MOBILE ROBOT MAPPING AND NAVIGATION USING ULTRASONIC SENSOR AND IMU SENSOR

A PROJECT REPORT

Submitted by

ESHA G (2019507014)

GANESH B (2019507016)

HARIHARAN Ve (2019507019)

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

PRODUCTION ENGINEERING

MADRAS INSTITUTE OF TECHNOLOGY CAMPUS

ANNA UNIVERSITY: CHENNAI 600 044

JUNE 2023

ANNA UNIVERSITY: CHENNAI 600 044

BONAFIDE CERTIFICATE

Certified that this project report entitled "**MOBILE ROBOT NAVIGATION USING ULTRASONIC SENSOR AND IMU SENSOR**" is the bonafide work of **ESHA G (2019507014), GANESH B (2019507016), HARIHARAN Ve (2019507019),** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr. A. SIDDHARTHAN,	**Dr. P. KARTHIKEYAN**,
Professor and Head,	Assistant Professor (Sr. Gr) and Supervisor,
Department of Production Technology,	Department of Production Technology,
Madras Institute of Technology,	Madras Institute of Technology,
Anna University, Chennai-600 044.	Anna University, Chennai-600 044.

ACKNOWLEDGEMENT

It is a pleasure to place on record our profound sense of gratitude to our Supervisor **Dr. P. Karthikeyan**, Assistant Professor, Department of Production Technology, Madras Institute of Technology, for his valuable guidance, advice, motivation, and encouragement which helped us immensely to accomplish the second phase of this project successfully.

We acknowledge our sincere thanks to the project committee members, **Dr. V. Mugendiran,** Assistant Professor, **Mr. R. Mathiyazhagan,** Teaching Fellow, **Mr. P. Mani,** Teaching Fellow and **Mr. K. Muthukumaran,** Teaching Fellow, Department of Production Technology, Madras Institute of Technology for their valuable suggestions.

We sincerely thank **Mr. P. Murali,** Teaching fellow, **Mr. V. Dinesh Kumar,** Professional Assistant, Department of Production Technology, Madras Institute of Technology Campus, for their constant support.

We express deep sense of gratitude to **Dr. A. Siddharthan**, the Head of the Department and Professor, Department of Production Technology, Madras Institute of Technology for his valuable help and in depth experienced advises which helped me in completion of this project successfully. We express heartful gratitude to **Dr. J. Prakash**, Dean, Madras Institute of Technology, for providing me an opportunity to carry out this project.

ESHA G

GANESH B

HARIHARAN Ve

ABSTRACT

This project focuses on developing a mobile robot navigation system that utilizes ultrasonic sensors and an Inertial Measurement Unit (IMU) sensor for improved obstacle detection, localization, and mapping. The objective is to create an intelligent system that enables the robot to autonomously navigate in indoor environments, avoiding obstacles and accurately determining its position. The existing navigation systems often face limitations in terms of obstacle detection and localization accuracy. By integrating ultrasonic sensors and IMU sensors, this project aims to overcome these limitations and provide a robust and reliable solution. The key challenges addressed in this project include obstacle avoidance, localization, and mapping. Furthermore, a localization and mapping algorithm is implemented to accurately determine the robot's position within the environment. This allows the robot to navigate autonomously and create a map of its surroundings. The outcomes of this project have significant implications for various applications, such as autonomous exploration, surveillance, and assistive robotics. The integration of ultrasonic sensors and IMU sensors enhances mobile robot navigation capabilities, leading to improved performance, reliability, and adaptability in indoor environments.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF ABBREVIATIONS

IMU - Inertial Measurement Unit

DIY - Do it yourself

DC Motor - Direct Current Motor

Wi-Fi – Wireless Fidelity

SLAM – Simultaneous Localization and Mapping

LiDAR – Light Detection and Ranging

DRL – Deep Reinforcement Learning

CNN- Convolutional neural network

EKF- Extended Kalman Filter

ROS – Robot Operating System

GUI- Graphic User Interface

ICSP – In-circuit Serial Programming

SS- Stainless Steel

DMP – Dynamic Movement Primitives

ESP- Electronic Stability Program

AI- Artificial Intelligence

AGV- Automated Guided Vehicle

IDE- Integrated Development Environment

UDP- User Datagram protocol

LIST OF TABLES

CHAPTER 1

1. INTRODUCTION

Mobile robots are becoming increasingly popular in various fields, including manufacturing, healthcare, and defence. These robots can perform tasks that are hazardous, repetitive, or require high precision. The ability of mobile robots to navigate autonomously is essential for their successful operation in these settings. Navigation involves identifying obstacles and determining the robot's position and orientation with respect to its environment.

The present project focuses on the development of a mobile robot navigation system using ultrasonic and IMU sensors. The robot is designed to navigate autonomously in an indoor environment while avoiding obstacles. The report outlines the various components of the robot and the challenges faced during the project.

The need for mobile robot navigation systems arises from the limitations of traditional navigation techniques. Conventional navigation methods, such as manual control and pre-programmed paths, are not suitable for mobile robots operating in dynamic environments. These robots need to be able to navigate autonomously to perform tasks efficiently and safely. Autonomous navigation also reduces the workload of human operators, allowing them to focus on other aspects of the task.

The development of mobile robot navigation systems has been driven by advances in sensor technology, computing power, and artificial intelligence. Ultrasonic and IMU sensors are commonly used in mobile robot navigation systems due to their low cost and high accuracy. These sensors can provide information about the robot's surroundings and its orientation, enabling it to navigate autonomously.

The present project involves the development of a mobile robot navigation system using ultrasonic and IMU sensors. The robot is designed to navigate autonomously in an indoor environment while avoiding obstacles.

The development of a mobile robot navigation system using ultrasonic and IMU sensors is an essential step towards the widespread adoption of mobile robots in various fields. The present project report outlines the various components of the robot, the methodology and the algorithm used, related dependencies and the challenges faced during its development. The project demonstrates the capabilities of mobile robots in navigating autonomously and avoiding obstacles using low-cost and high-accuracy sensors. The future work could involve the integration of additional sensors, such as cameras and lidar, to improve the robot's navigation capabilities further.

1.1 EXISTING SYSTEM AND THEIR DRAWBACKS

- **Vision-based navigation:** This system uses cameras to track the robot's position and orientation relative to its surroundings. Vision-based navigation systems are typically very accurate, but they can be susceptible to noise and occlusions. It is also difficult to implement in low light conditions.
- **Lidar-based navigation:** This system uses a laser scanner to create a 3D map of the environment. The robot's position and orientation can then be

determined by comparing the map to its own sensor data. Lidar-based navigation systems are very accurate and can be used in a variety of environments, but they are also expensive and complex. They are also difficult to use in cluttered environment.

- **Inertial navigation:** This system uses a combination of accelerometers, gyroscopes, and magnetometers to track the robot's motion. Inertial navigation systems are relatively inexpensive and easy to implement, but they are not very accurate over long distances. They can be affected by environmental factors such as temperature and vibrations.
- **Odometry:** This system uses wheel encoders to track the robot's movement. Odometry systems are inexpensive and easy to implement, but they are not very accurate in environments with uneven surfaces or obstacles. It can be affected by wheel's spinnage.

Each of these navigation systems has its own advantages and disadvantages. The best system for a particular application will depend on a number of factors, including the environment in which the robot will be operating, the accuracy requirements, and the budget. Hence based on the various factors, ultrasonic sensor and IMU sensor has been chosen due to its:

- **Low cost:** Ultrasonic and IMU sensors are relatively inexpensive, making them a cost-effective option for autonomous mobile robots.
- **Easy to use:** Ultrasonic and IMU sensors are relatively easy to use and install, making them a good choice for DIY projects.
- **Reliable:** Ultrasonic and IMU sensors are reliable and can be used in a variety of environments.

- **Accurate:** Ultrasonic and IMU sensors can provide accurate measurements of distance and orientation, making them a good choice for applications where precision is important.

1.2 PROBLEM STATEMENT

- Develop an intelligent sensor fusion algorithm that integrates data from ultrasonic sensors and IMU sensors to obtain reliable and precise information about the robot's environment, including obstacle detection, distance measurements, and position estimation.
- Design a robust obstacle avoidance strategy that utilizes the sensor data to detect and avoid obstacles in real-time. The system should be able to handle static and dynamic obstacles efficiently, ensuring the safety of the robot and its surroundings.
- Implement a localization and mapping algorithm that utilizes the combined sensor data to accurately determine the robot's position and orientation within the environment. This will enable the robot to navigate autonomously and create a map of its surroundings for future reference.

1.3 OBJECTIVE

- Design and build a mobile robot with a rectangular chassis made of lightweight material suitable for indoor environments.
- Use ultrasonic sensors for obstacle detection and avoidance and implement sensor fusion with IMU sensors to improve navigation accuracy and reliability.
- Develop a control algorithm for the robot to navigate through an obstacle course autonomously.

- Implement wireless communication using Bluetooth or WiFi to allow remote control and monitoring of the robot's position and status.
- Evaluate the performance of the robot in terms of accuracy, speed, and reliability, and identify potential areas for improvement and future development.

1.4 ORGANISATION OF REPORT

- The report consists of 5 chapters.
- Chapter 1 contains introduction, existing systems, and their drawbacks, defines the problem statement and the objective.
- Chapter 2 explains the literature survey and the summary of the research papers.
- Chapter 3 focusses on the process flow of the project, methodology of the processes, circuit diagram and the algorithm for Arduino and python code. It also briefs about the components used in the project, their specifications, and the bill of materials.
- Chapter 4 talks about the selection of the number of sensors, Design works which were involved, Libraries installed and the result of this project.
- Chapter 5 gives a conclusion, benefits, and drawbacks of this project.

CHAPTER 2

2. LITERATURE REVIEW

Chen et al. (2023) presents an improved navigation system for mobile robots using ultrasonic sensors. The system uses a combination of ultrasonic sensors and a genetic algorithm to estimate the robot's position and orientation. The system has been tested on a mobile robot and has been shown to be more accurate and robust than previous systems.

H. Zhang et al. (2023) proposes an IMU sensor-based mobile robot navigation system using a hybrid Kalman filter and neural network. The proposed system uses an IMU sensor to estimate the robot's position and orientation, and a hybrid Kalman filter and neural network to improve localization accuracy. Experimental results show that the proposed system can achieve accurate and reliable navigation in indoor and outdoor environments.

Giannoccaro et al. (2022) presents a method for processing LiDAR and IMU data to detect targets and estimate the odometry of a mobile robot. The method first uses the LiDAR data to create a point cloud of the robot's environment. The point cloud is then used to identify potential targets. The IMU data is then used to estimate the robot's pose relative to the target. The method has been tested on a mobile robot and has been shown to be effective in detecting targets and estimating the robot's odometry.

J. Zhang et al. (2022) presents a real-time mobile robot navigation system using IMU sensors and deep reinforcement learning (DRL). The proposed system uses an IMU sensor to estimate the robot's position and orientation, and a DRL algorithm to learn the optimal navigation policy. Experimental results show that the proposed system can achieve accurate and robust navigation in complex environments.

Yang et al. (2022) presents a novel ultrasonic sensor-based navigation system for mobile robots. The system uses a combination of ultrasonic sensors and a neural network to estimate the robot's position and orientation. The system has been tested on a mobile robot and has been shown to be effective in a variety of environments.

M. Chen et al. (2021) proposes an adaptive LiDAR-based obstacle avoidance algorithm for mobile robots. The algorithm uses a combination of LiDAR data and adaptive control techniques to generate smooth and safe trajectories for the robot. The proposed algorithm is evaluated on a mobile robot platform, and the results show that it can achieve effective obstacle avoidance in various environments.

Y. Zhang et al. (2021) presents a multi-sensor fusion approach for mobile robot navigation in complex environments. The approach combines LiDAR data, camera images, and inertial measurements to achieve accurate and robust localization and mapping. The proposed approach is evaluated on a mobile robot platform, and the results show that it can achieve high accuracy and efficiency in various indoor and outdoor environments.

S. Hosseini et al. (2021) proposed an IMU-based navigation system for autonomous mobile robots using a convolutional neural network (CNN). The proposed system uses an IMU sensor to estimate the robot's position and orientation, and a CNN to learn the correlation between IMU sensor data and robot position. Experimental results demonstrate that the proposed system can achieve accurate and robust navigation in both indoor and outdoor environments.

Liu et al. (2021) presents a method for obstacle detection and avoidance for mobile robots using ultrasonic sensors. The method uses a combination of ultrasonic sensors and a fuzzy logic controller to detect and avoid obstacles. The method has been tested on a mobile robot and has been shown to be effective in a variety of environments.

Zhang et al (2021) presents a research and implementation of autonomous navigation for mobile robots based on SLAM algorithm under ROS. The paper first introduces the SLAM algorithm and its applications in mobile robotics. The paper then discusses the implementation of the SLAM algorithm under ROS. The paper finally presents the experimental results of the autonomous navigation system. The results show that the system can successfully navigate the robot in a complex environment.

A. Farahat et al. (2020) proposes an efficient LiDAR-based path planning algorithm for mobile robots operating in dynamic environments. The algorithm considers the real-time sensor data from the LiDAR sensor and uses a grid-based representation of the environment to generate collision-free trajectories for the robot. The proposed algorithm is evaluated in simulation and on a real mobile robot, and the results show that it can generate safe and efficient paths in dynamic environments.

S. Choi et al. (2020) presents a LiDAR-based mobile robot navigation system for outdoor environments that considers dynamic objects such as pedestrians and vehicles. The system uses a combination of LiDAR data and deep learning-based object detection to detect and avoid dynamic obstacles in real-time. The proposed system is evaluated on a mobile robot platform, and the results show that it can achieve safe and efficient navigation in outdoor environments.

Y. Wang et al. (2020) presents a mobile robot navigation system using IMU sensors and particle filter localization. The proposed system uses a low-cost IMU sensor to estimate the robot's position and orientation, and a particle filter algorithm to improve localization accuracy. Experimental results show that the proposed system can achieve accurate and reliable navigation in indoor environments.

