Automatic Seed Sowing Robot

Author

Capt Rana Farid Riaz

00000241073

Maj Malik Muhammad Saddam

00000241073

Capt Muhammad Hamza Bin Zahid

00000241054

Capt Muhammad Kamran

00000241067

Supervisor

Col (R) Attiq Ahmad

Submitted to the faculty of Department of Electrical Engineering,

Military College of Signals, National University of Sciences and Technology,

in partial fulfillment for the requirements of B.E Degree in Electrical Engineering

June, 2021

CERTIFICATE OF CORRECTIONS & APPROVAL

Certified that work contained in this thesis titled "Automatic Seed Sowing Robot" carried out by Rana Farid Riaz, Muhammad Hamza Bin Zahid, Malik Muhammad Saddam and Muhammad Kamran under the supervision of Col (R) Attiq Ahmad for partial fulfillment of Degree of Bachelors of Electrical Engineering, in Military College of Signals, National University of Sciences and Technology, Islamabad during the academic year 2019-2021 is correct and approved. The material that has been used from other sources it has been properly acknowledged / referred.

Approved by

Supervisor

____

____

Date: ________________

DECLARATION

No portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

Plagiarism Certificate (Turnitin Report)

This thesis has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

Rana Farid Riaz
00000241073

Muhammad Hamza Bin Zahid
00000241054

Malik Muhammad Saddam
00000241073

Muhammad Kamran
00000241067

Signature of Supervisor

Acknowledgements

We are thankful to our Creator Allah Subhana-Watala to have guided us throughout this work at every step and for every new thought which we setup in our minds to improve it. Indeed we could have done nothing without His priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was His will, so indeed none be worthy of praise but Him.

We are profusely thankful to our beloved parents who raised us when we were not capable of walking and continued to support us throughout in every aspect of our life.

We would also like to express special thanks to our supervisor Col (R) Attiq Ahmad for his help throughout our thesis. Without his help we wouldn't have been able to complete our thesis. We appreciate his patience and guidance throughout the whole thesis.

Finally, we would like to express our gratitude to all faculty members and individuals who have rendered valuable assistance to our study.

Dedicated to our exceptional parents and adored siblings whose tremendous support and cooperation led us to this accomplishment.

Abstract

Agriculture is the mainstay of Pakistan's economy. It is also an important source of foreign exchange earnings and stimulates growth in other sectors. The government is focusing on supporting small and marginalized farmers and promote small scale innovative technologies to promote growth in this sector. According to the 6th Population and Housing Census of Pakistan 2017, the country's population is growing at the rate of 2.4 percent per annum. This rapid increase in population is raising demand for agricultural products. There are vast gaps between the acquired and actual output of production, which suffers due to a lack of appropriate technology, use of inputs at improper times and unavailability of water and sometimes over watering which not only negatively affects the production but also significantly reduces the fertility of soil as well. Another one of the greatest problem is the wastage of seed during sowing season. The present government is focused on developing this sector and in this connection initiated a number of measures such as crop diversification, efficient use of water and promotion of high value crops including biotechnology. But the problem of seed wastage is still persistent and very less importance is given to this issue. Another problem that we are facing is use of extensive manpower in lieu of low production. Our production do not meet the manpower ratio we use in this sector. One of the main reason is lack of use of technology in this sector.

To overcome this issue, we have analyzed the issue and proposed a prototype of Automatic Seed Sowing Robot. The robot is a sample for further improvements hence the paper can be read in this perspective. Use of robot in this area will not only curtail the wastage of seed but also reduce the manpower significantly. As our government is working on promoting small scale farming, keeping in view, Automatic Seed Sowing robot is the solution. This modal can also be replicated on a bigger scale with modification to be used in large fields and in sowing of different types of seeds.

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

Pakistan is an agrarian country. Agriculture has a significant role in shaping our economy. Agriculture contributed around 22% to its GDP in 2019 whereas it employed 42% of Pakistan's labor force. Comparison of agricultural share in GDP and use of labor for this contribution reflects that there is a dire need to increase the agricultural yield with respect to the manpower being used.

1.1 Scope and Motivation

Automatic Seed Sowing Robot is a prototype for seed sowing in a prepared field. It plays a major role in reducing manpower being used for seed sowing and in reduction of seed wastage. The robot is equipped with various devices and sensors for high precision and analysis of environment before seed sowing. It can be operated via android app, capable of avoiding obstacles and sow the seed according to the particular requirement of the seed. The robot sows the seeds as per the requirement of the desired crop keeping in view the inter seed and inter lane distances and depth. The chassis of the robot is a 4 wheeler vehicle made up of iron bars. Wheels of the robot is made up of iron which help it in movement in a muddy field. It operates on the principle of differential drive. Being a prototype, the robot is capable of carrying 2 kg of seeds.

The Automatic Seed Sowing Robot which we have chosen for our project is a robot whose aim is to reduce the use of manpower in seed sowing process and curtail the wastage of time and seed.

1.2 Design and Devices

Our proposed seed sowing robot is a 4 wheeler vehicle made up of iron and aluminum. The proposed chassis and wheels of the robot have been specially designed after repeated experimentation on a prepared field. Various types of robot designed were considered and trials were performed by the group on the desired terrain. After a detailed analysis and consideration the subsequent design has been proposed in this paper. Merits and de merits of various designs, components and parts which have been used and those which have not been considered suitable for this system is also discussed in coming paragraphs.

Footnote may be given with Font size 10 of Times New Roman.

1.2.1 Chassis

The proposed material of chassis can be aluminum. Aluminum being light weight offers reduced weight of robot. But, the use of aluminum as a chassis material results in increased cost of construction due to expensive raw material. Assembling and labor cost of aluminum chassis is also high due to lack of skillful manpower.

Figure 1 - Chassis

On contrary, Iron chassis is cost effective due to cheaper and easily available raw material and easily available manpower for its construction. Hence, in our proposed design, the robot housed the electronic circuitry and seed sowing mechanism on a 1.5 x 2.5 ft iron chassis. The shaft of the wheels are made up of aluminum to reduce the weight of the robot and facile movement of robot. Iron angle bars are installed in the middle of the structure of support the wheel shafts and bear the weight of robot. The design provides the ground clearance of 4 inches.

1.2.2 Wheels

The initially proposed design of the robot consisted of plastic tires. These tires are easily available in market, are cost effective and light weight. But these type of tires/ wheels lack grip in muddy terrains. The same very problem arises by the use of air filled rubber tires. This types of tires also pose the problem of punctures when driven in fields and rugged terrain. Trails have been made by this group on plastic and rubber wheels which lack grip in a prepared muddy terrain.

Figure 2 – Specially Designed Wheel

To resolve this problem, we have designed the custom made iron wheels for this robot. We have taken into consideration the working of tractor in a field. The rear wheels of this robot are analogous to the tires of a tractor. The proposed wheels are now made up of iron, having diameter of 10 inches. The rear wheels have specially made skewers on its circumference to grip the ground and push or pull the robot in desired direction.

Footnote may be given with Font size 10 of Times New Roman.

1.2.3 Drive/ Steering Mechanism

The robot works on the principle of differential drive hence, two motors along with chains and spur gear sets have been used for both left and right wheels. Both the wheels can independently be driven either forward or backward. The robot can be steered in any direction and can be turned on any angle just by the variation in the speed of both motors. The point at which the robot rotates about is called Instantaneous Center of Curvature.

Figure 3 – Gear Set

The spur gear set used in the robot is a commonly used in motor bikes. The drive gears of the robot have 14 teeth whereas the driven gears have 41 teeth. The gear ratio can be found by dividing the number of teeth on drive gear to the number of teeth on driven teeth. Hence the gear ratio of our robot is 1:2.92.

1.2.4 Seed Sowing Mechanism

The robot proposed in this paper is a prototype which is designed for the crops whose seed sowing is performed by throwing. These crops include wheat, corn and cotton et cetra. These crops do not require in depth sowing of seed. Seed is generally sown in a depth on 1 inch. Seed sowing has further two parts:

1.2.4.1 Ploughing Mechanism

Before the seed is sown in the ground, an adequate space is made in the mud to throw the seed and then cover it from the mud. For this purpose, we have incorporated a nut and a long bolt in the structure. Nut of the mechanism is welded on the front of robot. Bolt is then tightened in the nut. The length of the bolt can be adjusted both manually and automatically. For the automatic operation, bolt is driven through a belt and servo motor. When the servo motor rotate the bolt in clockwise direction, it moves deep inside the ground. On counter clockwise movement, the bolt moves upwards. An Iron angle is aligned to this nut and bolt and is installed at the rear end of robot. When the robot moves, the bolt makes the space for the seed to be sown. Sowing mechanism throws the seed in space and angle cover the seed with soil.

Footnote may be given with Font size 10 of Times New Roman.

1.2.4.2 Sowing Mechanism

Sowing mechanism consist of funnels and the stepper motors. The assembly is housed in a wooden structure. Funnels can take two different type of seed. Seed up to 2 kg can be stored in the funnels. At the bottom end of the funnels, wooden disc has been installed. Shafts of the stepper motor in engraved in the wooden disc which rotates the disc and throw the seed according to our requirement. The disc with its one complete revolution throws two seeds. Two more funnels are also installed below this arrangement and are connected to pipes. These funnels collect the seed thrown from the wooden disc and with the help of pipes, lay them on the desired location.

Figure 4 – Seed Sowing Mechanism

1.2.5 Devices

The Mechanical structure of the Automatic Seed Sowing Robot has been interfaced with the electronic circuitry especially designed for this purpose. The block diagram of the system is shown below followed by brief of the devices/ sensors used.

1.2.5.1 Raspberry Pi 4 Model B

Raspberry Pi is a low cost and small sized computer which is capable of performing every task that a desktop computer can do. Raspberry Pi 4 Model B is the first variant of the 4th Generation of Raspberry Pi computers/ micro controllers. The micro controller is just 3.4 x 2.2 x 0.4 inches in dimension and weighs only 40grams. It comes with Broadcom BCM2711B0 quad-core ARM processor and the 4K-capable

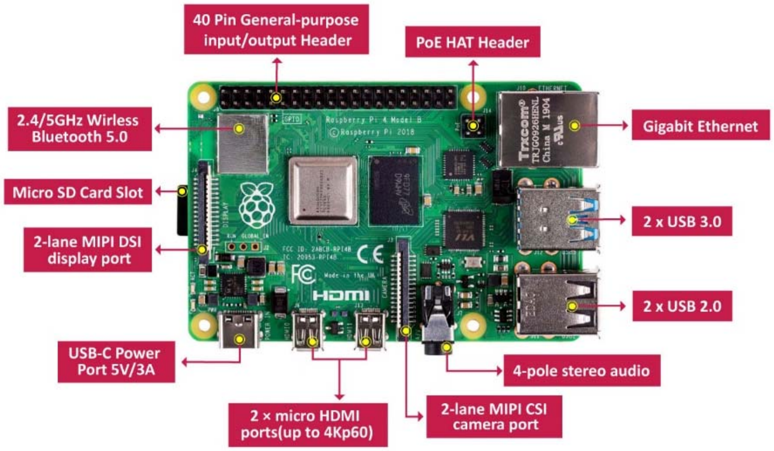


Figure 5 – Raspberry Pi

Footnote may be given with Font size 10 of Times New Roman.

Broadcom Video Core VI video processor. The variant also has upgraded 3.0 USB ports and Type C ports for power. The Pi provides ports for Ethernet, Wifi and Bluetooth. In addition to these common ports, the Pi 4 also provides Camera Serial interface (CSI), Display Serial Interface and Micro SD card slot to enhance the storage capacity.

The most important feature of Raspberry Pi 4 Model B in the 4 Pin GPIO header. These I/O pins provide direct access to connect external devices.

1.2.5.2 MPU 9250

MPU 9250 is a 9 axis Motion Processing Unit. It is system in package (SiP) that contains a 3-axis Accelerometer, 3-axis Gyroscope and a 3-axis Magnetometer. It also contain Digital Motion Processor which makes it capable to process complex motion fusion algorithms. MPU-9250directly provides complete 9-axis Motion Fusion output. The prime advantage of this SiP is the low power consumption. The device operate on 6.4 µA making it easy for use in circuits. It consists of 3 Analog to digital converters (ADC) for digitizing the outputs of accelerometer, gyroscope and magnetometer each. The device provides data in 3 axis about acceleration, angular velocity and magnetic strength. Microcontroller can be connected to MPU9250 easily by using I2C or SPI serial bus interface.

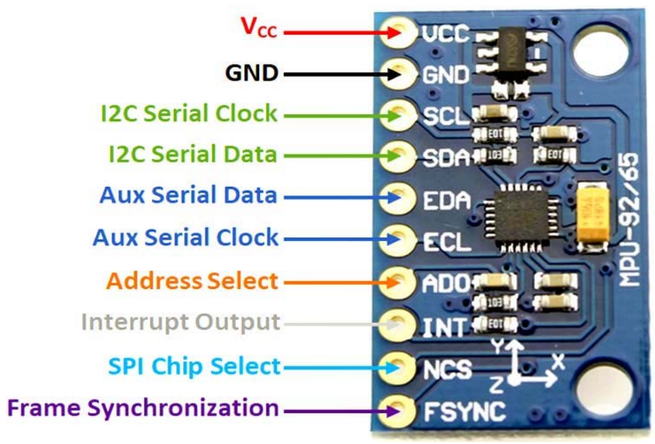


Figure 6 – MPU 9250

1.2.5.3 IBT - 2H Motor Driver

IBT – 2H is a high power module used as a motor driver. The module consists of 2 BTS 7960 chips and features logic level inputs, protection against over temperature, short circuit, over current and slew rate adjustments. The module has the following specifications:

Figure 7 – IBT 2H

- Input voltage: 6-27 V DC

Footnote may be given with Font size 10 of Times New Roman.

