University of Hertfordshire UH

FINAL YEAR PROJECT REPORT

MEng

Name: Amr Elshenawy

Supervisor: Dr. Zoe Jeffrey

Autonomous Self-Navigating Robot

School of Engineering and Technology

2018/2019

Bachelor of Engineering Degree with Honours in
Electrical and Electronic Engineering

MEng Individual Project Report
School of Engineering and Technology
University of Hertfordshire

Autonomous Self-navigating Robot

Report by - ***Amr Elshenawy***

Supervisor - ***Dr. Zoe Jeffrey***

Date - ***29-3-2019***

DECLARATION STATEMENT

I certify that the work submitted is my own and that any material derived or quoted from the published or unpublished work of other persons has been duly acknowledged (ref. UPR AS/C/6.1, Appendix I, Section 2 – Section on cheating and plagiarism)

Student Full Name: Amr Elshenawy

Student Registration Number: 14186242

Signed: Amr Elshenawy
Date: 26-2-2019

ABSTRACT

Robotics and Autonomy is a very fast-growing field that will at large affect future human life more than expected. Autonomous robots are increasing rapidly in numbers and are getting better and more sophisticated day-by-day. Significant amount of contributions and innovations have been done for the field in the past few decades.

This project, Autonomous Self-navigating Robot, is aimed at planning, designing and developing a robot that is able to help navigate itself around any environment avoiding any obstacles in the way. With the aid of several ultrasonic sensors that are used to detect objects in close proximity, the data received from the sensors will be analysed by a microcontroller that will then interpret the data and take a decision based on the outcome of the information received from the sensors. The outcome of the microcontroller is then observed in the form of physical movement in the robot body by the powering of four DC motors accordingly. Three navigation algorithms have been investigated and implemented for testing and development from selected papers. These are Binary Movement Algorithm, Wall-Follow Algorithm and Fixed Rotations Algorithm. Each of these methodologies have been tested and analysed for their advantages and drawbacks where they lacked satisfactory performance. A fourth algorithm has therefore been developed as a personal contribution which provides a combination of all previous algorithms. Variable Automated Rotations Algorithm aims and combining the positives of the previously three mentioned concepts while eliminating their drawbacks to allow for the best performance possible fit for this project.

In addition to the autonomous feature, the robot has been developed to allow external communication with nearby devices/users. This will allow for a remote manual control of the robot overriding the autonomous feature in case there is ever a need to override the microcontroller with manual control. Moreover, the robot has been programmed to self-test for faulty connections using a hardware testing algorithm. This allows the user to identify any loose or misconnected wires that might impact the performance of the robot.

This report will touch upon some of the shortcomings of this project, why certain initial goals failed to be implemented and what could be done in future work to circumvent these issues and hindrances faced during the development of the robot. Furthermore, the report will discuss further development, significant additions and improvements that could be done to allow for a much more sophisticated robot. How the functionality of this robot could be linked with Artificial Intelligence and machine learning and discussing opportunities where this robot could be taken to the next level of autonomy and efficiency.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Zoe Jeffrey for her true attention and care for my project. Her recommendations, advices, ideas and thoughts contributed to the success of this project. I'd also like to thank my parents and family for their constant supply of encouragement and motivation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

NASA	The National Aeronautics and Space Administration
DC	Direct Current
VSLAM	Vision Simultaneous Localization and Mapping
IR	Infrared
PWM	Pulse Width Modulation
PIC	Peripheral Interface Controller
PCB	Printed Circuit Board
RNG	Random Number Generator
4WD	4-Wheel Drive
BPS	Bits Per Second
IDE	Integrated Development Environment

1. INTRODUCTION AND BACKGROUND

To individuals who are not knowledgeable in the fields of Electronics or Electrical Engineering, robots and the field of Robotics in general can often seem like an extremely complicated and complex matter with lots of intricacies and challenges. One might even consider the field as complicated as "rocket science". However, this is not very true. There is a vast spectrum and varying levels of how complex robots can get. Surely, a robot that is designed to create a breeze of air at home using simple components is drastically inferior to NASA's Mars rover, Opportunity [1]. Therefore, this begs this question, what is a robot and when does one become classified as a Robot. What constitutes whether an electronic device is a robot or not.

First, Robotics is multi-disciplinary field that combines knowledge and expertise from a variety of fields in Engineering. Robotics requires concepts and implementations from the fields of Engineering and Science including mechanical engineering, electrical engineering, electronics engineering, information engineering, computer science and many more specialised fields. A definition from Lifewire states that a robot is [2] "**A machine capable of responding to its environment to automatically carry out complex or repetitive tasks with little, if any, direction from a human being**". Another definition from Robots homepage from IEEE [3] mentions that "**A robot is an autonomous machine capable of sensing its environment, carrying out computations to make decisions, and performing actions in the real world.**" By understanding and dissecting these definitions, it can be outlined what constitutes or makes up a robot. Essentially, a robot has to produce some sort of a measurable or observable output in response to an external event. The output is most commonly a mechanical output resulting in physical manipulation in the robot's environment. Robots are designed to make daily life for humans more easy, convenient, safe and less tedious. Simply put, robots are designed to serve humans to improve their quality of lives in all aspects imaginable. A common misconception is that the majority of electronic devices, especially ones that react to human input, are considered robots. This is in fact incorrect by definition. For instance, mobile phones are not considered robots mainly because they don't manipulate anything in their environment. While they do have sensors and several output measures in the form of visuals, audio etc., they are after all reacting to an input from a user rather than the phone reacting to its environment. That being said, different scientists have different definitions of what robots are based on what they define what a robot to be and their arguments. Defining a solid definition for robots would require several doctoral programmes with extensive research to pinpoint exactly what would be considered a robot and what is not. Defining what robots are is a hot-topic amongst scholars and scientists and always a topic with lots of room for debate.

The earliest forms of contributions that helped develop robots as we know them today initiated thousands of years ago. The earliest example of simple machines utilising fundamental mechanical concepts are dated back to around to the 3rd century BC. Archimedes, a Greek mathematician studied Archimedean simple machines. He investigated and discovered the mechanical advantages of levers. Archimedean simple machines are mechanical devices that alter or affect a force resulting a change in its direction or magnitude. They are known as simple mechanisms employing mechanical advantages, most notably leverage, to multiply and increase forces significantly. The six most classical machines defined by Renaissance scientists are [4] [5]:

- Lever
- Wheel and axle
- Pully
- Inclined Plane
- Wedge
- Screw

Modern robotics employ in one way or another fundamental mechanical concepts from these six simple machines. Even though Archimedes did not invent robots directly, his contributions along with other great scientists such as Sir Isaac Newton are what allowed the recent boom of development and innovation in robotics and their ability to carry out physical activities that require sheer amounts of forces and efforts no human would ever be able to carry out.

Moving forward, as humanity's knowledge and understanding of Engineering and Mathematics increased exponentially. Scientists started introducing rules and regulations of how robots should operate and what governs their actions. In 1942, science fiction author Isaac Asimov introduced three laws known as the "Three Laws of Robotics" in his anticipation of a rise in a power robotic industry [6]. During the formulation of these laws, the word "robotics" then came to be and was formed in the process. The three rules are:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Isaac Asimov has then went on to modifying and adding to the Three Laws to bring them better in line with how robots were perceived at his time and what their potential was understood to be. One of the most notable rules laws added is known as the Zeroth Law where it precedes all other previous laws and it reads:

- A robot may not harm humanity, or, by inaction, allow humanity to come to harm.

Isaac Asimov rules however have been a strong point of debates and arguments discussing the legitimacy of such rules. Their area of reach, what constitutes "harm" to humanity and realistically how implementable these rules are. In addition, ambiguities and loopholes were strong points of contention when it comes to governing these rules. In this modern time, whether these rules are still enforceable in all aspects of robotics is still widely up to debate.

Few years following Isaac Asimov's Three Laws of Robotics, one of the main milestones that truly skyrocketed robot's development took place. In 1948 and 1949, American neurophysiologist and robotician William Grey Walter in collaborations with the Burden Neurological Institute in Bristol, England, created the first electronic autonomous robot with intelligent behavior [7]. His first robot inventions, called Elmer and Elsie, where visualized as "tortoises" due to their shape and their extremely slow movement pace. Those robots were three-wheeled and were able to find their way to recharging stations when their batteries ran low. William's work inspired subsequent researchers and scientists in the field of robotics. In 1954, George Devol invented the first digitally operated and programmable robot and he named it Unimate. Devol's robot, Unimate, revolutionized the manufacturing processes and at large contributed to the boom of industrial revolution. It demonstrated how powerful these machines are at affecting the manufacturing industry in terms of efficiency, precision, power, costs-saving and safety. Unimate laid the foundations that then founded and formed modern robotics industry. Soon after his invention, Devol sold the first Unimate robot to General Motors in 1960 that revolutionized the automotive industry to unprecedented levels of productivity [8].

Figure 1-1: George Devol with his invention Unimate in 1954 [8].

With collaboration from Joseph Engelberger who is known as the Father of Robotics, George and Jospeh founded Unimation Inc. in 1961 in Connecticut to develop business in the newly created robotics industry. Following the sale of Unimate robots to General Motors, GM escalated the production rate drastically jumping ahead of all competition significantly. In 1969, GM refurbished their plant in Ohio to accommodate the installation of the new Unimate robots on their production floor. With help from Unimate, GM were able to achieve extraordinary levels of production, more than twice the rate of competitors at 110 built cars per hour [8]. With the immense efficiency seen from Unimate after GM's adoption of the ground-breaking technology, other companies followed suit as well. Companies like BMW, Mercedes Benz and Volvo started incorporating Unimate in their car production flow. Unimate is credited to be one of the most influential inventions in the past 100 years not just to industry and manufacturing, but to civilization and humanity as a whole. It has initiated an era of great efficacies and automations.

Thanks to Devol and his invention, Unimate, the field of robotics continues to grow in all aspects of human life beyond just manufacturing or automation.

1.1 Autonomous Robotic Cars and Motivation

Autonomous Robots are machines that are capable of performing tasks and instructions to a high degree of Autonomy with little to no human intervention. They have the ability to carry out repetitive tedious tasks for long periods of time without any need for rest or cooldown. The project in hand will focus on a specific field of Autonomous Robots, Autonomous Robot Cars. The concept behind Autonomous Cars is they do not require human control to steer the vehicle in a specific direction. Thanks to some crucial components that will be discussed shortly, the car has the ability to sense its environment and the world surrounding it, make a judgement of what it perceives and then reach a conclusion by making a decision. While it might sound like a ridiculous idea to have heavy machinery like a car not operated by a human in real-life streets and roads, it is expected that self-driving cars will far surpass the accuracy and precision of any human being and allow for a much safer, accident-free traffic.

According to published research data from World Health Organization, Global Health Observatory data (GHO) and the Parliament [9] [10], there were 1.25 million road traffic deaths worldwide in 2013 alone. Several million more civilians sustained serious injuries with permanent severe life-changing traumas that inhibited their ways of life forever. Road traffic accidents is the ninth leading cause of death globally and is projected to become seventh by 2030. While there have been improvements over the years, especially in developed countries, by introducing more strict traffic laws and regulations, the situation is not getting much better in the rest of the world. More prominently, developing countries and densely populated cities.

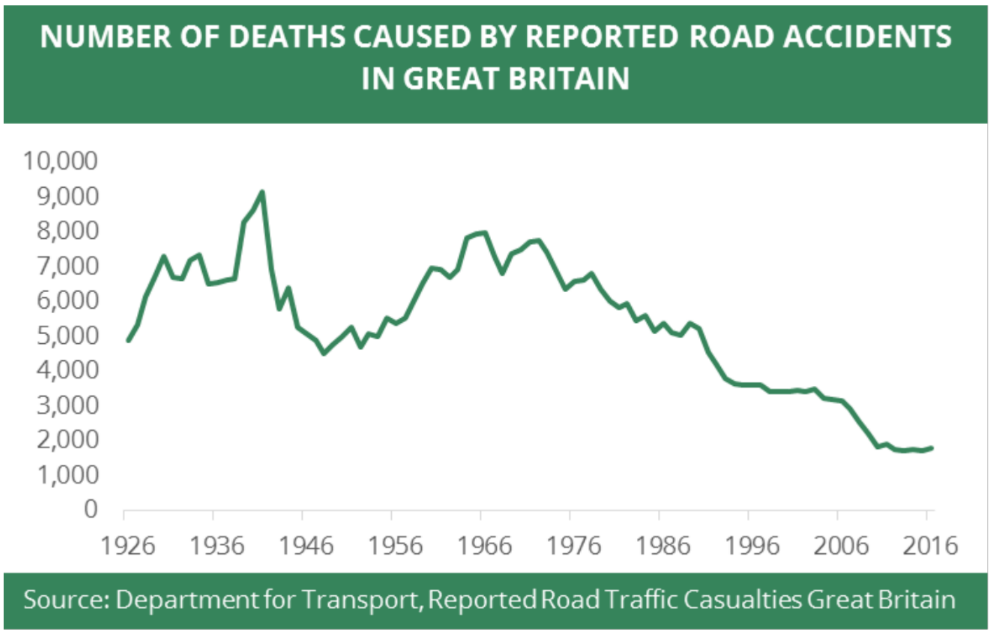


Figure 1-2: Number of Deaths by Road Accidents in UK [9]

Tesla is today's most renowned company for producing and manufacturing electric cars with varying levels of autonomy [11] [14]. It is leading the market for self-driving electric cars, promoting the overall humanity's shift away from the dependency on fossil fuels and manual control.

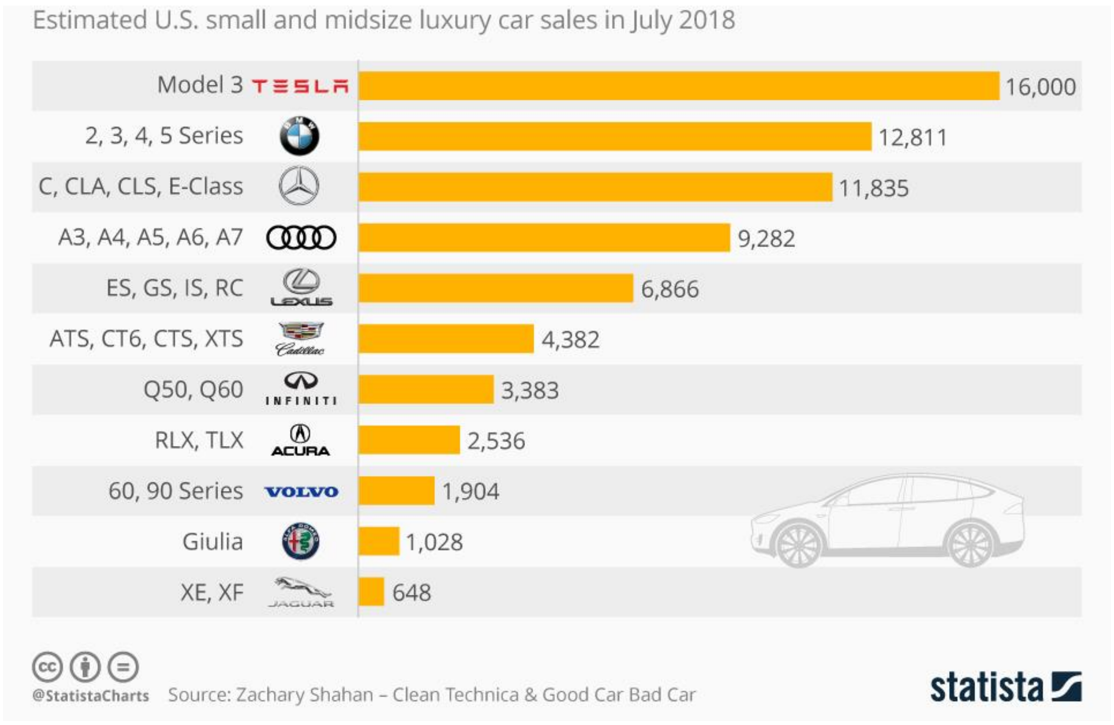


Figure 1-3: Tesla Model 3 sales against competition [11]

Their newest product Model 3 sales surpassed strong powerhouses in the industry in July 2018 including notable companies like BMW, Mercedes Benz and Audi. This demonstrates that it is only a matter of time before electric self-driving cars takeover the streets and alienating manually-driven gas cars. Tesla's founder and CEO Elon Musk has said [12] that he expects fully autonomous Tesla cars on the streets in 2019, completely overtaking human control. However, the safety and viability these cars and if they will ever take over 100% is still a question that is still yet to be answered.

Self-driving cars however are not the only focus of autonomous vehicles. There are multiple examples of fully autonomous vehicles currently incorporated deep within our society. Some of the more prominent examples are NASA's space rovers which have landed on the Moon and Mars have varying degrees autonomy. They performed functions like navigating the planet's surface, picture capturing, extracting and collecting soil samples etc. to send back valuable information back to Earth. One of the most notable rovers ever created is Mar's rover, Opportunity that roamed the Red Planet's surface for 14 years from 2004 to 2018 [1] [13]. Thanks to the ingenuity and brilliance of the engineers who worked on Opportunity, the rover has surpassed far and beyond the expectations that were first set out for it. Opportunity was anticipated to survive Mar's environment and atmosphere for a very short period of time. It has outlived its life expectancy of 92 days 55 times longer to reach 5498 days of operation!

Autonomy in Opportunity has allowed the robot to perform some tasks with independency away from human control which has proven to be very crucial to the safety of the rover during heavy dust storms and one that has allowed it to reach such a long period of operation. These dust storms created significant latency problems and lag with communications back to Earth control. In addition, the time delay between Mars and Earth of 24 minutes posed serious problems to controlling opportunity if Earth's control team can't redirect Opportunity in response to threats in time. That is where the autonomous capabilities of Opportunity took over.

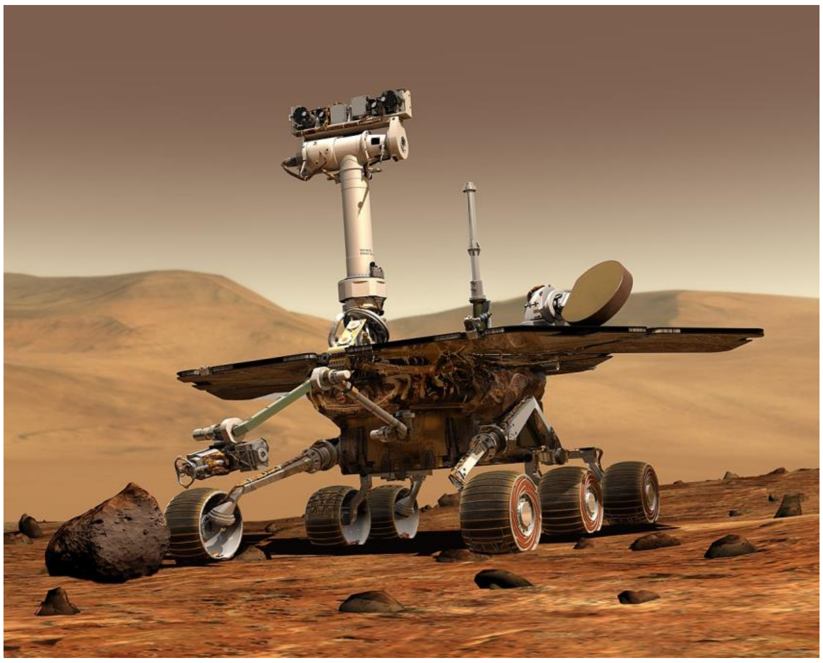


Figure 1-4: NASA's Mars Rover Opportunity [1]

Autonomous vehicles are employed in various sectors and industries for various tasks. Delivery robots are used to deliver packages and parcels to addresses while navigating real-life conditions like traffic, weather, landscapes and roads integrity and pedestrians. Autonomous robots are also utilised in remote hazardous applications in industries related to radiation, energy production, sewage systems etc. These are specific areas of fields that could pose real threat to human workers to carry out tasks for inspection and maintenance. In addition, as these robots usually take very long periods of times to complete a task, especially if it's on a one-way journey like traversing sewage underground pipes etc., they have the capabilities to perform self-maintenance. This is critical when one or more of the robot's hardware components breakdown which may act as a hindrance to the successfulness of the robot's mission.

1.2 Project Research Main Point of Interest

One of the most common applications of robots for domestic use is the use of autonomous vacuum cleaner robots to automatically vacuum home's surfaces without any human intervention. A well-known brand is a series of models known as Roomba developed by iRobot. The technology and concepts implemented within vacuum cleaner robots will be the main focus that will drive the research and development for this project. Vacuum robots such as Roomba have critical components that allows them to perform their functions properly. Each component or "sub-system" complements other sub-systems in a tight link. If a sub-system fails, the whole system is bound to fail as well. When it comes to vacuum robots, there are three main sub-systems which make up autonomous vacuum robots.

1. **Perception & Sensing**: Vacuum robots need to be equipped with appropriate sensors to allows the robot to 'sense' its environment and surroundings. These sensors would be the primary components used to navigate the robot around. A good analogy to explain how a robot would use its sensor is comparing it to a blindfolded person who is using their hands to feel objects in front of them. Some of the most common tools used to navigate robots around include:
    a. Infrared sensors
    b. Laser scanners
    c. Vision cameras
    d. Ultrasonic sensors
    e. Force-torque sensors
2. **Processing & Analysis**: Autonomous robots are equipped with a "brain" that will dictate what the robot does. Data processing and analysis is the core part of any autonomous robot because without a method present that can make use of the data received by the sensors, it is pointless. Depending on the varying levels of complexity and tasks to be carried out by the robot, the processor of the robot is selected. Robots are usually controlled by microcontrollers that have their own CPU along with RAM, ROM and other peripherals on the same chip. More complex robots would require more sophisticated tools to allow for increased functions and processing power. That includes faster processing speeds, larger storage spaces, memories and in general more options over a simpler one. These microcontrollers are programmed by engineers to instruct the robot how to analyse and process the information and then take a decision if certain conditions are met beforehand. A different approach for this sub-system of robots might make use of incorporating Artificial Intelligence or machine learning. However, the use of such advanced technology for simple or trivial tasks is usually unnecessary. This is a topic will be discussed further towards the end of the project.
3. **Actuation & Movement**: The third and final sub-system of vacuum robots, is the actual output seen as a result of the previous sub-system. Once the microcontroller of the robot has completed its processing of data from the sensors, it is now ready to perform actions than physically alter objects or cause manipulation in its surroundings. Some of the tools used to perform actuation or movement in robots are:
    a. Wheels
    b. DC Motors
    c. Comb drive
    d. Stepper Motors
    e. Servo Motors
    f. Hydraulic Cylinder
    g. Pneumatic Actuator
    h. Linear Motors

Methodologies and algorithms that are implemented by other researchers, innovators and hobbyists relevant to domestic robot applications like vacuum robots will be investigated, discussed and analysed.

1.3 Problem Statement and Project Objectives

This project has been set out to investigate and study state-of-the-art methodologies and concepts when it comes to developing autonomous robots. The project at large will be dedicated to improving the efficiency and proficiency of Autonomy within the robot. Experimentations and tests will need to be carried out to test different types of methodologies or algorithms currently existing to implement. In addition, where possible include improvements and own contributions that are specific to the project in hand which can help the robot function better. As the title of the of the project suggest, Autonomous Self-navigating Robot, the robot developed will need to be completely autonomous and is also self-navigating. The robot will need to navigate itself around in any environment allowing it to maneuver and avoid objects and obstacles along the way. The robot will also be expected to avoid falling over from high surfaces and stairs.

Furthermore, it is required to implement a way to allow for user manual control overriding the autonomous feature of the robot. This factors into real world applications where if systems fail and malfunction preventing the robot from carrying out its functions properly, a user would step in to override the system and manually take control of the robot. The complete project aims and objectives could be highlighted as below:

1. Autonomously navigate around objects/obstacles without collision.
2. Immediately stop functions and redirect the robot when faced with stairs or an event that would cause the robot to fall down a significant distance
3. Use of an application to wirelessly control the robot manually.
4. Constant improvement of algorithms to reach a high level of accuracy of detecting objects and obstacles through testing and development.

1.4 Project Components and Costs

Robots are notorious for requiring lots of small trivial components, yet crucial to the overall integrity of the robot. Electronic components, tools and parts for connectivity and connections, wires, mechanical fasteners like washers, nuts, bolts, screws. Additional components include wheels, sensors, microcontroller, DC motors, motor drivers etc. In addition, the chassis for the body for the robot needs to be designed and built properly to provide a strong hold for all components that will make up the robot. It has been discussed with the project supervisor Dr. Zoe Jeffrey who has recommended to purchase for a robot kit to avoid time wasted on tasks that are not targeted towards the overall contributions of the project. Moreover, Dr. Jeffrey has allowed for an increased project budget beyond the 75 pounds limit. if required to obtain components pertinent to the success of the project. The robot kit would include major parts needed to assemble the robot which include:

1. Robot body chassis
2. Wheels
3. DC Motors
4. Rechargeable batteries
5. Batteries housing
6. Batteries charger
7. Motors driver
8. USB connector
9. Mechanical fasteners that include:
    a. Screws
    b. Nuts
    c. Bolts
    d. Washers
    e. Pillars
    f. Spacers

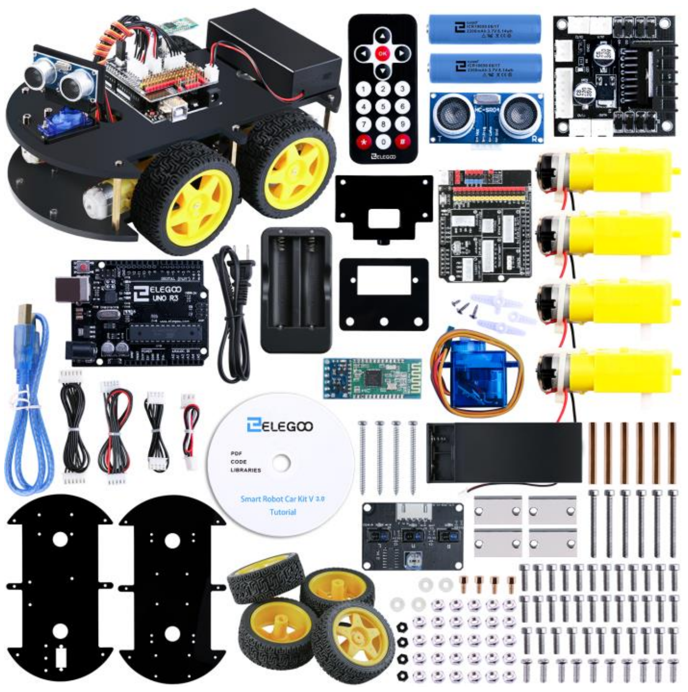


Figure 1-5: Robot Kit Parts [31]

Not all components in the kit were used as some of the components were irrelevant and not needed for the project.

#	Component	Quantity	Cost
1	Robot Kit	1	60
2	Ultrasonic Sensors	5	7.99
3	3D Printed Sensors Mount	7	25
4	Arduino Mega	1	28

Table 1-1: Project Components and Costs

The reasoning behind the purchase of some of these components will be discussed in-depth in later sections.

1.5 Project Marketability & Contributions

It is hoped that contributions and results of this project will help towards the marketability of more autonomous robots and increase their popularity for domestic use in daily human life. Whether its for home maintenance, entertainment or assistance for the elderly, these robots can improve human's lives drastically. In addition, concepts, algorithms and methodologies implemented in this project would be of help for implementation of autonomous navigation algorithms in different applications and various fields of robotics. All contributions and attempts at accelerating humanity's shift towards more autonomous robotic applications are invaluable to evolvement of societies and communities. They provide overwhelmingly positive arguments against human involvement and control.

