MAPPING AND LOCALIZATION OF MOBILE ROBOT USING IMU SENSOR

PR5712 - PROJECT – I REPORT

Submitted by

ARUNKUMAR R	**2019507004**
GANESH B	**2019507016**
HARIHARAN Ve	**2019507019**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

PRODUCTION ENGINEERING

DEPARTMENT OF PRODUCTION TECHNOLOGY

MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY:: CHENNAI 600 044

FEBRUARY 2023

ANNA UNIVERSITY:: CHENNAI 600 044

BONAFIDE CERTIFICATE

Certified that this project report **"MAPPING AND LOCALIZATION OF MOBILE ROBOT USING IMU SENSOR"** is the Bonafide work of "**ARUNKUMAR R (2019507004),** G**ANESH B (2019507016), HARIHARAN Ve (2019507019)"** who carried out the project under my supervision. Certified further that to the best of my knowledge, the work reported herein does not from part of any other thesis or dissertation on the basis of which a degree or award was conferred on earlier

Dr. A. SIDDHARTHAN
Professor and Head
Department of Production Technology,
Madras Institute of Technology,
Anna University, Chennai -600044.

Dr. P. KARTHIKEYAN
Assistant Professor and Supervisor
Department of Production Technology,
Madras Institute of Technology,
Anna University, Chennai -600044.

ACKNOWLEDGMENT

It is a pleasure to place on record our profound sense of gratitude to our supervisors **Dr. P. KARTHIKEYAN**, Assistant Professor, Department of Production Technology for his valuable guidance and which helped us immensely to accomplish this project successfully.

We wish to express our sincere thanks to our project coordinator **Dr. C. NANDAKUMAR,** Assistant Professor (Sl.Gr.), our Panel Member **Dr. G. KUMARESAN,** Teaching Fellow and our Faculty Advisor and Panel Member **Mr. K. MUTHUKUMARAN,** Department of Production Technology, Madras Institute of Technology, for their valuable suggestions and constant encouragement.

We are extremely grateful to **Dr. A. SIDDHARTHAN,** Professor and Head of the Department of Production Technology, for his constant encouragement and suggestions which helped us in the completion of this phase of the project success.

We are highly thankful to **Dr. J. PRAKASH,** Professor and Dean, Madras Institute of Technology, Chennai for providing the necessary facilities to carry out the project.

We are very grateful to all the staff members and non-teaching staff of the Department of Production Technology and my friends and family for their valuable suggestions and their help provided during this project.

ARUNKUMAR R

GANESH B

HARIHARAN Ve

ABSTRACT

A path planning is nothing but the strategy to find a sequence of valid configurations that moves the object from source to destination. It lets an autonomous vehicle or a robot find the shortest and most obstacle-free path from start to target point. The main problem in this research is the determination of localization in a very tough and dynamically changing environment as well as in tight spaces. Over time, the error in localization with LiDAR increases too much, in the presence of many new dynamic objects on the map. So, to find the absolute position of the robot, this project report focuses on the usage of IMU sensor for localizing and mapping the robot by interfacing it with Microcontroller. Primarily, this project focuses on, how to collect data from the IMU sensor in order to find the position of the robot . With the help of external data acquisition software called PUTTY, the robot's position is measured and plotted in Cartesian graph. While its result still needs to get fine-tuned in order to get the accurate real time data. So, in addition to that, this project also tries to find the tilt angle with respect to the ground with the help of IMU sensor. From the result, it is understandable that if the pointer moves towards its left, the robot is tilted rightwards, and when the pointer moves towards its right, the robot is tilted leftwards. With these, we can nullify those tilt angles and make sure of a smooth motion by identifying the robot's position. So, it is done to ensure that robot is stable. Despite all these information, the project initially suggests an RRT* path planning algorithm with the help of Python Programming, which is the most optimum and highly efficient method to rely on for the motion of the robot. This developed position-sensing method has to be cleared and calibrated to obtain the accurate displacement measurements of the robot. The integration of this prototype with the Lidar sensor configuration will enable feedback-based monitoring and controlled motion of the robot (autonomous).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

LiDAR	-	Light Detecting and Ranging
3D	-	Three-Dimensional
RADAR	-	Radio Detection and Ranging
GPS	-	Global Positioning System
INS	-	Inertial Navigational System
UAV	-	Unmanned Aerial Vehicle
IDE	-	Integrated Development Environment
IMU	-	Inertial Measurement Unit
AHRS	-	Attitude Heading Reference System
RRT	-	Rapidly exploring Random Tree
PRM	-	Probabilistic RoadMap
IPS	-	Indoor Positioning System
RFID	-	Radio Frequency Identification

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

This current century is mainly focused on modernization of industries by using modern concepts like industry 4.0, Industrial IoT, automation and development in which the human task are replaced by robots to attain good accuracy, high productivity, efficiency, speed and multiplicity. As the environment or working area may be changeable, the algorithm or the rules must be devised in path planning to ensure an optimistic collision-free path.

A path planning is nothing but the strategy to find a sequence of valid configurations that moves the object from source to destination. It is one of the most important issues in robotics for robot localization and navigation. It lets an autonomous vehicle or a robot find the shortest and most obstacle-free path from start to target point. In the case of mobile robots, the systems, which are responsible for their autonomous movement, are those for localization and navigation. These systems usually operate together and with the help of specialized sensors and algorithms manage to control the movement of mobile robots from one location to another. The localization is also related to mapping. Through recognizing the contours in environment around it, it can create maps of premises, which later can be used by navigation system.

1.2 NEED FOR THE PROJECT

1. Motion tracking enhances human-computer interaction and plays a vital role in computer animation of a 3-D model. It provides real-time information, and the amount of animation data produced by motion tracking within a given time is large.

2. An indoor positioning system (IPS) is a network of devices used to locate people or objects where GPS and other satellite technologies lack precision or fail entirely, such as inside multistory buildings, airports, alleys, parking garages, and underground locations.

3. Applications includes: Indoor Localization, Digitization (Digital Twin) Positioning and Navigation, Occupancy Analytics, Sensor Evaluations, Smart E-Labeling, Asset and People Tracking, Process Automation and Trigger Logics, Bluetooth Low Energy, Ultra-Wideband and RFID, Presence and Motion Monitoring, and Environmental Monitoring.

1.3 PROBLEM STATEMENT

1. Localization in mobile robots is fundamental problem, on which of recent years the scientists have proposed many different solutions and algorithms. Localization is the key component of one navigation system and the successful execution of autonomous movement depends on the accuracy of the positioning.

2. The main problem in this research is the determination of localization in a very tough and dynamically changing environment as well as in tight spaces. Problem formulation – over time, the error in localization with LiDAR increases too much, in the presence of many new dynamic objects on the map. That leads to a large deviation in finding the current position that will obstruct the navigation system.

1.4 OBJECTIVE

1. Therefore, this project is going to present a method for improvement of the localization of a mobile robot, using a LiDAR sensor and IMU sensor.

2. This project is going to find the position using distance measurement and tiltness using IMU sensor. The presented methos can be used for initial localization

of the robot, for correction of the accumulated error by other sensors involved in determination of localization and if the navigation passes into rotation recovery mode, (the algorithm cannot find alternative path).

3. This project also focuses on to find the best path planning algorithm for obstacle avoidance of mobile robot.

1.5 ORGANISATION OF CHAPTERS

Chapter 1	This chapter consists of the introduction and the need of this project. Further, this chapter discuss about the problem statement aroused from literature review and the objective of this project.
Chapter 2	This chapter consists of the Literature review where the statement of projects done by researchers are review and the final inference is taken from the review.
Chapter 3	This chapter states the methodology to achieve the objective and the definitions of path planning and its types. Further, the components used for this project is discussed.
Chapter 4	This chapter gets the simulation part where the different types of algorithms is simulated and shown. Further, the robot setup and the results are shown in this chapter.
Chapter 5	This chapter gives the conclusion and the future work required for this project. The appendix is shown which contains codes used in this project

CHAPTER 2

LITERATURE REVIEW

2.1 LITERATURE REVIEW

2.1.1 LiDAR Sensor and Path planning review

Ava Chikurteva, et al., (2021), the error in localization with LiDAR increases too much, in the presence of many new and dynamic objects on the map. That leads to large deviation in finding the current position that will obstruct the navigation system**.**

Mehmet Korkmaz, et al., (2018), they conclude their research by giving evidence shown below in table 1.1. These evidences are made under static environment.

Table 1.1 Comparison between Algorithm

Algorithm	**Time(s)**	**Distance**	**Efficiency**
A^*	76.96	368	1
RRT^*	1.98	434	32.95778
GA	24.62	524	2.195298
PRM	0.95	397	75.09288
RRT	2.38	446	26.68094

Yan Peng, et al., (2015), The technology of obstacle avoidance is divided into two parts, one is global obstacle avoidance method based on the known environment information and another is local obstacle eavoidance method based on the information from sensor.

Emilio Frazzoli et al., (2010), The robotic motion planning problem has received a considerable amount of attention, especially over the last decade, as robots started becoming a vital part of modern industry as well as our daily life.

Hang Yan, et al., (2018), IMU double integration is an approach with a simple principle: given a device rotation, one measures an acceleration, subtracts the gravity, integrates the residual acceleration once to get velocity and integrates once again to get positions.

2.1.2 IMU Sensor review

Jeff Ferguson (2015), Errors in IMU's Measurements can be broken into three main categories: biases, scale-factor errors, and misalignment errors. IMU biases, b_a and b_g for the accelerometer and gyroscope respectively, are errors in measurement that are present regardless of the forces or rates induced on the sensor.

Mr. Andrew Blake, et al., (2011), Adam Champy explains that by double integrating constant acceleration "Errors increases with the square of time; the error after 1000 seconds is 1,000,000 times greater than at 1 second. Any small offset error in acceleration measurement will soon produce an intolerable error level."

Naveen Prabhu Palanisamy (2016), The Kalman filter is simply an optimal recursive data processing algorithm. It processes all available measurements, regardless of their precision, to estimate the current value of the variables of interest.

2.2 INFERENCE FROM LITERATURE

1. From the observation, it is clearly seen that A* algorithm is the best one regarding the shortest distance. But this algorithm is in need of long computational time in order to obtain optimum distance. PRM (Probabilistic RoadMap) algorithm also seems to be good but it is consuming more memory space compare to A*. So, I would like to prefer RRT* Algorithm since it is less memory consuming algorithm and well efficient compare to A* algorithm. So, this gives a conclusion to prefer RRT* Algorithm since it is less memory consuming algorithm and well efficient compare to A* algorithm.
2. This project would like to prefer the Inertial Measurement Unit (IMU) Sensor for finding the current position and tiltness of the robot.
3. The method to find the distance measurement of the robot is to use IMU double integration method.
4. It is very important to reduce the errors occur in the sensor to give accurate positioning of the robot. Therefore, it is well good to use Kalman Filter for filtering the right data for measurement.