- Max Current: 43 A
- Control I/P Level: 3.3-5 V
- Duty Cycle: 0-100%

1.2.5.4 NEO – 6M GPS Receiver

NEO – 6M GPS module is a GPS receiver with a built in ceramic antenna. The module has a strong satellite search capabilities. One can monitor the status of the module through power and signal indicators. The module consists of built in EPROM in which configuration settings are stored. The module also has TTL level serial interface and can be connected easily to the microcontroller using TTL to USB adapter. The module generates a default NMEA data message which is used in most of the applications. The module has the following specifications:

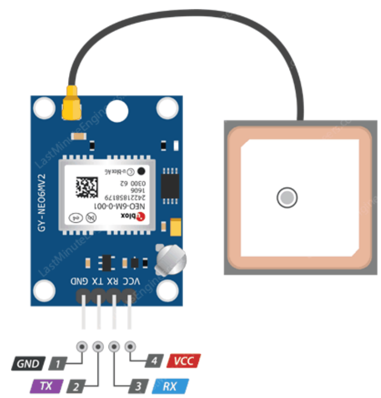


Figure 8 – NEO – 6M GPS

- Operating Voltage: 3-5V
- Positing Accuracy: 2.5 m
- Refresh Rate: 5Hz Max
- Protocol: NMEA (Default)/ UBX Binary

1.2.5.5 HC-SR04 Sensor

HC-SR04 is a 4-pin module used to detect the obstacles and calculating short range distances. The module works on the principle of ultrasonic waves having ultrasonic transmitter and receiver. The transmitter transmits the ultrasonic wave and receiver detects the reflection of the wave in case of any obstacle.

Figure 9 – HC- SR04

The sensor can be used with both microprocessor and microcontroller platforms like aurdino and raspberry pi. The module take 5v input from Vcc and and gnd pins. Current consumed by the sensor is 15mA, hence it can be directly connected on board. Trigger and Echo

Footnote may be given with Font size 10 of Times New Roman.

Pins in the sensor are I/O pins and can be connected to the microcontroller easily. For measurement and detection, trigger pin should be high for at least 10µs.

1.2.5.6 Moisture Detection Sensor

The working of the soil moisture sensor is simple. The fork-shaped probe with two exposed conductors, acts as a variable resistor (just like a potentiometer) whose resistance varies according to the water content in the soil. This resistance is inversely proportional to the soil moisture:

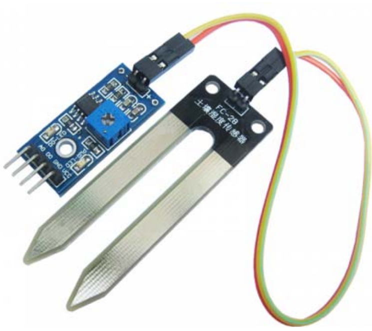


Figure 10 – Moisture Sensor

- The more water in the soil means better conductivity and will result in a lower resistance.
- The less water in the soil means poor conductivity and will result in a higher resistance.

The sensor produces an output voltage according to the resistance, which by measuring we can determine the moisture level.

1.2.5.7 IR Sensor

Widely used in motor speed detection, pulse count, the position limit, etc. This IR speed module sensor with the comparator LM393, we can calculate the speed of rotation of the wheels of our robot. If we place a ring gear that rotates attached to our wheel. It could also be used as an optical switch.

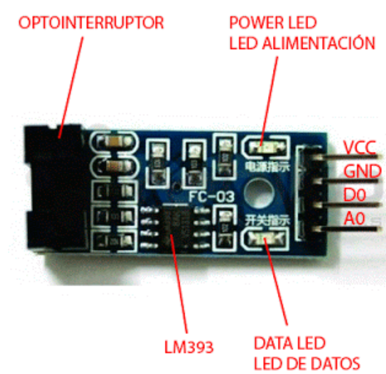


Figure 11 - IR Sensor

Footnote may be given with Font size 10 of Times New Roman.

CHAPTER 2: WORKING METHODOLOGY

The robot is equipped with various devices and sensors for high precision and analysis of environment before seed sowing. It can be operated via android app, capable of avoiding obstacles and sow the seed according to the particular requirement of the seed. All the activities of robot are performed simultaneously by different nodes of Robot Operating System. ROS is capable of coordinating and performing different tasks simultaneously. The overall operation of ROS is shown is the figure.

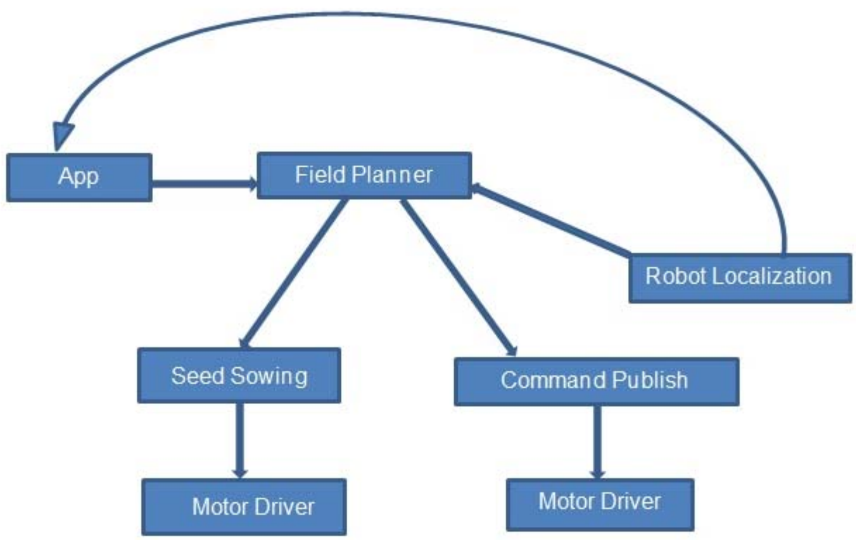


Figure 12 - ROS Operation

2.1 Robot Localization

Automatic seed sowing robot is a prototype for seed sowing. The robot works autonomously in a field by taking latitude and longitudes as inputs. On turning the robot on, the robot localize itself. Localization of robot denotes the Robot's capability to establish its own location and orientation. It is one of the fundamental concept in robotics automation as the knowledge of robot's own location is very important in making future decisions. In our robot, the localization is made possible with the use of three different types of sources. A block diagram of robot localization is shown in the figure and explained in subsequent headings.

Footnote may be given with Font size 10 of Times New Roman.

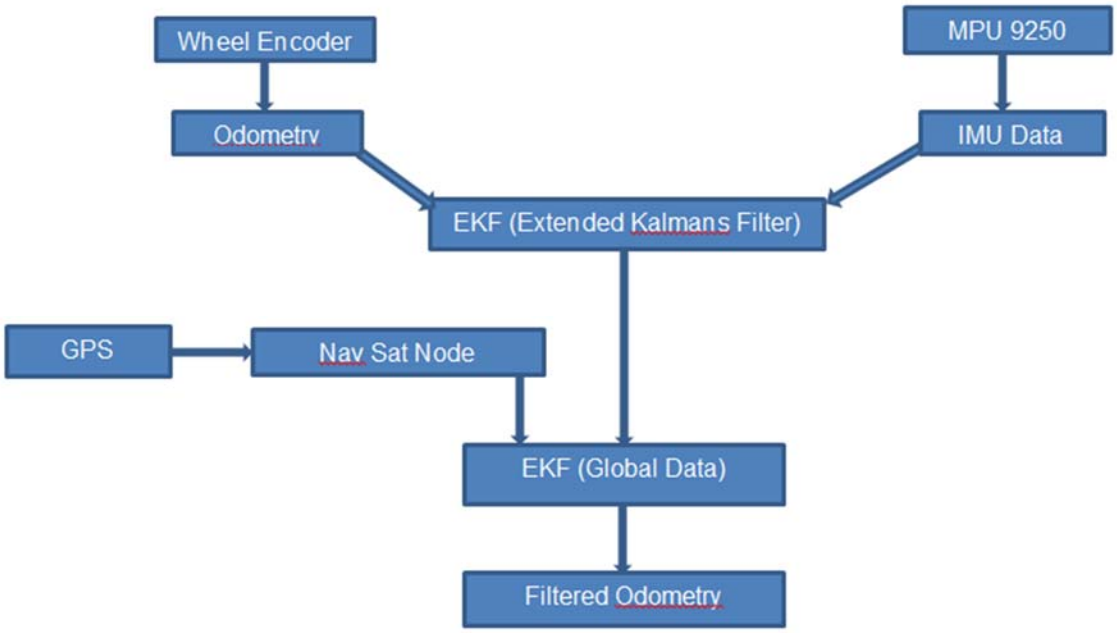


Figure 13 - Robot Localization

2.1.1 Inertial Measurement Unit

In seed sowing, continues tracking of robot's position with respect to the targeted plotted map is required along with high precision of maximum 1 meter. This technology is called SLAM, Simultaneous Localization and Mapping. The robot builds up the map of seed sowing while tracking its own position with respect to build up map. To enhance the precision of localization, we have used Inertial Measurement Unit (IMU). For this purpose, we have used MPU 9250. It is a 9-axis motion processing unit. The IMU produce output in 3-axis each about acceleration, angular velocity and magnetic strength. From this data we have calculated the Yaw, Pitch and Roll by using Madgwick Algorithm. IMU data along with Wheel encoder data are fused by using Extended Kalman Filter to obtain filtered odometery. C

2.1.1.1 Madgwick Filter

An IMU is a sensor suite complete with an accelerometer and gyroscope plus a Magnetometer. All three of these sensors measure physical qualities of Earth's fields or orientation due to angular momentum. Alone, these sensors have faults that the other sensors can make up for. The goal is to build an inertial measurement unit that will be able to sensor fuse an accelerometer, gyroscope, and optionally a magnetometer to provide roll, pitch, and yaw angles

Footnote may be given with Font size 10 of Times New Roman.

relative to the frame of Earth. The Madgwick Filter fuses the IMU data. It does this by using gradient descent to optimize a Quaternion that orients accelerometer data to a known reference of gravity. This quaternion is weighted and integrated with the gyroscope quaternion and previous orientation. This result is normalized and converted to Euler angles. Code is attached in annex **'B'.**

Gyroscope Model:

$$\omega=\hat{w}+bg+ng$$

Here, ω is the measured angular velocity from the gyro, $\hat{w}$ is the latent ideal angular velocity we wish to recover, b_g is the gyro bias which changes with time and other factors like temperature, n_g is the white gaussian gyro noise.The gyro bias is modelled as

$$\dot{b}_g = b_{bg}(t) \sim N(0,Q_g), \text{ where}$$

Q_g is the covariance matrix which models gyro noise.

Accelerometer Model:

$$a=R^T(\hat{a}-g)+b_a+n_a$$

Here, a is the measured acceleration from the accelerometer, $\hat{a}$ is the latent ideal acceleration we wish to recover, R is the orientation of the sensor in the world frame, g is the acceleration due to gravity in the world frame, b_a is the accelerometer bias which changes with time and other factors like temperature, n_a is the the white gaussian accelerometer noise.

The accelerometer bias is modelled as

$$\dot{b}_a=b_{ba}(t) \sim N(0,Q_a)$$

where Q_a is the covariance matrix which models accelerometer noise.

2.1.2 Wheel Encoder

Another technique employed to refine the precision and track the actual position of robot, we have used the wheel encoder. Wheel encoder is an IR sensor which is used to count the revolution of the wheel. As our proposed robot is working on the principle of differential drive, the number of revolution that both wheels undergo should be equal in order to drive the robot in straight line. Else if the both wheels have different velocities, the robot tends to turn in some direction. In order to evaluate the velocity and revolution of the wheel, we have installed specially designed hand-made encoder wheel with the drive gears of motors. The feedback encoder tracks how much a wheel rotates. Given that information and wheel radius, we can find out that how much a wheel has moved and how fast it is moving. The diameter of wheel is 8 cm. The 360 degrees of the circle is divided in 20 hollow segments. Every segment is of 5 degrees and is separated by 13 degrees from others. The encoder wheels rotate between the IR sensors which count the change in state of wheel. The transmitter of IR sensor transmit the infrared rays which is being received at receiver end. When IR is blocked by the wheel, it counts the change in state. This change in state is then cut into half to count only the number of hollow segments. When the number of ticks of left and right wheel are known, we can compute position and orientation of robot.

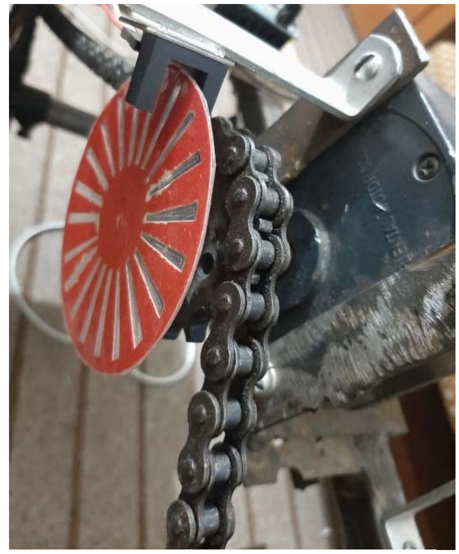


Figure 14 – Wheel Encoder

Now, we calculate how much a robot turn using wheel encoder. Let,

c= circumference of wheel= distance covered in one revolution

q= number of encoder ticks per revolution

S_{lt} = encoder tick for left wheel at time t

d_l = Distance covered by left wheel, then

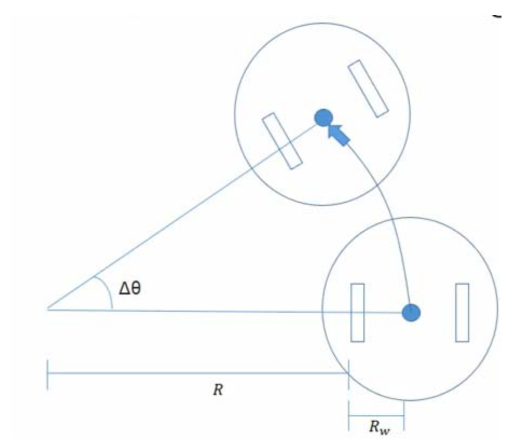


Figure 15 – Distance Calculation

$$dl = \frac{(Slt - sl(t-1)}{q} * c$$

Similarly for right wheel

$$dr = \frac{(Srt - sr(t-1)}{q} * c$$

Footnote may be given with Font size 10 of Times New Roman.

