Reducing Operational Cost of an Autonomous Robot Using Combined Data-Driven and Navigational Algorithms

by

Xu Ting (Pamela) She

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Applied Science in Mechanical Engineering

Department of Automotive and Mechatronics Engineering

Faculty of Engineering and Applied Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

April 2021

THESIS EXAMINATION INFORMATION

Submitted by: **Xu Ting (Pamela) She**

Masters of Applied Science in **Mechanical Engineering**

Thesis title: Reducing Operational Cost of an Autonomous Robot Using Combined Data-Driven and Navigational Algorithms

An oral defence of this thesis took place on April 19, 2021 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Martin Agelin-Chaab
Research Supervisor	Dr. Xianke Lin
Research Co-supervisor	Dr. Haoxiang Lang
Examining Committee Member	Dr. Jaho Seo
Thesis Examiner	Dr. Jing Ren

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

Autonomous mobile robots have become an important presence in various fields of work, but are restricted by short operation times due to the limitations of their onboard energy sources. This factor bottlenecks applications of mobile robots in many situations. In this thesis, a data-driven power consumption model is developed based on the motion data of the robot. A path planning algorithm combining the rapid-exploring random tree and artificial potential field was developed for navigation. From this, a combined framework utilizing model predictive control as the control strategy to optimize motion and conserve power is proposed. The algorithm is implemented in a popular research platform, TurtleBot3, for validation in a dynamic environment. These tests were conducted on a flat surface using five different obstacle configurations, with obstacles being hidden from initial detection. The experimentation results demonstrate the effectiveness of the proposed algorithm framework in reducing the required power and calculation time.

Keywords: Mobile Robot; Power Consumption; MPC; Path-planning; Data-driven

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

Xu She

STATEMENT OF CONTRIBUTIONS

Part of the work described in Chapter 2 and 4 has been published as:

X. T. P. She, X. Lin, and H. Lang, “A Data-Driven Power Consumption Model for Electric UAVs,” in *2020 American Control Conference (ACC)*, Denver, CO, USA, 72020, pp. 4957–4962.

I performed the writing of the manuscript, platform set up, platform data collection, and comparison tests of results. Review and technical support were provided by Co-authors.

ACKNOWLEDGEMENTS

I give my thanks to Dr. Xianke Lin for his support and hard work in guiding me through this thesis.

I give my thanks to Dr. Haoxiang Lang for providing me with the opportunity in getting started with this work.

I give thanks to my parents and friends for their love and support in getting me through everything.

TABLE OF CONTENTS

LIST OF TABLES

Chapter 3

Chapter 4

LIST OF FIGURES

Chapter 1

Chapter 2

Chapter 3

Chapter 4

LIST OF ABBREVIATIONS AND SYMBOLS

RRT	Rapidly-exploring Rapid Tree
ROS	Robot Operating System
NMPC	Non-linear Model Predictive Control
USV	Unmanned Surface Vehicles
AUV	Autonomous Underwater Vehicles
UAV	Unmanned Aerial Vehicles
PWM	Pulse-Width Modulation
LiPo	Lithium Polymer
Lidar	Light Distance and Range
PRM	Probabilistic Roadmap
v	Linear Velocity
ω	Angular Velocity
V	Voltage
I	Current
NN	Neural Network

Chapter 1. Introduction

1.1 Background and Motivation

Automation is becoming increasingly relevant in modern-day life. It is now prevalent not only in industrial spheres but also in the military, in agriculture, and within the home. For years we have been utilizing automation to increase the quality of human life [1]. Productivity, accuracy, and efficiency have improved while simultaneously reducing risk to human health caused by monotonous physical work or occupational hazards. The applications of automation allow people more free time to be allocated to other tasks. For companies, it also means a reduction in human labour costs and expenses. The presence of robots has provided many new job opportunities. However, it has also been responsible for eliminating others [2]. With the continuous ongoing development of technology, automation will only continue to improve and grow.

Robots are a popular form of automation, and just like general automation products, they can be used in many different circumstances, such as in factories, home environments, military, agriculture and even in medicine [3]. They can be applied for a variety of functions, such as performing repetitive tasks like packaging or moving boxes, vacuuming, checking for bombs, and planting and watering seeds, to name a few. They can also be used in social situations, like caring for the elderly (with studies on the topic mostly from Japan as opposed to other countries, and have more focus on companion types of robots instead of physical health assistance robots) or keeping children entertained and educated [4, 5].

Popular examples of robots used in education include the Lego Mindstorms, and a well-known companion robot is the AIBO by Sony [6].

There are a multitude of existing robot designs due to the variety of their applications. They can be stationary, such as robotic manipulators seen in factories, or mobile ones like Amazon warehouse robots. They can even be a mix of both, where the manipulator appendage is attached to the mobile robot base.

Fig 1.1. Amazon mobile robots [7]

Despite the differences in robots regarding their appearances and performable tasks, they share a few similar functionalities in regard to how they work in terms of locomotion, cognition, perception and navigation. For their locomotion, all robots have a form of kinematic and dynamic model, as well as a variety of control theories that are used to dictate how they move. Their perception, the means by which they view their surroundings, is dictated by external sensors. Examples are the light detection and ranging sensor (Lidar) which uses a laser to detect obstacles, the ultrasonic sensor which uses sound waves, the infrared sensor which emits radiation to detect objects, and cameras for sensing the external environment, including sonars (sound-based sensor) that can be used in underwater situations. There are also internal sensors that are used to measure the state of the robot,

such as encoders which count the rotation of the robot's joints and can be used to calculate position. Velocity sensors can be used to determine the velocity of the robot, and by extension the acceleration. Voltage and current sensors can be used to determine how much power the robot is using. Cognition is the robot's ability to read and analyze all sensor data to determine its next steps, and navigation is the robot's ability to determine how it should move in a location while using path-planning strategies to avoid obstacles [8].

Even with their many capabilities, robots still have their limitations. They are only as intelligent as they are programmed to be and depending on their energy source, there is a limit to how long they can run. Whereas robots that are powered via an outlet can run for as long as the outlet can provide electricity (and assuming there is no mechanical failure), the same cannot be said for mobile robots, especially ones that rely on a portable energy source such as batteries or fuel cells [9]. It is possible to use other forms of energy to power a mobile robot, but even those are limited in terms of run time and heavily dependent on the size of the robot. Gas for example, which can be used in larger mobile robots, needs eventual refuelling. However, it is a non-renewable energy source and is not eco-friendly. Renewable energy sources are also an option, such as solar or wind, but cases where the robot goes indoors or does not have access to renewable energy, it would still require a form of portable energy storage. This makes batteries a much more popular energy source for mobile robots since it is easy to implement and acquire. Rechargeable batteries in particular are commercially more sought out, and can be used for longer periods. They can also be recycled, making them more eco-friendly compared to fossil fuels.

There is however a considerable limitation to batteries, that being the length of their life cycle. For each time the battery is charged then discharged, or simply sitting at rest, its

available capacity reduces. This is called battery degradation. This degradation results in the battery running for shorter amounts of time because the amount of energy it can store has decreased. Battery degradation occurs the most when the battery goes through a charge cycle, hence why it is important to be able to efficiently use as much of a single charge as possible. By decreasing the number of times needed to charge the battery, effects such as capacity degradation can be reduced and the battery life can be prolonged. To improve the efficiency of battery usage in a system, there are a variety of methods that focus on modelling the battery to best prolong the lifespan. One can work towards prolonging the lifespan by assessing the battery life [10] or by using a battery management system (BMS) to better control the amount of power sent to each component of the robot. Proper storage can also help. For a mobile robot, optimizing the software is another method to prolong battery life. Swanborn *et al.* [11] provides a list of where power and energy are most heavily consumed, as well as a list of possible solutions that might make the system more energy efficient. These include a decreased number of motion changes, a slower motion overall, a decrease of idle time, and using a more advanced motion algorithm.

Navigation is not only a major contributor to the motion of the algorithm, but it is also highly important to the autonomy of the algorithm. It is the navigation algorithms that motion is derived from. From navigation, the mobile robot receives a planned path that it uses to avoid obstacles while the robot heads for a goal. On a known map, the mobile robot only sees the known obstacle locations. However, in a real environment, there may always be changes in obstacle placement, such as having new or hidden obstacles that the map did not indicate or were placed after the map's creation. Additionally, there may also be the need to account for obstacles that are moving. This makes dynamic path-planning a highly

important topic of study in relevant research. A dynamic path-planning algorithm allows for more spontaneity in the navigation of the robot, in case it encounters moving or unexpected obstacles [12].

There are a variety of different navigation strategies when it comes to path-planning, which can be split into multiple different categories. These categories can be listed as grid-based strategies, which include A* or D*, artificial potential field, geometric algorithms, and sampling-based algorithms such as rapidly-exploring random tree (RRT) [12]. Further research also includes variations of these algorithms. Research in path-planning is constantly evolving, as there is always a possibility of discovering newer methods to improve upon existing ones.

The motivations include generating a power reduction model through power prediction using motion data, reducing computational cost for real-time calculations, and using a combined control framework to implement on a physical robot. The objectives are to create a methodology that would allow for a more energy-efficient navigation control system that is robust in obstacle avoidance for a robot traversing a known semi-known space.

1.2 Objectives

The main objectives of this research are to design and develop a control strategy on a mobile robot so that the power consumption of the battery can be conserved by optimizing the motion in a dynamic map environment. This control strategy outputs robot control commands on a remotely controlled, battery-powered physical robot as it moves

simultaneously using real-time data. The power consumption prediction model would be trained offline based on real robot motion data. Since mobile robots, in general, have conceptual similarities in navigational algorithms, with the main difference being the models used in their control strategies, this study, in particular, would look at a small differential drive 3-wheeled robot that utilizes the robot operating system (ROS).

The detailed objectives of this research include:

1. To model a data-driven power consumption prediction for a LiPo battery-based on the motion data of a differential-drive, 3-wheeled mobile robot.

2. To develop a path-planning method that reduces the computational cost of when the robot is moving in real-time while avoiding obstacles along the way.

3. To design a non-linear model predictive control (NMPC) based strategy that optimizes the power consumption, local goal movement and obstacle avoidance

4. To develop a cost function for the above objectives and to tune the weighting factors and constraints to obtain optimal performance.

5. To make a comparative study on the effectiveness of using the power consumption prediction versus when the prediction is not used for power consumption conservation.

1.3 Contributions

To the knowledge of the researcher, the main contribution of this research is:

1. An adaptive methodology for a data-driven power consumption model based on the motion of the robot.

2. The implementation of a reduction in power consumed through the motion control of the robot, using power consumption prediction on a real robot.

3. A path-planning algorithm that uses the RRT* and artificial potential field path-planning methodologies for obstacle avoidance and robot navigation in a dynamic environment.

4. The combined framework under an NMPC control strategy for robot navigation through a variety of obstacle configurations

1.4 Outline

The following shows how the thesis is organized:

- **Chapter 1 Introduction** introduces the current technologies on which this study is based upon as well as the difficulties that have arisen in this field. The motivation, objective, and contributions are also showcased.

- **Chapter 2 Literature Review** introduces mobile robot configurations and reviews existing navigational and power consumption methods where the research gaps are identified and illustrated.

- **Chapter 3 Methodology** introduces the differential drive model that would be used as the basis for the robot's model used in the NMPC. Then explains the problem of robot navigation with power consumption to be solved which is then formed under an NMPC framework with a TurtleBot3 as a platform where the objectives are local goal navigation, obstacle avoidance and power consumption prediction based on real robot motion data.

- **Chapter 4 Results and Discussion** presents the experimentation details and results that the real robot presented based on the NMPC power consumption strategy with potential field obstacle avoidance. The analysis of the power consumption results is also provided.

- **Chapter 5 Conclusion and Future Works** looks at an overview of what was discussed in this work and lists possible future improvements.

Chapter 2. Literature Review

The review will first look at the different kinds of mobile robots, then at various methods of path planning and how they've been applied to autonomous wheeled robots or vehicles since, despite a difference in the platform, there are similarities to the methods in terms of navigation. It will also look at existing literature in the realm of energy conservation and the methodologies used there, as well as the control strategies used to implement motion in the robot. Finally, the research gaps in existing literature will be identified and analyzed.

2.1 Types of Robots

Mobile robots are capable of traversing a multitude of environments, with several types of functions applicable to them. They are a very popular tool and can be used in both industrial and domestic settings. Some wide known applications of mobile robots involve vacuuming, transportation of objects, and military usages (such as sweeping for mines). These robots can not only be remotely controlled but can also move autonomously. They can have wheels, tracks, or legs, and may be used to traverse underwater and aerial environments as well. A great deal of research has thus been focused on optimizing their movement and construction. For these robots, autonomy is a huge aspect, and the autonomy of mobile robots is a thoroughly investigated topic. Since the robot's ability to traverse a multitude of environments is one of its most important features, navigational methods for the robot have been heavily explored. When the robot traverses an environment, there are many factors to consider how it moves, what kind of environment it is in, and what obstacles are in the path. No matter the type of mobile robot, whether it be on land, in water, or air, they

all have similarities in terms of planning their navigation. Hence, people have developed various path-planning methods to improve robot movement. All path-planning algorithms are under certain categorizations, based on their main method of planning. Some of the known ones include grid-based search, artificial potential field, geometric algorithms and sampling-based algorithms. All these types of path-planning methods, with variation to the modelling behind the robot, can be adjusted to work for the robot's navigation.

However, the movement of the robot is not the only relevant factor. If the user has a task for the robot to complete at a far distance, the robot must be able to move in a way for it to conserve as much of its energy and in as little time as possible to ensure that it can reach the destination. The kind of energy the robot uses may limit the range of the robot. Some robots may be able to use renewable energy such as solar panels or, in the case of the Mars rovers, nuclear energy. However, there are a good number of robots that utilize some form of rechargeable battery. These batteries tend to have a greater limitation compared to the other renewable energy sources since, after long and continuous usage, the battery will degrade. Thus, it is necessary to decrease the number of times the batteries need to recharge, which elongates the battery's life cycle.

Robots of each type have their own variety of configurations. Wheeled robots can range from a minimum of 2 wheels to a maximum of however many the producer can afford. 1-wheeled (unicycle) robots do exist, as shown in [13] However, they are not the most stable of robot types. Nonetheless, some of its theoretical aspects, such as the kinematics and the variations of the kinematics, can be applied to the robots with a higher wheel-count. Many robots have their kinematics be based on the unicycle robot as shown by [14]. This is

because this is the traditional robot kinematic and is easier to understand. From the unicycle model, further kinematics for each configuration can be derived.

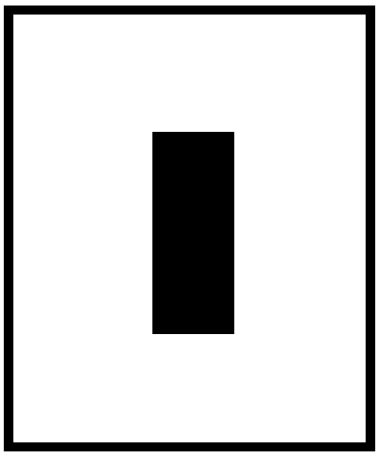


Fig 2.1. Single wheeled robot

2-wheeled robots are some of the simpler robots. Each wheel is powered by a motor individually, and the drive method is called differential drive. This means that when the robot turns, depending on the rate of the turn, one of the wheels would either move slower or begin moving in the opposite direction. Despite the simplicity of 2-wheeled robots, they are not the most stable, as it is very easy for the robot to tip either forward or backward. One of the most well-known examples of a 2-wheeled robot is the Roomba [15]. 3-wheeled robots can be driven using a differential drive or with steering. With differential drive, the two larger wheels are powered by individual motors while the third wheel is present for stability and is free-turning. If the third wheel is used for steering, then the two connected wheels on the same axle are powered by one motor while the third wheel is powered by a separate motor that controls the heading of the wheel. 4-wheeled robots can have a variety of steering methods. The first method is similar to differential steering with the exception

that there are two freewheels. Two motors are attached to what are usually the two front wheels, and the motors power the wheels individually while the third and fourth wheels in the back are mainly for balance. The second method is a tank-like motion where the four wheels are all powered individually but still connected along the track. The two wheels on the left are a pair, as are the two on the right. The pairs are connected so that they turn in the same direction. The third method is to steer the robot like a car. This means that two of the wheels, either the front two or the back two, would be connected on a single axle while the other two wheels are powered individually [8]. The specific kinematic model for the robot that is focused in this paper is the differential drive. Which is a model that can be applied on 2-wheeled, 3-wheeled and 4-wheeled robots [16]. This model would be looked at in detail in chapter 3.

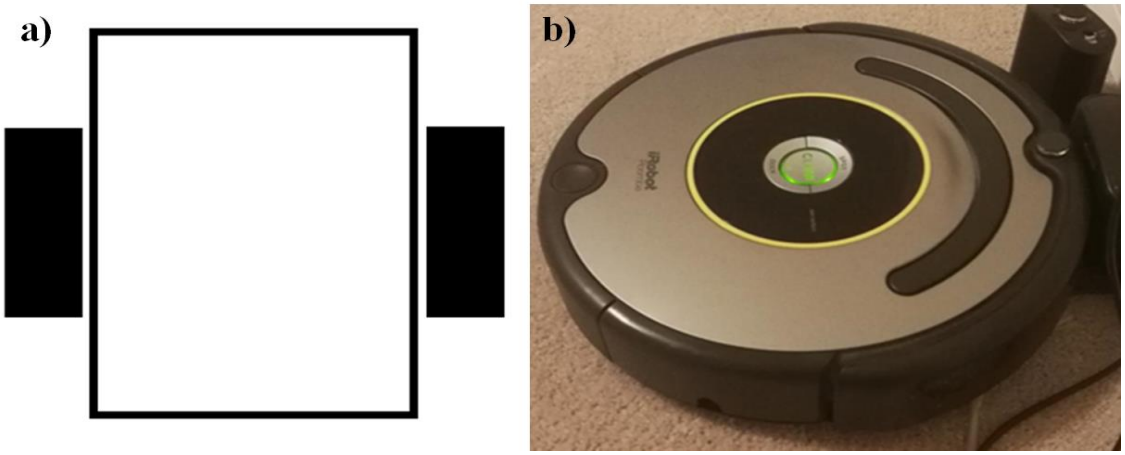


Fig 2.2. (a) 2-Wheeled robot configuration (b) Roomba

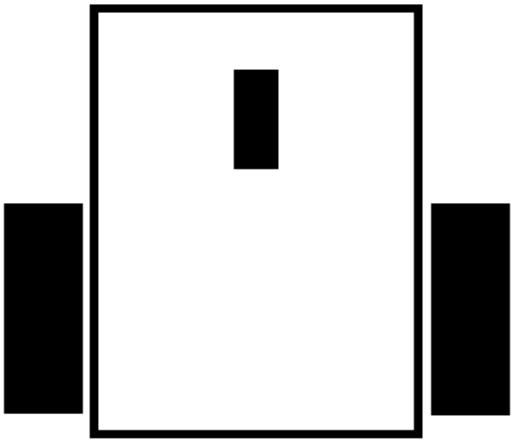


Fig 2.3. 3-Wheeled robot configuration

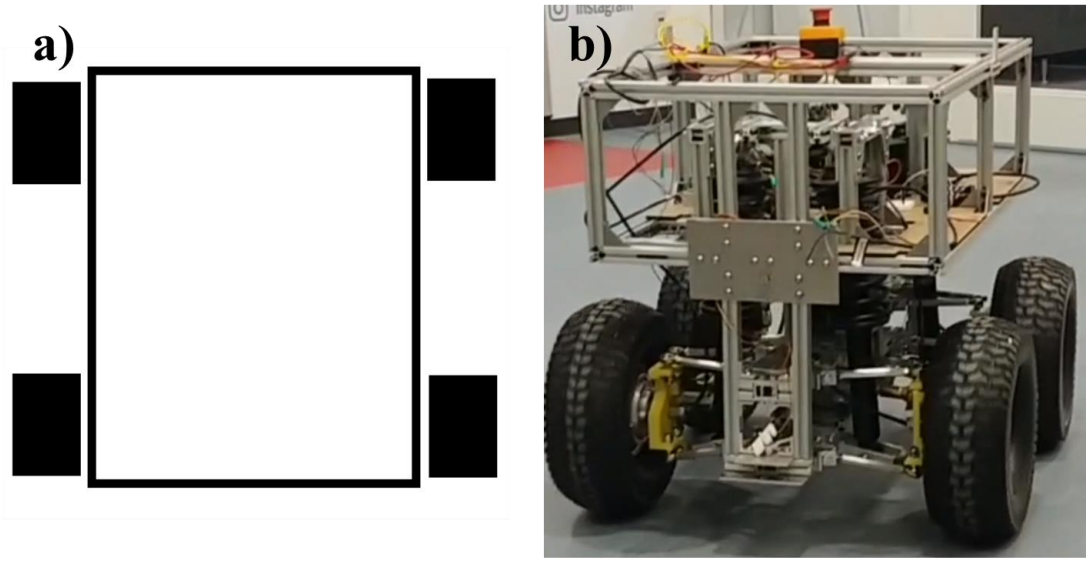


Fig 2.4. (a) 4-Wheel robot configurations (b) 4-Wheeled delivery robot

Robots with more than 4 wheels are less commonly seen. However, they have a great presence in space exploration. The Mars rovers by NASA are 6 wheeled robots that have all of their wheels touching the ground. This is done by the special chassis and suspension system that the robot has, where the arms of the system are much longer than what would

be seen on more commonly known mobile robots or autonomous vehicles [13]. Robots such as the Mars rover utilize nuclear fusion and solar energy as a power source. However, on earth, using nuclear fusion in a robot is not a viable solution due to the larger number of risks nuclear energy presents. Thus, the required calculations and methodologies applied on a Mars rover may not fully correlate to a robot on Earth.

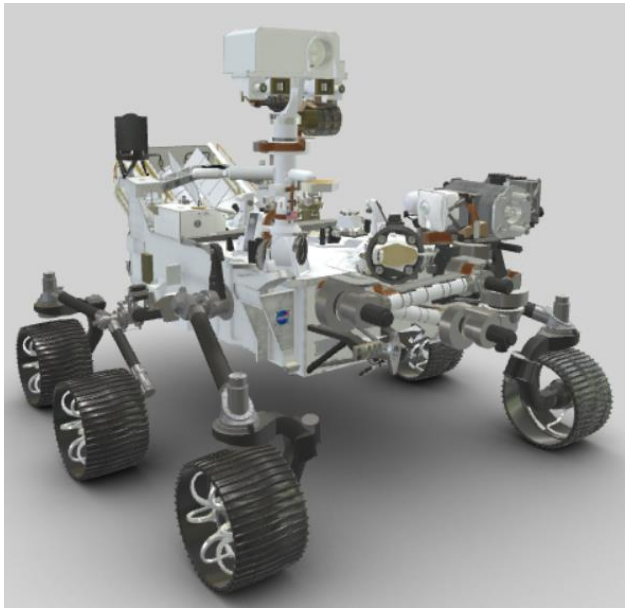


Fig 2.5. Mars rover Perseverance [17]

For wheeled robots, with every increase in the number of wheels, not only will the energy requirement rise, but so does the complexity of the dynamics. For example, a 4-wheeled robot may have more flexible maneuverability, but would consequently require more energy than a 2-wheeled robot due to the larger load and number of actuators required to move it. This increase in energy requirement makes it even more important to optimize the control of the motion in a simplified manner. Due to the adaptability of the method, developing a power consumption model based upon motion data is useful and can be

applied to various robots despite them not having identical configurations. Furthermore, the concept behind various path-planning algorithms applies to all types of robots since the method is not robot specific, however, when being applied to a physical robot, the size and shape have to be considered as even if a control strategy may stay the same, the tolerance for avoidance may change.

Unmanned aerial vehicles (UAVs), or more popularly known as drones, utilize propellers to fly. Depending on the size and type, the range of motion and method of control may change. The main configurations of UAVs are called fixed-wing, helicopter, and multi-copter. These systems tend to be 6-DOF, thus increasing the complexity of the kinematics. They can be powered either through batteries or gas for the larger UAVs and have been used in as many different applications as land mobile robots. UAVs are highly useful for military purposes such as surveillance, for transportation purposes such as Amazon delivery, or for agricultural purposes, to name a few [18]. The methodology of using a power prediction model using motion can also be used on unmanned aerial vehicles (UAV) as shown in [19, 20]. This shows that a data-driven power consumption prediction model can be used on all kinds of mobile robots as long as the correct data inputs are selected further emphasizing the versatility of the method.

Fig 2.6. Quadcopter UAV

2.2 Hardware Components of Robot

For this research, the robot used is the TurtleBot3, a 3-wheeled differential drive robot. This means that two of the wheels are powered individually with the third wheel, which tends to be smaller in size, free rotating. The mobile robot is comprised of a combination of different components in order to operate as shown in Fig. 2.7. These components include sensors, computers, and actuators, which together are capable of accomplishing a task while moving in an environment. This section takes a look at some of the detailed components of the robot.

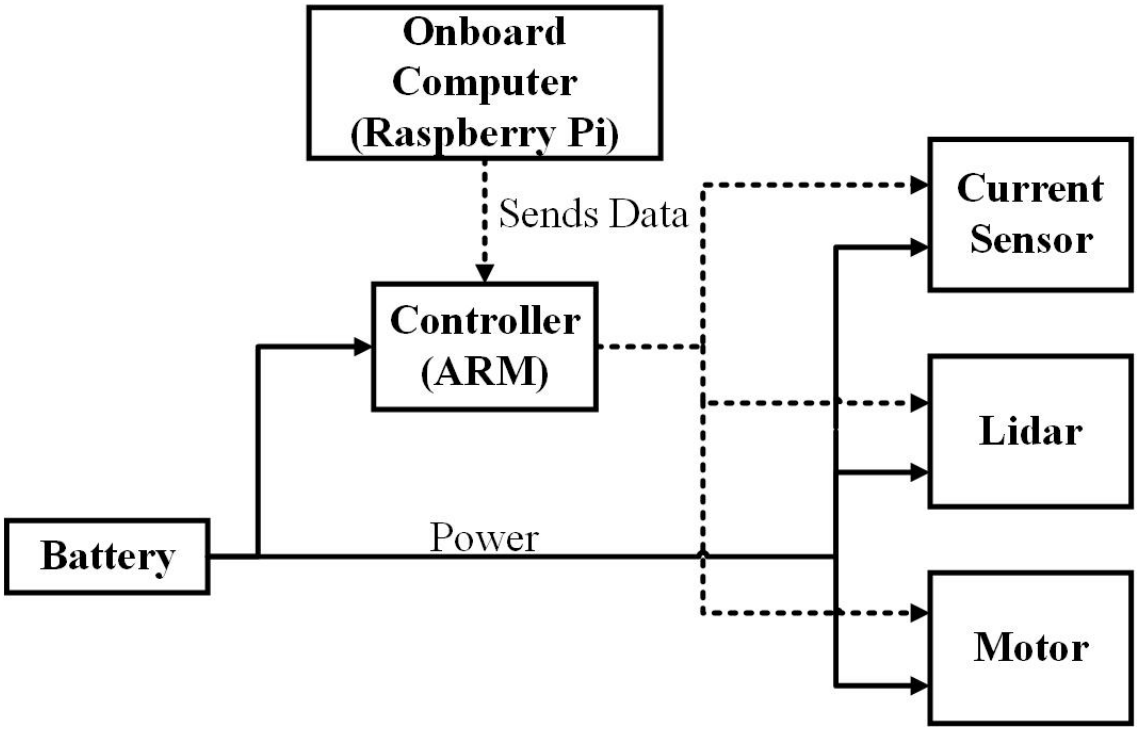


Fig 2.7. Robot's component configuration

The onboard computer of the robot is the raspberry pi. Its main function is to allow communication between the robot's controller and the host computer which remotely controls the robot. This communication is set up through ROS. To get this communication started, the ubuntu 16.4 operating system is installed onto the raspberry pi, where the Wi-Fi communication would be set up.

