

Rochester Institute of Technology

RIT Digital Institutional Repository

Theses

2022

Intelligent Navigation Service Robot Working in a Flexible and Dynamic Environment

Said Abdallah
sma6163@rit.edu

Follow this and additional works at: <https://repository.rit.edu/theses>

Recommended Citation

Abdallah, Said, "Intelligent Navigation Service Robot Working in a Flexible and Dynamic Environment" (2022). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

Rochester Institute of Technology



**INTELLIGENT NAVIGATION SERVICE ROBOT
WORKING IN A FLEXIBLE AND DYNAMIC
ENVIRONMENT**

by

Said Abdallah

A thesis submitted in partial fulfillment of the requirements for
the degree of Master of Science in Mechanical Engineering

Supervised by

Dr. Slim Saidi & Dr. Ghalib Kahwaji

Department of Mechanical & Industrial Engineering
Rochester Institute of Technology Dubai campus

United Arab Emirates

2022

RIT

Master of Science in

Mechanical Engineering

THESIS APPROVAL

**INTELLIGENT NAVIGATION SERVICE ROBOT WORKING IN A
FLEXIBLE AND DYNAMIC ENVIRONMENT**

Student Name: Said Abdallah

Dr. Slim Saidi
Associate Professor
Department of Mechanical & Industrial Engineering

(Thesis Advisor)

Dr. Ghalib Kahwaji
Professor and Chair
Department of Mechanical & Industrial Engineering

(Thesis Co-advisor)

Dr. Salman Pervaiz
Associate Professor
Department of Mechanical & Industrial Engineering

(Committee Member)

Dr. Mohamed Abdul Rahman
Assistant Professor
Department of Mechanical & Industrial Engineering

(Committee Member)

Acknowledgements

I would like to thank my esteemed supervisor – Dr. Slim S. for providing guidance and feedback throughout this project. My gratitude extends to Dr. Ghalib K. for his invaluable supervision, support, and tutelage. In addition, many thanks to all faculty and staff members of the mechanical and industrial department at the Rochester institute of technology for their valuable inputs. Besides, it is my pleasure to acknowledge the financial support of the Rochester Institute of Technology for this project.

Finally, the big and special thanks go out to my wonderful wife Thanaa; you provided me with the help, support, encouragement, and patience I needed to complete this MSc degree. Without you, nothing would have been possible.

ABSTRACT

Numerous sensor fusion techniques have been reported in the literature for a number of robotics applications. These techniques involved the use of different sensors in different configurations. However, in the case of food driving, the possibility of the implementation has been overlooked. In restaurants and food delivery spots, enhancing the food transfer to the correct table is neatly required, without running into other robots or diners or toppling over. In this project, a particular algorithm module has been proposed and implemented to enhance the robot driving methodology and maximize robot functionality, accuracy, and the food transfer experience. The emphasis has been on enhancing movement accuracy to reach the targeted table from the start to the end. Four major elements have been designed to complete this project, including mechanical, electrical, electronics, and programming. Since the floor condition greatly affecting the wheels and turning angle selection, the movement accuracy was improved during the project. The robot was successfully able to receive the command from the restaurant and go to deliver the food to the customers' tables, considering any obstacles on the way to avoid. The robot has equipped with two trays to mount the food with well-configured voices to welcome and greet the customer. The performance has been evaluated and undertaken using a routine robot movement tests. As part of this study, the designed service wheeled robot required to be with a high-performance real-time processor. As long as the processor was adequate, the experimental results showed a highly effective search robot methodology. Having concluded from the study that a minimum number of sensors are needed if they are placed appropriately and used effectively on a robot's body, as navigation could be performed by using a small set of sensors. The Arduino Due has been used to provide a real-time operating system. It has provided a very successful data processing and transfer throughout any regular operation. Furthermore, an easy-to-use application has been developed to improve the user experience, so that the operator can interact directly with the robot via a special setting screen. It is possible, using this feature, to modify advanced settings such as voice commands or IP address without having to return back to the code.

Table of contents

ACKNOWLEDGEMENTS	I
ABSTRACT.....	II
TABLE OF CONTENTS	III
LIST OF FIGURES	V
LIST OF TABLES	VII
CHAPTER 1: INTRODUCTION.....	1
1.1 PROBLEM STATEMENT	2
1.2 RESEARCH AIMS AND METHODOLOGY	2
1.3 STRUCTURE OF THE THESIS.....	3
CHAPTER 2 : BACKGROUND AND LITERATURE REVIEW.....	4
2.1 BACKGROUND.....	4
2.1.1 SERVICE ROBOT MECHANISMS	4
2.1.2 MOBILITY AND AUTONOMY	5
2.1.3 ROBOT NAVIGATION	6
2.2 LITERATURE REVIEW	8
2.2.1 NAVIGATION: THE EXPLORATION PATH.....	10
2.2.1.1 NAVIGATION STRATEGIES.....	12
2.2.1.2 ROBOT LOCALIZATION	16
2.2.2 SENSOR-FUSION CONTROL	16
2.3 CONCLUSIONS.....	21
CHAPTER 3 : THE DESIGN AND FABRICATION	23
3.1: INTRODUCTION	23
3.1.1: CONCEPTUAL DESIGN.....	24
3.2: MECHANICAL DESIGN.....	26
3.2.1 ROBOT BODY FABRICATION	27
3.2.1.1 CHASSIS.....	27
3.2.1.2 BODY COVER AND TRAYS HOLDERS	29
3.2.1.3 THE MAIN ROBOT FRAME BODY	30
3.2.2 WHEEL CONFIGURATION	32

3.3: ELECTRICAL AND POWER CIRCUIT DESIGN	34
3.3.1 THE POWER AND CHARGING CONFIGURATION	34
3.3.2 MOTORS DRICE AND POWER SYSTEM.....	40
3.4: ELECTRONIC CIRCUIT DESIGN.....	47
3.4.1.1 CIRCUIT DIAGRAM AND CONNECTION.....	51
3.4.1.2 CODING AND SOFTWARE	54
CHAPTER 4 : OBJECT DETECTION-BASED ON SENSOR-FUSION SYSTEM	86
4.1: INTRODUCTION.....	86
4.2: OBJECT DETECTION PROBLEM	90
4.3: SENSOR CONFIGURATION.....	90
4.3.1 SENSOR TYPES.....	92
4.3.1.1 LIDAR DETECTING SENSOR	92
4.3.1.2 ULTRASONIC RANGE SENSORS.....	92
CHAPTER 5 : CONCLUSIONS AND FUTURE WORK	94
5.1 CONCLUSION	94
5.2 RROPOSED FUTURE WORK.....	96

List of Figures

Figure 1: Different service robot categories	4
Figure 2: General schematic for robot navigation.....	7
Figure 3: Navigation architecture proposed in this work [65]	15
Figure 4: Robot localization using EKF sensor fusion.	18
Figure 5: Functional Diagram of multisensor integration and fusion (figure redrawn from [84])	20
Figure 6: The main units that describing the AMR robot	24
Figure 7: Project conceptual design sequence	25
Figure 8: The main project component breakdown elements	26
Figure 9: Extrusion Aluminum Profiles (T-slot aluminum profiles)	27
Figure 10: 20x20 mm Aluminum profiles base structure	28
Figure 11: Aluminum profile fasteners: (a)the three-way corner connector, (b)T-Slot nuts, (c)Screw bolt, (d)corner brackets.	28
Figure 12: The Aluminum profile used to fix the motors	29
Figure 13: The ultrasonic on the (a) sides, (b) front or bac.....	29
Figure 14:(a) Real model of main body of robot, (b) Inventor 3D design model of main body of the service robot.....	30
Figure 15: 2D CAD model for the base bottom design	33
Figure 16: (a) 12 V-12AH Lead Acid Rechargeable Battery, (b) 12V battery charger	36
Figure 17: The powering and charging circuit connection	37
Figure 18: 12V lead Acid battery state of charge (SOC) vs. voltage while under discharge.....	38
Figure 19: 12V lead Acid battery state of charge (SOC) vs. voltage while under charge	38
Figure 20: LM2596S DC-DC step-down buck converter	39
Figure 21: DC-DC Buck Converters circuit diagram	39
Figure 22: The motor and driver kit used to run the service robot	40
Figure 23: the stepper motor used to control the robot motor.....	42
Figure 24: Detailed one motor and driver configuration.	44
Figure 25: Stepper motor driver input signal waveform timing diagram	45
Figure 26: Motors power circuit diagram	47
Figure 27: Arduino Due board.....	48
Figure 28: Real-time operating system for microcontrollers	49
Figure 29: Arduino Pro	50
Figure 30: NodeMCU ESP8266 Wi-Fi SoC.....	50
Figure 31: the complete electronics controlling circuit wiring diagram	51
Figure 32: Functional block diagram of the proposed service robot model.....	52
Figure 33: top view diagram is showing 8 ultrasonics sensors around the robot.....	68
Figure 34: The main interface for the MIT app inventor	78
Figure 35: Fig 4: Robot application three main screens (a) Setting screen, (b) Welcoming screen, (c) Start screen	79
Figure 36: The common database non-visible component	79
Figure 37: First screen time setting.....	80
Figure 38: First screen Blockly code	80
Figure 39: The start screen database non-visible component	81
Figure 40: The application start screen.....	81
Figure 41: Part of the main screen components.....	82

Figure 42: Initiate the link between the NodeMCU and the application, and the tables buttons.....	82
Figure 43: The actions once the Start button pressed	83
Figure 44: Placing the setting button actions	83
Figure 45: the communication code from the NodeMCU to the application.....	84
Figure 46: The simplest algorithm procedures	87
Figure 47: The algorithm states	87
Figure 48: Control process.....	91
Figure 49: The Obtained Environment Map from RPLIDAR A1 Scanning.....	92
Figure 50: the working principle of the ultrasonic.....	93

LIST OF TABLES

Table 1: Parts of main body of 3D printer:	31
Table 2: The required calculation for the motors and batteries selections	35
Table 3: The main items required for the power circuit.....	36
Table 4: The robot's characteristics and system inputs	41
Table 5: Analysis results	41
Table 6: Tables options selected using 3 digits binary number	76

Chapter 1: INTRODUCTION

Defining the industrial robotics term is necessary before starting to discuss the history of the robot. This chapter gives a basic understanding of robots by examining their different fundamentals. Introducing robotics cannot be done without mentioning the fundamental laws that govern robots. Asimov [8] noted that one of the basic principles of robots is that they are good machines serving human beings, as he envisioned a robot as a mechanical replica of a person without emotions. Besides, General Motors developed and used the first industrial robot. Industrial robots have been extensively used to perform repetitive, heavy, and hazardous processes in the manufacturing industry, especially for repetitive, serious, and dangerous tasks. A typical industrial robot usually has a fixed working range and is designed to operate within it. Modern industrial robots can perform the same functions in different locations due to mobility [9]. Therefore, industrial robots must operate autonomously to carry out their tasks effectively. Therefore, they need to have the right tools for exploring their environment. Robot's design and creation was traditionally handled through the use of programming so that could perform various tasks, including those that were dangerous and may need to be repeated several times with an increased level of accuracy and reliability. Looking forward, robotic devices are used in place of human beings to perform tasks precisely and without error. However, robots are being used for service activities as well. They operate in the same manner as humans do. A few successful examples of service robot companies assist human beings by performing a wide range of tasks, including household chores. Robots have seen an increase in their use in the service sector in the past few years as compared with the industrial sector. Robots allow humans to live a better life and help them be more productive. They are equipped with many capabilities to assist humans in different scenarios. It is imperative for the robots to be able to navigate so they can follow a human's commands and move around the target.

Many applications have used service robots, including the manufacturing industry, and military operations. So, the robot interacts with a variety of tools and equipment and is tasked with modeling its environment, moving efficiently, and identifying objects while using its navigation system and management system to perform assigned tasks [1]. Real-time robot navigation controlling system is accomplished through three main functions: route planning [2], robot localization [3], and control the motion [4], [5]. Finding the shortest path from a point of start to a place of destination while avoiding obstacles is the first part of the process. Second, the robot must be capable of determining its position and orientation in relation to the environment. Providing the robot with the ability to function in its environment [4] is the third main task. Mobile robots are able to carry out specific useful functions within their environment as they perform assigned tasks, such as grasping and moving objects. Mobile robots are capable of searching, finding, and relocating objects in a variety of manufacturing operations and environments [1]. It is a robot's job to explore its environment and find specific objects, which it can then use to perform a useful task, such as moving them to a different location. Enhanced navigation systems and vision-based object recognition systems will be needed for such robots. It is crucial that the navigation system generates paths that cover the entire area and determines the robot's location within the area. Additionally, it needs to identify all obstacles in real-time in order to select the most effective path towards a destination [6]. To undertake the exploration task, vision-based object recognition is essential; it involves the use of a vision sensor and an appropriate technique for recognizing objects. At present, most robots in the workplace can perform specific and repeatable tasks that they are precisely programmed to perform. The robot must be reprogrammed to do something else, which is expensive and time-consuming. Researchers and experimenters are undergoing continuous improvement and

development in this sphere of robotics, which requires the robot to see and understand its surroundings. These developments made the robots able to perform multiple tasks and operate in static or dynamic environments. Different sets of complex systems are necessary to identify an object and pick it up without being taught that object to the robot, which is not broadly available in the marketplace at this time. Several well-known companies, such as Amazon and Google, have developed robots that can partially work in their warehouses where human hands still perform the bulk of the labor. In spite of the fact that robots cannot match human dexterity and see the world the same way humans do, they still have the potential to accomplish great things. In the last decade of the 20th century, the continuous deep learning revolution has advanced robotics and artificial intelligence research to the point where robot vision and motor control are approaching human levels.

1.1 PROBLEM STATEMENT

In recent years, service robots have been proposed for numerous applications, including those in the service industry and even in the civil service. Service robots can include automated vacuum cleaners or intelligent wheelchairs. They can also be used to deliver medicine or food. However, in order to perform the searching services, robots must be able to recognize specific objects they can approach, grasp, and relocate. However, developing service robots with navigation systems and reliable work plans has long been recognized as necessary by the industry. Although the robot's production lines are slated to reduce the time for deployment of that manufacturing equipment and the cost overruns, multiple lines are still plagued by delays and overruns, which can commonly be attributed to the factory's high running costs and its extended operating processes. First, the traditional complex control systems used as the basis of any robotics system significantly limit our capability to model and solve problems related to the functions of the robot during look-ahead scheduling. Secondly, if the idea and workflow are not identified correctly during scheduling, subsequent conflicts can occur. Today's industrial projects are becoming more and more technically complex and logistically challenging, which exposes the manufacturing operations to even more complex constraints. Thus, there is a need for a better understanding of building and designing a commercial-grade prototype service robot with the intelligence to navigate as good as the robots produced by industrial plants, at a reasonable price, and with more outstanding capabilities.

1.2 RESEARCH AIMS AND METHODOLOGY

Developing a novel real-time control system for serving food autonomously in a static restaurant environment. Unlike the currently available robotics' control systems, the new one is straightforward, simple, and easy to access and modify, as it is supported by multiple open-source platforms, configured in a novel designed circuit. Additionally, this research work is relatively cheap when compared to currently existing service robots, having a lower cost of about 76%. Completely different from anything currently available in the market, as available service robots are closed systems with no ability to access or perform any extra tasks, as the user is not able to run the robot seamlessly without the cloud. In contrast, our robot is completely operated locally with the support of a Wi-Fi connection. Furthermore, a newly developed application offers the ability to control the robot remotely from any android device, regardless of its location. In addition, the application is able to display unlimited extra options which could be included to be used for either controlling or monitoring purposes, as these are extremely new features not available in previous robots.

For the purpose of demonstrating the concept, the robot has designed, constructed, described, and shown as a part of the thesis. Throughout the work, C++ and C programming languages have been used. The major task of building the restaurant service robot and developing its control system, which allows the robot to find and track targets (tables), is divided into many stages, first is to design and fabricate the body frame in size, shape, and quality that is appropriate, second, design the power circuit diagram including the battery and the charging system, in the third step, integrate three different controllers so that real-time actions can be implemented based on the data collected from the sensors, fourth, the navigation task requires creating a path that incorporates as much sense as possible to encompass the entire environment. This is followed by constructing the path so that the robot can use it

1.3 STRUCTURE OF THE THESIS

This thesis is organized as a summary report and related research project which contributes to partial fulfilment of the requirements for the degree of MSc in Mechanical Engineering. This thesis is primarily about robotics, and it is a complex field. Thus, it is necessary to introduce some basic concepts in the first chapter before delving into more detailed discussions in the following chapters. A literature review and background are presented in the following chapter (chapter 2). The robot design and fabrication process are described in Chapter Three. These phases include mechanical and electrical design, power circuit design, and electronic circuit design. Path and motion control are discussed in Chapter four. Using the sensor-fusion system, chapter five describes how the object detection operation is implemented. In addition, it provides detailed explanations about the path exploration process. Chapter six presents the results of a simulation of a robot control system and introduces the algorithm. Finally concluding remarks are presented in Chapter 7. Additionally, the combination of all codes and results that summarize the results presented in the paper are also presented. Various possibilities and perspectives for the future are also discussed. In order to coordinate and perform a given task, the service robots need certain sensors and equipment so that they can receive feedback from the external environment. Therefore, the robot should constantly monitor the surrounding environment to interact smartly as an autonomous robot by avoiding unexpected obstacles. Several issues need to be carefully considered. Service robots work in an environment where certain things are not clearly prefigured, then collect data and turn it into a piece of knowledge, then actions. For instance, a bottle of water may appear on the table at first, but it may be somewhere else after a certain time. To find the bottle of water, the robot has to search through the whole space. If the robot knows how a human thinks, he will know the likely places we could find the bottle in, ensuring a more computationally effective search.

Chapter 2 : BACKGROUND AND LITERATURE REVIEW

2.1 BACKGROUND

In recent years, the importance of intelligent navigation has increased for mobile robots, as the interest in service robots has grown. The service robots can recognize their environment and react autonomously to any unexpected situations by using the sensors. Even after significantly more research has been done on developing autonomous service robots, the robots still face a challenge when it comes to navigating in real-life environments despite research reports supporting their capabilities. On the other hand, a smart and autonomous service robot has become a real consideration because of the rapid spread of COVID-19 worldwide in recent years. Governments and citizens required some additional support and assistance to fight the virus by reducing human interactions, ensuring that the people are wearing masks in public areas, and maintaining as far distance as possible. Therefore, it is necessary to realize the need for intelligent autonomous multipurpose service robots that can accomplish various tasks supported by navigation systems. An autonomous robot equipped with smart capabilities could go around inside malls as a sanitizing robot, ensuring a clean and safe environment all the time.

2.1.1 SERVICE ROBOT MECHANISMS

The service tasks are generally vague and complex, such as "bring me a cup of coffee" or "give me a glass of water" [10]. As a result, the service robot must possess high levels of reasoning and understanding to perform these tasks. The holistic robot service mechanisms have been investigated in this section to bridge this gap. In this case, the service robot can determine the time and provide active services accordingly. According to their way of locomotion on the ground, the mobile robots can be classified into three different categories: wheeled, legged, and articulated [11]. Each of them has unique characteristics that make it suitable for specific applications. Rotational devices like wheels and tracks are facilitated and driven by the movement of wheeled robots. That mechanism is usually simple, the weight is low, and it moves quickly and efficiently on structured and regular surfaces. Because of this, they are used nearly everywhere in the industry. However, they are not as effective on rough or soft surfaces, such as outdoor, unpaved terrain. Wheeled robots, for instance, are highly energy-intensive to move over uneven terrain or obstacles [11]. Due to their limitations, the other two types of robots are necessary because more than half of Earth's landmass cannot be accessed by existing track vehicles and wheeled vehicles [11], [12]. Figure 1 is listing the main service robot classifications.

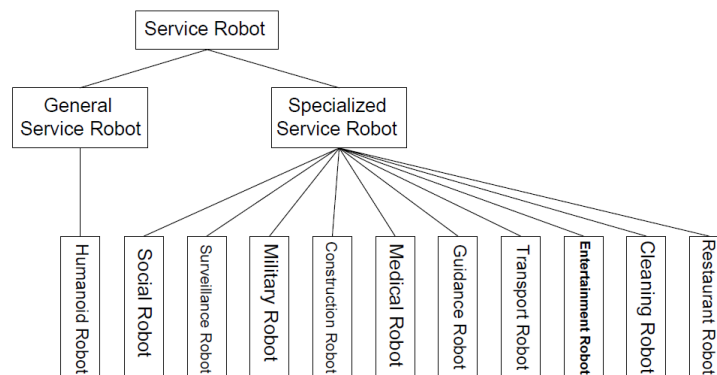


Figure 1: Different service robot categories

Legged and articulated robots share many characteristics with their biological counterparts. Since they use discrete footholds, robots with legs [12], are more mobile in soft and unstructured terrain. Support and traction are provided by only point contacts with the ground, As opposed to wheeled robots, wheeled robots need continuous support surfaces. Despite this, they still have some shortcomings, such as low speeds, complicated controls, heavyweight, and high energy consumption. The segments of articulated robots are gathered and interconnected in such a way as to resemble snakes or centipedes [13], [14]. They are beneficial for moving over irregular terrains and through narrow passages primarily due to their ability to move along and across them.

Usually, the service robots are designed and manufactured from scratch to meet the exact functional needs. For instance, these robots may be designed and programmed in an autonomous or manual mode. Normally, service robots operate in autonomous mode when they perform service tasks. Nevertheless, if a system malfunction or error occurs, it will automatically switch to manual mode. Besides that, an option to choose between the manual and the automatic modes on the control panel is available [15]. A service robot will be used as an example to highlight the cooperative relationships between the two modes. In addition to changing states dynamically, the service robot can also dynamically alter its behavior depending on the task. Cognitively, this task-driven state is characterized by high energy consumption, as well as being task-driven cognitively. The robot runs only once the task command is received in the autonomous mode. Once the robot system is initially configured, it will automatically run upon receiving the task command. Waiting, scheduling, and execution are three available state types in autonomous mode. The waiting state is one of the most energy-saving states of the robot. While in this state, the robot awaits the action to take place [16]. Additionally, there are two ways the robots can accept a task: either they receive instructions from the user to perform the task for them (passive cognition), or they find the task on their own (active cognition) [17]. The robot changes the status to schedule when it receives task orders. Multiple service robots within the service area are used to configure their status according to the task they are supposed to perform, while the others remain unchanged. A service task is assigned and managed once it has been received. Once it has been executed, the status changes to complete [18]. Once multiple tasks have been completed, the scheduling status will not return until all functions have been completed according to the priorities. The status will return to the schedule once all the tasks have been conducted [19]. In this case, if there is no task, it will return to the waiting state and continue to wait for service tasks. Accordingly, service robot states are directed by service tasks to improve the adaptability of service robots.

2.1.2 MOBILITY AND AUTONOMY

A robot's mobility refers to its ability to move freely between locations within a given environment in order to accomplish its tasks. The robot is considered non-autonomous if its movements are controlled remotely by another person [3]. As opposed to this, autonomous robots use various sensors to evaluate their surroundings. As long as the assigned tasks are achieved, the robot's motion is controlled by the sensors' measurements. The robot does this without any operator intervention. There are two main categories of robots in the world: industrial robots and service robots [3]. Robot applications have historically been built into industrial robots used in manufacturing and industries. Typically, industrial robots are only intended to carry out simple, repetitive tasks in standardized and restricted environments. The industrial robots have expanded rapidly because of this advantage. As opposed to industrial robots, service robots are designed to move autonomously and in various environments. Due to this, the field

of service robots has experienced challenges, and they have not been able to grow quickly. Nonetheless, service robots can be applied to the home, office, factory, and many other places where people live and work. It has moved from industrial technology to a market focused on consumers, homes, and services [20]. Numerous companies and laboratories have developed and demonstrated the concept of service robots [21], but it has been not easy to introduce them into homes or workplaces. A few commercially viable service robots have been developed over the past three decades, including Sony's AIBO [22] and iRobot's Roomba™ [23]. Since these practical service robots are easy to use and simple to operate, they have proven extremely effective at completing their assigned tasks. Robots of this type usually need a high level of intelligence and autonomy to function properly. It should appear natural for them to interact with humans to visit desirable locations and behave humanly. Also, they should be able to manipulate objects to recognize their surroundings, and so on [11]. The abilities of these service robots need to be interactive assistants for humans can be obtained through these features. A standardization study should be carried out before the service robots can be commercialized. A variety of service robots are considered in this research. An alternative development method for a service robot is suggested to commercialize a service robot. In addition to presenting an alternative form of service robot, the robot's locomotion has also been considered. On the other hand, general service robots can perform a wide range of functions instead of specialized service machines. It uses two manipulators' arms and several instruments to perform various tasks as humanoid robots. They are one of the most common types of general service robots. Besides, humanoid robots are considered as an alternative development form of service robots, as their locomotion is carefully considered.

Mobility is required for service robots to perform tasks in rough and complicated environments. Service robots can achieve locomotion in many ways: wheeling, walking, hopping, and so forth. When considering the primary tasks given to robots, researchers must consider what type of locomotion to use. Furthermore, each method must be considered for its merits as well as its shortcomings. Unlike industrial robots, service robots are divided into many categories. Various types of service robots are available with different capabilities.

2.1.3 ROBOT NAVIGATION

With the growing popularity of service robots, the importance of smart navigation for mobile robots has been recognized more and more. Sensors should enable robots to detect ambient environments, and they should act immediately in a real-time mode to combat situations that may arise. Researchers have been working hard to create a robot capable of performing navigation elegantly. Although many research results have been produced on autonomous navigation, it is not easy to enable a mobile robot to navigate in a real environment confidently. But this study presents a method for intelligently navigating service robots utilizing technologies for smart environments. Intelligent navigation comprises four major components: localization, mapping, path planning, and control. Similarly, each performance is highly dependent on the others since each functionality influences the others. Thus, it is considered crucial to use a system architecture that allows these functionalities to work together in order to achieve intelligent navigation with high reliability. The navigation management scheme comprises three main elements: a semantic map, wide area localization, and wide-area localization. By interacting with the components of the smart environment, gathering useful information about intelligent navigation would be very beneficial. This work proposes a semantic map structure for encapsulating topological, metric, and semantic information about the environment, and a wide-area navigation algorithm based on the semantic map has been developed. This research focuses on a tool we built to help service robots

navigate intelligently using the information contained in semantic maps. Although it is necessary to construct certain infrastructures to support our approach, we can expect that a smart environment will be able to provide better navigation abilities for service robots. The algorithms used to navigate autonomous robots are generally the following: map building, localization, and path planning, as shown in figure 2 below. In order to accomplish autonomous navigation, these algorithms need to be operated at high linkages and depend on each other. However, each algorithm has a lot of uncertainties and disturbances in real-life scenarios. Due to the complexity and dynamic nature of environments in which robots will operate and the fact that they have a variety of patterns in common, the uncertainty of an overall navigation system will be magnified, making it very difficult to implement autonomous navigation in practice.

The smart environment could solve navigation and localization problems more reliably. The operating server provides an environment map that provides topology and semantic information to the robot using location sensors. The map is distributed through a sensor network, and location information is distributed to the robot through the sensor network. Robust path planning, map building, and robot localization are all possible with reduced uncertainty. On the other hand, a comparison is made between existing methods and smart methods that use sensors to determine the environment's characteristics. Navigation in an unknown environment is a challenging task for robots, but it is necessary. Furthermore, in smart environments, previously created maps are stored in the environment server system and then made available to robots whenever the robot requests information about that map. With our smart, environment-based navigation methodology, the tedious and exhausting process of producing maps is no longer needed. Based on the existing system, maps are typically classified as either 'occupied' or 'empty'. However, in our study, we propose a map structure that enables wide-area location and navigation in smart environments by storing topology, metrics, and semantic information about the location. Using XML expressions, we can represent our map. We can also standardize our data structures in order to enable easy sharing and standardization.

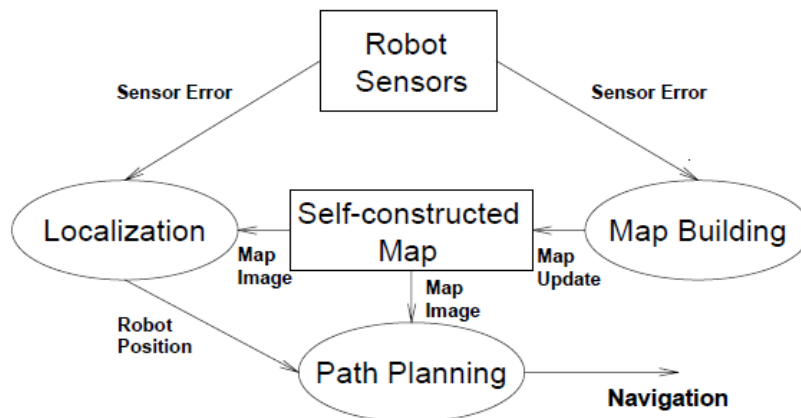


Figure 2: General schematic for robot navigation

2.2 LITERATURE REVIEW

Day after day, the integration between the robots and the human in the same environment increases; this significantly benefits us as humans in facilitating many of our tasks, but not for the robots. The service robots typically work in dynamic environments where people are walking around. The human presence poses a new challenge to the robot in that it must plan its motion in a controlled way, taking human comfort and safety into account [24]. These results indicate that robots should be able to detect human presence and recognize potential human interactions [25]. In the past, the main research focusing was on the robot's stability and the controlling methods [26]. In the past few years, many hardware sensors and drives started to be vital products, as researchers integrate the available models with specific functional robots to produce tasks like a hospital robot, house robot, delivery robot, etc. As a result, the need was raised for a significant amount of research on the topics related to robot navigation and map-making systems. A navigation system is needed for the robots to identify specific areas and locations to accomplish specific tasks. [27] used small devices fixed in various places within the robot's workspace to provide it with information that supports the robot's guidance. At the same time, a hybrid mapping strategy was applied for the robot's navigation [28]. Moreover, [29] presented a new navigation algorithm fully supported by the SLAM algorithm (Simultaneous Localization and Mapping), which can be used to generate a map and detect obstacles. In contrast, some researchers focused only on simple navigation tasks, as shown in Lei's research [29].

The use of the robot varies with the applications of the researchers involved. This allows them to try some simple and not complicated navigation tasks. In the following research, ultrasonic sensors have been used; the authors [29] developed a navigation system based on particle filtering and an obstacle-avoidance technique to determine where the robot is on the map and avoid obstacles. As part of [30] study, the limitations and advantages of the most commonly used localization techniques were explored, including but not limited to the sonar and radar. In contrast, Andreas [31] developed a system for tracking moving objects to predict their future positions before executing a dynamic path plan. On the other hand, [32] presented an RGB-D camera-based indoor robot. The main conclusion over the past discussion is that robots equipped with single sensors cannot precisely perform navigation tasks. Therefore, the idea of designing a robot platform integrating multiple sensors is a feasible solution. Based on that in our project, six ultrasonic sensors are used: three in the front half, the rest on the backside, and a Lidar sensor to increase the systems accuracy and reduce the percentage of the error to the maximum, which will ensure that the robot works in a safe and dynamic environment. The literature examined the most optimum algorithms that should be used to enhance robot mapping and navigation in a wide and deep fashion. According to [32], a newly proposed method that improved SLAM algorithms have been developed with a furniture and household object detection CNN network (Convolutional Neural Network) in order to optimize a robot's movements, as it is possible to generate a SLAM map and to detect the room in an unknown environment simultaneously. A new localization algorithm using vision-based localization enabled [33] to develop a new robot controller. This method was combined with several odometers and sensor fusion techniques.