CHAPTER 3

METHODOLOGY

3.1 METHODOLOGY

The Methodology of the project is represented below in figure 3.1.a and 3.1.b in a diagrammatic way. The flow chart or the process flow of the project represents the step-by-step process of mapping and localization of mobile robot using IMU Sensor. Figure 3.1.a discuss the process done by robot while it is stationary. Whereas, Figure 3.2.b discuss the process done by robot while it is in motion.

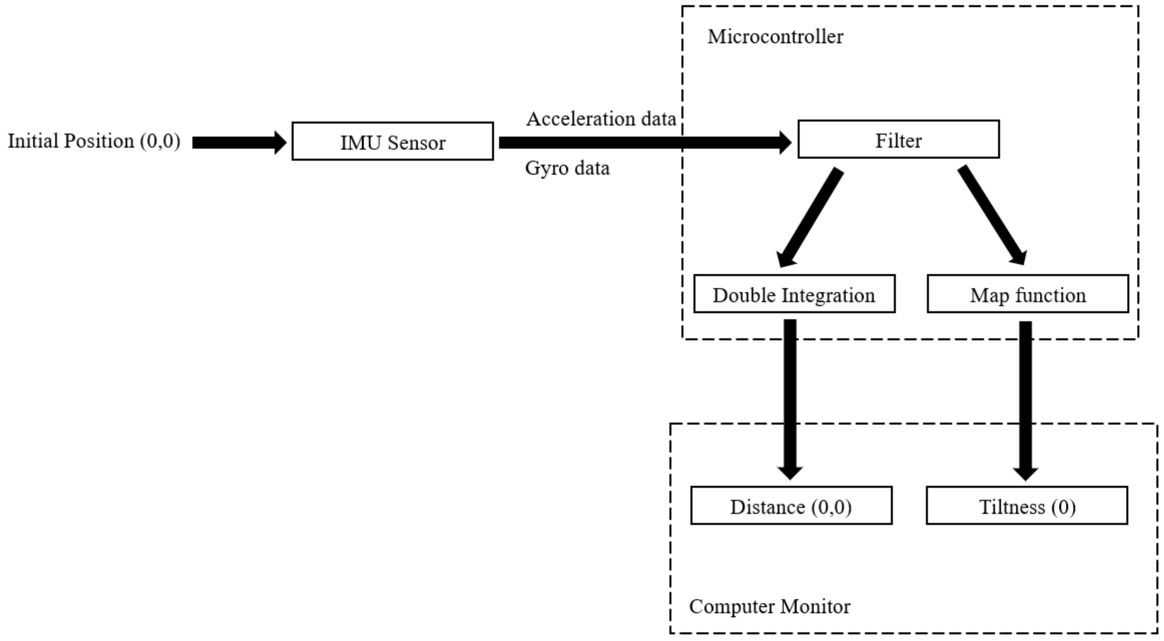


Figure 3.1.a Methodology I

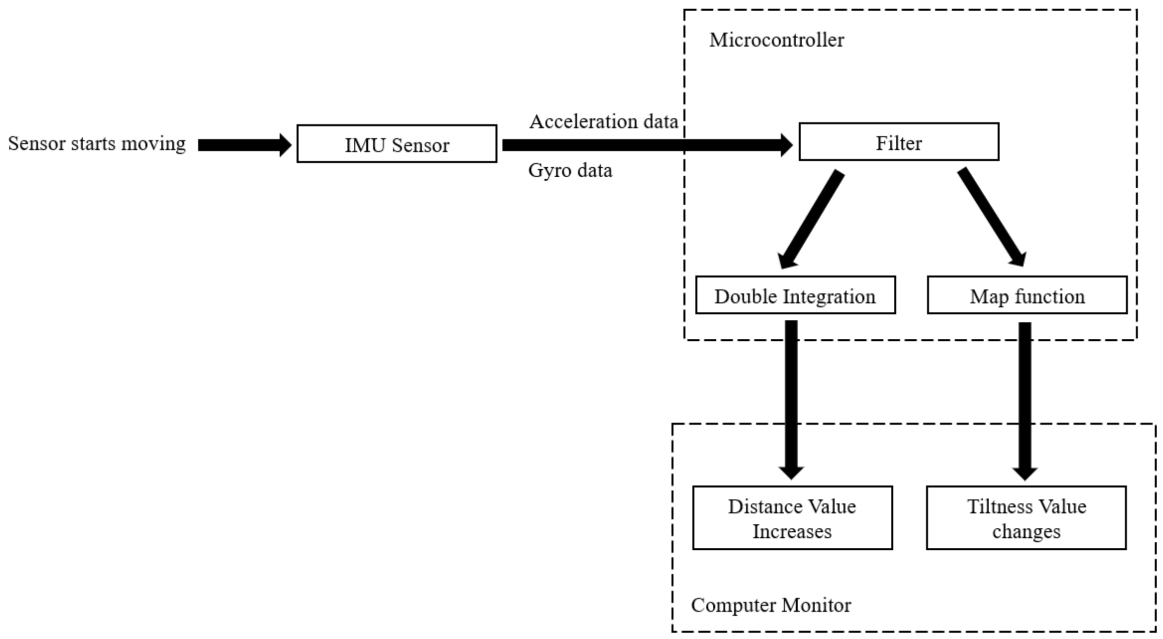


Figure 3.1.b Methodology II

3.2 PATH PLANNING

The path planning problem of mobile robots is defined by this following: mobile robots can find an optimal path from the starting state to the target state that avoids obstacles based on one or some performance indicators (such as the lowest working cost, the shortest walking route, the shortest walking time, etc.) in the motion space. The mobile robot path planning method can be divided into two types according to the known degree of environmental information: path planning based on global map information or local map information. These two path planning methods are referred to as global path planning and local path planning. Another type of classification is shown in figure 1.6.

3.2.1 Global Path Planning

The global path planning method can generate the path under the completely known environment (the position and shape of the obstacle are predetermined). Global path planning involves two parts:

- Establishment of the environmental model and
- The path planning strategy.

There are many mature methods for establishing an environment model for mobile robot path planning. Existing basic environment models mainly include a grid decomposition map, quad split graph, visibility graph, and Voronoi diagram.

After the environmental map is built, global path planning is carried out. After the environmental map is built, global path planning is carried out. Algorithms of global path planning are mainly divided into two types: heuristic search methods and intelligent algorithms. The A∗ algorithm is the most commonly used heuristic graph search algorithm for state space. In addition to solving problems based on state space, it is often used for the path planning of robots. Intelligent algorithms have lots of studies, including ant colony, particle swarm, genetic, bat, simulated annealing, and so forth.

3.2.2 Local Path Planning

Different from the global path planning method, the local path planning focuses on the current local environment information of the mobile robot and uses the local environment information obtained by the sensor to find an optimal path from the starting point to the target point that does not touch the obstacle in the environment. The path planning strategy needs to be adjusted in real time.

Commonly used methods for local path planning include the rolling window, artificial potential field, and various intelligent algorithms.

3.2.3 Classification Based on the Approach

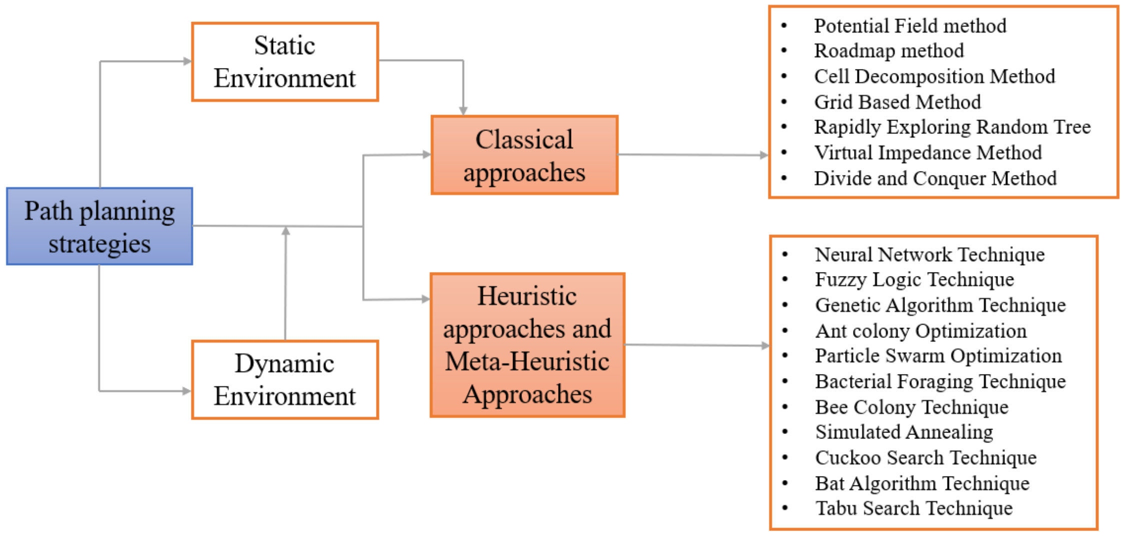

Static Environment
Path planning strategies
Dynamic Environment
Classical approaches
Heuristic approaches and Meta-Heuristic Approaches
• Potential Field method
• Roadmap method
• Cell Decomposition Method
• Grid Based Method
• Rapidly Exploring Random Tree
• Virtual Impedance Method
• Divide and Conquer Method
• Neural Network Technique
• Fuzzy Logic Technique
• Genetic Algorithm Technique
• Ant colony Optimization
• Particle Swarm Optimization
• Bacterial Foraging Technique
• Bee Colony Technique
• Simulated Annealing
• Cuckoo Search Technique
• Bat Algorithm Technique
• Tabu Search Technique

Figure 3.2 Classification based on the approach to solve problem

3.3 COMPONENTS

3.3.1 Microcontroller

For this project, Arduino UNO is used to collect data from IMU sensor. Arduino (shown in figure 3.3) is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

