ÉCOLE NATIONALE SUPÉRIEURE DES MINES DE SAINT-ÉTIENNE

MASTER'S 1 REPORT

Implementation of a robot behavior learning simulator

Author:
Kushagra Singh BISEN

Supervisor:
Dr. Mihaela JUGANARU-MATHIEU

A report submitted in fulfillment of the requirements for the degree of Master's 1 in Cyber-Physical and Social Systems

in the

Set of Intelligent Robots Project
Institu Henri Fayol, IT and Intelligent Systems department

August 27, 2021

Declaration of Authorship

I, Kushagra Singh BISEN, declare that this thesis titled, "Implementation of a robot behavior learning simulator" and the work presented in it are my own. I confirm that:

- This work was done wholly in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

ÉCOLE NATIONALE SUPÉRIEURE DES MINES DE SAINT-ÉTIENNE

Abstract

Mihaela JUGANARU-MATHIEU
Institu Henri Fayol, IT and Intelligent Systems department

Master's 1 in Cyber-Physical and Social Systems

Implementation of a robot behavior learning simulator

by Kushagra Singh BISEN

In this project, we present the implementation of digital twin of a robot to learn the behaviour. The behaviour will provide the robot intelligence to prevent collision while navigating. Digital Twin is a technology to use software to produce an exact replica of a physical twin. Digital Twins help to train and test before a physical twin. We have used ROS navigation stack with modifications suited to the project for autonomous navigation. We implemented a robot with sensors to execute autonomous navigation and technologies such as SLAM. We have developed a method to learn the behaviour of the robot without using information from the robot in an environment with obstacles. The method periodically learns and records the obstacle state around the robot to save in a log file.

Acknowledgements

It is a genuine pleasure to express my gratitude and appreciation to my guide Dr. Mihaela Juganaru-Mathieu, Dr. Nida Meddouri, Dr. Flavien Balbo and Dr.Mireille Batton-Hubert. Their dedication to the research, constant supervision, and an overwhelmingly warm and understanding attitude towards me have made it possible to complete my work. Their timely advice, diligent interest towards my work and suggestions when I was deviated from the path have helped me to an great extent to accomplish my task and understand research which developed a keen interest in me for the research area. I owe a deep sense of gratitude to Institut Henri Fayol for funding my internship with a great salary. I would also like to thank Saint Etienne Metropole for providing me with a grant to support my work at EMSE. I owe everything to my parents. Without their support and encouragement, none of it would have been possible.

Contents

Chapter 1

Context of the Work

1.1 Introduction

The work is based on an Industry 4.0 scenario, which is a cyber-physical environment consisting of various different actors and objects involved. The different actors involved are either stationary or mobile. Moreover, complexity of the environment increases when we account for heterogeneous actors with various decision making capabilities. Robots with various manufacturers present various transform frames, different software and sensors. Due to the heterogeneous nature of the robots involved, we can not depend on information we receive from the robot, as this particular information will differ from a robot to other based upon it's configuration. The problem is solved by creating a digital twin which records the information of the environment as well as the robot. The simulator is dedicated to capture the digital twin of the robot, whereas a script saves the log values of the obstacle space in a semi-structured format to be used later.

1.1.1 Structure of the Report

The report is structured in four chapters. The first chapter provides context to the work and introduces concepts required to study and implement the work. The second chapter details the technologies involved in the project, hardware and software. In the third chapter, details about implementation of technologies learnt from chapter two is discussed. The solution to problem is discussed in the fourth chapter and the report ends with a conclusion detailing the perspective of the work.

1.2 Motivation

The structure of a dynamic Industry 4.0 environment is highly volatile, the structure is defined through a stationary frame that has been declared before. The decision making capabilities of the robot to navigate the environment while avoiding obstacles and other robots can have a great impact on the performance and the utility of the environment.

1.3 Robot

1.3.1 What is a Robot?

The origin of the word robot can be found in Czech playwriter Karel Chapek's play titled "Rossum's Universal Robots (R.U.R)" in 1921. Thw word robot results from combining the Czech words *rabota* meaning compulsory work and *robotnik* meaning an agricultural bound labor. A **robot** is a system existing in a physical world,

with decision making capabilities of varying extent, can sense the environment it's in to achieve some goals. A goal can be differ according to the need of autonomous behaviour. Essentially, robot is a cyber-physical system combining sensing, actuation, and computation. With the advancements in technology and materials essential to build a robot, we can see numerous different robots with different applications. Robots such as :

- a self-foldable / self-actuated robot developed at MIT [8]
- a lightweight aerial robot developed at University of Penn
- consumer-grade drones by DJI
- Autonomous Vehicles developed at Google.
- Autonomous Surface Vehicles by ASV Global

Robots help humans to do *dirty*, *dull*, or *dangerous* tasks that no human wishes to do, although they are important to be done. As any machines, in an Industry 4.0 environment humans can integrate robots into the development/production process, thus these processes can be optimized. Optimizing robots with different applications can help us to exploit robot technologies to leviate pressure imposed by growing population by using in applications such as,

- mobility-on-demand
- automated highways
- drone swams for servillance
- truck platoons for long distance logistics

Along with these mobile wheel bearing vehicular robots, we have other robots as :

- autonomous behaviour on any terrain for search and rescue with Big Dog robots.
- Persobal Robots for help with menial tasks, for example, iCub Robot.
- Emotional Robots wit Human Computer Interface designed to ease the interaction for example, Pepper Robot.

1.4 Autonomous Behavior

For a robot to perform autonomous behavior in an environment, it must be able to model and percieve the world it is in, be able to process information and perform required actions and plan it's behavior in adverse conditions. The level of such autonomy varies with different use cases. These challenges are solved by deploying perception module, action module and decision-making module. These three modules will be mounted and developed on a cyber-physical system, differentiate cyber-physical systems with pure artificial intelligence. Architectures employed in Robotics combine the three modules to be used by the developer to develop such CPS systems.

1.4.1 Perception

For a robot to initiate any form important autonomous behavior of decision making, the robot should know where the robot is present in the given Industry 4.0 environment. A robot uses different sensors to infer it's position in the environment. A sensor is an entity that is used to measure the state of the world as well as the state of the robot relative to the world. The different sensors provide measurements of the environment and extract meaningful information for autonomous behavior. **Proprioceptive sensor** in a robot is used to determine the coordiante location of the robot relative to the frame it is in, by measuring internal information to the robot. Proprioceptive sensors can also measure odometry, speedometer, energy level and acceleration of the robot. These coordinates on change define the movement of the robot. Another exteroceptive sensor is used to acquire information regarding the environment the robot is currently present in by calculating light intensity and sound amplitude to measure the distance from theh nearest obstacle. The perception module will save the information about the map to be inherited in other modules. **Exteroceptive Sensors** measure something external to the robot, i.e the environment. Exteroceptive sensors are of two types :

- *Active* : These sensors affect the environment by emitting energy. Examples would be, calculating distance with Infrared Sensor, sonar range finder and calculating distance with laser scanners.
- *Passive* : These sensors have no affect on the environment, for example, the ambient light sensor, a sound sensor or camera being used to sense vision.

1.4.2 Action

Action module decides the force and orientation for a robot to perform the task assigned. Action module deals with low level control of the robot's motor. In presence of a predefined goal, the action module will calculate the rotational and forward velocities to reach the goal. Action module comprises of various equations responsible to calculate the linear and angular velocities.

1.4.3 Decision Making

In order to achieve a higher order goal, the robot will use the action and perception modules to initiate *navigation* to reach a predefined goal. **Perception** module has provided the neccesary information to the robot about the environment and location of obstacles. **Action** module provides the neccesary equations to calculate the velocities to pursue the motion towards the goal. Delibrative planning is executed by the decision-making module to compute a path that does not collide with the obstacles and respects robot's motion constraints. In real Industry 4.0 environment, we've multiple robots and mobile entities. Collaboration, communication and coordination among the robots for path planning to calculate efficient algorithms for calculating linear and angular velocities are an interesting subject for research. For example, collective movement between robots as well as aerial unmanned vehicles such as drones can be initiated by either having a distributed architecture or a centralized leader-follower control. A decentralized system is prone to failure much more than a leader-follower control system. A decision-making algorithm could be written in the leader-follower control system to be in a certain range or perform a certain action around the leader robot. Multi Robot control and coordination allows us to recreate this.

The simplest example of an autonomous robot would be the Roomba robot, which is employed to clean our houses. The robot would use the sensors to infer the world i.e the room around him and initiate navigation to the most suitable location. A simple roomba robot had, 1) A cliff sensor to make sure the robot is not falling off a level field. 2) A bump sensor to retrack it's behaviour and initiate recovery motion. 3) A wall sensor to detect the walls and distance to it. 4) An optical sensor to detect the odometry and force being exerted by the motor. The behaviour displayed by the simple robot is :

- **Wall-Following**: The robot would follow the wall if it bumps into something.
- **Straight** : The robot would go straight and turn a random angle if it bumps into something.
- **Dirty Spot** : If the robot is in contact with a dirty spot, it will spiral around the position untill it bumps into something.

This behaviour is very naive and primitive in relation to what robots can do in current scenario and advancements in the last two decades are huge.

1.4.4 History

The field of robotics emerged from combining and taking influences from various different fields which were present at a time around 1950s. Robotics can see it's influences from :

- **Control Theory** : Control theory [9] develops methods for control of dynamic systems and engineered processes in an environment. The objective is to build a model or an algorithm to decide the behaviour of the entity once a particular trigger event (or an action request) has been initiated, the decided behaviour will influence the state transition to drive it into a desired state while minimizing any delay, overshoot, steady-state behavior while ensuring stability often to achieve a degree of optimality. A controller is present in the robot to decide this behaviour. In robotics, the most important part of controk theory is to control the feedback.
- **Cybernetics** : Cybernetics [10] is an interdisciplinary approach important to regulate the structure and control of a device. The outcomes of the actions are further fed into the feedback loop. It is the integration of sensing, action and environment.
- **Artificial Intelligence** : In 1950s, artificial intelligence dealt with planning processes and reasoning which integrated to develop into robotics.