1.6 Upcoming Report Structure

In the upcoming sections of this report, in-depth subject and literature review will be conducted about different state-of-the-art concepts implemented in this field. Detailed information regarding the work that went into developing the robot including experimentations, tests, developments and results. The four main navigation algorithms investigated will be discussed in-depth with results and analysis. Ending conclusions will also be provided to summarise the complete project. Side information will also be discussed regarding project management, further improvements and additions and potential future applications of the work done. To preface what is to come, below are the main topics with the corresponding sections.

1. Subject & Literature Review – section 2
2. Hardware Selections – section 3
3. Build & Assembly – section 4
4. Programming the Arduino (Calibrations) – section 5
5. Binary Movement Algorithm – section 6
6. Wall-Follow Navigation Algorithm – section 7
7. Fixed Rotations Navigation Algorithm – section 8
8. Automated Variable Rotations Navigation Algorithm – section 9
9. Bluetooth Communications & Manual Control – section 10
10. Automated Hardware Testing – section 11
11. Final Conclusions – section 12
12. Further development opportunities & improvements – section 13
13. Project Management – section 14
14. References
15. Bibliography
16. Appendices

2 LITERATURE REVIEW

There are numerous publications and hobbyists' websites with lots knowledge and information about different approaches and methodologies engineers take when designing autonomous robots. This section will attempt to dissect and highlight keys concepts and algorithms from various papers and websites. As mentioned previously in section 1.2, one the most common applications of domestic uses for autonomous robots is vacuum cleaning robots. They have the ability to clean rooms with complete autonomy and no intervention from a user. They have sensors equipped to allow them to prevent bumping into objects around the house and additional cliff sensors that prevent them from falling off stairs. Roomba, one of the most common vacuum robots developed by iRobot will be investigated first.

Even though Roomba is designed by an industrial company with lots of sophisticated capabilities and functions, there are key takeaways that could be extracted from such a product to incorporate into smaller scale projects. In [15] [16], it is mentioned that Roomba runs on two separate wheels controlled by two different DC electric motors for directional rotation and a 360° castor wheel. The DC motors can be driven in reverse direction allowing Roomba to easily spin in a circular motion. Allowing Roomba to spin on a single spot for a few more seconds helps cleaning dirtier spots that need more time for dust suction. The robot is powered by an onboard rechargeable battery pack stationed within the robot's compartment. The user does not need to extract the batteries to recharge them. Instead, the robot is programmed to dock itself when the battery storage is running low or after it has finished its operation onto its "base" that allows for batteries recharge. With the help of self-charger contacts, it is only a matter of getting the contacts in touch with the dock recharging points without any need for connecting cables or manual intervention.

Roomba is equipped with several sensors allowing it to perform its functions adequately.

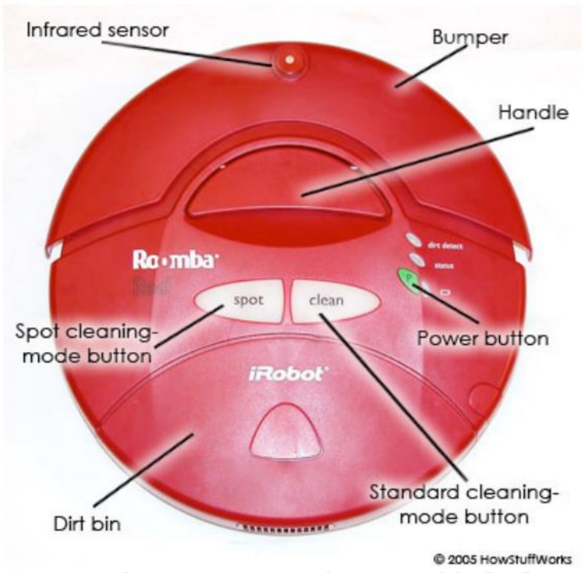


Figure 2-1: Roomba Top Side [16]

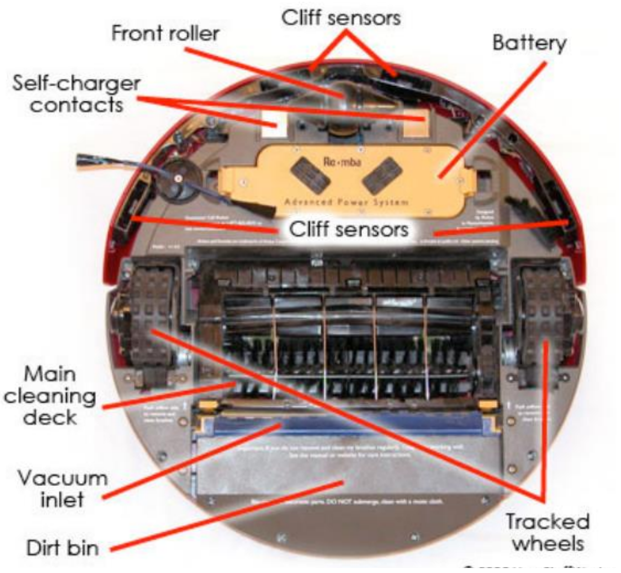


Figure 2-2: Roomba Bottom Side [16]

The robot is equipped with cliff sensors allowing it to detect and avoid steps immediately to prevent the robot falling down the stairs which will result in inevitable damage to the robot's internals. This is done by having the sensor constantly sending out infrared signal downwards vertically, the signal reflects off the ground's surface and is detected back by the sensor. Roomba is programmed to maintain a certain distance between its body and the ground. However, if the distance increases abruptly or gets lost and is not reflected back properly, that means that the robot is on a step's edge and changes direction in an instant. In addition, Roomba is equipped with a mechanical bumper that retracts upon collision with an object. The bumper performs two functions for the robot. First, it softens the impact and absorbs the collision by retracting slightly using mechanical springs which take the majority of the hit away from the robot's delicate components. Second, when the bumper retracts upon collision, this activates a mechanical object sensor that notifies the microprocessor that there is an object ahead, allowing it to change direction. This process is repeated and the bumper keeps retracting upon collisions until the robot finds a clear path.

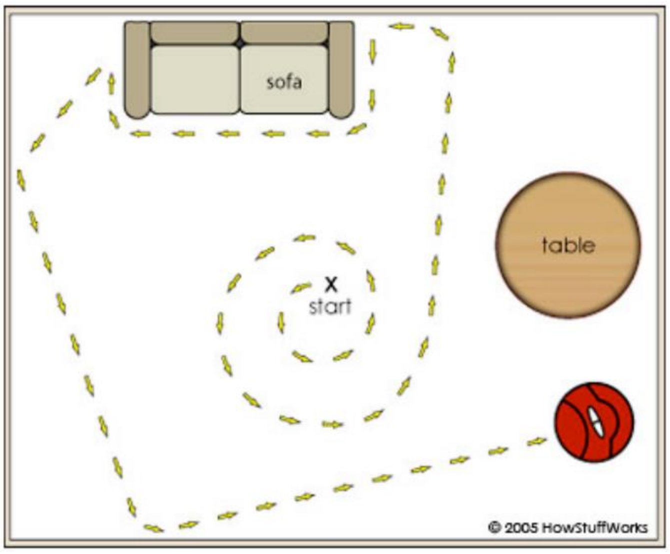


Figure 2-3: Roomba's Cleaning Algorithms & Patterns [15] [16]

Furthermore, the robot is also equipped with more infrared sensors called wall sensors. These wall sensors allow the robot to maintain operation and move very closely to a wall without collision or impact. This provides the robot with the ability to clean hard-to-reach edges and corners without risking hitting an object. The robot is additionally equipped with an infrared sensor which is used to "pan out" or estimate the room perimeter. This helps the robot to determine how long it would need to clean a room and its surroundings. There are no confirmations or hard evidence on what the implemented algorithms are for Roomba because this is an industrial product. However, [15] [16] believe that Roomba uses a mix of two concepts with help from all of its onboard sensors to perform its cleaning functions. Different Roomba models have different patterns and algorithms, however the most common patterns observed are:

1. Roomba starts with outward spiral movements. The radius of the outward movements mostly depends on two factors. If the robot thinks it has reached the perimeter of the room using initial data received from its main infrared sensor, it will cease moving in spiral movements and start moving along the perimeter of the room to clean all corners

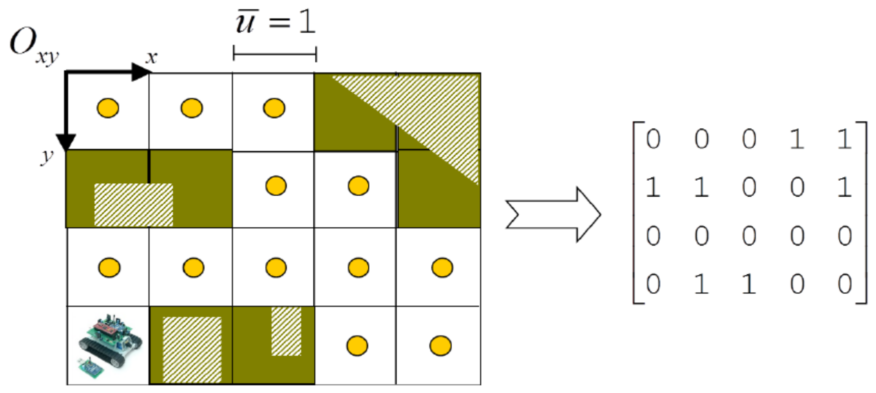
of the room. The second factor, while expanding out of the spiral, if the robot detects a close object, it changes direction and starts moving in a straight line.

2. The second pattern or algorithm observed from Roomba's operations is a "wall-follow" concept. Utilising the wall sensors mentioned previously, the robot moves alongside objects and walls carefully cleaning edges and corners without colliding at all with the object. The wall-follow methodology maintains a minimum distance between the robot's body and the object. If the distance gets too close, the robot moves out. If the distance gets too large, the robot moves in. Constant and quick data processing from the sensors allow the robot to react in time for the event.
3. The third pattern observed is the "wall-bounce" concept. This concept utilises the mechanical bumper mentioned earlier to navigate Roomba around objects to allow it to move in a straight clear path. Wall-bounce concept allows the robot to clean in a straight line until it hits an object then it diverges to a random direction and the concept is repeated.

Older Roomba models utilised navigation methods based on the above. These methods, while did the job, were not efficient enough. It took longs periods of time to completely clean a room and even then, there was no guarantee that all spots would be dusted. However, newer models are believed to make use of more modern and sophisticated ways to navigate the robot better around a room and obstacles. New versions of Roomba employ an ingenious concept called VSLAM (Vision Simultaneous Localization and Mapping). This concept makes use of onboard infrared sensors that captures pictures of small sections of a room which overtime, build up into a greater and more complete picture. This allows the robot to identify its route, where it's heading and where it has been before. This increases the efficiency of the robot drastically as it prevents it from cleaning areas that have already been cleaned earlier. Moreover, pictures and information created from past operations are stored and utilised for future cleaning sessions. The robot remembers the layouts of rooms overtime allowing it to operate more efficiently as time goes by. In a study done by Roomba's engineers [15], it was found the new employed method reduced operation time by 20%, saving more power. The robot also has the ability to allow for wireless connectivity allowing the user to control the robot wirelessly using a smartphone with the help of a compatible application.

In a second paper [17], it attempted to develop path-planning algorithms for indoor applications with low computational resources and restricted power supply. Modern path-planning algorithms are focused on finding the optimal route between two points for the robot to traverse through. However, these algorithms were not designed with processing and memory capacity in considerations, which more often than not, can be of great hindrance to the overall performance and scalability of the robot and its functions. Some of the most-popular concepts for developing intelligent robotic movement behaviour is through deterministic solution based on heuristic graph and reactive solution based on environment sampling [22]. Choosing between different approaches is dependent on several factors, what the robot is set out to accomplish, requirements, restrictions and hardware/software capabilities. The deterministic algorithms concepts offer incredibly quick execution times and optimum solution for a given problem. However, they are hindered by map scalability and expansions. On the other hand, reactive algorithms have no restrictions to map scalability and can adapt to dynamic changes. However, there is no assurance that the solution will be the minimum path possible.

To overcome the problems risen from both former concepts mentioned, [17] developed a navigation algorithm called HCTNav which has the advantages of very low computational requirements and shortest path-planning solutions for optimal solutions. Comparing the developed navigation algorithm HCTNav to more well-known popular approaches such as Dijkstra highlights some of the issues that HCTNav tried to overcome. Tests and results showed that HCTNav's memory peak during operation was nine times lower than Dijkstra's in mappings with greater than 150,000 blocks. Dijkstra's algorithm is attributed to deterministic solutions because of its heavy mathematical approach to path-planning which results in optimised solutions. However, utilising mathematical approaches which requires scanning for the whole frontier increases the algorithm's execution times which results in lowered speed performance. In addition, dynamic memory consumption exhibits quadratic growth which requires massive amounts of RAM in large graphs. HCTNav is conceived as a hybrid algorithm employing strategies and methodologies from both deterministic and reactive solutions. The solution attempted by HCTNav involves creating a 2D grid populated with cells that represent possible movement spaces. The navigation map can be reduced to a logical matrix with inputs from the Boolean domain. Each cell on the grid is identifiable by its (x,y) coordinates in a matrix. If a map is 10x10 cells, then the corresponding matrix is also 10x10. The formation of the matrix with 1's and 0's is dependent on what is perceived by the robot from its surroundings. A free cell indicating a clear a path is denoted by a 0 in the matrix while an occupied cell indicating a blocked path is denoted by a 1 in the matrix.



$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Figure 2-4: Mapping a 2D Map to a corresponding binary matrix model [17]

Obstacle detection in this model is implemented using two generic nodes Node[i] and Node[j] on the map. The first algorithm functions check for a clear trajectory ahead of the robot. If no clear path is detected, the one representing the first in sight from the robot's perspective will be returned. The obstacle detection strategy attempts to scan for possible collisions in the trajectory of the robot including the sides of the robot. This is to ensure that all the tiles belonging to the trajectory of the robot are clear of obstructions. The equation used to check for collisions between Node[i] to Obstacle [k] is

$$d^2(i,k) = (x_i - x_k)^2 + (y_i - y_k)^2 \tag{1}$$

The first algorithm function will return the obstacle 'k' in the form

$$d^2(i,k^*) = min_k\{d^2(i,k)\} \tag{2}$$

Which represents the first possible collision detect along the trajectory.

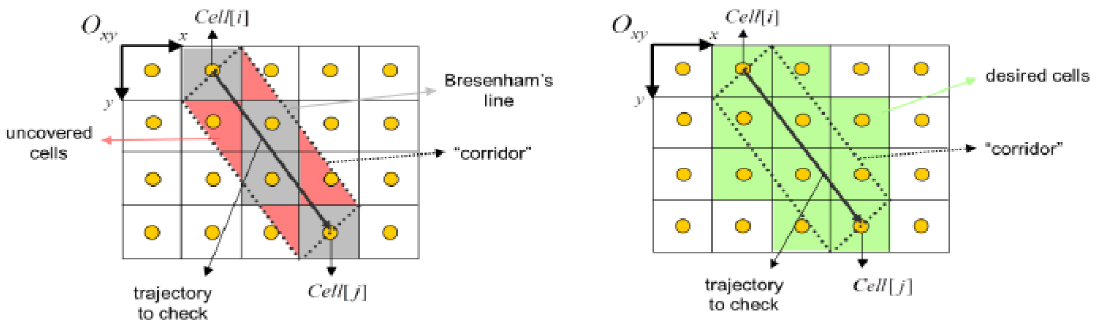


Figure 2-5: Visualizing the obstacle detection function demonstrating the "corridor" for the trajectory of the robot [17].

A final figure demonstrates the stages at which HCTNav reaches a conclusion and makes a decision based on generated maps and most optimal and efficient path-plan.

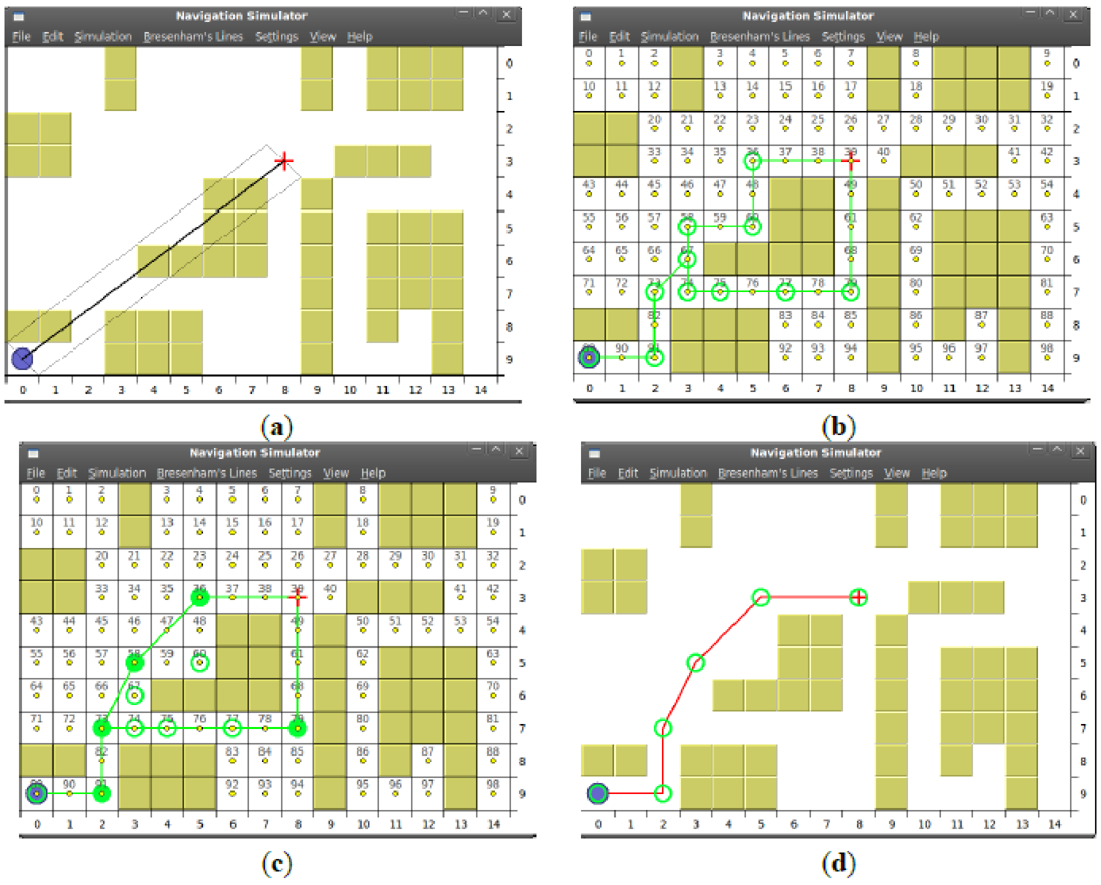


Figure 2-6: Complete HCTNav demo - (a) Initial map scenario; (b) Raw path-graph generated; (c) Optimized path-tree; (d) Shortest path returned [17].

A third paper [18] proposed an innovative approach for obstacle detection and avoidance for autonomous robots. The robot built comprised of an engine, motor controller, micro-controller and an array of sensors in a pentagonal formation as shown in the figure below. The simulation of the robot was carried out in MATLAB based on differential robot drive and equations of motion for a wheeled robot differential. The proposed navigation algorithm aims at getting the robot to learn and build its navigation rules without supervision or monitoring from an external party. This is attempted by employing chromosome encoding that is directly linked to behaviour. The architecture involves using a threshold to indicate a result related to what is perceived by the sensors. A value below the threshold indicates an obstacle detection while a value above the threshold indicates a clear path. The use of five sensors allows for 32 actions for each possible set of observations perceived by the sensors.

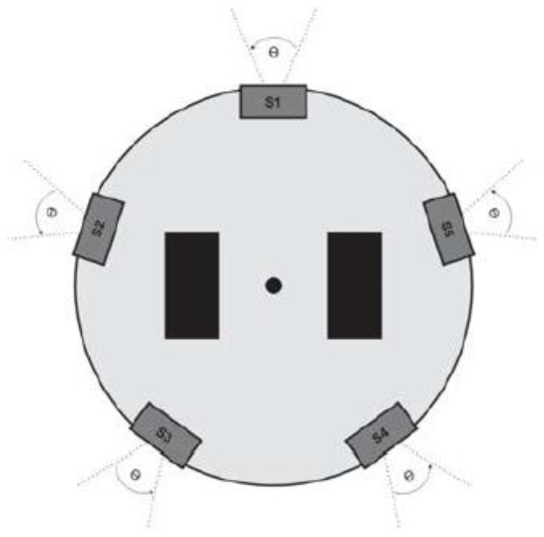
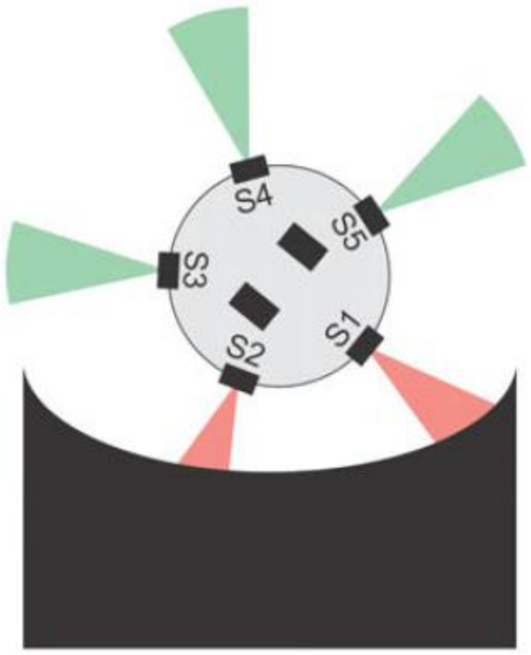
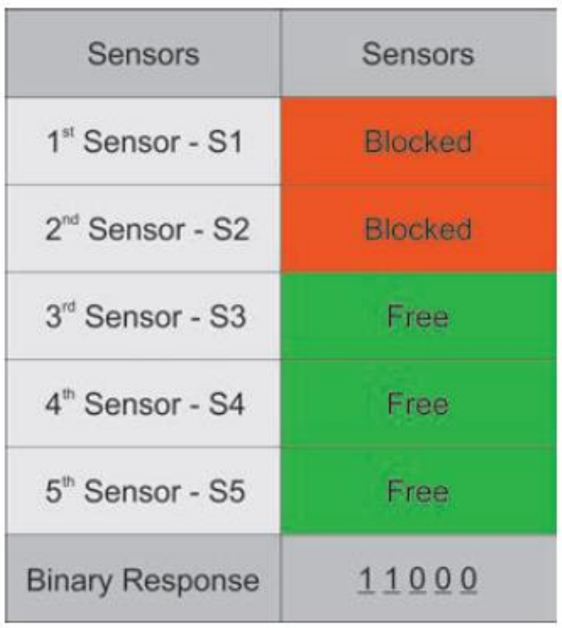


Figure 2-7: Proposed Robot Model [18]





Sensors	Sensors
1st Sensor - S1	Blocked
2nd Sensor - S2	Blocked
3rd Sensor - S3	Free
4th Sensor - S4	Free
5th Sensor - S5	Free
Binary Response	1 1 0 0 0

Figure 2-8: Proposed Architecture [18]

From figure 13, it can be seen that an array of sensors produces an integer output. Each possible integer output is linked with an action to perform. Therefore, the chromosome architecture is a transformed vector that encodes 32 states of the engine. The actions are indicated by the intensity of the Pulse Width Modulation (PWM) and is in the range of 0 - 255.

A fourth paper [19] put forth a compelling approach for the concepts and strategies observed earlier in Roomba's patterns and movements. Details of the navigation algorithms that were implemented in Roomba remained unconfirmed due it being a commercial robot. However, this paper has shed light on some of the methodologies employed in such a robot. The robot developed in this paper was set on two wheels controlled by two DC motors and a castor wheel in front allowing for easy 360 degrees of rotation. The main sensors employed in this robot were cliff sensors and bumper sensors. Similar to what has been discussed with Roomba's sensors, the cliff sensor is used to detect steps ahead of the robot and prevent it from falling down to its death. The bumper sensors are used to soften the hit to the robot's body and to signal the robot that the path ahead is obstructed and redirect navigation. The default value set for the mechanical bumper sensors is '0' indicating that there has been no collision and therefore no obstacle ahead. When the bumper retracts upon impact, the sensors goes to high '1' indicating an obstacle and thus allowing the robot to change direction. The block diagram shown in figure 15 displayed the complete architecture and flow of information in the system. All building blocks including power supply, clean brushes, motors, sensors and receivers are interlinked together that feed into the control unit which is the microcontroller. The microcontroller processes the information received from its side blocks and analyses the information then takes decisions based on pre-set requirements and conditions. The output can be observed as the wheels rotate when being controlled by the motor driver after it has received commands from the microcontroller.

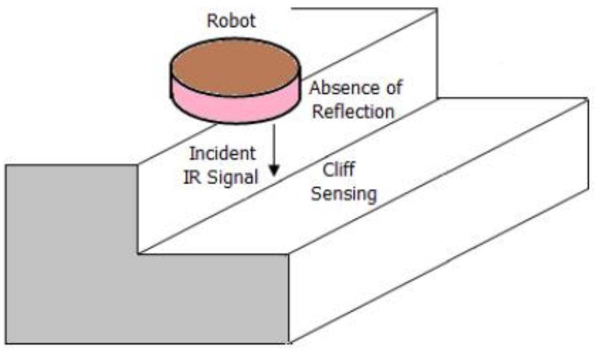


Figure 2-9: Steps detection using Cliff sensors illustration [19]

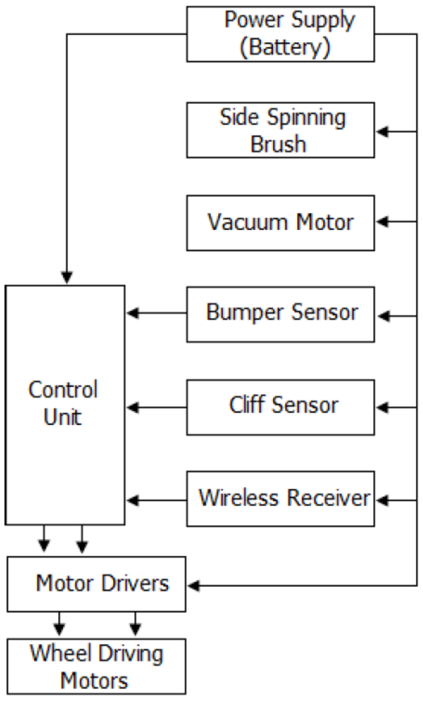


Figure 2-10: Robot's Block Diagram [19]

This paper has investigated four different navigation algorithms for autonomous control in the context of vacuum cleaning robots. These four strategies are:

1. **Random Walk** – The random walk algorithm, as the name suggests, has no perception of its environment or what the future path would look like. It is not guided by any mappings or intelligent approaches to dictate the routing of the robot. Instead, it makes random arbitrary movements

based on what the sensors detect. The robot keeps walking in a straight line until the mechanical bumper hits an object and retracts. Upon retraction, the robot processes data from its side sensors for any obstructions. If deemed clear, the robot rotates in a random direction based on an equation that introduces an element of randomness along with considerations to the robot's physical parameters. A robot with

a. Wheel radius 'r',
b. Wheel to centre point distance 'R'
c. Number of revolutions of wheel per second 'N'

Will have a random turning period of

$$\left[\frac{R\vartheta}{720.rN} + RNG\ (0\ to\ \frac{R\theta}{720.rN})\right]\ seconds \qquad (3)$$

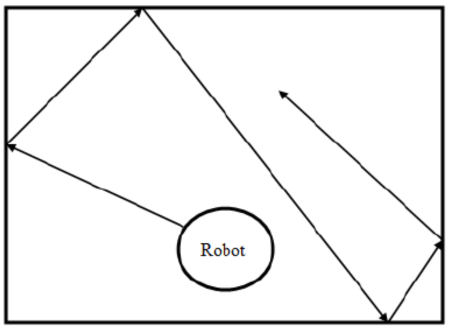


Figure 2-11: Random Walk movement motion illustration [19]

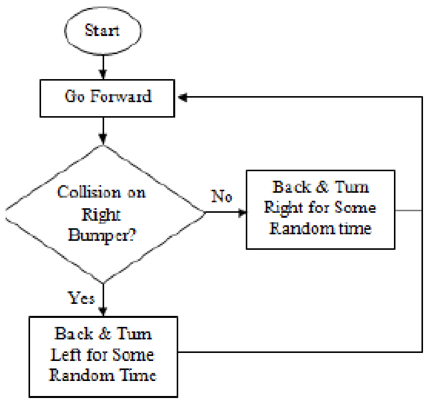

Start
Go Forward
Collision on Right Bumper?
No
Back & Turn Right for Some Random time
Yes
Back & Turn Left for Some Random Time

Figure 2-12: Random Walk Algorithm Concept [19]

2. **Spiral algorithm** – This algorithm is developed to simply allow the robot to rotate constantly in an outward expanding spiral. Before initiating the algorithm function, the robot checks for any obstructions in the way. If there is enough space allowing for spiral movement, the robot rotates in a direction in a constant expanding motion until an object is detected. This allows for quick and efficient cleaning of large area of space. Depending on what direction the robot is rotating the rotating of the wheels differ. If the robot is rotating anti-clockwise, the right wheel motor is rotating at full speed while the left wheel motor is rotating with a gradually increasing speed with respect to time. This results in the outward expansion of the spiral.

