BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 404 (January 2024)

Robotics and Automation Laboratory

Final Project Report

Group: 02

AgroBot: *An Intelligent Vision-Guided System for Sustainable Weed Detection & Elimination in Agriculture.*

Course Instructors:
Dr. Celia Shahnaz
Md. Jawwad Ul Islam

Signature of Instructor:

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. Type the student ID and name, and put your signature. *You will not receive credit for this project experiment unless this statement is signed in the presence of your lab instructor.*

"In signing this statement, We hereby certify that the work on this project is our own and that we have not copied the work of any other students (past or present), and cited all relevant sources while completing this project. We understand that if we fail to honor this agreement, We will each receive a score of ZERO for this project and be subject to failure of this course."

Signature: **Full Name: Progna Dipto Saha** **Student ID: 1906027**	**Signature:** **Full Name: Tanvir Ahmed Khan** **Student ID: 1906048**
Signature: **Full Name: Asif Akhtab Ronggon** **Student ID: 1906078**	**Signature:** **Full Name:Md. Hadayetuzzaman** **Student ID: 1906102**
Signature: **Full Name: Shadman Sobhan** **Student ID: 1906109**	**Signature:** **Full Name: Kazi Abrar Mahmud** **Student ID: 1906112**

Table of Contents

1 Abstract

As an agricultural country, agriculture plays a crucial role in the economy of Bangladesh, providing livelihoods for millions of people. However, one of the challenges faced by farmers is the **growth of weeds**, which reduce crop yields, increase manual labor.However,a rover bot designed to detect and treat weeds offers an innovative solution to this problem.Utilizing advanced technologies such as computer vision, machine learning, and precision spraying mechanisms, the rover bot can autonomously navigate fields, accurately identify weeds, and apply targeted treatments. This reduces the need for harmful chemical herbicides, minimizing environmental impact and preserving soil health. Furthermore, the automation of weed control can significantly reduce labor costs, particularly in rural areas where workforce shortages during peak seasons can be a challenge. For a country like Bangladesh, where agriculture is predominantly small-scale and resource-limited, the implementation of a weed-detecting and treating rover bot could enhance productivity, improve sustainability, and contribute to food security by allowing farmers to focus more on crop care and management while reducing their dependence on manual labor and harmful chemicals. **Our idea focuses on an affordable rover bot with the aforementioned features to support economically disadvantaged farmers.**

2 Introduction

For This project we developed an AI-based pesticide sprayer .The motivation behind this project was the fact that Agriculture is the backbone of Bangladesh's economy, providing livelihoods for millions and ensuring food security. However, farmers face significant challenges, including weed growth, which reduces crop yields and increases labor demands. To address this, an innovative rover bot has been conceptualized to detect and treat weeds autonomously. By integrating technologies like computer vision, machine learning, and precision spraying, the bot minimizes chemical herbicide use, preserves soil health, and reduces environmental impact. This affordable solution not only enhances productivity but also alleviates labor shortages, particularly in resource-limited, small-scale farming systems. It represents a sustainable step forward for Bangladesh's agricultural future.

3 Design

3.1 Problem Formulation

3.1.1 Identification of Scope

There is a big problem of weed management in Bangladesh and it increases the cost of production and decreases the production quantity of crops. Current methods used in weed control are labor intensive, adverse to the environment, or unaffordable to the small-scale farmers. There is a major gap in the market for a low-cost, autonomous and effective solution.

The creation of a weed identifying and removing rover bot solves the major problems of high reliance on manpower and the use of chemical herbicides. Through the integration of

technology in the weeding process, rover bot plays a crucial role in cutting down costs and decreasing the adverse effects on the environment. It presents a new way to look at agriculture and the use of technology in making it sustainable.
Weeds are the greatest enemies to crops as they struggle for water, nutrients and light, thus lowering crop yields and income of farmers. Other factors that also contribute to the suffering of farmers include the high cost of labor and the use of herbicides which are costly. This can have a revolutionary impact on the agricultural output, sustainability and the standard of living of farmers.
The implementation of the rover bot has the potential to transform the agricultural sector in Bangladesh. By reducing manual labor costs and minimizing the use of chemical herbicides, the rover can contribute to a more sustainable and eco-friendly farming system. The increased productivity and reduced production costs can enhance farmers' income and overall well-being.