Robotics shifted in 1980s, with introduction of new ideas such as 1) reactive control 2) hybrid control and 3) behaviour based control which produced faster and intelligent machines. Artificial Intelligence's approach to solve a robotics problem shifted from initiating a delibrative planning algorithm such as shortest distance. Instead robotics tried to imitate living objects and focussed on what an intelligent machine would behave like. By taking inspiration from organisms in nature, it was found that they follow a simple pattern of reactive rules to an external stimuli. This gave rise to reactive behaviours being applied into robotics. Behaviour based robotics was a key concept for the development of robotics at that point of time. We could use this behaviour based robotics concept and employ it in concepts such as swarm

robotics and collective intelligence, thus birthing a new research area in robotics itself. As years passed, and we had improvements in computation and hardware design for the sensors, we moved away from the behaviour based robotics. We still see concepts such as wall-following, obstacle avoidance is based upon behaviour based control paradigm. A huge trend recently is in leveraging neural networks and machine learning to improve the perception and control of the robot. Researchers are trying to integrate machine learning end to end in the robot itself promoting it's intelligent behaviour. As we use the robot's as a blackbox, with an end to end architecture it is very difficult for us to understand why a robot is doing what it is doing at a certain position. In our work, we are trying to record this behaviour of the robot while considering it as a black box with an end to end architecture and learning it's behaviour by making a digital twin.

1.5 Robot Control Architectures

The building block of an autonomous system such as a robot is a perception-action loop. Perception-Action loop is a cycle for the robot to process the thing that he experiences and the things that he does, simultaneously improving the decision making process of the robot. The robot is in a continous interaction with it's environment.

- **Reactive Control** : Reactive Control deals with using the sensors to present the current estimate of the world, recovery behavior rules in case of collision produce actions which are simple and fast to compute.
- **Delibrative Control** : Delibrative Control deals with predicting the future state of the robot, developing a plan for the same for the robot to decide the *sequence of actions* to pursue in order to follow the sequence. In our work, we have used delibrative planning to move the robot from a starting position to the goal position via different algorithms ; for example A*, Dijkstra and Greedy Best First Search Algorithm is used to compute the path of the robot.

Complicated actions require complicated control architectures, which combine the three elemental modules of autonomy which are perception, plan and action. Examples of such control architecture are,

- **Finite State Machines** : They are reactive and follow a sequential path. Consisting of a finite set of states and transitions between these states. A simple example in this case would be 'pick up the trash' robots.
- **Subsumption Architecture** : They are reactive and follow a concurrent path, which means that the three modules of perception action and decision making will execute simultaneously.
- **Sense-Plan-Act** : They follow a delibrative path planning method which use the modules in a step by step way.

As seen in the Figure 1.1, the motor controller is dynamic. The controller carries multiple processes simultaneously for the robot to move autonomously.

1.6 Motion Control in Robots

Motion Control in a robot deals with the process through which the delibrative or reactive control is passed down to actuators of the robot and finally to the motor

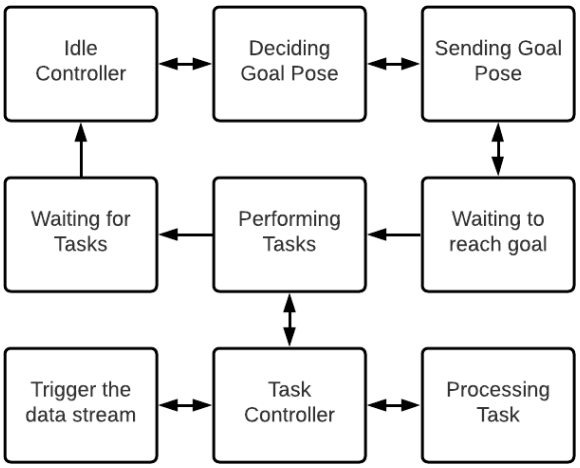

FIGURE 1.1: State Diagram of Controller

resulting in the physical change of state in a direction robot's decided to move to. The action component of the autonomy controls the robot with the motion control module.

1.6.1 Actuators

Actuators are used to convert a signal from a circuit board, e.g. from a Raspberry Pi or an Arduino into physical mechanical motion. Actuators serve various purposes, such as locomotion, manipulation, heating and sound emission. Other examples of electrical-to-mechanical actuators are, DC Motors, stepper motors and loudspeakers. *For example,* in a normal example of an automated car a driver can steer and acclerate, thus making 2 control points. There are examples of for uncertainty and noise in actuators. Examples of such are wheel slip, slack in the motion and environmental factors such as wind, friction and other natural uncontrollable distractions. Actuators influence a certain single degree of freedom of the robot. A robot is controllable in all of it degree of freedom if it has an actuator at each and every degree of freedom. *Degree of Mobility* is the number of DOFs which are operable by the actuators. If the DOF is equal to robot's DOM the robot is a *holonomic robot* and if the number of DOF is greater than it's DOM then it is a *non-holonomic robot*. Otherwise, if robot's DOM is more than it's DOF, the robot's actuation is redundant. The robot in our project is a **Differential Drive Robot** which can actuate with left and right wheels without any interference. DDR has 3 DOF but DOM is 2, differential drive robots are *non-holonomic* in nature.

Kinematics

Kinematics are important for us to infer and calculate the motion, i.e behaviour of the robot when it reaches a goal. Forward Kinematics decide where the robot would end up in the coordinate frame of the environment (x, y, θ) given that we have the control parameters and the time of movement. Reverse kinematics deal with finding the control parametetrs of the robot once the desired final position (x, y, θ) is provided.

1.7 Localization

Localization deals with positioning a robot in the coordinate frame of the environment. When you put a robot randomly in an Industry 4.0 environment, it does not know where it is or how the environment around the robot looks. There are various methods with which one can accomplish localization. *Dead-Reckoning*, is an initial/minimal method of localization where we know the initial position of the robot, and we further update the position of the robot blindly based on the differential movements. Thius method is odometry-based. *Global Localization*, is a method where initial position of the robot is not known. There are two major methods to infer the position of the robot and initialize localization :

- **Map-Based Localization** : In this method, we have various landmarks or obstacles which we can develop into a map. Sensors employed in this method are, laser sensor, camera for vision and proximity sensors. There are various map-matching techniques used to develop a map.
- **Beacon-Based** : In this methods, we employ an active infrastructure with bluetooth, WiFi, GPS Sensors for outdoors. The method used here is trilateration, fingerprinting and proximity.

In our implementation, as well as major implementations across different robotics projects. The project employs a combination of global localization and position tracking methods. There are multiple challenges faced by localization in dead-reckoning and global localization :

- **Dead-Reckoning**
  - a slip could occour in the wheel, thus displacing the localization and producing an error
  - errors caused by runaway
- **Global Localization**
  - various errors and failures
  - noise in the non-Gaussian sensor
  - unreliability of sensors access such as GPS
  - ambiguous map
  - environments which are continuously changing with dynamic obstacles
  - a robot is not able to know that it has been displaced, so it still displays the position relative to the previous reference frame.

1.7.1 Probabilistic Localization

In robotics, we employ localization probabilistically. The three key components to such localization are robot's information related to where it is (it's initial state), a predefined motion model and a sensor model to record the information of the surroundings by the robot.

1.8 Path Planning

Path Planning (see Figure 1.2) is done to predict the next successive states of the robot and form a path to which the robot can move by defining the configuration space of the environment and predicting a strategy employing various algorithms considering the space as well as the robot's motion constraints.

FIGURE 1.2: Path Planning

1.8.1 Configuration Space

An Industry 4.0 environment has broadly 2 types of objects, **robots** and **obstacles**. They are considered to be the part of the world. Robot's motion for path planning that a robot could execute as a rigid body in it's degree of freedom while maintaining robot's constraints of movement. The constraints are the size of robot's base and the degree to which it can move. The 'space' for the robot to move is possible coordinate motion space that can be applied for the robot to move in the environment without hitting an obstacle or obstructing the path of another robot. We can use similar motion planning algorithms to different problems and robots which have other geometry and motion controls. Robot's configuration can be described by a vector containing joint coordiantes of the robot. These coordinates are either an angle or a length for rotational and sliding joint respectively. Robot's Motion space is the environment space deducting the obstacle's space.

Minkowski Sum

Minkowski Sum is used to calculate the vector configuration space of the robot as a whole. The sum of two sets of position vectors **A** and **B** are calculated by adding a those vectors with one another, forming the set $A + B = a + b \mid a \in A$ and $b \in B$ To compute free space and an obstacle space for the robot, we need the vecotrs for the static obstacle as well as the robot in relation to a specific reference point. So, in case the robot collide, we add those two positional vectors, the result provides the value of obstacle space.

Path Planning Problem

The objective of a path planning problem is to succesfully predict the next states of the robot which the robot would follow. A trajectory is defined as the path that the robot takes, the trajectory is/can be different from the planned predicted path because of drift and size of the robot matter while cutting sharp corners in the path. Thus, in an Industry 4.0 environment a workspace, an obstacle region and a configuration space are defined. Suppose that we wish to go from Point A to Point B following the trajectory. The trajectory is calculated by a function where the initial and the final positions are both in the configuration space and the robot moves till the final destination coordiante is reached. The path planning is done by ROS's architecture, specifically the navigation stack.