The controller used on the robot is the OpenCR 1.0, an open-sourced Arduino UNO-based controller module for ROS. It is from here that power from the battery is transferred to all other components of the robot, and is where the data from these components are stored to be sent to the computer. It also does calculations that determine the pulse-width modulation (PWM) to be sent to the motors. It contains some sensors within itself, such as an inertial

measurement unit containing a 3-axis gyroscope, an accelerometer and a magnetometer. It has numerous pins and ports for the hardware communication between the components and is programmed under the Arduino software. Fig. 2.8 shows an image of the Raspberry Pi and the OpenCR controller.

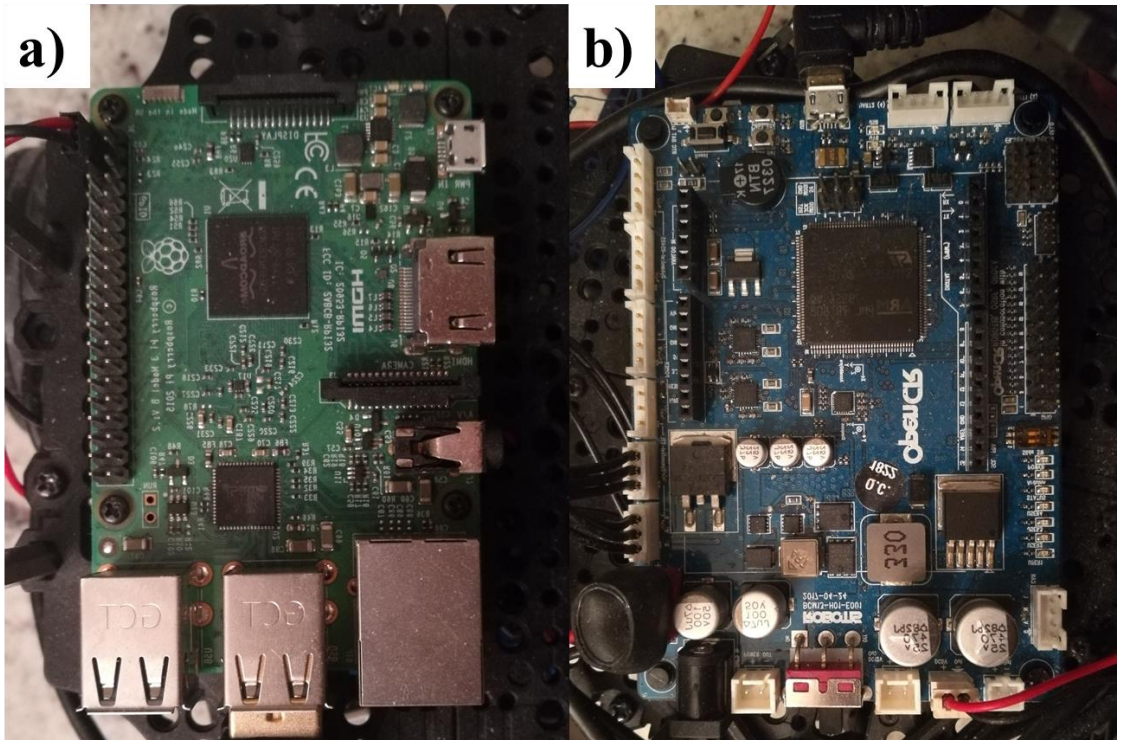


Fig 2.8. a) Raspberry Pi 3B b) OpenCR 1.0

The actuators on this robot are by the Dynamixel XL430-W250. This is a cored motor with a contactless absolute encoder as the joint's position sensor. There are various methods to control this motor, position, PWM, and velocity. For this robot, the control is done through velocity. This means that the user sends a velocity to the motor, which is then set as the goal velocity. This velocity gets converted into the desired velocity trajectory using an acceleration profile, which then goes through a PI controller that outputs a PWM, which

subsequently goes through a limiter that sets the final PWM value for the motor after going through an inverter [21]

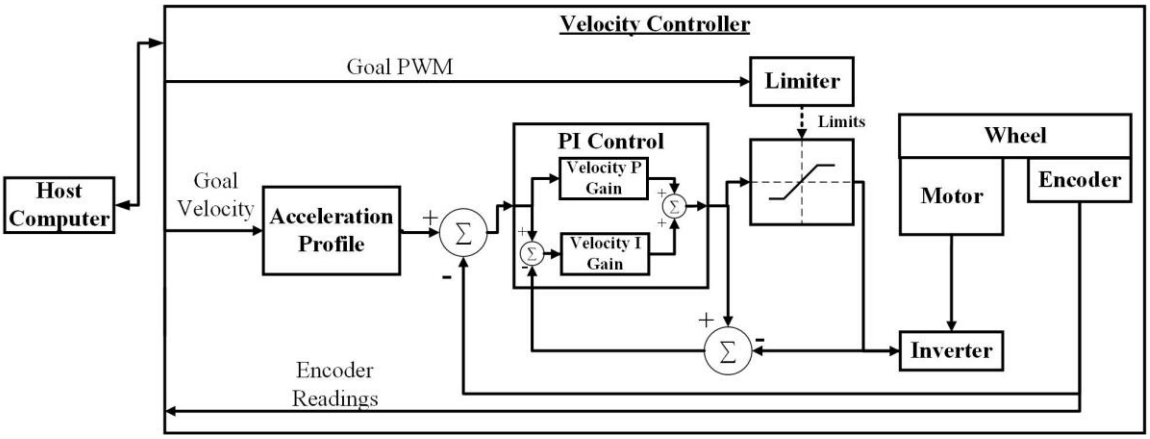


Fig 2.9. Velocity controller

The battery of the robot powers the components of the entire system. For the TurtleBot3, the battery used is a 3 cell Lithium polymer battery (LiPo) with a nominal voltage of 11.1 and a capacity of 1800 mAh. Lithium polymer batteries (or more specifically lithium-ion polymer batteries) differ from regular lithium-ion batteries due to the material used for the electrolyte. Usually, the electrolyte is in the form of liquid, while LiPo batteries use gelled polymer as the electrolyte [22]. The benefits of LiPo batteries are that they are safer, more flexible, and lighter in comparison to their liquid electrolyte counterparts, but still just as volatile if handled and cared for poorly [23]. However, the gelled polymer is less conductive, which means that the battery would have less capacity by comparison [24]. This makes good and proper care of the battery even more important since the battery is subject to degradation. Degradation of the battery cells can be both physical and chemical, with some of these degradation mechanisms being caused by various factors such as time

(batteries at rest will always have some degradation), external temperature (high or low temperatures are detrimental to the battery), high current load (both input and output can cause mechanical stress and high internal temperatures), high or low voltage cells (storage or use at either of these conditions) and too much change in energy cycles (excessive charge and discharge cycles) [25]. To minimize the degradation damage done to the battery, it is advantageous to limit the causes of the degradation. Preventing the robot's operation and storage in high heat or low temperatures is one consideration, as well as changing the method for the control of motion so that a lower current load is used and the number of charge cycles is lowered. The control of motion is the method that is focused on for this study.

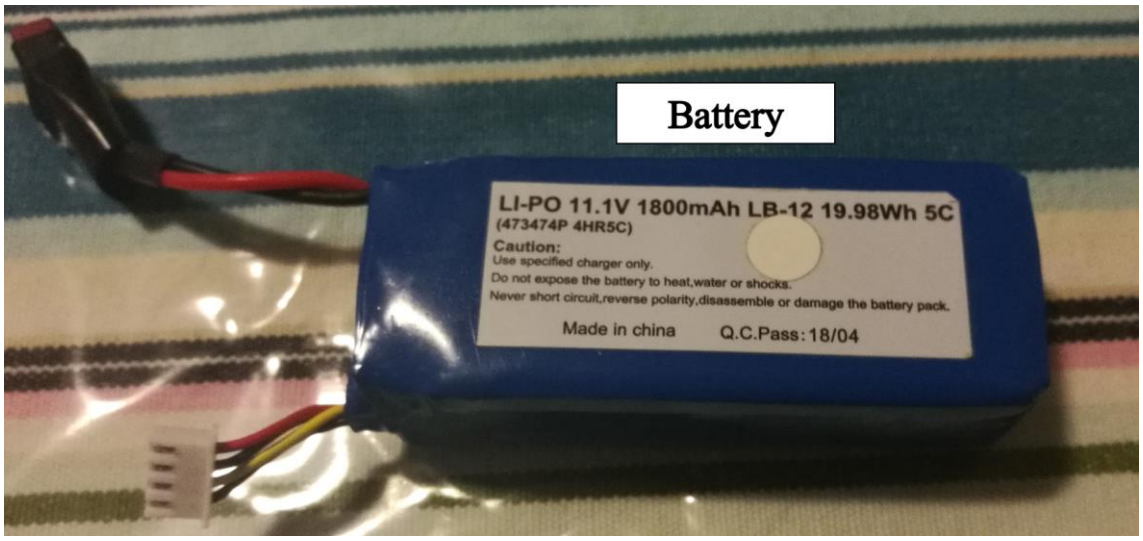


Fig 2.10. Battery used in TurtleBot3 with specifications listed

Light distance and range (Lidar) sensors are sensors that aim rapid pulses of laser light at the surroundings. If there is an object, the light is reflected back to the sensor and the time it takes for this reflection to return is used to calculate the distance and angle the object is

from the lidar. This creates a cloud of points that represents a model of the object [26, 27]. The lidar used in this research, is a 360 degree, 2D lidar. This means that the lidar can see the whole circumference around the lidar but cannot see the height of the object.

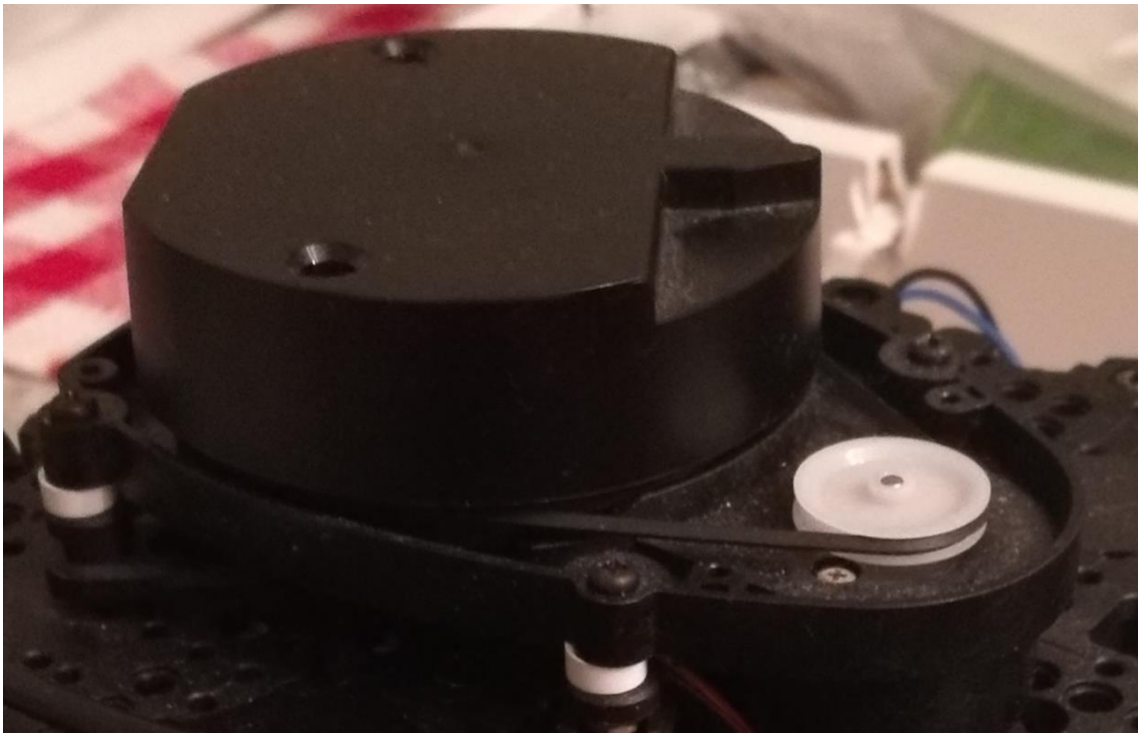


Fig 2.11. Light distance and range sensor on TurtleBot3

The lidar is fairly accurate with an accuracy of about 15 mm and a distance range from 120mm ~ 3,500mm [21]. An example of the point cloud as shown on Matlab can be seen in the figure below.

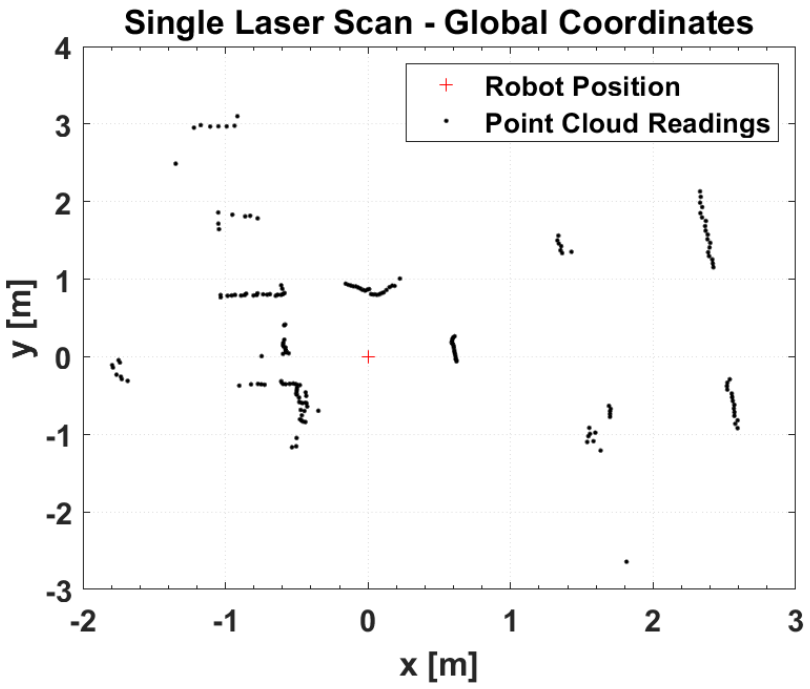


Fig 2.12. Point cloud derived from laser scan in the global coordinate system

The current sensor used is the ACS712 hall effect sensor. Hall effect sensors utilize magnets to sense the current. As the current goes through the sensing plate, the magnetic field causes a voltage based on this current to be sent out. This voltage is what is read by the OpenCR code and gets converted into current. The conversion equation is as follows:

$$V_{raw} = \frac{VCC}{1023} \times V_{in} \tag{2.1}$$

$$V = V_{raw} - 0.5VCC \tag{2.2}$$

$$I = \frac{V}{Sensitivity} \tag{2.3}$$

where VCC is the supply voltage of a value of 5, V_{raw} is the raw voltage data from the reading of V_{in}, and I represents the current and sensitivity is 0.1 from the datasheet [28].

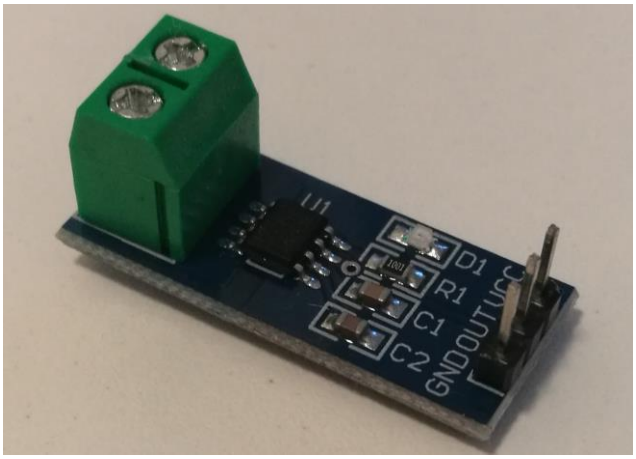


Fig 2.13. ACS712 current sensor

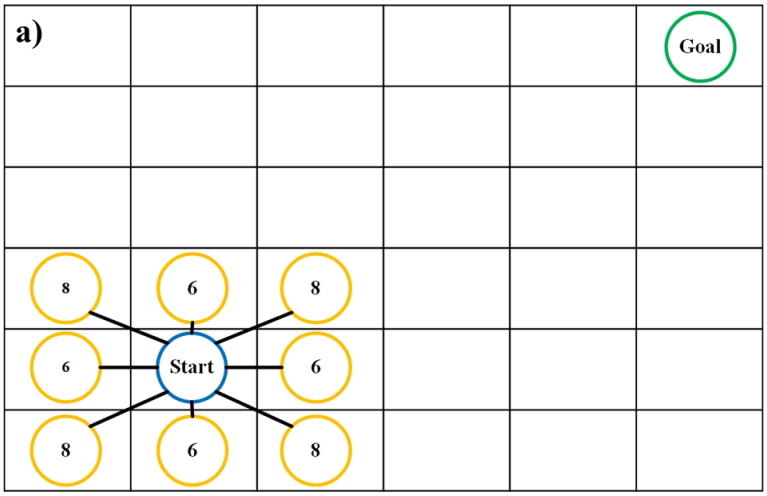
2.3 Path-Planning

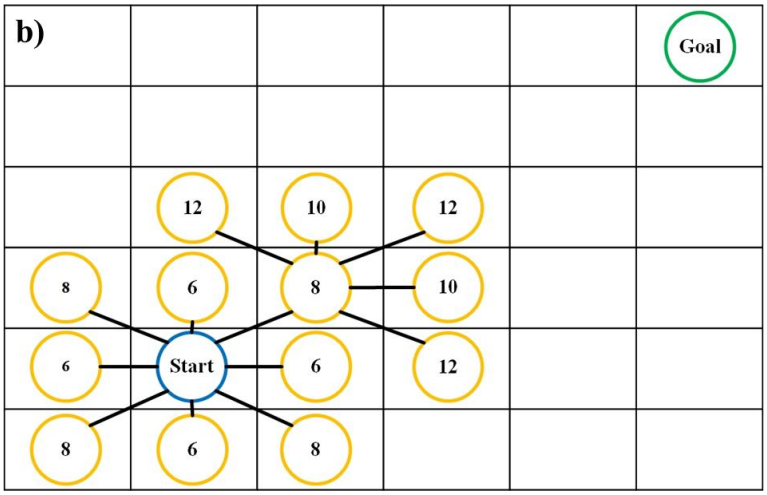
Path-planning is a heavily researched topic because it is an important feature for any sort of autonomous robot or vehicle. There are a variety of different path-planning algorithms such as A*, potential field, RRT*, genetic algorithm-based and their variations being some of the more well-known, and each of these falls under overarching categories. The categories and algorithms will be explained below:

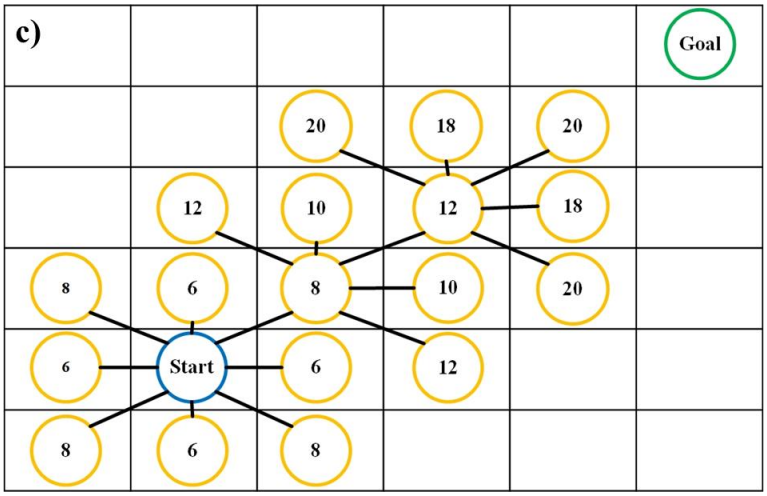
2.3.1 Grid-Based

Grid-based algorithms utilize cells to determine the best path. From the starting cell, the algorithm looks to adjacent cells to determine whether the next cell is feasible (if there is an obstacle) and if it's getting closer to the goal. The algorithm will keep looking at the next cell until it reaches the goal, where the algorithm proceeds to find and determine a cost for each of the cells, along with the connected cells, that results in the lowest path [29].

A* and D* algorithms are some of the more widely known methods of grid-based path planning. Saranya *et al.* [30] look to a comparison between A*, D* and a modified D* algorithm on a Lego NXT Mindstorms robot. It was determined that A* works better and faster in a static environment and is simple to implement, but their modified D* works faster in a dynamic environment compared to D* and is less complex than D*. However, A* is still able to work with complex environments as evidenced by [31], where a least-squares policy iteration is combined with A* to generate a hierarchical path-planning optimization where A* is used to create a more general path while the LSPI is used to generate a more optimal local path. A down-side to grid-based algorithms, however, is their computational complexity in regards to the memory space of the program [32]. A larger map with more grids would result in longer computational time and may not always have the smoothest paths. At times, the resulting paths can have very sharp turns, which are not efficient for the robot as not all robots are capable of such maneuvers, and even the edge of the robot may catch an obstacle due to the robot's centre of rotation.


a)
Goal
8
6
8
6
Start
6
8
6
8


b)
Goal
12
10
12
8
6
8
10
6
Start
6
12
8
6
8


c)
Goal
20
18
20
12
10
12
18
8
6
8
10
20
6
Start
6
12
8
6
8

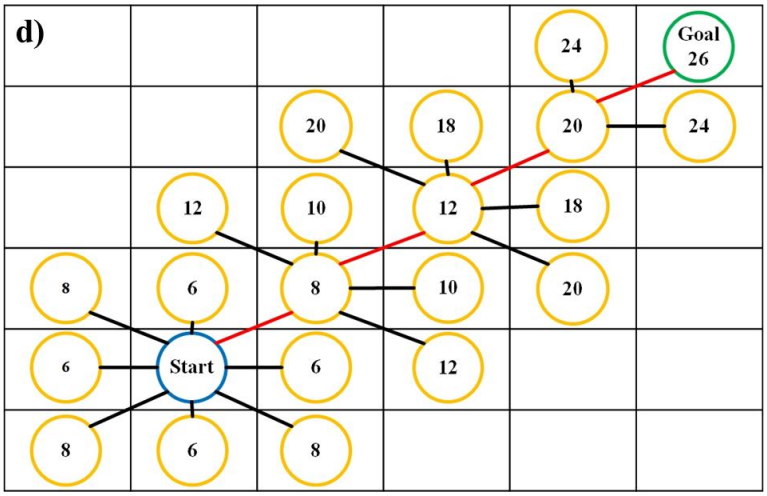


Fig 2.14. A* at various time steps a) Timestep 1, b) Timestep 2, c) Timestep 3, d) Timestep 4

2.3.2 Potential Field

The artificial potential field algorithm utilizes the concept of attractive and repulsive forces. Obstacles generate a repulsive force that pushes the robot away, while the goal generates an attractive force that pulls the robot towards it. These forces are all assigned virtually and a gradient is applied to determine the optimal path based on the steepness of the gradient [12]. The application of potential fields with nonlinear model predictive control-based algorithms has been done on unmanned surface vehicles (autonomous ships) to traverse a dynamic coastal environment in [33]. Potential field algorithms can also be used in autonomous vehicles as shown by [34], where a model predictive path-planner controller based on the potential field was used. This showcases the effectiveness of potential field algorithms in a dynamic environment, as both on coastal shorelines and a driven road, there are a lot of unexpected obstacles and the potential field is capable of adapting to these conditions. However, the flaw to the potential field is that it may trap itself in a local

minimum in some scenarios [35]. This tends to occur when the repulsive and attractive forces equal each other, which usually causes the robot to stop moving.

