_k = \begin{pmatrix} \cos x_\theta & -\sin x_\theta \\ \sin x_\theta & cos x_\theta \end{pmatrix} h_k + \begin{pmatrix} x_x \\ x_y \end{pmatrix}, \tag{3.7}$$

where x is the pose of the scan frame in the submap frame. Scan Matching is achieved using a non-linear optimization process. The optimization process seeks a transformation T_x that aligns the pose of the laser scan x frame in the submap frame. This is achieved using a *Ceres* based [8] scanner matcher formulated in a nonlinear least squares problem:

$$\underset{x}{\operatorname{argmin}} \sum_{k=1}^{K} (1 - M_{smooth}(T_x h_k))^2, \tag{3.8}$$

where function $M_{smooth} : \mathbb{R}^2 \to \mathbb{R}$ is a smooth version of the probability values in the local submap s_i which is calculated using bi-cubic interpolation. This smooth version is used because usually gives a better precision than the resolution of the map.

The local SLAM approach slowly accumulates error which is acceptable only for a few dozen scans. So, many submaps are created and the Global SLAM approach tries to align all submaps and scans poses to estimate an accurate map m of the unknown environment. This optimization process follows sparse pose adjustment (SPA) [9] and like scan matching, it is cast as a nonlinear least squares problem. This process runs every few seconds and *Ceres* is used to calculate a solution to:

$$\underset{X^s, X^h}{\operatorname{argmin}} \frac{1}{2} \sum_{i,j} \rho(E^2(x_i^s, x_j^h, \Sigma_{i,j}, x_{i,j})), \tag{3.9}$$

where $X^s = \{x_i^s\}_{i=1,2\ldots n}$ are submap poses and $X^h = \{x_j^h\}_{j=1,2\ldots\xi}$ are scan poses which are optimized given some constraints. These constraints take the form of relative poses $x_{i,j}$ and associated covariance matrices $\Sigma_{i,j}$. The pose $x_{i,j}$ describes where in the submap s_i the scan was matched. The term ρ represents a loss function and it is used to reduce the influence of outliers. Then a *branch-and-bound scan matching* approach is executed and seeks for the optimal, pixel accurate match:

$$x^\star = \underset{x \in W}{\operatorname{argmax}} \sum_{k=1}^{K} M_{\text{nearest}}(T_x h_k), \tag{3.10}$$

where W is the search window size and M_{nearest} an extended version of the submap. Search window is divided into steps, angular δ_θ step size is chosen based on the maximum range $d_{\max}$ of each scan and the resolution r of the map:

$$d_{\max} = \max_{k=1,..,K} \|h_k\|, \tag{3.11}$$

$$\delta_\theta = \arccos(1 - \frac{r^2}{2d_{\max}^2}). \tag{3.12}$$

Then each step is defined as:

$$w_x = \left\lceil \frac{W_x}{r} \right\rceil, w_y = \left\lceil \frac{W_y}{r} \right\rceil, w_\theta = \left\lceil \frac{W_\theta}{d_{\max}} \right\rceil. \tag{3.13}$$

Finally the pose $x^\star$ can be efficiently calculated using a Branch-and-bound approach. A more detailed description of the overall approach can be found in [7].

3.2 Localization - AMCL

In many situations, a robot is placed in a known environment where it performs the required task. For example, a robot could be placed in a library in order to find misplaced books. In this case, the map is given so it is not necessary to estimate it. This scenario is referred to as the Mobile robot localization problem where only the pose of the robot needs to be estimated. Robot Localization problems are distinguished into three types according to [10](see Chap. 7.2):

- **Position Tracking**, where initial pose is known.
- **Global Localization**, where initial pose is unknown.
- **Kidnapped Robot Problem**, where robot can be kidnapped and teleported to some other location.

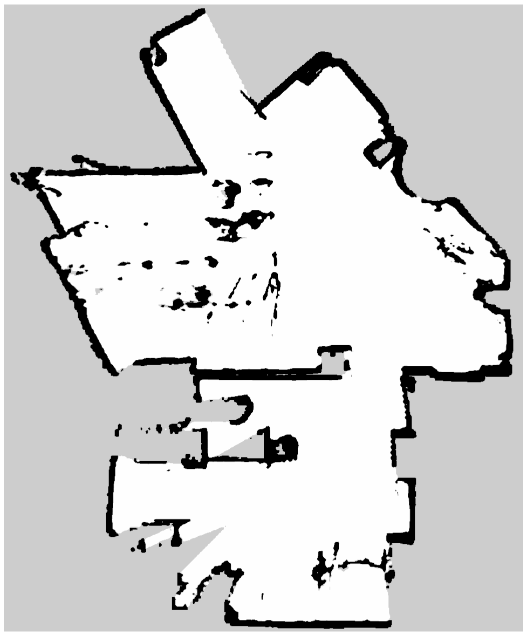


Figure 3.2 : An example of a map used for robot localization.

A variety of algorithms are available trying to solve the robot localization problem. The most popular of them is the *Monte Carlo Localization* (MCL), which takes advantage of particle filters to estimate the robot's pose. Algorithm 1 shows the basic MCL functionality. A set of M Particles $\mathcal{X}_t = \{x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}\}$ represents the belief $bel(x_t)$. The initial belief is calculated by uniformly spreading particles across the map. Line 4 of the algorithm samples from the motion model, using particles from present belief as starting points. Then measurements are used to calculate the importance weight of each particle (Line 5). Initial weight is set to $\frac{1}{M}$ for all particles. Lines 7-9 are the resampling step. Particles with stronger weights are more likely to be chosen for the next round. This lead to a new set of particles concentrated to more likely locations of the true pose.