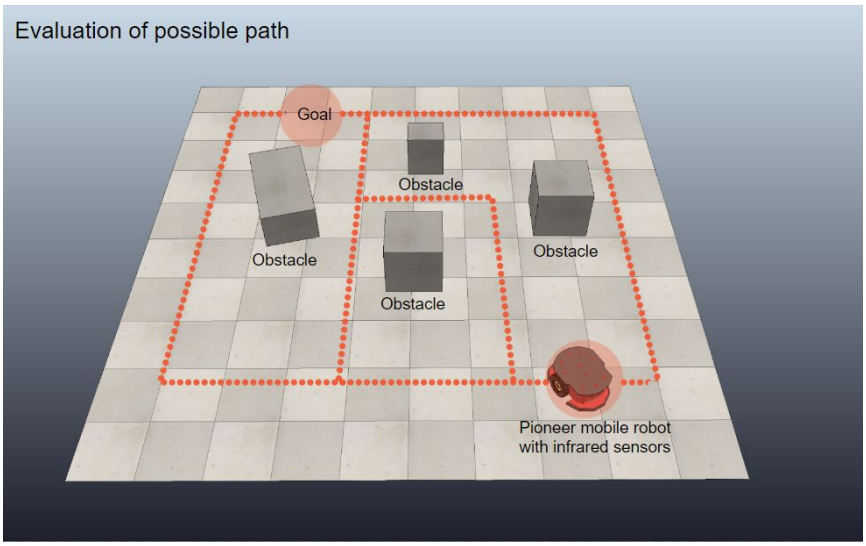

FIGURE 1.3: Evaluation of Possible Path to Goal.

There are various different approaches to path planning which use the configuration space of the environment for succesful next state planning. The different solutions provide levels of completeness to the solution. In our context, the path planning provides a complete solution i.e *If a solution exists, it will find one otherwise it will return a failure to form the path*. The work employs *combinatorial method* to find the path to a goal state.

Combinatorial Method

Combinatorial Method is employed to calculate a valid roadmap to access the connectivity to the final goal destination by maintaining the robot's constraints, with resoect to the obstacle space. Combinatorial Method is executed in the following steps :

- Calculating the Configuration space, and dividing them into free/obstacle space. This is employed by using SLAM (Simultaneous Localization and Mapping) to develop a map for the configuration space.
- Generation of a roadmap in the free configuration space, i.e calculating the free space in which the robot can reach the goal, and once the robot reaches the destination the exploration stops.

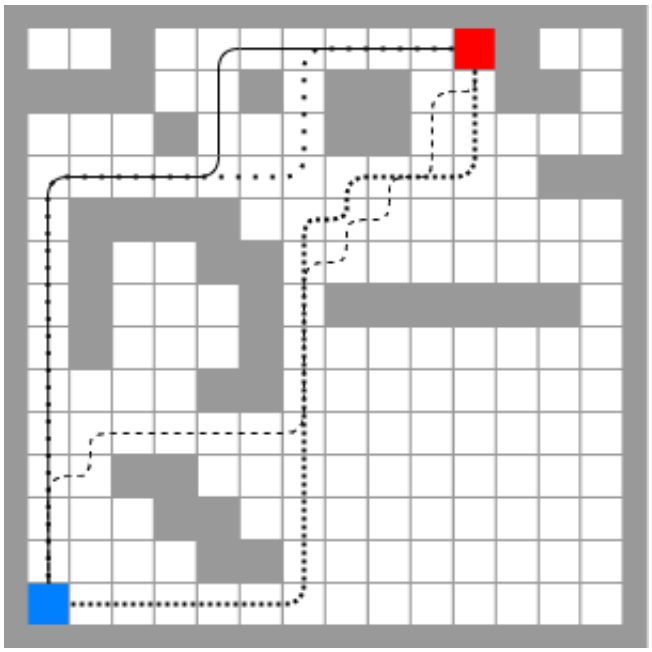

FIGURE 1.4: Possible ways of path planning.

- Computing the cost of the path from initial state to the final state, and choosing the path suited for implementation, for example, shortest path, optimal path or a path subjected to some constraints. It is totally user defined.
- The result is the final path in the free configuration zone.

1.9 Digital Twins

Digital Twin (DT) technology is now used and commercialized in various industries to optimize Industry 4.0 processes and environments. Digital Twin (DT) is used to model Industry 4.0 machine/process before it's inception in real-world using materials. Making an artificial digital twin model before a physical twin is required. It helps researchers to model for training testing and preventing errors. Based on the artificial digital twin, the researchers can further study and train and improve upon the previous present model. DTs can be defined as (physical and/or virtual) machines or computer-based models that are simulating, emulating mirroring, or "twinning" the life of a physical entity, which may be an object, a process, a human, or a human-related feature. Each DT is linked to its physical twin through a unique key [1]. A digital twin can be visualized by a simulation, but a DT is not a simulation, instead it is much more than a simulation. Digital Twin is an exact replica of an real-world entity, which means that it is a living, intelligent model which can evolve if needed to. All the abilities and capabilities of the physical twin can be realized such as monitoring and control. We could predict the future status of the digital twin, i.e we can predict future defects and failures. Digital Twin Techologies can enable continuous communication-collaboration-interaction with the physical twin simultaneously. Digital Twins are mostly a part of a Cyber-Physical System (CPS). To sum everything up, we can say that a Digital Twin is a technology which helps the researcher to utilize data fusion, Artificial Intelligence appliations with big data by using the IOT sensors without making the usual physical-twin.

In the context of our project, the cyber-physical system we have consists of a robot where we can employ different sensors to study it's behaviour in an Industry 4.0 environment. The robot is not supposed to have it's sensors do the computation. The hypothesis is to make a digital twin, with a god's eye view which can see and then log the behaviour of the robot around the obstacles in the configuration space.

1.10 Behavioural Learning

Multi-Agent Systems are rapidly growing in usage in a variety of domains. Multi Agent Systems contain, as the name suggests, multiple entities. In environments, where multi-agent systems rapidly collaborate with one another, i.e a dynamic system. The agent needs to learn and improve on basic behaviour using data mining and reinforcement learning. In the context of our project, the multi-agent system needs to prevent collision. Thus, we wish to learn the robot's behaviour around the obstacles in the particular space. For this, we note and log the position of the robot at every step, and it's nearby obstacles to produce data for a reinforcement learning model.

Chapter 2

Technologies Used

2.1 Introduction

In the previous chapter, we went over an introduction to the context of the work as well as the methods through which we can accomplish the goal of learning the behaviour of the robot as it moves in the configuration space. As we know a basic introduction to the work, in this chapter we would go over the technologies that are employed to achieve the goal of learning the behaviour of the robot. We will first go over the hardware and software requirements of the robot, and the environment.

2.2 Hardware Employed

The project uses a Turtlebot2 [12] robot. Turtlebot2 (see Figure 2.1) is a low-cost, personal robot kit with open-sourced software. The robot was created in 2010, but has been continuously envolved in both software strength and hardware with open-source contributions to the project. The Turtlebot2 consists of a base, two wheels, a single board computer and a LIDAR sensor. Other sensors for example a vision sensor, i.e. Camera can be attached later to the mainframe based upon the requirement. Turtlebot2 accomplishes SLAM and Navigation efficiently in the configuration space. It is developed to be employable in various scenarios, and can be controlled remotely by a computer or a joystick. Turtlebot2 can be attached with an arm to perform manipulator tasks.

FIGURE 2.1: Turtlebot2

2.2.1 LIDAR Sensor

LIDAR stands for light detection and ranging. LIDAR sensor measures the time of flight of the pulse of light. The light transmitted from LIDAR hits an obstacle and is noted. The light transmitted is usually infrared, where most of the energy is concentrated in a narrow beam of light. The distance to the obstacle is calculated by measuring the time of flight in seconds. The hardware involved here is a receiver emitter pairs (channels) which is combined wuth rotating mirrors to ensure that the whole area is covered. They are used in autonomous devices. The sampling rate of a LIDAR is high, therefore the light beams do not collide with one another resulting in precision. The range of LIDAR sensor are measured by maximum likelihood estimators. There are various types of LIDAR sensors with varying capabilities such as difference in the scanning area, resolution, detection range. The resuting data collected is a point cloud grid.

2.3 Software Employed

2.3.1 Gazebo Simulator

In the implementation, we are using **Gazebo** [6] as a simulator to visualize and execute the processes. Gazebo is a well-designed high quality simulation software with a real physics engine. Gazebo makes it easy for researchers to test AI algorithms, design Robots, perform regression and testing and train AI systems before using it in real-world applications. Gazebo can use 3D models to implement real-world scenarios for the robot to interact with. It is a trusted simulator, being used by reputed space agencies to simulate robots. Turtlebot2 has packages for Gazebo to present the robot in simulation. Gazebo's biggest advantage is it's huge community support around it.

Gazebo provides many features:

- *Dynamic Simulation :* Simulation is provided by many high-quality physics engine such as ODE, Bullet and DART.
- *Advanced 3D Graphics :* Gazebo provides great simulation which is very realistic including textures, lighting and shadows.
- *Sensors and Noise :* Simulator can be used to make real sensor data with noise from sensors and other equipments such as laser range finders, vision sensors and contact sensors etc.
- *Plugins :* Researcher can develop custom plugin for a specific sensor or a specific task in the simulation.
- *Robot Models:* There are many different robots which can be used in Gazebo, from simple robots such as Turtlebot2/3 to Mars Rover can be simulated in a Gazebo environment.
- *TCP/IP Transport:* We can run simulation on remote servers, and interface with Gazebo simulator through socket based message pushing.
- *Cloud Simulation :* CloudSim can be used to run Gazebo on Amazon Web services, or personal web service or personal architecture.
- *Command Line Tools :* Command line tools simplify simulation control and introspection.

Gazabo11 was the version being implemented with ROS Noetic in this report's context.