Consider the movement of robot shown in the figure. A robot turn from its initial position to the position shown in the diagram. Let,

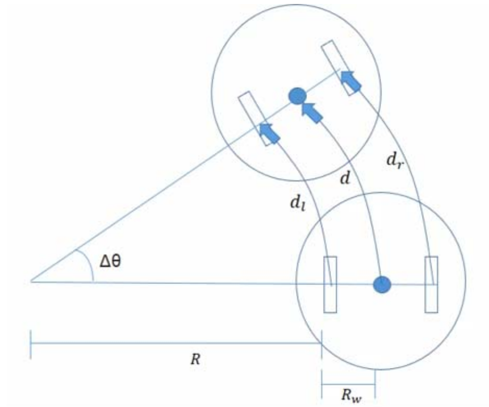


Figure 16 - Distance Calculation

Rw= Distance between wheel and centre

R = Radius of curved path

Δθ = Change in orientation

Now we calculate the distance covered by the center of robot and both wheels.

$d_l = R\Delta\theta$

$d = (R+R_w)\,\Delta\theta$

$d_r = (R+2\,R_w)\,\Delta\theta$

Now, we generally do not know the value of d and Δθ. We only know dr and dl from encoders and Rw from robot modal. So let's find out Δθ.

$dl == R\Delta\theta$

$d_r = (R+2\,R_w)\,\Delta\theta$

$\Delta\theta\, R_w = \frac{dr-R\Delta\theta}{2}$

$\Delta\theta = \frac{dr-dl}{2Rw}$

Now we have the expressions for d and Δθ, so we can find out d which comes out to be

$d = \frac{dl+dr}{2}$

Fusion of wheel encoder's data and IMU gives us the position of the robot according to the local frame of reference. Code is attached in annex **'C'.**

2.1.3 Global Positioning System

In the proposed robot, global localization is made possible with the use of GPS Receiver NEO 6M having accuracy up to 2.5 m. Each satellite transmits μwave signal towards earth. Receiver on earth use these signals to estimate their speed, location, direction and time. This information is then processed by receiver to determine the latitude, longitude, speed, altitude and time.

Footnote may be given with Font size 10 of Times New Roman.

The GPS gives the output in standard NMEA format. An example of NMEA format is as follow:

$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A

All NMEA messages start with the $ character, and each data field is separated by a comma.

Field	Meaning
0	Message ID $GPRMC
1	UTC of position fix
2	Status A=active or V=void
3	Latitude
4	Longitude
5	Speed over the ground in knots
6	Track angle in degrees (True)
7	Date
8	Magnetic variation in degrees
9	The checksum data, always begins with *

Table 1 – NMEA Message Format

The data provided by GPS is subscribed by Nav Sat Node of ROS. The output of this node is again subscribed by the ekf node along with the filtered odometry published by the fusion of IMU and Wheel encoders data. Ekf node again fuse the data from both nodes and provides the position of the robot globally. Code is attached in annex **'D'.**

2.2 Localization Data Fusion

2.2.1 Robot Operating System (ROS)

Robot operating system is a set of tools and software libraries which aims to make the complex robotic operation easy. The basic function of the ROS is to run a number of executable files simultaneously that are able to exchange data synchronously or asynchronously. For example, ROS get the data from the robot sensors at set frequency, retrieve that data, pass it to data processing and in return control the motors according to desired function.

Footnote may be given with Font size 10 of Times New Roman.

2.2.2 Node

Node is an executable which uses ROS to communicate with other nodes. Every node that start working, declare itself to Master. Nodes communicate with each other by publishing and subscribing messages. They also exchange request and response messages as a part of ROS service call. These are defined as srv files.

2.2.3 Topic

The data in ROS is called a topic. A topic defines the types of messages that will be published about that topic. The nodes that sends data publish the topic name and the type of message to be sent. The actual data is published by the node. Nodes can publish messages to a topic as well as subscribe to a topic to receive messages.

2.2.4 Robot Localization Package

It is a collection of state estimation nodes. Each node is a state estimator of the robot that is moving in a 3 dimensional space. In our proposed robot, we have used ekf (Extended Kalman's Filter) localization node for state estimation. This package also provide us nav sat node which enable us to integrate the data of GPS also. These nodes allow the user to fuse the data of multiple sensors together. Hence they do not restrict the number of inputs. Moreover, these nodes provide continues estimation. The state estimation begins with the reception of even a single measurement. If there is a delay in sensor data, the filter will continue to estimate the state through internal motion model. Every state estimator node estimates the state in 15 dimensions: $(X,Y,Z,roll,pitch,yaw,\dot{X},\dot{Y},\dot{Z},\dot{roll},\dot{pitch},\dot{yaw},\ddot{X},\ddot{Y},\ddot{Z})$.

2.2.5 Extended Kalman's Filter

The extended Kalman filter is utilized for nonlinear problems like bearing-angle target tracking and terrain-referenced navigation (TRN). The Extended Kalman Filter (EKF) is an extension of the classic Kalman Filter for non-linear systems where non-linearity are approximated using the first or second order derivative. It is based on:

- linearizing dynamics and output functions at current estimate
- propagating an approximation of the conditional expectation and covariance

Consider $\varphi : R^n \rightarrow R^m$

Suppose $E x = \bar{x}$, $E(x - \bar{x})(x - \bar{x})^T = \Sigma_x$, and $y = \varphi(x)$

If Σ_x is small, φ is not too nonlinear, $y \approx \tilde{y} = \varphi(\bar{x}) + D\varphi(\bar{x})(x - \bar{x})$

Approximation for mean and covariance of nonlinear function of random variable:

$$\bar{y} \approx \varphi(\bar{x}), \Sigma_y \approx D\varphi(\bar{x})\Sigma_x D\varphi(\bar{x})^T$$

2.3 Obstacle Avoidance

The robot is capable of avoiding the obstacle when it is en route to the field. During this stage when the robot is not in seed sowing mode, the robot uses it supersonic sensor for the detection of obstacles. The transmitters emit a high frequency ultrasonic sound, which bounce off any nearby solid objects, and the receiver listens for any return echo. That echo is then processed by the control circuit to calculate the time difference between the signal being transmitted and received. This time can subsequently be used, along with some clever math, to calculate the distance between the sensor and the reflecting object.

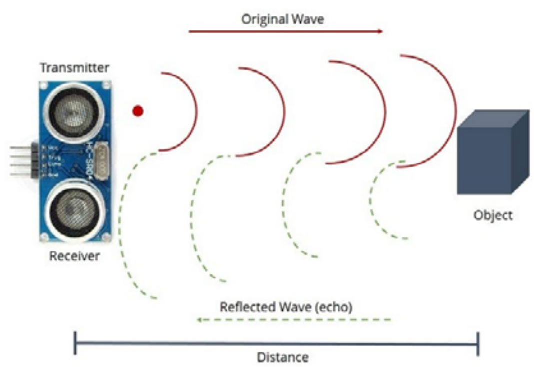


Figure 17 - Obstacle Avoidance Scheme

2.4 Robot's Movement and Control

For the coordinating and controlling robot's motion, we have designed various nodes in ROS. These nodes are performing various functions simultaneously which helps in controlling the Robot. Following nodes are being used in the ROS for functioning of our proposed robot.

ROS Node 1: This nodes subscribes to the topic of Robot localization package. It is connected with application and publishes data on ROS. This node actually transfers input data from application to ROS. Using the algorithm, the robot calculates the present and targeted location. This node further publishes the Present and targeted location continuously which can be viewed on GUI. Code is attached in annex **'E'.**

Field Planner Node: This node decides the target location of the robot. It moves the robot towards its target. During movement within field it divides the field into small sub targets and moves it in a matrix. This node publishes two type of commands, one is target and present location with bearing. This data is subscribed by the command publish node. Second is seed

Footnote may be given with Font size 10 of Times New Roman.

sowing commands which is further subscribed by seed sowing node. Code is attached in annex **'F'.**

ROS Node Command: This node subscribe to the topic of Present and targeted location from Field Planner Node. After the brief calculation as per the algorithm, it calculates the distance and velocity required to reach the target location. The distance between two points in a cartesian plane is found by following formula.

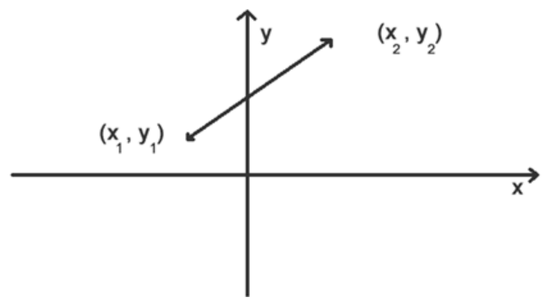


Figure 18 - Point to Point Distance Calculation

$$d = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$$

The node then publishes the velocities of the motors ranging from -1 to 1. Code is attached in annex **'G'.**

ROS Node Motor Driver: This node subscribes to the velocities calculated by the Node Command. The motors drive the robot in forward direction when velocity passed to them ranges between 0-1. In case of velocities ranging between 0 to -1, the robot will move in backward direction. Code is attached in annex **'H'**

ROS Node Seed Sowing: This node subscribe to the seed type and calculate the velocity of stepper motors that are used in the seed sowing mechanism. The velocities are then published by this node and operation of seed sowing is performed.

ROS Node Obstacle Detection: This node subscribe to the data provided by sensor HC-SR04. This is an IR Sensor for detection of the obstacles which robot is en route to the field from its starting position. In case of any obstacle, the node estimates the re-routing of the robot and publish the data accordingly.

Footnote may be given with Font size 10 of Times New Roman.

Chapter 3: Graphic User Interface and Integration with Robot

For digitized control of our robot system we have developed an android application which is integrated with the robot system. Use of this application enables us to control our robot remotely,

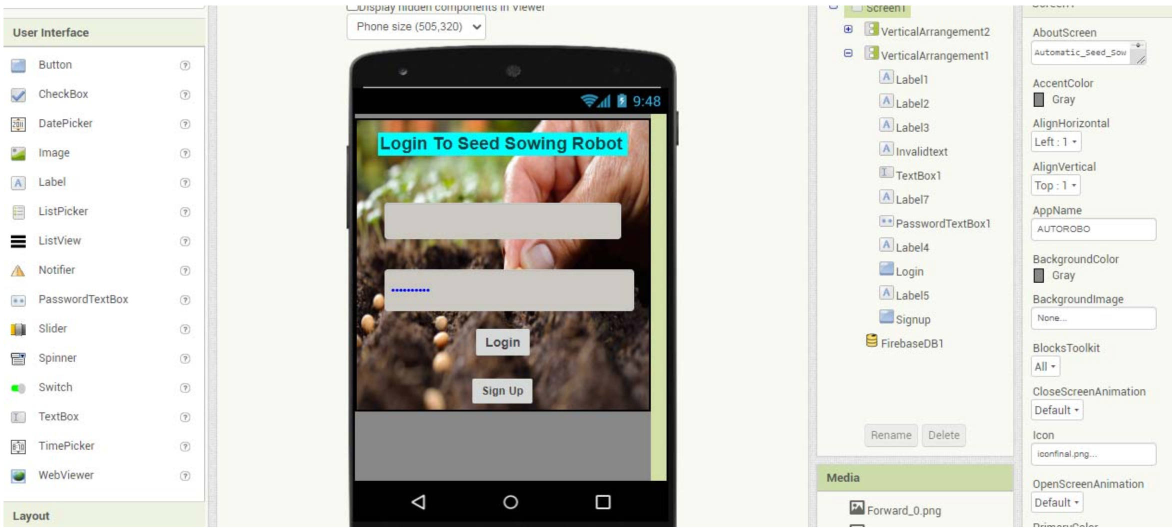


Figure 19 – MIT App Developer

get information about the robot's location and state which can be used for further decision making. It eliminate the need of the operator to be in direct contact with the robot. Operator can control the robot from the position of his own ease in a safer manner away from harmful dust or venomous reptiles. Moreover, the operation of robot from a distant location decrease the fatigue of manpower.

It requires just a click on screen in order to use its functionalities. GUI makes it very easy to be use by novice as it is user friendly. It looks very attractive and multi colored.

Figure 20 - GUI

3.1 Development of GUI

For the easy operation of the robot remotely, we have specially designed an android application. The development of this Graphical user interface and its interfacing with the robot's operation is described subsequently. We have use the GUI builder MIT App Inventor for this purpose. MIT have two

Footnote may be given with Font size 10 of Times New Roman.

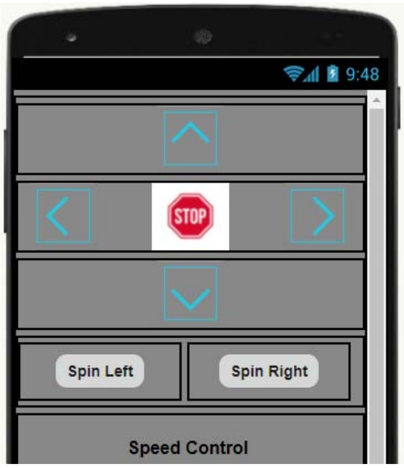


Figure 21 - GUI

different platforms. The first is the designer block. Developer design the out lay of GUI in this arrangement. The second is coding block. The coding block provides functionality to the GUI (Blocks in Annex **I**). Without a GUI builder, a GUI must be built by manually specifying each widget's parameters in source-code, with no visual feedback until the program is run. The main advantage of the app builder is that the user do not need to write extensively large codes. The user just have to design the screen graphically and the app builder will write the code for it. The user further manipulates the code according to his own requirement.