The SLAM algorithm depends on a camera using an image processing technology to identify whatever it is programmed for. Thus, visibility is a critical and significant point. [34] discussed the SLAM algorithm in the scenario of low visibility indoor as a weak point, e.g., due to smoke or the dust in many outdoor scenarios. Instead of using the SLAM algorithms and the robotics operating system, we intend to develop a novel method that teaches the robot about its environment without using a camera. Changing the map will be developed to enhance the user experience, as any user for the robot is supposed to be

able to update the map with no prior knowledge in robotics or even coding. Our robot extended with more sensing input information, e.g., sonars, Lidar and IMU sensors, which will overcome most of the weaknesses of the other systems discussed in the literature. A study conducted by [33] revealed that some design methods already implemented into ROS (Robot Operating System) resulted in poor theoretical analysis and lackluster navigation performance study. Furthermore, a dynamic obstacle like pedestrians will greatly influence the experimental result since such barriers will remain on the map for some micro or milli-seconds due to the non-real-time operating system. In that case, it is difficult for the robot to plan its path around an obstacle when the obstacle area is very large or even if it moves dynamically. [35] revealed the same results in their research regarding the current challenges that involve real-time determinations of routes under dynamic environments. It requires constant evaluation (recalculation) of the determined routes (trajectories)—the limited time for generating a new free collision trajectory with multiple dynamic obstacles is still a serious challenge. A major contributor to this situation is the time-consuming perception algorithms [36], which force the motion planning decision window to be dramatically reduced. They are therefore unable to overcome this limitation with their current implementations. This problem may be solved by combining object detection results with the cost map used in the move base package to reduce the effects of dynamic obstacles [37]. But, due to the limitations of device sensor accuracy, the overall experimental result still isn't accurate enough.

Several articles covering sufficient details about robot path planning have been analyzed in this review. A survey has been conducted among the most popular techniques involved in robot path planning. It has been observed that researchers are working on reducing time and complexity [38] as They concluded that there are several factors involved in path planning that need to be considered in order to produce the best possible results, including length, stability, efficiency, and safety of the robot. While authors have attempted to reduce complexity by using one algorithm initially, it has emerged that no single algorithm can meet all the requirements. So, they suggested that to produce the most consistent results. Algorithms need to be hybridized that minimize error-producing effects, combine robustness with response time and precision, and deal with errors as efficiently and intelligently as possible. This thesis focuses on developing a customized source code that works along with the Fuzzy logic algorithm, which requires a complete circuit that can run in a real-time scenario that is compatible with a microprocessor rather than using ROS, which indirectly contributes to overcoming the difficulties and weaknesses mentioned in the literature regarding the real-time planning calculations on the non-real-time scenario.

Developing intelligent planning algorithms is one of the key issues when designing robotic architectures [39], [40]. So, since our robot is planned to serve food in three relatively high levels pallets, the system's stability is required and mandatory to ensure food safety from falling. For the robot, three stages are critical: start moving, turn left or right, and then stop. One of the project contributions is to design a special customized python code will be developed to control the change in the wheels speed ratio based on the live updates feedback from the ultrasonic and the Lidar, as this discovers the need for a real-time operating system that will guide to high accuracy in applying the speed changing directly after receiving the area information, as the ratio will vary based on the available area around the robot. Once there is enough space, the robot will rake a higher angle to ensure a more smoothy in turning, but once we have a smaller space, the two motors will keep rotating in the same direction but at lower speeds, reducing the turning curve smoothly. This code must be used with the fuzzy logic algorithm, allowing us to apply the Pulse Width Modulation technique. While to achieve a smooth start/stop action, the proportional-integrated differential (PID) algorithm will be used.

There is surprisingly little literature on services robotics in the form of a literature review due to their relatively short history and the fact that they are interdisciplinary. Recent literature reviews tend to be divided into technology-focused and human-focused studies. As part of their overview of fuzzy set models for humanoids, Kahraman et al. [41] highlighted recent technological advancements. However, although fuzzy sets are useful for developing humanoid robots, they have noted that the current fuzzy sets are insufficient to accurately model complex human behavior. To understand how human/managerial aspects of service robots interact, a three-part framework has been proposed [42]. In this framework, robots are designed, customers are defined, and service encounters are characterized. Several studies have examined how acceptance of service robots in travel and hospitality is influenced by factors such as employee loyalty and consumer trust [43]. An article published in the Electronics journal by Lu et al. [44] evaluated the impact of service robots on customers and employees. In their study, the authors concluded that current service robot research is fragmented, conceptual, and concerned with the stage of initial adoption. Additional literature streams should be included from information systems, computer science, and engineering in their recommendation. Given the limited scope of literature reviews in this area, the present study aims to comprehensively investigate successful models, methodologies, techniques, and technologies for service robot development. This objective is met by conducting a systematic literature review in order to identify pertinent papers discussing tools, standards, implementations, practices, and any other technology related to service robots and other robots in general.

2.2.1 NAVIGATION: THE EXPLORATION PATH

The main task of a rescue robot is to explore the environment. The mapping process and victim search are reliant upon exploration. As a result of the robot's navigation capabilities, it will be able to navigate autonomously. This will improve navigation efficiency and effectiveness, decreasing operator workload and particularly helpful in situations in which communication is interrupted. There are many obstacles that exist in the environments in which search robots operate, such as starting points, targets, and obstacles of unknown sizes and shapes. As a result, the starting point has been identified, but the final destination remains unknown. Thus, the robot moves towards its final destination by moving from its starting point. The robot must be able to find a continuous path that is free of obstacles so that it may cover the entire environment. In addition, the search engine should be able to determine where it is within the environment and know when it has been successful. The mobile robots are capable of autonomously planning their motions in real time and navigating from one place to another safely. Three factors are required in robust navigation: path planning, self-localization, and motion control. In the first place, path planning is defined as finding a route that avoids collisions from a given starting point to a given destination [45]. The robot localizes itself in the environment by estimating its position in relation to specific objects [3]. Motion control refers to the capability of robots to convert sensory information into accurate physical movement in a realistic environment [3]. It is not only a complex and technological problem, but also determines how autonomous a robot is and how reliable it is at performing its assigned tasks. Since the 1970s, robot navigation has received extensive research [46]. The navigation problem has been approached with a variety of solutions and techniques, but it remains challenging. Instead of reflecting the limited capabilities of navigation algorithms, it reflects the need for secure and reliable methods of obtaining and collecting data about the environment so that it can be included in the navigation map [47]. In addition to the ongoing difficulties in robust robot navigation outlined by Negenborn [3], other problems include limitations of computational power (CPUs), difficulties detecting and recognizing different objects, and complexities when avoiding obstacles.

The task of exploration in a rescue scenario is comprised of two parts: First, decide the next destination, taking into consideration exploring the environment as quickly as possible and that there may be places with greater chances of finding victims; After the robot reaches the target position, it must move through an environment with cluttered and rough terrain. Depending on the environment to explore, the First decision involves various issues. When exploring an unknown environment, SLAM algorithms are employed, and in general, these algorithms should be coordinated with exploration decisions (e.g. [48]). The amount of error accumulated when localizing or SLAM is small enough not to cause a significant problem when closing loops, especially in smaller environments such as Rescue Arenas. Thus, we can safely assume that exploration can be accomplished without SLAM (that is, without the need to re-visit known locations to refute current localization hypotheses) since there is no need for SLAM to assist. We use a scan-match-based localization approach in our system ([49]), but determining which positions to examine depends entirely on unexplored frontiers, as described in [50]. Frontiers were chosen because we are interested in exploring unexplored territory on the map, and utilizing them allows us to think beyond just the possibility of reaching unexplored territory. The solution to the general motion planning problem is extremely compute-intensive when a simple problem formulation is used. Several experiments have shown that the "generalized mover's problem" is PSPACE-hard. Thus, we develop heuristic algorithms that can quickly identify various options. Generally, they can relax some of the rules, such as completeness and optimization. Additionally, some assumptions made to solve this problem do not hold in a rescue environment.

For example:

- If the robot is to negotiate narrow passages, it is essential to consider the size and shape of the robot before assuming that it will move at a safe distance from the obstacles.
- Furthermore, the sensors do not provide accurate data, and they suffer from a discretization error; furthermore, some maneuvers may be unexpectedly failed due to undetected obstacles, so one cannot assume that any maneuver is possible.
- the map is unknown beforehand. Therefore, a straightforward application of path-planning and exploration techniques in a rescue context cannot be successful. As a result, maneuvering is so tricky because it requires extremely precise maneuvering, but at the same time, it requires planning long paths to avoid difficult/blocked passages.

Increasingly complex robot motions require probabilistic methods to reduce computation costs compared to deterministic and geometric approaches. The future will see these controls applied successfully to the autonomous control of many manipulators and Kino dynamic vehicles. A method such as this relaxes the requirement of completeness to probabilistic completeness while still guaranteeing formal guarantees about an algorithm's behavior. Both Probabilistic RoadMap [51] and Rapid Exploring Random Trees [52] are probabilistic algorithms for robot motion; they compute a graph or tree of possible paths and then choose one of those paths to reach the target. For robot navigation, it is common to use a local and a global planner. Local planners calculate steering based on sensor information gathered from the local map, whereas global planners take into account the

entire environment. Using this method, the problem can be decoupled by first solving a simple path-planning problem and then finding a trajectory that can follow the computed path. The method used by [53] includes constructing a topological graph of the environment, from which a topological path can be determined for the low-level motion planner to follow. The following are the main advantages of the proposed approach:

- Probabilistic Roadmaps and Growing Neural Gases are presented as a new method of global path planning that features the ability to be observed incrementally while exploring a map.
- As far as local motion planning is concerned, the Randomized Kinodynamic Planning method has been enhanced with an interleaved planning and execution method, feedback control, and online pruning.

Throughout the remainder of this section, the local motion planner and a high-level topological path planner are introduced to reduce the search space and improve computation performance by devising a path for guiding the execution of the local planner.

2.2.1.1 NAVIGATION STRATEGIES

There are different navigation strategies based on whether a hostile environment contains static or dynamic obstacles [54]. They can be classified as unknown and known environments, respectively. During the latter, information about obstacles is provided before the motion begins. Many different algorithms have been developed to address the robot navigation problem throughout a wide range of environments [46]. Any navigation planning algorithm assumes that the robot is aware of the locations of the start and target, along with a direction between them, in order to find an optimal path. Depending on the algorithm, additional environmental information or even a map might be required. In terms of navigation algorithms, Zhu, et al. [55] separate global planning from local planning.

Global navigation planning

The algorithm searches a graph representing a map of the global environment to determine the robot's path from the start to the goal. Off-line or online graphs can be constructed respectively. Before the robot begins its trajectory, the navigation algorithm determines the best path based on the comprehensive map initially loaded into the robot. The authors of Jan, et al. [45] present some optimal path planning algorithms for locating images and images within their environments. The view of the environment is one of the discrete cells in which the robot appears. One criticism of this method is that it involves a camera that is fixed in position. Huiying, et al. [56] employed a similar approach in which they combined Voronoi diagrams with the Dijkstra algorithm to find a path for robotic navigation. On the other hand, offline methods assume that the robot moves in a static environment and has a precise motion system that meets navigation requirements. The statements in this statement are unrealistic and so this method is rarely used to navigate robots in reality.

Compared to the online method, the online technique uses a navigation algorithm that updates the environmental map continuously as the robot collects data. By doing so, the robot can calculate its position within the map, and adjust its position accordingly, while navigating in dynamic environments. A navigation algorithm has been developed by using an A* search algorithm in conjunction with potential field methods and Monte Carlo localization (MCL) methods. Graphs of visibility were generated using images generated by a camera and image processing software. After using A* search to plan, we applied a potential field method and applied a path to avoid obstacles. We continuously update the robot's steps in the environment with the MCL algorithm. Nguyen Hoang, et al. [12] developed an algorithm for determining the optimal path in a simulation and avoiding obstacles. All environmental information could be examined simultaneously through both of these strategies [11], [12]. Nevertheless, recalculating the path due to changes in the environment is extremely costly. Global planning methods have three inherent weaknesses: they are difficult to construct, expensive to compute, and they are hard to construct a graph model that is accurate. In order to model environments and to construct exploration paths for mobile search robots, global navigation planning approaches are typically used. Robots are given a set of points to use to cover their surroundings completely by utilizing Fukazawa, et al.'s [5] point-distribution, path-generation algorithm. It continuously looked for an object while it traveled along the path, and once it found one, it relocated it. In the study, the robot sought out the shortest path that encompassed all points. Assuming the robot has an exhaustive map of the environment, we conducted this analysis. The researchers said zigzag paths, spiral paths, and random walks could cover the entire environment in exploration applications. Random walks failed to guarantee that the exploration task would be accomplished, according to the authors. By joining line segments within the environment, the other two techniques generate the exploration path. There is no doubt that generating a path costs more the more line segments there are. Another study proposed an efficient approach for modeling the search path by minimizing the expected time required to find the target [57]. In that work, assumptions were made that the mobile was equipped with efficient sensors, that the environment containing the object was fully understood and that the robot's motion strategy allowed the robot to find the target rapidly. [58] divided the known environment into regions for the robot that searched for multiple targets. The robot had to determine the correct sequence of motions to reduce the time it took to find the targets. A robot was not described in detail as to how it discovered and recognized objects in [57], [58]. In addition, these studies included simulations of robotic operations rather than actual robots.

There have been attempts to avoid building a comprehensive environmental map by some researchers. On the basis of online sensor measurements, such as crossing lines, Tovar's [59] robot created a minimal model of the world around it. Using a simply connected planning environment, the authors introduced a visibility tree as a means to encode enough information dynamically to generate optimal paths. In a study [60], an approach was described for guiding a mobile robot from a starting point to a target location. A trial has the benefit of enabling the mobile robot to reach the target location with minimal autonomous navigation skills required. In order for the robot navigation process to work, however, the trail must be shaped.

Local navigation planning

As opposed to building a global map [55], local navigation algorithms use sensor data directly to control the robot's movement. These algorithms are applied to guide the robot along one straight path in such an unknown or dynamic environment. To navigate, the robot must avoid obstacles in its approach and constantly monitor essential information, such as the distance it has left from where it is now and where it intends to go. Local navigation algorithms are often simple to construct and are ideal for real-time applications. The local navigating technique extensively uses the potential field algorithm [60]. An artificial potential field surrounds the robot to determine its location. The potential of the target position attracts the robot, whereas the possibility of obstacles repels it. To control the robot's movement, the robot calculates the potential field as it moves toward the target, then determines the force induced by this field. As robots typically move from one field to another, they move from a higher potential field to a lower potential field. We construct the optimal potential field to prevent the robot from becoming trapped in a local minimum before reaching its target, but it is impossible to create one. Therefore, it is combined with other navigation algorithms for increased efficiency, as described in [61].

One of the most popular navigation techniques, the Bug algorithms [2], has the advantage of solving the navigation problem without building full environment maps and only saving a few points of the path curve. Therefore, they are the same as local planning techniques since the robot only needs local environmental information instead of global environmental information. The algorithms will terminate its motion and report that the target is unreachable if the robot discovers no such path, that is, no local minimum. Those techniques assume that the robot has perfect localization capabilities, perfect sensors, and no size (point object), say the authors [2], who compared eleven robots in this family. This means that the algorithms have no direct application to real-life robot navigation. The Bug movement strategies seem appropriate since the robot is designed to navigate in an uncertain, constantly changing environment [2], [55]. When navigating in unfamiliar and changing environments, autonomous service robots face their greatest challenge, which is maintaining their safety. Here is an example of motivation: imagine that a robot enters a building to deliver a package to a particular office, and this is its first time doing this. Since it has an unknown environment, people move around by changing the environment. In addition, there is no global information such as GPS data, the system does not have a global map of the environment. A robot cannot reach the targeted table without human assistance, so it requires a human to assist it. The robot could follow a human to its destination so that it could track and avoid potential obstacles while it followed the human. Instead of simply following a path until you reach the second intersection, then turning left and entering the second door, the doorman could provide a set of high-level, humanlike commands to the robot, and the robot could execute these commands independently. Similarly, if someone did not know where they were, they would act the same way in an unfamiliar building. By utilizing this strategy, you will be able to locate the destination even without a global map and without requiring global localization. This is because only basic sensing capabilities such as corridors, intersections, and doors will be required. In order to solve this type of problem, we propose and evaluate robot navigation strategies as one of the objectives of this work. The field of robotic navigation has seen a few attempts since its emergence [62]. These strategies are categorized as reactive. Detecting obstacles that have recently been detected causes robots to react by avoiding them, making a deliberate decision to do so and ignoring the

consequences. Maps of the environment can be used by robots to plan paths avoiding obstacles and complete missions.

Autonomous robots have been proposed with a navigation architecture that does not require global mapping or localization. In the proposed strategy, local coordinate frames are used to prevent globalization. These frames are only necessary when the robot is planning a path within its field of view or following a previous path. If they are not present, the robot can do nothing. Due to the short interval between two frames, it is possible to determine the coordinate transformation between two frames with some uncertainty. The reason for this is that we can use standard wheel odometry. It is also possible to avoid moving obstacles within the range of the sensor in addition to static obstructions. Due to autonomous robots' autonomous functions, it is required to encode user commands in artificial vector fields to ensure that the robots behave as needed [63]. Moreover, these vector fields can also encompass social and cultural expectations. As soon as a human command is entered, the planner uses a vector field to optimize the solution. The system determines the safest path to take based on the desired behavior. It may be a legal requirement for a robot to follow a corridor close to a wall at certain points due to a social rule. Several modifications were made to reference [64] to make it the planner.

Chang's study focused on the model which also considers social rules [65], requires global metric localization, which can be difficult to achieve in practice. Given this, the autonomous service robot has the following main features. A system for autonomous navigation of service robots in unknown work environments without a worldwide map or worldwide positioning system is the first aspect of the project. Another approach involves using local coordinate frames to eliminate the requirement for global localization in order to address obstacle avoidance. Three applications of this technology deal with encoding human commands and modifying them as they occur. The proposed navigation architecture is shown in figure 3.

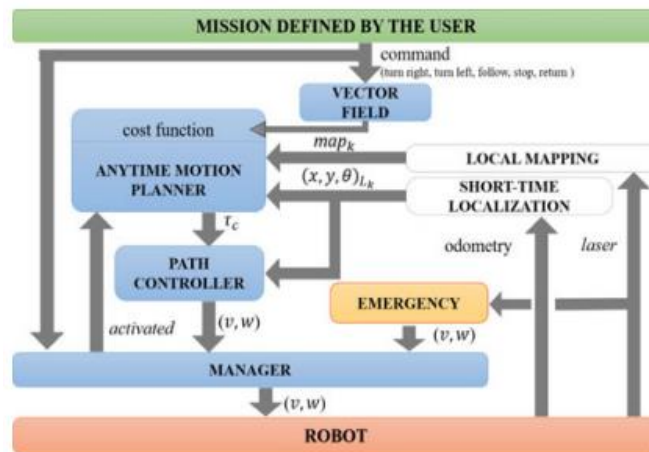


Figure 3: Navigation architecture proposed in this work [66]

2.2.1.2 ROBOT LOCALIZATION

By using whatever sensors are available, robot localization determines the robot's location in relation to certain aspects of its environment. A relative localization process or an absolute localization process can be used [3], [67].

Relative localization

Based on previous positions, trajectories, and velocities, a robot performing relative localization calculates its position. As a result, the robot should know its initial position before it can continue calculating its current location using the direction, speed, and time it has traveled [[3]]. As a low-cost and easy-to-implement method of measuring relative position, odometry is widely used. An electromagnetic compass is used to determine the direction of the robot (see [[67]]). To implement this method, wheel encoders calculate the rotation of each wheel and an orientation sensor counts the revolutions of each wheel (see [[67]]). When the robot measures its distance from the start location over time, any measurement errors caused by drift or wheel slippage will compound over time.

Absolute localization

With the absolute localization method, a robot can estimate its current location by computing the distance from a predefined location, regardless of previous estimates [[67]]. As a result, any measurement error is not exacerbated by the absolute method. Landmarks are used to estimate a robot's location in this method. Active landmarks are differentiated from passive landmarks. An object of the first type is a satellite or other radio-transmitting object, which actively transmits the location of the robot. Therefore, the robot doesn't need to know anything about the environment. In some cases, however, the robot may not be able to receive the signals from the active landmarks, thereby causing measurement errors [3]. Robots that operate with active landmarks (see [[65], [68]]) frequently locate their precise location with the help of the Global Positioning System (GPS). Passive landmarks must be actively observed and recognized by the robot in order for it to determine its location [47]. A robot's ability to recognize landmarks will depend on the kind of sensor it uses.

2.2.2 SENSOR-FUSION CONTROL

Using the low-cost sensor fusion is solving the problem of robot formation in unknown environments by using swarms of robots for self-localization and global path planning. The multi-followers, for example, form symmetrical shapes according to the master's location. Utilizing the global path planning strategy, the robots can avoid obstacles and reach their desired destination. Several robots follow the master robot's route to the desired location by following the path defined by him. That framework reduces the complexity of the swarm system since the advance decision is processed at the master robot, the follower issues merely the middleware commands from motor driver to actuators. The system is also deployed as service bots where robots transport particular payloads from one place to another as instructed by the operator. Furthermore, it is extremely important that the axis of the coordinate system to which the robot is located relates to the orientation provided by the

IMU coordinate system in order to facilitate the operation of the robot. Our sensor fusion approach makes use of an ultrawideband (UWB) beacon-based positioning system with an orientation system IMU and wheel odometry encoders to locate, orient and navigate our robot on an unknown map. ROS (robotic operating system) framework is used to control the robots. The robots operate through Wi-Fi or a 4G network. Throughout the system, all robots can access the position topics sent by the stationary robot. The vision sensor is controlled by deciding where to look next [69] and by deciding where to move next. This method examines all of the camera's configurations, including zooming, one by one, while in the present position. This approach will not work if the camera's settings are complex or if the image processing is lengthy. In relation to this, the authors of [69] proposed the 'best-first' strategy, where all possible camera configurations should be examined at the location of the navigation start. As a result of the application of this strategy as a first step, and if the target is detected, the subsequent search will be faster and easier. In this study, robots were equipped with stereo cameras, which were not capable of zooming in. The authors cite the concept of placing the robot closest to the target and the target at the greatest distance. The next best viewpoint is determined if, during the second stage ('where to move next'), the target cannot be detected from the current viewpoint. A high probability of reaching the next position should be followed by detection of the object. Data from IMU and beacon sensors cannot be directly fused since they are in different reference frames. It is necessary to determine the transformation between frames in order to determine the orientation of the robot within the beacon frame. In order to enhance self-localization in an unknown environment, we use sensor fusion.

Unlike lidar, the swarm robots do not use sensors to map the environment. They are randomly deployed in an unknown environment [70]. To provide autonomous navigation and warmth, the individual agent of this robot system must be continuously updated with information on its own location and that of other team members. Position and orientation relative to the reference global map should be included in this information. To further refine the orientation of the robot, the wheel odometry is combined with the yaw value of the IMU. Sensor values from the IMU and beacon are not directly integrated since their reference frames are different. Our goal is to determine the robot's orientation in the beacon frame by determining the transformation between frames. This package integrates raw pitch yaw values from the IMU with EKF [71] to eliminate sensor noise for robot localization. In order to compensate for the errors in right and left wheel velocities resulting from differences in relative wheel speeds, the feedback control method is used to calculate the left and right wheel velocities. In order to provide information to the control loop, EKF filters the robot's location, as the global frame includes the position, and the current frame includes the heading. Figure 4 is a schematic illustration of the flow of ROS-based sensor fusion using an EKF Kalman filter. As the filtered odometry is passed to the global and local path generators as odometry, the filtered odometry is used in navigation in the UWB workspace.

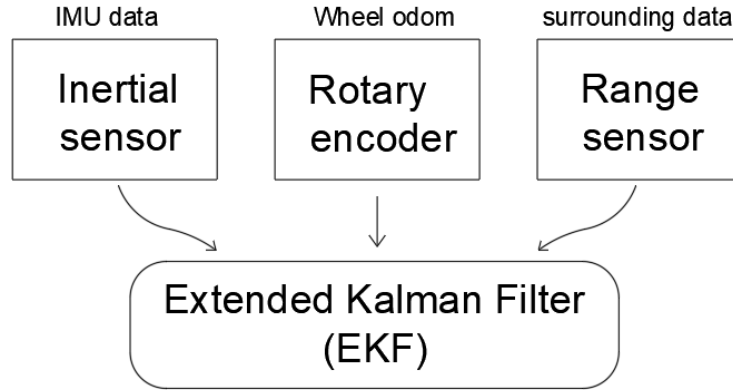


Figure 4: Robot localization using EKF sensor fusion.

Recent years have seen much interest in the use of multiple sensors for robot navigation and localization. A laser scanner is one of the most common types of range image sensors used to localize robots. In addition to vision, sonar, odometry, and radio sensor networks, localization was also achieved [65]. As a general rule, the localization problem involves determining the position of a robot using sensor data. A robot can be equipped with either a multi-sensor or a single sensor. One type of sensor, for example, is fitted outside the robot, such as an indoor GPS. In addition, there are odometry encoders, gyroscopes, and laser scanners inside the robot-like device. Currently, researchers employ multiple sensor types as composites, and many researchers combine data from sensors from multiple sources. Montesano et al. describe a method that combines the bearing-only information that cameras provide with the motion of vehicles in order to locate a pair of robots cooperatively [72]. The SLAM techniques used by Diosi and Kleeman [73] included laser scanners and advanced sonar. Batalin and Sukhatme have proposed a robot navigation algorithm based on embedded sensor networks [74]. Many systematic approaches have been proposed for localization using sensor fusion, starting with Kalman filters. There are several branches of probabilistic localization, but the branch that has received the most attention is Bayesian localization. Since both methods reduce computation time and memory consumption, the Markov Localization [75] and Monte Carlo Localization [75] methods are especially useful.

In the previous study, we used most of the odometry and range images to execute the localization of the service robot [76]. The use of robot guides may create problems for museums and other large public buildings offering guided tours. The robot's range image sensor becomes extremely polluted when people pass by or keep close to it. Situations like this are likely to happen all the time. A cranky, tender, or dizzy floor can cause the odometry data to be inaccurate, which can negatively affect the localization results. The aim of this study is to develop a probabilistic model that combines laser scan data with data from an indoor GPS system to assist in solving those challenges. Until recently, the only way to locate a service robot was via an odometry chart and a range image. Because of this, it was not always possible to handle errors. We propose an indoor GPS system and the Monte Carlo method as a probability-based localization method to overcome these challenges. A scheme is also presented that we demonstrate experimentally is effective for selecting the correct sensor in different situations. A probabilistic model is adapted for the results of a location calculation in view of the possibility that an indoor GPS system's measurement is uncertain. As part of this process, we will also

demonstrate a reasonable method for updating samples' probability. The service robot needs to be able to map a partially or completely unknown environment and maintain localization parameters in order to explore it. Accessing the range information is the primary objective of mobile robot mapping, followed by converting the range information into an internal representation. An internal representation is needed for the robot to continuously update its position. As a result, the robot can achieve full autonomy and operate without human assistance. Without updating the previous state of the environment, it is extremely difficult for mobile robots to make decisions in highly dynamic environments. In order to update the environment recursively, the system in the service robot collect the local and surrounding environmental information and merge it with another local data to enhance the sensing quality. In the past decade, researchers have made significant progress in solving problems associated with the combination and fusion of data from multiple sources for decision-making purposes [77], [78]. Information fusion is widely used in engineering, healthcare, military, and robotics applications.

The literature defined the fusion sensor systems into many key, as the Joint Directors of Laboratories defined it as process of collecting, analyzing, and integrating information from many different sources to build a complete idea about the situation around and identify threats, along with providing a complete and timely assessment of the situation. Throughout the process, estimates and assessments are continually made and evaluated to determine whether additional sources are needed or whether the process itself needs to be modified to improve results [79]. As Durrant-Whyte defines in [80]. The sensor-fusion system utilizes the data that been collected from many different sensors and merge them in the best estimation state that really describe the situation. According to Luo, the sensor-fusion is the process of combining (or merging) different sources of sensory information into one composite representation. Hall et al. defined fusion techniques as the process of combining data from several sensors with information from associated databases to achieve greater accuracy and specificity than otherwise possible [80]. "Data fusion" refers to the method of combining data in order to refine state estimates and predictions, as defined by Steinberg [81]. According to Dasarathy, "information fusion" refers to the blending of multiple sources of information, such as sensors, databases, and human-collected data. When the sources are used individually without utilizing synergies, better decisions or actions are taken (quantitatively and qualitatively in terms of accuracy, robustness, etc.) [82]. Dietz (2008) classified sensor fusion into different levels, such as 'high-level fusion'. In a dynamic environment, significant objects and events are analyzed in relation to each other as a high-level fusion [83]. As defined by Linas, information fusion is the process of assembling, coordinating, and combining data, information, and events from different sources for the purpose of estimating parameters, characteristics, and behaviors specific to entities within one's field of view. You can implement it either fully automated or with human assistance for analysis and/or decision support [79].

Fusion refers to the process of combining various sources of information during the integration phase of service mobile robot applications. Different types of information are fused together at different levels of a hierarchical model. Information Fusion can be divided into four levels, as shown in Figure 5, the Multisensory integration is represented as a composite of multiple functions. First, Signal-Level

Fusion: This involves signal enhancement techniques such as beamforming developed by using microphone arrays. In general, the resulting signal from multiple sensors is identical to the original, but it has a higher quality. Another type of fusion is pixel-level fusion, where information based on pixels is combined. For example, in CMOS and CCD cameras, such information is produced by sensors. It is possible to merge pixels pixel-by-pixel or to merge local neighborhoods of pixels in each image to create the fused image. In addition to these types of Fusion, there are feature-level Fusions, which have applications in various fields, including robot mapping, person tracking, and automatic speech recognition. By combining the scene's features with other sensors, like microphones, the scene's features are extracted from it. As a fourth method, conditional probability can be used to achieve symbol-level fusion through statistical inference.

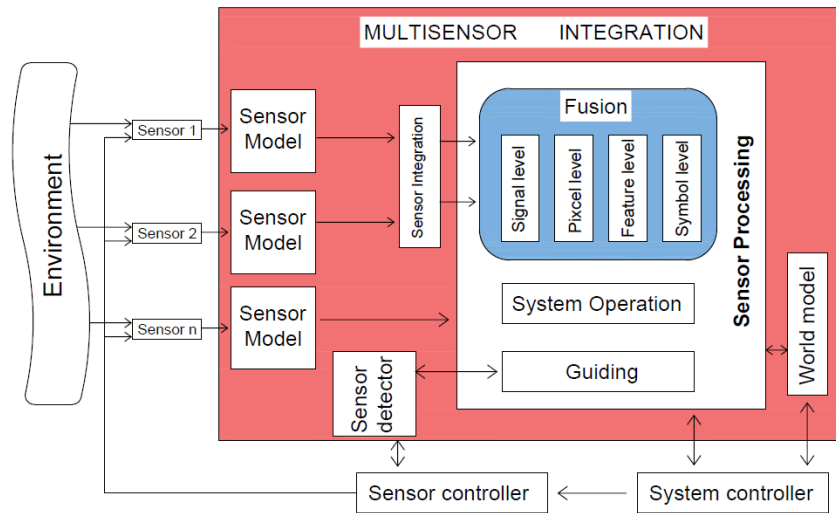


Figure 5: Functional Diagram of multisensor integration and fusion (figure redrawn from [84])

The combination of information from multiple sensing devices enhances a system's ability to perform a specific task. When multiple sensory devices are incorporated at the architectural level of a system, some general integration issues arise [85]. Structures of hierarchical integration can be helpful for representing the different levels and nodes of fusion in architecture in an efficient manner. Examples are the National Bureau of Standards (NBS) sensory and control hierarchy [86]. A review of the fusion-sensor integration systems shows any elements, ranging from the sensors to the sensors, which are the inputs for the integration process. Filtering and enhancement of the signal are commonly used to enhance the data. Furthermore, the sensor model converts the range data from different sensors into a standardized representation. Range information can be provided by a sensor in the form of voltage, current, pulse width modulated signal, or an image [51]. Prior to the actual fusion process, sensor registration is crucial to describe the unique and temporal dimensions of sensor information. When processing the sensor, the fusion is done at the symbol, feature, pixel, and signal levels. Separating the sensor's data from fusion may be worthwhile if the sensor's data differs significantly from other sensors. The navigation of mobile robots uses sensor data to build local maps based on the current location of the device. A world model is created by updating the information from the

device with previous information. Navigation with multisensory fusion in mobile robots is typically described by world models that represent high levels of representation. A multisensory system, on the other hand, uses a sensor selection procedure to select the most appropriate combination of sensors [87]. There are three general stages to selecting a sensor: a) Pre-Selection: This is the first step to identifying the most appropriate sensor based on environmental conditions. The location of the robot's sensors and its surroundings are used to determine pre-selection within a mobile environment. Using real-time sensor performance evaluation [88], the real-time sensor selection approach was demonstrated. In the event that an algorithm determines that a particular sensor is underpowered, then that sensor is excluded from integration. System Controller: the system controller sends commands to the mobile robots. Feedback signals from sensors are used to control algorithms such as path planning, collision avoidance, and navigation.