Environment's Entities

In Gazebo's simulation, we get presented with a blank environment to build upon. We use Computer Aided Design (CAD) to create required entities and obstacles. In our implementation, we had to develop boxes, tiles and borders to the environment. The model is saved in *sde* format, and rendered in Gazebo. Each box is placed at a specific position, and the borders are placed to enclose the environment. The whole entity is made over a sheet of tile. The Gazebo environment with a turtlebot is demonstrated in the Figure 2.2

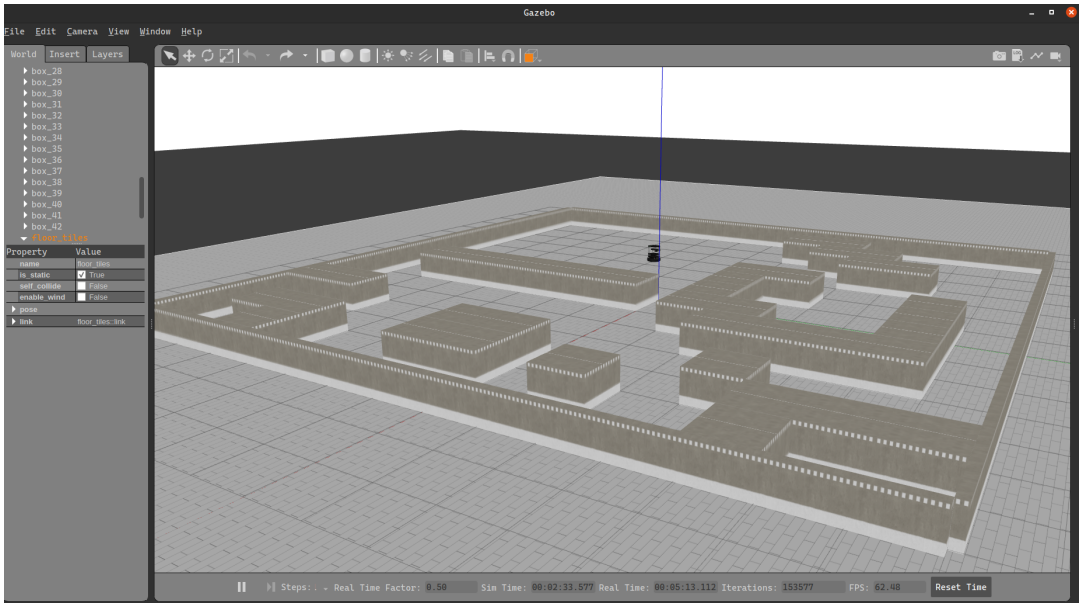

FIGURE 2.2: Turtlebot2 in Gazebo Environment

2.4 Robot Operating System

Robot Operating System or ROS, is known for it's flexibility as a framework to write and simulate robot software. ROS has a collection of open-source tools and libraries are used to run robots. It is very helpful in the field of robotics as it reduces the hassle of writing and figuring most of the robot's driver and actuator code. It is at a higher level in abstraction, thus providing an option to use the same code in simulation in Gazebo as well as to run the real robot. It provides hardware abstraction, libraries, device drivers, package management and more. There are various different versions of ROS, each combined with a specific version of Ubuntu's Long Term Service distribution. As the software we wish to build is supposed to be utilized later, a stable framework is a neccesity. Ubuntu 20.04 LTS and ROS Noetic were chosen as distributions for the project.

2.4.1 ROS Packages

In ROS, we have different packages for differnt softwares. A package is created by specific ROS commands. A ROS Package structure is :

- *include/package name*: C++ scripts include headers which are stored in this directory.
- *msg* : This folder contain msg types, i.e the types of messages being interchanged in processes.
- *src* : This folder contains the source file for the package.
- *srv* : This folder contains files which show the simplified service description languages.
- *scripts* : This folder contains the scripts (in python) required to run the robot or do a specific task.
- *CMakeLists.txt* : This file describes the ROS Package and helps the developer to specify runtime and execution time dependancies to build the catkin project.
- *package.xml* : Describe the dependancies of the package.
- *CHANGELOG.srt* : Many packages will define a changelog which can be automatically injected into binary packaging and into the wiki page for the package

2.4.2 ROS Command Line Tools

Packages are a central concept that are used to manage different software. Some command line tools help you to manage such packages :

- *rospack* : finds and retrieves information about the package.
- *catkin-create-pkg* : creates a new ROS package.
- *catkin-make* : build the workspace containing different ROS packages.
- *rosdep* : to install the system dependencies of the package.
- *rqt* : RQT has a a plugin called "Introspection/Package Graph", which visualizes package dependencies as a graph.

2.5 Simulataneous Localization and Mapping

As we studied in the previous chapter about localization, which is a process to inform the robot about it's position in the environment it is in. Simultaneous localization and mapping (SLAM) is a technique applied in artificial intelligence mobile robot for a self-exploration in numerous geographical environment. SLAM becomes fundamental research area in recent days as it promising solution in solving most of problems which related to the self-exploratory oriented artificial intelligence mobile robot field [5]. SLAM is one of the most fundamental part of autonomous mobile robot navigation. It is used by the mobile robot to navigate around the environment and generate a map. The map generated is used to find the obstacle location and grid surrounding the robot to make an appropriate path planning scenario for the robot. Although Localization and Mapping are different technologies, SLAM

incorporates them together where localization is done concurrently with the robot creating the map. The map is used to note than landmark position and be able to generate an efficient path trajectory. SLAM has the advantage of doing this together at the same time making it the most important algorithm in autonomous robot navigation. SLAM methods implement Extended Kalman filters. Figure 2.3 demonstrates the SLAM process.

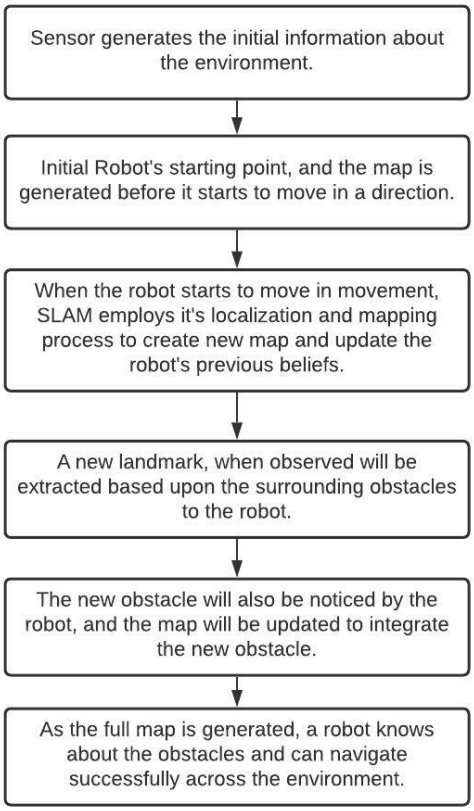

FIGURE 2.3: SLAM Workflow [5]

2.5.1 Features of SLAM

Three major features of SLAM exist and are discussed below :

Mapping

For a robot to move around autonomously in an environment. It should know where it is, providing prior knowledge. Mapping provides his capability for the robot to generate a map using the hardware sensor (mostly LIDAR) to get information from the environment. There are various ways to represent the map, which are used to generate a path in subsequent steps. The data will be used by the robot to localize and recognize it's own position and landmark.

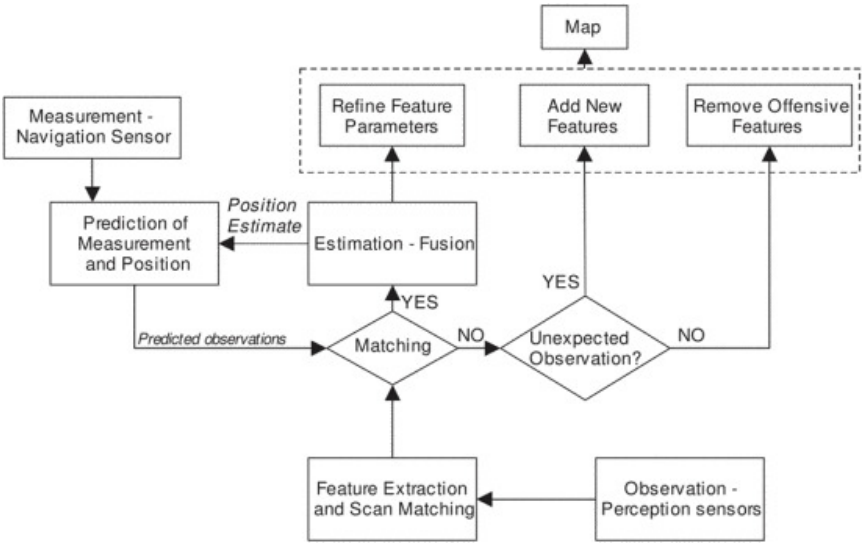

FIGURE 2.4: SLAM Block Diagram [2]

Localization

Localization is calculated by using the odometry of the robot. It is one of the features of SLAM which are able to calculate and estimate the poisition of the robot, and the obstacles close to it. The mobile robot should be able to *think* for itself by calculating and estimating the path travelled and calculating the nearby obstacles.

Navigation

Navigation employs both Mapping and Localization, by combining the results of both. When the robot navigates through the environment, mapping and localization process are executed periodically and repeatedly recursively to update the robot's information from the environment and saves the map. The robot's exploration is implemented and is able to backtrack to origin point or starting position. Path Planning is done with various algorithms, such as Dijkstra, A*, Potential Field and other algorithms. In our implementation, we used different algorithms which we will describe in 2.6.1

2.5.2 Isssues in SLAM

As every method, SLAM has it's owm issues which are uncertainty, correspondence and data association.

Uncertainity

There are both location and hardware uncertainity in SLAM which affect the performance of robot while performance. Uncertainity in location determines how capable the robot is to handle multiple paths in environment's location. For a robot to move in a linear direction, and come back to the same position, the path planning is very easy without any uncertainty because simplicity in motion. In real world, as well as in our implementation the robot is subjected to different paths in the environment. Therefore, in this case location's uncertainity as well as the final trajectory of the

path is often tough and included with a degree of uncertainity. Inaccurate information can be corrected in the post treatment of the data and processed to recognize the robot and obstacles nearby.

Correspondence

Identification of the obstacles in the environment is a problem in SLAM. If we consider two different obstacles, with similar shapes but one obstacle is slightly bigger than the other in height, the light radiated from the obstacle when LIDAR will hit shows the same obstacle for both. In such cases, SLAM is not able to update robot's map.

Data Association

In data association issues, the SLAM is not accurate in returning to it's original point and previously mapped area. Data association processes are used to estimate the odometry frame of the mobile robot to backtrack from where it came from i.e, it's original state and nearby obstacles.