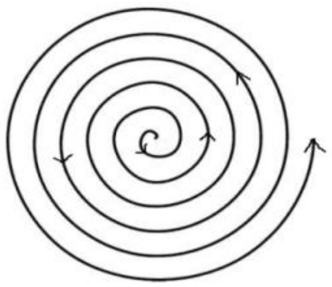
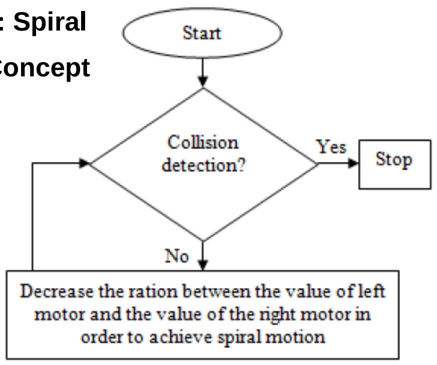


Figure 2-13: Spiral movement motion illustration [19]

Figure 2-14: Spiral Algorithm Concept [19]


Start
Collision detection?
Yes
Stop
No
Decrease the ration between the value of left motor and the value of the right motor in order to achieve spiral motion

3. **'S' shaped pathway** – The 'S' shaped algorithm allows the robot to move in S shaped movements back and forth in a given area. According to [19], this method more efficient and faster at covering an entire room compared to the Spiral movement. Employing this algorithm, the robot walks in a straight line ahead until confronted with an obstruction. Upon detecting an object, the robot follows a set of fixed instructions:
   a. Move backward
   b. Rotate 90 degrees to the right/left
   c. Move forward
   d. Rotate 90 degrees to the left/right

   Assuming that the robot body has a diameter of 'L' and backward distance = $\frac{L}{3}$ then the required times to perform the functions above in seconds can be formulate as:

$$\text{Back} = \frac{L}{6\pi r N} \tag{4}$$

$$\text{Rotate} = \frac{R}{4rN} \tag{5}$$

$$\text{Forward} = \frac{L}{2\pi r N} \tag{6}$$

   These equations are formulated specifically for the context of this paper. However, concepts and methods can be adopted to different similar projects and applications.

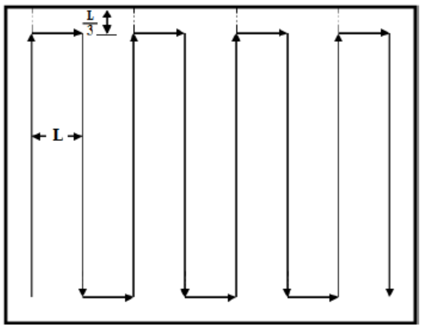


Figure 2-15: 'S' shaped pathway movement motion illustration [19]

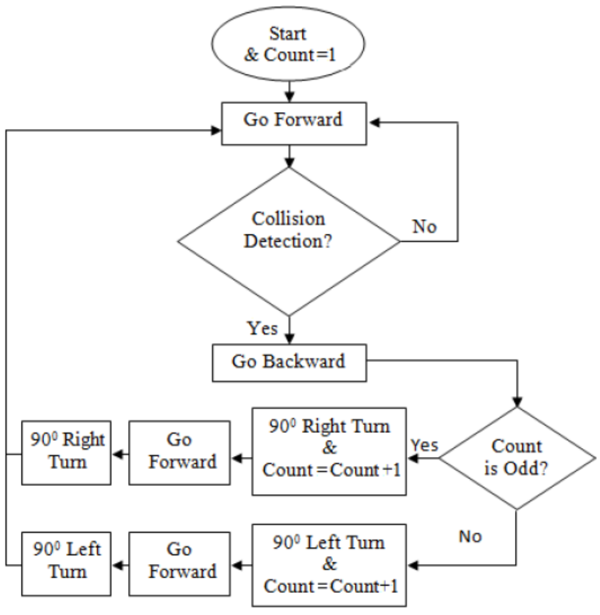


Figure 2-16: 'S' shaped pathway Algorithm Concept [19]

4. **Wall Follow** – As indicated by the name, the 'Wall-follow' algorithm controls the robot's body by maintaining a minimum distance between the robot and a wall. Using the side sensors on the robot, the control unit can constantly monitor for the distance and perform corrections as required. This strategy allows vacuum robots to clean dust-off corners and areas that are hard to reach or not clean often. This motion can be visualized in the figure 2-17 below.

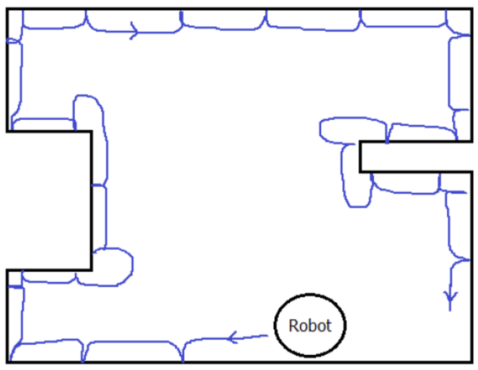


Figure 2-17: Wall-Follow movement motion illustration [19]

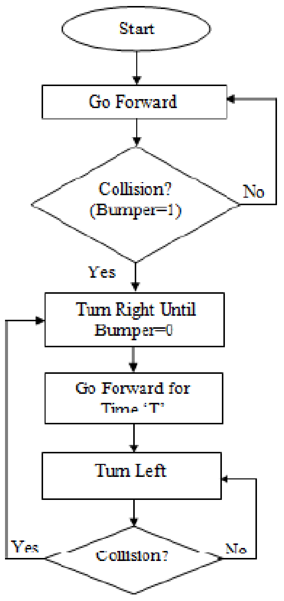


Figure 2-18: Wall-Follow Algorithm Concept [19]

In addition to the very valuable information and methodologies extracted from this paper that will affect how this project will progress, it also highlighted the importance of combining all previously mentioned algorithms to create a more complete and efficient robot. The use of a mixture of algorithms complements each other and eliminates some of the weakness that come with each strategy. This is visualized in figure 2-20 below where the performance graphs display the shortest time to cover the largest room area took place by employing a combination of all algorithms. This further reinforces the concept that a less efficient algorithm isn't valid for use at all as it can be improved and made more useful with the use of other algorithms.

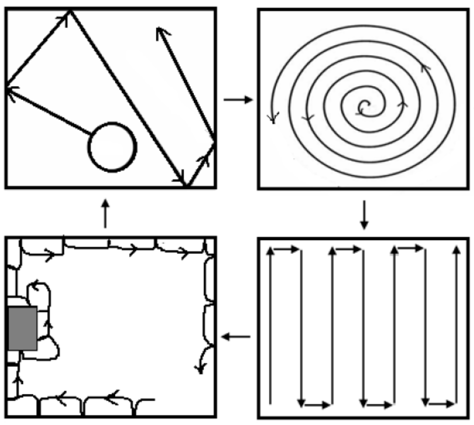


Figure 2-19: Execution of different algorithms in cyclic order [19]

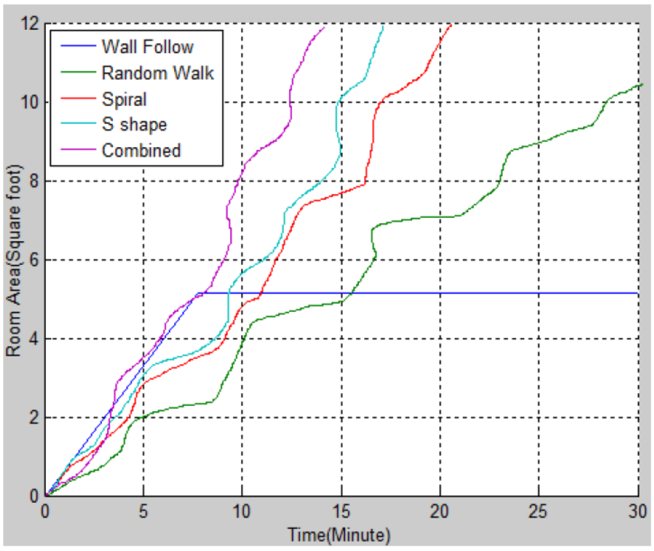


Figure 2-20: Performance Graph for different algorithms [19]

A fifth paper [20] attempted developing an obstacle avoidance robot using different techniques and components compared to previously mentioned papers. Previous projects investigated used to employ IR sensors for obstacle detection. However, this paper attempted using an ultrasonic sensor instead. An ultrasonic sensor works by emitting sound waves which are then reflected off an object's surface and analysed. The hardware utilised for this robot were an Arduino UNO microcontroller, ultrasonic sensors, DC motors, motor driver and a compass. The navigation algorithm employed in this paper is a basic strategy that forms the basis for all other navigation algorithms. Sensory data is sent to the microcontroller, in this case an Arduino UNO, which processes and analyses the information. If the data received from the sensor indicate a clear path, the robot keeps moving forward in a straight line. However, if the data from the sensor indicates an obstructed path, based on additional sensory information from its sides, the robot makes a rotation towards the clearer path, i.e. left or right. If the robot is surrounded from all direction, ahead, left and right, the robot moves backwards and goes back to where it came from. It can be concluded that the navigation algorithm employed in this paper is rather binary and restricted in how it reacts to obstacles and its movements. However, as mentioned previously using the result from [19], combining this algorithm with additional methodologies can enhance its performance and effectiveness.

A sixth paper [21] proposed and developed a path planning algorithm in dynamic environment using a more complex and sophisticated approach. Developing an autonomous robot that is able to traverse in a dynamic environment is considerably more challenging to a static environment. This is due to the fact that there are a lot more factors to analyse and consider that aren't present in static environment. Such examples include constantly moving objects, their velocities, trajectories, end point etc. To solve the problem at hand, problems were formulated to help assess the final goal. Such problems that were acknowledged are: How a robot can detect and separate an obstacle; how to privilege situation avoidance to overcome and how to treat a situation where a robot is faced with more than one obstacle. In order to answer these questions, some of the main assumptions that were made are:

a) The initial condition of the mobile robot satisfies $\alpha_R(0) = \alpha_g$ where $\alpha_g \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ is the desired direction and is assumed to be known to the microcontroller. (X_g , Y_g) is the target's cartesian coordinates.

$$\alpha_g = \arctan\left(\frac{Y_g - Y_R}{X_g - X_R}\right) \quad (7)$$

($X_R(0)$, $Y_R(0)$) is the initial condition of the robot and let τ be the sampling rate. The moving obstacle should satisfy only one constraint which is – the translation velocity of the dynamic obstacle is less than the translation velocity of the robot: $V_{obs} < V$.

b) In order to facilitate the development of the robot and its algorithms, all obstacles that are to be detected by the robot's sensors are to be approximated to circular objects. Disc 'C' with radius 'R' centred at point O ahead of the robot. The reasoning behind the approximations of all objects to circles is to allow for an efficient and uniform detect of obstacles. The intersection points between the circular object and the sensory waves can be depicted as

$$\theta_1 = \arctan\left(\frac{Y_{N1} - Y_R}{X_{N1} - X_R}\right) \quad (8)$$

With the help of few more initial assumptions and steps carried out in [21], the following illustrations demonstrate the obstacle detection and navigation concepts for the robot.

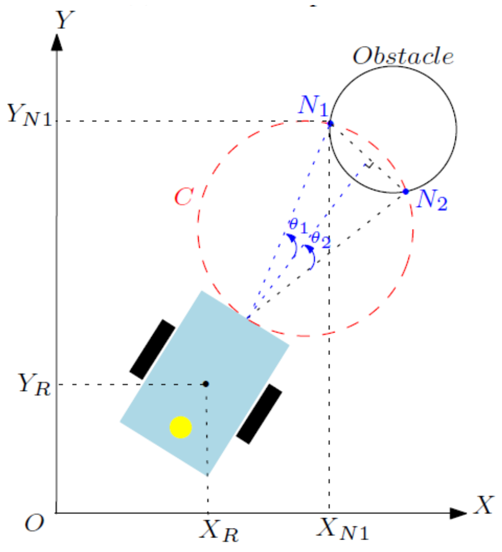


Figure 2-21: Obstacle Detection [21]

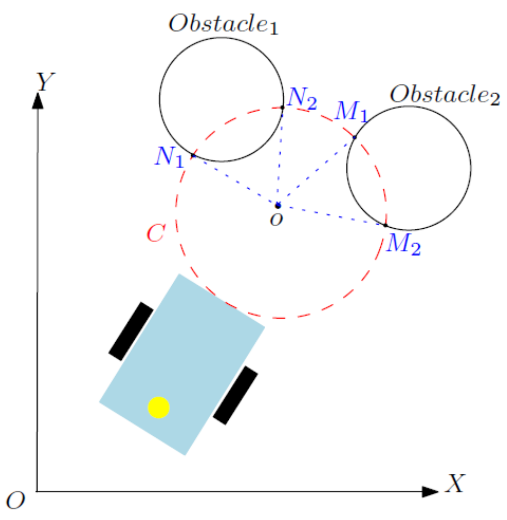


Figure 2-22: Intersection Points N1, N2 [21]

These fundamental equations, assumptions and illustrations along with additional more equations helped formulate a way to design a navigation algorithm in dynamic environments. A major advantage of this approach is that only is the algorithm able to navigate a robot around in a dynamic environment but is also able to navigate around in static environments and still objects. Below are simulation results for navigation in both static and dynamic environments.

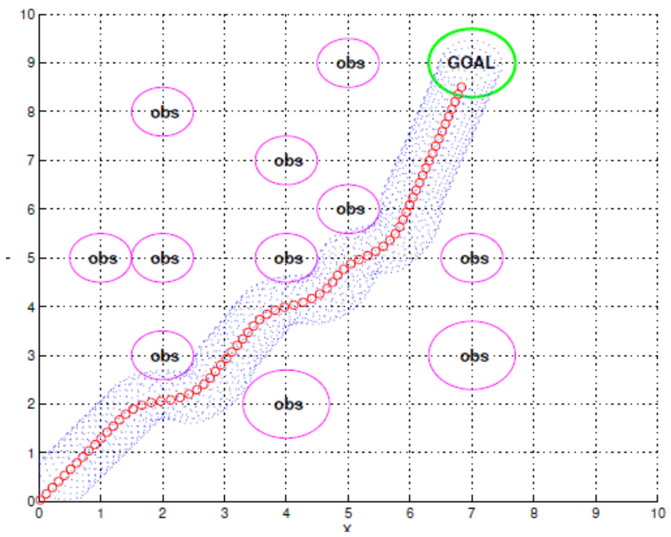


Figure 2-23: Robot Navigation in Static Environment [21]

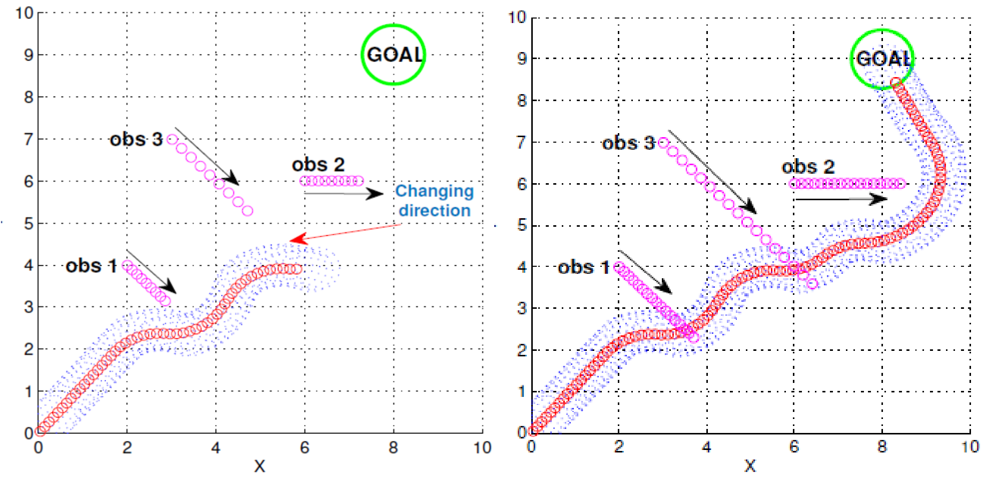


Figure 2-24: Robot Navigation in Dynamic Environment [21]

2.1 Conclusion and Selections

Following extensive research and investigation beyond what has been covered in this report to understand some of the latest state-of-the-art methodologies and concepts when it comes to designing autonomous robots. Papers [19] [20] which also tie closely with [15] [16] have been selected for further investigation for the project in-hand. [20] provides a solid foundation for autonomous navigation algorithm that can be built upon and enhanced to reach higher capabilities with the help of more advanced algorithms. Due to the simplicity and inherit foundational knowledge provided by [20], it will be combined with and improved upon with the help of [19] which provides large and extensive amount of information regarding strategies and approaches that are employed in modern autonomous robots. It discusses algorithms such as ‘Random Walk’ also known as ‘Wall-bounce’ and ‘Wall-follow’ which will be investigated extensively for implementation and development of this project. It also dives in-depth and discusses possible strategies that are implemented to develop Roomba vacuum robots in [15] [16] which have a lot of interesting and compelling approaches to navigate robots successfully with the additions of cliff and wall sensors.

3 HARDWARE SELECTIONS

Arguably, the two most influential and critical components that will dictate the success and the approach of this project will be:

- Microcontroller
- Sensor

The microcontroller is the main core of the robot that will bring all other subsystems or functions together. It shall hold all programming and navigation algorithms as well as processing and analysis of data received from the sensors. Choosing a microcontroller will need to be carefully assessed and compared with other alternatives to see which is a better fit for the project requirements in terms of reliability, performance, capabilities, scalability, limitations and cost. The second most critical component will be the sensor chosen to be used for obstacle detecting and navigation. Moreover, addition components worth discussing are motor driver to drive the DC motors for wheels rotation and Bluetooth module for wireless control of the robot using an external or separate unit.

1. **Microcontrollers**: For the research and investigation done across various research papers, publications and hobbyists' websites, there was a clear indication of the most common microcontrollers used to implement autonomous robots. Two of the most common microcontrollers used are Raspberry Pi and Arduino. Both products are aimed to a specific market of researchers and students working on personal and scientific project of mild to significant complexity. However, the main differences between both are the features and functions available in both controllers. Raspberry Pi is often referred to as a miniature small-scale general-purpose computer more so than a microcontroller with dimensions similar to a credit card. Raspberry Pi is an extremely sophisticated and advanced piece of technology bring users of all ages to explore and investigate computing challenges, programming mainly complex projects in a wide variety of languages. It has its own dedicated processor, memory and is capable of running an operating system which runs on Linux. It supports several functions and features a normal computer would be expected to do. Users have the ability to browse the internet, stream high definition videos, create spreadsheets, word documents and even playing video games. It also has the massive advantage of running several programs simultaneously at the same time. On the other hand, an Arduino is an open-source electronics motherboard based on simple hardware and software. An Arduino is a prime example of what a microcontroller is. An easy-to-use simple computer that can be used by young students getting started into the wide world of electronics. It is capable of one program at a time repeatedly and perform actions by triggering suitable outputs using inputs from external sources. Arduino boards are best suitable for simple repetitive tasks, ones that can be run one at a time. Raspberry Pi's are used for more

complicated projects that requires several functions or programs to be run together, for instance, weather forecast applications.

When it comes to comparing technical specification of the two boards, there is a ginormous difference between the capabilities of each board. A Raspberry Pi 3 Model B boasts a powerful 64-bit quad-core processor with a staggering 1.2 GHz processing speed [23]. It has 1GB RAM and expansion option for MicroSD. It has four USB ports, 40 GPIO pins, an ethernet port, in-built Wi-Fi and Bluetooth modules. In addition, it has in-build audio output, a graphic driver for HDMI output and ability to install operating systems like Windows 10. On the other hand, an Arduino Mega 2560 has an 8-bit microchip microcontroller with 16 MHz processing speed and a flash memory of 256 KB. It does not support external operating system installations, series processing architecture of one program at a time, no media capabilities, no in-built communication modules [24]. Furthermore, an Arduino can only be programmed in C/C++ while Raspberry Pi supports additional languages including Python and Ruby. Arduino boards are open-source allowing for a large database of information and tutorials

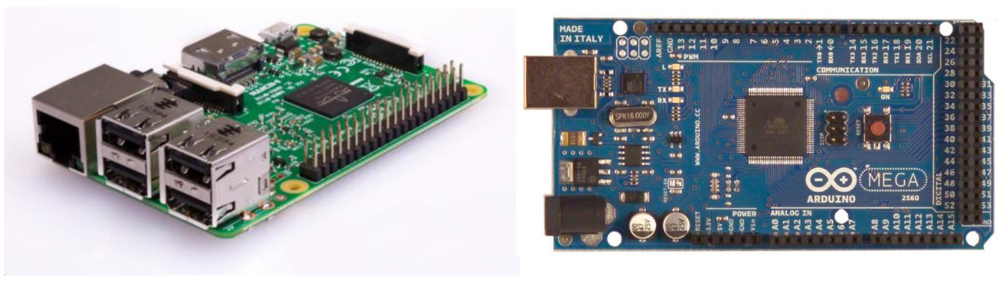


Figure 3-1: Raspberry Pi 3 (left) and Arduino Mega 2560 (right) [23] [24]

available on how to interact and work with them while Raspberry Pi’s are not open-source. Due to the simplicity of Arduino boards, interfacing with external components like sensors, motors and other electronic components is simpler compared to Raspberry Pi where it requires libraries and software to be installed to allow for the proper integration of the require electronic component.

Despite the overwhelming and clear advantages Raspberry Pi boards have over Arduino, selecting such a sophisticated board is unwise if the features available will not be utilised. Due to the nature of this project, an Arduino Mega 2560 is more than enough to support the requirements and the aims of this project. In addition to the substantial price difference of over £25, the ease-of-use, support and documentation available for Arduino compared to Raspberry Pi swayed the final choice selection towards an Arduino Mega 2560. Honourable mentions and contenders for selecting a microcontroller were an Arduino UNO and the use of basic PIC (Peripheral Interface Microcontroller). When it comes to the use of PIC, it required additional level of design and planning that was not contributing towards the overall project goal. Integrating PICs will require the design and build of suitable circuitry to facilitate the functions of a PIC including LC and RC filters, power supply and chopping circuits. The additional work was deemed unnecessary and not worthwhile for the time investment as it had no effects on project aims. The choice of an Arduino Mega over UNO came down to the availability of more I/O pins in addition to larger memory for larger code requirements when required. The use of a Mega allows for future expansions and scalability of the project to include more features and functions which would have been impossible with an UNO. Additional Mega 2560 data specifications could be found in the appendices.

2. **Sensors**: Sensors are the only components that will provide input to the microcontroller providing information about the surroundings of the robot and what is ahead. Therefore, the sensor selected and the quality of the component will play a significant role in how accurate the robot autonomously navigates itself avoiding obstructions in the way. The research carried out has shown that two of the most common sensors used for autonomous robots are IR sensors and Ultrasonic sensors. Both sensors work in different ways and have restrictions and limitations that will have drastic effects based on the application. IR sensors operate in the electromagnetic spectrum in the near infrared region of 700 nm to 1400nm [25]. Infrared has higher frequency range than microwave and lower than visible light. The simple explanation behind IR sensors is they transmit an infrared signal which reflects off an object's surface and is detected back by a sensor. This can be observed from figure 2-27. The majority of infrared sensors have distance range of around 15-20 cm and rises to 80 cm for more expensive modules. Due to the limited range of the sensors, the positioning of the IR modules is very important to ensure accurate and precise reflection of the transmitted signals for proper obstacle detection. Failure to correctly position the sensors will result in failure to detect object at specific angles. In addition, "signal contamination" between different IR modules is possible if they are placed in close proximity to each other as the transmitted signal of module can be received by a different module, further increasing accuracies and errors. Furthermore, due to the inherit operation of IR sensors, they are incapable of operating in outdoor environments where there is bright sunlight as sun rays contain IR waves which will interfere with the sensor module. On the other hand, Ultrasonic sensor work with complete independence from the electromagnetic spectrum. Ultrasonic sensor, as the name suggests, emit sound waves at a very high frequency that is in audible to humans. The emitted sound waves are reflected off object's surface which are then detected back

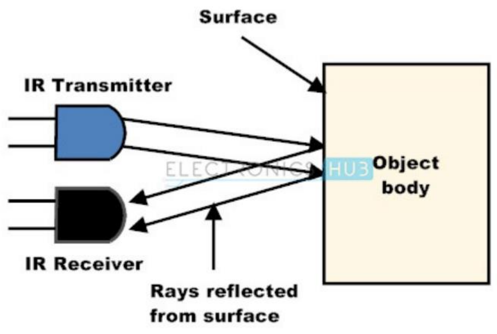


Figure 3-2: Operation of an Infrared Sensors for Obstacle Detection [25]

by the receiver of the sensor module. Distance measurement includes time calculations involving the speed of sound which is 343 m/s. Unlike IR sensors, Ultrasonic sensors are not hindered by sunlight rays or the presence of IR absorbing objects. Moreover, they have greater operating range of up to 350 cm which is far superior to IR sensors. Due to the advantages of Ultrasonic sensor and lesser limitations, they were chosen to be the main sensor for obstacle detection for this project. Chosen Ultrasonic model SR-04 has a trigger input pulse width of 10us, effectual angle of 15 degrees and measuring angle of 30 degrees and a resolution of 0.3 cm. Further detailed information could be found in the appendices.

