

# **MULTIPURPOSE AGRICULTURAL ROBOT**

## **BTech Project Report**

**Submitted in partial fulfilment for the award of the Degree of  
Bachelor of Technology in Electrical and Electronics Engineering**

By

**ABHIJITH C V (B180840EE)**

**ABHISHEK K PRADEEP (B180251EE)**

**ANAGHA P (B181051EE)**

**ANEESH KUMAR C M (B180781EE)**

**KRISHNAJITH K (B180884EE)**

Under the guidance of

**Dr. PREETHA P**



Department of Electrical Engineering  
NATIONAL INSTITUTE OF TECHNOLOGY, CALICUT  
NIT Campus P. O., Calicut-673601, India

2022



## CERTIFICATE

*This is to certify that the thesis entitled “MULTIPURPOSE AGRICULTURAL ROBOT” is a bonafide record of the Project done by **ABHIJITH CV** (B180840EE), **ABHISHEK K PRADEEP** (B180251EE), **ANAGHA P** ( B181051EE), **ANEESH KUMAR C M** (B180781EE) and **KRISHNAJITH K** (B180884EE) under my supervision and guidance, in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Electrical & Electronic Engineering from National Institute of Technology Calicut for the year 2022.*

**Dr. PREETHA P**

Associate Professor and Head,  
Dept. of Electrical Engineering

Place: NIT Calicut

Date: 03/05/2022

## **ACKNOWLEDGEMENT**

The accomplishment of this report benefits from the help and direction from our project guide, **Dr. PREETHA P**, Associate Professor and Head, Department of Electrical Engineering, to whom we express our heartfelt gratitude. We would also like to pay our sincere gratitude to those who have helped and supported us for the preparation of this project.

ABHIJITH C V

ABHISHEK K PRADEEP

ANAGHA P

ANEESH KUMAR C M

KRISHNAJITH K

## ABSTRACT

Agriculture is the backbone of main nations across the globe. In India, it provides employment to around 70% of the population and accounts for about 17% to the total GDP of India. It includes different processes like sowing of seed, fertilization, weeding, irrigation etc which can be tedious and requires large manpower. This project aims to develop an automated multipurpose agricultural robot which uses less power and human effort in the agronomic applications, thereby reducing farmer's effort. This work aims to design, develop and model the robot in the ROS platform with the help of Fusion 360 and MATLAB Simulink, which can sow the seeds, cut the weeds and perform irrigation without any manual control. The structural schematics and control algorithm for the coordination of various processes involved are studied. The robot is powered by a DC motor, the speed control of which is also studied. The structure of the robot was designed and developed in Fusion 360 and Solidworks. The various data and image processing algorithms for object detection and tracking were carried out in ROS using OpenCV. The cost analysis of hardware implementation of the robot is also carried out.

**Keywords:** Agriculture, Autonomous, Multibody Dynamics, Computer Vision, Image Processing, Path Tracking, Artificial Neural Networks, Object Detection.

## TABLE OF CONTENTS

Chapter No.	Title	Page No.
	List of Figures	iii
	List of Tables	v
1	INTRODUCTION	
	1.1 Introduction	1
	1.2 Motivation	3
	1.3 Literature review	4
	1.4 Objectives	5
	1.5 Organisation of thesis	5
2	SYSTEM DESCRIPTION.	
	2.1 Fundamentals of Multibody Dynamics.	7
	2.2 Kinematics of the Differential Robot.	7
3	MODELLING AND SIMULATION	
	3.1 Multibody System Assembly	11
	3.2 Speed Control of DC Motor	13
	3.3 Design of PID Controller	14
	3.4 Supervisory Control	17
	3.5 Simulation Results (MATLAB)	19
4	CROP ROW DETECTION AND NAVIGATION	
	4.1 Introduction	21
	4.2 Row detection algorithm	21
	4.3 Row navigation and proportional controller	23
	4.4 ROS and Gazebo simulation	25
	4.5 Simulation Results (ROS)	26
	4.6 Improved algorithm for row detection and navigation.	28

5	EXECUTION OF AGRICULTURAL TASKS	
	5.1 Seeding	34
	5.2 Irrigation	35
	5.3 Weeding	36
6	MATERIALS AND SPECIFICATIONS	
	6.1 Selection of DC Motor	42
	6.2 Battery Specifications	44
	6.3 Electronic Components	46
	6.4 Material Used	50
	6.5 Estimation of Cost for Hardware Implementation	56
7	CONCLUSION	
	7.1 Conclusion	58
	7.2 Future Scope of Work	58
	REFERENCE	60
	APPENDIX	61

## List of Figures

<b>Figure No.</b>	<b>Title</b>	<b>Page No.</b>
1	Basic Architecture of Multipurpose Agricultural robots	3
2	Differential Drive Kinematics	8
3	CAD Model	11
4	Simscape Multibody System	12
5	DC Motor Schematic	13
6	Simulation with and without speed control	14
7	Feedback network for speed control	15
8	Simulation with speed control	16
9	Motor for Steer Action	17
10	Stateflow Diagram	18
11	Virtual Field View in Mechanics explorer (MATLAB)	18
12	Wheel Speed Characteristics	19
13	Steer Angle Characteristics	20
14	Raw image data received from the front facing camera	21
15	Mask created by splitting images into HSV channels applying filter ranges.	22
16	Contours detected using the HSV mask.	22
17	Circles enclosing the detected contours	22
18	Line fitted using OpenCV function	23
19	Fitted line and reference line superimposed on raw image.	23
20	Camera frame showing error angle and distance from reference line	24
21	Various ROS nodes and topics active during gazebo simulation	25
22	Row detection and follow controller block diagram	26

23	Gazebo simulation environment with spawned robot model and crop field.	26
24	Angular velocity plots of right and left motor from simulation	27
25	Plots of error values computed from camera frame	28
26	Markers for identification	29
27	Marker aligned along crop rows.	29
28	Agribot finite state machine for row detection and navigation	31
29	Error dist v/s time (above) and error angle v/s time (below)	32
30	Gazebo simulation environment with spawned robot model and crop field.	33
31	Seeder	34
32	Water Tank	35
33	Weeder	37
34	Weed and Plant Detection	38
35	Free body diagram of the robotic wheel	42
36	Motor, Battery and Electronic Components	44
37	Materials and Parts	50



## **List of Tables**

<b>Table No.</b>	<b>Title</b>	<b>Page No.</b>
1	PID Controller Parameters	14
2	Control Parameters in Open loop and Closed Loop Response	16
3	Errors Quantification in Closed Loop Control in Step Input	17
4	Confusion Matrix for Plants and Weeds	41
5	Cost Estimation	56

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 Introduction**

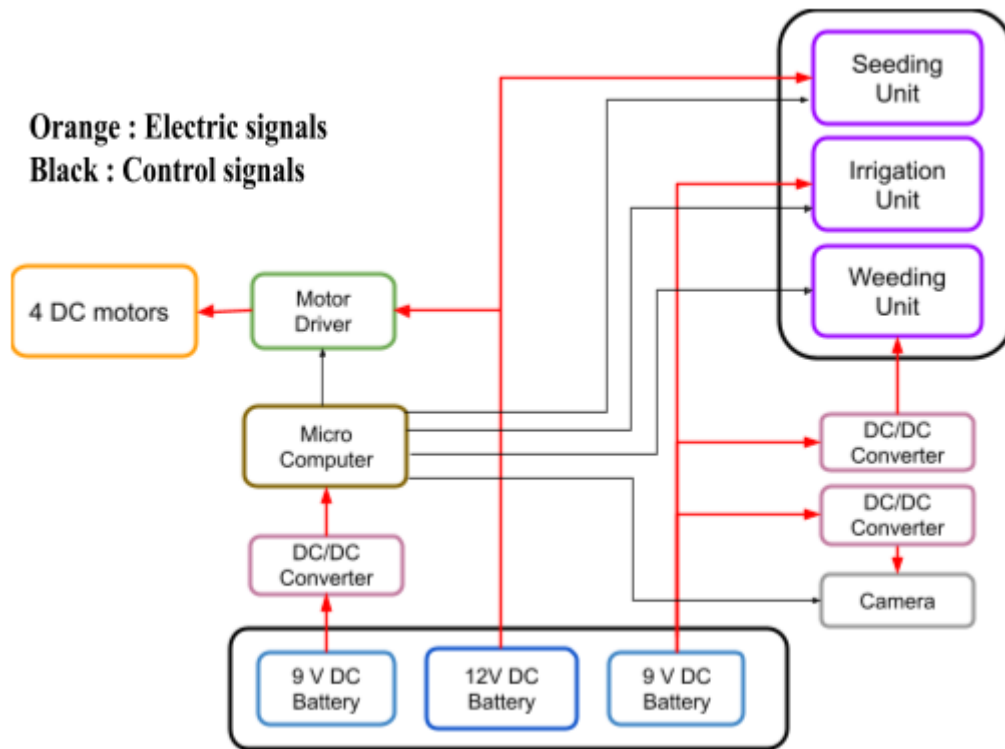
Agriculture constitutes an important part in the economy of nations across the globe. Since the dawn of history, agriculture has been one of the significant earnings of producing food for human utilization. Over 70 percent of the Indian rural household is still dependent on agriculture for livelihoods. The demand for food is also increasing due to increasing population. Apart from that, some of the environmental phenomena affect agricultural production. Hence the requirements in the field of agriculture are growing at a faster rate. Usually, the traditional methods like spreading, scattering are performed by humans, which takes time and results in low production in the field of agriculture. Lag in collecting, spreading and improper seedbed preparation reduces the yield of the crop. The inadequate accessibility of power also disturbs the efficiency of products. Hence, the modernization or automation in the field of agriculture helps to overcome the above challenges.

The current trend is to incorporate automation of technologies involved in the agricultural process so as to reduce the human labour required and to increase the efficiency and productivity. In the 1980's many agricultural robots were started for research and development. Over history, agriculture has evolved from a manual occupation to a highly industrialized business, utilizing a wide variety of tools and machines. Presently, there is a growing trend towards the realization of autonomous agricultural vehicles. The idea of applying robotics technology in agriculture is very new and opens up a plethora of opportunities. This is definitely going to have a large impact on this sector.

Robotics and automation can play a significant role in enhancing agricultural production needs, it can perform elementary agricultural operations like sowing, weeding, fertilization, irrigation etc, either on manual control or on its own, by the given algorithm. There are many advantages to it. Apart from reducing effort and increasing productivity, the applications of robotics also

ensures safety, by replacing the human operators, when the duties that are needed to be performed are harmful for workers. Also, it results in high speed of operation. One is that it will do a job very repetitively and very much the same every time, so you can get huge quality improvements in a number of areas. Another key advantage in agriculture is that robots can work 24 hours a day – often when there's no light, which can be a big factor with certain crops. Autonomous vehicles are small and light, and it can replace the heavy machinery which requires a large power to operate, resulting in soil compaction, which further limits soil aeration, water infiltration and root penetration.

First and foremost, a 3D physical model or an electromechanical model of an autonomous robot is developed using CAD (Fusion 360) software in this project. This CAD model is then imported into the MATLAB Simulink environment as well as the ROS Platform. Then, the model is made physically accurate by including dynamics, hydraulics, and other factors and using the SimMechanics toolbox in MATLAB, the designed parts are assembled to follow multibody dynamics. DC motors provide the rotational actuation for the wheels and a PID controller was also implemented in the model for the speed control of this motor. Standard approaches involving ranging sensors and image processing were explored to detect safe traversal paths for the navigation of robot. State flow algorithms are used for decision making in real time, to perform task planning based on the given conditions and actions. The image processing algorithms are implemented into the model for object detection and tracking, which helps us in navigation and weeding purposes, which is done with the help of visual feedback obtained from the camera fitted on the robot chassis. Apart from this, a visualisation tool, ROS RViz tool is used to visualise agricultural functions like spraying and seeding. Finally, an estimation of costs for implementing the hardware model of the multipurpose agricultural robot is performed.



**Fig 1: Basic Architecture of Multipurpose Agricultural Robot**

## 1.2 Motivation

Agriculture is one of the important economic activities that provides us with food, fibre, and a feeder for our survival. In present days, farmers face a lack of manpower and are not able to cope with the growing demands. The field of agriculture also involves different tasks like ploughing which require heavy machinery, which might be hazardous. To sum up, some of the major issues in the Indian agriculture field involve increasing input costs, availability of skilled labours, and crop monitoring.

Hence, automation technologies were used in agriculture to overcome these obstacles and to help farmers to reduce their efforts and maximise the output and productivity. But the agricultural robots present day work in a predefined way, so only repetitive tasks can be programmed, and also it requires manual control to coordinate various tasks.

The main motive of this project is to apply robotic technologies to develop a multipurpose agricultural robot that performs various agricultural processes like seeding, weeding, and irrigation in an automated way. The robot also provides optional switching between various tasks according to the conditions. The manual control of the robot is also provided for the convenience of farmers. Once this technology is adopted into agriculture, the cost will come down, and it increases speed and accuracy of labour, leading to increased productivity.

### **1.3 Literature Review**

Many research papers have been based on the problems faced in the field of agriculture like increasing costs, lack of skilled labour and low productivity due to delay in time while performed manually. Therefore [1] discusses how these obstacles can overcome the automation technologies, by which farmers can reduce their effort and increase their yield. A robot that has been designed in [1] for agricultural tasks with the aid of various new technologies and research. The agrobot performs basic elementary functions like ploughing, seeding, levelling and water spraying and also tests for various parameters like soil condition etc with minimal human interventions. However, the monitoring based on camera and sensors were absent in this agrobot.

For reducing the human efforts further and making it fully automated, sensors can be used. IOT sensors or GPS are used in order to provide the information about the field and help in crop monitoring. By integrating IOT sensors in the robots, the robot will be able to act upon the input given by the users. Hence, by using cutting edge technology like Arduino, IOT and wireless sensor network, the paper aims at making a smart agricultural ecosystem, better monitoring of crops and environmental conditions, which would act as a major factor to improve the yield of the crop production. However, this would increase the cost of the robot.

Sowing of seed is one of the important processes in agriculture, which requires a good amount of human effort and time. The agrobot [3] is focused on designing and developing a smart seed sowing agrobot to solve this issue. This smart seed sowing robot consists of one robotic arm, whose position is controlled through a mobile application, and it sows the seeds from the seed's container.

The present-day robots are mostly focused on doing predefined tasks repeatedly, often a single task. The multipurpose robots available in the market are based on the manual control to switch tasks and are heavily dependent on the decisions made by the operating person. The aim of this

project is to develop a multipurpose agricultural robot which can perform functions like automatic seeding, weeding and irrigation based on the decisions made by the controller in response to the input given, events and the other time-based conditions. The sensor and camera information provided to the controller for aiding the decision-making process. By this manner, the human effort and the time required can be reduced drastically, thereby increasing productivity. A single robot can be used to perform multiple tasks which is economical and reduces the cost. Furthermore, greater focus will be laid on the costs and the efficiency, while selecting the components and materials needed for the hardware implementation. These limitations inspired the development of a multipurpose agricultural robot which is automated and economically feasible.

## **1.4 Objectives**

The objectives of the work are:

- To develop a model of path following an Agricultural robot in a Simulation environment and test for various conditions.
- To control the motion and actions of the robot by means of supervisory control.
- To navigate the field using markers, as well as to auto correct its trajectory while traversing the field.
- To detect plants and weeds and visualization of spraying and in ROS RViz.
- To automate agricultural functions like sowing, spraying and fertilisation so as to integrate all the above functions into a single machine
- To determine the DC motor and Battery specifications and also identify different materials required for implementing the robot, with a detailed cost analysis.

## **1.5 Organisation of Thesis**

Chapter 1 introduces the subject on which the project is based on, explains the motive and states the objectives.

Chapter 2 deals with the theoretical aspects, explaining the system description, the fundamentals of multibody dynamics and the kinematics of the system.

Chapter 3 deals with the modelling of the robot and its simulation in MATLAB Simulink and the control algorithm developed in Stateflow.

Chapter 4 deals with the exploration of crop row detection and navigation, using techniques like Image Processing, in ROS.

Chapter 5 deals with the implementation and visualisation of various agricultural tasks such as seeding, weeding and irrigation.

Chapter 6 deals with the specifications and materials used for the robot model, with a detailed cost estimation.

Chapter 7 includes the conclusion and future scope of the project.

## **CHAPTER 2**

### **SYSTEM DESCRIPTION**

#### **2.1 Fundamentals of Multibody Dynamics**

The system configuration or the position of all material points of the body must be able to be characterized by the description variables at any instant of time. The generalized coordinates include both linear and angular coordinates.

The smallest number of variables required to fully characterise a system's configuration, or the smallest number of independent generalised coordinates required to describe a system's configuration uniquely is referred to as system's degrees of freedom (DoF) .When a kinematic joint is added to a system, the overall number of degrees of freedom is reduced due to the amount of constraints imposed by the joint, which varies depending on the number and kind of joints used, and where the constraints must be independent of one another. The difference between the system coordinates and the number of independent constraints can be used to calculate the number of degrees of freedom of a multibody system.

$$n_{DoF} = 6n_b - m \quad (1)$$

where  $n_b$  represents the number of bodies that compose the multibody system and  $m$  is the number of independent constraints

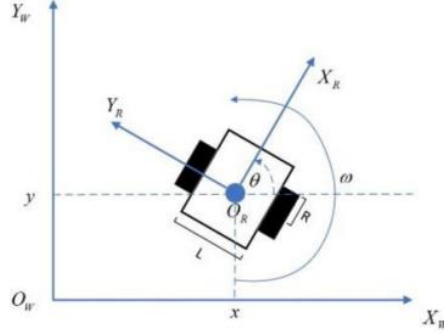
There are two forms of analysis that can be used to investigate the dynamics of a mechanical system: forward dynamics and inverse dynamics. The external forces acting on the bodies of a system are known in forward dynamic analysis, and the resulting motion is derived by solving the equations of motion. Inverse dynamic analysis, on the other hand, seeks a specific motion for a multibody system with the goal of determining the forces required to achieve that motion. In the context of the present work, methods of kinematics and forward dynamics are employed.

#### **2.2 Kinematics of a differential drive mobile robot**

Agricultural mobile robots are a subtype of wheeled robots with differential drives. It is made up of two driving wheels that are positioned on a single axis and can be driven forward or



backward independently. Hence, the kinematics of this mobile robot can be derived as follows:



**Fig 2: Differential Drive Kinematics[9]**

The generalised coordinate constraints  $q = (x, y, \theta)$ , where  $(x, y)$  is the position and  $\theta$  is the heading of the centre of the axis of the wheels represent the configuration of the robot platform. It can also be represented with respect to a global inertial frame,  $\{O_w, X_w, Y_w\}$ . Let the robot frame be  $\{O_r, X_r, Y_r\}$  and the robot's linear velocities in the  $x$  and  $y$  directions of the robot frame are  $v_x$  and  $v_y$  respectively.  $v_l$  is the left wheel angular velocity,  $v_r$  is the right wheel angular velocity, and  $\omega$  is the heading rate.  $L$  is the distance between the wheels and  $R$  is the radius of the wheels. The kinematic model in the robot frame is given by:

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ 0 & 0 \\ -\frac{R}{L} & \frac{R}{L} \end{bmatrix} \begin{bmatrix} v_l \\ v_r \end{bmatrix} \quad (2)$$

The kinematic model in the world frame is given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ \omega \end{bmatrix} \quad (3)$$

where,

$$\begin{aligned} v_l &= \frac{2v_x - \omega L}{2R} \\ v_r &= \frac{2v_x + \omega L}{2R} \end{aligned} \quad (4)$$

At any instant R and  $\omega$  can be solved as follows:

$$R = \frac{l}{2} \frac{V_l + V_r}{V_r - V_l}; \quad \omega = \frac{V_r - V_l}{l}; \quad (5)$$

There are three possible cases with these kinds of drives:

1. If  $V_l = V_r$ , then the forward linear motion in a straight line. R becomes infinite, and there is no rotation effectively, that is  $\omega$  is zero.
2. If  $V_l = -V_r$ , then  $R = 0$ , and rotation is about the midpoint of the wheel axis - ve rotate in place.
3. If  $V_l = 0$ , or  $V_r = 0$ , then the rotation about the left wheel or right wheel respectively is obtained.