Monte Carlo Localization can be modified to solve more challenging localization problems like Kidnapped Robot Problem. Those variations are further described in [10](Chapter 8.3). A popular approach is the Adaptive Monte Carlo Localization (AMCL) which is based on KLD sampling [11]. The basic idea of this algorithm is to dynamically change the number of

Algorithm 1 MCL

```
 1: procedure MCL($\mathcal{X}_{t-1}, u_t, z_t, m$)
 2:     $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
 3:     for $m = 1$ to $M$ do
 4:         $x_t^{[m]} =$ sample motion model$(u_t, x_{t-1}^{[m]})$
 5:         $w_t^{[m]} =$ measurement model$(z_t, x_t^{[m]})$
 6:         $\bar{\mathcal{X}}_t = \mathcal{X} + \langle x_t^{[m]}, w_t^{[m]} \rangle$
 7:     for $m = 1$ to $M$ do
 8:         draw $i$ with probability $\propto w_t^{[i]}$
 9:         add $x_t^{[i]}$ to $\mathcal{X}_t$
10:     return $\mathcal{X}_t$
```

particles based on a statistical approach. In our experiments, we used an implementation of this algorithm. The implementation is provided through a ROS package [12]. The package is included as a part of Navigation Stack. Navigation Stack contains packages for localization, path planning, obstacle avoidance, cost map creation and action planning. An example of navigation visualized by rviz can be seen in Figure 2.19. An example of AMCL initial particle spread can be seen in Figure 3.3.

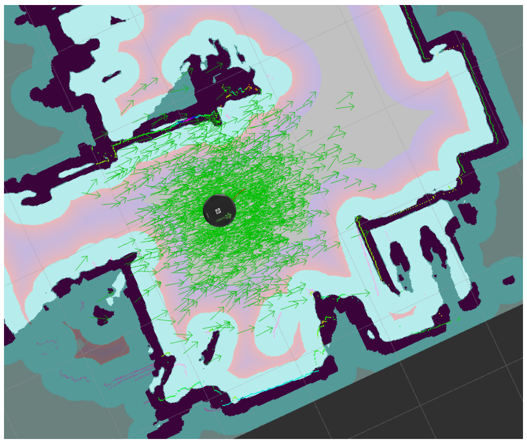


Figure 3.3 : Particle cloud visualized.

Chapter 4

RFID Tag Localization Algorithms

Radio Frequency Identification or RFID is a wireless, non-contact, battery-less method of data transfer. An RFID tag contains an antenna and an IC chip. Its tiny size factor makes it possible to be integrated into cards, key-chains or even stickers. RFID technology can be used for access control, inventorying, object tracking etc. There 3 main RFID bands: Low Frequency (120 kHz to 150 kHz), High Frequency (13.56 MHz) and Ultra High Frequency (865 MHz to 868 MHz Europe, 902 MHz to 928 MHz USA). We are interested in UHF RFID Tags. RFID Tags are passive and are powered from the energy of the electromagnetic wave transmitted from the Reader. They do not transmit some kind of signal, they just reflect the incoming signal in such a way that they encode the necessary data. Each tag is identified by a unique code called the Electronic Product Code (EPC). This is achieved with antenna load switching and it is based on the backscatter effect.

Because of the distance between Reader and tag, a phase offset is introduced in the received signal. This phase offset can be modeled as follows:

$$\phi_R - 2\pi f_c \tau = \phi_R - 2\frac{2\pi d_0}{\lambda} = \phi_R - 2kd_0, \tag{4.1}$$

where $\tau = 2\frac{d_0}{c} = 2\frac{d_0}{\lambda f_c}$, c is the speed of light, f_c is the carrier frequency, ϕ_R is the carrier phase, d_0 is the Euclidean distance between Reader and tag which is multiplied by 2 because signal travels from reader-to-tag and back and $k = \frac{2\pi}{\lambda}$ is the angular wavenumber. The reader calculates that offset for each read and report it in 2π intervals. This measurement θ_{meas} also include phase offset due to cabling ϕ_{cable}, phase noise ϕ_{noise} from reader's receive chain, a constant offset θ_{tag} introduced by tag electronics and phase θ_{refl} caused by the effect of the RF signal reflections (multipath). So each phase measurement can be modeled as follows:

$$\theta_{\text{meas}} = \overbrace{\phi_R - 2kd_0 + \underbrace{\theta_{\text{tag}} + \phi_{\text{cable}}}_{\theta_t}}^{\theta_i} + \underbrace{\phi_{\text{noise}} + \phi_{\text{refl}}}_{\phi_{\text{var}}} . \tag{4.2}$$

In this thesis ϕ_{var} is omitted, so we assume that $\theta_{\text{meas}} = \theta_i$. Then the equation $\phi_i(x_t, y_t, \theta_t)$ is defined as:

$$\begin{aligned} \phi_i(x_t, y_t, \theta_t) &= \left(\frac{2\pi}{\lambda} 2d_i + \theta_t\right) \mod 2\pi \\ &= \left(\frac{4\pi}{\lambda}\sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} + \theta_t\right) \mod 2\pi, \end{aligned} \tag{4.3}$$

where $i = 1, ..., N$, (x_i, y_i) are reader's coordinates and (x_t, y_t) are tag's coordinates. Reader's coordinates are known from Robot Localization. Consequently the only unknowns that need estimation are tag's coordinates and constant phase offset.