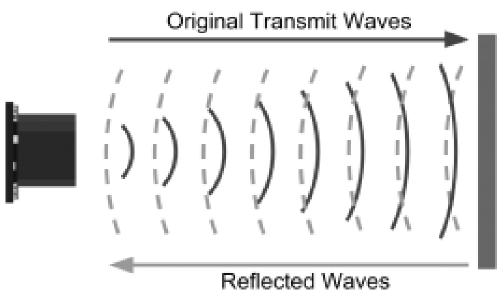


Figure 3-3: Ultrasonic Sensor Operation for Obstacle Detection [26]

3. **Bluetooth Module**: The choice of a Bluetooth module, while important, was not as critical as the previously discussed components. The main aim behind a Bluetooth module is to provide means of wireless communication with the robot using an independent external application which is to be controlled by a user. The chosen model HC-05 supports baud rates of up to 460800 bps and is capable of operating in master or slave mode [28]. Direct alternatives to this model where HC-06 and HC-08. The choice came down to the availability and price of the product as the detailed specifications were irrelevant for the application of this project. Further technical information regarding the chosen Bluetooth module can be found in the appendices.
4. **Motor Driver**: The motor driver control model L298N is used to drive the four DC Motors on the robot. The L298N is a dual H-Bridge motor driver that allows the user to control the speed and the rotating direction of the motors. The motor driver contains switching elements like transistors and MOSFETs to form an H-like configuration. With the use of PWM along with L298N, multiple DC Motors can be controlled by a single driver allowing for wide range of movement options, speeds and rotations. Further in-depth and technical information could be found regarding the motor driver in the appendices.

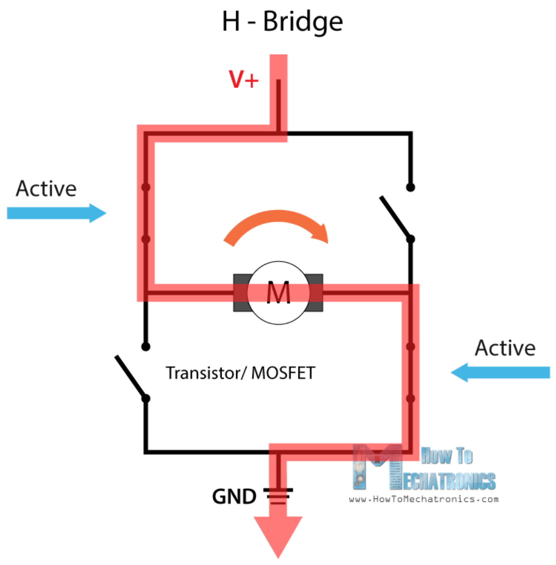


Figure 3-4: H-Bridge Illustration for L298N [29]

4 BUILD AND ASSEMBLY

The initial step towards building the robot together is it assemble the robot parts properly to ensure that the body is robust enough to with-handle impacts and hits during the development stage. As mentioned previously in section 1.4, the robot kit purchased contained the majority of the components required to carry out this project. However, as the kit is for general purpose projects, the components of the kit were not of complete fit for this project. There were few shortcomings within the kit that needed to be covered for.

First, this project was planned to be navigated around using five Ultrasonic sensors. A sensor would be used to detect objects directly ahead of the robot, two sensors positioned to either side of the robot angled at 45 degrees, a sensor placed looking backwards behind the robot and final sensor to be used as a cliff sensor, as discussed extensively in Literature Review. However, the robot kit came equipped with only a single Ultrasonic sensor, five additional sensors had to be purchased to cover for the shortcoming with an additional one as a backup in the case of sensor malfunction or failure. Second, the sensor modules are comprised of an emitter and a receiver placed on a PCB (Printed Circuited Board). The sensor modules have no housing or mechanical parts to allow them to be safely secured on a surface. Therefore, housings needed to be developed to allow the sensors to stand up correctly in a position for accurate readings and measurements. There were several options available to choose from for selecting a housing for the sensor. One solution was to purchase ready-made housings that were made-to-fit for the sensor module. However, those were expensive and with the need for five of them, that was a heavy burden on the project's budget. After further research and investigation, the next best and most suitable alternative was to 3D print the required sensor housings or holders.

Furthermore, the robot kit was provided with an Arduino UNO which was planned to accommodate one ultrasonic sensor. However, as mentioned in section 2.1, an Arduino Mega 2560 was selected over an UNO for the increased amounts of I/O pins to facilitate the addition of four more sensors. The Mega also allows for additional possible future requirements and expansions, which would have been a problem with an UNO. This brings up next the final shortcoming of the kit.

The UNO was made to fit on the acrylic plates provided which made up the robot chassis. The UNO has dimensions of 68.6 by 53.4 mm while the Mega has dimensions of 101.52 by 53.3 mm [24]. They both have different footprints and as a result have mismatching screw hole locations on the PCB. That meant that the pre-set mounting holes for the UNO on the acrylic plates were not matching for the Mega. Moreover, the additional four sensors had no holes for fixation and mounting on the acrylic plates. Therefore, additional initial steps had to be taken into account to accommodate the additions made to the robot, starting with the 3D printing of the Ultrasonic sensor mounts. This allows determining the required hole sizes before drilling.

4.1 Ultrasonic Sensors 3D Printed Mounting Brackets

A 3D model for the sensor mounting brackets needed to be selected with specific requirements. The mounting bracket would need to have a small footprint in order to fit five of them on the same acrylic plate size. In addition, the mounting bracket should have flexibility or freedom of movement to allow for easy change of sensor alignment, angle and orientation. The following 3D model was selected which had satisfied the requirements for a mounting bracket [30].

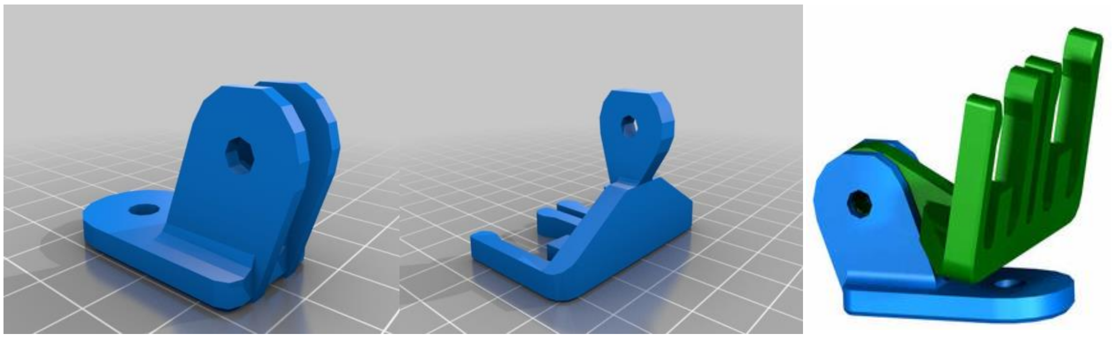


Figure 4-1: Sensor Mounting Bracket, base (left), body (middle), complete (right) [30]

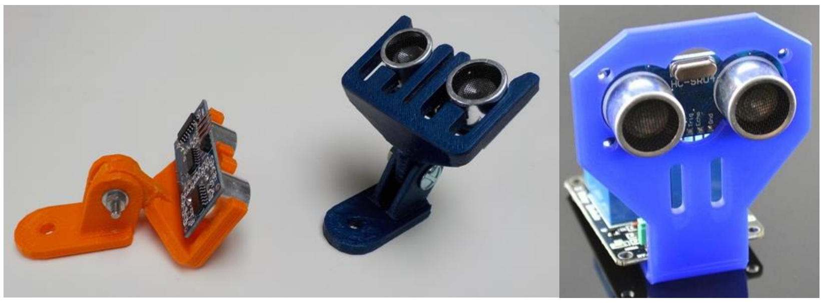


Figure 4-2: Flexible sensor mount vs Fixed mount [30]

The 3D model was of perfect fit for the project requirements as the base allowed for easy 360 degrees of rotation and vertical flexibility for elevated or lowered line of sight. Liaising and coordinating with laboratory technician was necessary to produce the required 3D models. Seven sensor mounts were manufactured in total allowing for two spares in the case of a break for one of the mounts. The building process took around 14 hours to complete and was completed within the University of Hertfordshire workshops and equipment.

4.2 Acrylic Plates Adjustments

Following the 3D printing of the mounting brackets, the next step was to work on the acrylic plates. The adjustments that needed to be done on the acrylic plates were additional drill holes of varying sizes to accommodate all new components introduced to the robot. This included new four M3 hole size locations for Arduino Mega and four additional M4 holes for the sensors mounting brackets. Coordination was carried out with the workshop technicians to drill the required holes based on guidance and directions provided to them. The following figures 3-3 and 3-4 demonstrate the required hole sizes and location on the respective plates top and bottom. All drill holes were drilled as clearance holes and no tapped holes were required. The side sensors holes were positioned far enough from the edge to allow enough room for 360 degrees of rotation without having the mounting bracket base sticking over an edge.

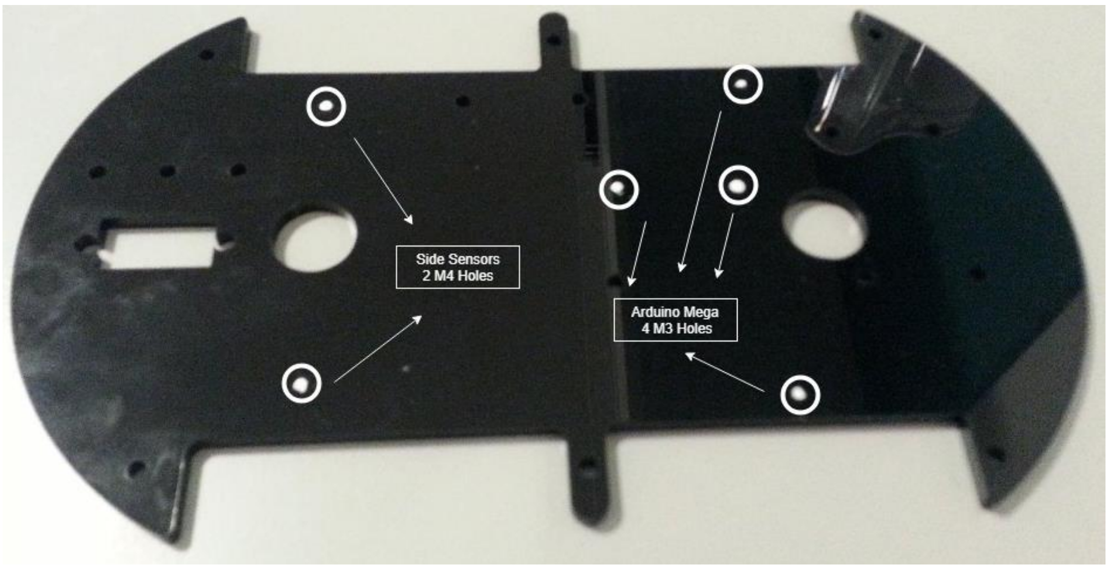


Figure 4-3: TOP Acrylic Plates Drill Holes

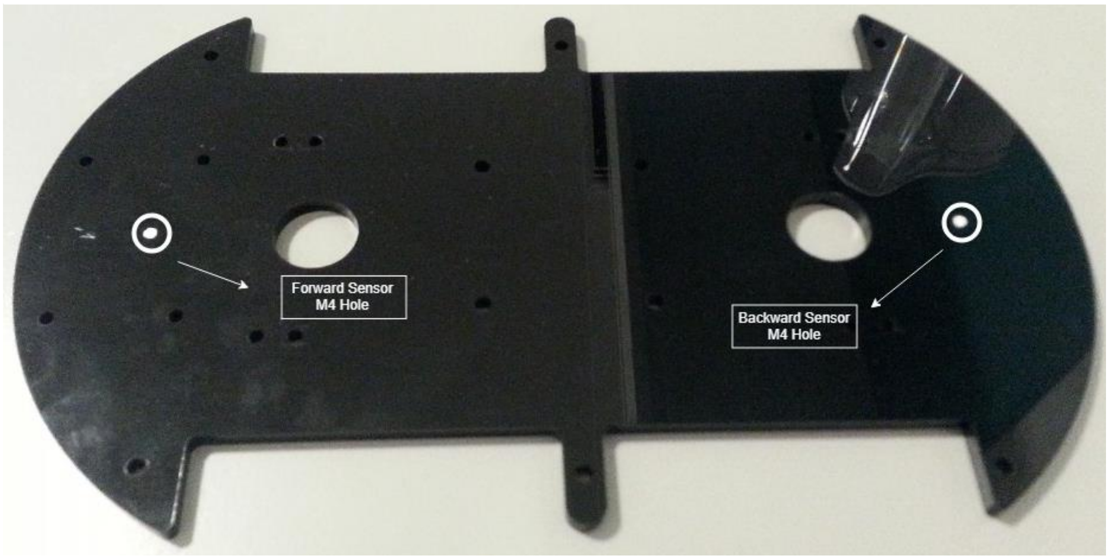


Figure 4-4: BOTTOM Acrylic Plates Drill Holes

4.3 Final Complete Robot Assembly

Following the adjustment made to the acrylic plates to accommodate the additional required components, the final step of the assembly was to put everything together and wiring all components together. DC motors were connected to the motor driver which was then connected to the shield provided within the kit. The Ultrasonic sensors were connected to a 5V power source while the emitter and receiver pins were connected to any digital pins available. The Trig pin on the sensor is used to trigger and emit the sound waves while the Echo pin is used detect and received the reflected sound waves. The Bluetooth module was connected to the Rx (receive) and Tx (transmit) pins on the shield. The following figure 3-5 shows the final build of the robot put together in its final state. The batteries pack is placed at the back-side of the robot in the black compartment which powers the complete robot. It should be noted that when programming the microcontroller and uploading code to the Arduino, power is not needed

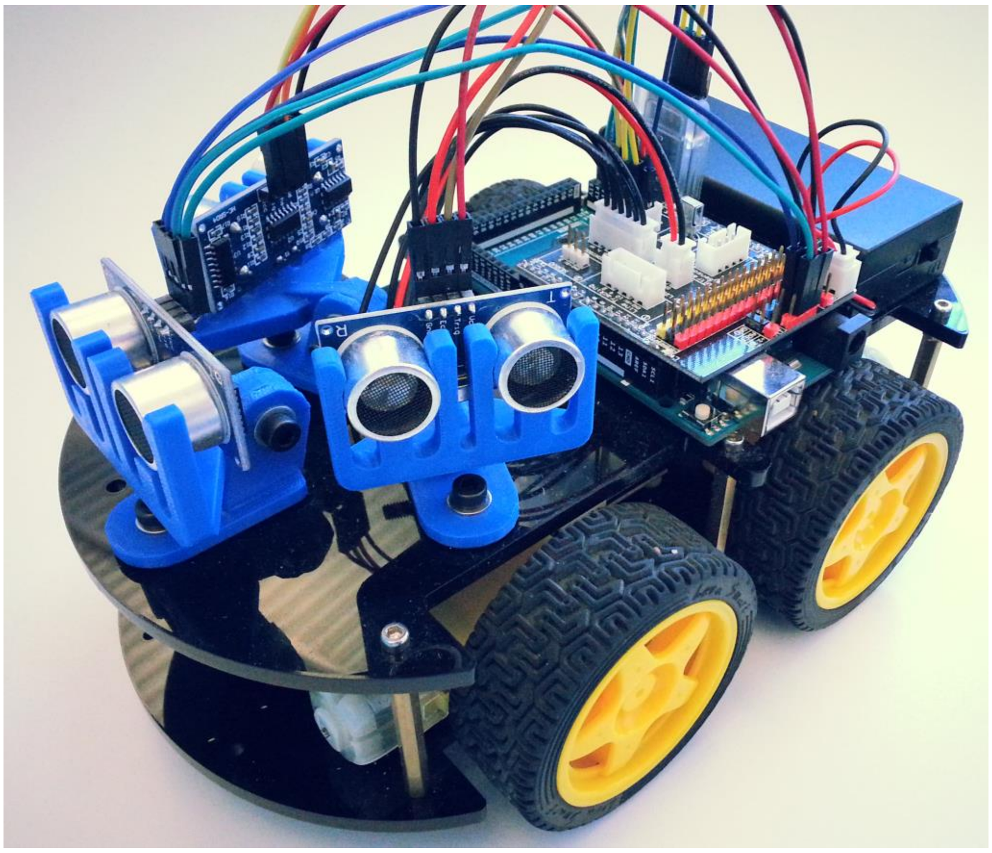


Figure 4-5: Final Robot Assembled Stage

from the batteries and therefore is not required to be turned on. A USB connection from a computer or a laptop is valid to power the Arduino and sensors for testing and development. In addition, in the case of using both power inputs, a USB plus the batteries, the power safe is selected automatically. It is advised however to turn off the batteries when not required to save power.

5 PROGRAMMING THE ARDUINO

Following the completion of the robot assembly, the next step of completing the project is to initiate programming the microcontroller to achieve the project aims and objectives. This is the most critical stage of the project which will at large dictate how successful the project is. The programming of the Arduino will involve coding several functions that will perform different tasks which combine to build the complete code together. Future sections will be split up to focus on each separate sub-system of the code on its own. To preface what is to come, the following is the breakdown of the sections:

- Sensors Calibration and Fine-tuning
- Binary (+) sign movement navigation algorithm
- Wall-Follow navigation algorithm
- Fixed Rotations navigation algorithm
- Automated Variable Rotations navigation algorithm
- Setting up Bluetooth communications
- Manual control
- Automated Hardware Testing

Selected sections from the above form different functions and sub-systems of the code that make up the final version of the project's code. The navigation algorithms investigated are selected from the papers mentioned in section 2.1. Starting with the Binary (+) sign movement as the base and foundation for developing the navigation algorithms, it will be designed and evaluated for its performance and improved upon with help from the rest of the navigation algorithms selected. The automated variable rotations navigation algorithm will be the end version of the code which will include personal contribution based on personal judgement and analysis that would result in a better and smoother performance for the robot overall.

5.1 *Sensors Calibration and Fine-tuning*

To ensure accurate and precise obstacles detection and autonomous control, it is paramount to ensure the sensors are providing the microcontroller with correct and true measurements. Failure to achieve precise readings from the sensors will result in a poor performance from the robot overall regardless of how advanced the navigation algorithms are. Before initialising sensor calibration, it is crucial to first read raw data from the sensor to understand the original margin of error being dealt with. This will provide a better and stronger indication how much error the calibration and fine-tuning of the sensor will try to fix. For the purposes of this project and its application, a reading precision of ±0.5-1 cm variance from true value will be deemed sufficient for what is required out of this robot. Figure 5-1 shows the sensor readings read by the microcontroller prior to calibration. For an actual distance of around 42cm, the sensor readings are seen to fluctuate rapidly from 40cm to 47cm which has a very high error margin and therefore is unsatisfactory. The error margin increased significantly for distances greater than 100cm as sounds waves would reflect wider at such distances, providing the sensor with even less accurate readings. The sensor calibration strategy used was implemented for all sensors as they all operated similarly, with respect to different flags and variables. First, the sensor module transmits sound waves by setting Trig pin on a High state for 10us. This High state sends out 8 cycle sonic bursts which travel at the speed of sound 343m/s. The sonic bursts are then detected back by the receiver, Echo. Based on the time duration it took for the sound waves to travel back to Echo, the sensor modules produces the time duration in microseconds. From the code's perspective, the time duration is dumped into a variable storing microseconds. Time is then converted to distance using the equation S=D/T. This results in the equation used to calculate the distance D = T * 0.0343 / 2 (9) where 0.0343 is the speed of sound in microseconds and the division by 2 is for the 2-way sound travel. To increase the accuracy of the readings, it is was required to eliminate readings which were considered extreme, contributing to the errors produced. Due to the resolution of the sensors, readings below 3cm and above 350cm were eliminated as they often resulted in very sporadic and inaccurate readings. This allowed for distances between 3cm and 350cm to be only considered. To increase the accuracy of the readings even further, readings in the specified range were sampled 10 times every 10 milliseconds and their average was taken. The average of 10 readings taken every 10 milliseconds produced far more accurate and precise results closer to the true value. The effects of the sensor calibration and fine-tuning is seen in figure 5-2. For an actual distance of 15cm, the sensor returned values between 15cm and 16cm was deemed very satisfactory for the requirements of the project. The principle above was then carried out to the rest of the sensors who each had their own flags and variables. After having the sensors set and ready for accurate distances measurements, next steps shall involve developing the navigation algorithms.

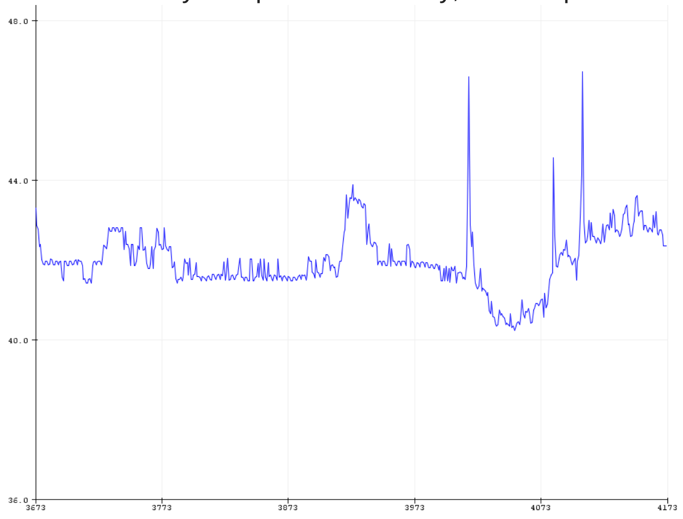


Figure 5-1: Sensor Readings BEFORE calibration

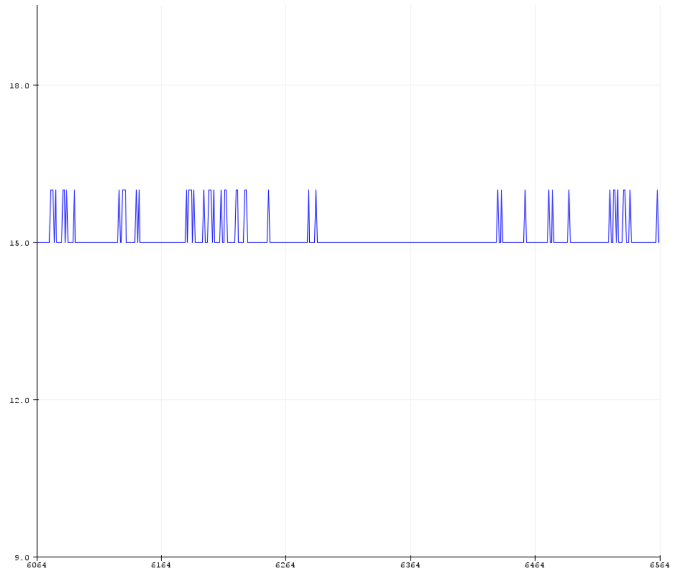


Figure 5-2: Sensor Readings AFTER calibration

6 BINARY MOVEMENT NAVIGATION ALGORITHM

The first navigation algorithm to be investigated is the most basic one out of the three selected algorithms mentioned in section 2.1. Binary Movement does not have any advanced strategy or approach when it comes to avoiding obstacles detected by the sensor. It does not utilise any advanced rotations or micro-adjustments to sway away from obstacles. Instead, the algorithm makes the robot change direction and move in the opposite direction. The algorithm results in making very binary, rigid and fixed movements that can be attributed to a (+) sign movement; forward-backward and left-right. Part of the unspoken objectives of the project is to produce a fluid and smooth robot movement which has flexible movements and maneuvers. The algorithm concept can be visualized in figure 6-1. The rotations involved in this algorithm can be simply categorized as sharp 90° to either side or a complete 180° rotation to the opposite direction. In addition, distances used to set the limits for the forward and side sensors we set based on the optimal performance of the robot observed during the development phase. In this algorithm, 30cm for all sensors was deemed satisfactory for the robot's performance. However, it is not a hard limit and can be varied slightly. The same concept carries forward to the next algorithms. Distance limits for the next algorithms are slightly different to better fit the individual algorithm's performance. The selected distances demonstrated the best reasonable performance observed while testing the robot.

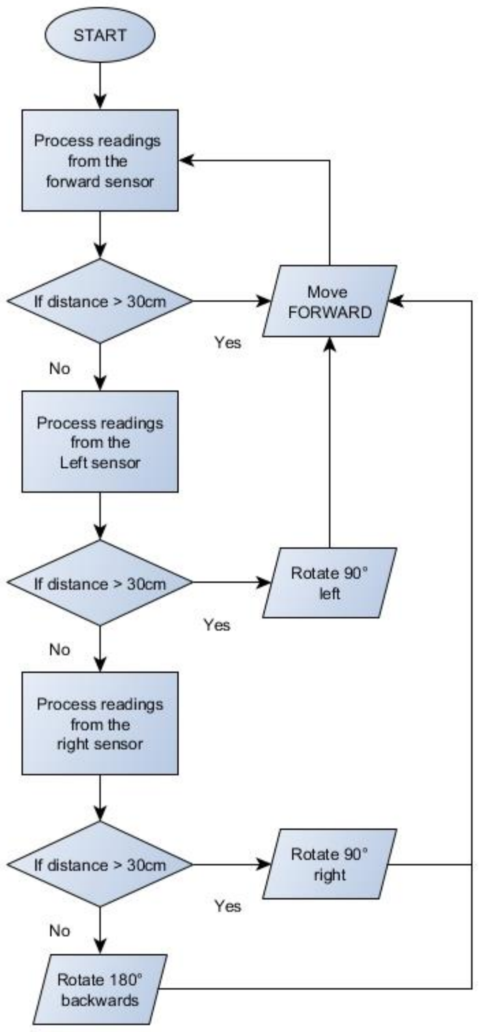


Figure 6-1: Binary Movement Algorithm Concept

Adapting the algorithm to a different robot with bigger or small dimensions would require adjustments of the distance limits based on the robot. For a bigger robot dimensions, large distance limits would be required to provide a safe distance of operation away from obstacles. On the other hand, for a small robot, distance limits could be lowered as the robot has a shorter breaking distance and therefore can operate with lower distance limits.

6.1 Test Run – Experiment

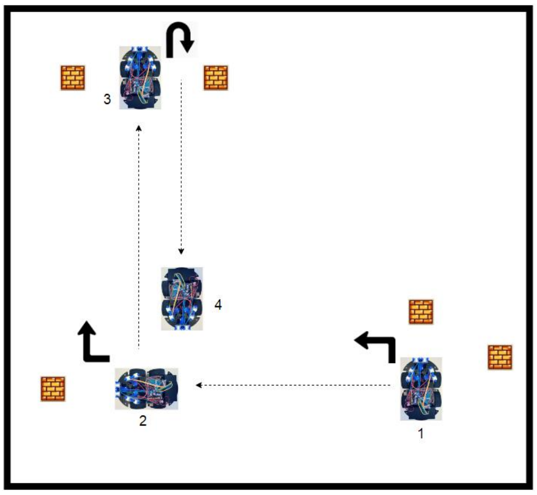


Figure 6-2: Binary Movement Algorithm Experiment Illustration

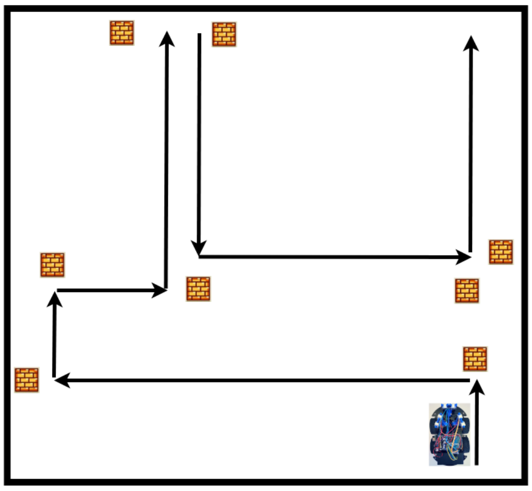


Figure 6-3: Path overview taken by robot during test run

A test run, or an experiment was carried out to evaluate and judge the performance of the algorithm when it comes to reacting to obstructions in the way. The experiment was carried out in an empty space in a small section of a room and obstacles were randomly positioned in the empty space, where the robot will traverse through. The motion of the robot was critically observed and analysed. Analysis and comparisons were made against the project aims to understand what other routes or movements the robot could have carried out to result in a better navigation performance. An illustration of the experiment can be seen in figure 6-2. A summarised path taken by the robot can be seen in figure 6-3 for a longer test run.

