Digital Twin of an Indoor Navigation System using TurtleBot3 WafflePi

Bansal, Saharsh

Award date:
2021

Department of Mathematics and Computer Science
Software Engineering and Technology

Digital Twin of an Indoor Navigation System using TurtleBot3 WafflePi

Saharsh Bansal
1423819

Supervisor:
dr. ir. Ion Barosan
dr. ir. Eugene Lepelaars (TNO)

Eindhoven, August, 2021

Abstract

This project aims at developing a system centered around a CPS device, which is able to direct the robot to specified locations in a known indoor environment. The concerned region is mapped using a LIDAR and the scanned map is then used for identifying locations as well as to plan the path. A multitude of path planning algorithms are used to plan the path and a comparison between them is performed to find which algorithm would work best in which scenario. Once the path is calculated the physical device is directed to its location along the path. Additionally the project also focuses on mapping the magnetic field of the concerned region. To do so the magnetic field is modelled using Gaussian process regression. The entire project is developed using a MDSE approach. This approach is selected because of the advantages it poses in terms of identifying the system requirements and developing each aspect of the system individually and comprehensively. Using this systematic approach also helps in identifying any contradictions or pitfalls early in the development cycle. Additionally using this approach allows developing each service as an individual model, which can be translated to future/other systems. Doing so allows the project to be scaled up as a whole or even individual components of it. It was achieved by implementing aspects of it in the ongoing projects at the TruckLab Automotive Lab at TU/e.

List of Abbreviations

amcl Adaptive Monte Carlo Localization.

BDD Block Decision Diagram.

BFS Breadth First Search.

CN Digital Twin Communication Network.

CPS Cyber-Physical Systems.

DD Digital Twin Digital Data.

DT Digital Twin.

GP Gaussian Process.

IMU Inertial Measurement Unit.

LIDAR Light Detection and Ranging.

MDSE Model Driven System Engineering.

PE Digital Twin Physical Entity.

SBC Single-Board Computer.

Ss Digital Twin Services.

TB3 TurtleBot3 WafflePi.

TBT TurtleBot Truck.

TCP Transmission Control Protocol.

TRIZ Theory of Inventive Problem Solving.

VE Digital Twin Virtual Entity.

Contents

List of Figures

List of Tables

1 Introduction

Over the past decade since its introduction, cyber physical systems(CPS) have had a significant contribution in various industries. They play key roles in automation, manufacturing and robotics. As a result, they have applications in many fields of research including health-care, robotics etc. This report focuses on the development of a system for an indoor navigation system. In this section the motivation for this project, its overall scope and use cases will be focused upon.

1.1 Digital Twin(DT) and CPS

In general a CPS is one which includes engineered networks between physical and computational devices [17]. Using a CPS device has various advantages such as distributed computing, interconnected and integrated systems. A Digital Twin (DT), while being a relatively new concept, first introduced in 2006, refers to a system which has both a physical and virtual component. In this system it is widely accepted that the purpose of the virtual component is to replicate the actions of the physical component. As can be seen from the generally accepted meanings of the two aforementioned concepts, there is some overlap between the understanding of what a CPS device and a DT is. The inter-connectivity of a CPS device can be considered to be equivalent to the link formed between the physical and virtual components of a DT. Whereas, the distributed computing ability of the CPS, can be the equivalent of any external services added into the DT. This is further explained in Section 2. It can therefore be assumed that a DT is a type of CPS device in itself, as is done in this project.

Given the features mentioned above, a DT system can act as the perfect medium for developing, executing and testing new systems, as it can replicate the physical motions onto a virtual environment. Doing so, helps in visualizing the actions of the physical device remotely. It also allows emulating the actions of the physical device entirely on a virtual platform. This allows for sped up simulations and behaviour monitoring. While, the concept of a DT has expanded over the years, there is still no industry/academic standard. The model used in this project will be explained in detail in Chapter 2, and its relevance to the targeted goals.

However, taking into consideration even the rudimentary understanding of what a DT is, it can be seen that it would be a suitable platform to develop and test a navigation system designed for indoor environments. A navigation system would require the implementation of various algorithms to calculate an appropriate path and then testing to validate its accuracy.

1.2 Problem Definition and Goal

Navigation has held a special place in human history over the centuries. From the times of the colonial explorers discovering new lands, to finding the path to the nearest supermarket. A key aspect in navigation and location is a pre-requisite knowledge of the concerned environment. The environment may be either internal, an enclosed region such as a building, or even external, intra/inter-city. This can be done by constructing a map of the area. However, for the purposes of navigation, the map need not necessarily be restricted to geographic landmarks, but could be about various features such as magnetic fields, radio signatures, heat maps etc. as shown by Brena et al. [8]. Performing the survey operation for generating these maps is a tedious and time consuming process, as it requires a physical walk-through of the concerned region and collecting data. However, this procedure can be automated. It requires a self exploring CPS device with the ability to store and process data.
These CPS devices enable unguided exploration and data collection possibilities in unexplored environments. They can also be expanded to path planning including collision avoidance. This has several advantages.

1. Acting as a safety precaution by allowing an alternate mapping option, thereby protecting humans from potentially dangerous situations.

2. Instant mapping and access of environment map.

3. Consideration of other environmental parameters, such as heat signatures, geo-magnetic fields thereby providing multiple maps reflecting various conditions.

For example, a single device could be configured in a way that it can at the same time map not only the geographical map of an area but also characterise other features such as temperature, pressure, radiation etc. It further helps illustrate the importance of autonomous robotic mapping as, if humans were to perform the same task various safety considerations would restrict access to potentially harmful environments.
Outdoor mapping has been tackled extensively and is now possible with satellite detection, Li and Zhijian [15], Malarvizhi et al. [16]. However, the challenge lies in indoor mapping, along with GNSS(Global Navigation Satellite System) denied regions. Much work has been done to tackle the indoor problem with using WiFi signals. However, it is susceptible to interference from external factors. Especially in a shared public space such as a university or a library where there may be multiple such networks added or removed at any given time. This necessitates the need for mapping using a relatively stable source of information, which is not as susceptible to change. A potential source of mapping information is the geo-magnetic field of the region as explored by Brena et al. [8], Yeh et al. [24]

Introduction to magnetic field mapping

Identifying the magnetic field of a region is not a completely new concept. Birds have been using it for centuries to navigate while migrating. Bats also use it for characterizing their environment. There have been concerns in the past about the effect of magnetic field on human health and various studies have been done regarding it as shown in Kheifets et al. [13], Stein and Udasin [20]. These papers mainly focus on the effects of induced magnetic fields due to proximity to high power electric lines, and specify the limits of exposure to humans in various places where these lines are present. Various governments across the EU including the Netherlands have limitations on the acceptable levels of magnetic field, especially in regions where children are exposed the most[18]. This provides another reason for mapping the magnetic field of an indoor region, other than to use it for navigation. The field map can be used to identify magnetic hot-spots and therefore be used to ensure that the magnetic field is always below the prescribed levels.
The next concern would be regarding the issue of information stability over time. As shown in Yeh et al. [24], Gozick et al. [12], Angermann et al. [7] the geo-magnetic field in a region remains fairly constant over extended periods of time. It may be argued that the presence of ferromagnetic materials in the building such as iron bars or steel beams and such may have an effect on the geo-magnetic field in that space. However, once construction is done, it is fairly improbable that there would be much deviation in the magnetic signature of the region. A potential case when there may be significant changes is when there are renovations/alterations made to the structure of the building. In that case, a simple rerun of the mapping procedure would be in order to get the new data. Doing this would be a simple enough task for an automated robot. This further elaborates the advantage of having an automated bot do this task rather than have a human perform it.

1.3 Scope of the project

Up till now the need for a system for indoor navigation has been described. Additionally, a potential new source for information for generation of the map, magnetic fields was introduced. The project aims at developing an indoor navigation system which can also map the magnetic field of a given region. Setting up such a system can be further divided into developing two working systems.

- The first part aims to develop a proper navigation system in an indoor setting. It encompasses a CPS device which is able to accurately navigate from a given source location to a destination in the shortest available path.

- The second part of the solution is to accurately map the magnetic field of the given region.

These parts will be tackled separately before merging them to present a final solution to this project. Generating this map will help identify location based on the magnetic signature of specific landmarks, which is the underlying target of this project. However, identifying the specific magnetic signatures requires extensive knowledge in magneto-statics and falls outside the scope of this project. Therefore in this project the magnetic map generated can be used to direct the CPS device to location of highest/lowest magnetic field values. This project can therefore act as a stepping stone to future research into identifying locations by their magnetic signatures. It can also be used to test a technology other than magnetic fields to act as the driving force behind the generation of the map. That map and its feasibility as a viable location identifier source can be tested using the navigation system being developed in this project.

1.4 Research Questions

The system being developed in this project aims at answering particular questions concerning both indoor magnetic field mapping and indoor navigation. They are:

- RQ1 How to map magnetic fields of a region on a low resource device?

  Aims at evaluating the feasibility of deploying existing magnetic field mapping solutions on devices with limited RAM availability such as the Raspberry Pi 3b+.

- RQ2 Which navigation algorithm is optimal in an indoor environment?

  Aims at comparing the various path finding algorithms implemented on the following criteria

  - SRQ2_1 Which algorithm has the least mean error in directing to target location?
  - SRQ2_2 Which algorithm yields the path in shortest amount of time?
  - SRQ2_3 Which algorithm utilizes the least resources to find the solution?

- RQ3 How feasible is it to use both magnetic fields and LIDAR for indoor navigation?

  Aims at trying to establish the reasonableness of using both the mentioned sensors as the basis for navigation indoors.

1.5 System Context

Having defined the scope of this project it is now important to discuss the overall method which will be used in the implementation of the various aspects of this project. Given the various facets of this project a structured approach will be required which properly defines each task and is able to merge all of them together. For this reason a model driven system engineering(MDSE) approach will be taken. It helps in properly defining the individual components of the project.
The system being developed in this project has various features being implemented simultaneously. This therefore necessitates the development of a singular controller which is able to oversee and control all the required features. Developing such a system and deploying it directly in the physical environment runs a few risks, of damages due to unpredicted consequences of implemented solutions. It is therefore wise to first test the developed system onto a virtual platform replicating the behaviour of the actual physical system. As a result the system developed in this project will be on a digital twin.

1.5.1 Digital Twin using MDSE approach

MDSE is able to develop extremely complex systems with relative ease. Each of the services being tackled in the project are represented as individual models which interact with the necessary components or each other if required. This helps in compartmentalizing tasks and tackling them individually. Thereby making the development cycle relatively smooth. It is therefore an ideal approach to proceed with the development of the digital twin in this project.
Developing the digital twin of a system allows for a virtual platform, where multiple possible theories/algorithms can be tested without expending any physical resources. Additionally since its on a virtual environment the simulation can be sped up and thus verification can be expedited. For this project, should an accurate virtual replica of the PE be made on the VE, the entire area can be mapped without having to do any physical testing. This also allows for various routing algorithms to be tested. Executing a model based system will also give the shortcomings of the tested system without risking any damage to the physical devices.
It also provides a platform to test the response time of the actual designed system by, connecting it to the simulation environment and check any possible implementation virtually before being implemented physically. This provides a environment for ensuring that implemented systems work as intended. This not only saves time as the simulated environment can be sped up, but also saves on potential physical damage to any of the concerned devices. All the reasons explained above further reaffirm the choice of developing the digital twin for this project.

This section covers the overall target of this project, while simultaneously explaining the base concepts which will be employed in its development. However, some base information specific to topics tackled in this project is required before the system can be developed. These topics will be elaborated in detail in the following section. Once a basic understanding of the base concepts is established, the report focuses on the implementation of the DT. The results of the the implementation of the system and its services are then visualized and evaluated on various parameters. Finally the proposed research questions are answered based on the implementation and findings made during the project, before presenting the final conclusions on the project and suggestions on any potential future work concerning it.

2 Background Information

Before starting the development of the project, basic information about various concepts utilized in this project needs to be explored in further detail. In this section a brief explanation of the DT concept is presented along with some background information about magnetic field mapping. In addition to it, the tools used to develop the system are also presented along with the rationale to do the same are also explored.

2.1 Digital Twin model

A DT typically has no fixed definition but various concepts, have been commonly accepted by researchers and academics in the field. The most widespread concept is the three dimensional model, as explained in Wu et al. [23]. It contains a virtual replica of the physical environment, the physical devices and a communication network between the two components. However, this architecture does not account for any external services which can be incorporated into the DT system. A different architecture is the four dimensional model, which includes the digital data, the virtual replica, an additional model which is the predictive twin and the decision making twin. This model ignores the effects and needs of the physical entity. It is best suited to evaluate a purely virtual system which would be useful for simulation based testing in the initial stages of product development. Due, to the lack of the physical device in the architecture, it is not best suited to this project which focuses on incorporating a physical robot and collecting data from sensors interacting with the physical environment. Wu et al. [23] also discusses a six dimensional model which has the following components: 1) physical data source, 2) local data repository, 3) data information transformation level, 4) cloud-based information repository, 5) emulation and 6) simulation module. While this model covers the issue of the previous one by incorporating the physical entity and accounts for data received from the physical sensors, it does not discuss the interactions between the various components. It is an important part of this project as there is constant data transfer between the various components especially between the services offered and the physical device.

Accepted Model

The final model considered is a five dimensional model as described in Wu et al. [23]. This model consists of the Physical Entity(PE), the Virtual Entity(VE), the Services(Ss), Digital Data(DD) and the Communication Network(CN). It is better represented in Figure 1. This model better considers the requirements of this project and allows for a proper conceptual understanding of the various aspects of the system. It is the model which will be used for this project. Figure 1a shows how the components of the system of this project fits into the different dimensions of the DT.

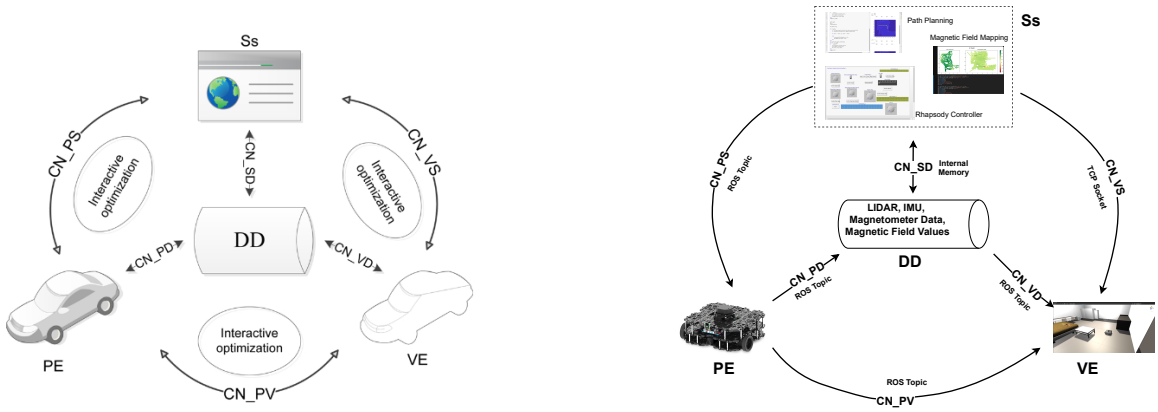
PE: Encompasses the physical TB3, the sensors taking readings from the environment, namely the LIDAR, the IMU and the RM3100 magnetometer.

VE: Consists of the virtual replication of the TB3, its movement and the virtual visualization of the predicted path.

Ss: Includes indoor mapping, path planning and magnetic field extrapolation. A final service offered is a MBSE based system controller which controls not just the TB3 itself, but also the other offered services.

DD: Accounts for the data read from the sensors listed previously as well as the results of the services offered, namely the extrapolated magnetic fields and the results of the path planning service.

CN: The communication network is achieved using a distributed ROS network over a WiFi connection.



(a) Conceptual [23] **(b)** DT Framework for project

Figure 1: Five-Dimensional Framework for DT

A visual representation of these components in the framework format can be seen in Figure 1b. The Ss is generally associated with additional services used for analysis or functional services which assist the DT itself. However for this project the scope of the Ss is expanded to allow for control services of the PE and the VE. This is to allow for distributed, parallel control of the services offered. In addition to this doing so also allows for a system which could easily be expanded to allow for cloud computing and thereby allow for complex controls and services be incorporated to this project.

2.2 Magnetic Field Mapping

While forming the map of an indoor region there are 2 potential magnetic field sources that we can consider. The first is source is the geo-magnetic field. The second source can be artificially induced by physically placing (electro)magnets in strategic location to induce a magnetic field, which will be used to form a map. The disadvantage of introducing external magnets is that we are physically changing the magnetic field of the region. Should the strength of the field be high it may start adversely affecting human health [6]. There have additionally been multiple studies covering the effects on static magnetic fields on humans, Ueno [21], Kheifets et al. [13]. However, no conclusive correlation between magnetic fields and human health has been established for now. Despite this there are restrictions on the levels of acceptable magnetic fields in different regions, such as industrial factories, proximity to power stations etc. [10]. Taking all these factors into consideration, the mapping of induced magnetic fields falls outside the scope of this project. We will therefore be focusing on mapping the geomagnetic field.

A key requirement when developing a navigation system based on geomagnetic field is that the magnetic field should remain stable across large periods of time. Gozick et al. [12] show that the magnetic field in a region remains fairly stable across a period of nearly a year. The slight variations in data collected over time can be attributed to human error as they have mentioned. Hence it can be reasonably assumed that barring any major changes to the region or changes made to the structure of the building/massive renovations involving ferromagnetic metals, the magnetic field should remain reasonably constant over time.

To get the most accurate readings to form the magnetic map, an ideal implementation would be to take readings from each cubic unit of the given region. However, as this method is not feasible in vast regions, this necessitates the requirement of some form of interpolation of readings taken over some region. A proposed method to do so has been shown by Solin et al. [19] using Gaussian Process Regression. A few methods have been proposed in this paper. One method illustrates capturing the magnetic field readings in all 3 axes along a random path. Using Gaussian Process regression this data is then used to predict the magnetic field along the unidentified regions.

2.3 Path Finding Algorithms

All the path finding algorithms used in this project are based on graph theory[14]. In this context a graph refers to a set of vertices(V), which are connected to each other via edges(E). Using various algorithms, it is possible to find a route from a source vertex(V_s) to a destination vertex(V_d), as a list of intermediate nodes. The edges could have certain weights associated with it which signifies the cost of using it. The following path finding algorithms are used in the implementation of this project:

BFS Algorithm

Breadth First Search(BFS), is a graph search algorithm which searches all adjacent nodes from the V_s, till it reaches V_d. The order in which the nodes are searched can be seen in Figure 2.

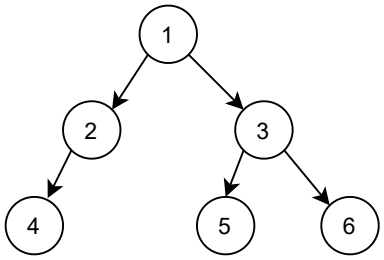


Figure 2: BFS exploration sequence of graph

BFS, assumes that all the edges have the same weight.

Dijkstra Algorithm

Dijkstra algorithm works similar to the BFS algorithm in that it keeps exploring all unvisited neighbours of a vertex, till the V_d is found. It differs from BFS in the process of exploring the next vertices. Once a vertex is reached the next vertex is chosen based on the shortest distance from the current set of visited vertices.

A* Algorithm

A* works in a similar way to Dijkstra, but instead of simply trying to minimize the cost to the current node, it associates a cost function which accounts for the cost to reach the destination from the current node. This cost is calculated by means of a heuristic function. The heuristic function is problem specific and the complexity of A* varies with the choice of heuristic. Possible examples of heuristic functions for A* path planning are Manhattan distance, absolute distance between 2 points, and Euclidian distance, distance along a straight line.

2.4 Equipment/Tools Used

In this section the equipment and tools used to develop the individual models of the DT in this project are explored in detail.

2.4.1 PE Equipment

This section outlines the tools used in the PE of the digital twin. The robot that was used to design the aspects of this project is the TB3. In addition to it, the magnetometer used to read the geomagnetic values is the RM3100. The following sections give a more detailed explanation on the reason for the selection of these components.

DT Component	Equipment/Tool	Description
PE	TB3 RM3100	Physical Device Physical Sensor
VE	Unity	Software
Ss	MATLAB Python Rhapsody	Software Software Software
CN	ROS Kinetic TCP Socket	Communication Platform Communication Platform

Table 1: Equipment/Tools Used

TurtleBot3 WafflePi

The specific device being used for this project is the "TurtleBot3 Waffle Pi" robotTur [4]. It was recently purchased by the Model-Driven Software Engineering Group at Eindhoven University of Technology. It has various sensors inbuilt onto it, such as the 360° LIDAR and a 9-axis IMU among others. These sensors detect the physical information from the surroundings and the data is then used to guide the bot as needed. A digital twin of this device has already been developed Busch [9] and it will be used in this report to develop the navigation system.

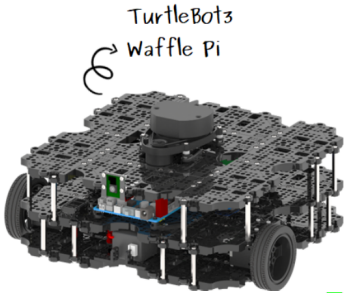


Figure 3: TurtleBot3[4]

RM3100 Geomagnetic Sensor

This requires getting magnetic field data from all 3 axis which can be used to predict the field for the entire region. In the current TB3, there are 2 potential sources of field data. The IMU magnetometer and the RM3100 magnetometer. For this project we select the RM3100 magnetometer. The reasons for doing so are as follows:

- It has an adjustable output data rate which can be tuned depending on the application and scenario.
- Has a measurement range of $\pm 1100 \mu$T which falls well within the limits for the geomagnetic field.
- Has a sensitivity which can measure changes upto 13nT(Geo-Magnetic field typically lies in the range of $\pm 50 \mu$T). However the onboard IMU(MPU9250) has an accuracy of 0.6μT. This difference in potential accuracy lead to favoring the RM3100 over the onboard IMU magnetometer.

2.4.2 VE Tools

The VE in this project is used to mainly visualize the movement of the bot on the PE on a virtual platform as well as visualize other services offered by this project such as the planned path from

source to destination. In order to replicate both the motion of the bot and see the path calculated the visualization tool used in this project is the *Unity Game Engine*.

2.4.3 Ss Tools

The final model developed in the DT in this project is the Ss and this section discusses the tools used to develop the services of this project.

MATLAB

The first tool used to develop the Ss is *MATLAB*. This was done, because the main operating system used during the development of this project on the remote PC(system which hosts the Ss, DD and the VE of the DT) is Windows, whereas the communication platform between the TB3 and the remote PC is ROS kinetic. *MATLAB* was the only programming tool found which provided a link to the ROS network hosted on a remote machine(Linux Virtual Machine hosted on the same PC), as well as provide a platform where the parts of the system can be developed.

Python

Python is the tool used to form the GP model for magnetic field extrapolation. *Python* was the tool selected as it has a wide extension of comprehensive libraries for GP modelling, which helps in optimizing the model hyperparameters for magnetic field extrapolation.

IBM Rational Rhapsody

Rhapsody is the modelling tool which uses SysML to develop the project using the MDSE approach.

2.4.4 CN Tools

A ROS network established over a WiFi connection acts as the primary CN tool for communication between the different components of the DT. It allows for communication between Ss-PE, VE-PE and the Ss-VE. Additionally TCP socket communication is also implemented to communicate between services.

With a basic understanding of the concepts utilized in this project, we now have an accepted model of the DT framework for the system. Additionally the information required to proceed to magnetic field mapping is also discussed along with the tools used to develop the project. Using this, we can now move onto the development of the individual models of the system, using the appropriate methodology, as is discussed in the subsequent section.

3 Methodology

The next step in of the project is to focus on the development of the individual features offered. To tackle the development in a systematic method, the methodology used to define the entire system is the SYSMOD methodology. In this chapter the details of this methodology and its relevance to this project will be explored. Additionally, the process of development of each individual aspect of the DT, will be explored.