### 2.2.1 Forward Kinematics

Assume the robot is at some point (x, y) and is moving in a direction that makes an angle with the X axis. It is considered that the robot is located in the middle of the wheel axle. This gets the robot to move to different places and orientations by altering the control parameters  $V_l$ ,  $V_r$ . (Note:  $V_l$  and  $V_r$  denote ground wheel velocities.) The ICC location is found using the velocities  $V_l$  and  $V_r$  and equation 3:

$$ICC = [x - R \sin(\theta), y + R \cos(\theta)] \quad (6)$$

and the robots pose at the at time  $t + \delta t$  will be:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega \delta t) & -\sin(\omega \delta t) & 0 \\ \sin(\omega \delta t) & \cos(\omega \delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega \delta t \end{bmatrix} \quad (7)$$

The motion of a robot rotating a distance  $R$  about its ICC with an angular velocity of  $\omega$  is described by this equation. Another way to think about it is that the robot's movement is equivalent to 1) translating the ICC to the coordinate system's origin, 2) rotating the origin by an angular amount  $\theta$ , and 3) translating back to the ICC.

### 2.2.2 Inverse Kinematics

In general, the position of a robot capable of moving in a particular direction  $\Theta(t)$  at a given velocity  $V(t)$  can be described as:

$$\begin{aligned}x(t) &= \int_0^t V(t) \cos[\theta(t)] dt \\y(t) &= \int_0^t V(t) \sin[\theta(t)] dt \\ \Theta(t) &= \int_0^t \omega(t) dt\end{aligned}\tag{8}$$

The inverse kinematics challenge is when they control the robot to reach a specific configuration  $(x, y, \theta)$ . When determining its location, a differential drive robot imposes non-holonomic limitations. The robot, for example, is unable to move laterally along its axle. . The motion equations for the special instances of  $v_l = v_r = v$  (robot travelling in a straight line) are:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + v \cos(\theta) \delta t \\ y + v \sin(\theta) \delta t \\ \theta \end{bmatrix}\tag{9}$$

If  $v_r = -v_l = v$ , then the robot rotates in place and the equations become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + 2v\delta t/l \end{bmatrix}\tag{10}$$

This motivates a strategy of moving the robot in a straight line, then rotating for a turn in place, and then moving straight again as a navigation strategy for differential drive robots.

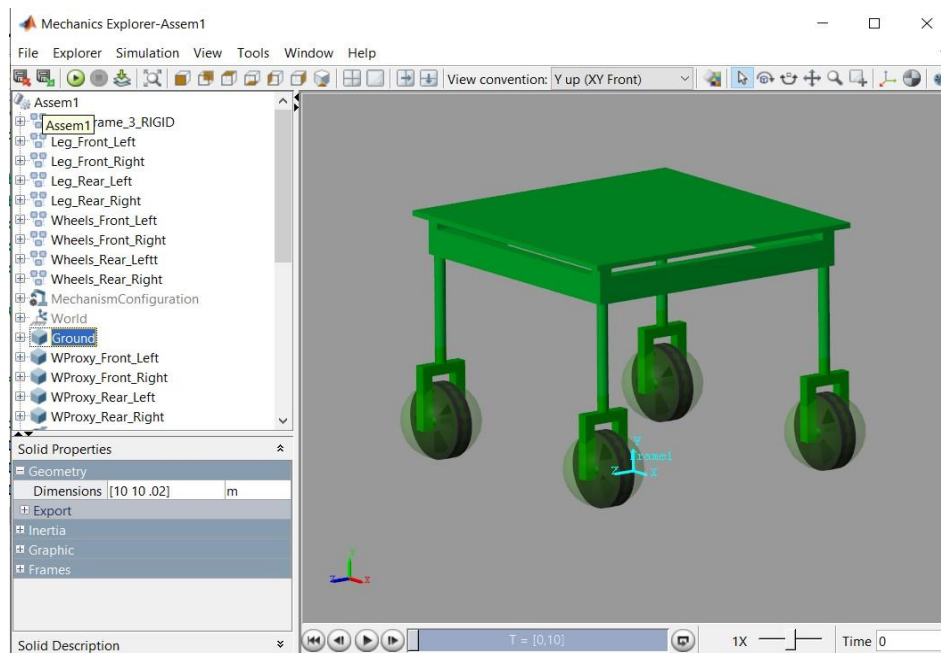
## CHAPTER 3

### MODELLING AND SIMULATION

#### 3.1 Multibody system assembly

Model-Based Design approach in engineering is employed for integrating design and testing to be done simultaneously through simulations. Accordingly, the robot can be designed and built on the CAD model, and then it can be imported to MATLAB Simulink Environment for employing multibody dynamics.

The CAD model can be imported into Simscape by breaking into sub assemblies and connecting them together using Simscape Multibody [5]. The independent sub-assemblies used are Chassis, Wheels etc. The model can be then simplified by removing components that don't affect the accuracy of simulation. After importing, these sub-assemblies can be linked together by different joints. Revolute joints and 6DoF are primarily used for this purpose. The Rigid Transform blocks are used to shift the joint and wheel centre of mass to the correct location. With the help of these blocks, each part's location relative to the Chassis can be obtained and all the parts together can be arranged to create the full robot.



**Fig 3 CAD Model**

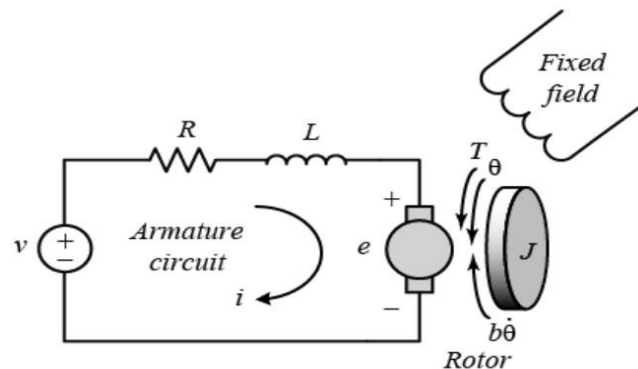
After compiling the model, the mechanic explorer can be used to verify part locations and adjust



### 3.2 Speed Control of Motor

Once the CAD model is imported into the Simulink environment, the necessary connections are made and actuations are provided to the appropriate joints. Upon simulation, it's seen that the speed of the model is not regulated and constantly varies. In order to attain speed control, a controller must be designed that can produce the desired speed.

The following is the schematic of the DC Motor system [6].



**Fig.5. DC Motor Schematic [6]**

The motor parameters chosen are as follows (for demonstration):

(K) Torque Constant/Back emf Constant = 0.01 Nm/A or Vs/rad

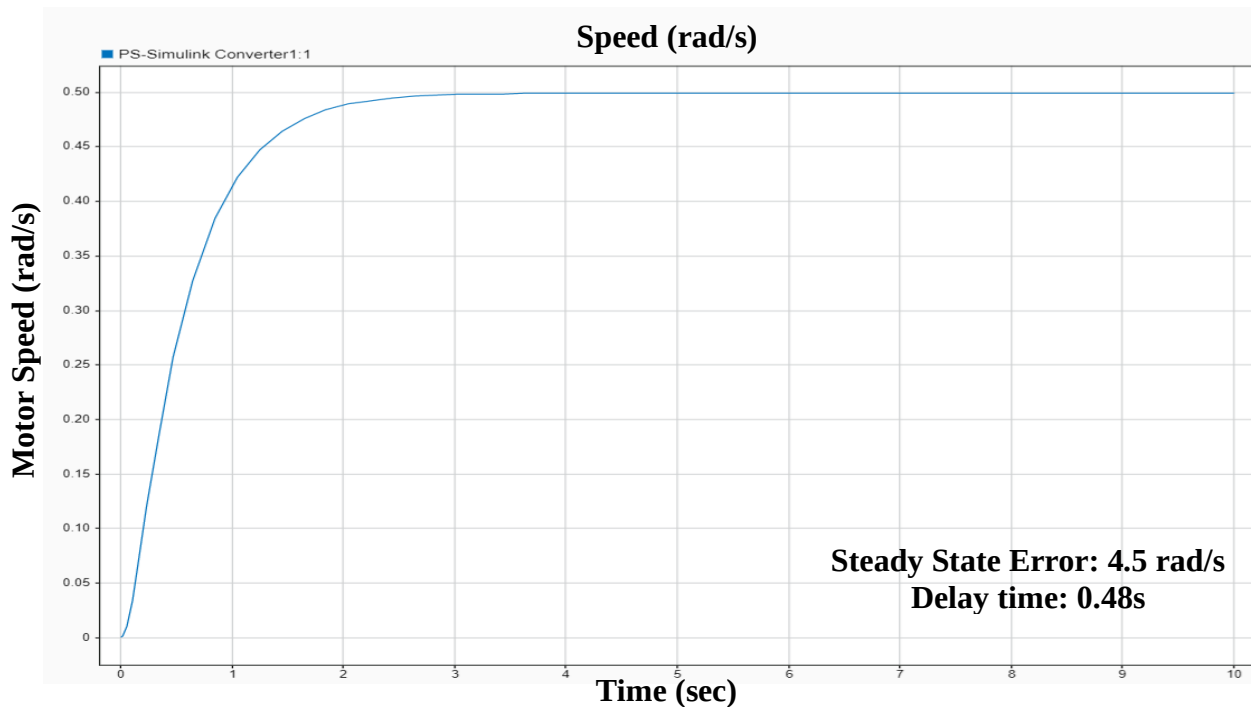
(J) Rotor Inertia = 0.01 kg m<sup>2</sup>

(b) Rotor damping = 0.1 Nms/rad

(L) Armature Inductance = 0.5 H

(R) Armature Resistance = 1 ohms

After modelling the DC motor with the given parameters, the simulation is run and plotted for the desired speed of 5 rad/s.



**Fig.6 Simulation without speed control (Open Loop Response)**

The speed settles at around 0.5 rad/s which is significantly less than the required 5 rad/s. Hence a large steady-state error is observed.

To achieve speed regulation, an appropriate controller is needed to be designed. For this purpose, a PID Controller is considered. A PID Controller is one type of controller known to be able to reduce steady-state error.

### 3.3 Design of PID Controller

By using the PID Tuner App available in MATLAB, the controller gains are estimated which is necessary to match the requirements.

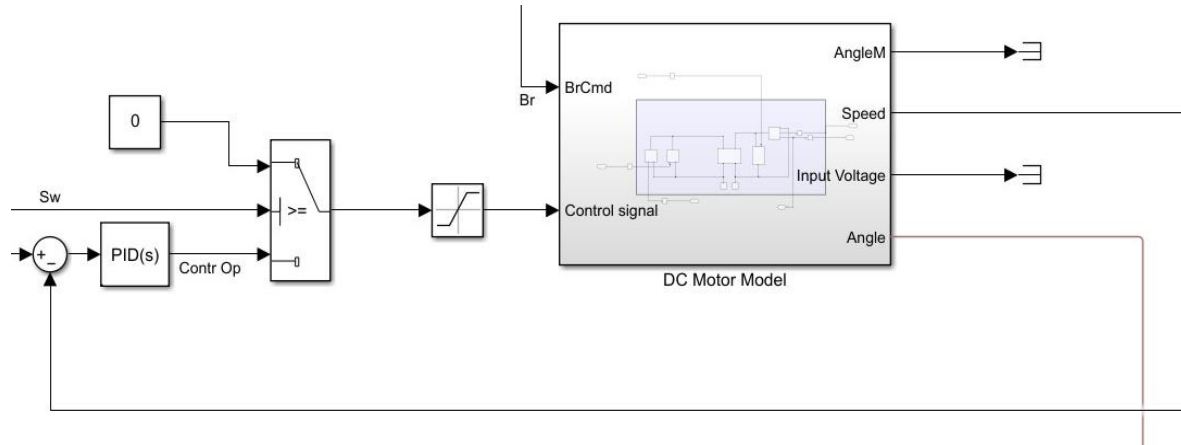
The design specifications chosen are: Settling time < 2 secs ; Max overshoot < 5 %.

From these specifications, a reasonable controller is designed whose gains are given as:

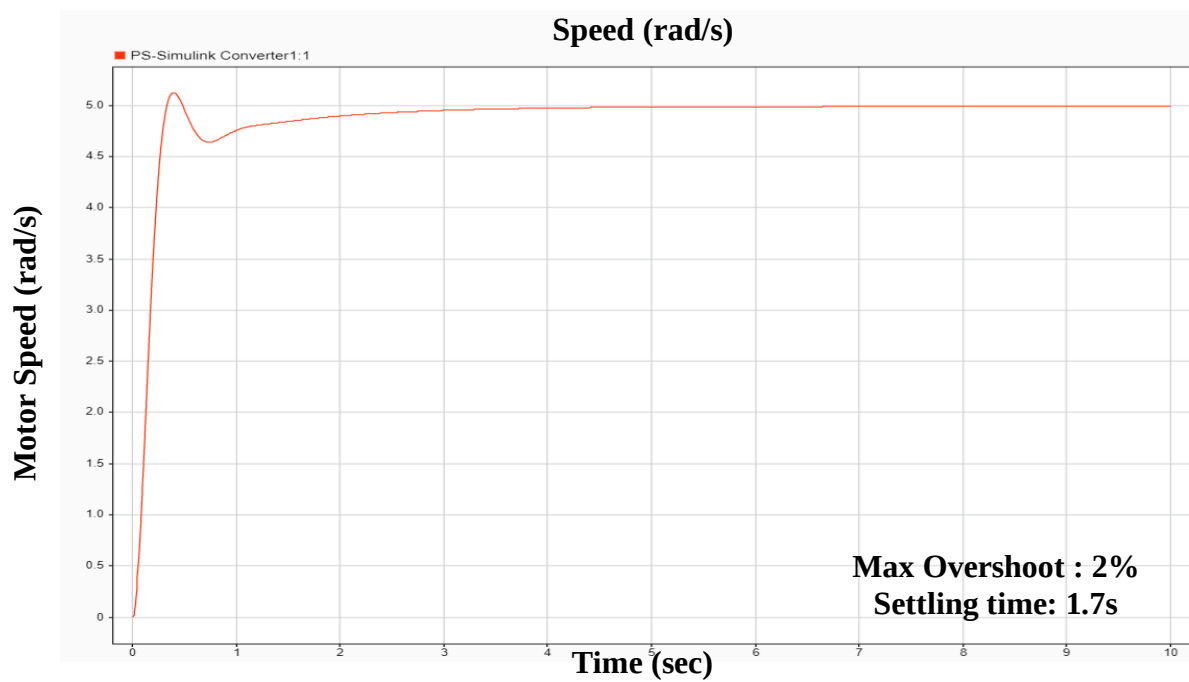
Parameters	Gains
Proportional (P)	20.9022
Integral (I)	39.7486
Derivative (D)	1.9387

**Table 1: PID Controller Parameters**

This controller is added to the feedback network and the simulation is run and the results are plotted in Fig 8.

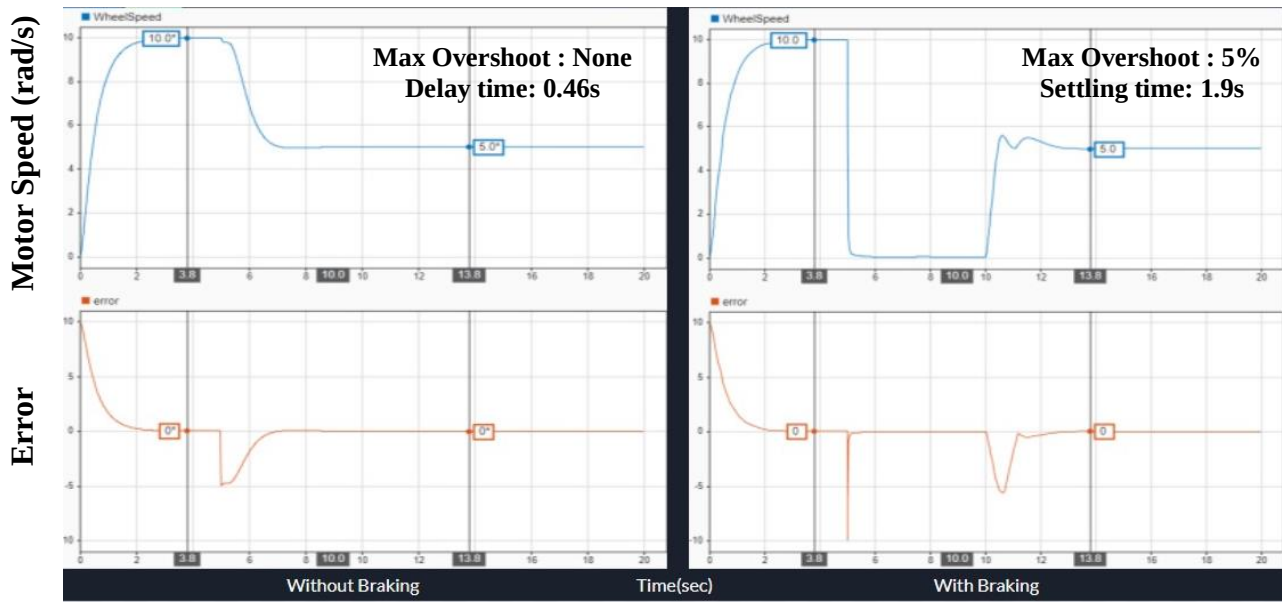


**Fig.7. Feedback network for speed control**



**Fig.8 a)**





**Fig.8 b). Simulation with speed control (Closed Loop Response)**

	Steady State Error (rad)	Max Overshoot (%)	Settling Time (secs)	Delay time (secs)
Open Loop Response	4.5	None	None	0.48
Closed Loop Response (Step Input)	0	2	1.7	0.2
Closed Loop Response (Time varying input)	0	5	1.9	0.46

**Table 2: Control Parameters in Open loop and Closed Loop Response**

In Fig 8. The a) part shows the speed control when a step input is applied while b) shows the speed control when time varying step input is applied (10 - 5 rad/s).

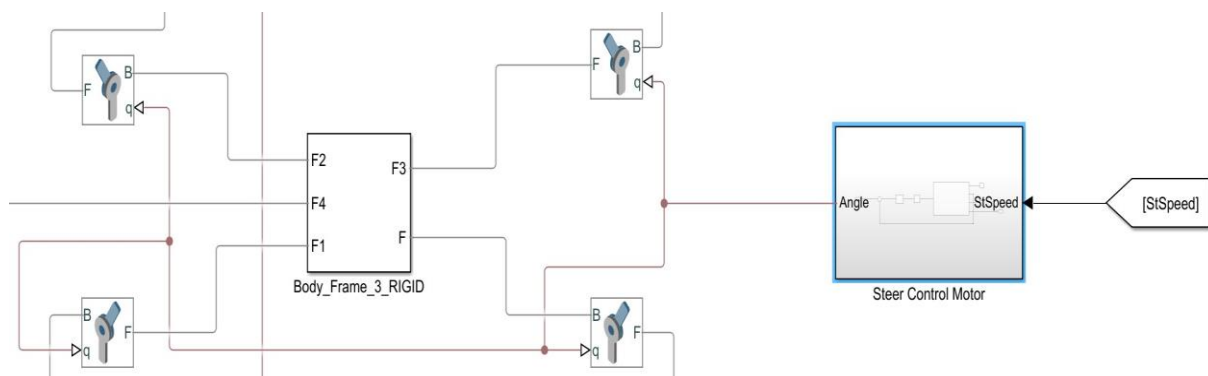
As seen from the simulations in Fig..8 , there is better control on the speed with reasonable performance by eliminating the steady state error with overshoot as 4% and settling time around 2 secs for both single and time varying inputs. Hence design specifications are satisfactory.

The PID controller designed meets all of the stated design requirements. A steer action is also performed by the robot with the help of motors as shown below to negotiate a turn to move across the field.

The following table shows the error quantification in closed loop control for a step input.