Zhang et al. (2020) presents a navigation system for mobile robots in indoor environments using ultrasonic sensors. The system uses a combination of ultrasonic sensors and a particle filter to estimate the robot's position and orientation. The system has been tested on a mobile robot and has been shown to be effective in a variety of indoor environments.

J. Li et al. (2019) proposed for LiDAR-based mobile robot navigation in indoor environments. The approach combines graph-based SLAM and pose graph optimization to achieve accurate and robust localization and mapping. The proposed approach is evaluated on a mobile robot equipped with a LiDAR sensor, and the results show that it can achieve high accuracy and efficiency in various indoor environments.

A. S. Mujawar et al. (2019) proposed a robust navigation system for a mobile robot using IMU sensors and an extended Kalman filter (EKF). The system is capable of accurately estimating the robot's position and orientation in real-time, even in the presence of external disturbances and sensor noise. The proposed system was tested on a mobile robot platform, and the results demonstrate its effectiveness in achieving accurate and robust navigation.

Li et al (2019) provides an overview of the sensors and data used in mobile robotics for localisation. The paper discusses the different types of sensors that can be used for localisation, including ultrasonic sensors, IMU sensors, LiDAR sensors, and cameras. The paper also discusses the different types of data that can be used for localisation, including odometry data, landmark data, and map data. The paper concludes by discussing the challenges of localisation in mobile robotics and the future trends in localisation research.

Wang et al (2019) presents a low-cost and robust navigation system for mobile robots using ultrasonic sensors. The system uses a combination of ultrasonic sensors and a Kalman filter to estimate the robot's position and orientation. The system has been tested on a mobile robot and has been shown to be effective in a variety of environments.

Suresh et al (2017) presents a method for obstacle detection using ultrasonic sensor for a mobile robot. The method first uses the ultrasonic sensor to measure the distance to the nearest obstacle. The distance is then used to determine if the obstacle is within the robot's safe range. If the obstacle is within the safe range, the robot will continue to move forward. If the obstacle is outside the safe range, the robot will stop and turn in a different direction. The method has been tested on a mobile robot and has been shown to be effective in detecting obstacles.

2.1 SUMMARY OF LITERATURE REVIEW

Ultrasonic sensors and IMU sensors can be used to estimate the position and orientation of a mobile robot. Ultrasonic sensors can be used to detect obstacles in the robot's path. IMU sensors can be used to track the robot's movement. A combination of ultrasonic sensors and IMU sensors can be used to create a navigation system for a mobile robot. The accuracy of the navigation system will depend on the number and placement of the ultrasonic sensors and the accuracy of the IMU sensor. The navigation system will be more robust to environmental conditions if it uses a combination of ultrasonic sensors and IMU sensors. The navigation system will be easier to use if it uses a combination of ultrasonic sensors and IMU sensors.

Overall, ultrasonic sensors and IMU sensors are a good option for mobile robot navigation in applications where cost, robustness, and ease of use are more important than accuracy and range. And moreover, on referring to those papers, it is found that ultrasonic sensor is a better option for an indoor application whereas LIDAR is beneficial and accurate in a dynamically complex environment. Since the scope of this project is limited to indoor environment, ultrasonic sensor is preferred.

The type of ultrasonic sensors to use will depend on the environment in which the robot will be operating. For example, if the robot will be operating in a noisy environment, ultrasonic sensors that are less susceptible to noise is chosen. The number of ultrasonic sensors used will depend on the size of the robot and the environment in which it will be operating. For example, if the robot is large and will be operating in a large environment, more ultrasonic sensors are used. The placement of the ultrasonic sensors will depend on the shape of the robot and the environment in which it will be operating. For example, if the robot is a cube, you will need to place the ultrasonic sensors on all sides of the robot. The accuracy of the IMU sensor will depend on the quality of the sensor. You will need to select an IMU sensor that is accurate enough for your application. The robustness of the navigation system will depend on the number and placement of the ultrasonic sensors and the accuracy of the IMU sensor.

You will need to design the navigation system to be robust enough for the environment in which the robot will be operating. The ease of use of the navigation system will depend on the user interface. You will need to design the user interface to be easy to use for the intended users.

CHAPTER 3

3.1 PROCESS FLOW

The process flow of the project represents the step-by-step process of Mobile robot navigation with ultrasonic sensor and IMU sensor and is shown in figure 3.1.

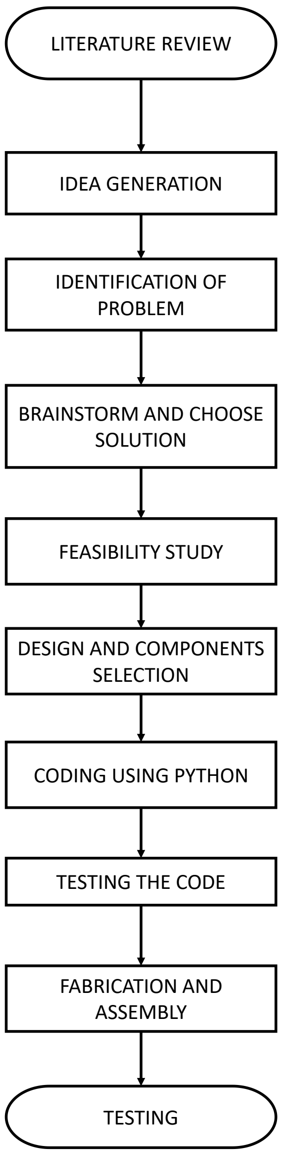


Figure 3.1: Process flow of this project

3.2 METHODOLOGY OF THE PROCESS

The Methodology of the project is represented below in figure 3.2 in a diagrammatic way. Figure 3.2 discuss the process done by robot while it is stationary. Whereas Figure 3.3 discuss the process done by robot while it is in motion. When the robot starts to move, the acceleration data of IMU sensor and the time duration data of the ultrasonic sensor is collected by the microcontroller. Then, the microcontroller will process those data like: The IMU's data is given in low frequency noise filter to eliminate noise in data and the filtered data is double integrated to get the position value. Whereas the ultrasonic sensor's data is formulated to calculate the distance between obstacle and sensor. Then the position and distance are visualized in computer screen.

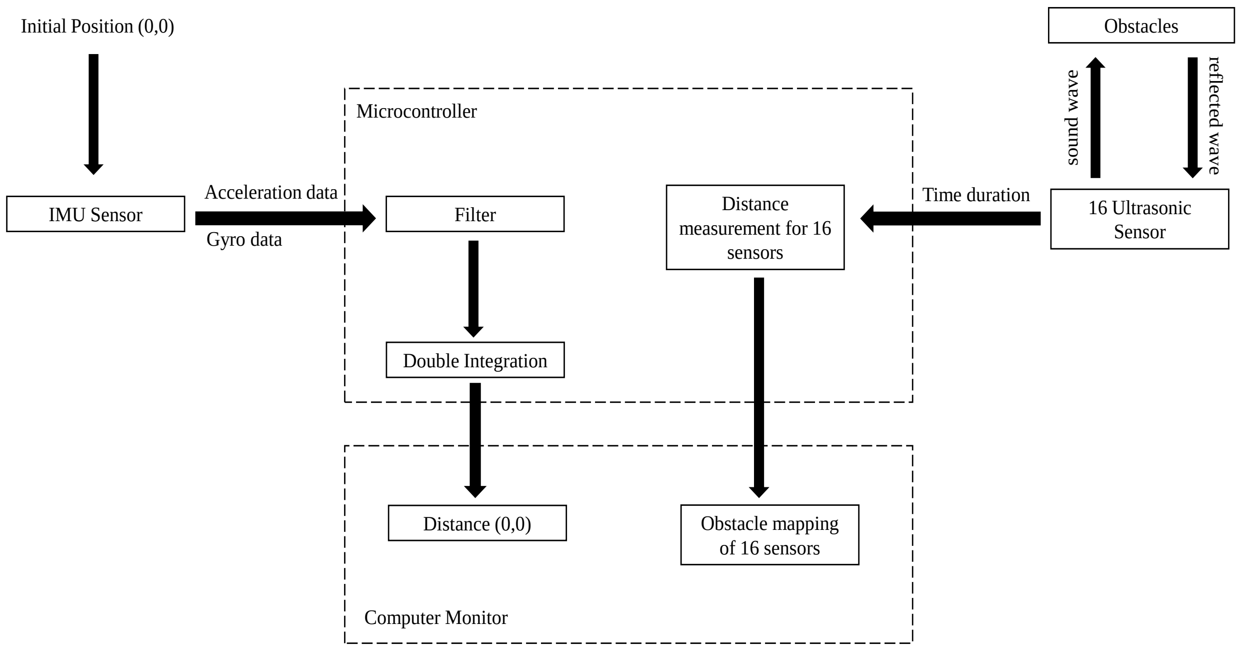


Figure 3.2: Methodology of the process when the robot is at initial position.

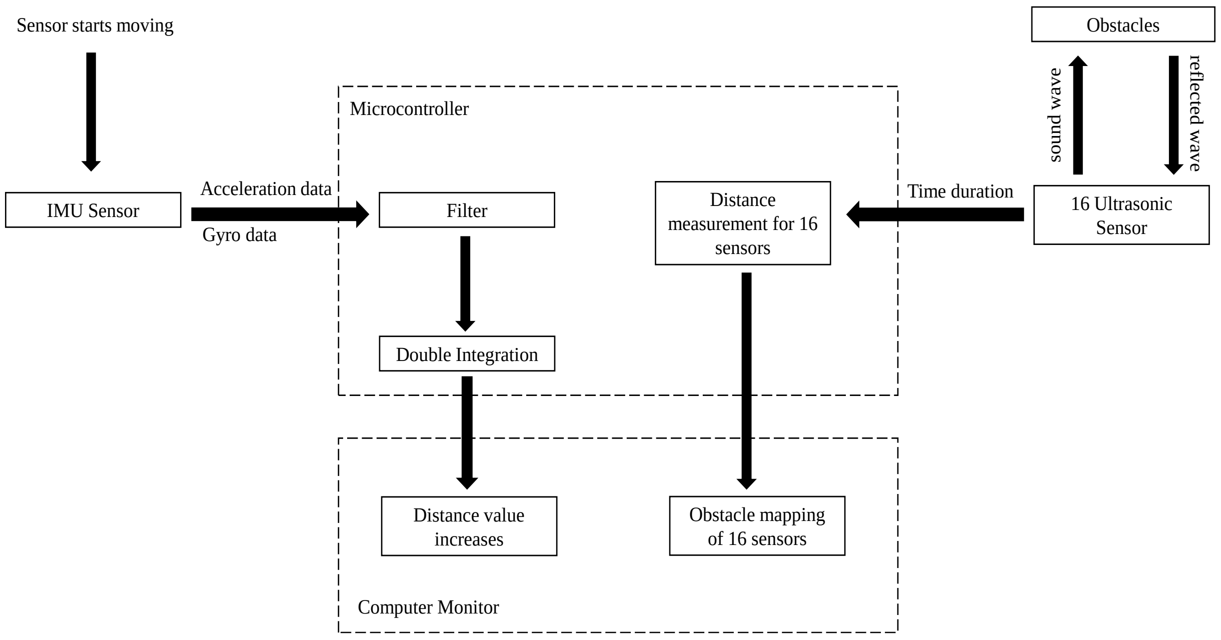


Figure 3.3: Methodology of the process when the robot is in motion.

3.3 CIRCUIT DIAGRAM

The below figure shows the pin diagram and connections of the entire setup graphically. The arrangement of 16 ultrasonic sensor, the power supply connections and Arduino mega connections are depicted in figure 3.4.

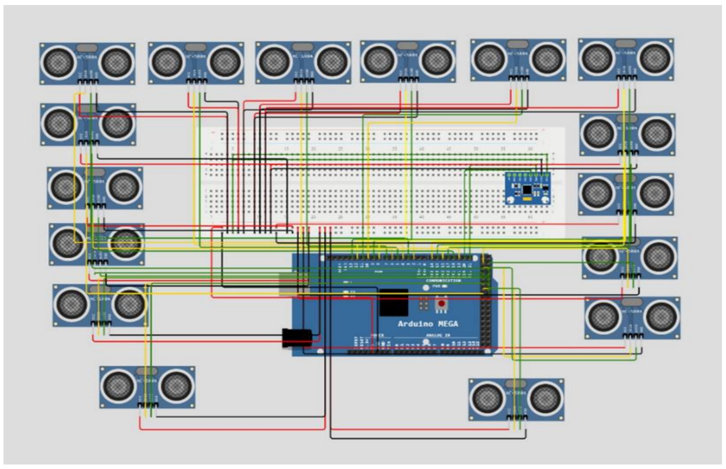


Figure 3.4: Circuit Diagram of the setup

3.4 ALGORITHM FOR ARDUINO CODE

STEP 1: Start

STEP 2: Set the variables for calculation of IMU sensor.

STEP 3: Set the variables for trigger pin and echo pin for 16 sensors.

STEP 4: Set the serial Baud rate to 9600Hz.

STEP 5: Get accelerated data from IMU sensor and the time duration data from 16 sensors.

STEP 6: Set the position from acceleration data.

STEP 7: Set the distances from time duration data.

STEP 8: Print the position and distances.

STEP 9: Stop

3.5 ALGORITHM FOR PYTHON CODE

STEP 1: Start

STEP 2: Set the turtle objects from the turtle GUI tool for animation.

STEP 3: Dump the Arduino code.

STEP 4: Get the output from microcontroller both the position and distances.

STEP 5: Separate the animation window into sixteen parts by using turtle objects.

STEP 6: Use a turtle object to visualise the distances data of 16 ultrasonic sensors.

STEP 7: Use a turtle object to visualise the position data of IMU sensor.

STEP 8: Repeat step 6 and 7 until stop.