4.1 Phase Relock

An interesting approach for Phase based RFID Tag Localization is proposed in [3]. The proposed method is trying to minimize the following function:

$$\begin{aligned} F(x_t, y_t, \theta_t) &= \sum_{i=1}^{N} [\phi_{it}(x_t, y_t, \theta_t) - \theta_i]^2 \\ &= \sum_{i=1}^{N} \Bigg[\Bigg(\left(\frac{4\pi}{\lambda}\sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} + \theta_t\right) \mod 2\pi - \theta_i\Bigg) \\ &\quad \mod 2\pi\Bigg]^2, \end{aligned} \tag{4.4}$$

where θ_i is the measured phase. This problem is treated as an nonlinear optimization problem. The global minimum of $F(x_t, y_t, \theta_t)$ are the estimated tag coordinates (x_t, y_y) and the constant phase offset θ_t. Measured phase samples contain discontinuities because of the mod function. These discontinuities introduce many local minimums to $F(x_t, y_t, \theta_t)$ and the optimization would fail to find the global one. To overcome this issue, a phase unwrapping method is applied to eliminate those discontinuities. Samples are divided into groups that need to be shifted vertically by $k \cdot 2\pi$; $k \in \mathbb{Z}$, k is calculated so that a continuous curve is produced. This method is modeled as follows:

$$\hat{\Theta}_j = \Theta_j + k \cdot 2\pi, j = 1, ..., m \ , \tag{4.5}$$

where Θ_j are phase sample groups. An example of this process can be seen in Figure 4.1.

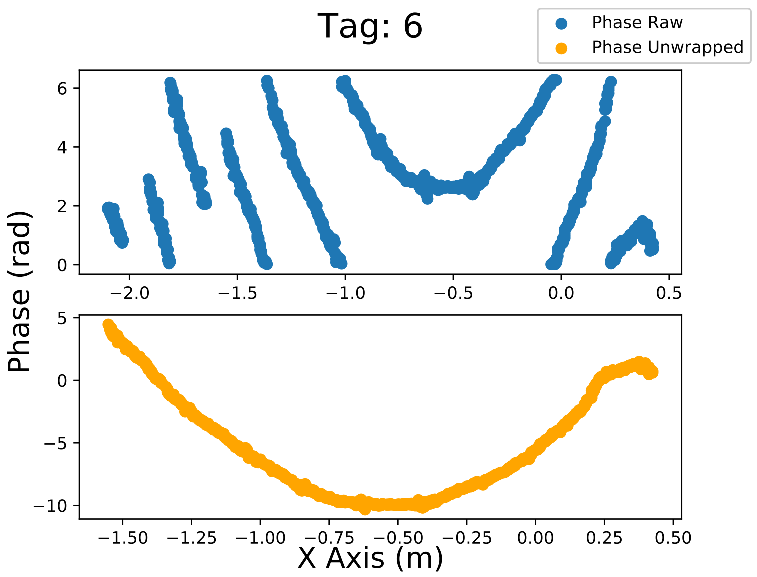


Figure 4.1 : Phase Unwrap Example.

Now $F(x_t, y_t, \theta_t)$ is simplified to:

$$\begin{aligned}\hat{F}(x_t, y_t, \theta_t) &= \sum_{i=1}^{N} [\phi_i(x_t, y_t, \theta_t) - \hat{\theta}_i]^2 \\ &= \sum_{i=1}^{N} \left[\left(\frac{4\pi}{\lambda} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} + \theta_t \right) - \hat{\theta_{it}} \right]^2,\end{aligned} \tag{4.6}$$

and it is ready for further processing.

The optimization process that we used is Gradient Descent and it is based on derivative observation. An initial point of ϵ_1 is randomly chosen. Then next ϵ (ϵ_{n+1}) is calculated from the following equation:

$$\epsilon_{n+1} = \epsilon_n - \alpha \nabla \hat{F}(\epsilon_n), \tag{4.7}$$

where $\alpha \in \mathbb{R}_+$ represents a small number (e.g. 0.0000001) and ϵ is defined as:

$$\epsilon_n = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} \tag{4.8}$$

This calculation is repeated over a finite amount of times. When ϵ does not significantly change, calculation is stopped and ϵ is our estimate. Algorithm 2 provides a brief overview of the optimization process. Partial derivatives of $\hat{F}(x_t, y_t, \theta_t)$ were calculated using Python's library Sympy [13]. Another useful library for our implementation was Scipy [14].

4.2 Particle Filter Based

4.2.1 PF1

Another approach taking advantage of phase measurements for RFID Tag Localization is by using particle filters [4] [5]. Antenna's trajectory and orientation are known from Robot Localization (Chap. 3) and TF tree (Chap.

Algorithm 2 Gradient Descent

```
1: procedure GD($\Theta, X$)
2:     phase unwrap
3:     $\epsilon = \text{random}(X)$
4:     for $n = 0$ to $N$ do
5:         $x_t = x_t - \alpha \frac{\partial F(x_t, y_t, \theta_t)}{\partial x_t}$
6:         $y_t = y_t - \alpha \frac{\partial F(x_t, y_t, \theta_t)}{\partial y_t}$
7:         $\theta_t = \theta_t - \alpha \frac{\partial F(x_t, y_t, \theta_t)}{\partial \theta_t}$
8:         if $x_t, y_t, \theta_t$ does not significantly change then
9:             break
10:    return $x_t, y_t, \theta_t$
```