Time Complexity

This issue deals with implementation of SLAM algorithms and methods employed to calculate and compute the information received from the sensors. In SLAM the process of localization and mapping are carried out together recursively during navigation in a direction. The processes are executed concurrently in a very short amount of time, the handling and management of the values from the sensors are not managed and calculated efficiently. Therefore the performance and time complexity are important to produce reliable results for the robot to explore the environment.

2.6 Path Planning Methods

The previous chapter has introduced the term *motion planning* and *path planning* to us. Path plannng algorithms are used to take the design of the workspace and obstacle space to return the path. The robot follows the path which is returned by the algorithm. A such algorithm is called a *planner*. According to Steven M. LaValle's definition of a planner [7] : "A planner simply constructs a plan and may be a machine or a human. If the planner is a machine, it will generally be considered as a planning algorithm. In many circumstances it is an algorithm in the strict Turing sense, however, this is not necessary. In some cases, humans become planners by developing a plan that works in all situations"

There are two types of path planning algorithms :

- Global Planner Algorithms
- Local Planner Algorithms

2.6.1 Global Planner Algorithms

Global Path Planner utilises the definition of configuration space of the robot which is based upon the global map. The global path planning algorithms are known as offline algorithms because the calculation of path is done before the robot moves. Robot's configuration space is used to calculate a path to the final state. ROS is

utilized to convert the map file which was generated by SLAM to divide the map into square boxes and classifying them into either obstacle or a free configuration space.

General Idea

After SLAM, the made map file is generated and provided to the navigation stack to describe the configuration space. ROS uses the concept of global costmaps to divide the map into squares based upon it's resolution and the value to each square given in the range of 0-255. 0 signifies a free space while any value greater than one is an obstacle space. We use this information to decide and calculate the path taken by our robot to reach the goal. Thus, after describing the configuration space we will use Graphs to calculate a viable path to the final destination of the robot. There are different paths to consider at each and every node in the graph. A path is considered if and only if, a particular next node provides the minimum cost or a user/task specific requirement. Different Algorithms we implemented in the project provide a different method to solve this problem and predict the next states of the robot to follow. Figure 2.5 shows all the possible paths to reach the goal.

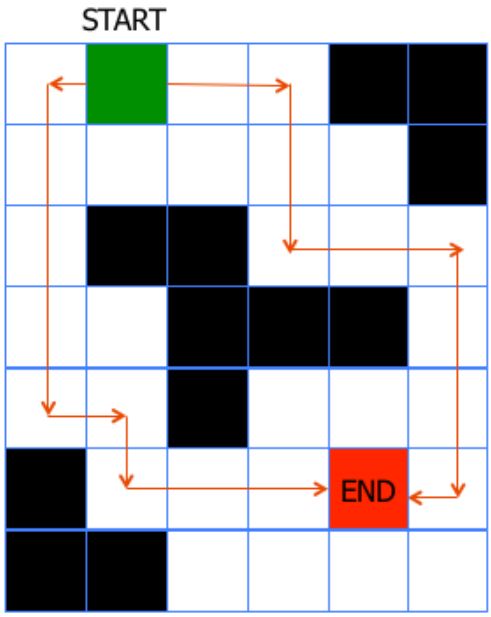

FIGURE 2.5: Path Planning Grid

Dijkstra Algorithm

Dijkstra's algorithm is a very generic way to find the path to the goal node which are connected by edges if we consider the configuration space to be a graph. Dijkstra's algorithm was developed by Edsger W. Dijkstra [11]. The algorithm was proposed to solve the path planning by finding the minimum cost path.

A* Algorithm

A* algorithm is pretty similar to Dijkstra's algorithm but it has increased performance. Dijkstra's algorithm is a new approach as it starts to explore the global obstacle grid with a starting node and doesn't stop till it reaches the ending node. It will identify the list of nodes which connect the end node to the starting node to form a path. But there is an issue with Dijkstra, which requires it to explore all the nodes in an equal fashion. This is fine if computation speed and space is not a requirement. But exploring all the nodes equally increases the time complexity of the algorithm. If the map is big, using Dijkstra's algorithm can be very computationally expensive. We use A* algorithm to improve the efficiency of search and finding the path. A* does this by introducing the concept of a *heuristic function* which will help the node exploration towards the goal. The function calculates a correlation of the node to a cost value (a positive value). Heuristic functions are calculated by satisfying these basic criteria.

- H(goal) = 0
- H(x) <= H(y) + cost(x,y), where cost(x,y) is the cost to travel from one node to the other and x,y are the two adjecent nodes.

By keeping the cost of the goal node 0, it helps us increase the cost efficiency because we do not have a direction. Thus, if the cost of travelling from a node to the other can bring us closer to the goal we use it as it increases our exploration time. Heuristic function cost(x,y) can be calculated by using either Euclidian Distance or Manhattan Distance depending on one's preference.

Greedy Best First Search Algorithm

Greedy Best-First Search (Greedy Best-First Search) Algorithm is a search algorithm. The configuration space is represented as a graph of different nodes. The algorithm will explore the whole graph by expanding the nodes which can have promising next nodes according to a rule. Pearl J. [3] defines GBFS as an estimation of the most promising next node N by a heuristic evaluation function *f(n)* which, in general, depends on the description of N, the information of the goal, the information gathered by the process upto this point, and other knowledge about a problem domain. Configuration space for GBFS algorithm is accomplished by using Visibility Graph Method. The method states that the path is defined as a line connecting the starting configuration with the ending goal configuration by employing the non-free configuration space. As we define the non-free configuration paths, possible non-collision paths between the start and goal positions are defined too. As the information about the robot's workspace increases, the quality of the algorithm increases too. The algorithm will work better when we have much more information about the robot's workspace.

2.6.2 Local Planner Algorithms

Local Planners are dynamic path planning algorithm, which means that the algorithm works online while the robot is moving, unlike global planner algorithms where the next states are already decided and the robot follows the specific path. Local Planner Algorithms require constant update and calculation of the values recieved from sensors, wheather it is a LIDAR sensor or a camera vision sensor. The

algorithm computes those values returned to predict a short term relapse path towards the goal. There are several local planner algorithms such as *Dynaamic Window Approach (DWA) local planner*, *Time-Elastic-Band (TEB) local planner* already implemented in the ROS package, which we will discuss about in this section. Local Planner also helps the robot to maintain it's path that was calculated earlier by the global planner algorithm. In this subsection, we will discuss about the local path planning algorithms.

TEB local planner

Configuration space's representation in TEB local planner is done by using a *Sampling-Based approach*. The planner uses a novel approach named *Time-Elastic-Band algoritm* for obstacle avoidance purposes.The algorithm is described in the paper [4]. The motion planning is to move the robot from A(x_o, y_o) to B(x_N, y_N) without any collision with relation to the global and local reference frames. The local path planner provides a turn in the angle of rotation in y axis and a forward motion to avoid the obstacle.

TEB (see Figure 2.7) is done by describing a fixed number of points to the final goal, (x_0, y_0), (x_1, y_1), $(x_2, y_2) \ldots (x_N, y_N)$. The different points are put apart by a time interval. The time interval is the difference in travelling two points. Time Elastic Band is made up from two different set of points and the time interval, i.e B = (Q,T) and the time band is computed by minimizing a sum of multiple objectives and cost functions to find the new path to get to the final position. The differences betweent the consecutive points are used to calculate the velocity and acceleration for the robot.

2.6.3 DWA local planner

DWA local planner (see Figure 2.6) stands for using *Dynamic Window Approach* to navigation in a regular updating fashion on the surface. After the global path planner gives a path and a costmap configuration space, the local planner will send the velocity to the robot to avoid any obstacles. DWA local planner utilises the BaseLocalPlanner interface which is specified in the navigation package. The *dwa local planner* provides a controller to control the mobile motor of the robot. The controller will connect the path planning process to the robot. The robot uses a map to create a planner trajectory to get from the start to goal location. Grid is created around the robot using the LIDAR sensor. The grid cells are utilized to calculate cost value, and the controller will use utilize the value function to send the values of velocity_x, velocity_y and velocity rotation theta to the robot.

Major ideas in Dynamic Window Approach (DWA) are described below :

- Getting values from robot's LIDAR sensor and discretely calculating the robot's control space.
- Velocity is calculated from the function to calculate the current state and predicting the next sampled velocity with next state in case of deflection with an obstacle.
- The velocity will be applied for a very short period of time.
- Different trajectories are generated to avoid the obstacle.
- The trajectory with the highest score and providing the lowest cost to move towards the goal is chosen.

- The trajectory is chosen, and the associated velocity is sent to the motor by the controller.
- You repeat the algorithm everytime you find an obstacle.

Our choice was to use Dijkstra algorithm for the global planner algorithm and dynamic window approach for the local planner algorithm. Implementation for A* and GBFS algorithm is also done, and can be implemented based upon preference.