6.2 Results & Analysis

Based on the illustrations and figures shown in 6-2 and 6-3, it can be observed that movements taken by the robot under the Binary Movement Navigation Algorithm were rigid and fixated by sharp rotations of 90° or complete 180°. The robot has no sense of alternative rotations allowing for smaller adjustments less than 90°**.** For instance, in figure 6-2, robot positions 1 and 2 made sharp rotations of 90° when an obstacle was detected ahead of the robot. A more efficient and a sophisticated maneuver could have rotated by around 45° to get by the robot without having to change direction completely. Not only does this save power by reducing longer unnecessary rotations, but also allows the robot to travel longer distances in a shorter period of time by utilising smaller angle adjustments. As a result, the performance of this navigation method was deemed unsatisfactory as it does not help towards a smoother and more fluid movement for the robot.

Nonetheless, knowledge and concepts utilised in this algorithm will form foundations for future algorithms to help develop a better and more efficient algorithm. Indeed, all forms of autonomous robots use some form of sharp rotations which are built upon to make them more relevant for the projects' requirements.

7 WALL-FOLLOW NAVIGATION ALGORITHM

The second navigation algorithm to investigate is the Wall-Follow concept. As discussed in section 2, the Wall-Follow is a navigation algorithm that is widely used in robots that require operation near objects or a wall. When it comes to most navigation algorithms, the main obvious aim is to detect and navigate away from objects and obstructions in the robot's path. However, using Wall-Follow does quite the opposite as it requires a constant presence of an object in close proximity to be detect by the robot in order to navigate itself around. Using onboard sensors, the robot maintains a specific distance from the wall. As the robot starts to diverge away from the wall, it performs small adjustments to swerve back in. Similarly, as the robot starts to move closer to the wall, it performs small adjustments to swerve away from the wall. Wall-Follow algorithm is utilised mainly in autonomous robots such as vacuum cleaning robots such as Roomba and in inspection robots that perform analysis on structures, for instance, sewer and pipeline inspection and maintenance robots. The concept of the algorithm and decision-making process can be seen from figure 7-1. Unlike Binary Movement algorithm

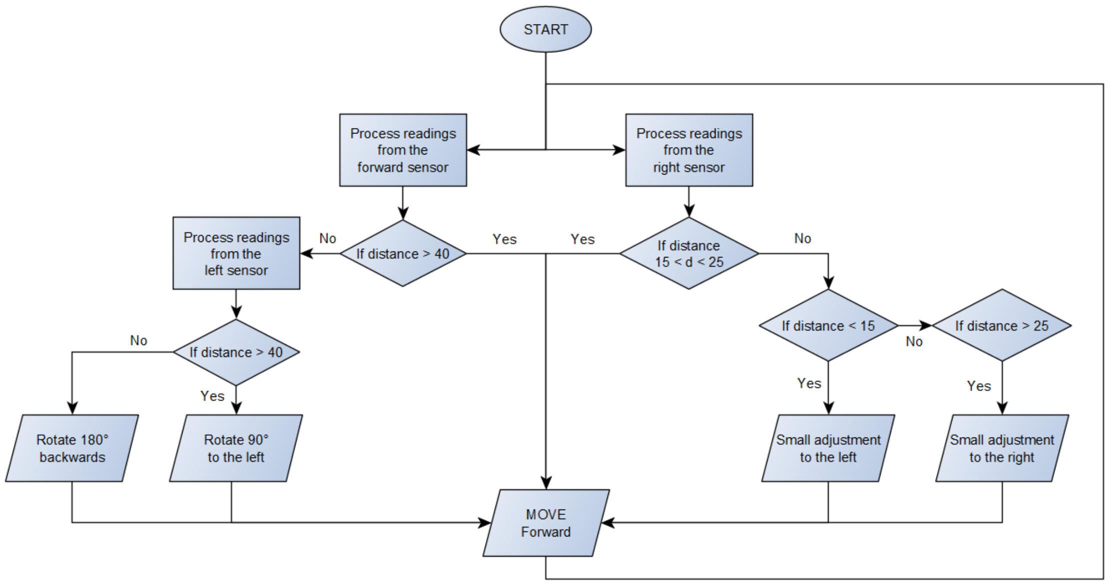

START
Process readings from the forward sensor
Process readings from the right sensor
Process readings from the left sensor
No
If distance > 40
Yes
Yes
If distance 15 < d < 25
No
If distance < 15
No
If distance > 25
No
If distance > 40
Yes
Yes
Yes
Rotate 180° backwards
Rotate 90° to the left
Small adjustment to the left
Small adjustment to the right
MOVE Forward

Figure 7-1: Wall-Follow Algorithm Concept

investigated earlier, Wall-Follow incorporates a more sophisticated turning concept where it is not restricted by fixed 90° rotation. It utilises small adjustments as necessary to slowly turn away from or towards the wall. Movements can be as small as 10°, just enough to allow for necessary adjustment required for the operation of the robot.

7.1 *Test Run – Experiment*

To judge the performance of the Wall-Follow algorithm, the robot has been put to test in the same testing space conducted for section 6.1. For the purposes of this testing, the robot is programmed to initiate operation with the right side set as a reference for the wall. Therefore, all tests require the robot to be physically placed close enough to a wall or an object on the right side. Figures 7-2 and 7-3 demonstrate the experiment and the behaviour of the robot

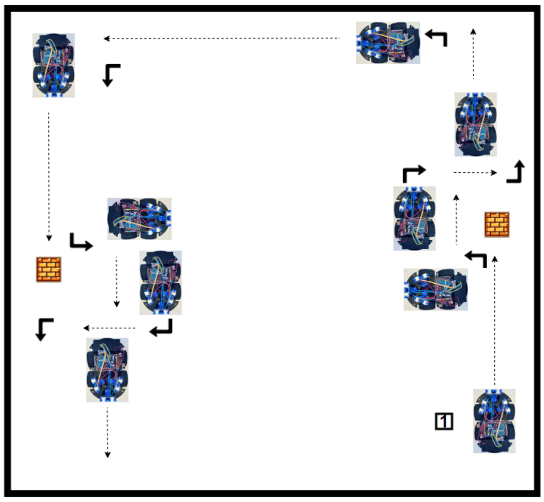


Figure 7-2: Wall-Follow Algorithm Experiment Illustration

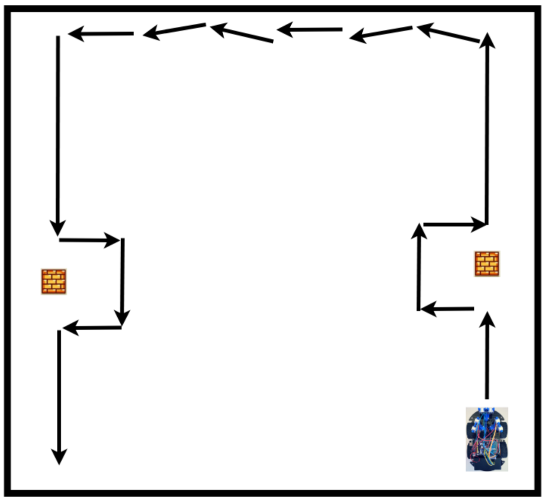


Figure 7-3: Path overview taken by robot during test run

observed under Wall-Follow algorithm. Two objects have been placed in the robot's path when moving adjacent to a wall to observe its reaction to obstacles using Wall-Follow algorithm.

7.2 Results & Analysis

It can be observed that the robot maintains constant proximity to the wall when programmed to having the right side as a reference. When the robot detects an object ahead, it makes a 90° rotation to the left if clear and then performs another rotation 90° to the right once the object has been cleared. It then performs another 90° to the left to get back on the its original path and maintain close operating proximity to the wall. Move ahead, it was observed that when the robot would start getting closer to the wall, the robot would perform a small angular adjustment of around 10° to 45° as required to stay away from the wall, once the distance gets back to safe levels, the robot performs more small adjustments to go back in closer to the wall and maintain the specified distance range. The robot performs this cycle of maneuvers to ensure to stay in close proximity to the wall within the specified limits.

Furthermore, it is seen that due to the nature and inherit feature of the Wall-Follow algorithm, the majority of the test area space towards the centre has not been explored by the robot at all. Utilising this algorithm means that the robot will ignore large areas of space if there is no wall nearby that can be used as a reference. Tests where Wall-Follow algorithms excel include testing conditions in mazes. Due to the behaviour observed while comparing against the overall project aims and objectives, the Wall-Follow algorithm was concluded to be unsatisfactory as it is limited to what it can do and restricts the overall flexibility of the robot.

Despite the shortcomings, the Wall-Follow algorithms has demonstrated better maneuvers and flexibility when it comes to moving around obstacles. It utilised better adjustments decisions to clear objects compared to the Binary Movement algorithm. This will be carried onto the next algorithms helping towards improving the final version.

8 FIXED ROTATIONS NAVIGATION ALGORITHM

The third algorithm to be investigated is the Random Walk navigation strategy discussed previously in [19]. The algorithm utilises elements of randomness incorporated commonly in autonomous vacuum cleaning robots. The randomness helps vacuum cleaning robots to move in all directions to clean as much area as possible. However, in the context of this project, the randomness is meaningless and illogical. Therefore, equation (3) mentioned earlier has been adjusted to remove the RNG in the equation and substituted with a specific rotation periods, thus the name Fixed Rotations Navigation Algorithm. This navigation strategy combines advantages from the previously discussed strategies in sections 6 and 7 while eliminating their drawbacks. The fundamental concepts in Binary Movement realized earlier help to produce the basic movements in a (+) sign. In addition, the more sophisticated adjustments in Wall-Follow allow for more enhanced fluid rotations. The combination of these movements eliminated the drawbacks of each algorithm by cancelling out the restriction of limited rotations in Binary Movement of 90° and 180° and the constant need for the presence of a wall for navigation in Wall-Follow. With Fixed Rotations algorithm, the robot can move freely in an environment without the need for a wall as a reference and has more flexible adjustments. The decision-making process behind this algorithm can be seen in figure 8-1.

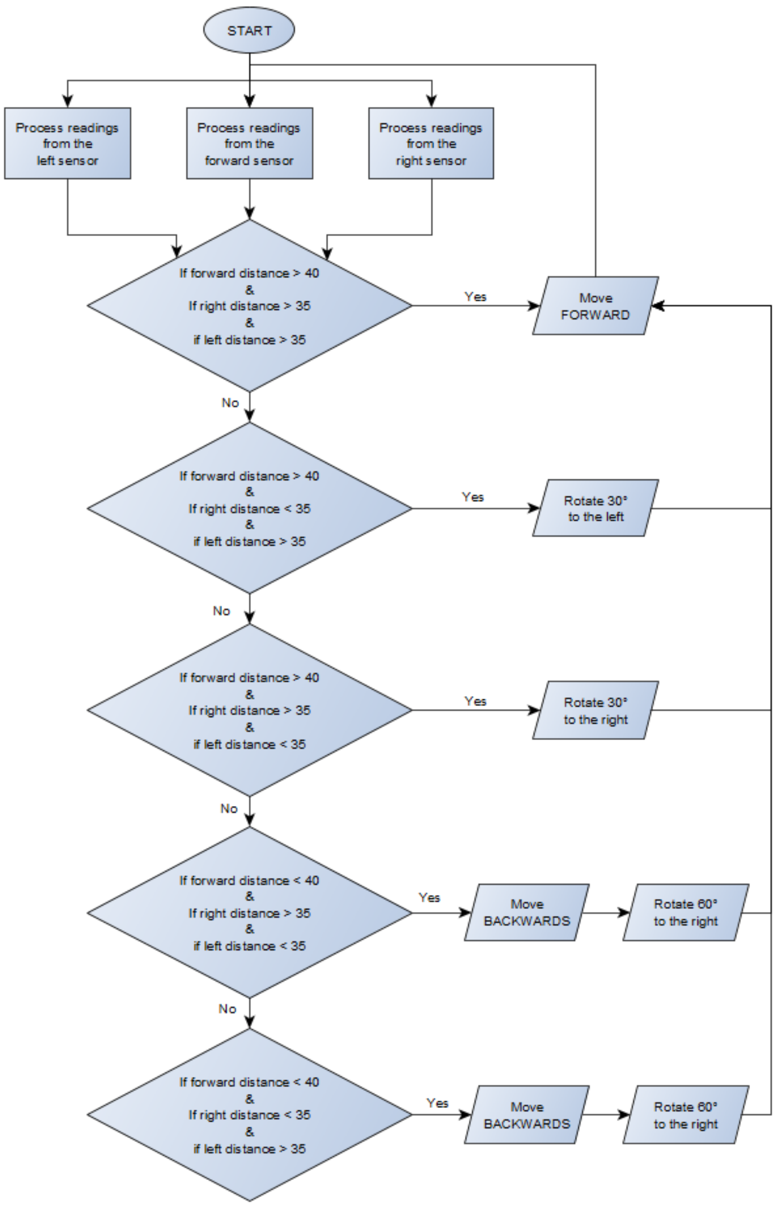

START
Process readings from the left sensor
Process readings from the forward sensor
Process readings from the right sensor
If forward distance > 40 & If right distance > 35 & if left distance > 35
Yes
Move FORWARD
No
If forward distance > 40 & If right distance < 35 & if left distance > 35
Yes
Rotate 30° to the left
No
If forward distance > 40 & If right distance > 35 & if left distance < 35
Yes
Rotate 30° to the right
No
If forward distance < 40 & If right distance > 35 & if left distance < 35
Yes
Move BACKWARDS
Rotate 60° to the right
No
If forward distance < 40 & If right distance < 35 & if left distance > 35
Yes
Move BACKWARDS
Rotate 60° to the right

Figure 8-1: Fixed Rotations Algorithm Concept

8.1 Test Run – Experiment

A test run has been set up to judge the performance of the algorithm to assess whether the performance is satisfactory or further improvements needed to be done. Experiment conditions were similar to previous test runs. Figures 8-2 and 8-3 demonstrate the behaviour of the robot under Fixed Rotations Algorithm when faced with obstructions in the way.

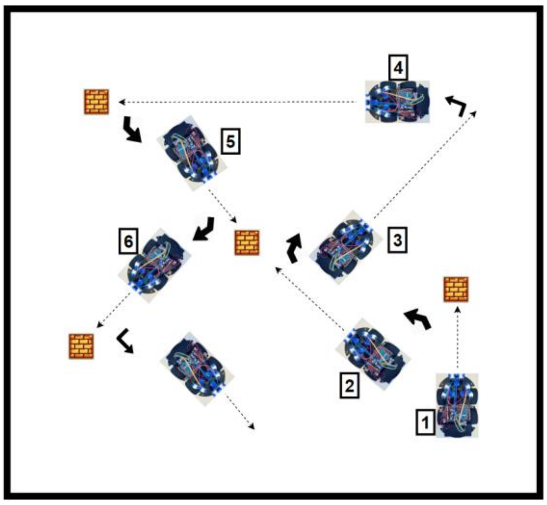


Figure 8-2: Fixed Rotations Algorithm Experiment Illustration

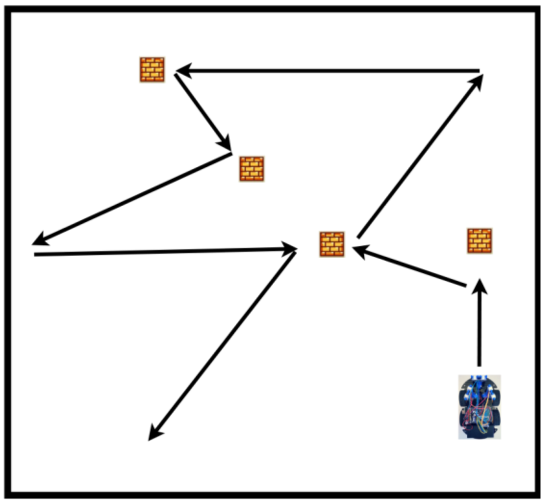


Figure 8-3: Path overview taken by robot during test run

Figure 8-2 shows the behaviour of the robot in a specific set of movements while 8-3 displays a clearer overview of the path taken by the robot for a longer test run. Unlike the Wall-Follow algorithm, the robot is not restricted to the right-hand side and is capable of starting anywhere.

8.2 Results & Analysis

By observing the illustrations above, the sophistication of the algorithm can be assessed. Comparing to the previous two algorithms, it can be clearly seen that it provides a much better performance in terms of movement, smoothness and fluidity. It eliminated the rigidness and restricted movement from the Binary Movement algorithm and is not dependant on a wall for reference using Wall-Follow. In addition, the robot is free to roam in the area towards the centre away from walls. However, despite the improvements observed; the robot's movement was not as natural as it was aimed to be. There were several situations where the robot could have taken a better route or made a more efficient rotation. Due to the nature of the Random Walk equation (3) with eliminated RNG element, the robot rotates small fixed movements every time, regardless of whether that rotation was suitable or not. This can be analysed by inspecting figure 8-2 carefully. In positions 3, 4 and 5, the robot could have made smaller angle rotations instead of making the same angular rotation every time. Improving the robot's rotations efficiency is crucial to the robot's overall agility and ability to traverse through obstacles as quickly and efficiently as possible. Furthermore, additional critical problems introduced challenges and uncertainties that affected the robot's rotations. These issues will be covered in section 9.2. and will explain why Automated Variable Rotations is better at dealing with them.

9 AUTOMATED VARIABLE ROTATIONS ALGORITHM

The final algorithm to be developed is the Automated Variable Rotations algorithm. This algorithm aims at combining all the advantages of previously mentioned algorithms while eliminating all of their drawbacks. Bringing together a more complete and fine-tuned algorithm to better fit the aims and objectives of this project. With the previous algorithm, Fixed Rotations, it has been highlighted that the major drawback is the fixed angle rotations made regardless of the position of the object. This means that for differently positioned objects, the robot will make the same rotations regardless of whether that full rotation was necessary or not. This is due to the inherit fixed randomness that was available from [19]. With this new algorithm, it is aimed to eliminate this restriction by introducing rapid monitoring of the objects during the rotation. This allows the robot to keep rotating **until** the object has been cleared, i.e. once the object is cleared, the rotation stops. With this implementation, the robot will have far more fluid and natural movement making only necessary rotations as required. In addition, the reduced rotations overall allow the robot to operate faster making it cover longer distances in a shorter period of time as rotations are done as minimally as possible. This strategy of combining different algorithms together follows the benefits realized in figure 2-20 from [19] where the best performing algorithm was the combination of all strategies. Automated Variable Rotations methodology can be observed in figure 9-1.

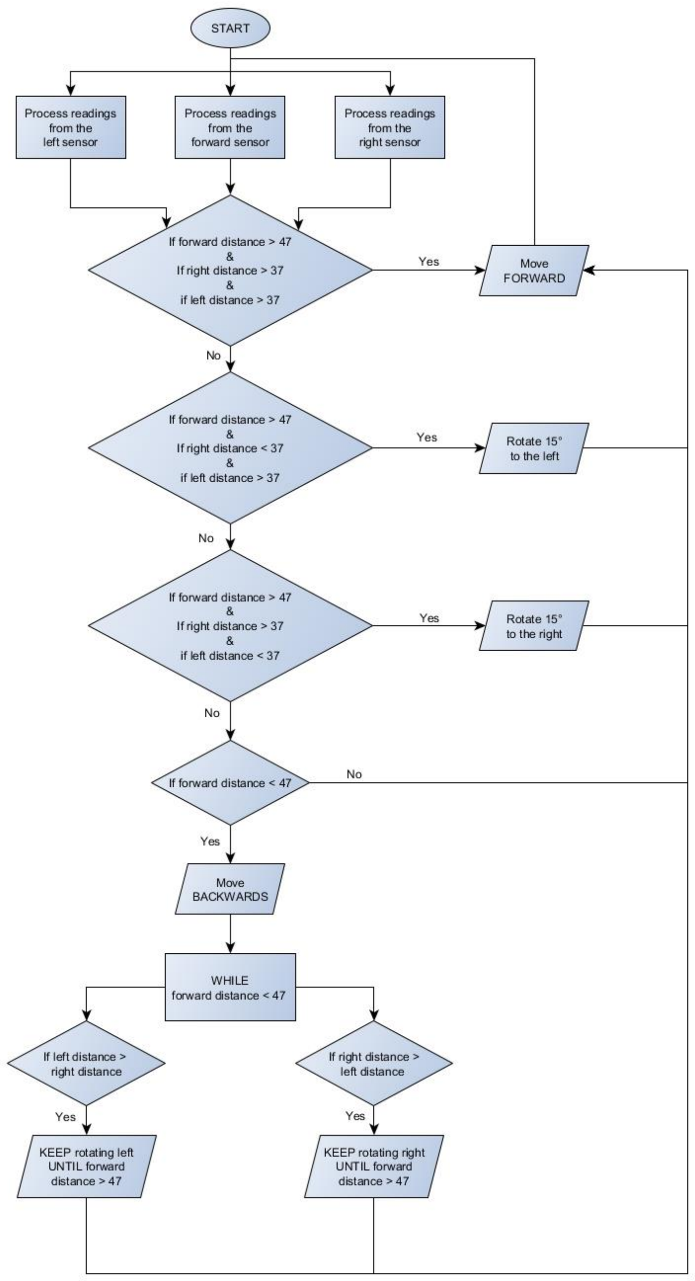

START
Process readings from the left sensor
Process readings from the forward sensor
Process readings from the right sensor
If forward distance > 47 & If right distance > 37 & if left distance > 37
Yes
Move FORWARD
No
If forward distance > 47 & If right distance < 37 & if left distance > 37
Yes
Rotate 15° to the left
No
If forward distance > 47 & If right distance > 37 & if left distance < 37
Yes
Rotate 15° to the right
No
If forward distance < 47
No
Yes
Move BACKWARDS
WHILE forward distance < 47
If left distance > right distance
If right distance > left distance
Yes
Yes
KEEP rotating left UNTIL forward distance > 47
KEEP rotating right UNTIL forward distance > 47

Figure 9-1: Automated Variable Rotations Algorithm Concept

9.1 Test Run - Experiment

The final test run will operate under the same set of conditions from previous experiments. Objects will be positioned in the test area to observe how the robot interacts with the objects.

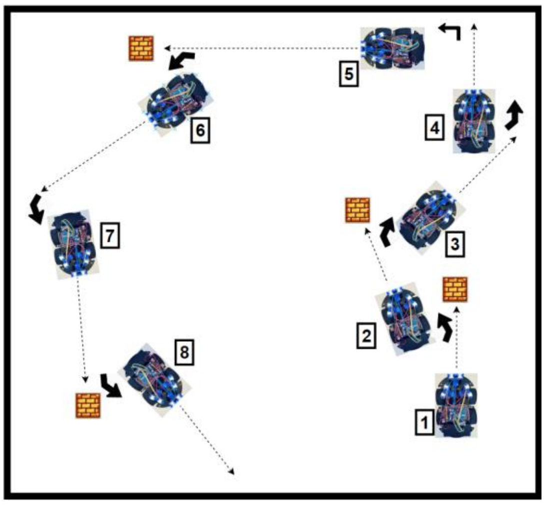


Figure 9-2: Automated Variable Rotations Algorithm Experiment Illustration

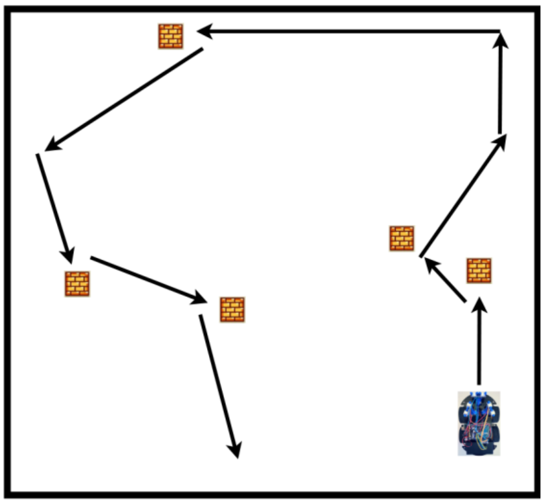


Figure 9-3: Path overview taken by robot during test run

Figures 9-2 and 9-3 demonstrate the robot's maneuvering abilities around obstacles placed in the testing area. This test should be able to confirm the efficiency of the final algorithm. The experiment was carried out several times more compared to other algorithms to ensure the competence and the results required from the robot.

9.2 Results & Analysis

By observing the test run results illustrations, a significant improvement can be seen from comparisons with section 8.1. As the robot gets closer to and is obstructed by an object, the robot makes a very tight turn allowing it to just get past the object without doing any over-turns. In positions 2, 3 and 4 of figure 9-2, the robot is seen to carry out small angular rotations that are just enough to avoid the object ahead. When compared to Fixed Rotations Algorithm, for the same position, the robot made wider rotations more than what was necessary to avoid the object. The tighter turns and accurate rotations realized from Automated Variable Rotations algorithm allow for a much fluid, natural and smooth robot movement. The efficiency of the robot is increased greatly as it only carries out actions needed to fulfil its main objective, avoiding obstacles. The combination of all previous algorithms in sections 6, 7 and 8 together has allowed the development and better sophistication of the final algorithm. The fusing of different methodologies and strategies together help to achieve a greater performance by utilising their individual advantages while eliminating their individual drawbacks. Magnified performance benefits employing this philosophy has been demonstrated in figure 2-20 in [19].

In addition to the advantages mentioned reaped by employing this algorithm, there are a couple more factors that Automated Variable Rotations is better at dealing with compared to all other previous algorithms. By carrying out numerous test-runs and experiments for the algorithms, few factors were observed that affect the rotational movement of the robot. These are:

• **Floor texture and resistance**: One of the most influential factors that affected the rotational speed of the robot is the floor texture and resistance the robot is tested on. From a mechanical perspective, the robot makes effective rotations by turning the wheels on either side in the opposite direction. For example, for a right turn, the robot turns the left set wheels forward while the right set wheels backward. This cause the robot to a rotation to the right and vice versa. The rotation ensued causes the wheels to be dragged across the surface, making floor frictions and resistance a significant factor when it comes to rotational movements. A smooth surface with low resistance like a wooden or marble flooring would have very minimal resistance and friction compared to a much rougher surface like a carpet or a rug. Two surfaces were mainly used for testing of the algorithms, wooden flooring and carpet. Both of which have very different textures. When running experiments on both surfaces, it has been noticed that for a specific period where the motors are powered ON, the angle rotation on the wooden surface was wider compared to the rotation on the carpet. This is due to the fact that if the motors were powered on for instance for 100ms, the robot had less resistance to work with and therefore and an easier time to skid across the surface creating a wider rotation compared to a rotation on a carpet with much higher resistance resulting in a smaller rotation. This variance in operation results created problems with the Fixed Rotations algorithms as the rotations were uncertain and varied widely between different surfaces. A code developed to mainly operate on a carpet surface would prove to be an overshoot for smoother surfaces where the angles can go wider, making the wider unnecessary rotations an even bigger problem. This issue has been mitigated significantly with the implementation of the Automated Variable Rotations algorithm as it is not dependant on a certain surface for operation. Since there is rapid monitoring of the objects in the final algorithm, that dictates how long the rotations should go on for. This means that regardless of the surface resistance or friction, the robot should be able to perform refined rotations based on the object’s position. This realizes the greater benefit of the final algorithm as it is not restricted for operation on a specific surface and is free to operate on different kinds of surfaces while the Fixed Rotations Algorithm produced different and varying results when operating on different surfaces. Automated Variable Rotations algorithms allow the robot to have a great flexibility and higher level of sophistication.