Now, we have created the application screens in different alignments for different functions with the help of labels, buttons, text boxes, clock, spinners and firebase extension. First is the login screen. The application is designed for use in mobile. Every action on the application screen is linked with the data base. The user needs to sign up initially. Afterwards, the user login to start the operation of robot. The main page of the GUI offers different option to the user as shown in the figure. The user can find present location, can operate the robot manually, sow the seed automatically, reaches to the targeted field and shows status of the robot.

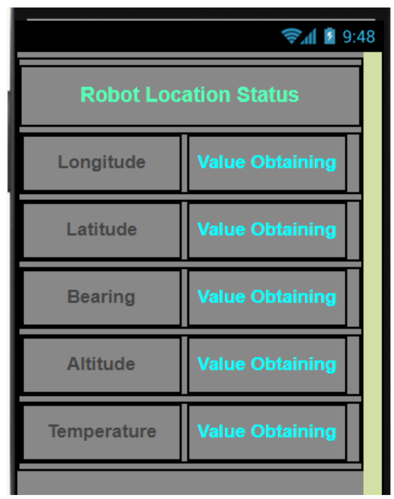


Figure 22 - GUI

Screen in figure allows the user to operate the robot manually. In case of any eventuality the user can over take the autonomous operation of the robot and control the robot on his own wish and will. Direction, rotation and speed of the robot can be controlled according to requirement.

The screen shown in the figure provides information to the user about robot's present location. The information includes latitude, longitude, bearings, altitude and temperature. The information provided by the robot is very useful in further decision making and controlling the operation of the robot.

Footnote may be given with Font size 10 of Times New Roman.

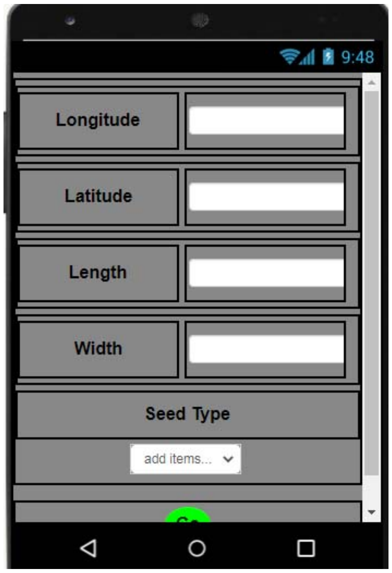


Figure 24 - GUI

The shown screen of the GUI allow user to move the robot from its initial position to the desired location of the field. This feature is introduced keeping in mind the heavy weight of the robot which restrict the user to move the robot manually. The user inputs the latitude, longitude or the x, y coordinates of the field or desired location. On entering the required inputs, the robot will reach the targets destination. In case of any obstacle, the robot will re-route itself as discussed in preceding chapter and reach the desired location

Figure shows the screen used to start the autonomous operation of the robot. The GUI takes the Latitude, longitude, length and width of the field. Seed type is also fed as an input to the GUI. The robot then plot the sowing according to the requirements of the given seed type and length and width of the field. Go button in the bottom is pressed to start the operation of the robot.

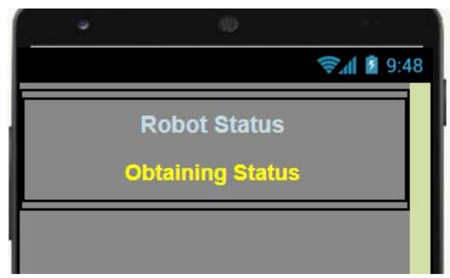


Figure 23 - GUI

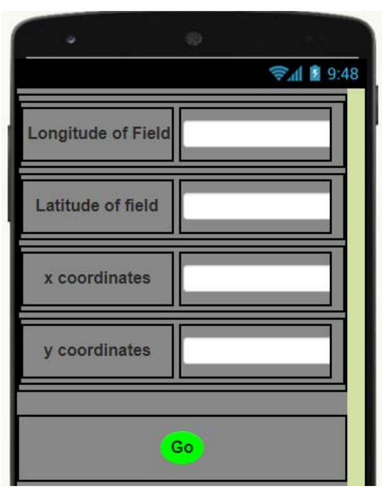


Figure 25 - GUI

The Shown robot screen shows the robot status either idle, seed sowing or approaching the desired field.

3.2 GUI – Fire Base Interface

For interfacing the GUI with robot, we have used firebase. In our GUI development, we have linked the Python 3 data with python 2. This is because the pyre base library is only

Footnote may be given with Font size 10 of Times New Roman.

compatible with python 3. Therefore data can only be get from the GUI with the use of python 3. On the other hand, ROS is only compatible with python 2. Therefore, to link the two, we have used SQL database which is a built in database of python. Firebase offers authentication, databases, real time database, storage and hosting services. It uses Real time database to store the data and implement the function in real time. Firebase use the method of authentication to link the built application in MIT with the Robot's working. The Firebase Rea time Database lets you build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, real time events continue to fire, giving the end user a responsive experience. When the device regains connection, the Real time Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

The Real time Database provides a flexible, expression-based rules language, called Firebase Real time Database Security Rules, to define how your data should be structured and when data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data, and how they can access it.

The Real time Database is a No SQL database and as such has different optimizations and functionality compared to a relational database. The Real time Database API is designed to only allow operations that can be executed quickly. This enables you to build a great real time experience that can serve millions of users without compromising on responsiveness.

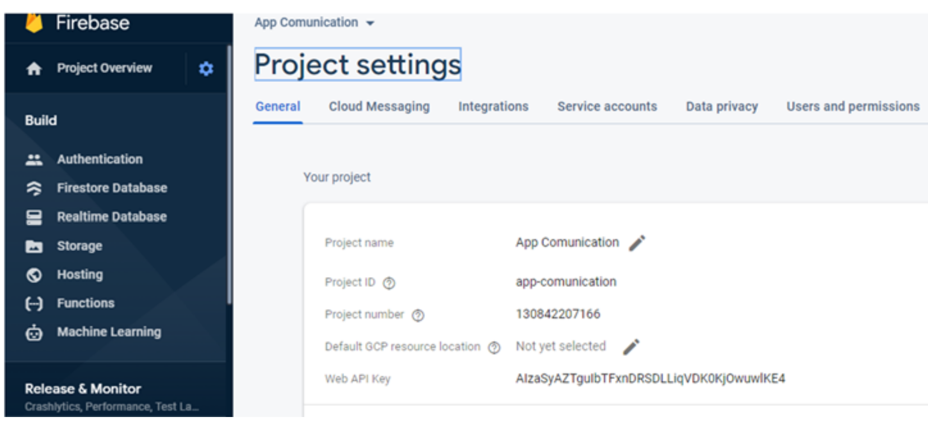


Figure 26 - GUI Authentication

Footnote may be given with Font size 10 of Times New Roman.

3.2.1 Firebase Authentication

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices. Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. The Firebase UI Auth component implements best practices for authentication on mobile devices and websites, which can maximize sign-in and sign-up conversion for your app. It also handles edge cases like account recovery and account linking that can be security sensitive and error-prone to handle correctly.

Firebase UI can be easily customized to fit in with the rest of your app's visual style, and it is open source, so you aren't constrained in realizing the user experience you want.

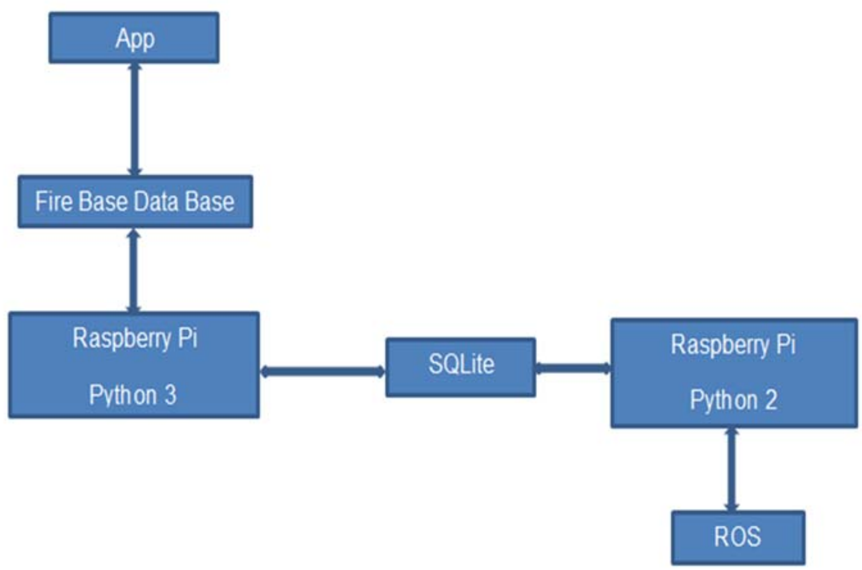


Figure 27 - GUI Working

Footnote may be given with Font size 10 of Times New Roman.

Appendices

Appendix 'A'

```
import math
import time
import numpy as np
import rospy
from sensor_msgs.msg import Imu
from MadgwickAHRS import MadgwickAHRS
from mpu9250_i2c import mpu6050_conv, AK8963_conv


def sensor_msg_imu(accl_gyro, roll, pitch, yaw):
    quaternion = euler_to_quaternion(yaw, pitch, roll)
    imu = Imu()
    imu.header.frame_id = 'base_link'
    imu.header.stamp = rospy.Time.now()
    imu.orientation.x = quaternion[0]
    imu.orientation.y = quaternion[1]
    imu.orientation.z = quaternion[2]
    imu.orientation.w = quaternion[3]
    imu.linear_acceleration.x = (accl_gyro[0] * 9.8)
    imu.linear_acceleration.y = (accl_gyro[1] * 9.8)
    imu.linear_acceleration.z = (accl_gyro[2] * 9.8)
    imu.angular_velocity.x = (accl_gyro[3] * math.pi / 180)
    imu.angular_velocity.y = (accl_gyro[4] * math.pi / 180)
    imu.angular_velocity.z = (accl_gyro[5] * math.pi / 180)
    return imu
def euler_to_quaternion(roll, pitch, yaw):
    qx = np.sin(roll / 2) * np.cos(pitch / 2) * np.cos(yaw / 2) - np.cos(roll / 2) * np.sin(pitch / 2) * np.sin(
        yaw / 2)
    qy = np.cos(roll / 2) * np.sin(pitch / 2) * np.cos(yaw / 2) + np.sin(roll / 2) * np.cos(pitch / 2) * np.sin(
        yaw / 2)
    qz = np.cos(roll / 2) * np.cos(pitch / 2) * np.sin(yaw / 2) - np.sin(roll / 2) * np.sin(pitch / 2) * np.cos(
        yaw / 2)
    qw = np.cos(roll / 2) * np.cos(pitch / 2) * np.cos(yaw / 2) + np.sin(roll / 2) * np.sin(pitch / 2) * np.sin(
        yaw / 2)
    return [qx, qy, qz, qw]


time.sleep(1)  # delay necessary to allow mpu9250 to settle
rospy.init_node('imu_data', anonymous=True)
pub = rospy.Publisher('/imu', Imu, queue_size=10)
pub.publish("Connection initiated")
euler = MadgwickAHRS()
while not rospy.is_shutdown():
    accl_gyro = mpu6050_conv()
    magno = AK8963_conv()
    euler.MadgwickAHRSupdate(
        gx=accl_gyro[3] * math.pi / 180.0,
```


Footnote may be given with Font size 10 of Times New Roman.

```
        gy=accl_gyro[4] * math.pi / 180.0,
        gz=accl_gyro[5] * math.pi / 180.0,
        ax=accl_gyro[0],
        ay=accl_gyro[1],
        az=accl_gyro[2],
        mx=magno[0] + 40,
        my=magno[1] + 40,
        mz=magno[0] + 40
    )
    euler_roll = euler.GetRoll()
    euler_pitch = euler.GetPitch()
    euler_yaw = euler.GetYaw()
    imu_data = sensor_msg_imu(accl_gyro, euler_roll, euler_pitch, euler_yaw)
    rospy.sleep(0.1)
    pub.publish(imu_data)
```

MPU 9250

```
import smbus
import time
def MPU6050_start():
    # alter sample rate (stability)
    samp_rate_div = 0  # sample rate = 8 kHz/(1+samp_rate_div)
    bus.write_byte_data(MPU6050_ADDR, SMPLRT_DIV, samp_rate_div)
    time.sleep(0.1)
    # reset all sensors
    bus.write_byte_data(MPU6050_ADDR, PWR_MGMT_1, 0x00)
    time.sleep(0.1)
    # power management and crystal settings
    bus.write_byte_data(MPU6050_ADDR, PWR_MGMT_1, 0x01)
    time.sleep(0.1)
    # Write to Configuration register
    bus.write_byte_data(MPU6050_ADDR, CONFIG, 0)
    time.sleep(0.1)
    # Write to Gyro configuration register
    gyro_config_sel = [0b00000, 0b010000, 0b10000, 0b11000]  # byte registers
    gyro_config_vals = [250.0, 500.0, 1000.0, 2000.0]  # degrees/sec
    gyro_indx = 0
    bus.write_byte_data(MPU6050_ADDR, GYRO_CONFIG, int(gyro_config_sel[gyro_indx]))
    time.sleep(0.1)
    # Write to Accel configuration register
    accel_config_sel = [0b00000, 0b01000, 0b10000, 0b11000]  # byte registers
    accel_config_vals = [2.0, 4.0, 8.0, 16.0]  # g (g = 9.81 m/s^2)
    accel_indx = 0
    bus.write_byte_data(MPU6050_ADDR, ACCEL_CONFIG, int(accel_config_sel[accel_indx]))
    time.sleep(0.1)
    # interrupt register (related to overflow of data [FIFO])
    bus.write_byte_data(MPU6050_ADDR, INT_ENABLE, 1)
    time.sleep(0.1)
    return gyro_config_vals[gyro_indx], accel_config_vals[accel_indx]
```