2.3 CONCLUSIONS

It is possible to analyze images and detect objects using various techniques, but they also have limitations. Researchers have combined several of these techniques to improve their performance. Utilizing a camera and image processing requires a lot of processing time; therefore, this thesis will not focus as heavily on using a camera. As a result of the robot's sensor-fusion system, the robot detects and locates the target (Table), approaches, delivers and moves it to its station back indicated by an earlier defined landmark. Depending on the algorithm used, it is either a global or a local navigation path. Almost all algorithms assume robots know where they are starting from and where they are going, and their task is to find the best route between them.

An understanding of the area to be searched is essential before using a mobile search robot. A lack of information was available regarding the robot's tools for detecting the target. The robot was responsible for identifying the target. The robot should search the entire area since the target location in this study is unknown. A comprehensive exploration plan should therefore include all possible areas of exploration. Generally speaking, our case assumes that the boundaries of the explored area are completely known, but its internal configuration is unknown. During its motion, the robot follows walls or obstacles that it encounters on its way. When it finds the target, it delivers the container to the starting position. If it finds the target, it moves the container there. When the robot reaches the starting point again, it completes its motion. In this case, the robot would follow a path until it reached the starting point again. The proposed motion planning is based on adapting bug algorithms. Developing better and more efficient systems is challenging when designing mobile service robots that operate in complex environments. These techniques can be incorporated into existing fusion-sensor frameworks, according to research. Bayesian processing is an ancient and highly dependable method. One method also had fundamental problems, including information uncertainty, conflicts, and incompleteness. Fusion-sensor techniques must be trained over a long period when dealing with complex environments with large variations. A significant advantage of using the sonar and Lidar and their ability to perform computations quickly. Indoor and outdoor environments are more readily adapted to the fusion-sensor technique as a practical matter. Sensory information must be preprocessed and post-processed, followed by an internal representation in the form of a map for

mapping and localization to be robust. Researchers reviewed localization approaches used by different authors to position mobile robots, including probabilistic maps-based localization and automated mapping schemes designed for localizing mobile service robots. Mobile robots operating in unfamiliar environments can effectively and efficiently locate themselves using the SLAM model. The localization accuracy and orientation are improved when probabilistic localization approaches are combined with SLAM. Moreover, different methods provide accurate solutions and higher convergence rates in low-noise environments. The use of RFID tags in restricted environments, such as indoors, has proven to be very effective in tracking robots. The use of evolutionary techniques in conjunction with RFID improves the robustness of these systems by estimating nearly optimal paths. We observed that the combination of RFID and SLAM could improve localization accuracy, particularly in indoor environments.

In research on localizing brain-controlled autonomous service robots, a focus has been placed on applying map-oriented and SLAM-based localization methods. Control of these robots is made possible by an electroencephalogram (EEG) signal. The brain-computer interface (BCI) automates the direct control of robots by sending commands directly to them. This type of robot can either be controlled by the BCI or shared with it instantly. An intelligent-based autonomous navigation system is used in a shared controlled model to allow the user to control robots that a BCI controls remotely. Optimal use of these systems requires robots in both categories to know their current location and incorporate appropriate path planning and navigation strategies. Developing SLAM-based RFID systems for robotic positioning in indoor environments and applying evolutionary techniques to SLAM-based localization schemes can reduce errors and guide the robot to avoid collisions in the future. The location and orientation of robots can also be determined not only by SLAM but also by continuous localization and component mapping schemes such as ARIEL and feature-based extraction. Researchers have conducted numerous EEG-powered robotic systems to determine the optimal orientation. Thanks to this research's advanced effort and success, we will be able to design robots that can plan routes and navigate autonomously or partially autonomously.

Chapter 3 : THE DESIGN AND FABRICATION

3.1: INTRODUCTION

In this work, the practical and theoretical challenges have been addressed involved in scaling different model robots for use in restaurant food service operations. This research focused also on the developing and manufacturing this robot in a commercial grade prototype level. Extensive study has been investigated different limitations on the navigation system, as well as the different strategies for planning out a travelling route (trajectory) due to many factors in the surrounding environment. Three stages of testing will be carried out on the designed control systems. As a first step, the robot should move blindly with no sensor inputs and no obstacles around it. In order for the robot to navigate within its environment, the sensors system platform for the robot has been designed and constructed. As a result, activating the sensors and the trajectory generation systems will be used in the second step, along with the methodology for avoiding obstacles by considering the shortest possible route. A major objective of this work is to overcome the theoretical difficulties associated with scaling model robots to industrially relevant sizes. Researchers may be able to identify methods of overcoming these limitations by identifying limitations in navigation systems and path planning methods. Testing the functionality of the developed and proposed control system will be conducted with a service mobile robot, as described in the first and second chapters. Robots are designed and built to navigate their indoor environments using their sensor platform. Many industrial applications involve the use of service robots in the current day, but these robots may be obstructed by objects dropped on the ground. In spite of the fact that these obstacles are small, and the robot can navigate over them, the robot will consume large amounts of energy. As a result, the robot is forced to follow the obstacles' boundaries, resulting in a longer travel time and an inefficient navigation path. Due to the way the caster wheels create a continuously supporting surface on the surface of the two-wheel robots, they are easier to maneuver on soft and unstructured terrains. However, they are having a difficult time moving and overcoming small obstacles.

Service robots are used in a wide range of industrial and real-life applications. Despite this, the obstacles still pose a serious challenge to the robot's movement, so it should be able to navigate around them. In the event of any sudden obstacles, the robot will continue to follow the predefined trajectory and update it in the event of any sudden obstacle's appearance. In general, wheeled robots work less well on rough surfaces, such as outdoor, unpaved terrains. In general, prototypes can be divided into four categories: Simulation prototypes, in which a simulation could be used to create something that looks like the final product, rough prototypes, High fidelity prototypes and Commercial grade prototypes: usually ready for patenting and marketing, which is the case in this research. It is a challenging task to be able to describe what an autonomous mobile robot (AMR) is and how it works in a few words. However, the benefits that come from the utilization of an autonomous mobile robot (AMR) can be summarized in a few different ways. The main benefit of using this technology is that they are able to move in a specific space without human assistance, in a fully autonomous state, excluding manufacturing operations. When referring to the naming, mobile refers to

technologies of autonomous driving, while service refers to the role of robots. Figure 6 illustrates the driving unit for the project in terms of mobility and the application unit in terms of service.

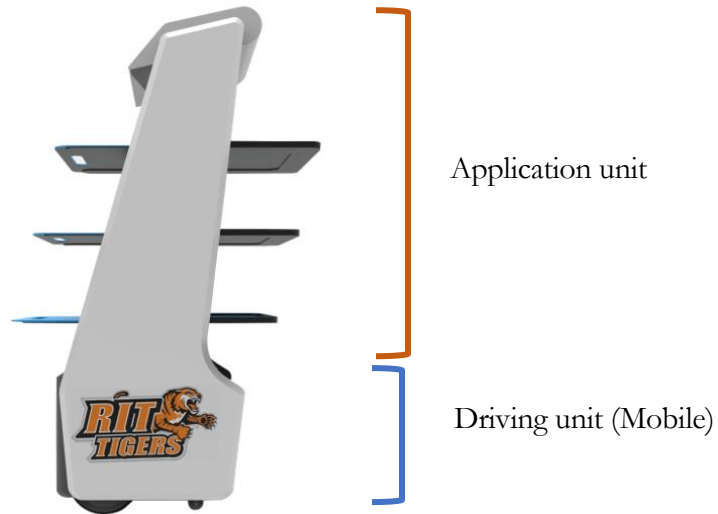


Figure 6: The main units that describing the AMR robot

Unlike AGV, AMR does not follow a defined path, but creates it. While the AGV runs autonomously based on a guided path. It moves according to various guidance methods such as QR code, magnets and rails which are mainly used in the workspace. The mobile robot mechanism requires different inputs to autonomously drive such as sensors, radar, lidar, and camera to recognize the environment. While it needs a motor, battery, and IMU to drive it.

3.1.1: Conceptual design

Successful things are always the result of careful planning. All of the things we use today were once concepts, like the first automobile, electricity, and our smartphone. Coming up with a plan for its design is a crucial part of refining a concept. That is where conceptual design fits into this category. Even though design engineers use conceptual design all the time, it is a technique that is applicable to any field. Designing a new product involves a multiphase process, starting with conceptual design. It is important to begin by coming up with a general concept before starting a project, whether it is for building a software program, machine or even a robot. After conceptualizing a design, the schematics are drawn. An integral part of conceptual design is convincing the project supervisor that the idea is worth pursuing and has the level of thesis work. Schematic design entails reviewing the feasibility of the concept as presented. Even so, it does not mean it is impossible to evaluate the feasibility before selling the concept. An initial project brief is often the guide for the design team, and it is the stage at which information is gathered and markets are researched. In recent years, projects have combined the conceptual design phase with the schematic design phase. This is the stage of project development when they use the term “concept”. The most challenging aspect of any product design is identifying

a working idea, and figuring out how it will be looks like. Creating an idea is affected by a variety of factors, including market design trends, raw material costs, local and international competitors, as well as client needs. The literature showed that today's industrial projects became more and more technically complex and logistically challenging, which exposes the manufacturing operations to even more complex constraints. Thus, there is a need for a better understanding of how to build and design a commercial-grade prototype service robot with the required intelligence to work and navigate autonomously in an in-door restaurant for serving food, or in a specific industrial plants at a reasonable price, and with more outstanding capabilities. Since the RIT university has already shifted to the new campus, the need raised to enhance the food delivery experience for students by using this project robot which will be able to interact with the students, display the menu and carrying the plates without going through the conventional food serving method. After defining the idea and doing enough extensive research, the idea has been finalized and it becomes ready for initiating the prototype. The rough design idea has been represented by preparing some hand sketches, 2D CAD design and finally by using the Inventor 3D CAD models. Due to the variety of disciplines that been used to create this project, a conceptual design was required to break down the project work, and to ensure covering all topics in a clear and organized manner. The main topic is represented by the term 'project' which produce many subsystems that contain the components which divided into elements. Figure 7 is showing the organized sequence.

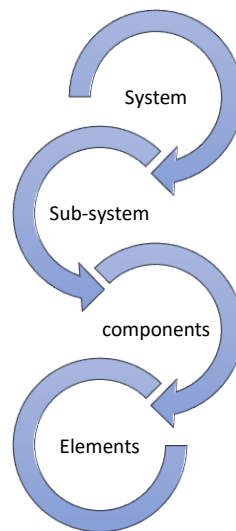


Figure 7: Project conceptual design sequence

Conceptual design initiates the process of multi-stage development. Conceptual design phases, which kick start any new project, typically consist of defining the problem as a design brief, conducting background research, and specifying requirements as what has been discussed in chapter 2. When these phases get completed, they proceed to brainstorming solutions and choosing the best one, at which point they are able to start real project planning. The conceptual design schematic is shown in

Figure 8. As soon as the solution has been decided, it is time to progress to the development phase and build a prototype. In addition to testing the prototype before it can be built, the design should be reconsidered several times until all problems get resolved. The engineering design steps are usually ordered a certain way; however, design teams are not required to follow those steps precisely. The decision may even change the order or go back to a previous phase once they are further along in the process if something is discovered that needs to be addressed.

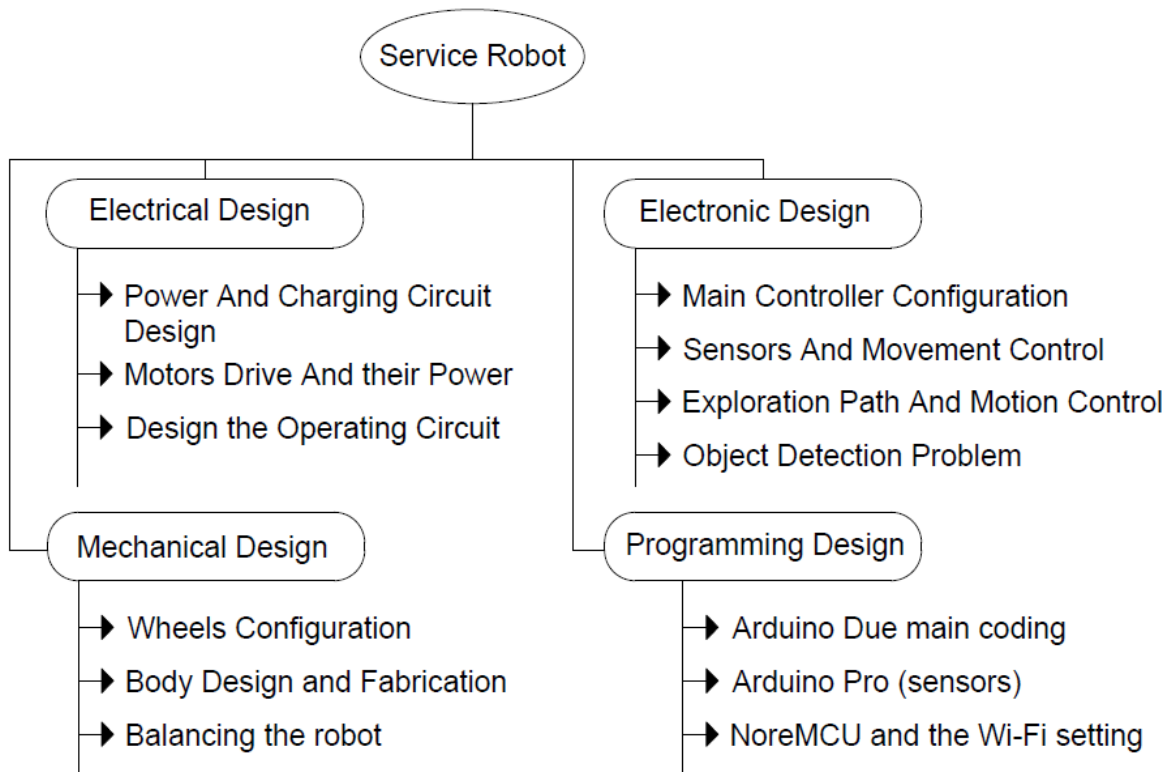


Figure 8: The main project component breakdown elements

3.2: Mechanical Design

The mechanical structure consists of the base robot body (Aluminum), Acrylic body, wheel configuration, and the motor drives. The service robot is like any other robotics mechanical machine, contains partially a mechanical components. In this part, the working principle and the applications of the mechanical components will be discovered, and will provide an overview regarding those components and the reason behind chosen them. Also, understanding the simple mechanical components provides the necessary background to explore more complex systems in future to be used with many types of mobile service robots.

The mechanical component of this work has tested different methodologies within some software and electronic components. Consequently, a completely new wheeled robot was developed and implemented to test the proposed methodology. The robot must have an on-board processor that has a high computational capacity so it can do the implementation of all algorithms within the controller, there are two main aspects to consider during this robotics project. Firstly, it is essential to use a board supports real-time operating systems, which require a high processing speed and special components; therefore, the robot will move quickly to the targeted position in the off-line mode, preventing unnecessary delays and interruptions. In addition, the board must be quick during the entire process, which will allow the robot to initiate its trajectory faster with more processing speed. This chapter will provide an overview of the mechanical design and the development of the platform for an autonomous wheeled mobile service robot. The food will be served with this robot as it will move toward its target upon receiving the instructions from the user in an indoor environment. The robot is primarily composed of both mechanical and electronic components. This section will discuss the mechanical structure which consisting mainly of a robot body (Aluminum chassis), two wheel derived by two stepper motors supported by two spherical caster wheels.

3.2.1 ROBOT BODY FABRICATION

3.2.1.1 Chassis

Main body frame is consisting of Aluminum square cross section profiles to give the stability and to focus the weight down of the robot. Also, it used to give some strength, stiffness, and rigidity for the robot. Due to the fact that this project is consisting of many mechanical parts and electronics, the need raised for a customized frame for the enclosure to contain the main parts/weight. The extruded aluminum channels profiles are coming in a variety of shapes, sized that include many channel configuration and connecting accessories to ensure an enough secured joints and links. The 20x20mm aluminum profiles in figure 9 have been used in addition to many different related fasteners shown in figure 11 , which used for reinforcing such as, the Aluminum corner brackets, M5 x 10mm T-Slot nuts, M5x10mm hex socket screw bolt, and the L Type 3-Dimensional Bracket 3-Way Corner Connector to ensure the body building in an exact 90 degrees.

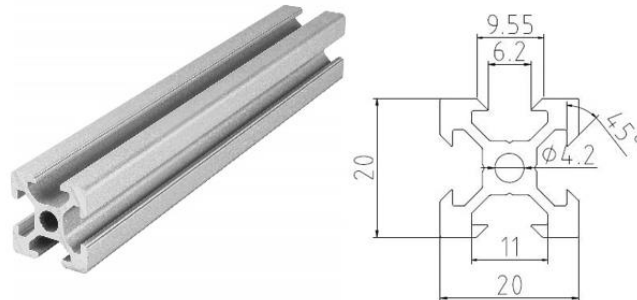


Figure 9: Extrusion Aluminum Profiles (T-slot aluminum profiles)

The entire robot's base with the dimension of 502 x 425 x 233 mm has constructed using the previous mentioned 20 x 20 mm Aluminum profiles in a total lengths of 3240 mm long, 1920 mm wide and 932 height producing the frame that includes two floors divided into 8 bars 405 mm long, 4 bars 480 mm long, and 4 bars 233 mm long as shown in figure 10.

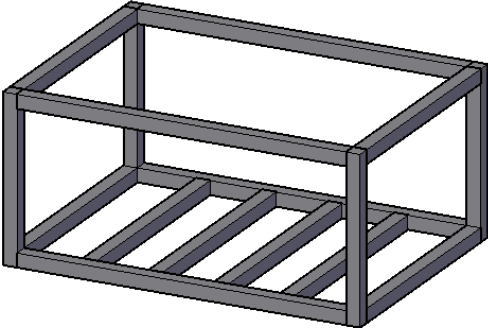


Figure 10: 20x20 mm Aluminum profiles base structure

Besides, the same dimension Aluminum profiles used to fix the motors in their location strictly form all sides to ensure a proper alinement for the robot movement.

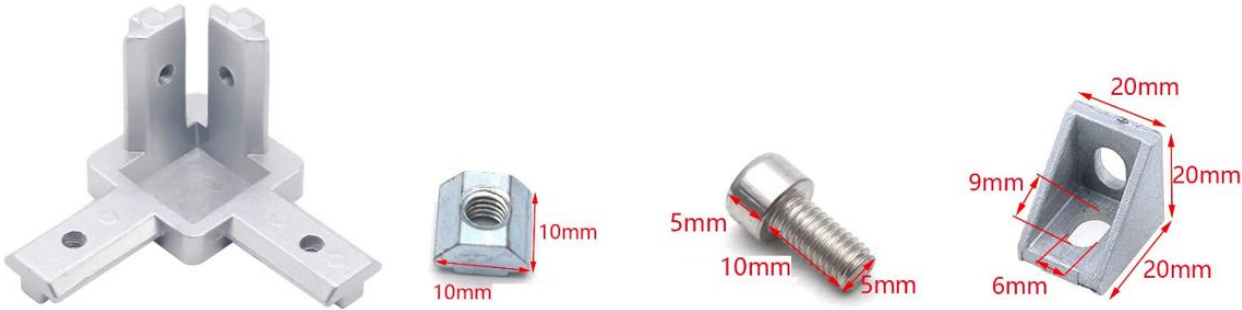


Figure 11: Aluminum profile fasteners: (a)the three-way corner connector, (b)T-Slot nuts, (c)Screw bolt, (d)corner brackets.

The bottom floor has built using 4 separated holding bars to support the entire body, and to have the ability to fix and support the motors as show in figure 12, each 405 mm long. It is used to attach the Servo motors, motor's drivers, 12 VDC batteries, and the charger.

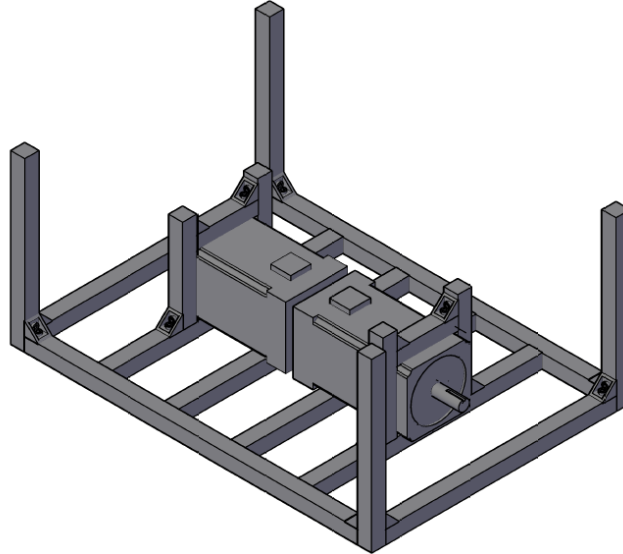


Figure 12: The Aluminum profile used to fix the motors

The upper floor of the frame is shown as a support and cover for the box, however, there is an internal acrylic floor used to attach the controllers (Arduino Due, Arduino Pro, and NodeMCU), and holding two DC to DC converters, and the USB hub. The Lidar range sensors is also placed and mounted at the same floor with will discuss later.

3.2.1.2 Body cover and trays holders

As a second item/material, acrylic sheets are used to build the body. This special plastic material has transparency similar to glass. The high impact resistance of this material enables it to endure high loads for a long period of time, which lends itself to a large number of applications. Besides having the option to order it in a range of colors, some people also choose to use it in high-quality transparent. Total of eight ultrasonic sensors are placed around the robot in eight different positions (two sensors per side) that were already opened by laser on the acrylic sides, front, and back. Figure 13 is showing the front two ultrasonic sensors.



Figure 13: The ultrasonic on the robot's body

3.2.1.3 The main robot frame body

After building the aluminum profile base, the need arises for the acrylic sheet to cover the base and to hold the trays and the robot screen above. Also, it is used to mount the ultrasonics on the sides, as shown in figure 13. The whole body has been implemented exactly same as planned and designed, no single variation indicates the precise and accurate design. Both CAD 3D design, and the actual robot photo are shown in figure 14(a) and (b).

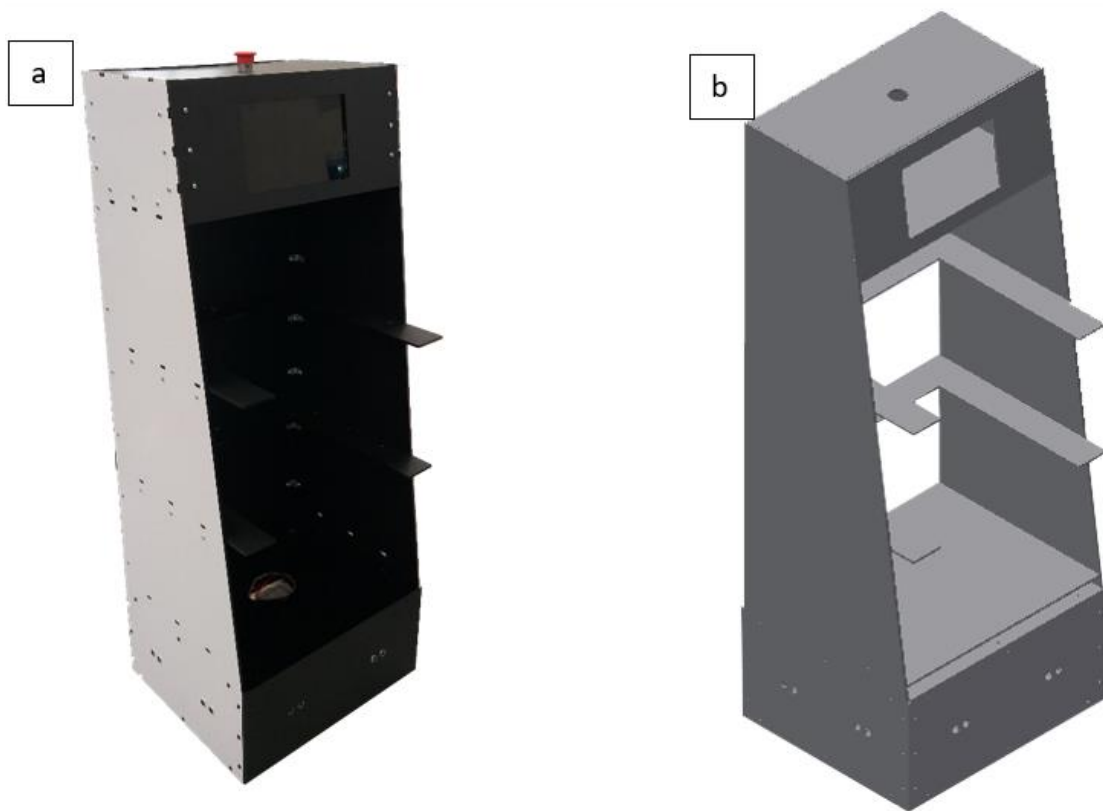
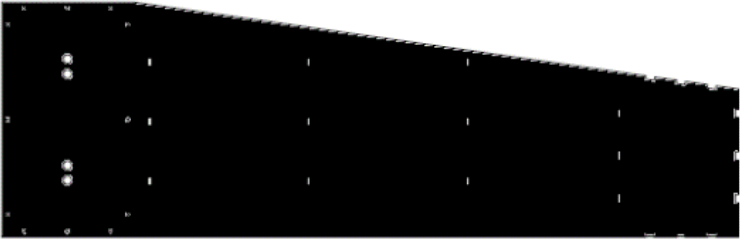
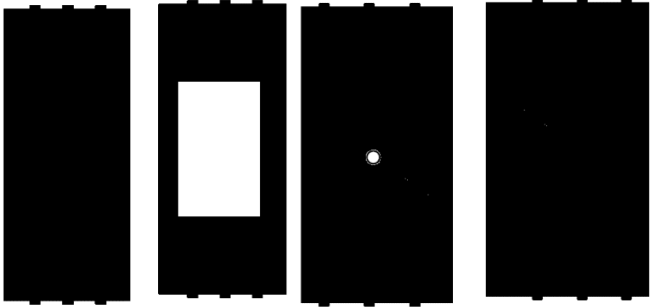



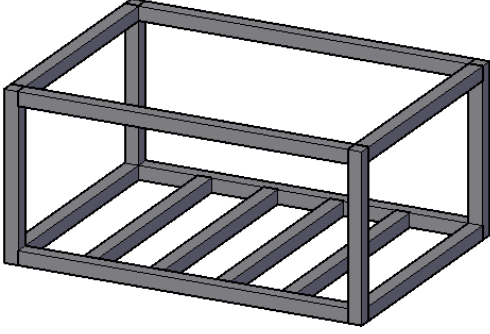
Figure 14: (a) Real model of main body of robot, (b) Inventor 3D design model of main body of the service robot

The table below is showing the main parts of the body and reflecting each one of them with an illustrative picture. The main items are mentioned below:

1. Main body
2. Trays
3. C-shape tray holder
4. Top cover
5. Lidar sensor cover

Table 1: Main Parts of the robot's body:

Part name	Required number	Picture
Side panels	2	
Top box	1	
Tray slider C - Shape	2	

Aluminum frame	1	
-------------------	---	--

3.2.2 WHEEL CONFIGURATION

Using two spherical caster wheels with a two-wheeled drive was chosen for this project as shown in figure 15. In this configuration, the front pair of wheels are rotated in a particular direction, allowing a robot to reorient itself. The back pair of caster wheels act as followers to guide the robot in the direction it is moving. When the robot is required to drive in a straight line, its wheels will rotate in the same direction at the same speed. Wheels configured in this manner have some advantages over alternative designs. A wheel-based locomotion system is the most popular method for mobile robotics because they are simple to program, easy to control, and require relatively little mechanical engineering. Additionally, the wheeled motion is also more efficient on structured, flat surfaces when compared to legged movement. There are various types of wheel and motor drive configurations on wheeled robots. As a result, wheel configuration plays a vital role in controlling and stabilizing mobile robots. For robots, wheels can be configured in various ways, including two wheels, four wheels, or even more. The following describes some wheel configurations consisting of a different number of wheels. This project is a robot with two wheels, as some robots do initially. Differential drive mechanisms equipped with two wheels are usually used to move or change direction. The differential drive mechanism allows mobile robots to drive their wheels independently from each other. Therefore, the robot does not need an additional steering mechanism since the wheels can change their relative angular velocity in order to adjust their direction. Two-wheeled robots need to be balanced accurately and have their weight distributed across their wheels in order to be stable, and they are less agile than four-wheeled robots.

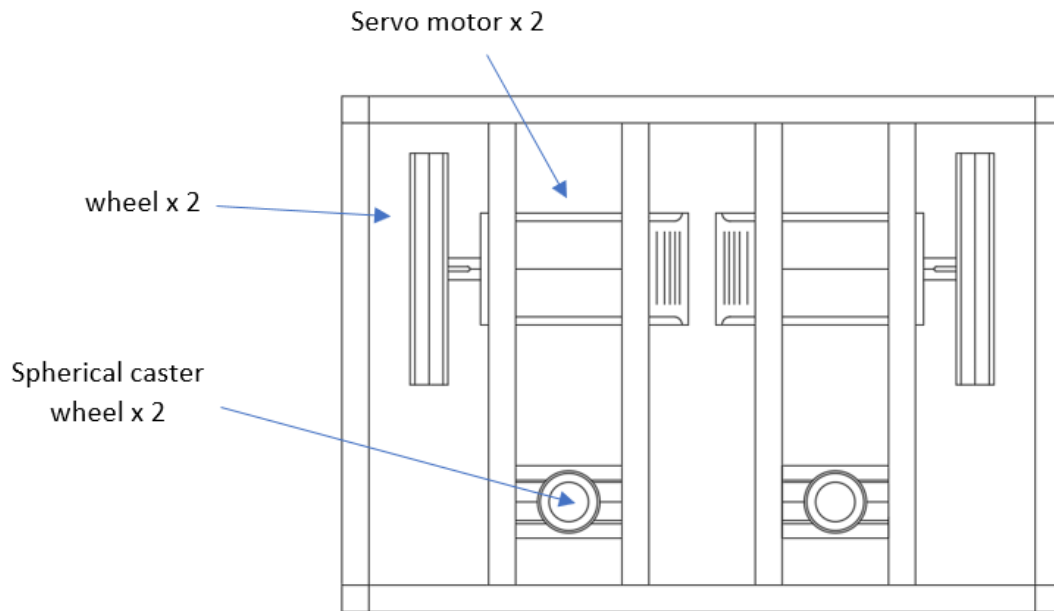


Figure 15: 2D CAD model for the base bottom design

It is common for robot designers to include a third wheel on their machines to improve their balance. By using a stepper motor or a servo motor, the third wheel of the robot can be driven, allowing it to steer. In addition to being easy to operate, this type of drive, called a tricycle drive, is efficient. Due to this drive, however, the robot must wait a long time before it can change direction. Some robots use a three-wheeled drive system to integrate the drive and steering mechanisms. For this wheel, there are two motors, a DC motor to drive the movement and a stepper or servo motor to provide the steering. The other two wheels rotate freely and are symmetrical. This is a complicated design due to the complexity of building both the steering and driving mechanisms simultaneously. In addition, it has the disadvantages of the previous tricycle drive. Synchronized drives are used in other three-wheel configurations where two motors are used. The three wheels are driven by two motors, and the third motor steers them all. It is possible to build synergistic configurations for robots that move straight, but the mechanism requires considerable development.