Time (s)	Input (rad)	Output (rad)	Error	Error (%)
0	5	0	5	100
0.2	5	1.2	3.8	76
0.4	5	3.1	1.9	38
0.6	5	4.4	0.6	12
0.8	5	4.9	0.1	2
1	5	5.2	-0.2	-4
1.7	5	5.1	-0.1	-2
2.2	5	5	0	0
2.4	5	5	0	0

**Table.3: Errors quantification in closed loop response for step inputs**



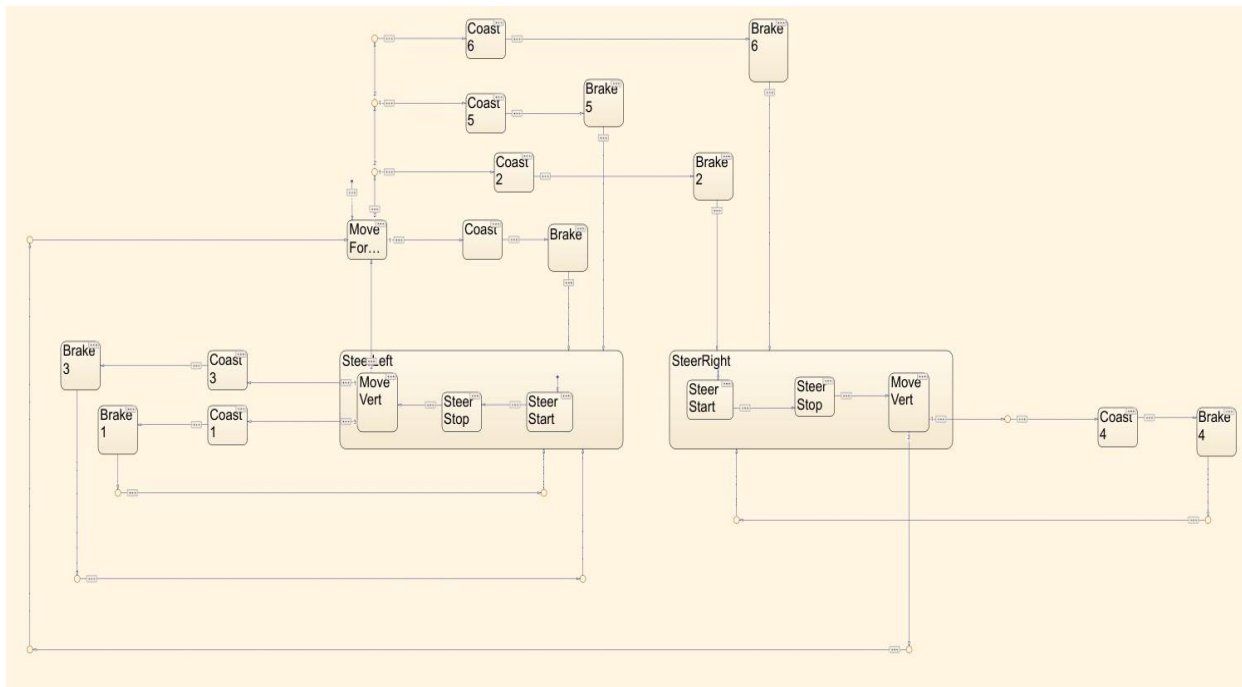
**Fig 9 Steering Motor**

### 3.4 Supervisory Control

For supervisory control of the robot, a Stateflow algorithm has been used which uses state transition diagrams, flow charts, state transition tables, and truth tables to make real time decisions based on input signals, events and time based conditions. With the help of Stateflow charts available through MATLAB a supervisory control logic is created to command the actions and motion of the robot.

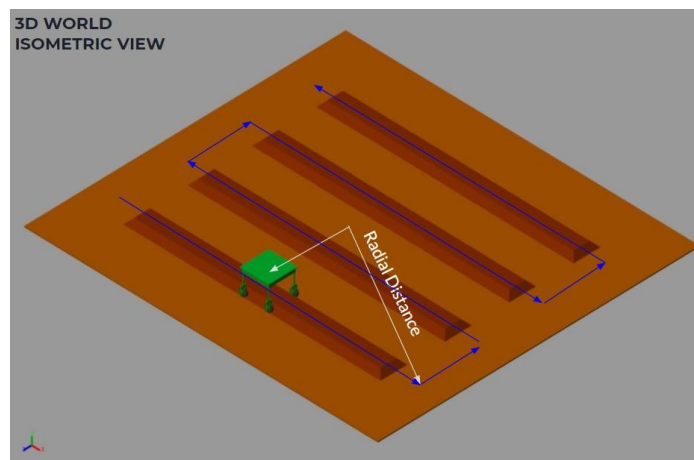
Firstly, the motion and the path of the robot is considered that is to be followed. For this purpose a sensor is required that measures the radial distance from the center of the field to the robot's center. This is implemented as follows: The Transform block senses the distances along the x and z axes and an operation is carried out on these variables to obtain the required radial distance.

By making use of this sensing element to measure the distances, a control logic is implemented to navigate around the field. This is done with the help of Stateflow charts as shown below.



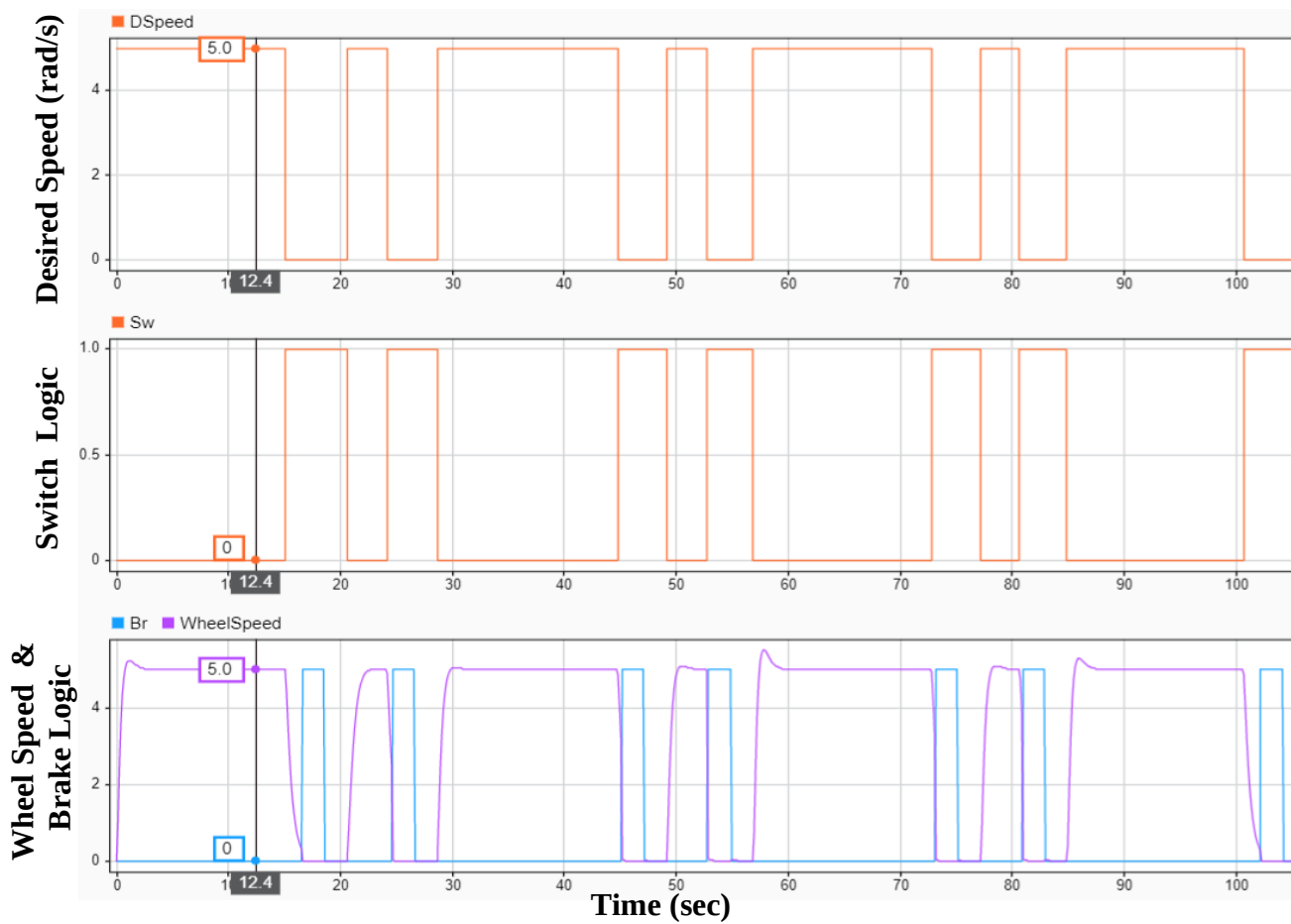
**Fig 10: Stateflow chart**

Here 3 actions are performed on the drive motors ie, Forward movement, Vertical movement, Coasting and then Braking. Whereas, the steer motors are responsible for Left and Right steering action. The chart outputs the following variables, Speed for the drive motor, Steering angle, Switch logic and the Brake command. By making use of this sensor distance data a logic is implemented such that it follows a specified path correctly and navigates around the field as shown below.



**Fig 11: Virtual Field View in Mechanics explorer (MATLAB)**

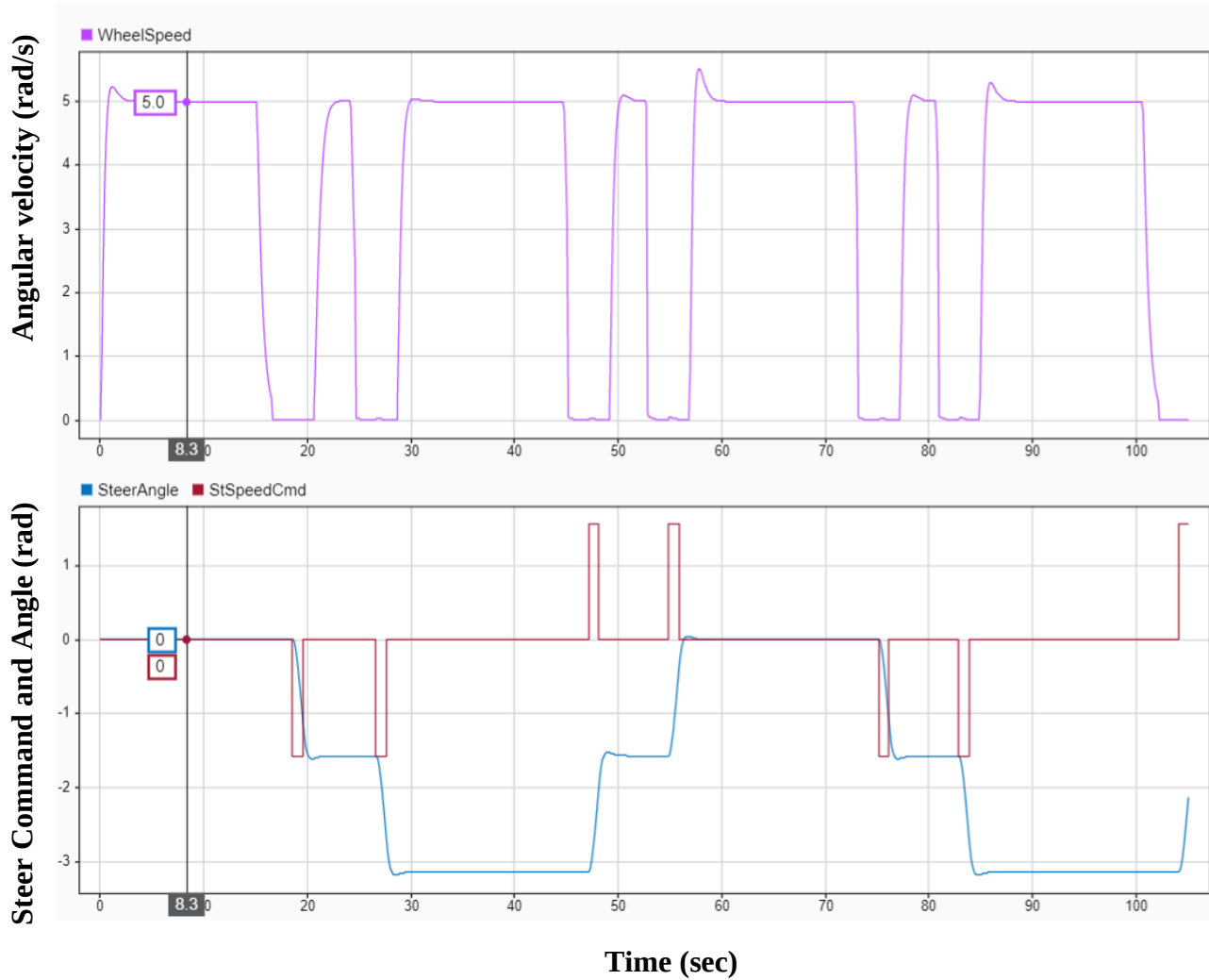
### 3.5 Simulation Results (MATLAB):



**Fig 12: Wheel Speed Characteristics**

The first row shows the Desired speed command given to the PID controller which then gives an appropriate voltage signal to activate the Drive motors. First the robot moves across the lane and makes it to the opposite end.

It is evident from the above plots, that the Stateflow algorithm decides and strategically outputs Switch and Brake Commands one after the other such that the robot stops at the specified spot to negotiate the turn to move across the lanes. The narrow peaks in the Wheel Speed implies the vertical motion of the robot across the field while the wider ones indicate horizontal motion along the crop lanes. The whole simulation took around 105 secs to complete for the robot to reach the destination through the specified path.



**Fig 13: Steer Angle Characteristics**

This graph shows the signals of Steering Commands and Steering Angle to actuate the steer motors. It's clear from the graph that the Steer commands are given by Stateflow when the Wheel Speed is decreased to 0 i.e. the robot is fully stopped after applying the brakes. The Steer action is completed fully before the Drive motors are actuated for vertical motion. The negative Steer command implies a Left Turn while the positive one indicates the Right Turn.

The data obtained on various parameters essential for material selection for both the structure and the circuitry by running these simulations.

## **CHAPTER 4**

### **CROP ROW DETECTION AND NAVIGATION**

#### **4.1 Introduction**

Detection of crop row is done by processing images received from a camera fitted at the front of the robot chassis. This camera gives a visual feedback of forthcoming crop row structure and helps the robot steer accordingly. Image processing is done using OpenCV C++ library. A simple proportional controller will command a differential drive controller which produces rotational velocity commands to speed controllers of four motors connected to four wheels. ROS and gazebo were used for row detection algorithm testing.

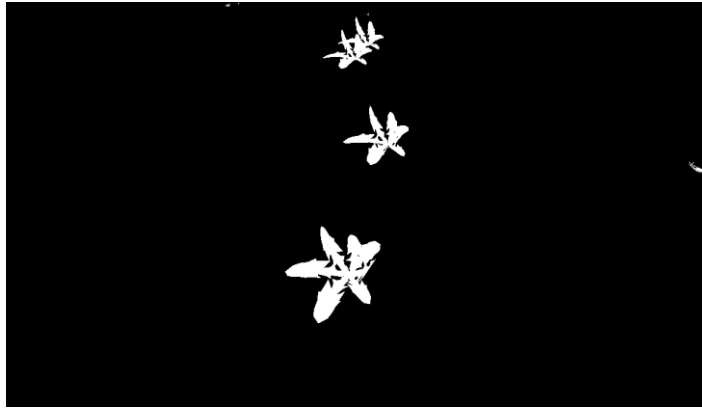
#### **4.2 Row detection algorithm**

Algorithm for detecting crop row is given below:

- 1) Image received from the front camera is splitted into its hue, saturation and value channels as it is easier to create a mask using HSV color space than in RGB color space. These HSV channels are used to create a mask to detect the plants. Pixels with hue value between 50 to 80 is retained and for saturation and values channels values between 0 to 255 is retained.

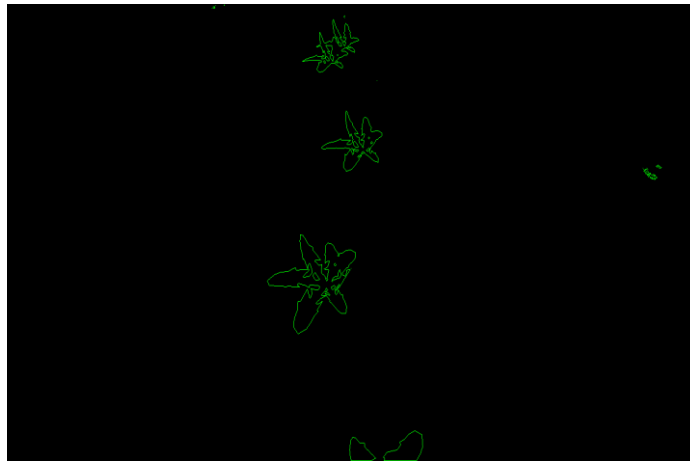


**Fig 14:Raw image data received from the front facing camera**



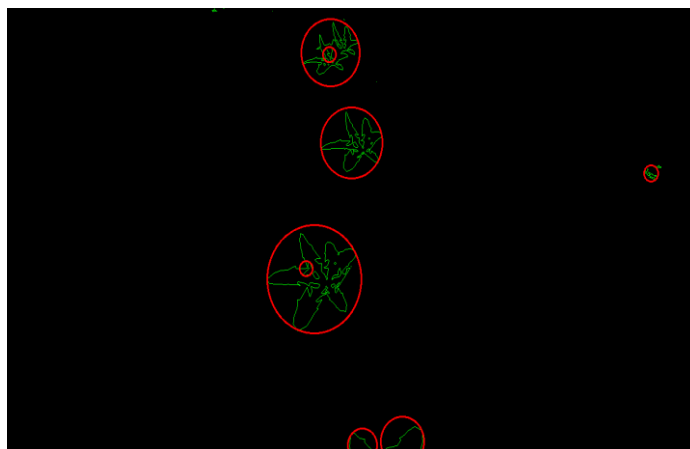
**Fig 15 :Mask created by splitting images into HSV channels and applying filter ranges.**

- 2) Mask created in the above step is used to find contours or clusters.



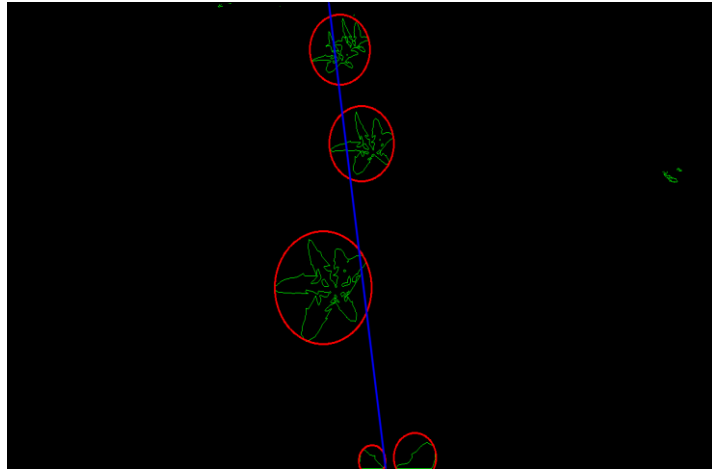
**Fig 16: Contours detected using the HSV mask.**

- 3) Finding a polygon that can enclose the given contour and also finding a minimum circle that can enclose the polygon.



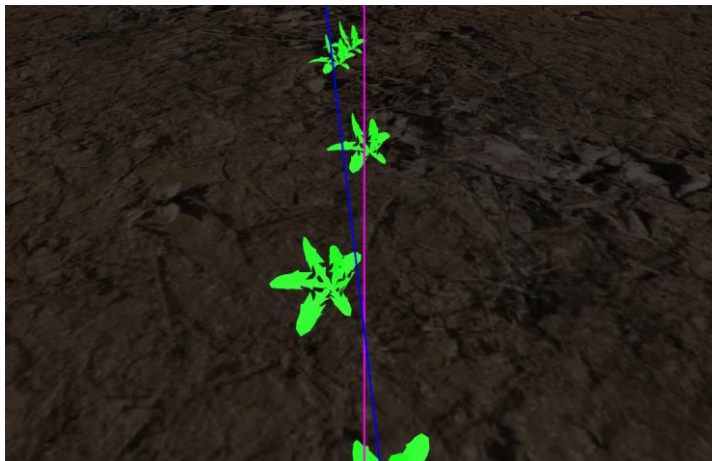
**Fig 17:Circles enclosing the detected contours**

- 4) After getting minimum enclosing circles their center is calculated and given to a line fitting function that tries to fit a line through the given centers points.



**Fig 18: Line fitted using OpenCV function given the enclosing circle's center point coordinates**

- 5) The fitted line is used to compute the error angle and error distance from the camera frame center line (purple line in figure given below).



**Fig 19 : Fitted line(blue) and reference line(pink) superimposed on raw image.**

### **4.3 Row navigation and proportional controller**

After computing the best fit average line given the plant center coordinates, the angle between the reference line passing vertically through the midpoint of the camera frame and the fitted line is taken as the error angle. The difference between the x-coordinate of mid-points of the reference line and fitted line is taken as the error distance. The error distance and error angle is



used to compute the angular velocity of the robot base. Forward velocity of the robot base is kept at a constant velocity of 0.4 m/s.