Footnote may be given with Font size 10 of Times New Roman.

```
def read_raw_bits(register):
    # read accel and gyro values
    high = bus.read_byte_data(MPU6050_ADDR, register)
    low = bus.read_byte_data(MPU6050_ADDR, register + 1)

    # combine higha and low for unsigned bit value
    value = ((high << 8) | low)

    # convert to +- value
    if (value > 32768):
        value -= 65536
    return value


def mpu6050_conv():
    # raw acceleration bits
    acc_x = read_raw_bits(ACCEL_XOUT_H)
    acc_y = read_raw_bits(ACCEL_YOUT_H)
    acc_z = read_raw_bits(ACCEL_ZOUT_H)

    # raw temp bits
##    t_val = read_raw_bits(TEMP_OUT_H) # uncomment to read temp

    # raw gyroscope bits
    gyro_x = read_raw_bits(GYRO_XOUT_H)
    gyro_y = read_raw_bits(GYRO_YOUT_H)
    gyro_z = read_raw_bits(GYRO_ZOUT_H)

    # convert to acceleration in g and gyro dps
    a_x = (acc_x / (2.0 ** 15.0)) * accel_sens
    a_y = (acc_y / (2.0 ** 15.0)) * accel_sens
    a_z = (acc_z / (2.0 ** 15.0)) * accel_sens

    w_x = (gyro_x / (2.0 ** 15.0)) * gyro_sens
    w_y = (gyro_y / (2.0 ** 15.0)) * gyro_sens
    w_z = (gyro_z / (2.0 ** 15.0)) * gyro_sens

##    temp = ((t_val)/333.87)+21.0 # uncomment and add below in return
    return a_x, a_y, a_z, w_x, w_y, w_z


def AK8963_start():
    bus.write_byte_data(AK8963_ADDR, AK8963_CNTL, 0x00)
    time.sleep(0.1)
    AK8963_bit_res = 0b0001  # 0b0001 = 16-bit
    AK8963_samp_rate = 0b0110  # 0b0010 = 8 Hz, 0b0110 = 100 Hz
    AK8963_mode = (AK8963_bit_res << 4) + AK8963_samp_rate  # bit conversion
    bus.write_byte_data(AK8963_ADDR, AK8963_CNTL, AK8963_mode)
    time.sleep(0.1)


def AK8963_reader(register):
    # read magnetometer values
    low = bus.read_byte_data(AK8963_ADDR, register - 1)
    high = bus.read_byte_data(AK8963_ADDR, register)
    # combine higha and low for unsigned bit value
```


Footnote may be given with Font size 10 of Times New Roman.

```
    value = ((high << 8) | low)
    # convert to +- value
    if (value > 32768):
        value -= 65536
    return value


def AK8963_conv():
    # raw magnetometer bits

    loop_count = 0
    while 1:
        mag_x = AK8963_reader(HXH)
        mag_y = AK8963_reader(HYH)
        mag_z = AK8963_reader(HZH)

        # the next line is needed for AK8963
        if bin(bus.read_byte_data(AK8963_ADDR, AK8963_ST2)) == '0b10000':
            break
        loop_count += 1

    # convert to acceleration in g and gyro dps
    m_x = (mag_x / (2.0 ** 15.0)) * mag_sens
    m_y = (mag_y / (2.0 ** 15.0)) * mag_sens
    m_z = (mag_z / (2.0 ** 15.0)) * mag_sens

    return m_x, m_y, m_z


# MPU6050 Registers
MPU6050_ADDR = 0x68
PWR_MGMT_1 = 0x6B
SMPLRT_DIV = 0x19
CONFIG = 0x1A
GYRO_CONFIG = 0x1B
ACCEL_CONFIG = 0x1C
INT_ENABLE = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
TEMP_OUT_H = 0x41
GYRO_XOUT_H = 0x43
GYRO_YOUT_H = 0x45
GYRO_ZOUT_H = 0x47
# AK8963 registers
AK8963_ADDR = 0x0C
AK8963_ST1 = 0x02
HXH = 0x04
HYH = 0x06
HZH = 0x08
AK8963_ST2 = 0x09
AK8963_CNTL = 0x0A
mag_sens = 4900.0  # magnetometer sensitivity: 4800 uT

# start I2C driver
bus = smbus.SMBus(1)  # start comm with i2c bus
```


Footnote may be given with Font size 10 of Times New Roman.

```
gyro_sens, accel_sens = MPU6050_start()  # instantiate gyro/accel
AK8963_start()  # instantiate magnetometer
```


Footnote may be given with Font size 10 of Times New Roman.

Appendix 'B'

```
import math
def invSqrt(x):
    import struct
    i = struct.unpack('>i', struct.pack('>f', x))[0]
    i = 0x5f3759df - (i >> 1)
    y = struct.unpack('>f', struct.pack('>i', i))[0]
    return y * (1.5 - 0.5 * x * y * y)


class MadgwickAHRS:

    def __init__(self, quaternion=None, beta=None):
        self.sampleFreq = 3.0  # sample frequency in Hz
        self.beta = 0.1  # 2 * proportional gain
        if quaternion is not None:
            self.quaternion = quaternion
        if beta is not None:
            self.beta = beta
        self.q0 = 1.0
        self.q1 = 0.0
        self.q2 = 0.0
        self.q3 = 0.0

    def MadgwickAHRSupdate(self, gx, gy, gz, ax, ay, az, mx, my, mz):
        # Use IMU algorithm if magnetometer measurement invalid (avoids NaN in magnetometer
normalisation)
        # if((mx == 0.0f) && (my == 0.0f) && (mz == 0.0f)) {
        #    MadgwickAHRSupdateIMU(gx, gy, gz, ax, ay, az)
        #    return
        # }
        q0 = self.q0
        q1 = self.q1
        q2 = self.q2
        q3 = self.q3
        sampleFreq = self.sampleFreq
        beta = self.beta
        # Rate of change of quaternion from gyroscope
        qDot1 = 0.5 * (-q1 * gx - q2 * gy - q3 * gz)
        qDot2 = 0.5 * (q0 * gx + q2 * gz - q3 * gy)
        qDot3 = 0.5 * (q0 * gy - q1 * gz + q3 * gx)
        qDot4 = 0.5 * (q0 * gz + q1 * gy - q2 * gx)

        # Compute feedback only if accelerometer measurement valid (avoids NaN in accelerometer
normalisation)
        if ((ax != 0.0) and (ay != 0.0) and (az != 0.0)):
            # Normalise accelerometer measurement
            recipNorm = invSqrt(ax * ax + ay * ay + az * az)
            ax *= recipNorm
            ay *= recipNorm
            az *= recipNorm

            # Normalise magnetometer measurement
            recipNorm = invSqrt(mx * mx + my * my + mz * mz)
```


Footnote may be given with Font size 10 of Times New Roman.

```
mx *= recipNorm
my *= recipNorm
mz *= recipNorm

# Auxiliary variables to avoid repeated arithmetic
_2q0mx = 2.0 * q0 * mx
_2q0my = 2.0 * q0 * my
_2q0mz = 2.0 * q0 * mz
_2q1mx = 2.0 * q1 * mx
_2q0 = 2.0 * q0
_2q1 = 2.0 * q1
_2q2 = 2.0 * q2
_2q3 = 2.0 * q3
_2q0q2 = 2.0 * q0 * q2
_2q2q3 = 2.0 * q2 * q3
q0q0 = q0 * q0
q0q1 = q0 * q1
q0q2 = q0 * q2
q0q3 = q0 * q3
q1q1 = q1 * q1
q1q2 = q1 * q2
q1q3 = q1 * q3
q2q2 = q2 * q2
q2q3 = q2 * q3
q3q3 = q3 * q3

# Reference direction of Earth's magnetic field
hx = mx * q0q0 - _2q0my * q3 + _2q0mz * q2 + mx * q1q1 + _2q1 * my * q2 + _2q1 * mz * q3 -
mx * q2q2 - mx * q3q3
hy = _2q0mx * q3 + my * q0q0 - _2q0mz * q1 + _2q1mx * q2 - my * q1q1 + my * q2q2 + _2q2 *
mz * q3 - my * q3q3
_2bx = math.sqrt(hx * hx + hy * hy)
_2bz = -_2q0mx * q2 + _2q0my * q1 + mz * q0q0 + _2q1mx * q3 - mz * q1q1 + _2q2 * my * q3 -
mz * q2q2 + mz * q3q3
_4bx = 2.0 * _2bx
_4bz = 2.0 * _2bz

# Gradient decent algorithm corrective step
s0 = -_2q2 * (2.0 * q1q3 - _2q0q2 - ax) + _2q1 * (2.0 * q0q1 + _2q2q3 - ay) - _2bz * q2 * (
        _2bx * (0.5 - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (-_2bx * q3 + _2bz * q1) * (
            _2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + _2bx * q2 * (
            _2bx * (q0q2 + q1q3) + _2bz * (0.5 - q1q1 - q2q2) - mz)
s1 = _2q3 * (2.0 * q1q3 - _2q0q2 - ax) + _2q0 * (2.0 * q0q1 + _2q2q3 - ay) - 4.0 * q1 * (
        1 - 2.0 * q1q1 - 2.0 * q2q2 - az) + _2bz * q3 * (
            _2bx * (0.5 - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (_2bx * q2 + _2bz * q0) * (
            _2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + (_2bx * q3 - _4bz * q1) * (
            _2bx * (q0q2 + q1q3) + _2bz * (0.5 - q1q1 - q2q2) - mz)
s2 = -_2q0 * (2.0 * q1q3 - _2q0q2 - ax) + _2q3 * (2.0 * q0q1 + _2q2q3 - ay) - 4.0 * q2 * (
        1 - 2.0 * q1q1 - 2.0 * q2q2 - az) + (-_4bx * q2 - _2bz * q0) * (
            _2bx * (0.5 - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (_2bx * q1 + _2bz * q3) * (
            _2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + (_2bx * q0 - _4bz * q2) * (
            _2bx * (q0q2 + q1q3) + _2bz * (0.5 - q1q1 - q2q2) - mz)
s3 = _2q1 * (2.0 * q1q3 - _2q0q2 - ax) + _2q2 * (2.0 * q0q1 + _2q2q3 - ay) + (-_4bx * q3 + _2bz *
q1) * (
        _2bx * (0.5 - q2q2 - q3q3) + _2bz * (q1q3 - q0q2) - mx) + (-_2bx * q0 + _2bz * q2) * (
            _2bx * (q1q2 - q0q3) + _2bz * (q0q1 + q2q3) - my) + _2bx * q1 * (
```


Footnote may be given with Font size 10 of Times New Roman.

```
                    _2bx * (q0q2 + q1q3) + _2bz * (0.5 - q1q1 - q2q2) - mz)
            recipNorm = invSqrt(s0 * s0 + s1 * s1 + s2 * s2 + s3 * s3)  # normalise step magnitude
            s0 *= recipNorm
            s1 *= recipNorm
            s2 *= recipNorm
            s3 *= recipNorm

            # Apply feedback step
            qDot1 -= beta * s0
            qDot2 -= beta * s1
            qDot3 -= beta * s2
            qDot4 -= beta * s3

        # Integrate rate of change of quaternion to yield quaternion
        q0 += qDot1 * (1.0 / sampleFreq)
        q1 += qDot2 * (1.0 / sampleFreq)
        q2 += qDot3 * (1.0 / sampleFreq)
        q3 += qDot4 * (1.0 / sampleFreq)

        # Normalise quaternion
        recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3)
        q0 *= recipNorm
        q1 *= recipNorm
        q2 *= recipNorm
        q3 *= recipNorm

        self.q0 = q0
        self.q1 = q1
        self.q2 = q2
        self.q3 = q3

    def MadgwickAHRSupdateIMU(self, gx, gy, gz, ax, ay, az):
        q0 = self.q0
        q1 = self.q1
        q2 = self.q2
        q3 = self.q3
        sampleFreq = self.sampleFreq
        beta = self.beta
        # Rate of change of quaternion from gyroscope
        qDot1 = 0.5 * (-q1 * gx - q2 * gy - q3 * gz)
        qDot2 = 0.5 * (q0 * gx + q2 * gz - q3 * gy)
        qDot3 = 0.5 * (q0 * gy - q1 * gz + q3 * gx)
        qDot4 = 0.5 * (q0 * gz + q1 * gy - q2 * gx)
        if ((ax != 0.0) and (ay != 0.0) and (az != 0.0)):
            # Normalise accelerometer measurement
            recipNorm = invSqrt(ax * ax + ay * ay + az * az)
            ax *= recipNorm
            ay *= recipNorm
            az *= recipNorm

            # Auxiliary variables to avoid repeated arithmetic
            _2q0 = 2.0 * q0
            _2q1 = 2.0 * q1
            _2q2 = 2.0 * q2
            _2q3 = 2.0 * q3
            _4q0 = 4.0 * q0
```


Footnote may be given with Font size 10 of Times New Roman.

```
            _4q1 = 4.0 * q1
            _4q2 = 4.0 * q2
            _8q1 = 8.0 * q1
            _8q2 = 8.0 * q2
            q0q0 = q0 * q0
            q1q1 = q1 * q1
            q2q2 = q2 * q2
            q3q3 = q3 * q3

            # Gradient decent algorithm corrective step
            s0 = _4q0 * q2q2 + _2q2 * ax + _4q0 * q1q1 - _2q1 * ay
            s1 = _4q1 * q3q3 - _2q3 * ax + 4.0 * q0q0 * q1 - _2q0 * ay - _4q1 + _8q1 * q1q1 + _8q1 * q2q2 +
_4q1 * az
            s2 = 4.0 * q0q0 * q2 + _2q0 * ax + _4q2 * q3q3 - _2q3 * ay - _4q2 + _8q2 * q1q1 + _8q2 * q2q2 +
_4q2 * az
            s3 = 4.0 * q1q1 * q3 - _2q1 * ax + 4.0 * q2q2 * q3 - _2q2 * ay
            recipNorm = invSqrt(s0 * s0 + s1 * s1 + s2 * s2 + s3 * s3)  # normalise step magnitude
            s0 *= recipNorm
            s1 *= recipNorm
            s2 *= recipNorm
            s3 *= recipNorm

            # Apply feedback step
            qDot1 -= beta * s0
            qDot2 -= beta * s1
            qDot3 -= beta * s2
            qDot4 -= beta * s3

        # Integrate rate of change of quaternion to yield quaternion
        q0 += qDot1 * (1.0 / sampleFreq)
        q1 += qDot2 * (1.0 / sampleFreq)
        q2 += qDot3 * (1.0 / sampleFreq)
        q3 += qDot4 * (1.0 / sampleFreq)

        # Normalise quaternion
        recipNorm = invSqrt(q0 * q0 + q1 * q1 + q2 * q2 + q3 * q3)
        q0 *= recipNorm
        q1 *= recipNorm
        q2 *= recipNorm
        q3 *= recipNorm

        self.q0 = q0
        self.q1 = q1
        self.q2 = q2
        self.q3 = q3

    def GetRoll(self):
        q0 = self.q0
        q1 = self.q1
        q2 = self.q2
        q3 = self.q3
        roll = math.atan2(q0 * q1 + q2 * q3, 0.5 - q1 * q1 - q2 * q2)
        roll = roll * 57.29578
        return roll

    def GetPitch(self):
```


Footnote may be given with Font size 10 of Times New Roman.

```
        q0 = self.q0
        q1 = self.q1
        q2 = self.q2
        q3 = self.q3
        pitch = math.asin(-2.0 * (q1 * q3 - q0 * q2))
        pitch = pitch * 57.29578
        return pitch

    def GetYaw(self):
        q0 = self.q0
        q1 = self.q1
        q2 = self.q2
        q3 = self.q3
        yaw = math.atan2(q1 * q2 + q0 * q3, 0.5 - q2 * q2 - q3 * q3)
        yaw = yaw * 57.29578 + 180.0
        return yaw
```


Footnote may be given with Font size 10 of Times New Roman.