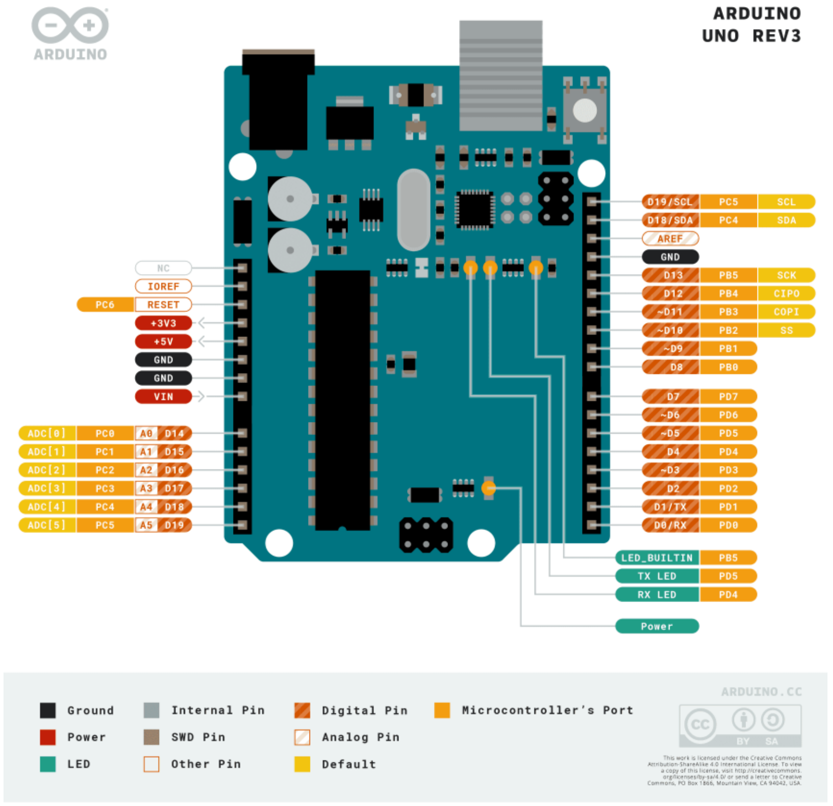


Figure 3.3 Pin diagram of Arduino UNO

3.3.2 Inertial Measurement Unit (IMU) Sensor

The Inertial Measurement Unit sensor (Shown in figure 1.4) is an electronic device used to calculate and reports an exact force of body, angular rate as well as the direction of the body, which can be achieved by using a blend of 3 sensors like Gyroscope, Magnetometer, and Accelerometer. These sensors are normally used to plan aircraft including UAVs (unmanned aerial vehicles), between several others, & spacecraft, comprising landers and satellites. Modern developments permit the manufacture of IMU-based GPS devices. The pin diagram of MPU6050 IMU sensor is shown in figure 3.4.

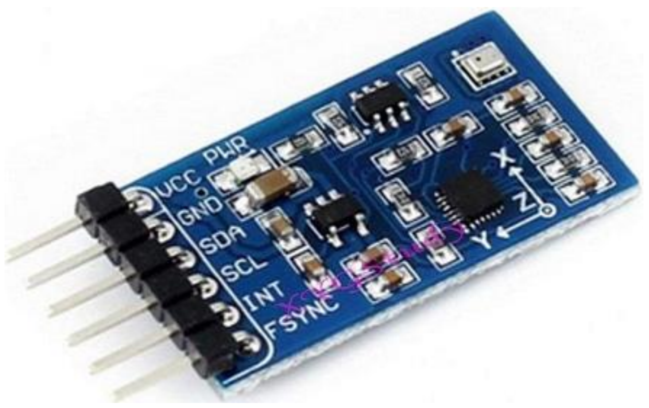


Figure 3.4 An IMU Sensor

3.3.2.1 Working Principle

An IMU sensor unit working can be done by noticing linear acceleration with the help of one or additional accelerometers & rotational rate can be detected by using one or additional gyroscopes. Some also contain a magnetometer which can be used as a heading reference. This sensor includes some usual configurations which include gyro, one accelerometer, and magnetometer for each axis used for each of the 3-vehicle axes like roll, yaw, and pitch.

3.3.2.2 Uses

It measures force, magnetic field, and angular rate. These sensors include a 3-axis accelerometer and gyroscope. So, this would be measured as a 6-axis IMU sensor. These can also comprise an extra 3-axis magnetometer, so it can be considered like a 9-axis IMU.

Officially, the name IMU is just the sensor; however, these are frequently connected with the software like sensor fusion. This sensor merges data from numerous sensors to offer heading & orientation measures. Commonly, this sensor can be used to refer to the blend of the sensor & sensor fusion software, which can also be referred to as an Attitude Heading Reference System (AHRS).

3.3.2.3 Components of imu sensor

IMU consists of three components. They are,

- Accelerometer
- Gyroscope
- Magnetometer (Optional).

ACCELEROMETER

Accelerometers measure linear acceleration in a particular direction. An accelerometer can also be used to measure gravity as a downward force.

GYROSCOPE

Gyroscopes measure angular velocity about three axes: pitch (x axis), roll (y axis) and yaw (z axis). When integrated with sensor fusion software, a gyro can be used to determine an object’s orientation within 3D space.

MAGNETOMETER

A magnetometer measures and can detect fluctuations in Earth’s magnetic field, by measuring the air’s magnetic flux density at the sensor’s point in space.

3.3.2.4 Pin Diagram of MPU6050 IMU Sensor

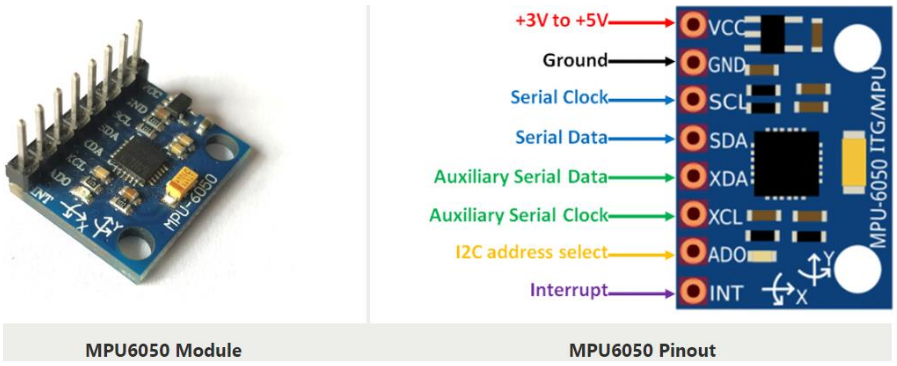


Figure 3.5 Pin Diagram

3.3.2.5 MPU6050 Module Pinout

- **VCC** supplies power to the module.
- **GND** is the ground pin.
- **SCL** is a serial clock pin for the I2C interface.
- **SDA** is a serial data pin for the I2C interface.
- **XDA** is the external I2C data line. The external I2C bus is for connecting external sensors, such as a magnetometer.
- **XCL** is the external I2C clock line.
- **AD0** allows you to change the I2C address of the MPU6050 module. It can be used to avoid conflicts between the module and other I2C devices or to connect two MPU6050s to the same I2C bus. When you leave the ADO pin unconnected, the default I2C address is 0x68$_{\text{HEX}}$; when you connect it to 3.3V, the I2C address changes to 0x69$_{\text{HEX}}$.
- **INT** is the Interrupt Output pin. The MPU6050 can be programmed to generate an interrupt upon detection of gestures, panning, zooming, scrolling, tap detection, and shake detection.

3.3.3 Chassis

Chassis is the load-bearing framework of an artificial object, which structurally supports the object in its construction and function. An example of a chassis is a vehicle frame, the underpart of a motor vehicle, on which the body is mounted; if the running gear such as wheels and transmission, and sometimes even the driver's seat, are included, then the assembly is described as a rolling chassis. The basic model is shown in figure 3.6.

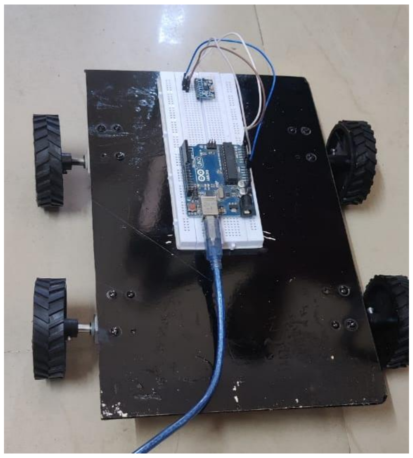


Figure 3.6 chassis of the robot

3.3.4 Wheel

A wheel is a circular component that is intended to rotate on an axle bearing. The wheel is one of the key components of the wheel and axle which is one of the six simple machines. Wheels, in conjunction with axles, allow heavy objects to be moved easily facilitating movement or transportation while supporting a load, or performing labor in machines. A wheel reduces friction by facilitating motion by rolling together with the use of axles. In order for wheels to rotate, a moment needs to be applied to the wheel about its axis, either by way of gravity or by the application of another external force or torque. The basic model is shown in figure 3.7.

Figure 3.7 wheel of the robot

3.3.5 Permanent Magnet DC Motor

Permanent magnet DC motor is an advanced type of motor similar to induction motors. It utilizes the power of electromagnetic principles to generate torque. As the name indicates, this motor uses a permanent magnet to create the magnetic field for operating the DC motor.

A PMDC motor has two major components: Stator and Armature

Stator

The stator is the outer part of the PMDC motor which makes up its housing. Magnets are mounted on the inner side of the stator in such a way that the North and South pole of the magnets alternatively face the armature. Apart from housing the magnets, the stator also serves as a low reluctance return path for the magnetic flux. In case the magnets somehow lose their power, an additional field coil is provided to compensate for the same.

3.3.7 L293D Motor Driver

A motor driver is an integrated circuit chip which is usually used to control motors in autonomous robots. Motor driver act as an interface between Arduino and the motors. The most commonly used motor driver Ics are from the L293 series such as L293D, L293NE, etc. These Ics are designed to control 2 DC motors simultaneously. The basic model is shown in figure 3.9.

Figure 3.8 L293D Motor Driver

CHAPTER 4

SIMUALATION

4.1 DIJKSTRA'S ALGORITHM

Dijkstra's algorithm is a classic algorithm for finding the shortest path between two points due to its optimization capability. The working of this algorithm is shown in figure 4.1.