3.1 SYSMOD

SYSMOD [2] is a general purpose methodology which provides a good foundation to develop models with the SysML language. Therefore it is the methodology which will be employed in this project. Following this method yields a structured and well formed final result. It also helps in determining the interactions and dependencies of the various components. The specific steps required in the SYSMOD method are shown below.

- *Requirement Analysis*: Identify properties of the system which are needed or wanted by the stakeholder of the project.
- *System Context*: It is represented by a Block Decision Diagram(BDD) which shows the interaction between the system and external components/actors/stakeholders
- *Use Cases*: Use Cases help identify the different services offered by the system
- *Architecture*: Architecture is a BDD, which shows the interactions between the Use Cases and the flow of data/commands between them.
- *Behaviour Diagram*: This shows the realization of the individual use cases as a state chart diagram.

These are the 5 typical steps followed in the SYSMOD methodology. However, these steps do not cover the development cycle of the project. Additionally it does not relate the requirements of the system to the DT model implemented in this project as explained in Section 2. In order to cover these, the SYSMOD methodology is extended. TRIZ 9 boxes is drawn to visualize the development cycle of the project. Additionally a functional analysis of the system to relate the requirements to the individual components of the developed DT is also performed.

3.1.1 TRIZ Analysis

Having identified the framework for the DT that will be followed, as seen in Section 2, the next step would be to start identifying the specifics of the system and how it fits into the different components of the DT. To do so, first the requirements of this project are identified and how it relates to past work done. Additionally potential improvement in the future are also considered, giving a more streamlined development cycle to the project. The tool utilized in this project to do so is the TRIZ 9 box. Performing an analysis with the TRIZ 9 boxes helps better realize the system. Figure 4 shows the 9 box analysis performed for the system developed in this project.

	PAST	PRESENT	FUTURE
Super System	Individual Room	Individual Room, TuckLab	Industrial/ Academic/ Commercial Complex
System	Virtual Visualization of Magnetic Fields. Virtual Replication of motion of bot	Path Planning and Magnetic Field extrapolation	Magnetic Signature Extraction. Localization using visual features
Sub System	LIDAR, Motor, IMU	LIDAR, Motor, IMU, Magnetometer	Camera, Odometer, QR Scanner

Figure 4: TRIZ 9 Box analysis

While drawing up the 9 box diagram does not help in developing any individual component of the DT, such as the PE, VE or the Ss, it does help put the project in the correct context and visualize the progression that it should take during its development cycle.

3.1.2 Requirements Analysis

Deriving precise requirements before the design of the system is crucial. A requirement can be defined as a property of a system that is either needed or wanted by a stakeholder. Coming up with the necessary requirements in the early stages of the project will increase confidence. When the requirements of the system are understood, the system can be demonstrated and accepted. Customers could already agree with the project at the initial or conceptual stages.
Requirements have to be chosen carefully as they will drive the project. Every aspect should be traceable back to the source requirement. A complete list of all the requirements that have been considered and the requirement diagrams can be seen in Table 2.

Display	**Magnetic Mapping**	**Movement**	**Path Planning**
Show Magnetic Field Map	Mapping using Individual Axis Models	Change Direction of Bot	Plot Path with A* Algorithm
Show Physical Obstacle Map	Mapping using Joint Axis Model	Change Speed of Bot	Plot Path with Dijkstra Algorithm
Show Calculated Path	Read Instantaneous Field Values	Move Bot in Straight Line	Plot Path with BFS Algorithm
		Stop the Bot	Identify Destination
			Localize the Bot
			Go to Specified Target

Table 2: Requirements Table

Here we see that the requirements of the entire system can be divided into a few broad categories and have been listed as such. A full description of all the requirements along with their relevant requirement diagram is elaborated in Appendix A.1.

3.1.3 TRIZ contradictions and solutions

Having defined the requirements of the system, the next step is to analyze them and identify any contradictions which may arise between them in any given scenario. Having determined the contradictions, we will be using the TRIZ 40 principles to find a solution. The 40 principles are chosen here as they were created to solve ambiguities and contradictions between requirements or if there are any overlapping requirements.

The first contradiction detected is that the *Show Path Predicted* requirement would be futile if no path has been calculated. This further can be broken down into the selection of a path finding algorithm and identifying a destination. The solution for this is to segment the path predicted function which will work only once an algorithm and a destination have both been selected.
The second contradiction is similar to the first in that the *Go To Specified Target*, requirement only makes sense when an algorithm and a destination have been selected. A similar segmentation solution can be followed for this contradiction.
A final contradiction detected is between the *Go To Specified Target* requirements of the Path Planning category and the requirements of the Movement category. Trying to control the bot manually while having it move to its destination could lead to unfavorable results and can deviate the bot from its target. We use the property of local quality[3] and ensure that the bot can move automatically to its destination only when the bot is not being controlled manually.

3.1.4 System Context

After the elicitation of the requirements, the view of the system in terms of actors and the rest of the instances which interact with our system, is clear. We are now able to construct the context of the system, where the high level interactions of this system is depicted. The system context diagram is created in IBM Rhapsody with a Block Definition Diagram, which is presented in Figure 5, where the high level entities which are related to our system are shown.

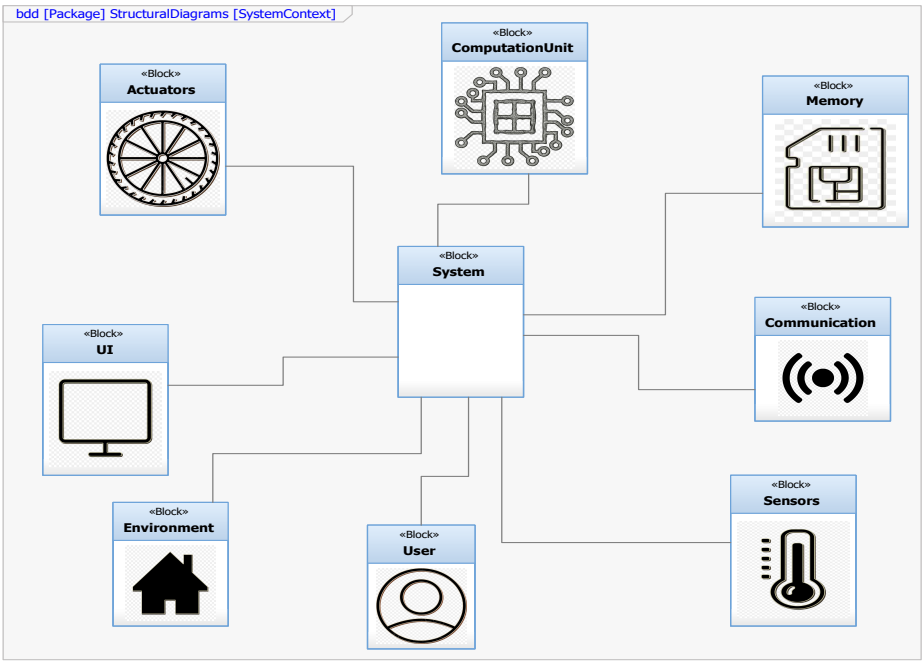


Figure 5: System Context of DT

As can be seen from the system context we have various different components each working in tandem to perform all the tasks of the system. The *Actuators* represent the wheels/motors of the TB3, which cause it to move. The *UI* has various different aspects. It includes the display unit of the VE, the display aspects of the *Computation Unit* which allow for the path display and various other aspects which come out of calculations and a GUI for the controller which allows the User to provide commands and manual control over the system. The *Communication* block is a rather important block as it allows for the inter-operation of the different aspects of this project as well as helps relay commands and data to the different software being used over WiFi. *Memory* accounts for the aspect which helps store the information of the DD, which is read from

Sensors and generated from the *Computation Unit*. The final aspect of the system context is the *Environment* which provides the source for all information used to model not just the physical environment but the magnetic field information as well.

3.1.5 Use Cases

Use cases have been created based on the requirements which were going to be implemented. These Use Cases provide a description which identifies the services the system provides for the stakeholders. The System Use Cases give an outside-in view on the system functions from the perspective of the system actors. This enables the system development to build a system that satisfies the needs of the system actors. A well defined Use Case should address certain aspects such as a short description of its goal, preconditions, post conditions, error situations, the trigger that starts the Use Case, associated system actors, the standard process and an alternative process (if applicable). All use cases are depicted in their respective diagrams in Appendix A.2

Name	PathCalcUC: GoToDestination
Short Description	The bot is made to go to targeted destination
Precondition	Path is calculated from current location to destination
Post Condition	The bot is ready for new target or manual motion control
Error Situation	No path planned or manual control
System State in State of error	Error message displayed
Actors	User, TB3
Trigger	Go To Destination Command
Standard Process	Enable movement control to destination
Alternative Process	NA

Table 3: Use Case Description: Go To Destination

3.1.6 Architecture of the system

The Controller system, consists of separate utilities and functions, which serve a specific purpose for the smooth functioning of the system as a whole. This abstraction of a system to its respective subsystems is represented, using a Block Definition Diagram, and the composition association of the subsystems with the system as shown in Figure 6. The subsystems are a directed composition of the main (high level) system, because they structure it as parts, exactly like the parts of an engine.
In the case of this system, the subsystems are the realized Use Cases.

3.1.7 Behaviour diagrams

The functionalities of the system are conceived through the Use Cases and realized with behavioural diagrams. Behavioural diagrams execute the scenarios of a specific Use Case in a structured way. In certain cases, a behavioural diagram can realize more than one Use Case, if their scenarios are short and can be combined. Behavioural diagrams can be of different types, for example state machine diagrams, sequence diagrams, activity diagrams, etc. For this project, the most effective solution is the state machine diagram implementation, because the system changes states using event messages, or guards. An example of the state diagram is shown in Figure 7. The complete list of the behavioural diagrams of this project is shown in Appendix A.3

3.1.8 Functional Analysis

Having identified a brief overview of the system, we now move to defining the conceptual model of the digital twin using the TRIZ framework as discussed by Wu et al. [23]. The first step in

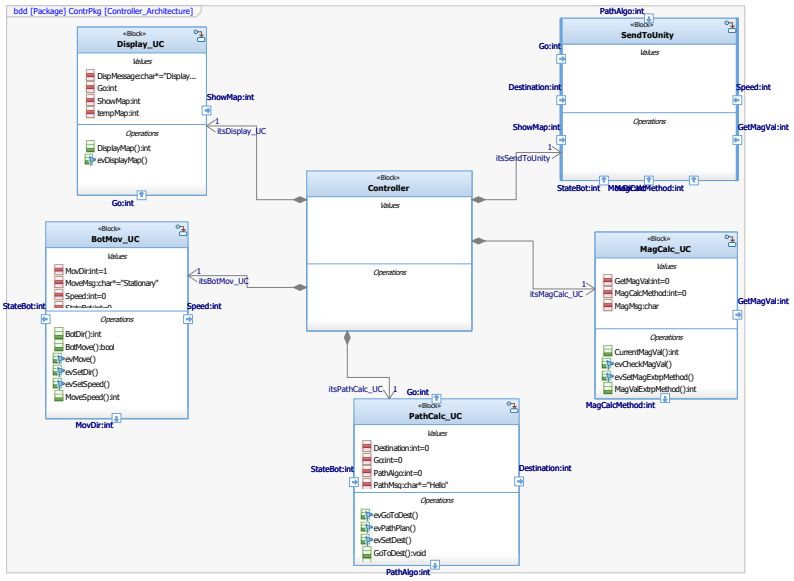


Figure 6: Architecture of the System

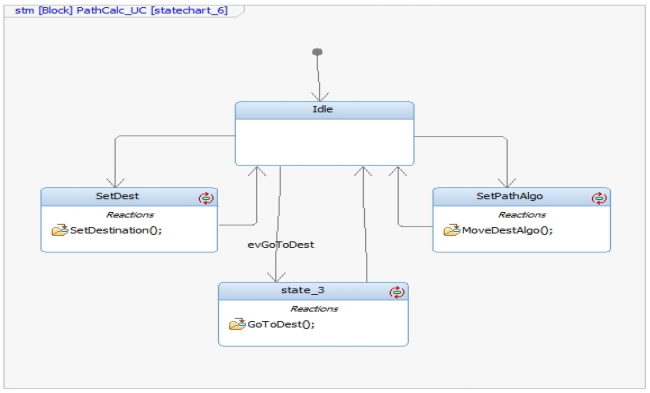


Figure 7: Path Planning Behaviour Diagram

doing so is to identify the functions that are expected to be performed by the digital twin. For this project the basic functions expected to be performed can be broadly categorized as data collection, mapping and navigation. These functions can be further decomposed into further smaller functions. The complete function analysis can be seen in Figure 8.

The functions above can now be categorized in the concerned components of the DT. The data collection and its sub-components are all performed on the physical environment and therefore become a component of the PE. Mapping the environment, and especially formulating the magnetic map, requires computation post data collection and would have results which concern the user and other stakeholders involved in the project. It therefore is a part of the Ss. Finally the navigation function, while requiring computation and control of its own, has a sub-function in it which requires changes in the PE. We can therefore consider that the sub-functions of the navigation function barring the direct to destination, which belongs to the PE and the VE(as the VE of the TB3 needs to replicate the movements of the PE), all are a part of the Ss. The final component which is the system controller would be a part of the Ss.

The next step is identifying the various components involved in the PE and analysing their functions. In addition to this it is determined if enough information is available about the operation of the

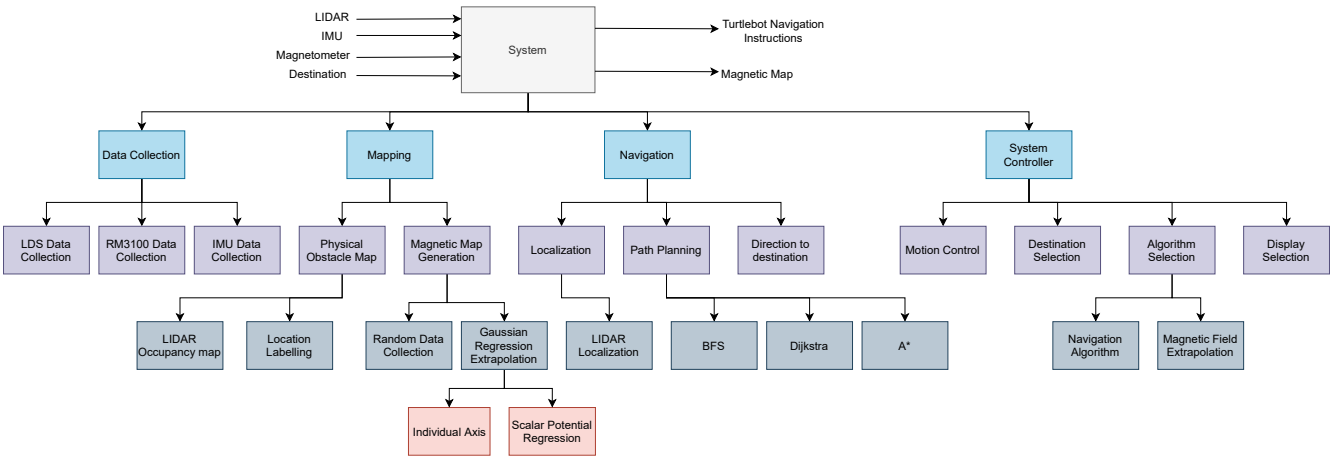


Figure 8: Function analysis of the system

component or if the entire scope of the device is currently unknown. The results of this analysis can be seen in Table 4

Component	**Function Description**
Battery	Power Supply
IMU	Sensor Information
Accelerometer	Acceleration Information
Gyroscope	Orientation Information
Magnetometer	Magnetic Field Information
RM3100	Magnetic Field Information
Motors	Motion of robot
LIDAR	Obstacle Detection, Position
Obstacles	Information Input
Floor Plan	Information Input
Geo-Magnetic Field	Information Input
Raspberry Pi	Computation, information input
OpenCR 1.0	Computation, information input
TurtleBot Remote	Steering Input
Wifi	Connection

Table 4: PE Component Analysis

3.1.9 Conceptual Models of Individual Components

With the individual components of the PE, the next step is to visualize the interactions between the components. Doing so presents us with a conceptual model elaborating the connections between the components of the PE and a first view of how the overall architecture of the DT will look like. This is shown in Figure 9

In the conceptual model of the PE we can see not only the physical components and their interactions but also what the purpose of the interactions are. From the figure we should also note the WiFi block. It acts as the medium to host all the inter component communication, i.e. the CN dimension of the DT.

This conceptual model can now be elaborated upon by adding the required features of the VE as shown in Figure 10.

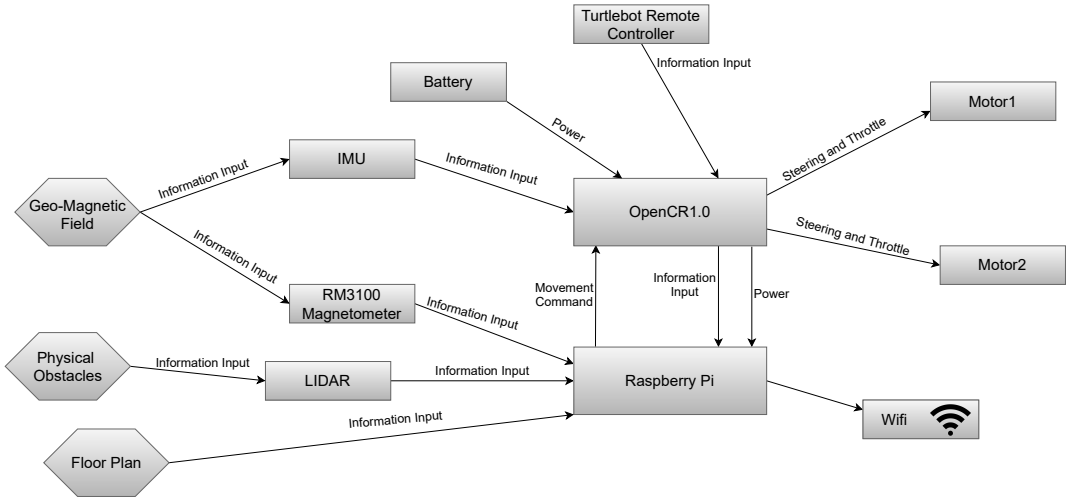


Figure 9: Conceptual Model of PE

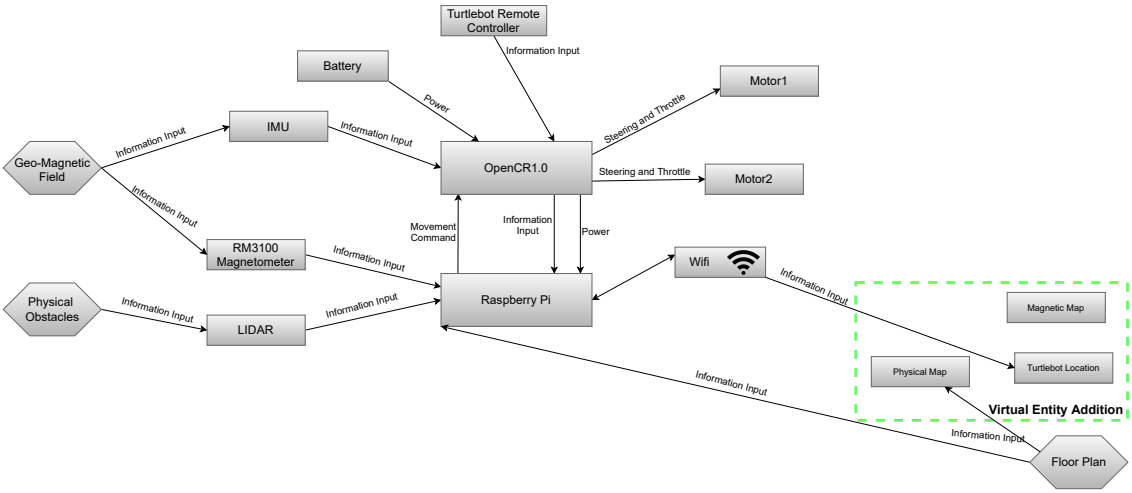


Figure 10: Conceptual Model of VE

The VE conceptual model shows the source of information for the various aspects of the virtual replication. As noted in the figure the source of the physical map in the VE is the floor plan of the concerned region. This simply means that the region has to be replicated into the virtual replication manually beforehand. At present there is no provision wherein live information from the physical sensors can be modelled live onto the VE. In addition to this the source of information for the magnetic map is presently not shown, however it will be discussed in a later figure.

To the model containing both the PE and VE we now add the features of the DD as shown in Figure 11

While not providing any additional interaction information other than the source of the various information at this time, the DD conceptual model provides a platform on which the components of the Ss can be added. Doing so provides us with a complete picture of the entire DT and can be seen in Figure 12

From the figure we can now see what the entire DT should look like.

Having discussed the methodology used to define the system and models of the DT, the next step is to develop these individual models of the DT. The following sections cover the development of

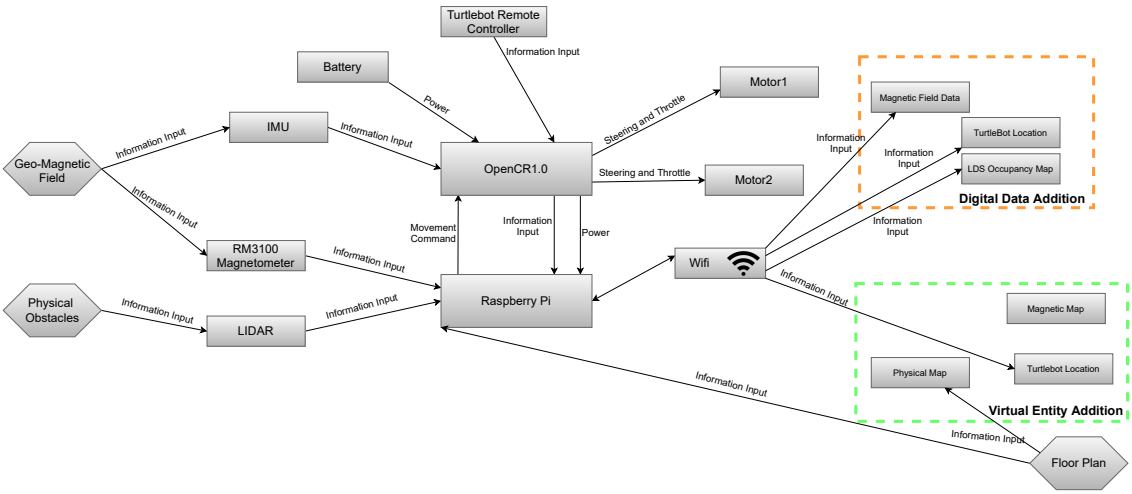


Figure 11: Conceptual Model of DD

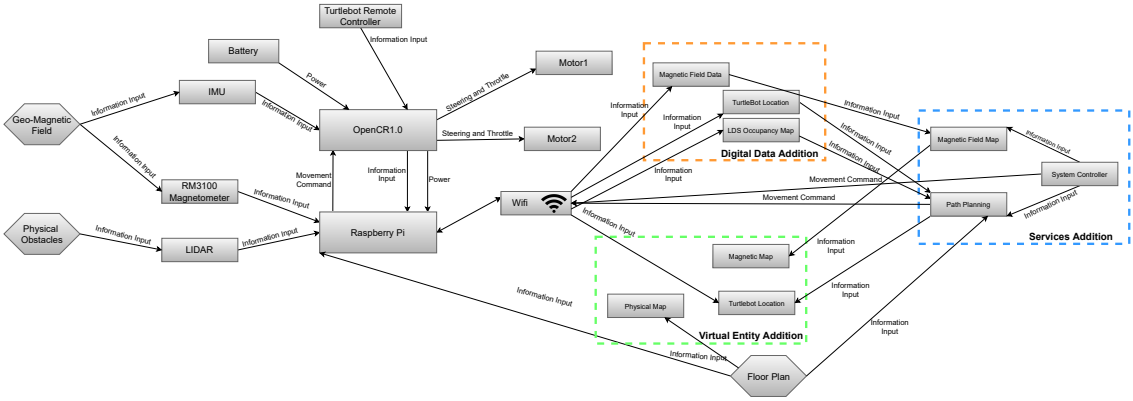


Figure 12: Conceptual Model of Ss

the individual aspects of the DT.