3.1.2 Literature Review

Weeding robots utilize either chemical or non-chemical methods to remove weeds. Chemical approaches use targeted spraying systems to minimize herbicide usage, while non-chemical approaches rely on mechanical or laser-based tools to eliminate weeds.

Weeding robots use various tools to remove weeds depending on the weeding method. Spraying robots utilize targeted nozzles for precise herbicide application, while mechanical robots use tools like brushes, blades, and cutting discs. Laser-based robots use high-power lasers to burn and eliminate weeds.[2]

Weeding robots navigate agricultural fields using vision-based row-following, RTK-GPS guidance, and coverage planning algorithms. These strategies ensure accurate movement, precise localization, and efficient coverage of the field.[2]

Vision-based weed detection is the dominant approach for weed detection. The YOLO (You Only Look Once) series of models has become a prominent tool in agricultural applications, particularly for weed detection. YOLOv5, the latest iteration, offers significant improvements in real-time object detection capabilities, making it suitable for precision agriculture [1,8].

YOLOv5 has three main components:

1. **Backbone**:
    - The backbone of YOLOv5 is based on CSPDarknet, which is a variant of the traditional Darknet architecture. It is responsible for feature extraction from the input images. The CSP (Cross Stage Partial) design allows for better gradient flow and improved learning during training by splitting feature maps into two parts and processing them in parallel before merging them back together. This structure enhances the model's ability to capture various features at different levels of granularity.

2. **Neck**:
    - The neck of YOLOv5 incorporates a modified Path Aggregation Network (PANet) along with a Spatial Pyramid Pooling (SPP) layer, specifically a variant known as SPPF. PANet improves feature aggregation across different scales, facilitating better localization and context understanding. The SPPF layer increases the receptive field without compromising speed, allowing the model to focus on relevant features more effectively.
3. **Head**:
    - The head of YOLOv5 processes the aggregated features to predict bounding boxes, class labels, and objectness scores. It consists of several convolutional layers that output the coordinates (x, y, width, height) for each detected object along with its class probabilities.