Appendix 'C'

Encoder Node

```
import rospy
from std_msgs.msg import Int16, String
from encoder_tick import encoder_ticks



def callback(data):
motor_direction = data.data.split('/')
ticks.update_motor_direction(motor_direction[0], motor_direction[1])



rospy.init_node('encoder', anonymous=True)
pub_l_wheel = rospy.Publisher('/lwheel', Int16, queue_size=10)
pub_r_wheel = rospy.Publisher('/rwheel', Int16, queue_size=10)
ticks = encoder_ticks(0, 0)
```


Footnote may be given with Font size 10 of Times New Roman.

```
if __name__ == '__main__':

""" main """

try:

while not rospy.is_shutdown():

pos_sub = rospy.Subscriber('motor_feedback', String, callback)

rospy.sleep(0.1)

l_wheel_ticks, r_wheel_ticks = ticks.get_value()

print(ticks.get_value())

l_wheel_ticks = Int16(l_wheel_ticks)

r_wheel_ticks = Int16(r_wheel_ticks)

pub_l_wheel.publish(l_wheel_ticks)

pub_r_wheel.publish(r_wheel_ticks)

rospy.sleep(0.5)

except rospy.ROSInterruptException:

pass
```


Footnote may be given with Font size 10 of Times New Roman.

Encoder Ticks

```
import time

from RPi import GPIO

class encoder_ticks():

def __init__(self, left_motor, right_motor):

self.left_encoder = 11

self.right_encoder = 13

self.left_counter = 0

self.right_counter = 0

self.left_motor_direction = left_motor

self.right_motor_direction = right_motor

self.last_state = 0

GPIO.setmode(GPIO.BOARD)
```


Footnote may be given with Font size 10 of Times New Roman.

```
GPIO.setup(self.left_encoder, GPIO.IN,
pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(self.right_encoder, GPIO.IN,
pull_up_down=GPIO.PUD_DOWN)

GPIO.add_event_detect(self.left_encoder, GPIO.BOTH,
callback=self.my_callback)

GPIO.add_event_detect(self.right_encoder, GPIO.BOTH,
callback=self.my_callback1)


def my_callback(self, channel):

if GPIO.input(self.left_encoder) != self.last_state:

if self.left_motor_direction == '1':

self.left_counter += 1

elif self.left_motor_direction == '-1':

self.left_counter -= 1


def my_callback1(self, channel):

if GPIO.input(self.right_encoder) != self.last_state:

if self.right_motor_direction == '1':
```


Footnote may be given with Font size 10 of Times New Roman.

self.right_counter += 1

elif self.right_motor_direction == '-1':

self.right_counter -= 1

def get_value(self):

return self.left_counter, self.right_counter

def reset(self):

self.left_counter = 0

self.right_counter = 0

def update_motor_direction(self, left_motor, right_motor):

self.left_motor_direction = left_motor

self.right_motor_direction = right_motor

Wheel Odometry

```
#!/usr/bin/env python

import rospy
from math import sin, cos, pi
from geometry_msgs.msg import Quaternion
from geometry_msgs.msg import Twist
```


Footnote may be given with Font size 10 of Times New Roman.

```
from nav_msgs.msg import Odometry
from tf.broadcaster import TransformBroadcaster
from std_msgs.msg import Int16

class DiffTf:

    def __init__(self):
        rospy.init_node("raw_odometry")
        self.nodename = rospy.get_name()
        rospy.loginfo("-I- %s started" % self.nodename)

        #### parameters #######
        self.rate = rospy.get_param('~rate', 10.0)  # the rate at which to publish the transform
        self.ticks_meter = float(
            rospy.get_param('ticks_meter', 72))  # The number of wheel encoder ticks per meter of travel
        self.base_width = float(rospy.get_param('~base_width', 0.53))  # The wheel base width in meters

        self.base_frame_id = rospy.get_param('~base_frame_id', 'base_link')  # the name of the base frame of the robot
        self.odom_frame_id = rospy.get_param('~odom_frame_id', 'odom')  # the name of the odometry reference
frame

        self.encoder_min = rospy.get_param('encoder_min', -32768)
        self.encoder_max = rospy.get_param('encoder_max', 32768)
        self.encoder_low_wrap = rospy.get_param('wheel_low_wrap',
                                    (self.encoder_max - self.encoder_min) * 0.3 + self.encoder_min)
        self.encoder_high_wrap = rospy.get_param('wheel_high_wrap',
                                    (self.encoder_max - self.encoder_min) * 0.7 + self.encoder_min)

        self.t_delta = rospy.Duration(1.0 / self.rate)
        self.t_next = rospy.Time.now() + self.t_delta

        # internal data
        self.enc_left = None  # wheel encoder readings
        self.enc_right = None
        self.left = 0  # actual values coming back from robot
        self.right = 0
        self.lmult = 0
        self.rmult = 0
        self.prev_lencoder = 0
        self.prev_rencoder = 0
        self.x = 0  # position in xy plane
        self.y = 0
        self.th = 0
        self.dx = 0  # speeds in x/rotation
        self.dr = 0
        self.then = rospy.Time.now()

        # subscriptions
        rospy.Subscriber("lwheel", Int16, self.lwheelCallback)
        rospy.Subscriber("rwheel", Int16, self.rwheelCallback)
        self.odomPub = rospy.Publisher("odom", Odometry, queue_size=10)
        self.odomBroadcaster = TransformBroadcaster()

    def spin(self):
        r = rospy.Rate(self.rate)
        while not rospy.is_shutdown():
```


Footnote may be given with Font size 10 of Times New Roman.

```
        self.update()
        r.sleep()

    def update(self):
        now = rospy.Time.now()
        if now > self.t_next:
            elapsed = now - self.then
            self.then = now
            elapsed = elapsed.to_sec()

            # calculate odometry
            if self.enc_left == None:
                d_left = 0
                d_right = 0
            else:
                d_left = (self.left - self.enc_left) / self.ticks_meter
                d_right = (self.right - self.enc_right) / self.ticks_meter
            self.enc_left = self.left
            self.enc_right = self.right

            # distance traveled is the average of the two wheels
            d = (d_left + d_right) / 2
            # this approximation works (in radians) for small angles
            th = (d_right - d_left) / self.base_width
            # calculate velocities
            self.dx = d / elapsed
            self.dr = th / elapsed

            if (d != 0):
                # calculate distance traveled in x and y
                x = cos(th) * d
                y = -sin(th) * d
                # calculate the final position of the robot
                self.x = self.x + (cos(self.th) * x - sin(self.th) * y)
                self.y = self.y + (sin(self.th) * x + cos(self.th) * y)
            if (th != 0):
                self.th = self.th + th

            # publish the odom information
            quaternion = Quaternion()
            quaternion.x = 0.0
            quaternion.y = 0.0
            quaternion.z = sin(self.th / 2)
            quaternion.w = cos(self.th / 2)
            self.odomBroadcaster.sendTransform(
                (self.x, self.y, 0),
                (quaternion.x, quaternion.y, quaternion.z, quaternion.w),
                rospy.Time.now(),
                self.base_frame_id,
                self.odom_frame_id
            )

            odom = Odometry()
            odom.header.stamp = now
            odom.header.frame_id = self.odom_frame_id
            odom.pose.pose.position.x = self.x
```


Footnote may be given with Font size 10 of Times New Roman.

```
        odom.pose.pose.position.y = self.y
        odom.pose.pose.position.z = 0
        odom.pose.pose.orientation = quaternion
        odom.child_frame_id = self.base_frame_id
        odom.twist.twist.linear.x = self.dx
        odom.twist.twist.linear.y = 0
        odom.twist.twist.angular.z = self.dr
        self.odomPub.publish(odom)

    def lwheelCallback(self, msg):
        enc = msg.data
        if (enc < self.encoder_low_wrap and self.prev_lencoder > self.encoder_high_wrap):
            self.lmult = self.lmult + 1

        if (enc > self.encoder_high_wrap and self.prev_lencoder < self.encoder_low_wrap):
            self.lmult = self.lmult - 1

        self.left = 1.0 * (enc + self.lmult * (self.encoder_max - self.encoder_min))
        self.prev_lencoder = enc

    def rwheelCallback(self, msg):
        enc = msg.data
        if (enc < self.encoder_low_wrap and self.prev_rencoder > self.encoder_high_wrap):
            self.rmult = self.rmult + 1

        if (enc > self.encoder_high_wrap and self.prev_rencoder < self.encoder_low_wrap):
            self.rmult = self.rmult - 1

        self.right = 1.0 * (enc + self.rmult * (self.encoder_max - self.encoder_min))
        self.prev_rencoder = enc


if __name__ == '__main__':
    """ main """
    try:
        diffTf = DiffTf()
        diffTf.spin()
    except rospy.ROSInterruptException:
        pass
```


Footnote may be given with Font size 10 of Times New Roman.

Appendix 'D'

```
import serial
import time
import string
import pynmea2
import rospy
from sensor_msgs.msg import NavSatFix

rospy.init_node('GPS', anonymous=True)
pub1 = rospy.Publisher('/sensor_msg/NavSatFix', NavSatFix, queue_size=5)
port = "/dev/ttyAMA0"
while not rospy.is_shutdown():
    ser = serial.Serial(port, baudrate=9600, timeout=0.5)
    dataout = pynmea2.NMEAStreamReader()
    newdata = ser.readline()

    if newdata[0:6] == "$GPRMC":
        newmsg = pynmea2.parse(newdata)
        lat = newmsg.latitude
        lng = newmsg.longitude
        gps = "Latitude=" + str(lat) + "and Longitude=" + str(lng)
        print(gps)
        gpsmsg = NavSatFix()
        gpsmsg.header.stamp = rospy.Time.now()
        gpsmsg.header.frame_id = "gps"
        gpsmsg.latitude = lat
        gpsmsg.longitude = lng
        pub1.publish(gpsmsg)
```


Footnote may be given with Font size 10 of Times New Roman.

Appendix 'E'

ROS Node 1

```
import rospy
from std_msgs.msg import String

from sq_update import sq_update_down

sql1 = sq_update_down()
global status
status = 0
rospy.init_node('app_link', anonymous=True)
pub1 = rospy.Publisher('/field_work', String, queue_size=5)


def set_to_tgt(x, y):
    goal = '{}/{:.2f}/{:.2f}/Nil/{:.2f}/{:.2f}'.format(0, 0.0, 0.0, x, y)
    return goal


def set_to_fd(x, y, l, w, s):
    goal = '{}/{:.2f}/{:.2f}/{}/{:.2f}/{:.2f}'.format(1, l, w, s, x, y)
    return goal


# SLite communication methods
def update_current_loc_sq(la, lo, brg, alt, tem):
    sql1.update('PresentLocationLat', 'Present lat', la)
    sql1.update('PresentLocationLong', 'Present Long', lo)
    sql1.update('PresentLocationB', 'Present Bearing', brg)
    sql1.update('PresentLocationA', 'Present Altitude', alt)
    sql1.update('PresentLocationT', 'Present Temp', tem)


def update_status_sq(tgt_la, tgt_lo, disp, seed, area_c, area_r, approx_t):
    sql1.update('Robot Status', 'Status', status)
    if status == 1:
        sql1.update('Robot Status tgt Loc', 'Target Latitude', tgt_la)
        sql1.update('Robot Status tgt Loc', 'Target Longitude', tgt_lo)
        sql1.update('Robot Status tgt Loc', 'Displacement', disp)
    elif status == 2:
        sql1.update('Robot Status to fd', 'Field Latitude', tgt_la)
        sql1.update('Robot Status to fd', 'Field Longitude', tgt_lo)
        sql1.update('Robot Status to fd', 'Displacement', disp)
    elif status == 3:
        sql1.update('Robot Status in fd', 'Seed Sown', seed)
        sql1.update('Robot Status in fd', 'Area Covered', area_c)
        sql1.update('Robot Status in fd', 'Area Remaining', area_r)
        sql1.update('Robot Status in fd', 'Approx time', approx_t)


def get_tgt_loc_sq():
```