Angular velocity of base is given by:

$$\omega = K_1 \times \theta + K_2 \times d \quad (11)$$

where,

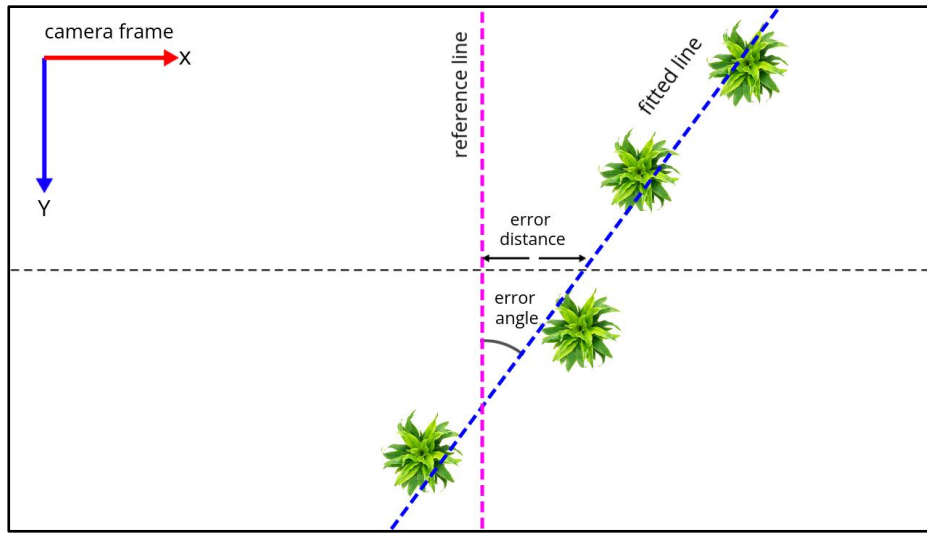
$\omega$  = angular velocity of the robot base.

$K_1$  =gain for angle error.

$K_2$  =gain for distance error.

$\theta$  = angle error

$d$  =distance error.



**Fig 20: Camera frame showing error angle and distance from reference line**

Through trial and error, satisfactory values for  $K_1$  and  $K_2$  were obtained and they are:

$$K_1 = 30 \text{ and } K_2 = 0.2.$$

Forward velocity and angular velocity computed by the controller is given to a differential drive controller which converts these commands into angular velocity for the four motor by using following equations:

$$v_l = \frac{2v_x - \omega L}{2R}$$

$$v_r = \frac{2v_x + \omega L}{2R} \quad (12)$$

where,

$v_l, v_r$  are left and right motor angular velocities respectively

$v_x$  =forward velocity of robot base.

$\omega$  = angular velocity of the robot base.

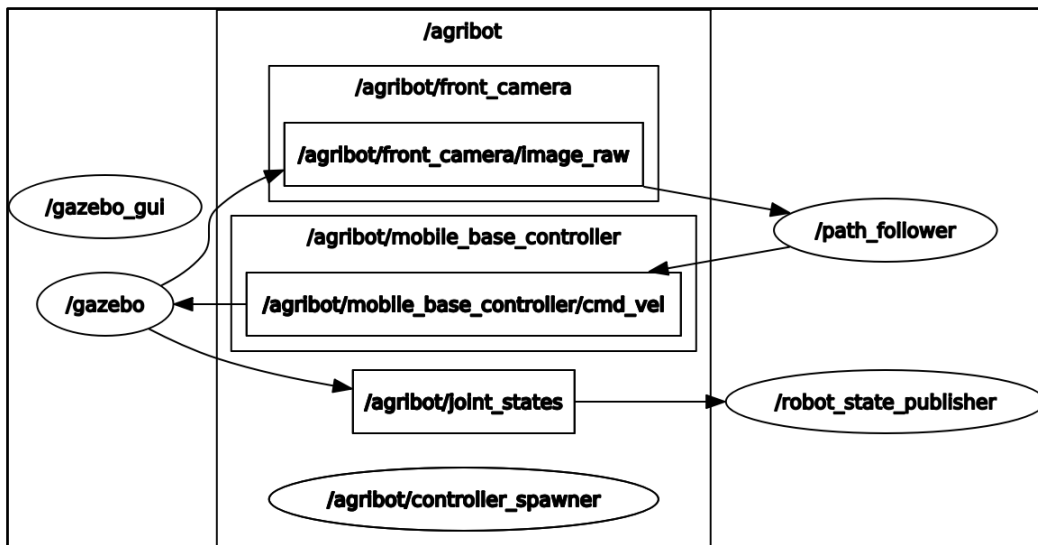
$R$  = radius of the wheel

$L$  =distance between left and right wheels.

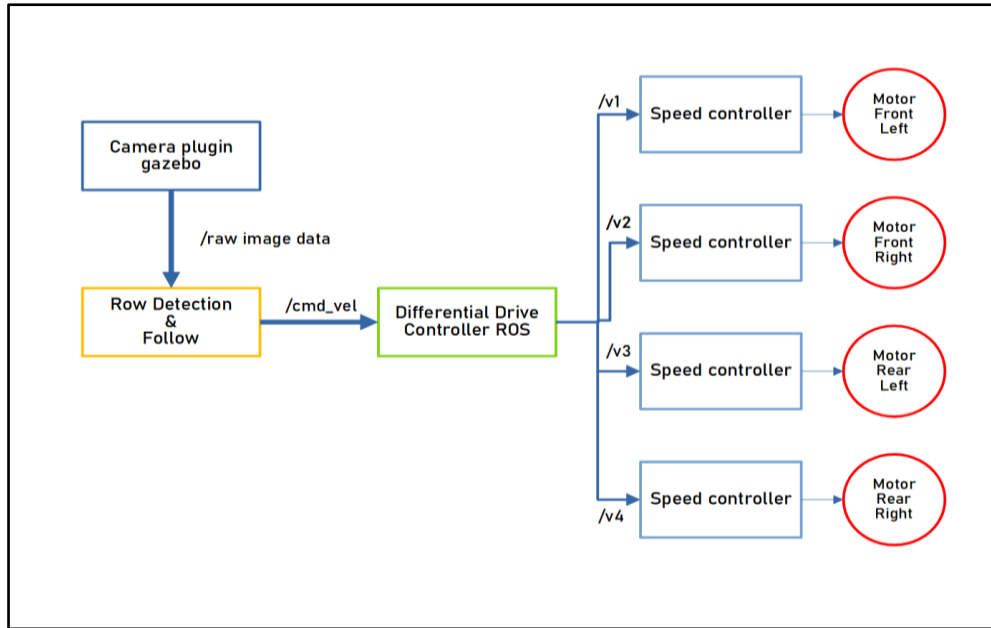
The velocity command generated by the differential controller is fed to the speed controllers of four motors.

#### 4.4 ROS and Gazebo simulation

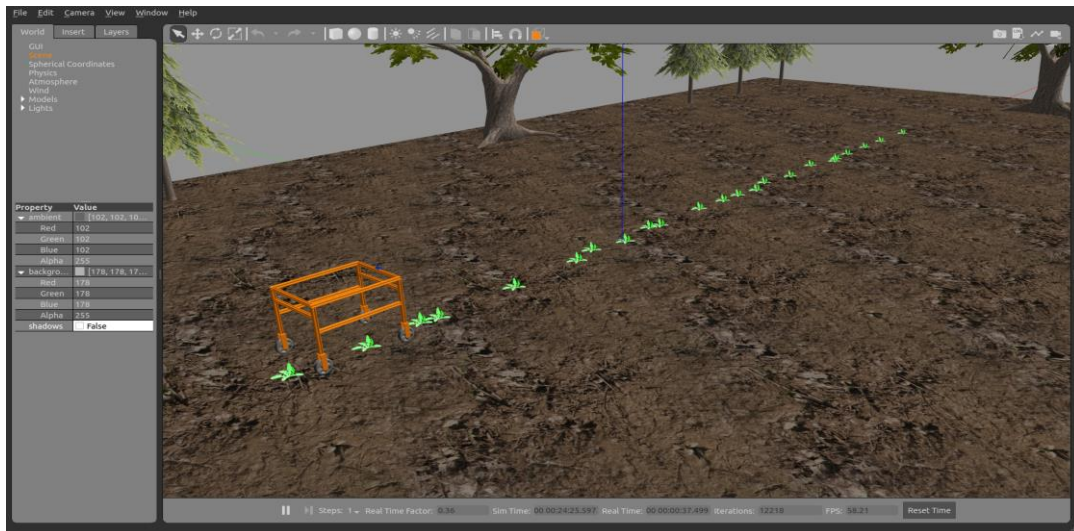
Robot operating system(ROS) and gazebo simulation environment were utilized for complete physical simulation of the robot and testing of row detection and navigation algorithms. ROS have bindings that connect the OpenCV library with ROS and ROS nodes where programmed in C++ programming language. An agricultural farm was designed in gazebo for testing the detection algorithm. A single row of plants were placed in the environment. A robot model was loaded into the gazebo via spawn\_urdf node. Row detection and the following algorithm was implemented as a ROS node using C++. Row detection node(/path\_follower) will publish geometry twist messages(/cmd\_vel). This message is picked up by a differential controller(/agribot/mobile\_base\_controller) which will convert this twist message into wheel joints angular velocities.



**Fig 21: Various ROS nodes and topics active during gazebo simulation.**



**Fig 22:Row detection and follow controller block diagram**

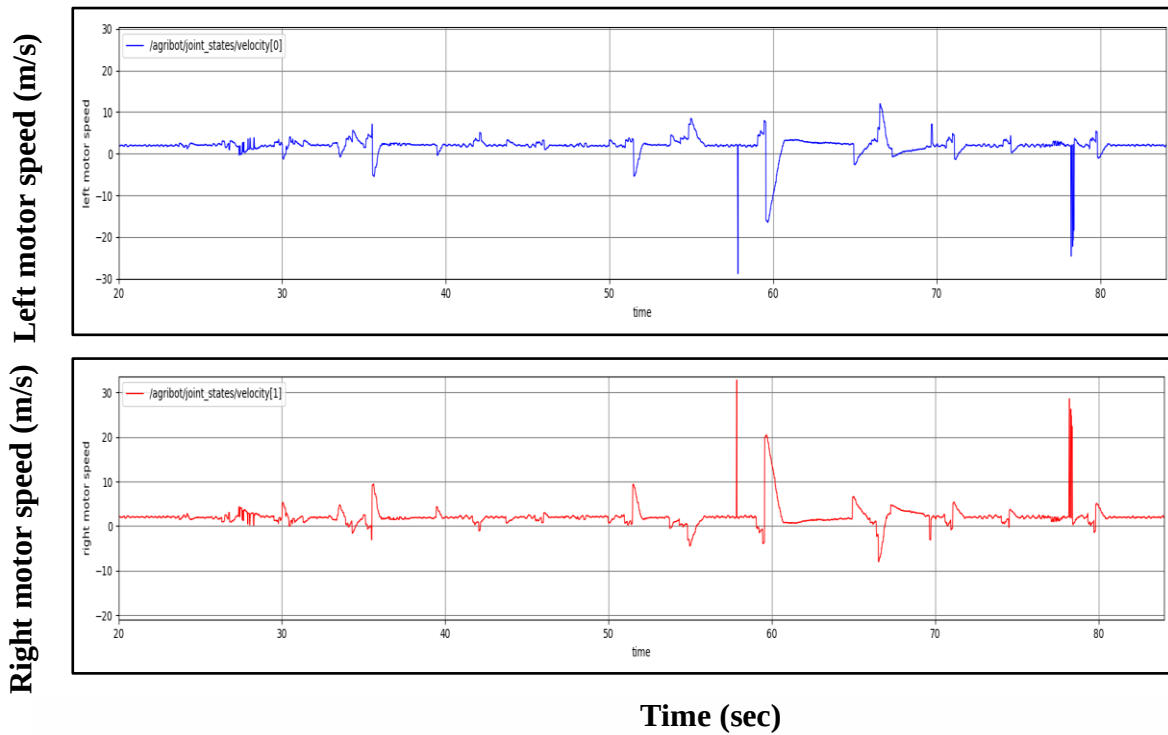


**Fig 23: Gazebo simulation environment with spawned robot model and crop field.**

#### 4.5 Simulation results (ROS)

From the simulation the angular velocity of left and right motor joints were plotted from 20 to 80 simulation time units. The plots of left and right motor speed is a mirror image as it is a differential drive control and to rotate the mobile base wheel on one side turn in one direction and set of wheels on the other side turn in the opposite directions. The spikes in certain areas of

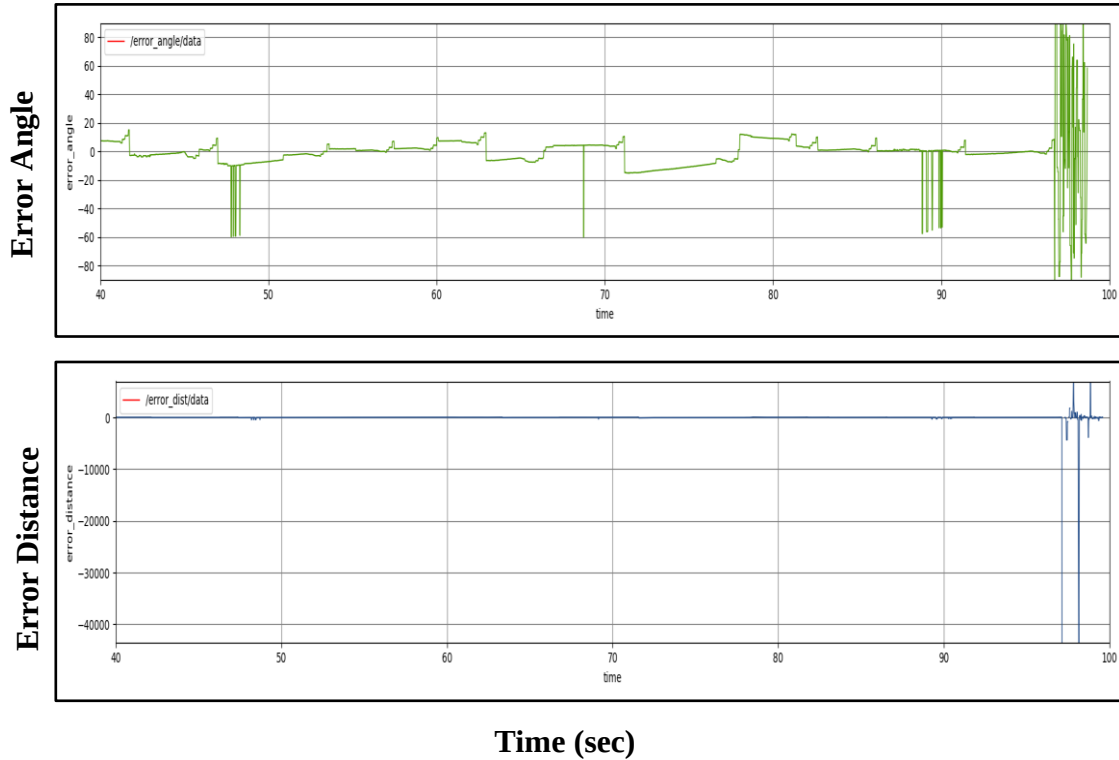
the speed vs time plot is when the vehicle is correcting its orientation with respect to the crop row line and trying to align the row line. Only angular velocity of two motors at the back are plotted since the set of wheels on one side have the same velocity again because of the differential control method.



**Fig 24: Angular velocity plots of right and left motor from simulation a)(Top)left motor speed vs time b) (Bottom)right motor speed vs time**

Also, the error angle and error distance were also plotted with respect to simulation time. At the end of the row lots of spiking in errors values is noticed. This can be explained by the fact that at the end of a row, there is a moment when only a single plant is visible to the camera. This only produces a single point for a line to be fitted according to the row detection algorithm proposed earlier. Since many lines can be passed through a single point, the line fitting function tries to fit lines in random directions. This direction keeps on changing since the algorithm runs in a loop. The drastic change in the averaged fitted line's slope or direction is manifested as a huge amount of spiking at the end of the plot when reaching the end of the crop row. In order to stop the mobile base from rotating left and right heavily during the end of a row, the angular velocity of the robot is limited to a value of 10 rad/s which was found by simulation. And if the

angular velocity computed by the proportional controller goes beyond the threshold the angular velocity is set to 0 rad/s.

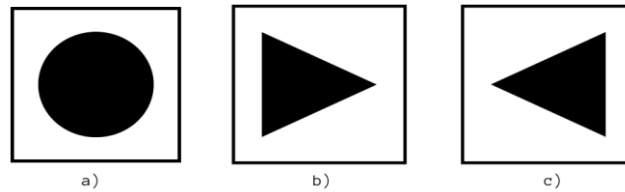


**Fig 25: Plots of error values computed from camera frame a)angle error (top) b) distance error(bottom)**

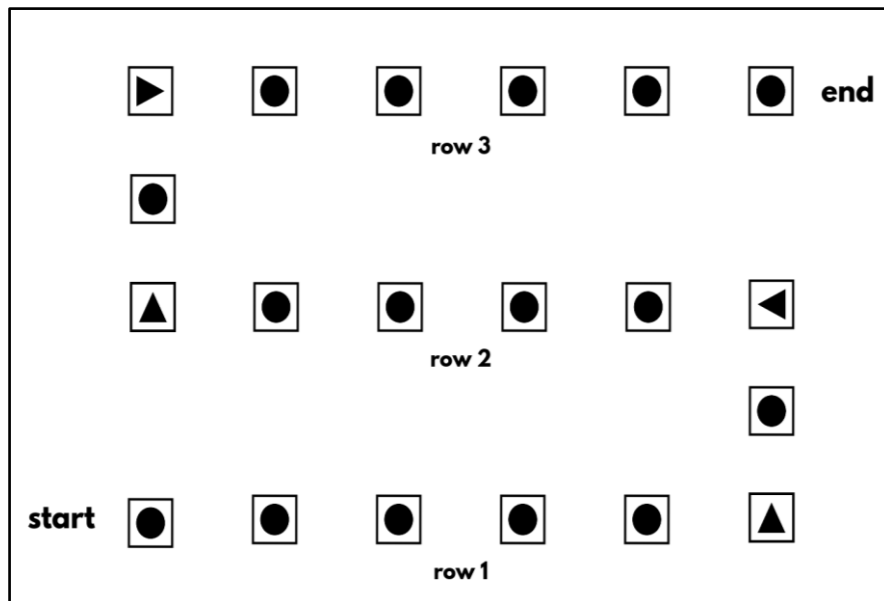
#### 4.6 Improved algorithm for crop row detection and navigation

Algorithm proposed in section 4.2 for crop row detection works by detecting plants directly in front of the robot and fitting a line, then a proportional controller tries to reduce the error angle between desired line and fitted line thus resulting in following the fitted line. The image processing technique used to identify the plant does simply filter all objects in the image falling in a certain hue value range(50 to 80). This method has a drawback that it assumes that only green objects in the crop field are plants but usually it's not the case. Furthermore, no row switching mode has been implemented and it cannot follow a row if plants are not present eg. during seeding. Thus a new algorithm was developed which solves all of the above problems. The way the new algorithm solves the above problem is by removing the dependency of detecting plants and converting to detecting specially created markers kept in the field by the user.

There are three unique markers(symbols) with distinct meanings used by the marker following algorithm as shown in Fig 26. All the markers are painted in black color to identify it using image processing. The navigation algorithm implements a simple finite state machine which is used to manage states of the robot while traversing the rows(Fig 27). The row switching is achieved by a combination of triangle and circle markers placed at the corners for turning the robot and making it enter the next row.



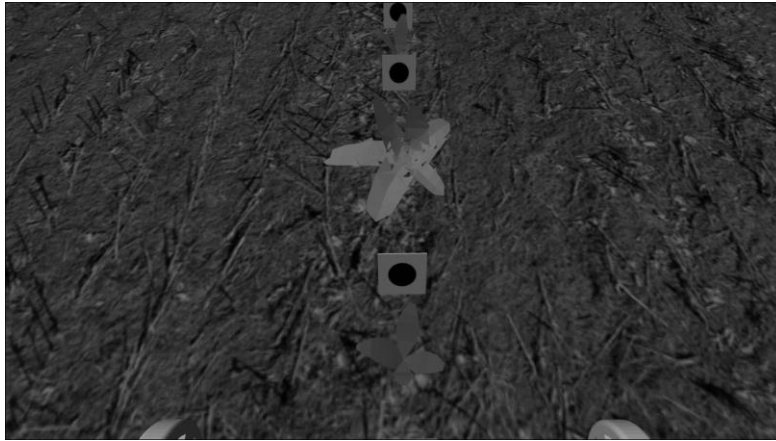
**Fig 26: Markers a) circle marker: to follow the row b) triangle right marker: to turn right c) triangle left marker: to turn left**



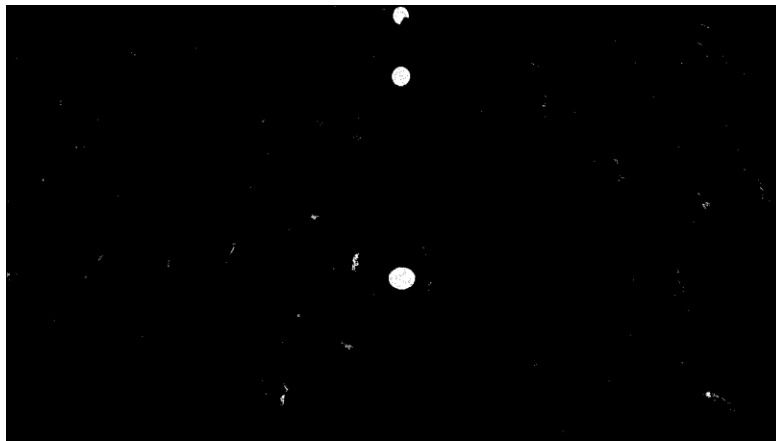
**Fig 27: top view of markers placed in rows corresponding to the rows in the field.**

Marker detection via image processing:

- Convert image received from camera into grayscale.