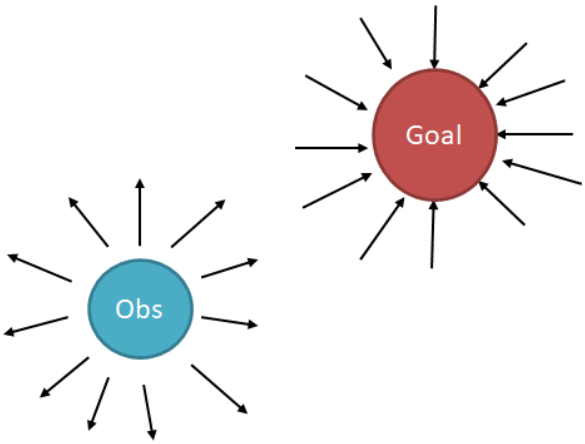
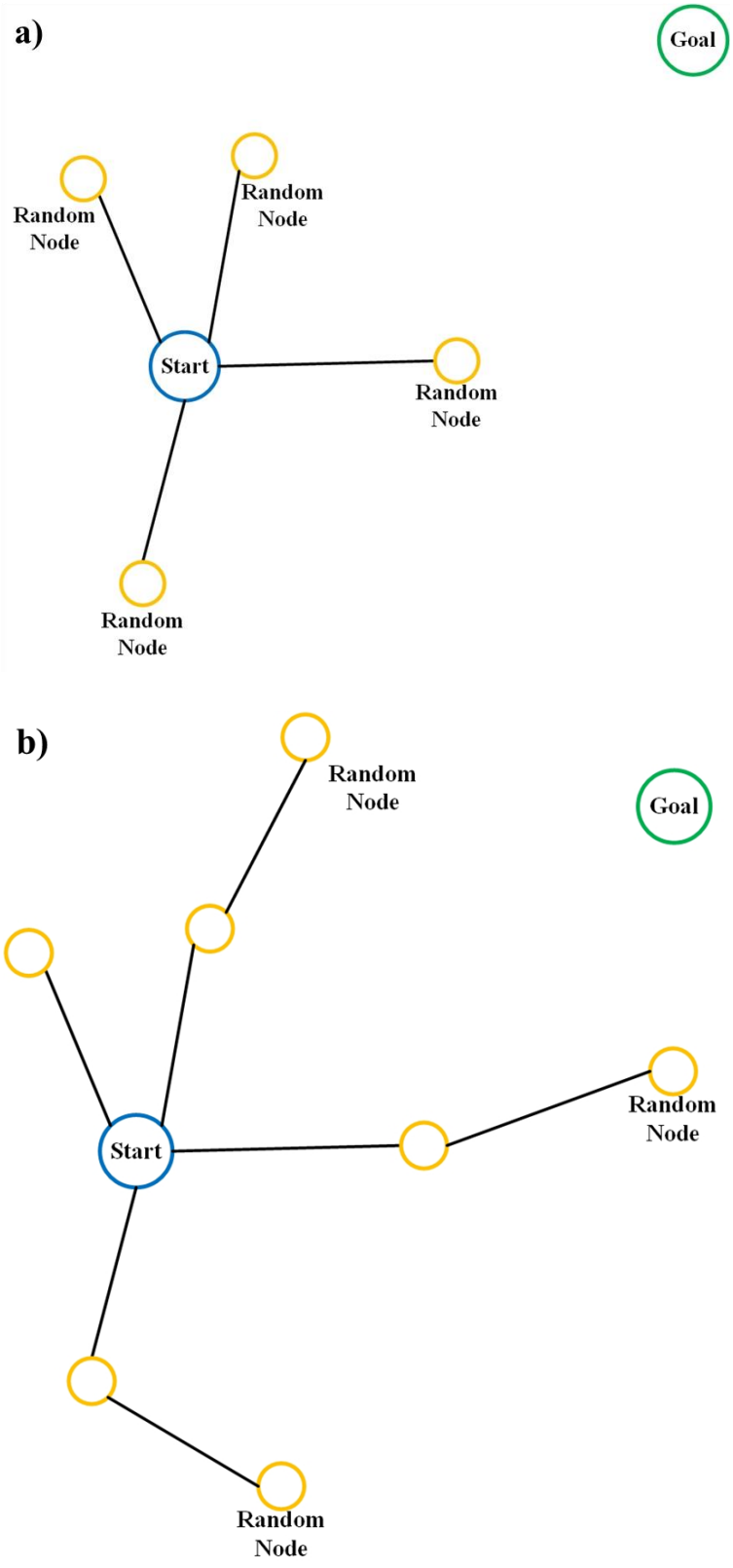


Fig 2.15. Potential field forces

2.3.3 Sampling-Based

Sampling-based algorithms like the randomly-exploring rapid tree (RRT), its variant, RRT* and the probabilistic roadmap (PRM) generates random nodes and look for the connections between the nodes to create the shortest path that goes from the start to the goal [36]. PRM takes in the configuration space and notes the free space on the map. It then places nodes all over the map at random and connects all nodes to create a multitude of possible trajectories, which allows it to then figure out where the from start to goal path is after it has found all possible trajectory connections. RRT and its variant RRT* differs from PRM as RRT generates a selection of nodes of specified distances, and from these initial nodes, it expands randomly (like tree-branches) until it reaches the goal [37]. However, the path results from RRT are not always the smoothest or most ideal, as it can generate paths that, though feasible, are not optimal or considered the “shortest and

smoothest” path. The variant RRT* furthers the ability of RRT by generating nodes until the algorithm deems the path can no longer be optimized or it reaches the maximum number of search iterations. La Devin Connell [38] places the RRT* star in a dynamic environment and showcases its robustness at planning and replanning an optimal path. However, it makes no mention of the length it takes for the algorithm to plan and replan the path. Xinyu *et al.* [39] utilize a combined method of potential field and RRT* called potential guided RRT*. The method is shown to be effective in generating an optimal path in a short amount of computation time. However, the resulting paths are very close to the obstacles and if applied to real environments, depending on the size of the robot, may result in a collision. It also does not account for a dynamic environment, which is where real-world applications are usually seen in. RRT* requires a longer computational time when finding the optimal path, which decreases the efficiency and is also less beneficial when running the program in a real environment, where the robot is expected to constantly be in motion. For the robot to stop moving to recalculate a new path if there are unexpected obstacles wastes not only time but also battery power.


a)
Goal
Random Node
Random Node
Start
Random Node
Random Node
b)
Random Node
Goal
Start
Random Node
Random Node

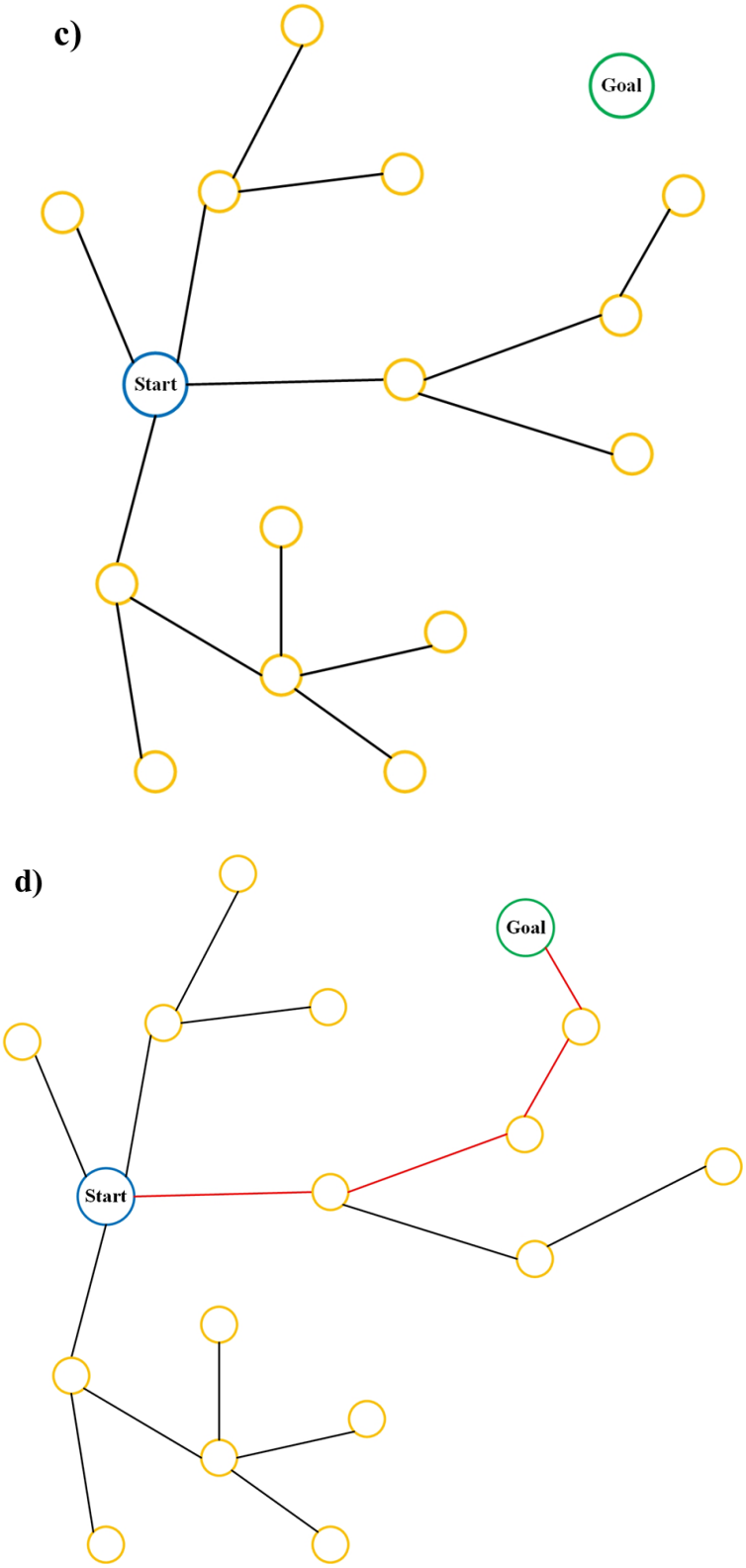


Fig 2.16. RRT at various time steps a) Timestep 1, b) Timestep 2, c) Timestep 4, d) Timestep 5

2.3.4 Heuristic Path-Planning

Heuristic algorithms are popular when it comes to dynamic global path planning, especially the genetic algorithm (GA). These approaches focus on using optimization techniques such as ant colony optimization [40, 41], particle swarm optimization [42, 43] and simulated annealing [44, 45], to name a few, and are generally more efficient than the other approaches mentioned above. However, they may not always solve the path-planning problem or be the most optimal solution [46]. A good representation of how GA is used for path planning is shown by [47]. However, when utilized in combination with other path-planning approaches, it can be more effective as explained in [48].

2.4 Power Analysis in Mobile Robots

When looking at the power consumption minimization of battery-powered mobile robots, there are a few methodologies applied. These methodologies typically require the user to know and understand more information about the system that is being used, which can be implemented into a complex model that may be difficult to compute or implement in a wider variety of systems. Abdilla *et al.* [49] use a power model to determine an endurance model for a UAV, to establish how long the UAV can run. The power model would require information about the rotor's thrust and efficiency, which is affected by the UAV's weight and propeller size. Not all of the information can be easily gathered without extensive experimentation. The paper, though it validates their model, doesn't further indicate what it can do to improve battery usage. Battery management for rescue robots is studied in [50],

where they consider what system on the robot might cause more power consumption and how much the robot may consume overall. Mei *et al.* [51] develop a power model from real data results to determine how to reduce the power consumption of mobile robots. They created their model by determining the coefficients for generic power consumption models for motion, sensor, and the controllers, as well as the embedded computer. Afterwards, they look at techniques on how power consumption can be reduced, but with no implementation. An energy consumption model is developed by [52], which is then implemented into an omnidirectional robot. The energy consumption model requires a mathematical analysis of the kinetic energy, frictional energy, heat energy from motors, and mechanical energy from frictional torque in the motors. The model is then implemented as a cost for the ROS dynamic window approach local trajectory planner. Despite the effectiveness of the model, the experiments were limited to a local planner and a global planner was not implemented, nor does it demonstrate performance in a more complicated environment. The energy consumption model is also complicated due to requiring a significant amount of information in order to implement the model, such as the moment of inertia and frictional torque of the actuator. Cauwer *et al.* [53] present a data-driven approach for energy consumption on electric vehicles. It uses a neural network to create a speed profile prediction and linear regression model to estimate the energy consumption. The resulting prediction was shown to be fairly accurate. However, it does not show the amount of time it would take to run the vehicle, and although it may work well for a global setting, it doesn't cover what would happen locally and in response to unexpected changes to the path. A power model was created by gathering the power consumption of each component for their robot [54]. The techniques they used involved shutting down idle components and

scheduling the time of the tasks. Despite the results demonstrating a detectable decrease in power consumed, this model requires knowledge of the power consumption of each component, which is less efficient to test for compared to taking to account the power consumption of the whole system.

2.5 Research Gaps

As reviewed thus far, there exists a variety of path-planning algorithms, each with their own pros and cons. Most literature in path planning focuses on static environments, which when implemented in a real environment will not perform the same. Environments are always changing, even with an initial mapping of the area, hence having a path-planning algorithm that can adapt to a dynamic environment is extremely important. Use of heuristic path-planning algorithms is fairly popular for dynamic path-planning [47]. However, heuristic algorithm path-planning will not always generate the most optimal result. In addition, many methods focus on recalculating the path, which increases the computational cost of the algorithm. Hence, in this thesis, artificial potential field was combined with an initial path planning algorithm to reduce the computation cost.

There are a few ways to consider the power consumption of mobile robots. One is to develop power models for individual components of the robot, then place them together afterward to be used in the overall system [52, 54]. However, this would require multiple different tests on the system components, as opposed to only the motors, in order to gather all the data needed to develop the mathematical models. Another method of preserving

power is to pre-plan an energy-efficient path for the robot, which can be useful if the robot is in a static environment. Finally, the last method is to make the motion more energy-efficient [54]. This thesis optimizes the motion of the robot in order to reduce the power consumed based on data-driven methodology, which is more versatile compared to model-based ones.

2.6 Summary

In this chapter, various mobile configurations were introduced and some of their functionalities were discussed. Afterwards, the hardware components of the robot platform were explained in detail. Various path planning algorithms' operations were analyzed, as well as how they were implemented. Later, literature reviews of existing power analysis of robots were viewed and their research gaps considered. Finally, research gaps, in general, were reviewed between existing literature reviews.

Chapter 3. Methodology

3.1 Modelling of Differential Drive Robots

Since the experimental tool used is a TurtleBot, a 3-wheeled differential drive robot, it is necessary to determine the kinematics of the TurtleBot for model predictive control in order to determine the control of the robot.

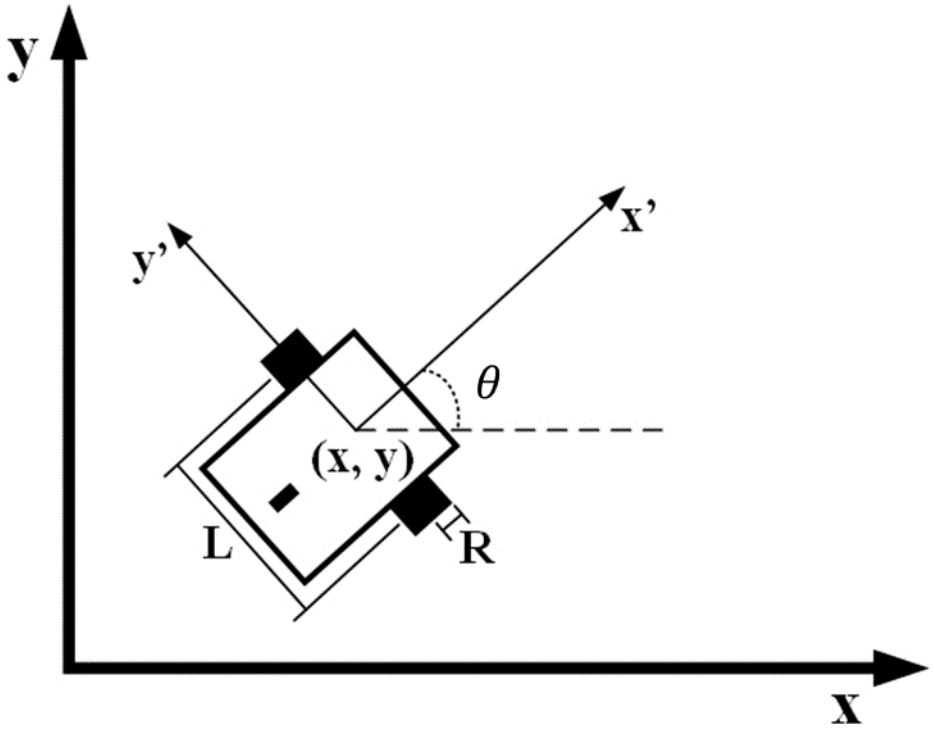


Fig 3.1. Geometry and coordination of the robot

Differential drive robots are wheeled robots with two independently motor-driven wheels. The motion of the robot can be modelled in the form of the following equation which is based on the kinematic model of a unicycle [14].

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{3.1}$$

where x and y represent the global planar coordinates, θ represents the orientation with respect to the x-axis, and v and ω indicate the linear and angular velocities. $\dot{x}$, $\dot{y}$ and $\dot{\theta}$ are the velocities in the x coordinate, y coordinate, and angular velocity respectively. This is the model that will be used with the non-linear model predictive control (NMPC) in order to obtain the control actions of v and ω for the robot. From this, it can also be noted that the differential drive robot follows the nonholonomic constraint:

$$\dot{x}sin\theta \; - \dot{y}cos\theta \; = \; 0 \tag{3.2}$$

3.2 Problem Formulation

As mentioned in the objective, the research is focused on generating an NMPC-based control strategy, using a new combination of a primary and secondary path-planning algorithm with consideration to power-saving through motion optimization. The control inputs to the robot would be the linear and angular velocity and the state variables would be the position (x, y), orientation (θ), and the SoC of the robot. In order to optimize the motion, the cost function must be minimized based off of these three objectives: 1) the local goal, 2) obstacle avoidance and 3) power consumption prediction. Eq. 3.3 represents the generalized function for the NMPC.

$$J_{cost} = \sum_{k=1}^{\infty} f(X_k, U_k) \tag{3.3}$$

where J_{cost} is the cost that is meant to be minimized for each iteration and $f(X_k, U_k)$ is a function that correlates the state and control variables to a cost at one time step for the whole of the prediction horizon. This formulation would be discretized based on the robot's kinematic model. The local goal objective utilizes the reference path to the final goal, generated from the RRT* algorithm. It is determined by choosing a point a few steps ahead on the reference path (from the closest point to the current position of the robot). The power consumption prediction objective is determined using the power prediction model, which is obtained from motion data. The obstacle avoidance objective utilizes an artificial potential field, where the closest obstacle point is set to generate the repulsion force and the local goal is set to generate the attraction force. Since the objectives are the distance to the local goal, the power consumption cost, and the obstacle avoidance cost, Eq. (3.3) can be expanded as shown below.

$$J_{cost} = w_1 * \Sigma J_{localgoal} + w_2 * \Sigma J_{power} + w_3 * \Sigma J_{OA} \quad (3.4)$$

$J_{localgoal}$ is the cost of the local goal, where the distance gap between the local goal and the positions on the prediction horizon is minimized. J_{power} is the power consumption prediction based on a data-driven model shown in Eq. (3.8), and J_{OA} is the obstacle avoidance cost is based on artificial potential fields governed by Eq. (3.9-3.11). w_1, w_2, w_3 represents the weighting factor that controls the trade-off between costs. The values of these weights are 2, 0.001, and 12 respectively. These weights were tuned manually and were decided upon by observing how the robot acts as well as the final cost value for each time step. The tuning is done systematically through activating, deactivating and changing the values individually until the final weight values were obtained. Since the cost of distance gap to local goal is required to get the robot to the final destination, this cost is

always active. Afterwards, the cost of power consumption is activated and the weight is adjusted to get the approximate value so that the robot is capable of reaching the final destination while showcasing power conservation. Then the cost of power consumption is deactivated and cost of obstacle avoidance is activated to obtain the approximate value of the weight for obstacle avoidance. With both these values obtained, the cost of power consumption and obstacle avoidance gets activated so that further adjustments to the weight value can be done which results in the final weight values mentioned above. The tuning of the weights is required due to the necessity of equalizing the importance of the objectives for the NMPC.

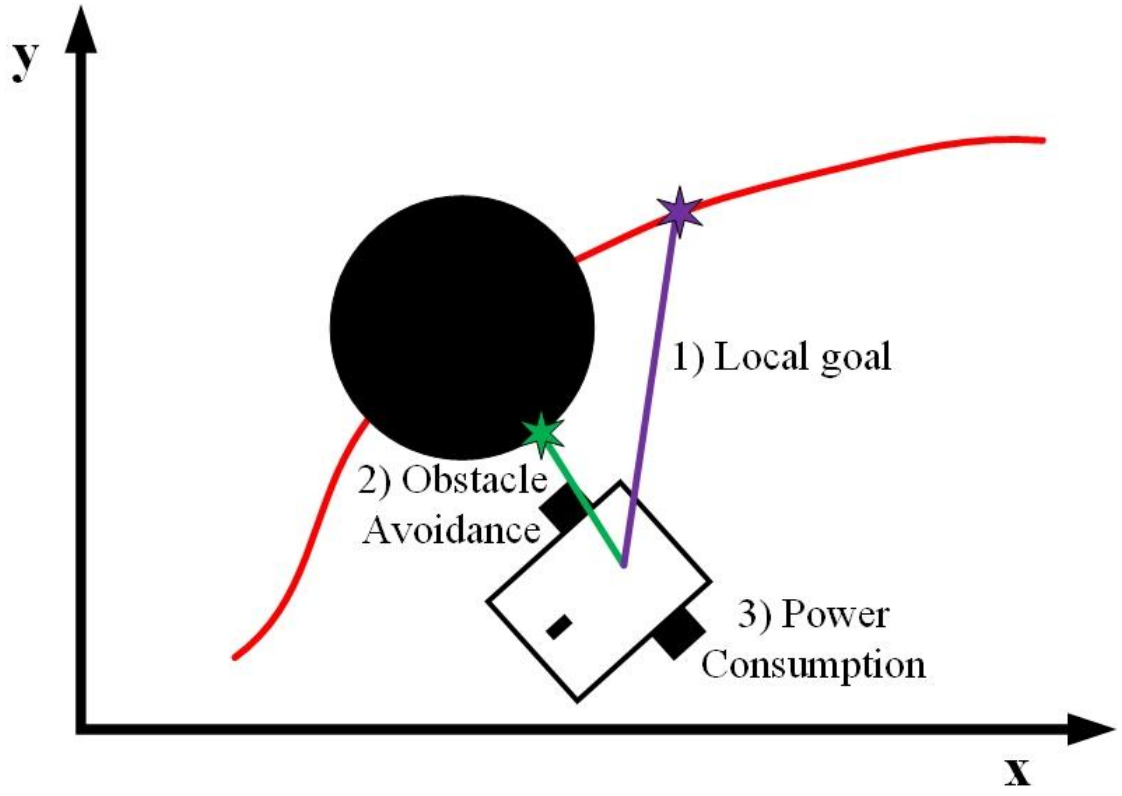


Fig 3.2. Visual representation of the objectives

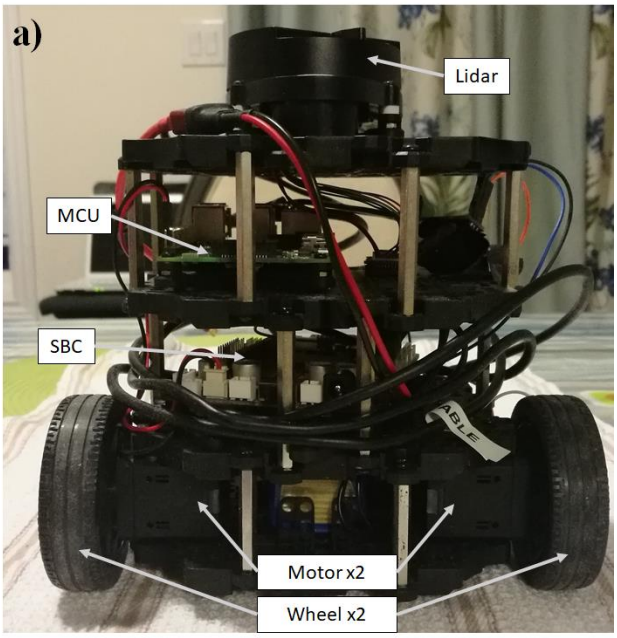
The platform that will be used to analyze this problem is the TurtleBot3, a differential-driven 3-wheeled robot operating system (ROS) based robot, and a Linux computer that contains Matlab. The specifications of the platforms and how they communicate would be discussed.

3.3 Platform Specifications

The sections below list the specifications of the apparatuses used.

3.3.1 TurtleBot 3

The images below show how the physical appearance of the TurtleBot3:



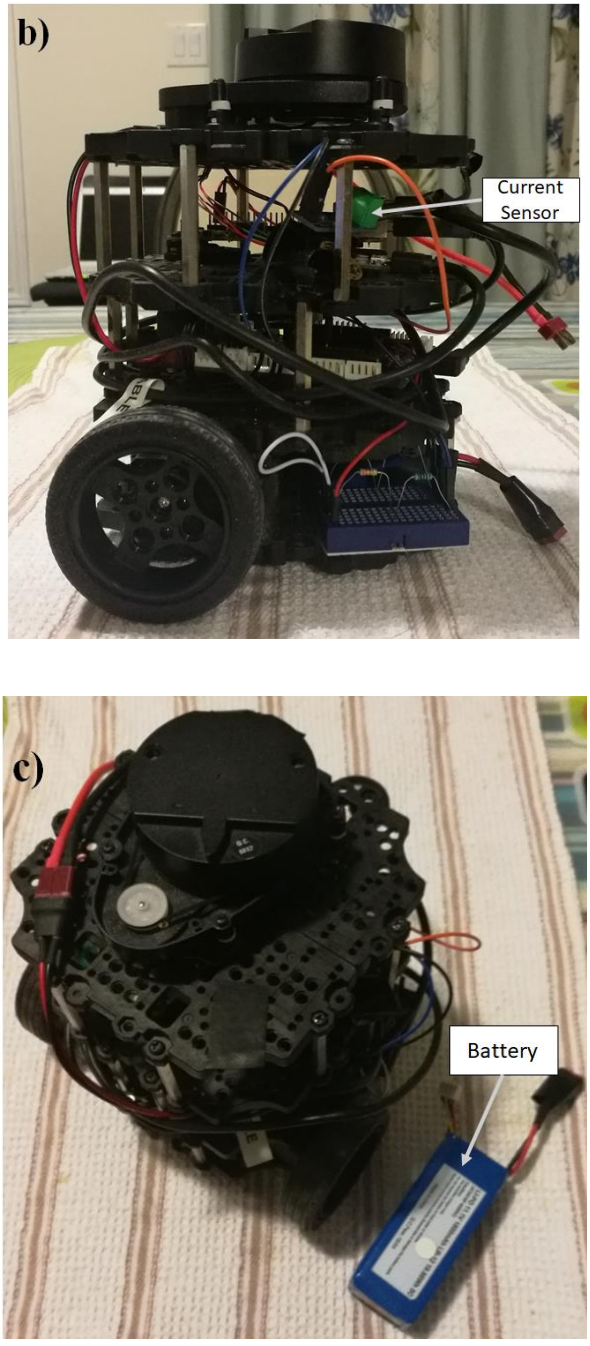


Fig 3.3. TurtleBot3 a) Front view, b) Side view, c) Angled view

The TurtleBot3 is a small (L:138 mm, W: 178 mm, H: 192 mm) 3-wheeled differential drive robot with a third caster wheel for stability. The system is coded under the robot

operating system (ROS), an open-source robot software development system. It consists of a Lidar, IMU, two motors, and an 11.1V LiPo battery. The full hardware specifications are listed in the following table:

Table 3.1 TurtleBot3 specifications [21]

Specification	**Value**
Max translation velocity	0.22 m/s
Max rotational velocity	2.84 rad/s
Max payload	15 kg
Size (LxWxH)	138mm x 178mm x 192mm
Weight	1.1 kg
Climbing Threshold	10mm or lower
Single Board Computer (SBC)	Raspberry Pi3 Model B+
Microcontroller Unit (MCU)	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Motors	XL430-W250 (Cored)
Laser Distance Sensor	360 Laser Distance Sensor LDS-01
Inertial Measurement Unit	GyROScope 3 Axis

	Accelerometer 3 Axis Magnetometer 3 Axis
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
Current Sensor	ACS712 20A Hall Current Sensor

Table 3.2 Lidar specifications [21]

Lidar Specification	Value
Distance range	120mm ~ 3,500mm
Distance Accuracy	±15mm
Distance Precision	±10 mm

3.3.2 Host Computer

The host computer is an ASUS laptop with the following specifications:

Table 3.3 Host computer specifications

Specification	Detail
CPU	Intel Core i7 - 4500U, 1.8GHz
Memory	7.7 GB
HDD	1 TB
Operating System	Linux Ubuntu 16.04 LTS

3.4 Robot Communication

The code that dictates the control actions from NMPC is written in Matlab on the host computer, which allows for ease of debugging and implementation. Therefore, it is necessary to establish communication between Matlab and the robot. Additionally, unless

the TurtleBot is connected to a display through HDMI, it is unable to show the user how to run its functions without the use of a host computer. Thus, the robot operating system (ROS) serves as the communication link between the TurtleBot and Matlab.

The first stage is the communication between the host computer and the robot. This is done through ROS and Wi-Fi. On the host computer where ROS is installed, there is a module called the ROS master. This module overlooks all operations that occur within ROS, otherwise known as the “nodes”. These nodes send and receive messages, where the nodes that send are called publishers and nodes that receive are called subscribers. It is possible for a node to be both publisher and subscriber. The messages themselves are transmitted through a uniquely named topic on the ROS network. Figure 4.2 shows how the ROS master and nodes connect and communicate with each other. The ROS master module uses an IP address specifically for the laptop (based on the Wi-Fi) and this same address is placed on the TurtleBot computer for the two systems to send messages to each other over the network. Once the Wi-Fi connection is set up, the host computer is now able to control the TurtleBot [21].