3.2 VE Component

The first component being developed is the VE model. The basis for this is the project developed by Busch [9]. The aforementioned model was developed further by adding a new environment to it(the development environment for the project). Additionally a service was also added which displays the path calculated onto the VE replica of the physical environment. Doing so helps visualize the predicted path and how it would be around any physical obstacles.

3.2.1 Forming the virtual replication of the indoor region

Having defined the system controller, the next step would be to develop the various aspects of the services offered in this project. Among them the first one tackled is creating a virtual replica of the concerned indoor region. The virtual replica is developed on Unity. It was chosen due to ease of use, as well as pre-existing solutions for connections to a ROS network. An existing ROS1 bridge, which allows for data from the TB3 to be directly read into the VE is used as the CN_PV and CN_DD links of the DT
The first environment which was chosen to be worked on was my residence at the time of writing. shows the 1:1 virtual model which was made.

Figure 13: Floor Plan Replication on VE

The model created was a 1:1 model to accurately represent the movement of the TB3 in the physical world accurately in the virtual environment. One requirement of for the virtual model is that the origin of this model, i.e. (0,0,0) be at the point where the TB3 origin lies. This is done because the point where the TB3, is positioned in Figure 13 is the location where it was positioned initially when the map using the LIDAR was formed.

3.2.2 Displaying the planned path

One of the requirements of the system as mentioned in Section 3.1.2, is the display of the predicted path based on the path calculation algorithm implemented. This path is displayed in the VE, highlighting the path to be taken by the TB3. This is shown visually in Section 4.2.1.
The reason for the implementation of this is to add to the visual asthetic of the project at this point. However, this implementation can have far reaching implementations in the future, especially if AR/VR is introduced into a future implementation of the project.

3.3 PE Component

The next development component is working on the PE model. This step works on integrating the external geomagnetic sensor onto the TB3.

3.3.1 Integrating RM3100 with the TurtleBot

In a previous project by Busch [9], the RM3100 sensor was integrated to the TB3, via means of a Arduino Pro Mini data proxy. However, doing so limited the data rate of the sensor to around 5Hz. In this project, the magnetometer was integrated directly into the RaspberryPi of the TurtleBot. Doing so allows for a much higher data rate, thereby providing more data and helping smooth out any disturbances caused by noisy readings. An additional reason why the Arduino was done away with was that it had a magnetic signature of its own and provided an unnecessary source of magnetic field disturbance.

3.3.1.1 Calibrating the RM3100

Out of the box the RM3100 provides magnetic field values, however given the various cycle counts and output rates, this data can get distorted. This requires calibrating the data so as to get accurate and meaningful data from the sensor.

To calibrate the sensor, readings from it needed to be taken in a region with known magnetic field data. This was achieved in the TNO facility at Den Haag. The setup is shown in Figure 15. Figure 14 shows the orientation in which the magnetometer was calibrated.

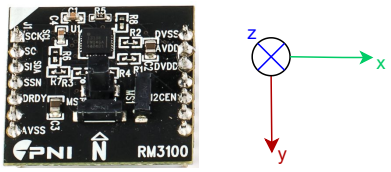
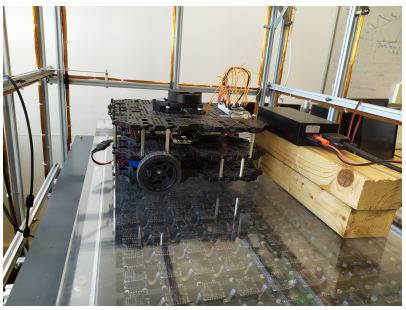


Figure 14: Magnetometer Orientation

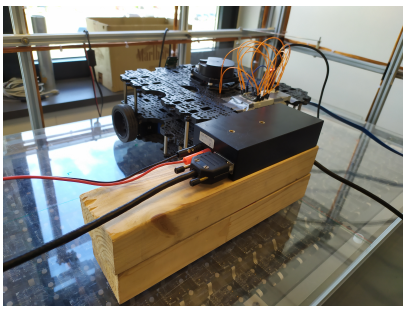
To get the accurate known readings of the field a previously calibrated Mag658 magnetometer [1] was used which can be seen in Figure 15b. The positioning of the Mag658 was kept such that it is as close to the position of the rm3100 on-board magnetometer. This was done to get the most accurate readings as possible. The positioning of the rm3100 itself was kept such that it is as far away as possible from potential sources of magnetic disturbance such as the motors, and the on-board SBC, the RaspberryPi and the OpenCR1.0. In addition to this the work done by Busch [9] further confirms that the positioning of the magnetometer at said position causes it to experience the least interference.
In Figure 15 we can see that the entire calibration setup is placed in a cage of sorts. This cage has wires running along its periphery on all 3 of its dimensions. These coils can be used to control the magnetic field inside the cage, and provide us with a known magnetic field environment using which we can calibrate the sensor. In this setup readings were taken when custom fields were setup using the following scenarios:

- No external field
- ± 0.3A and ± 0.6A in the coil corresponding to a disturbance only in the x-axis.
- ± 0.3A and ± 0.6A in the coil corresponding to a disturbance only in the y-axis.
- ± 0.3A and ± 0.6A in the coil corresponding to a disturbance only in the z-axis.

Readings were take for the above scenarios to cover the entire range of potential geo-magnetic field of $\pm 50\mu T$, along all 3 axis. Now with both the raw data from the rm3100 and the expected values from the Mag658, a linear relation can be found such that the data from the rm3100 closely resembles that from the Mag658.



(a) Calibration orientation

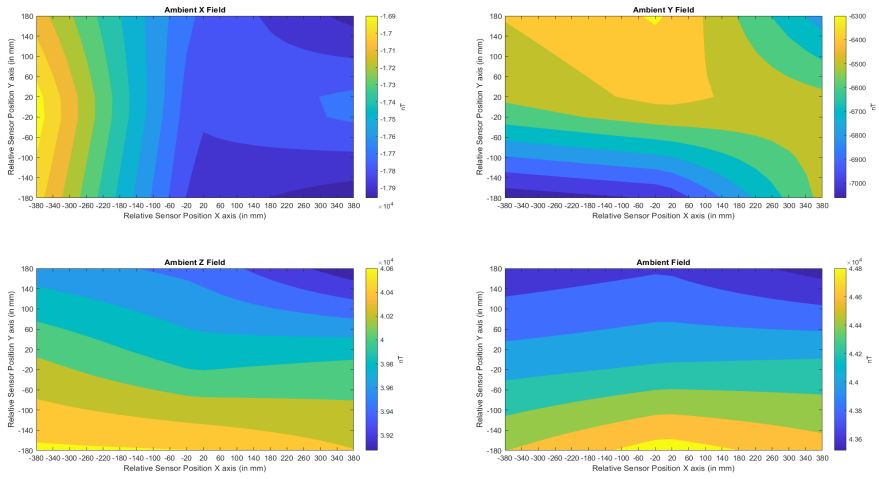


(b) Known Sensor vs RM3100

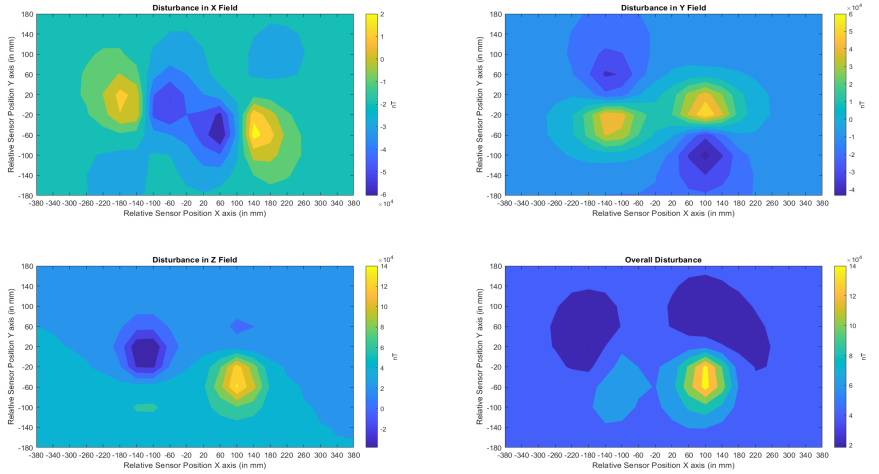
Figure 15: Calibration of Magnetometer

As can be seen from Figure 15, the sensor is calibrated while it is placed on board the TB3. This was done to primarily account for the magnetic field disturbance caused by the TB3 itself. Figure 16b shows the magnetic disturbance that the TB3 causes in the 3 magnetic fields whereas

the ambient undisturbed field can be seen in Figure 16a. Looking at the 2 figures, we can clearly see that the TurtleBot even in a powered down state has a significant magnetic signature in all 3 axis. Should an externally calibrated sensor have been placed on the TB3, the field values it would have read, would be corrupted by the presence of the TB3. However having calibrated the sensor while it is atop the TB3, it ensures that the values now read are the actual field values.



(a) Ambient Field Contour



(b) TB3 Field Contour

Figure 16: Contour of Magnetic Field

Unlike the Arduino as explained above, the TB3, itself could not be done away with and as a result it was determined that the calibration of the magnetometer would be done in a manner such that the data read from it would account for any disturbances caused by the TB3.

3.4 Ss Component

The final DT component developed as part of the DT, is the Ss. Various features are developed as a part of the Ss model of the DT, such as physical and magnetic field mapping, path planning and movement to destination. They are discussed in detail in the following sections.

3.4.1 Physical Map

As mentioned, the physical map defines the physical aspects of the indoor region. Forming this map requires identifying the locations of all physical obstacles. This is done with the use of the on-board LIDAR. The exact steps taken to form the map are the same as listed on the TurtleBot website Tur [4]. An example map generated using the listed procedure can be seen in.
The map generated using this procedure is of the format *.map* and is accessed using the corresponding *.yaml* file. This map is transferred from the host machine(virtual machine hosting the ROS network), to the corresponding service provider software, Matlab. It is read in Matlab in the form of a binary occupancy grid of size (n, m), where n denotes the no of rows of the grid and m denotes the no of columns of the grid. A binary occupancy grid of a region defines a region in small boxes of known dimensions. Each cell has a value signifying if the cell is either occupied or empty. A cell in the occupancy grid denotes a 5x5 cm^2 area in the physical environment.
This binary occupancy grid, is then used to generate an adjacency matrix of size (k, k), where $k = n \times m$. Each row and column denotes a single cell in the occupancy grid. The value of each cell in the adjacency matrix is determined by Equation (1)

$$AM_{ij} = \begin{cases} 1 & \text{if Equation (2) is true} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$OG_{x_i,y_i} = free \ \& \ OG_{x_j,y_j} = free \ \& \\ \Big(\Big((x_i == x_j) \ \& \ (y_i == y_j \pm 1)\Big) || \Big((y_i == y_j) \ \& \ (x_i == x_j \pm 1)\Big)\Big) \tag{2}$$

In the above equation AM_{ij} represents cell (i, j) of the adjacency matrix and OG_{x_i,y_i} represents cell $((i/n), (i\%n))$ of the occupancy grid. In Equation (2), the portion of $x_i == x_j$ ... $x_i == x_j \pm 1$, considers only adjacent nodes to be connected to each other. It excludes nodes diagonally adjacent as in this project we restrict ourselves to straight line motion of the TB3.
The generated adjacency grid is then used to make an undirected, unweighted graph. The edges of the graph are all unweighted or have equal weight signifying that they each have the same significance. The generated graph is then used for path planning as explained further in Section 3.4.3

3.4.2 Magnetic Field Mapping

Having mapped the physical region we can now move onto mapping the magnetic field. The procedure followed to map the magnetic field is the one which has been discussed in detail by Solin et al. [19], Wahlstrom et al. [22]. The process followed will be discussed in detail in the following sections.

3.4.2.1 Data Collection

Before trying to model the magnetic fields, a set of training data needs to be collected. The data required for the magnetic field mapping is the x, y and z field values and the positional data of the magnetometer. In order to do this, the TB3 is controlled manually and made to follow a random path in the indoor environment along which the magnetic field readings from the magnetometer are read and stored. This data will now be used to model the Gaussian model which will then be used to predict the magnetic field for the entire region. As seen in Section 3.3.1, the orientation of a magnetometer is crucial to taking meaningfully consistent readings of the magnetic field data. However, the TB3 has 2 degrees of freedom in regards to its motion. Movement in the forward direction and rotating around its axis. This can lead to shifting the magnetometer from the orientation in which it was calibrated. This changes the X-Y axis orientation of the magnetometer and can therefore lead to inconsistent readings. However, since there is no change in the orientation of the Z-axis, its readings are always consistent.

This issue is tackled by reading magnetic field values in the X and Y axis only when the TB3 faces a fixed orientation, i.e. when the orientation of the TB3 is 0°
Having considered the collection of the magnetic field values, we now consider the collection of the positional values of the magnetometer. The position of the magnetometer onboard the TB3, is shown in Figure 17. The positional data, (x,y) coordinates at a time of the TB3 can be obtained from the localization method followed in this project. This is explained in further detail in Section 3.4.3. However the positional data received is of the base link point of the TB3 as seen in Figure 17. This therefore leads to the requirement of converting the positional data corresponding to the base link of the TB3, to the position of the magnetometer. In Figure 17a helps visualize the position of the magnetometer, (p_b,q_b), with respect to the base link position, (p_a,q_a). Equation (3) shows the conversion of the positional data from base link to magnetometer location.

$$(p_b, q_b) = \begin{cases} (p_a - 0.18, q_a), & \text{if } \theta = 0. \\ (p_a + 0.18, q_a), & \text{else if } \theta = 180. \\ (p_a, q_a - 0.18), & \text{else if } \theta = 90. \\ (p_a, q_a + 0.18), & \text{else if } \theta = 90. \\ \text{Solve Equations (4) to (6)}, & \text{else.} \end{cases} \tag{3}$$

$$\sqrt{(p_b - p_a)^2 + (q_b - q_a)^2} = 0.18 \tag{4}$$

$$\left(\frac{q_b - q_a}{p_b - p_a}\right) = \tan(\theta) \tag{5}$$

$$\big((p_b < p_a) \wedge (\cos(\theta) > 0) \big) \vee \big((p_a < p_b) \wedge (\cos(\theta) < 0) \big) \tag{6}$$

Equation (4) accounts for the x axis offset between (p_a,q_a) and (p_b,q_b) as seen in Equation (4). Equation (5) calculates the position upon rotation of the TB3.Using Equation (4), and Equation (5), we get 2 point along the x axis of the TB3. However, a single point needs to be selected from the two. This selection can be made using Equation (6), which determines which of the 2 points generated by the previous 2 equations is to be finally selected.
In Equations (3) and (4), the 0.18 indicates the 18cm offset between the base link point and the magnetometer position.

(a) Magnetometer Position Translation

(b) Magnetometer Position on TB3

Figure 17: Positional Value of magnetometer

3.4.2.2 Magnetic Field Extrapolation

Having collected the training data for modelling the Gaussian process, we now move on to actually building the model. A Gaussian model in general follows the principle outlined in Equation (8). From the equation we see that the GP posterior y_i can be modelled as a result of a GP prior for value x_i and some noise value ε_i.

$$f(x) \sim \mathcal{GP}(0, \kappa(x, x')) \tag{7}$$

$$y_i = f(x_i) + \varepsilon_i \tag{8}$$

The observed noise is also a Gaussian model $\sim \mathcal{N}(0, \sigma^2_{noise})$. Given that both the prior and the noise are Gaussian it stands to reason that the posterior y_i is also Gaussian.
A key aspect of the GP prior is the covariance matrix which is signified by $\kappa(x, x')$ in Equation (7). While there are various covariance functions available, after looking at the trend that the magnetic field training data takes, it was concluded that the squared exponential covariance function is what will be followed in the following models. The equation for the function is shown in Equation (9).

$$\kappa_{SE}(x, x') = \sigma^2_{SE} \exp\left(-\frac{\|x - x'\|^2}{2\ell^2_{SE}}\right) \tag{9}$$

Here the values of σ^2_{SE} and l^2_{SE} are called the magnitude and length scale hyperparameters respectively. The values for these are learned from training the model on the data.

$$f_d(x) \sim \mathcal{GP}(0, \kappa_{const}(x, x') + \kappa_{SE}(x, x')) \tag{10}$$

The independent predictor variables(x_i in Equation (8)) are the spatial coordinates at which the field values are taken. After the generation of the model the purpose of this section is to predict the magnetic field(y) for previously unknown locations, such that ($\mathbf{y} : \mathbb{R}^2 \to \mathbb{R}^3$). $\mathbb{R}^2$ denotes the spatial coordinates($\mathbf{x}_i$, $\mathbf{y}_i$) in a known environment.
With this basic knowledge on GP and the concerned covariance functions we move onto the formation of the models. 2 different processes are followed. They are as follows.

Individual Axis Model

The first method followed considers the field values for each axis separately. The field for each axis is modelled as a prior separately. However, the covariance function listed in Equation (7) assumes that the mean of the prior is 0. However, as the magnetic field is a physical field which does not have a zero mean, this assumption becomes invalidated. Therefore as mentioned in Solin et al. [19], the covariance for the GP prior is adjusted as shown in Equation (11)

$$f_d(x) \sim \mathcal{GP}(0, \kappa_{const}(x, x') + \kappa_{SE}(x, x')) \qquad , d \,\epsilon\, \{X, Y, Z\} \tag{11}$$

where κ_{const} is a constant value kernel which can be expressed as σ^2_{const}.
This GP prior for each individual axis is now used to predict the magnetic field for the respective axis at the

$$y_{d,i} = f_d(x_i) + \varepsilon_{i,d} \tag{12}$$

Here $\varepsilon_{i,d}$ denotes the Gaussian noise corrupting the output data with a noise variance of σ^2_{noise}.
Each axis model has 4 hyperparameters:

- σ_{SE}(magnitude scale for SE kernel)
- ℓ_{SE}(length scale for SE kernel)
- σ_{noise}(noise variance)
- σ_{const}(variance for const kernel)

According to what was reported by Solin et al. [19], allowing the model to tune these hyperparameters can cause it to be stuck at local minima points. However, despite this when experiments were run to optimize the parameters by maximizing the log marginal likelohood of the functions, the results seemed favorable enough to be accepted. This will be further elaborated in Section 4.

Joint Model

The magnetic field was modelled using the method described above. However considering the observations of Solin et al. [19], about individual model hyperparameters converging at local minima another model was also considered. This process was also laid out in the same report. It allows for shared hyperparameter tuning of the 3 axis fields. To do so, a single model was constructed with the same kernel as the previous section, however for training instead of having a single field as the predictor variable all 3 fields are considered the predictor variables for the same model. The result for this is seen in Section 4.

3.4.3 Planning the path

With the completion of the mapping part of this project the path planning and movement to destination is developed. All the path planning algorithms used in this project are born out of graph theory and the graph generated in 3.4.1 is used. The section focuses of forming the required path and following it to reach the destination.

3.4.3.1 Localization

A pre-requisite to planning a path is knowledge of the current location/start point of the path. In addition to finding the start position for path planning the localization of the TB3, plays a key role in the magnetic field mapping as was explained in paragraph 3.4.2.2. This is because the model developed relates the magnetic field location to the X-Y coordinate of a region. It is therefore extremely important that we have the accurate values of the X-Y coordinates as they are directly related to the magnetic field values.
There are 2 potential methods considered for localizing the TB3.

- ***Odometry(odom)***: Localizing by integrating the data from the wheel encoder over time. This data is calculated by the TB3 itself and is readily available on the */odom* ROS topic.
- ***Adaptive Monte Carlo Localization(amcl)***: Localizing using the scan data from the LIDAR. This data is then used in the turtlebot navigation node as shown in Tur [5], giving the current X-Y coordinates of the bot.

The *odom* method returns the current location of the TB3, by keeping track of the wheel rotations and accumulating it over time. This method is highly accurate as can be seen in Figure 18. As a consequence of this the location calculated using this method is independent of any pre-formed map. Thus making it immune to any localization errors which may occur when the localization algorithm is run in an environment which may have slight changes compared to the previous map, such as a new table/chair being introduced which was not accounted for in the map which was previously formed. Additionally it was observed that the data rate for this localization method given the current sensors aboard the TB3, is 20 Hz. After testing experimentally it was observed that this data rate was updating the location at a rate sufficient for not only directing the bot to its final destination(as will be explained in Section 3.4.4), but also was sufficiently higher than the data rate of the magnetic field values read. Having location values at a higher rate than the magnetic field values ensures that when the magnetic field data is available it is represented by the accurate location values, thereby avoiding any data spillage/inaccuracies. A concern when considering this method is that, there would be errors arising and accumulating over time, should there be any inaccuracies in the position calculation, which would compound over time. This would give incorrect values of the position as more time passed. To verify this a test was performed at 3 different speeds of the TB3. Different speeds(linear:Lin and angular:Ang) were tested, to ensure that the localization algorithm would return accurate locations, even when the location was updated at a faster rate. The results of this test can be seen in Figure 18. In the graph, the y axis represents the squared error between the reported location from the *odom* topic and the actual expected location. It is calculated with Equation (13). All reported speeds are in m/s

$$score = \sqrt{(x_a - x_r)^2 + (y_a - y_r)^2} \tag{13}$$

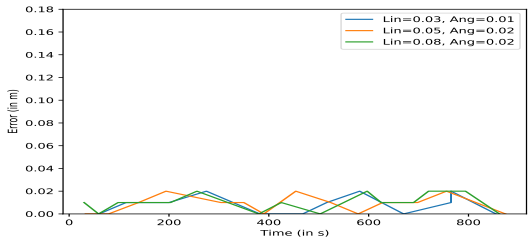


Figure 18: Absolute Error vs Time

Here (x_a, y_a) represents the actual position and (x_r, y_r) is the reported location position from the odom topic. From the graph we can see, that at varying speeds upto 0.08m/s the error is recorded position never exceeds 0.02m. This error itself could be attributed to slight discrepancies which were made during recording of readings, because there are instances when the error does go to 0. This further supports that at the instances when there were errors the readings were taken when the bot was at a slight offset from the expected location. Hence, from the results presented in the graph we can conclude that the results of using this localization method are accurate.
However, a limitation of using this localization method is that the origin gets reset to the point where the TB3 is turned on every time the bot is rebooted.
The *amcl* method performs the localization by reading the data from the LIDAR scans, and relating it to a previously formed map using the LIDAR. The accuracy of this method is dependant on the relevance of the previously formed map to the present time. This method of localization would be susceptible to any slight changes which may have been made in the map. In addition to this the data rate of updating the location for this method is dependant on the rate at which new information is read from the LIDAR. In other words it is dependant on the speed of the bot. This can be seen in Figure 19.

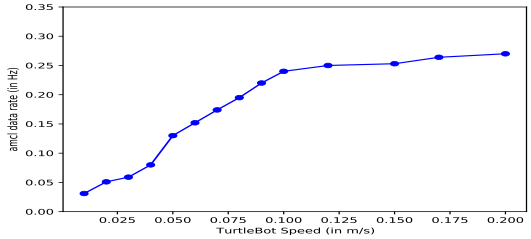


Figure 19: amcl Data Rate vs TB3 speed

The data rate for localization is important in the magnetic field data sampling explained in paragraph 3.4.2.1, to ensure that magnetic field values are collected along every 0.01m. During the collection of the data the speed of the bot is kept relatively low, 0.05m/s, so as to record maximum data points for training the Gaussian models. At 0.05m/s the observed data rate of the *amcl* method is 0.13Hz, which equates to a new data point approximately every 7s. In this duration the bot would move forward by 0.35m and we do not collect any data over this distance. This limits the amount of training data points available, thereby affecting the accuracy of the trained Gaussian models. Additionally during the movement of the bot to its target destination, as will be elaborated in Section 3.4.4, the speed of the bot is kept at 0.1m/s. At this speed the observed data rate of the amcl method is approximately 0.24Hz. This results in a new data point every 4s. This

causes the bot to move a distance of .4m before a new localization result is observed which could cause the bot to overshoot its destination, which is not an acceptable outcome.
Considering the assessment done of both potential localization algorithms, it was concluded that though both algorithms give an accurate value for the localization, the *amcl*, is limited by its data rate. This limitation is a result computing the location from the amcl algorithm and additionally it is also limited by the data rate of the onboard LIDAR, which is 5Hz. Considering all the outlined parameters the *odom*, method was selected for localization. This was primarily due to the advantage of higher data rate it has over the *amcl* method without compromising accuracy. Even though doing so, presents us with the limitation that every time the bot has to be booted up at the same initial position, it is a compromise that offsets the errors which may occur by low location update rates.