- Apply thresholding which colors pixel to full white(255) whose grey scale value is less than a given threshold(here it is 3). Since the markers are all painted black this thresholding will filter out only the markers black painted shapes(square and triangle).



- Next the image after thresholding is used to detect contours or clusters.
- Contours detected are used to find the polygon enclosing the contour. This step gives the number of sides of the polygon from which it is decided whether the polygon is triangle or a circle.

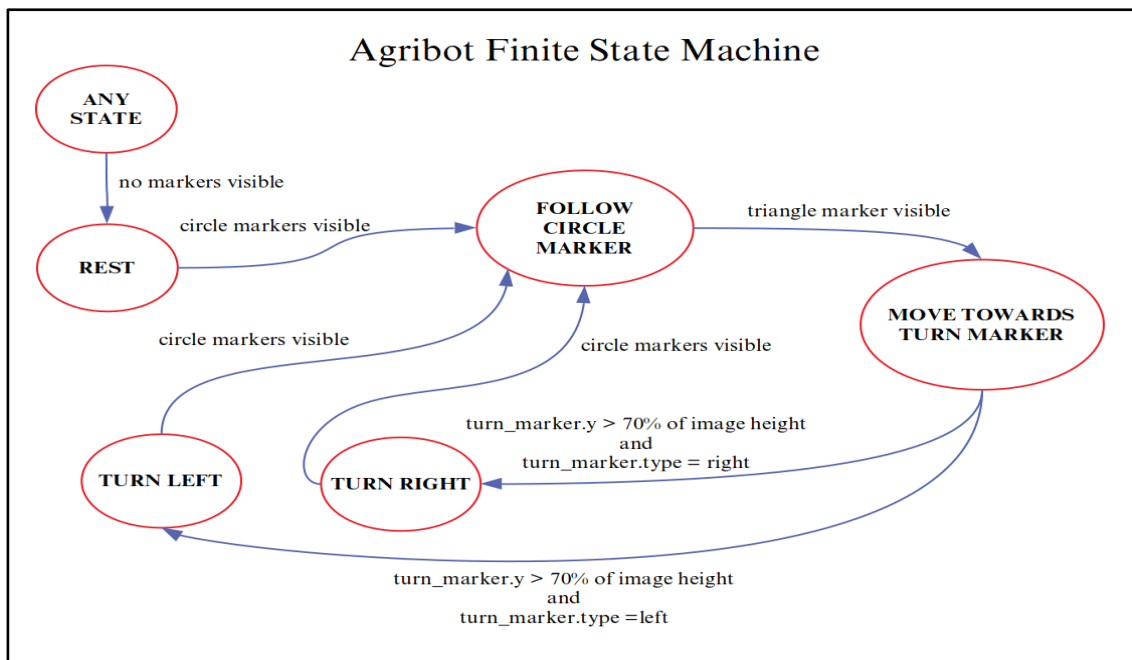


- Depending on if the marker is circle or triangle the FSM changes its state appropriately. Working of FSM is given in the next section.

- If the state is '**FOLLOW CIRCLE MARKER**', then the center point of circle markers in the image plane is calculated and these center points are used to fit a line same as the previous algorithm and these line is followed by the proportional controller based on error angle and error lateral distance.

Finite State Machine:

1. FSM starts in '**REST**' state and moves to '**FOLLOW CIRCLE MARKER**' state if circle markers are visible.
2. When one of the triangle markers is visible in the frame, the state changes to '**MOVE TOWARDS TURN MARKER**' state.
3. If triangle marker's y coordinate in the image frame is greater than 70% of the image height, then it changes to '**TURN LEFT**' or '**TURN RIGHT**' state depending on whether the triangle marker type is left or right.
4. Being in '**TURN LEFT**' or '**TURN RIGHT**' state it will change to '**FOLLOW CIRCLE MARKER**' state if the camera sees a circle marker while turning left or right.
5. When in any of the states the FSM will move to '**REST**' state if no markers(circle or triangle) are present in the image frame.

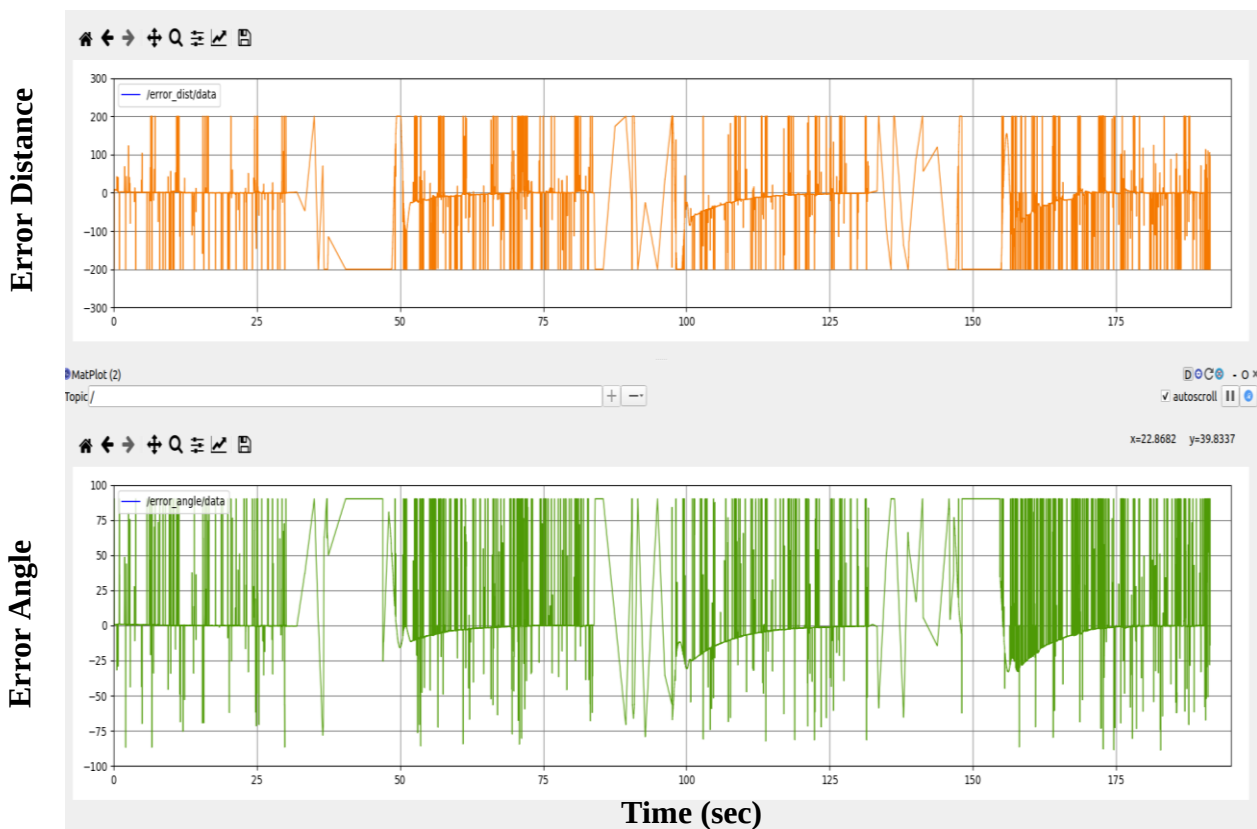


**Fig 28: agribot finite state machine for row detection and navigation**

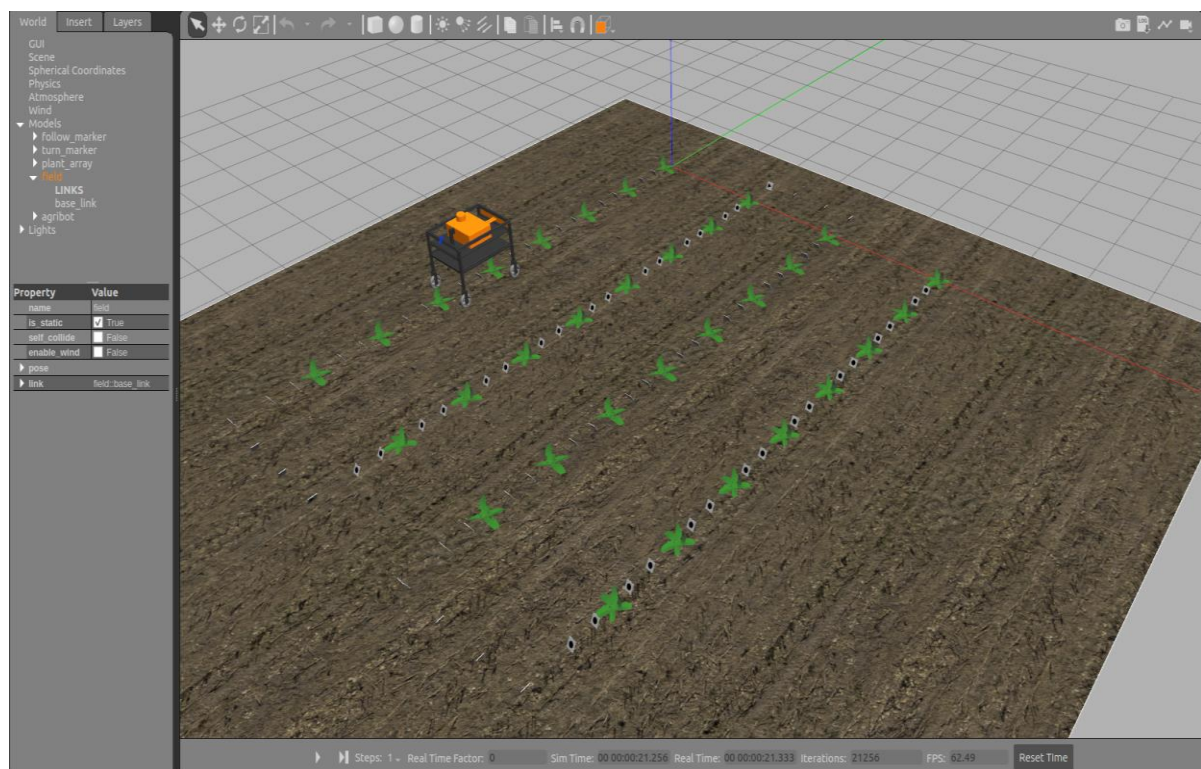


FSM state to differential drive ROS controller velocity command:

1. **REST:** forward velocity = 0 m/s, angular velocity = 0 rad / s.
2. **FOLLOW CIRCLE MARKER:** forward velocity= 0.4 m/s, angular velocity = commanded by proportional controller(aligning robot with fitted line), max. angular velocity =  $\pm 10$  rad / s.
3. **MOVE TOWARDS TURN MARKER:** forward velocity= 0.4 m/s; angular velocity = commanded by proportional controller(aligning robot towards the turn marker).
4. **TURN LEFT:** forward velocity= 0 m/s; angular velocity = -10 rad / s.
5. **TURN RIGHT:** forward velocity= 0 m/s; angular velocity = 10 rad / s.



**Fig 29: error dist v/s time (above) and error angle v/s time (below)**



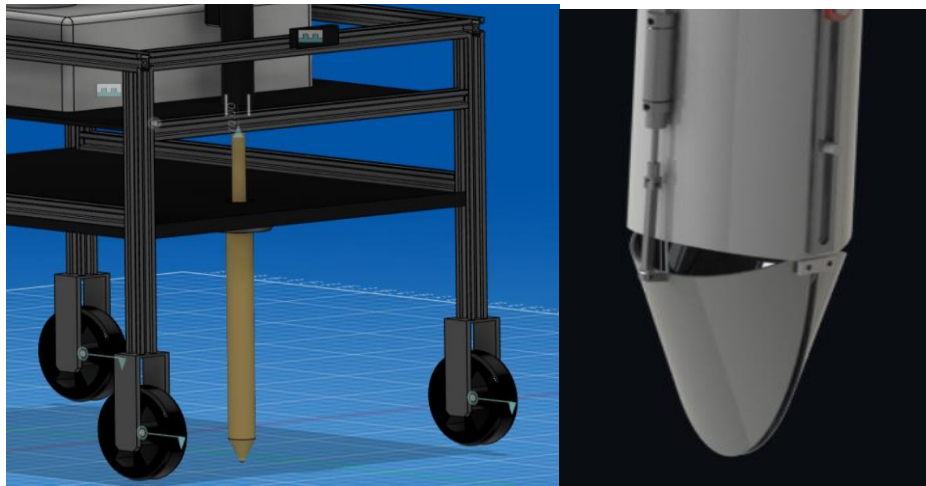
**Fig 30: Gazebo simulation environment with spawned robot model and crop field.**

## CHAPTER 5

### EXECUTION OF AGRICULTURAL TASKS

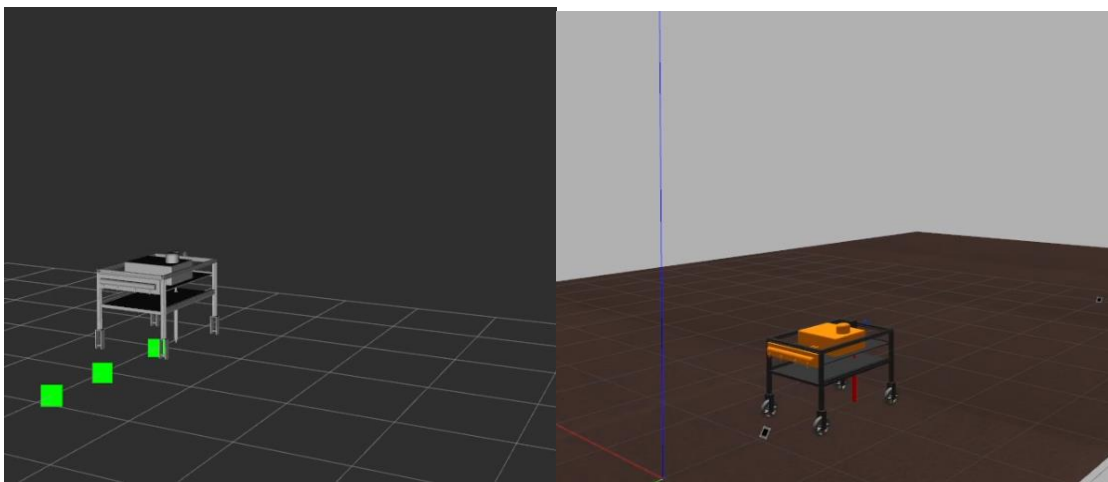
#### 5.1 Seeding:

This action is done using the seeder designed by us and its sliding motion is described by the linear actuator.



**Fig 31: a) Seeder used for simulation (Yellow) b) Actual Seeder (White)**

The seeder shown in Fig 31.a) (yellow ) is a simplified version of the actual intended seeder which is shown in Fig 31.b) (white). This was done so for optimising simulation. The seeder part contains a storage unit which can store the seeds for the entire run.



**Fig31 c) . Simulation showing seeding operation in Rviz(Left) and Gazebo(Right) in Real Time**

RViz helps in visualisation of seeding operation by placing a green marker on the region of contact of the seeder. The seeding action is triggered by square visual markers (Fig 31.c) when the mode selected is seeding. While the seeding mode is active, the linear actuator swings up

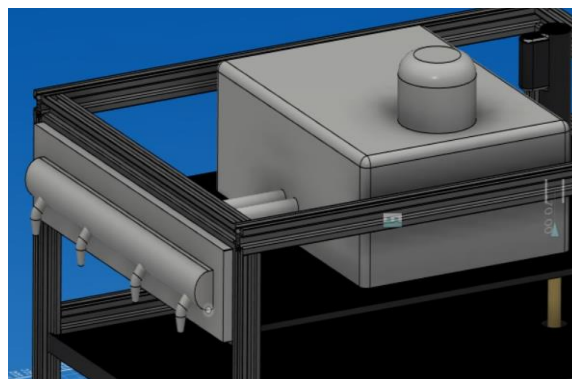
and down in the vertical direction, repeatedly penetrating the earth until the square marker emerges to terminate the operation.

The time interval with which the seeder digs can be adjusted to specify the gaps between planted seeds. For simulation, a 3 sec interval is given which means a 3 m gap (velocity = 1 m/s) between planted seeds which is not a real world example and is just taken for demonstration purposes. When the seeder is dug into the ground, the small actuators present on the either side (Fig 31.d) helps in opening and closing of the seeder when the seeding mechanism has dug into the ground. When the seeder opens the seeds are discharged in intervals as a set.



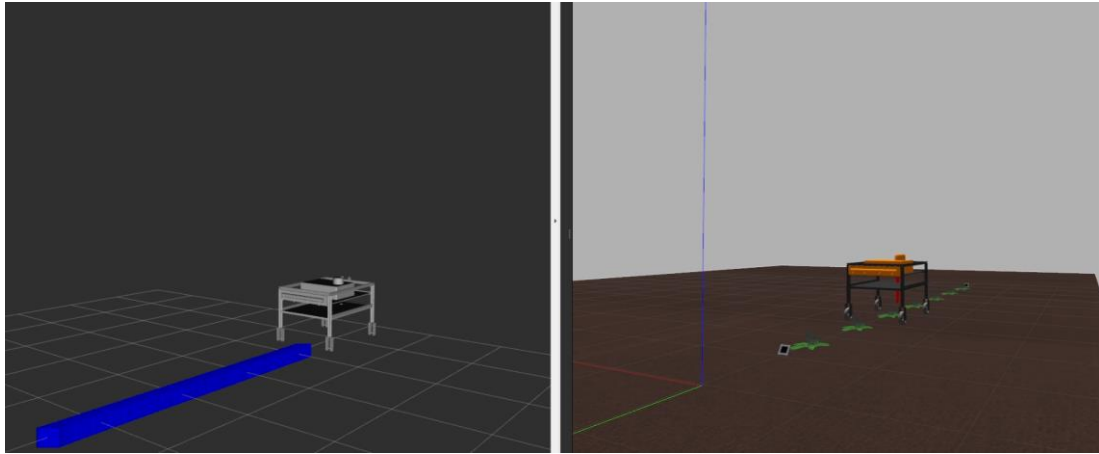
**Fig 31 d): i) Seeder Closed ii) Seeder Open**

## **5.2 Irrigation:**



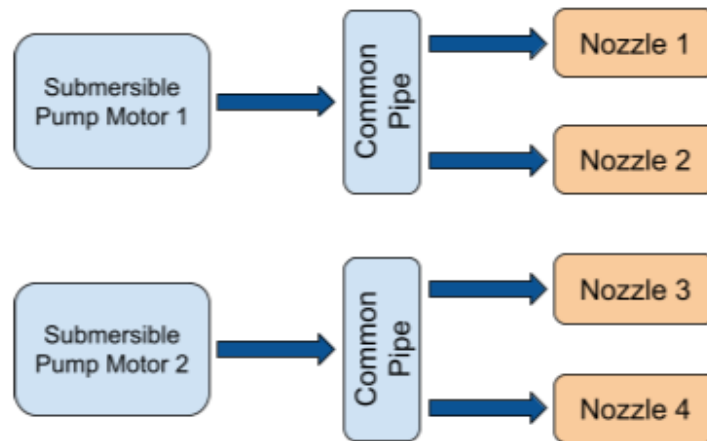
**Fig 32 a). Water Tank connected to sprayer**

RViz helps in visualisation of spraying operation by placing blue line markers on the path of the robot.



**Fig 32 b) . Simulation showing spraying operation in Rviz(Left) and Gazebo(Right) in Real Time**

The sprayer attached to the tank at the back of the robot is used for this action. Water pump motors are also housed in this tank, which spray water at a fixed flow rate. The spraying action is triggered by square visual markers (Fig 32.b) when the mode selected is spraying. The motors are connected to the nozzles as shown below. While the spraying mode is active, the water pump motors are initiated and start pumping water through the nozzles as in the layout shown below. This operation is terminated when the next square marker shows up.