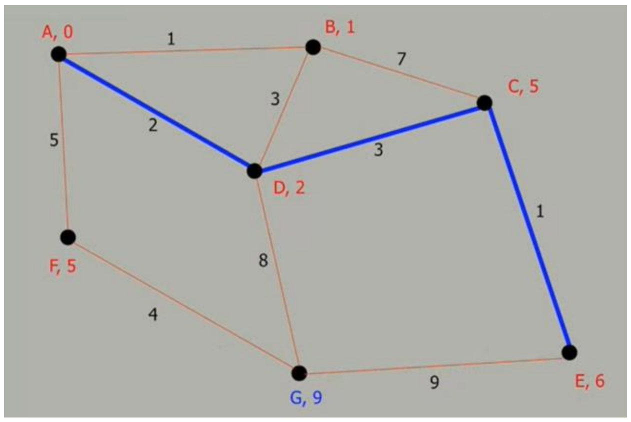
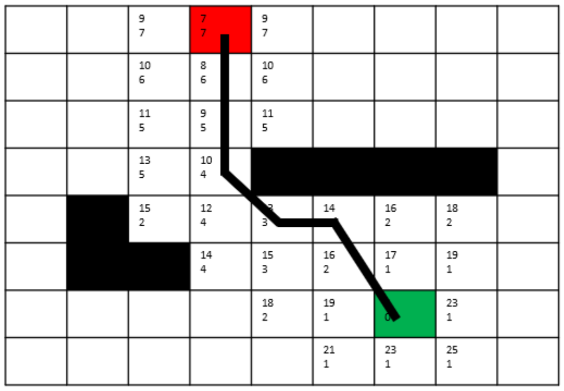


Figure 4.1 Dijkstra's algorithm

4.2 A* ALGORITHM

A* algorithm is a heuristic function-based algorithm for proper path planning. It calculates heuristic function's value at each node on the work area and involves the checking of too many adjacent nodes for finding the optimal solution with zero probability of collision. Hence, it takes much processing time and decreases the work speed. A* is a computer algorithm. It must need processors of high configuration in order to check various nodes successively. The working of this algorithm is shown in figure 4.2.



SYMBOL	DESCRIPTION
🟥	GOAL NODE
🟩	START NODE
⬛	WALL
⬜	PATH

Figure 4.2 A* Algorithm

4.3 RAPIDLY EXPLORING RANDOM TREE(RRT) ALGORITHM

The premise of RRT is actually quite straight forward. Points are randomly generated and connected to the closest available node. Each time a vertex is created, a check must be made that the vertex lies outside of an obstacle. Furthermore, chaining the vertex to its closest neighbour must also avoid obstacles. The algorithm ends when a node is generated within the goal region, or a limit is hit. The working of this algorithm is shown in figure 4.3.

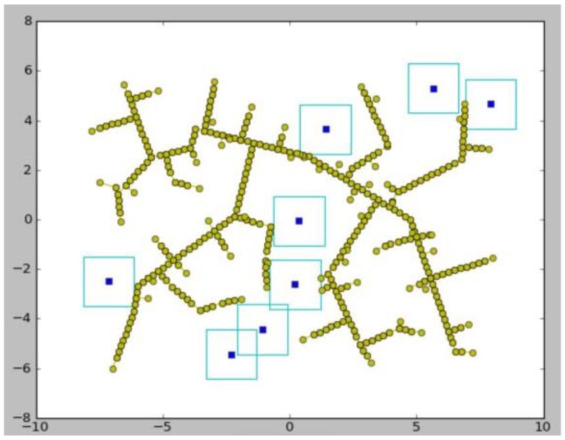


Figure 4.3 RRT Algorithm

4.4 RRT* ALGORITHM

RRT* is an optimized version of RRT. When the number of nodes approaches infinity, the RRT* algorithm will deliver the shortest possible path to the goal. First, RRT* records the distance each vertex has travelled relative to its parent vertex. After the closest node is found in the graph, a neighbourhood of vertices in a fixed radius from the new node are examined. If a node with a cheaper than the proximal node is found, the cheaper node replaces the proximal node. After a vertex has been connected to the cheapest neighbour, the neighbours are again examined. Neighbours are checked if being rewired to the newly added vertex will make their cost decrease. If the cost does indeed decrease, the neighbour is rewired to the newly added vertex. This feature makes the path smoother. RRT* creates incredibly straight paths. The working of this algorithm is shown in figure 4.4.

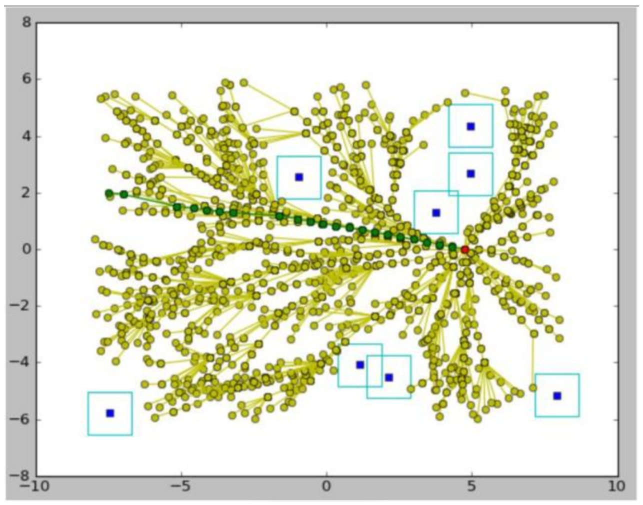


Figure 4.4 RRT* Algorithm

4.5 SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

Simultaneous Localization And Mapping is a method to built model of the environmental map surrounded by a mobile robot, while estimating the robot's pose (Pose means the location and orientation of robot). The working of this technique is shown in figure 4.5.

Slam algorithms have classification:

- Filtering where the robot state is estimated ‘on the go’ with latest measurement.
  E.g.: EKF (Extended Kalman Filter), Particle filter method
- Smoothing where the full robot trajectories are estimated using complete set of measurement.
  E.g.: Pose graph optimization

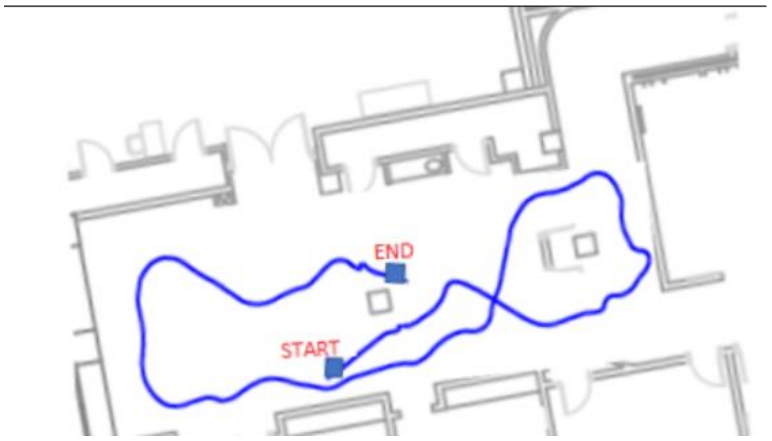


Figure 4.5 SLAM Technique

4.6 OUTPUT

4.6.2 Dynamic RRT

The dynamic RRT code is shown in Appendix and the output is shown in figure 4.6.

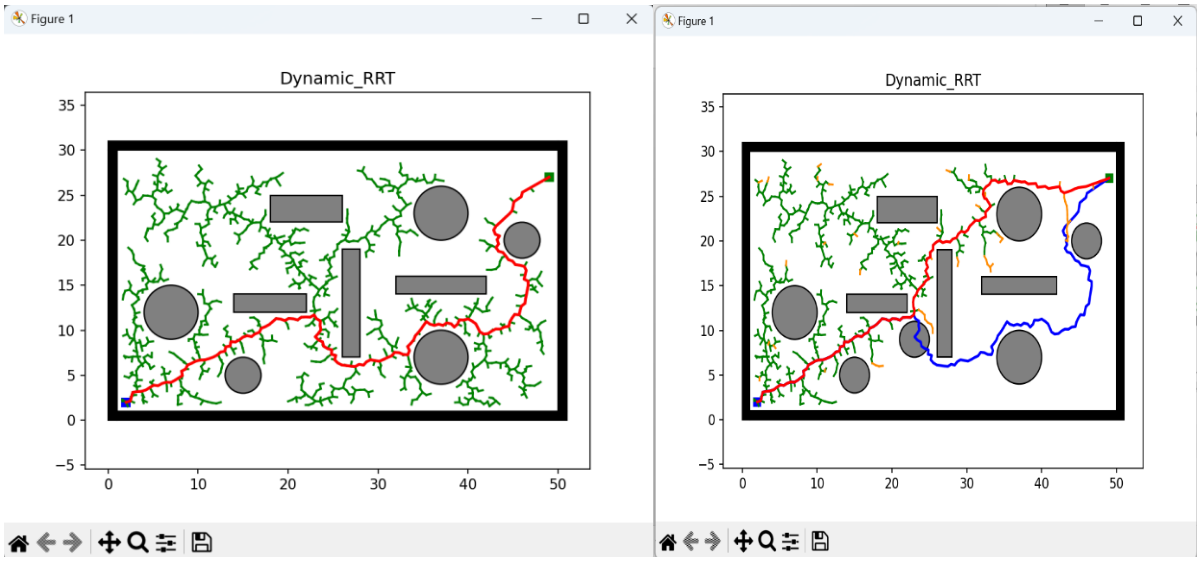


Figure 4.6 Dynamic RRT Output

4.7 CALCUATING TILTNESS OF ROBOT

As we are using LiDAR sensor, we can find the position and orientation (or pose of robot) of a robot by manipulating the data from LiDAR. But the error in measuring the pose of robot using LiDAR will get increased in the presence of dynamically moving obstacles in the environment. To overcome this problem, we are interfacing LiDAR with IMU sensor in microcontroller. IMU (Inertial Measurement Unit) is a device used to measure the angular velocity, linear acceleration and overall orientation of the robot.

4.8 INTERFACE MPU6050 WITH MICROCONTROLLER

Connections are straightforward. Begin by connecting the VCC pin to the Arduino’s 5V output and the GND pin to ground. Now we are left with the pins that are used for I2C communication. Note that each Arduino board has different I2C pins that must be connected correctly. On Arduino boards with the R3 layout, the SDA (data line) and SCL (clock line) are on the pin header close to the AREF pin. They are also referred to as A5 (SCL) and A4 (SDA). The interface diagram is shown in figure 4.7.

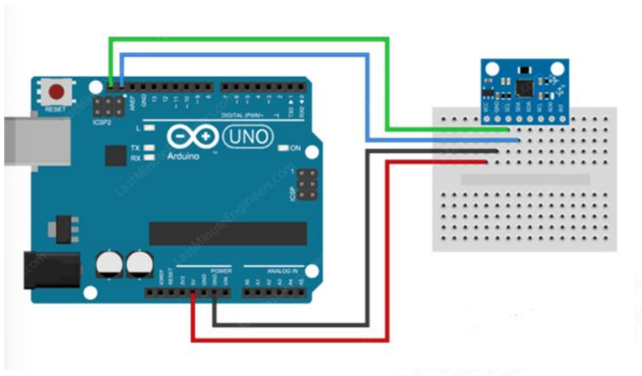


Figure 4.7 Setup Diagram of interface between MPU6050 and Arduino Uno

4.9 DESCRIPTION OF ROBOT SETUP

The interfaced IMU Setup is placed on the four-wheeled robot. The robot consists of a chassis where the sensor is placed, 2 dummy wheels which are placed in front end of the robot and 2 wheels connected to DC motor which are placed in rear end of the robot. Further, the LiDAR sensor with its holder is interfaced with IMU sensor and placed near the IMU Setup.