3.4.3.2 Destination Selection

Another pre-requisite of implementing path planning is determining the destination node for the algorithm. The destination node is determined based on the coordinates of the destination. While the destination can be set to any coordinate which has been previously mapped a general practice throughout the rest of the project is to select destination coordinates based on pre-determined landmarks of the region.
In addition to the pre-determined landmarks, a system has been implemented such that the system is able to detect the region of largest magnetic field intensity, and can use that point as its target destination.
The coordinates as previously mentioned inSection 3.4.1 are based on the previously mapped region. Having a destination coordinate which was not previously mapped by the bot is not supported by this project.

3.4.3.3 Path Planning Algorithms

In this project the path planning service is offered using Matlab as the computation unit. Doing so allows us to take advantage of pre-existing solutions in Matlab to calculate a path from one node to another in a graph. The implementations of the concerned algorithms are discussed below. For this project we restrict ourselves to shortest path algorithms.

BFS
BFS is one of the most basic shortest path finding algorithms. In order to implement the BFS shortest path finding algorithm, the in-built MATLAB function *shortestpath()*, is used.
To ensure that *BFS* algorithm is used for path calculation the *Method* parameter of the function is set to *unweighted*. Doing this makes the function return a path from the start node to destination node which was calculated using the BFS algorithm.

Dijkstra
Dijkstra's algorithm is one of the most popular path finding algorithms in graph theory. It has the advantage of being able to find a path using a weighted graph where each edge has a penalty value. However, for this project this does not matter as all edges in our graph have the same weight/penalty.
Similar to the BFS algorithm an in-built Matlab function is used to find a path using Dijkstra's algorithm on a graph. It is done using the same *shortestpath* function.
To ensure that *Dijkstra* algorithm is used for path calculation the *Method* parameter of the function is set to *positive*. Doing this makes the function return a path from the start node to destination node which was calculated using the Dijkstra algorithm.

A*
*A** is the last algorithm considered for path planning. It works in a similar manner as Dijkstra's algorithm, but has an added feature of using a heuristic function in addition to the distance to

the destination to calculate the path from one node to another. 2 different heuristic functions are considered:

- *Euclidian Distance*: the length of a straight line from source to destination
- *Manhattan Distance*: the sum of the individual distance between the source and destination along both axis(as project considers a 2D space)

To the best of knowledge there are no pre-existing functions in Matlab which provide A* path finding in a graph. To that extent a custom code was written for finding the path using $A*$. This code incorporates a selection parameter which allows it to switch between the 2 considered heuristics. The full code is provided in Appendix B.3.

3.4.4 Move to destination

Having determined the path to the destination from the current location, the next step is to direct the TB3 to the destination along the determined path. Given that the path only consists of straight lines[1], it was determined that the robot only ever moves forward. In the case of a route along the other axis the TB3 is made to rotate along its central axis till it faces the direction it needs to move in. Figure 20 shows the steps taken when directing the TB3 to its target destination.
The path which is returned by the path planning algorithm described above provides a constant list of adjacent nodes from source to destination. However implementing the movement over this constant adjacent nodes, led to issues of jerking like behaviour of the bot. This is because of the constant start and stop which happens due to the next node in the path being the one adjacent to the current node. This led to a disturbance in the orientation of the bot and ultimately caused to bot to drift significantly away from its destination. To overcome this, the path calculated was filtered down to identify the edges of the path. Doing so ensured that the local target node would not always be the adjacent node(unless in the case of a quick s-bend). This prevented the constant movement and stop of the bot and mitigated the problem of jerking and drifting of the bot to a large extent.

3.4.4.1 Obstacle Detection

During the motion of the TB3 to its destination the calculated path refers to a map made at a previous time. However, from the time the map was made, there could be new obstructions or obstacles along the way. As a result of this obstacle detection was added to the code. Obstacle detection in this case uses the data from the LIDAR. The data from the LIDAR comes in as an array of length 360, where each index of the array contains a float value which indicates the distance to the closest obstacle at that angle from the central lateral axis of the TB3, in a clockwise direction.
Given that the motion of the TB3 is restricted to straight line movement in the forward direction, as explained earlier, we need to focus on the data concerning only the front facing side of the TB3. Figure 21a shows the range of values which are considered when searching for obstacles. For this project we assume that the TB3, will stop in its motion if there is an object in front of it at a distance of $\leq$ 0.5m. Considering this distance and that the width of the turtlebot is approximately 0.3m, we need to search a distance of at least 0.3 m parallel to the TB3, for obstacles, which could interfere with the bot's motion. For the sake of safety we consider a buffer of 0.025m on each side as well, and therefore we scan a distance of 0.35m, 0.5m in front of the TB3. Considering these values and using the equation Equation (14), where S = 0.35m and R = 0.5m, we get that the angle, θ is approximately 0.69 rad, or 40°. As a result of this, we look at data for 20° from either side of the central axis of the TB3. A continuous scan of the array points returned from the LIDAR data at the aforementioned points tells us the distance to the closest obstacle in the

[1]This is because of the choices made during the graph forming where only non-diagonal adjacent matrices were considered connected

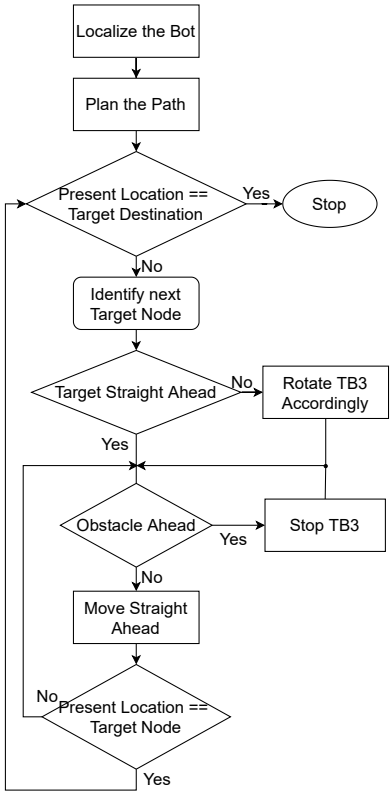
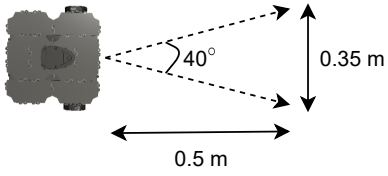


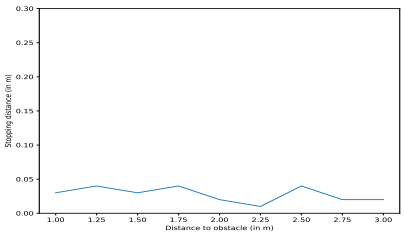
Figure 20: Flow Chart for TB3 Motion Control

defined range, and should a value on any of those points be less than 0.5, the motion of the TB3 is halted until all the values are > 0.5 again.

$$S = R \times \theta \tag{14}$$



(a) TurtleBot Cone of View



(b) Stopping Distance

Figure 21: TB3 Obstacle Detection

A test was performed to check the braking distance of the TB3. This test was necessary to ensure that the distance selected of 0.5m was sufficient and that there would be no collisions. The result of the test performed are shown in Figure 21b[2]. As can be seen from the results, that the TB3 stops well within a safe distance from an obstacle at any distance. However, despite this the scanning distance is still kept at 0.5m to ensure the safety of the TB3 and any potential obstacles, human or

[2]The speed of the turtlebot in all these cases was kept at 0.1 m/s(approximately 33% of the top speed of the bot)

otherwise. The data from the graph about distance less than 0.5m tells us that should any obstacle come suddenly in front of the bot, it should still be able to stop within a reasonable distance. The complete code for the motion control of the bot to its target destination is provided in Section B.3.

This concludes the methodology used for the development of the project as a whole as well as the services offered in the individual models of the DT. With the developed individual aspects, the next step is to consider the results of implementing the developed system.

4 Implementation

The previous chapter elaborated on the procedure followed to create the project. This chapter focuses on visualizing the results of the implementation. In addition to the services discussed, this chapter also presents a control interface which was developed as an over-arching interface to the DT developed.

4.1 Rhapsody GUI

The services being offered in this project have multiple choices in them, such as a variety of path planning algorithms and magnetic field extrapolation methods. Given these choices, a Graphical User Interface(GUI) was developed on Rhapsody which allows the user to select from the different available choices, during the final implementation of the system. It consists of various elements such as buttons, display screens, lights and knobs. It allows the user to make various selections such as navigation algorithm, target destination and choices on the different map to be displayed. The GUI is shown in Figure 22.

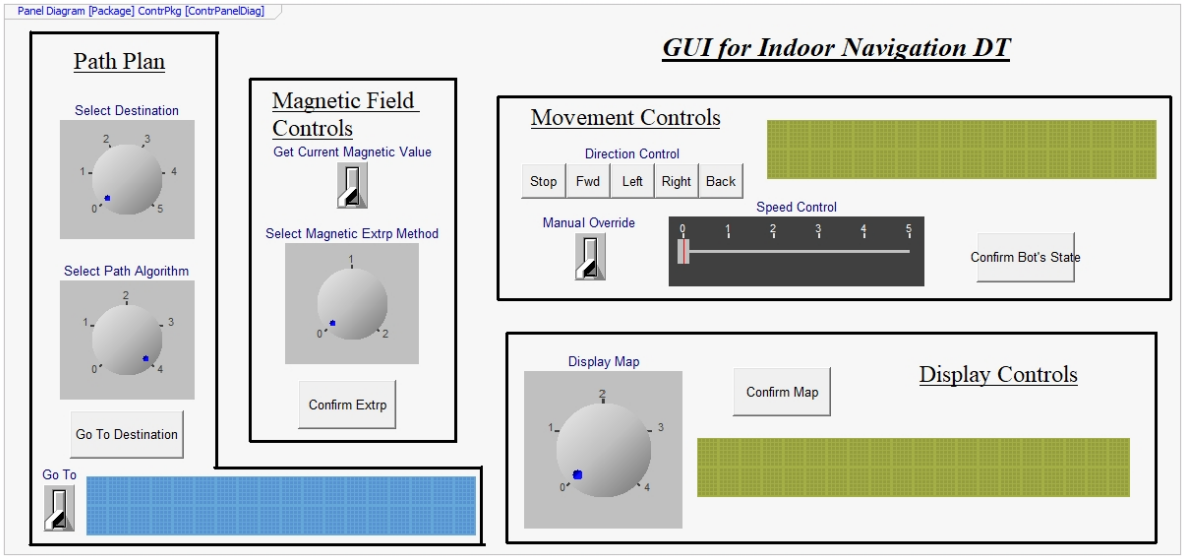


Figure 22: Rhapsody GUI

The GUI is divided into various parts. Each part is labelled appropriately. The movement controls allow for override manual control of the motion of the TB3. Additionally it also indicates if the bot is in manual control or not. Path Plan portion allows the user to select the destination(hard-coded coordinates of the target) and the path planning algorithm(1: BFS, 2: Dijkstra, 3: A* Euclidian, 4: A* Manhattan). Additionally it also sends the command to send the TB3 to the target destination. The GUI also has controls for the magnetic field mapping service of the system. It allows the user to choose between the 2 field extrapolation methods(1: Individual Axis Model, 2: Joint Axis Model) and also gets the system to read the instantaneous field value at the current position of the TB3. The final aspect of the GUI is the display option. The GUI allows the user to make the following choice options about the display:

1: Display the predicted path

2: Display the X-axis extrapolated magnetic field

3: Display the Y-axis extrapolated magnetic field

4: Display the Z-axis extrapolated magnetic field

In addition to allowing for choices the GUI also provides feedback messages which are pertinent to system. Doing so helps visualize and ensure that the TRIZ contradictions are ensured. The results of this can be seen in Figure 23. As can be seen from the image when the bot is in manual control mode, it cannot be used for path planning.

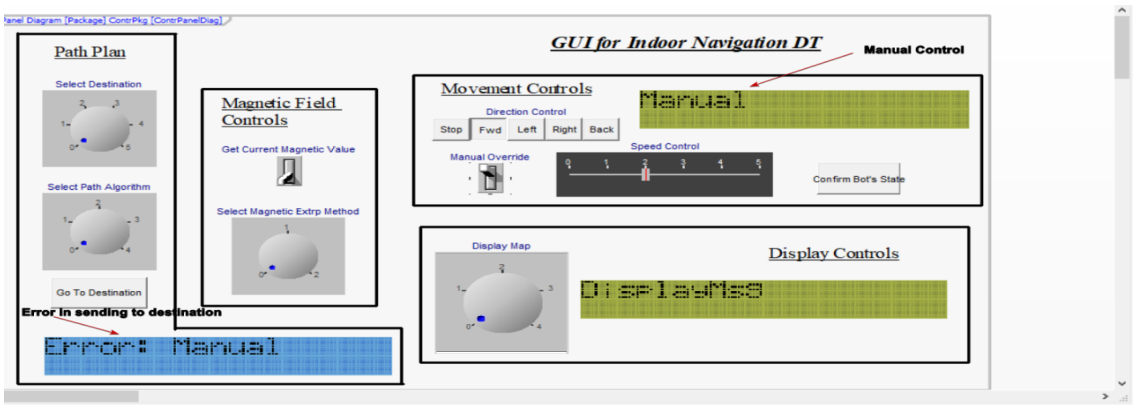
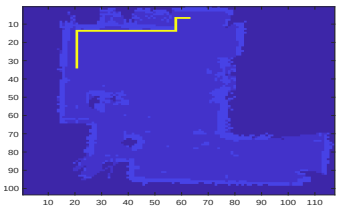


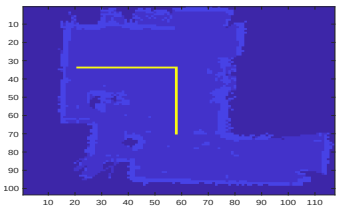
Figure 23: Manual Control Contradiction

4.2 Path Planning

The path planning implemented in this project as explained in Section 3.4.3, returns a path from point A to B. The result of this path planning can be seen in Figures 24 to 27. As can be seen from the figure, the path consists only of straight lines. This is due using graph theory algorithms and that the entire region is mapped as nodes of a graph. Establishing a path as such has the advantage in controlling the motion of the TB3, as it allows for motion along a singular axis at a time.

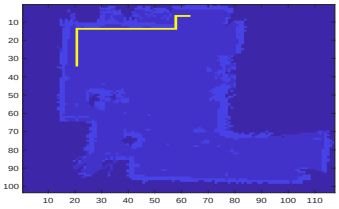


(a) TurtleBot Station to Door

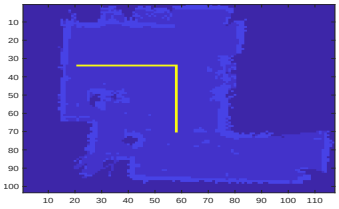


(b) TurtleBot Station to Table

Figure 24: BFS Generated Path

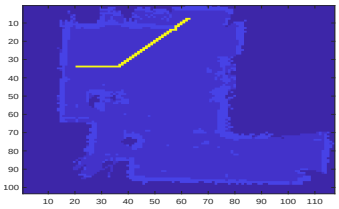


(a) TurtleBot Station to Door

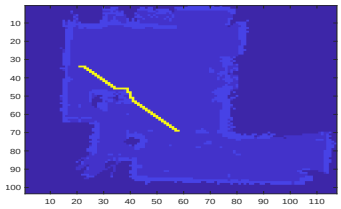


(b) TurtleBot Station to Table

Figure 25: Dijkstra Generated Path

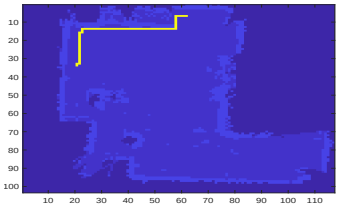


(a) TurtleBot Station to Door

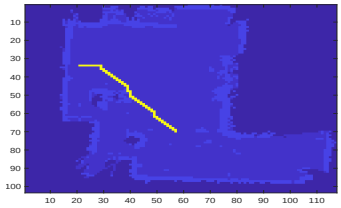


(b) TurtleBot Station to Table

Figure 26: A* Euclidian Generated Path



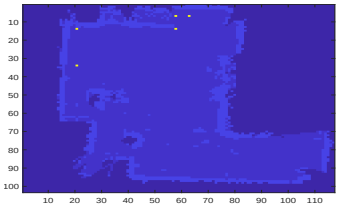
(a) TurtleBot Station to Door



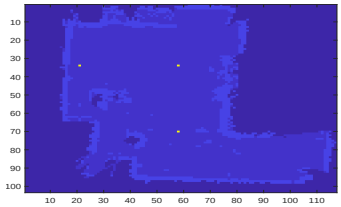
(b) TurtleBot Station to Table

Figure 27: A* Manhattan Generated Path

The edges of the path are extracted and the results of this are shown in Figures 28 to 31.

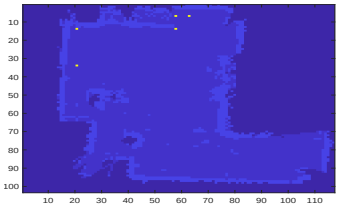


(a) TurtleBot Station to Door

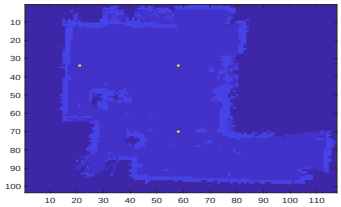


(b) TurtleBot Station to Table

Figure 28: BFS generated Path: Edges

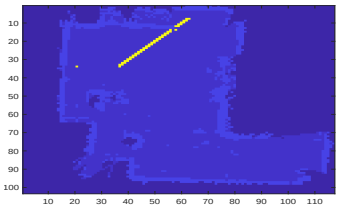


(a) TurtleBot Station to Door

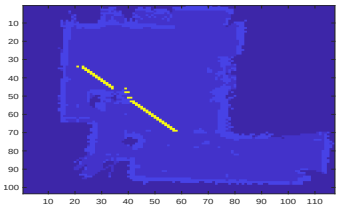


(b) TurtleBot Station to Table

Figure 29: Dijkstra Generated Path: Edges

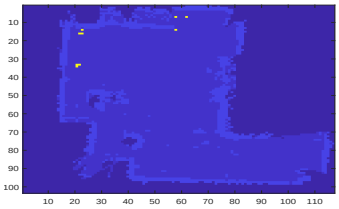


(a) TurtleBot Station to Door

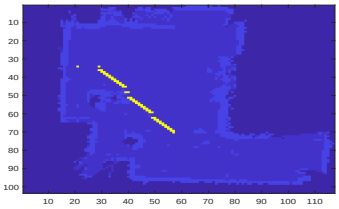


(b) TurtleBot Station to Table

Figure 30: A* Euclidian Generated Path: Edges



(a) TurtleBot Station to Door

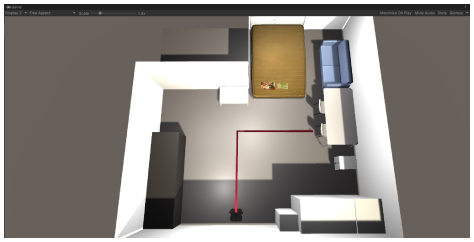


(b) TurtleBot Station to Table

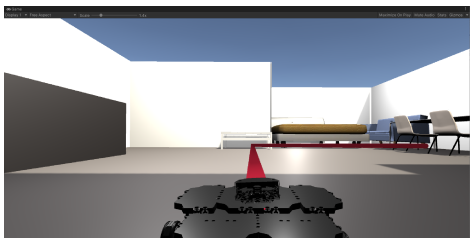
Figure 31: A* Manhattan Generated Path: Edges

4.2.1 Visualizing Path on VE

Having developed the path planning system for all 4 algorithms, we now visualize the path on the VE model of the space. The results can be seen in Figure 32.



(a) Top View



(b) First Person View

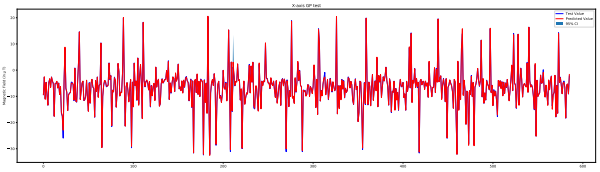
Figure 32: Visualization of Path on VE

4.3 Magnetic Field Mapping

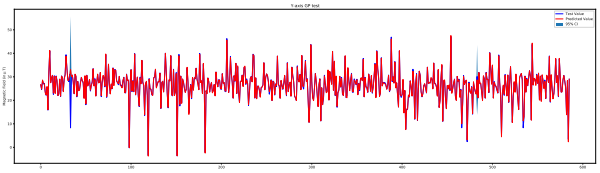
Having developed the models for the magnetic field extrapolation we now move onto visualizing the results of the efforts. The results of both extrapolation methods are demonstrated below.

4.3.1 Individual Axis

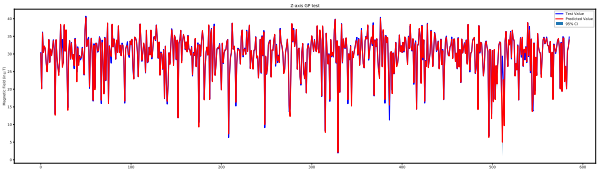
Firstly we visualize the results obtained from the models developed individually for each field. Figures 33a to 33c shows the result of the predictions of the model, compared against the measured value at those points.
Looking at the results in Figure 35, we can see that the predictions made by the models are very close to the actual expected values. An in-depth analysis on this is done in Section 5. The next step is to predict and visualize the individual field components for the entire test space. The results of this are seen in Figure 34. In this figure for each field the figure on the left shows the field value measured at each point which is used as the training point for the Gaussian Model for the



(a) X Axis



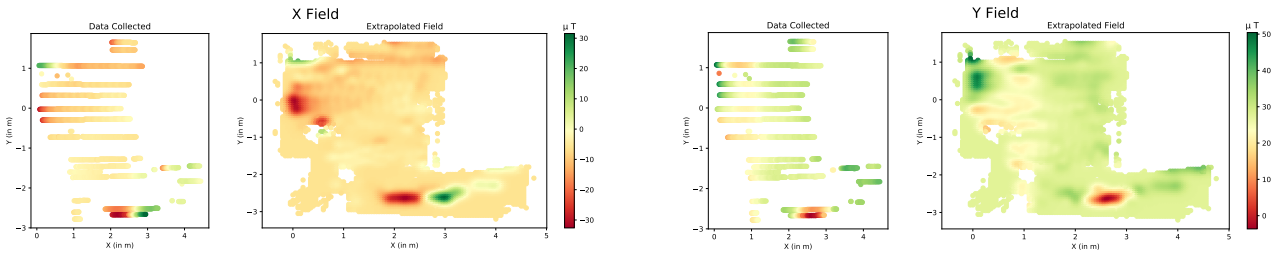
(b) Y Axis



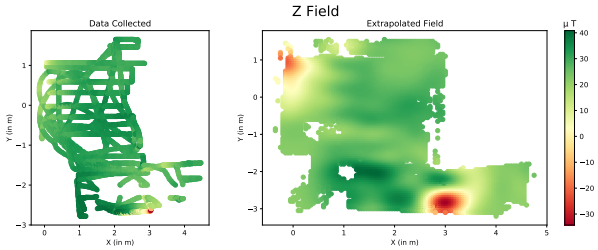
(c) Z Axis

Figure 33: Expected vs Predicted Field Values For Individual Models

concerned field. The figure on the right then represents the results of the extrapolation done using the trained model.