STEP 9: Stop

3.6 FLOWCHART FOR MICROCONTROLLER CODE

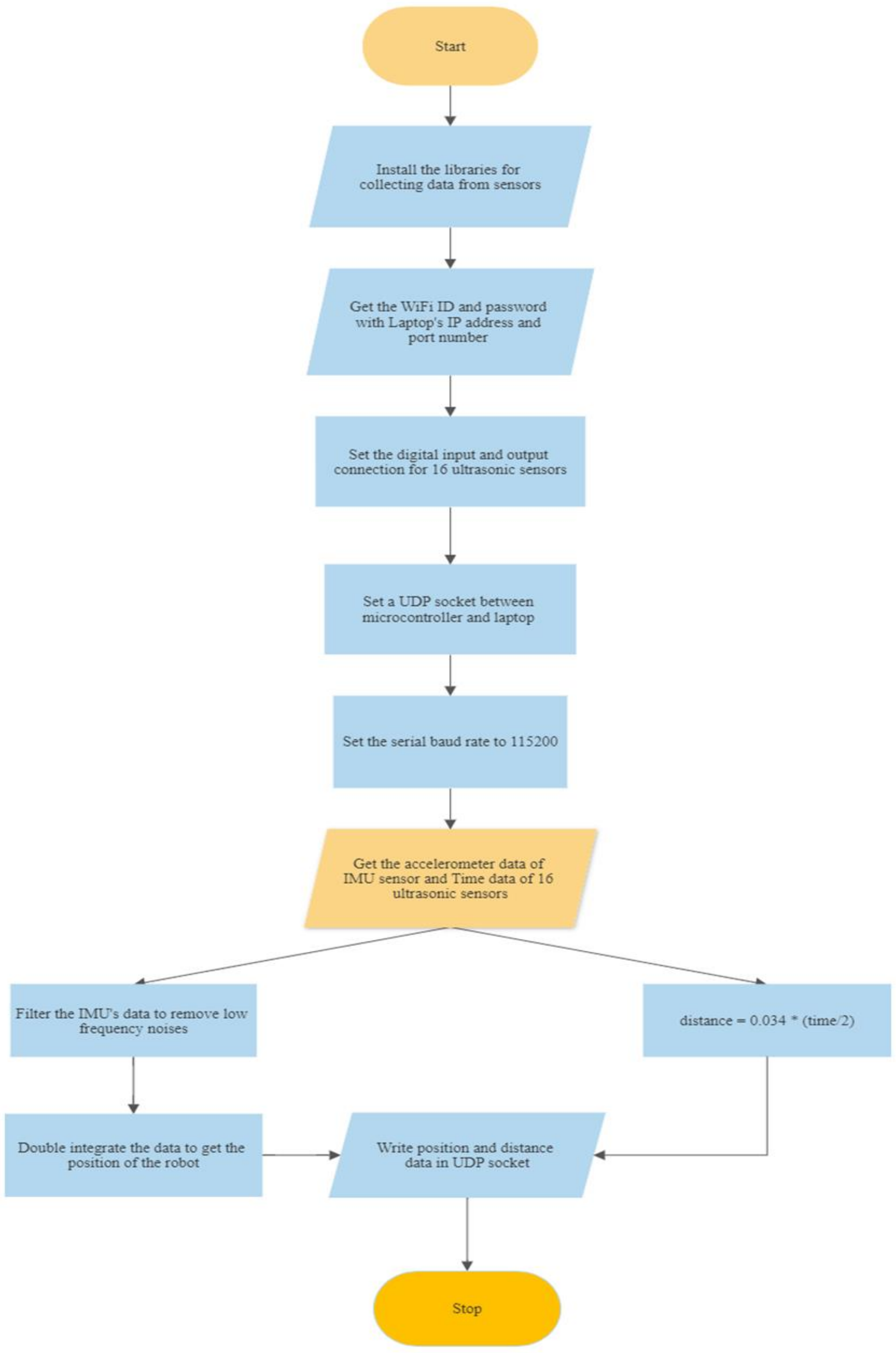

Start
Install the libraries for collecting data from sensors
Get the WiFi ID and password with Laptop's IP address and port number
Set the digital input and output connection for 16 ultrasonic sensors
Set a UDP socket between microcontroller and laptop
Set the serial baud rate to 115200
Get the accelerometer data of IMU sensor and Time data of 16 ultrasonic sensors
Filter the IMU's data to remove low frequency noises
distance = 0.034 * (time/2)
Double integrate the data to get the position of the robot
Write position and distance data in UDP socket
Stop

Figure 3.5: Flowchart for microcontroller code

3.7 FLOWCHART FOR PYTHON CODE

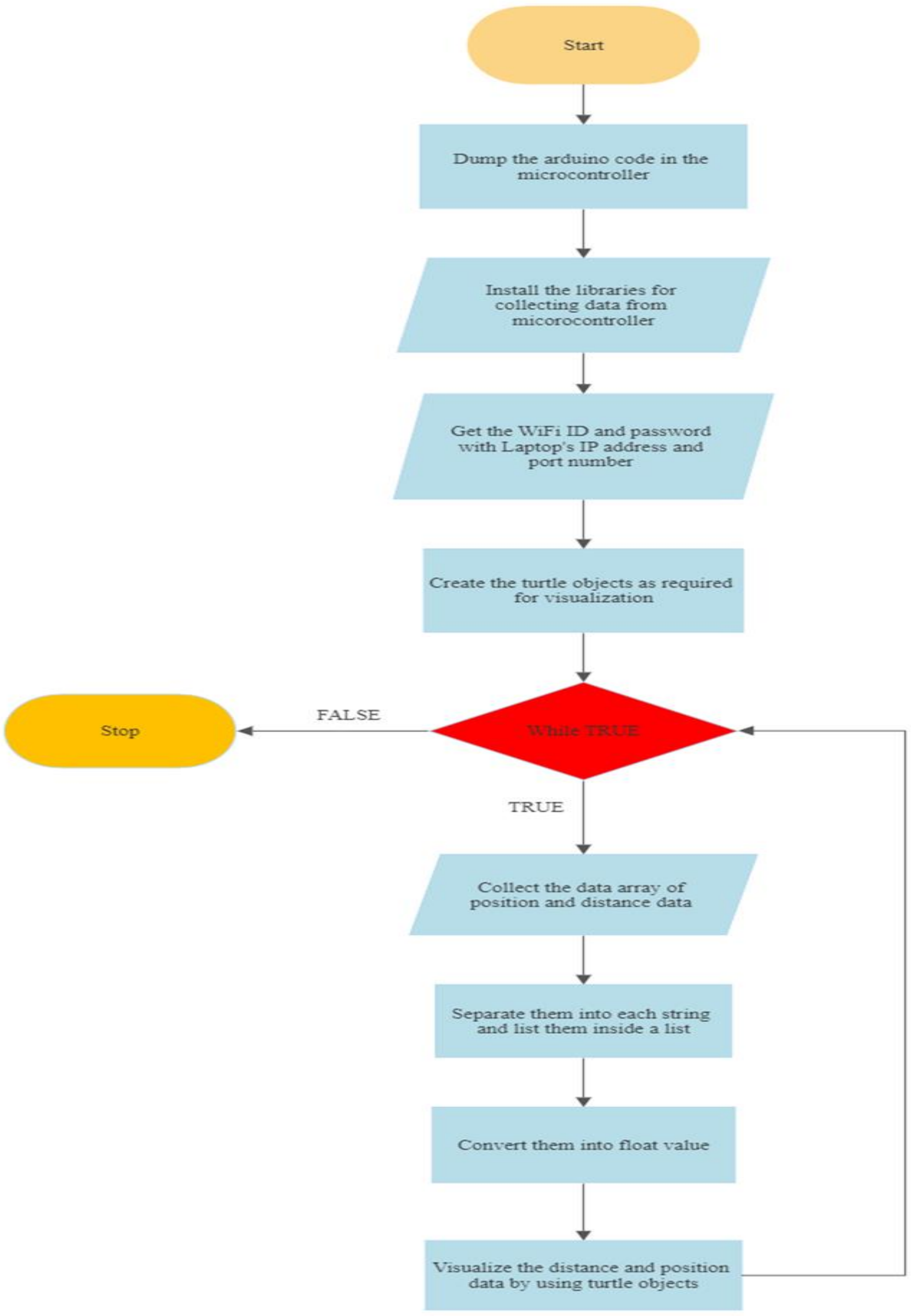

Start
Dump the arduino code in the microcontroller
Install the libraries for collecting data from micorocontroller
Get the WiFi ID and password with Laptop's IP address and port number
Create the turtle objects as required for visualization
While TRUE
FALSE
Stop
TRUE
Collect the data array of position and distance data
Separate them into each string and list them inside a list
Convert them into float value
Visualize the distance and position data by using turtle objects

Figure 3.6: Flowchart for Python code

3.8 COMPONENTS

- Microcontroller
- Ultrasonic sensor
- DC motor
- IMU sensor
- Mecanum Wheel with coupler
- L293D shield
- Chassis
- Aluminium ring
- Stainless Steel rods
- Bread Board
- Battery- 9V, 12V
- Bluetooth Module -HC05
- Clamp
- Nut and Bolt
- Jumper Wire

3.8.1 Microcontroller

The main microcontroller for the robot is Arduino Mega plus WIFI R3 . It is responsible for controlling the motors, sensors, and other components. It also has a built-in WIFI module, which can be used to control the robot remotely. It has 54 digital input/output pins, 16 analog input pins, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, an ICSP header, and a reset button. The image of the components used is shown in figure 3.5.

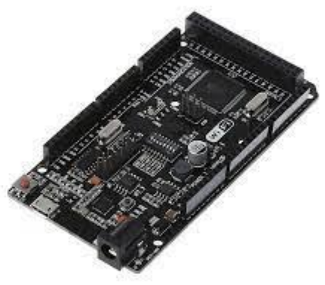


Figure 3.7: Arduino Mega R3

3.8.2 Ultrasonic sensor

These sensors are used to detect obstacles in the robot's path. Sixteen sensors are arranged in a ring around the robot, so that they can detect obstacles in all directions. This sensor’s range of detection is from between 2 centimetres to 4 meters and it’s measuring angle is $30^{o.}$ The model of ultrasonic sensor used is shown in the figure 3.6.

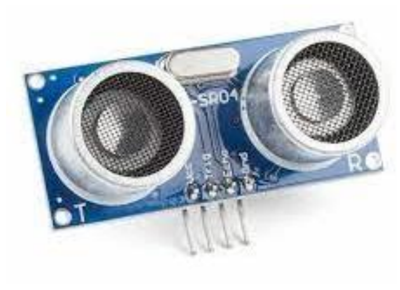


Figure 3.8: Ultrasonic Sensor (HC-SR04)

3.8.3 DC Motor

These motors are used to drive the robot's wheels. They are controlled by the Arduino Mega. The voltage supply is 12v and the armature shaft diameter is 6mm. For four wheels, 4 motors are used. Below is the image of DC motor in figure 3.7.

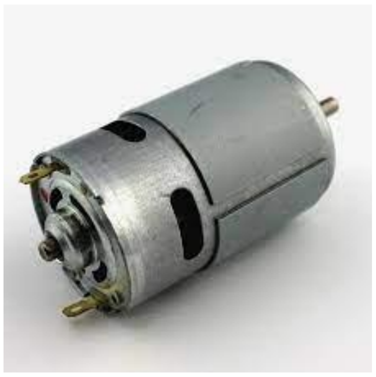


Figure 3.9: DC Motor

3.8.4 IMU Sensor

Inertial measurement unit (IMU) is used to measure the robot's orientation. We are using MPU6050 module for using acceleration and gyro data. It is used in conjunction with the ultrasonic sensors to help the robot navigate its environment. Below is the image of IMU sensor in figure 3.8.

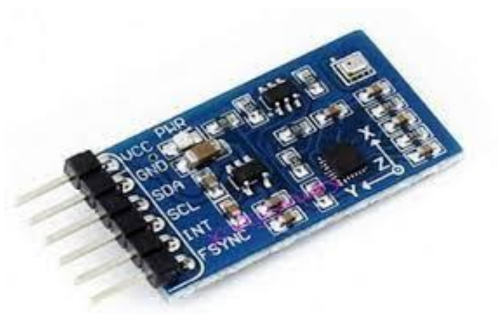


Figure 3.10: IMU Sensor

3.8.5 Mecanum Wheels with coupler

These wheels are used to allow the robot to move in any direction. They are controlled by the Arduino Mega. Their outer diameter is 80mm. A 6.7 mm coupler is used. Below is the image of mecanum wheel in figure 3.9.

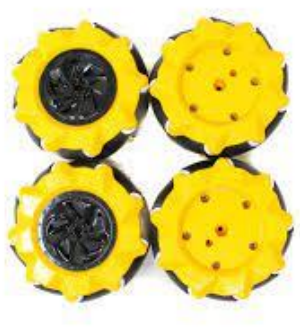


Figure 3.11: Mecanum wheel

3.8.6 L293D Shield

This shield is used to connect the DC motors to the Arduino Mega. It also provides power to the motors. It is shown in figure 3.10.

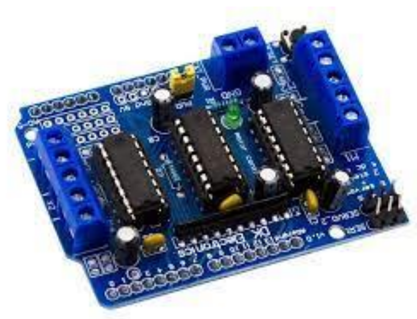


Figure 3.12: L293D Shield

3.8.7 Chassis

This is the frame of the robot as shown in figure 3.11. It is made of wood and provides a platform for the other components to be mounted on.

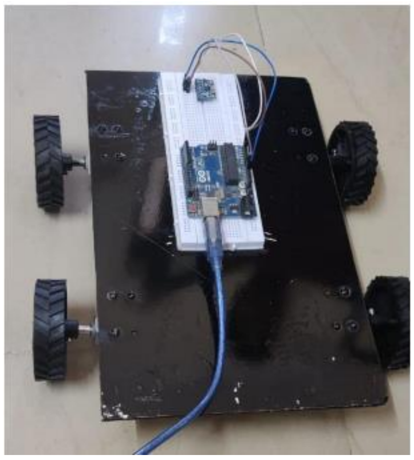


Figure 3.13: Chassis

3.8.8 Stainless Steel rods

2 SS rods to hold the ring of ultrasonic sensors: These rods are used to hold the ring of ultrasonic sensors in place. They are attached to the chassis using a clamp. It is shown in figure 3.12.