4.10 LIBRARY INSTALLATION

Setting up the MPU6050 module to begin capturing the device’s raw data output is fairly simple. Manipulating the data into something meaningful, on the other hand, is more difficult, but there are some libraries at our disposal. To install the library, navigate to Sketch > Include Library > Manage Libraries… Wait for the

Library Manager to download the library index and update the list of installed libraries. Filter your search by entering ‘mpu6050’. Look for the Adafruit MPU6050 Library by Adafruit. Click on that entry and then choose Install. The Adafruit MPU6050 library makes use of the Adafruit Unified Sensor Driver and Adafruit Bus IO Library internally. So, search the library manager for Adafruit Unified Sensor and BusIO and install them as well.

Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation is used in I2Cdev.h. I2Cdev and MPU6050 must be installed as libraries, or else the .cpp /.h files for both classes must be in the include path of your project.

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. It can also connect to a serial port.

4.11 RESULT

The robot setup and the IMU sensor distance calculation are done and simulated successfully. The data is collected and converted into excel file by using PuTTY configuration software. Then, the motion is visualized in excel sheet. This process is shown in figures 4.8, 4.9, 4.10, 4.11.

Figure 4.8 The path of robot

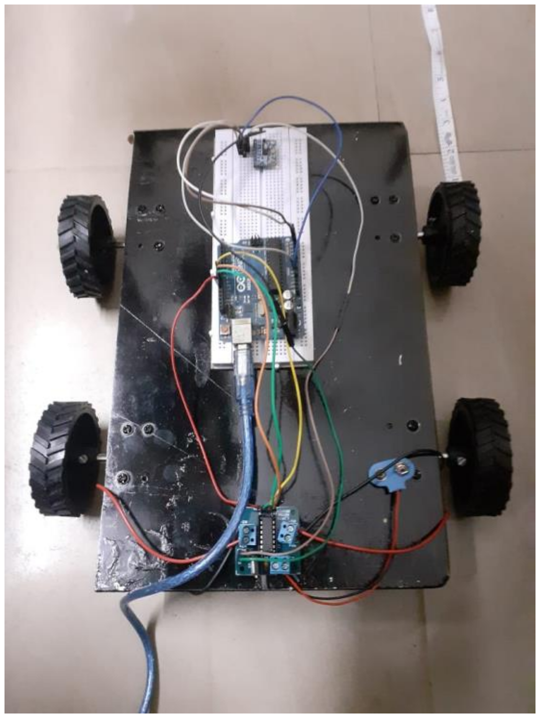


Figure 4.9 Robot at starting position

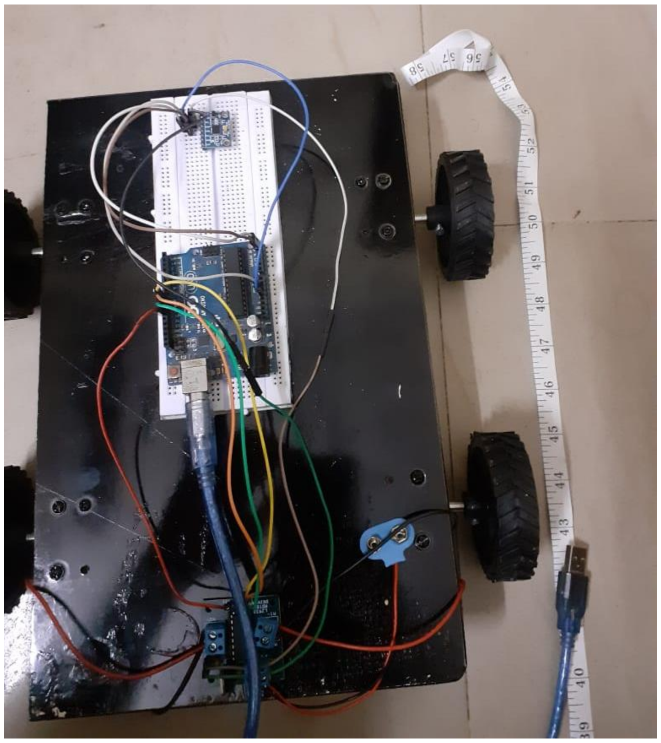


Figure 4.10 Robot at end position

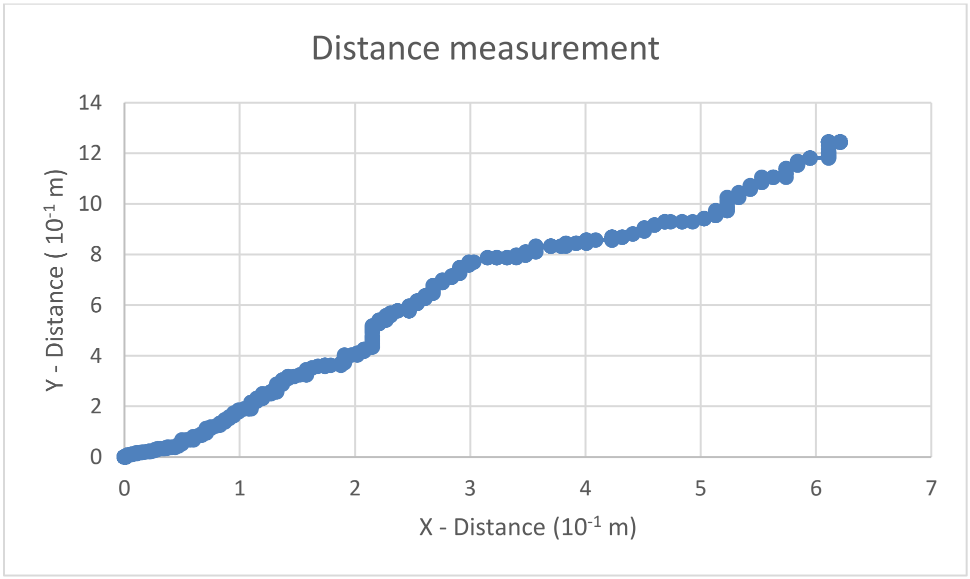


Figure 4.11 The motion plot

However, this calculation is not valid and has much error to be rectified and calibrated. This project requires further study to reduce those errors and the data should be collected in real time.

Adding on to the result, the IMU sensor shows the tilt of robot successfully and is shown in the figures 4.12, 4.13, 4.14, 4.15, 4.16, 4.17 respectively.