Due to the stability features of the four-wheel configuration, many designers tend to this configuration. Some wheeled mobile robots have steering mechanisms similar to those found in cars. Several motors provide motion to the robot, and one or two of them are used for steering. Two or one motors are attached to the robot's back or front, and one or two engines are attached to the back or front. The problem with this configuration is that the robot must travel a long arc to change directions. However, the robot can also be driven using differential drive instead of four wheels. It is easy to implement differential drives into robotics because they are commonly used. While it is possible to control the steering of this robot, it is more complicated than controlling the steering of a robot using a car-like configuration. Finally, some robots are equipped with motor drives that rotate

at zero degrees. The wheels mounted on one side of the machine are rotated in the same direction, while the wheels on the other side are rotated in the opposite direction. All of the wheels on the robot's legs rotate simultaneously and in the same direction when it wishes to drive straight. In addition to moving to the desired position and then turning in place to achieve the desired orientation, the four-wheel configuration will help ensure the robot's stability. Therefore, the four wheels control the steering and motion of the robot, making it more powerful. In addition to its many benefits, the use of four wheels with four motors has some disadvantages, namely the robot's controllability; it has been difficult to perform straight-line motion control since all four motors must rotate at the same speed. On the other hand, precise wheel alignment is essential to ensure a straight line of motion. It is imperative that all wheels are in contact with the ground simultaneously in order to maintain straight-line motion. The robot motion method also requires powerful motors on both sides to achieve the zero-point turn radius.

3.3: ELECTRICAL AND POWER CIRCUIT DESIGN

Any machine or robot is requiring electricity source to provide the power for the complete circuit, as it supposed to provide the controllers with the suitable voltage and current to control the operations and the rest of components to energize them. This project is requiring an electrical circuits to control, drive, and sense the robot operations, as these can be achieved by a combination of many electrical components in a circuit.

3.3.1 THE POWER AND CHARGING CONFIGURATION

In order to provide the robot with power all the time during its operation, batteries and charging systems have been used to achieve the best approach circuit in terms of performance and durability. Before purchasing, it is necessary to do proper calculations for the appropriate selection. According to estimates, the robot makes 160 deliveries per day (20 deliveries/hour) and works for eight hours every day. In order to estimate the robot's power consumption, the weight must also be considered. Then, the amount of electrical power needed to charge batteries can be determined.

Three main forces have been considered: Force for acceleration(F_a), Force for Tire resistance(F_r), and the Force for Air Drag (F_d).

$$(\text{Total mass} * (\text{max velocity}/3.6)) / \text{time to achieve max velocity} \tag{1}$$

$$\text{friction coefficient}(0.02) * [\text{total mass} * 9.8 \text{ (normal force)}] \tag{2}$$

$$(0.5 * C_d * \rho * \text{max velocity}^2 * \text{Frontal Area}) \tag{3}$$

Then the total force is the summation for equations 1,2, and 3.

Total Force = 50 + 9 + 50 = 109N. Thus, the Peak Torque Requirement in the unit of Nm is 11.1. At this stage, the total wattage in watt is calculated by the formula of $(\text{RPM} * 0.1472) * \text{Peak Torque} +$

Power required by electronics) which give 655.2 W. By dividing the Peak Torque Requirement (Nm) on the number of motors; 2, we ended with 5.5 Nm for each motor. The Power per Motor (W) is calculated by dividing the Peak Power on the no. of motors. This give a 327.6 W. the peak power is used to find the Energy Required in Battery Bank (Wh) using the formula peak power * (milage / nominal velocity) which give 100.6.

Finally, the required Ah battery capacity at 12V is calculated by the Energy Required in Battery Bank (Wh) on the used voltage, then multiply by 2. This gives a 16.8 Ah. By assume that the batteries are working on 80% of their capacity. Then $16.8/0.8 = 21$ Ah. So, the selection went to using two batteries 12Ah for each connecting in series to give 24Ah which achieve the requirements smoothly.

Table 2: The required calculation for the motors and batteries selections

Technical Calculations		
FORCES		
Total Mass of robot (M)	45	<i>Mass of Robot</i>
Force for acceleration(Fa)	50	<i>{Total mass * (max velocity/3.6)} / time to achieve max velocity</i>
Force for Tyre resistance(Fr)	9	<i>friction coefficient(0.02) * [total mass * 9.8 (normal force)]</i>
Force for Air Drag(Fd)	50	<i>(0.5 * Cd * ρ * max velocity^2 * Frontal Area)</i>
Total Force (F)	109	<i>Fa + Fr + Fd</i>
RESULTS		
Peak Torque Requirement (Nm)	11.1	<i>Wheel radius * (Fa + Fr+ Fd)</i>
Max Wheel RPM Requirement (RPM)	522.4	<i>(Nominal velocity * 16.667) / (2 * pi * (radius of tire / 39.37))</i>
Power Required by Tablet and Electronics	50.0	
Total Wattage (W)	655.2	<i>overdrive at startup (RPM*0.1472)*Peak Torque) + Power required by electronics)</i>
Torque per Motor (Nm)	5.5	<i>Peak Torque / no. of motors</i>
Power per Motor (W)	327.6	<i>Peak Power / no. of motors</i>
Energy Required in Battery Bank (Wh)	100.6	<i>peak power * (milage / nominal velocity)</i>
Ah for the batteries at 12V	21.0	

Ultracell 12V, 12Ah batteries have been used, the rest of components are listed in table 3 as a summary:



Figure 16: (a) 12 V-12AH Lead Acid Rechargeable Battery, (b) 12V battery charger

Table 3: The main items required for the power circuit

Item name	Description	Quantity
Lead Acid Rechargeable Battery	12VDC, 12AH	2
Lead Acid Battery charger	12VDC or 24VDC	1
Normally open switches	N.O Switch	5
Emergency stop	N.C Switch	1
DC-DC Buck Converter	Step down	2
Wires, luges, and connectors	-	-

Two batteries and one charger have been used to charge and run the robot of this project. Figure 16(a) and (b) are showing the used units. Generally, multiple batteries of the same voltage rating and similar capacities can be paralleled with no problem, even at slightly different states of charge. However, the most important note is doing the parallel operation accurately, + to + and - to -, otherwise, it will cause a serious issue. One way to make the parallel operation safer is to do the actual paralleling with an appropriate fuse. Once a parallel connection is established for a while, the state of charge of the batteries will have equalized, at which time they can be treated as one battery. The charger could be connected to the paralleled and equalized pair to continue to charge the battery pair. By using this method, many more batteries in parallel could be added as long as the proper polarity is observed for each one added.

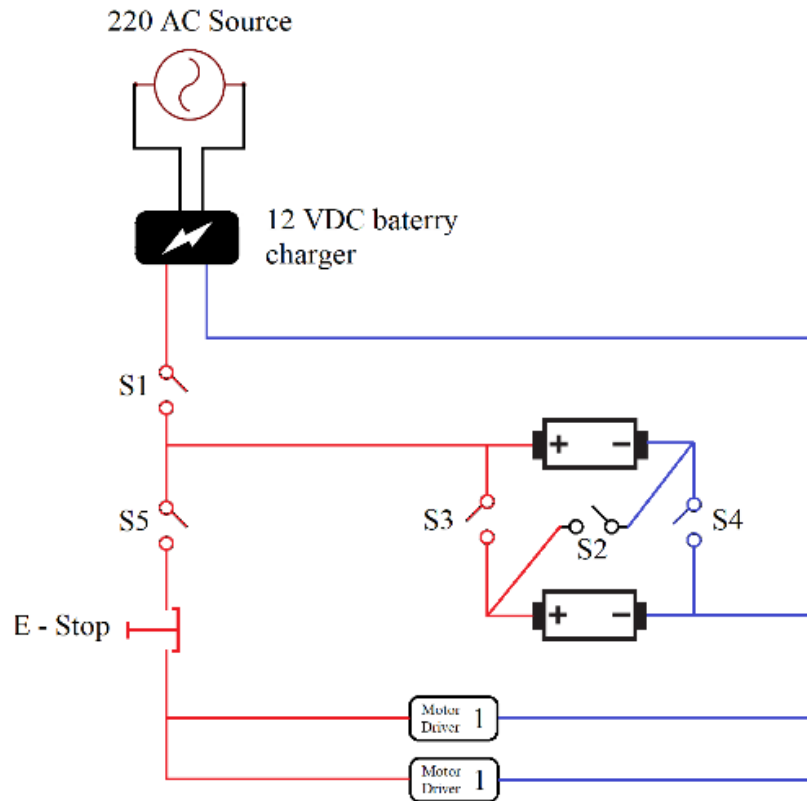


Figure 17: The powering and charging circuit connection

The power circuit must provide a 24VDC, as the used stepper motor in this project (Model No. 34HS38-4204D-E1000) is requiring 24 VDC to run on its full capacity, two lead acid batteries are used in a series connection to provide 24VDC. However, charging the batteries with an external charger while they are in series is practically not recommended, as this will usually result an overcharge and causing the batteries to be faulty after some time. So, the best way to charge them is to change their circuit to parallel during the charging, and change it back to series during the running and operation. Figure 17 is showing an electrical circuit that has been designed and developed to accomplish charging and running circuit connections.

The used 12DC voltage battery is a popular Lead Acid battery voltage. Figure 18 and figure 19 are the typical chart for a generic 12V Led Acid Battery. It is important to check the battery voltage while it is being discharged with a light load. Without any load, a spurious voltage will be shown which would be higher and won't accurately represent the actual state of charge.

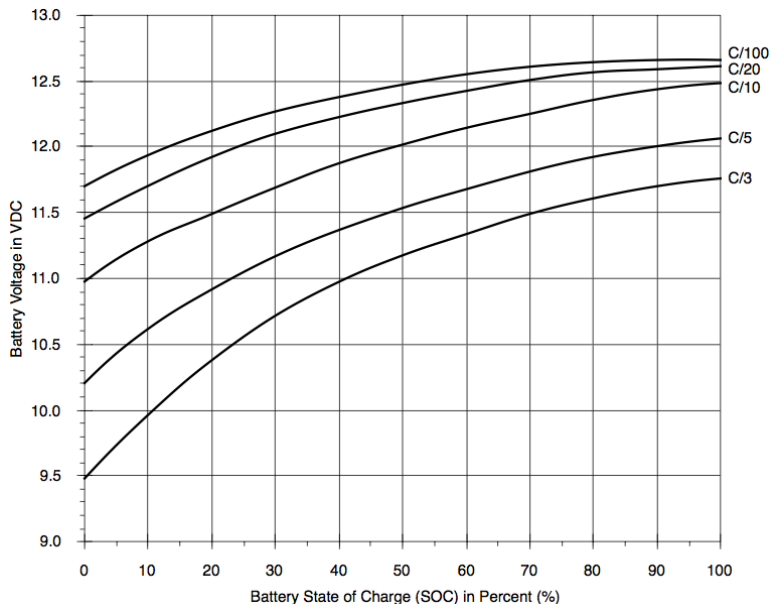


Figure 18: 12V lead Acid battery state of charge (SOC) vs. voltage while under discharge

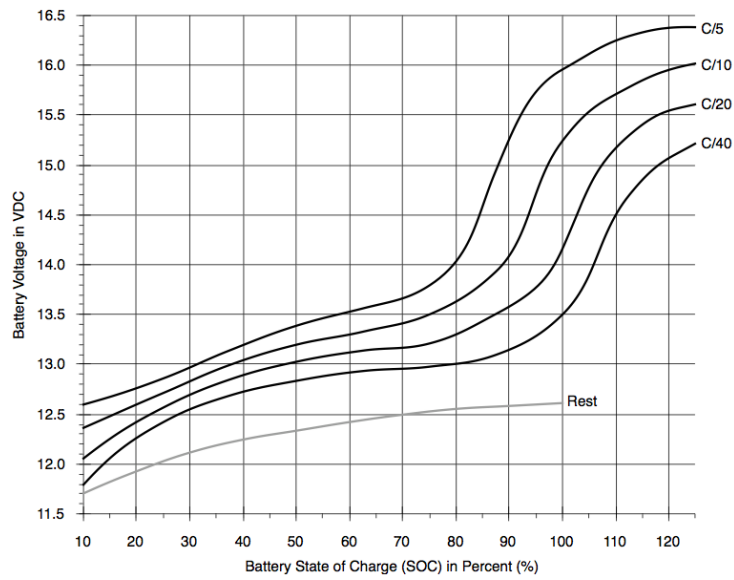


Figure 19: 12V lead Acid battery state of charge (SOC) vs. voltage while under charge

Five different normally open (N.O.) switches have been used as shown in figure 17. In the case of charging, S1, S3, and S4 would be close to generate a parallel charging connection of 12V. The voltage will be equalized in both batteries until the charger read 12DCV between its terminals, then the charger will automatically turn off. In the operation case, the batteries connection must be in series connection to provide the 24VDC required to run the motors, for this case, S5, S2 will be close, while the others open. This connection will isolate the charger and flow a 24 VDC from the batteries to run the whole circuit. During the running mode, the batteries will provide a 24VDC on their terminals. The Arduino Due is requiring a power supply in the range of 6 to 20 volts. However, if the pin is supplied with less than 7V, the board may produce less than 5V which leads to instability. When more than 12V is used, the voltage regulator may overheat and destroy the circuit board. It is therefore advisable to keep the voltage between 7 and 12 volts. So, to provide the Arduino Due with a 8 DCV, a step-down LM2596S DC-DC Buck Converter is required to step down the batteries 24 DC volt to 8DCV (shown in figure 20). The DC-DC Buck Converter will take a 24VDC as an input, and give a 8VDC as an output to feed the Arduino Due. On the other hand, another DC-DC Buck Converter is required to give a 5VDC as an output to feed the electronics and to charge the tablet which used to give the instructions for the robot.



Figure 20: LM2596S DC-DC step-down buck converter

In this particular DC-DC Buck Converter model, it allows the input voltage to be any value between 4 and 40VDC and it gives an adjusted voltage value between 1.25 and 37VDC. The voltage can be change by adjusting small potentiometer on the board. Also, it has a volt stabilizer and LED display to show the input voltage and the adjusted output voltage, the Buck Converters circuit is shown in figure 21 below.

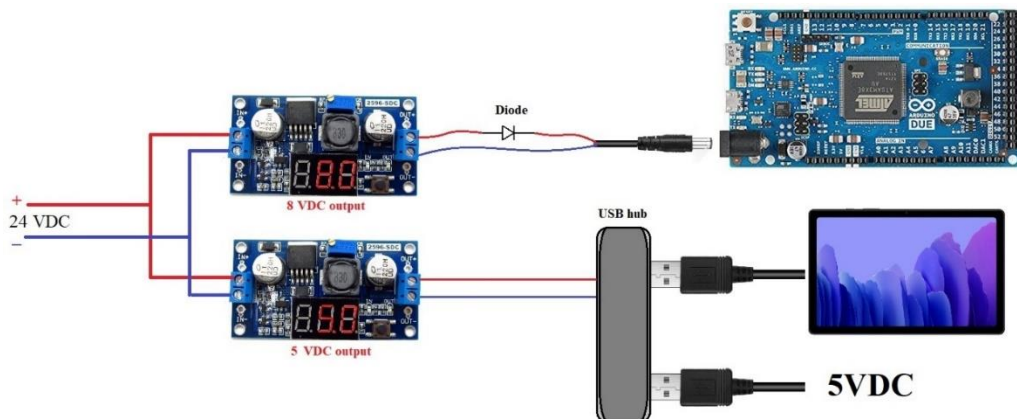


Figure 21: DC-DC Buck Converters circuit diagram

A small diode 1N4001 has been used on the positive line of the Barrel Jack Power Cable to prevent the power to go in opposite way in the case of connecting the Arduino Due with the computer using the USB micro, as without this diode, some small amount of voltage will go through the Arduino to the barrel jack terminal to flow back to the DC-DC Buck Converters and energize it. So, this diode has been added to avoid that.

3.3.2 MOTORS DRIVE AND POWER SYSTEM

An essential first step to creating a high-quality moving robot is choosing the motor that will be appropriate for its function. Ideally, the motor should be able to withstand static and dynamic loads without bending or fractioning. Several factors have been considered during the selection of the stepper motor which used in this project robot. Stepper motors may not be ideal for every application, but in the right environment, they can be beneficial. Nema 34 8.5Nm closed loop stepper motor 86EBP147ALC has been used with the driver model ZDM-2HA865 as shown in figure 22. Those are a heavy load kit used usually with the big CNC routing machines to provide robot with the ability to handle the whole weight along with the stability and the accuracy in the movement during the running mode.

Stepper motor (34HS38-4204D-E1000):

The stepper motor is brushless DC an electromechanical device that converts electrical impulses from a microcontroller in discrete angular displacements. In this case, bipolar stepper two phase motors are used, this type of motors consists of a rotor with permanent magnets and a number of coils on the stator.



Figure 22: The motor and driver kit used to run the service robot

Nema 32 stepper bipolar rotate for each step angle 1.80, so it needs for 200 step for one revolution, the current for phase 4.2 Amps, and the rated current 7.2 A, the holding torque 7.07 N.m, this has been used due to the required accuracy and the torque. The data sheet of stepper motor and the driver in the appendix.

The main, technical input data that helps in selecting the motors are listed in table 4.

Table 4: The robot’s characteristics and system inputs

INPUTS	
Mass of Robot (Kg)	45
Required Number of Deliveries per day	160
Distance travelled per day	3.072
Nominal Velocity (Km/h)	20
Time to achieve maximum velocity (s)	5
Drag Coefficient	0.4
Radius of Robot tire (in)	4
Number of Motors (Nos)	2
Width of Robot (m)	0.413
Length of Robot (m)	0.5
Height of Robot (m)	1.306

The calculation’s result already mentioned in table 5, as it was found that each motor must be minimum able to handle the load of 5.5 Nm. The calculations lead to select a motor with 8.9 Nm to handle the load easily and to give more smoothness in the control.

Table 5: Analysis results

RESULTS		
Peak Torque Requirement (Nm)	11.1	<i>Wheel radius * (Fa + Fr+ Fd)</i>
Max Wheel RPM Requirement (RPM)	522.4	<i>(Nominal velocity * 16.667) / (2 * pi * (radius of tire / 39.37))</i>
Power Required by Tablet and Electronics	50.0	
Total Wattage (W)	655.2	<i>overdrive at startup (RPM*0.1472)*Peak Torque) + Power required by electronics)</i>
Torque per Motor (Nm)	5.5	<i>Peak Torque / no. of motors</i>

Stepper motor driver (ZDM-2HA865)

For easy operation, the used driver is a complete micro-stepping motor driver with a built-in translator. An output drive capacity of up to 24VDC and a range of 0 – 7.2Amps makes this one suitable for operating bipolar stepper motors in full, half, quarter, eighth, and sixteenth step modes. This driver utilizes a digital signal processor chip to improve closed-loop control. It has a pulse response frequency of up to 200kHz. As the motor provides electronic gear matching to multiple pulse sources (no matter the subdivided value), it has 16 general subdivision options and 256 subdivisions (51200 pulses/rotations). As far as safety is concerned, the device has protections against overcurrent, overheating, overvoltage, and tracking error. The used driver is shown in figure 23.



Figure 23: the stepper motor used to control the robot motor.

The parameter settings are the key for defining the working scenario of the motor, this specific driver is composed of 4 LED digital displays and 4 keys, which is used to display various state and parameter Settings of the system. Each time the drive is powered up, the version number used by the current drive will be displayed. 3 seconds later, the status of the current drive will be displayed (standby operating speed is 0, and the current fault code will be displayed when there is a fault). When the driver enters the normal operation mode, the revolution of the current motor (revolution/minute) is displayed in real time. When the motor is reversed, the leftmost (highest) of the digital tube display value flashes. When the driver has multiple fault alarms, the corresponding fault code is displayed in turn. To begin with, stepper motors have full torque at rest, and the angle of rotation of the motor depends on the pulse input. The main advantages of stepper motors are their great speed control, their accurate positioning, and their high repeatability.

Furthermore, stepper motors are very reliable since they are brushless motors. This minimizes mechanical failure and maximizes the operation lifespan of the motor. These motors are intended to be used in an array of different environments. The reason for this is that since the frequency of pulses affects the speed of rotation, there can be a wide range of rotational speeds. Stepper motors require drivers in order to run and control their movements. This is due to the requirement for a motor driver

that does not expose the stepper motor to excessive heat or noise. Furthermore, the selection should be based on a suitable stepper motor driver that will not overpower a small stepper motor.

The speed and direction of the robot is defined by the stepper motor and how it has been controlled by the microcontroller board; however, the motor's power is coming from an external 24VDC batteries charging system as discussed earlier, as the microcontroller will not be able to handle and provide such power. Consequently, the stepper motor driver is connected directly to the batteries so it can support the required power and can also be controlled by the microcontroller. The detailed wiring is shown in figure 24.

In order to control the motor, the Arduino Due is used as a control board. There are two major cables on each motor; the first one controls the direction, and the second one controls the number of pulses. A zero-turn radius is one of the requirements for the robot. In this case, each motor pair must be connected to one channel and controlled simultaneously. For instance, if the robot wishes to move forward in a straight line, it will run both pairs of wheels at the same speed and in the same direction. Alternatively, if it wishes to change its orientation, it will rotate both motor pairs in opposite directions. As a result of using this method, the robot is able to change direction by spinning around its axis.

The robot is designed to move at a nominal speed of 0.2 m/s to facilitate the transfer of food to people around. Obviously, this is enough. There are still other factors to be identified, which will determine the maximum speed. Nonetheless, we can expect the maximum speed to lie in a range of 0.20 to 0.35 m/s. This gives us the wheel rotation speed we need to calculate. Additionally, the robot's maximum speed needs to be considered. To calculate wheel rotation speed, we need to know the wheel diameter (190 mm).

$$N_T = \frac{60V_N}{\pi D_W} = \frac{60 * 0.2}{\pi * 0.19} = 20 \text{ RPM}$$

Where:

$D_W = 0.19\text{m}$, The wheel diameter

$V_N = 0.2\text{m/s}$, The normal robot speed

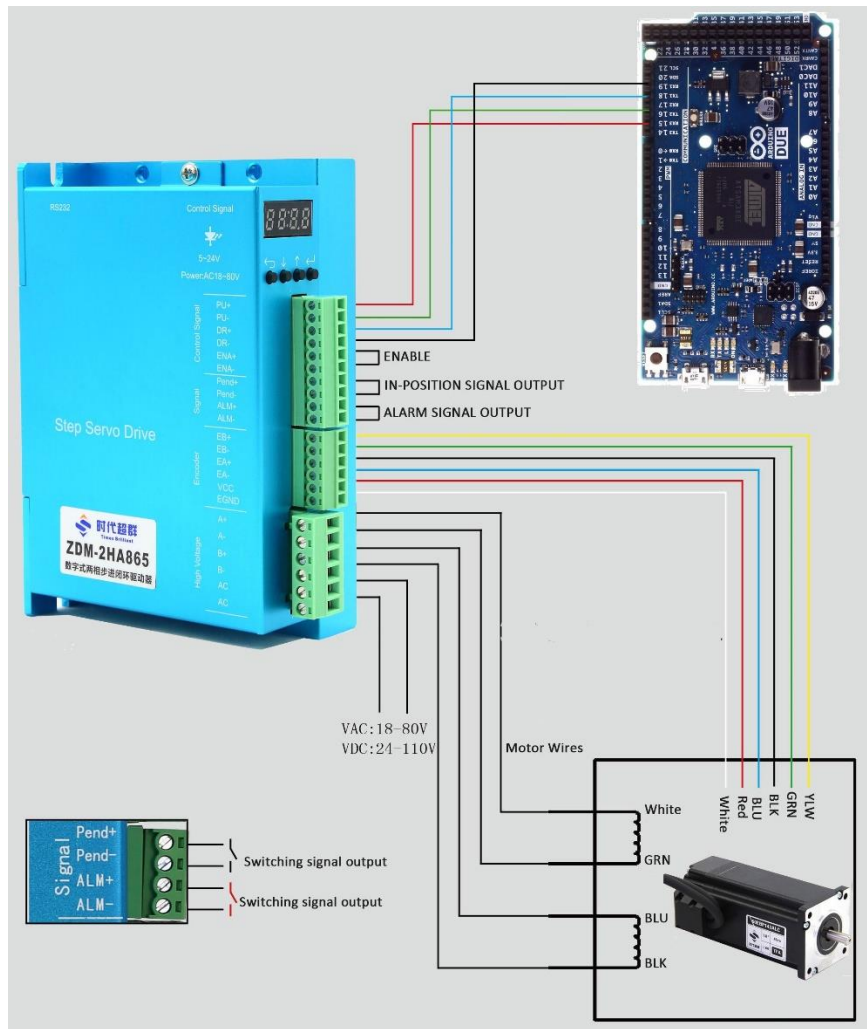


Figure 24: Detailed one motor and driver configuration.

A high motor of 12N.m Nema 34 Hybrid Closed Loop 2 Phase Stepper Motor 86J18156EC-1000 Plus Encoder Cable has been used with an 8.2A Stepper Driver Motor Stepper ZDM-2HA865. One of the advantages of closed loop stepper motors is that they can be controlled by position feedback and/or speed feedback, which can help them control the phase transition with the rotor's position, which can help them perform better. Additionally, when it comes to the closed-loop control system (which includes the closed-loop stepper driver) of the stepper motor, we should expand the operating speed range, improve tracking and positioning accuracy at a given speed, or get the speed limit and accuracy index limit.

Stepper motor and servo motor technologies are combined in the closed-loop step motor system. Compared to a standard stepper motor setup, a closed-loop stepper motor system performs better and with less resistance. Despite providing feedback and control for the closed-loop system, as well as short transient and free oscillation times, the closed-loop system will not lose or gain steps. In applications that require improved energy efficiency and smooth operation, such as those that operate at high loads, a closed-loop stepper motor system, such as the 34HS38-4204D-E1000, is the best

choice. In addition to having higher torque at low RPMs than servo motor systems, a closed-loop system has a number of other advantages. In addition to these features, short transient times, less packaging, precise positioning using encoders integrated into the motors, and comparatively low pricing are also advantages. The stepper motor driver ZDM-2HA865 is suitable for various small and medium sized automation equipment and instruments such as, engraving machine, laser cutting machine, high-speed plotter, small CNC machine tools, automatic assembly equipment, etc. Low noise required, smooth operation, High speed response equipment is especially effective.

The DSP digital step closed loop driver ZDM-2HA865 represents a new generation of DSP digital step drivers. This driver employs advanced vector closed loop control technologies. With a closed-loop stepper motor driver, compared with conventional open-loop drivers, the problem of step loss is eliminated, along with increasing the high-speed performance of the stepper motor, reducing the motor, and reducing its vibration, further increasing the working speed, and making the equipment's precision and turning degree even better. Moreover, the driver will emit an alarm signal with the same reliability as an ac servo when the motor is continuously overloaded. Dimensions compatible for motor mounting on conventional stepper motors (57/60) and 86 series motors. This driver is using a new DSP chip with a 32-bit motor control, controlled by an advanced vector closed loop control technology, as the user is able to adjust the controller setting using a LED digital tube display buttons operation in an easy to operate mode. A Static and dynamic current can be set arbitrarily in a range of 0 to 7.2 Amps, with adaptive drive (57/60) and 86 series hybrid closed loop stepper motors, the input/output of photoelectric have an isolation for the signals with a pulse response frequency up to 200KHz. In addition, it provides an electronic gear matching to various pulse sources, and protection of over-current, over-heat, over-voltage, and tracking error. The driver input sequence is shown in figure 25.

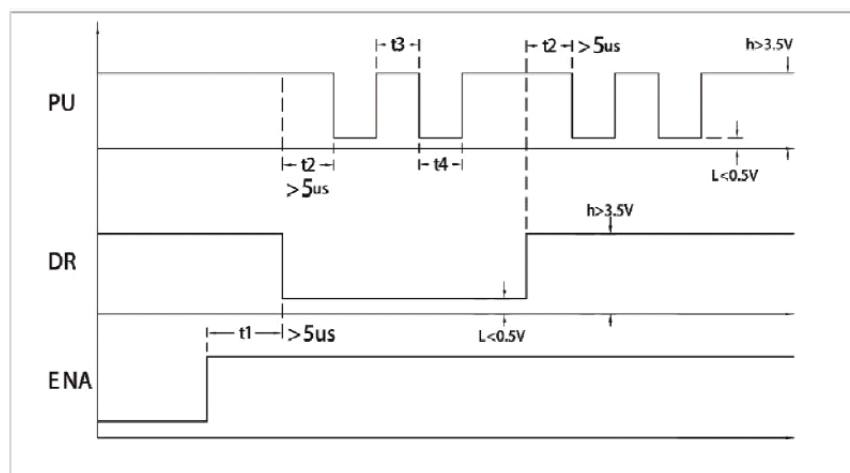


Figure 25: Stepper motor driver input signal waveform timing diagram

The closed-loop drive system cannot simply change the direction of the motor by replacing the motor line. If the motor runs in an inconsistent direction with the given direction, the setting should be change

of the parameter P002 to be “change the direction”. While when the subdivision value selected in P001 is Set, driver subdivision is defined by the electronic gear variable. The unit pulse command input to the driver can be defined by the electronic gear to move the transmission device any distance. The pulse command generated by the upper controller does not need to consider the gear ratio, reduction ratio or the number of motor encoder lines of the transmission system. It can be easily matched with various pulse sources to achieve the user's ideal control resolution (Angle/pulse).

Computational formula: $P \times G = N \times C \times 4 \rightarrow G = \frac{4 \times N \times C}{P}$

Where:

P: number of pulses for input instruction

G: electronic gear ratio, = Frequency division of molecular / Dividing the denominator

N: motor rotation number

C: number of encoder lines/r this system C = 1000

For example, when the output pulse of the upper controller is 6000, the motor rotates for 1 turn.

$$G = \frac{4 \times N \times C}{P} = \frac{4 \times 1 \times 1000}{6000} = 2 / 3$$

Then, based on the above result, the P006 parameter to be set to 2 and the P007 parameter is set to 3. The above results are calculated by mathematical deduction and the minimum common divisor are taken as far as possible. The recommended range of electronic gear ratio is $1/20 \leq G \leq 20$. Figure 26 is showing the complete circuit after adding the motors and their drivers to it, the drivers taking a 24VDC from the batteries all the time. As once the batteries lose their charge, the motors will not be able to work unless recharge them again.