Figure 3.14: Stainless steel

3.8.9 Breadboard

This is used to connect the different components of the robot together. A breadboard is a reusable prototyping platform for electronic circuits. It consists of a grid of holes that are connected in a specific pattern. The breadboard can be used to connect electronic components together without soldering. This makes it a convenient way to prototype circuits and to test new ideas. In this project, the breadboard will be used to connect the ultrasonic sensors to the Arduino Mega, the DC motors to the L293D shield, and the other components to the breadboard. The breadboard will also be used to provide power to the components. It is shown in the figure below in figure 3.13.

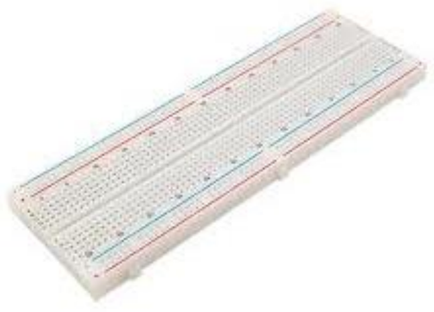


Figure 3.15: Breadboard

3.8.10 Batteries

12V Battery provides power to the Arduino Mega, IMU sensor, and ultrasonic sensors. It is shown in figure 3.14.

Figure 3.16: 12V Battery

9V Battery provides power to Arduino UNO. It is shown in figure 3.15.

Figure 3.17: 9v Battery

3.8.11 Battery to Arduino connector

This is used to connect the 12V battery to the Arduino Mega. The battery to Arduino connector is a simple device used to connect a battery to an Arduino board. It typically consists of a black lead connected to one of the Arduino's ground pins, and a lead from the battery's positive terminal connected to either the Arduino's Vin pin or a DC socket on the board. The connector is often custom-made to deliver power from a specific type of battery, such as a 9V battery, to the Arduino. It is important to be careful when using the VIN pin to avoid damaging the board, as it does not have reverse polarity protection.

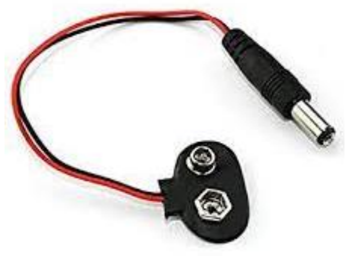


Figure 3.18: Arduino to battery connector

3.8.12 Bluetooth Module HC05

This module is used to connect the robot to a computer or smartphone. It allows the robot to be controlled remotely. The Bluetooth module HC05 as shown in figure 3.17 is a wireless communication module that can be used to connect two devices together. It is a popular choice for Arduino projects because it is easy to use and relatively inexpensive. In this project, the Bluetooth module HC05 will be used to connect the robot to a computer or smartphone. This will allow the robot to be controlled remotely.

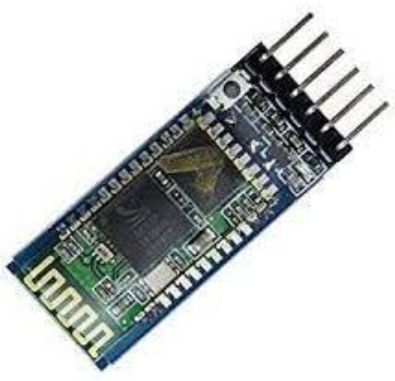


Figure 3.19: Bluetooth module (HC05)

3.8.13 Aluminium Ring

The main body of the robot is made of aluminium ring. It is a plate of aluminium bent into a sixteen-sided polygon using a notching machine and a folding machine. The result of it is shown in figure 3.19.

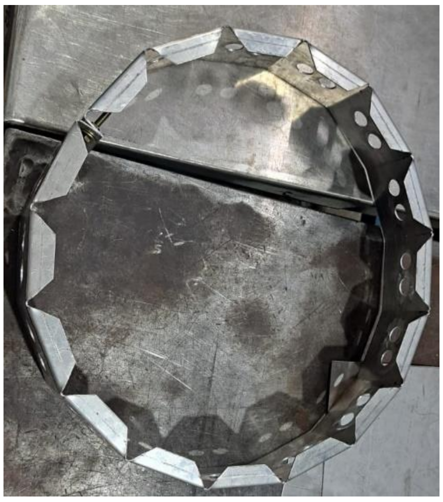


Figure 3.20: Aluminium Ring

3.8.14 Jumper wire

Jumper wires are short, insulated wires with two male connectors on each end and they are shown below in figure 3.20. They are used to connect different components on a breadboard or other circuit board. In this project, jumper wires will be used to connect the ultrasonic sensors to the Arduino Mega, the DC motors to the L293D shield, and to the other components .

Figure 3.21: Jumper Wires

3.9 BILL OF MATERIALS

S NO.	COMPONENT	QUANTITY	COST(Rs.)
1	Ultrasonic Sensor	16	800
2	IMU Sensor	1	500
3	Arduino mega Wi-Fi R3	1	1000
4	DC Motor	4	600
5	Aluminium	1	700
6	Mecanum Wheels	4	700
7	Stainless Steel rod	2	100
8	L293D shield	1	300
9	Bread Board	1	70
10	Jumper Wires	Multiple	80
11	Battery- 9V	1	190
12	Battery-12V	1	700
13	Araldite	1	65
14	Bluetooth Module	1	200
15	Battery to Arduino connector	1	30
Total cost			**6035**

Table 3.1: Bill of Materials

CHAPTER 4

4.1 SELECTION OF THE NUMBER OF SENSORS

A simulation was done based on the range and uncovered region of the sensors around the ring. By using 8 sensors, (figure 4.1) there was an observation of 45 degrees of uncovered region. When an object is present in that region, the system could not detect them. Hence, an observation was done using 10 sensors (Figure 4.2). Though the degree of uncovered region has drastically reduced, it was still not efficient. Hence, on using 16 sensors (Figure 4.3), it was found that the ultrasonic waves converge at 5.3 metres. Hence 16 sensors were chosen as the ideal number for this project.

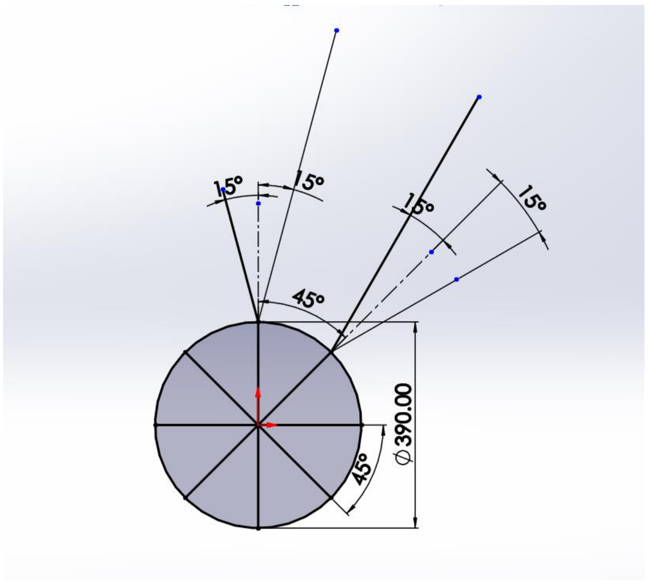


Figure 4.1: Sensor ring using 8 sensors.

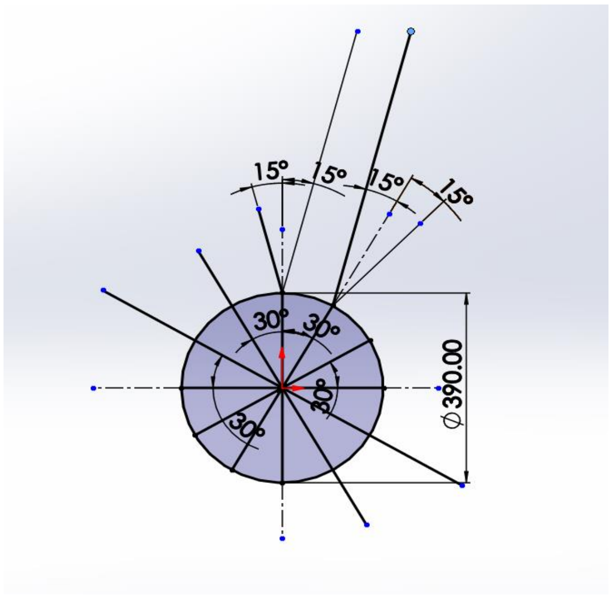


Figure 4.2: Sensor ring using 10 sensors.

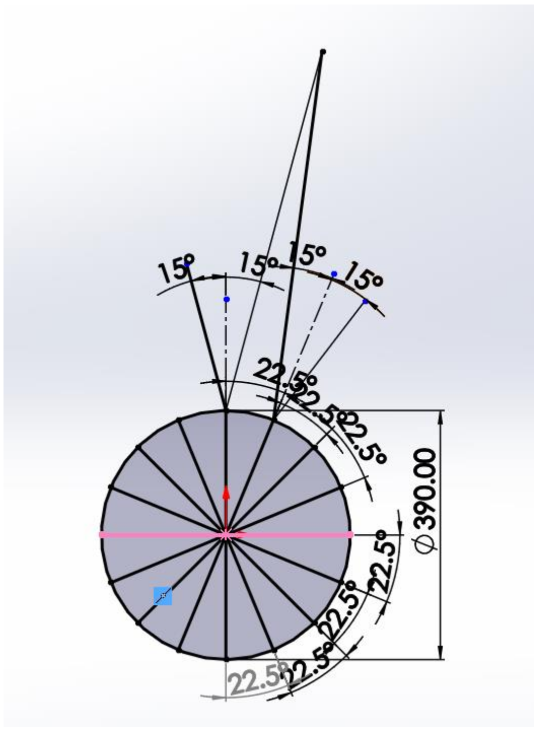


Figure 4.3: Sensor ring using 16 sensors.

4.2 DESIGN WORK

After brain storming, it is decided to position the ultrasonic sensor in the form of ring, so that 360 degrees view is attained. All the sensors are mounted on a 16-sided polygon made of aluminium. The top view, side view, bottom view and isometric view of the chassis is shown below in figure 4.4, 4.5, 4.6 and 4.7 respectively.

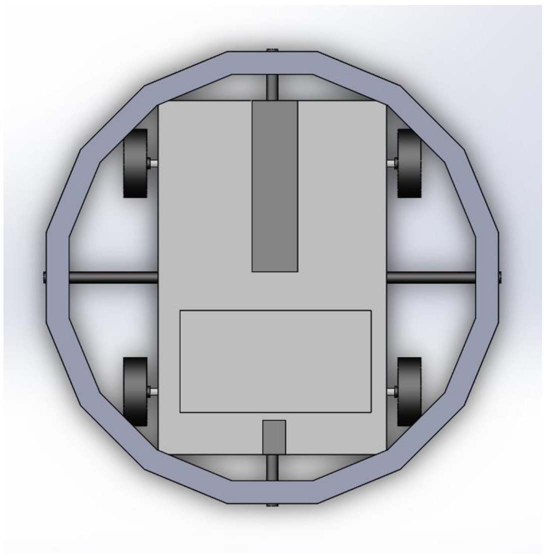


Figure 4.4: Top View of the design

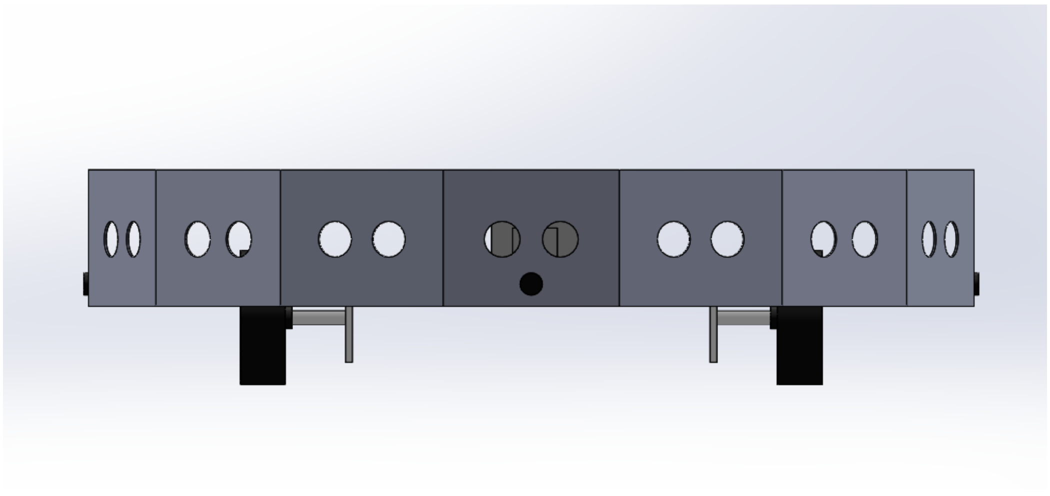


Figure 4.5: Side View of the aluminium ring

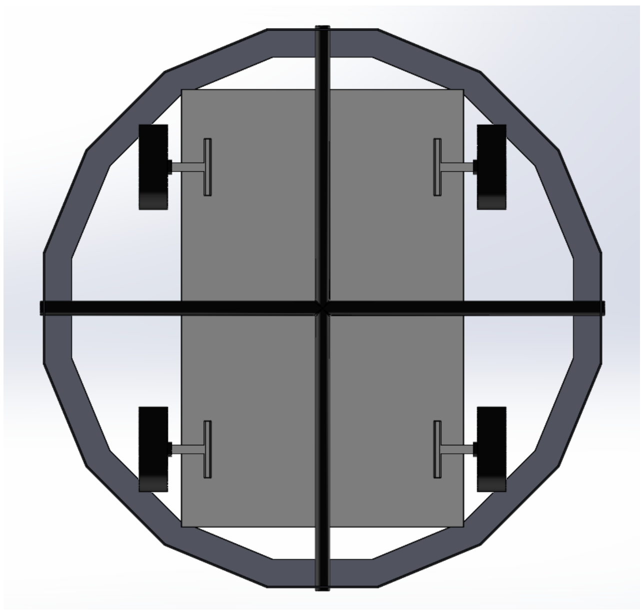


Figure 4.6: Bottom view of the ring

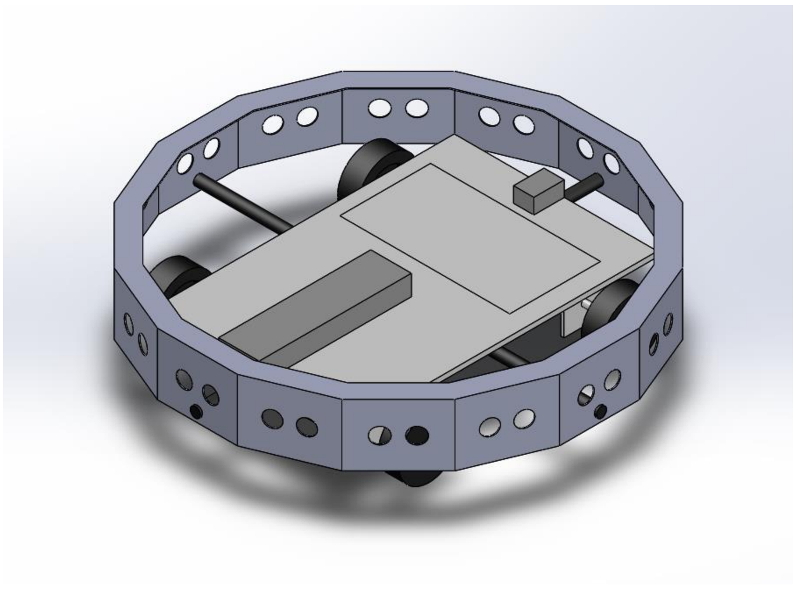


Figure 4.7: Isometric View of the ring

4.3 SOFTWARES USED

PyCharm: PyCharm is an integrated development environment (IDE) used for programming in Python. It is used to visualize the data of the sensors with the help of turtle GUI tool.