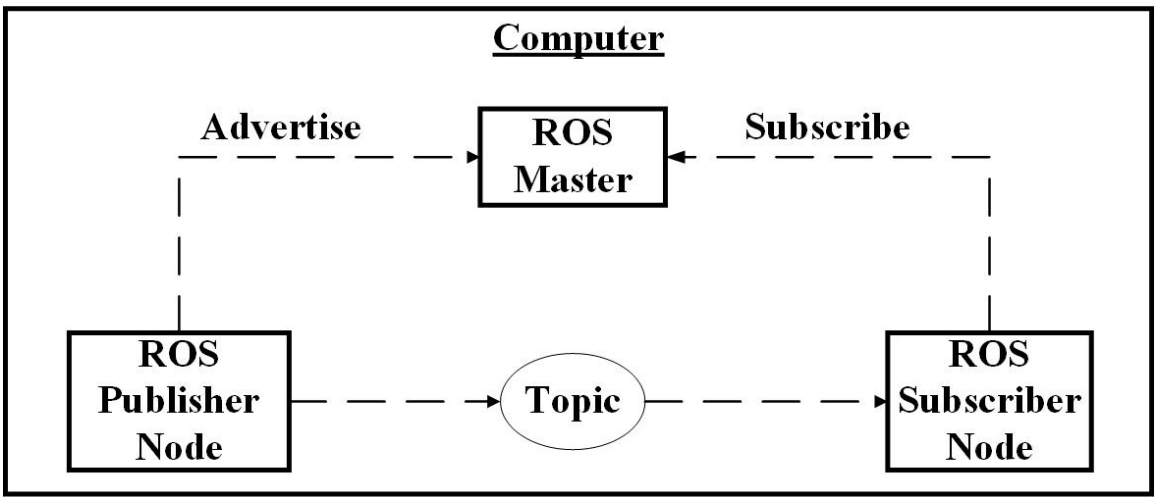


Fig 3.4. Relation between ROS master and ROS nodes

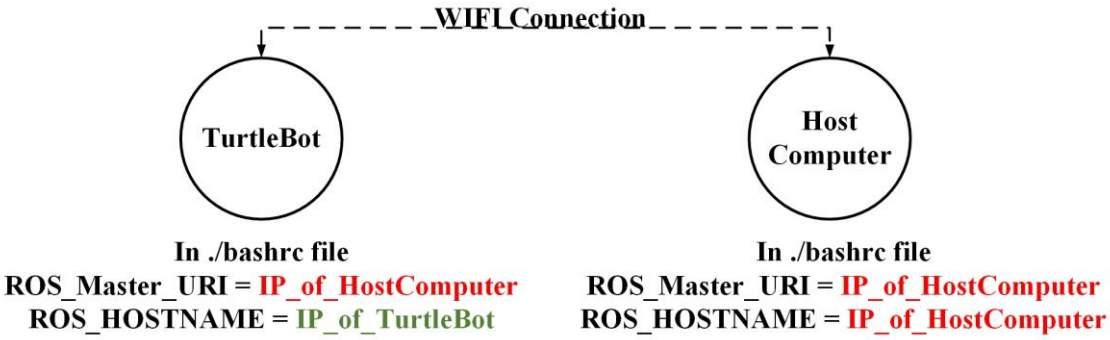


Fig 3.5. Requirements for TurtleBot3 communication

Once the connection between the host computer and the TurtleBot has been established and tested, Matlab has to be connected to the ROS system. Matlab requires a toolbox that allows it to communicate with the nodes on ROS. Once the toolbox has been acquired, the line *rosinit* is launched on the Matlab command window and a global node is generated for Matlab on ROS, thus allowing Matlab to subscribe and publish messages on ROS. From

there, the TurtleBot would be able to read messages that Matlab publishes, establishing communication between the two devices.

A program is run on the TurtleBot through the host computer, instructing it to start sending data to the host computer. This sets up the initial state of the robot and is always set as (0,0,0) on the odometry settings. Figure 4.4 shows a simplified version of the architecture of the communication. Figure 4.5 shows the connection between the nodes and topics on ROS, with table 4.4 giving an explanation of each node, topic and what they publish and subscribe to.

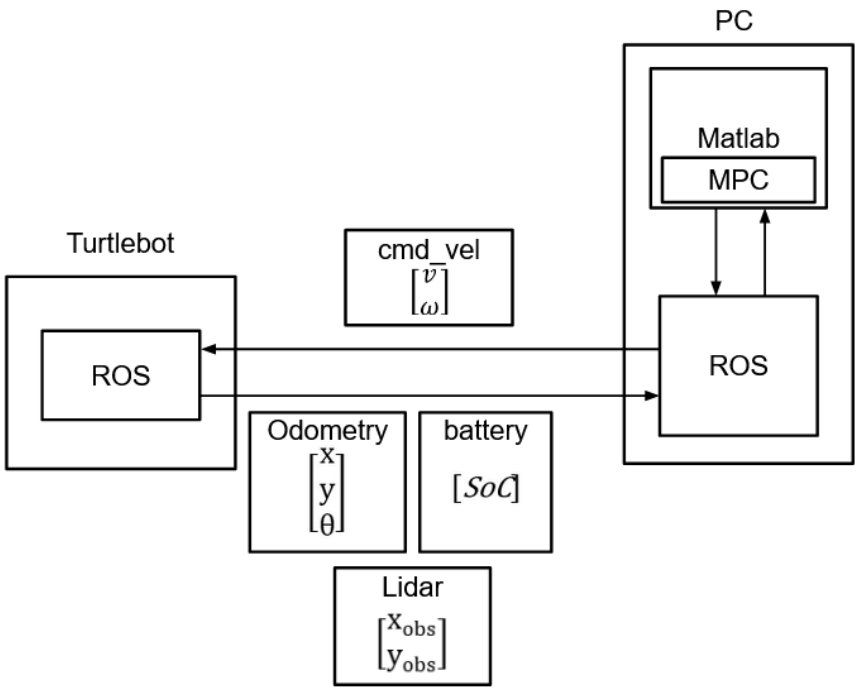


Fig 3.6. Simplified communication architecture of the robot and host computers

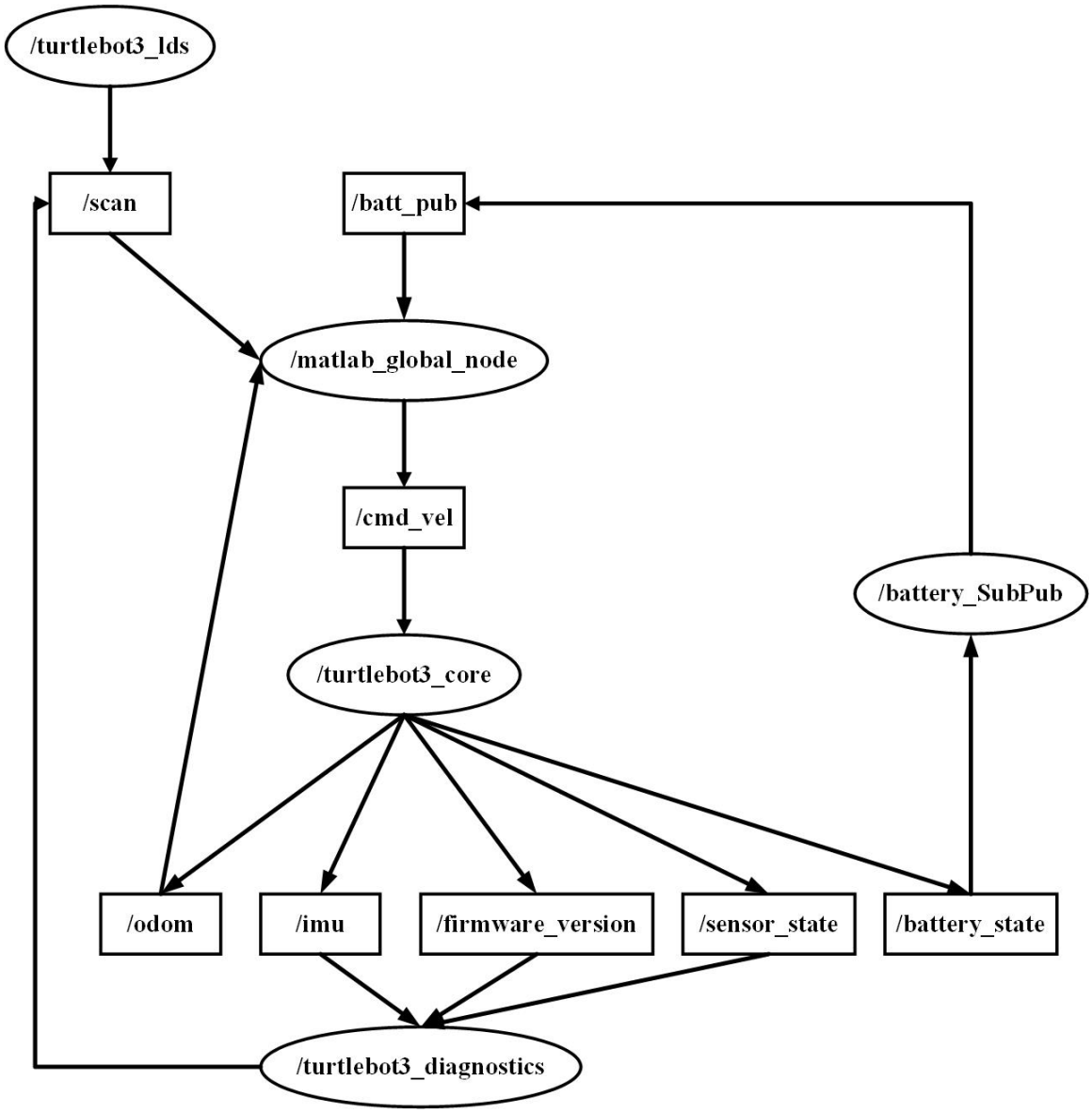


Fig 3.7. ROSgraph connections of the nodes and topics

Table 3.4 Node and topic explanation of Fig. 4.5

Name	**Description**	**Publish/Subscribe**
Nodes		
TurtleBot3_lds	reads the laser information and converts it into user-friendly messages on *scan*	Publish to *scan*
TurtleBot3_core	sets up all publisher and subscribers that are needed to run the robot and is where all the main robot processes occur converts *v* and *w* into left and right wheel velocities	Subscribe to *cmd_vel* Publish to: - *battery_state* - *sensor_state* - *firmware_version* - *imu* - *odom*
TurtleBot3_diagnostics	contains the status of the components connected to the TurtleBot3	Subscribe to: - *sensor_state* - *firmware_version* - *imu* - *odom* - *scan*
Battery_SubPub	allows for *battery_state* topic messages to be transferred to the Matlab program and is a custom node because Matlab does not have a pre-existing subscriber for the *battey_state* topic	Subscribe to *battery_state* Publish to *batt_pub*
Matlab_global_node	allows for Matlab to read the battery data, the lidar scan and the odometry which then uses NMPC to find the control actions for the	Subscribe to: - *batt_pub* - *scan* - *odom*

	robot *v* and *w*	Publish to *cmd_vel*
Topics		
scan	2D point cloud scan of the environment with the ranges and angles from the robot in the robot's local coordinate system	Relationships are established under the nodes section.
cmd_vel	linear and angular velocity instructions	
battery_state	messages related to battery condition, current and voltage information	
sensor_state	messages that contain encoder values, battery voltage and torque	
firmware_version	message with the robot's type, firmware and software information	
imu	message containing the velocity and acceleration of the robot	
odom	message containing the odometry of the robot which is the orientation and position based on encoder data	
batt_pub	custom topic that copies the information from battery state into the Matlab_global_node	

To summarize the table, the TurtleBot3 gathers the robot's environment and state data such as the obstacle location, current x-y position, orientation and battery state, then sends the

information to the host computer through Wi-Fi. The host computer conveys this information to Matlab, which uses this data to calculate a reference line to a user-defined goal, as well the control actions (v, w) using NMPC to direct the robot to the goal. This data is then sent back to the TurtleBot through ROS, where it is converted into the individual left and right wheel velocities using the following equation:

$$v_r = \frac{2v + \omega L}{2R} \tag{3.5}$$

$$v_l = \frac{2v - \omega L}{2R} \tag{3.6}$$

where L is the length between the centre of the two wheels (160mm) and R represents the radius of the wheel (66mm).

The robot and the computer continue their communication until the robot's program is stopped.

3.5 Data-Driven Power Consumption Prediction

Data-driven methods involve using multitudes of real data from either experimentation by the researcher or provided online by companies under various conditions. Data collection tends to be based on experimentation for what the researcher is looking for. Since the research described in this paper was conducted to predict power consumption based on the movement of the TurtleBot, motion data (which includes any variables related to motion such as position, velocity, and acceleration) was collected for this purpose. Battery data

(such as voltage and current) was also recorded, as it is required to calculate the power usage.

Initially, for a UAV platform, a neural network (NN) data-driven power consumption prediction model was developed and tested against a linear regression power consumption prediction model. The results indicated that the NN model was more accurate in its prediction. The results of this model were then placed in an NMPC control strategy framework in Matlab, which although the model itself was effective, the computational time was too slow for real-time computation [19]. Thus, the software Eureqa was used to generate a model for the UAV. It was shown to be decently accurate, and due to the model being merely an expression, it made the NMPC calculations a lot faster due to being more suited for Matlab and not requiring a GPU for faster computation.

The power consumption prediction model was developed using a software called Eureqa, that originally belonged to Nutonian but as of the date of writing is owned by DataRobot. Eureqa is more commonly used in industrial settings. [55, 56] demonstrate the effectiveness of the software and how it might be used. Eureqa is a symbolic regression program with supervised learning. The user inputs the data and sets up the building blocks (addition, subtraction, multiplication, division, exponential, etc.) that will be used to generate the expressions. The data is split according to the user's wants (in this case, 70% training and 30% validation). After the setup is complete, it begins to generate random equations through the evolutionary search algorithm, genetic algorithm, utilizing the more successful expressions and placing them with more combinations, in order to constantly improve the fitting of the expression until it can no longer find a better one. How fit the expression becomes is dependent on the inputted data and the building blocks it is allowed

to use. This fit is based on the R^2 goodness of fit. The software can be allowed to run for a long time before it finds a sufficient expression. The software also shows the maximum error, mean squared error, and mean absolute error for each expression it comes up with. Furthermore, the expression takes less time to calculate on Matlab since Matlab is more efficient with expression types of calculation and does not require any external hardware such as GPU to speed up the process.

The data values for the robot were obtained by driving the robot remotely using WASD keys through the program:

ROSlaunch TurtleBot3_teleop TurtleBot3_teleop_key.launch

where "teleop" stands for teleoperation and the "key" stands for the keyboard. This means that the robot is controlled at a distance. Once the robot had been started, a program was launched to start recording the data, and the robot runs through different movements until the battery dies (the range goes from 12.3 V to 11.2 V for each data collection run). The recorded data files include battery data (contains voltage and current which is then used to calculate power), command velocity (the velocity provided by the user or program to the robot), IMU (contains velocity and acceleration data), odometry (contains position and orientation) and joint state (contains individual wheel velocities) of the motors. The files were saved as comma-separated values (CSV) files that used epoch time to log the data. In this manner, it is possible to combine all files logging at different frequencies to generate a larger file, containing all the relevant variables with an even sampling time and lowering the amount of data that needs to be trained with. Given how the variables work, it was decided upon that the variables of some files (such as command velocity) were redundant for the motion of the robot. Once all the log files were obtained, due to the original files

having a significant amount of redundant variables, they were cleaned up to leave only a single CSV file that contained the wheel velocities, linear and angular velocity and acceleration, and power data. The variable of time was removed, since the time at which the movement occurs or is logged at is not a relevant variable for power consumption. For example, the robot can run at a linear velocity of 0.01 m/s and slowly consume its power, while moving at a linear velocity of 0.05 m/s will consume more power, thus allowing it to run for a shorter amount of time. Hence, time is too arbitrary of a variable to be used for the purpose of this research.

There were a variety of different movements used. Some of the movements served as training data, such as, for instance, moving in a constant straight line at a single linear velocity for as long as possible before being required to turn due to space limitations. These constant linear velocities range from 0.01 m/s to 0.2 m/s for forward linear velocity and -0.01m/s to -0.06 m/s for backward linear velocity. The changing angular velocities, with the unit of rad/s, include 0.1, 0.5, 1, -0.1, -0.5, and -1. It also has the motion of constant angular velocity with changing linear velocity. Going in a circle, clockwise and counterclockwise at different velocities, 8-shaped motion as well as speeding up and slowing down for both linear and angular velocities. After the data was gathered, it was then scaled to normalize the data. The scaling is based on the maximum and minimum values for each variable.

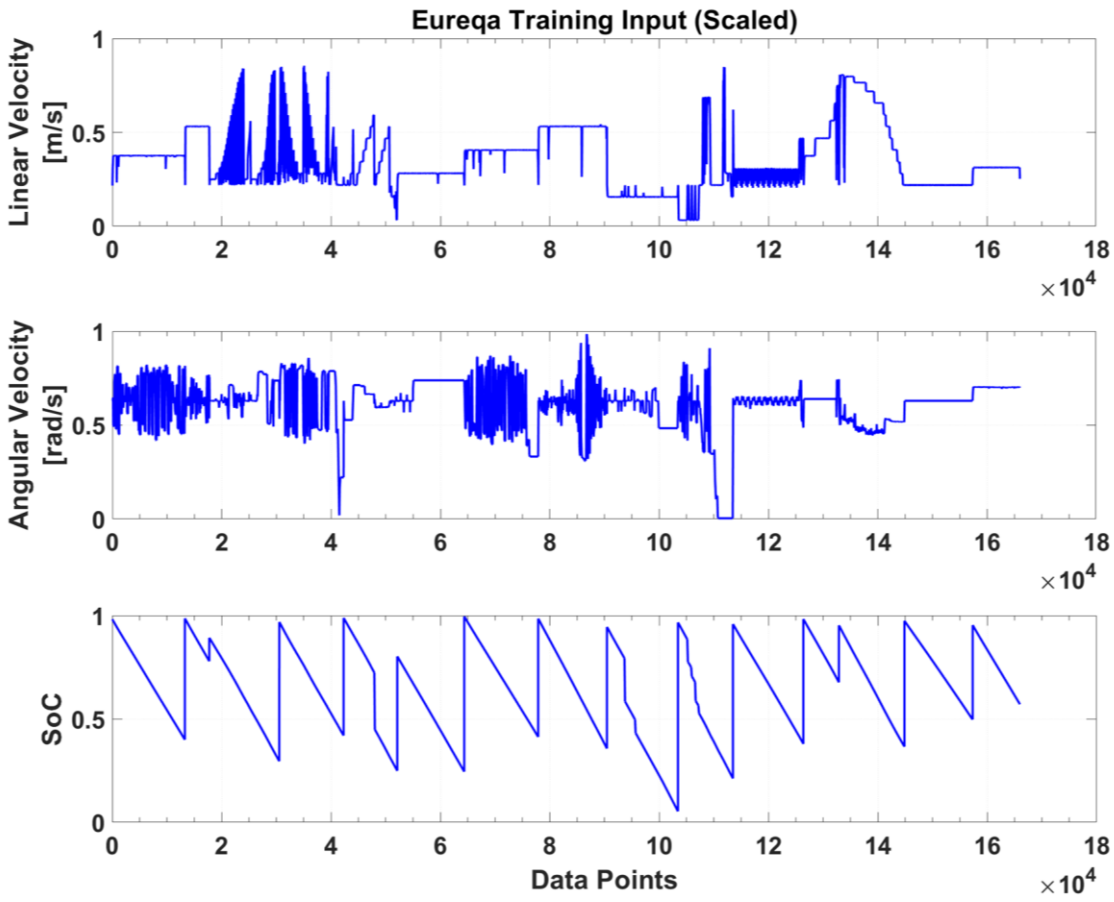


Fig 3.8. Final input data used in Eureqa training expression

Once the data has been normalized and narrowed down, it is trained under a machine learning algorithm to find matching patterns between the motional data and power consumption [57].

The final variables that were placed in the software are listed below:

Table 3.5 Variables used in Eureqa

Variable Symbol	**Variable Description**
Inputs	
v	Linear Velocity
w	Angular Velocity
v_L	Left Wheel Velocity
v_R	Right Wheel Velocity
a	Linear acceleration
α	Angular acceleration
SoC	State of Charge
Output	
P	Power

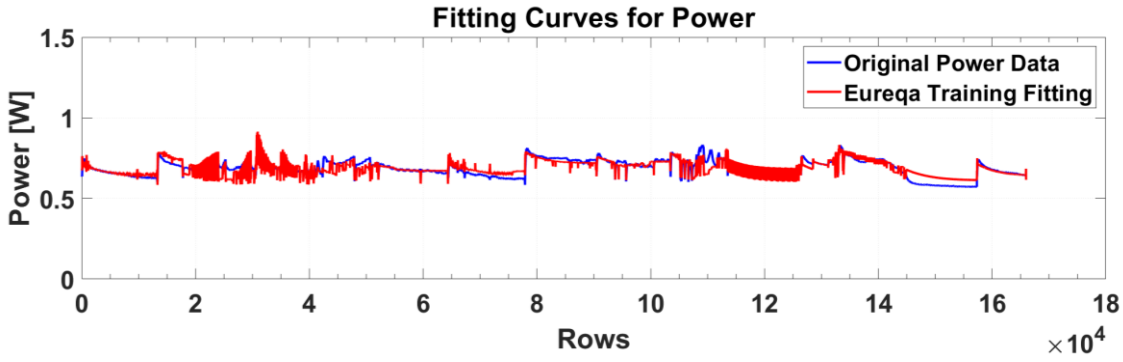


Fig 3.9. Eureqa fitting curves for power prediction

The equation that was produces by Eureqa is shown below:

$$J_{power} = 0.693 + 0.404v\omega + 0.196\omega^2(SoC^6) + (0.008 + \frac{0.010sin((v - 0.739)^{-3}))}{(v - 0.222\omega)} \quad (3.8)$$

This shows the variables that are used to create the final expression. Linear and angular velocities, along with SoC, are the most relevant variables to the development of this expression. This expression was then tested with randomly selected data to check for accuracy.

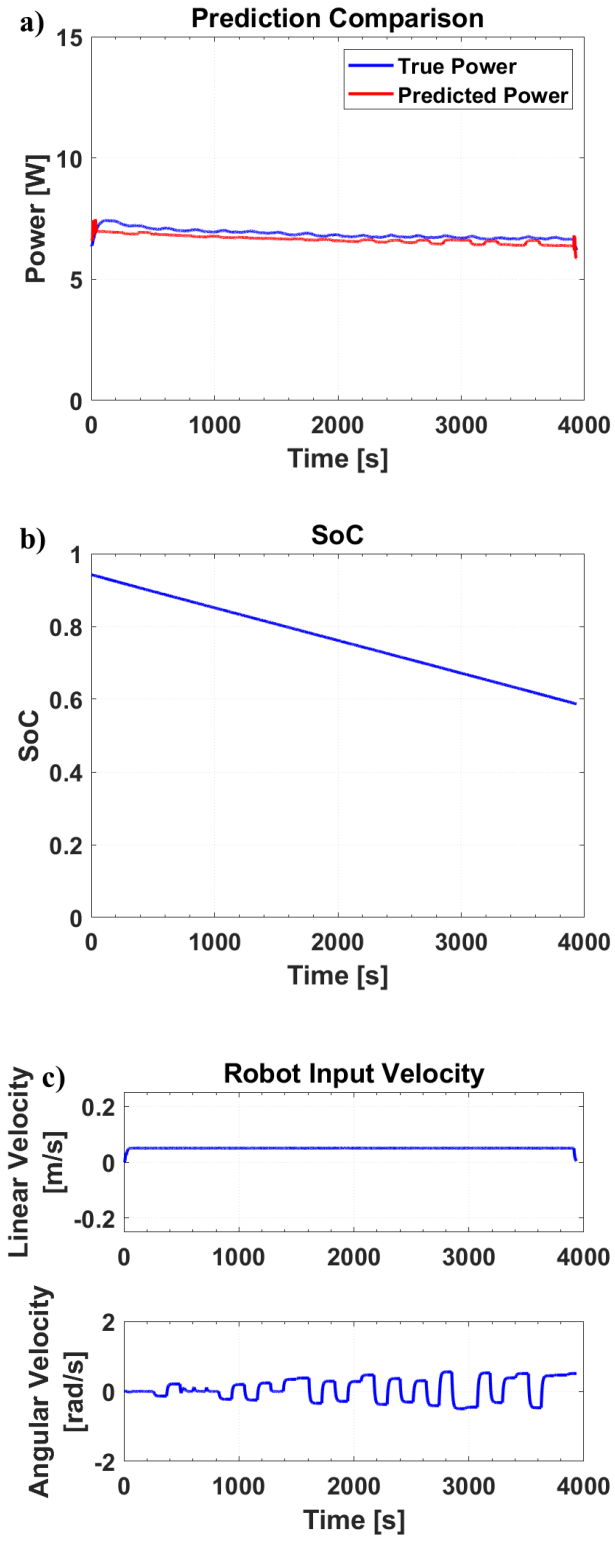


Fig 3.10. Constant linear velocity, turning angular velocity (a) Power Prediction, (b) SoC, (c) Velocity

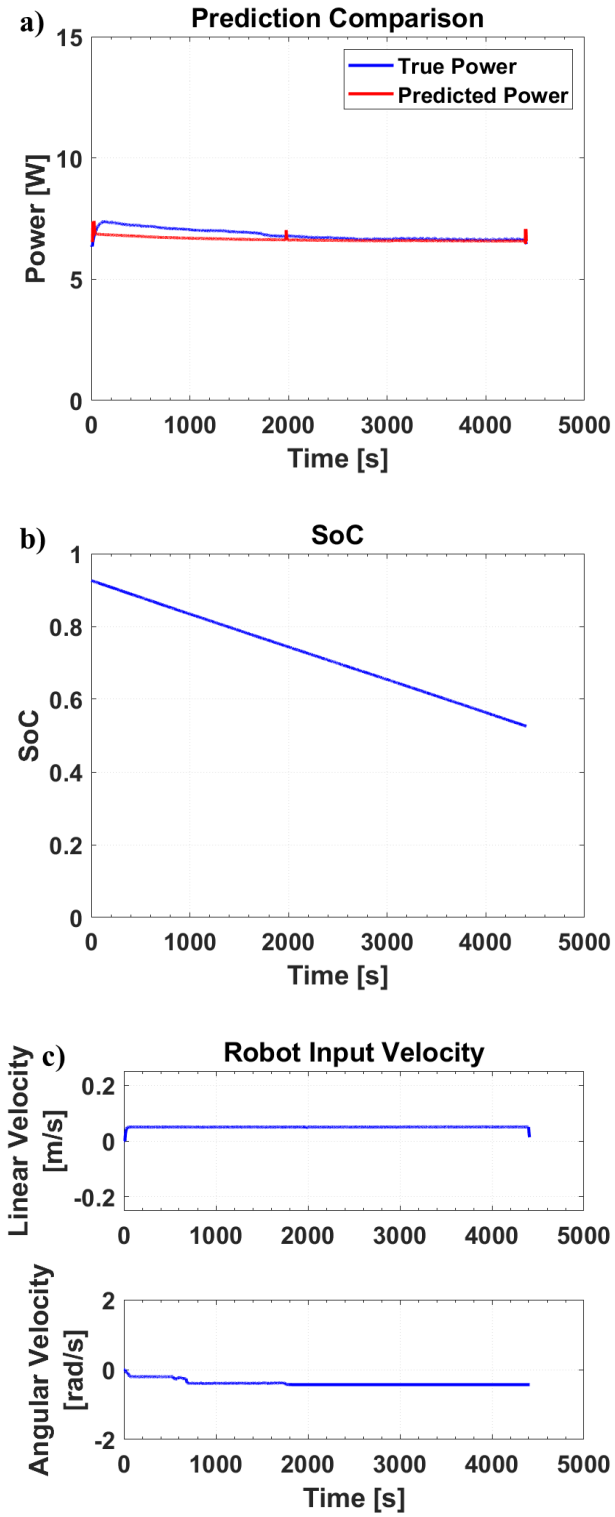


Fig 3.11. Constant linear velocity, turning left angular velocity (a) Power Prediction, (b) SoC, (c) Velocity

The results of the power prediction, when compared with the true power between test scenarios, are that most of the prediction errors are within the range of 10%. The only exceptions are when the robot is idle for the entire duration of the discharge. Though the expression is not perfect, it does show that it is capable of generating an adequate prediction of power usage based on the motion data. It is noted that Fig. 3.11 does have a larger discrepancy in regards to the prediction at the beginning. This can be due to the motion not being as well represented by the input data since the model is derived from data values. As such, there is always room for improvement in regards to the data that is inputted into Eureqa and even if the motion itself may be similar, the exact values may differ hence the discrepencies. However, the results still show a sufficient level of power prediction and the benefits of using it out weigh the cost. As such, the benefits to using a data-driven model are that it is more robust, removes the inclusion of more complex modelling, and doesn't require the user to know the details of a vehicle model that may be difficult to find or calculate.