Footnote may be given with Font size 10 of Times New Roman.

```
    x_raw = sql1.read('Target_Locationx', 'x axis in field')
    x = float(x_raw[0])
    y_raw = sql1.read('Target_Locationy', 'y axis in field')
    y = float(y_raw[0])
    if x == 0 and y == 0:
        x_raw = sql1.read('Target_Locationlat', 'Latitude in field')
        x = float(x_raw[0])
        y_raw = sql1.read('Target_LocationLong', 'Longitude in field')
        y = float(y_raw[0])
    return x, y


def get_seed_loc_sq():
    x_raw = sql1.read('Seed_Typelat', 'Latitude')
    x = float(x_raw[0])
    y_raw = sql1.read('Seed_Typelong', 'Longitude')
    y = float(y_raw[0])
    len_raw = sql1.read('Seed_TypeLen', 'Length')
    len = float(len_raw[0])
    width_raw = sql1.read('Seed_Typew', 'Width')
    width = float(width_raw[0])
    seed = sql1.read('Seed_Type_s', 'Seed')
    return x, y, len, width, seed[0]


while not rospy.is_shutdown():
    # get present location data from ROS and update in database
    lat = 2
    lon = 3
    bearing = 4
    altitude = 507
    temp = 32
    update_current_loc_sq(lat, lon, bearing, altitude, temp)

    # get status from ROS and update in database
    tgt_lat = 5
    tgt_lon = 6
    displace = 7
    no_of_seed = 8
    area_cov = 119
    area_rem = 678
    approx_time = 76
    update_status_sq(tgt_lat, tgt_lon, displace, no_of_seed, area_cov, area_rem, approx_time)

    # get target from app through database and send it to ROS
    target = sql1.read('Target_Locationgo', 'go')
    target_flag = float(target[0])
    if target_flag == 1:
        x_tgt, y_tgt = get_tgt_loc_sq()
        print(x_tgt, y_tgt)
        # publish tgt on ROS
        work = set_to_tgt(x_tgt, y_tgt)
        pub1.publish(work)
    field = sql1.read('Seed_Typego', 'go')
    field_flag = float(field[0])
    if field_flag == 1:
```


Footnote may be given with Font size 10 of Times New Roman.

```
        x_fd, y_fd, len_fd, width_fd, seed_type = get_seed_loc_sq()
        print('field', x_fd, y_fd, len_fd, width_fd, seed_type)
        # publish field data on ROS
        work = set_to_fd(x_fd, y_fd, len_fd, width_fd, seed_type)
        pub1.publish(work)

    rospy.sleep(0.1)
    print("Working")
```


Footnote may be given with Font size 10 of Times New Roman.

Appendix ‘F’

Field Planner Node

```
import time
import rospy
from std_msgs.msg import String
from nav_msgs.msg import Odometry


def callback(data):
    global field_data
    field_data = data.data.split('/')


def pose(data):
    print(2)
    global present_loc
    present_loc = data


def seed_spacing_selection(seed):
    if seed == 'wheat':
        return 0.17
    elif seed == 'onion':
        return 0.06
    elif seed == 'sorghum':
        return 0.12
    elif seed == 'soy_bean':
        return 0.08
    else:
        return 0


# noinspection SpellCheckingInspection
def seed_location(leng, wid, seed):
    sp = seed_spacing_selection(seed)
    x = present_loc.pose.pose.position.x
    a = present_loc.pose.pose.position.y
    leng += present_loc.pose.pose.position.x
    wid += present_loc.pose.pose.position.y
    fwd = True
    turn = False
    while x <= leng:
        if not turn:
            if fwd:
                y = a + wid - sp
                fwd = False
            else:
                y = a + sp
                fwd = True
        # set data in pose_data format
        q_x = float(present_loc.pose.pose.orientation.x)
        q_y = float(present_loc.pose.pose.orientation.y)
        q_z = float(present_loc.pose.pose.orientation.z)
```


Footnote may be given with Font size 10 of Times New Roman.

```
        q_w = float(present_loc.pose.pose.orientation.w)
        euler = quaternion_to_euler(q_x, q_y, q_z, q_w)
        goal = '{:.2f}/{:.2f}/{:.2f}/{:.2f}/{:.2f}'.format(x, y, present_loc.pose.pose.position.x,
present_loc.pose.pose.position.y, euler[2])
        pub1.publish(goal)
        # set data in seed_sowing format
        if not turn:
            go = 'sow/{:.2f}/{:.2f}/'.format(abs(present_loc.pose.pose.position.y - y), sp)
            pub2.publish(go)
        else:
            go = 'idle/{:.2f}/{:.2f}/'.format(abs(present_loc.pose.pose.position.y - y), sp)
            pub2.publish(go)
        # check if reached to goal iterate it
        rospy.sleep(1)
        while True:
            if x == present_loc[0] and y == present_loc[1]:
                break
        turn = not turn
        x += sp


def go_to_target(tgt_x, tgt_y):
    tgt_x = float(tgt_x)
    tgt_y = float(tgt_y)
    p_x = float(present_loc.pose.pose.position.x)
    p_y = float(present_loc.pose.pose.position.y)
    q_x = float(present_loc.pose.pose.orientation.x)
    q_y = float(present_loc.pose.pose.orientation.y)
    q_z = float(present_loc.pose.pose.orientation.z)
    q_w = float(present_loc.pose.pose.orientation.w)
    euler = quaternion_to_euler(q_x, q_y, q_z, q_w)
    goal = '{:.2f}/{:.2f}/{:.2f}/{:.2f}/{:.2f}'.format(tgt_x, tgt_y, p_x, p_y, euler[2])
    print(goal)
    pub1.publish(goal)


def quaternion_to_euler(x, y, z, w):
    import math
    t0 = +2.0 * (w * x + y * z)
    t1 = +1.0 - 2.0 * (x * x + y * y)
    X = math.degrees(math.atan2(t0, t1))

    t2 = +2.0 * (w * y - z * x)
    t2 = +1.0 if t2 > +1.0 else t2
    t2 = -1.0 if t2 < -1.0 else t2
    Y = math.degrees(math.asin(t2))

    t3 = +2.0 * (w * z + x * y)
    t4 = +1.0 - 2.0 * (y * y + z * z)
    Z = math.degrees(math.atan2(t3, t4))

    return X, Y, Z


rospy.init_node('field_planner', anonymous=True)
pub1 = rospy.Publisher('/pose_data', String, queue_size=5)
```


Footnote may be given with Font size 10 of Times New Roman.

```
pub2 = rospy.Publisher('/seed_sowing', String, queue_size=5)
pub3 = rospy.Publisher('/robot_status', String, queue_size=5)
global field_data
field_data = [2, 0, 0, 'nil', 0, 0]
global present_loc
# get confirmation to start field work with field parameters
while not rospy.is_shutdown():
    field_sub = rospy.Subscriber('field_work', String, callback)
    rospy.sleep(0.5)
    location = rospy.Subscriber('odometry/filtered', Odometry, pose)
    rospy.sleep(0.1)
    condition = field_data[0]
    print(condition)
    if condition == '1':
        print('in field')
        field_length = float(field_data[1])
        field_width = float(field_data[2])
        seed_type = field_data[3]
        tgt_x = float(field_data[4])
        tgt_y = float(field_data[5])
        if present_loc.pose.pose.position.x != tgt_x and present_loc.pose.pose.position.y != tgt_y:
            stat = '3'
            pub3.publish(stat)
            go_to_target(tgt_x, tgt_y)
        stat = '4'
        pub3.publish(stat)
        seed_location(field_length, field_width, seed_type)
    elif condition == '0':
        print('to field')
        target_x = float(field_data[4])
        target_y = float(field_data[5])
        stat = '1'
        pub3.publish(stat)
        go_to_target(target_x, target_y)
    else:
        print('no command received')
        stat = '0'
        pub3.publish(stat)
```


Footnote may be given with Font size 10 of Times New Roman.

Appendix 'G'

Node Command
Command Publish

```
from goal_controller import GoalController
from controller import Controller
from pose import Pose

import rospy
from std_msgs.msg import String


def callback(data):
    global pose_data
    pose_data = data.data.split('/')


# making objects
current_position = Pose()
target_goal = Pose()
goal_info = GoalController()
R5 = Controller()
#R5.setWheelSeparation(5)
#R5.setMaxMotorSpeed(1)
#R5.setTicksPerMeter(1000)
rospy.init_node('controller', anonymous=True)
pub1 = rospy.Publisher('/command2', String, queue_size=5)
pub1.publish("Connection initiated")

while not rospy.is_shutdown():
    pos_sub = rospy.Subscriber('pose_data', String, callback)
    rospy.sleep(1)
    target_goal.x = float(pose_data[0])
    target_goal.y = float(pose_data[1])
    current_position.x = float(pose_data[2])
    current_position.y = float(pose_data[3])
    current_position.theta = float(pose_data[4])

    desired_speeds = goal_info.get_velocity(current_position, target_goal, 3)
    lin = desired_speeds.xVel
    ang = desired_speeds.thetaVel

    speed = R5.getSpeeds(lin, ang)

    if speed.left < 0:
        if speed.right < 0:
            cmd = 'Set/{:.2f}/{:.2f}/R5/Reserve'.format(speed.left, speed.right)
        elif speed.right >= 0:
            cmd = 'Set/{:.2f}/{:.2f}/R5/LEFT'.format(speed.left, speed.right)
    elif speed.left > 0:
        if speed.right > 0:
            cmd = 'Set/{:.2f}/{:.2f}/R5/Forward'.format(speed.left, speed.right)
        elif speed.right <= 0:
            cmd = 'Set/{:.2f}/{:.2f}/R5/RIGHT'.format(speed.left, speed.right)
```


Footnote may be given with Font size 10 of Times New Roman.

```
    else:
        cmd = 'Set/{:.2f}/{:.2f}/R5/STOP'

    #print(cmd)
    print(speed.left, speed.right)

    # do whatever you want here
    pub1.publish(cmd)
    rospy.sleep(1)  # sleep for one second
```

Controller

```
from __future__ import division

#import rospy


class MotorCommand:
    """Holds motor control commands for a differential-drive robot.
    """

    def __init__(self):
        self.left = 0
        self.right = 0


class Controller:
    """Determines motor speeds to accomplish a desired motion.
    """

    def __init__(self):
        # Set the max motor speed to a very large value so that it
        # is, essentially, unbound.
        self.maxMotorSpeed = 1  # ticks/s
        self.ticksPerMeter = 100
        self.wheelSeparation = 5

    def getSpeeds(self, linearSpeed, angularSpeed):
        tickRate = linearSpeed * self.ticksPerMeter
        diffTicks = angularSpeed * self.wheelSeparation * self.ticksPerMeter

        speeds = MotorCommand()
        speeds.left = tickRate - diffTicks
        speeds.right = tickRate + diffTicks

        # Adjust speeds if they exceed the maximum.
        if max(abs(speeds.left), abs(speeds.right)) > self.maxMotorSpeed:
            factor = self.maxMotorSpeed / max(abs(speeds.left), abs(speeds.right))
            speeds.left *= factor
            speeds.right *= factor

        #speeds.left = int(speeds.left)
        #speeds.right = int(speeds.right)
        return speeds
```


Footnote may be given with Font size 10 of Times New Roman.

```
    def setWheelSeparation(self, separation):
        self.wheelSeparation = separation

    def setMaxMotorSpeed(self, limit):
        self.maxMotorSpeed = limit

    def setTicksPerMeter(self, ticks):
        self.ticksPerMeter = ticks
```

Goal Controller

```
from __future__ import division, print_function
from math import pi, sqrt, sin, cos, atan2
from pose import Pose
#import rospy

class GoalController:
    """Finds linear and angular velocities necessary to drive toward
    a goal pose.
    """

    def __init__(self):
        self.kP = 3
        self.kA = 8
        self.kB = -1.5
        self.max_linear_speed = 10
        self.min_linear_speed = 0
        self.max_angular_speed = 3
        self.min_angular_speed = 0
        self.max_linear_acceleration = 10
        self.max_angular_acceleration = 10
        self.linear_tolerance = 0.025 # 2.5cm
        self.angular_tolerance = 3/180*pi # 3 degrees
        self.forward_movement_only = False

    def set_constants(self, kP, kA, kB):
        self.kP = kP
        self.kA = kA
        self.kB = kB

    def set_max_linear_speed(self, speed):
        self.max_linear_speed = speed

    def set_min_linear_speed(self, speed):
        self.min_linear_speed = speed

    def set_max_angular_speed(self, speed):
        self.max_angular_speed = speed

    def set_min_angular_speed(self, speed):
        self.min_angular_speed = speed

    def set_max_linear_acceleration(self, accel):
        self.max_linear_acceleration = accel

    def set_max_angular_acceleration(self, accel):
        self.max_angular_acceleration = accel
```

```
    def set_linear_tolerance(self, tolerance):
        self.linear_tolerance = tolerance

    def set_angular_tolerance(self, tolerance):
        self.angular_tolerance = tolerance

    def set_forward_movement_only(self, forward_only):
        self.forward_movement_only = forward_only

    def get_goal_distance(self, cur, goal):
        if goal is None:
            return 0
        diffX = cur.x - goal.x
        diffY = cur.y - goal.y
        return sqrt(diffX*diffX + diffY*diffY)

    def at_goal(self, cur, goal):
        if goal is None:
            return True
        d = self.get_goal_distance(cur, goal)
        dTh = abs(self.normalize_pi(cur.theta - goal.theta))
        return d < self.linear_tolerance and dTh < self.angular_tolerance

    def get_velocity(self, cur, goal, dT):
        desired = Pose()

        goal_heading = atan2(goal.y - cur.y, goal.x - cur.x)
        a = -cur.theta + goal_heading

        # In Automomous Mobile Robots, they assume theta_G=0. So for
        # the error in heading, we have to adjust theta based on the
        # (possibly non-zero) goal theta.
        theta = self.normalize_pi(cur.theta - goal.theta)
        b = -theta - a

        # rospy.loginfo('cur=%f goal=%f a=%f b=%f', cur.theta, goal_heading,
        #               a, b)

        d = self.get_goal_distance(cur, goal)
        if self.forward_movement_only:
            direction = 1
            a = self.normalize_pi(a)
            b = self.normalize_pi(b)
        else:
            direction = self.sign(cos(a))
            a = self.normalize_half_pi(a)
            b = self.normalize_half_pi(b)

        # rospy.loginfo('After normalization, a=%f b=%f', a, b)

        if abs(d) < self.linear_tolerance:
            desired.xVel = 0
            desired.thetaVel = self.kB * theta
        else:
            desired.xVel = self.kP * d * direction
```