• **Battery charge levels**: The second most influential factor that affected the rotational speeds of the robot where the battery charge levels. As experiments went on for several hours on a single charger, it has been noticed that the performance of the robot started to vary as the battery levels dropped lower. The batteries are used to power everything on the robot including the motor driver, DC motors, microcontroller, ultrasonic sensors and Bluetooth module. DC motors are by far the most power consuming components on the robot which requires sufficient power to run properly. However, as the power levels dropped low, the robot would start to run slower thus resulting in difficulties for robot movement. This effect could be observed to be stronger on rougher surfaces as greater DC motors power is required to operate on a rougher surface compared to a smoother one. Furthermore, the weakened and slow robot functions due to low battery levels meant that the rotational speeds were also affect greatly. For a specific surface, rotational speeds at full battery charge levels were faster and more agile compared to low charge levels. This affected the performance of the robot greatly especially under Fixed Rotations algorithm. However, the implementation of the Automated Variable Rotations helped reducing these side effects due to inherit advantages of the algorithms mentioned earlier.

• **4-wheel drive**: One of the main factors that resulted in varying performances on different surfaces was employing a 4-wheel drive in the robot body. As shown before, the robot is driven by 4 DC motors to help the navigate the robot in rough environments at relatively high speeds. The presence of 4WD allows the robots to traverse on uneven rough surfaces with mild ease. However, with the presence of 4WD, it multiplies the robot's friction with surfaces significantly. A different approach is the implementation of 2WD with a castor wheel in front that is free to rotate in 360° without a motor. This strategy is seen to be utilised in various robot examples including the vacuum cleaning robot in [19]. A castor wheel allows for easier rotations which are not hindered as much by floor resistance compared to 4WD.

Tests have been carried out to measure the required operation time for the DC motors to make main key rotational movements on two different surfaces, carpet and wooden flooring. The tables below demonstrate the various test results. Tests were carried out at same battery levels.

Carpet	
Angle	Operation Time
90°	850ms
180°	1300ms
270°	1800ms
360°	2500ms

Table 9-1: Different angles required operation times on Carpets

Wooden Flooring	
Angle	Operation Time
90°	400ms
180°	800ms
270°	1500ms
360°	2100ms

Table 9-2: Different angles required operation times on Wooden Flooring

From the results observed in the tables, it can be seen how significant the surface resistance and the friction it provides against the wheels during rotations. Tests carried on a carpet produced widely varying results due to unpredictable random behaviour. Due to the high friction on the carpet, for wider angle rotations, there were instances where the wheels would start "skipping" or jumping on the floor as it cannot maintain a constant contact with the floor. Numerous tests were carried out and averages were taken to arrive at the most accurate results. On the other hand, operation on wooden flooring which is a much smoother surface seemed to give a much better performance for the robot. There were no wheel skipping observed and less severe varying results. Comparing both results, it can be seen that all angular rotations require a less operation time of 400-500ms which is quite significant. There is where Automated Variable Rotations advantages come in as they allow for freedom of operations on different surface textures and is not hindered to a specific environment, allowing for greater flexibility and adaptability.

10 BLUETOOTH COMMUNICATIONS & MANUAL CONTROL

An additional requirement for robot functions is to allow the ability for external communications between the robot and separate users/devices. This is to provide the ability for user manual control to override the system in case of an emergency or a component failure. Wireless communications will also allow for additional tools like automated hardware testing will be discussed ahead. There is more room for adding in additional future requirements as needed with proper programming beforehand. Wireless control will be done through an android application which provide programmable "buttons" free to be assigned for various functions as needed. A speed of 9600 bits per second (bps) is used which is a common value for non-critical applications. The Arduino is set to intercept incoming data from the external application in the form of characters. Each button pressed in the application has a designated character to it. When the microcontroller detects a specific character, it is programmed to understand the required functions associated with that character and thus allows for an action to be taken. Figure 10-1 shows the application interface used to interact with the robot. There is an abundance of buttons available to be mapped to different functions to fit project requirements and possible future needs. Directional buttons are used to control the robot movements while the shaped buttons are used to perform specific functions. Figure 10-2 shows the characters associated with each button press. For left, forward, right and backward movements, letters 'l', 'f', 'r' and 'b' have been set respectively. Therefore, the microcontroller can be programmed to expect these characters and carry out respective movements when these characters are received from the application. Furthermore, the shaped buttons are used to perform more additional binary functions. The square, triangular and cross buttons have been assigned 's', 'a' and 't' respectively. The square button is used to bring the robot to a complete stop and interrupting any movements. The triangular button is used to switch between manual and autonomous navigation mode while the cross button is used to test for hardware testing which will be discussed in the next section. Any press on the direction buttons performs two functions, first it carries out the movement set to that specific button. Second, it indicates that the robot is now in manual control, disabling autonomous mode. To go back to autonomous navigation mode, the triangular button is pressed, disabling manual control.

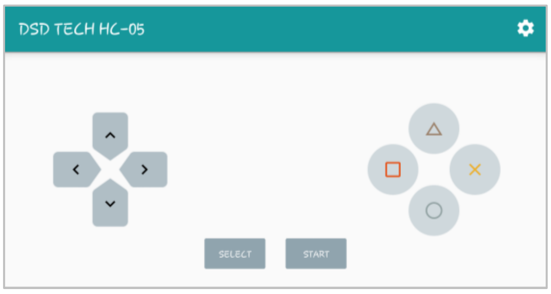
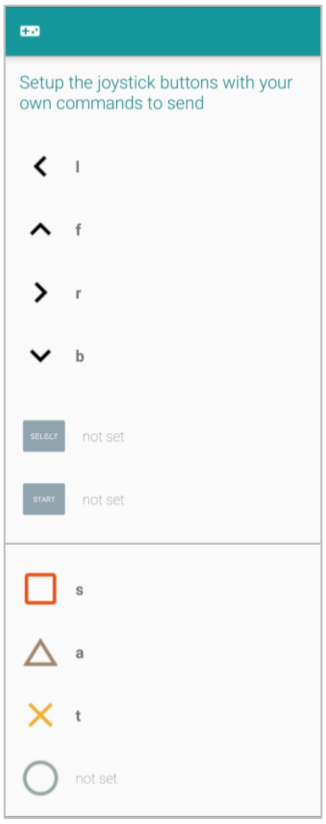


Figure 10-1: Wireless Control Application Interface


Setup the joystick buttons with your own commands to send

<	l
^	f
>	r
v	b
SELECT	not set
START	not set
□	s
△	a
✕	t
○	not set

Figure 10-2: Buttons Assignments Configuration

11 AUTOMATED HARDWARE TESTING

Following the final development of the core functions of the robot, it was required to develop an automated method to detect loose wire or connections to ensure that all components are connected properly, and they are performing their required functions. Figure 4-5 shows a significant amount of connections and wires that could be easily snatched out of place and thus resulting in a faulty performance. Due to the density of the writes in a small area, it might be difficult to spot the unconnected wires manually and therefore an automated method would be more beneficial to implement. The most crucial components which also account for the most connections on the robot are the Ultrasonic sensors. An unconnected wire to/from the sensor means that the sensor isn't working properly and thus isn't sending accurate data, if at all, to the microcontroller. Malfunctioning sensors, especially the forward sensor, will result in a complete failure and cause the robot to crash. An algorithm has been developed to detect data received from the sensors, an unconnected sensor will send no data to the microcontroller while a connected one will. If the algorithm picks up no data from the sensors when it should be receiving data, the LED will constantly blink to indicate faulty connections. However, if the algorithm does receive data from the sensors, then the LED remains on permanently. To access testing mode, the user can press the cross button in the application interface to initiate the test mode shown in figure 10-1. Figure 11-1 and 11-2 display the LED when a faulty connection is detected or good condition otherwise. Moreover, a complete block diagram for the design and concept behind the robot is demonstrated in figure 11-2. It shows the various components and their links to each other and how they operate to bring the system together.

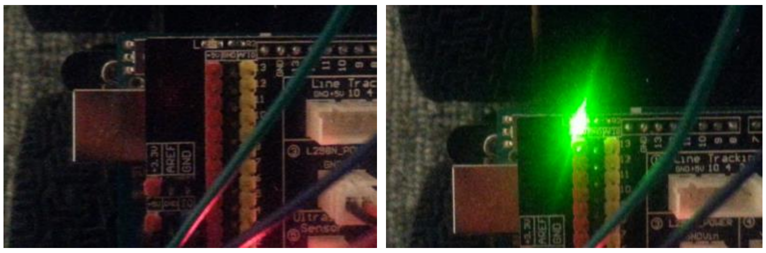


Figure 11-1: Blinking LED indicating a connection failure OR permanently ON otherwise

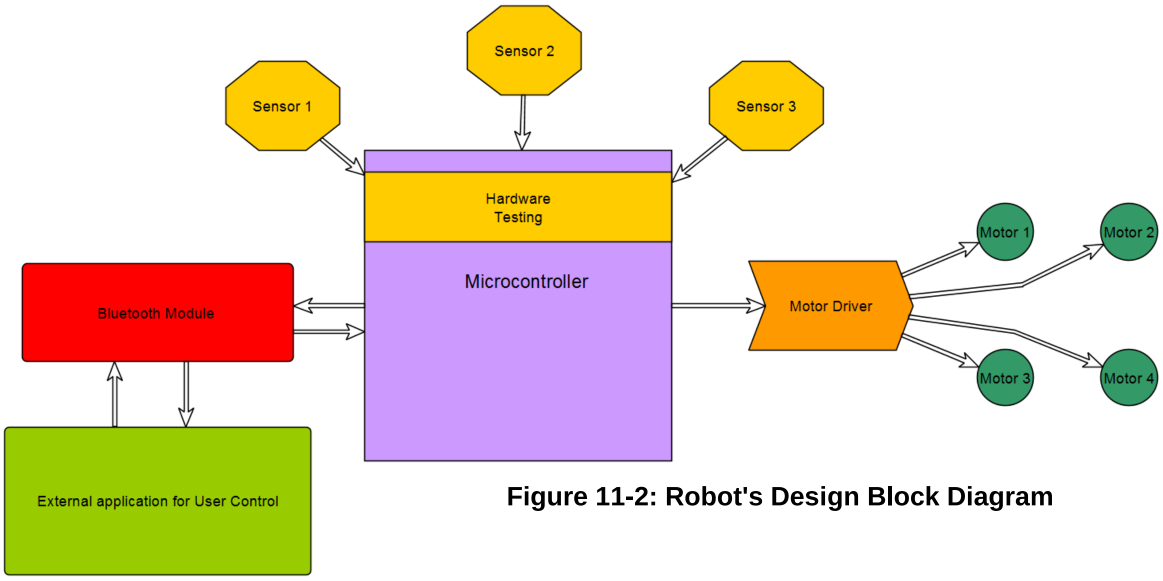


Figure 11-2: Robot's Design Block Diagram

12 FINAL CONCLUSIONS

This project was set out to design, build and develop an autonomous robot that is able to navigate itself in any environment around obstacles and obstructions in its path. The robot was planned to have a high accuracy and precision of obstacle avoidance and demonstrate a fluid and smooth operation. A major part of the project is the intensive and extensive research and investigation into a variety of different navigation algorithms and developing an understanding regarding the current state-of-the-art in this field.

Several navigation strategies have been selected for further investigation with varying levels of complexity and sophistication. These algorithms were Binary Movement, Wall-Follow, Fixed Rotations and Automated Variable Rotations. Binary Movement algorithm was the first strategy investigated which involved fundamental concepts when it comes to carrying out basic robot movements in a (+) sign. Due its simplicity, it provided a very basic and straightforward maneuvering abilities that hindered a lot of the potential expected from the robot. It was lacking severely in smooth and natural movement and was inefficient and slow while navigating around objects. However, it laid the foundations for other algorithms to follow. Wall-Follow provided much better results in terms of movement and navigation. It provided better sophistication and rotational movement compared to Binary Movement, and was not as restricted and rigid in its movement. However, Wall-Follow displayed obvious problems that were deemed not suitable for the project. Wall-Follow was heavily dependant on the constant presence of a wall for navigation reference. This was a major liability as the robot functions would fail being far away from a wall; which also meant that large portions of the environment the robot was tested in, specifically towards the centre remained unexplored. A better navigation algorithm was needed to merge the positives of Binary Movement and Wall-Follow while eliminating their drawbacks. The employment of Fixed Rotations algorithms showed very promising results for the performance of the robot. Navigation around objects was considerably more accurate compared to Binary Movement. The algorithm also provided small angular adjustments for better turning as opposed to sharp 90 degrees. In addition, the algorithm was not restricted to the presence of a wall for reference like Wall-Follow as therefore was free to explore anywhere. On the contrary, the algorithm still indicated flaws within its design. Fixed angle rotations were utilised that inhibited its movement and reaction to objects. Consistent fixed angle rotations were made constantly in reaction to objects regardless for their positioning in respect to the robot’s body. This resulted in some rotations being an overshoot and ended way beyond what was required to avoid the object. These flaws were magnified dramatically under several external factors like floor resistance and battery charge levels. A more proficient algorithm was therefore required to better circumvent some of these drawbacks. Automated Variable Rotations showcased an intelligent and sophisticated approach to navigating around objects that is not as affected heavily by floor resistance and battery charge levels. Due to the constant rapid monitoring of objects and feedback, rotations were only carried out for as long as needed to clear the object safely. Angles overshooting were mostly eliminated, and the robot demonstrated a much more efficient and quick navigation routes around its surroundings. The use of Automated Variable Rotations has allowed for a much smoother and fluid robot movement and the freedom of operation in various environments and different floor textures.

Additional efforts allocated to the project went into hardware selections and comparisons. Mainly, close comparisons were made between different microcontrollers and sensors for distance measurement. Ultrasonic sensors were selected as they provided a greater deal of advantages and suitability compared to their counterparts. Five Ultrasonic sensors were initially planned to be used on the robot, forward, backward, left-right and cliff sensors. After long development period, the backward and cliff sensors were eliminated from the project and were not implemented. The backward sensor was tested for implementation and was concluded to be redundant as it did not provide any additional beneficial information to the algorithm to help its navigation. When a reverse movement was carried out for the robot, it was mainly during a miniscule readjustment for a wide angle rotation and was never for extended reverse movement. Furthermore, the cliff sensor failed to be implemented due to the way Ultrasonic sensors work and their incompatibility with the project's design. It was found that when the Cliff sensor was placed at an angle looking forward to detect edges or steps, all transmitted sound waves reflected off at an angle away from the receiver and were not detected back. This provided no information to the sensor about the presence of any edges.

Reflecting back on the initial project aims and objectives, in addition to developing an autonomous robot that navigates around obstacles and the constant development and improvements to navigation algorithms capabilities, it was also required to introduce means of wireless communications for the robot to allow wireless control. The robot was programmed to allow manual control by a user operating with an application to manually control the robot, overriding the autonomous feature of the robot. Additional features were also added that allowed for automatic detection of faulty connections and wires to ensure hardware functionalities. The project proved to be successful in the end with the development of a well-round sophisticated robot that employs an intelligent algorithm for efficient navigation and obstacle avoidance. The constant improvements and mixing of different navigation algorithms meant that the final version was a robust and a more complete methodology that provided lots of advantages with minimal drawbacks. The robot demonstrated impressive performance when navigating around objectives autonomously with efficient rotations. Moreover, the wireless communications added a necessary feature to the robot to allow for wireless manual control and allowed for the possibility of additional features that may be required in the future.

Despite the success, the following section will touch on some of the improvements and additions that could be made to deal with problems faced during the development of the robot. Possible additions to improve the complexity and sophistication of the robot and the employment of completely different approach for navigation in different applications.

13 FURTHER DEVELOPMENTS

There are several ways this project could be further developed to allow for additional features to help arrive at an even better performance. A cliff sensor would be a valuable addition to the robot to allow it stay clear of edges and steps. Implementing a cliff sensor in this project failed due to the mechanical restriction of the robot chassis as well as the nature of the Ultrasonic sensor. To help accommodate a cliff sensor, mechanical adjustments could be made to the chassis to allow for the instillation of a sensor perpendicular to the ground so that transmitted waves are not reflected away. Beside mechanical adjustments, a different sensor type could be selected altogether which is not hindered by the outward reflection of transmitted waves off the ground. Moreover, during the testing and development phases of the robot, it was found that Ultrasonic sensors inherently have blind spots during operation. If an Ultrasonic sensor transmits a sound wave that reflects off an object at a certain angle, all sound waves are reflected away from the receiver and are not detected back. This causes partial performance failure as an object to the side of the robot is not detected and may result in a collision. To overcome this issue, a bigger array of sensors could be used to overcome the blind spots of the sensors and allow for proper detection of objects at all angles. This will of course also entail the expansion of the robot chassis to facilitate the addition of more sensors.

Furthermore, an upgrade to the central core of the robot which is the microcontroller is a very valuable upgrade to the expansion of the robot. As discussed extensively in section 3, a Raspberry Pi is vastly superior to an Arduino. In the case where higher computing and processing power is required, a Raspberry Pi would be a worthwhile investment in terms of memory, processor speed, RAM etc. In addition, they come equipped with media options allowing for media processing, HDMI ports as well as in-built Bluetooth and Wi-Fi modules. Transition of the code from an Arduino to a Raspberry Pi is possible subject to proper adjustment and modification to the code. Certain keywords and functions are defined within the Arduino IDE which may not be recognizable by Raspberry Pi environment.

With the additional power that comes with a Raspberry Pi, utilising machine learning and AI is a prospect that can widely open more options and possibilities for what the robot can accomplish. While complex and intricate, machine learning can allow the robot to carry out more complex functions and tasks with greater independence rather on than relying on pre-defined sets and conditions. AI, deep learning and machine learning are very rapidly increasing fields of technology, specifically in robotics. The implementation of AI could open possibilities for advanced additions including navigation cameras and VSLAM technology. The robot could be trained to recognize locations and rooms to help it increase its efficiency and precision.

Additional tools to help controlling the robot’s movement include optical encoders to monitor wheel speeds, rotations and angle deviations and errors. Compasses and GPS could aid in the navigation of the robot as well.

14 PROJECT MANAGEMENT

One of the main factors pertinent to the successful delivery of any project is smart and realistic planning. There is a common say that goes by “If you fail to plan, you plan to fail.” Efforts have been paid at the initial stage of the project to set out a clear, realistic and obtainable plan to be adhered by for the duration of the project.

Throughout the duration of the project, the project plan initially set up was generally accurate and was followed to a great extent. However, as with any other project, there are inevitable unforeseen hurdles that show up during the project development which need to be dealt with. These will ultimately affect the tasks to follow. On the other hand, there are tasks that were completed faster than anticipated. One of the main crucial tasks that were expected to take a considerable amount of time but ended up being otherwise was the manufacturing of 3D printed sensors mount. Due to the uncertainty regarding the availability of the workshop and the lab technician as well as the required time for manufacturing, 2 weeks were allocated for the task. However, due to how early the task was tackled at the start of the project, the manufacturing of the sensor mounts ended up taking around 3 days.

The majority of the tasks were completed relatively close to their originally planned periods with few exceptions. Disruption happened towards the end of the project period as it was required to develop additional functionalities that were not planned for initially. Moreover, a period of 3 weeks approximately from late January to mid-February 2019 were spent researching the implementation of additional more sophisticated functions and features. This period push the progress of report writing significantly towards early March from the originally start period planned early February. As a result of the significant delay, work hours were increased significantly to around 8-10 hours per day to help completing the final report on time. In addition, there was a significant work load and time pressure in March as there were four additional assignments due submission for other modules within 10 days period of each other. The addition of the second slippage period proved to be extremely useful as it allowed for the write-up of the report which would have otherwise impossible if other tasks were allocated for that period. In retrospect, delaying the report writing was not a wise decision to attempt adding more additional features. A better plan for the future could have included more regard to the module assignments anticipated in March and integrating them in the project plan.

Nonetheless, consistent scheduled weekly meetings were held with the project supervisor. Dr. Zoe Jeffrey updating her on project progress and any issues encountered. Ideas and thoughts were exchanged to help direct the project in the right direction that help successfully delivering the project. Comparisons between the initial Gantt chart and the final Gantt chart could be found in the appendices along with detailed tabulated information of task details and key dates.

REFERENCES

[1] Mars.nasa.gov. (2019). *Mars Exploration Rovers*. [online] Available at: https://mars.nasa.gov/mer/ [Accessed 26 Mar. 2019].

[2] Nelson, T. (2018). *What Is a Robot? Do you know how to recognize a robot?*. [online] Lifewire. Available at: https://www.lifewire.com/what-is-a-robot-4148364 [Accessed 26 Mar. 2019].

[3] Guizzo, E. (n.d.). *What Is a Robot? - ROBOTS: Your Guide to the World of Robotics*. [online] Robots.ieee.org. Available at: https://robots.ieee.org/learn/ [Accessed 26 Mar. 2019].

[4] Idahoptv.org. (n.d.). *Simple Machines: Facts (Science Trek: Idaho Public Television)*. [online] Available at: http://idahoptv.org/sciencetrek/topics/simple_machines/facts.cfm [Accessed 26 Mar. 2019].

[5] VEX Robotics. (n.d.). *Simple Machines & Motions - The Six Types of Simple Machines*. [online] Available at: https://www.vexrobotics.com/vexiq/education/iq-curriculum/simple-machines-and-motion/six-types-of-simple-machines [Accessed 26 Mar. 2019].

[6] Wells, P. (n.d.). *Issac Asimov's Three Laws of Robotics*. [online] Csl.mtu.edu. Available at: http://www.csl.mtu.edu/winter98/cs320/AI/pmwellsasimov.html [Accessed 26 Mar. 2019].

[7] Historyofinformation. (2015). *The First Electronic Autonomous Robots: the Origin of Social Robotics: HistoryofInformation.com*. [online] Available at: http://www.historyofinformation.com/detail.php?entryid=854 [Accessed 26 Mar. 2019].

[8] Robotics Online. (n.d.). *Unimate - The First Industrial Robot*. [online] Available at: https://www.robotics.org/joseph-engelberger/unimate.cfm [Accessed 26 Mar. 2019].

[9] Jackson, L. and Cracknell, R. (2018). *Road accident casualties in Britain and the world*. [online] Researchbriefings.parliament.uk. Available at: https://researchbriefings.parliament.uk/ResearchBriefing/Summary/CBP-7615 [Accessed 26 Mar. 2019].

[10] World Health Organization. (n.d.). *Road traffic deaths*. [online] Available at: https://www.who.int/gho/road_safety/mortality/en/ [Accessed 26 Mar. 2019].

[11] McCarthy, N. (2018). *The Tesla Model 3 Is Blowing Away The Competition*. [online] Statista Infographics. Available at: https://www.statista.com/chart/14921/the-tesla-model-3-is-blowing-away-the-competition/ [Accessed 26 Mar. 2019].

[12] Su, J. (2018). *Tesla Could Have Full Self-Driving Cars On The Road By 2019, Elon Musk Says*. [online] Forbes.com. Available at: https://www.forbes.com/sites/jeanbaptiste/2018/11/07/tesla-could-have-full-self-driving-cars-on-the-road-by-2019-elon-musk-says/ [Accessed 26 Mar. 2019]

[13] Mars.nasa.gov. (n.d.). *Opportunity Rover Updates*. [online] Available at: https://mars.nasa.gov/mer/mission/rover-status/opportunity/recent/all/ [Accessed 26 Mar. 2019]

[14] Statista. (n.d.). *Tesla Statistic and Facts*. [online] Available at: https://www.statista.com/topics/2086/tesla/ [Accessed 26 Mar. 2019]

[15] Woodford, C. (2018). *How do Roomba robot vacuum cleaners work?*. [online] Explain that Stuff. Available at: https://www.explainthatstuff.com/how-roomba-works.html [Accessed 26 Mar. 2019]

[16] Layton, J. (2017). *How Robot Vacuums Work*. [ebook] Howstuffworks. Available at: http://www.sci.brooklyn.cuny.edu/~sklar/teaching/f06/cis1.0/papers/roomba-howstuffworks.pdf [Accessed 26 Mar. 2019]

[17] Pala, M., Eraghi, N., López-Colino, F., Sanchez, A., de Castro, A. and Garrido, J. (2013). HCTNav: A Path Planning Algorithm for Low-Cost Autonomous Robot Navigation in Indoor Environments. *ISPRS International Journal of Geo-Information*, 2(3), pp.729-748.

[18] Assis, L., Soares, A., Coelho, C. and Van Baalen, J. (2016). An Evolutionary Algorithm for Autonomous Robot Navigation. *Procedia Computer Science*, 80, pp.2261-2265.

[19] Hasan, K., Nahid, A. and Reza, K. (2014). Path planning algorithm development for autonomous vacuum cleaner robots. *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*.

[20] Robotshop. (2013). *How to make a simple Autonomous Vehicle*. [online] Available at: https://www.robotshop.com/community/blog/show/how-to-make-a-simple-autonomous-vehicle [Accessed 26 Mar. 2019]

[21] F. Cherni, Y. Boutereaa, C. Rekik and N. Derbel, "Autonomous mobile robot navigation algorithm for planning collision-free path designed in dynamic environments," *2015 9th Jordanian International Electrical and Electronics Engineering Conference (JIEEEC)*, Amman, 2015, pp. 1-6.

[22] Fu, L., Sun, D. and Rilett, L. (2006). Heuristic shortest path algorithms for transportation applications: State of the art. *Computers & Operations Research*, 33(11), pp.3324-3343.

[23] Raspberry Pi. (n.d.). *Raspberry Pi 3 Model B - Raspberry Pi*. [online] Available at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Accessed 26 Mar. 2019]

[24] Store.arduino.cc. (n.d.). *Arduino Mega 2560 Rev3*. [online] Available at: https://store.arduino.cc/mega-2560-r3 [Accessed 26 Mar. 2019]

[25] Electronics Hub. (2015). *IR (Infrared) Obstacle Detection Sensor Circuit*. [online] Available at: https://www.electronicshub.org/ir-sensor/ [Accessed 26 Mar. 2019]

[26] Burnett, R. (2018). *Understanding How Ultrasonic Sensors Work*. [online] MaxBotix Inc. Available at: https://www.maxbotix.com/articles/how-ultrasonic-sensors-work.htm [Accessed 26 Mar. 2019]

[27] Ultrasonic Ranging Module HC - SR04 Datasheet. (n.d.). [ebook] Elec Freaks. Available at: https://www.mouser.com/ds/2/813/HCSR04-1022824.pdf [Accessed 26 Mar. 2019]

[28] HC-05 -Bluetooth to Serial Port Module Datasheet. (n.d.). [ebook] Itead Studio. Available at: http://www.electronicaestudio.com/docs/istd016A.pdf [Accessed 26 Mar. 2019]

[29] Nedelkovski, D. (2017). *Arduino DC Motor Control Tutorial - L298N* | *PWM* | *H-Bridge - HowToMechatronics*. [online] HowToMechatronics. Available at: https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/ [Accessed 26 Mar. 2019]

[30] Ducros, A. (2017). *Fixation articulée pour HC-SR04 by AlainDucros*. [online] Thingiverse.com. Available at: https://www.thingiverse.com/thing:2592461/files [Accessed 26 Mar. 2019]

[31] Elegoo.com. (n.d.). *Elegoo Industries,Ingenious & fun DIY electronics and kits*. [online] Available at: https://www.elegoo.com/ [Accessed 26 Mar. 2019]

BIBLIOGRAPHY

Arkin, R. (2009). *Governing Lethal Behavior in Autonomous Robots*. 1st ed. Boca Raton, Fla.: Chapman & Hall/CRC.