Arduino IDE: The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

Solidworks: SolidWorks is a solid modelling computer-aided design and computer-aided engineering application published by Dassault Systems. Solid works is used to make model of the project and do design work and dimension calculations.

4.4 LIBRARY INSTALLATION

4.4.1 I2C device library

The I2C Device Library (i2cdevlib) is a collection of mostly uniform and well-documented classes to provide simple and intuitive interfaces to I2C devices. Each device is built to make use of the generic "I2Cdev" class, which abstracts the I2C bit- and byte-level communication away from each specific device class, making it easy to keep the device class clean while providing a simple way to modify just one class to port the I2C communication code onto different platforms (Arduino, PIC, MSP430, Jennic, simple bit-banging, etc.). Device classes are designed to provide complete coverage of all functionalities described by each device's documentation, plus any generic convenience functions that are helpful.

There are examples in many of the classes that demonstrate basic usage patterns. The I2Cdev class is built to be used statically, reducing the memory requirement if you have multiple I2C devices in your project. Only one instance of the I2Cdev class is required. Recent additions as of late 2021 have also made it possible to pass in non-default Wire objects (in the Arduino environment) to allow using multiple I2C transceivers at the same time, specifically because of the number of people who wanted to use up to four MPU-6050 IMUs without I2C mux ICs involved.

4.4.2 MPU6050 library

MPU-6050 6-axis accelerometer/gyroscope Arduino Library adapted for Arduino Library Manager by Electronic Cats, Feb 2019. The MPU6050 combines a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die together with an onboard Digital Motion Processor (DMP) which processes complex 6-axis Motion Fusion algorithms.

4.4.3 NewPing library

NewPing allows interfacing with ultrasonic sensors simple, fast & powerful. Initially, I was not happy with how poorly ultrasonic sensors performed. I soon realized the problem was not the sensor, it was the available ping and ultrasonic libraries causing the problem. The NewPing library totally fixes these problems, adds many new features, and breathes new life into these very affordable distance sensors.

4.4.4 WiFiEsp library

With an ESP8266 board, WiFiEsp library allows an Arduino board to connect to the internet. It can serve as either a server accepting incoming connections or a client making outgoing ones. The WiFiEsp library is very similar to the Arduino WiFi and Ethernet libraries, and many of the function calls are the same. Supports ESP SDK version 1.1.1 and above (AT version 0.25 and above). This library features are:

- APIs compatible with standard Arduino WiFi library.
- Use AT commands of standard ESP firmware (no need to flash a custom firmware).
- Support hardware and software serial ports.
- Configurable tracing level.

4.4.5 Turtle graphics library

Turtle is a pre-installed Python library that enables users to create pictures and shapes by providing them with a virtual canvas. The onscreen pen that you use for drawing is called the turtle and this is what gives the library its name. In short, the Python turtle library helps new programmers get a feel for what programming with Python is like in a fun and interactive way.

Turtle is mainly used to introduce children to the world of computers. It is a straightforward yet versatile way to understand the concepts of Python. This makes it a great avenue for kids to take their first steps in Python programming. The Python turtle library is not restricted to little ones alone! It is also proved extremely useful for adults who are trying their hands at Python, which makes it great for Python beginners.

4.5 DRIVER INSTALLATION AND CODE DUMPING

For visualizing the data, dump the code for collecting data array from the IMU and 16 ultrasonic sensors to the microcontroller. The data array is then collected from microcontroller to laptop wirelessly through UDP transport layer protocol for per second time. To promote UDP protocol, the WiFi module should be paired with the microcontroller. So, a driver should be installed between WiFi Module and Microcontroller to transfer the data array. The video link for installation of driver is given in the references called "Setup ESP8266 on Mega with build-in WiFi | macOS". After the driver installation, the code given in appendix is dumped into the microcontroller through serial port. Then the port is disconnected and a 5 V power supply is given to the microcontroller.

4.6 VISUALIZING THE IMU'S DATA

As this project uses IMU sensor to identify the robot's position, it is necessary to filter the data coming from the sensor, eliminating the low frequency noises. To filter the data, Adafruit_MPU6050, Adafruit_sensor and "Wire.h" library is used. the "Adafruit_MPU6050" library implements the hardware functions of the MPU6050, while the "Adafruit_Sensor" library implements the unified sensor abstraction layer. "Wire.h," which allows us to communicate with I2C devices, is also included. The Adafruit MPU6050 library makes use of the Adafruit Unified Sensor Driver and Adafruit Bus IO Library internally. The "setFilterBandwidth()" function, which is used in this project, sets the bandwidth of the Digital Low-Pass Filter. The bandwidth selection allows you to alter the low-pass filter's cut-off frequency, allowing you to smooth out the signal by removing high-frequency noise. This function accepts the following values:

- MPU6050_BAND_260_HZ – for 260 Hz bandwidth (According to the documentation, this disables the filter)
- MPU6050_BAND_184_HZ – for 184 Hz bandwidth
- MPU6050_BAND_94_HZ – for 94 Hz bandwidth
- MPU6050_BAND_44_HZ – for 44 Hz bandwidth
- MPU6050_BAND_21_HZ – for 21 Hz bandwidth
- MPU6050_BAND_10_HZ – for 10 Hz bandwidth
- MPU6050_BAND_5_HZ – for 5 Hz bandwidth

By using turtle GUI tool and the filtered data, we can track the position of robot easily. For testing the sensor, we made the robot to move 2 meters in y axis direction from initial point 10 times. The result is that the sensor shows the error of up to ± 0.24 meters which is 12% error. It's the best rectification the sensor can do. The results are shown in figure 4.8 and 4.9.

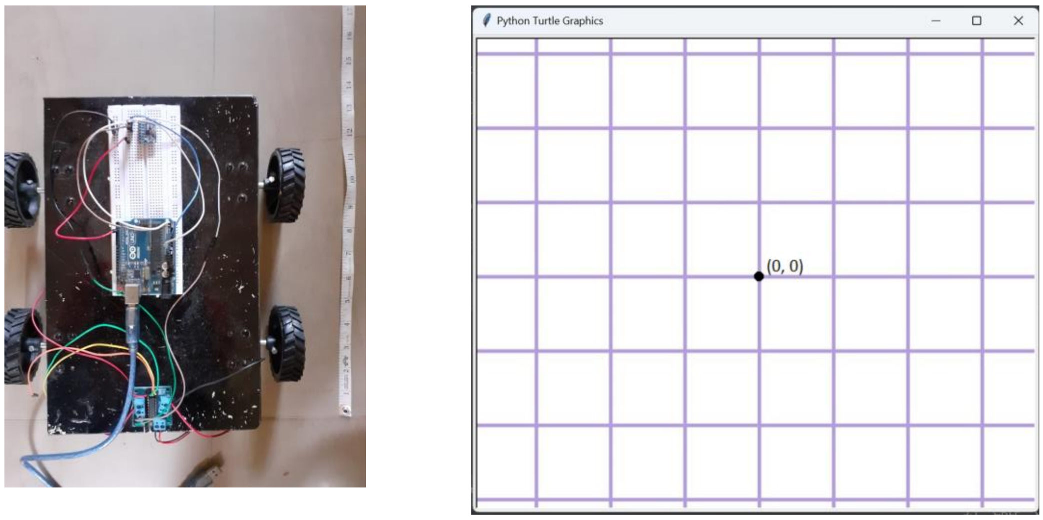


Figure 4.8: The robot at Initial position

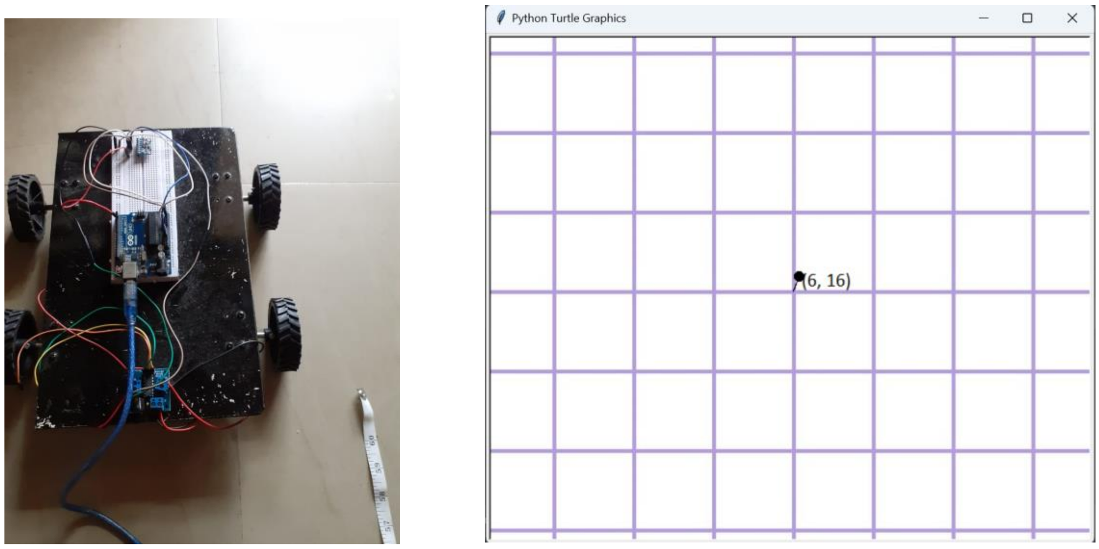
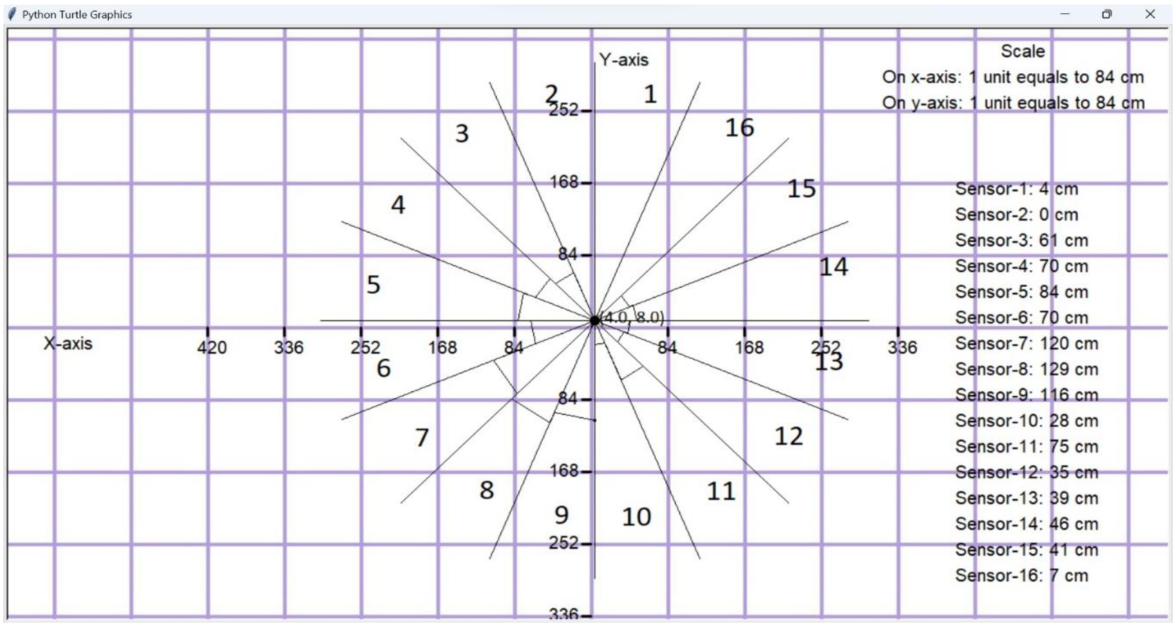


Figure 4.9: The robot after moving to 15.5 decimeters (61 inches)