3.6 Non-Linear Model Predictive Control Strategy

Model predictive control is a control strategy that takes in the reference and current state of the robot and runs it through a cost function, helping it determine the optimal control actions for the robot over a prediction horizon. In the case of the MPC, the reference input for this robot is the pre-planned path that comes from the RRT*, as shown in the architecture below.

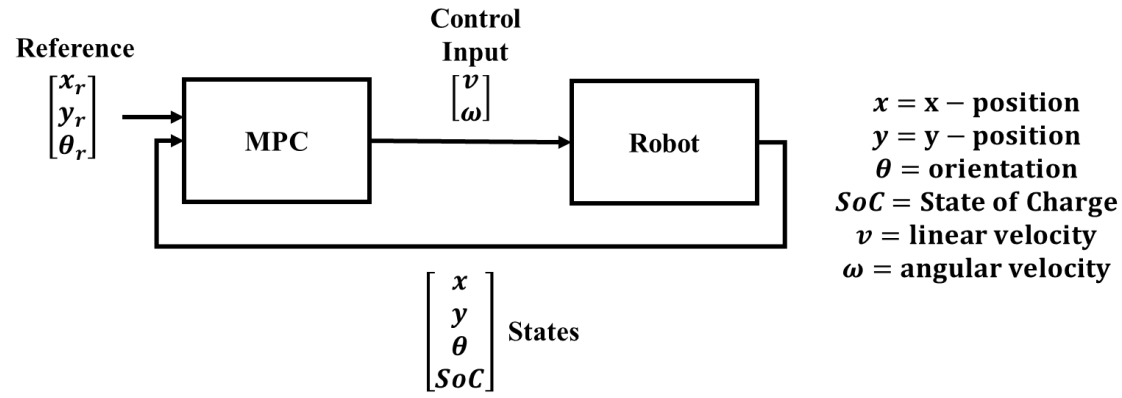


Fig 3.12. General architecture of MPC with the robot

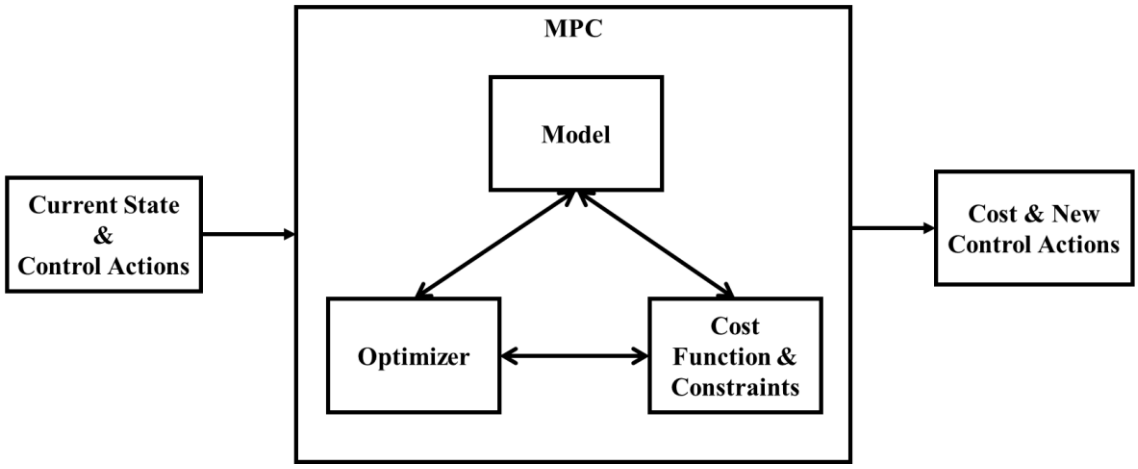


Fig 3.13. Internal MPC structure

The cost function of the MPC (which dictates the movement of the robot) consists of the power consumption prediction cost from the equation in the previous section, the potential field obstacle avoidance cost, and the local goal cost that determines the next goal of the robot on its current reference path.

The local goal cost minimizes the distance between the prediction horizon and the local goal. The aforementioned local goal is determined and set 29 points ahead of the reference point the robot is currently closest to.

The reference path is generated from RRT* which creates an optimal path with limited initial information and it takes a long time to generate. Traditionally, RRT* is required to rerun whenever the current path is insufficient in avoiding the obstacle which is costly. In this thesis, artificial potential field is proposed to handle the dynamic obstacle avoidance to avoid the high computational cost from recalculating the path using RRT*.

As mentioned previously, the dynamic obstacle avoidance is based on an artificial potential field where the obstacles "repulse" the robot while the goal "attracts" it. This means that if the robot gets closer to the obstacle, the cost will increase, hence the repelling force. Conversely, the attraction force will decrease the cost instead. However, this only occurs when two conditions are satisfied. The first condition requires that the obstacle intersects the reference path and the second mandates that the closest obstacle point is a threshold distance from the obstacle. In this case, the threshold distance is written to be 45 cm. This is done so that the robot does not unnecessarily avoid obstacles that are not in its way. The potential field expression is shown in equation (3.9 – 3.11):

$$J_{obs} = \alpha_{obs} * e^{-\mu_{obs}*((x_p - x_{obs})^2 + (y_p - y_{obs})^2)} \tag{3.9}$$

$$J_{goal} = -\alpha_{goal} * e^{-\mu_{goal}*((x_p - x_{goal})^2 + (y_p - y_{goal})^2)} \tag{3.10}$$

$$J_{OA} = J_{obs} + J_{goal} \tag{3.11}$$

where J_{obs}, J_{goal} is the potential cost for the obstacle and local goal. α and μ represent the height and width of the repulsion (8 and 15 for obstacle potential) and attraction (1 and 12

for local goal potential). The values for the height and width were determined manually through observation of the robot's avoidance capability around the obstacle. $x_p, y_p, x_{obs}, y_{obs}, x_{goal}, y_{goal}$ are the x, y coordinate points for predicted position, closest obstacle point and local goal point. J_{OA} is the total cost of obstacle avoidance. The obstacle point coordinates and goal point coordinates within the cost function is fixed for the specific time step whilst the predicted positions are changing with respect to the control actions linear and angular velocity. With the use of the exponential, a large slope can be generated showcasing that the closer the robot gets to the obstacle avoidance point, the stronger the repulsive push value whereas for the goal point, it would be a stronger pull value. Thus the cost gets exponentially increased or decreased depending on how far the robot is in prediction for the repulsive and attractive forces.

The closest obstacle point coordinates, x_{obs}, y_{obs} were determined using simultaneous localization and mapping (SLAM) technique based on the point cloud from the lidar sensor. SLAM is the robot's ability to construct and update a map in an unknown environment while keeping track of it's location within the environment. This means that as the robot moves, a new scan of the robot's environment is obtained from the lidar sensor, generating a new point cloud for the robot's local coordinate system at each time step. The values of this point cloud is provided in the form of x and y coordinates where the robot is always considered to be at the centre point. These values then gets converted into the global coordinate system for mapping and the robot's state is used to determine the current position of the robot in the global coordinate system. From these coordinate values, the distance the point is from the robot is calculated for all the possible obstacle position point,

From this information, the closest obstacle point in the global coordinate system is obtained which is the coordinate that is utilized by the artificial potential field algorithm.

The NMPC parameter are shown in Table 3.6. The time step was chosen based on the calculation and communication time between the host computer and the robot. The horizon values were manually determined through trial and error where the movement of the robot was used to examine the value. These value from the tuning results in movement with a minimal amount of oscillation. The number of states and outputs are shown in Fig. 3.12 with the variable of $x, y, \theta,$ and SoC and the number of inputs are the control actions, v, ω.

Table 3.6 NMPC Parameters

Parameter	**Value**
Prediction Horizon	4
Control Horizon	1
Time Step [s]	1.8
Number of States	4
Number of Outputs	4
Number of Inputs	2

3.6.1 NMPC Control Velocity Constraints

During the physical tests, it was discovered that there was heavy oscillation at higher speeds, which required experimentation to compare command velocity and measured velocity. The conditions analyzed were changing linear velocity, changing angular velocity, constant linear velocity at 0.01, 0.1, and 0.2 m/s with changing angular velocity, and changing linear velocity with a constant angular velocity at 0.1 rad/s. By testing these conditions, it was discovered that when the individual wheel velocities reached a certain threshold, command velocity ceased to matter. Regardless of code alterations, the robot would not allow more power to be sent to the motor, thus limiting the maximum speed. Fig. 3.13 shows the position for the velocity test, while Fig. 3.14 to 3.18 depicts the linear velocity, angular velocity, and velocities for the left and right wheel respectively. It observed that at lower velocities, the measured velocities are the same as the command velocities, whereas there is a gap between them at higher velocities. This indicates that at high velocities, the robot moves slower than the commanded velocity.

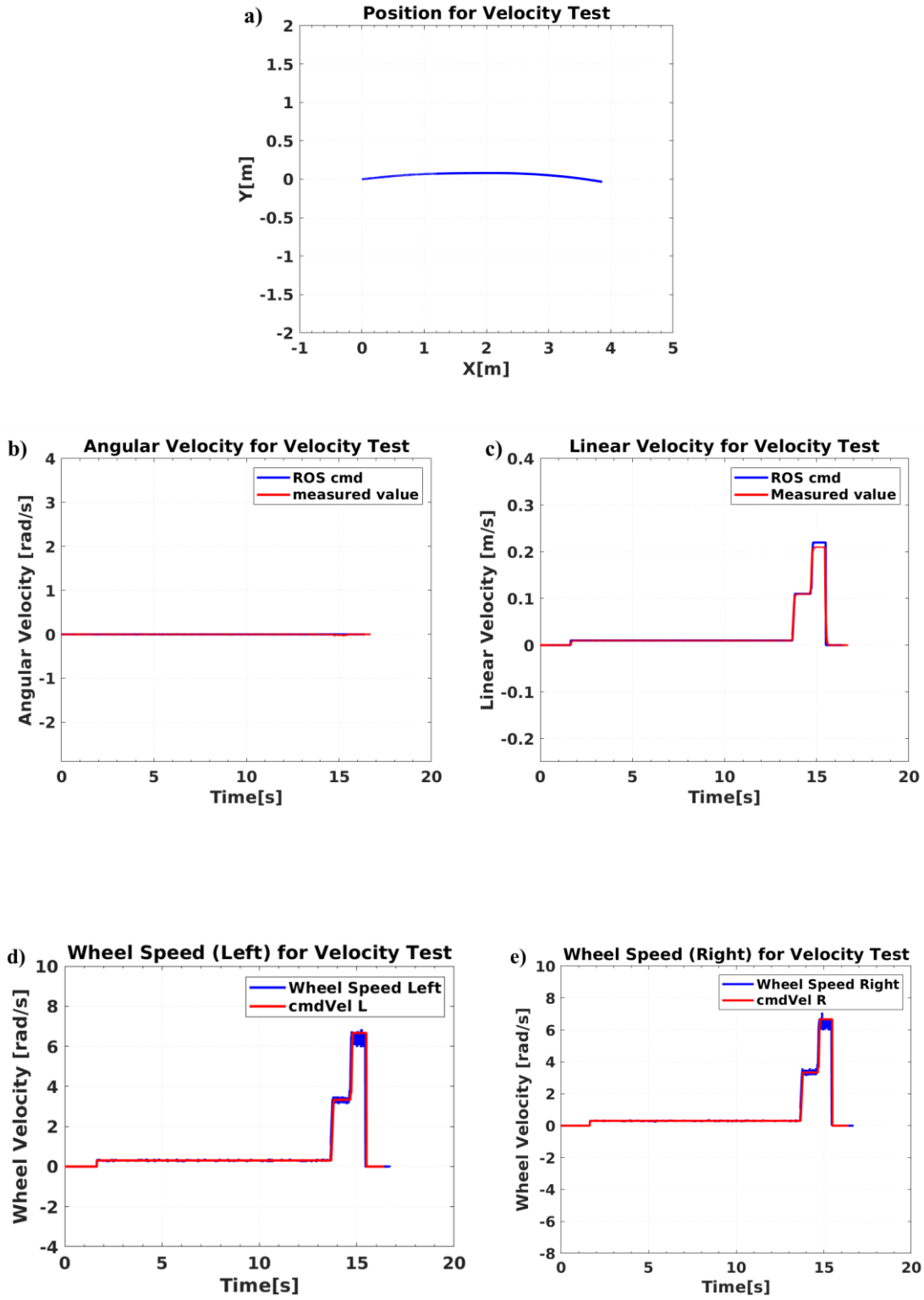


Fig 3.14. Speeding up linear velocity, zero angular velocity (a) Position, (b) Linear velocity, (c) Angular velocity, (d) Left wheel speed, (e) Right wheel speed

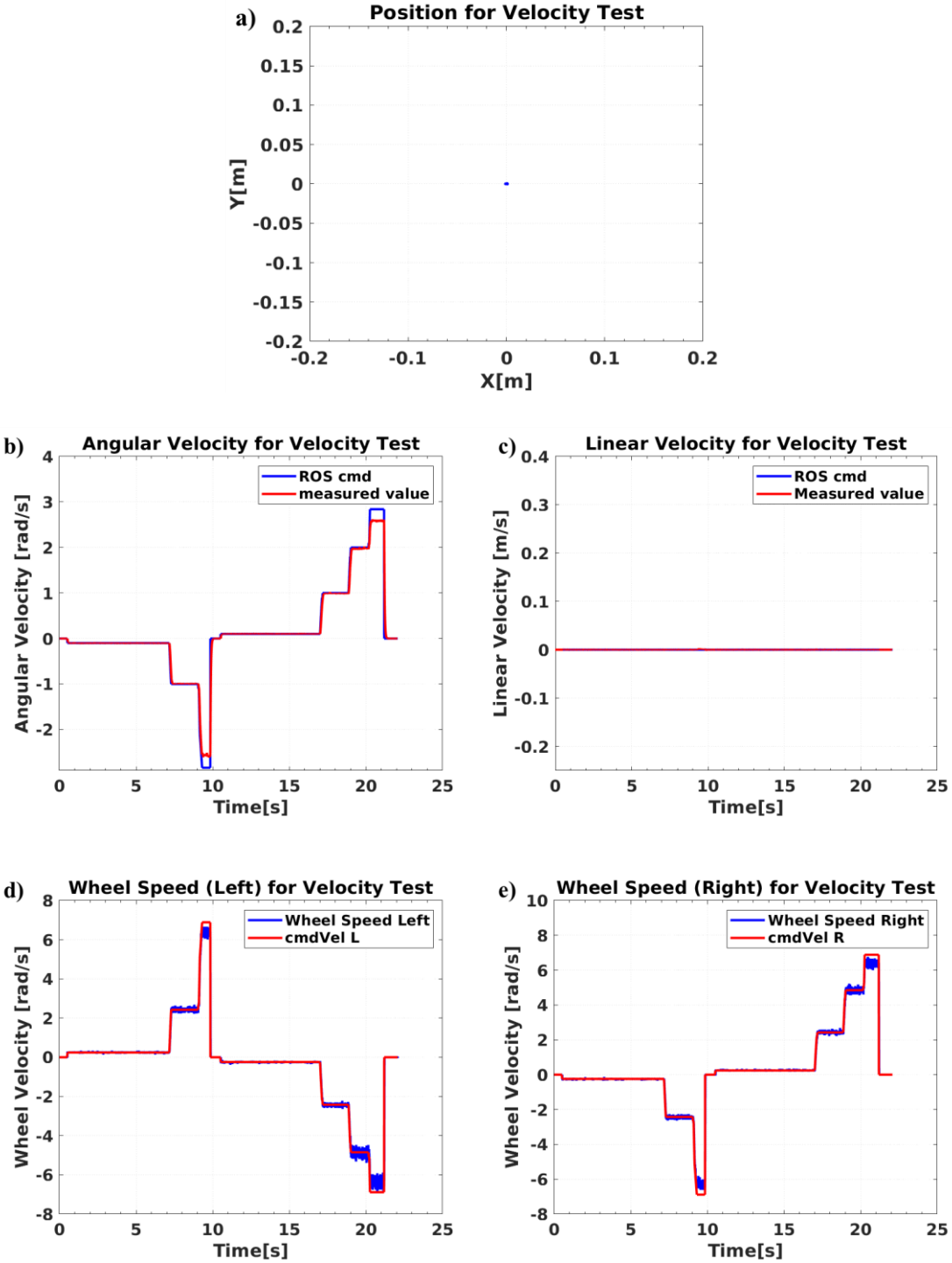


Fig 3.15. Zero linear velocity, changing angular velocity (a) Position, (b) Linear velocity, (c) Angular velocity, (d) Left wheel speed, (e) Right wheel speed

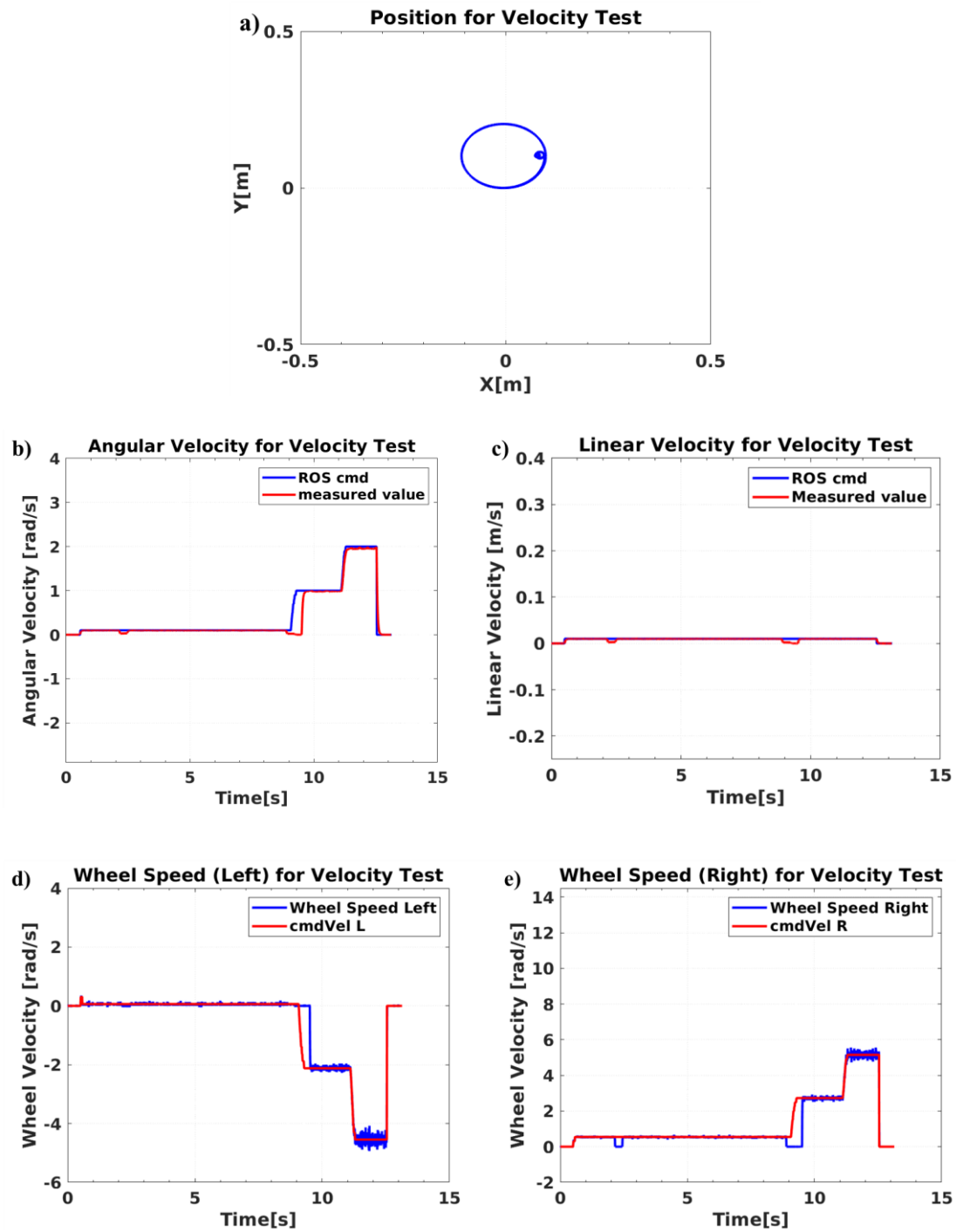


Fig 3.16. Linear velocity at 0.01m/s, changing angular velocity (a) Position, (b) Linear velocity, (c) Angular velocity, (d) Left wheel speed, (e) Right wheel speed

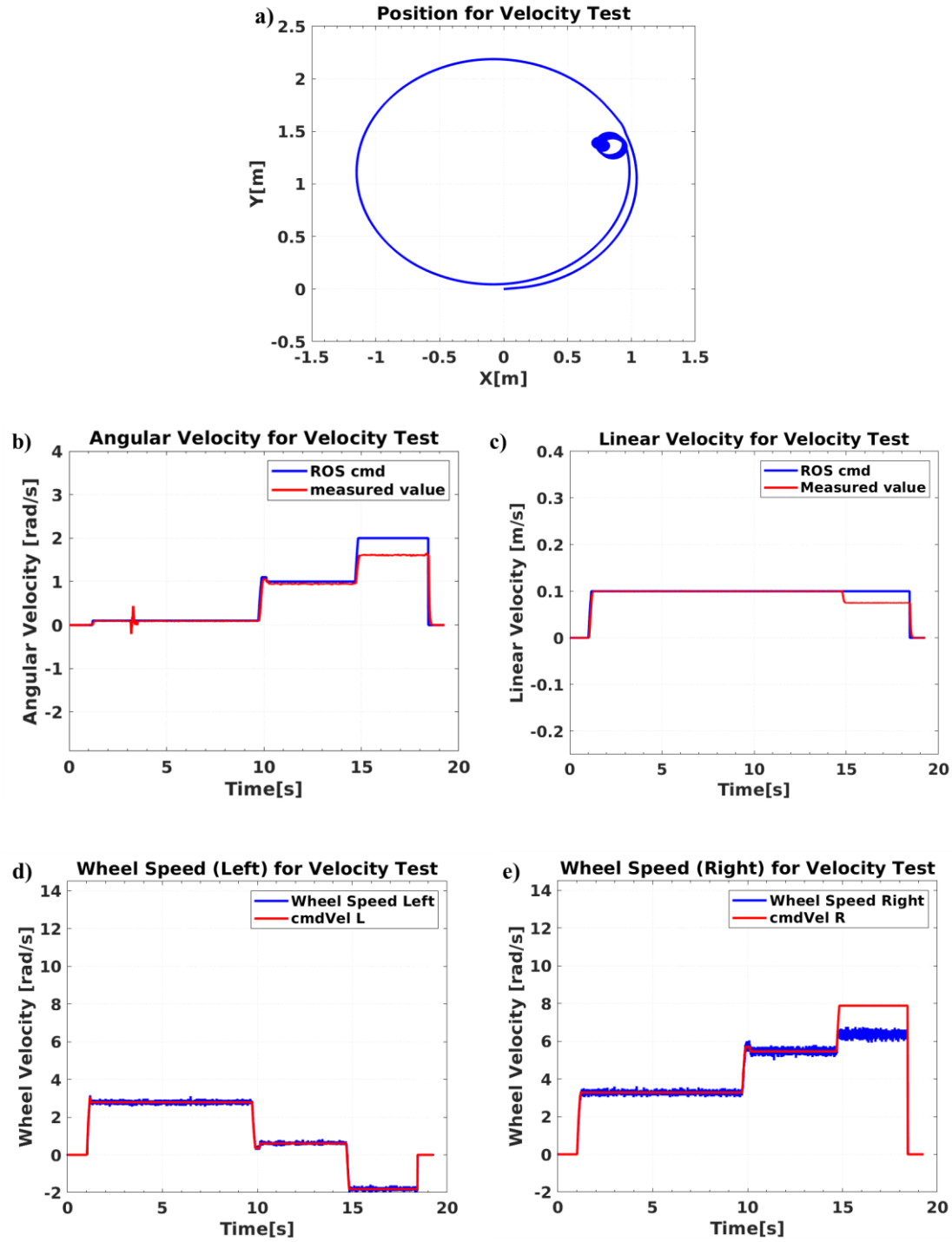


Fig 3.17. Linear velocity at 0.1m/s, changing angular velocity (a) Position, (b) Linear velocity, (c) Angular velocity, (d) Left wheel speed, (e) Right wheel speed

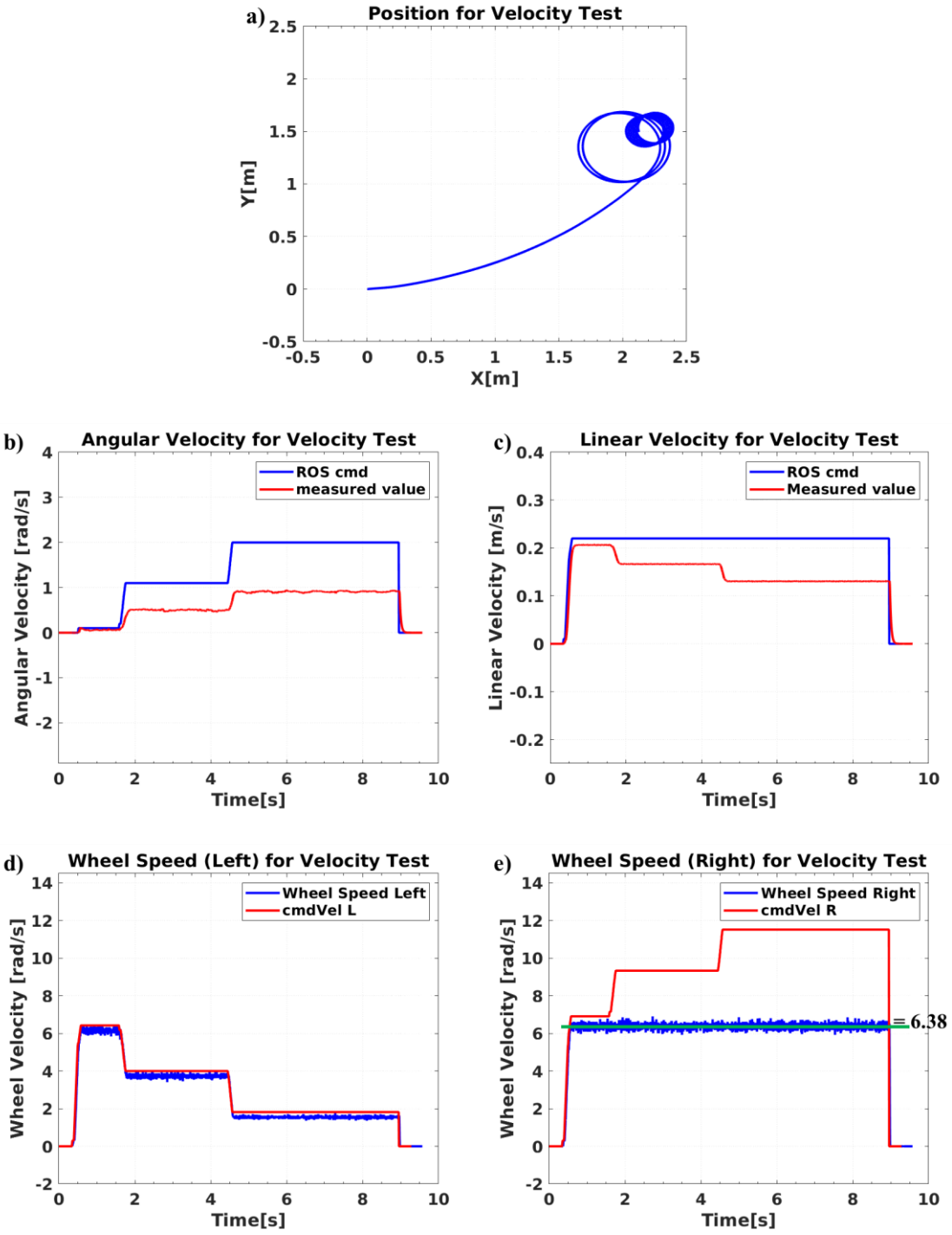


Fig 3.18. Linear velocity at 0.22m/s, changing angular velocity (a) Position, (b) Linear velocity, (c) Angular velocity, (d) Left wheel speed, (e) Right wheel speed

After further analysis, it was discovered that the motors have a limited specification for the speed shown in the table below:

Table 3.7 Motor rotation limitations based on voltage input [21]

Input Voltage [V]	**rev/min**	**rad/s**
9.0	47	4.92
11.1	57	5.97
12.0	61	6.39

Therefore, the linear and angular velocity constraints on the NMPC are limited to slower command velocities because the TurtleBot is physically incapable of reaching higher ones. This is due to a limitation placed on the TurtleBot's internal motor speed code for the individual velocities. To ensure that the TurtleBot would always reach its actual command velocity, the max amount of control actions that the NMPC allows is much lower than the max linear and angular velocities provided by the TurtleBot 3 manual.