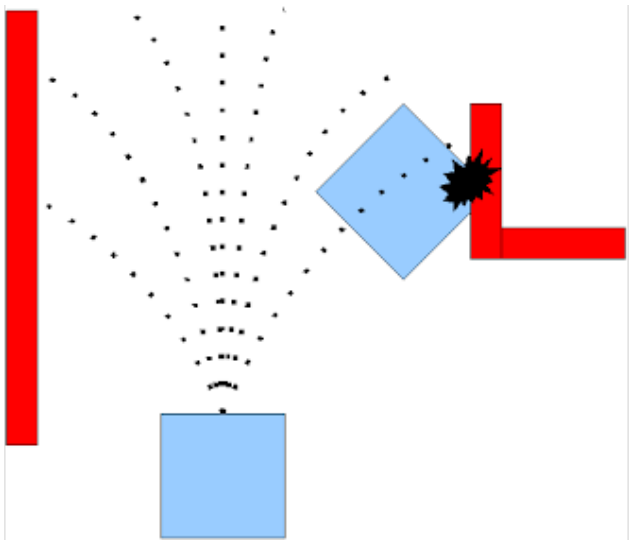

FIGURE 2.6: DWA Planner

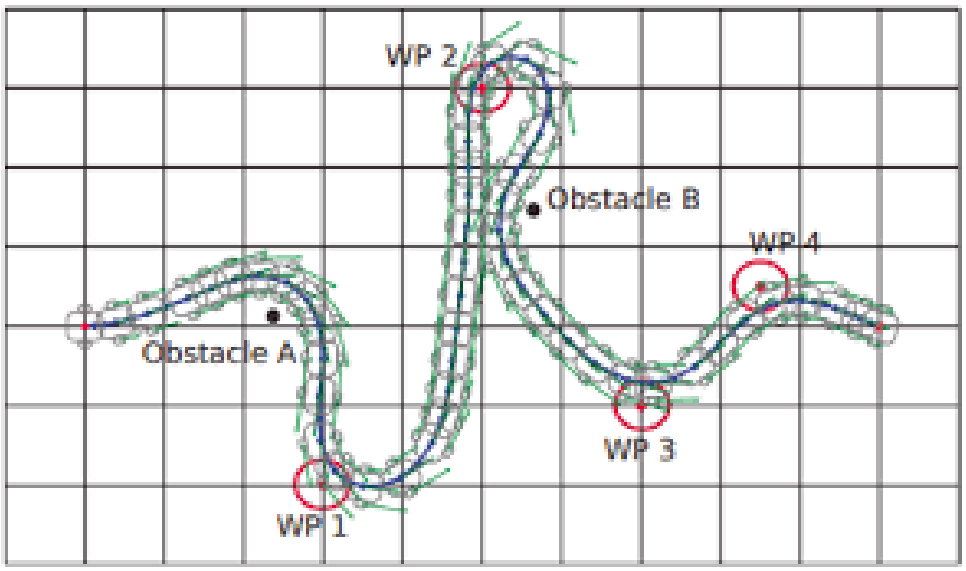

FIGURE 2.7: TEB Planner

Chapter 3

Implementation

3.1 Introduction

In the previous chapters, we discussed about the context of the work and the technologies, both software and hardware required to execute the project. In this chapter, we would go through methods and steps with which we can utilise those technologies to produce results. In our implementation, we wish to study the behavior of robot with respect to it's neighbourhood obstacles. To achieve studying of goals, we will use the relative position of the robot with respect to the obsatcles and result in a log file for data mining purposes.

3.2 Environment Development

The simulation environment used in the project is Gazebo11 [6]. Gazebo is a simulation environment where we create a DT [1]. ROS framework utilizes Gazebo as the simulation environment for both our robot as well as the obsatcle environment. To simply simulate the robot into Gazebo, we need to define *launch* files, which contain arguments to define the robot and it's specifications. In a launch file, we define the initial position of the robot, the world file we might use, import the neccesary equipments of the robots and define the sensors of the robot, it's base, and the manufacturer of each. The equipments are defined by using *xacro* files which are saved in the same directory. We can also call other processes and scripts to run, as well as visualization in the launch file. The *launch* file also helps us define and call **AMCL** launch file, which helps us localize the robot in the environment frame. The launch files contain arguments to visualize the processes with RViz (ROS Visualization Tool). The launch file is programmed in XML-like structure. In the project, we have floor tiles, square boxes and walls surrounding the environment. These entities can be created in CAD and imported as we intend to implement. These entities are stored in *.gazebo* directory and therefore can be used to develop other environments with different configurations. The digital twin developed along with the robot is noted and studied from the top at *god's eye view*. In ROS package, launch folders contain the launch files, which call models from models and world directory. Note that, models are building blocks of world files. World files with different configurations are stored in *worlds* directory in the ROS package.

3.3 SLAM

SLAM (Simultaneous Localization and Mapping) is done by simulating the turtlebot in a particular environment. As the robot does not know anything about the environment, a *teleoperation* command, i.e using the keyboard to move the robot

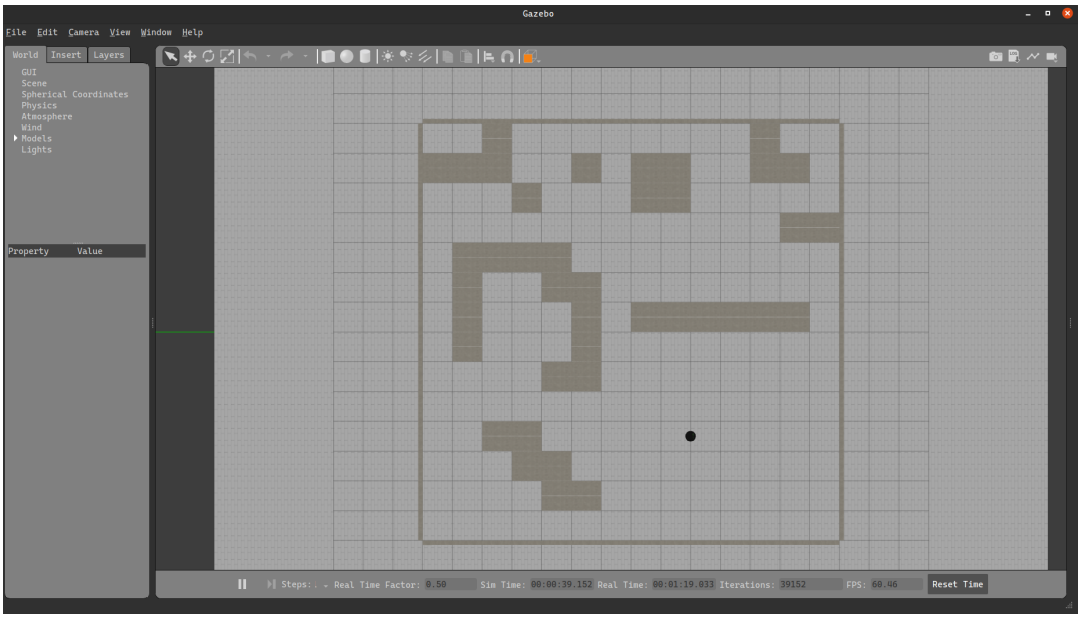

FIGURE 3.1: View of 14x14 grid

around the environment. As the robot has a 360º LIDAR sensor, concurrent mapping and localization is done. The map is generated and saved for further global path planning and navigation.The user, controlling the robot stop the teleoperation will stop moving around the robot when the mapping is done completely. In SLAM methods in ROS, we can use different methods to execute SLAM such as "gmappping" (`wiki.ros.org/gmapping`) which uses a two dimensional occupancy grid map describing the configuration space. Another such method is "cartographer" (`wiki.ros.org/cartographer`) utilizing an algorithm employing loop closure method. In our work, we implemented the *gmapping* package to implement SLAM see Figure 3.3.

- Simulate the robot in environment with *launch* files.
- Using turtlebot package to initiate SLAM
- Starting the teleoperation node in another terminal.
- Moving the robot around the environment to record the map.
- Saving the map.

In the Figure 3.3, the robot is moving in (1) using the teleoperation node i.e controlled by the keyboard. As the robot moves around and covers the environment, the map is generated.

3.4 Navigation

Navigation in ROS based environment is done through the navigation stack. Navigation stack is an open-source framework in ROS which accompanies everthing required for a robot to move from a point to the other. The navigation stack is completely configurable, and developer can add their code into the stack when we wish

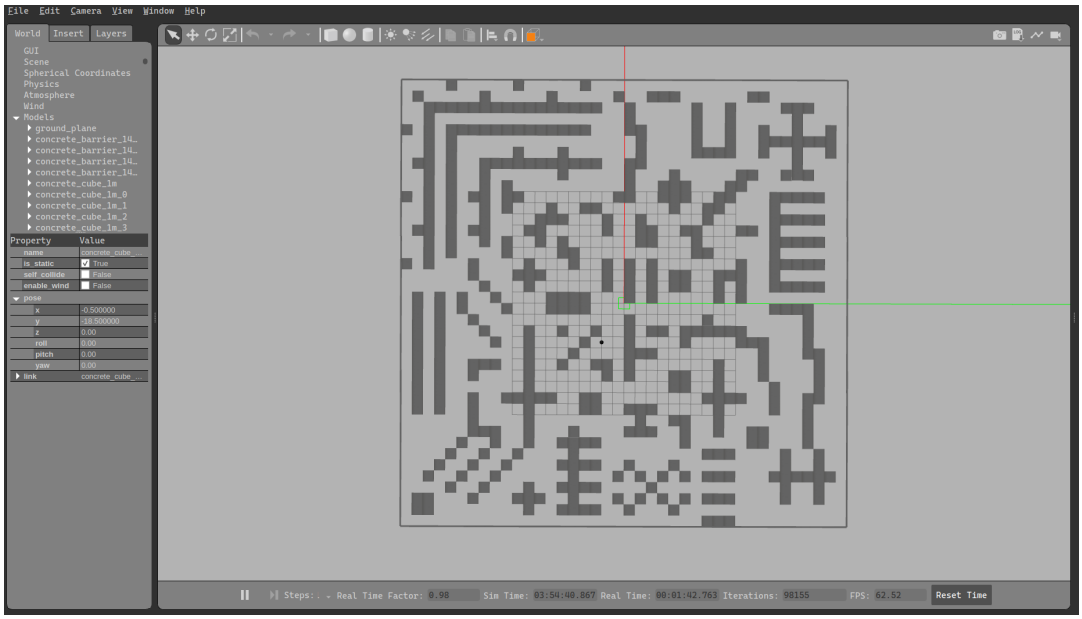

FIGURE 3.2: View of 30x30 grid

to. Navigation stack is available on Github, for anyone to use and utilise. ROS Navigation stack is on a conceptual level. The information is taken from odometry, and goal position of the robot and sensors. Navigation stack deals with publishing velocity to the robot controller, the values will be published according to the need to reach the goal position. There are many prerequisites in Navigation Stack :

- The robot should be running ROS.
- coordinate frame of the robot, i.e *tf transforms* should be known.
- global plan coordinate frame of the robot should be known.
- correct *message* files to exchange information in between processes.