4.7 DATA VISUALISATION

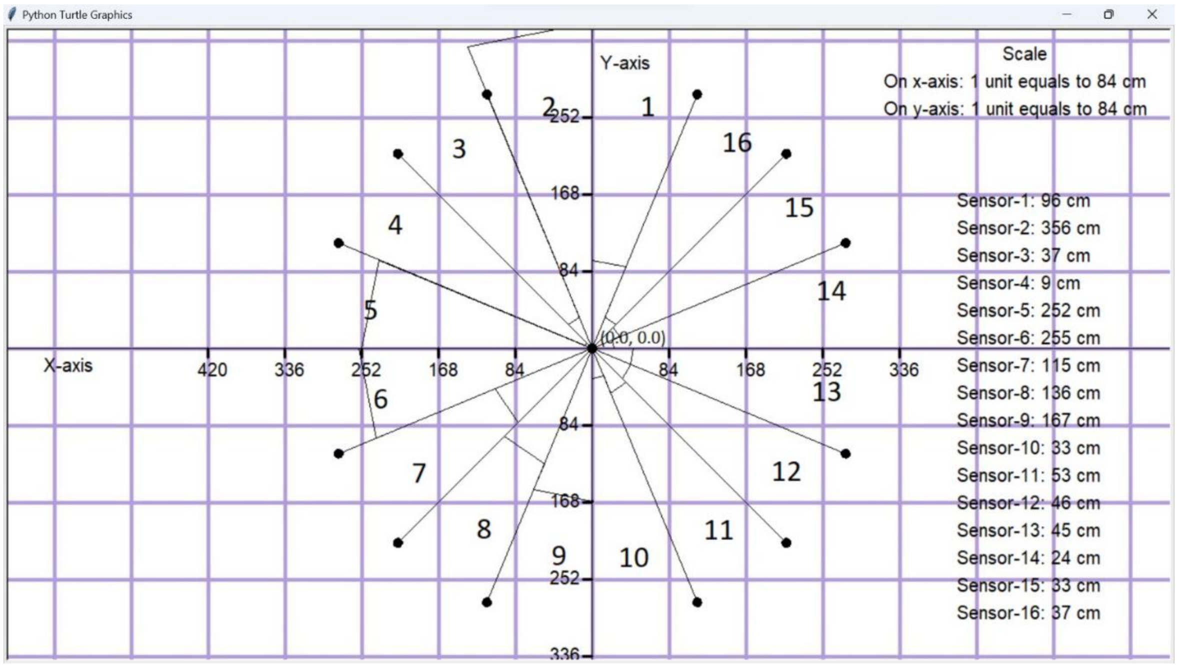
At first, the sensors are mounted and connected according to the figure 3.4. Then the python code given in the appendix is ran in PyCharm software. The data is collected from the sensors, both ultrasonic sensor and IMU sensor. Those data are visualised using Turtle GUI tool. Turtle GUI is a useful tool for visualizing the data from 16 ultrasonic sensors and IMU sensors. Also, it can be used to display the distance to obstacles and the robot's position in real time. This information can be used to improve the performance of mobile robot navigation algorithms. The visualisation is shown in figure 4.10 and figure 4.11. For example, if the distance to an obstacle is displayed on the Turtle GUI, the robot can be programmed to avoid the obstacle. If the robot's position is displayed on the Turtle GUI, the robot can be programmed to keep them heading in a desired direction. Turtle GUI can also be used to debug mobile robot navigation algorithms. As shown in figure 4.12, the robot position is also depicted when it is motion.



```
Scale
On x-axis: 1 unit equals to 84 cm
On y-axis: 1 unit equals to 84 cm

Sensor-1: 4 cm
Sensor-2: 0 cm
Sensor-3: 61 cm
Sensor-4: 70 cm
Sensor-5: 84 cm
Sensor-6: 70 cm
Sensor-7: 120 cm
Sensor-8: 129 cm
Sensor-9: 116 cm
Sensor-10: 28 cm
Sensor-11: 75 cm
Sensor-12: 35 cm
Sensor-13: 39 cm
Sensor-14: 46 cm
Sensor-15: 41 cm
Sensor-16: 7 cm
```

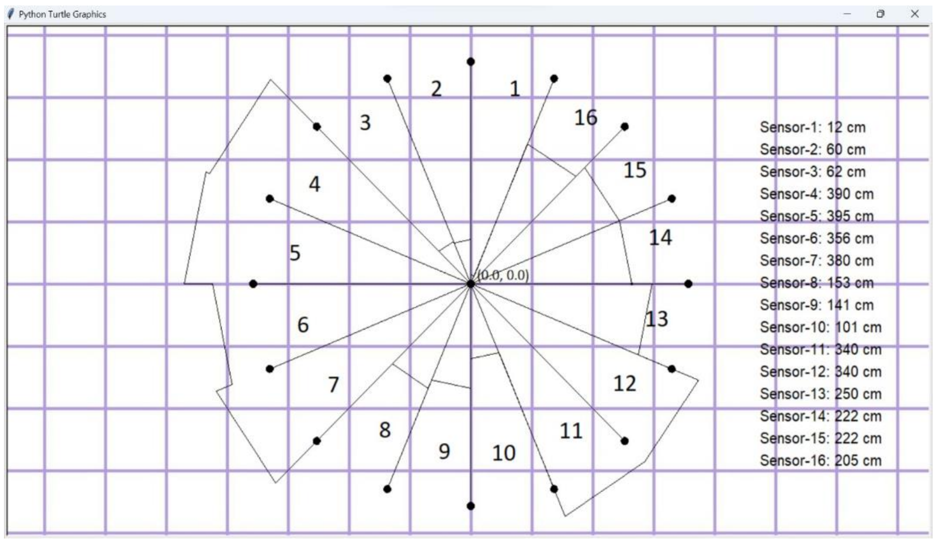
Figure 4.10: Visualization of obstacles when in initial position



```
Scale
On x-axis: 1 unit equals to 84 cm
On y-axis: 1 unit equals to 84 cm

Sensor-1: 96 cm
Sensor-2: 356 cm
Sensor-3: 37 cm
Sensor-4: 9 cm
Sensor-5: 252 cm
Sensor-6: 255 cm
Sensor-7: 115 cm
Sensor-8: 136 cm
Sensor-9: 167 cm
Sensor-10: 33 cm
Sensor-11: 53 cm
Sensor-12: 46 cm
Sensor-13: 45 cm
Sensor-14: 24 cm
Sensor-15: 33 cm
Sensor-16: 37 cm
```

Figure 4.11: Visualization of obstacles after the robot was moved.



```
Sensor-1: 12 cm
Sensor-2: 60 cm
Sensor-3: 62 cm
Sensor-4: 390 cm
Sensor-5: 395 cm
Sensor-6: 356 cm
Sensor-7: 380 cm
Sensor-8: 153 cm
Sensor-9: 141 cm
Sensor-10: 101 cm
Sensor-11: 340 cm
Sensor-12: 340 cm
Sensor-13: 250 cm
Sensor-14: 222 cm
Sensor-15: 222 cm
Sensor-16: 205 cm
```

Figure 4.12: Obstacles visualised between the range of the sensor.

The range of ultrasonic sensor is between 4cm to 4 m. In figure 4.12, the obstacles have been visualised, the sensor 1, 2 and 3, depicts the range between 4 cm to 1 m. The sensors 8, 9 and 10 depicts the sensors' ability to visualise the range between 1m to 2 m. The sensors 13, 14, 15 and 16 captures the obstacles between the range of 2m to 3m. Similarly, the sensor 4, 5, 6, 7, 11 and 12 depicts the sensors' ability to detect the obstacles between the range of 3 to 4m.

The observations here:

- The black line between the quadrants depicts the distance between the sensor and the obstacle.
- The unit of the grid is 84cm on either direction as shown in the X any Y axis.
- The numbering between quadrants depicts the sensor numbering.
- The (X, Y) co-ordinates specify the position of the robot. In figure 4.10, the robot is at (0,0) and in figure 4.11, it is shown that the robot is at (4,8).

- On the right side of the grid depicts the numerical value of the distance between the ultrasonic sensor and obstacle.

4.8 MOTION CONTROL

The robot has 4 DC motors, and those motors are controlled by L293D motor shield. The DC motors are joined with Mecanum wheels. The microcontroller like Arduino UNO is merged with the motor shield by connecting them with pins below the motor shield. A HC05 Bluetooth module is connected with the motor shield. The connections and the procedures are shown in the video lecture given in the reference called "How to make Bluetooth controlled car with Mecanum wheel | Indian LifeHacker." The code for the motion is dumped in the microcontroller and the 5 V power supply is given to it. A 12 V battery power supply is given to the driver. Please ensure to remove the power jumper on the motor shield before giving 12 V power supply to avoid short circuiting. Then the Bluetooth module is connected with remote control wirelessly. Finally, the motion of the robot is achieved by using the remote control.

4.9 RESULT

It is a successful navigation system that can be used to navigate a mobile robot in an indoor environment. It uses a combination of ultrasonic sensors and IMU sensors to determine the robot's position and orientation. Ultrasonic sensors are used to detect obstacles, and the IMU sensors are used to track the robot's movement.

The system was tested in a variety of indoor environments, and it was able to successfully navigate around obstacles and avoid collisions. The system was also able to maintain its position and orientation in the presence of noise and disturbances. The system is a valuable tool for mobile robots that need to navigate in indoor environments. The system is easy to use and can be easily implemented on a variety of mobile robots. The system is also relatively inexpensive, making it a cost-effective solution for many applications. The final assembly is shown in figure 4,9 and 4.10. The obstacles position and their relative distance from the sensor is also graphically represented using Turtle GUI tool. The objects between 4cm and 4m from the obstacles can be graphically represented as shown above.

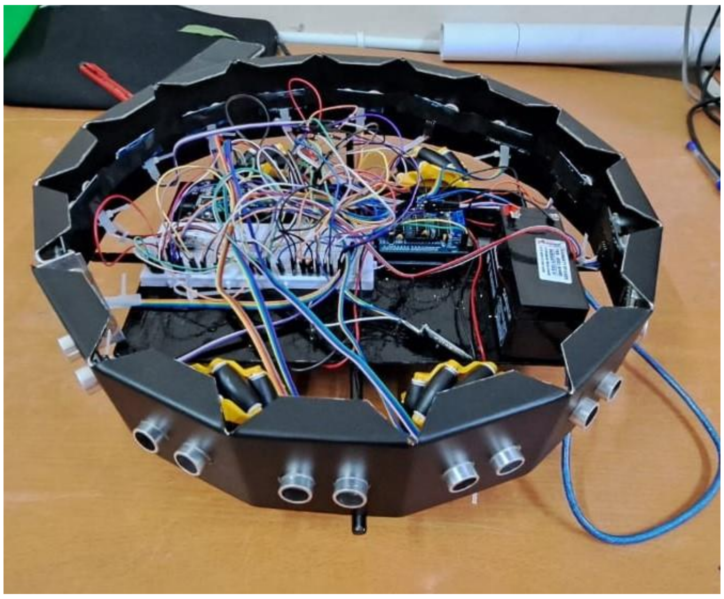


Figure 4.14: The final assembly of the project

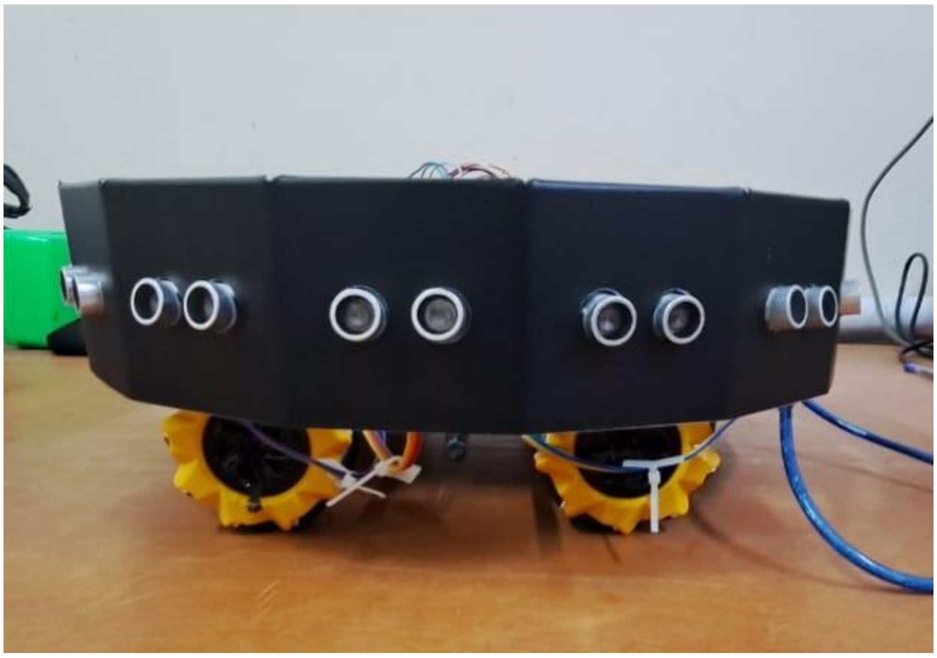


Figure 4.15: The side panel

CHAPTER 5

5.1 CONCLUSION

In the conclusion, it is stated that the proposed method for mobile robot navigation using ultrasonic sensors and IMU sensors was able to accurately estimate the robot's position and orientation in both indoor and outdoor environments. It is also stated that the method was able to navigate the robot through a maze without colliding with any obstacles. The proposed method uses a combination of ultrasonic sensors and IMU sensors to estimate the robot's position and orientation. The ultrasonic sensors are used to measure the distance to objects in the robot's path, and the IMU sensors are used to measure the robot's orientation. The two sensor measurements are fused using a Kalman filter to produce a more accurate estimate of the robot's position and orientation. The proposed method was evaluated in simulation and on a real-world robot.

The simulation was used to evaluate the performance of the method in a variety of environments, including indoor and outdoor environments. The real-world robot was used to evaluate the performance of the method in a real-world environment. The results of the simulation and the real-world experiment showed that the proposed method was able to accurately estimate the robot's position and orientation in both indoor and outdoor environments. The proposed method is a promising approach for mobile robot navigation. The method is simple to implement and can be used with a variety of sensors. The method is also robust to noise and can be used in a variety of environments. The proposed method can be improved in several ways. First, the method can be extended to use more sensors, such as cameras. Second, the method can be improved to handle more complex environments, such as cluttered environments. Third, the method can be improved to be more efficient.