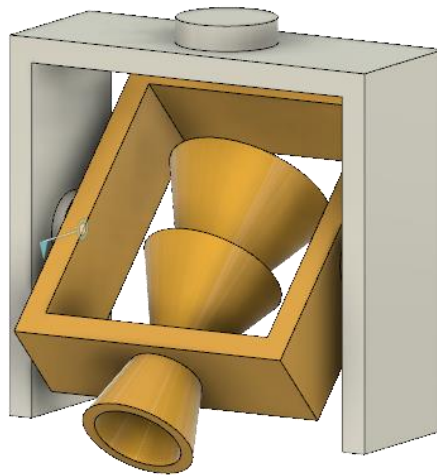
**Fig 32 c): Motor Layout in Tank**

By setting up the water pumps in the layout as shown above, it is expected to have a flow rate in the range 50 - 60 Litres Per Hour from each nozzle.

### 5.3 Weeding:

Earlier techniques involve manual ways in the removal of weeds. Two features which are being used till date are:

- Mechanical weeding keeps the soil surface loose by producing soil mulch which results in better aeration and moisture conservation. In spite of its advantages, mechanical weeding may at times kindle the growth of more weeds due to the seeds that get spilled during the process.
- Chemical weeding is mostly preferred because the weedicide application is generally more effective, easier and selective.



**Fig 33 a). Weeder**

The weeding action is triggered by square visual markers (Fig 33.b) when the mode selected is weeding. Once the weeding mode is active, a 2-axis gimbal controls the direction of the nozzle, thereby allowing us to have a better control over the exact region to be sprayed, which is detected by image processing. The weed and plant detection is done using the yolov4 darknet framework by which an accurate classification of both is obtained.



**Fig .33 b) Weed and plant detection**



### 5.3.1 Weed and Plant detection

YOLO V4 technique for real-time object detection is used to detect plants and weeds. Yolo stands for “You Only Look Once”. The input, backbone, neck, and dense network layers make up the yolo v4 framework. Yolo v4's backbone, which was employed in the project, is csp darknet 53. It's a convolution network that's been trained on the coco dataset, which includes roughly 80 classes. The feature extraction from the photos is done using the backbone. Through the neck, the retrieved features are delivered to the dense network layer. The dense network layer or the yolo layer does some data augmentation techniques to improve the prediction results. The dense network layer predicts the class that is present. The dense network layer's output is used to draw the boundary boxes. The dense network layer's output is a six-valued vector. These values aid in the identification of the image's class and the creation of bounding boxes.

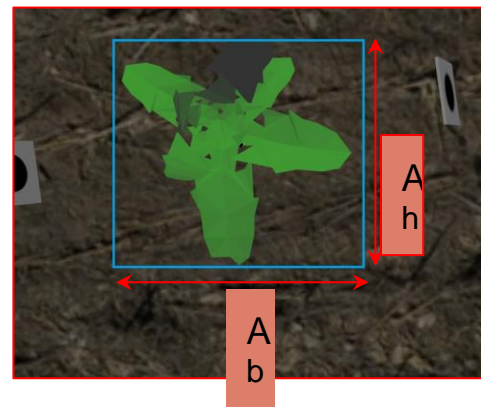
Output  $y=(P_c, A_x, A_y, A_h, A_b, C)$

$P_c$  : Probability of the class

$A_x, A_y$ : Center of the bounding box

$A_h, A_b$ : height and width of the bounding box respectively

$C$ : class name of the bounding box

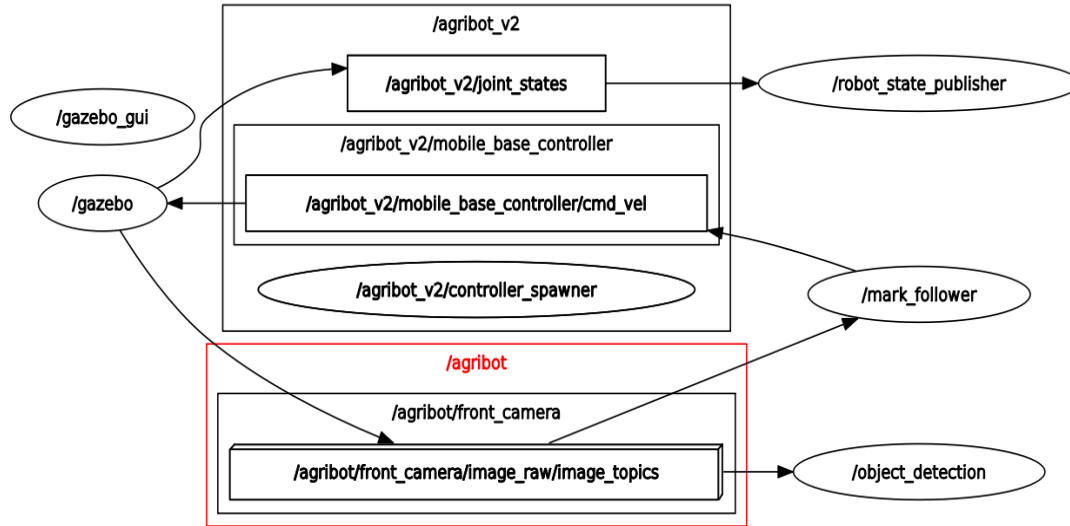


**Fig:34 a) Bounding box around predicted class**

### 5.3.2 Steps involved in training and testing

- Prepare custom dataset – Multiple pictures were taken from the gazebo simulation environment which contains plant and weed around 170 pictures were collected.
- Labelling software was used to label the positions of plants and weeds in the pictures. A text file was created which contains the class name and coordinates of the object.
- Training of objects is done in google colab by rendering the gpu that is available online. For training the model darknet framework is used. Darknet is a CNN framework tested on coco dataset. Parameters for training the model are set in the cfg file. The cfg file along with the pre-trained weight files used for feature extraction from input images. After training a weight file is generated for the custom dataset.
- The yolo model is tested using a modified cfg file and a weight file obtained during training. The object detection node which subscribes to the camera stream from the

gazebo was used to detect objects. Plants and weeds are categorised in real time, and boundary boxes are drawn around them. Threshold probability level above 0.4 is used to classify the plants and weeds.



**Fig :34 b) Object detection node subscribing from image topic**

### Parameters used for training

All the parameters for training the model were supplied through the cfg file. The learning rate for the network was 0.00261. A total of 20 convolution layers, 3 max pooling layers and 9 routes layers were used for training. Batch size of images passed for the training is 64. And a total of 2500 iterations were used for training. The model was trained till the accuracy turned into 99.2%.

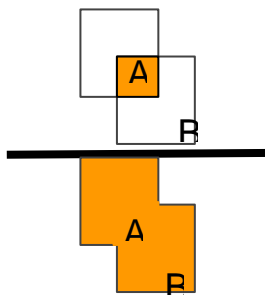
### Performance of the trained model

To test the performance of the trained model, IOU (intersection over union) and Average precision are calculated from a test data set of 110 images.

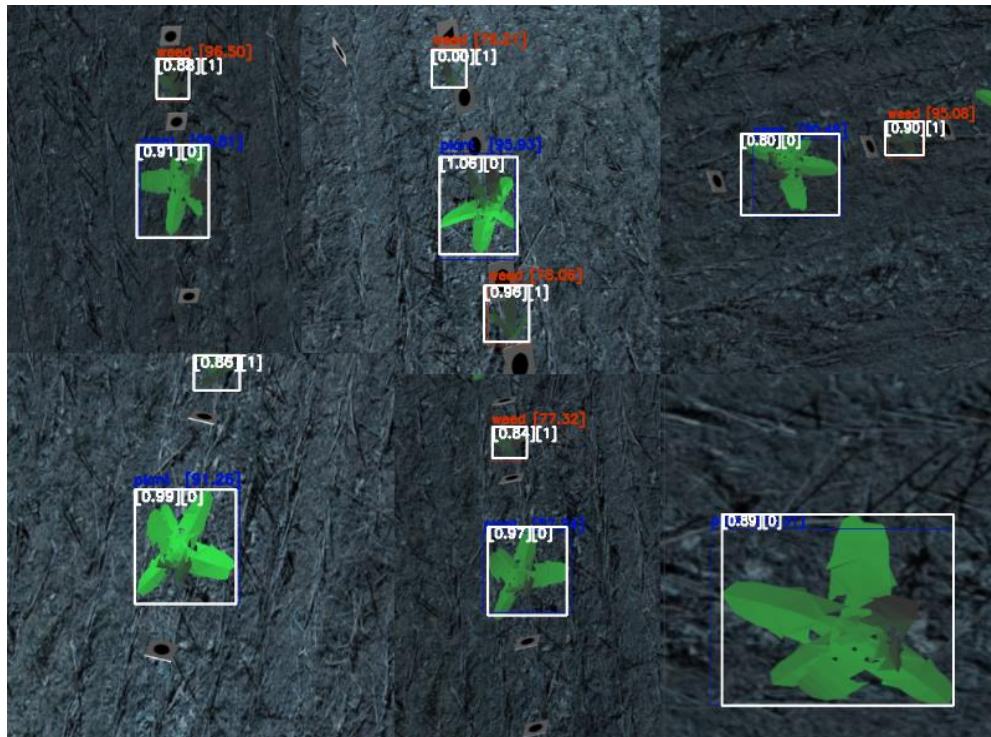
IOU(Intersection Over Union) Tested:

Intersection over union is a common error parameter used in object detection. The amount to which prediction bounding boxes and actual bounding boxes intersect is described by this terminology. IOU has a scale of 0.0 to 1.0, with 1.0 being the best scenario. IOU for the test dataset was calculated by comparing predicted and actual bounding boxes. IOU is calculated using the given formula.



$$IOU = A \cap B / A \cup B =$$


**Fig 34 c) IOU formula**



**Fig 34 d) Performance testing on test dataset**

The white bounding box is the ground truth .Blue and red bounding box are the predicted bounding boxes for plant and weed respectively. The values in white are IOU for respective bounding boxes. The values in blue and red are prediction percentage values for plant and weed class respectively. The dataset used for testing had 80 positive samples and 30 negative samples. The precision of the model is calculated by dividing true positives with total number of positive predictions.

The average IOU for the test dataset is 0.92 .

Accuracy for plant class set from the test dataset is 98.18%.

Precision for the plant class set from the test dataset is 0.9756 .

Accuracy for weed class set from the test dataset is 94.54%.

Precision for the weed class set from the test dataset is 0.9625 .

Confusion matrix:

**For plant:**

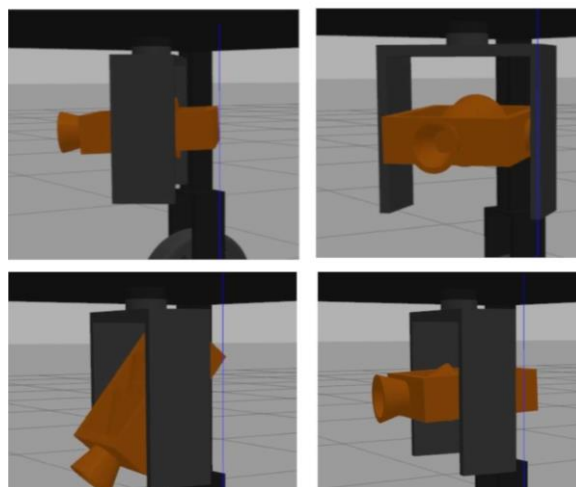
N=110	Positive (Plant)-Predicted	Negative(Plant) -Predicted
Positive(Plant)-Actual	80	0
Negative(Plant)-Actual	2	28

**For weed:**

N=110	Positive (Weed)-Predicted	Negative(Weed) -Predicted
Positive(Weed)-Actual	77	3
Negative(Weed)-Actual	3	27

**Table 4: Confusion Matrix for plant and weed**

From the confusion matrix the true positive cases and true negative cases are high. This indicates the high precision and accuracy of the trained model. Chemical method of weed removal is implemented through the above shown mechanism , which involves the sprinkling of ash and powdered farmyard manure. The weeder is operated by spraying air at high pressure through a specially designed nozzle where the air mixes with liquid or powder and then exits via a nozzle. The weeder's range of rotation to target the weeds while spraying is shown in the diagram below. Two steppers motors are connected, allowing yaw and pitch rotation. When the next square marker appears, the operation is terminated.



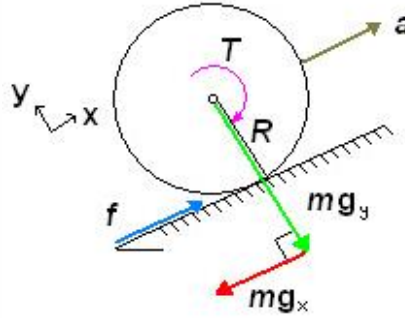
**Fig . 34 e) 2 Axis Gimbal (Yaw and Pitch Rotation)**

## CHAPTER 6

### MATERIALS AND SPECIFICATIONS

#### 6.1 Selection of Drive Motor

A DC motor is used for the drive system. The specifications required for the motor are calculated as follows from the values of base parameters.



**Fig 35: Free body diagram of the robotic wheel [10]**

On considering the free body diagram;

$$mg_x = mg \sin \theta \quad (13)$$

$$mg_y = mg \cos \theta \quad (14)$$

For the robot not to slide down the incline, there should be a friction between the wheel and the surface, which produces a torque. Also, it is considered to accelerate along the incline, to account for the irregularities on the surface of the field.

Torque required is:  $T = f \cdot R \quad (15)$

On balancing the force equation,

$$\sum F_x = M \cdot a = f - m \cdot g_x \quad (16)$$

Inserting the equation for torque above, and the equation for  $mg_x$ :

$$M \cdot a = T/R - M \cdot g \sin \theta \quad (17)$$

$$T = R \cdot M \cdot (a + g \sin \theta) \quad (18)$$

This torque value represents the total torque required to accelerate the robot up an incline. For obtaining the torque needed for each drive motor,

$$T = ((a+g*\sin \theta)*M*R)/N \quad (19)$$

On consideration of the total efficiency of the system as e:

$$T = (100/e) * ((a+g*\sin \theta)*M*R)/N \quad (20)$$

The value for efficiency here represents the total efficiency of the motor and can be estimated as follows. The individual efficiency of the components are assumed to be their typical values.

- Battery ~ 96% efficient
- Motor Driver ~ 95% efficient
- DC motor~90% efficient

so the total efficiency of the DC motor would be:  $96\% \times 95\% \times 90\% = 82\% \sim 80\%$

Total power (P) per motor can be calculated using the following relation:

$$P = T*\omega \quad (21)$$

,where ‘w’ is the angular velocity of the system. Hence, from the following base parameters, the specifications of DC motor as:

#### **Base parameters**

Mass of the robot : 24.2 kg

Maximum weight :75 kg ( with fully filled water tank)

Number of drive motors: 4

Radius of Drive wheel: 0.1 m

Robot velocity: 1 m/s

Maximum Incline : 4 deg

Supply Voltage: 12 V

Desired acceleration:  $0.01 \text{ m/s}^2$

#### **Specifications:**

Angular velocity : $10 \text{ rad/s} = 95.49 \text{ rpm}$

Torque:1.6273 Nm

Power: 16.273 W

Based on these requirements, the following DC motor is selected for the actuation in the Multipurpose Agricultural Robot.The specifications of the motor are as follows.

## DC motor:



**Fig 36 a) DC motor**

<b>Product Name</b>	Orange OG555 High Torque DC Motor (SKU: 764983)
<b>Supply Voltage</b>	12 V DC
<b>Power rating</b>	18 W
<b>Output speed</b>	100 rpm
<b>Maximum Output Torque</b>	1.73 Nm
<b>Current Rating</b>	4.134 A
<b>Motor body length</b>	94 mm
<b>Motor body diameter</b>	37 mm

## 6.2 Battery Specs:

### 1) 12 Battery:

Torque of a motor is 1.63 Nm, so a 12V battery is chosen of matching specifications:

Nominal Voltage of 12 V and a motor current of 5 A. For seeder, a motor current of 2 A is needed. Also 4 motors with a total current requirement of almost 20 A have to be considered too.

From this, the wattage requirement of each motor is=  $12 \times (5 \times 4 + 2) = 264 \text{ W}$ .

Let us take the average running time of our system to be 1 Hr 48 min ~ 2 Hrs.

Then, the total battery capacity will be =  $22 \times 1.8 = 40 \text{ Ah}$

Energy =  $(0.264) \times (1.8) = 0.475 \text{ kWh}$ .



**Fig 36 b) 12V Battery**

<b>Product Name</b>	Luxnur (TM) Portable Power Power Bank Lithium Battery
<b>Voltage</b>	12 V
<b>Capacity</b>	40 Ah
<b>Dimension</b>	9.37 x 6.77 x 7.99 inches
<b>Weight</b>	148 g

## 2) 9V Battery:

A 9 V battery is used for powering water pumps( 2 Nos) and the camera.

Nominal Voltage of 12 V and a current of 3 A. From this, the wattage requirement of each motor is=  $9 \times (3) = 27 \text{ W}$ .

Let us take the average running time of the system to be 1 Hr 48 min ~ 2 Hrs.

Then, the total battery capacity will be =  $3 \times 2 = 6 \text{ Ah}$ .

Energy =  $(27) \times (2) = 0.054 \text{ kWh}$ .



**Fig 36 c) 9V Battery**

<b>Product Name</b>	LFP (LiFePo4) Battery Pack
<b>Voltage</b>	9 V
<b>Capacity</b>	6 Ah
<b>Dimension</b>	Max. 32.5×70.5mm

### 6.3 Electronic Components:

#### 1) Micro Computer:



**Fig 36 d) Micro Computer**

The Coral Dev Board Mini is a single-board computer that provides fast machine learning (ML) inferencing in a small form factor, the on-board Edge TPU coprocessor is capable of performing 4 trillion operations (tera-operations) per second. It can act as a fully-functional embedded system with SoC + ML + wireless connectivity.

<b>CPU</b>	MediaTek 8167s SoC (Quad-core Arm Cortex-A35)
<b>GPU</b>	IMG PowerVR GE8300 (integrated in SoC)
<b>ML accelerator</b>	Google Edge TPU coprocessor: 4 TOPS (int8); 2 TOPS per watt

<b>RAM</b>	2 GB LPDDR3
<b>Flash memory</b>	8 GB eMMC
<b>Input/output</b>	40-pin GPIO header; 2x USB Type-C (USB 2.0)

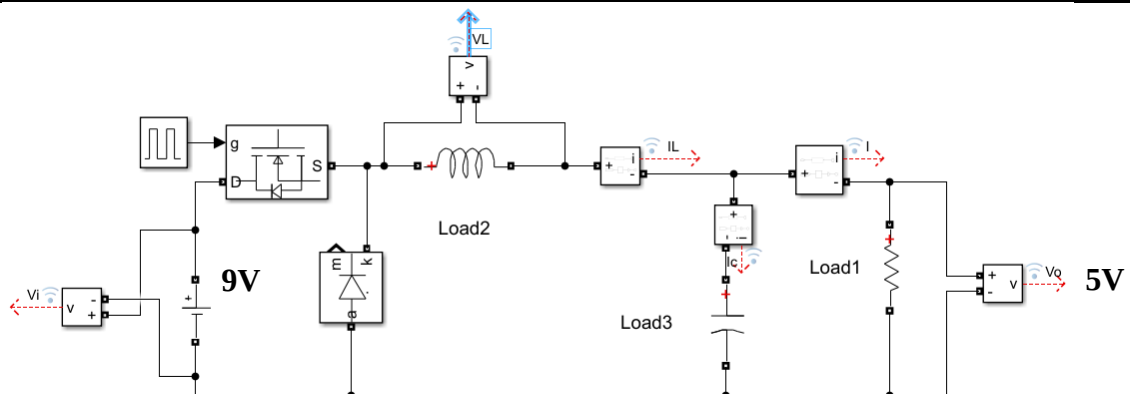
## 2) DC - DC 9V - 5V Converter:

This converter steps down the 9 V to 5 V which is used to power the 2 water pump motors, camera module and 2 stepper motors in the weeder.