2.2.2). Particles are defined as follows:

$$p^{[m]} = \begin{pmatrix} x^{[m]} \\ y^{[m]} \\ \theta^{[m]} \\ w^{[m]} \end{pmatrix}, \quad m = 1, 2, ..., M, \tag{4.9}$$

where $(x^{[m]}, y^{[m]})$ are particle's coordinates, $\theta^{[m]}$ is the phase offset a tag would introduce and $w^{[m]}$ is particle's weight. Particles are spread uniformly on the 2D plane, around some point based on robot's trajectory and antenna's orientation. For each particle, the constant phase offset $\theta^{[m]}$ is uniformly assigned and initial weight is set to $w^{[m]} = 1$. On every round, Euclidean distance $d^{[m]}$ between each particle and robot is calculated:

$$d^{[m]} = \sqrt{(x^{[m]} - x_i)^2 + (y^{[m]} - y_i)^2}. \tag{4.10}$$

Then we test whether a phase measurement for a tag at the particle's location could introduce that offset. Phase measurement θ_i is converted into distance δ_0 using the following formula:

$$\delta_0 = \frac{\lambda \theta_i}{4\pi}. \tag{4.11}$$

Taking into account that $\theta_i \in [0, 2\pi)$, $\delta_0 \in [0, \frac{\lambda}{2})$ can be deduced. Then the calculated distance d_m distance can be model as follows:

$$d^{[m]} = \delta^{[m]} + n\frac{\lambda}{2}, n \in \mathbb{N}, \tag{4.12}$$

$$\delta^{[m]} = d^{[m]} \bmod \frac{\lambda}{2}. \tag{4.13}$$

Then we set:

$$\delta_{\rho 1} = \max(\delta^{[m]}, \delta_0), \ \ \delta_{\rho 2} = \min(\delta^{[m]}, \delta_0). \tag{4.14}$$

Particles that are more likely to introduce the measured phase offset are these with the minimum distance Δ where:

$$\Delta^{[m]} = \min\left(\delta_{\rho 1} - \delta_{\rho 2}, \frac{\lambda}{2} - (\delta_{\rho 1} - \delta_{\rho 2})\right). \tag{4.15}$$

This formula ensures that the minimum distance is always calculated even in cases where $\delta_{\rho 1}, \delta_{\rho 1}$ are near 0 or $\frac{\lambda}{2}$. In Figure 4.2 an example is shown.

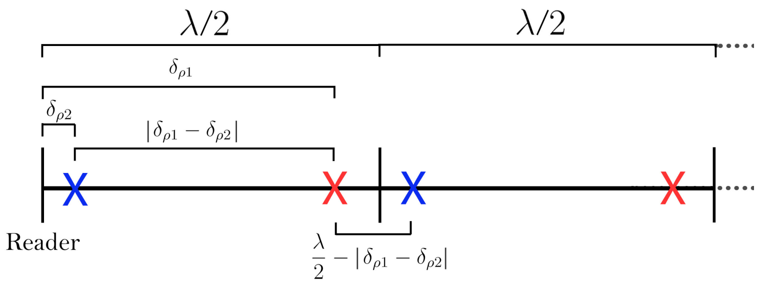

$\lambda/2$
$\lambda/2$
$\delta_{\rho 1}$
$\delta_{\rho 2}$
$|\delta_{\rho 1} - \delta_{\rho 2}|$
Reader
$\frac{\lambda}{2} - |\delta_{\rho 1} - \delta_{\rho 2}|$

Figure 4.2 : Distances visualized.

Then a Gaussian Distribution $\mathcal{N}(\Delta^{[m]}, \mu, \sigma^2)$ is used to update the particles' weight:

$$\mathcal{N}(\Delta^{[m]}, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\Delta^{[m]} - \mu\right)^2 / 2\sigma^2} \tag{4.16}$$

where μ is the distribution's mean value and σ is the standard deviation. So the new weight $\hat{w}^{[m]}$ will be:

$$\hat{w}^{[m]} = w^{[m]} \mathcal{N}(\Delta^{[m]}, 0, \sigma^2). \tag{4.17}$$

Finally the RFID Tag's Location estimate is calculated as a weighted mean:

$$x_t = \frac{\sum_{m=1}^{M} w^{[m]} x^{[m]}}{\sum_{m=1}^{M} w^{[m]}}, \quad y_t = \frac{\sum_{m=1}^{M} w^{[m]} y^{[m]}}{\sum_{m=1}^{M} w^{[m]}}. \tag{4.18}$$

Algorithm 3 implements the particle filter described above. This version of the algorithm is called PF1.

4.2.2 PF2

Constant phase offset of each particle is subtracted from phase measurement. This way that offset can be also estimated. An improved version of PF1 ignores the estimation of θ_t and sets $\theta^{[m]}$ to a constant value $\theta_c \in [0, 2\pi)$. Then the estimation algorithm is executed K times and the final location estimation is the mean of all estimations. In every execution, θ_c is set to a different value according to the following formula:

$$\theta_c = k \left\lceil \frac{2\pi}{K} \right\rceil, k = 0, ..., K-1. \tag{4.19}$$

So algorithm's 3 line 6 can be changed to $\theta^{[m]} = \theta_c$. Because of smaller search space, only x_t, y_t need estimation, the number of particles M can be reduced. Carefully choosing K, M, total execution time can be improved.

Algorithm 3 PF1

```
1: **procedure** PF1($\Theta, X_r$)
2:     **calculate spread point P**
3:     **for** $m = 1$ **to** $M$ **do**
4:         $x^{[m]} = \mathcal{U}(P \pm r)$
5:         $y^{[m]} = \mathcal{U}(P \pm r)$
6:         $\theta^{[m]} = \mathcal{U}[0, 2\pi)$
7:         $w^{[m]} = 1$
8:     **for** $n = 1$ **to** $N$ **do**                    ▷ N is number of measurements
9:         **for** $m = 1$ **to** $M$ **do**
10:            $\delta_0 = \frac{\lambda(\theta_n - \theta^{[m]}) \mod 2\pi}{4\pi}$
11:
12:            $d^{[m]} = \sqrt{(x^{[m]} - x_n)^2 + (y^{[m]} - y_n)^2}$
13:            $\delta^{[m]} = d^{[m]} \mod \frac{\lambda}{2}$
14:            $\delta_{\rho 1} = \max(\delta^{[m]}, \delta_0)$
15:            $\delta_{\rho 1} = \min(\delta^{[m]}, \delta_0)$
16:            $\Delta^{[m]} = \min\left(\delta^{[m]} - \delta_0, \frac{\lambda}{2} - (\delta^{[m]} - \delta_0)\right)$
17:            $w^{[m]} = w^{[m]} \mathcal{N}(\Delta^{[m]}, 0, \sigma^2)$
18:     $x_t, y_t$ = **calculate weighted mean**
19:     **return** $x_t, y_t$
```

Chapter 5

Experimental Results

5.1 Robot Localization

To evaluate the functionality of the complete system as well as the accuracy of implemented algorithms, real world experiments were conducted. First of all, robot localization had to be tested, so the creation of maps was necessary.