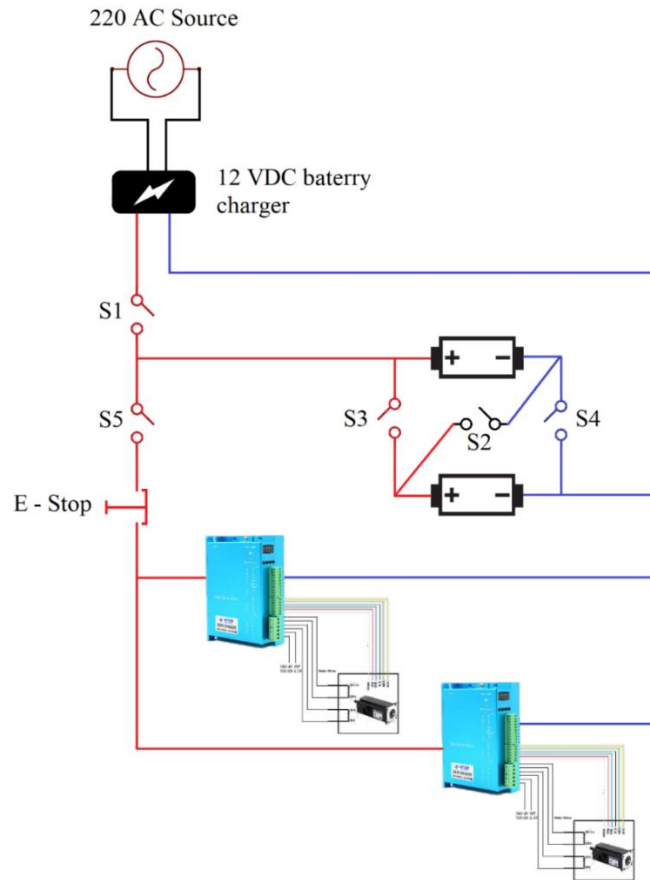


Figure 26: Motors power circuit diagram

3.4: ELECTRONIC CIRCUIT DESIGN

It is important to mention the main requirements for designing an advanced service robot before describing the electronic system. A mobile robot with autonomous capabilities is required to locate and find the trajectory to arrive at the tables and deliver food within an indoor environment. Sensors are needed to navigate and detect obstacles, as well as to control the robot's motion in this case. In order to accomplish these tasks, an electronic module system is required. Detecting a target is typically achievable through the use of sensor fusion systems that require a high level of integration of code. In addition, some sensors use low-level software (low-level processing) utilizing another controller to process their signals. In order to run high-level real-time programs, the processor should have a large processing power. Considering that it is mounted on the robot, the processor should also be in standby mode. In this case, the commands come from the robot application and are communicated via Wi-Fi between the robot application and the main microcontroller (Arduino Due). Nevertheless, this board is not equipped with the sensor and motor drivers, and I/O circuitry necessary for handling the sensors and driving the motors. You have two options to address this problem. For the first method, the Arduino Due (main microcontroller) is connected to sensors and motors via adapters. However, this creates two further problems. First, the Arduino Due lacks the required logic HIGH voltage in its terminals that is needed to drive and process the sensors; thus, the sensors cannot be operated. A second problem is that even

if the necessary software is installed, the Arduino Central Processor requires extra time and effort in order to process signals from sensors.

Using the second method would solve the previous two problems, as it uses a different microcontroller to handle and manage the low-level software (sensors). In that case, the microcontroller would be the interface unit between the robot's key electronics: the main microcontroller (Arduino Due), the sensors, and the motor driver. To implement the main electronic system, it was decided to integrate these two modules, a large microcontroller (Arduino Due) and a smaller one (Arduino Pro).

3.4.1 THE MAIN CONTROLLING PARTS AND CONTROLLERS

Arduino Due (FreeRTOS):

This is the main motherboard used to organize the working sequence of the robot, as it is working on the real-time system, which is the ideal solution for this service robot project, as there are many data need to be transfer and treat during the running program, such as the Lidar and the Ultrasonics readings, and the received data from the NodeMCU from the robot application. This amazing microcontroller board working on a 32-bit arm core. The number of the I/O pins is 54, along with 12 analog inputs. In addition, it is incorporating two DACs and two CANs, which make it an ideal board for large-scale Arduino projects. Arduino due is shown in figure 27.



Figure 27: Arduino Due board

The core CPU it has is Atmel SAM3X8E ARM with four hardware serial ports different from any other Arduino board, its clock working at the frequency of 84 MHz Also like any other Arduino board, it has a power jack, an SPI header, a JTAG header, a reset button, and an erase button. With a 9V battery, the Arduino Due works great with the barrel plug connector. Some additions are made to this board to operate on the robot.

FreeRTOS

Since the Arduino IDE and environment are readily available within reach, Arduino libraries and drivers are available; however, the Arduino environment only supports `setup()` and `loop()`, as well as not supporting multitasking and other features. With FreeRTOS shown in figure 28, users can easily integrate FreeRTOS into the Arduino development environment through a library that can easily be shimmed into the IDE for seamless integration.



Figure 28: Real-time operating system for microcontrollers

It appears that most operating systems allow multiple threads or programs to run simultaneously. The process is known as multitasking. Typically, each processor core can only execute one program at a time. Operating systems utilize a component called the scheduler to determine which program to run when. The scheduler also makes it seem as if multiple programs are being executed simultaneously. RTOS schedulers are designed to provide predictable (often referred to as deterministic) execution patterns. Embedded systems, such as Arduino, often have real-time requirements, so this is particularly relevant for embedded systems. The scheduling mechanisms used in traditional real-time operating systems, such as FreeRTOS, enable determinism by allowing users to set priority levels for threads of execution. Schedulers then determine which thread to execute next by using the priority. It is the Task in FreeRTOS that executes the thread.

Arduino Pro:

Using the ATmega328 as its core, Arduino Pro is an embedded microcontroller board with a range of features. Both versions of the Pro are available at 3.3 V/8MHz and 5 V/16MHz. For this project, the 5V version is preferred because the ultrasonic sensors need to be energized. Among the features on the board are 14 digital inputs and outputs. There are six PWM outputs (four of which can be used as PWM inputs), six analog inputs, a battery connection, a power switch, and holes for mounting an ICSP header and pin headers as shown in figure 29. A breakout board can be powered and communicated via USB by connecting a six-pin header to an FTDI cable. With the Arduino Pro, you can have a semi-permanent electronic component built into exhibitions or into objects. Due to the board's lack of pre-attached connectors, it can be directly soldered or connected with various types of connectors, unlike headers pre-attached to the board. The pinout of this shield is compatible with the board, unlike many other Arduino shields.



Figure 29: Arduino Pro

With many of its features, this board is able to communicate with the outside world, such as with other Arduino boards, or with other microcontrollers. In addition to UART TTL serial communication, the ATmega328 also provides UART RX and TX digital pins, 0 and 1 pins, respectively. The Arduino software includes a serial monitor that allows basic textual data to be sent and received through the FTDI USB to TTL FT232RL cable.

NodeMCU (ESP8266)

Objects are increasingly being connected through IoT applications, and the use of connected objects is on the rise. Controlling the service robot is done through a customized app downloaded onto a tablet. Therefore, this tablet is supposed to communicate with the microcontroller to send and receive data. However, Bluetooth is a very limited option, as it has two main limitations that prevent this project from using it. In the first place, Wi-Fi offers a faster connection, which makes it more suitable for running full-scale networks. The second advantage of Wi-Fi is that it has a wider range and better security than Bluetooth (if configured properly). The used NodeMCU is shown in figure 30.



Figure 30: NodeMCU ESP8266 Wi-Fi SoC

However, as the robot will be controlled from different locations other than the tablet, it is possible to use the Wi-Fi to connect multiple devices at the same time in contrast to the Bluetooth situation. In conclusion, wireless connections were determined to be the most appropriate method of communication between the application and the microcontroller. Since the NodeMCU contains an Arduino-compatible programmable part, it is the best option for this purpose. The NodeMCU platform is an open-source platform that allows objects to be connected and data to be transferred over Wi-Fi

using the ESP8266. Furthermore, the microcontroller also features several important microcontroller features, such as PWM, ADC, and so on, that will be able to solve most of the project's problems on its own. Due to the fact that it was necessary to modify the Arduino IDE to make alternate toolchains available for compiling Arduino C/C++ for these more recent processors, Arduino.cc modified the Arduino IDE to support new MCU boards such as the ARM/SAM MCU included in the Arduino Due. With the introduction of the SAM Core and the Board Manager, they made this possible. As the name implies, the Arduino Core is made up of various software components that work together to compile C/C++ source files from the Board Manager and Arduino IDE.

The Wi-Fi SoC of the ESP8266 has been supported by a custom Arduino core developed by ESP8266 enthusiasts known as the "ESP8266 Core for the Arduino IDE". There are many ESP8266-based development boards and modules, including NodeMCUs, on which software can be developed.

3.4.1.1 CIRCUIT DIAGRAM AND CONNECTION

A robot's wiring is primarily intended to serve two purposes. A wiring's primary function is to provide power to the robot's devices. Second, communication networks are needed to link the many devices that comprise the robot's control system. These networks, in turn, help to keep the wiring system more secure and organized. They also make it easier to detect any loose wires or connections that may be incorrect. Additionally, by setting up the wires correctly and connecting them securely, this improves the robot's performance, eliminates intermittent electrical problems, and makes it easier to troubleshoot and resolve electrical or signal-related problems.

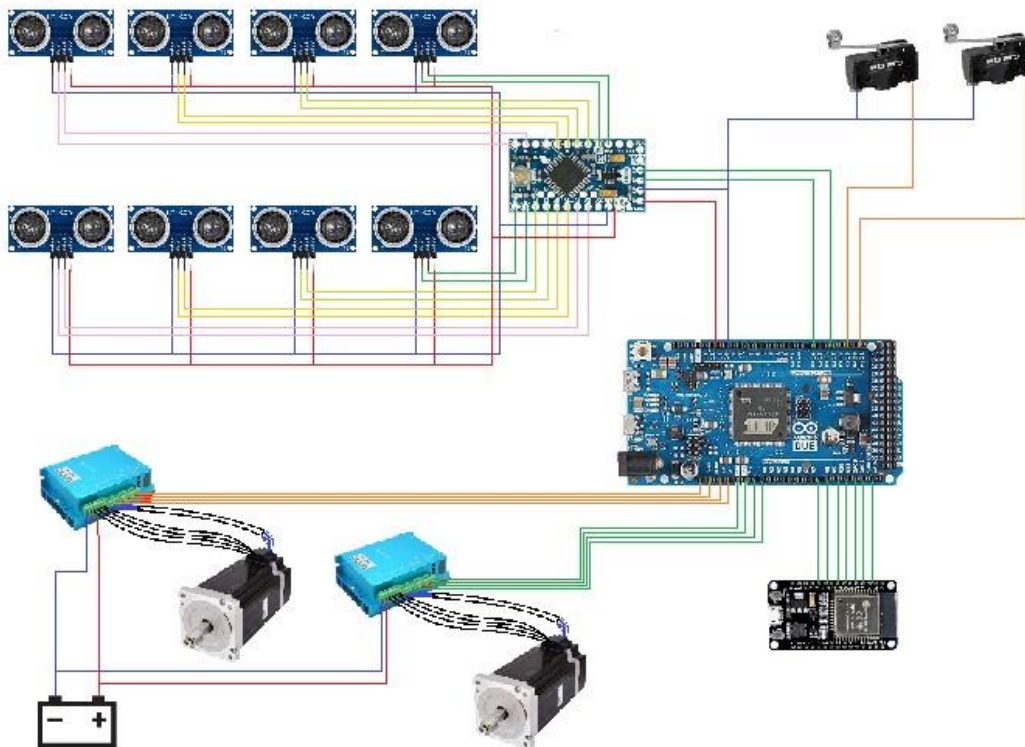


Figure 31: the complete electronics controlling circuit wiring diagram

Figure 31 illustrates the proposed robot's circuit diagram. There will be a central processing unit composed of 54 digital input/output pins and 12 analog input pins, which we call an Arduino Due. Developing the power supply has been designed to come through a charging circuit that will use two batteries of 12 VDC each, which will be connected in a series to provide the 24VDC for the motors. In order to step down the current in the circuit, DC to DC converters have been used with different settings and values. Microcontrollers interface with Wi-Fi modules using RX1 and TX1 sets of data pins on a Wi-Fi module. On the Arduino Pro microcontroller, the ultrasonic sensors have been connected to the digital pins echo and trig. A ZDM-2HA865 motor driver is composed of 22 pins; the first two pins are used to get the 24DCV from the power supply, while the second four pins supply the motors with the controlled power. Additionally, six input pins are connected to the motor's encoder, so the motors can be controlled by monitoring their current position. Further, four additional pins are connected to the microcontroller to control the operation of the whole movement.

Robotic movements are generally controlled by the microcontroller (Arduino Due) by following the user commands. The user will operate the robot by using an android app already included on the robot tablet. Additionally, the same application can be installed on some other mobile devices to control the robot as well. The commands can be retrieved using an app that converts them into physical actions as well as multiple voices using the tablet's speaker. Wireless NodeMCU module is used to connect the tablet to the microcontroller. As soon as the operator sends the table number command, the app will send the necessary data to the microcontroller, and then the microcontroller will receive the data through the NodeMCU. Depending on the command, the robot moves forward, backward, left, right, or autonomously to reach the desired table. In order to drive the robot, there are two heavy-load stepper motors that are mounted on ball bearings internally, and those motors are controlled by a stepper motor driver. When in autonomous mode, the system utilizes ultrasonic and Lidar sensors to detect obstacles and avoid them. A set of signals is sent by Arduino Pro depending on the data received from the Lidar sensors and the ultrasonic sensors. These signals give the robot information about any obstacles in its path. If the robot is not stopped at the moment, it detects any object, the avoidance action will begin. This happens because there are specific commands that stop the robot after any object is detected.

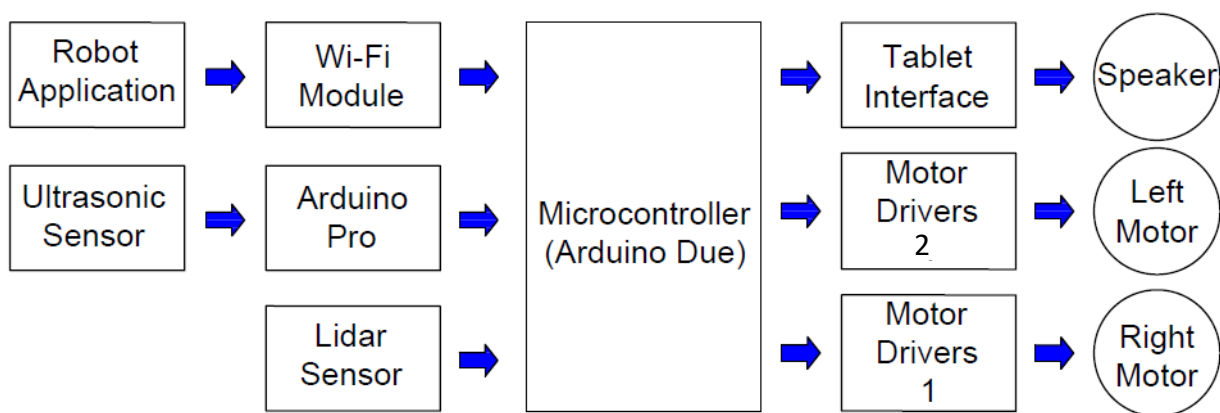


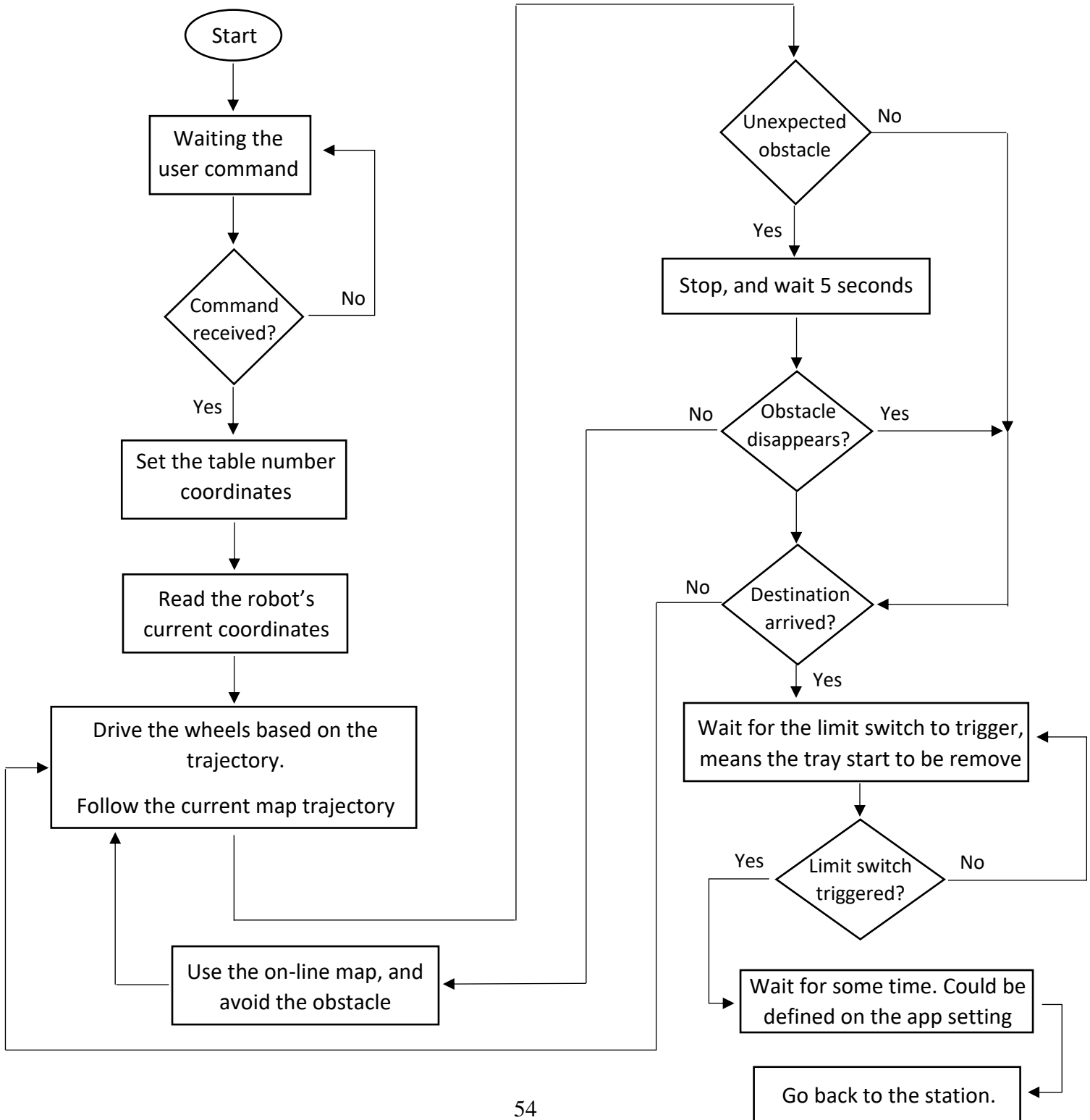
Figure 32: Functional block diagram of the proposed service robot model.

As shown in figure 32, the proposed robot has a functional design. Through a Wi-Fi module, the Arduino Due microcontroller connects to an Android application. The app interface will be used to give the movement commands and the data will be sent to the microcontroller using the same Wi-Fi module. Motor drivers were needed for controlling the movement of the robot. The microcontroller was used to control the rotation of motors on the left and right according to the required direction of rotation. An ultrasonic sensor will be integrated to detect obstacles and enable the autonomous operation of the robot.

3.4.1.2 CODING AND SOFTWARE

Design the code is requiring a complete understanding for the working process. The below process flow diagram is summarizing the process and the possible case.

Process Flow Diagram



Coding

Arduino Due

```
#include <Arduino.h>
#include <FreeRTOS_ARM.h>
```

Since the Arduino is used, it has its own system library like the pin mode and special commands link to it, so defining this library is must to allow the visual studio to understand the Arduino and talk with it. About the “FreeRTOS” library is a very common free version of real time operating systems used to initiate a real time mode on the code as we are using a Arduino Due which providing the structure of the real time system.

```
const uint16_t Coordinates_Table_X[9] = {2000,1000,3000,1000,3000,1000,3000,1000,3000};
const uint16_t Coordinates_Table_Y[9] = {0,1800,1800,4300,4300,12000,12000,16000,16000};
```

Defining the coordinate of the tables using arrays, this method of definitions making the code simpler and easier to track, edit or adjust. The first coordinates are for the starting station (2000,0), and the second is for the first table, and the third is for the second table, etc.

```
#define DEBUG 0
#define WHEEL_DIAMETER_CM 19
#define WHEEL_TO_WHEEL_DISTANCE 37.5
#define MICRO_STEPPING 8
#define PIN_Motor_Left_Pul 9
#define PIN_Motor_Left_Dir 10
#define PIN_Motor_Right_Pul 11
#define PIN_Motor_Right_Dir 12
#define PIN_Motor_Enable 13
#define PIN_Reach_Signal 21
#define PIN_Tray_Removed 20
#define PIN_Limit_1 17
#define PIN_Limit_2 16
#define X_Length_MAX 4000
#define Y_Length_MAX 16000
#define X_Home 2000
#define Y_Home 0
#define time_period 350
#define time_period_rotation 600
#define obstacle_wait_interval 7000
#define obstacle_avoid_width 800
#define obstacle_avoid_length 1100
#define obstacle_pulses_correction (((obstacle_avoid_length/10) * PULSE_PER_CM) * 2)
```

In the case of the "#define", the variable that is written in front of it is not a variable any longer because it is fixed and cannot be changed inside the code (Fixed value). These fixed variables include Arduino

pins in some cases, and distances and different values in others. The variables in general are stored in memory and consume memory space. Nevertheless, some variables are not required to be stored in memory while others are. Therefore, we can avoid saving them in memory by writing them into the code compiler so it will replace the value directly with the signed name. By using this method, any fixed constants can be defined efficiently. As an example, whenever the X_Home is called, it will be replaced by 2000, as the name; "X_Home" is only used for programmer information, whereas the code itself uses only the value; 2000.

```
// Motor Parameters
const float PULSE_PER_CM = ((200.0 * MICRO_STEPPING) / (3.1416 * (WHEEL_DIAMETER_CM)));
const float PULSE_PER_Degree = 1.07 * ((200.0 * MICRO_STEPPING) / 360.0) * (WHEEL_TO_WHEEL_DISTANCE / WHEEL_DIAMETER_CM);
```

In the normal variables section, six different major variables to control the robot have been defined. The motor parameter is used to find the amount of the required pulses to provide the motor so it can move precisely as per the coordination commands. Earlier to that, the wheel diameter and the distance between the motors must be found and defined in the Fix variables section above.

```
// Location Variables
int X_Position = X_Home;
int Y_Position = Y_Home;
int Theta = 0;
```

In the Location variable section, both coordinates variable X and Y have been defined to save the position every time during the robot movement, and to be use later once the robot take the next command.

```
// Table Selection
uint8_t prev_table = 0;
uint8_t table_no = 0;
```

This variable will save the requested table number and save it into the memory, this is very important to track the location of the robot and to ensure the robot to not move if the second command asked him to go to the same table where its location now is.

```
// Motor Drive Parameters
bool pulse = 0;
bool reached = 1;
bool go_home = 0;
volatile bool obstacle = 0;
```

Bool variable definition is used to work with the values of 0 or 1, four different variables initialized to control and to provide the feedback base on the motor situation

```
// Path ReRouting
bool avoidance = 0;
```

```
// ultrasonics Readings
byte ultrasonics;
```

In the ultrasonics section a special method has used to read from them, so this function is totally required to handle them and processing the data inside.

```
int limit_Switch1_status = 0;
int limit_Switch2_status = 0;
```

Since two limit switches are used, the needs for two special variables to save their value on it

```
// User Defined Functions
inline uint32_t trapezoidal_profile(int i, int n);
void Drive_Forward(uint16_t distance);
void Turn(uint16_t angle);
void Turn_Clockwise(uint16_t angle = 90);
void Turn_CounterClockwise(uint16_t angle = 90);
void trace_rectangle(uint16_t length, uint16_t width);
void Home(void);
inline void move_to_X(uint16_t X);
inline void move_to_Y(uint16_t Y);
void move_to_XY(uint16_t X, uint16_t Y);
void go_to_table(uint8_t table);
inline void avoid (void);
```

In this part of the code, the functions have been defined. The functions let the programmer to write a block of code that does just one specific thing and then use it whenever it is required rather than having to type it again and again. The above piece of code is showing many functions described below:

```
inline uint32_t trapezoidal_profile(int i, int n);
```

This function is providing the ability to accelerate and deaccelerate during the robot movement in the starting and the end of the paths.

```
void Drive_Forward(uint16_t distance);
```

Once this function called, both wheels will move to one direction together to allow the robot to move straight forward toward the target.

```

void Turn(uint16_t angle);

void Turn_Clockwise(uint16_t angle = 90);

void Turn_CounterClockwise(uint16_t angle = 90);

```

The main function is the “Turn” function. Once this function is required, a numerical calculations will be done to estimate the left distance to arrive the target. The shortest turning side will be decided based on that then directly it will call the related function and turn to that direction

```

void Home(void);

```

This function will be called once going to the home is required, either by receiving the command from the used to go back home or in the case once the food get delivered.

```

inline void move_to_X(uint16_t X);

inline void move_to_Y(uint16_t Y);

void move_to_XY(uint16_t X, uint16_t Y);

void go_to_table(uint8_t table);

```

Going to any table means going to a specific X and Y coordinators. During the robot movement, sometimes it moves only in the direction of X, sometime on the direction of Y, and some, in both directions together to complete the order and deliver the food.

```

// RTOS Tasks
void Read_Node_MCU(void *pvParameters);
void Read_Ultrasonics(void *pvParameters);

```

The real time operating system functions (RTOS) are running in the operating system not with the code, which mean that the functions are running every specific time does not matter what the robot is doing, as every one second as an example, the code will call the functions and check their reading doing an interrupting job. The piece of code above it is showing two real time functions, the first one is to keep read from the NodeMCU for any new command coming from the user from the tablet, while the second one is the reading from the ultrasonics to avoid any obstacles at any time during the job.

```

void setup() {

```

This section of code is used to setup the different controller’s pins, and to initialize some variables with initial values so the code starting by them.

```
Serial.begin(115200);  
Serial1.begin(9600);
```

Inside the `void setup()` section, the communication band should be clarifying and defined to allow the controller to connect with the computer (Visual Studio software) to send and receive the data.

```
// Motor Driver Pins  
pinMode(PIN_Motor_Left_Pul, OUTPUT);  
pinMode(PIN_Motor_Left_Dir, OUTPUT);  
pinMode(PIN_Motor_Right_Pul, OUTPUT);  
pinMode(PIN_Motor_Right_Dir, OUTPUT);  
pinMode(PIN_Motor_Enable, OUTPUT);
```

On the Arduino Due, many pins have been assigned as an OUTPUT's to open the channel between the controller and the motor driver to send the required control signals. The first section of the above brackets is the name of the pin, as they have already linked with the pin numbers on the beginning of the code, in the “#define” section.

```
// Limit Switches Pins  
pinMode(PIN_Reach_Signal, OUTPUT);  
pinMode(PIN_Tray_Removed, OUTPUT);  
pinMode(PIN_Limit_1, INPUT_PULLUP);  
pinMode(PIN_Limit_2, INPUT_PULLUP);
```

Two limit switches are used, and linking them with Arduino should be through a pull-up or pull-down resistor. A good feature in the Arduino that its pins have an internal PULL-UP resistor, and to activate any one of the, just required to right in the code “INPUT_PULLUP” instead of “INPUT”.

```
// NodeMCU Read Pins  
pinMode(A1, INPUT);  
pinMode(A2, INPUT);  
pinMode(A3, INPUT);
```

The signal that comes from the NodeMCU is coming in a three digits form A1,A2,A3, those will come to the Arduino as an input, so they have to be defined as INPUT's as shown in the above piece of code.

```

//initialization
digitalWrite(PIN_Motor_Enable, 0);
digitalWrite(PIN_Reach_Signal, 0);
digitalWrite(PIN_Tray_Removed, 0);
delay(10000);

```

Initialize the motor signal to stop mode, and the signals for limit switched to zero, to ensure that all readings are ready to be update during the running mode.

```

// RTOS Functions
xTaskCreate(Read_Ultrasonics, "Read Ultrasonics", 200, NULL, 1, NULL);
xTaskCreate(Read_Node_MCU, "Node", 200, NULL, 1, NULL);

vTaskStartScheduler();
for (;;)

```

Activate the real time function in the setup block, and initiate the values that functions receive and send to consider it once receive the data for both if the NodeMCU and the ultrasonic

```
void loop() {}
```

In this part, the code it written which utilize whatever described before, such as the functions, variables, and the parameters. All the coding lines after this is the program that the robot will follow and apply.

```

inline uint32_t trapezoidal_profile(uint32_t i, uint32_t n){
    uint32_t speed = 0;
    n = n >> 1;
    if (i < n){
        speed = n - i;
    } else {
        speed = i - n;
    }
    speed = (speed * 300) / n;
    return (time_period + speed);
}

```

This part is doing the needful to calculate how much the robot already cross from the signed command, this used to increase and decrease the speed gradually to ensure a smooth acceleration and deacceleration specially in the start, stop and turning.

```

void Drive_Forward(uint16_t distance) {
  if (DEBUG) {
    Serial.print("Driving Forward: ");
    Serial.println(distance);
  }
  uint32_t total_pulses = ((distance/10) * PULSE_PER_CM) * 2;
  digitalWrite(PIN_Motor_Left_Dir, 0);
  digitalWrite(PIN_Motor_Right_Dir, 1);
  for (uint32_t i = 0; i < total_pulses; i++) {
    int obs = 0;
    while(obstacle){
      if (obs > 3){
        if (obstacle_pulses_correction < (total_pulses - i)){
          avoid();
          digitalWrite(PIN_Motor_Left_Dir, 0);
          digitalWrite(PIN_Motor_Right_Dir, 1);
          total_pulses = total_pulses - obstacle_pulses_correction;
          break;
        }
      }
      delay(1000);
      obs++;
    }
    digitalWrite(PIN_Motor_Left_Pul, pulse);
    digitalWrite(PIN_Motor_Right_Pul, pulse);
    pulse = !pulse;
    delayMicroseconds(trapezoidal_profile(i, total_pulses));
  }
}

```

This code is utilizing many predefined functions, as it is worked to receive the distance that the robot should move forward as per the user needs (user command), then calculate the required pulses to achieve that target and give to the motor to it can drive forward. During that movement, this function is providing the obstacles avoidance feature, so the robot will avoid any obstacles come and stayed more than three seconds in front of the robot.

```

void Turn(uint16_t angle) {
    uint16_t angle_difference = abs(Theta - angle);
    if (DEBUG) {
        Serial.print("angle: ");
        Serial.println(angle_difference);
    }
    if (angle_difference == 270){
        if (Theta > angle) {
            Turn_Clockwise(90);
        } else {
            Turn_CounterClockwise(90);
        }
    } else{
        if (Theta > angle) {
            Turn_CounterClockwise(angle_difference);
        } else {
            Turn_Clockwise(angle_difference);
        }
    }
    Theta = angle;
}

```

Once the robot is requiring a turning, there are two options, either to turn to the left side (counterclockwise), or turn to the right side (clockwise). A separate function for both turning already defined below, and both of them are really utilize in the turn function and the selection of the right or left specified based on the nearest turning direction to the target. The comparison calculations and the decision are coming as a result of the above code.

```

void Turn_Clockwise(uint16_t angle) {
    uint32_t total_pulses = (angle * PULSE_PER_Degree) * 2;
    digitalWrite(PIN_Motor_Left_Dir, 1);
    digitalWrite(PIN_Motor_Right_Dir, 1);
    for (uint32_t i = 0; i < total_pulses; i++) {
        digitalWrite(PIN_Motor_Left_Pul, pulse);
        digitalWrite(PIN_Motor_Right_Pul, pulse);
        pulse = !pulse;
        delayMicroseconds(time_period_rotation);
    }
}

```



```

void Turn_CounterClockwise(uint16_t angle) {
  uint32_t total_pulses = (angle * PULSE_PER_Degree) * 2;
  digitalWrite(PIN_Motor_Left_Dir, 0);
  digitalWrite(PIN_Motor_Right_Dir, 0);
  for (uint32_t i = 0; i < total_pulses; i++) {
    digitalWrite(PIN_Motor_Left_Pul, pulse);
    digitalWrite(PIN_Motor_Right_Pul, pulse);
    pulse = !pulse;
    delayMicroseconds(time_period_rotation);
  }
}