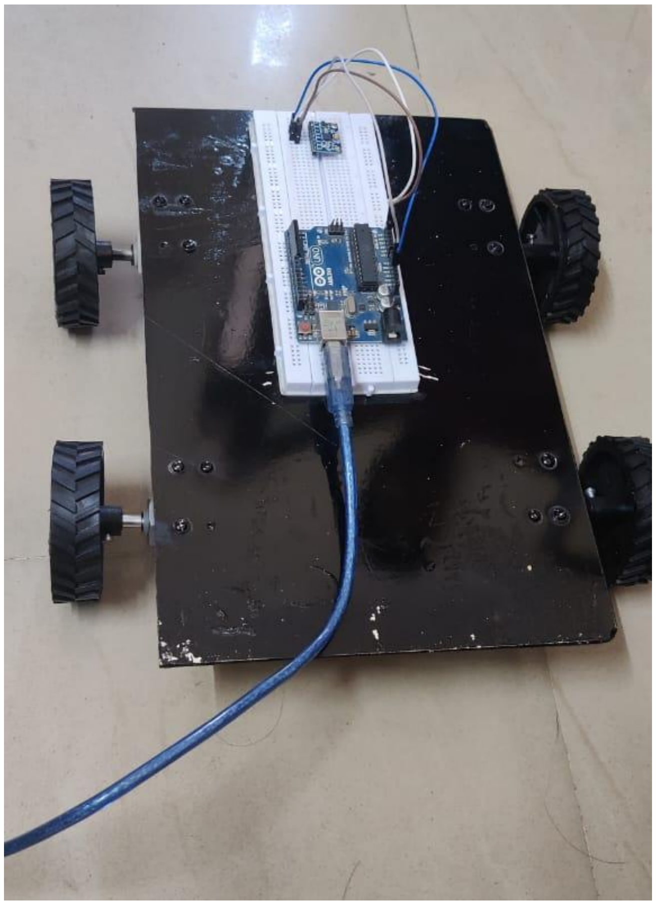


Figure 4.12 Robot in stable condition

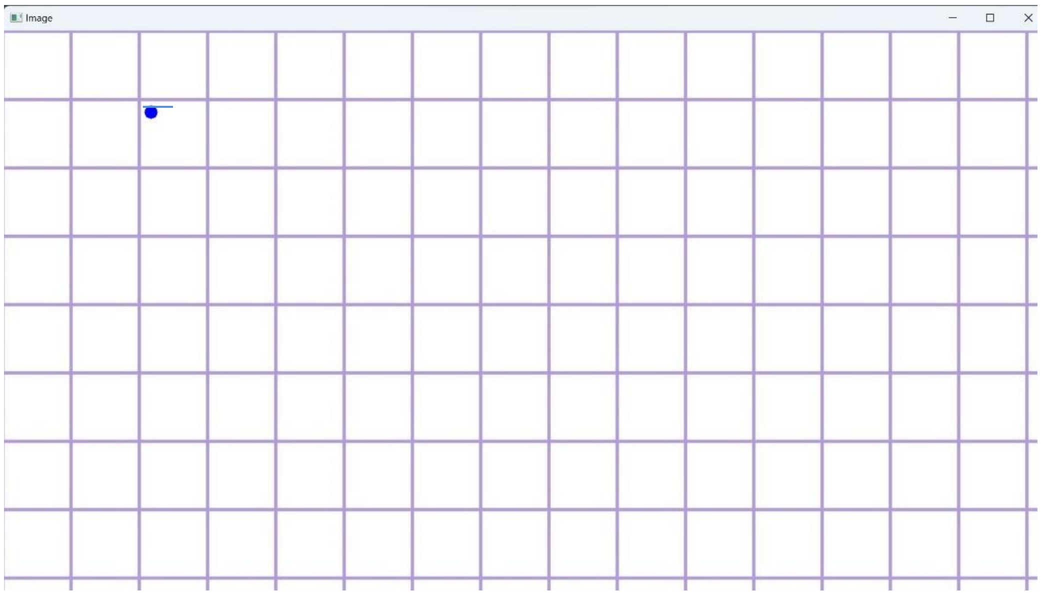


Figure 4.13 Output of data in stable condition

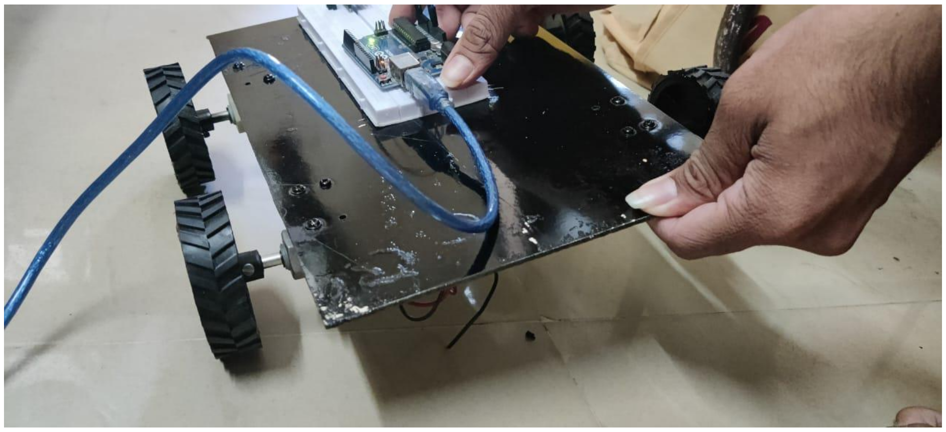


Figure 4.14 Robot tilted towards left

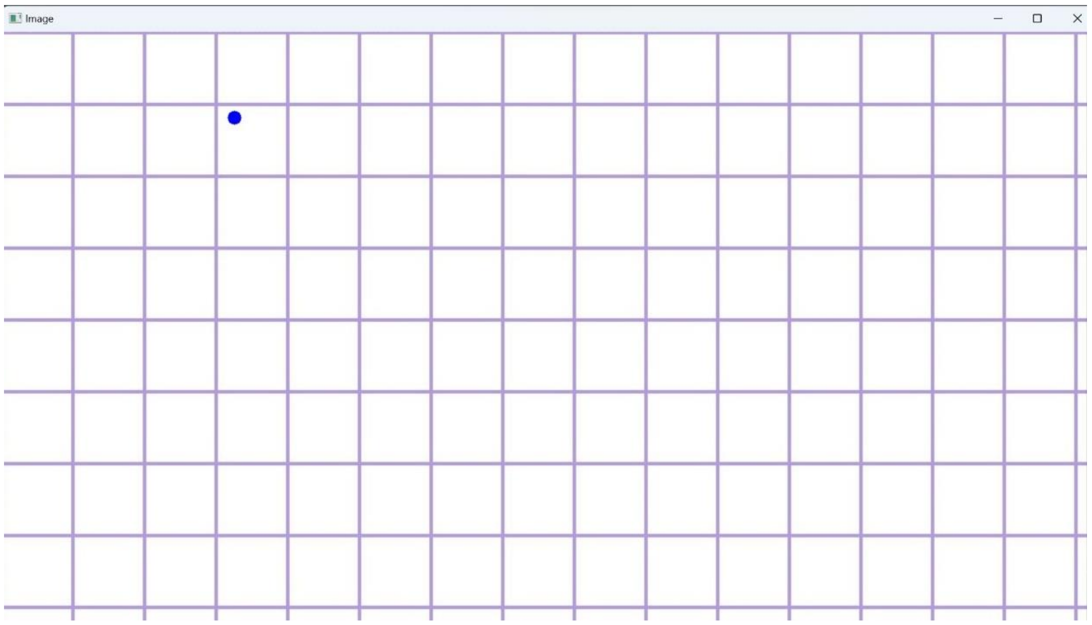


Figure 4.15 Output of data after left tilt

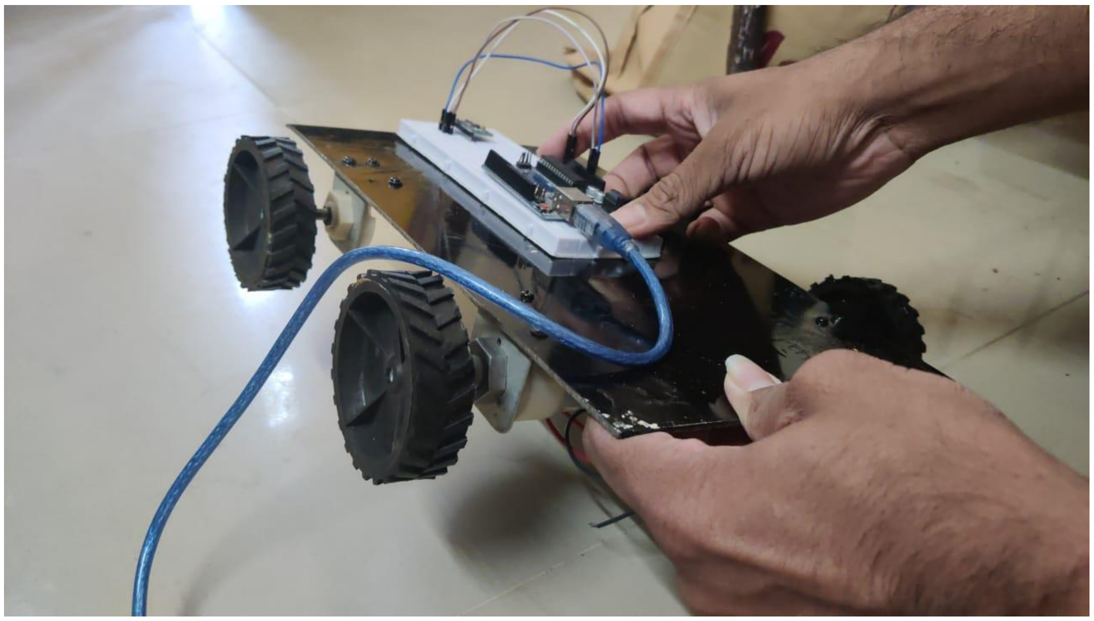


Figure 4.16 Robot tilted towards right

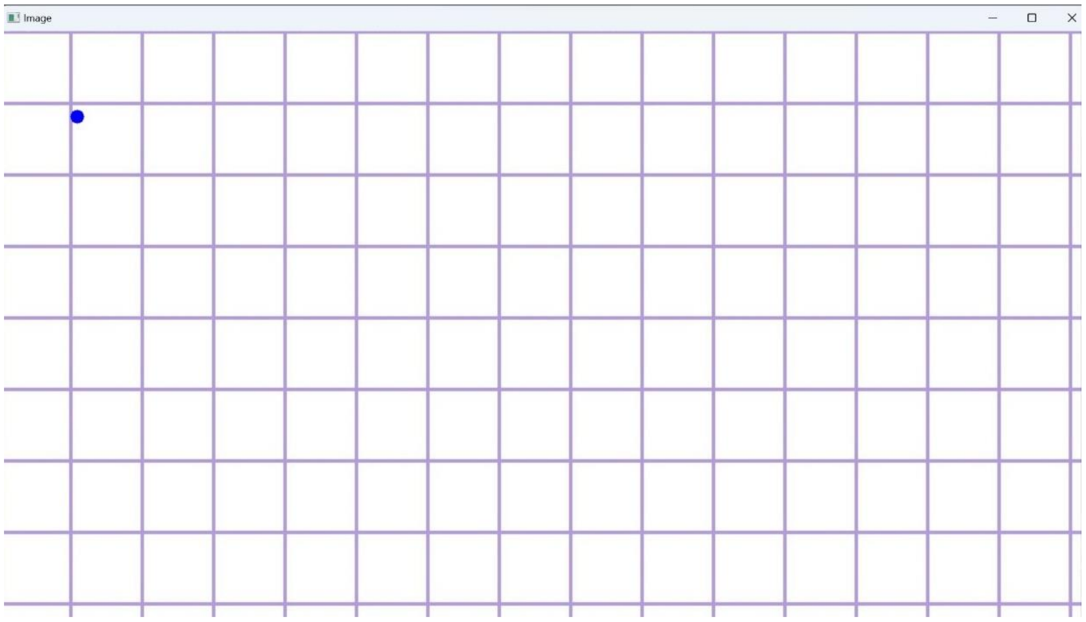


Figure 4.17 Output of data after right tilt

CHAPTER 5
CONCLUSION

5.1 CONCLUSION

Thus, with this setup, we can identify the distance measurement and tilt angle of the robot to ensure that there is no significant error while simulating the path planning algorithm. It is evident from the above result that if the pointer moves towards its left, the robot is tilted rightwards, and when the pointer moves towards its right, the robot is tilted leftwards. With these, we can nullify those tilt angles and make sure of a smooth motion by identifying the robot's position. Furthermore, The attempt of error rectification and calibration for distance measurement is failed for this phase.

5.2 FUTURE WORK

1. This developed position-sensing method has to be cleared and calibrated to obtain the accurate displacement measurements of the robot.
2. The integration of this prototype with the Lidar sensor configuration will enable feedback-based monitoring and controlled motion of the robot (autonomous).
3. This method will allow us to visualize the optimized shortest path planning with instantaneous identification of the obstacles in a dynamic environment.