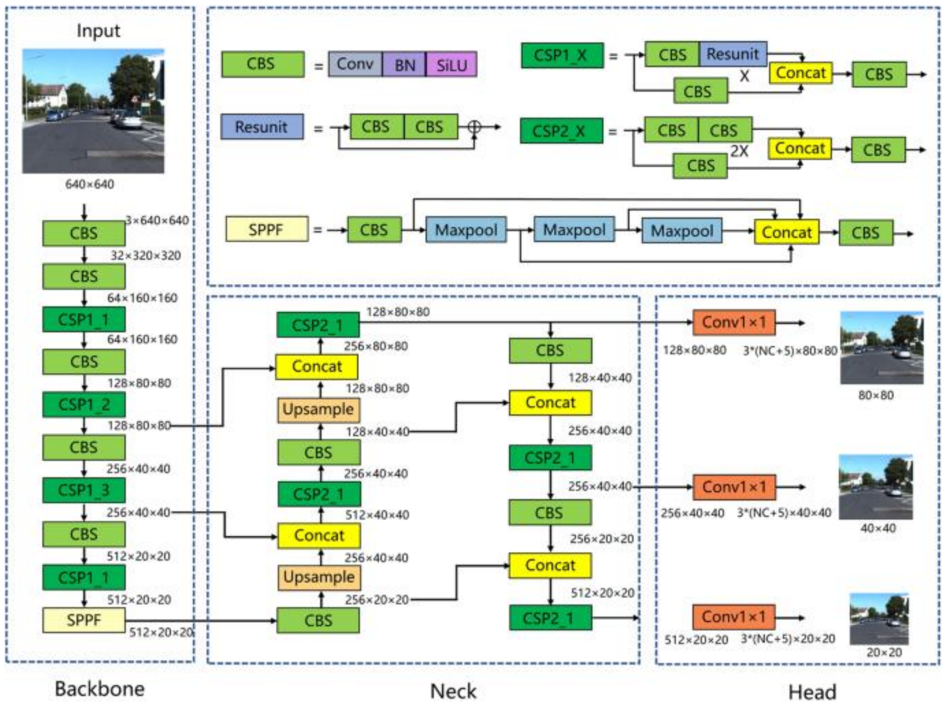


Fig: Yolo V-5 architecture(Source: https://www.nature.com/articles/s41598-023-36868-w)

This design allows for efficient processing and high accuracy in detecting objects within images. Its lightweight nature makes it particularly suitable for deployment on edge computing platforms, enabling real-time applications in field environments

Despite technical challenges, several commercial weeding robots are actively used in agriculture, including Ecorobotix AVO[3] (solar-powered, targeted spraying), EarthSense[4] TerraMax (vision-based navigation and herbicide application), and Naïo Technologies Ted [5] (autonomous electric weeding in vineyards). Other notable systems include Carbon Robotics LaserWeeder[6] (laser-based weeding) and FarmDroid FD20 [7] (solar-powered seeding and weeding).

3.1.3 Formulation of Problem

The primary objective of our project is to design a cost-effective, autonomous rover bot that can detect and spray weeds in the agricultural field. We aim to address high labor costs and the environmental impact of excessive herbicide use. The boundaries of the design include affordability, simplicity of operation, and adaptability to local environmental conditions.
The rover bot will have three core functionalities: (1) weed detection using image processing or sensor-based systems, (2) autonomous navigation to move through crop fields, and (3) targeted spraying to minimize herbicide use. Our design also requires robustness, low maintenance, and compatibility with local farming practices.
Our design is constrained by cost limitations, as it must be affordable for small-scale farmers. We also consider scalability for various environmental factors such as high humidity, uneven terrain, and varying crop types.

3.1.4 Analysis

This project combines advanced technologies such as computer vision, machine learning, autonomous navigation, and robotics to create an Vision based Weed-detection and elimination system. It addresses the critical challenge of reducing pesticide use by deploying an intelligent targeted spraying system. The project has the potential to significantly reduce harmful and toxic pesticide use and usher in a new era of Ai-based agricultural solutions.

3.2 Design Method

Our over all project consists of 3 main component :
1. Complete ROS (noetic) integration.
2. Weed detection using vision model
3. Autonomous navigation using SOTA odometry techniques
4. Dedicated targeted spray system.

ROS system architecture:

Our complete project is built using ROS 1 noetic.We deployed our system on a Single Board Computer(SBC) namely raspberry pi5. Raspberry uses Arm-architecture hence we had to create a custom Docker image for encapsulating all the project files from our (amd/x86) architecture based development PC.The overall software architectural view is given below:

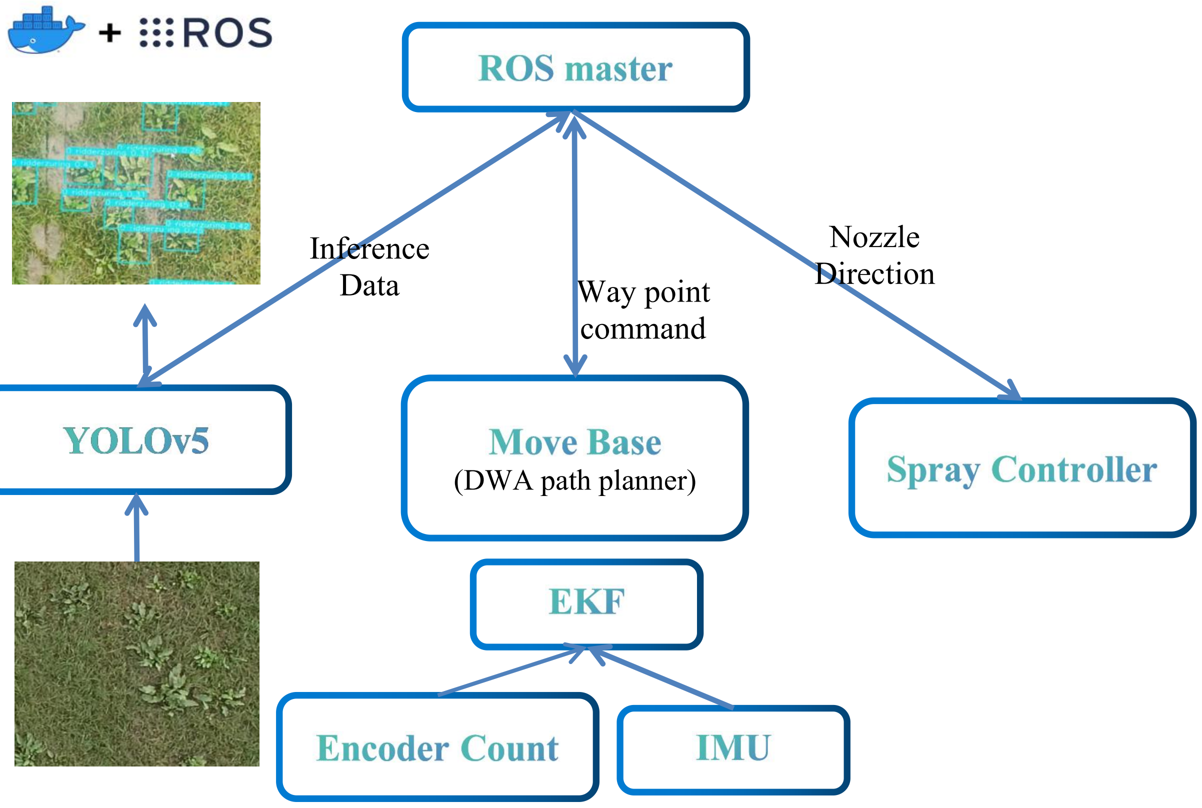


Fig : Over all system architecture.

Ai-Vision:

Yolo V5 Architecture:

YOLOv5, developed by Ultralytics, is a state-of-the-art object detection model that follows the "You Only Look Once" (YOLO) architecture, known for its speed and accuracy. It utilizes a single convolutional neural network (CNN) to predict bounding boxes and class probabilities directly from full images in one forward pass. YOLOv5 features a modular design with four key components: backbone, neck, head, and detection layers. The backbone extracts essential features from images, typically using CSPDarknet, while the neck enhances feature fusion through PANet or FPN. The detection head predicts object locations and classes using anchor-based or anchor-free approaches. Its flexibility, ease of deployment, and support for multiple resolutions make YOLOv5 widely popular for real-time applications.

Dataset Description:

We trained our model with weed detection dataset available on kaggle {https://www.kaggle.com/datasets/jaidalmotra/weed-detection}. This dataset contains 1906 images, among which1661 images belong to train set and the remaining 245 images belong to test set. The labels are available in COCO (Common Objects in Context) format, which is widely used for object detection tasks. In this format, the labels are provided in a JSON file. Images of the dataset with corresponding label is given :