Reif, J. and Wang, H. (1999). Social potential fields: A distributed behavioral control for autonomous robots. *Robotics and Autonomous Systems*, 27(3), pp.171-194.

R. G. Simmons, "Structured control for autonomous robots," in *IEEE Transactions on Robotics and Automation*, vol. 10, no. 1, pp. 34-43, Feb. 1994.

Dautenhahn, K. (1995). Getting to know each other—Artificial social intelligence for autonomous robots. *Robotics and Autonomous Systems*, 16(2-4), pp.333-356.

APPENDIX A

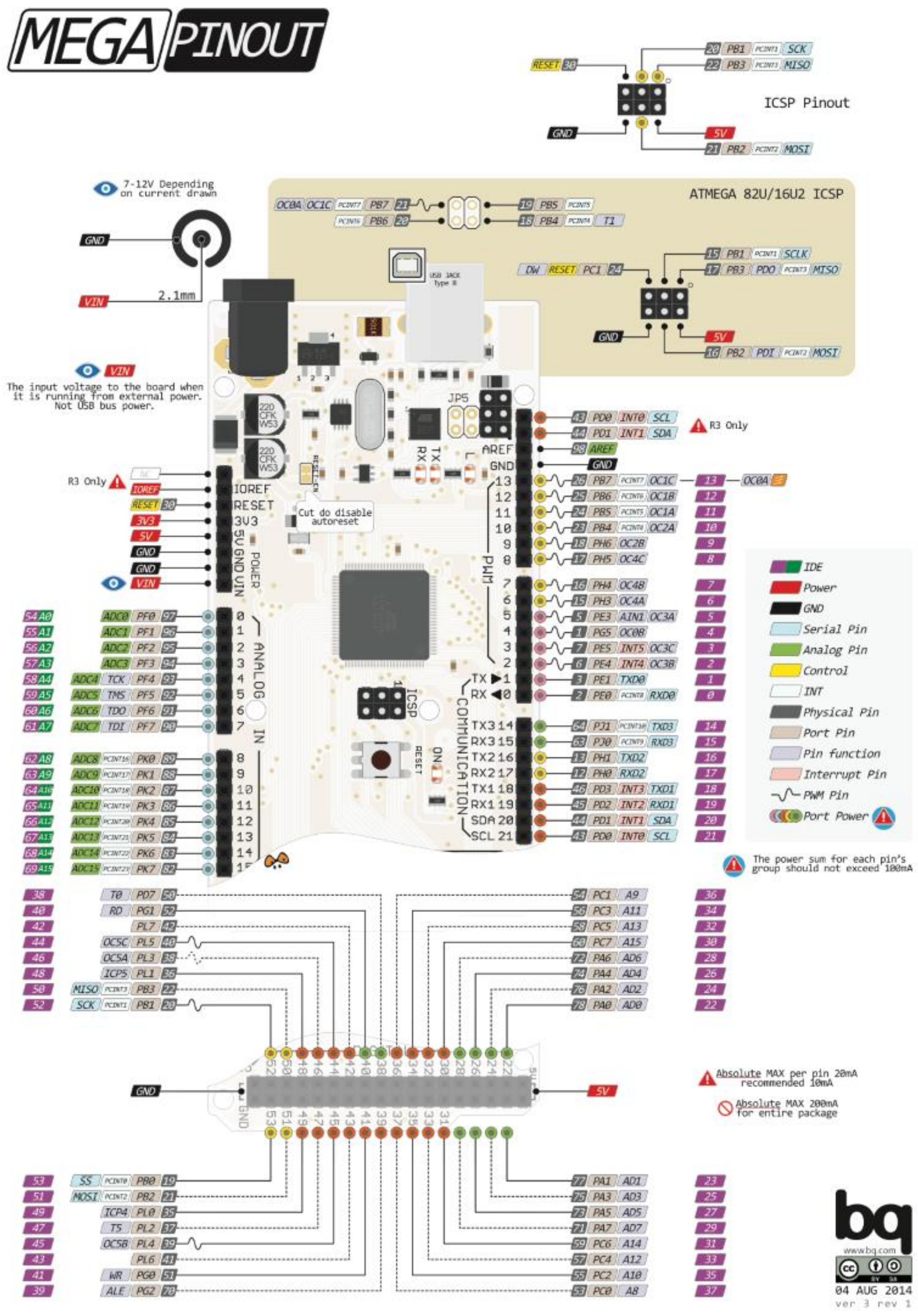
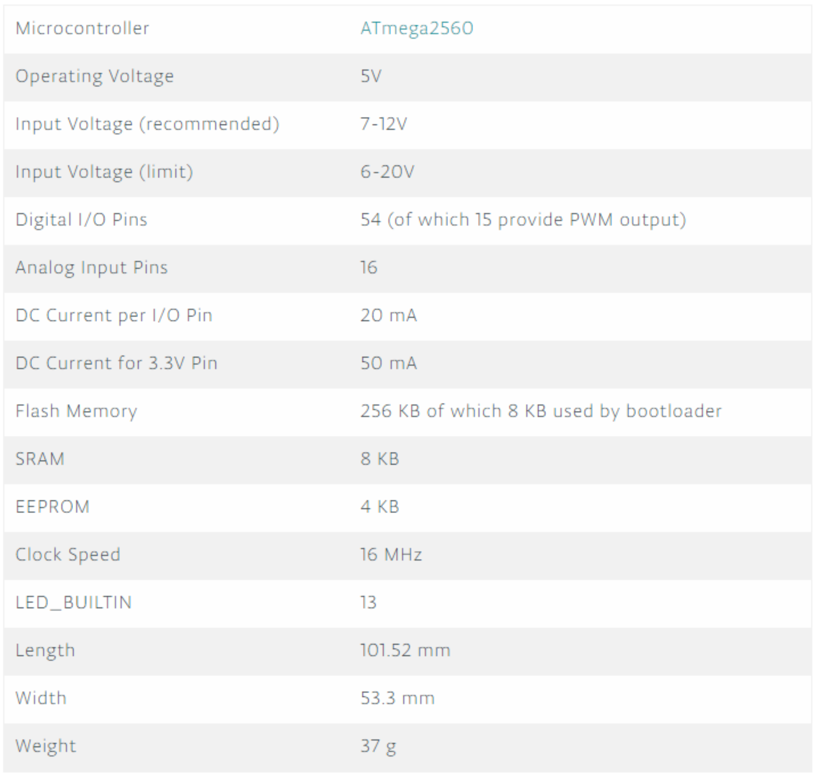
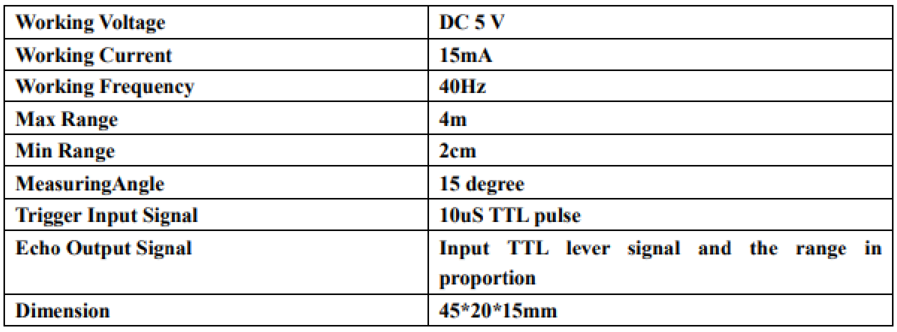

MEGA PINOUT
ICSP Pinout
ATMEGA 82U/16U2 ICSP
7-12V Depending on current drawn
2.1mm
The input voltage to the board when it is running from external power. Not USB bus power.
R3 Only
Cut do disable autoreset
IDE
Power
GND
Serial Pin
Analog Pin
Control
INT
Physical Pin
Port Pin
Pin function
Interrupt Pin
PWM Pin
Port Power
The power sum for each pin's group should not exceed 100mA
Absolute MAX per pin 20mA recommended 10mA
Absolute MAX 200mA for entire package
04 AUG 2014
ver 3 rev 1

Figure A-1: Arduino Mega 2560 Pinout [24]



Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Figure A-2: Arduino Mega 2560 Technical Specs [24]



Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Figure A-3: Ultrasonic Sensor HC-SR04 Technical Specs

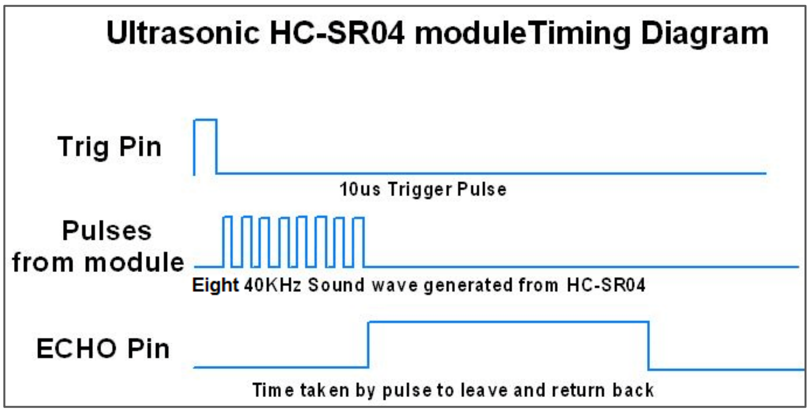
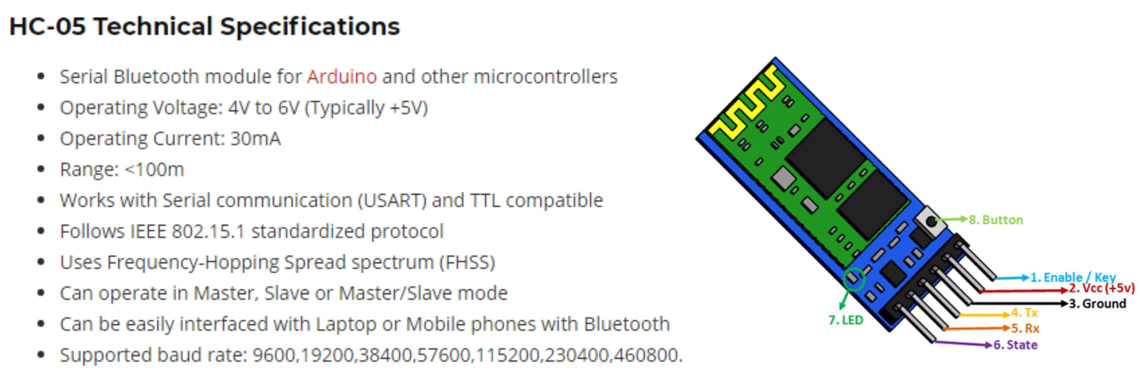

Ultrasonic HC-SR04 moduleTiming Diagram
Trig Pin
10us Trigger Pulse
Pulses from module
Eight 40KHz Sound wave generated from HC-SR04
ECHO Pin
Time taken by pulse to leave and return back

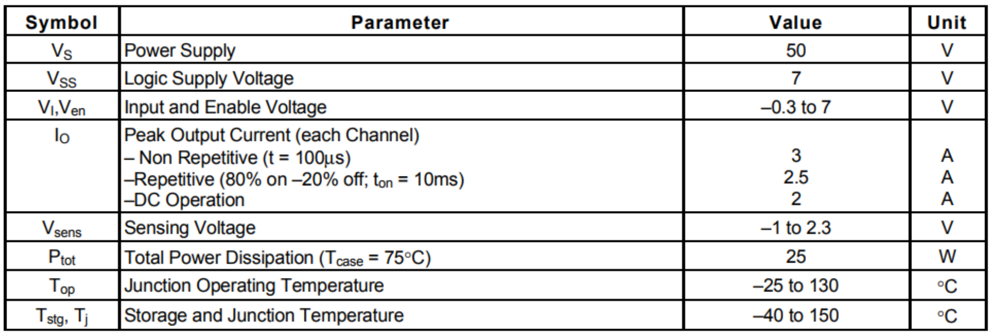
Figure A-4: Ultrasonic Sensor Timing Diagram [27]



HC-05 Technical Specifications

- Serial Bluetooth module for Arduino and other microcontrollers
- Operating Voltage: 4V to 6V (Typically +5V)
- Operating Current: 30mA
- Range: <100m
- Works with Serial communication (USART) and TTL compatible
- Follows IEEE 802.15.1 standardized protocol
- Uses Frequency-Hopping Spread spectrum (FHSS)
- Can operate in Master, Slave or Master/Slave mode
- Can be easily interfaced with Laptop or Mobile phones with Bluetooth
- Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.

Figure A-5: Bluetooth Module HC-05 Technical Specifications [28]



Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

Figure A-6: Motor Driver L298N Technical Specifications [29]

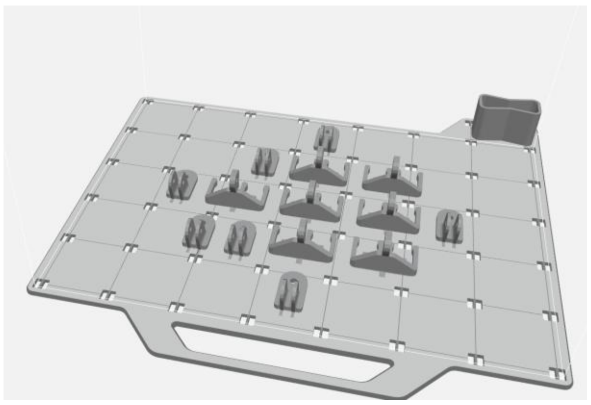


Figure A-7: 3D Printed Sensors Mount on CAD

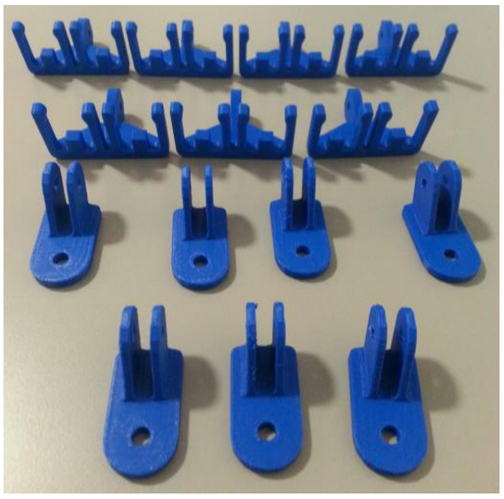


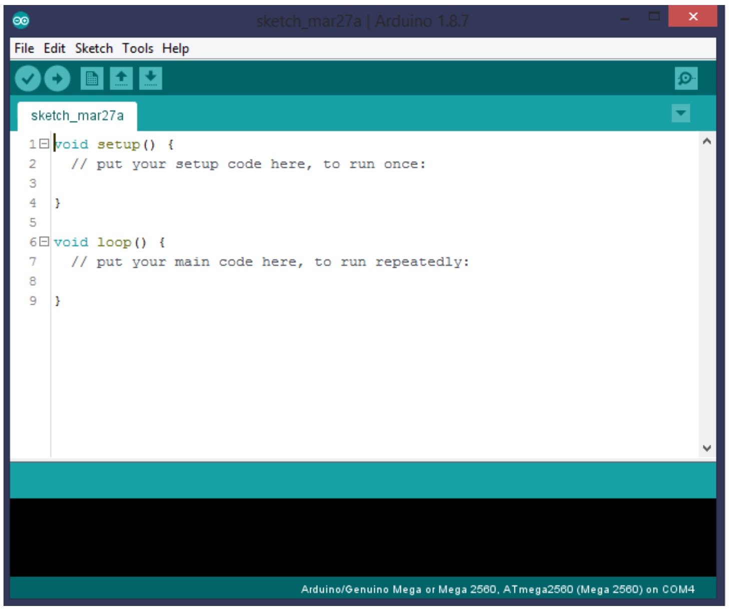
Figure A-8: Final Printed Sensors Mount

APPENDIX B

Getting started with an Arduino – how to run and operate.

In order to run an Arduino microcontroller, several steps need to be taken first to allow the computer to communicate with the microcontroller. The steps required can be summarized as:

1. Download the Arduino IDE – The Arduino IDE allows the computer to communicate with the microcontroller. This can be downloaded directly from their website [24] under the software tab.
2. Connect the microcontroller – The next step after installation is to connect the microcontroller board to the computer using a USB cable.
3. Setting up the IDE – Following the installation of the IDE, the IDE now needs to be configured to run the specific microcontroller at hand. For the purpose of this report, the microcontroller is Mega 2560.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM4

Figure B-1: Arduino IDE

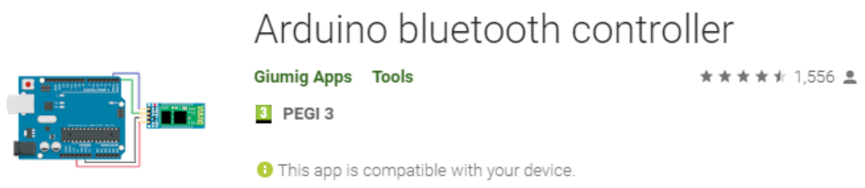
Figure B-1 displays the initial stage of the IDE after installation. To configure the IDE for communication with Mega 2560, select Tools > Board > Mega 2560. Again, Tools > Processor > Mega 2560. Once more, Tools > Port > COM3 or higher (usually COM4). To find out which serial port should be selected, disconnect the microcontroller and re-check the port menu. The entry that disappears should be the one to select, reconnect the microcontroller and select that serial port.

4. Enter Code – After the configuration of the IDE, the microcontroller is now ready to be programmed. There are two main loops in the code space that need to be there at all times for the code to be compiled successfully. The setup loop is used to insert parts of the codes that needs to be run once only, for example: pins initializations and setup. The second loops will contain the rest of the code which will run continuously. To import previously written code, File > Open a currently existing project.

IMPORTANT NOTE: Arduino is incapable of both communicating to a computer via USB and have the Bluetooth module connected at the same time. If the Bluetooth module is connected to the microcontroller and it is attempted to upload a code to the microcontroller at the same time, an error will be produced preventing the user from uploading the code. To resolve this issue, disconnect the wire connected to the Vcc (+5v) pin which is responsible for powering the module. This will prevent the microcontroller from communicating with the Bluetooth module. After uploading the required code to the microcontroller, the Bluetooth module can then be reconnected. To summarise, it should always be **either a USB connection or a Bluetooth module connected at a time, but not both.**

Bluetooth Control App:

There is a wide variety of applications available to download with different features to wirelessly communicate with Arduino. The application employed in this project can be found on Android's Play Store. The application is called "Arduino Bluetooth Controller" developed by Giumig Apps.


Arduino bluetooth controller

Giumig Apps Tools ★★★★½ 1,556

PEGI 3

This app is compatible with your device.

Figure 0-2: Bluetooth App on Play Store

Figure B-2 above displays the application as seen on the Play Store. The application can be downloaded directly by searching for the application on Play Store on Android or can be downloaded from a computer and transferred over using this link:

https://play.google.com/store/apps/details?id=com.giumig.apps.bluetoothserialmonitor

Algorithms Code – the final algorithm version B.4 has been commented. B.1, B.2 and B.3 mostly carry the same functional operations which can be understood by referring to B.4

B.1 Binary Movement Navigation Algorithm

```
//Amr Elshenawy MEng Individual Major Project
//Autonomous Self-Navigating Robot
//(+) SIGN BINARY MOVEMENT CONCEPT ALGORITHM
//3 Ultrasonic Sensors - Forward, Left, Right
//4WD Drive Chassis
//Bluetooth Module
//-----------------------------------------------------------------------------------------------------------------------
-----
//Sensors Intialisations, Flags and Variables

int ForwardEcho = A12;
int ForwardTrig = A13;
int RightEcho = A10;
int RightTrig = A11;
int LeftEcho = A8;
int LeftTrig = A9;

long FSduration, LSduration, RSduration;
float IFSdistance, FSdistance, ILSdistance, LSdistance, IRSdistance, RSdistance;

unsigned int sum = 0;
unsigned int sum2 = 0;
unsigned int sum3 = 0;
unsigned int Sample_Count = 0;
unsigned int Sample_Count2 = 0;
unsigned int Sample_Count3 = 0;
int Num_Samples = 10;
int KeepLooking = 250;

//-----------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------
-----
//Motor Intialisations

#define ENALeft 5
```

```
#define ENBRight 6
#define LeftForward 7     //IN1
#define RightBackward 9   //IN3
#define LeftBackward 8    //IN2
#define RightForward 11   //IN4

int FRota = 80;
int RRota = 200;

//------------------------------------------------------------------------------------------------------------------------------
-----
//------------------------------------------------------------------------------------------------------------------------------
-----
//Enumeration List

enum StatusMode
{
  IDLE,
  Autonomous,
  Manual
} func_mode = IDLE;

enum MovementMode
{
  STOP,
  FORWARD,
  BACKWARD,
  LEFT,
  RIGHT
} mov_mode = STOP;

//------------------------------------------------------------------------------------------------------------------------------
-----
//------------------------------------------------------------------------------------------------------------------------------
-----
//Operation Functions

void MoveForward()
{
  analogWrite(ENALeft, FRota);
  analogWrite(ENBRight, FRota);
  digitalWrite(RightForward, HIGH);
  digitalWrite(LeftForward, HIGH);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, LOW);
}

void MoveBackward()
{
  analogWrite(ENALeft, FRota);
  analogWrite(ENBRight, FRota);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, HIGH);
  digitalWrite(LeftBackward, HIGH);
}

void TurnLeft()
{
  analogWrite(ENALeft, RRota);
  analogWrite(ENBRight, RRota);
  digitalWrite(RightForward, HIGH);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, HIGH);
}

void TurnRight()
{
  analogWrite(ENALeft, RRota);
  analogWrite(ENBRight, RRota);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, HIGH);
  digitalWrite(RightBackward, HIGH);
  digitalWrite(LeftBackward, LOW);
}

void Stop()
{
  digitalWrite(ENALeft, LOW);
  digitalWrite(ENBRight, LOW);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, LOW);
}

float ForwardSensor()
{
  digitalWrite(ForwardTrig, LOW);
  delayMicroseconds(5);

  digitalWrite(ForwardTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(ForwardTrig, LOW);

  FSduration = pulseIn(ForwardEcho, HIGH);

  IFSdistance = FSduration * 0.0343 / 2;

  if(IFSdistance > 3 && IFSdistance < 300)
  {
    while(Sample_Count < Num_Samples)
    {
      sum += IFSdistance;
      Sample_Count++;
      delay(10);
    }
  }

  FSdistance = ((float)sum / (float)Num_Samples);

  Sample_Count = 0;
  sum = 0;

  Serial.print("Distance: ");
  Serial.println(FSdistance);

  return FSdistance;
}
```

```
float LeftSensor()
{
  digitalWrite(LeftTrig, LOW);
  delayMicroseconds(2);

  digitalWrite(LeftTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(LeftTrig, LOW);

  LSduration = pulseIn(LeftEcho, HIGH);

  ILSdistance = LSduration * 0.0343 / 2;

  if(ILSdistance > 3 && ILSdistance < 300)
  {
    while(Sample_Count2 < Num_Samples)
    {
      sum2 += ILSdistance;
      Sample_Count2++;
      delay(10);
    }
  }

  LSdistance = ((float)sum2 / (float)Num_Samples);

  Sample_Count2 = 0;
  sum2 = 0;

  Serial.print("Distance: ");
  Serial.println(LSdistance);

  return LSdistance;
}

float RightSensor()
{
  digitalWrite(RightTrig, LOW);
  delayMicroseconds(5);
  digitalWrite(RightTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(RightTrig, LOW);

  RSduration = pulseIn(RightEcho, HIGH);

  IRSdistance = RSduration * 0.0343 / 2;

  if(IRSdistance > 3 && IRSdistance < 300)
  {
    while(Sample_Count3 < Num_Samples)
    {
      sum3 += IRSdistance;
      Sample_Count3++;
      delay(10);
    }
  }

  RSdistance = ((float)sum3 / (float)Num_Samples);

  Sample_Count3 = 0;
  sum3 = 0;

  Serial.print("Distance: ");
  Serial.println(RSdistance);

  return RSdistance;
}

void AutonomousMode()
{
    if(func_mode == Autonomous)
    {
      ForwardSensor();
      RightSensor();
      LeftSensor();

      if(FSdistance > 40)
      {
        MoveForward();
      }
      else if(RSdistance > 40)
      {
        TurnRight();
        delay(850);
        ForwardSensor();
        if(FSdistance > 40)
        {
          MoveForward();
        }
      }
      else if(LSdistance > 40)
      {
        TurnLeft();
        delay(850);
        ForwardSensor();
        if(FSdistance > 40)
        {
          MoveForward();
        }
      }
      /*if(RSdistance < 40)
      {
        TurnLeft();
        delay(80);
      }
      if(LSdistance < 40)
      {
        TurnRight();
        delay(80);
      }
      /*else
      {
        MoveBackward();
        delay(800);
        MoveForward();
      }*/
    }
}
```

```
void BluetoothData()
{
 if(Serial.available())
 {
   switch(Serial.read())
    {
     case 'f': func_mode = Manual; mov_mode = FORWARD;      break;
     case 'b': func_mode = Manual; mov_mode = BACKWARD;     break;
     case 'l': func_mode = Manual; mov_mode = LEFT;         break;
     case 'r': func_mode = Manual; mov_mode = RIGHT;        break;
     case 's': func_mode = Manual; mov_mode = STOP;         break;
     case 'a': func_mode = Autonomous;
     default:                                               break;
    }
 }
}

void ManualMode()
{
 if(func_mode == Manual)
 {
   switch(mov_mode)
   {
    case FORWARD: MoveForward();    break;
    case BACKWARD: MoveBackward();  break;
    case LEFT: TurnLeft();          break;
    case RIGHT: TurnRight();        break;
    case STOP: Stop();              break;
    default:                        break;
   }
 }
}

//-----------------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------------
-----
```

```
void setup()
{
  pinMode(ENALeft, OUTPUT);
  pinMode(ENBRight, OUTPUT);
  pinMode(LeftForward, OUTPUT);
  pinMode(RightForward, OUTPUT);
  pinMode(LeftBackward, OUTPUT);
  pinMode(RightBackward, OUTPUT);

  pinMode(ForwardEcho, INPUT);
  pinMode(ForwardTrig, OUTPUT);
  pinMode(LeftEcho, INPUT);
  pinMode(LeftTrig, OUTPUT);
  pinMode(RightEcho, INPUT);
  pinMode(RightTrig, OUTPUT);

  Serial.begin(9600);
}

void loop()
{
  /*BluetoothData();
  ManualMode();
  AutonomousMode();*/
  ForwardSensor();
  RightSensor();
  LeftSensor();

  if(FSdistance > 40)
  {
    MoveForward();
  }
  if(FSdistance > 40 && RSdistance < 15)
  {
    TurnLeft();
    delay(700);
    ForwardSensor();
    if(FSdistance > 40)
    {
      MoveForward();
    }
  }
  if(FSdistance > 40 && LSdistance < 15)
  {
    TurnRight();
    delay(700);
    ForwardSensor();
    if(FSdistance > 40)
    {
      MoveForward();
    }
  }
  if(FSdistance < 40)
  {
    if(RSdistance > LSdistance)
    {
      TurnRight();
      delay(700);
      ForwardSensor();
      if(FSdistance > 40)
      {
        MoveForward();
      }
    }
    else
    {
      TurnLeft();
      delay(700);
      ForwardSensor();
      if(FSdistance > 40)
      {
        MoveForward();
      }
    }
  }
}
```