(a) X Field **(b)** Y Field



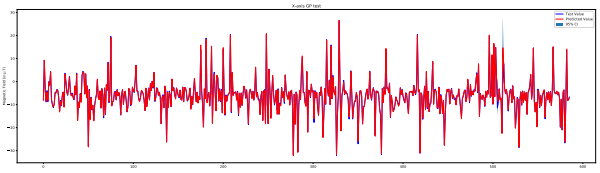
(c) Z Field

Figure 34: Predicted Magnetic Field on Map using Individual Axis Models

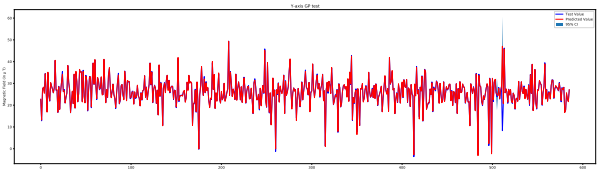
4.3.2 Joint Model

We next move onto visualizing the predictions from the model developed for all the axes together. Similar to the results visualized in the individual axis, we now move onto the visualization of the results of the model developed for all 3 axis together. Figure 35 shows the results of the field values

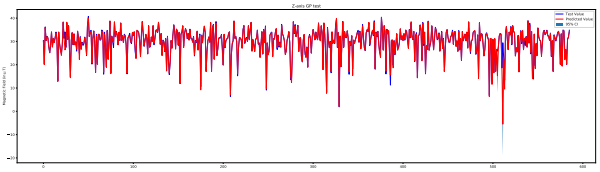
predicted by the model compared to their actual values. As it seems other than a few outliers, all predicted values lie in the range of actual values.



(a) X Axis

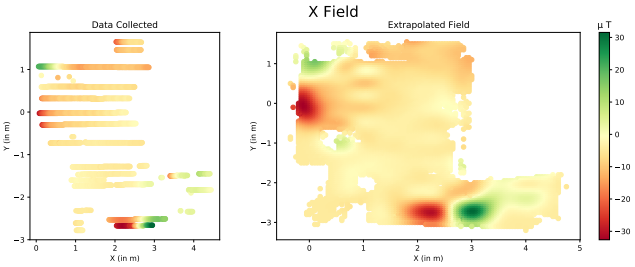


(b) Y Axis

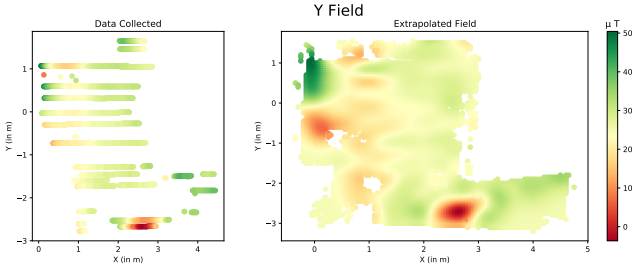


(c) Z Axis

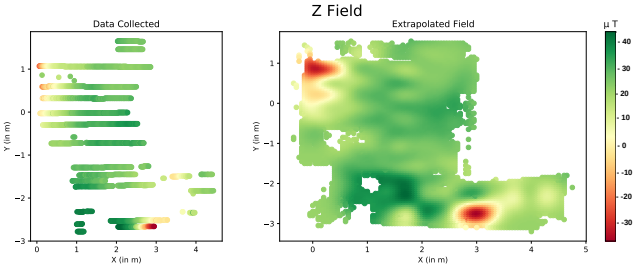
Figure 35: Expected vs Predicted Field Values For Joint Model



(a) X Field



(b) Y Field



(c) Z Field

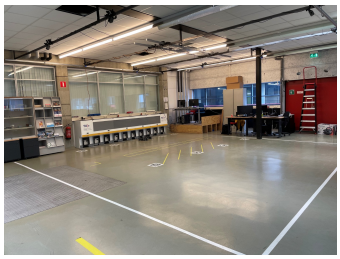
Figure 36: Predicted Magnetic Field on Map using Joint Axis Model

4.4 TruckLab Implementation

A main requirement during the development of this project was its ability to be scaled up to a larger environment or even to other projects. Considering this, each service offered was developed

as an individual model. Using these individual service models, the concerned features can be ported to a different project.
The TruckLab Automotive Technology Department, uses TurtleBot Burger to replicate physical trucks, which are referred to as TurtleBot Trucks(TBT), Figure 37a, hereon. The service developed in this project can be ported to be used with the trucks in the Trucklab workspace, Figure 37b. The workspace considered here is a near 6×6 m^2 area. Using the TBT and the physical space as mentioned earlier the following services of this project are implemented in the TruckLab environment. This development provides a proof of the scalability of the services of this project and helps reaffirm confidence in the development process taken.

(a) TBT



(b) TruckLab Space

Figure 37: TruckLab PE

4.4.1 Path Planning

The path planning service of this project was scaled to be implemented in the TruckLab environment. Firstly a map of the region was developed, using the onboard LIDAR. However, the onboard LIDAR, has a maximum range of 3.5m. However, the free space of the lab has a free space much larger than the range of the LIDAR. This led to issues in mapping the physical region and its obstacles. Incidentally the entire experimental space is a completely empty rectangle which was then hard-coded as an occupancy grid matrix. In this matrix each cell is represented as a 5x5 cm cross section of the region. The matrix is then used to generate the graph of the region with each node representing the individual cells of the matrix and edges representing a connection between any two nodes. The generated graph is then used to plot a path from a given source node to a different destination node. The results of this is shown in Figure 38.

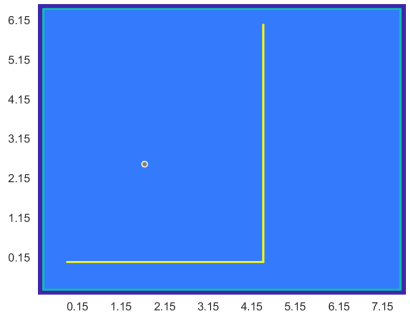


Figure 38: TruckLab Path Planning

The TBT, used in this implementation is not a two wheeled differential robot, the same as the TB3. It is a 4 wheeled vehicle based model. As a result it cannot be used to make 90° turns around its own axis. Therefore even though an accurate path was generated using the algorithms proposed in this project for the TruckLab space, the motion control service developed in this project cannot be

translated for the TBT. To control the motion of the TBT to be directed to its target destination along the calculated path, a separate motion control algorithm would be required which is able to accurately control a vehicle based model. Given the model based implementation of this project, integrating this control algorithm, once it is developed, into this project would be a relatively easy task.

4.4.2 Obstacle Detection

The work being implemented at the TruckLab by Ayush Maheshwari, includes controlling the motion of the TBT across the physical space. As an additional feature to the motion control of the truck, the obstacle detection implemented in this project was ported to be accommodated in the motion control of the truck developed by Ayush. The implementation of the obstacle detection in this section differs slightly from the implementation as described in Section 3.4.4. The difference lies in the cone of view, which is 10°, instead of 40°. The change was made for the following reasons.

- The width of the truck is less than that of the TB3. The scanned region is around 0.25m, 1.5m in front of the truck. This is appropriate for the truck as its width is approximately 0.18m.

- The obstruction in the side views of the truck caused due to presence of the 3D printed covering on top of it.

Figure 39, shows the results of suddenly sliding an obstacle in front of the truck during its motion. Though it is not evident from the figures, the truck halted in its motion upon detecting the hindrance in its path, and only moved forward upon the removal of the obstacle.

Figure 39: TruckLab TBT Obstacle Detection

4.4.3 Magnetic Field Mapping

The last service offered in this project which was successfully scaled up to be accommodated into the TruckLab environment is the magnetic field mapping of the environment. However, due to the limitations of magnetometer calibration and the disturbances caused by the device the magnetometer is placed on, as explained in paragraph 3.3.1.1, the TB3 was the device used to collect the magnetic field data along the 3 axis. The results of this data collection is shown in Figure 40. As can be seen the data is collected only along a fixed orientation of the magnetometer, while the TB3 is moving in a straight direction. As mentioned previously this affects the fields generated for the X and Y axis, however does not affect the Z axis fields. However, the Z axis training data was also limited for this implementation, due to a shortage of computation resources. The data collected in all possible exceeded 22,000 data points. When these data points were attempted to be fit into a GP, the amount of memory required exceeded the maximum available memory. As a result the maps generated here, are not the most accurate representation which would be available. However, this can be considered a stepping stone to generating the magnetic field map of the TruckLab environment in a future project with access to remote cloud services.

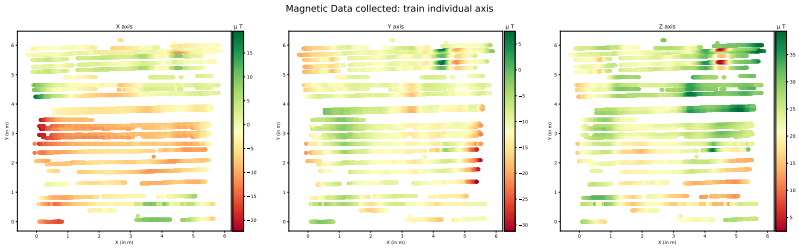
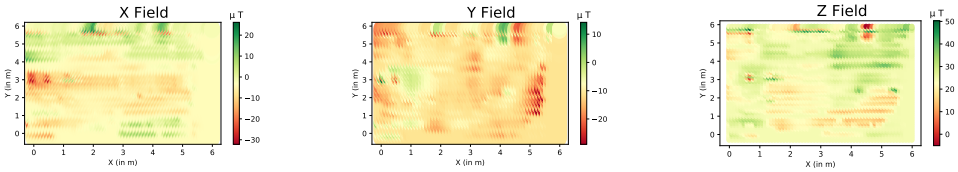
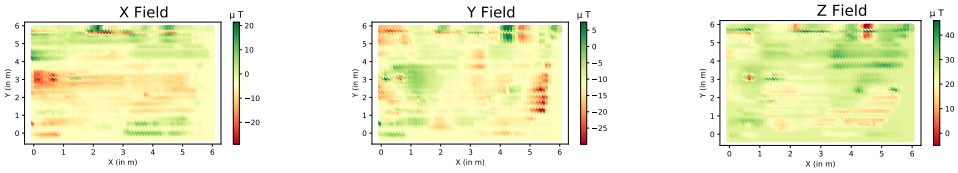


Figure 40: Magnetic Field Data Collected



(a) X Field **(b)** Y Field **(c)** Z Field

Figure 41: TruckLab Predicted Magnetic Field on Map using Individual Axis Model



(a) X Field **(b)** Y Field **(c)** Z Field

Figure 42: TruckLab Predicted Magnetic Field on Map using Joint Axis Model

All code listings and individual components of the DT developed in this project are provided in a github repository at the following link: *https://github.com/SaharshB/ThesisF*.

5 Evaluation

Having completed the development of the various aspects of the project and observed the results, the next step is to evaluate the performance of the various aspects. Doing so not only helps find the feasibility of the proposed solution, but will also help in answering the research questions proposed in this project.

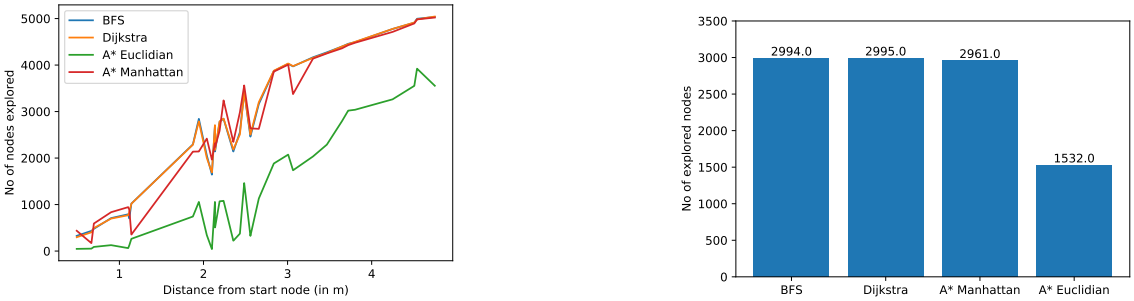
5.1 Evaluation of path planning

The first aspect to be evaluated is the implemented path planning module. Evaluation for this is important as it helps understand the physical requirements that these algorithms impose on the device(SBC) that they are deployed on. Additionally we also try to evaluate the time that it takes each of the implemented algorithms as getting a prompt path is also an important requirement.

All the data presented in this subsection is presented as an average of 10 repeatability tests performed on the same experiment. This was done to try to remove any outliers in data which may have been caused by certain unexpected occurrences such as a lag in the processing of the utilized computing device.

5.1.1 Memory Requirement

The first parameter to be evaluated is the amount of memory required to calculate the path. However, measuring the amount of memory(storage space and RAM) is not easy as Matlab does not have a method to accurately measure memory usage during a function execution. As a result a compromise evaluation has to be performed. This will be done by measuring the amount of available memory at the start of the function execution and the amount of available memory at the end of the function execution. Doing so gives an approximate on the amount of memory consumed during the function. It however, does not account for any memory used and released during the execution of the function. Even so, measuring the memory usage in this case gives a crude understanding about the storage requirements for the execution of the algorithms. In addition to this in order to understand the computation requirements for the algorithms, a comparable approximation could be made from observing the number of nodes explored by each algorithm before the optimal path is returned. Although the number of nodes explored is not a reasonable indicator of comparative memory consumption, as some algorithms may have other memory storage requirements other than maintaining the stack of explored nodes. For example, the A* algorithm needs to keep track of the cost value of each node(both g and h), which the BFS does not have to, which leads to more memory being consumed by A* compared to BFS. However, if a comparative relationship between the amount of memory consumed and the number of nodes explored can be found for each algorithm, it can be used to have an approximate understanding of the memory requirement in a different scenario. This helps in getting an understanding of the scalability of the algorithms which is extremely important should it be deployed on a SBC, with extremely limited available memory.



(a) Trend of explored nodes over distance

(b) Average number of explored nodes

Figure 43: Number of explored nodes in Path Finding Algorithms

Figure 43 shows the number of nodes explored by each algorithm before a path was determined. Looking at Figure 43a we can see that *BFS* and *Dijkstra* behave in nearly the same manner in this respect. This is the expected outcome as in a graph with unweighted edges *Dijkstra* is essentially the same as the *BFS* algorithm. However, the main change in number of explored nodes can be seen in the *A** algorithm. While the manhattan heuristic does not change the explored nodes number by a lot, implementing the euclidian heuristic reduces the number of nodes explored by nearly half. This general response can be verified from the data presented in Figure 43b. This can be attributed to that the euclidian considers the shortest straight line distance, whereas manhattan considers the shortest distance by measuring the distance along both axis. Given that the straight line distance is shorter than the distance calculated by considering both axes' separately, it is an acceptable result that the Euclidian heuristic needs to explore fewer nodes than the Manhattan heuristic.

This assumption is however not supported by the data represented in Figure 44. It shows that the amount of memory used remains comparable across the 2 heuristics implemented for the A* algorithm. Additionally the amount of memory used by BFS and Dijkstra seem comparable and seem to hold the same trend as well, which was an expected outcome due to reasons mentioned earlier. Figure 45, shows the average amount of memory used by each of the implemented algorithms. Using that we can see that the amount of memory used by both BFS and Dijkstra is nearly comparable, but it is less than half of that utilized by the A* algorithms. This could be attributed to the requirement of the A* algorithm to calculate the heurestics. The heuristic is effectively an array containing the cost to the destination from the current path. The amount of memory consumed by the A* algorithm would be subject to the size of the heurestic array which in turn is dependant on the number of size of the grid being used[3]. The memory consumption of the A* algorithm can therefore be considered to be scalable in $\mathcal{O}(n)$, but it would still be more than the amount of memory consumed by the BFS and Dijkstra algorithms due to their lack of any such heuristic matrix.

[3]In the experiments run in this project the size of the grid is 103×117

(a) BFS and Dijkstra **(b)** A* Algorithms

Figure 44: Memory Consumption for path planning algorithms

(a) BFS and Dijkstra **(b)** A* Algorithms

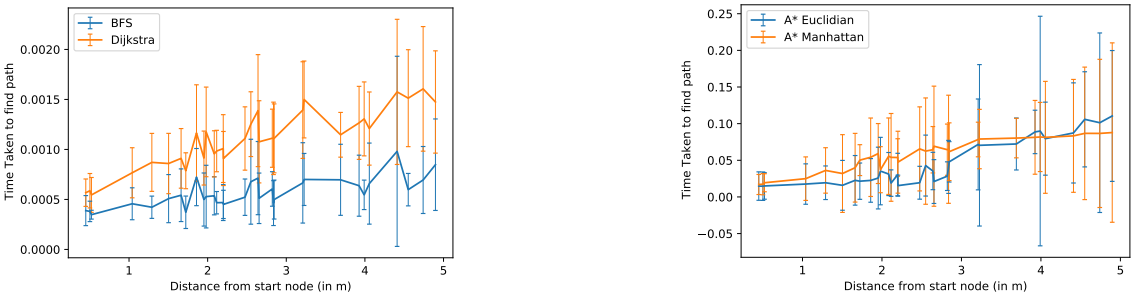
Figure 45: Average memory consumption per algorithm

5.1.2 Timing Characteristics

Having considered the memory requirements of the path planning algorithms we now consider their timing characteristics. It will help evaluate the response times and determine the fastest potential algorithm to get a solution.

Figure 46 shows the general trend of time consumption for each implemented algorithm. The time mentioned in the figure is in seconds. Here we see that both BFS and Dijkstra are multiple time faster than the A* algorithms. There could be multitude of reasons for this. The first being that each time the A* algorithm is run, a new heuristic array needs to be computed. This is done, because the heuristic matrix depends on the destination node. Given that the destination would most likely change at every iteration a new heuristic matrix needs to be calculated every time the algorithm is run. After running separate experiments to estimate the time taken to calculate the heuristics, it was found that the average time taken to calculate the heuristic matrix is approximately 2 ms. This itself seems to be the same amount of time taken to calculate the entire path using either BFS or Dijkstra. The next reason which is in all probability the more dominant reason is the inefficiency of the written code. The comparisons here are made between an inbuilt function(*shortestpath*) to find the BFS and Dijkstra paths, whereas a custom code was written for the A* algorithm. This custom code upon reflection was not optimized for timing performance rather, it was written for the accuracy of the solution. The current implementation of the A* algorithm works in a sequential manner on the occupancy grid of the region. Each element of the grid is then represented as a structure which keeps track if the vertex is explored, the score to reach the vertex and the cost from the current vertex to the destination. Additionally, there are certain redundancy checks at stages to ensure that a path even exists. All these factors could potentially contribute in slowing the result than what the actual algorithm would take.

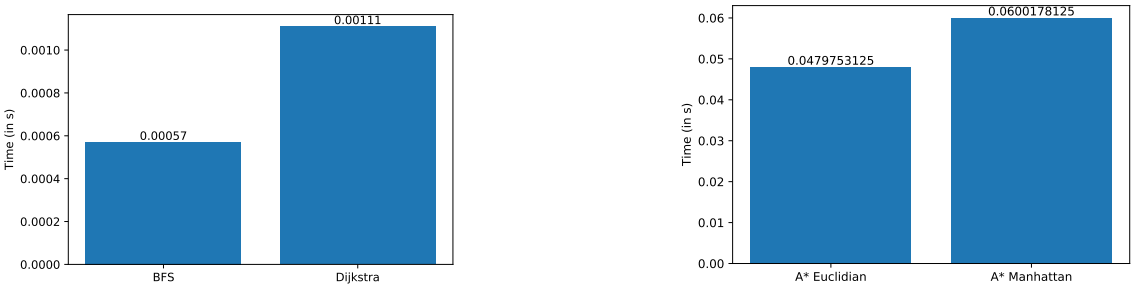
The data in Figure 47, shows that the average time taken by both BFS and Dijkstra is comparable and extremely low. Whereas the time taken by the 2 A* implementations are comparable to each other. Additionally we can see that the heuristic used has quite an effect on the time taken.



(a) BFS and Dijkstra **(b)** A* Algorithms

Figure 46: Time taken for path finding

Between the Manhattan and the Euclidian heuristics we can see an almost 20% improvement in timing performance of the latter.



(a) BFS and Dijkstra **(b)** A* Algorithms

Figure 47: Average Time taken for path finding

From the results as seen in Figure 46, it can be seen that as the amount of distance increases(causing an increase in the number of explored nodes), each algorithm scales differently with respect to the amount of time taken. BFS scales at a rate of approximately 2.5, whereas Dijkstra scales at a rate of nearly 3. The A* algorithms scale at a rate of ≈ 5. This indicates that BFS is the fastest algorithm in this situation, where the graph considered is unweighted.

5.2 Evaluation of magnetic field extrapolation

Having presented the results of the evaluation of the path planning algorithms, the next step is to evaluate the results magnetic field extrapolation methods as seen in Section 4.3. The data presented in this subsection is presented as an average of 5 repeated tests performed on the same experiment similar to the manner of the previous section.
In this section we evaluate the performance of both the individual axis models as well as the joint axis model developed, as shown in paragraph 3.4.2.2. In order to evaluate the performances of the developed models, we use the MSE(mean squared error)metric. The MSE score tells us about the accuracy of the fit of the model. It accounts for any outliers in the data and reports accordingly. Doing so helps us put a quantitaive perspective on the accuracy of the models developed.

5.2.1 Individual Axis Model

The first models evaluated are the ones developed individually for each axis separately. Figure 48, shows the results of the repeatability experiments performed on the models for individual field regression for varying training sizes. Looking at the range of MSE scores we see that the X and Y

field scores lie in the range of 0.2 to 0.5. The Z field score lies in the range of 2 to 3. The range of values predicted(the magnetic field values) is ± 50 μT. Considering this range of values the MSE scores obtained are fairly well and can be considered to be a good result.

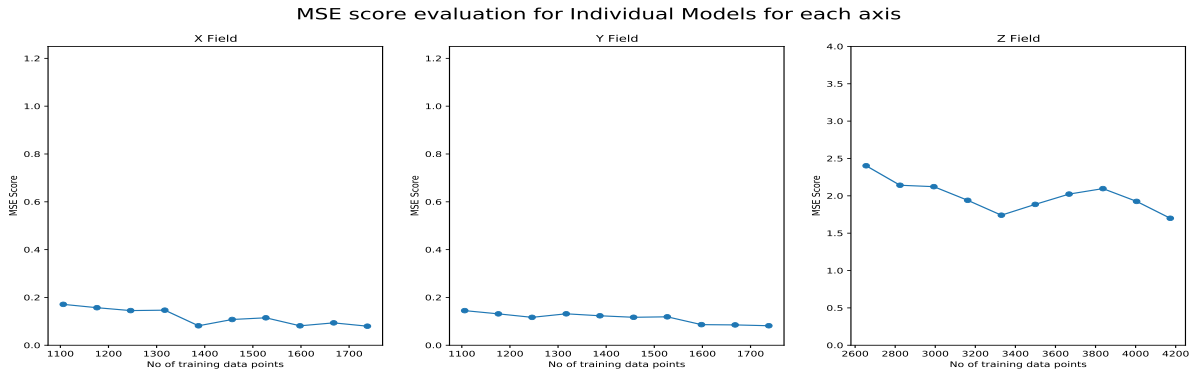


Figure 48: MSE Score for Individual Axis Model

Additionally observing the trend of the score we can see that increasing the number of training points does increase the performance of the models by reducing the MSE score.

5.2.2 Joint Model

We now move onto evaluating the mode developed for all the 3 axes together. The results are depicted in Figure 49. Similar to the results of the previous model, the MSE scores lie in a range which represents a positive result considering the range of values which it is predicting. Additionally the results are slightly better than that of the individual axis models for the Y axis, whereas for the Z axis it is much better. However this could be attributed to the fewer train/test points.