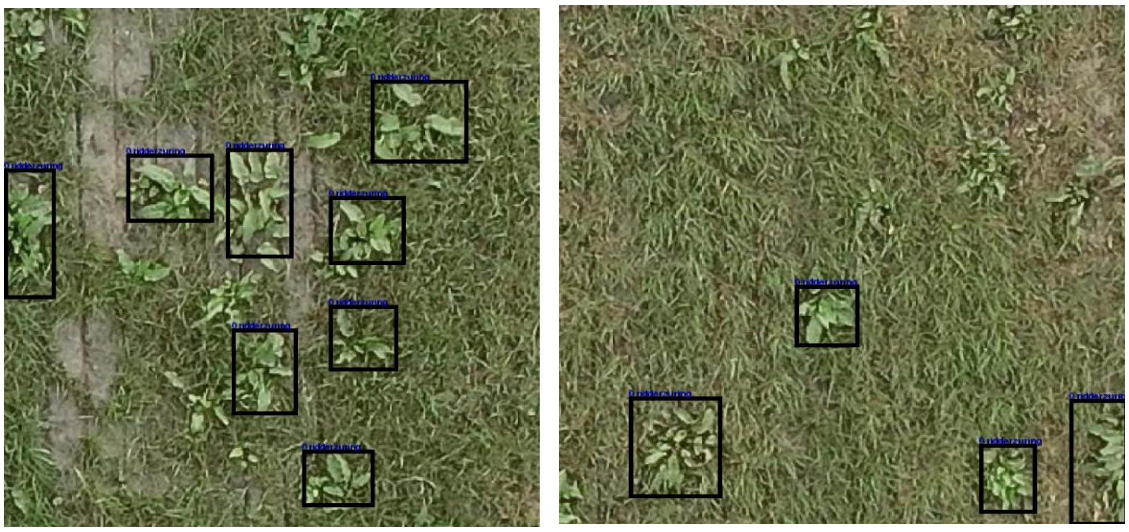


Fig:Trained YOLOv5 inference snapshot.

Autonomus Navigation :

Now in order to give our rover body a sense of its relative position ,we used MPU6050 IMU(Inertial Measurements Unit) based odometric system.As our whole system is developed using ROS, we needed to create ROS compatible driver file. Our esp32 driver file uses rosserial node to communicate with the master. The main task of the esp32 based control circuit is to publish wheel encoder and imu data in separate topics.

Odometry:

For filtering out the noisy wheel encoder data we encororated a Extended-Kalman-filter to fuse this incoming wheel encoder data with that of Inertial measurement unit (IMU).The configuration parameter for EKF_localization is given below.

Configuration parameters for any senor in the localization package must follow the standard structure below:

$$sensor\_config = \begin{pmatrix} X & Y & Z \\ roll & pitch & yaw \\ \dot{X} & \dot{Y} & \dot{Z} \\ \dot{roll} & \dot{pitch} & \dot{yaw} \\ \ddot{X} & \ddot{Y} & \ddot{Z} \end{pmatrix};$$

Configuration parameters for odometry form wheel encoder only

$$odm\_config = \begin{pmatrix} false & false & false \\ false & false & false \\ true & true & true \\ false & false & false \\ false & false & false \end{pmatrix}$$

$odom\_que\_size: 10$
$\text{odom\_nodelay}: \text{true}$
$\text{odom\_differential}: \text{false}$
$\text{odom\_relative}: \text{false}$

Configuration parameters for orientation form IMU only

$$imu\_config = \begin{pmatrix} false & false & false \\ true & true & true \\ false & false & false \\ true & true & false \\ true & true & true \end{pmatrix}$$

$imu\_nodelay: false$
$imu\_differential: false$
$imu\_relative: false$
$imu\_queue\_size: 10$
$imu\_remove\_gravitational\_acceleration: true$

The process and noise covariance matrix for our system is given below:

Process Noise Covariance Matrix

$$\mathbf{Q} = \begin{bmatrix}
0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.06 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.03 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.06 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0.025 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.025 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.04 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.02 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.015
\end{bmatrix}$$

Initial Estimate Covariance Matrix

$$\mathbf{P}_0 = \begin{bmatrix}
1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\times10^{-9}
\end{bmatrix}$$

Fig : Covariance matrix for our Extended Kalman Filter(EKF) configuration

Wheel odometry offers velocity and position data but is prone to drift, while IMUs provide orientation and angular velocity, which counteract odometry errors through complementary measurements. The robot_localization package allows fine-tuning configurations for state estimation, enabling reliable fusion by setting covariance parameters, selecting relevant data streams, and handling time delays, making it ideal for mobile robots.

Below we demonstrate the performance improvement gained from fusing the wheel odometry and IMU data.