To fully setup the essential navigation stack, it's configuration can be done through `http://wiki.ros.org/navigation` In our implementation, we are using the local path planner (DWA Local Path Planner) but we are not employing the LIDAR sensor to avoid obstacle dynamically. The robot in our implementation, does not intend to path plan in a dynamic fashion. Instead, we wish to read the behaviour of the robot in relation with it's global path plan. In the data flow in Navigation Stack, odometry and LIDAR are the chosen combination. Odometry is the most important information in the control. As we wish to *see and record* the behavior of the robot from a god's eye view (see Figure 3.4), we will only use Odometry information to see the x *and* y values of the robot in relation to the coordinate frame. The goal position is chosen from using the ROS Visualization tool. Navigation stack will be turned on to work, when a goal position is chosen. The navigation then publishes the trajectory position and velocity from the robot's controller. The transform tree, is utilized to specify the motion constraints of the robot like it's height and width of the mobile base. There are various such parameters attached in the *config* folder describing different patterns in numerous *.yaml* files.

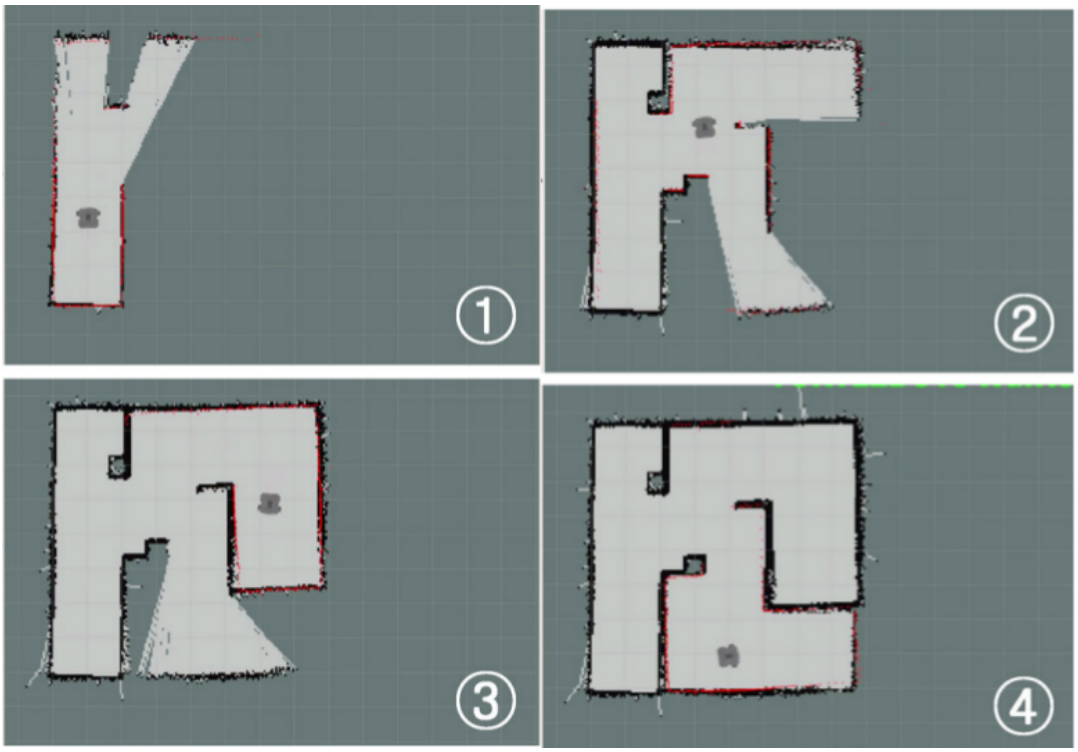

FIGURE 3.3: Step-by-Step SLAM

3.4.1 Customized Global Planner Algorithm

In our implementation, the robot's possible movements are only in 4 directions. We don't wish the optimization of the path, instead we wish to focus on defining the movement in those directions. Thus, after defining the *global costmap* and it's grids. The algorithms should only iterate and explore over the *top, left, right and down* nodes only at each and every step. In the Figure 3.5, the node **R** is an obstacle, which is describe as a value greater than 1 in global costmap configuration. Thus, the robot's global planner can not explore that node and further nodes that can exist and explored from using **R** as a referemce point. As the nodes **L,P and V** are free, in the next step the robot's global planner will proceed to explore the other nodes built upon those. Let's consider that **L** was node chosen in order to move to the goal position. We will proceed to explore the nodes adjecent to the **L** node. In the Figure 3.6, the node L has 4 neighbours, namely **K,G,M,Q**. As **G and M** are obstacles with **Q** being the previous explored node. We have no other option to chose node K as our next node.

Strategies of Planning

As we discussed that, there are three algorithms implemented in our project which are namely, *Dijkstra's Algorithm, A* Algorithm and GBFS algorithm*. These three algorithms will follow a similar node exploration strategy but the incentives to explore a new node will differ from algorithm to algorithm, resulting in different strategies to reach the goal position node.

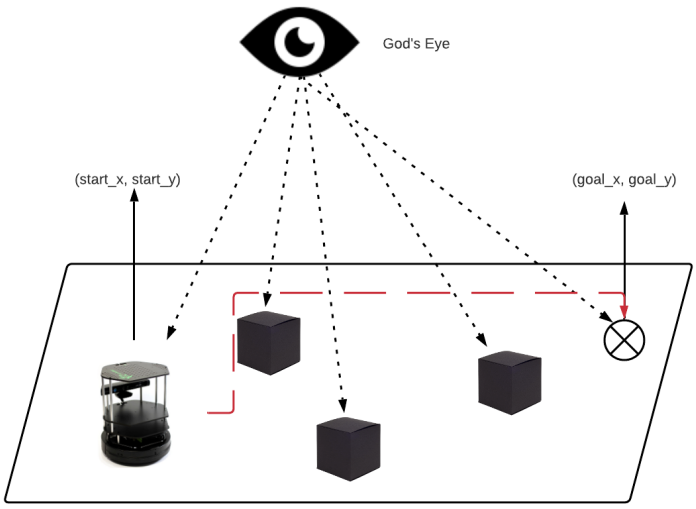

FIGURE 3.4: God's eye view on Digital Twin

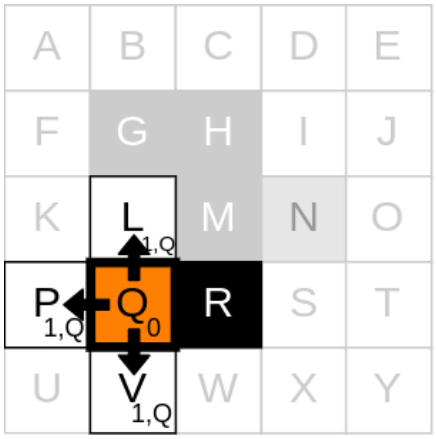

FIGURE 3.5: Exploration Strategy of Nodes

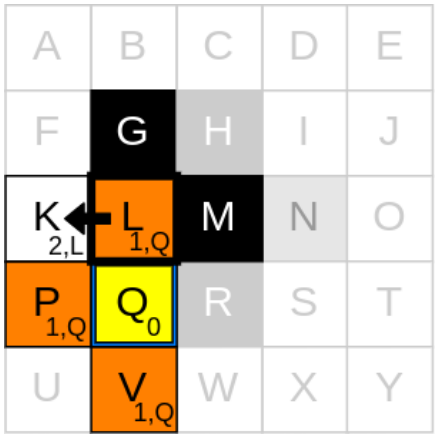

FIGURE 3.6: Next stage of exploration.

Chapter 4

Solution to Log Generation

4.1 Introduction

In the previous chapters, we have discussed the method to implement the solution to generate a path. Path Generation and Implementation of a method to store the behaviour of the robot with respect to it's neighbours is another issue. In this chapter, we will cover the method implemented for generation of the log file with respect to the nearby obstacles.

4.2 Constraints

As we are implementing the solution by following the hypothesis that we wish to see and generate the log in the Digital Twin from the perspective of the God's eye. We can not use any information that has been published by the robot. Thus, we can not use *scan* topic and sensors to infer data and proceed with data mining. Therefore, we are only limited to *'use what we can see'*. Odometry describe the position of the robot in the context of the environment. From the god's eye perspective, we can only see the positional information or *position* of the robot. Therefore, to solve the log generation problem we will only use the **x** and **y** positions of the robot.

4.3 Using Odometry

Odometry is a *topic* in ROS which describes the postion of the robot in the coorrdinate system. Odometry message type contains :

- **std_msgs/Header *header***, providing a header to the information.
- **string child_frame_id**, providing the ID to the frame generated from positional transforms of the robot model.
- **PoseWithCovariance**, describing the linear positional information about the robot i.e the x,y,z position of the robot in the coorrdinate frame.
- **TwistWithCovariance**, describing the angular positional information about the robot i.e the x,y,z angular rotation in the coorrdinate frame.

Odometry is the best choice for generation of log file in our implementation. This is because, in our hypothesis, the values should be recorded from the god's eye view (see Figure 3.4). Implementation can be done from the robot's LIDAR sensor, although we will get the values from the robot. This implementation will deviate us from our hypothesis. In implementation with the LIDAR sensor, we found out that the values were not regular. This could be due to incompetency of the LIDAR

sensor. We needed to find a different definitive method, using values from odometry. In the Figure 4.1, the light grey grid is generated with LIDAR sensor. The process is continous, and not positional. This results in varying discrepency in the data generated. We recorded loss in data, making this approach unusable and further solidifying the use of odometry.