All required ROS nodes were launched and we drove our robot around so it could map the environment. In our first experiments, only our lab was mapped (Fig. 3.2). Map results were acceptable, so we proceeded to larger scale mapping like Science's Building East Wing, Level 0 (Fig. 3.1 or West Wing and Core Level 1 (Fig. 5.1). In Figure 5.3 loop closing optimization can be seen. Indeed the claim for large trajectory loop closing was confirmed.

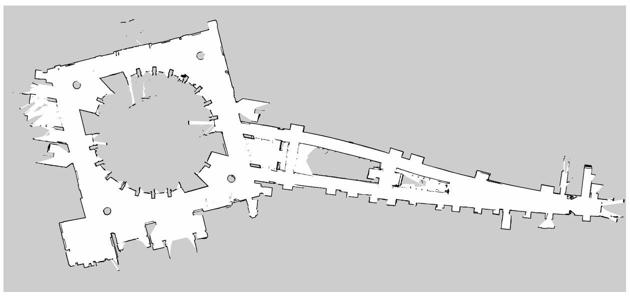


Figure 5.1 : Science's Building West Wing and Core Level 1 Map.

Maps were saved using a special version of *map server* package, created to support Cartographer. From the experiments above we acquired maps useful for our next experiments and the capabilities of the selected SLAM algorithm got explored. In Figure 5.2 map creation snapshots can be found.

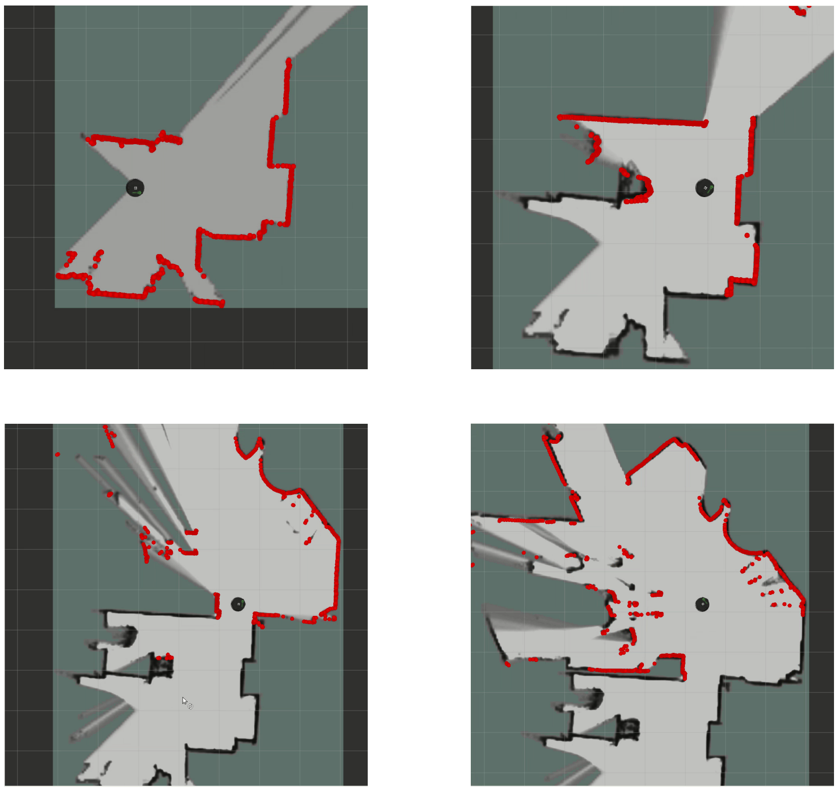


Figure 5.2 : Map creation process.

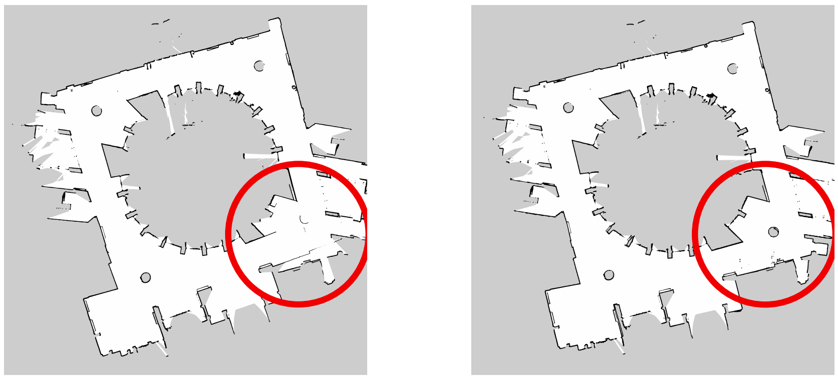


Figure 5.3 : Loop Closure Example.