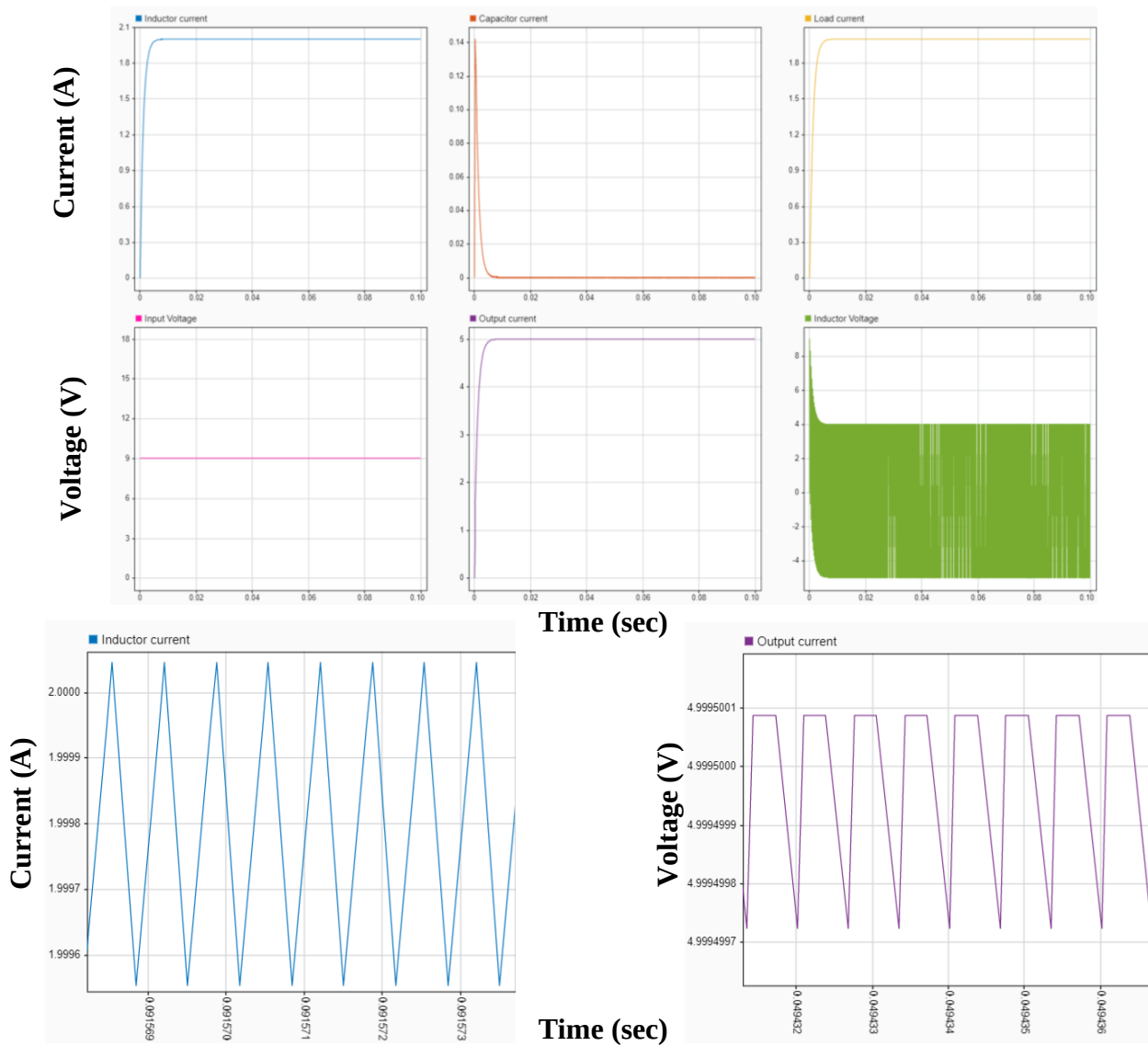
**Fig 36 e) DC/DC Converter**

<b>Product Name</b>	DC-DC USB Step Down Converter Module KG344
<b>Input Voltage</b>	9 V
<b>Output Voltage</b>	5 V
<b>Output Current</b>	3 A
<b>Dimension</b>	6 x 5 x 3 cm
<b>Max Switching Frequency:</b>	1.5 MHz



By running simulations on the required DC/DC Converter, the desired 5V was obtained within fraction of a second.





The current ripple in the Inductor (3 mH) is 0.9 mA and the voltage ripple in Capacitor (40  $\mu$ F) is 3.6  $\mu$ V. These results give an indication that the device is working as intended.

### 3) Motor Driver IC:

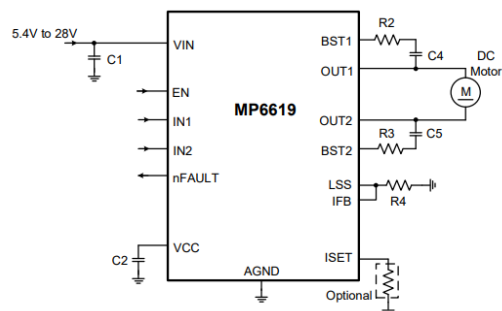


Fig 36 f) Motor Driver IC

<b>Product Name</b>	MP6619GQ-P
<b>Input Voltage</b>	5.4 V to 28 V
<b>Output Current</b>	5 A
<b>Dimension</b>	3 * 3 mm

#### 4) Camera:



**Fig 36 g) Camera Module**

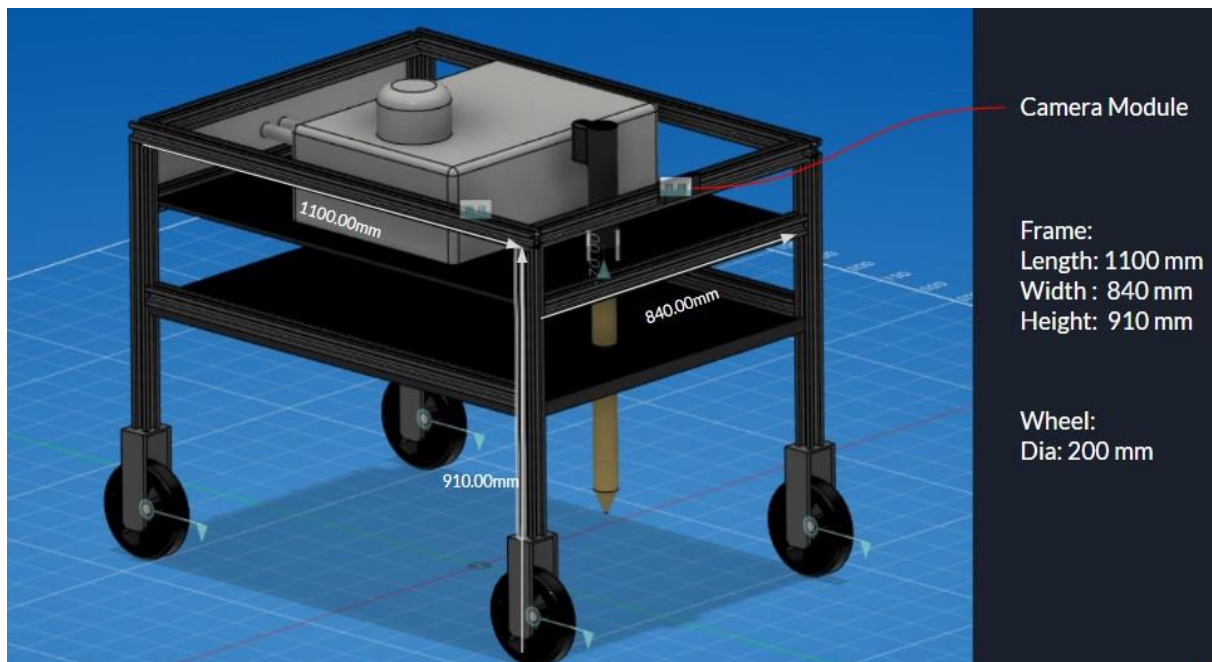
<b>Camera Type</b>	USB Camera
<b>Maximum Image &amp; Transfer Rate</b>	1920x1080 @ 30fps MJPEG/ 1280x720 @ 30fps MJPEG
<b>Resolution</b>	5MP
<b>Sensor</b>	Aptina MI5100 1/2.5 inch
<b>Max. Resolution</b>	2592 *1944
<b>Operating Voltage</b>	5V DC
<b>Power Consumption</b>	150mW - 200mW
<b>Dimension</b>	4.1 * 4.1 * 3 cm
<b>Weight</b>	150 g

This camera is attached to the front of the robot and the visuals obtained from it are used for image processing for both path following and classification.

## 6.4 Materials Used:

### 1. Frame:

One of the most common choices for building robots are metals as they are strong, rigid, hard, tough, heat resistant, and isotropic. Metals are typically shaped by forging, bending, machining (sawing, drilling, milling turning), grinding or casting. Metals are basically unmatched in their combination of strength (hardness) and toughness. Some of the most commonly used metal or its alloys in robots are steel, aluminium, copper, bronze etc. In this project, Aluminium Extrusion is chosen to build the frame of the robots because of the following reasons. Aluminum is one of the lighter metals, decreasing the overall weight of the robot. Another advantage with the aluminium is that it does not rust over time. Since it is a softer metal, it becomes easier to machine with hand tools like drills and saws.



**Fig 37 a): Structural frame of Robot**



**Fig 37 b): Aluminium Extrusion**

<b>Product Name</b>	Aluminium Extrusion Profile
<b>Profile Size</b>	20*20
<b>Extrusion Length</b>	500 mm
<b>Material</b>	Aluminium Alloy 6030

## 2. Wheels:

The PU8074-M20 by Richmond is a 200mm diameter polyurethane wheel with a maximum capacity of 600 kg and has a tyre width of 50mm. The axle diameter is 20mm.



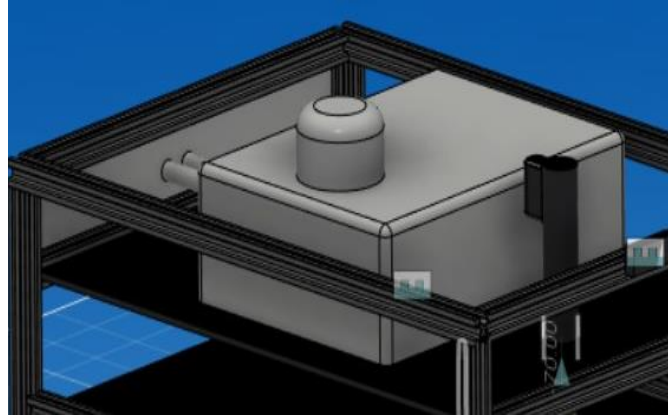
**Fig 37 c): PU8074-M20 wheel**

<b>Product Name</b>	PU8074-M20
<b>Diameter(mm)</b>	200 mm
<b>Tyre Width(mm)</b>	50
<b>Wheel Type</b>	Polyurethane
<b>Axle Diameter</b>	20mm
<b>Bearing Type:</b>	Ball Bearing
<b>Load Capacity(kg):</b>	600/500

Polyurethane wheels have very good load bearing capacities, are highly resistant to wear, tear and general impact damage. Polyurethane wheels and castors are highly durable products with

good resistance to many chemicals, greases and oils. Polyurethane wheels maintain their shape well and can operate efficiently at both high and low temperatures.

### 3.1 Water Tank:

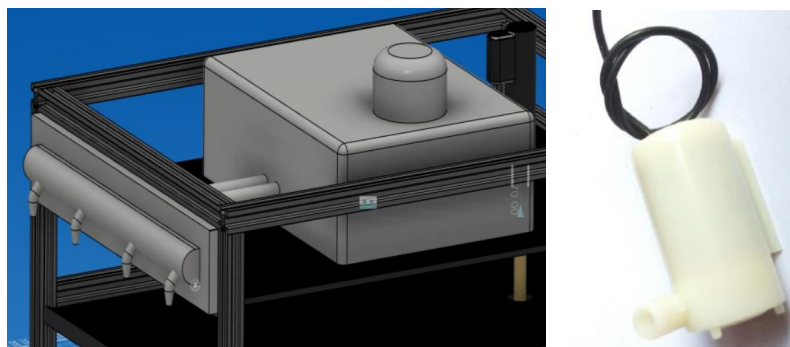


**Fig 37 c): Submersible Pump Motor**

<b>Material</b>	Plastic (Polyethylene)
<b>Maximum Capacity</b>	54 L
<b>Dimensions</b>	60 × 45 × 20 cm
<b>Weight (Fully Filled)</b>	54 kg

The primary reason why Plastic Tanks is used is because of its affordability and availability. Polyethylene, which is the raw material used to manufacture such tanks using solidified rubber with ideal balance between being too brittle and flexible. Apart from that, they are lightweight and can also endure mechanical stresses fairly.

### 3.2 Water Pump Motor (in Tank):

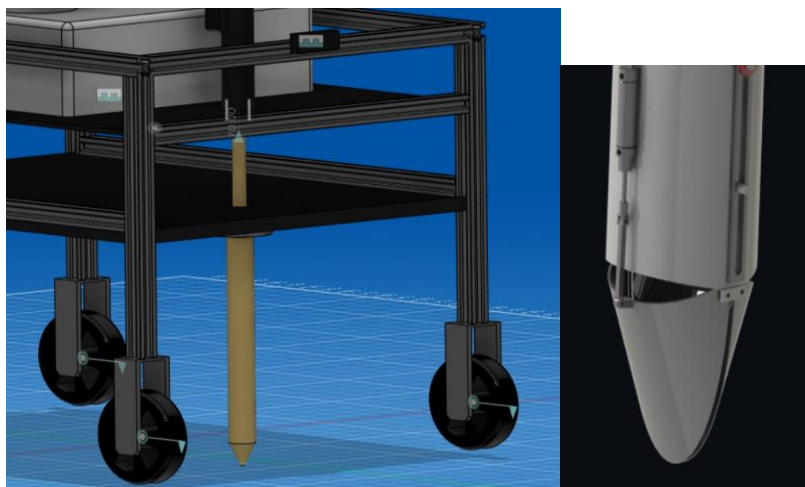


**Fig 37 d): Submersible Pump Motor**

<b>Brand Name</b>	SR ROBOTICS
<b>Item Weight</b>	120 g
<b>Material</b>	Acrylonitrile Butadiene Styrene
<b>Maximum Flow rate</b>	100 Litres per hour
<b>Voltage</b>	2.5 ~ 9 V
<b>Maximum Rated Current</b>	0.25 A
<b>Power</b>	0.36 W
<b>Pump Inlet/Outlet Diameter</b>	7 mm
<b>Part Number</b>	SR45
<b>Power Source Type</b>	Battery Powered
<b>Number of items</b>	2
<b>Package Dimensions</b>	13.2 x 13.1 x 2.5 cm

A small size submersible pump motor is used, which can be operated from a 2.5 ~ 9V power supply. It can take up to 100 Liters Per Hour with very low current consumption of 220ma. Features include - Smooth operation, high efficiency, good performance, long service life. It uses advanced electronic components and a wear-resistant shaft.

#### 4.1 Seeder:

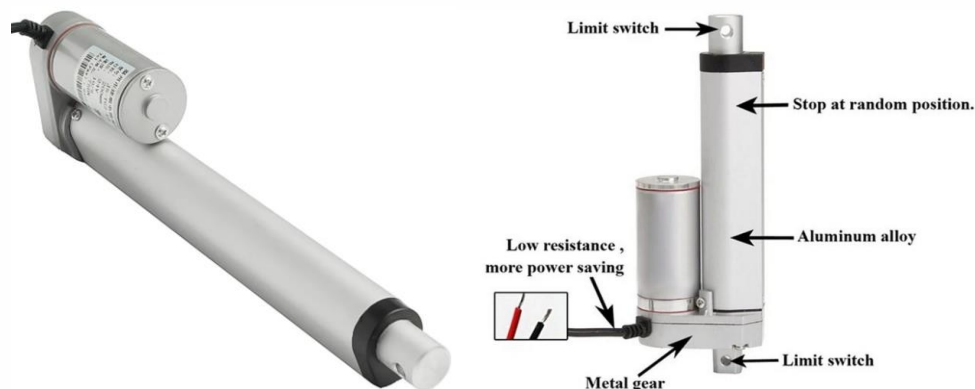


**Fig 37 e): i) Seeder used for simulation (Yellow) ii) Actual Seeder (White)**

The seeder shown in Fig 37.e) (i) (yellow ) is a simplified version of the actual intended seeder which is shown in Fig 37.e) (ii) (white). This was done so for optimising simulation.

<b>Material</b>	ABS (acrylonitrile butadiene styrene)
<b>Seed Storage Height</b>	49 cm
<b>Seed Storage Radius</b>	2.5 cm
<b>Seed Storage Volume</b>	961.625 cm <sup>3</sup>
<b>Total Height</b>	80 cm

#### 4.2 Linear Actuator (Connected to seeder):



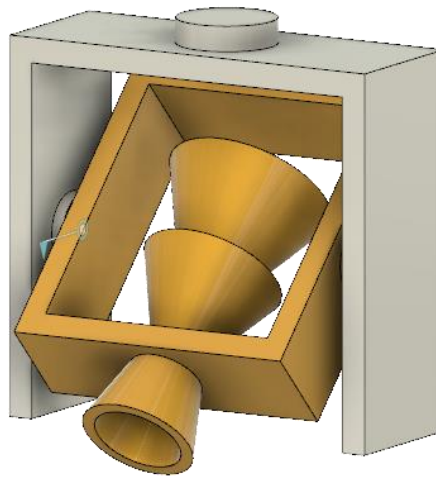
**Fig 37 f): 200 mm Stroke Length Linear Actuator**

This Miniature Linear Actuator Stroke Length 200MM,1500N,12V is an electric push rod and an ideal solution for industrial, agricultural machinery, construction and many other applications. This electric push rod is a device which transforms the rotary motion of a motor into the linear reciprocating motion of the push rod. The electric push rod is composed of drive motor, reduction gear, guide sleeve, push rod, sliding seat, shell, micro motion control switch.

<b>Brand Name</b>	Generic
<b>Product Name</b>	Linear Actuator
<b>Stroke Length</b>	200 mm

<b>Permanent magnet DC motor drive</b>	12V DC
<b>Operating temperature</b>	-20°C to +63°C
<b>Maximum static load</b>	337.2 lbs. ( 1500 Newtons )
<b>Maximum current</b>	2 A
<b>Dimensions</b>	33 × 9 × 4.5 cm
<b>Weight</b>	1.05 kg

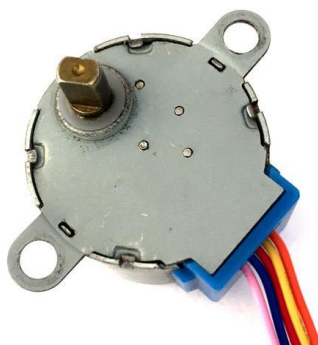
### 5.1 Weeder:



**Fig 37 g) : Weeder**

<b>Material</b>	ABS (acrylonitrile butadiene styrene)
<b>Dimensions</b>	75 * 30 * 5 cmm

### 5.2 Stepper Motor for Yaw and Pitch Rotation (in Weeder)



**Fig 37 h) : Stepper Motor**



<b>Product Name</b>	28BYJ-48 Stepper Motor
<b>Torque</b>	300 gf-cm
<b>Voltage</b>	5 V
<b>Frequency</b>	100 Hz
<b>Insulated Power:</b>	600VAC/1mA/1s
<b>Diameter</b>	25 mm
<b>Step Angle</b>	5.625 deg

### 6.5 Estimation of Cost for Hardware Implementation

Sl No	Component	Cost /piece	Quantity	Total Cost(Rs)
1	DC Motor	699	4	2796
2	Motor Driver Circuit	244	4	976
3	Battery (12 V)	10500	1	10500
4	Battery (9 V)	1028	2	2056
5	DC-DC converter	249	3	747
6	Camera	3482	1	3482
7	Micro Computer	7653	1	7653
8	Aluminium Extrusion (for frame)	379	12	4548
9	Aluminium sheet (for body)	400	2	800
10	Wheels	982	4	3928
11	Water Tank	400	1	400
12	Water Pump Motor	249	2	498
13	Linear Actuator	4500	1	4500
14	Seeder & Weeder (for material)	175	1 kg	175
15	Stepper Motors (for yaw and pitch rotation)	97	2	194

**Table 5: Cost Estimation**

**Total cost of all components and materials = Rs 42603**

This design is substantially cheaper than existing market alternatives, and those robots are only capable of doing a single task.

## **CHAPTER 7**

### **CONCLUSION**

#### **7.1 Conclusion**

In agriculture, the opportunities for robot enhanced productivity are immense and robots are appearing on farms. The other problems associated with autonomous farm equipment can probably be overcome with technology. Crop production may be done better and cheaper with small machines than with a few large ones. The main focus of the project was to develop a multipurpose agricultural robot that reduces the human effort, time and cost and increases productivity.