B.2 Wall-Follow Navigation Algorithm

```
//Amr Elshenawy MEng Individual Major Project
//Autonomous Self-Navigating Robot
//WALL-FOLLOW CONCEPT ALGORITHM
//3 Ultrasonic Sensors - Forward, Left, Right,
//4WD Drive Chassis
//Bluetooth Module
//-----------------------------------------------------------------------------------------------------------------------
-----
//Sensors Intialisations, Flags and Variables

int ForwardEcho = A12;
int ForwardTrig = A13;
int RightEcho = A10;
int RightTrig = A11;
int LeftEcho = A8;
int LeftTrig = A9;

long FSduration, LSduration, RSduration;
float IFSdistance, FSdistance, ILSdistance, LSdistance, IRSdistance, RSdistance;

unsigned int sum = 0;
unsigned int sum2 = 0;
unsigned int sum3 = 0;
unsigned int Sample_Count = 0;
unsigned int Sample_Count2 = 0;
unsigned int Sample_Count3 = 0;
int Num_Samples = 10;
int KeepLooking = 250;

//-----------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------
-----
//Motor Intialisations

#define ENALeft 5
#define ENBRight 6
#define LeftForward 7     //IN1
#define RightBackward 9   //IN3
```

```
#define LeftBackward 8    //IN2
#define RightForward 11   //IN4

int FRota = 65;
int RRota = 160;

//---------------------------------------------------------------------------------------------------------------------------
-----
//---------------------------------------------------------------------------------------------------------------------------
-----
//Enumeration List

enum StatusMode
{
  IDLE,
  Autonomous,
  Manual
} func_mode = IDLE;

enum MovementMode
{
  STOP,
  FORWARD,
  BACKWARD,
  LEFT,
  RIGHT
} mov_mode = STOP;

//---------------------------------------------------------------------------------------------------------------------------
-----
//--------------------------------------------------------------------------------------------------------------------------- ---
-----
//Operation Functions

void MoveForward()
{
  analogWrite(ENALeft, FRota);
  analogWrite(ENBRight, FRota);
  digitalWrite(RightForward, HIGH);
  digitalWrite(LeftForward, HIGH);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, LOW);
}

void MoveBackward()
{
  analogWrite(ENALeft, FRota);
  analogWrite(ENBRight, FRota);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, HIGH);
  digitalWrite(LeftBackward, HIGH);
}

void TurnLeft()
{
  analogWrite(ENALeft, RRota);
  analogWrite(ENBRight, RRota);
  digitalWrite(RightForward, HIGH);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, HIGH);
}

void TurnRight()
{
  analogWrite(ENALeft, RRota);
  analogWrite(ENBRight, RRota);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, HIGH);
  digitalWrite(RightBackward, HIGH);
  digitalWrite(LeftBackward, LOW);
}

void Stop()
{
  digitalWrite(ENALeft, LOW);
  digitalWrite(ENBRight, LOW);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, LOW);
}

float ForwardSensor()
{
  digitalWrite(ForwardTrig, LOW);
  delayMicroseconds(5);

  digitalWrite(ForwardTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(ForwardTrig, LOW);

  FSduration = pulseIn(ForwardEcho, HIGH);

  IFSdistance = FSduration * 0.0343 / 2;

  if(IFSdistance > 3 && IFSdistance < 300)
  {
    while(Sample_Count < Num_Samples)
    {
      sum += IFSdistance;
      Sample_Count++;
      delay(10);
    }
  }

  FSdistance = ((float)sum / (float)Num_Samples);

  Sample_Count = 0;
  sum = 0;

  Serial.print("Distance: ");
  Serial.println(FSdistance);

  return FSdistance;
}

float LeftSensor()
{
  digitalWrite(LeftTrig, LOW);
  delayMicroseconds(2);

  digitalWrite(LeftTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(LeftTrig, LOW);

  LSduration = pulseIn(LeftEcho, HIGH);

  ILSdistance = LSduration * 0.0343 / 2;

  if(ILSdistance > 3 && ILSdistance < 300)
  {
    while(Sample_Count2 < Num_Samples)
    {
      sum2 += ILSdistance;
      Sample_Count2++;
      delay(10);
    }
  }

  LSdistance = ((float)sum2 / (float)Num_Samples);

  Sample_Count2 = 0;
  sum2 = 0;

  Serial.print("Distance: ");
  Serial.println(LSdistance);

  return LSdistance;
}

float RightSensor()
{
  digitalWrite(RightTrig, LOW);
  delayMicroseconds(5);

  digitalWrite(RightTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(RightTrig, LOW);
  RSduration = pulseIn(RightEcho, HIGH);

  IRSdistance = RSduration * 0.0343 / 2;

  if(IRSdistance > 3 && IRSdistance < 300)
  {
    while(Sample_Count3 < Num_Samples)
    {
      sum3 += IRSdistance;
      Sample_Count3++;
      delay(10);
    }
  }

  RSdistance = ((float)sum3 / (float)Num_Samples);

  Sample_Count3 = 0;
  sum3 = 0;

  Serial.print("Distance: ");
  Serial.println(RSdistance);

  return RSdistance;
}

void AutonomousMode()
{
    if(func_mode == Autonomous)
    {
      ForwardSensor();
      RightSensor();
      LeftSensor();

      if(FSdistance > 45 && (RSdistance > 15 && RSdistance < 30))
      {
        MoveForward();
      }
      else if(FSdistance > 45 && RSdistance < 15)
      {
        TurnLeft();
        delay(80);
        MoveForward();
      }
      else if(FSdistance > 45 && RSdistance > 30)
      {
        TurnRight();
        delay(80);
        MoveForward();
      }
      if(FSdistance < 45 && LSdistance > 37)
      {
        MoveBackward();
        delay(100);
        TurnLeft();
        delay(200);
        ForwardSensor();
        if(FSdistance > 45)
        {
          MoveForward();
        }
      }
    }
}

void BluetoothData()
{
  if(Serial.available())
  {
    switch(Serial.read())
      {
        case 'f': func_mode = Manual; mov_mode = FORWARD;      break;
        case 'b': func_mode = Manual; mov_mode = BACKWARD;     break;
        case 'l': func_mode = Manual; mov_mode = LEFT;         break;
        case 'r': func_mode = Manual; mov_mode = RIGHT;        break;
        case 's': func_mode = Manual; mov_mode = STOP;         break;
        case 'a': func_mode = Autonomous;
        default:                                               break;
     }
  }
```

```
}

void ManualMode()
{
  if(func_mode == Manual)
  {
    switch(mov_mode)
    {
      case FORWARD: MoveForward();    break;
      case BACKWARD: MoveBackward();  break;
      case LEFT: TurnLeft();          break;
      case RIGHT: TurnRight();        break;
      case STOP: Stop();              break;
      default:                        break;
    }
  }
}

//-----------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------
void setup()
{
  pinMode(ENALeft, OUTPUT);
  pinMode(ENBRight, OUTPUT);
  pinMode(LeftForward, OUTPUT);
  pinMode(RightForward, OUTPUT);
  pinMode(LeftBackward, OUTPUT);
  pinMode(RightBackward, OUTPUT);

  pinMode(ForwardEcho, INPUT);
  pinMode(ForwardTrig, OUTPUT);
  pinMode(LeftEcho, INPUT);
  pinMode(LeftTrig, OUTPUT);
  pinMode(RightEcho, INPUT);
  pinMode(RightTrig, OUTPUT);

  Serial.begin(9600);
```

```
}

void loop()
{
  /*BluetoothData();
  ManualMode();
  AutonomousMode();*/
  ForwardSensor();
  RightSensor();
  LeftSensor();

  if(FSdistance > 40 && (RSdistance > 15 && RSdistance < 30))
  {
    MoveForward();
  }
  else if(FSdistance > 40 && RSdistance < 15)
  {
    TurnLeft();
    delay(80);
    MoveForward();
  }
  else if(FSdistance > 40 && RSdistance > 30)
  {
    TurnRight();
    delay(80);
    MoveForward();
  }
  if(FSdistance < 40 && LSdistance > 30)
  {
    MoveBackward();
    delay(100);
    TurnLeft();
    delay(200);
    ForwardSensor();
    if(FSdistance > 40)
    {
      MoveForward();
    }
  }
}
```

B.3 Fixed Rotations Navigation Algorithm

```
//Amr Elshenawy MEng Individual Major Project
//Autonomous Self-Navigating Robot
//FIXED ROTATIONS CONCEPT ALGORITHM
//3 Ultrasonic Sensors - Forward, Left, Right,
//4WD Drive Chassis
//Bluetooth Module
//--------------------------------------------------------------------------------------------------------------------------
-----
//Sensors Intialisations, Flags and Variables

int ForwardEcho = A12;
int ForwardTrig = A13;
int RightEcho = A10;
int RightTrig = A11;
int LeftEcho = A8;
int LeftTrig = A9;

long FSduration, LSduration, RSduration;
float IFSdistance, FSdistance, ILSdistance, LSdistance, IRSdistance, RSdistance;

unsigned int sum = 0;
unsigned int sum2 = 0;
unsigned int sum3 = 0;
unsigned int Sample_Count = 0;
unsigned int Sample_Count2 = 0;
unsigned int Sample_Count3 = 0;
int Num_Samples = 10;
int KeepLooking = 250;

//--------------------------------------------------------------------------------------------------------------------------
-----
//--------------------------------------------------------------------------------------------------------------------------
-----
//Motor Intialisations

#define ENALeft 5
#define ENBRight 6
#define LeftForward 7     //IN1
#define RightBackward 9   //IN3
```

```
#define LeftBackward 8    //IN2
#define RightForward 11   //IN4

int FRota = 80;
int RRota = 200;

//-----------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------
-----
//Enumeration List

enum StatusMode
{
  IDLE,
  Autonomous,
  Manual
} func_mode = IDLE;

enum MovementMode
{
  STOP,
  FORWARD,
  BACKWARD,
  LEFT,
  RIGHT
} mov_mode = STOP;

//-----------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------
-----
//Operation Functions

void MoveForward()
{
  analogWrite(ENALeft, FRota);
  analogWrite(ENBRight, FRota);
  digitalWrite(RightForward, HIGH);
  digitalWrite(LeftForward, HIGH);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, LOW);
}

void MoveBackward()
{
  analogWrite(ENALeft, FRota);
  analogWrite(ENBRight, FRota);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, HIGH);
  digitalWrite(LeftBackward, HIGH);
}

void TurnLeft()
{
  analogWrite(ENALeft, RRota);
  analogWrite(ENBRight, RRota);
  digitalWrite(RightForward, HIGH);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, HIGH);
}

void TurnRight()
{
  analogWrite(ENALeft, RRota);
  analogWrite(ENBRight, RRota);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, HIGH);
  digitalWrite(RightBackward, HIGH);
  digitalWrite(LeftBackward, LOW);
}

void Stop()
{
  digitalWrite(ENALeft, LOW);
  digitalWrite(ENBRight, LOW);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, LOW);
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, LOW);
}

float ForwardSensor()
{
  digitalWrite(ForwardTrig, LOW);
  delayMicroseconds(5);

  digitalWrite(ForwardTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(ForwardTrig, LOW);

  FSduration = pulseIn(ForwardEcho, HIGH);

  IFSdistance = FSduration * 0.0343 / 2;

  if(IFSdistance > 3 && IFSdistance < 300)
  {
    while(Sample_Count < Num_Samples)
    {
      sum += IFSdistance;
      Sample_Count++;
      delay(10);
    }
  }

  FSdistance = ((float)sum / (float)Num_Samples);

  Sample_Count = 0;
  sum = 0;

  Serial.print("Distance: ");
  Serial.println(FSdistance);

  return FSdistance;
}

float LeftSensor()
{
  digitalWrite(LeftTrig, LOW);
  delayMicroseconds(2);

  digitalWrite(LeftTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(LeftTrig, LOW);

  LSduration = pulseIn(LeftEcho, HIGH);

  ILSdistance = LSduration * 0.0343 / 2;

  if(ILSdistance > 3 && ILSdistance < 300)
  {
    while(Sample_Count2 < Num_Samples)
    {
      sum2 += ILSdistance;
      Sample_Count2++;
      delay(10);
    }
  }

  LSdistance = ((float)sum2 / (float)Num_Samples);

  Sample_Count2 = 0;
  sum2 = 0;

  Serial.print("Distance: ");
  Serial.println(LSdistance);

  return LSdistance;
}

float RightSensor()
{
  digitalWrite(RightTrig, LOW);
  delayMicroseconds(5);

  digitalWrite(RightTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(RightTrig, LOW);
  RSduration = pulseIn(RightEcho, HIGH);

  IRSdistance = RSduration * 0.0343 / 2;

  if(IRSdistance > 3 && IRSdistance < 300)
  {
    while(Sample_Count3 < Num_Samples)
    {
      sum3 += IRSdistance;
      Sample_Count3++;
      delay(10);
    }
  }

  RSdistance = ((float)sum3 / (float)Num_Samples);

  Sample_Count3 = 0;
  sum3 = 0;

  Serial.print("Distance: ");
  Serial.println(RSdistance);

  return RSdistance;
}

void AutonomousMode()
{
    if(func_mode == Autonomous)
    {
      ForwardSensor();
      RightSensor();
      LeftSensor();

      if(FSdistance > 45 && RSdistance > 37 && LSdistance > 37)
      {
        MoveForward();
      }
      else if(FSdistance > 45 && RSdistance < 37 && LSdistance > 37)
      {
```

```
    TurnLeft();
    delay(80);
    MoveForward();
  }
  else if(FSdistance > 45 && LSdistance < 37 && RSdistance > 37)
  {
    TurnRight();
    delay(80);
    MoveForward();
  }
  else if(FSdistance < 45 && RSdistance > 37 && LSdistance > 37) //Begin test line
  {
    if(RSdistance > LSdistance)
    {
      TurnRight();
      delay(700);
      if(FSdistance > 45)
      {
        MoveForward();
      }
    }
    else
    {
      TurnLeft();
      delay(700);
      if(FSdistance > 45)
      {
        MoveForward();
      }
    }
  }
  /*else if(FSdistance < 45 && RSdistance < 30 && LSdistance > 30)
  {
    TurnLeft();
    delay(200);
    if(FSdistance > 45)
    {
      MoveForward();
    }
  }                                                    //End test line
```

```
    /*if(FSdistance < 45)
     {
       MoveBackward();
       delay(400);
       while(FSdistance < 45)
       {
         if(RSdistance > LSdistance)
         {
           TurnRight();
           delay(250);
           ForwardSensor();
         }
         else if(LSdistance > RSdistance)
         {
           TurnLeft();
           delay(250);
           ForwardSensor();
         }
         /*ForwardSensor();
       }
       MoveForward();
     }*/
   }
}

void BluetoothData()
{
  if(Serial.available())
  {
    switch(Serial.read())
     {
       case 'f': func_mode = Manual; mov_mode = FORWARD;      break;
       case 'b': func_mode = Manual; mov_mode = BACKWARD;     break;
       case 'l': func_mode = Manual; mov_mode = LEFT;         break;
       case 'r': func_mode = Manual; mov_mode = RIGHT;        break;
       case 's': func_mode = Manual; mov_mode = STOP;         break;
       case 'a': func_mode = Autonomous;
       default:                                               break;
     }
  }
```

```
}

void ManualMode()
{
  if(func_mode == Manual)
  {
    switch(mov_mode)
    {
      case FORWARD: MoveForward();    break;
      case BACKWARD: MoveBackward();  break;
      case LEFT: TurnLeft();          break;
      case RIGHT: TurnRight();        break;
      case STOP: Stop();              break;
      default:                        break;
    }
  }
}

//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------
void setup()
{
  pinMode(ENALeft, OUTPUT);
  pinMode(ENBRight, OUTPUT);
  pinMode(LeftForward, OUTPUT);
  pinMode(RightForward, OUTPUT);
  pinMode(LeftBackward, OUTPUT);
  pinMode(RightBackward, OUTPUT);

  pinMode(ForwardEcho, INPUT);
  pinMode(ForwardTrig, OUTPUT);
  pinMode(LeftEcho, INPUT);
  pinMode(LeftTrig, OUTPUT);
  pinMode(RightEcho, INPUT);
  pinMode(RightTrig, OUTPUT);

  Serial.begin(9600);
}
void loop()
```

```
{
  /*BluetoothData();
  ManualMode();
  AutonomousMode();*/
  ForwardSensor();
  RightSensor();
  LeftSensor();

  if(FSdistance > 45 && RSdistance > 37 && LSdistance > 37)
  {
    MoveForward();
  }
  if(FSdistance > 45 && RSdistance < 37 && LSdistance > 37)
  {
    TurnLeft();
    delay(80);
    MoveForward();
  }
  else if(FSdistance > 45 && LSdistance < 37 && RSdistance > 37)
  {
    TurnRight();
    delay(80);
    MoveForward();
  }
  if(FSdistance < 45) //Begin test line
  {
    if(RSdistance > LSdistance)
    {
      TurnRight();
      delay(700);
      MoveForward();
    }
    else
    {
      TurnLeft();
      delay(700);
      MoveForward();
    }
  }
}
```

B.4 Automated Variable Rotations Algorithm

```
//Amr Elshenawy MEng Individual Major Project
//Autonomous Self-Navigating Robot
//FINAL VERSION - AUTOMATED ROTATION
//3 Ultrasonic Sensors - Forward, Left, Right
//4WD Drive Chassis
//Bluetooth Module
//----------------------------------------------------------------------------------------------------------------------
-----
//Sensors Intialisations, Flags and Variables

int ForwardEcho = A12;
int ForwardTrig = A13;
int RightEcho = A10;
int RightTrig = A11;
int LeftEcho = A8;
int LeftTrig = A9;

long FSduration, LSduration, RSduration;
float IFSdistance, FSdistance, ILSdistance, LSdistance, IRSdistance, RSdistance;

unsigned int sum = 0;
unsigned int sum2 = 0;
unsigned int sum3 = 0;
unsigned int Sample_Count = 0;
unsigned int Sample_Count2 = 0;
unsigned int Sample_Count3 = 0;

int Num_Samples = 10;
int KeepLooking = 250;

//----------------------------------------------------------------------------------------------------------------------
-----
//----------------------------------------------------------------------------------------------------------------------
-----
//Motor Intialisations

#define ENALeft 5
#define ENBRight 6
#define LeftForward 7     //IN1
```

```
#define RightBackward 9   //IN3
#define LeftBackward 8    //IN2
#define RightForward 11   //IN4

int FRota = 80;
int RRota = 200;

//-------------------------------------------------------------------------------------------------------------------------
-----
//-------------------------------------------------------------------------------------------------------------------------
-----
//Enumeration List

enum StatusMode
{
  IDLE,
  Autonomous,
  Manual,
  Test
} func_mode = IDLE;

enum MovementMode
{
  STOP,
  FORWARD,
  BACKWARD,
  LEFT,
  RIGHT
} mov_mode = STOP;

//-------------------------------------------------------------------------------------------------------------------------
-----
//-------------------------------------------------------------------------------------------------------------------------
-----
//Operation Functions

void MoveForward()
{
  analogWrite(ENALeft, FRota);
  analogWrite(ENBRight, FRota);
  digitalWrite(RightForward, HIGH);
  digitalWrite(LeftForward, HIGH);    //MOVE FORWARD FUNCTION
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, LOW);
}

void MoveBackward()
{
  analogWrite(ENALeft, FRota);
  analogWrite(ENBRight, FRota);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, LOW);   //MOVE BACKWARD FUNCTION
  digitalWrite(RightBackward, HIGH);
  digitalWrite(LeftBackward, HIGH);
}

void TurnLeft()
{
  analogWrite(ENALeft, RRota);
  analogWrite(ENBRight, RRota);
  digitalWrite(RightForward, HIGH);
  digitalWrite(LeftForward, LOW);   //TURN LEFT FUNCTION
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, HIGH);
}

void TurnRight()
{
  analogWrite(ENALeft, RRota);
  analogWrite(ENBRight, RRota);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, HIGH);    //TURN RIGHT FUNCTION
  digitalWrite(RightBackward, HIGH);
  digitalWrite(LeftBackward, LOW);
}

void Stop()
{
  digitalWrite(ENALeft, LOW);
  digitalWrite(ENBRight, LOW);
  digitalWrite(RightForward, LOW);
  digitalWrite(LeftForward, LOW);   //STOP FUNCTION
  digitalWrite(RightBackward, LOW);
  digitalWrite(LeftBackward, LOW);
}

float ForwardSensor()
{
  digitalWrite(ForwardTrig, LOW); //Set soundwaves transmitter to LOW
  delayMicroseconds(5);   //Wait 5us

  digitalWrite(ForwardTrig, HIGH);  //Set soundwaves transmitter to HIGH
  delayMicroseconds(10); // Wait 10us
  digitalWrite(ForwardTrig, LOW);   //Set soundwaves trasnmitter to LOW

  FSduration = pulseIn(ForwardEcho, HIGH);    //Receiver soundwaves duration dumped into
variable

  IFSdistance = FSduration * 0.0343 / 2;    //distance calculation

  if(IFSdistance > 3 && IFSdistance < 350)
  {
    while(Sample_Count < Num_Samples)
    {
      sum += IFSdistance;                 //Further distance calibration
      Sample_Count++;
      delay(10);
    }
  }

  FSdistance = ((float)sum / (float)Num_Samples); //Average distance calculated

  Sample_Count = 0;
  sum = 0;

  //Serial.print("Distance: ");
  //Serial.println(FSdistance);

  return FSdistance;
}
```

```
float LeftSensor()       // SAME COMMENTS FROM FORWARD SENSOR
{
  digitalWrite(LeftTrig, LOW);
  delayMicroseconds(2);

  digitalWrite(LeftTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(LeftTrig, LOW);

  LSduration = pulseIn(LeftEcho, HIGH);

  ILSdistance = LSduration * 0.0343 / 2;

  if(ILSdistance > 3 && ILSdistance < 350)
  {
    while(Sample_Count2 < Num_Samples)
    {
      sum2 += ILSdistance;
      Sample_Count2++;
      delay(10);
    }
  }

  LSdistance = ((float)sum2 / (float)Num_Samples);

  Sample_Count2 = 0;
  sum2 = 0;

  //Serial.print("Distance: ");
  //Serial.println(LSdistance);

  return LSdistance;
}

float RightSensor()   // SAME COMMENTS FROM FORWARD SENSOR
{
  digitalWrite(RightTrig, LOW);
  delayMicroseconds(5);
  digitalWrite(RightTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(RightTrig, LOW);

  RSduration = pulseIn(RightEcho, HIGH);

  IRSdistance = RSduration * 0.0343 / 2;

  if(IRSdistance > 3 && IRSdistance < 350)
  {
    while(Sample_Count3 < Num_Samples)
    {
      sum3 += IRSdistance;
      Sample_Count3++;
      delay(10);
    }
  }

  RSdistance = ((float)sum3 / (float)Num_Samples);

  Sample_Count3 = 0;
  sum3 = 0;

  //Serial.print("Distance: ");
  //Serial.println(RSdistance);

  return RSdistance;
}

void BluetoothData()    //BLUETOOTH function for categorizing the data recieved from the
app
{
  if(Serial.available())
  {
    switch(Serial.read())
      {
        case 'f': func_mode = Manual; mov_mode = FORWARD;      break;
        case 'b': func_mode = Manual; mov_mode = BACKWARD;     break;
        case 'l': func_mode = Manual; mov_mode = LEFT;         break;
        case 'r': func_mode = Manual; mov_mode = RIGHT;        break;
        case 's': func_mode = Manual; mov_mode = STOP;      break;
        case 'a': func_mode = Autonomous;                   break;
        case 't': func_mode = Test;                         break;
        default:                                            break;
      }
   }
}

void AutonomousMode()     //Autonomous Navigation Algorithm Control
{
    if(func_mode == Autonomous)
    {
      ForwardSensor();
      RightSensor();
      LeftSensor();

      if(FSdistance > 47 && RSdistance > 37 && LSdistance > 37)
      {
        MoveForward();
      }
      else if(FSdistance > 47 && RSdistance < 37 && LSdistance > 37)
      {
        TurnLeft();
        delay(80);
        MoveForward();
      }
      else if(FSdistance > 47 && LSdistance < 37 && RSdistance > 37)
      {
        TurnRight();
        delay(80);
        MoveForward();
      }
      if(FSdistance < 47)
      {
        MoveBackward();
        delay(400);
        while(FSdistance < 53)
        {
          if(RSdistance > LSdistance)
          {
            TurnRight();
            delay(250);
            ForwardSensor();
          }
          else if(LSdistance > RSdistance)
          {
            TurnLeft();
            delay(250);
            ForwardSensor();
          }
          //ForwardSensor();
        }
        MoveForward();
      }
    }
}

void ManualMode()   //Manual Control Function
{
  if(func_mode == Manual)
  {
    switch(mov_mode)
    {
      case FORWARD: MoveForward();    break;
      case BACKWARD: MoveBackward();  break;
      case LEFT: TurnLeft();          break;
      case RIGHT: TurnRight();        break;
      case STOP: Stop();              break;
      default:                        break;
    }
  }
}

void TestMode()   //Automated Hardware Testing Algorithm
{
  if(func_mode == Test)
  {
    ForwardSensor();
    LeftSensor();
    RightSensor();
    if(FSdistance == 0 || LSdistance == 0 || RSdistance == 0)
    {
      digitalWrite(LED_BUILTIN, HIGH);
      delay(300);
      digitalWrite(LED_BUILTIN, LOW);
      delay(300);
    }
    else if(FSdistance > 0 && LSdistance > 0 && RSdistance > 0)
    {
      digitalWrite(LED_BUILTIN, HIGH);
    }
  }
  if(func_mode == Manual || func_mode == Autonomous)
  {
    digitalWrite(LED_BUILTIN, LOW);
  }
}

//-----------------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------------
-----
//-----------------------------------------------------------------------------------------------------------------------------
-----

void setup()
{
  pinMode(ENALeft, OUTPUT);
  pinMode(ENBRight, OUTPUT);
  pinMode(LeftForward, OUTPUT);
  pinMode(RightForward, OUTPUT);    //INTILISATIONS OF MOTOR PINS FUNCTION
MODE
  pinMode(LeftBackward, OUTPUT);
  pinMode(RightBackward, OUTPUT);

  pinMode(ForwardEcho, INPUT);
  pinMode(ForwardTrig, OUTPUT);
  pinMode(LeftEcho, INPUT);
  pinMode(LeftTrig, OUTPUT);    //INITIALISATIONS OF SENSOR PINS FUNCTION MODE
  pinMode(RightEcho, INPUT);
  pinMode(RightTrig, OUTPUT);

  pinMode(LED_BUILTIN, OUTPUT);

  Serial.begin(9600);     //SERIAL COMMUNCATION BAUD RATE SPPED
}

void loop()
{
  BluetoothData();
  AutonomousMode();   //CALLING OF ALL MAIN FUNCTIONS
  ManualMode();
  TestMode();
  /*unsigned long startt = micros();
  AutonomousMode();
  unsigned long endt = micros();
  unsigned long delta = endt - startt;
  Serial.println(delta);*/
}
```