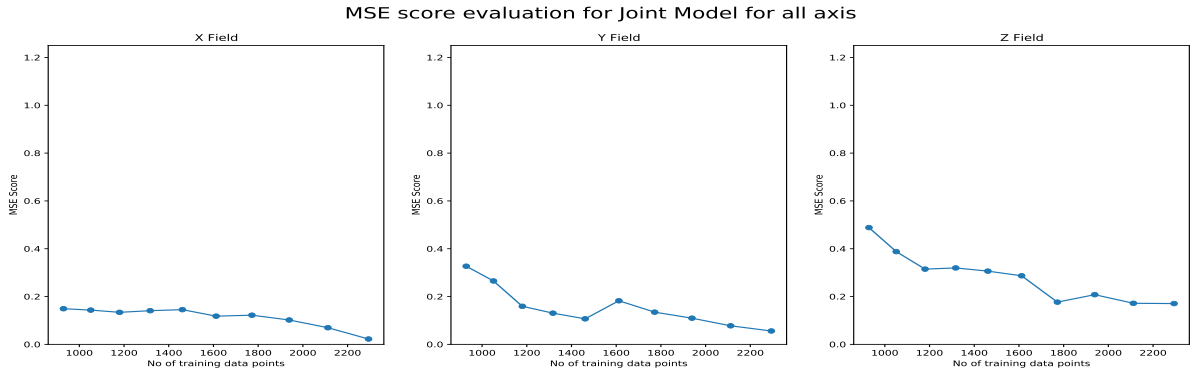


Figure 49: MSE Score for joint Model

5.3 Assessment of magnetic field extrapolation

In this section we evaluate the system requirements of the magnetic field regression models. Considering this data will help determine the possibility of deploying this solution on a low resource device such as the on-board raspberry pi SBC of the TB3.
The first parameter measured is the amount of memory taken to train the model. The results of the tests can be seen in Figure 50. From the figure a clear relationship can be seen between the number of training data points and the amount of memory taken. It is a relatively linear relationship, which seems to hae a scalability of $\mathcal{O}(n)$, with n being the number of training data

points. Further conclusions about the feasibility of implementing this solution will be discussed in Section 6

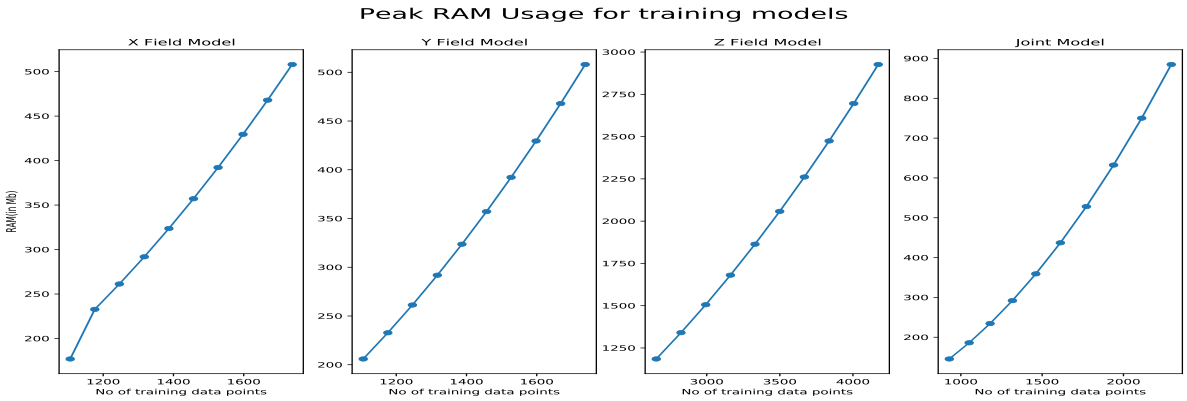


Figure 50: Peak RAM taken to fit Gaussian Regression Model

6 Discussion

Having developed the project and presented its results, a brief reflection upon the project as a whole and the answers to the proposed research questions is the next step. The assumptions/limitations made during this project are also considered along with mentioning the potential for any future development on the project.

6.1 Research Questions

In this section, the proposed research questions are considered and answered based upon the findings made during the development and evaluation of this project.

6.1.1 RQ1 How to map magnetic fields of a region on a low resource device?

Performing the magnetic field extrapolation in this project is done based on the predictions from a Gaussian Regression Model. Therefore the deciding factor on determining the feasability of magnetic field mapping on any device would be dependent on the feasibility of forming the regression model and training it on the training data points on said device. The factors in consideration is primarily the amount of RAM used over the course of the fitting of the model. From the results seen in Figure 50, the amount of memory consumed is dependent on the number of training data points. As can be seen a relatively small space, which has an area of around 20 m^2, there can be more than a 2000 training points. Given the range of RAM usage which can be seen, especially in the Z field model, which is more than 3Gb, it will be difficult to replicate this solution on a device like the RaspberryPi, which itself has a maximum of 1Gb RAM available, it would not be advisable to export this solution to the onboard SBC. This can be additionally verified with the results of extrapolating the field for the TruckLab environment. In that scenario, the number of training data points is approximately 10,000 and the peak RAM usage was around 13Gb.
Based on the results observed in this project the optimal solution for mapping the magnetic field of a region would be the method employed here, i.e. collect the necessary data and export it to a external device with more memory and computation capabilities to perform the map extrapolation. Once the map is formed the results can be transferred to the device back in the form of a map/array which can then be used to identify locations which can subsequently be used for the path planning.

6.1.2 RQ2 Which navigation algorithm is optimal in an indoor environment?

SRQ2_1 Which algorithm has the least mean error in directing to target location?
The method in which the path is calculated is modelling the entire space as a grid, wherein each cell in the grid is representative of a certain cross section in the area. The resolution of the grid is an important aspect here(the area of each cell in the grid). Having a grid with cells having too large of an area will lead to a slower solution as it increases the number of searchable nodes, however it can also lead to a more exact and accurate path to exact desired point. However, keeping the resolution of the grid constant the accuracy of finding the destination, should a viable path exist to it from the current location, is relatively constant across all considered algorithms. Therefore for this point all considered algorithms have relatively the same performance.

SRQ2_2 Which algorithm yields the path in shortest amount of time?
Section 5.1.2 discusses the results of the experiments to determine the timing characteristics of the path finding algorithms. It also briefly tries to explain any shortcomings of the implementation of the A* algorithms. Taking these findings in consideration it can be clearly seen that BFS is the algorithm which yields the path in the shortest amount of time. Additionally looking at the trend that the data takes, it can also be concluded that BFS scales most optimally(least amount of increase in time, with an increase in distance). However, it should be noted that these results are subject to the algorithms being implemented on an unweighted graph. Should a different mapping technique be employed, which yields a weighted edges graph, the results of this project would not

hold in that scenario and new tests would need to be performed.

SRQ2_3 Which algorithm utilizes the least resources to find the solution?
The main resources we concern ourselves with, to answer this question are the RAM usage and the amount of memory used. Concerning the RAM the test results of the test performed to measure the RAM as shown in Section 4. The amount of RAM used remains rather the same across all the algorithms. The results of the test for memory usage are depicted in Figure 44. As can be seen the amount of memory used is highest in A* Manhattan. Similar to the results of the timing characteristics, we see that the amount of memory usage is the least for BFS, despite it being the algorithm which explores the most number of nodes.

Considering the 3 parameters taken into account when evaluating the different path finding algorithms, it can be seen that BFS, appears to be the most efficient algorithm in terms of both time required and memory taken. However, it should be noted again that these findings are subject to an unweighted graph. With respect to the accuracy of the algorithms, all have a similar performance with respect to the accuracy of the solution, although is it mainly dependant on the accuracy of the mapping technique rather than the algorithm itself. It can therefore be concluded that the optimal algorithm for path finding in an indoor environment is BFS.

6.1.3 RQ3 How feasible is it to use both magnetic fields and LIDAR for indoor navigation?

Having developed the magnetic field map of the region on all 3 axis and generated the physical obstacle map using the LIDAR, we can now assess if using the data from these 2 maps in conjunction is a reasonable combination for path planning and navigation. As reported by Gozick et al. [11], the magnetic field in an indoor environment remains relatively stable across a time period of atleast 6 months. Considering this we can take that the map formed for the magnetic fields will be constant and this satisfies the assumption that the data in the extrapolated magnetic field map will remain accurate at a later time. Hence using the magnetic field map as a source of identifying target locations is possible. Generating the magnetic field map, as seen in paragraph 3.4.2.2 requires a map of the physical region as well. The coordinate data from the physical map is used to predict the results of the Gaussian model trained on the magnetic field data collected. This physical coordinate map is provided by the data from the LIDAR scans of the region. This further demonstrates the feasability of using the magnetometer and the LIDAR in conjunction with each other. However, using the magnetic field map, we are unable to visualize the physical obstacles in the path. This would be an integral part when considering the movement of the bot from the source to destination. This detection can be performed using the LIDAR scan data in real time, as shown in Section 3.4.4. Seeing the results of this project it can be concluded that using a conjunction of both the data from the magnetometer and the physical map generated using the LIDAR, a system for controlling the navigation in an indoor environment can be developed. The extrapolated magnetic field map, provides a magnetic field value for each axis at every point in the physical space. These values can be used to identify the coordinates of a destination based on the characteristics of the magnetic field values. With the coordinates for the destination identified the data from the LIDAR can be used in real time for avoiding collisions with physical obstacles which may be on the calculated path. It can therefore be concluded that using both magnetic fields and a live scan of the LIDAR, is a viable combination of data for indoor navigation.
The accuracy of the developed map is a concern while answering this question. For this project it is assumed that the map generated during a previous time is still relevant at the time of path planning, and is not inaccurate due to any external disturbances.

6.2 Project Constraints

While we have displayed the results of the projects and derived our conclusions from it, we now discuss the constraints of this project.

Firstly the movement control of the path planning section of the project is dependant on the values read from the */odom* topic. While paragraph 3.4.3.1, covers the positional localization of the bot and its accuracy, it does not touch upon the rotational accuracy of the TB3. The rotational values are calculated from the orientation quaternion in the *odom* topic. The values in the topic are provided by the onboard IMU, gyroscope. Thus making it susceptible to IMU drift errors. This leads to potential errors in rotation angle calculation if the bot is moved at high speeds(experimentally it was observed speeds > 0.08m/s). The alternative solution of the *amcl* method as discussed earlier has a data rate too low to be of any viable use.
Another limitation concerns the magnetic field data collection. As mentioned in paragraph 3.4.2.1, the X and Y field magnetic field values are dependant on the orientation of the sensor. As a result, the solution employed in this project was to read the magnetic field values for the X and Y axis only along a specified orientation. This limits the amount of training data points available for those axes'. In addition to this the model is also limited in detecting any patterns along the axis perpendicular to the direction in which the values were recorded. To address this issue, a system was implemented to shift the magnetic field values from the current orientation to the expected orientation. However, rotating the bot has a separate problem of its own. The movement of the TB3 is controlled from the front end of the bot, whereas the magnetometer is placed at the back end of the bot. The distance between the wheel and the magnetometer is approximately 225mm. Whereas the distance from the magnetometer to the centre of the bot is approximately 100mm. Due to this offset in position, rotating the bot, causes the X-Y position of the magnetometer to also change. Therefore, while rotating the magnetic field values using the angle of rotation(yaw angle), leads to an accurate field reading in the expected orientation, the position of the sensor is offset from the reading reported by the *odom* topic. This makes reading data of the X and Y magnetic fields when the TB3 is rotated from its expected orientation, not feasible presently.

7 Conclusions

Digital Twin technology helps us visualise, understand, simulate and develop complex behaviour of products or services. Therefore, having a Digital Twin of an indoor navigation system, allows a platform to test various algorithms, and validate their accuracy. In addition to this developing each service as individual models allows translating them to other projects such as the one being developed by Ayush Maheshwari.

The first aspect of this project was the conceptual model of the digital twin. As discussed earlier the model used in this digital twin is the 5 component architecture as explained in Section 2. Using this model allows for more flexibility and abstraction about the development of various aspects of the project. Doing it so, also aids in the portability of the individual features to different projects or systems. While the portability is an important feature of the abstraction followed here, it also plays a key role in maintainability and scalability of this project.

The purpose of this project was to demonstrate a system which can be scaled up to any other device with minor changes, such as a new map of the region or adjusting the speed of the bot as per requirement. In this project we were able to successfully map a region and visualize the magnetic field map of the region at the same time. In addition to this the system also controls the motion of a robot(TB3 in this case), and accurately navigate it to a desired location. For the purpose of the navigation and path planning, we treat the TB3 as a point model which can freely move about in a 2D space, and is not limited by multi-body dynamics thereby freely allowing it to make even the sharpest of movements(a complete 360° rotation about its base link axis). Doing so allows the TB3 to follow the paths determined by the algorithms used in this project as shown in Section 3.4.3.

The magnetic field mapping developed in this project was able to successfully develop an accurate magnetic field map of the concerned indoor region. However, this mapping technique is subject to the accuracy of not only the field data, but also on the accuracy of the positional and rotational data of the magnetometer. As discussed above, for the path planning part of this project we treat the TB3 as a point model. However, for the data collection for the magnetic field mapping, the point model may lead to data spillage as there is an offset in the position of the magnetometer to the point whose position is returned using the localization method. Therefore the positional translation becomes imperative.
While this project covers a wide range of topics, it is by no means the end. There are various features and services which could be added.
Firstly the main focus of this project was to develop the Ss of the proposed digital twin model. There can be plenty of features which can be added to the VE model. An accurate physical model of the TB3, with proper physics engines which accurately predict the movement of not just the robot itself, but also accommodate various geographical and physical features could be developed. This would introduce a new source of data form the VE and not be reliant on just the PE, which would allow for proper simulations to take place. Adding these models to the VE, would further help validate the performance and data received from the sensors in the PE.

This project can be extended to accommodate for various additional features. Some of which are listed below.

- Adding physical models of the actuators and sensors to the VE. Causing the VE to not be limited to being a mirror of the data generated from the PE. It would also allow to testing any new algorithms or services purely on the VE. This would help save many resources in the future, as the simulations on the VE, could be sped up and we could obtain faster results. Additionally it would also bypass the need of any physical devices, thereby making testing in any environment much easier without access to any of the actual components.

- Implement a magnetic field extrapolation method which learns the field in real time as the bot moves around in the environment, rather than collecting the data and then performing

post-processing on it at a later time, as is done in this project.

- A more efficient localization method which is not susceptible to IMU drift which may cause errors in determining the rotation angles. A system such as the OptiTrack mechanism which is currently setup in the TruckLab environment at TU/e. This would ensure accurate location and orientation values of the bot, and would also overcome the shortcoming of having the location being reset to (0,0) every time the bot is rebooted.
- While this project provides a path planning mechanism in the provided space, the path yielded, consists of only straight lines with right angle turns. Another potential avenue of future work would be to implement a more efficient path planning algorithm which accounts for turning the bot in a circular motion, like an actual car, whenever the direction of motion needs to be changed. This would lead to the bot reaching its destination faster, not because a shorter path was established, but because now the bot does not have to stop and rotate. It can simply make the appropriate circular turn.
- Additional sensors such as a camera can be integrated onto the bot, which can be used to form a visual map of the region. A camera could also be used to detect and identify certain objects, which could then be scaled up to a object tracking system.

Bibliography

[1] Mag658 Magnetometer. URL https://www.bartington.com/mag658/.

[2] SYSMOD. URL https://mbse4u.com/sysmod/.

[3] TRIZ 40 Principles. URL https://triz.org/triz/principles.

[4] TurtleBot3 Waffle Pi, . URL https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.

[5] TurtleBot Navigation, . URL https://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/#run-navigation-nodes.

[6] WHO Magnetic Field Health Effects. URL https://www.who.int/peh-emf/publications/facts/fs299/en/#:~:text=HEALTHEFFECTS&text=Staticmagneticfieldsexertforces,impedetheflowofblood.

[7] M. Angermann, M. Frassl, M. Doniec, B. J. Julian, and P. Robertson. Characterization of the indoor magnetic field for applications in Localization and Mapping. *2012 International Conference on Indoor Positioning and Indoor Navigation, IPIN 2012 - Conference Proceedings*, (November):13–15, 2012. doi: 10.1109/IPIN.2012.6418864.

[8] R. F. Brena, J. P. García-Vázquez, C. E. Galván-Tejada, D. Muñoz-Rodriguez, C. Vargas-Rosales, and J. Fangmeyer. Evolution of Indoor Positioning Technologies: A Survey. *Journal of Sensors*, 2017, 2017. ISSN 16877268. doi: 10.1155/2017/2630413.

[9] R. P. Busch. Developing a Digital Twin of 'TurtleBot3 Waffle Pi' Robot in Unity Game Engine With Integration and Visualisation of RM3100 Geomagnetic Sensor. (October), 2020.

[10] N. I. o. P. H. Environment and the. Comparison of International Policies on Electromagnetic Fields. URL https://rivm.openrepository.com/handle/10029/623629.

[11] B. Gozick, K. P. Subbu, R. Dantu, and T. Maeshiro. Magnetic maps for indoor navigation. *IEEE Transactions on Instrumentation and Measurement*, 60(12):3883–3891, 2011. ISSN 00189456. doi: 10.1109/TIM.2011.2147690.

[12] B. Gozick, K. P. Subbu, R. Dantu, and T. Maeshiro. Magnetic maps for indoor navigation. *IEEE Transactions on Instrumentation and Measurement*, 60(12):3883–3891, 2011. ISSN 00189456. doi: 10.1109/TIM.2011.2147690.

[13] L. Kheifets, M. Repacholi, R. Saunders, and E. Van Deventer. The sensitivity of children to electromagnetic fields. *Pediatrics*, 116(2), 2005. ISSN 00314005. doi: 10.1542/peds.2004-2541.

[14] D. Laparra. Pathfinding Algorithms in Graphs and Applications. 2019. URL http://hdl.handle.net/2445/140466.

[15] H. Li and L. Zhijian. The study and implementation of mobile GPS navigation system based on Google Maps. *Proceedings of ICCIA 2010 - 2010 International Conference on Computer and Information Application*, pages 87–90, 2010. doi: 10.1109/ICCIA.2010.6141544.

[16] K. Malarvizhi, S. V. Kumar, and P. Porchelvan. Use of High Resolution Google Earth Satellite Imagery in Landuse Map Preparation for Urban Related Applications. *Procedia Technology*, 24:1835–1842, 2016. ISSN 22120173. doi: 10.1016/j.protcy.2016.05.231. URL http://dx.doi.org/10.1016/j.protcy.2016.05.231.

[17] NIST CPS PWG. Framework for Cyber-Physical Systems - v1.0. (May), 2016.

[18] RIVM. National precautionary policies on magnetic fields from power lines in. 2017. doi: 10.21945/RIVM-2017-0118R.

[19] A. Solin, M. Kok, N. Wahlstrom, T. B. Schon, and S. Sarkka. Modeling and Interpolation of the Ambient Magnetic Field by Gaussian Processes. *IEEE Transactions on Robotics*, 34(4): 1112–1127, 2018. ISSN 15523098. doi: 10.1109/TRO.2018.2830326.

[20] Y. Stein and I. G. Udasin. Electromagnetic hypersensitivity (EHS, microwave syndrome) – Review of mechanisms. *Environmental Research*, 186(August 2018):109445, 2020. ISSN 10960953. doi: 10.1016/j.envres.2020.109445. URL `https://doi.org/10.1016/j.envres.2020.109445`.

[21] S. Ueno. Biological Effects of Magnetic Fields. *IEEE Translation Journal on Magnetics in Japan*, 7(7):580–585, 1992. ISSN 08824959. doi: 10.1109/TJMJ.1992.4565451.

[22] N. Wahlstrom, M. Kok, T. B. Schon, and F. Gustafsson. Modeling magnetic fields using Gaussian processes. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 3522–3526, 2013. ISSN 15206149. doi: 10.1109/ICASSP.2013.6638313.

[23] C. Wu, Y. Zhou, M. V. Pereia Pessôa, Q. Peng, and R. Tan. Conceptual digital twin modeling based on an integrated five-dimensional framework and TRIZ function model. *Journal of Manufacturing Systems*, 58(October 2019):79–93, 2021. ISSN 02786125. doi: 10.1016/j.jmsy.2020.07.006. URL `https://doi.org/10.1016/j.jmsy.2020.07.006`.

[24] S. C. Yeh, W. H. Hsu, W. Y. Lin, and Y. F. Wu. Study on an Indoor Positioning System Using Earth's Magnetic Field. *IEEE Transactions on Instrumentation and Measurement*, 69 (3):865–872, 2020. ISSN 15579662. doi: 10.1109/TIM.2019.2905750.