APPENDIX

VIRTUAL SIMULATION OF PATH

DYNAMIC RRT

```
"""
DYNAMIC_RRT_2D
@author: huiming zhou
"""

import os
import sys
import math
import copy
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches

sys.path.append(os.path.dirname(os.path.abspath(__file__)) +
                "/../../Sampling_based_Planning/")

from Sampling_based_Planning.rrt_2D import env, plotting, utils


class Node:
    def __init__(self, n):
        self.x = n[0]
        self.y = n[1]
        self.parent = None
        self.flag = "VALID"


class Edge:
    def __init__(self, n_p, n_c):
        self.parent = n_p
        self.child = n_c
        self.flag = "VALID"


class DynamicRrt:
    def __init__(self, s_start, s_goal, step_len, goal_sample_rate,
waypoint_sample_rate, iter_max):
        self.s_start = Node(s_start)
        self.s_goal = Node(s_goal)
        self.step_len = step_len
        self.goal_sample_rate = goal_sample_rate
        self.waypoint_sample_rate = waypoint_sample_rate
        self.iter_max = iter_max
        self.vertex = [self.s_start]
        self.vertex_old = []
        self.vertex_new = []
        self.edges = []

        self.env = env.Env()
        self.plotting = plotting.Plotting(s_start, s_goal)
        self.utils = utils.Utils()
        self.fig, self.ax = plt.subplots()

        self.x_range = self.env.x_range
        self.y_range = self.env.y_range
        self.obs_circle = self.env.obs_circle
        self.obs_rectangle = self.env.obs_rectangle
        self.obs_boundary = self.env.obs_boundary
        self.obs_add = [0, 0, 0]

        self.path = []
        self.waypoint = []

    def planning(self):
        for i in range(self.iter_max):
            node_rand = self.generate_random_node(self.goal_sample_rate)
            node_near = self.nearest_neighbor(self.vertex, node_rand)
            node_new = self.new_state(node_near, node_rand)

            if node_new and not self.utils.is_collision(node_near, node_new):
                self.vertex.append(node_new)
                self.edges.append(Edge(node_near, node_new))
                dist, _ = self.get_distance_and_angle(node_new, self.s_goal)

                if dist <= self.step_len:
                    self.new_state(node_new, self.s_goal)
            path = self.extract_path(node_new)
            self.plot_grid("Dynamic_RRT")
            self.plot_visited()
            self.plot_path(path)
            self.path = path
            self.waypoint = self.extract_waypoint(node_new)
            self.fig.canvas.mpl_connect('button_press_event', self.on_press)
            plt.show()

            return

    return None

def on_press(self, event):
    x, y = event.xdata, event.ydata
    if x < 0 or x > 50 or y < 0 or y > 30:
        print("Please choose right area!")
    else:
        x, y = int(x), int(y)
        print("Add circle obstacle at: s =", x, ",", "y =", y)
        self.obs_add = [x, y, 2]
        self.obs_circle.append([x, y, 2])
        self.utils.update_obs(self.obs_circle, self.obs_boundary, self.obs_rectangle)
        self.InvalidateNodes()

        if self.is_path_invalid():
            print("Path is Replanning ...")
            path, waypoint = self.replanning()

            print("len_vertex: ", len(self.vertex))
            print("len_vertex_old: ", len(self.vertex_old))
            print("len_vertex_new: ", len(self.vertex_new))

            plt.cla()
            self.plot_grid("Dynamic_RRT")
            self.plot_vertex_old()
            self.plot_path(self.path, color='blue')
            self.plot_vertex_new()
            self.vertex_new = []
            self.plot_path(path)
            self.path = path
            self.waypoint = waypoint
        else:
            print("Trimming Invalid Nodes ...")
            self.TrimRRT()

            plt.cla()
            self.plot_grid("Dynamic_RRT")
            self.plot_visited(animation=False)
            self.plot_path(self.path)

        self.fig.canvas.draw_idle()

    def InvalidateNodes(self):
```

```
    for edge in self.edges:
        if self.is_collision_obs_add(edge.parent, edge.child):
            edge.child.flag = "INVALID"

def is_path_invalid(self):
    for node in self.waypoint:
        if node.flag == "INVALID":
            return True

def is_collision_obs_add(self, start, end):
    delta = self.utils.delta
    obs_add = self.obs_add

    if math.hypot(start.x - obs_add[0], start.y - obs_add[1]) <= obs_add[2] + delta:
        return True

    if math.hypot(end.x - obs_add[0], end.y - obs_add[1]) <= obs_add[2] + delta:
        return True

    o, d = self.utils.get_ray(start, end)
    if self.utils.is_intersect_circle(o, d, [obs_add[0], obs_add[1]], obs_add[2]):
        return True

    return False

def replanning(self):
    self.TrimRRT()
```

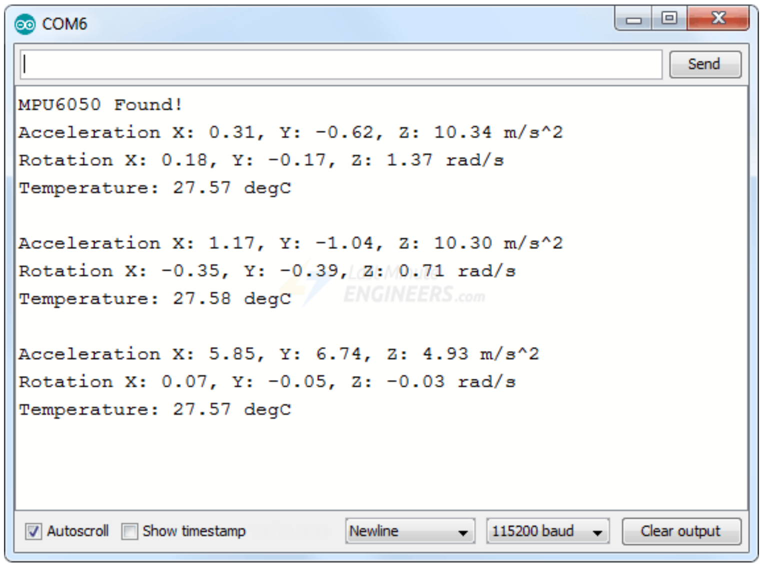
```
        for i in range(self.iter_max):
            node_rand = self.generate_random_node_replanning(self.goal_sample_rate,
    self.waypoint_sample_rate)
            node_near = self.nearest_neighbor(self.vertex, node_rand)
            node_new = self.new_state(node_near, node_rand)

            if node_new and not self.utils.is_collision(node_near, node_new):
                self.vertex.append(node_new)
                self.vertex_new.append(node_new)
                self.edges.append(Edge(node_near, node_new))
                dist, _ = self.get_distance_and_angle(node_new, self.s_goal)

                if dist <= self.step_len:
                    self.new_state(node_new, self.s_goal)
                    path = self.extract_path(node_new)
                    waypoint = self.extract_waypoint(node_new)
                    print("path: ", len(path))
                    print("waypoint: ", len(waypoint))

                    return path, waypoint

        return None

    def TrimRRT(self):
        for i in range(1, len(self.vertex)):
            node = self.vertex[i]
            node_p = node.parent
            if node_p.flag == "INVALID":
                node.flag = "INVALID"

        self.vertex = [node for node in self.vertex if node.flag == "VALID"]
        self.vertex_old = copy.deepcopy(self.vertex)
        self.edges = [Edge(node.parent, node) for node in
self.vertex[1:len(self.vertex)]]

    def generate_random_node(self, goal_sample_rate):
        delta = self.utils.delta

        if np.random.random() > goal_sample_rate:
            return Node((np.random.uniform(self.x_range[0] + delta, self.x_range[1] -
delta),
                         np.random.uniform(self.y_range[0] + delta, self.y_range[1] -
delta)))

        return self.s_goal

    def generate_random_node_replanning(self, goal_sample_rate,
waypoint_sample_rate):
        delta = self.utils.delta
        p = np.random.random()

        if p < goal_sample_rate:
            return self.s_goal
        elif goal_sample_rate < p < goal_sample_rate + waypoint_sample_rate:
            return self.waypoint[np.random.randint(0, len(self.waypoint) - 1)]
        else:
            return Node((np.random.uniform(self.x_range[0] + delta, self.x_range[1] -
delta),
                         np.random.uniform(self.y_range[0] + delta, self.y_range[1] -
delta)))

    @staticmethod
    def nearest_neighbor(node_list, n):
        return node_list[int(np.argmin([math.hypot(nd.x - n.x, nd.y - n.y)
                                        for nd in node_list]))]

    def new_state(self, node_start, node_end):
        dist, theta = self.get_distance_and_angle(node_start, node_end)

        dist = min(self.step_len, dist)
        node_new = Node((node_start.x + dist * math.cos(theta),
                         node_start.y + dist * math.sin(theta)))
        node_new.parent = node_start

        return node_new

    def extract_path(self, node_end):
        path = [(self.s_goal.x, self.s_goal.y)]
        node_now = node_end
        while node_now.parent is not None:
            node_now = node_now.parent
            path.append((node_now.x, node_now.y))

        return path

    def extract_waypoint(self, node_end):
        waypoint = [self.s_goal]
        node_now = node_end

        while node_now.parent is not None:
            node_now = node_now.parent
            waypoint.append(node_now)

        return waypoint

    @staticmethod
    def get_distance_and_angle(node_start, node_end):
        dx = node_end.x - node_start.x
        dy = node_end.y - node_start.y
        return math.hypot(dx, dy), math.atan2(dy, dx)

    def plot_grid(self, name):

        for (ox, oy, w, h) in self.obs_boundary:
            self.ax.add_patch(
                patches.Rectangle(
                (ox, oy), w, h,
                edgecolor='black',
                facecolor='black',
                fill=True
            )
        )

    for (ox, oy, w, h) in self.obs_rectangle:
        self.ax.add_patch(
            patches.Rectangle(
                (ox, oy), w, h,
                edgecolor='black',
                facecolor='gray',
                fill=True
            )
        )

    for (ox, oy, r) in self.obs_circle:
        self.ax.add_patch(
            patches.Circle(
                (ox, oy), r,
                edgecolor='black',
                facecolor='gray',
                fill=True
            )
        )
```

```
        plt.plot(self.s_start.x, self.s_start.y, "bs", linewidth=3)
        plt.plot(self.s_goal.x, self.s_goal.y, "gs", linewidth=3)

        plt.title(name)
        plt.axis("equal")

    def plot_visited(self, animation=True):
        if animation:
            count = 0
            for node in self.vertex:
                count += 1
                if node.parent:
                    plt.plot([node.parent.x, node.x], [node.parent.y, node.y], "-g")
                    plt.gcf().canvas.mpl_connect('key_release_event',
                                                 lambda event:
                                                 [exit(0) if event.key == 'escape' else None])
                    if count % 10 == 0:
                        plt.pause(0.001)
        else:
            for node in self.vertex:
                if node.parent:
                    plt.plot([node.parent.x, node.x], [node.parent.y, node.y], "-g")

    def plot_vertex_old(self):
        for node in self.vertex_old:
            if node.parent:
                plt.plot([node.parent.x, node.x], [node.parent.y, node.y], "-g")
    def plot_vertex_new(self):
        count = 0

        for node in self.vertex_new:
            count += 1
            if node.parent:
                plt.plot([node.parent.x,          node.x],          [node.parent.y,          node.y],
color='darkorange')
                plt.gcf().canvas.mpl_connect('key_release_event',
                                             lambda event:
                                             [exit(0) if event.key == 'escape' else None])
                if count % 10 == 0:
                    plt.pause(0.001)

    @staticmethod
    def plot_path(path, color='red'):
        plt.plot([x[0] for x in path], [x[1] for x in path], linewidth=2, color=color)
        plt.pause(0.01)


def main():
    x_start = (2, 2)  # Starting node
    x_goal = (49, 27)  # Goal node

    drrt = DynamicRrt(x_start, x_goal, 0.5, 0.1, 0.6, 5000)
    drrt.planning()
```

```
if __name__ == '__main__':
    main()
```