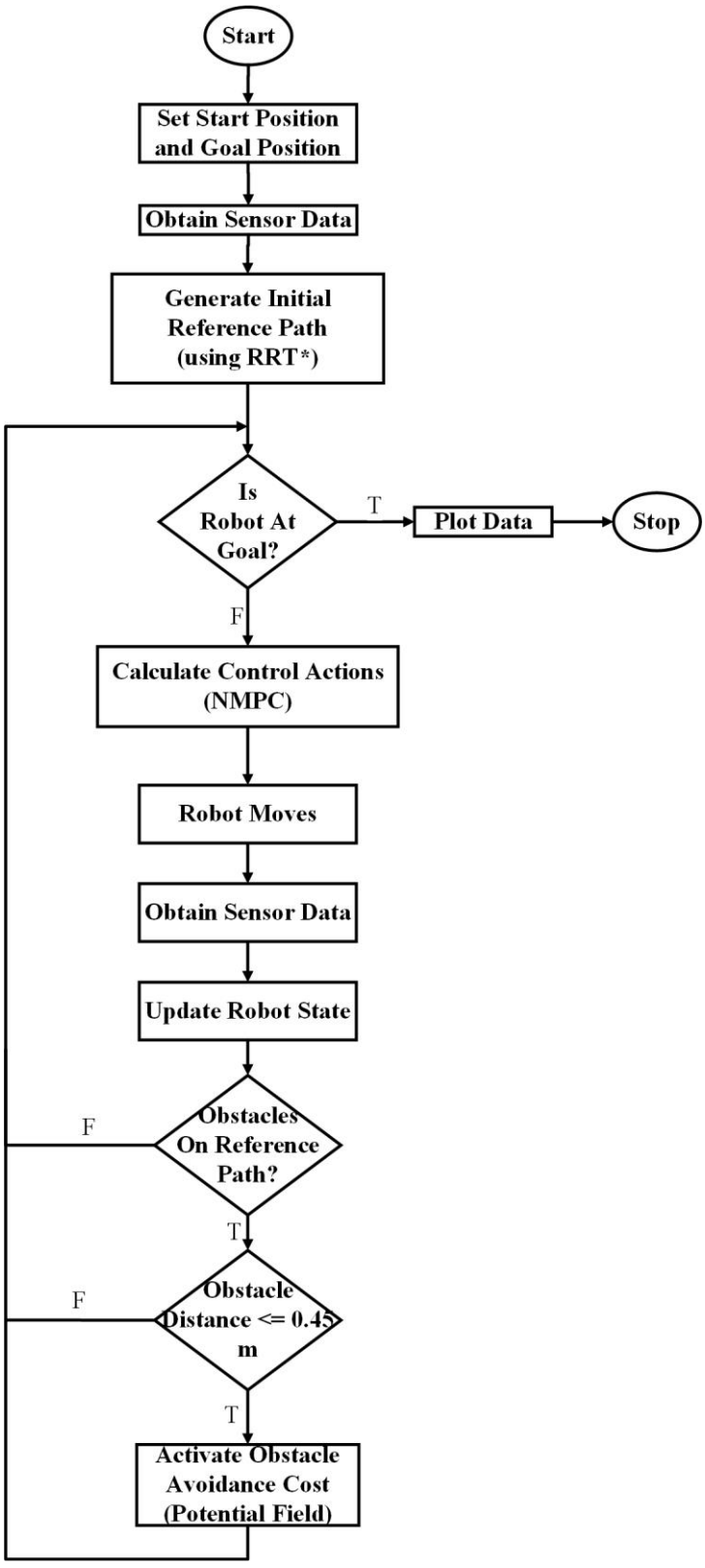


Fig 3.19. Algorithm flowchart

3.7 Summary

In this chapter, the goal of this study is defined by formulating the problem and the platform was defined.

To achieve this goal, the specifications of the mobile robot used in this study were analyzed and the method of communication between the TurtleBot3, the host computer and Matlab was established. Motion data of the TurtleBot3 and battery data were collected to be used in a supervised learning algorithm called Eureqa, generating an expression that models the power consumption of the TurtleBot based on motion. After testing a variety of variable combinations, it was linear and angular velocity, as well as SoC, that were the final variables deemed relevant to the model for power consumption. Afterwards, an NMPC based control strategy was designed for the TurtleBot3. Under this framework, the multi-objective problem was determined, with said objectives being 1) to follow the local goal on the reference path to get to the final goal, 2) to avoid obstacles along the path and 3) to use the power prediction model to conserve power consumption. A cost function was set for each of these objectives to generate the final algorithm used for the experimentation. Figure 4.17 shows a flowchart of the entire algorithm process.

Chapter 4. Results and Discussion

4.1 Experimental Setup

The robot was placed in an open-spaced room with obstacles of various sizes and shapes. There were five obstacle configurations tested, for conditions with and without the power consumption cost. The robot was then operated using MATLAB, which would coordinate with the ROS code to move the robot. Fig 4.1 shows the experiment environment without obstacles present. The areas within the green box are used as floor bump coverings. The yellow boxed items are obstacle indicators, marking the exact location of the obstacle. The grey circle is the starting position of the robot across all runs.

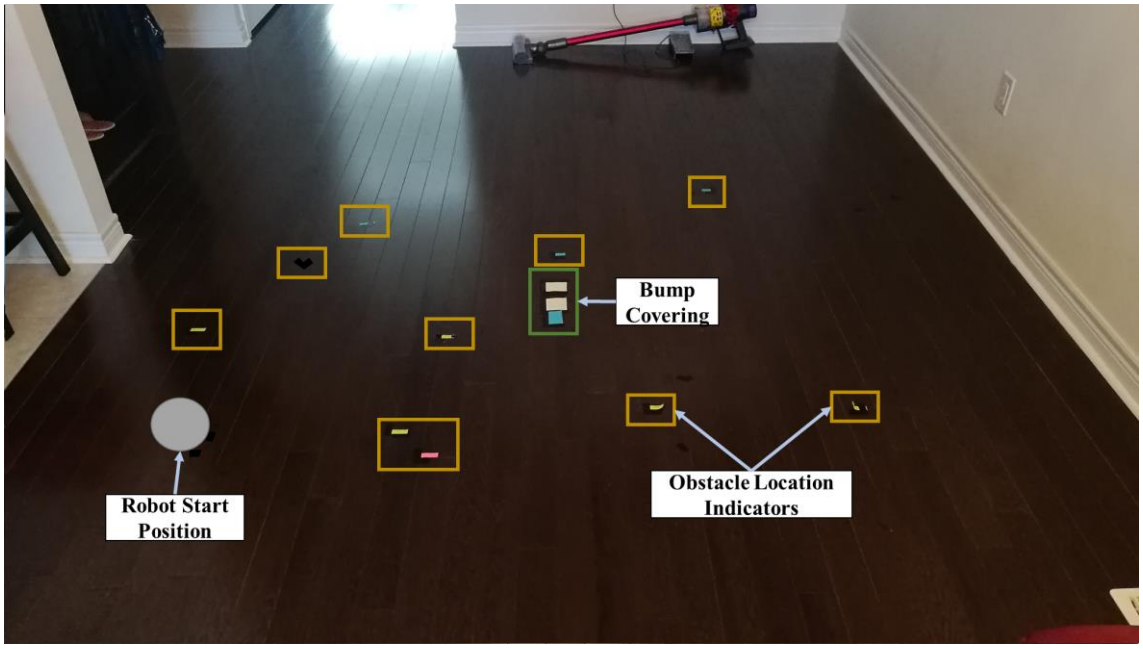


Fig 4.1. Test environment with no obstacles present

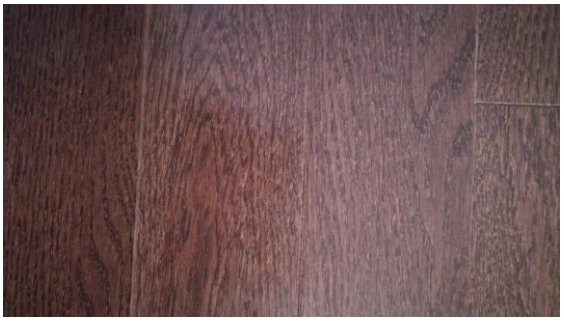


Fig 4.2. Close up of floorboards

Due to the floor being comprised of wooden planks, the surface was not perfectly even. Most of the time, the gaps had no effect on the movement of the robot. However, the area that is outlined by the green box has uneven flooring, creating a gap that was greater than the climbing threshold. This resulted in the caster wheel being caught between the gaps while the robot was moving at a slower speed, causing the TurtleBot to slip due to the main wheels continuing their momentum. This interfered with the encoder count hence why the affected area was covered with tape and sticky notes to smoothen the surface. It is unknown the exact reasoning for how this uneven flooring occurred. It may be due to poor installation, or relative humidity. This solution was effective in preventing the robot's caster wheel from being caught on the floor's surface on subsequent tests.

The obstacle location indicators are present to ensure the exact knowledge of the obstacle's position is known. During the testing phases of the algorithm, which included parameter testing, there were a few conditions in which the test would be considered an algorithm failure. These conditions include not following the reference path or hitting the obstacle. When the robot hits an obstacle (for example, obstacle three in fig 4.3), it may be light enough to displace. The indicators allow for the obstacles to be easily reset to their original

configurations, in the event that the obstacle itself is shifted even slightly. Additionally, they are more intuitive to the obstacle position when changing obstacle configuration. There are also sticky notes connected to the location indicators that have the obstacle configuration number written on them, specifying which configuration could be used at that location

As mentioned earlier, the grey circle represents the starting position of the robot for all obstacle configurations. This is to keep all test runs as uniform as possible. Tape was used to identify the robot's front and side positions in the environment to ensure it started at the same location as consistently as possible. Part of this is due to how the position is established when starting the TurtleBot. Wherever the TurtleBot starts up, the initial position is set at (0,0,0) since position count is determined by the encoder values of the robot. Furthermore, the (0,0) point on the occupancy grid map is also where the TurtleBot starts from when the map was generated, therefore to keep things simpler to keep track of, the robot always starts at the same position.

Since the experiment took place in a closed environment, wind is not present and the temperature is regulated. Therefore, neither of these were regarded as relevant factors. For temperature in particular, although normally an important consideration for its effects on battery usage, it did not present as a significant element as during the testing period, the room temperature remained at a constant of 25℃.

4.2 Obstacle Configuration

There are five different obstacle configurations to showcase the variety of scenarios that the robot can traverse in as shown from fig 4.3 to fig 4.7. The blue circle represents the goal.

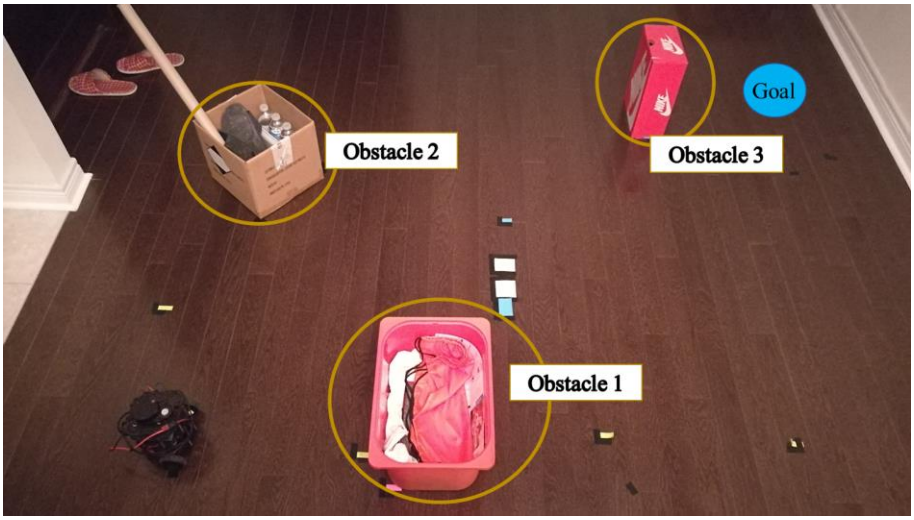


Fig 4.3. Obstacle configuration one

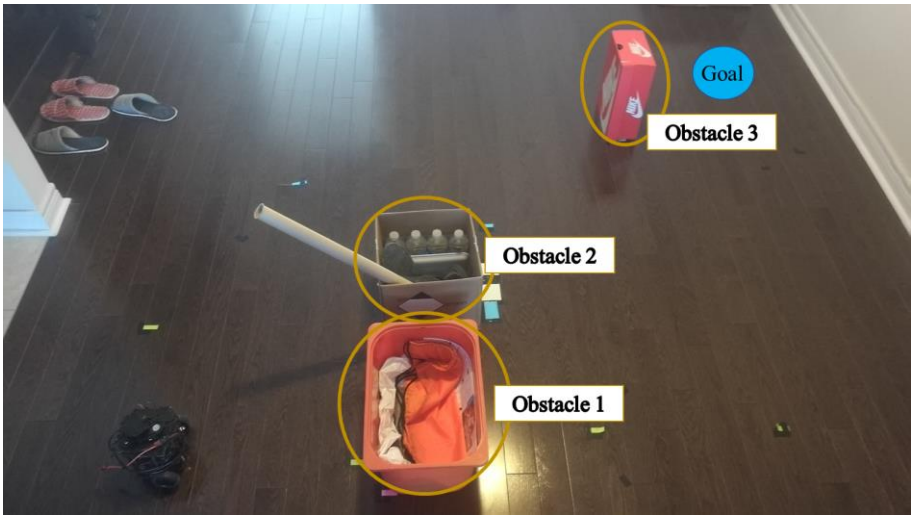


Fig 4.4. Obstacle configuration two

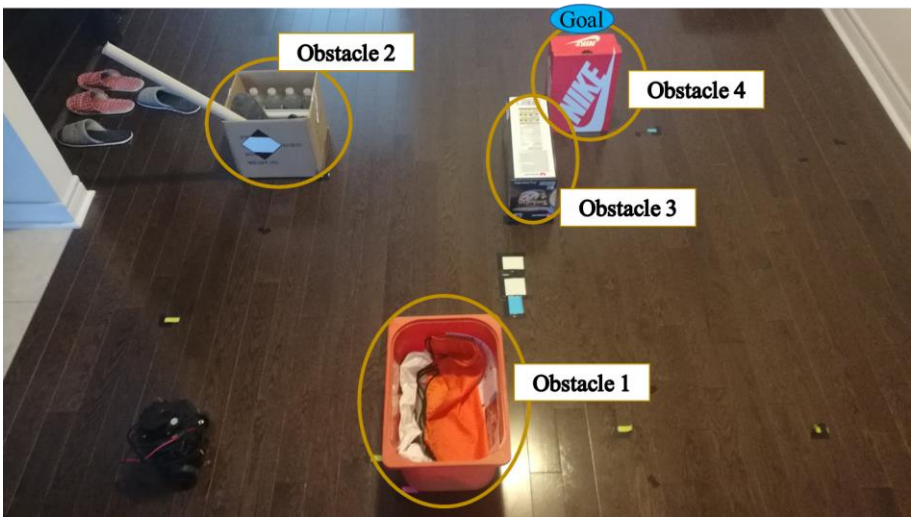


Fig 4.5. Obstacle configuration three

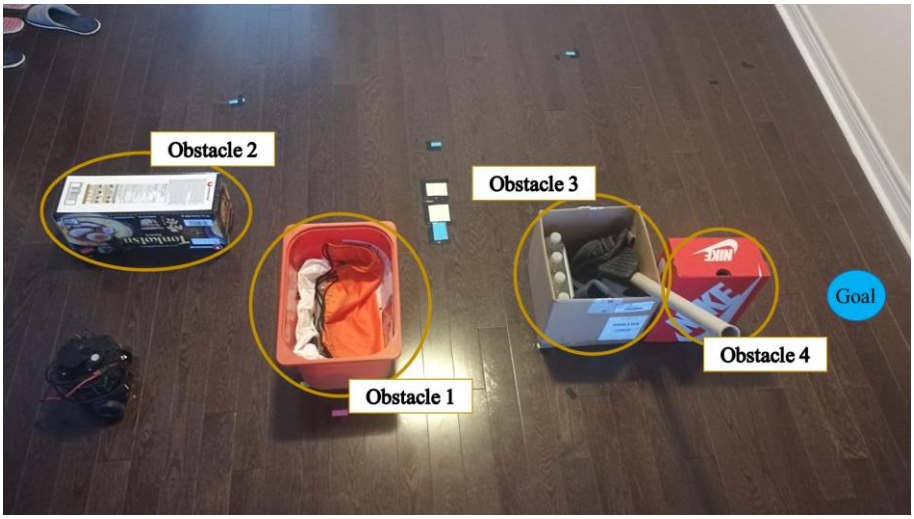


Fig 4.6. Obstacle configuration four

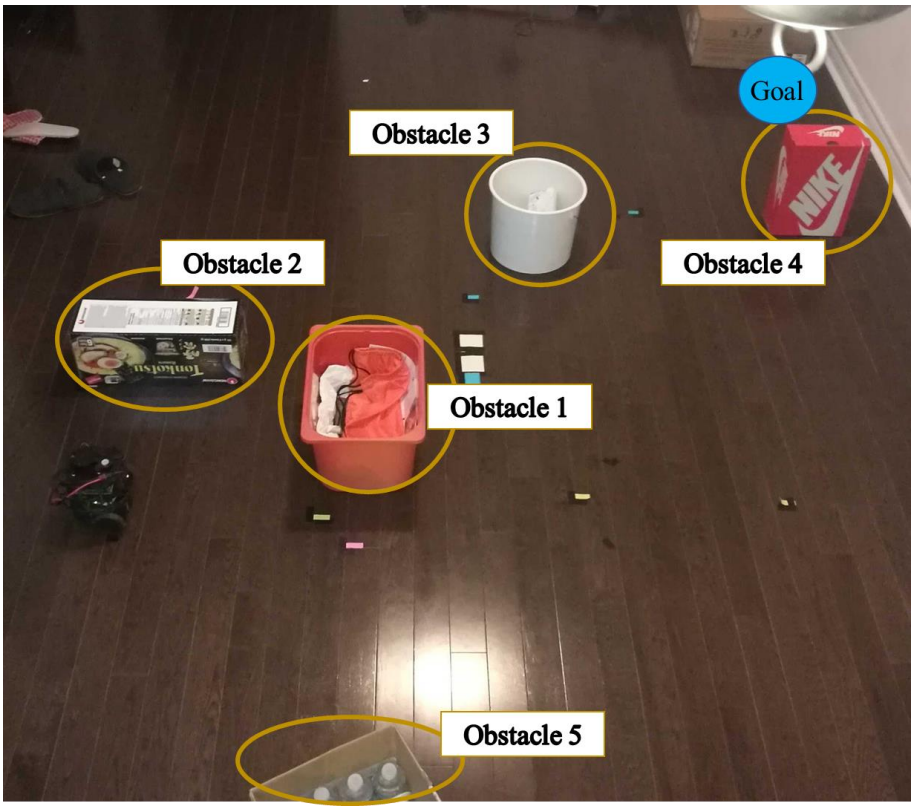


Fig 4.7. Obstacle configuration five

4.3 Experimental Procedure

The main objective of the experiments was to check the effectiveness of the path-planning algorithm and power consumption prediction model, in reducing the power consumed while the robot is moving.

The first step to accomplishing this is to develop an occupancy grid map. Occupancy grid maps determine the environment in the form of a grid and based on the lidar sensor, it determines where in the environment there is free space (and by extension, where there is not). There are two types of occupancy grid maps. The first type uses binary values. Therefore, each section of the grid map is labelled either 0 (free space) or 1 (occupied

space). The second type uses probabilistic values, which is a more detailed representation of the map due to its higher fidelity and robustness and uses values from 0 to 1. For example, in the case of a moving obstacle in the far distance, the algorithm may not be certain if the spot is occupied and will continue the original path it was on. The probability of the value for those grids would be low at that present moment. As the robot approaches, the probability value would increase, thereby initiating the rerouting process. However, if the obstacle moved out of the way, by the next time the robot checks the lidar data, the probability would have decreased again. For the experiments done in this research, the occupancy grid map is probability-based and the initial map was created offline.

To get the occupancy grid map, the robot is required to first drive around the empty environment without any obstacles. At the same time, it records what the lidar reads and the position at which it is read. Once the robot has driven around the area a few times, the occupancy grid map is generated as shown in the figure below.

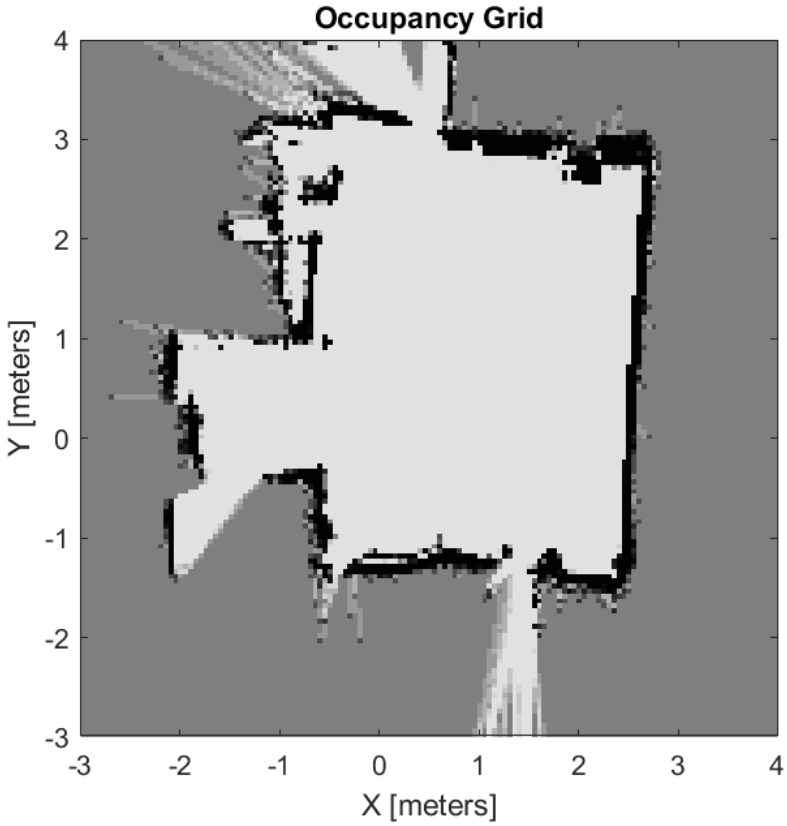


Fig 4.8. Occupancy grid map

The resolution of the occupancy grid map is 20. This means that there are 20 cells for every meter

Afterwards, the obstacles were set up and the robot was placed at the starting position. The goal is set and the sensors gather the initial data, allowing the RRT* path-planning algorithm to generate the initial reference path. This ensures that the same path is used and saves the number of times needed to test the power consumption prediction. By focusing on the comparison between when the power consumption prediction model is activated, this initial reference path is used for their respective obstacle configurations.

4.3.1 Initial Maps

Figs 4.9 to 4.13 show the maps of the initial path planned by the RRT* algorithm, with the following goal positions and the amount of time taken for the RRT* algorithm to find the optimal reference path in the first sitting depicted in table 4.1.