Afterward, AMCL and Navigation Stack got tested. Modification to parameters were made to improve the performance as well as to include all of the installed sensors. For example, the default Turtlebot AMCL package converts depth images to laser scans because LiDAR is an optional module. Localization using depth images does not provide the desired results because of the smaller field of view, so the dedicated files were changed so that LiDAR's node is launched and used for AMCL. Depth information was used only for obstacle avoidance. Rviz was very useful in those experiments because it visualizes all navigation stack information and makes troubleshooting very simple. An example of a navigation goal action can be seen in Figure 5.4. Red arrows represent AMCL particles. It is worth mentioning that the initial pose estimate is spread over a large area but as the robot moves, that estimate is corrected and quickly converges around the true pose.

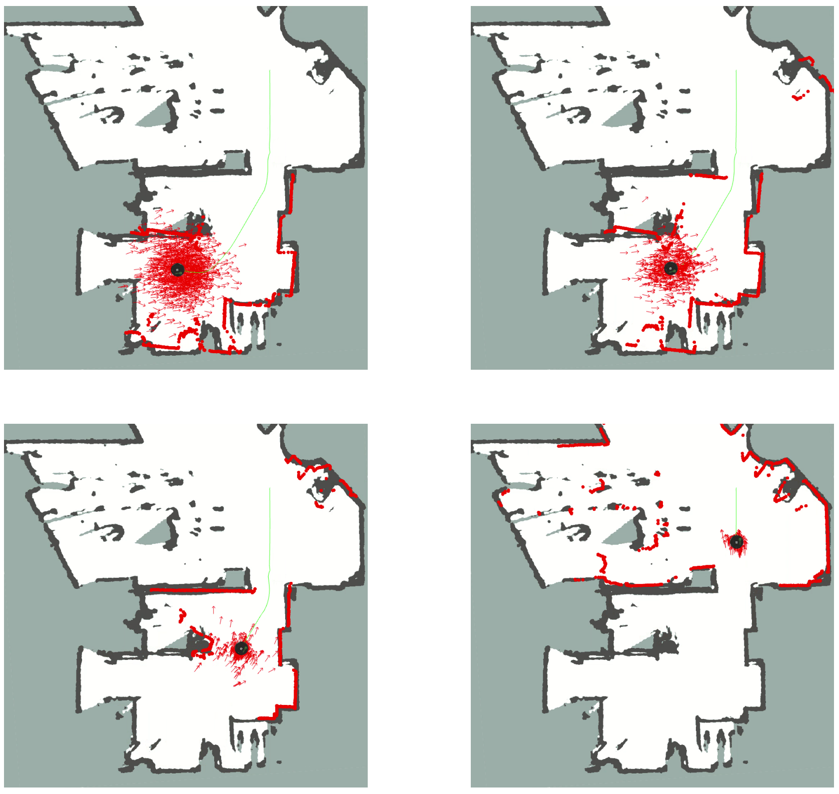


Figure 5.4 : Robot navigation example.

At this point mapping and localization provided us with acceptable results so the next step was to evaluate the RFID Tag Localization Algorithms. But before that Log Synchronization had to be tested. As described in Chap.

2.2.3 initial attempts failed but our proposed method based on timestamp synchronization produced the desired results. Some minor modifications were made to the Pose Logger's log rate so it could match the RFID Reader's read rate. The result of the synchronization was a file containing the following information for each tag interrogation:

- **x**, **y** coordinates of the antenna location, when the tag was read,
- **RSSI**,
- **Phase**,
- **EPC**.

5.2 RFID Tag Localization

The scenario for RFID tag localization follows: 15 books were arranged in a bookshelf spaced approximately 5.5 cm apart. Then, Alien ALN9740 (Higgs-4) RFID tags were attached to the books in such a way that the center of the tag was at the same height as the reader's antenna (1.1 m). To find each tag's true location, a measuring tape was used. Reader's transmit power is set to 30 dB_m (maximum). The map is created beforehand and AMCL provides robot's pose estimate.

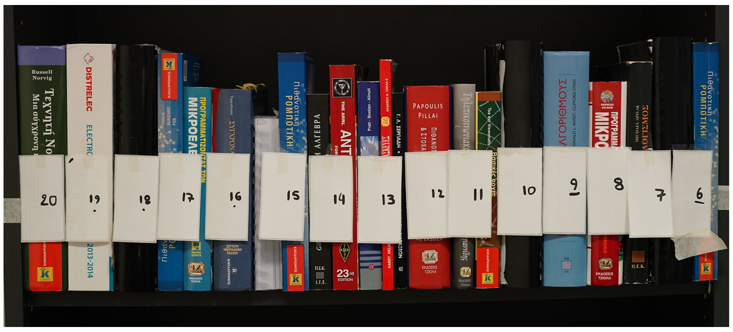


Figure 5.5 : Book arragement overview.

Our robot moves on a straight line approximately 1 m away from tags, with its antenna facing the tags. Robot moves with a translational velocity of 10 cm/s along a 3 m trajectory.

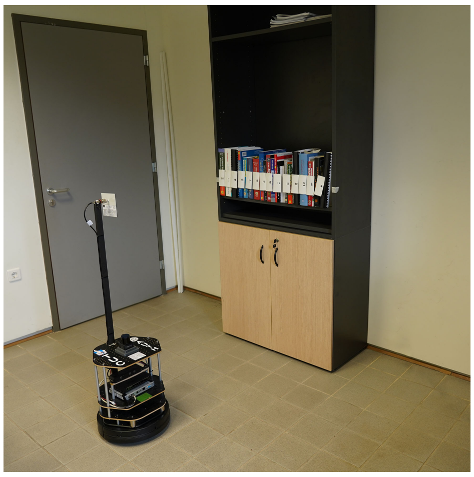


Figure 5.6 : Scenario overview.