In this work, firstly, a CAD model of the proposed architecture is designed with the help of the Solidworks platform. The model is then imported to MATLAB using SimMechanics link Add-in which helps to assemble the designed parts into multibody dynamics. The model is tested using basic parameters and controlled to verify the movements of joints. Actuation for rotation of the wheels is given by DC motors, whose speed needs to be controlled. A PID controller was designed for this purpose. A virtual world was created in the Simulink environment for the robot to perform its task. After which, the various control strategies are implemented to automate the specific tasks given to the robot. This includes image processing algorithms like OpenCV that will help the robot in object detection, tracking path planning. The Stateflow implementation is also explored, which enables the robot to make real-time decisions based on the conditions and given input, without any manual interference. Additionally, a plant detection system is developed in ROS to further assist in various farming applications. Detection of crop rows is done in the ROS platform by processing images received from a camera fitted at the front of the robot chassis. This camera gives a visual feedback of forthcoming crop row structure and helps the robot steer accordingly. Image processing is done using OpenCV C++ library. The use of markers assists in row navigation. Weed detection is performed using YOLO V4 Darknet Framework. Apart from this, a visualisation tool, ROS RViz tool is used to visualise agricultural functions like spraying and seeding. Finally, an estimation of costs for implementing the hardware model of the multipurpose agricultural robot is performed. It is expected that the robot will support the farmers in improving the efficiency of operations in their farms.

#### **7.2 Future Scope of Work**

Through this project, a simulation model has been created of a multipurpose agricultural robot

that performs various agricultural functions such as seeding, weeding and watering . In the future, a hardware model of an agrobot that performs the agricultural tasks in the field in real time can be developed. The robot can be built in the full scale version incorporating more features like ploughing and harvesting. Also, solar powered batteries can be used to power the motors and other components, which would help to reduce the electricity consumption and thereby reduce the operating charges.

Some of the limitations of the robot are as follows. Since the navigation is based on the markers, the robot is only autonomous once it is in the agricultural field, human effort will be required to bring the robot to the field. The Watertank in the robot is of fixed capacity which is only desirable for small and medium sized fields, for operating in large fields ,the water tank would need to be constantly refilled . The machine learning algorithm used for the weed detection has an accuracy of about 91 percent, in the future the possibility of implementing a more accurate and efficient algorithm can be explored. This will improve the quality and effectiveness of the robot.

### **Novelty Report**

Most of the robots available in the market are specifically designed to perform a single agricultural task. In this project, the possibility of incorporating multiple functional units in the single robot for performing different agricultural tasks have been explored. For navigation, a marker based algorithm is used, instead of a GPS based one. The markers have been used to decrease the cost of computational processing.

## REFERENCES

- [1] K. D. Sowjanya, R. Sindhu, M. Parijatham, K. Srikanth and P. Bhargav, "Multipurpose autonomous agricultural robot," 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), 2017, pp. 696-699, doi: 10.1109/ICECA.2017.8212756.
- [2] J. B. Han, K. M. Yang, D. H. Kim and K. H. Seo, "A Modeling and Simulation based on the Multibody Dynamics for an Autonomous Agricultural Robot," 2019 7th International Conference on Control, Mechatronics and Automation (ICCMA), 2019, pp. 137-143, doi: 10.1109/ICCMA46720.2019.8988607.
- [3] Paulo Flores,"Fundamental Concepts in Multibody Dynamics", Springer Briefs in Applied Sciences and Technology 168, In book: Concepts and Formulations for Spatial Multibody Dynamics, March 2015,, doi:10.1007/978-3-319-16190-7\_2
- [4] Alexa Sanchez, 'Creating Virtual Robot Environments in Simscape: Assembling Robots and Importing from CAD',, August 2020, Mathworks
- [5] Simscape Multibody - MATLAB & Simulink (mathworks.com)
- [6] Control Tutorials for MATLAB and Simulink - Motor Speed: Simulink Simscape (umich.edu)
- [7] Ahmadi, Alireza and Nardi, Lorenzo and Chebrolu, Nived and Stachniss, Cyrill, "Visual servoing-based navigation for monitoring row-crop fields", 2020 IEEE International Conference on Robotics and Automation (ICRA).
- [8] Visual servoing-based navigation for monitoring row-crop fields - <https://github.com/PRBonn/visual-crop-row-navigation>.
- [9] Dudek and Jenkin, "Differential Drive Robots", Computational Principles of Mobile Robotics.
- [10] C benson, Dynamic Tools Mechanics, Drive Motor Sizing Tutorial, 2014

## Appendix:

### Code:

mark\_follower.cpp :

```
1.  #include <ros/ros.h>
2.  #include <sensor_msgs/image_encodings.h>
3.  #include <geometry_msgs/Twist.h>
4.  #include <std_msgs/Float32.h>
5.
6.  #include <cv_bridge/cv_bridge.h>
7.  #include <image_transport/image_transport.h>
8.  #include <opencv2/imgproc/imgproc.hpp>
9.  #include <opencv2/highgui/highgui.hpp>
10.
11. #include <vector>
12.
13. using namespace std;
14. using namespace cv;
15.
16. float error_angle = 0.0;
17. float error_dist = 0.0;
18.
19. unsigned int frameCount = 0;
20.
21. enum ROBOT_STATE
22. {
23.     REST,
24.     FOLLOW,
25.     TURN_LEFT,
26.     TURN_RIGHT,
27.     MOVE_TOWARDS_TURN
28. };
29.
30. enum DIRECTION
31. {
32.     LEFT,
33.     RIGHT
34. };
35.
36. unsigned int state = REST;
37.
38. char *StateToString(int state)
39. {
40.     switch (state)
41.     {
42.         case REST: return "rest";
43.         case TURN_LEFT: return "turn left";
44.         case TURN_RIGHT: return "turn right";
45.         case FOLLOW: return "follow";
46.         case MOVE_TOWARDS_TURN: return "towards turn sign";
47.         default: return "unknown state";
48.     }
49. }
50.
51. // given three points of a triangle returns pointing direction(assuming triangle marker is isosceles)
52. int GetTriangleDirection(vector<Point> points)
53. {
54.     Point sorted[3];
55.
56.     //sorting points based on distance from x axis
57.     if(points[0].x >= points[1].x)
58.     {
59.         sorted[0] = points[0];
60.         sorted[1] = points[1];
61.     }
62.     else
63.     {
64.         sorted[0] = points[1];
65.         sorted[1] = points[0];
66.     }
67.
68.     if(points[2].x >= sorted[0].x)
69.     {
```

```

70.     sorted[2] = sorted[1];
71.     sorted[1] = sorted[0];
72.     sorted[0] = points[2];
73. }
74. else
75. {
76.     if(points[2].x >= sorted[1].x)
77.     {
78.         sorted[2] = sorted[1];
79.         sorted[1] = points[2];
80.     }
81.     else
82.     {
83.         sorted[2] = points[2];
84.     }
85. }
86.
87. int d1 = sorted[0].x - sorted[1].x;
88. int d2 = sorted[2].x - sorted[1].x;
89.
90. if(abs(d1) > abs(d2)) return RIGHT;
91. else return LEFT;
92. }
93.
94. void DetectMarkers(cv::Mat &image)
95. {
96.     frameCount++;
97.
98.     error_angle = 0;
99.     error_dist = 0;
100.
101.     // cv::imshow("opencv_window", image);
102.     // cv::waitKey(3);
103.
104.     // convert to grey scale image
105.     cv::Mat greyImage;
106.     cv::cvtColor(image, greyImage, CV_BGR2GRAY);
107.
108.     // every object having grey scale value greater than threshold is made 0
109.     cv::Mat binary;
110.     int threshold = 3;
111.     cv::threshold(greyImage, binary, threshold, 255, THRESH_BINARY_INV);
112.
113.     // finds contour in the binary image
114.     vector<vector<Point>> contours;
115.     vector<Vec4i> hierarchy;
116.     cv::findContours(binary, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
117.
118.     cv::Mat img_contour = image;
119.     cv::putText(img_contour, StateToString(state), Point(5, 25), cv::FONT_HERSHEY_SIMPLEX, 1, cv::Scalar(0, 255,
120. 255), 2, 8, false);
121.
122.     if (contours.size() > 0)
123.     {
124.         // cv::Mat img_contour = Mat::zeros(greyImage.size(), CV_8UC3);
125.
126.         // approximate contours to polygons + get bounding rects and circles
127.         vector<vector<Point>> contours_poly(contours.size());
128.         vector<Point2f> center(contours.size());
129.         vector<float> radius(contours.size());
130.         vector<Point2f> circle_center;
131.         Point2f triangle_center;
132.
133.         // find enclosing polygon which fits around the contours and get their centers
134.         for (size_t i = 0; i < contours.size(); i++)
135.         {
136.             cv::approxPolyDP(Mat(contours[i]), contours_poly[i], 2, true);
137.             cv::minEnclosingCircle((Mat)contours_poly[i], center[i], radius[i]);
138.
139.             // accepting polygons with only certain radius
140.             if(radius[i] > 7)
141.             {
142.                 if (contours_poly[i].size() == 3)
143.                 {
144.                     // triangle marker

```

```

144.         cv::drawContours(img_contour, contours, i, cv::Scalar(0, 255, 0), 2, 8, hierarchy, 0, Point());
145.         triangle_center = center[i];
146.
147.         state = MOVE_TOWARDS_TURN;
148.
149.         // checking if robot reached the turn sign
150.         if(center[i].y > (image.rows * 0.6))
151.         {
152.             int dir = GetTriangleDirection(contours_poly[i]);
153.             if (dir == LEFT)
154.             {
155.                 state = TURN_LEFT;
156.                 cv::putText(img_contour, "LEFT", center[i], cv::FONT_HERSHEY_SIMPLEX, 1, cv::Scalar(0, 0, 255),
2, 8, false);
157.             }
158.             else if (dir == RIGHT)
159.             {
160.                 state = TURN_RIGHT;
161.                 cv::putText(img_contour, "RIGHT", center[i], cv::FONT_HERSHEY_SIMPLEX, 1, cv::Scalar(0, 0,
255), 2, 8, false);
162.             }
163.             frameCount = 0;
164.         }
165.         break;
166.     }
167.     else if (contours_poly[i].size() > 5)
168.     {
169.         // circle marker
170.         cv::drawContours(img_contour, contours, i, cv::Scalar(0, 255, 0), 2, 8, hierarchy, 0, Point());
171.         cv::putText(img_contour, "FOLLOW", center[i], cv::FONT_HERSHEY_SIMPLEX, 1, cv::Scalar(0, 0, 255),
2, 8, false);
172.         circle_center.push_back(center[i]);
173.
174.         if(state == TURN_LEFT || state == TURN_RIGHT)
175.         {
176.             if(frameCount > 30)
177.             {
178.                 state = FOLLOW;
179.                 frameCount = 0;
180.             }
181.         }
182.         else
183.         {
184.             state = FOLLOW;
185.         }
186.     }
187. }
188. else
189. {
190.     cv::drawContours(img_contour, contours, i, cv::Scalar(255, 255, 255), 2, 8, hierarchy, 0, Point());
191. }
192. }
193.
194. // correcting course while moving towards turn sign
195. if(state == MOVE_TOWARDS_TURN)
196. {
197.     float angle = CV_PI / 2;
198.     if (abs(triangle_center.x - (image.cols / 2)) > 0.0f)
199.     {
200.         float slope = (triangle_center.y - image.rows) / (triangle_center.x - (image.cols / 2));
201.         angle = atan(slope);
202.     }
203.     error_angle = (CV_PI / 2) - abs(angle);
204.
205.     if (angle < 0.0f)
206.     {
207.         error_angle *= -1;
208.     }
209. }
210.
211. if (circle_center.size() > 0 && state == FOLLOW)
212. {
213.     // line fitting
214.     Vec4f linefit;
215.     cv::fitLine(circle_center, linefit, CV_DIST_L2, 0, 0.01, 0.01);

```



```

216.
217.     Point2f pt1;
218.     pt1.x = linefit[2] - linefit[0] * 1000;
219.     pt1.y = linefit[3] - linefit[1] * 1000;
220.
221.     Point2f pt2;
222.     pt2.x = linefit[2] + linefit[0] * 1000;
223.     pt2.y = linefit[3] + linefit[1] * 1000;
224.
225.     // ROS_INFO("P1{x: %0.3f, y: %0.3f}, P2{x: %0.3f, y: %0.3f}", pt1.x, pt1.y, pt2.x, pt2.y);
226.
227.     // draw fitted line
228.     cv::line(img_contour, pt1, pt2, cv::Scalar(255, 0, 0), 2, cv::LINE_AA);
229.
230.     // draw desired path line(center of the camera frame)
231.     cv::line(img_contour, Point(image.cols / 2, 0), Point(image.cols / 2, image.rows), cv::Scalar(255, 0, 255), 2,
cv::LINE_AA);
232.
233.     // compute angle error;
234.     float angle = CV_PI / 2;
235.     if (abs(pt1.x - pt2.x) > 0.0f)
236.     {
237.         float slope = (pt2.y - pt1.y) / (pt2.x - pt1.x);
238.         angle = atan(slope);
239.     }
240.     error_angle = (CV_PI / 2) - abs(angle);
241.
242.     if (angle < 0.0f) error_angle *= -1;
243.
244.     // compute lateral distance error (distance between mid points of the desired and detected line)
245.     float lateral_dist = 0.0;
246.     if (abs(angle) < CV_PI / 2)
247.     {
248.         float slope = tan(angle);
249.         float y_intercept = pt1.y - (slope * pt1.x);
250.         float x_coord = ((image.rows / 2) - y_intercept) / slope;
251.         lateral_dist = x_coord - (image.cols / 2);
252.     }
253.     else
254.     {
255.         lateral_dist = pt1.x - (image.cols / 2);
256.     }
257.     error_dist = lateral_dist;
258.
259.     float error_dist_limit = 200.0f;
260.     if (abs(error_dist) > error_dist_limit)
261.     {
262.         if (error_dist > 0.0)
263.         {
264.             error_dist = error_dist_limit;
265.         }
266.         else
267.         {
268.             error_dist = -error_dist_limit;
269.         }
270.     }
271.     // ROS_INFO("err_dist: %0.3f, err_angle: %0.3f deg", error_dist, error_angle * 180 / CV_PI);
272. }
273.
274.     cv::imshow("opencv_window", img_contour);
275.     cv::waitKey(3);
276. }
277. }
278.
279. void ImageCallBack(const sensor_msgs::ImageConstPtr &msg)
280. {
281.     cv_bridge::CvImagePtr cv_ptr;
282.
283.     try
284.     {
285.         cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
286.         DetectMarkers(cv_ptr->image);
287.     }
288.     catch (cv_bridge::Exception &e)
289.     {

```

```

290.     ROS_ERROR("cv_bridge exception: %s", e.what());
291. }
292. }
293.
294. int main(int argc, char **argv)
295. {
296.     ros::init(argc, argv, "mark_follower");
297.
298.     ros::NodeHandle nh;
299.     ros::Subscriber image_subscriber = nh.subscribe("/agribot/front_camera/image_raw", 1, ImageCallBack);
300.
301.     ros::Publisher cmdvel_pub = nh.advertise<geometry_msgs::Twist>("/agribot_v2/mobile_base_controller/cmd_vel",
1);
302.     ros::Publisher error_dist_pub = nh.advertise<std_msgs::Float32>("/error_dist", 1);
303.     ros::Publisher error_angle_pub = nh.advertise<std_msgs::Float32>("/error_angle", 1);
304.
305.     ros::Rate loopRate(100);
306.
307.     cv::namedWindow("opencv_window", cv::WindowFlags::WINDOW_KEEPRATIO);
308.
309.     float rotSpeedLimit = 30.0;
310.     float speedLimit = 1;
311.
312.     while (ros::ok())
313.     {
314.         geometry_msgs::Twist cmd_vel;
315.
316.         // setting cmd_vel depending on the state of the robot
317.         if(state == FOLLOW)
318.         {
319.             cmd_vel.linear.x = speedLimit;
320.             cmd_vel.linear.y = 0.0;
321.             cmd_vel.angular.z = (error_angle * 30.0) + (-error_dist * 0.2);
322.
323.             // limiting angular velocity
324.             if (abs(cmd_vel.angular.z) >= rotSpeedLimit)
325.             {
326.                 cmd_vel.angular.z = cmd_vel.angular.z > 0 ? rotSpeedLimit : -rotSpeedLimit;
327.             }
328.
329.             std_msgs::Float32 err_dist;
330.             err_dist.data = error_dist;
331.             error_dist_pub.publish(err_dist);
332.
333.             std_msgs::Float32 err_angle;
334.             err_angle.data = error_angle * 180.0 / CV_PI;
335.             error_angle_pub.publish(err_angle);
336.         }
337.         else if(state == TURN_LEFT)
338.         {
339.             cmd_vel.linear.x = 0.0;
340.             cmd_vel.linear.y = 0.0;
341.             cmd_vel.angular.z = 3.0;
342.         }
343.         else if(state == TURN_RIGHT)
344.         {
345.             cmd_vel.linear.x = 0.0;
346.             cmd_vel.linear.y = 0.0;
347.             cmd_vel.angular.z = -3.0;
348.         }
349.         else if(state == MOVE_TOWARDS_TURN)
350.         {
351.             cmd_vel.linear.x = speedLimit;
352.             cmd_vel.linear.y = 0.0;
353.             cmd_vel.angular.z = (error_angle * 30.0);
354.
355.             // limiting angular velocity
356.             if (abs(cmd_vel.angular.z) >= rotSpeedLimit)
357.             {
358.                 cmd_vel.angular.z = cmd_vel.angular.z > 0 ? rotSpeedLimit : -rotSpeedLimit;
359.             }
360.         }
361.         else if(state == REST)
362.         {
363.             cmd_vel.linear.x = 0.0;

```

```

364.         cmd_vel.linear.y = 0.0;
365.         cmd_vel.angular.z = 0.0;
366.     }
367.
368.     cmdvel_pub.publish(cmd_vel);
369.
370.     // ROS_INFO("d: %0.3f, a: %0.3f, z: %0.3f", error_dist, error_angle * 180 / CV_PI, cmd_vel.angular.z);
371.
372.     ros::spinOnce();
373.     loopRate.sleep();
374. }
375.
376. return 0;
377. }

```

## weed\_detection.py :

```

1.  from ctypes import *
2.  import math
3.  import random
4.  import os
5.  import cv2
6.  import numpy as np
7.  import time
8.  import darknet
9.  import rospy
10.
11.  from sensor_msgs.msg import Image # Image is the message type
12.  from cv_bridge import CvBridge
13.
14.  # print ("bridge")
15.  bridge = CvBridge()
16.  network = None
17.  class_names = None
18.  class_colors = None
19.
20.  def convertBack(x, y, w, h):
21.      xmin = int(round(x - (w / 2)))
22.      xmax = int(round(x + (w / 2)))
23.      ymin = int(round(y - (h / 2)))
24.      ymax = int(round(y + (h / 2)))
25.      return xmin, ymin, xmax, ymax
26.
27.  def main(args=None):
28.      global network, class_names, class_colors
29.
30.      configPath = "./yolov4-custom.cfg" # Path to cfg
31.      weightPath = "./yolov4-custom_best.weights" # Path to weights
32.      metaPath = "./data/multiple_images.data" # Path to meta data
33.
34.      if not os.path.exists(configPath): # Checks whether file exists otherwise return ValueError
35.          raise ValueError("Invalid config path `" +
36.                           os.path.abspath(configPath)+"`")
37.      if not os.path.exists(weightPath):
38.          raise ValueError("Invalid weight path `" +
39.                           os.path.abspath(weightPath)+"`")
40.      if not os.path.exists(metaPath):
41.          raise ValueError("Invalid data file path `" +
42.                           os.path.abspath(metaPath)+"`")
43.
44.      network, class_names, class_colors = darknet.load_network(configPath, metaPath, weightPath, batch_size=1)
45.
46.      rospy.init_node("object_detection")
47.      sub = rospy.Subscriber("/agribot/front_camera/image_raw", Image, img_callback)
48.      rospy.spin()
49.
50.      cv2.destroyAllWindows()
51.
52.  def img_callback(img):
53.      current_frame = bridge.imgmsg_to_cv2(img, desired_encoding='bgr8')
54.      YOLO(current_frame)
55.
56.  def YOLO(img):

```

```

57.
58.     prev_time = time.time()
59.
60.     frame_height, frame_width, channels = img.shape
61.     darknet_image = darknet.make_image(frame_width, frame_height, 3) # Create image according darknet for
compatibility of network
62.
63.     frame_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)    # Convert frame into RGB from BGR and resize
accordingly
64.     frame_resized = cv2.resize(frame_rgb, (frame_width, frame_height), interpolation=cv2.INTER_LINEAR)
65.
66.     darknet.copy_image_from_bytes(darknet_image, frame_resized.tobytes())
67.
68.     detections = darknet.detect_image(network, class_names, darknet_image, thresh=0.3)
69.
70.     image = darknet.draw_boxes(detections, frame_resized, class_colors)
71.     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
72.
73.     print(time.time()-prev_time)
74.
75.     cv2.imshow('Demo', image)                                # Display Image window
76.     cv2.waitKey(3)
77.
78. if __name__ == "__main__":
79.     main()

```