CODE FOR GETTING ACCELERATION DATA FROM IMU

```
#include <Adafruit_MPU6050.h>

#include <Adafruit_Sensor.h>

#include <Wire.h>



Adafruit_MPU6050 mpu;



void setup(void) {

  Serial.begin(9600);



  // Try to initialize!

  if (!mpu.begin()) {

    Serial.println("Failed to find MPU6050 chip");

    while (1) {

      delay(10);

    }
  }

  // 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);

  delay(100);
}

void loop() {
  /* Get new sensor events with the readings */
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  /* Print out the values */

  Serial.print(a.acceleration.x);

  Serial.print(",");

  Serial.print(a.acceleration.y);

  Serial.print(",");

  Serial.print(a.acceleration.z);

  Serial.print(", ");

  Serial.print(g.gyro.x);

  Serial.print(",");

  Serial.print(g.gyro.y);

  Serial.print(",");

  Serial.print(g.gyro.z);

  Serial.println("");


  delay(10);

}
```



```
MPU6050 Found!
Acceleration X: 0.31, Y: -0.62, Z: 10.34 m/s^2
Rotation X: 0.18, Y: -0.17, Z: 1.37 rad/s
Temperature: 27.57 degC

Acceleration X: 1.17, Y: -1.04, Z: 10.30 m/s^2
Rotation X: -0.35, Y: -0.39, Z: 0.71 rad/s
Temperature: 27.58 degC

Acceleration X: 5.85, Y: 6.74, Z: 4.93 m/s^2
Rotation X: 0.07, Y: -0.05, Z: -0.03 rad/s
Temperature: 27.57 degC
```

Figure 5.1 Output of the code

CODE FOR GETTING DISTANCE MEASUREMENT

// 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"

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files

// for both classes must be in the include path of your project

#include "I2Cdev.h"

```
#include "MPU6050.h"


// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
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



void setup() {

// join I2C bus (I2Cdev library doesn't do this automatically)

Wire.begin();
```

```
// initialize serial communication

// (38400 chosen because it works as well at 8MHz as it does at 16MHz, but

// it's really up to you depending on your project)

Serial.begin(9600);


// 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);
  delay(10);
}


void loop() {
// read raw accel/gyro measurements from device


count=count+1;
//sensors_event_t a, g, temp;
//  mpu.getEvent(&a, &g, &temp);
accelgyro.getMotion9(&ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz);
```

```
//ax=a.acceleration.x;

//ay=a.acceleration.y;

//az=a.acceleration.z;

//gx=g.gyro.x;

//gy=g.gyro.y;

//gz=g.gyro.z;


bx=(float)ax/(1638.4);

by=(float)ay/(1638.4);

bz=(float)az/(1638.4);

float pitch = atan(bx/sqrt(pow(by,2) + pow(bz,2)));

float roll = atan(by/sqrt(pow(bx,2) + pow(bz,2)));

pitch = pitch * (180.0/PI);

roll = roll * (180.0/PI);

float yawRaw=atan2( (-my*cos(roll) + mz*sin(roll) ) , (mx*cos(pitch) +
my*sin(pitch)*sin(roll)+ mz*sin(pitch)*cos(roll)) ) *180/PI;

float YawU=atan2(-my, mx) *180/PI;

bx=bx-chax;

by=by-chay;

bz=bz-chaz;
```

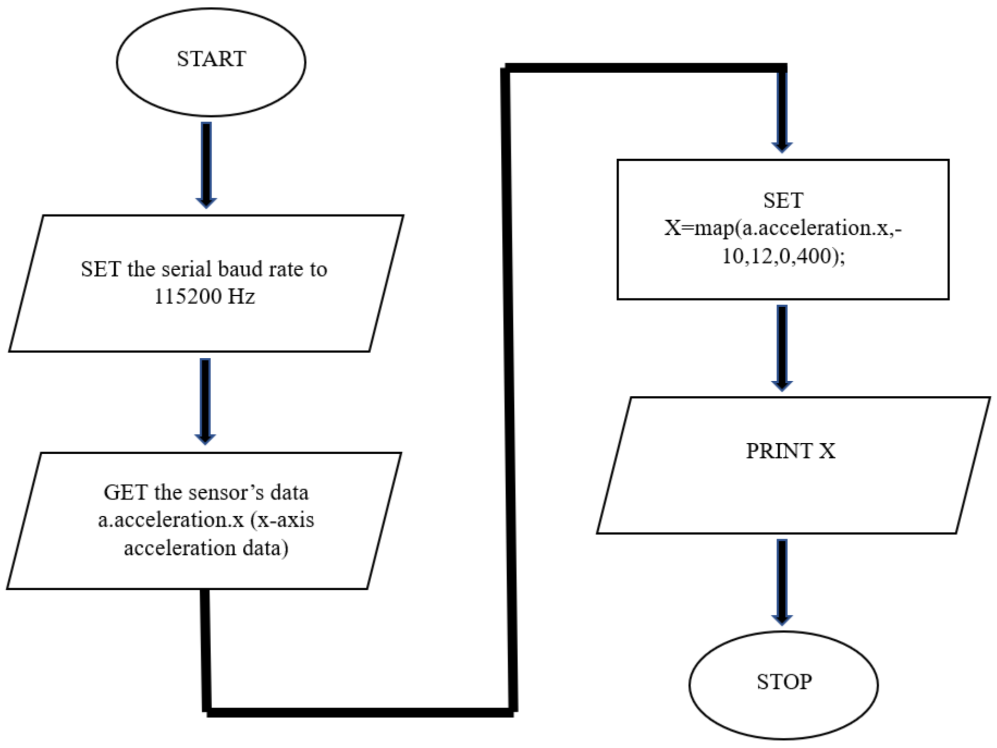
```
cx=gx/65.5;

cy=gy/65.5;

cz=gz/65.5;


//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;

}


//Serial.print("\n");
```

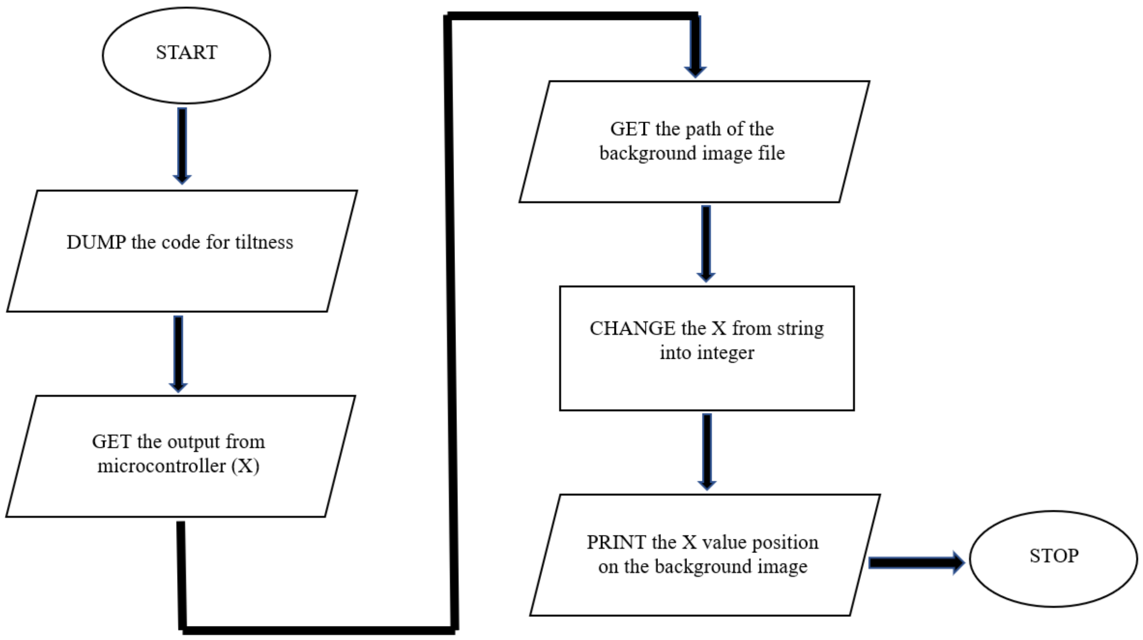
```
Serial.print((s*100)); //PRINT

Serial.print(",");

Serial.println((sy*100));

//Serial.print("\n");

//Serial.print((sz*1000));

//Serial.print(" ");

//Serial.print(interval);

//Serial.print(" ");

//Serial.print(pitch);

//Serial.print(" ");

//Serial.print(YawU);

//Serial.print(" ");

//Serial.println(roll);



//Serial.print(" ");

// Serial.print(interval,10);



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);

}
```

FLOWCHART OF CODE FOR GETTING TILTNESS DATA


START
SET the serial baud rate to 115200 Hz
GET the sensor's data a.acceleration.x (x-axis acceleration data)
SET X=map(a.acceleration.x,-10,12,0,400);
PRINT X
STOP

FLOWCHART OF PYTHON CODE FOR VISUALIZING THE DATA


START
DUMP the code for tiltness
GET the output from microcontroller (X)
GET the path of the background image file
CHANGE the X from string into integer
PRINT the X value position on the background image
STOP

REFERENCES

1. Akif Durdu. Mehmet Kormaz. (2018) “Comparison of optimal path planning algorithms” Vol. 1, No. 1, pp. 255-257

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

3. 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.

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

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

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

7. Jeff Ferguson. (2015) “Calibration of Deterministic IMU Errors” Vol. 1, No. 1, pp. 4-9

8. Jay A. Farrell. et al., (2022) “IMU Error Modeling Tutorial” Vol. 42, No. 6, pp. 40-66.

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

10. Yan Peng. Et al., (2015) “The Obatacle Detection and Avoidance Algorithm Based on 2-D Lidar” Vol. 97, No. 15, pp. 1648-1653

11. 1.3: Dijkstra's Algorithm, Robotics: computational Motion Planning (online course on Coursera given by University of Pennsylvania). Citated at 2015

12. MathWorks Help Centre, "Implement Simultaneous Localization And Mapping (SLAM) with Lidar Scans". Citated at 2015

13. https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/. Citated at 25th march 2018

14. Video Series: "MATLAB Tech Talk by Brian Douglas". https://youtube.com/playlist?list=PLn8PRpmsu08rLRGrnF-S6TyGrmcA2X7kg . Citated at 2015