5.2 BENEFITS

- Ultrasonic sensors are low-cost and easy to use.
- IMU sensors are accurate and can track the robot's movement in real time.
- The Kalman filter can fuse the data from the two sensors to create a more accurate estimate of the robot's position.
- **Improved accuracy and reliability**. By combining the data from two different sensors, you can improve the accuracy and reliability of your robot's navigation. The ultrasonic sensor can be used to measure the distance to obstacles, while the IMU can be used to track the robot's orientation. This combination of sensors can help to prevent the robot from colliding with obstacles and can also help it to stay on track.
- **Increased autonomy.** By using ultrasonic sensors and IMU sensors, you can make your robot more autonomous. This means that the robot will be able to navigate its environment without human intervention. This can be useful for applications such as warehouse automation and delivery robots.
- **Reduced costs.** By using ultrasonic sensors and IMU sensors, you can reduce the cost of your robot. These sensors are relatively inexpensive, and they can be easily integrated into a robot's design. This can make your robot more affordable and accessible to a wider range of users.

Overall, mobile robot navigation using ultrasonic sensor and IMU sensor has the potential to provide a number of benefits. By improving the accuracy, reliability, autonomy, and cost-effectiveness of mobile robots, this project could have a significant impact on a variety of industries.

5.3 CHALLENGES

- Ultrasonic sensors are not accurate at long distances.
- IMU sensors can be affected by noise and disturbances.
- The Kalman filter can be computationally expensive.
- **Accuracy.** Ultrasonic sensors are not very accurate, and their readings can be affected by factors such as the distance to the obstacle, the angle of the sensor, and the presence of noise. This can lead to the robot colliding with obstacles or getting lost.
- **Obstructions.** Ultrasonic sensors can be obstructed by objects, such as walls, furniture, and people. This can prevent the robot from detecting obstacles and can lead to collisions.
- **Complexity.** Using ultrasonic sensors and IMU sensors can make the robot's navigation system more complex. This can make it more difficult to design, build, and maintain the robot.

5.4 FUTURE WORKS

- You could use multiple sensors, such as ultrasonic sensors, cameras, and LiDAR sensors, to improve the accuracy and reliability of your robot's navigation.
- You could use AI to develop algorithms that can help your robot to navigate more autonomously.
- You could use cloud computing to process the data from your robot's sensors and to develop more accurate and reliable navigation algorithms.
- It can also be used as an automated AGV in an industrial setup.

APPENDIX

MICROCONTROLLER CODE FOR ULTRASONIC AND IMU SENSOR

```
 // Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation

// is used in I2Cdev.h

#include <Adafruit_MPU6050.h>

#include <Adafruit_Sensor.h>

#include "Wire.h"

#include "I2Cdev.h"

#include "MPU6050.h"

#include <Mouse.h>

#include <NewPing.h>

#include <WiFiEsp.h>

#include <WiFiEspUdp.h>




MPU6050 accelgyro;

Adafruit_MPU6050 mpu;

unsigned long prev=0;

int16_t ax, ay, az;

int16_t gx, gy, gz;
```

```
int16_t mx, my, mz;

float gyy=0;

float bx,by,bz;

float cx,cy,cz;

float dx,dy,dz;

float s, lv;

float sy, lvy, sz, lvz;

float ac, chax=0;

float chay=0;

float chaz=0;

float cherror;

int count=0;

float inv=0, invy=0, invz=0;

#define LED_PIN 13

bool blinkState = false;

unsigned long previousMillis = 0;

//Motor A

const int inputPin1  = 10;    // Pin 15 of L293D IC

const int inputPin2  = 11;    // Pin 10 of L293D IC

//Motor B

const int inputPin3  = 9;   // Pin  7 of L293D IC

const int inputPin4  = 8;   // Pin  2 of L293D IC
```

```
// Wi-Fi settings

char ssid[] = "Ganesh";

char password[] = "11111111";

IPAddress serverIP(192, 168, 207, 183); // Replace with the IP address of
your server

unsigned int serverPort = 139; // Replace with the port number of your
server



// Ultrasonic sensor settings

#define TRIGGER_PIN1 2

#define ECHO_PIN1 3

#define MAX_DISTANCE1 5000



#define TRIGGER_PIN2 4

#define ECHO_PIN2 5

#define MAX_DISTANCE2 5000



#define TRIGGER_PIN3 6

#define ECHO_PIN3 7

#define MAX_DISTANCE3 5000



#define TRIGGER_PIN4 8

#define ECHO_PIN4 9
```

```
#define MAX_DISTANCE4 5000


#define TRIGGER_PIN5 10

#define ECHO_PIN5 11

#define MAX_DISTANCE5 5000


#define TRIGGER_PIN6 12

#define ECHO_PIN6 13

#define MAX_DISTANCE6 5000


#define TRIGGER_PIN7 36

#define ECHO_PIN7 37

#define MAX_DISTANCE7 5000


#define TRIGGER_PIN8 38

#define ECHO_PIN8 39

#define MAX_DISTANCE8 5000


#define TRIGGER_PIN9 40

#define ECHO_PIN9 41

#define MAX_DISTANCE9 5000

#define TRIGGER_PIN10 22
```

```
#define ECHO_PIN10 23

#define MAX_DISTANCE10 5000


#define TRIGGER_PIN11 24

#define ECHO_PIN11 25

#define MAX_DISTANCE11 5000


#define TRIGGER_PIN12 26

#define ECHO_PIN12 27

#define MAX_DISTANCE12 5000


#define TRIGGER_PIN13 28

#define ECHO_PIN13 29

#define MAX_DISTANCE13 5000


#define TRIGGER_PIN14 30

#define ECHO_PIN14 31

#define MAX_DISTANCE14 5000


#define TRIGGER_PIN15 32

#define ECHO_PIN15 33

#define MAX_DISTANCE15 5000
```

```
#define TRIGGER_PIN16 34

#define ECHO_PIN16 35

#define MAX_DISTANCE16 5000


// Create objects for ultrasonic sensor and UDP client

NewPing sonar1(TRIGGER_PIN1, ECHO_PIN1, MAX_DISTANCE1);

NewPing sonar2(TRIGGER_PIN2, ECHO_PIN2, MAX_DISTANCE2);

NewPing sonar3(TRIGGER_PIN3, ECHO_PIN3, MAX_DISTANCE3);

NewPing sonar4(TRIGGER_PIN4, ECHO_PIN4, MAX_DISTANCE4);

NewPing sonar5(TRIGGER_PIN5, ECHO_PIN5, MAX_DISTANCE5);

NewPing sonar6(TRIGGER_PIN6, ECHO_PIN6, MAX_DISTANCE6);

NewPing sonar7(TRIGGER_PIN7, ECHO_PIN7, MAX_DISTANCE7);

NewPing sonar8(TRIGGER_PIN8, ECHO_PIN8, MAX_DISTANCE8);

NewPing sonar9(TRIGGER_PIN9, ECHO_PIN9, MAX_DISTANCE9);

NewPing sonar10(TRIGGER_PIN10, ECHO_PIN10,
MAX_DISTANCE10);

NewPing sonar11(TRIGGER_PIN11, ECHO_PIN11,
MAX_DISTANCE11);

NewPing sonar12(TRIGGER_PIN12, ECHO_PIN12,
MAX_DISTANCE12);

NewPing sonar13(TRIGGER_PIN13, ECHO_PIN13,
MAX_DISTANCE13);

NewPing sonar14(TRIGGER_PIN14, ECHO_PIN14,
MAX_DISTANCE14);
```

```
NewPing sonar15(TRIGGER_PIN15, ECHO_PIN15,
MAX_DISTANCE15);

NewPing sonar16(TRIGGER_PIN16, ECHO_PIN16,
MAX_DISTANCE16);

WiFiEspUDP udp;


void setup() {

  // initialize serial for debugging

  Serial.begin(115200);


  // initialize device

//Serial.println("Initializing I2C devices...");

accelgyro.initialize();


// verify connection

//Serial.println("Testing device connections...");

//Serial.println(accelgyro.testConnection() ? "MPU6050 connection
successful" : "MPU6050 connection failed");

for(int i=0;i<20;i++)

{

accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);

chax=chax+ax;

chay=chay+ay;

chaz=chaz+az;
}

chax=(float)chax/32768;

chay=(float)chay/32768;

chaz=(float)chaz/32768;

//Serial.println("New Chax");
//Serial.print(chax);

// configure Arduino LED for
pinMode(LED_PIN, OUTPUT);
accelgyro.setXAccelOffset(0);
  accelgyro.setYAccelOffset(3);
  accelgyro.setZAccelOffset(16384);
  // set accelerometer range to +-8G
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);

  // set gyro range to +- 500 deg/s
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  // set filter bandwidth to 21 Hz
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
  pinMode(inputPin1, OUTPUT);
  pinMode(inputPin2, OUTPUT);
  pinMode(inputPin3, OUTPUT);
  pinMode(inputPin4, OUTPUT);

  // initialize serial for ESP module
  Serial3.begin(115200);
  WiFi.init(&Serial3);

  // Connect to Wi-Fi
  if (WiFi.begin(ssid, password) == WL_CONNECTED) {
    Serial.println("Connected to Wi-Fi");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("Failed to connect to Wi-Fi");
    while (1);
  }
  udp.begin(serverPort);
}
```

```
void loop() {


  // read raw accel/gyro measurements from device

  count=count+1;

 accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my,
&mz);


  bx=(float)ax/(1638.4);

  by=(float)ay/(1638.4);

  bz=(float)az/(1638.4);

  bx=bx-chax;

  by=by-chay;

  bz=bz-chaz;


  //Motion in one simple direction, let y=0.

unsigned long currentMillis = millis(); //record time of new reading

float interval = float((float(currentMillis) - float(previousMillis))/1000) ;
//Time of previous reading-Time of Current reading= Interval

if(bx>1||bx<(-1))

{

ac=abs(bx)*interval; //acceleration x time=velocity

//ac=(bx)*interval;

lv=inv; // initial velocity=lv, final velocity=inv
```

```
inv=inv+ac; // vf =vi+1t;

s=s+(lv*interval)+0.5*ac*interval; // Distance= Previous Distance + Vit +
1/2 x a x t^2


}


if(by>1||by<(-1)){

ac=abs(by)*interval;

//ac=(by)*interval;

lvy=invy;

invy=invy+ac;


sy=sy+(lvy*interval)+0.5*ac*interval;

}


if(bz>1||bz<(-1))

{

ac=abs(bz)*interval;


lvz=invz;

invz=invz+ac;


sz=sz+(lvz*interval)+0.5*ac*interval;
```

```
}
  delay(0); // Adjust the delay according to your needs
  // Read distance from the ultrasonic sensor
  unsigned int distance1 = sonar1.ping_cm();
  unsigned int distance2 = sonar2.ping_cm();
  unsigned int distance3 = sonar3.ping_cm();
  unsigned int distance4 = sonar4.ping_cm();
  unsigned int distance5 = sonar5.ping_cm();
  unsigned int distance6 = sonar6.ping_cm();
  unsigned int distance7 = sonar7.ping_cm();
  unsigned int distance8 = sonar8.ping_cm();
  unsigned int distance9 = sonar9.ping_cm();
  unsigned int distance10 = sonar10.ping_cm();
  unsigned int distance11 = sonar11.ping_cm();
  unsigned int distance12 = sonar12.ping_cm();
  unsigned int distance13 = sonar13.ping_cm();
  unsigned int distance14 = sonar14.ping_cm();
  unsigned int distance15 = sonar15.ping_cm();
  unsigned int distance16 = sonar16.ping_cm();
  // Prepare the data to send
  String data1 = String(distance1);
  String data2 = String(distance2);
  String data3 = String(distance3);

  String data4 = String(distance4);

  String data5 = String(distance5);

  String data6 = String(distance6);

  String data7 = String(distance7);

  String data8 = String(distance8);

  String data9 = String(distance9);

  String data10 = String(distance10);

  String data11 = String(distance11);

  String data12 = String(distance12);

  String data13 = String(distance13);

  String data14 = String(distance14);

  String data15 = String(distance15);

  String data16 = String(distance16);

  String data17 = String(s);

  String data18 = String(sy);

  // Send the data via UDP

  udp.beginPacket(serverIP, serverPort);

udp.print(data1+","+data2+","+data3+","+data4+","+data5+","+data6+","+
data7+","+data8+","+data9+","+data10+","+data11+","+data12+","+data13
+","+data14+","+data15+","+data16+","+data17+","+data18);

  udp.endPacket();
  Serial.print("Distance: ");

  Serial.print(distance2);

  Serial.println(" cm");

  previousMillis=millis(); //record current time

// blink LED to indicate activity

blinkState = !blinkState;

digitalWrite(LED_PIN, blinkState);

digitalWrite(inputPin1, HIGH);

    digitalWrite(inputPin2, LOW);

    digitalWrite(inputPin3, HIGH);

    digitalWrite(inputPin4, LOW);

    delay(0);

}
```