Loggers are executed and when the trajectory is finished, the synchronization script produces the log file described above. Finally, Tag Localization Algorithms are executed for each tag and 1D, 2D errors are calculated using Euclidean distance. For PF1 and PF2, the number of particles M is set to 10^5 and 10^4 respectively. The experimental results are presented in Table 5.1.

Table 5.1 : Experimental errors (meters) across all tags.

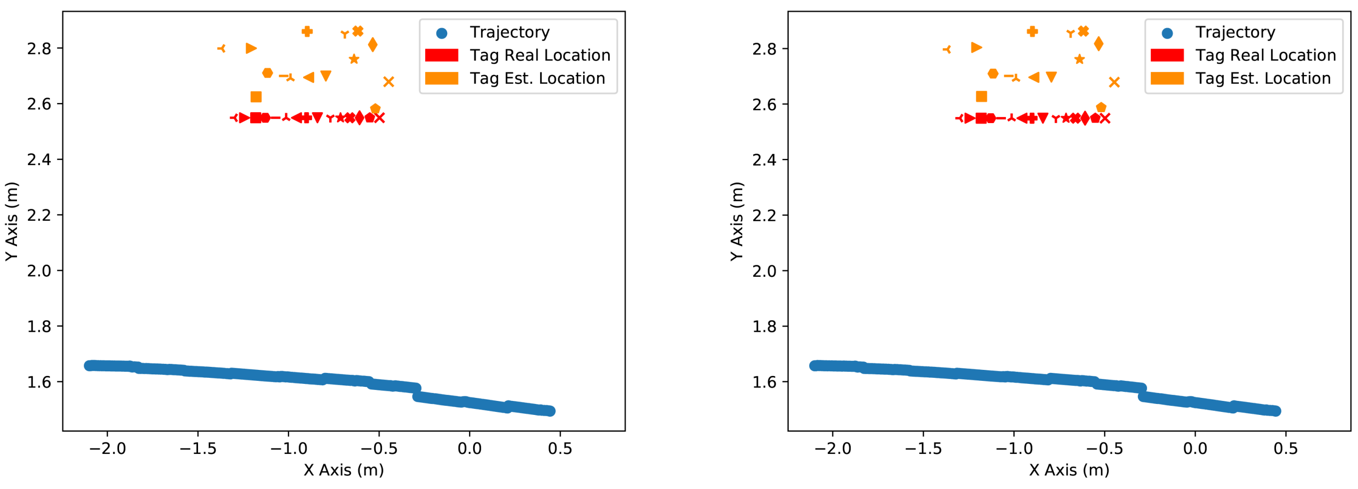
	Relock		**PF1**		**PF2**	
TagID	**1D error**	**2D error**	**1D error**	**2D error**	**1D error**	**2D error**
6	0.059	0.256	0.051	0.137	0.0516	0.1392
7	0.163	0.326	0.031	0.045	0.0332	0.0496
8	0.095	0.427	0.075	0.281	0.0756	0.278
9	0.042	0.436	0.044	0.314	0.0436	0.3168
10	0.066	0.207	0.075	0.223	0.0748	0.2256
11	0.078	0.312	0.082	0.319	0.08	0.3176
12	0.037	0.175	0.047	0.159	0.0484	0.1548
13	0.006	0.340	0.003	0.308	0.0008	0.3132
14	0.009	0.178	0.067	0.157	0.0656	0.1612
15	0.014	0.157	0.023	0.147	0.0236	0.1444
16	0.034	0.159	0.043	0.157	0.042	0.158
17	0.004	0.083	0.013	0.158	0.0124	0.1608
18	0.031	0.070	0.003	0.077	0.002	0.0788
19	0.030	0.230	0.037	0.258	0.0376	0.2572
20	0.096	0.389	0.069	0.253	0.0684	0.2576

In this experiment 14 506 measurements were logged and processed (approximately 937 logs-per-tag). Mean values of errors and execution time are shown in Table 5.2. A visualization of estimated-true locations for each algorithm, are shown in figures 5.7a, 5.7b, 5.8 (tag IDs increase from right to left).

Table 5.2 : Experimental mean values and standard deviation.

	Relock	**PF1**	**PF2**
Mean 1D error	0.05 m	0.04 m	0.04 m
STD 1D error	0.043 m	0.025 m	0.025 m
Mean 2D error	0.25 m	0.2 m	0.2 m
STD 2D error	0.114 m	0.084 m	0.083 m
Mean Ex. Time	5.6 s	18.6 s	1.70 s

Taking a look at mean error values, it is clear that all methods are similarly accurate. Particle filter methods provide better results. The reason PF1 is slower than PF2 is that we use more particles. This is essential because, in PF1, more unknowns are estimated, so more particle combinations are needed. The lack of importance re-sampling also contributes because particles are not renewed. Improved results can be obtained if more parameters are taken into account, for example, the estimation of phase noise. Phase's Relock execution time is slower than expected because our Python script implementation is not optimized. Single axis error is smaller than tag-to-tag distance so tag order can be retrieved. This proves to be true in almost every case if we observe Figures 5.7a, 5.7b, 5.8. In all three algorithms, 13 tags' order is calculated correctly. Another worth mentioning fact is that all the estimations are located behind the true location. From this, we can assume that the measured phase does not correspond only to the direct path of the signal but also to possible reflections coming from objects in the environment (e.g. walls). Furthermore, if we take a closer look at the robot's logged trajectory, some inconsistencies can be noticed. These "jumps" are present due to floor tiling. This causes some z-axis movement when the robot's wheels fall into tiling grooves. Finally taking into account execution times suggest and robot's velocity all implementations were real time.



PF1. PF2.

Figure 5.7 : Particle filters location estimation.