```

Those are the two codes which already defined in the turning function, but they are showing how they are turning by giving the two wheels with different direction signals, so they rotate oppositely.

```

void trace_rectangle(void) {
  Drive_Forward(250);
  Turn_Clockwise();
  Drive_Forward(100);
  Turn_Clockwise();
  Drive_Forward(250);
  Turn_Clockwise();
  Drive_Forward(100);
  Turn_Clockwise();
}

```

Short piece of code has designed to calibrate the turning angle after setting it from the early stages code, this code will command the robot to move in a rectangular shape so the turning angle and the speed could be easily track and monitor.

```

void Home(void) {
  if (!(X_Position == X_Home && Y_Position == Y_Home)) {
    move_to_XY(X_Home, Y_Home);
  }
  reached = 1;
}

```

This function is working when the user give a command to the robot to go home, so the code will check its location if that at home or not, if not, the robot will take the way direct to home, if not, it will stay on the same position.

```

void move_to_X(uint16_t X){
    uint16_t distance = abs(X - X_Position);
    if (X_Position < X) {
        Turn(90);
    } else if (X_Position > X) {
        Turn(270);
    }
    Drive_Forward(distance);
    X_Position = X;
}

```

```

void move_to_Y(uint16_t Y){
    uint16_t distance = abs(Y - Y_Position);
    if (Y_Position < Y) {
        Turn(0);
    } else if (Y_Position > Y) {
        Turn(180);
    }
    Drive_Forward(distance);
    Y_Position = Y;
}

```

```

void move_to_XY(uint16_t X, uint16_t Y) {
    move_to_Y(Y);
    move_to_X(X);
}

```

Three embedded functions with each other's, move to x and y together is requiring a separate command to move x, and another command to move y, then compiling them in one function to either more of them together or individual. Before any movement, the code must check the current position, and you cannot tell the robot to go to a specific location where it is already right now.

```

void go_to_table(uint8_t table) {
  if (DEBUG) {
    Serial.print("Going to table: ");
    Serial.println(table);
  }

  move_to_XY(Coordinates_Table_X[table], Coordinates_Table_Y[table]);
  digitalWrite(PIN_Reach_Signal, 1);
  delay(1000);
  digitalWrite(PIN_Reach_Signal, 0);
  delay(500);
  //Serial.println("Reached");
  reached = 1;
  limit_Switch1_status = 0;
  limit_Switch2_status = 0;
  while (true) {
    limit_Switch1_status = digitalRead(PIN_Limit_1);
    limit_Switch2_status = digitalRead(PIN_Limit_2);
    if(limit_Switch1_status>0){
      break;
    }
    if(limit_Switch2_status>0){
      break;
    }
  }

  digitalWrite(PIN_Tray_Removed, 1);
  delay(1000);
  digitalWrite(PIN_Tray_Removed, 0);
  delay(500);
  //Serial.println("tray removed");
  if(DEBUG) {Serial.print("Done ");}
}

```

This code will receive the command for the targeted table from the NodeMCU, and convert to a signal that send to the motors to move in a specified X and Y coordinates, so it arrive the target. Also, this code is telling the robot to stay beside the table once it arrives until the customer remove the table and the limit switch release. Then it will send a signal to the Tablet to say “Thank You” or any other works used define on the setting button on the tablet. Finally, once the food gets delivered, the robot directly will go back to home position from where it already started.

```

inline void avoid(void){
    Turn(90);
    Drive_Forward(obstacle_avoid_width);
    Turn(0);
    Drive_Forward(obstacle_avoid_length);
    Turn(270);
    Drive_Forward(obstacle_avoid_width);
    Turn(0);
}

```

This part is the key function in the obstacle's avoidance, as the robot will avoid the obstacles in a specified width and length to come back after that to the track and continue the trip. This function will not be called if the remained distance to the target is not enough to do the avoidance, otherwise, the robot will get confused and loose the way.

```

void Read_Node_MCU(void *pvParameters) {
    TickType_t xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();
    for (;;) {
        prev_table = table_no;
        table_no = digitalRead(A1);
        table_no = table_no << 1 | digitalRead(A2);
        table_no = table_no << 1 | digitalRead(A3);
        if(DEBUG) {Serial.println(table_no);}
        if (prev_table != table_no) {
            if(reached){
                reached = 0;
                if(table_no == 0){
                    Home();
                }else {
                    go_to_table(table_no);
                }
            }
        }
        vTaskDelayUntil(&xLastWakeTime, (500 / portTICK_PERIOD_MS));
    }
}

```

This piece of code it latterly the gate for the entire code to the external world, as it links the whole code with the Wi-Fi through the NodeMCU for the purpose of connecting it to the tablet and the control application. Any command comes from the supplier through the application will be handled by this code for treatment and doing the required actions related to it on the microcontroller.

```

void Read_Ultrasonics(void *pvParameters) {
    TickType_t xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();
    for (;;) {
        if (Serial1.available()){
            ultrasonics = Serial1.parseInt();
            ultrasonics = ultrasonics & B00110000;
            obstacle = ultrasonics > 0 ? 1: 0;
            //obstacle = 0;
            //if(1) {Serial.println(obstacle);}
        }
        vTaskDelayUntil(&xLastWakeTime, (100 / portTICK_PERIOD_MS));
    }
}

```

Eight ultrasonics are connected to one controller and collect the data about the surrounding if any obstacles are around either static or dynamic. This code is able to handle and treat the data, and based in it the action will be done by conducting an avoidance or just keep the robot stopped until the obstacles removed if the remains distance is not enough for the avoidance.

Arduino Pro

```
#include <Arduino.h>
```

Since the Arduino is used (Arduino Pro), it has its own system library like the pin mode and special commands link to it, so defining this library is must to allow the visual studio to understand the Arduino and talk with it. Without this library, most of the definition lines below will be meaningless.

```

#define PIN_Ultrasonic_RB_Trig 2
#define PIN_Ultrasonic_RB_Echo 3
#define PIN_Ultrasonic_RF_Trig 4
#define PIN_Ultrasonic_RF_Echo 5
#define PIN_Ultrasonic_FR_Trig 6
#define PIN_Ultrasonic_FR_Echo 7
#define PIN_Ultrasonic_FL_Trig 8
#define PIN_Ultrasonic_FL_Echo 9
#define PIN_Ultrasonic_LF_Trig 10
#define PIN_Ultrasonic_LF_Echo 11
#define PIN_Ultrasonic_LB_Trig 12
#define PIN_Ultrasonic_LB_Echo 13
#define PIN_Ultrasonic_BL_Trig A0
#define PIN_Ultrasonic_BL_Echo A1
#define PIN_Ultrasonic_BR_Trig A2
#define PIN_Ultrasonic_BR_Echo A3

#define thres 40

```

In the case of the "#define", the variable that is written in front of it is not a variable any longer because it is fixed and cannot be changed inside the code (Fixed value). These fixed variables are the Arduino pins where the ultrasonics are connected and assigning a names for them as shown in figure 33. Each ultrasonic has two data pins Trig and Echo, so they have assigned with the direction names as per the below diagram.

The last code line "#define thres 40" is used to define the distance where the ultrasonic start detects if any obstacles is there. This could be change if another distance is required.

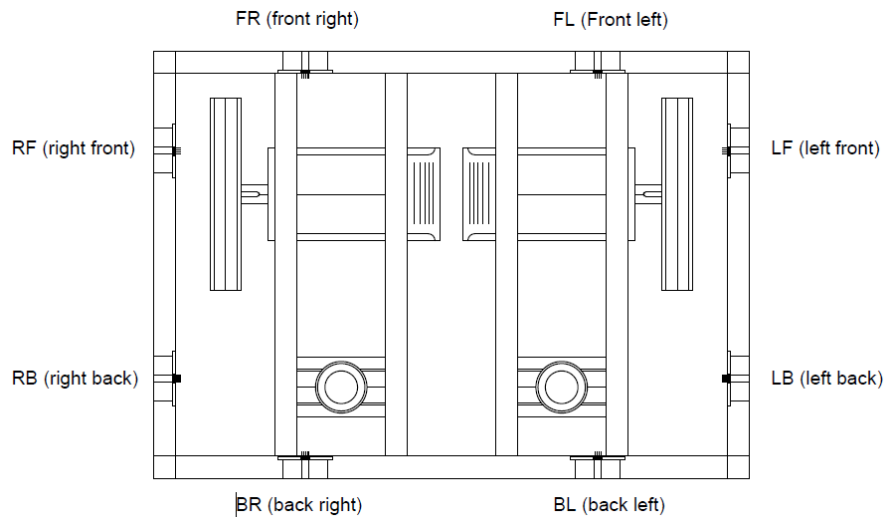


Figure 33: top view diagram is showing 8 ultrasonics sensors around the robot

```
bool U[8] = {0};
byte ultrasonics=0;

bool read_ultrasonic(int, int);
```

The reading of the eight ultrasonics required an array to put them together to send them one time to the Main controller (Arduino Due), this coding block is used to initiate that array.

```
void setup() {
```

This section of code is used to setup the different controller's pins, and to initialize some variables with initial values so the code starting by them.

```
Serial.begin(9600);
```

Inside the void setup() section, the communication band should be clarifying and defined to allow the controller to connect with the computer (Visual Studio software) to send and receive the data.

```

pinMode(PIN_Ultrasonic_RB_Trigger, OUTPUT);
pinMode(PIN_Ultrasonic_RB_Echo, INPUT);

pinMode(PIN_Ultrasonic_RF_Trigger, OUTPUT);
pinMode(PIN_Ultrasonic_RF_Echo, INPUT);

pinMode(PIN_Ultrasonic_FR_Trigger, OUTPUT);
pinMode(PIN_Ultrasonic_FR_Echo, INPUT);

pinMode(PIN_Ultrasonic_FL_Trigger, OUTPUT);
pinMode(PIN_Ultrasonic_FL_Echo, INPUT);

pinMode(PIN_Ultrasonic_LF_Trigger, OUTPUT);
pinMode(PIN_Ultrasonic_LF_Echo, INPUT);

pinMode(PIN_Ultrasonic_LB_Trigger, OUTPUT);
pinMode(PIN_Ultrasonic_LB_Echo, INPUT);

pinMode(PIN_Ultrasonic_BL_Trigger, OUTPUT);
pinMode(PIN_Ultrasonic_BL_Echo, INPUT);

pinMode(PIN_Ultrasonic_BR_Trigger, OUTPUT);
pinMode(PIN_Ultrasonic_BR_Echo, INPUT);

```

On the Arduino Pro, the ultrasonics are sending and receiving a data from the two pins it has, the Trig pin is used to receive data, so the Arduino pro will send to it, then it considered as OUTPUT for the Arduino. While the Echo is sending data from the ultrasonic to the Arduino Pro, so it considered as an INPUT to the Arduino. The above code assigned 16 pins from the Arduino (8 for the inputs, and 8 for the outputs).

```

digitalWrite(PIN_Ultrasonic_RB_Trigger, 0);
digitalWrite(PIN_Ultrasonic_RF_Trigger, 0);
digitalWrite(PIN_Ultrasonic_FR_Trigger, 0);
digitalWrite(PIN_Ultrasonic_FL_Trigger, 0);
digitalWrite(PIN_Ultrasonic_LF_Trigger, 0);
digitalWrite(PIN_Ultrasonic_LB_Trigger, 0);
digitalWrite(PIN_Ultrasonic_BL_Trigger, 0);
digitalWrite(PIN_Ultrasonic_BR_Trigger, 0);

```

The pins that supposed to send a data to the ultrasonic must be Initialize in the beginning, to ensure that all readings are ready to be update during the running mode.

```

bool read_ultrasonic(int trig, int echo){
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  int duration = pulseIn(echo, HIGH, 50000);
  duration = duration*0.034/2; //here we get distance
  //Serial.println(duration);
  if (duration == 0){
    return 0;
  }
  return (duration < thres);
}

```

It read from the ultrasonics and measure the distance away from that ultrasonic and check that if there is an obstacles by sending '1', or if there are no obstacles by sending '0'

```

void loop() {
  U[0] = read_ultrasonic(PIN_Ultrasonic_RB_Trig, PIN_Ultrasonic_RB_Echo);
  U[1] = read_ultrasonic(PIN_Ultrasonic_RF_Trig, PIN_Ultrasonic_RF_Echo);
  U[2] = read_ultrasonic(PIN_Ultrasonic_FR_Trig, PIN_Ultrasonic_FR_Echo);
  U[3] = read_ultrasonic(PIN_Ultrasonic_FL_Trig, PIN_Ultrasonic_FL_Echo);
  U[4] = read_ultrasonic(PIN_Ultrasonic_LF_Trig, PIN_Ultrasonic_LF_Echo);
  U[5] = read_ultrasonic(PIN_Ultrasonic_LB_Trig, PIN_Ultrasonic_LB_Echo);
  U[6] = read_ultrasonic(PIN_Ultrasonic_BL_Trig, PIN_Ultrasonic_BL_Echo);
  U[7] = read_ultrasonic(PIN_Ultrasonic_BR_Trig, PIN_Ultrasonic_BR_Echo);
  ultrasonics = U[0]<<7 | U[1]<<6 | U[2]<<5 | U[3]<<4 | U[4]<<3 | U[5]<<2 | U[6]<<1 | U[7] ;
  Serial.println(ultrasonics);
  delay(100);
}

```

On the first element of the array, it gets the first reading from the ultrasonic. Similarly, for all the other array elements. All the readings are compiles and saved in one variable called "ultrasonics" and sending them as one variable to the Arduino Due instead of sending them 8 separate times

As what the code shown, the **ultrasonics** variable will have the reading from the ultrasonics organized as the following :

```

U[0] U[1] U[2] U[3] U[4] U[5] U[6] U[7]
  RB  RF  FR  FL  LF  LB  BL  BR

```

NodeMCU

```

#include <Arduino.h>
#include <ESP8266WiFi.h>

```


Since the Arduino is used, it has its own system library like the pin mode and special commands link to it, so defining this library is must to allow the visual studio to understand the Arduino and talk with it. The ESP8266 library is mainly focused on Wi-Fi. In order to begin sending and receiving data, the NodeMCU board enables the user to connect the ESP8266 module to a Wi-Fi network. Using this library, the user will be able to understand and program this board's specific Wi-Fi networking capabilities

```
#define debug 1
#define led 2
#define PIN1 D5
#define PIN2 D6
#define PIN3 D7
#define PIN4 D8
#define PIN_Reached_Signal D0
#define PIN_Limit_Switch_Signal D1
```

In the case of the "#define", the variable that is written in front of it is not a variable any longer because it is fixed and cannot be changed inside the code (Fixed value). These fixed variables are the NodeMCU pins where the Arduino Due is connected to transfer the data from/to the Wi-Fi network. Two extra pins are added to show the status of the limit switches to reflect the status of the tray if removed or not yet.

```
const char *ssid = "AndroidAP";
const char *password = "said1234";
```

When the NodeMCU starts, it searches for the name 'AndroidAP' on the robot surrounding, and once it finds it, directly it links it with the password 'said1234' automatically. The library 'ESP8266WiFi' is handling all of this communication in the background.

```
int port = 8888;
```

The port is a communication line between the application and the NodeMCU to send and receive data. The same port number should be defined on the application to allow a two-way data transferring. In other words, the Wi-Fi name is the same as the name of a city, while the port name is the name of the street, then the IP address same as the address of house. Additionally, the port should be only for this specific work, as if another one already using it, it will not be able to link by showing an error, while it will work perfectly if it is only booked for transferring the data between the tablet application and the NodeMCU.

```
WiFiServer server(port);
```

This line of code to initialize the Wi-Fi server to the port viewed above ;8888

```
void setup(void) {  
  Serial.begin(9600);  
  WiFi.mode(WIFI_STA);  
  WiFi.begin(ssid, password);  
  
  pinMode(led, OUTPUT);  
  pinMode(PIN1,OUTPUT);  
  pinMode(PIN2,OUTPUT);  
  pinMode(PIN3,OUTPUT);  
  pinMode(PIN_Reached_Signal, INPUT);  
  pinMode(PIN_Limit_Switch_Signal, INPUT);  
  digitalWrite(PIN1,0);  
  digitalWrite(PIN2,0);  
  digitalWrite(PIN3,0);  
  digitalWrite(PIN4,0);  
  digitalWrite(led,0);  
  
  while (WiFi.status() != WL_CONNECTED) {  
    | delay(500);  
  }  
  
  digitalWrite(led,1);  
  Serial.println(WiFi.localIP());  
  server.begin();  
}
```

```
void setup(void) {
```

This section of code is used to setup the different controller's pins, and to initialize some variables with initial values so the code starting by them.

```
Serial.begin(9600);
```

Inside the `void setup()` section, the communication band should be clarifying and defined to allow the controller to connect with the computer (Visual Studio software) to send and receive the data.

```
WiFi.mode(WIFI_STA);  
WiFi.begin(ssid, password);
```

This defines the Wi-Fi mode of the NodeMCU as a Wi-Fi station, which means that the NodeMCU will connect to an external Wi-Fi which is the access point (mobile hotspot). The access point is the device that broadcasting the name, and the Wi-Fi station is the device that connect to the access point

```
pinMode(led, OUTPUT);
pinMode(PIN1,OUTPUT);
pinMode(PIN2,OUTPUT);
pinMode(PIN3,OUTPUT);
pinMode(PIN_Reached_Signal, INPUT);
pinMode(PIN_Limit_Switch_Signal, INPUT);
```

On the NodeMCU, the pins are defined to be as an outputs except the limit switches signals, as they have to be defined as an Input to take the data from the Arduino

```
digitalWrite(PIN1,0);
digitalWrite(PIN2,0);
digitalWrite(PIN3,0);
digitalWrite(PIN4,0);
digitalWrite(led,0);
```

The above lines of code are used to Initialize the pins of the NodeMCU in the beginning, to ensure that all readings are ready to be update during the running mode.

```
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
}
```

The code will wait sometime to ensure that the Wi-Fi already connected before start receiving and sending any information in the running mode

```
digitalWrite(led,1);
Serial.println(WiFi.localIP());
server.begin();
```

Once the NodeMCU get connected with the Wi-Fi router, the light turns OFF as per the code above, then it shows the IP on the monitor. The code '`server.begin();`' is telling the ModeMCU to work and act as a server. For the server it waits for some requests, so once it get the request, it perform the operation for that request instantly.

```

void loop(void) {
  WiFiClient client = server.available();
  if(client) {
    while(client.connected()) {
      while(client.available()>0) {
        uint8_t rec_data = client.read();
        rec_data = rec_data - 48;
      }
    }
  }
}

```

This check if the NodeMCU get any request or no from the application (from the user). If the requests are available, the variable 'client' will turn to 1. Once the client variable is one, the code will check again if the NodeMCU is connected, if yes, the code will make a short subtraction to convert the ASCII to integer by subtraction a 48 from the received data.

```

if (rec_data == 0) {
  | handleHome();
} else if(rec_data == 1){
  | handleTable1();
} else if (rec_data == 2) {
  | handleTable2();
} else if (rec_data == 3) {
  | handleTable3();
} else if (rec_data == 4) {
  | handleTable4();
} else if (rec_data == 5) {
  | handleTable5();
} else if (rec_data == 6) {
  | handleTable6();
}
}

```

This portion will treat the data and make the required comparison to know which request have been requested. If 0, then activate the 'handlehome0();' function, if 1, then activate the 'handleTable1();' function. Similarly, up to the table number 7.

```

    reached_signal = digitalRead(PIN_Reached_Signal);
    limit_switch_signal = digitalRead(PIN_Limit_Switch_Signal);
    if(reached_signal) {
        client.write("r");
        delay(1000);
    }
    if (limit_switch_signal) {
        client.write("l");
        delay(1000);
    }
}
client.stop();
}
}

```

This part of the code is for the other way of communication, as this code is sending the information from Arduino to NodeMCU to tablet. If the robot reached, the NodeMCU will send “r” to the tablet, then once the tray removed, the NodeMCU will send “l” to the table. Those two letters will be received from the tablet and the tablet will do some specific action based on that. The tablet actions and details will be explained later in details in CH4

```

void handleHome(void){
    if(debug) {Serial.println("home");}
    digitalWrite(PIN1,0);
    digitalWrite(PIN2,0);
    digitalWrite(PIN3,0);
}

```

```

void handleTable1() {
    if(debug) {Serial.println("1");}
    digitalWrite(PIN1,0);
    digitalWrite(PIN2,0);
    digitalWrite(PIN3,1);
}

```

```

void handleTable2() {
    if(debug) {Serial.println("2");}
    digitalWrite(PIN1,0);
    digitalWrite(PIN2,1);
    digitalWrite(PIN3,0);
}

```

```

void handleTable3() {
    if(debug) {Serial.println("3");}
    digitalWrite(PIN1,0);
    digitalWrite(PIN2,1);
    digitalWrite(PIN3,1);
}

```

```

void handleTable4() {
  if(debug) {Serial.println("4");}
  digitalWrite(PIN1,1);
  digitalWrite(PIN2,0);
  digitalWrite(PIN3,0);
}
void handleTable5() {
  if(debug) {Serial.println("5");}
  digitalWrite(PIN1,1);
  digitalWrite(PIN2,0);
  digitalWrite(PIN3,1);
}
void handleTable6() {
  if(debug) {Serial.println("6");}
  digitalWrite(PIN1,0);
  digitalWrite(PIN2,0);
  digitalWrite(PIN3,0);
}

```

Once the command asked for a specific table, a 3 binary number will be generated as one set, then it will be sent to the Arduino to treat it and convert it to a table number then start the action based on that. The table below is showing the details:

Table 6: Tables options selected using 3 digits binary number

Position	Binary number
Home	000
Table 1	001
Table 2	010
Table 3	011
Table 4	100
Table 5	101
Table 6	111

Table 7: Main differences between the ROS systems and the new real time approach.

	The new approach	Robot Operating System (ROS)
Cost and expenses	<3,000 \$ this study excludes any expensive high processor computer, as everything runs on three small microcontrollers (Arduino Due, Arduino Pro, and NodeMCU)	+7,000 \$ ROS is using a very expensive sensors and actuator modules, some through high-latency USB, and processing the data always too late in a complex message passing architecture inside an expensive computer
Lidar reading accuracy	A customized code created after getting LIDAR (rotating laser scanner) data out for mapping took advantage of almost no-latency wheel odometry and gyroscope data while the robot was moving during the acquisition	ROS simply just ignores this issue entirely. Although this problem is straightforward to solve on the integrated low level, it is complex to extend to the integrated high level like ROS
Industry deployment	Can be simply deploy in the industry by enhancing the ability to overlook some unexpected industry object. Also, it has the ability to automatically fix errors	commercial industrial robots do not use the Robotic Operative System (ROS) since they cannot automatically correct errors during their tasks, which is a limitation of ROS
Systematic problem solving	This is having the ability to handle the rest 90%	ROS would actually solve around 10% of the robotics problems
Complexity	Straightforward control system; simple, and easy to access and modify. Supported by multiple open-source platforms	Complex, separate modules (not designed to be compatible with each other) together is an inefficient and weak use of development time; even when it's done using a middleware such as ROS, even if it's the legendary "someone else" doing the drivers and data conversion
Control power consumption	Much less power using 200mW of power to do the job.	5 - 6 times more than our research system

Software (Robot Application design)

App Inventor is an integrated development environment that allows you to develop web applications. Google initially developed it, and now it is handled and managed by the MIT university. Developers with little or no experience in computer programming can now produce software applications for the apple operating system (iOS) and the Android. By using Star Logo, users can drag and drop visual objects in a graphic user interface to create custom Android apps or use scratch logic (a programming language) to build a more complex application. As of today, App-Inventor Companion is still under development, the program that allows you to run and debug your app on iOS devices. App

Inventor was created after extensive research was conducted in the field of the computations in the educational field, along with the working on the development environment which is belong to Google company.

In addition to App Inventor, all the projects are informed by constructionist learning theories that emphasize the importance of active learning through programming. App Inventor's experimental Firebase Realtime Database component also supports cloud data storage.

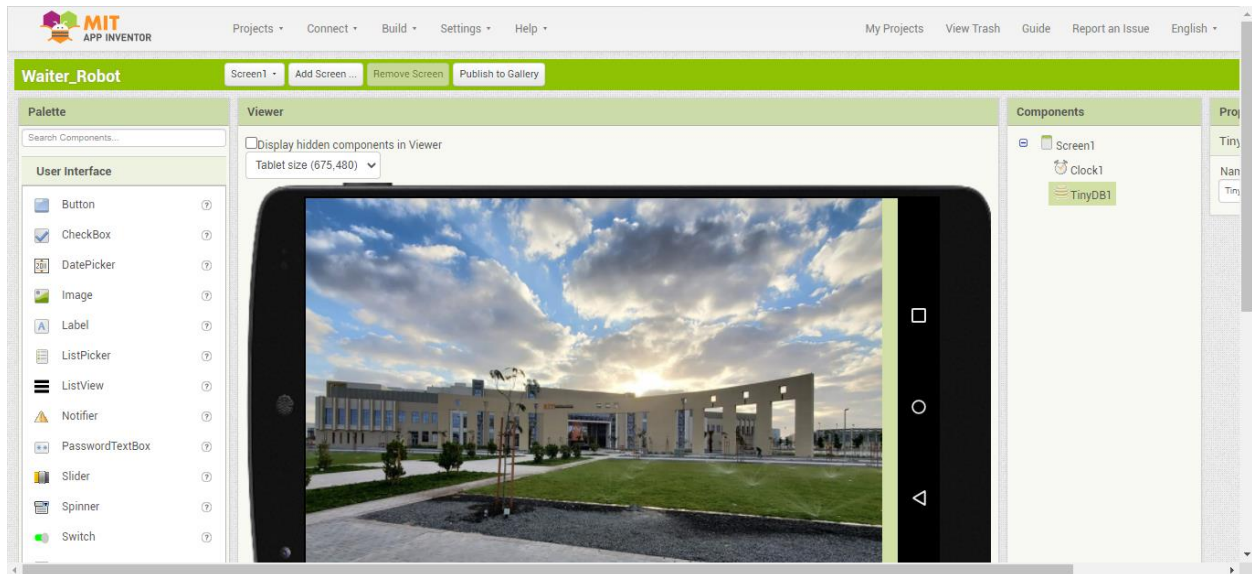


Figure 34: The main interface for the MIT app inventor

The MIT app tool inventor shown in figure 34, is allowing the people from all ages and all levels experience to learn and develop from the basic level of application up to a high one. It has the features of drag and drop programming interface that allows anybody to create mobile application for android devices. The robot control application interface has been developed using the MIT app inventor platform. The MIT app inventor has two main pages or tap; Designer and the blocks. On the other hand, for this specific project, three different screens illustrated in figure 35 have been initiated; the first one is the initial welcoming screen that shows a picture for the RIT new campus in the beginning, the second screen is the main one called the start screen to control the robot and to give the instructions and command, while the third one is used for the app setting.



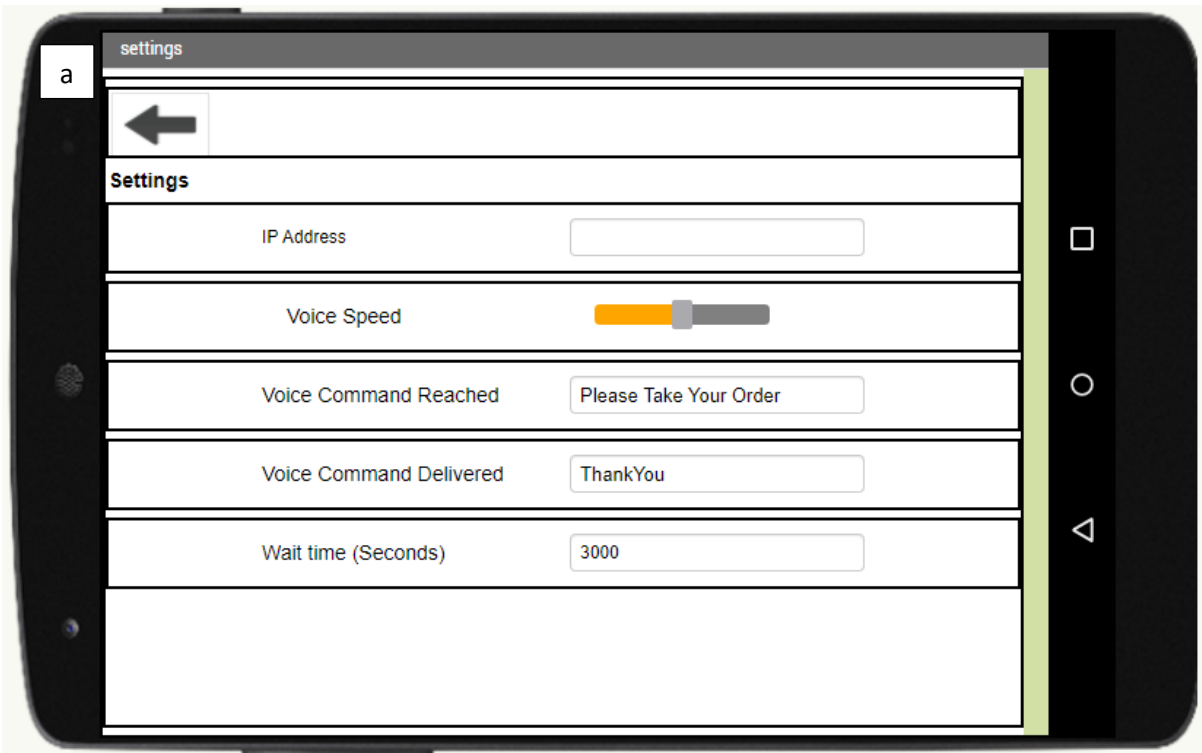


Figure 35: Robot application three main screens (a) Setting screen, (b) Welcoming screen, (c) Start screen

Each screen has a components or elements which defined the function of the screen and what it is doing, there is a component called 'TinyDB1' shown in figure 36, this component is represent the database of the screen, as each screen much has one to memorize the whole component on it.



Figure 36: The common database non-visible component

First screen (Welcoming screen)

The first screen is used to show a quick photo before initializing the control screen, as a timer should be set to identify how much time it should take before it moves to the second one. Figure 37 is showing this feature.

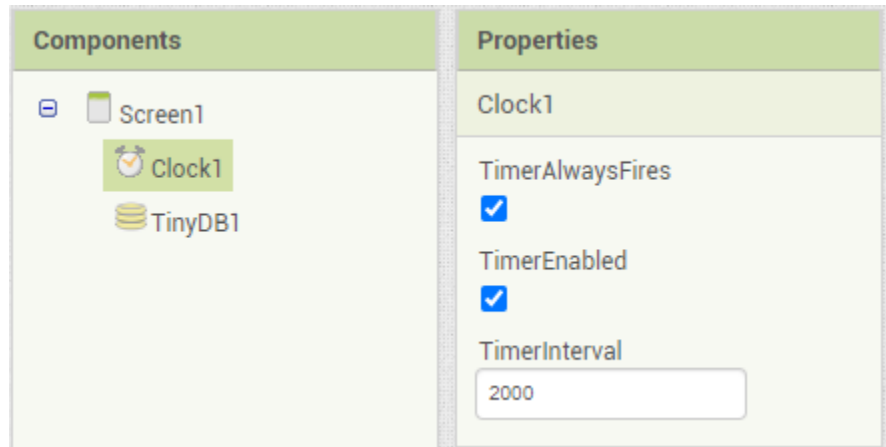


Figure 37: First screen time setting

Behind whatever we are seeing in the first screen, there is a Blockly code that control the entire screen. What to do, from where to bring, etc. Figure 28 is showing the first Blockly programming step.