PYTHON CODE FOR VISUALIZATION

```
import socket
import turtle
import serial
from turtle import *
import math

# Arduino IP address and port
arduino_ip = "192.168.207.183"  # Replace with the Arduino's IP address
arduino_port = 139  # Replace with the port number used in the Arduino
code

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
# Bind the socket to the Arduino IP and port
sock.bind((arduino_ip, arduino_port))

t1 = turtle.Turtle()

t2 = turtle.Turtle()
t3 = turtle.Turtle()
t4 = turtle.Turtle()
t5 = turtle.Turtle()
t6 = turtle.Turtle()
t7 = turtle.Turtle()
t8 = turtle.Turtle()
t9 = turtle.Turtle()

t10= turtle.Turtle()

t11= turtle.Turtle()
t12= turtle.Turtle()
t13= turtle.Turtle()
t14= turtle.Turtle()
t15= turtle.Turtle()
t16= turtle.Turtle()
t17= turtle.Turtle()
t18= turtle.Turtle()

t = turtle.Turtle()
t.hideturtle()
t.penup()
t.forward(10)
t.left(90)
path = r"D:\Downloads\gridbackground.png"
style = ('calibri', 15, 'normal')
t.write((0,0),font=style)

t2.speed(20)
t3.speed(20)
t4.speed(20)
t5.speed(20)
t6.speed(20)
t7.speed(20)
```

```
t8.speed(20)
t9.speed(20)
t11.speed(20)
t12.speed(20)
t13.speed(20)
t14.speed(20)
t15.speed(20)
t16.speed(20)
t17.speed(20)
t18.speed(20)


style1 = ('calibri', 25, 'normal')
ta = turtle.Turtle()
ta.speed(20)
ta.hideturtle()
ta.penup()
ta.left(77.5)
ta.forward(250)
ta.write('1',font=style1)

tb = turtle.Turtle()
tb.speed(20)
tb.hideturtle()
tb.penup()
tb.left(102.5)
tb.forward(250)
tb.write('2',font=style1)

tc = turtle.Turtle()
tc.speed(20)
tc.hideturtle()
tc.penup()
tc.left(127.5)
tc.forward(250)
tc.write('3',font=style1)

td = turtle.Turtle()
td.speed(20)
td.hideturtle()
td.penup()
td.left(152.5)
td.forward(250)
td.write('4',font=style1)

te = turtle.Turtle()
te.speed(20)
te.hideturtle()
te.penup()
te.left(175)
te.forward(250)
te.write('5',font=style1)

tf = turtle.Turtle()
tf.speed(20)
tf.hideturtle()
tf.penup()
tf.left(197.5)
tf.forward(250)
tf.write('6',font=style1)

tg = turtle.Turtle()
tg.speed(20)
tg.hideturtle()
tg.penup()
tg.left(218.5)
tg.forward(250)
tg.write('7',font=style1)

th = turtle.Turtle()
th.speed(20)
th.hideturtle()
th.penup()
th.left(240)
th.forward(250)
th.write('8',font=style1)

ti = turtle.Turtle()
ti.speed(20)
ti.hideturtle()
ti.penup()
ti.left(260)
ti.forward(250)
ti.write('9',font=style1)

tj = turtle.Turtle()
tj.speed(20)
tj.hideturtle()
tj.penup()
tj.left(277)
tj.forward(250)
tj.write('10',font=style1)

tk = turtle.Turtle()
tk.speed(20)
tk.hideturtle()
tk.penup()
tk.left(299.5)
tk.forward(250)
tk.write('11',font=style1)

tl = turtle.Turtle()
tl.speed(20)
tl.hideturtle()
tl.penup()
tl.left(322)
tl.forward(250)
tl.write('12',font=style1)

tm = turtle.Turtle()
tm.speed(20)
tm.hideturtle()
tm.penup()
tm.left(344.5)
tm.forward(250)
tm.write('13',font=style1)

tn = turtle.Turtle()
tn.speed(20)
tn.hideturtle()
tn.penup()
tn.left(370)
tn.forward(250)
tn.write('14',font=style1)

tp = turtle.Turtle()
tp.speed(20)
tp.hideturtle()
tp.penup()
tp.left(392.5)
tp.forward(250)
tp.write('15',font=style1)

tq = turtle.Turtle()
tq.speed(20)
tq.hideturtle()
tq.penup()
tq.left(415)
tq.forward(250)
tq.write('16',font=style1)


t10.shape("circle")
t10.shapesize(0.5,0.5,1)
screensize(500,500)
bgpic(path)

t1.shape("circle")
t1.shapesize(0.5,0.5,1)
t2.shape("circle")
t2.shapesize(0.5,0.5,1)
t3.shape("circle")
t3.shapesize(0.5,0.5,1)
t4.shape("circle")
t4.shapesize(0.5,0.5,1)
t5.shape("circle")
t5.shapesize(0.5,0.5,1)
t6.shape("circle")
t6.shapesize(0.5,0.5,1)
t7.shape("circle")
t7.shapesize(0.5,0.5,1)
t8.shape("circle")
t8.shapesize(0.5,0.5,1)
t9.shape("circle")
```

```
t9.shapesize(0.5,0.5,1)
t10.shape("circle")
t10.shapesize(0.5,0.5,1)
t11.shape("circle")
t11.shapesize(0.5,0.5,1)
t12.shape("circle")
t12.shapesize(0.5,0.5,1)
t13.shape("circle")
t13.shapesize(0.5,0.5,1)
t14.shape("circle")
t14.shapesize(0.5,0.5,1)
t15.shape("circle")
t15.shapesize(0.5,0.5,1)
t16.shape("circle")
t16.shapesize(0.5,0.5,1)
t17.shape("circle")
t17.shapesize(0.5,0.5,1)
t18.shape("circle")
t18.shapesize(0.5,0.5,1)

t2.right(22.5)
t2.forward(300)
t3.right(45)
t3.forward(300)
t4.right(67.5)
t4.forward(300)
t5.right(90)
t5.forward(300)
t6.right(112.5)
t6.forward(300)
t7.right(135)
t7.forward(300)
t8.right(157.5)
t8.forward(300)
t9.right(180)
t9.forward(300)
t11.right(202.5)
t11.forward(300)
t12.right(225)
t12.forward(300)
t13.right(247.5)
```

```
t13.forward(300)
t14.right(270)
t14.forward(300)
t15.right(292.5)
t15.forward(300)
t16.right(315)
t16.forward(300)
t17.right(337.5)
t17.
forward(300)
t18.right(360)
t18.forward(300)


# Main program loop to receive data
for i in range(10000000):
    data, address = sock.recvfrom(1024)  # Receive data from Arduino
    string = data.decode()
    # distance = int(str)  # Convert data to integer

    str = string.split(",")
    str1 = str[0]
    str2 = str[1]
    str3 = str[2]
    str4 = str[3]
    str5 = str[4]
    str6 = str[5]
    str7 = str[6]
    str8 = str[7]
    str9 = str[8]
    str10 = str[9]
    str11 = str[10]
    str12 = str[11]
    str13 = str[12]
    str14 = str[13]
    str15 = str[14]
    str16 = str[15]
    str17 = str[16]
    str18 = str[17].rstrip()

    num1 = float(str1)
    num2 = float(str2)  # convert the unicode string to an int
    num3 = float(str3)
    num4 = float(str4)
    num5 = float(str5)
    num6 = float(str6)
    num7 = float(str7)
    num8 = float(str8)
    num9 = float(str9)
    num10 = float(str10)
    num11 = float(str11)
    num12 = float(str12)
    num13 = float(str13)
    num14 = float(str14)
    num15 = float(str15)
    num16 = float(str16)
    num17 = float(str17)
    num18 = float(str18)

    print(num1, num2, num3, num4, num5, num6, num7, num8, num9,
num10, num11, num12, num13, num14, num15, num16, num17, num18)

    t10.speed(20)
    t10.goto(num17, num18)
    t.undo()
    t.write((num17, num18), font=style)
    delay(0)

    t1.speed(20)
    t1.goto(num10 * math.cos(math.radians(90)), -(num10 *
math.sin(math.radians(90))))
    t1.goto(num10 * math.cos(math.radians(67.5)), -(num10 *
math.sin(math.radians(67.5))))
    t1.goto(num11 * math.cos(math.radians(67.5)), -(num11 *
math.sin(math.radians(67.5))))
    t1.goto(num11 * math.cos(math.radians(45)), -(num11 *
math.sin(math.radians(45))))
    t1.goto(num12 * math.cos(math.radians(45)), -(num12 *
math.sin(math.radians(45))))
    t1.goto(num12 * math.cos(math.radians(22.5)), -(num12 *
math.sin(math.radians(22.5))))
    t1.goto(num13 * math.cos(math.radians(22.5)), -(num13 *
math.sin(math.radians(22.5))))
    t1.goto(num13 * math.cos(math.radians(0)), 0)
    t1.goto(num14 * math.cos(math.radians(0)), 0)
    t1.goto((num14 * math.cos(math.radians(22.5))), num14 *
math.sin(math.radians(22.5)))
    t1.goto((num15 * math.cos(math.radians(22.5))), num15 *
math.sin(math.radians(22.5)))
    t1.goto((num15 * math.cos(math.radians(45))), num15 *
math.sin(math.radians(45)))
    t1.goto((num16 * math.cos(math.radians(45))), num16 *
math.sin(math.radians(45)))
    t1.goto((num16 * math.cos(math.radians(67.5))), (num16 *
math.sin(math.radians(67.5))))
    t1.goto((num1 * math.cos(math.radians(67.5))), (num1 *
math.sin(math.radians(67.5))))
    t1.goto((num1 * math.cos(math.radians(90))), (num1 *
math.sin(math.radians(90))))
    t1.goto(-(num2 * math.cos(math.radians(90))), (num2 *
math.sin(math.radians(90))))
    t1.goto(-(num2 * math.cos(math.radians(67.5))), (num2 *
math.sin(math.radians(67.5))))
    t1.goto(-(num3 * math.cos(math.radians(67.5))), (num3 *
math.sin(math.radians(67.5))))
    t1.goto(-(num3 * math.cos(math.radians(45))), (num3 *
math.sin(math.radians(45))))
    t1.goto(-(num4 * math.cos(math.radians(45))), (num4 *
math.sin(math.radians(45))))
    t1.goto(-(num4 * math.cos(math.radians(22.5))), (num2 *
math.sin(math.radians(22.5))))
    t1.goto(-(num5 * math.cos(math.radians(22.5))), (num5 *
math.sin(math.radians(22.5))))
    t1.goto(-(num5 * math.cos(math.radians(0))), 0)
    t1.goto(-(num6 * math.cos(math.radians(0))), 0)
    t1.goto(-(num6 * math.cos(math.radians(22.5))), -(num6 *
math.sin(math.radians(22.5))))
    t1.goto(-(num7 * math.cos(math.radians(22.5))), -(num7 *
math.sin(math.radians(22.5))))
    t1.goto(-(num7 * math.cos(math.radians(45))), -(num7 *
math.sin(math.radians(45))))
    t1.goto(-(num8 * math.cos(math.radians(45))), -(num8 *
math.sin(math.radians(45))))
```

```
    t1.goto(-(num8 * math.cos(math.radians(67.5))), -(num8 *
math.sin(math.radians(67.5))))
    t1.goto(-(num9 * math.cos(math.radians(67.5))), -(num9 *
math.sin(math.radians(67.5))))
    t1.goto(-(num9 * math.cos(math.radians(90))), -(num9 *
math.sin(math.radians(90))))

    delay(-1000000000000000000)
    if i % 2 == 0:
        t1.clear()

    # print(str)
    # print(distance, "cm")

# Close the socket
sock.close()
```

REFERENCES

[1] Alok Sanyal. et al., (2018) “Path Planning Approaches for Mobile Robot Navigation in Various Environments: A Review” Vol. 2, No. 1, pp. 230-232.

[2] Andrew Blake. et al., (2011) “Deriving Displacement from a 3 axis Accelerometer” Vol. 25, No. 12, pp. 6-12.

[3] Ava Chikurteva, et al., (2021) “Mobile robot localization and navigation using LiDAR and indoor GPS” Vol. 54, No. 13, pp. 351-356.

[4] Chen, et al., (2023). “Improved Navigation of Mobile Robots Using Ultrasonic Sensors”

[5] Chen. M, et al., (2021). “Adaptive LiDAR-based Obstacle Avoidance for Mobile Robots”

[6] Choi. S, et al., (2020). “LiDAR-based Mobile Robot Navigation in Outdoor Environments using Dynamic Object Detection”

[7] Farahat. A et al., (2020) “Efficient LiDAR-based Path Planning for Mobile Robots in Dynamic Environments”

[8] Giannoccaro, et al., (2022). “Processing of LiDAR and IMU data for target detection and odometry of a mobile robot”

[9] Hang Yan. et al., (2018) “RIDI: Robust IMU Double Integration” Vol. 34, No. 9, pp. 1-6.

[10] Hosseini. S, et al., (2021). “IMU-Based Navigation of Autonomous Mobile Robots using a Convolutional Neural Network”

[11] Jeff Ferguson. (2015) “Calibration of Deterministic IMU Errors” Vol. 1, No.1, pp. 4-99.

[12] Li, et al., (2019) “Sensors and Data in Mobile Robotics for Localisation”

[13] Li. J, et al., (2019). “A Hybrid SLAM Approach for LiDAR-Based Mobile Robot Navigation in Indoor Environments”

[14] MathWorks Help Centre, “Implement Simultaneous Localization And Mapping (SLAM) with Lidar Scans”. Citated at 2015.

[15] Mujawar. A. S, et al., (2019) “A Robust Navigation System for a Mobile Robot using IMU Sensors and Extended Kalman Filter”

[16] Naveen Prabu Palanisamy (2016) “Filtering of IMU data using Kalman Filter” Vol. 3, No. 5, pp. 1-5.

[17] Pandu Sandi Pratama. (2019) “Experimental Comparison of A* and D* Lite Path Planning Algorithms for Differential Drive Automated Guided Vehicle” Vol. 20, No. 1, pp. 122-125.

[18] Video Series: “MATLAB Tech Talk by Brian Douglas”. https://youtube.com/playlist?list=PLn8PRpmsu08rLRGrnFS6TyGr mcA2X7kg . Citated at 2015.

[19] Video Series: “Setup ESP8266 on Mega with build-in wifi | macOS”. https://www.youtube.com/watch?v=E250CVUWNB0. Citated at 2020.

[20] Wang. Y, et al., (2020). “Mobile Robot Navigation using IMU Sensors and Particle Filter Localization”

[21] Yan Peng. Et al., (2015) “The Obstacle Detection and Avoidance Algorithm Based on 2-D Lidar” Vol. 97, No. 15, pp. 1648-165361.

[22] Zhang. Y, et al., (2021). “Multi-Sensor Fusion for Mobile Robot Navigation in Complex Environments”

[23] Zhang. J, et al., (2022). “Real-Time Mobile Robot Navigation using IMU Sensors and Deep Reinforcement Learning”

[24] Zhang. H, et al., (2023). “IMU Sensor-Based Mobile Robot Navigation using a Hybrid Kalman Filter and Neural Network”