Table 4.1 Goal positions for each obstacle configuration

Obstacle Configuration	**Goal (x [m], y [m])**	**Time to find reference path [s]**
1	(1.8, 1.8)	472.97
2	(1.8, 1.8)	470.07
3	(1.5, 2.0)	524.63
4	(2.0, 0.5)	550.59
5	(2.0, 2.0)	524.58

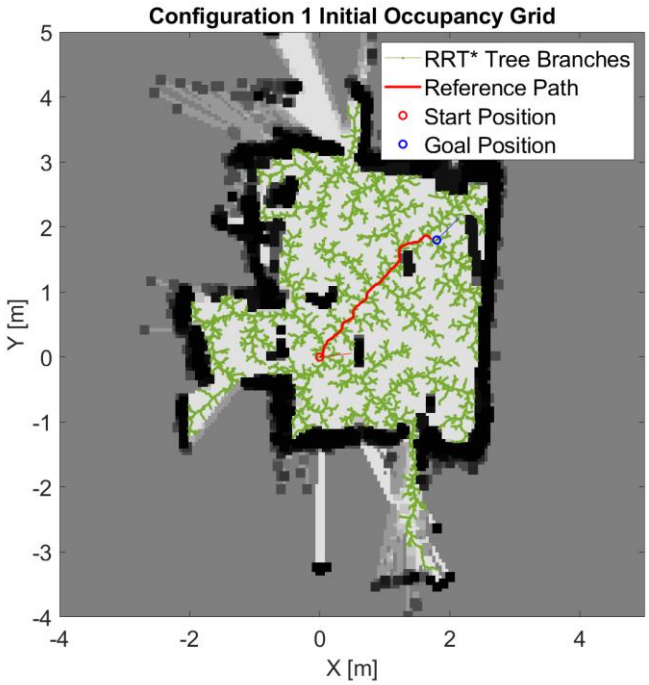


Fig 4.9. Reference path for obstacle configuration one

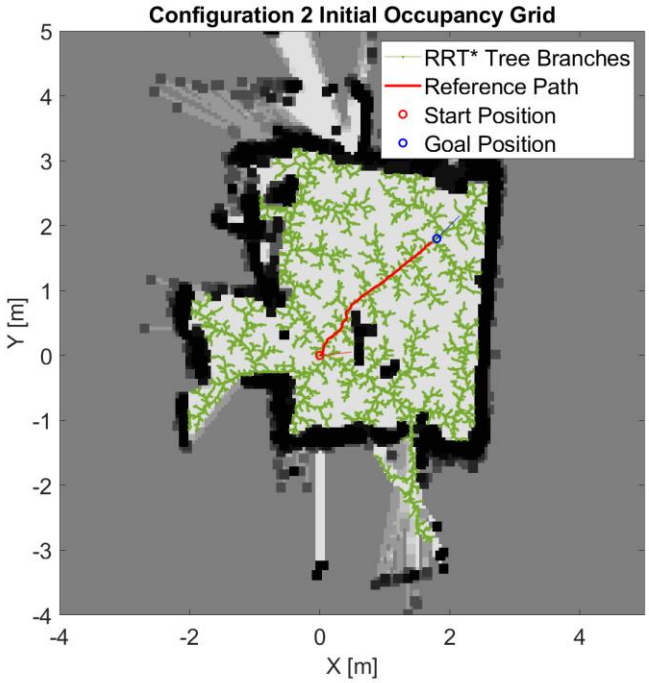


Fig 4.10. Reference path for obstacle configuration two

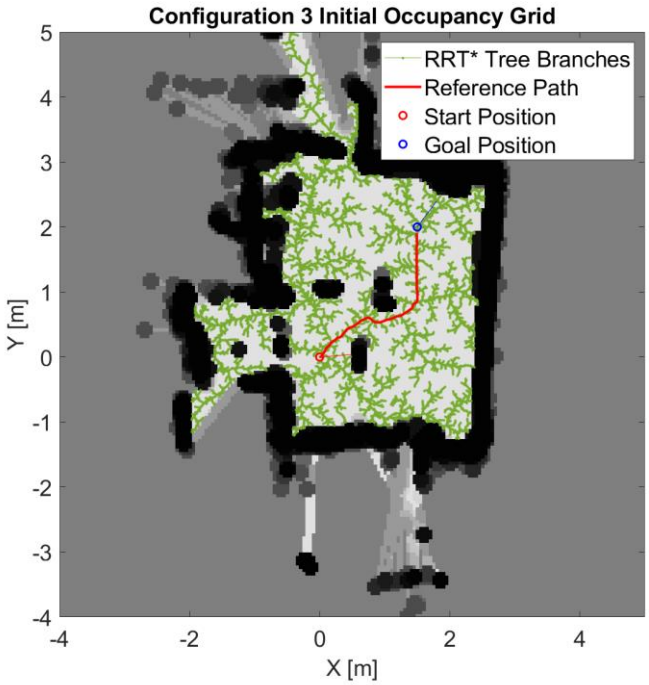


Fig 4.11. Reference path for obstacle configuration three

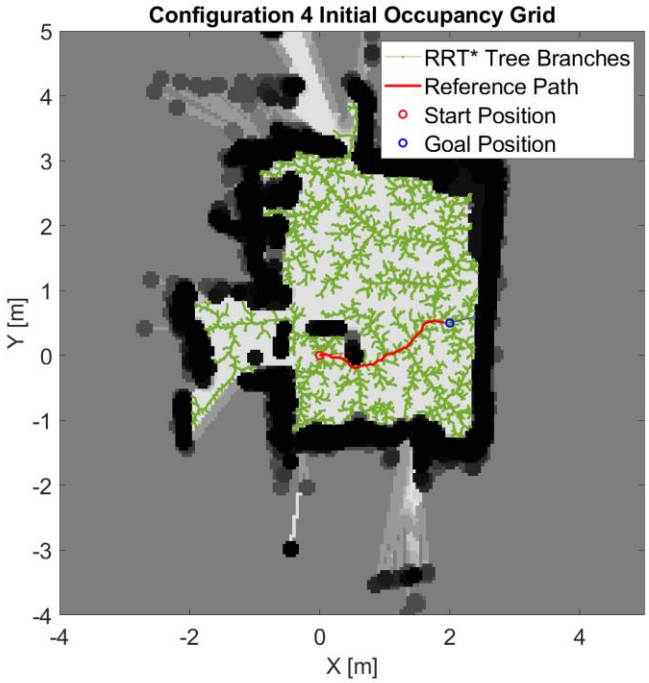


Fig 4.12. Reference path for obstacle configuration four

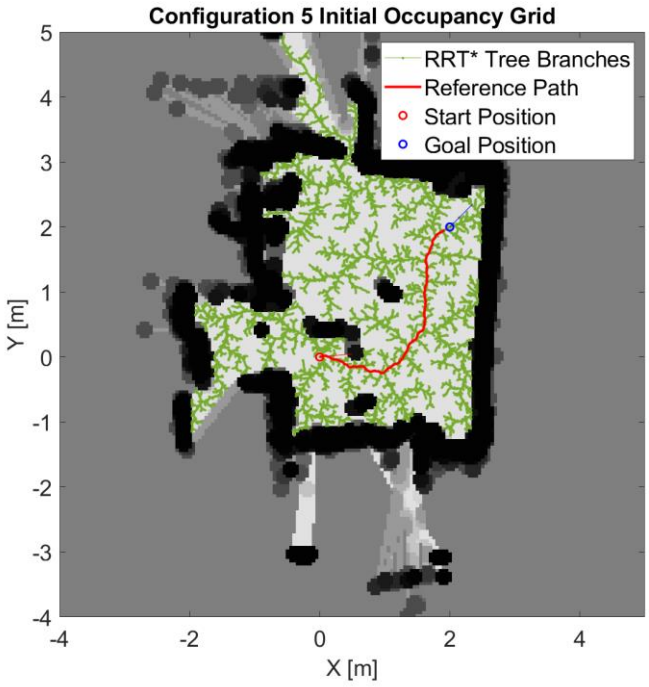


Fig 4.13. Reference path for obstacle configuration five

When the live images are compared to what the robot sees at the start of the run, it is evident that the robot is unable to detect all the obstacles, due to some being hidden behind other obstacles at the initial stage. Therefore, when the robot moves to an area where it can see the hidden obstacle and 1) the obstacle is shown to be in the way of the pre-planned path or 2) the occupancy map shows that the robot is still at risk of hitting the obstacle, the potential field would kick in to avoid a collision.

For the obstacles themselves on the occupancy grid maps, once they are detected by the lidar and placed on the map, the algorithm will “inflate” them. This means that to the algorithm, the object will seem bigger than it is in reality. This is established from the very initial reference path creation stage so that the path would avoid the obstacle without grazing it in the real environment. Many path planning algorithms provide solutions that

have the robot move as close to the obstacles as possible. However, this may result in the robot either hitting the obstacle or scraping along the side of it. This also makes working with multiple sizes of robots inefficient, as it would require measuring the robot, determining the centre, and readjusting the robot model for each time the robot needs to be adjusted. This is why modifying the inflation ratio of the obstacle is useful, since it is only one variable that needs to be adjusted. A larger inflation at the very beginning (when the initial map is created) is beneficial, as it gives the starting path a larger threshold in which to avoid the obstacle. Of course, if the value is too large, then the occupied space on the map may be filled in unnecessary locations, covering up potential paths that the robot could have followed. The inflation is based on a radius value provided by the user, taking the centre point of the obstacle and generating from it a radius, which converts all grids that touch the radius from free space to occupied space based on the resolution of the map, as shown in figure 5.14. In the algorithm for RRT*, the obstacles for the map were inflated by 0.055 m.

Since the local obstacle avoidance portion of the code results in a wider avoidance radius, subsequent iterations of the robot's movement have a smaller inflation radius (0.01 m), as it only needs to check if the obstacle would be in the way of the reference path.

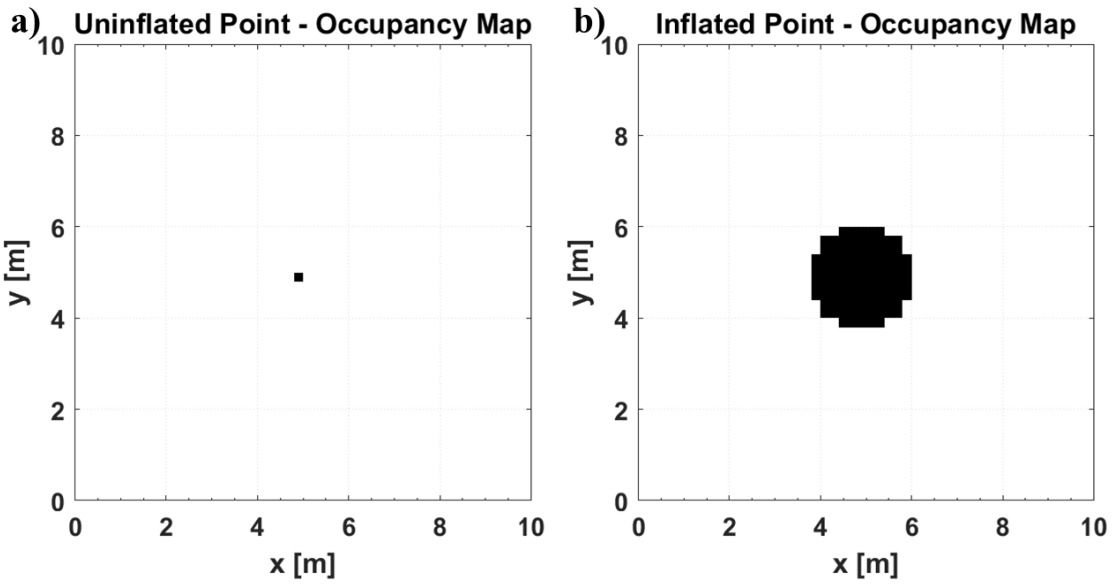


Fig 4.14 (a) Uninflated obstacle point, (b) Inflated obstacle point

With the initial occupancy maps and reference path generated, testing can begin for the computational time comparison for obstacle avoidance between RRT* algorithm recalculation and potential field cost function, as well as comparison to the effectiveness of power consumption prediction. For all experiments, the time taken to compute the initial reference time was not included in the final result because the reference path was the same for all of the experiments per obstacle configuration.

4.4 Obstacle Avoidance

The first test was to check the run time between recalculating the reference path with the RRT* algorithm versus using potential field cost to avoid the obstacles. For both types of movement, the recalculation and the activation of the potential field cost only occurs under

two conditions. The first condition is that an obstacle is in the way of the reference path. The second condition is that the closest obstacle point is less than or equal to 0.45 m. For the recalculation method, if these conditions are met, the robot will stop moving until the new reference path is generated. Should the robot continue to move, by the time a new reference path is generated for the MPC, the robot would be in a new spot unrepresentative of its mapped location. For the potential field cost function method, the robot continuously shifts the sample time at which it is instructed to move. For example, if the sample time is written to be 1.8 s, then the robot would continuously move for the given time while simultaneously performing control action calculations from the MPC. For these tests, power consumption was not considered as there was a greater focus on computational time.

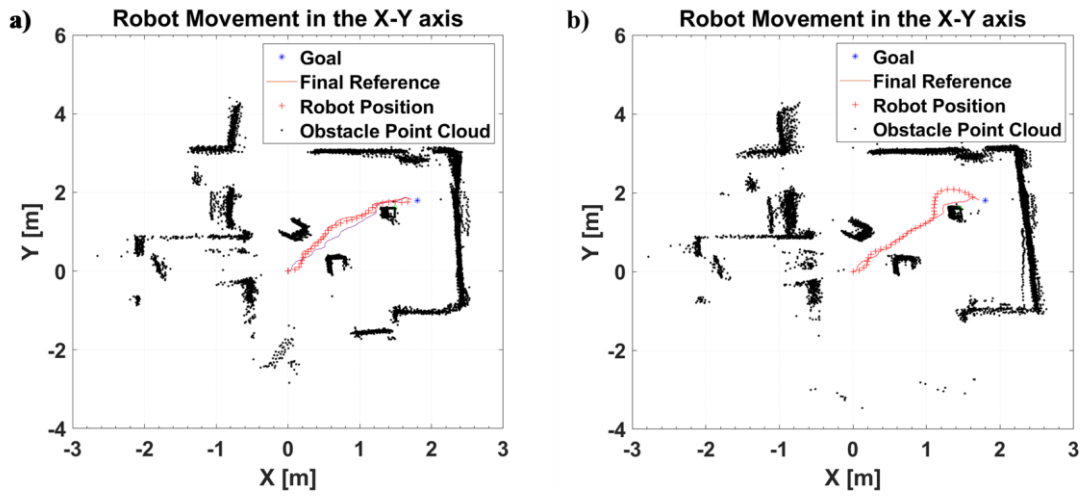


Fig 4.15. Configuration one, final robot paths for (a) Obstacle avoidance through recalculation, (b) Obstacle avoidance through potential field cost function

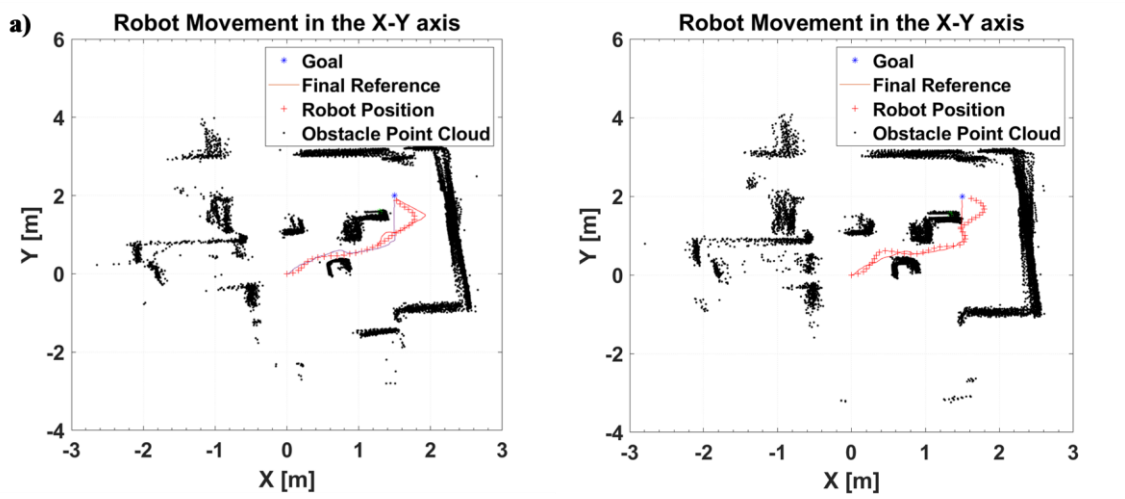


Fig 4.16. Configuration three, final robot paths for (a) Obstacle avoidance through recalculation, (b) Obstacle avoidance through potential field cost function

Table 4.2 Computation time for global recalculation vs. local obstacle avoidance

Obstacle Configuration	**Recalculation Time [s]**	**Potential Field Avoidance Time [s]**
1	712.14	70.60
3	742.71	73.1

From the results shown, it can be seen that the method utilizing RRT* recalculation takes a greater amount of time compared to the potential field cost function method in the face of unknown or larger than expected obstacles. This is inefficient as transportation of the robot is time-constrained. Additionally, when the robot ceases moving to calculate the new reference path, it still consumes power as the other components of the robot such as the SBC, Lidar, Current Sensor, and MCU are still running despite the robot's lack of motion. If power consumption were to be considered during this test, then an idle power

consumption calculation would be required, which is unnecessary for the potential field cost function method as it allows for continuous movement. Thus, the potential field cost function method for obstacle avoidance was determined to be more time-efficient than the recalculation method.

Obstacle avoidance using only potential field cost was another option that was considered. However, upon testing the runs, despite being fast while on the move, the initial reference path is not as robust compared to using a global path planner. Since the initial path is simply a straight line from start to goal and doesn't consider obstacle avoidance at the beginning, it can result in abnormally large curvatures of movement for the NMPC.

4.5 Power Consumption

To perform a comparison between the no power consumption scenario and the power consumption scenario, the same reference path for each respective obstacle was used for both. This is done for each obstacle configuration several times. A mean calculation was used to determine the average time and power consumption for each scenario.

Table 4.3 Power vs. no power average of runs comparison

	No Power		**Power**	
Configuration	**Time Travelled [s]**	**Energy Consumption [J]**	**Time Travelled [s]**	**Energy Consumption [J]**
1	75.3	515.7396	71.8	490.8996
2	68.7	471.2364	65.5	442.143
3	73.1	494.1198	68.9	467.055
4	74.3	512.7948	72.4	488.7918
5	76.7	505.242	73.67	483.786

As seen in table 4.3, the mean of the results for each obstacle configuration shows that when the power consumption prediction cost is activated, not only is the power usage conserved but the robot also travels shorter times. Therefore, for an average of a few runs for each obstacle configuration, the robot shows an average improvement of 4% for time and 5% for power, based on the complexity of the robot's final movement

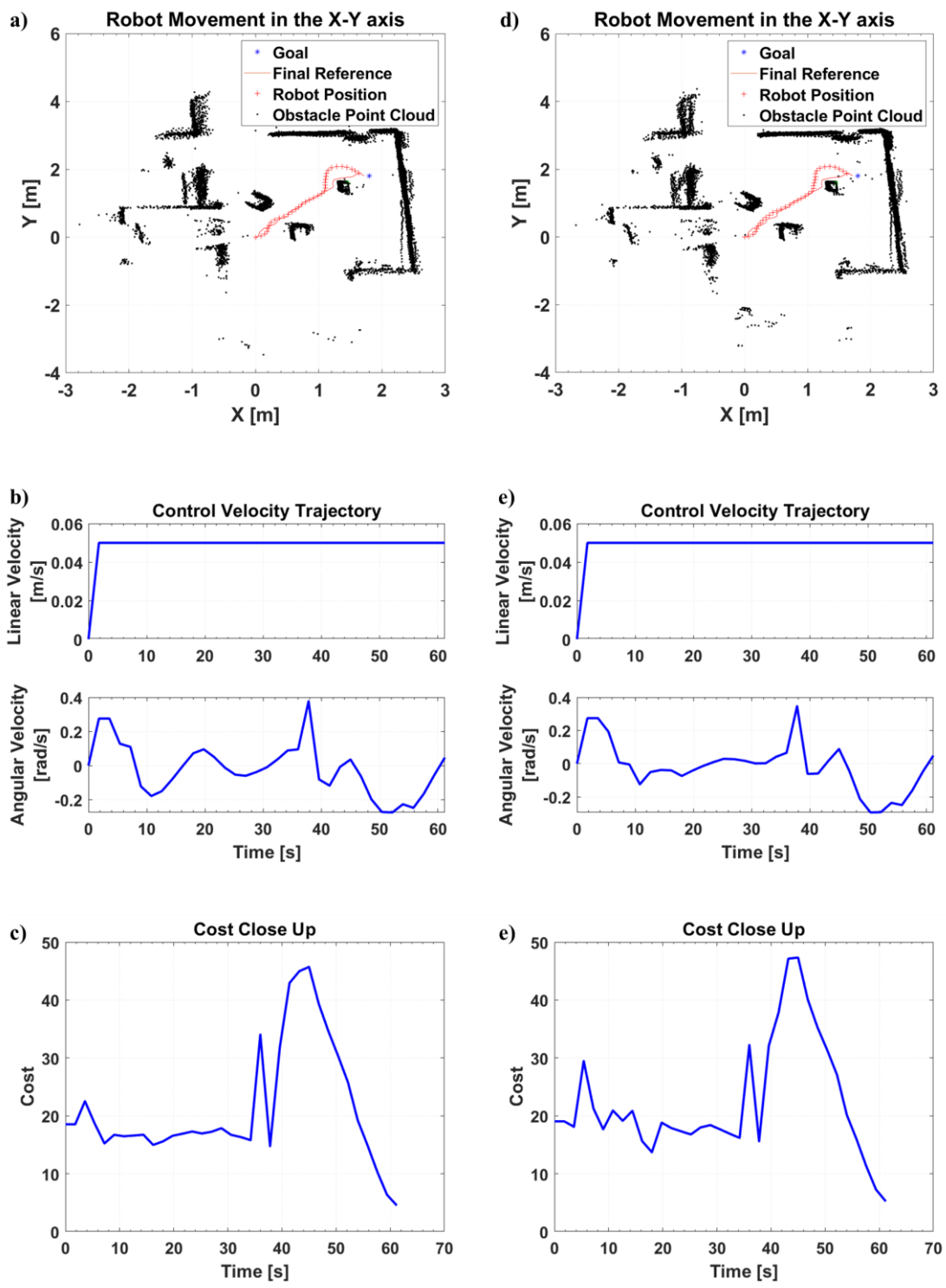


Fig 4.17. Position, velocity and cost plots of configuration one (a - c) Without the power consumption, (d - f) With power consumption

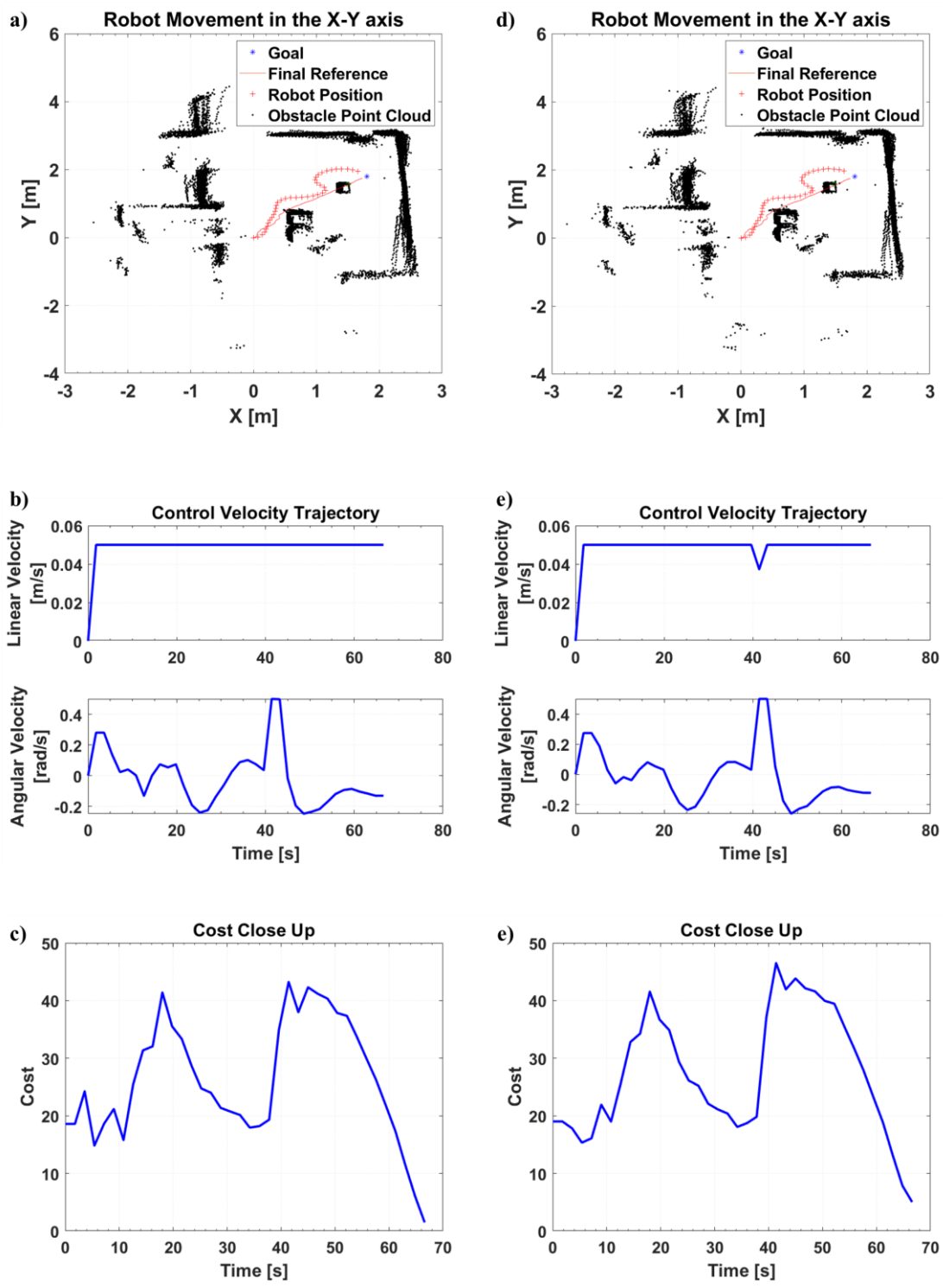


Fig 4.18. Position, velocity and cost plots of configuration two (a - c) Without the power consumption, (d - f) With power consumption