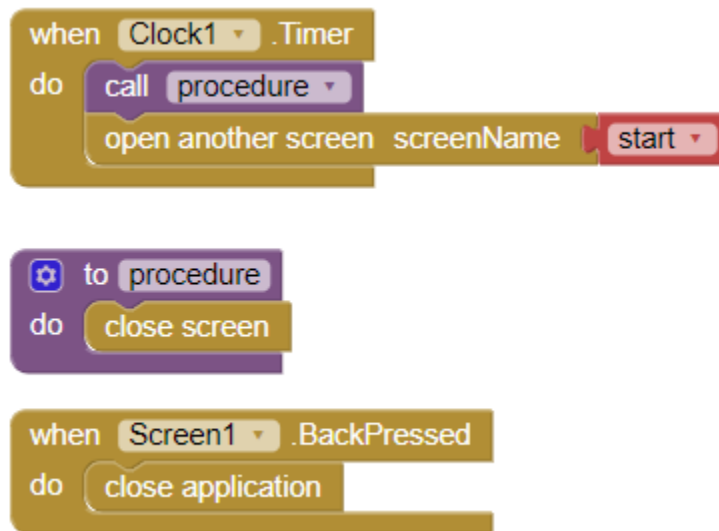


Figure 38: First screen Blockly code

The Blockly code that shown in fig.7 will be called after the 2000ms of showing the main screen as defined in the component timer. So, after two seconds the first block will run and call the “procedure” function to close the screen, then another screen name “Start” will open which represent the main control screen.

Second screen (Start screen)

This screen is considered as the main control screen, the user interacts with the robot through this screen by giving the command or the required table, figure 40 is showing the same application screen. One the

food is loaded on one of the trays, the operator of the robot needs to press on the table number, then start. To allow this features work in a proper way, a huge efforts required to design and link this screen of the application.

Three basic component shown in figure 39 have been used, TinyDB1, TextToSpeech1, and the imported additional tool ClientSocketAI2Ext1.this was the most complicated imported item to allow the two-way data transferring between the app and the NodeMCU.

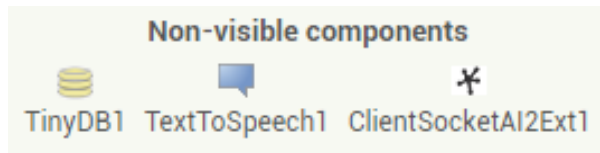


Figure 39: The start screen database non-visible component

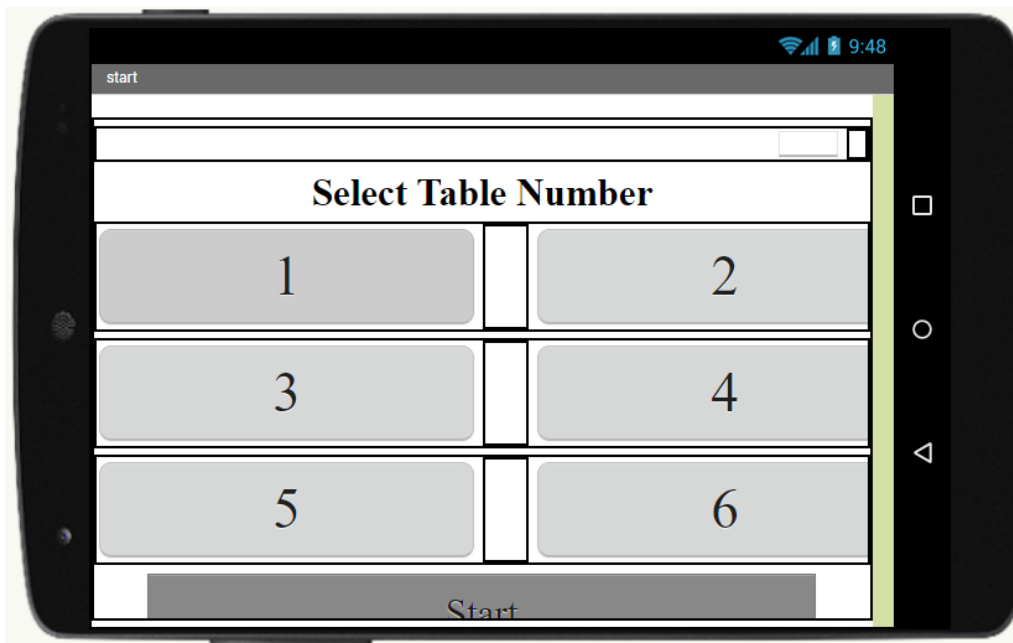


Figure 40: The application start screen

This screen basically consists of two components, a buttons, and a text box as shown in figure 41, where the user is able to write things.

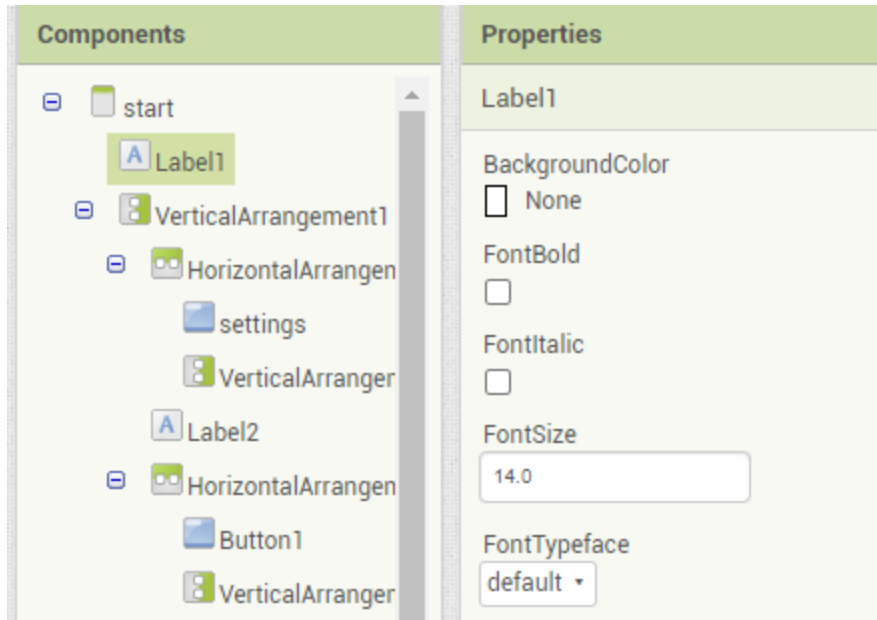


Figure 41: Part of the main screen components

The code behind this screen is a bit long and complicated, it will be divided into a portions for ease description and discussion. Figure 42 is showing the second part of the program.

```

when start - .Initialize
do
  set ClientSocketAI2Ext1 - . ServerAddress - to get global Base_IP -
  set ClientSocketAI2Ext1 - . ServerPort - to 8888
  call ClientSocketAI2Ext1 - .Connect

initialize global i to 0
initialize global Base_IP to "192.168.52.143"
initialize global Table_no to 0
initialize global Prev_Table_no to 0

when Button1 - .Click
do
  set global Table_no - to 1

when Button2 - .Click
do
  set global Table_no - to 2

when Button3 - .Click
do
  set global Table_no - to 3

when Button4 - .Click
do
  set global Table_no - to 4

when Button5 - .Click
do
  set global Table_no - to 5

when Button6 - .Click
do
  set global Table_no - to 6
  
```

Figure 42: Initiate the link between the NodeMCU and the application, and the tables buttons.

In the first screen setting, it was calling the “start” screen to open after it finished. So, in this screen the start screen code initialized. This first line is used to set the IP address as a variable which could be

change by the app developer only, then it will set the port to the same port that defined in the NodeMCU C++ code, as both of the application and the code must show the same port number to communicate. Finally, the connect line is called to initiate the line of contact between the NodeMCU and the application. The "table number" and the "previous table number" variables have been initialized and set to zero, so once the table number 1 is pressed as an example, the code will set "table number" to 1, and if the table number 2 is pressed as an example, the code will set "table number" to 2, etc. This shown in figure 43.

```

when st .Click
do
  set Label1 . Text to get global Table_no
  set Label1 . FontSize to 500
  set Label1 . HeightPercent to 100
  call ClientSocketAI2Ext1 .SendData
  data get global Table_no
  
```

Figure 43: The actions once the Start button pressed

One the table number is pressed, the robot will not start taking the actions until the used press the "Start" button, once this button pressed, the number of this table will be shown in the entire screen during the trip until it arrives the destination. This Blockly code above is showing the magnification of the table number text to a 500 x 100 times. Finally, the data will be transfer to the Arduino thought the NodeMCU to apply the changings by calling the "SendData" block as shown in figure 44.

```

when settings .Click
do
  call close_screen
  open another screen screenName settings

to close_screen
do
  close screen

when start .BackPressed
do
  close application
  
```

Figure 44: Placing the setting button actions

Once the setting button pressed, the Blockly code that shown in fig.11 will be called. So, after it will call the function of "close screen" to close the control screen, then another screen name "Setting" will appear which represent the setting screen where the user can add and change some settings. All above was a coding using one way direction from the application to NodeMCU, While the below code is doing the opposite by getting the data from the NodeMCU to tablet to say something like: 'Please take your order' or 'Thank you' or whatever is added to the setting button.

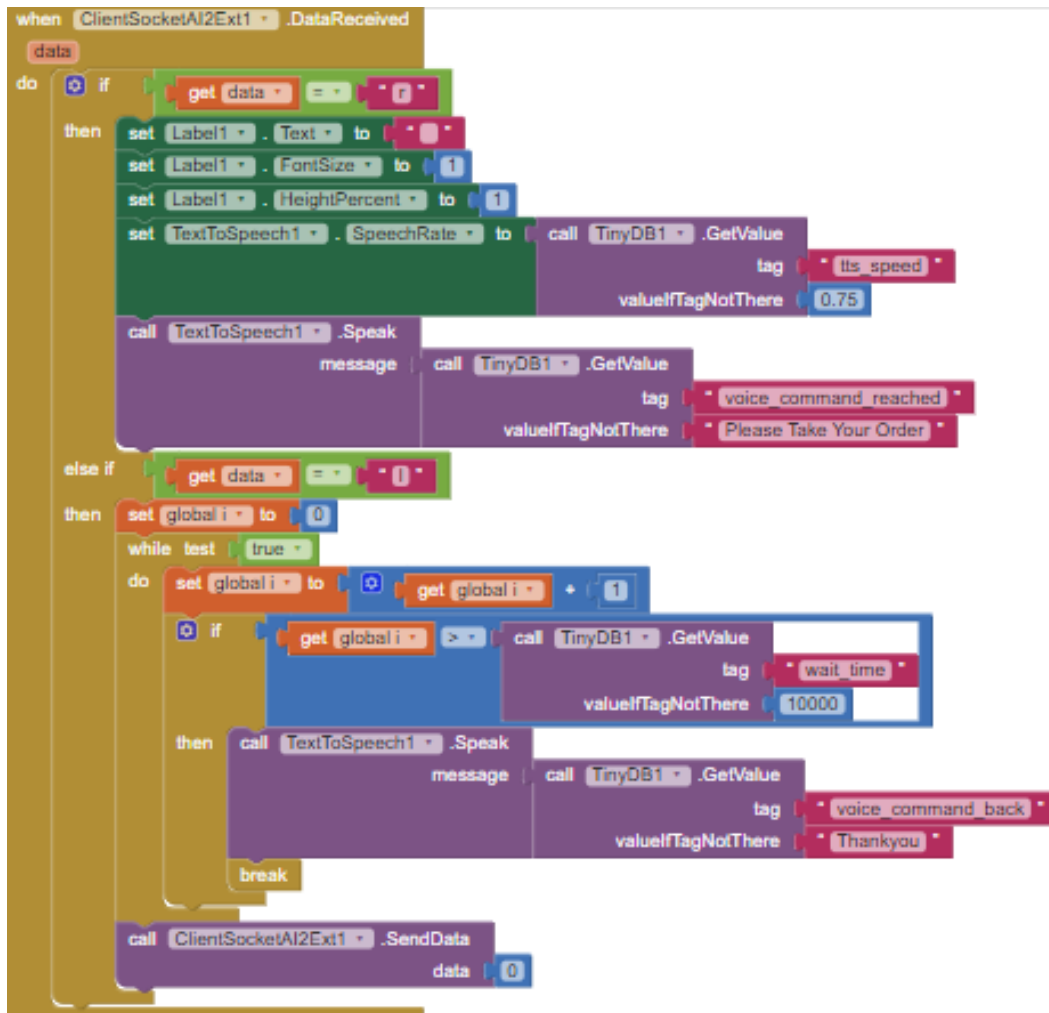


Figure 45: the communication code from the NodeMCU to the application

Whenever any data received from the NodeMCU, the “ClientSocketA12Ext1” will be called, and two options will be checked, whether it is ‘r’ (reached), or ‘P’ (Limit switch). This described in a Blockly language in figure 45. If the received data was ‘r’, the first action is to minimize the table number screen to a very small size 1 x 1 times to make it invisible. Now the most important code block is “TextToSpeech” which used to convert the written text in the text box on the setting to the speaker. First the rate of speed should be set (the speed of saying the words), in the setting, a scroll already added to do changing on the speed, so this Blockly code will take that value and ably it, otherwise a 0.75 rate will be set.



The rate has already set, now the statement that the robot should say one arrive need to set, the block “call TextToSpeech” is used to link the voice command of the robot once it reached with the app, if the text box was empty, the robot will say automatically “Please Take Your Order”

```

call TextToSpeech1 .Speak
  message call TinyDB1 .GetValue
    tag "voice_command_reached"
    valueIfTagNotThere "Please Take Your Order"
  
```

Similarly, for the case of “1”, of the application received the letter ‘1’ from the NodeMCU, which mean the limit switch already release, that mean the tray already removed, so the robot will start count some seconds as what added to the setting text box, otherwise if nothing is there, it will wait for 10 seconds then will say the voice command for the back action before it start moves back to the Home and initial position.

```

else if get data == "1"
then
  set global i to 0
  while test true
  do
    set global i to get global i + 1
    if get global i > call TinyDB1 .GetValue
      tag "wait time"
      valueIfTagNotThere 10000
    then
      call TextToSpeech1 .Speak
        message call TinyDB1 .GetValue
          tag "voice_command_back"
          valueIfTagNotThere "Thankyou"
      break
    call ClientSocketA12Ext1 .SendData
      data 0
  
```

Since there is no delay function in the android or the java application, the ‘global i’ is used to initiate some delay by counting from 0 to 1000, which equal to almost 3 sec delay. This is very important to have to give some time for the data to be transfer between the application and the NodeMCU to ensure the existence of the data.

Chapter 4 : OBJECT DETECTION-BASED ON SENSOR-FUSION SYSTEM

4.1: Introduction

Most service robots operate within an environment with a stationary starting point, a target position (table), and an array of randomly shaped obstacles that have finite surfaces. A continuous path must be found between the start and the target positions without colliding with other objects. To accomplish this, the controller along with the navigation system will be used. The navigation system is intended to allow the robot to plan its movements. Three factors must be considered when planning the robot's movements: A mapping module gathers the environmental information, transfers it into real-time for the robot to navigate, then generates a path so the robot can avoid collisions, in addition to controlling the robot's motion. Mobile search robots require the use of motion planning so that they can follow planned paths and continue to search and explore their environment. In light of the unknown position of the target, the search path must be carefully prepared in order to ensure it covers the entire searching area. In order for the service robot to accomplish its tasks, the navigation system must model the environment. Modelling consists of going through the environment using information from the mobile robot's sensors to determine the location of various objects, like tables, plant pots and other fixed or moving objects. Without mapping the environment, the mobile robot cannot navigate to the required tables, find a given location, or plan its path to the given location. Most applications for mobile robots do not have knowledge of the environment or the target location, so the environment modelling must be based on the similarities between different environments. Many aspects of the indoor environment can be utilized as a reference for the robot's motion, such as walls and doors. As the walls of an indoor environment allow the robot to operate autonomously in different indoor environments, they are crucial for designing an autonomous navigation system. Objects located on the floor of the environment may be obstructed by pieces of furniture and must be considered in designing the navigation system. A description of the exploration path and motion control is provided in this chapter. As it explains how the robot is guided by its environment, it uses the Bug algorithms to adapt and implement its exploration. Then, it describes the process of controlling the robot's motion, which is based on its feedback sensors information.

Many algorithms that are being employed by the robot modify those developed in the literature in small ways. According to this idea, the robot will be located at a predetermined point, have perfect localization abilities, and have perfect sensors. The system assumes that the robot knows a minimum number of places along its path, in addition to knowing the distance and direction between its starting point and the tables it is targeting. Specifically, the algorithms are designed to operate in two ways: to navigate to a target and to avoid obstacles by avoiding their boundaries. Adapting these algorithms is typically possible for robots that possess tactile sensors and an appropriate localization strategy, such as landmark detection. Figure 46 below is a simple illustration of the algorithm.

1. Drive towards the table Tx. Go to 2.
2. Is the table reached ?
 - a- Yes, STOP and say "Please take your order"
 - b- No, Go to (3)
3. Is any obstacle encountered ?
 - a- Yes, Go to (4)
 - b- No, Go to (1)
4. Go around the obstacle, and continue updating and saving the the remaining distance between the robot and the table, until:
 - a- The target reached. Stop
 - b- No, Go to (1)

Figure 46: The simplest algorithm procedures

Figure 47 represents the working sample environment. The blue box indicates the starting point of the robot, while the green identifies the target T. The dashed black line represents the desired path between the start and the goal, which is defined for the robot, while the green arrows denote the actual path of the robot's motion. The aims of all algorithms are explained as follows. The robot starts moving directly towards the target when it can; if it hits an obstacle, it goes forward along the intervening obstacle boundaries until that obstacle can be overcome and the robot continues directly towards the goal. However, if no path exists to the target, the algorithms are able to identify this state and they terminate, reporting that the goal is unreachable and keep stopping until any updates happen and the way is open for arrival.

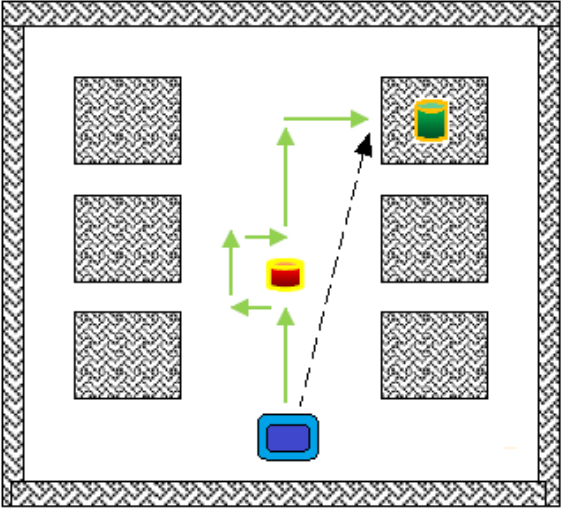


Figure 47: The algorithm states

Validation tests have been developed to demonstrate the amount of inaccuracy in robot placement, either at the table or at the station where the robot returns every time.

Table 8: motion angle deviation

Angle deviation during running the robot (One hour continues working, then charge it again)						
#	1st targeted table (1.5m)		2nd targeted table (3.5m)		3rd targeted table (5.5m)	
	At the Station	At the Table	At the Station	At the Table	At the Station	At the Table
01	0	3	0	1	0	5
02	1	3	3	5	7	4
03	3	1	4	5	6	7
04	4	5	3	5	4	7
05	2	3	2	6	5	8
06	3	4	9	2	9	10
07	1	2	5	5	7	13
08	1	1	4	3	11	15
09	2	5	4	7	15	16
10	6	4	3	7	12	12
11	7	3	6	8	20	11
12	9	4	5	6	17	17
13	8	7	7	8	13	19
14	10	5	10	12	14	16
15	6	2	11	14	19	21

Because of the lack of orientation feedback data, a cumulative error occurs on the orientation throughout the travel and after it arrives at its destination. Table 9 and Table 11 have listed a variety of tests that were performed to address this problem and to start enhancing the rotation accuracy.

Table 9: 1st Test summary:

		Number of Tests		
		Close distance targeted table (1.5m)	Mid distance targeted table (3.5m)	Far away distance targeted table (5.5m)
The angles variation at the station	Angle ≤ 7	12/15	12/15	07/15
	Angle > 7	03/15	03/15	08/15
The angles' variation at the table	Angle ≤ 7	15/15	11/15	04/15
	Angle > 7	00/15	04/15	11/15

To confirm the case and determine the specific fault that has to be reported for future system changes, the initial testing scenario has been reproduced 15 times. In comparison to the findings revealed in the mid and far targeted tables, the initial testing summary given in table 6 shows reasonably excellent outcomes. The orientation sensor allows the robot to maintain continuous angle adjustment, eliminating errors and avoiding the accumulation of errors.

Table 10: 1st Test summary:

Angle deviation during running the robot (One hour continues working, then charge it again)						
#	1st targeted table (1.5m)		2nd targeted table (3.5m)		3rd targeted table (5.5m)	
	At the Station	At the Table	At the Station	At the Table	At the Station	At the Table
01	0	2	0	2	0	6
02	3	1	6	7	8	6
03	3	4	4	4	7	8
04	2	5	3	6	3	6
05	4	2	1	5	4	9
06	3	6	6	3	8	11
07	1	5	4	5	10	14
08	2	3	5	6	12	18
09	5	2	7	7	14	17
10	7	8	8	6	18	14
11	7	4	8	9	21	12
12	8	6	9	12	18	19
13	11	7	10	11	17	21
14	11	8	12	15	16	15
15	9	9	15	19	20	19

Table 11: Replication test summary:

		Number of Tests		
		Close distance targeted table (1.5m)	Mid distance targeted table (3.5m)	Far away distance targeted table (5.5m)
The angles variation at the station	Angle ≤ 7	11/15	09/15	04/15
	Angle > 7	04/15	06/15	11/15
The angles' variation at the table	Angle ≤ 7	12/15	10/15	04/15
	Angle > 7	03/15	05/15	11/15

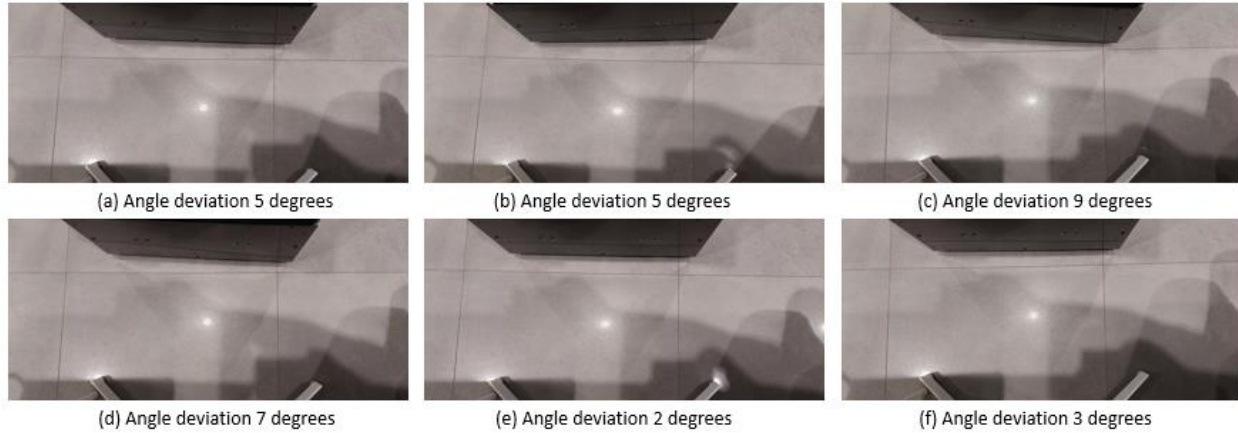


Figure 48: The angle deviation at the targeted table.

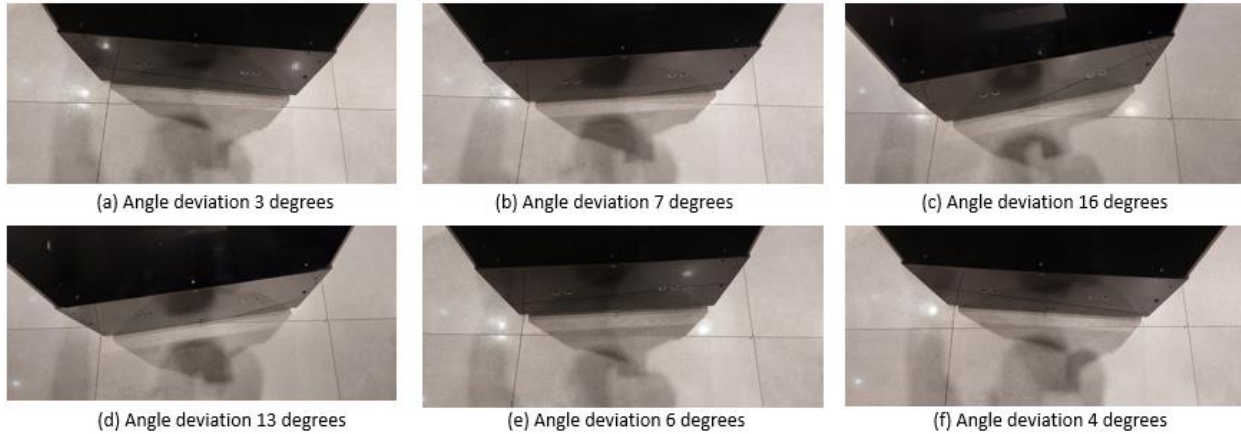


Figure 49: The angle deviation at the initial station.

4.2: OBJECT DETECTION PROBLEM

4.3: Sensor Configuration

In order for service robots to be autonomously operated, robust motion control systems are fundamental. According to the current state of a robot's environment, the control system chooses its actions. There are a variety of sensors that can be used to monitor the surrounding environment, including sensors that determine vision, range, and force. By transforming the external environment information into digital or electrical signals, the sensors provide the control system with the ability to control the robot physically. Motions pertaining to mobility are typically performed using the real hardware of mobile robots, including wheels, legs, etc. Figure 48 depicts the control process for a wheeled robot.

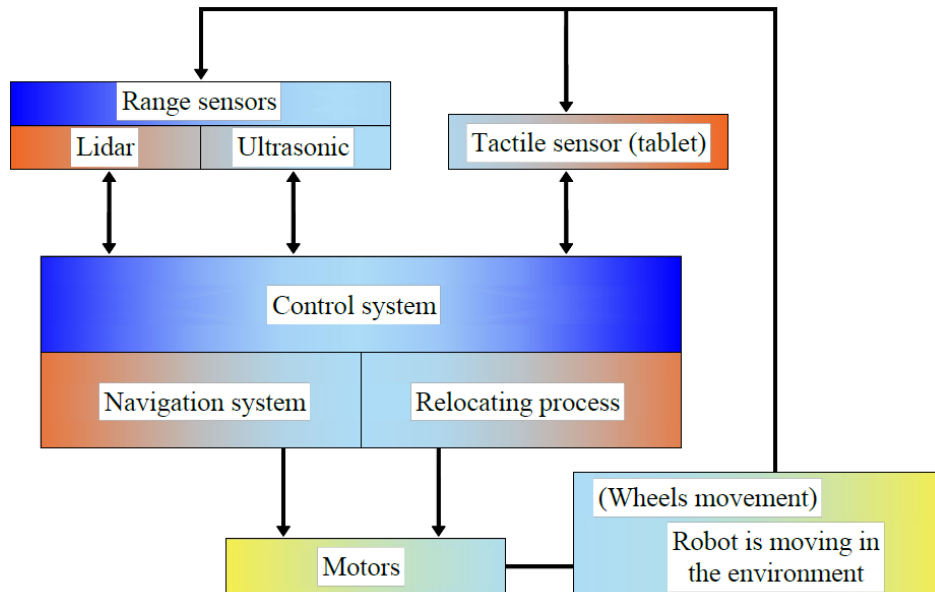


Figure 50: Control process

Depending on the type of sensor, as well as where the sensors are distributed on the robot's body, sensor configuration is determined. In the former case, specifications are mostly determined by the robot application and the amount of information provided. Vision sensors, ultrasonic range sensors, tactile sensors, and laser sensors are the most common types of sensors used to acquire environmental information. Sensors that are designed to detect vision are generally employed in applications that need a lot of information about their environment. For example, exploration and search and rescue applications. This is despite the fact that these sensors require processors with the ability to analyze images and extract information. Thus, it is clear why this project avoided such processors. In terms of cost and power consumption, range sensors are attractive. Furthermore, they only require a low amount of processing power to analyze their signals, so two kinds of range sensors have been employed: ultrasonics and lidars. Touch, pressure, force, and tactile sensors are transducers that sense touch, force, and pressure. Most commonly, they are used for improving the stability (balance) of robots with legs, as well as for controlling grip force. Laser sensors are widely used in navigation systems to map the environment and to localize robots because of their high accuracy. According to the size and application of the robot, the number and placement of sensors on its body will vary. If the robot is equipped with a high number of sensors, the accuracy of its measurement system will rise dramatically. The disadvantage of this is that it drives up the price of the robot and leads to a more complicated control system in its implementation. It should therefore be possible to reduce the number of sensors without affecting the movement of the robot.

4.3.1 Sensor Types

This section describes three types of sensors implemented in robot sensory platforms.

4.3.1.1 LIDAR DETECTING SENSOR

Throughout the study, 2D RPLIDAR 360 Degree was used as the primary range sensor. Using an infrared laser, it emits a modulated signal that is reflected by the object to be detected as shown in figure 49. RPLIDAR's vision acquisition system samples and processes a signal returned by the RPLIDAR and exhibits distance and angle values between the object and the RPLIDAR through the communication interface. The system measures distance data more than 2000 times per second and with high-resolution distance output (<1% of the distance).

With 360 points being sampled per round, the scanning frequency of RPLIDAR reached 5.5 Hz. And it can be configured to operate at a maximum rate of 10 Hz. Rx and Tx data ports are used for connecting the sensor to the mainboard. A lidar sensor is attached to the base of the robot for exploring its front and back sides.

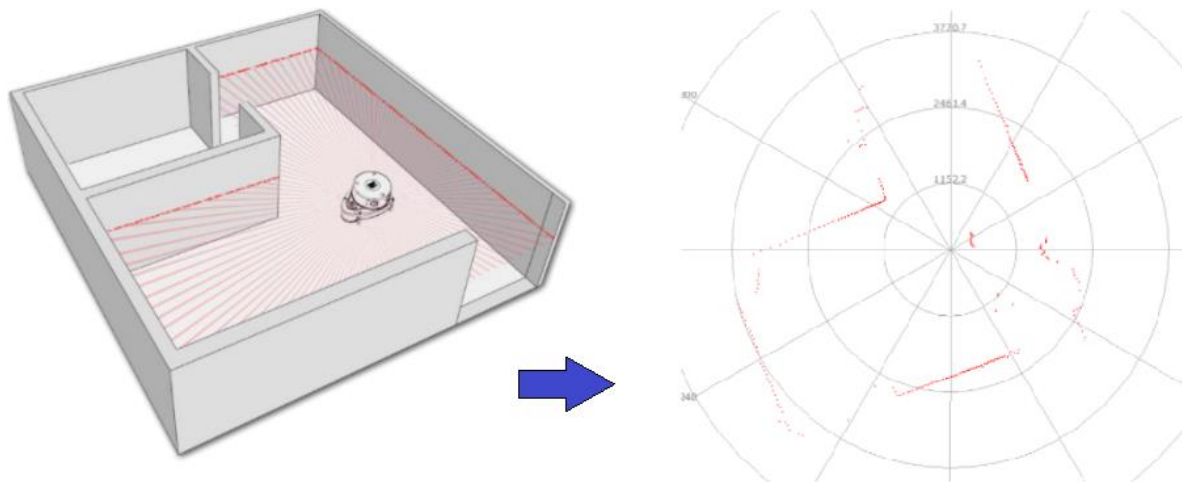


Figure 51: The Obtained Environment Map from RPLIDAR A1 Scanning

4.3.1.2 ULTRASONIC RANGE SENSORS

In a typical ultrasonic range sensor, an ultrasonic beam of pulses is emitted, with the reflected echoes returning back to the sensor as illustrated in figure 50. By calculating the time interval between emitting the signal and receiving the echo, the sensor determines how far the object is. Signals are generated in a cone shape, which covers a fixed volume. It depends on the transducer type whether there is a cone volume or a beamwidth. When choosing these sensors for robotics applications, beamwidth is a crucial parameter because it determines the boresight or the angle of range that the transducer covers when

facing straight-ahead. The robot could, for example, detect objects not directly in its path with a sensor that emits a beam within a large boresight.