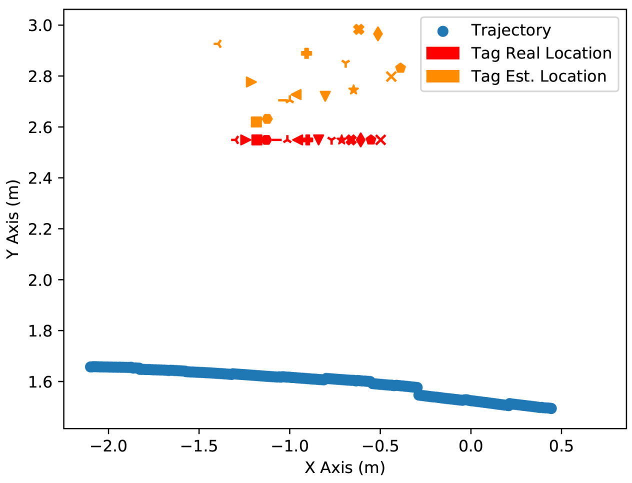


Figure 5.8 : Phase Relock location estimation.

Chapter 6

Conclusions

6.1 Conclusion

The purpose of this work was to combine RFID tags with robotics. This was accomplished and a mobile system was created, capable of autonomously navigating on a known map. During navigation RFID tags can be read and then using the implemented RFID tag localization algorithms, their location can be accurately estimated. Experimental results demonstrate that satisfactory estimation can be obtained, taking into account that some parameters were ignored (e.g. phase noise). This is visible if we compare true and estimated locations. All estimations are located further away from true locations so we can assume that the complete phase model is correct and further examination is needed. The importance of middlewares, like ROS, was appreciated, because of all the advantages frameworks like these can provide. Finally, all the implemented hardware modifications ensure that the robot is flexible and ready for any kind of experiment.

6.2 Future Work

This work exploited only a few of the possible capabilities of a system like this. There are a lot of possible improvements to both hardware and software.

From the hardware's perspective, some of them refereed below. First of all multiple antennas can be installed, so tags can be read in more directions. A multicopter can be used so tags can be localized at more heights (e.g. huge warehouses shelves). A software-defined radio can be installed on the robot, so more detailed data can be obtained. Furthermore, a single board computer can be used instead of a laptop. This will make it easier to start an experiment because until now, the laptop has to be placed on the robot every time it is required. Lastly, a smaller and cheaper robot can be created, so we can deploy multiple of them in an environment and program them to collaborate.

Regarding software improvements, a major one could be to include more noise parameters into the modeled phase. This will probably reduce the error because as it is suspected the multipath component of the phase measurement is a significant term. Multipath estimation could be possible by map examination. Also, 3D RFID localization can be performed with some modifications to our implementations. This means that we would localize tags at a height different from that of the robot's antenna. Another major improvement could be to implement importance resampling and particle position variations based on the robot's movement in PF algorithms. Then even fewer particles will be needed and their population could be dynamically changed. Finally, we can use a single particle set for all tags instead of a set-per-tag.

Bibliography

[1] J. Wang and D. Katabi, “Dude, where’s my card? rfid positioning that works with multipath and non-line of sight,” in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM ’13, Aug 2013, p. 51–62.

[2] Z. Luo, Q. Zhang, Y. Ma, M. Singh, and F. Adib, “3D backscatter localization for fine-grained robotics,” in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, Boston, MA, Feb. 2019, pp. 765–782.

[3] A. Tzitzis, S. Megalou, S. Siachalou, T. Yioultsis, A. Kehagias, E. Tsardoulias, A. Filotheou, A. Symeonidis, L. Petrou, and A. G. Dimitriou, “Phase ReLock - Localization of RFID tags by a moving robot,” in *Proc. IEEE Europ. Conf. on Antennas and Propagation (EuCAP)*, Krakow, Poland, Mar. 2019, pp. 1–5.

[4] E. Giannelos, K. Skyvalakis, M. Andrianakis, A. G. Dimitriou, and A. Bletsas, “Robust RFID Localization in Multipath with Phase-Based Particle Filtering and a Mobile Robot,” submitted to *IEEE Int.Conf. on RFID*, Orlando, USA, April 2020.

[5] E. Giannelos, “Intelligent Wireless Networks and Robots for Low-Cost Battery-less Sensing and Localization,” Ph.D dissertation, School of ECE, TUC, in preparation, Feb. 2020.

[6] Stanford Artificial Intelligence Laboratory et al., “Robotic operating system.” [Online]. Available: https://www.ros.org

[7] W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2D LIDAR SLAM,” in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016, pp. 1271–1278.

[8] S. Agarwal, K. Mierle, and Others, “Ceres solver.” [Online]. Available: http://ceres-solver.org

[9] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, “Efficient sparse pose adjustment for 2D mapping,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Taipei, Taiwan, Oct 2010, pp. 22–29.

[10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[11] D. Fox, “KLD-sampling: Adaptive particle filters,” in *Proc. Int. Conf. on Neural Information Processing Systems: Natural and Synthetic (NIPS)*, Vancouver, Canada, Oct. 2001, pp. 713–720.

[12] Brian P. Gerkey, “AMCL ROS package.” [Online]. Available: http://wiki.ros.org/amcl

[13] A. Meurer, C. Smith, M. Paprocki, O. Čertík, S. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. Moore, S. Singh, T. Rathnayake, S. Vig, B. Granger, R. Muller, F. Bonazzi, H. Gupta, F. Johansson, F. Pedregosa, and A. Scopatz, “Sympy: Symbolic computing in python,” *PeerJ Computer Science*, vol. 3, p. e103, 01 2017.

[14] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, 2020.