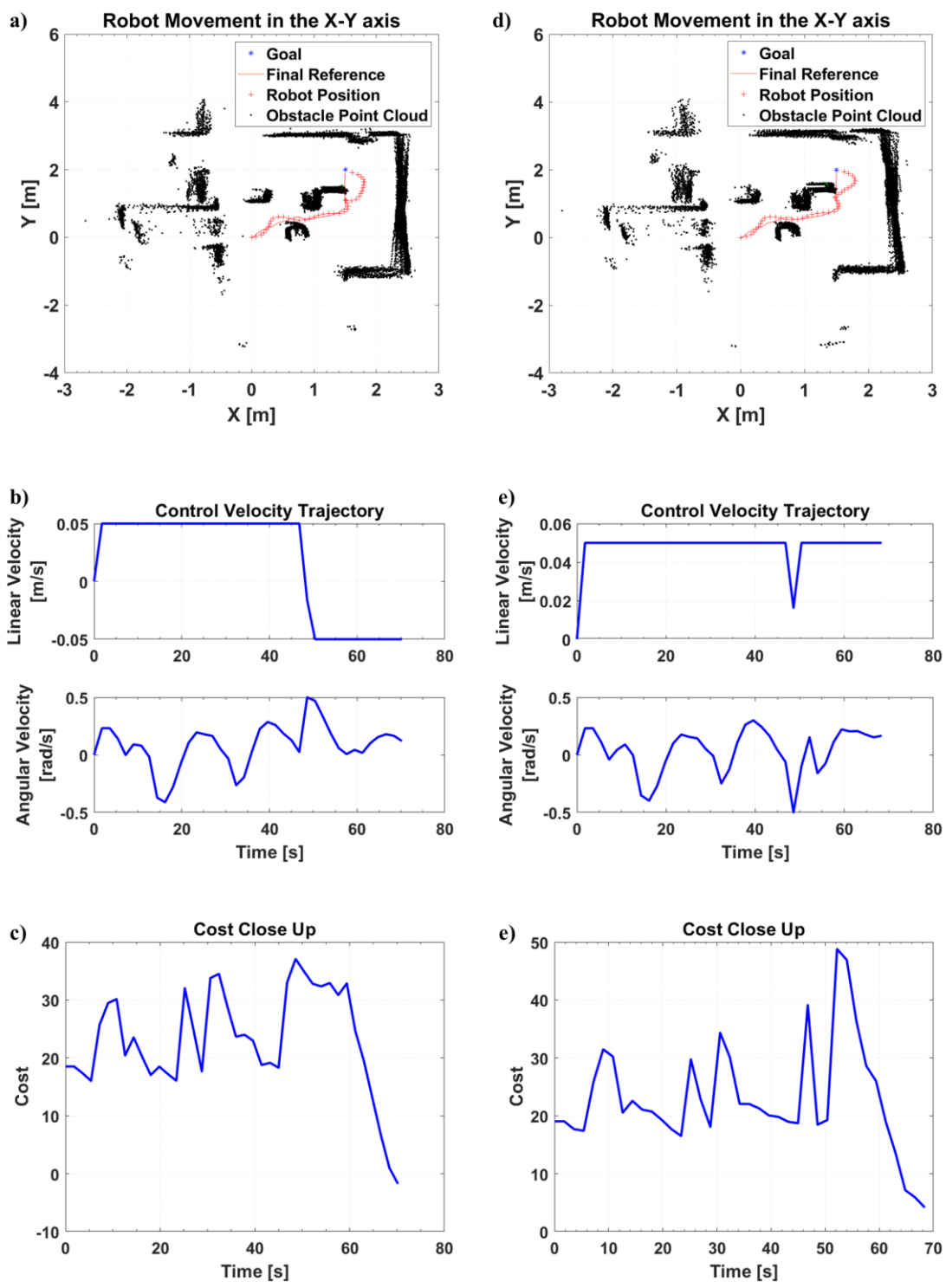


Fig 4.19. Position, velocity and cost plots of configuration three (a - c) Without the power consumption, (d - f) With power consumption

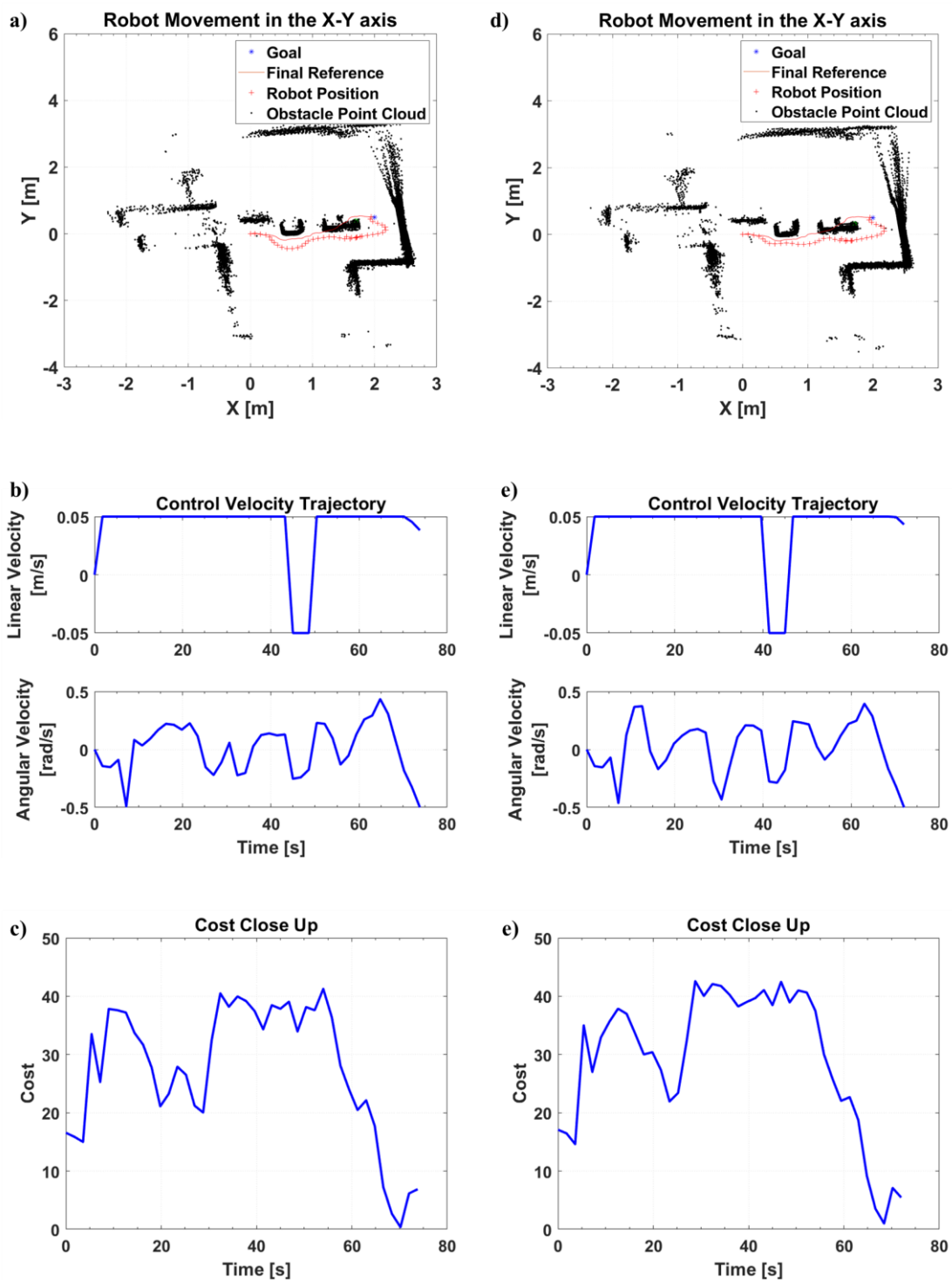


Fig 4.20. Position, velocity and cost plots of configuration four (a - c) Without the power consumption, (d - f) With power consumption

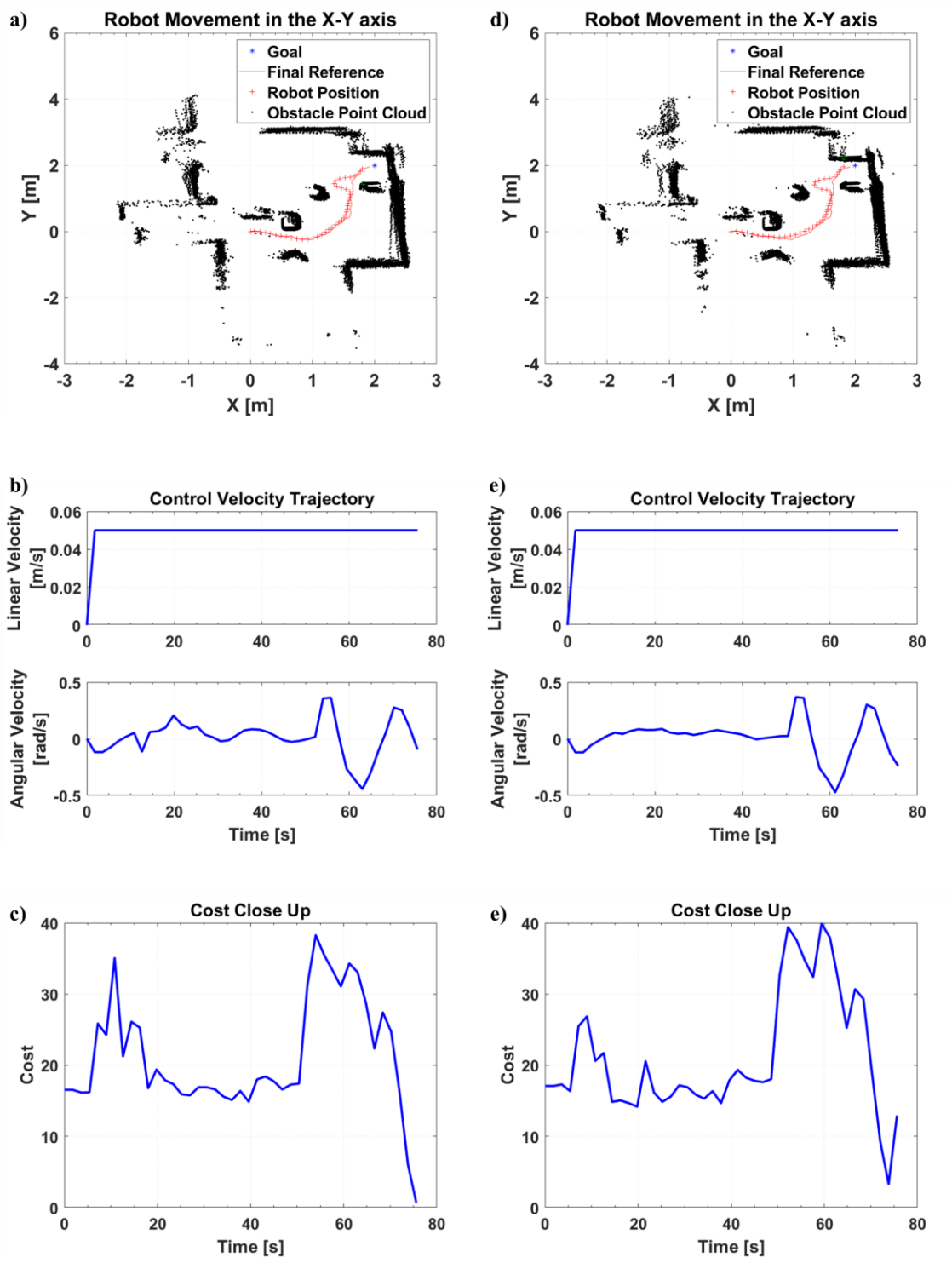


Fig 4.21. Position, velocity and cost plots of configuration five (a - c) Without the power consumption, (d - f) With power consumption

It is understood that, in general, the cost for the NMPC is expected to be constantly decreasing. However, due to the method the cost function uses for calculations (wherein the set goal and obstacle avoidance are more locally generated), the cost does not consistently decrease for each iteration over the entirety of a run unless the robot's local goal is the final goal. However, within the optimization code itself, the NMPC looks for the lowest cost between all the feasible control actions available. Therefore, while the cost does not seem to be decreasing in a global sense, it does so locally, since the local goal is constantly changing. Additionally, the obstacle avoidance is only activated under the previously mentioned conditions (obstacle intersecting reference path, closest point of obstacle being 0,45 m away). On every other occasion, the obstacle avoidance cost would be 0. This means that when the obstacle fulfils these conditions, it is sensible that the cost would suddenly spike, as it can be considered as a sudden appearance if the obstacle would have suddenly been from greater than 0.45 m away to less than 0.45 m away.

4.6 Summary

This chapter examined the performance of power consumption prediction by comparing the robot's movement between when there is and is not power consumption prediction present. First, an occupancy grid map of the empty experimentation environment was generated by moving the robot around the room. Next, 5 different obstacle configurations and goal paths were decided. These obstacle configurations had a variety of different sizes and shapes. An initial reference path was developed for each of these configurations to guarantee a consistent reference path for the configuration, removing the time needed for

the robot to recalculate the reference path every time it underwent a test. The time it took to run the robot's obstacle avoidance using reference path recalculation and the potential field cost method was evaluated, clearly concluding that the potential field cost method was faster by about 90%. Having determined that the potential field cost method was more efficient time-wise, the power consumption prediction cost based on the robot's movement was calculated. The robot was tested a few times for each obstacle configuration and for both power and no power consumption cost scenarios. The results of each of the scenarios were averaged out, showing that the power consumption cost was capable of reducing the amount of power consumed (based on the movement) by about 5% average (if all the power consumption difference errors are combined), while the time travelled was also reduced by about 4%. This shows that the proposed algorithm can successfully reduce the power consumed.

Chapter 5. Conclusions and Future Works

5.1 Conclusion

In this study, an NMPC based control strategy was proposed for a differential drive mobile robot. The main objective of this strategy was to use real motion data to develop a power consumption prediction model and a navigational method utilizing simultaneous localization and mapping and a path-planning algorithm to conserve computational time while traversing a dynamic environment.

Chapter 2 discusses some of the mobile robots in existence, their capabilities and limitations, and an overview of existing path-planning algorithms. Afterwards, the literature on the power analysis of mobile robots was discussed and analyzed. The research gaps were highlighted which showcased the need for a different strategy.

Chapter 3 shows the model that would be used for NMPC of the differential-drive robot. The problem is formulated as a multi-objective optimization problem for the NMPC control strategy. It determines that objectives are to minimize the power consumption, the local goal the robot must travel and avoid the obstacles. The problem solution utilizes a TurtleBot3 that works with Matlab, using a ROS communication method to control the robot. The power consumption minimization is solved by developing a power consumption

prediction model expression through the use of machine learning software that uses evolutionary search algorithms to develop the model with a 10% prediction error.

Chapter 4 shows the results of the physical robot traversing an environment with 5 different obstacle configurations, where some obstacles were hidden behind others for a more dynamic scenario. It also compares the computational time of using global reference path recalculation versus potential field cost for obstacle avoidance and concludes that the potential field cost was notably faster than using recalculation. Afterwards, the power consumptions of the robot in motion were compared, where it was shown that there was an improvement of 5% for the combined average of all configurations. The time it took for the robot to traverse from start to goal also decreased by a combined average of 4% for all configurations. This demonstrates that the algorithm is effective, though there is still room for improvement.

5.2 Future Works

Since this method was implemented on a smaller robot, there is potential for future implementation on larger systems, such as larger 3-wheeled robots with more applicability than a smaller robot which may also allow for the algorithm to be operated on the onboard computer as opposed to a host computer for further efficient calculations. This methodology can also be implemented on different configurations of wheeled robots, such as 4-wheeled robots with a different set of dynamics compared to 3-wheels. With a larger

system, battery management can be added to better control how power is assigned to various components in future works. Furthermore, the dynamic of the environment can be further increased by the use of moving obstacles, which were not available for this research due to the limitations of the testing facility. A larger environment or an outdoor environment can also be considered in future works to test the effects of different ground textures that might affect the wheels of the robot, in addition to the effects of different weather conditions.

Currently, many of the tuning for values such as the weight, NMPC parameters as well as other parameters of the cost function is done manually and checked through trial and error. This results in limited optimality of the variables which can be autonomized to further optimize the tuning. Another consideration that can be looked into is generating an overall data-driven cost function that incorporates all the objectives in one model to reduce further simplify the system. The stability of the NMPC control algorithm can also be further looked into as it is known that the receding horizon approach under constraints may run the risk of unstable behaviour. Hence why there is value in employing a stability theorem for the NMPC control algorithm.

References

[1] M. A. Kamarul Bahrin, M. F. Othman, N. H. Nor Azli, and M. F. Talib, "INDUSTRY 4.0: A REVIEW ON INDUSTRIAL AUTOMATION AND ROBOTIC," *Jurnal Teknologi*, vol. 78, 6-13, 2016, doi: 10.11113/jt.v78.9285.

[2] J. Dixon, B. Hong, and L. Wu, "The Employment Consequences of Robots: Firm-Level Evidence," *SSRN Journal*, 2019, doi: 10.2139/ssrn.3422581.

[3] B. Preising, T. C. Hsia, and B. Mittelstadt, "A literature review: robots in medicine," *IEEE engineering in medicine and biology magazine : the quarterly magazine of the Engineering in Medicine & Biology Society*, vol. 10, no. 2, pp. 13–22, 1991, doi: 10.1109/51.82001.

[4] J. Broekens, M. Heerink, and H. Rosendal, "Assistive social robots in elderly care: a review," *Gerontechnology*, vol. 8, no. 2, 2009, doi: 10.4017/gt.2009.08.02.002.00.

[5] O. Mubin, C. J. Stevens, S. Shahid, A. A. Mahmud, and J.-J. Dong, "A REVIEW OF THE APPLICABILITY OF ROBOTS IN EDUCATION," *Technology for Education and Learning*, vol. 1, no. 1, 2013, doi: 10.2316/Journal.209.2013.1.209-0015.

[6] aibo, *aibo.* [Online]. Available: https://us.aibo.com/feature/feature1.html (accessed: Mar. 12 2021).

[7] amazon.jobs, *Amazon Robotics.* [Online]. Available: https://www.amazon.jobs/en/teams/amazon-robotics (accessed: Mar. 12 2021).

[8] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, 172988141983959, 2019, doi: 10.1177/1729881419839596.

[9] J. S. Artal, J. A. Dominguez, and J. Caraballo, "Autonomous mobile robot with hybrid PEMfuel-cell and ultracapacitors energy system. Dedalo 2.0," *REPQJ*, pp. 1795–1800, 2012, doi: 10.24084/repqj10.838.

[10] J. L. Sullivan and L. Gaines, "A Review of Battery Life-Cycle Analysis. State of Knowledge and Critical Needs," 2010.

[11] S. Swanborn and I. Malavolta, "Energy efficiency in robotics software," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering Workshops*, Virtual Event Australia, 09212020, pp. 144–151.

[12] X. Hu, L. Chen, B. Tang, D. Cao, and H. He, "Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles," *Mechanical Systems and Signal Processing*, vol. 100, pp. 482–500, 2018, doi: 10.1016/j.ymssp.2017.07.019.

[13] J. Lee, S. Han, and J. Lee, "Decoupled Dynamic Control for Pitch and Roll Axes of the Unicycle Robot," *IEEE Trans. Ind. Electron.*, vol. 60, no. 9, pp. 3814–3822, 2013, doi: 10.1109/TIE.2012.2208431.

[14] T. P. Nascimento, C. E. T. Dórea, and L. M. G. Gonçalves, "Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots,"

International Journal of Advanced Robotic Systems, vol. 15, no. 1, 172988141876046, 2018, doi: 10.1177/1729881418760461.
[15] R. P. M. Chan, K. A. Stol, and C. R. Halkyard, "Review of modelling and control of two-wheeled robots," *Annual Reviews in Control*, vol. 37, no. 1, pp. 89–103, 2013, doi: 10.1016/j.arcontrol.2013.03.004.
[16] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*. Cambridge: Cambridge University Press, 2010.
[17] Mars.nasa.gov, *Learn About the Rover.* [Online]. Available: https://mars.nasa.gov/mars2020/spacecraft/rover/ (accessed: Mar. 9 2021).
[18] Q. Quan, *Introduction to Multicopter Design and Control*. Singapore: Springer Singapore, 2017.
[19] X. T. P. She, X. Lin, and H. Lang, "A Data-Driven Power Consumption Model for Electric UAVs," in *2020 American Control Conference (ACC)*, Denver, CO, USA, 72020, pp. 4957–4962.
[20] C.-M. Tseng, C.-K. Chau, K. Elbassioni, and M. Khonji, "Autonomous Recharging and Flight Mission Planning for Battery-operated Autonomous Drones," Mar. 2017. [Online]. Available: http://arxiv.org/pdf/1703.10049v2
[21] *ROBOTIS e-Manual.* [Online]. Available: https://emanual.robotis.com/docs/en/parts/controller/opencr10/ (accessed: Mar. 13 2021).
[22] M. Doyle and J. Newman, "The use of mathematical modeling in the design of lithium/polymer battery systems," *Electrochimica Acta*, vol. 40, 13-14, pp. 2191–2196, 1995, doi: 10.1016/0013-4686(95)00162-8.
[23] J. M. Amanor-Boadu and A. Guiseppi-Elie, "Improved Performance of Li-ion Polymer Batteries Through Improved Pulse Charging Algorithm," *Applied Sciences*, vol. 10, no. 3, p. 895, 2020, doi: 10.3390/app10030895.
[24] B. Scrosati and J. Garche, "Lithium batteries: Status, prospects and future," *Journal of Power Sources*, vol. 195, no. 9, pp. 2419–2430, 2010, doi: 10.1016/j.jpowsour.2009.11.048.
[25] C. R. Birkl, M. R. Roberts, E. McTurk, P. G. Bruce, and D. A. Howey, "Degradation diagnostics for lithium ion cells," *Journal of Power Sources*, vol. 341, pp. 373–386, 2017, doi: 10.1016/j.jpowsour.2016.12.011.
[26] B. Behroozpour, P. A. M. Sandborn, M. C. Wu, and B. E. Boser, "Lidar System Architectures and Circuits," *IEEE Commun. Mag.*, vol. 55, no. 10, pp. 135–142, 2017, doi: 10.1109/MCOM.2017.1700030.
[27] J. Hecht, "Lidar for Self-Driving Cars," *Optics & Photonics News*, vol. 29, no. 1, p. 26, 2018, doi: 10.1364/OPN.29.1.000026.
[28] Allegro, *ACS712GenMkt.* [Online]. Available: https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf (accessed: Mar. 13 2021).
[29] P. Yap, "Grid-Based Path-Finding," in *Lecture Notes in Computer Science, Advances in Artificial Intelligence*, G. Goos, J. Hartmanis, J. van Leeuwen, R. Cohen, and B. Spencer, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 44–55.

[30] C. Saranya, K. K. Rao, M. Unnikrishnan, V. Brinda, V. R. Lalithambika, and M. V. Dhekane, “Real Time Evaluation of Grid Based Path Planning Algorithms: A comparative study,” *IFAC Proceedings Volumes*, vol. 47, no. 1, pp. 766–772, 2014, doi: 10.3182/20140313-3-IN-3024.00050.

[31] L. Zuo, Q. Guo, X. Xu, and H. Fu, “A hierarchical path planning approach based on A $*$ and least-squares policy iteration for mobile robots,” *Neurocomputing*, vol. 170, pp. 257–266, 2015, doi: 10.1016/j.neucom.2014.09.092.

[32] H. Myint, “Development of Robot Navigation System with Collision Free Path Planning Algorithm,” *MLR*, vol. 3, no. 3, p. 60, 2018, doi: 10.11648/j.mlr.20180303.12.

[33] N. Tran, I. Prodan, E. I. Grøtli, and L. Lefèvre, “Potential-field constructions in an MPC framework: application for safe navigation in a variable coastal environment,” *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 307–312, 2018, doi: 10.1016/j.ifacol.2018.11.049.

[34] Y. Rasekhipour, A. Khajepour, S.-K. Chen, and B. Litkouhi, “A Potential Field-Based Model Predictive Path-Planning Controller for Autonomous Road Vehicles,” *IEEE Trans. Intell. Transport. Syst.*, vol. 18, no. 5, pp. 1255–1267, 2017, doi: 10.1109/TITS.2016.2604240.

[35] H. Bing, L. Gang, G. Jiang, W. Hong, N. Nan, and L. Yan, “A route planning method based on improved artificial potential field algorithm,” in *2011 IEEE 3rd International Conference on Communication Software and Networks*, Xi'an, China, 052011, pp. 550–554.

[36] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011, doi: 10.1177/0278364911406761.

[37] F. Peralta, M. Arzamendia, D. Gregor, D. G. Reina, and S. Toral, “A Comparison of Local Path Planning Techniques of Autonomous Surface Vehicles for Monitoring Applications: The Ypacarai Lake Case-study,” *Sensors (Basel, Switzerland)*, vol. 20, no. 5, 2020, doi: 10.3390/s20051488.

[38] H. La Devin Connell, *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC): Banff Center, Banff, Canada, October 5-8, 2017*. Piscataway, NJ: IEEE, 2017. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=8114675

[39] W. Xinyu, L. Xiaojuan, G. Yong, S. Jiadong, and W. Rui, “Bidirectional Potential Guided RRT* for Motion Planning,” *IEEE Access*, vol. 7, pp. 95046–95057, 2019, doi: 10.1109/ACCESS.2019.2928846.

[40] J. Wang, Ed., *2010 International Conference on Computer Design and Applications: ICCDA 2010 ; Qinhuangdao, China, 25 - 27 June 2010*. Piscataway, NJ: IEEE, 2010.

[41] S.-H. Chia, K.-L. Su, J.-H. Guo, and C.-Y. Chung, “Ant Colony System Based Mobile Robot Path Planning,” in *2010 Fourth International Conference on Genetic and Evolutionary Computing*, Shenzhen, 122010, pp. 210–213.

[42] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, WA, Australia, 1995, pp. 1942–1948.
[43] B. B. V. L. Deepak, D. R. Parhi, and B. M. V. A. Raju, “Advance Particle Swarm Optimization-Based Navigational Controller For Mobile Robot,” *Arab J Sci Eng*, vol. 39, no. 8, pp. 6477–6487, 2014, doi: 10.1007/s13369-014-1154-z.
[44] R. A. Rutenbar, “Simulated annealing algorithms: an overview,” *IEEE Circuits Devices Mag.*, vol. 5, no. 1, pp. 19–26, 1989, doi: 10.1109/101.17235.
[45] H. Miao and Y.-C. Tian, “Dynamic robot path planning using an enhanced simulated annealing approach,” *Applied Mathematics and Computation*, vol. 222, pp. 420–437, 2013, doi: 10.1016/j.amc.2013.07.022.
[46] E. Masehian and M. R. Amin-Naseri, “A Tabu Search-based Approach for Online Motion Planning,” in *2006 IEEE International Conference on Industrial Technology*, Mumbai, India, 122006, pp. 2756–2761.
[47] M. Samadi and M. F. Othman, “Global Path Planning for Autonomous Mobile Robot Using Genetic Algorithm,” in *2013 International Conference on Signal-Image Technology & Internet-Based Systems*, Kyoto, Japan, 122013, pp. 726–730.
[48] D. Ferguson, M. Likhachev, and A. Stentz, “A Guide to Heuristic-based Path Planning,” 2005.
[49] A. Abdilla, A. Richards, and S. Burrow, “Power and endurance modelling of battery-powered rotorcraft,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 92015, pp. 675–680.
[50] P. Sattayasoonthorn and J. Suthakorn, “Battery management for rescue robot operation,” in *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Qingdao, China, 122016, pp. 1227–1232.
[51] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. Lee, “A case study of mobile robot's energy consumption and conservation techniques,” in *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, Seatle, WA, USA, 2005, pp. 492–497.
[52] L. Xie, C. Henkel, K. Stol, and W. Xu, “Power-minimization and energy-reduction autonomous navigation of an omnidirectional Mecanum robot via the dynamic window approach local trajectory planning,” *International Journal of Advanced Robotic Systems*, vol. 15, no. 1, 172988141875456, 2018, doi: 10.1177/1729881418754563.
[53] C. de Cauwer, W. Verbeke, T. Coosemans, S. Faid, and J. van Mierlo, “A Data-Driven Method for Energy Consumption Prediction and Energy-Efficient Routing of Electric Vehicles in Real-World Conditions,” *Energies*, vol. 10, no. 5, p. 608, 2017, doi: 10.3390/en10050608.
[54] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. Lee, “A case study of mobile robot's energy consumption and conservation techniques,” in *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, Seatle, WA, USA, 2005, pp. 492–497.

[55] R. Dubčáková, “Eureqa: software review,” *Genet Program Evolvable Mach,* vol. 12, no. 2, pp. 173–178, 2011, doi: 10.1007/s10710-010-9124-z.
[56] D. R. Stoutemyer, “Can the Eureqa symbolic regression program, computer algebra and numerical analysis help each other?,” Mar. 2012. [Online]. Available: http://arxiv.org/pdf/1203.1023v1
[57] J. Liu, A. Saxena, K. Goebel, B. Saha, and W. Wang, “An Adaptive Recurrent Neural Network for Remaining Useful Life Prediction of Lithium-ion Batteries,” 2010.