A SYSMOD Steps

A.1 Requirements Analysis

Display Requirements

ID	*Name*	*Description*
DS	Display requirements	The system should be able to display the required data in an interpretable manner
DS1	Functional Requirements	The system should be able to display functional data
DS11	Display Magnetic Map	The system should be able to display the extrapolated magnetic field map of the region
DS12	Display Physical Map	The system should be able to display the 2D top view physical obstacle map of the region
DS2	Extra Requirements	The system should be able to display interesting features in a fun manner
DS21	Display Predicted Path	The system should be able to display the path to be taken by the bot
DS3	Assessment Requirements	The system should be able to display data which helps in the assessment of the system
DS31	Display Path Calc Mem	The system should be able to display the amount of memory taken to calculate the path
DS32	Display Path Calc Time	The system should be able to display the amount of time taken to calculate the path

Table 5: Requirements Table: Display

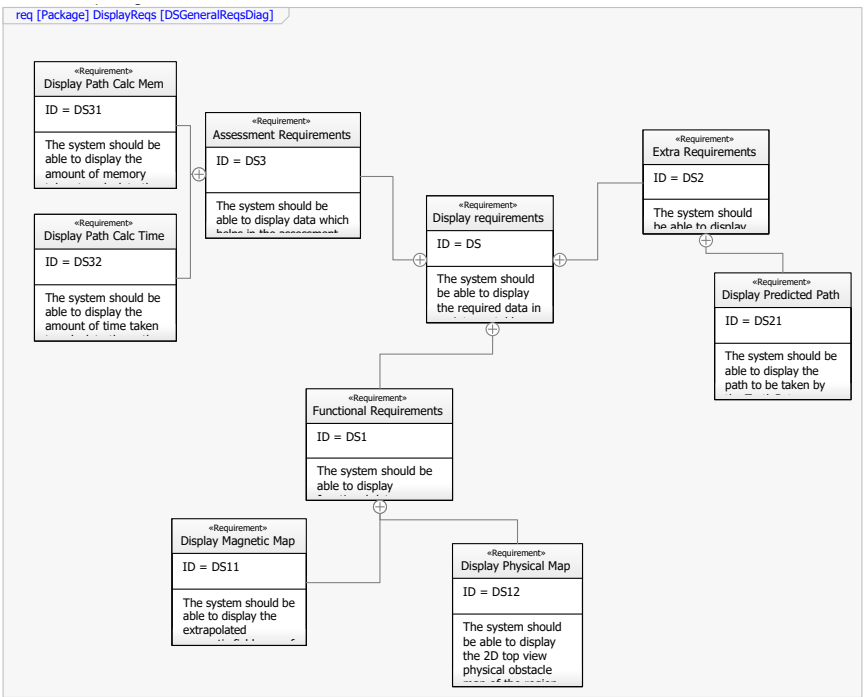


Figure 51: Requirement Diagram: Display

Magnetic Mapping Requirements

ID	*Name*	*Description*
MM	Magnetic Map Requirements	The system should be able to estimate the magnetic field of the region
MM1	Functional Requirements	The system should be able to accurately extrapolate the magnetic field of the region
MM11	Extrapolate Mag Field Joint Model	The system should be able to extrapolate the field of the region using a single, joint model
MM12	Extrapolate Mag Field X-axis	The system should be able to extrapolate the magnetic field of the region along the x-axis using individual model
MM13	Extrapolate Mag Field Y-axis	The system should be able to extrapolate the magnetic field of the region along the y-axis using individual model
MM14	Extrapolate Mag Field Z-axis	The system should be able to extrapolate the magnetic field of the region along the z-axis using individual model
MM2	Verification Requirements	The system should be able to verify the extrapolated magnetic field
MM21	Inst Mag Field	The system should be able to read the instantaneous magnetic field values at the current position

Table 6: Requirements Table: Magnetic Mapping

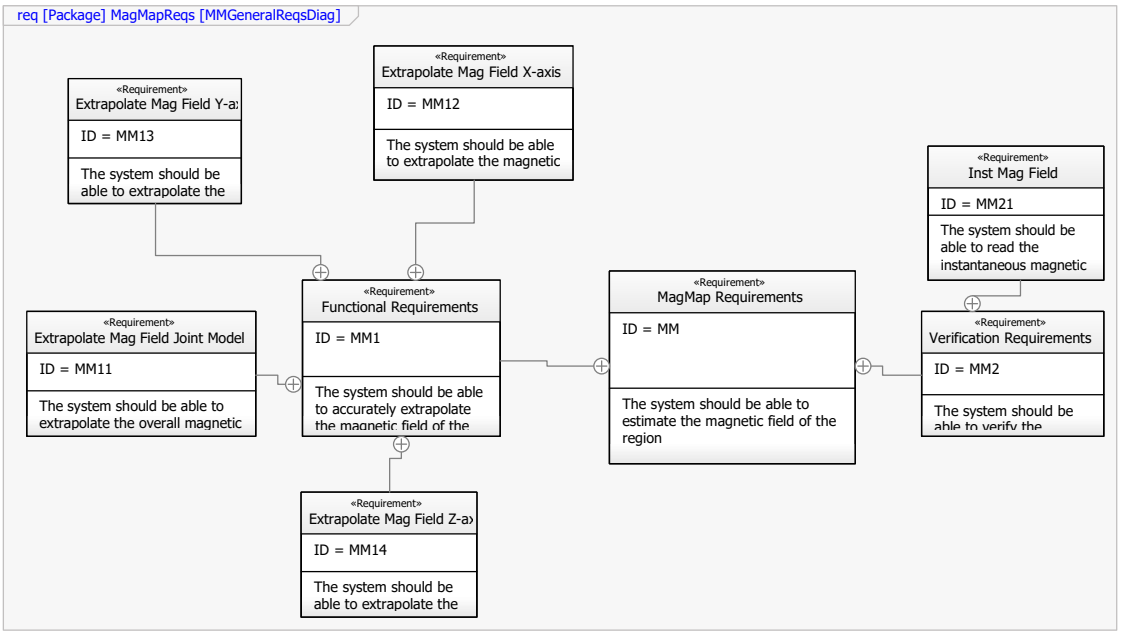


Figure 52: Requirement Diagram: Magnetic Mapping

Movement Requirements

ID	*Name*	*Description*
MV	Movement Requirements	The system should be able to control the movements of the bot
MV1	Functional Requirements	The system should be able to control the individual aspects of motion of the bot upon command
MV11	Change Direction	The system should be able to change the direction in which the bot is moving along a 360 degree axis
MV12	Decrease Speed	The system should be able to decrease the speed of the bot
MV13	Increase Speed	The system should be able to increase the speed of the bot
MV14	Move Straight	The system should be able to get the bot to move in a straight direction
MV15	Stop	The system should be able to stop the motion of the bot
MV16	Turn Axis	The system should be able to rotate the bot around its central axis

Table 7: Requirements Table: Movement

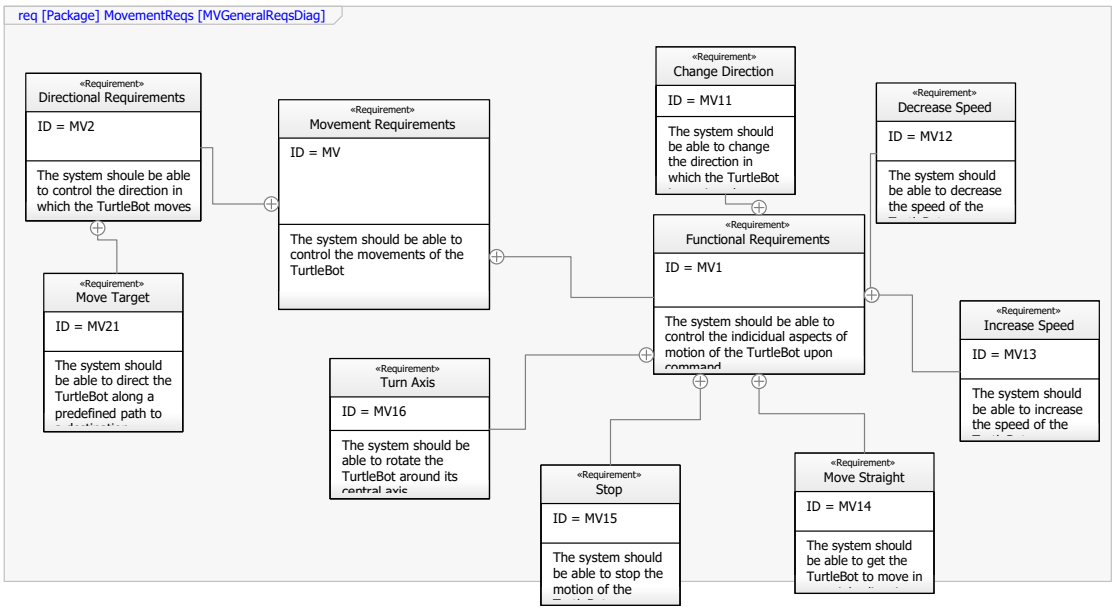


Figure 53: Requirement Diagram: Movement

Path Planning Requirements

ID	*Name*	*Description*
PC	Path Calculation Requirements	The system should be able to calculate a path for the bot to follow
PC1	Functional Requirements	The system should be able to calculate paths using various algorithms
PC11	Calculate Path Ast	The system should be able to calculate the path using A* algorithm
PC12	Calculate Path Djk	The system should be able to calculate the path using Dijkstra algorithm
PC13	Calculate Path BFS	The system should be able to calculate the path using BFS algorithm
PC14	Identify Destination	The system should be ale to identify the coordinates of the final destination.
PC15	Identify Location	The system should be able to identify coordinates of its current position
PC2	Directional Requirements	The system should be able to control the direction in which the bot moves
PC21	Move Target	The system should be able to direct the bot along a predefined path to a destination

Table 8: Requirements Table: Path Planning

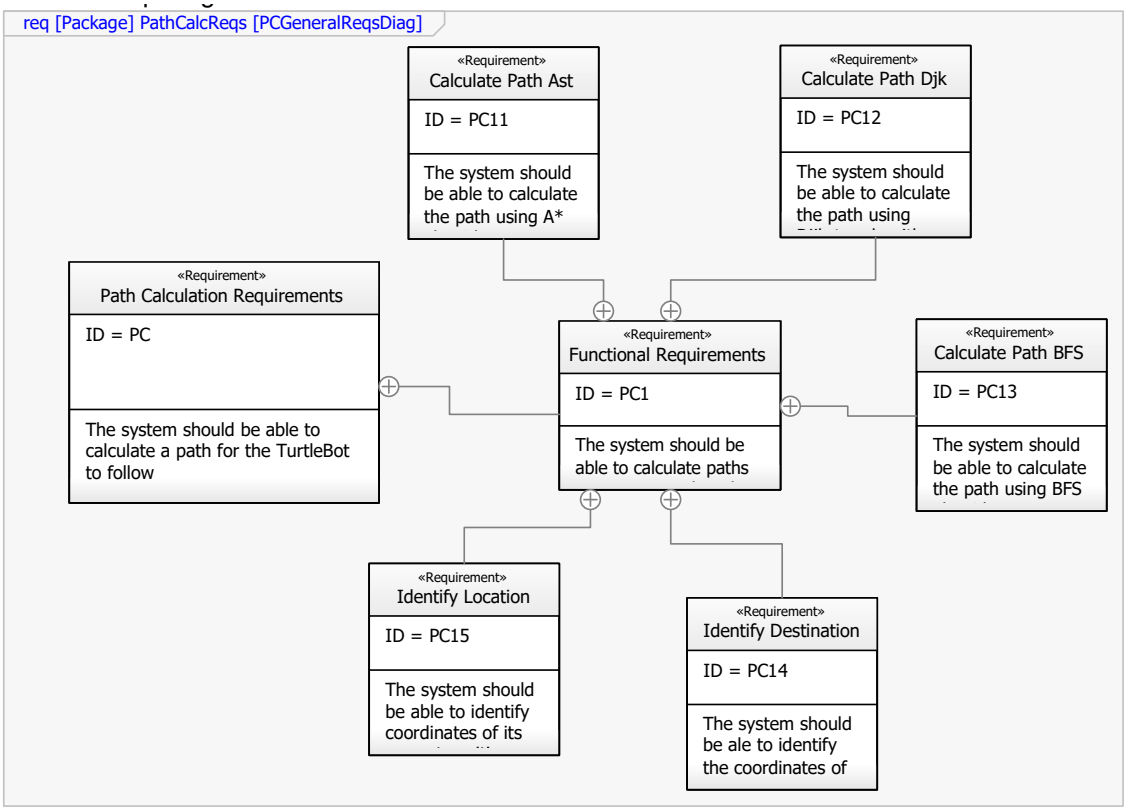
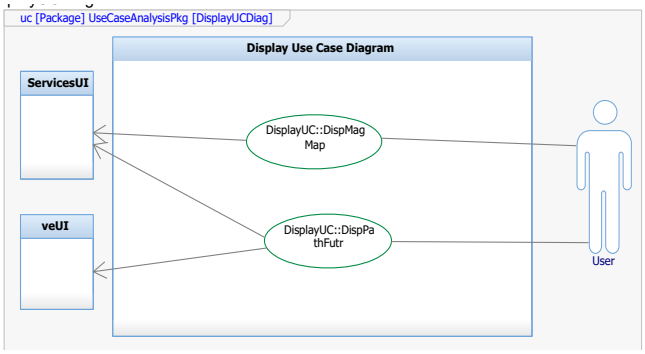
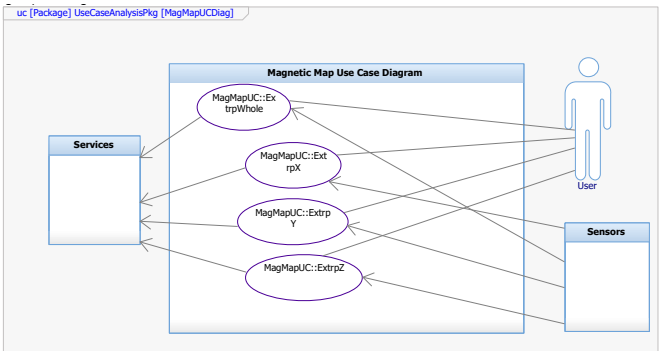


Figure 54: Requirement Diagram: Path Planning

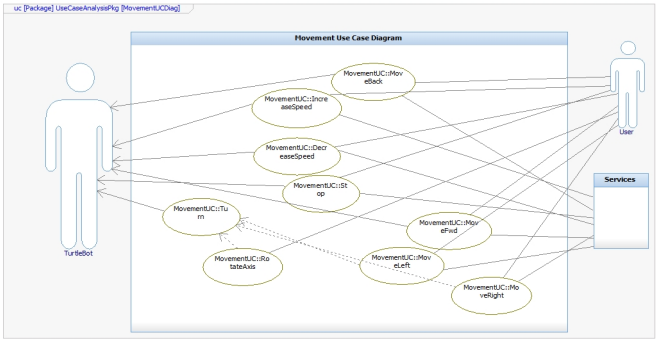
A.2 Use Cases



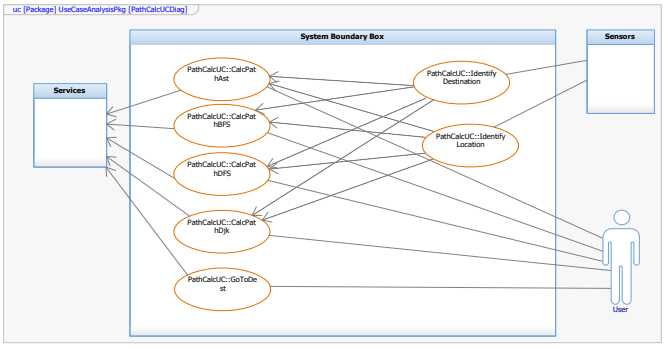
(a) Display UC



(b) Magnetic Mapping UC



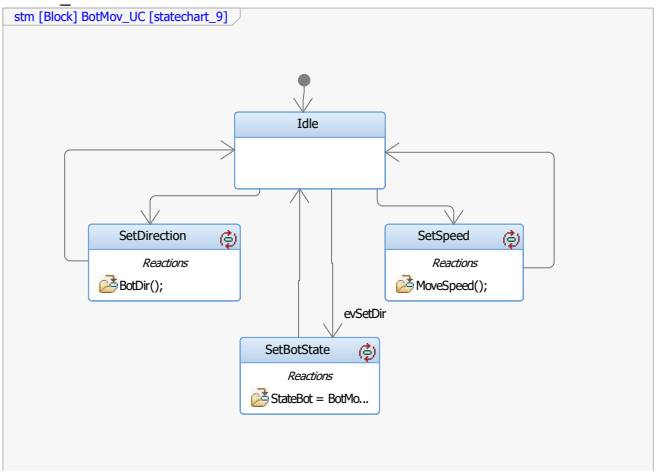
(c) Movement UC



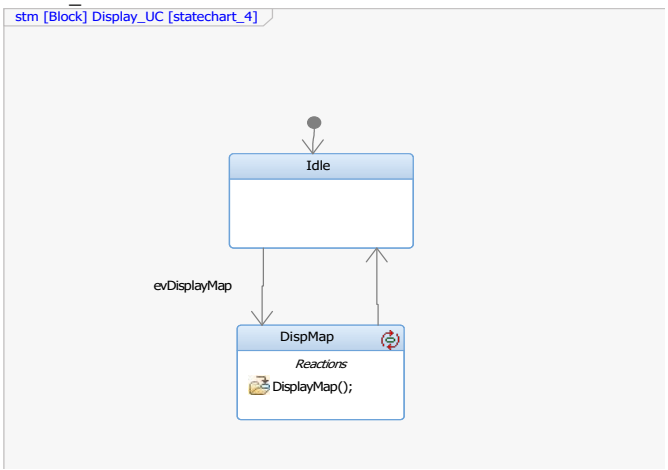
(d) Path Planning UC

Figure 55: Use Case Diagrams

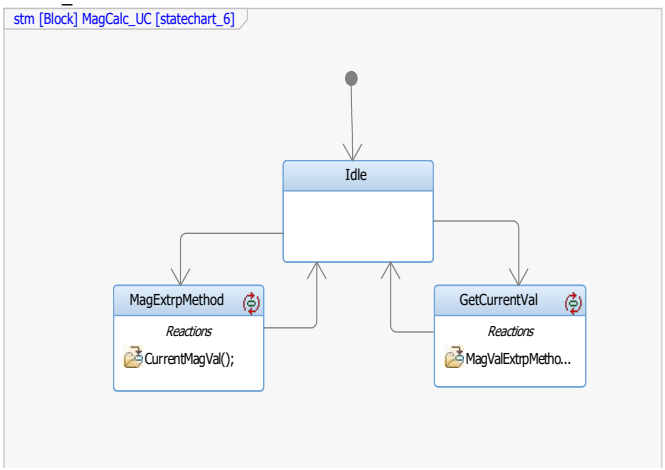
A.3 Behaviour Diagrams



(a) Movement State Chart



(b) Display State Chart



(c) Magnetic Mapping State Chart

Figure 56: Behaviour State Chart Diagrams

B Code Listings

B.1 Listings for the PE

B.1.1 Magnetometer Connections

Magnetometer Pin	Description	Magnetometer Pin	Description
SCK	RPi GPIO 2 (SDA)	*SDA*	RPi GPIO 3 (SCL)
SC	GND	*SSN*	GND
AVSS	GND	*DVSS*	GND
AVDD	+5V	*DVDD*	+5V
I2CEN	+5V		

Table 9: RM3100 Pinout

B.1.2 Code to read RM3100 Magnetometer readings

```
import rospy
from std_msgs.msg import String
from sensor_msgs.msg import MagneticField
from rm3100read import read_rm3100
import smbus, struct
from math import atan2, pi

def recast24to32(byte0,byte1,byte2):
    # pack 24 bits (3 bytes) into 32 bits byte-type
    b24 = struct.pack('xBBB',byte0,byte1,byte2)

    # unpack to unsigned long integer
    uL = struct.unpack('>L',b24)[0]

    # this is for 2's complement signed numbers -
    #   if negative assign sign bits for 32 bit case
    if (uL & 0x00800000):
        uL = uL | 0xFF000000

    # repack as 32 bit unsigned long byte-type
    packed = struct.pack('>L', uL)
    # unpack as 32 bit signed long integer
    unpacked = struct.unpack('>l', packed)[0]

    return unpacked

def read_rm3100():
    bus = smbus.SMBus(1)
    address = 0x20
    rm3100_POLL    = 0x00
    rm3100_CMM     = 0x01

    rm3100_Mx2w    = 0x24

    bus.write_byte_data(address, rm3100_POLL , 0x00)

    bus.write_byte_data(address, rm3100_CMM , 0b01111001)

    x=[]
    y=[]
    z=[]

```

```
43      x_coeff=[-14.45, 2219.57]
44      y_coeff=[-14.20, 7147.13]
45      z_coeff=[14.02,-20454.22]
46
47      for j in range(100):
48          raw=bus.read_i2c_block_data(address, rm3100_Mx2w, 9)
49          values=[]
50          for i in range(0, 9, 3):
51              data = float(recast24to32(raw[i],raw[i+1],raw[i+2]))
52              values.append(data)
53          x.append(values[0])
54          y.append(values[1])
55          z.append(values[2])
56      x_sum=0
57      y_sum=0
58      z_sum=0
59
60      for i in range(100):
61          x_sum=x_sum+x[i]
62          y_sum=y_sum+y[i]
63          z_sum=z_sum+z[i]
64      x_sum=x_sum/100
65      y_sum=y_sum/100
66      z_sum=z_sum/100
67
68      values=[(x_sum*x_coeff[0] + x_coeff[1]), (y_sum*y_coeff[0] + y_coeff[1]),
69      (z_sum*z_coeff[0] + z_coeff[1])]
70
71      return values
72
73  def mag_vals():
74      mag_msg=MagneticField()
75      pub = rospy.Publisher('magnetic_vals', MagneticField, queue_size=10)
76      rospy.init_node('mag_vals', anonymous=True)
77      rate = rospy.Rate(200) # 10hz
78      while not rospy.is_shutdown():
79          values=read_rm3100()
80          mag_msg.magnetic_field.x=values[0]*(10**-9)
81          mag_msg.magnetic_field.y=values[1]*(10**-9)
82          mag_msg.magnetic_field.z=values[2]*(10**-9)
83          heading=atan2(values[0],values[1])*180/pi
84          pub.publish(mag_msg)
85          rate.sleep()
86
87  if __name__ == '__main__':
88      try:
89          mag_vals()
90      except rospy.ROSInterruptException:
91          pass
```

B.2 Listings for the VE

B.2.1 Unity Listing for Displaying Predicted Path

```
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using System.Net.Sockets;
using UnityEngine;
using System;
using System.Net;
using UnityEngine.UI;
using System.Text;
using UnityEngine.SceneManagement;


namespace RosSharp.RosBridgeClient
{
    public class MyPathSubscriber : UnitySubscriber<MessageTypes.Std.
    Int32MultiArray>
    {
        public TcpListener server;
        private LineRenderer line;
        private Vector3 mousePos;
        public Material material;
        private int currLines = 0;
        private double xc, yc, xp, yp;
        private bool isMessageReceived = false;
        private int x, y;
        private int[] dest;
        private int[] dest_t = new int[1];
        string host = "127.0.0.1";
        int portPathstatus = 52063;
        float pathstatusfromServer;

        Thread tcpListenerThread;

        protected override void Start()
        {
            base.Start();
            line = new GameObject("PathLine").AddComponent<LineRenderer>();
            tcpListenerThread = new Thread(() => ListenForMessages());
            tcpListenerThread.Start();
        }

        private void Update()
        {
            Debug.Log("pathstatus: " + pathstatusfromServer);
            if (isMessageReceived == true)
            {
                ShowLine();
            }
            if (pathstatusfromServer != 1)
            {
                line.enabled = false;
            }
            if (pathstatusfromServer == 1)
            {
                line.enabled = true;
            }
        }

        public void ListenForMessages()
        {

            try
            {
```

```
63                // Set the TcpListener on port 13000.
64                IPAddress localAddr = IPAddress.Parse(host);
65
66                // TcpListener server = new TcpListener(port);
67                server = new TcpListener(localAddr, portPathstatus);
68
69                // Start listening for client requests.
70                server.Start();
71
72
73                Byte[] bytes = new Byte[1024];      //The byte array containing the
     sequence of bytes to decode.
74                String data = null;
75
76                while (true)
77                {
78                    using (TcpClient client = server.AcceptTcpClient())
79                    {
80                        data = null;
81
82                        // Get a stream object for reading and writing
83                        NetworkStream stream = client.GetStream();
84
85                        int i;      //The number of bytes to decode
86
87                        while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
88                        {
89                            data = System.Text.Encoding.ASCII.GetString(bytes, 0, i
    );  // 0 is The index of the first byte to decode.
90                            float pathstatusfromRhapsody = (float)Convert.ToDouble(
    data);
91
92                            pathstatusfromServer = pathstatusfromRhapsody;
93                            data = data.ToUpper();
94
95                            byte[] msg = System.Text.Encoding.ASCII.GetBytes(data);
96
97                            stream.Write(msg, 0, msg.Length);
98                        }
99                        //client.Close();
100                   }
101               }
102           }
103           catch (SocketException e)
104           {
105               Debug.LogError(String.Format("SocketException: {0}", e));
106           }
107           finally
108           {
109               server.Stop();
110           }
111       }
112
113       protected override void ReceiveMessage(MessageTypes.Std.Int32MultiArray
    message)
114       {
115           Debug.Log("Hello from path");
116           GetPosition(message);//.Ros2Unity();
117           isMessageReceived = true;
118       }
119
120       private void GetPosition(MessageTypes.Std.Int32MultiArray message)
121       {
122           Debug.Log("Received data " + message.data.Length);
123           dest = message.data;
124           if (dest == dest_t)
125           {
```

```
126                 isMessageReceived = false;
127             }
128             else
129             {
130                 dest_t = dest;
131                 isMessageReceived = true;
132             }
133 
134         }
135 
136         void ShowLine()
137         {
138             createLine(dest.Length);
139             line.SetPosition(0, new Vector3(0, 0.1f, 0));
140             for (int i = 0; i < dest.Length; i++)
141             {
142                 x = dest[i] / 117 + 1;
143                 y = dest[i] % 117;
144                 xc = (float)y / 20 - 1.05;
145                 yc = 1.7 - (float)x / 20;
146                 Debug.Log("Coordinates for " + i + " are: " + x + " " + y);
147 
148                 line.SetPosition(i + 1, new Vector3(-(float)yc, 0.1f, (float)xc));
149             }
150         }
151 
152         void createLine(int vertices)
153         {
154             line.material = material;
155             line.positionCount = vertices + 1;
156             line.startWidth = 0.05f;
157             line.endWidth = 0.05f;
158             line.useWorldSpace = false;
159             line.numCapVertices = 50;
160         }
161     }
162 }
```