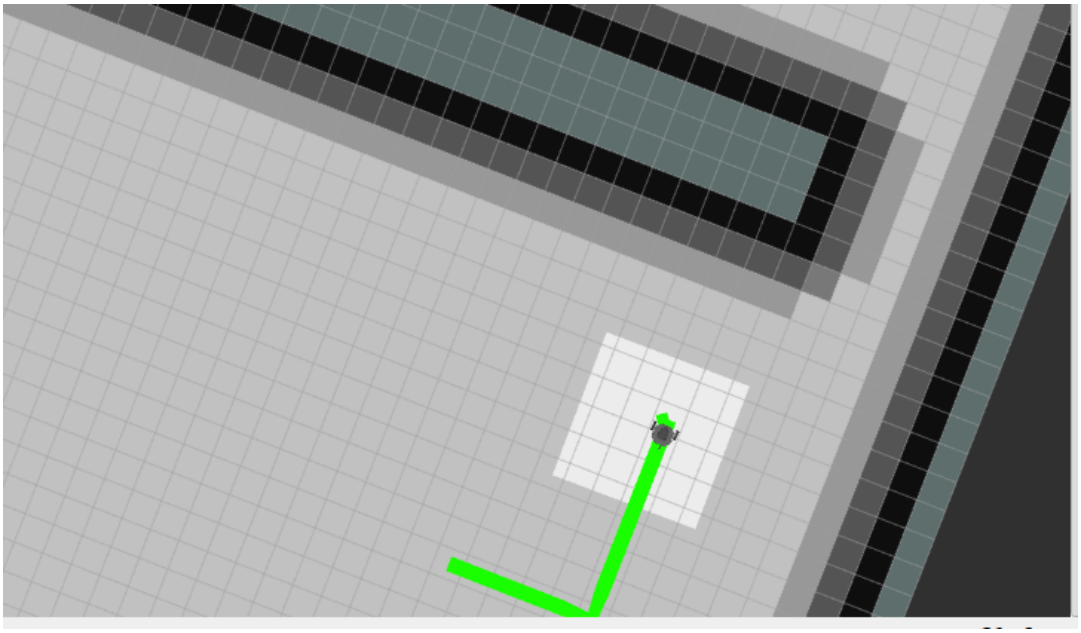

FIGURE 4.1: Grid Generated by Robot

4.4 Solution

4.4.1 Obstacle Space

While generation of the environment in Gazebo, every obstacle is given a position in it. The position of the obstacle is also described by using **x** and **y** positions of the obstacle. Thus, if we have 44 boxes, we should have 44 different pairs of **x** and **y** positions for the obstacle space.

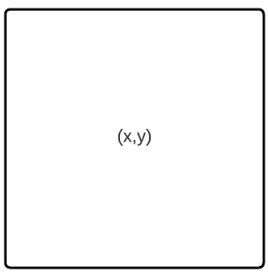

FIGURE 4.2: Obstacle positioning

We implemented the log generation algorithm using just 44 different boxes and 44 pairs of **x** and **y** positions. We rounded the **x** and **y** position to the nearest *point*

5 To generate a 3 by 3 grid, we calculated if the **x** and **y** poisitions and adding 0.5 to the values collides with the center of the robot. The method was basic, produced results but was not accurate and thus was not suitable for our implementation. We had to increase the number of contact points and change the algorithm, see 4.5.

Correlation of obstacle with robot

In our digital twin, we wish to use the Odometry positions of the robot as well as the obstacles around the robot. To find if there is an obstacle near the robot, we will have to find a way to correlate the both positions. A simple algorithm to do so could be,

- Finding the x, y position of the robot.
- Using the distance of the center of the obstacle with it's borders.
- Adding the particular value to the obstacle's position
- If it is equal, obstacle is there else not.
- Recording this behaviour.

The algorithm can be used to produce the obstacle space in a grid around the robot. The algorithm is simple in it's definition but lacks in many use cases. For example, if the robot just passes around the corner of the obstacle, the algorithm will not be able to show if the position is close to an obstacle or not. Thus, it needs improvement.

The improvement should be noted in correlation to the constraints so solve the problem, i.e by only using the position of the robot as well as the obstacle.

4.5 Final Solution

In our implementation, the length of the obstacle is 1 unit. Since the obstacle is a square, the length as well as the breadth are 1 unit each. There is only one position of the obstalcle i.e at the middle (see Figure 4.2). We need to improve on the position, and create more pairs of positions where the obstacle is. We approached this problem by dividing a square into 8 parts with different positional points. The obstacle's distance from the center to an edge is **0.5 units**. In order to improve and give more information about the obstacle space to the robot, we had to divide the obstacle into 8 lines with 5 points on each, signifying the coordinates of the obstacle space. Each of these points are situated at **0.1 unit** distance from one another.

As seen in Figure 4.3, there are multiple points described in an obstacle space as compared to just one in the previous approach. Eventhough, we are not using any sensors to get the information, we can name a *perception* range defining the points the robot will check in order to find a point that is present in the obstacle space. There are 44 obstacles with 72 points each, which means that the obstacle space data structure (python list in our implementation) contains, 3168 pairs of **x** and **y** positions. As the robot will move across the configuration space, the x and y coordinates of the robot will be used to calculate a nearby obsatcle point, if it exists based upon the *perception range* defined in the algorithm. The obstacle space is saved in a list, and the algorithm will find a point in the perception range of the robot and if it is present in the obstacle space, it will denote it as an obstacle. Another list will is created, which saves the position values in different grid sizes based upon the perception

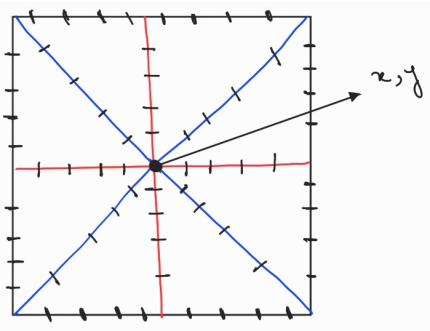

FIGURE 4.3: Divided Obstacle Space

range of the robot. For example, in a *3 by 3 grid*, the different positions are written to the list, and further to a semi-structured csv file. The log file is then to be used for data mining and reinforcement learning applications. The log will contain :

- ROS timestamp in seconds.
- x and y coordinate of the robot.
- the obstacle grid (*any size*) calculated by the algorithm.
- x and y coordinates of the goal.
- relative x and y coordinates to the goal
- manhattan distance of the robot to the goal.
- euclidian distance of the robot to the goal.
- angular orientation in z axis i.e the rotation of the robot.

4.6 Conclusion and Perspective

In this work, we had to develop a method to record the behavior of the robot in an environment with obstacles. We developed a method to navigate a robot from a position to goal position. We implemented different methods to generate the log file, and decide upon using positional values of robot as well as environment's entities to develop the obstacle space around the robot. The advantage of the method is we can expand it to any size of the grid. To implement this, I had to learn about autonomous navigation of the robot in an environment with ROS. I learnt about digital twins and methods to implement digital twins for behavioural learning. The goal of the work was to study the behavior of the robot around different obstacles. Development of navigation methods and an algorithm to infer the data to produce logs was required in completion. By producing the method for behavioral learning, the work can be expanded for multiple robots as digital twins to learn their behaviour in relation to one another. The generated log file can be used for machine learning, training or testing the developed strategy with digital twins.

Appendix A

Parameters for Navigation

A.1 Robot's Parameters

TABLE A.1: Parameters for Costmap

Parameter	Value
max_obstacle_height	0.60
robot_radius	0.17
obstacle_layer	:
enabled	true
max_obstacle_height	0.6
origin_z	0.0
z_resolution	0.2
z_vowels	10
unknown_threshold	15
mark_threshold	0
combination_method	: 1
track_unknown_space	true
obstacle_range : 5.5	
raytrace_range : 6.0	
origin_z	0.0
publish_voxel_map	false
observation_sources	scan
inflation_layer	:
enabled	true
cost_scaling_factor :	
5.0	
inflation_radius : 0.4	
static_layer	:
enabled	true

TABLE A.2: Global Costmap Parameters

Parameter	Value
global_frame	map
robot_base_frame	base_footprint
update_frequency	0.2
publish_frequency	0.2
transform_tolerance	0.5

TABLE A.3: Global Planner Parameters

Parameter	Value
old_navfn_behavior	false
use_quadratic	true
use_dijkstra	true
use_grid_path	false
allow_unknown	true
planner_window_x	0.0
planner_window_y	0.0
default_tolerance	0.0
publish_scale	100
lethal_cost	253
neutral_cost	50
cost_factor	3.0
publish_potential	true

Bibliography

[1] Barbara Rita Barricelli, Elena Casiraghi, and Daniela Fogli. "A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications". In: *IEEE Access* 7 (2019), pp. 167653–167671. DOI: 10.1109/ACCESS.2019.2953499.

[2] Franco Hidalgo and Thomas Bräunl. "Review of underwater SLAM techniques". In: *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*. Feb. 2015, pp. 306–311. DOI: 10.1109/ICARA.2015.7081165.

[3] Pearl J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, p. 48.

[4] Martin Keller et al. "Planning of Optimal Collision Avoidance Trajectories with Timed Elastic Bands". In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 9822–9827. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20140824-6-ZA-1003.01143. URL: https://www.sciencedirect.com/science/article/pii/S1474667016431691.

[5] Alif Ridzuan Khairuddin, Mohamad Shukor Talib, and Habibollah Haron. "Review on simultaneous localization and mapping (SLAM)". In: *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*. 2015, pp. 85–90. DOI: 10.1109/ICCSCE.2015.7482163.

[6] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[7] Steven M.LaValle. *Planning Algorithms*. 2006, pp. 20–22.

[8] C. Sung. "Computational design of foldable robots via composition". In: 2016. URL: https://dspace.mit.edu/handle/1721.1/113734.

[9] Wikipedia contributors. *Control theory — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Control_theory&oldid=1039308545. [Online; accessed 21-August-2021]. 2021.

[10] Wikipedia contributors. *Cybernetics — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Cybernetics&oldid=1037814219. [Online; accessed 22-August-2021]. 2021.

[11] Wikipedia contributors. *Edsger W. Dijkstra — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-August-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Edsger_W._Dijkstra&oldid=1039157161.

[12] Wise and Foote. *Turtlebot2 - Open-source robot development kit for apps on wheels*. https://https://www.turtlebot.com/turtlebot2/. [Online, accessed 24-August-2021]. 2021.