Footnote may be given with Font size 10 of Times New Roman.

```
        desired.thetaVel = self.kA*a + self.kB*b

    # Adjust velocities if X velocity is too high.
    if abs(desired.xVel) > self.max_linear_speed:
        ratio = self.max_linear_speed / abs(desired.xVel)
        desired.xVel *= ratio
        desired.thetaVel *= ratio

    # Adjust velocities if turning velocity too high.
    if abs(desired.thetaVel) > self.max_angular_speed:
        ratio = self.max_angular_speed / abs(desired.thetaVel)
        desired.xVel *= ratio
        desired.thetaVel *= ratio

    # TBD: Adjust velocities if linear or angular acceleration
    # too high.

    # Adjust velocities if too low, so robot does not stall.
    if abs(desired.xVel) > 0 and abs(desired.xVel) < self.min_linear_speed:
        ratio = self.min_linear_speed / abs(desired.xVel)
        desired.xVel *= ratio
        desired.thetaVel *= ratio
    elif desired.xVel==0 and abs(desired.thetaVel) < self.min_angular_speed:
        ratio = self.min_angular_speed / abs(desired.thetaVel)
        desired.xVel *= ratio
        desired.thetaVel *= ratio

    return desired

def normalize_half_pi(self, alpha):
    alpha = self.normalize_pi(alpha)
    if alpha > pi/2:
        return alpha - pi
    elif alpha < -pi/2:
        return alpha + pi
    else:
        return alpha

def normalize_pi(self, alpha):
    while alpha > pi:
        alpha -= 2*pi
    while alpha < -pi:
        alpha += 2*pi
    return alpha

def sign(self, x):
    if x >= 0:
        return 1
    else:
        return -1
```

Pose

```
from __future__ import division

class Pose:
```


Footnote may be given with Font size 10 of Times New Roman.

```
def __init__(self):
    self.x = 0
    self.y = 0
    self.theta = 0
    self.xVel = 0
    self.yVel = 0
    self.thetaVel = 0


def __str__(self):
    return str({'x': self.x, 'y': self.y, 'theta': self.theta,
                'xVel': self.xVel, 'yVel': self.yVel,
                'thetaVel': self.thetaVel})
```


Footnote may be given with Font size 10 of Times New Roman.

Appendix 'H'

ROS Node Motor Driver
Motor Controller

```
import time
import rospy
from std_msgs.msg import String

from motor_drive import motor_driver

global CheckCrash
global NewCommand
global driveLeft_motor
global driveRight_motor
Forward = 1
Reverse = 2
Stop = 0
Right_Left = 3
Non_Stop = -1
Non_Crash = -2


def callback(data):
   global CheckCrash
   global NewCommand
   if data.data.find('Off') > -1:
      CheckCrash = Stop
      print("Stop")
   else:
      if data.data.find('/Forward') > -1:
          CheckCrash = Forward
          print("Forward")
      if data.data.find('/Reserve') > -1:
         CheckCrash = Reverse
         print("Reverse")
      if data.data.find('None') > -1:
         CheckCrash = Right_Left
         print("None")
   NewCommand = True
   move_robot(data.data)


def move_robot(command):
   global driveLeft_motor
   global driveRight_motor

   if command.startswith("Set"):
      # Motor power setting: Set/driveLeft_motor/driveRight_motor
      parts = command.split('/')
      if len(parts) > 0:
         try:
            driveLeft_motor = float(parts[1])
            driveRight_motor = float(parts[2])
         except:
```


Footnote may be given with Font size 10 of Times New Roman.

```
            # Bad values
            driveRight_motor = 0.0
            driveLeft_motor = 0.0
    else:
        # Bad message
        driveRight_motor = 0.0
        driveLeft_motor = 0.0
    if parts[0]>0:
        left = 1
    elif parts[0]<0:
        left = -1
    if parts[1]>0:
        right = 1
    elif parts[1]<0:
        right = -1
    feedback = '{}/{}'.format(left,right)
    pub.publish(feedback)

    drive.driver(driveLeft_motor, driveRight_motor)


# Init node
rospy.init_node('control_node3', anonymous=True)
pub = rospy.Publisher('/motor_feedback', String, queue_size=5)

time.sleep(1)
CheckCrash = Stop
NewCommand = False
driveRight_motor = 0.0
driveLeft_motor = 0.0
drive = motor_driver()

# Run the GetCommands until we are told to close
try:
    print('Press CTRL+C to terminate the controller')
    time.sleep(2)
    subscriber = rospy.Subscriber("command2", String, callback)
    rospy.spin()
except KeyboardInterrupt:
    # CTRL+C exit
    print('\nUser shutdown')
```

Motor Driver

```
import RPi.GPIO as GPIO

class motor_driver:
    def __init__(self):
        self.left_motor_fr = 33
        self.left_motor_rv = 35
        self.right_motor_fr = 12
        self.right_motor_rv = 32
        GPIO.setmode(GPIO.BOARD)
        GPIO.setwarnings(False)
        self.initial_setup()

    def driver(self, left_v, right_v):
        if left_v > 1:
```


Footnote may be given with Font size 10 of Times New Roman.

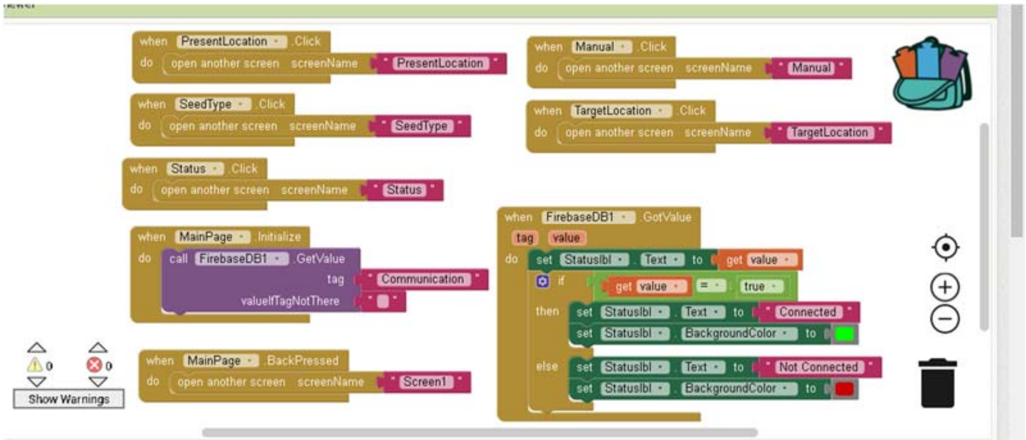
```
            left_v = 1
        elif left_v < -1:
            left_v = -1
        if right_v > 1:
            right_v = 1
        elif right_v < -1:
            right_v = -1
        if left_v == 0 and right_v == 0:
            self.stop()
        elif left_v > 0 and right_v > 0:
            self.forward(left_v, right_v)
        elif left_v < 0 and right_v < 0:
            self.reverse(left_v, right_v)
        elif left_v < 0 and right_v > 0:
            self.turn_left(left_v, right_v)
        elif left_v > 0 and right_v < 0:
            self.turn_right(left_v, right_v)

    def initial_setup(self):
        GPIO.setup(self.left_motor_fr, GPIO.OUT)
        GPIO.setup(self.left_motor_rv, GPIO.OUT)
        GPIO.setup(self.right_motor_fr, GPIO.OUT)
        GPIO.setup(self.right_motor_rv, GPIO.OUT)
        self.left_motor_pwm_fr = GPIO.PWM(self.left_motor_fr, 1000)
        self.left_motor_pwm_rv = GPIO.PWM(self.left_motor_rv, 1000)
        self.right_motor_pwm_fr = GPIO.PWM(self.right_motor_fr, 1000)
        self.right_motor_pwm_rv = GPIO.PWM(self.right_motor_rv, 1000)
        self.left_motor_pwm_fr.start(0)
        self.left_motor_pwm_rv.start(0)
        self.right_motor_pwm_fr.start(0)
        self.right_motor_pwm_rv.start(0)

    def stop(self):
        self.left_motor_pwm_fr.start(0)
        self.left_motor_pwm_rv.start(0)
        self.right_motor_pwm_fr.start(0)
        self.right_motor_pwm_rv.start(0)

    def forward(self, x, y):
        self.left_motor_pwm_fr.start(x*100)
        self.left_motor_pwm_rv.start(0)
        self.right_motor_pwm_fr.start(y*100)
        self.right_motor_pwm_rv.start(0)

    def reverse(self, x, y):
        self.left_motor_pwm_fr.start(0)
        self.left_motor_pwm_rv.start(abs(x)*100)
        self.right_motor_pwm_fr.start(0)
        self.right_motor_pwm_rv.start(abs(y)*100)

    def turn_left(self, x, y):
        self.left_motor_pwm_fr.start(0)
        self.left_motor_pwm_rv.start(abs(x) * 100)
        self.right_motor_pwm_fr.start(y * 100)
        self.right_motor_pwm_rv.start(0)
```

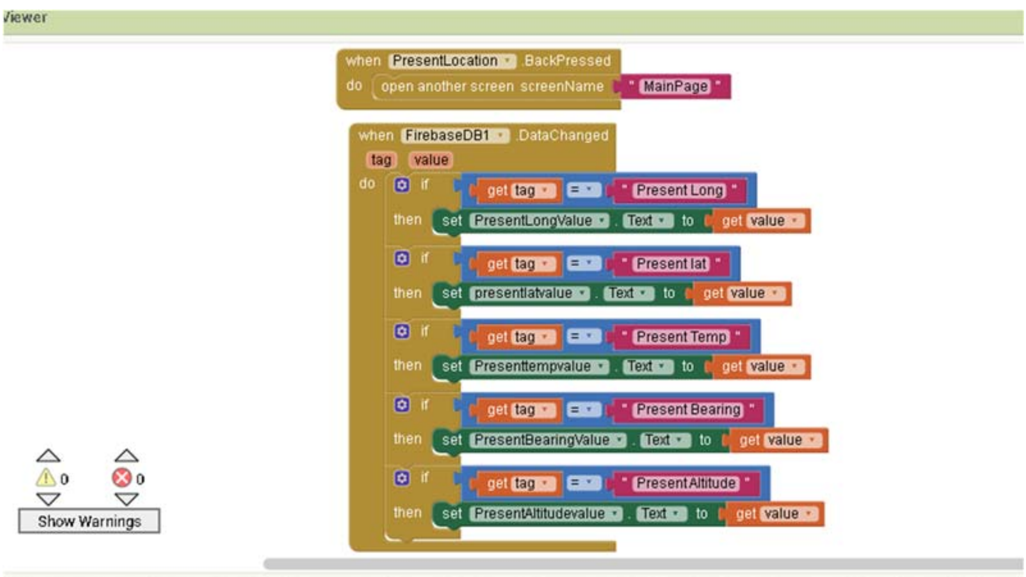

Footnote may be given with Font size 10 of Times New Roman.

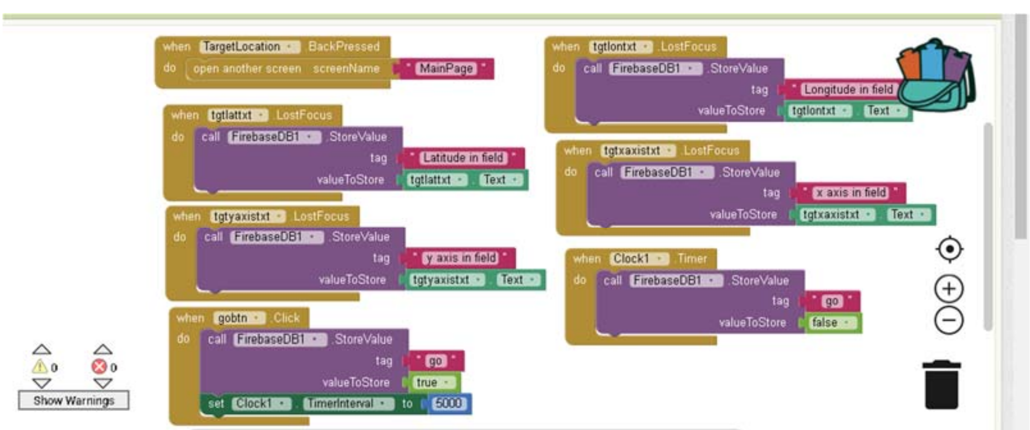
```
def turn_right(self, x, y):
    self.left_motor_pwm_fr.start(x * 100)
    self.left_motor_pwm_rv.start(0)
    self.right_motor_pwm_fr.start(0)
    self.right_motor_pwm_rv.start(abs(y) * 100)
```


Footnote may be given with Font size 10 of Times New Roman.

Annex 'I'







Footnote may be given with Font size 10 of Times New Roman.

REFERENCES

[1] Seed Sowing Machine by Alessandro Lupi & Fedrico Lucca - 2016

[2] Automatic Seed Sowing Robot by Abdul Rehman & Ahmed Akbar – Theme College of Engineering, India - 2017

[3] Automatic Seed Sowing robot by Vidya Yadave, Punam Bhosale & Jyoti Shinde – IRJET 2019

[4] https://appinventor.mit.edu/

[5] https://firebase.google.com/

[6] https://www.ros.org/

Footnote may be given with Font size 10 of Times New Roman.