B.3 Listings for the Ss

B.3.1 Listing for Ss: MATLAB

B.3.1.1 Matlab:Rhapsody Interface

```
function [] = rhapsodyInterface(grph,grd)

curr=rossubscriber("/odom");
magn=rossubscriber("/magnetic_vals");
[path_pub, path_msg]=rospublisher("/path","std_msgs/Int32MultiArray");
[mov_pub, mov_msg]=rospublisher("/cmd_vel");

dest_x=[-0.04, 1.87, -0.54, 1.26, 2.1, 2.52, 0.52, 1.87, 2.54, 1.87];
dest_y=[0.84, -1.82, 0.94, 0.97, 1.35, 0.89, -1.977, -1.82, -1.023, -1.97];
path=[];

g=grph;
A=grd;

while 1
    dest=str2double(python('serv.py','52060'))-48;
    goto=str2double(python('serv.py','52068'))-48;
    movd=str2double(python('serv.py','52059'))-48;
    p_algo=str2double(python('serv.py','52056'))-48;
    sped=str2double(python('serv.py','52058'))-48;
    stbt=str2double(python('serv.py','52066'))-48;
    locz=[curr.LatestMessage.Pose.Pose.Position.X, curr.LatestMessage.Pose.Pose.
    Position.Y];
    if dest
        sprintf("Destination is %d", dest)
    end
    if p_algo
        sprintf("p_algo is %d", p_algo)
    end
    if goto
        sprintf("Go to dest is %d", goto)
    end
    if movd
        sprintf("Move Direction is %d", movd)
    end
    if sped
        sped=0.26*sped/5;
        sprintf("Speed is %d", sped)
    end
    if stbt
        sprintf("State of bot is %d", stbt)
    end

    if dest & p_algo
        destn=[dest_x(dest), dest_y(dest)];
        path=planning(g,A,p_algo,locz,destn);
        path_msg.Data=path;
        send(path_pub,path_msg);
    end

    if goto == 1
        GoToDest(path,destn);
        disp("Going to dest");
    end

    if stbt == 1
        if sped
            if movd == 1
                mov_msg.Linear.X=sped;
                send(mov_pub,mov_msg);
                disp("Move fwd");
```

```
            elseif movd == 4
                mov_msg.Linear.X=-sped;
                send(mov_pub,mov_msg);
                disp("Move bck");
            end
        else
            mov_msg.Linear.X=0;
            send(mov_pub,mov_msg);
            disp("No speed");
        end
        if movd == 2
            mov_msg.Angular.Z=0.1;
            send(mov_pub,mov_msg);
            disp("Rotate left");
        elseif movd == 3
            mov_msg.Angular.Z=-0.1;
            send(mov_pub,mov_msg);
            disp("Rotate right");
        end

    else
        mov_msg.Linear.X=0;
        mov_msg.Angular.Z=0;
        send(mov_pub,mov_msg);
        disp("Stop");
    end

    if getmag == 1
        disp("Magnetic Field Values: X: %f Y: Z: ",magn.LatestMessage.
    MagneticField_.X,magn.LatestMessage.MagneticField_.Y,magn.LatestMessage.
    MagneticField_.Z)
    end

    pause(0.5);
end

end
```

Python Code: *serv.py*

```
import socket
import sys

HOST = '127.0.0.1'  # Standard loopback interface address (localhost)

def get_data(port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, port))
        s.listen()
        conn, addr = s.accept()
        with conn:
            data = conn.recv(1024)
            return data[0]

if __name__ == '__main__':
    prt = float(sys.argv[1])
    x=str(get_data(int(prt)))
    sys.stdout.write(x)
```

B.3.1.2 Path Planning

```
function [new_path] = planning(graph,grd,algo,currp,dest)
```

```

nav_algo=algo;

c_loc=currp;
d_loc = dest;

y_w_limits=[-3.4, 1.7];
x_w_limits=[-1.05,4.75];
res=20;
pth=[];
c_pos=[round(((y_w_limits(2)-c_loc(2))*res)) round(((c_loc(1)-x_w_limits(1))*res))
    ];
d_pos=[round(((y_w_limits(2)-d_loc(2))*res)) round(((d_loc(1)-x_w_limits(1))*res))
    ];

c_p=(c_pos(1)-1)*117+c_pos(2);
d_p=(d_pos(1)-1)*117+d_pos(2);

if nav_algo == 1
    path = shortestpath(graph,c_p,d_p,'Method','unweighted'); %BFS
elseif nav_algo == 2
    path = shortestpath(graph,c_p,d_p,'Method','positive');   %Dijkstra
elseif nav_algo == 3
    pth = AStar(1,c_pos,d_pos,grd);
    path=flip(pth);
elseif nav_algo == 4
    pth = AStar(2,c_pos,d_pos,grd);
    path=flip(pth);
end

path1=path;
curp=path1(1);

new_path=curp;

for i=path1
    a = [(floor(curp/117)+1) mod(curp,117)];
    b = [(floor(i/117)+1) mod(i,117)];

    if (a(1) == b(1)) || (a(2) == b(2))

    else
        curp=path1(find(path1==i)-1);
        new_path=[new_path curp];
    end
end
new_path=[new_path path1(end)];
end
```

B.3.1.3 A* Code

```
function [path] = AStar(heur,start, dest, Grid)
% clc; clear all;
% clear classes;
cost=1;
Found=false;
Resign=false;
grid=Grid;
init=start;
goal=dest;
path=[];
if heur ==1
    Heuristic=CalculateEuclidian(grid,goal); %Calculate the Heuristic
elseif heur ==2
    Heuristic=CalculateManhattan(grid,goal); %Calculate the Heuristic
end

```

```
ExpansionGrid(1:size(grid,1),1:size(grid,2)) = -1; % to show the path of expansion
ActionTaken=zeros(size(grid)); %Matrix to store the action taken to reach that
    particular cell
%how to move in the grid
delta = [-1,  0; % go up
          0, -1; % go left
          1,  0; %go down
          0,  1]; % go right


for i=1:size(grid,1)
    for j=1:size(grid,2)
        gridCell=search();
        if(grid(i,j)>5)
           gridCell=gridCell.Set(i,j,1,Heuristic(i,j));
        else
            gridCell=gridCell.Set(i,j,0,Heuristic(i,j));
        end
        GRID(i,j)=gridCell;
        clear gridCell;
    end
end
Start=search();
Start=Start.Set(init(1),init(2),grid(init(1),init(2)),Heuristic(init(1),init(2)));
Start.isChecked=1;
GRID(Start.currX,Start.currY).isChecked=1;
Goal=search();
Goal=Goal.Set(goal(1),goal(2),grid(goal(1),goal(2)),0);

OpenList=[Start];
ExpansionGrid(Start.currX,Start.currY)=0;
small=Start.gValue+Start.hValue;
count=0;
while(Found==false || Resign==false)
    small=OpenList(1).gValue+OpenList(1).hValue+cost;
    for i=1:size(OpenList,2)
        fValue=OpenList(i).gValue+OpenList(i).hValue;
        if(fValue<=small)
            small=fValue;
            ExpandNode=OpenList(i);
            OpenListIndex=i;
        end
    end
    OpenList(OpenListIndex)=[];
    ExpansionGrid(ExpandNode.currX,ExpandNode.currY)=count;
    count=count+1;

    for i=1:size(delta,1)
        direction=delta(i,:);
        if(ExpandNode.currX+ direction(1)<1 || ExpandNode.currX+direction(1)>size(
    grid,1)|| ExpandNode.currY+ direction(2)<1 || ExpandNode.currY+direction(2)>
    size(grid,2))
            continue;
        else
            NewCell=GRID(ExpandNode.currX+direction(1),ExpandNode.currY+direction
    (2));

            if(NewCell.isChecked~=1 && NewCell.isEmpty~=1)
                GRID(NewCell.currX,NewCell.currY).gValue=GRID(ExpandNode.currX,
    ExpandNode.currY).gValue+cost;
                GRID(NewCell.currX,NewCell.currY).isChecked=1; %modified line from
    the v1
                OpenList=[OpenList,GRID(NewCell.currX,NewCell.currY)];
                ActionTaken(NewCell.currX,NewCell.currY)=i;
            end

            if(NewCell.currX==Goal.currX && NewCell.currY==Goal.currY && NewCell.
```

```
    isEmpty~=1)
                Found=true;
                Resign=true;
                disp('Search Successful');
                GRID(NewCell.currX,NewCell.currY).isChecked=1;
                ExpansionGrid(NewCell.currX,NewCell.currY)=count;
                GRID(NewCell.currX,NewCell.currY)
                break;
            end
        end
    end
    if(isempty(OpenList) && Found==false)
        return;
    end
end

X=goal(1);Y=goal(2);

while(X~=init(1)|| Y~=init(2))
    x2=X-delta(ActionTaken(X,Y),1);
    y2=Y-delta(ActionTaken(X,Y),2);
    path(end+1)=(x2-1)*size(grid,2)+y2;
    X=x2;
    Y=y2;
end

end
```

```
classdef search

    properties
        gValue;
        currX;
        currY;
        isEmpty;
        isChecked;
        hValue;
    end

    methods
        function obj=search()
            obj.currX=0;
            obj.currY=0;
            obj.gValue=0;
            obj.isEmpty=0;
            obj.isChecked=0;
            obj.hValue=0;
        end

        function obj=Set(obj,X,Y,EmptyStatus,heuristic)
            obj.currX=X;
            obj.currY=Y;
            obj.isEmpty=EmptyStatus;
            obj.hValue=heuristic;
        end

    end
end
```

Euclidian Distance

```
function [Heuristic]=CalculateEuclidian(grd,goal)
Heuristic=zeros(size(grd));
for i=1:size(grd,1)
    for j=1:size(grd,2)
        Heuristic(i,j)=sqrt((i-goal(1))^2+(j-goal(2))^2);
    end
```

```
end
end
```

Manhattan Distance

```
function [Heuristic]=CalculateManhattan(grd,goal)
Heuristic=zeros(size(grd));
for i=1:size(grd,1)
    for j=1:size(grd,2)
        Heuristic(i,j)=(i-goal(1))+(j-goal(2));
    end
end
end
```

B.3.1.4 Motion Control of TB3

```
function [] = GoToDest(path,d_loc)

subscriber.loc=rossubscriber("/odom");
subscriber.scan=rossubscriber("/scan");
publisher.mov=rospublisher("/cmd_vel");
[path_pub, path_msg]=rospublisher("/path","std_msgs/Int32MultiArray");
path_msg.Data=path;
send(path_pub,path_msg);
yahaan=subscriber.loc.LatestMessage.Pose.Pose;
mov_msg=rosmessage(publisher.mov);
x_w_limits=[-1.05,4.75];
y_w_limits=[-3.4,1.7];
res=20;
y_coor=[round(yahaan.Position.X*100)/100 round(yahaan.Position.Y*100)/100];
y_c=[round(((y_w_limits(2)-y_coor(2))*res)) round(((y_coor(1)-x_w_limits(1))*res))
    ];
res=20;
pos=1;
req_pos=1;
new_path=path(2:end);
i=new_path(2);
while (y_coor < 0.95*d_loc) | (y_coor > 1.05*d_loc)
    for i=new_path
        pause(1)
        scan=subscriber.scan.LatestMessage.Ranges;
        yahaan=subscriber.loc.LatestMessage.Pose.Pose;
        y_coor=[round(yahaan.Position.X*100)/100 round(yahaan.Position.Y*100)/100];
        y_c=[round(((y_w_limits(2)-y_coor(2))*res)) round(((y_coor(1)-x_w_limits(1)
    )*res))];

        dest_coor=[(floor(i/117)+1) mod(i,117)];

        if dest_coor(1) > y_c(1)
            req_pos=3;
        elseif dest_coor(1) < y_c(1)
            req_pos=2;
        elseif dest_coor(2) > y_c(2)
            req_pos=1;
        elseif dest_coor(2) < y_c(2)
            req_pos=4;
        end
        theta=atan2(yahaan.Orientation.X*yahaan.Orientation.Y+yahaan.Orientation.W*
    yahaan.Orientation.Z,0.5-yahaan.Orientation.Y^2-yahaan.Orientation.Z^2)*180/pi;

        if theta>-1 && theta<1
            pos=1;
        elseif theta>89 && theta<91
            pos=2;
        elseif theta>-91 && theta<-89
            pos=3;
        elseif theta>179 && theta<180 && theta>-180 && theta<-179
```

```
49              pos=4;
50          end
51
52          poses=[0 89 -91 179];
53          if pos~=req_pos
54              while round(theta) ~= poses(req_pos)
55                  mov_msg.Angular.Z=0.08;
56                  send(publisher.mov,mov_msg);
57                  yahaan=subscriber.loc.LatestMessage.Pose.Pose;
58                  theta=atan2(yahaan.Orientation.X*yahaan.Orientation.Y+yahaan.
    Orientation.W*yahaan.Orientation.Z,0.5-yahaan.Orientation.Y^2-yahaan.
    Orientation.Z^2)*180/pi;
59              end
60              mov_msg.Angular.Z=0.0;
61              send(publisher.mov,mov_msg);
62          end
63
64          if y_c(1)~=dest_coor(1)
65              temp_d = 1;
66          elseif y_c(2)~=dest_coor(2)
67              temp_d = 2;
68          end
69
70          while (y_c(temp_d)~=dest_coor(temp_d))
71              yahaan=subscriber.loc.LatestMessage.Pose.Pose;
72              scan=subscriber.scan.LatestMessage.Ranges;
73              y_coor=[round(yahaan.Position.X*100)/100 round(yahaan.Position.Y*100)
    /100];
74              y_c=[round(((y_w_limits(2)-y_coor(2))*res)) round(((y_coor(1)-
    x_w_limits(1))*res))];
75              while lidarstop(scan)==1
76                  scan=subscriber.scan.LatestMessage.Ranges;
77                  mov_msg.Linear.X=0.00;
78                  send(publisher.mov,mov_msg);
79              end
80              %%start movement
81              if dest_coor(temp_d) ~= y_c(temp_d)
82                  mov_msg.Linear.X=0.05;
83                  send(publisher.mov,mov_msg);
84              end
85          end
86          mov_msg.Linear.X=0.0;
87          send(publisher.mov,mov_msg);
88      end
89  end
90  mov_msg.Linear.X=0.0;
91  send(publisher.mov,mov_msg);
92  end
```

B.3.2 Listing for Ss: Python

B.3.2.1 Gaussian Regression Code

```
import scipy.io
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from IPython.display import display, clear_output
import GPy
from server1 import serv_mag, serv_disp

mat = scipy.io.loadmat('Data/data_04_06b.mat')

def intersection(lst1, lst2):
    lst3 = [value for value in lst1 if value in lst2]
    return lst3


# Assign the data from mat file
magnetic_field=mat['mag']
xz_train=mat['xz_m'][0]
yz_train=mat['yz_m'][0]
x_train=mat['x_m'][0]
y_train=mat['y_m'][0]
x_test=mat['x_test_m'][0]
y_test=mat['y_test_m'][0]

train_mag_x=np.zeros(len(x_test))
train_mag_y=np.zeros(len(x_test))
train_mag_z=np.zeros(len(x_test))

mag_x=magnetic_field[0][0][0][0]*10**6
mag_y=magnetic_field[0][0][1][0]*10**6
mag_z=magnetic_field[0][0][2][0]*10**6
magz=[]

data_locz={'X': xz_train, 'Y': yz_train,'Mag_z': mag_z}
data_locx={'X': x_train, 'Y': y_train,'Mag_x': mag_x}
data_locy={'X': x_train, 'Y': y_train,'Mag_y': mag_y}
Xx_train=pd.DataFrame(data=data_locx)
Xy_train=pd.DataFrame(data=data_locy)
Xz_train=pd.DataFrame(data=data_locz)

# Create training dataframes
X_train_z=Xz_train.groupby(['X','Y']).mean()
X_train_z=X_train_z.reset_index()
X_train_x=Xx_train.groupby(['X','Y']).mean()
X_train_x=X_train_x.reset_index()
X_train_y=Xy_train.groupby(['X','Y']).mean()
X_train_y=X_train_y.reset_index()

for i in range(len(X_train_x)):
    index=intersection(X_train_z.index[X_train_z['X']==X_train_x['X'][i]].tolist(),
    X_train_z.index[X_train_z['Y']==X_train_x['Y'][i]].tolist())[0]
    magz.append(X_train_z['Mag_z'][index])

data_loc={'X': X_train_x['X'], 'Y': X_train_x['Y'],'Mag_x': X_train_x['Mag_x'],'
    Mag_y': X_train_y['Mag_y'],'Mag_z': magz}
data_locz={'X': X_train_x['X'], 'Y': X_train_x['Y'], 'Mag_z': magz}
X_train=pd.DataFrame(data=data_loc)
x_train_z=pd.DataFrame(data=data_locz)


# Create test data frame
data_test={'X': x_test, 'Y':y_test}
```

```
X_test=pd.DataFrame(data=data_test)

# Individual Axes

# X Axis
kern_x = GPy.kern.RBF(input_dim=2, variance=3., lengthscale=1.) + GPy.kern.Bias(1)
gpr_x = GPy.models.GPRegression(X_train_x[['X','Y']].to_numpy(),X_train_x[['Mag_x']].to_numpy(),kern_x)

gpr_x.optimize()

# Y Axis
kern_y = GPy.kern.RBF(input_dim=2, variance=0.8, lengthscale=1.) + GPy.kern.Bias(1)
gpr_y = GPy.models.GPRegression(X_train_y[['X','Y']].to_numpy(),X_train_y[['Mag_y']].to_numpy(),kern_y)

gpr_y.optimize()

# Z Axis
kern_z = GPy.kern.RBF(input_dim=2, variance=5., lengthscale=1.) + GPy.kern.Bias(1)
gpr_z = GPy.models.GPRegression(Xz_train[['X','Y']].to_numpy(),Xz_train[['Mag_z']].to_numpy(),kern_z)

gpr_z.optimize()

# Joint Axis Models
kern = GPy.kern.RBF(input_dim=2, variance=3., lengthscale=0.2) + GPy.kern.Bias(1)
gpr_t = GPy.models.GPRegression(Xt_train[['X','Y']].to_numpy(),yt_train[['Mag_x','Mag_y','Mag_z']].to_numpy(),kern)

gpr_t.optimize()

# Predict the data
predict_x=gpr_x.predict(X_test.to_numpy())
predict_y=gpr_y.predict(X_test.to_numpy())
predict_z=gpr_z.predict(X_test.to_numpy())
predict = gpr_t.predict(X_test.to_numpy())

xpred=[]
ypred=[]
zpred=[]
for i in range(len(predict[:][0])):
    xpred.append(predict[0][i][0])
    ypred.append(predict[0][i][1])
    zpred.append(predict[0][i][2])

fig1, (ax1, ax2) =plt.subplots(1,2,gridspec_kw={'width_ratios': [1,2]})
cb_x=ax1.scatter(X_train_x['X'],X_train_x['Y'],c=X_train_x['Mag_x'],cmap='RdYlGn')
ax1.set_xlabel("X (in m)")
ax1.set_ylabel("Y (in m)")
ax2.scatter(x_test,y_test,c=list(predict_x)[0],cmap='RdYlGn')
ax2.set_xlabel("X (in m)")
ax2.set_ylabel("Y (in m)")
cbx=fig1.colorbar(cb_x)
cbx.ax.set_title("\u03BC T")
fig1.set_figheight(5)
fig1.set_figwidth(15)
ax1.set_title("Data Collected")
ax2.set_title("Extrapolated Field")
fig1.suptitle("X Field",fontsize=20)

fig2, (ax3, ax4) =plt.subplots(1,2,gridspec_kw={'width_ratios': [1,2]})
cb_y=ax3.scatter(X_train_y['X'],X_train_y['Y'],c=X_train_y['Mag_y'],cmap='RdYlGn')
ax3.set_xlabel("X (in m)")
ax3.set_ylabel("Y (in m)")
ax4.scatter(x_test,y_test,c=list(predict_y)[0],cmap='RdYlGn')
ax4.set_xlabel("X (in m)")
```

```
ax4.set_ylabel("Y (in m)")
cby=fig2.colorbar(cb_y)
cby.ax.set_title("\u03BC T")
fig2.set_figheight(5)
fig2.set_figwidth(15)
ax3.set_title("Data Collected")
ax4.set_title("Extrapolated Field")
fig2.suptitle("Y Field",fontsize=20)

fig3, (ax5, ax6) =plt.subplots(1,2,gridspec_kw={'width_ratios': [1,2]})
cb_z=ax5.scatter(X_train_z['X'],X_train_z['Y'],c=X_train_z['Mag_z'],cmap='RdYlGn')
ax5.set_xlabel("X (in m)")
ax5.set_ylabel("Y (in m)")
ax6.scatter(x_test,y_test,c=list(predict_z)[0],cmap='RdYlGn')
ax6.set_xlabel("X (in m)")
ax6.set_ylabel("Y (in m)")
cbz=fig3.colorbar(cb_z)
cbz.ax.set_title("\u03BC T")
fig3.set_figheight(5)
fig3.set_figwidth(15)
ax5.set_title("Data Collected")
ax6.set_title("Extrapolated Field")
fig3.suptitle("Z Field",fontsize=20)

fig4, (ax11, ax12) =plt.subplots(1,2,gridspec_kw={'width_ratios': [1,2]})
cb_x=ax11.scatter(X_train['X'],X_train['Y'],c=X_train['Mag_x'],cmap='RdYlGn')
ax11.set_xlabel("X (in m)")
ax11.set_ylabel("Y (in m)")
ax12.scatter(x_test,y_test,c=xpred,cmap='RdYlGn')
ax12.set_xlabel("X (in m)")
ax12.set_ylabel("Y (in m)")
cbx=fig4.colorbar(cb_x)
cbx.ax.set_title("\u03BC T")
fig4.set_figheight(5)
fig4.set_figwidth(15)
ax11.set_title("Data Collected")
ax12.set_title("Extrapolated Field")
fig4.suptitle("X Field",fontsize=20)

fig5, (ax13, ax14) =plt.subplots(1,2,gridspec_kw={'width_ratios': [1,2]})
cb_y=ax13.scatter(X_train['X'],X_train['Y'],c=X_train['Mag_y'],cmap='RdYlGn')
ax13.set_xlabel("X (in m)")
ax13.set_ylabel("Y (in m)")
ax14.scatter(x_test,y_test,c=ypred,cmap='RdYlGn')
ax14.set_xlabel("X (in m)")
ax14.set_ylabel("Y (in m)")
cby=fig5.colorbar(cb_y)
cby.ax.set_title("\u03BC T")
fig5.set_figheight(5)
fig5.set_figwidth(15)
ax13.set_title("Data Collected")
ax14.set_title("Extrapolated Field")
fig5.suptitle("Y Field",fontsize=20)

fig6, (ax15, ax16) =plt.subplots(1,2,gridspec_kw={'width_ratios': [1,2]})
ax15.scatter(X_train['X'],X_train['Y'],c=X_train['Mag_z'],cmap='RdYlGn')
ax15.set_xlabel("X (in m)")
ax15.set_ylabel("Y (in m)")
cb_z=ax16.scatter(x_test,y_test,c=zpred,cmap='RdYlGn')
ax16.set_xlabel("X (in m)")
ax16.set_ylabel("Y (in m)")
cbz=fig6.colorbar(cb_z)
cbz.ax.set_title("\u03BC T")
fig6.set_figheight(5)
fig6.set_figwidth(15)
ax15.set_title("Data Collected")
ax16.set_title("Extrapolated Field")
```

```
fig6.suptitle("Z Field",fontsize=20)


method=serv_mag()-48
show_map=serv_disp()-48

if method==1:
    if show_map==2:
        display(fig1)
    if show_map==3:
        display(fig2)
    if show_map==4:
        display(fig3)
if method==2:
    if show_map==2:
        display(fig4)
    if show_map==3:
        display(fig5)
    if show_map==4:
        display(fig6)
```

B.3.2.2 Server Codes for interaction with Rhapsody

```
import socket

def serv_mag():
    HOST = '127.0.0.1'  # Local Address
    PORT = 52064        # Port to listen on for magnetic extrapolation method
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        conn, addr = s.accept()
        with conn:
            data = conn.recv(1024)
            return data[0]

def serv_disp():
    HOST = '127.0.0.1'  # Local Address
    PORT = 52063        # Port to listen on for display
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        conn, addr = s.accept()
        with conn:
            data = conn.recv(1024)
            return data[0]
```