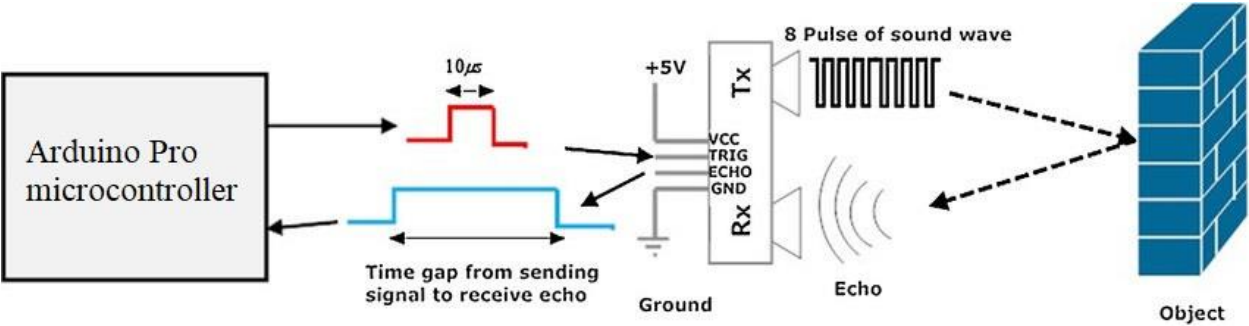


Figure 5482: the working principle of the ultrasonic

Chapter 5 : CONCLUSIONS AND FUTURE WORKS

5.1 CONCLUSION

A new framework for restaurant-based intelligent navigation systems is developed and implemented. These goals were achieved by developing an information-rich map structure called a semantic map structure. Several ideas are tested using a real restaurant and a mobile robot platform. Many tests have been carried out in actual scenarios to validate the efficacy and correctness of suggested navigation algorithms. With the use of simulations and tests, we were able to demonstrate the viability of our ideas and demonstrate that they work well in the areas of localization and navigation.

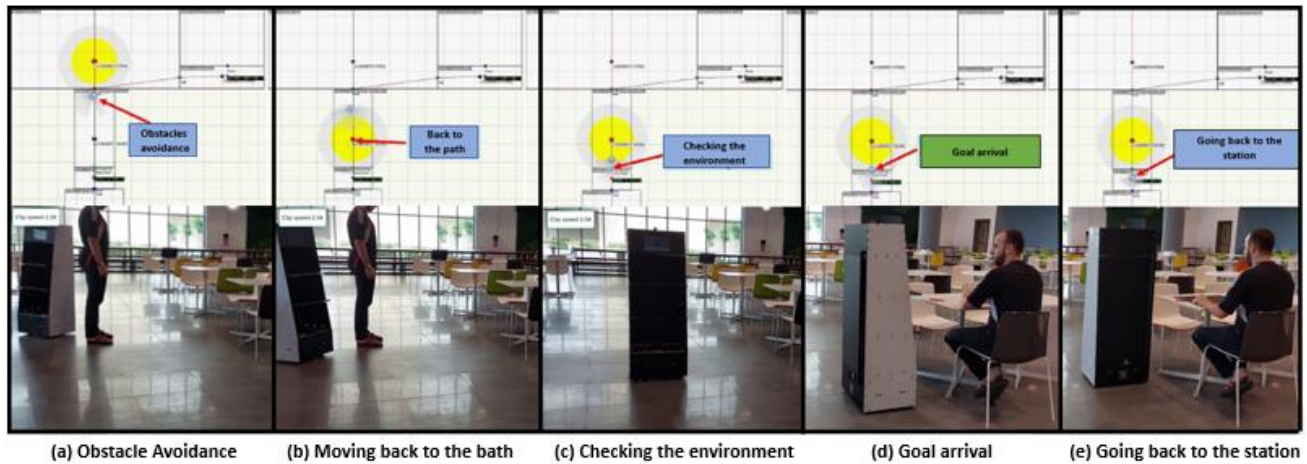


Figure 53: Overall processes that the robot navigates to designated goal position; a – e.

As a result of the main findings of this research work, the following conclusions have been drawn: According to the problem statement, numerous challenges are found in creating a robotics navigation system that works in a flexible restaurant environment. A novel autonomous control circuit has been implemented and tested, as it achieved the ability to overcome environmental difficulties. The robot has two sensors with customized calibration to ensure maximum area coverage. It shows a high sensitivity to the environment and the obstacles to reading and avoidance. 2.8% is the error found in reading the obstacles, as 1 out of 35 trials failed in reading the obstacles for many reasons, including the differences between the obstacle and the sensors' heights. The sensors are reading in a 2D scanning mode, which means that the obstacle must be at the same height or more to be detected by the sensor, which leads to a percentage of error. Adding a camera to the current sensor vision system will significantly improve the reading result and increase the system's accuracy percentage. Besides, by including one lidar and eight ultrasonics as an input data sensor, the robot technically requires more support by using an orientation sensor. Table 9 and Table 11 above showed the accumulated error produced once upon running the robot in a continuous running mode without stopping. The best temporary way to fix this issue is to relocate the robot using the same orientation every time. Ideally, a directional orientation sensor must be added to provide the orientation's auto-correction, which reduces the error to the minimum (+ 1.2)

Second, considering the matter being somewhat convoluted and essential for robotic research and development, a new customized real-time robot operating system has been developed, as it showed a competitive result compared with using ROS. For instance, the isolation technique used in this study gave a great result during the navigation process. A customized code created after getting LIDAR (rotating laser scanner) data out for mapping took advantage of almost no-latency wheel odometry and gyroscope data to process each LIDAR range data point, which was the current robot coordinate estimate, removing any stretching artifacts that would seep into the data if a LIDAR scan were handled as a single stationary "frame" while the robot was moving during the acquisition, as opposed to the ROS technique, which typically ignores this issue entirely. Although this problem is straightforward to solve on the integrated low level, extending to the integrated high level like ROS is complex. No extra code is used. The newest information is just used since all the information needed is already available in the main CPU. For ROS, this task becomes significantly more challenging when you have a non-real-time system interacting with non-real-time data coming from multiple, incompatible hardware devices and when there is a modularity barrier. In terms of money, this study excludes any expensive high processor computer, as everything runs on three small microcontrollers (Arduino Due, Arduino Pro, and NodeMCU). The new system is feature-rich, agile, quick, and energy-efficient. Even from the environmental point alone, we cannot afford to convert tens of watts of power into heat by running inefficient code inside the millions of robots we will swarm this planet with. So while others like ROS were putting together \$100-\$1000 sensor and actuator modules, some through high-latency USB, and processing the data always too late in a complex message passing architecture inside an expensive computer, our solution directly uses the \$1 components found inside those \$100 modules, all connected to a single microcontroller, running simple low-level real time code with no operating system, and providing the primary "central nerve" system of the robot, using 200mW of power to do that.

The designed control system for that robot was established and described, and the wheeled robot's construction was discussed. The design of a wheeled mechanism is utilized to demonstrate that the methodology is suitable for a search robot that operates independently in an indoor restaurant environment. The location of the robot has been effectively detected using an efficient method. Because the sensors are placed on the robot's body, it gives perfect autonomous navigation results, and the proposed search path covers the complete area for a robot wandering in a foreign environment. Furthermore, the proposed search approach was appropriate for the planned environment. This thesis aimed to design and build a navigation system for a service robot. It focuses on the issues of designing and running a restaurant delivery robot, which is meant to bring meals to consumers in an indoor restaurant setting. Several sub-problems were involved, such as mechanical design, alignments, reaching the tables by a predetermined path, and serving clients. The robot can walk along a predetermined path while continuously detecting its surroundings. Also, creating and implementing the mobile service robot was a significant problem; this paper detailed the robot's manufacturing approach. It mentioned a lengthy explanation that attempted to demonstrate most of the robot's modeling. However, the control system for such a robot was presented, as was the building of the wheeled robot. For the exploratory path, similar algorithms were utilized, with the robot designed to travel depending on the tables or the obstacle bounds. While moving beside the tables, the robot had to adjust its position and avoid the obstructions in its route, and it successfully said what it was supposed to say in both the arrival and when it left the targeted client table, which the user may edit.

5.2 PROPOSED FUTURE WORK

The mobile service robot encountered several challenges to reach this point, and the study examined its capabilities to find and deliver food in an indoor environment. Many more ideas can be implemented, and they will bring increased value to this robot by increasing its functionality and improving its accuracy, but due to many constraints, like timing and finances, the project has stopped at the point where it is now but listed several improvement ideas that could be adopted in the future to improve this robot. Future relevant work could include:

1. In the development of robot navigation (exploration paths), other navigation algorithms are implemented and simulated for comparison, as well as their effectiveness is demonstrated in field tests. Furthermore, some robot localization methods include simultaneous localization, SLAM, and Kalman filters.
2. With the help of a pan-tilt platform mounted on a robot with vision sensor capabilities, the robot can also search a larger area. As a result, the sensor fusion system will be improved, and the robot's perception of its surroundings will be enhanced. Also, the device's data feedback system is enhanced by integrating a directional sensor that keeps monitoring the trip orientation.
3. Since the robot has to work around people, it would be important to see the ability to improve the mechanical design to make it smoother and with fewer sharp edges, and in a way that helps in improving the sensor vision range, which will enhance the robot's ability to discover and understand the working environment aspects.
4. Adding a camera to the already existing sensor fusion system will make the robot more suitable, as this will increase the robot's ability to explore its environment in addition to deploying the robotic intelligence as a part of an autonomous system that uses artificial intelligence, such as a fuzzy controller or neural network-based control system.
5. Combined two or more robots to work in the same area. This will significantly increase the delivery time and service quality significantly.
6. Set up a microphone and speaker so that receptionists can place orders with clients at the table while communicating through the microphone
7. The charging system should be improved so that once the robots get back to the station, it automatically starts charging in a wireless mode
8. Enhance the robot's abilities to work independently and without human assistance (Self-ordering incorporating QR codes).
9. Finally, the idea is to design the base completely independent of the rest of the body, and this will provide flexibility by changing the robot's function to be, for instance, a reception robot that can receive people and talk with them by changing the user interface along with the required configuration for the speech and face recognition. This feature will provide a multi-use for the same robot base.

References

- [1] A. S. Khusheef, G. Kothapalli, and M. Tolouei-Rad, "Simulation of a mobile robot navigation system," *MODSIM 2011 - 19th Int. Congr. Model. Simul. - Sustain. Our Futur. Underst. Living with Uncertain.*, no. December, pp. 318–323, 2011, doi: 10.36334/modsim.2011.a3.khusheef.
- [2] J. Ng and T. Bräunl, "Performance comparison of Bug navigation algorithms," *J. Intell. Robot. Syst. Theory Appl.*, vol. 50, no. 1, pp. 73–84, 2007, doi: 10.1007/s10846-007-9157-6.
- [3] R. Negenborn, "Robot Localization and Kalman Filters," *Www. Negenborn Net/Kalman*, p. 145, 2003, [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.9672&rep=rep1&type=pdf>
<http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Robot+Localization+and+Kalman+Filters+On+nding+your+position+in+a+noisy+world#2%5Cnhttp://scholar.google.co>
- [4] J. Taheri and N. Sadati, "A fully modular online controller for robot navigation in static and dynamic environments," *Proc. IEEE Int. Symp. Comput. Intell. Robot. Autom. CIRA*, vol. 1, pp. 163–168, 2003, doi: 10.1109/CIRA.2003.1222082.
- [5] Y. Fukazawa, C. Trevai, J. Ota, H. Yuasa, T. Arai, and H. Asama, "Controlling a Mobile Robot that Searches for and Rearranges Objects with Unknown Locations and Shapes," *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2, no. October, pp. 1721–1726, 2003, doi: 10.1109/iro.2003.1248892.
- [6] T. Sogo, H. Ishiguro, and T. Ishida, "Mobile robot navigation by a distributed vision system," *New Gener. Comput.*, vol. 19, no. 2, pp. 121–137, 2001, doi: 10.1007/BF03037250.
- [7] A. Khusheef, "Investigation on the mobile robot navigation in an unknown environment," p. 9, 2013.
- [8] J. Wallén, "The history of the industrial robot," *Linköpings Univ.*, p. 18, 2008, [Online]. Available: <http://liu.diva-portal.org/smash/get/diva2:316930/FULLTEXT01.pdf>.
- [9] G. Bolmsjo, "Reconfigurable and Flexible Industrial Robot Systems," *Adv. Robot. Autom.*, vol. 03, no. 01, 2014, doi: 10.4172/2168-9695.1000117.
- [10] Y. Zhang, G. Tian, and H. Chen, "Exploring the cognitive process for service task in smart home: A robot service mechanism," *Futur. Gener. Comput. Syst.*, vol. 102, pp. 588–602, 2020, doi: 10.1016/j.future.2019.09.020.
- [11] M. F. Silva and J. A. T. MacHado, "A historical perspective of legged robots," *JVC/Journal Vib. Control*, vol. 13, no. 9–10, pp. 1447–1486, 2007, doi: 10.1177/1077546307078276.
- [12] M. H. Raibert, "Legged robots," *Commun. ACM*, vol. 29, no. 6, pp. 499–514, 1986, doi: 10.1145/5948.5950.
- [13] T. W. Mather and M. Yim, "Modular configuration design for a controlled fall," *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*, pp. 5905–5910, 2009, doi: 10.1109/IROS.2009.5354027.
- [14] A. A. Transteth and K. Y. Pettersen, "Developments in snake robot modeling and locomotion," *9th Int. Conf. Control. Autom. Robot. Vision, 2006, ICARCV '06*, 2006, doi: 10.1109/ICARCV.2006.345142.
- [15] S. Paluch, J. Wirtz, and W. H. Kunz, "Marketing Weiterdenken," *Mark. Weiterdenken*, no. October, 2020, doi: 10.1007/978-3-658-31563-4.

- [16] A. Corrales-Paredes, M. Malfaz, V. Egado-García, and M. Salichs, "Waymarking in social robots: Environment signaling using human–robot interaction," *Sensors*, vol. 21, no. 23, pp. 1–26, 2021, doi: 10.3390/s21238145.
- [17] L. Kunze and M. Beetz, "Envisioning the qualitative effects of robot manipulation actions using simulation-based projections," *Artif. Intell.*, vol. 247, pp. 352–380, 2017, doi: 10.1016/j.artint.2014.12.004.
- [18] Z. Cu-, "Th E Progress in Th E St U Dy of Oxygen Isot Ope s in Grou Ndwat Er N It Rat E," vol. 4, pp. 10–11, 2003.
- [19] SPARC, "Robotics 2020 - Multi-Annual Roadmap For Robotics in Europe," *Horiz. 2020 Call ICT-2016 (ICT-25 ICT-26), Release B 03/12/2015*, vol. 2016, 2015.
- [20] M. Prats, P. J. Sanz, A. P. Del Pobil, E. Marínez, and R. Marín, "Towards multipurpose autonomous manipulation with the UJI service robot," *Robotica*, vol. 25, no. 2, pp. 245–256, 2007, doi: 10.1017/S0263574706003304.
- [21] T. Breuer *et al.*, "Johnny: An autonomous service robot for domestic environments," *J. Intell. Robot. Syst. Theory Appl.*, vol. 66, no. 1–2, pp. 245–272, 2012, doi: 10.1007/s10846-011-9608-y.
- [22] J. Pransky, "AIBO - the no. 1 selling service robot," *Ind. Rob.*, vol. 28, no. 1, pp. 24–26, 2001, doi: 10.1108/01439910110380406.
- [23] J. Y. Sung, L. Guo, R. E. Grinter, and H. I. Christensen, "'My Roomba is Rambo': Intimate home appliances," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4717 LNCS, pp. 145–162, 2007, doi: 10.1007/978-3-540-74853-3_9.
- [24] M. L. Walters, D. S. Syrdal, K. Dautenhahn, R. te Boekhorst, and K. L. Koay, "Avoiding the uncanny valley: Robot appearance, personality and consistency of behavior in an attention-seeking home scenario for a robot companion," *Auton. Robots*, vol. 24, no. 2, pp. 159–178, 2008, doi: 10.1007/s10514-007-9058-3.
- [25] K. L. Koay, E. A. Sisbot, D. S. Syrdal, M. L. Walters, K. Dautenhahn, and R. Alami, "Exploratory study of a robot approaching a person in the context of handing over an object," *AAAI Spring Symp. - Tech. Rep.*, vol. SS-07-07, no. January, pp. 18–24, 2007.
- [26] B. He, S. Wang, and Y. Liu, "Underactuated robotics: A review," *Int. J. Adv. Robot. Syst.*, vol. 16, no. 4, pp. 1–29, 2019, doi: 10.1177/1729881419862164.
- [27] C. A. A. Calderon, C. Zhou, and R. E. Mohan, "Development of an autonomous service robot for social interactions," *ICICS 2011 - 8th Int. Conf. Information, Commun. Signal Process.*, 2011, doi: 10.1109/ICICS.2011.6174259.
- [28] L. Cao and X. Zhu, "An autonomous service mobile robot for indoor environments," *Proc. 2020 Asia-Pacific Conf. Image Process. Electron. Comput. IPEC 2020*, pp. 8–15, 2020, doi: 10.1109/IPEC49694.2020.9115180.
- [29] P. Maolanon, K. Sukvichai, N. Chayopitak, and A. Takahashi, "Indoor Room Identify and Mapping with Virtual based SLAM using Furnitures and Household Objects Relationship based on CNNs," *10th Int. Conf. Inf. Commun. Technol. Embed. Syst. IC-ICTES 2019 - Proc.*, pp. 1–6, 2019, doi: 10.1109/ICTEmSys.2019.8695966.

- [30] J. Torres-Solis, T. H., and T. Chau, "A Review of Indoor Localization Technologies: towards Navigational Assistance for Topographical Disorientation," *Ambient Intell.*, 2010, doi: 10.5772/8678.
- [31] A. Ess, K. Schindler, B. Leibe, and L. Van Gool, "Object detection and tracking for autonomous navigation in dynamic environments," *Int. J. Rob. Res.*, vol. 29, no. 14, pp. 1707–1725, 2010, doi: 10.1177/0278364910365417.
- [32] S. I. A. P. Diddeniya, A. M. S. B. Adikari, H. N. Gunasinghe, P. R. S. De Silva, N. C. Ganegoda, and W. K. I. L. Wanniarachchi, "Vision Based Office Assistant Robot System for Indoor Office Environment," *2018 3rd Int. Conf. Inf. Technol. Res. ICITR 2018*, pp. 1–6, 2018, doi: 10.1109/ICITR.2018.8736141.
- [33] D. Tick, A. C. Satici, J. Shen, and N. Gans, "Tracking Control of Mobile Robots Localized via Chained Fusion of Discrete and Continuous," vol. 43, no. 4, pp. 1237–1250, 2013.
- [34] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System," *2013 IEEE Int. Symp. Safety, Secur. Rescue Robot. SSRR 2013*, 2013, doi: 10.1109/SSRR.2013.6719348.
- [35] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, 2016, doi: 10.1109/TITS.2015.2498841.
- [36] T. Gandhi and M. M. Trivedi, "Pedestrian protection systems: Issues, survey, and challenges," *IEEE Trans. Intell. Transp. Syst.*, vol. 8, no. 3, pp. 413–430, 2007, doi: 10.1109/TITS.2007.903444.
- [37] F. Gul, W. Rahiman, and S. S. Nazli Alhady, "A comprehensive study for robot navigation techniques," *Cogent Eng.*, vol. 6, no. 1, pp. 1–25, 2019, doi: 10.1080/23311916.2019.1632046.
- [38] H.-J. Kwak and G.-T. Park, "Study on the Mobility of Service Robots," *Int. J. Eng. Technol. Innov.*, vol. 2, no. January, pp. 97–112, 2012.
- [39] Y. K. Hwang and N. Ahuja, "Gross Motion Planing - A Survey," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 219–291, 1992, [Online]. Available: <http://dl.acm.org/citation.cfm?id=136037>.
- [40] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014, doi: 10.1109/ACCESS.2014.2302442.
- [41] C. Kahraman, M. Deveci, E. Boltürk, and S. Türk, "Fuzzy controlled humanoid robots: A literature review," *Rob. Auton. Syst.*, vol. 134, p. 103643, 2020, doi: 10.1016/j.robot.2020.103643.
- [42] D. Belanche, L. V. Casaló, C. Flavián, and J. Schepers, "Service robot implementation: a theoretical framework and research agenda," *Serv. Ind. J.*, vol. 40, no. 3–4, pp. 203–225, 2020, doi: 10.1080/02642069.2019.1672666.
- [43] G. McCartney and A. McCartney, "Rise of the machines: towards a conceptual service-robot research framework for the hospitality and tourism industry," *Int. J. Contemp. Hosp. Manag.*, vol. 13, no. 12, pp. 3835–3851, 2020, doi: 10.1108/IJCHM-05-2020-0450.
- [44] V. N. Lu *et al.*, "Service robots, customers and service employees: what can we learn from the academic literature and where are the gaps?," *J. Serv. Theory Pract.*, vol. 30, no. 3, pp. 361–391, 2020, doi: 10.1108/JSTP-04-2019-0088.

- [45] G. E. Jan, K. Y. Chang, and I. Parberry, "Optimal path planning for mobile robot navigation," *IEEE/ASME Trans. Mechatronics*, vol. 13, no. 4, pp. 451–460, 2008, doi: 10.1109/TMECH.2008.2000822.
- [46] N. Sariff and N. Buniyamin, "An overview of autonomous mobile robot path planning algorithms," *SCORED 2006 - Proc. 2006 4th Student Conf. Res. Dev. "Towards Enhancing Res. Excell. Reg.*, no. August, pp. 183–188, 2006, doi: 10.1109/SCORED.2006.4339335.
- [47] G. N. Coelho, "Autonomous Mobile Robot Navigation using Smartphones e Guilherme Nogueira Coelho dos Santos Dissertation for the achievement of the degree Master in Information Systems and Computer Engineering," no. November, 2008.
- [48] C. Stachniss, G. Grisetti, and W. Burgard, "Recovering particle diversity in a Rao-Blackwellized particle filter for SLAM after actively closing loops," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2005, no. May 2005, pp. 655–660, 2005, doi: 10.1109/ROBOT.2005.1570192.
- [49] A. Censi, L. Locchi, and G. Grisetti, "Scan matching in the hough domain," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2005, no. January, pp. 2739–2744, 2005, doi: 10.1109/ROBOT.2005.1570528.
- [50] B. Yamauchi, "A frontier-based exploration for autonomous exploration," *IEEE Int. Symp. Comput. Intell. Robot. Autom. Monterey, CA*, pp. 146–151, 1997, [Online]. Available: <https://pdfs.semanticscholar.org/9afb/8b6ee449e1ddf1268ace8efb4b69578b94f6.pdf>.
- [51] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996, doi: 10.1109/70.508439.
- [52] J. J. Kuffner and S. M. La Valle, "RRT-connect: an efficient approach to single-query path planning," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2, no. Icara, pp. 995–1001, 2000, doi: 10.1109/robot.2000.844730.
- [53] D. Calisi, A. Farinelli, L. Locchi, and D. Nardi, "Autonomous navigation and exploration in a rescue environment," *Proc. 2005 IEEE Int. Work. Safety, Secur. Rescue Robot.*, vol. 2005, no. June 2014, pp. 54–59, 2005, doi: 10.1109/SSRR.2005.1501268.
- [54] H. Miao and Y. C. Tian, "Robot path planning in dynamic environments using a simulated annealing based approach," *2008 10th Int. Conf. Control. Autom. Robot. Vision, ICARCV 2008*, no. December, pp. 1253–1258, 2008, doi: 10.1109/ICARCV.2008.4795701.
- [55] Y. Zhu, T. Zhang, J. Song, and X. Li, "A new bug-type navigation algorithm considering practical implementation issues for mobile robots," *2010 IEEE Int. Conf. Robot. Biomimetics, ROBIO 2010*, pp. 531–536, 2010, doi: 10.1109/ROBIO.2010.5723382.
- [56] H. Dong, W. Li, J. Zhu, and S. Duan, "The path planning for mobile robot based on Voronoi diagram," *Proc. - 3rd Int. Conf. Intell. Networks Intell. Syst. ICINIS 2010*, pp. 446–449, 2010, doi: 10.1109/ICINIS.2010.105.
- [57] A. Sarmiento, R. Murrieta, and S. A. Hutchinson, "An Efficient Strategy for Rapidly Finding an Object in a Polygonal World," *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2, no. October, pp. 1153–1158, 2003, doi: 10.1109/iros.2003.1248801.
- [58] H. Lau, S. Huang, and G. Dissanayake, "Optimal search for multiple targets in a built environment," *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, pp. 3740–3745, 2005, doi:

- 10.1109/IROS.2005.1544986.
- [59] B. Tovar, S. M. La Valle, and R. Murrieta, "Optimal navigation and object finding without geometric maps or localization," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 1, pp. 464–470, 2003, doi: 10.1109/robot.2003.1241638.
- [60] D. H. Kim and S. Shin, "Local path planning using a new artificial potential function composition and its analytical design guidelines," *Adv. Robot.*, vol. 20, no. 1, pp. 115–135, 2006, doi: 10.1163/156855306775275530.
- [61] P. Raja and S. Pugazhenth, "Path planning for a mobile robot in dynamic environments," *Int. J. Phys. Sci.*, vol. 6, no. 20, pp. 4721–4731, 2011, doi: 10.5897/IJPS11.573.
- [62] H. P. Moravec, "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover, Tech. Report, Robotics Institute, Carnegie-Mellon University, pp.1-175," *Obs. Avoid. Navig. Real World by a Seeing Robot Rover*, pp. 1–175, 1980, [Online]. Available: <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA092604>.
- [63] T. Balch and R. Arkin, "Avoiding the past: A simple but effective strategy for reactive navigation," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 1, pp. 678–684, 1993, doi: 10.1109/robot.1993.292057.
- [64] G. A. S. Pereira, S. Choudhury, and S. Scherer, "A framework for optimal repairing of vector field-based motion plans," *2016 Int. Conf. Unmanned Aircr. Syst. ICUAS 2016*, pp. 261–266, 2016, doi: 10.1109/ICUAS.2016.7502525.
- [65] H. Chang, J. S. Choi, and M. Kim, "Experimental research of probabilistic localization of service robots using range image data and indoor GPS system," *IEEE Int. Conf. Emerg. Technol. Fact. Autom. ETFA*, vol. 00, pp. 1021–1027, 2006, doi: 10.1109/ETFa.2006.355414.
- [66] G. A. S. Pereira and E. J. R. Freitas, "Navigation of Semi-autonomous Service Robots Using Local Information and Anytime Motion Planners," *Robotica*, vol. 38, no. 11, pp. 2080–2098, 2020, doi: 10.1017/S0263574719001838.
- [67] G. Macesanu and T. Cocias, "Development of GTBoT, a High Performance and Modular Indoor Robot." .
- [68] A. K. Ray, L. Behera, and M. Jamshidi, "GPS and sonar based area mapping and navigation by mobile robots," *IEEE Int. Conf. Ind. Informatics*, pp. 801–806, 2009, doi: 10.1109/INDIN.2009.5195905.
- [69] K. Shubina and J. K. Tsotsos, "Visual search for an object in a 3D environment using a mobile robot," *Comput. Vis. Image Underst.*, vol. 114, no. 5, pp. 535–547, 2010, doi: 10.1016/j.cviu.2009.06.010.
- [70] A. V. Le, K. Ganesh, S. Apuroop, S. Konduri, H. Do, and M. R. Elara, "SS symmetry Multirobot Formation with Sensor Fusion-Based Localization in," pp. 1–18, 2021.
- [71] A. V. Le *et al.*, "Towards optimal hydro-blasting in reconfigurable climbing system for corroded ship hull cleaning and maintenance," *Expert Syst. Appl.*, vol. 170, no. December 2020, p. 114519, 2021, doi: 10.1016/j.eswa.2020.114519.
- [72] L. Montesano, J. Gaspar, J. Santos-Victor, and L. Montano, "Cooperative localization by fusing vision-based bearing measurements and motion," *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*

- IROS*, pp. 2333–2338, 2005, doi: 10.1109/IROS.2005.1544953.
- [73] A. Diosi and L. Kleeman, “Advanced sonar and laser range finder fusion for simultaneous localization and mapping,” *2004 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, vol. 2, pp. 1854–1859, 2004, doi: 10.1109/iros.2004.1389667.
- [74] M. A. Batalin, G. S. Sukhatme, and M. Hattig, “Mobile robot navigation using a sensor network,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2004, no. 1, pp. 636–641, 2004, doi: 10.1109/robot.2004.1307220.
- [75] G. National and H. Pillars, “Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation,” 1998.
- [76] P. K. Panigrahi and S. K. Bisoy, “Localization strategies for autonomous mobile robots: A review,” *J. King Saud Univ. - Comput. Inf. Sci.*, no. xxxx, 2021, doi: 10.1016/j.jksuci.2021.02.015.
- [77] D. Blanco, B. L. Boada, L. Moreno, and M. A. Salichs, “Local mapping from online laser Voronoi extraction,” pp. 103–108, 2002, doi: 10.1109/iros.2000.894589.
- [78] M. Asada, Y. Fukui, and S. Tsuji, “Representing Global World of a Mobile Robot with Relational Local Maps,” *IEEE Trans. Syst. Man Cybern.*, vol. 20, no. 6, pp. 1456–1461, 1990, doi: 10.1109/21.61215.
- [79] M. E. Liggins, D. L. Hall, and J. Llinas, *Handbook of Multisensor Data Fusion: Theory and Practice*. 2009.
- [80] G. National and H. Pillars, *Integration Coordination and Control of Multi-sensor Robot Systems*. .
- [81] A. Steinberg and C. Bowman, “Revisions to the JDL Data Fusion Model,” pp. 45–67, 2008, doi: 10.1201/9781420053098.ch3.
- [82] B. V. Dasarathy, “Information Fusion – what, where, why, when, and how?,” *Inf. Fusion*, vol. 2, no. 2, pp. 75–76, 2001, doi: 10.1016/s1566-2535(01)00032-x.
- [83] S. Das, *High level data Fusion*, vol. 53, no. 9. 2015.
- [84] K. Nagla, M. Uddin, and D. Singh, “Multisensor Data Fusion and Integration for Mobile Robots: A Review,” *IAES Int. J. Robot. Autom.*, vol. 3, no. 2, pp. 131–138, 2014, doi: 10.11591/ijra.v3i2.4075.
- [85] B. Ayrulu, B. Barshan, I. Erkmen, and A. Erkmen, “Evidential logical sensing using multiple sonars for the identification of target primitives in a mobile robot’s environment,” *IEEE Int. Conf. Multisens. Fusion Integr. Intell. Syst.*, vol. 1, pp. 365–372, 1996, doi: 10.1109/mfi.1996.572202.
- [86] O. Cohen and Y. Edan, “A sensor fusion framework for online sensor and algorithm selection,” *Rob. Auton. Syst.*, vol. 56, no. 9, pp. 762–776, 2008, doi: 10.1016/j.robot.2007.12.002.
- [87] S. J. Henkind and M. C. Harrison, “An Analysis of Four Uncertainty Calculi,” *IEEE Trans. Syst. Man Cybern.*, vol. 18, no. 5, pp. 700–714, 1988, doi: 10.1109/21.21598.
- [88] A. Elfes, “Sonar-Based Real-World Mapping and Navigation,” *IEEE J. Robot. Autom.*, vol. 3, no. 3, pp. 249–265, 1987, doi: 10.1109/JRA.1987.1087096.