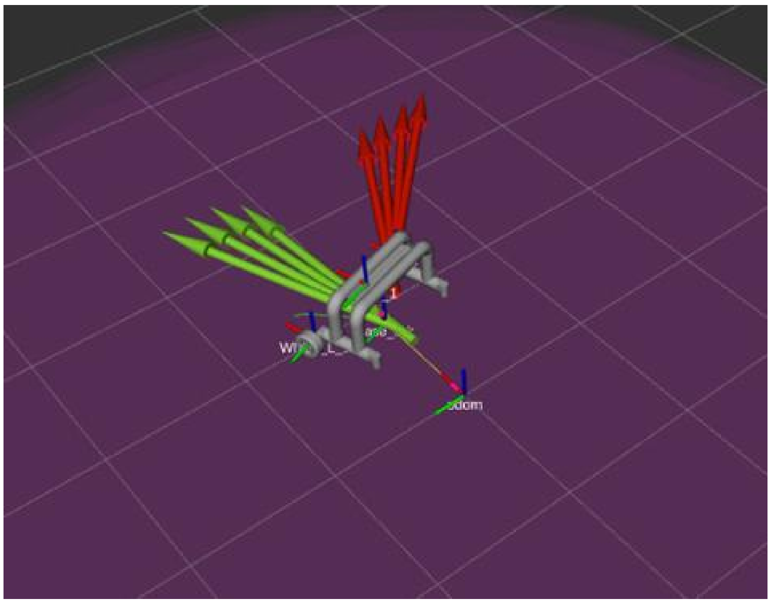


Fig : Comparison between raw odometry(Red) and filtered odometry(green)

From the above plot of the sensor values in '***rviz***' we can clearly see that the unfiltered value (Red arrowed) seems to deviate a lot from the actual physical position of the rover body, on the other hand fused odometry from the EKF algorithm seems to much accurately represent the actual position of the rover body.

Path planning:

For path planning and execution we implemented DWA(dynamic window approach) as the local path planner .Given a global plan to follow and a cost-map, the local planner produces velocity commands to send to a mobile base.

The configuration for our Local planner is given below:

```
DWAPlannerROS:

# Robot Configuration Parameters
  max_vel_x: 4.52
  min_vel_x: -4.52
  max_vel_y: 0.0
  min_vel_y: 0.0
# The velocity when robot is moving in a straight line
  max_vel_trans:  1.22
  min_vel_trans:  1.11
  max_vel_theta: 1.9
  min_vel_theta: -1.9
  acc_lim_x: 2.5
  acc_lim_y: 0.0
  acc_lim_theta: 3.2
# Goal Tolerance Parameters
  xy_goal_tolerance: 0.18   #<---Must be hailf of base_local_planner tolerance
  yaw_goal_tolerance: 0.17
  latch_xy_goal_tolerance: false
# Forward Simulation Parameters
  sim_time: 1.5
  vx_samples: 20
  vy_samples: 0
  vth_samples: 40
  controller_frequency: 10.0
```

Waypoint generation:

This project implements an autonomous navigation system for a robot to cover a defined field area using a zigzag pattern. The waypoints are generated dynamically based on the field’s dimensions and are formatted as ROS (Robot Operating System) goal messages. These goal messages are then sent to the ROS move_base node to execute the navigation. The system was developed and tested using Python for waypoint generation and ROS for motion control.

The results of waypoint generation is given below:

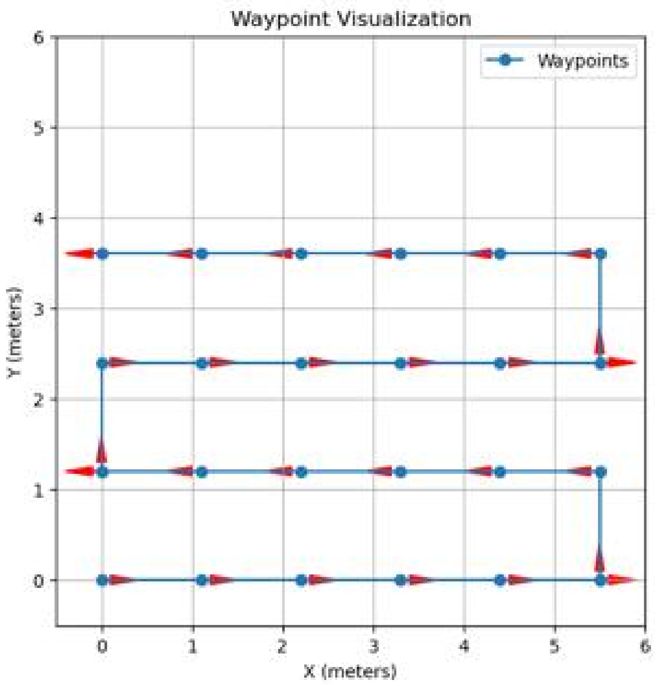
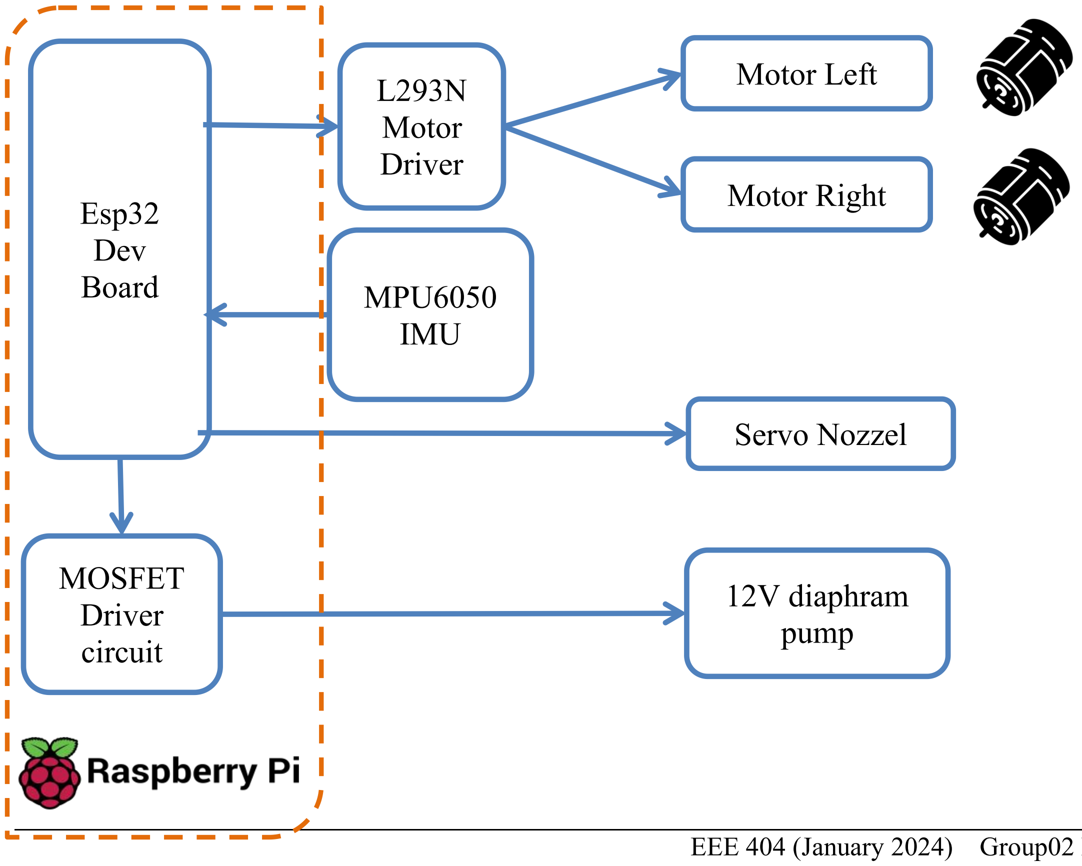


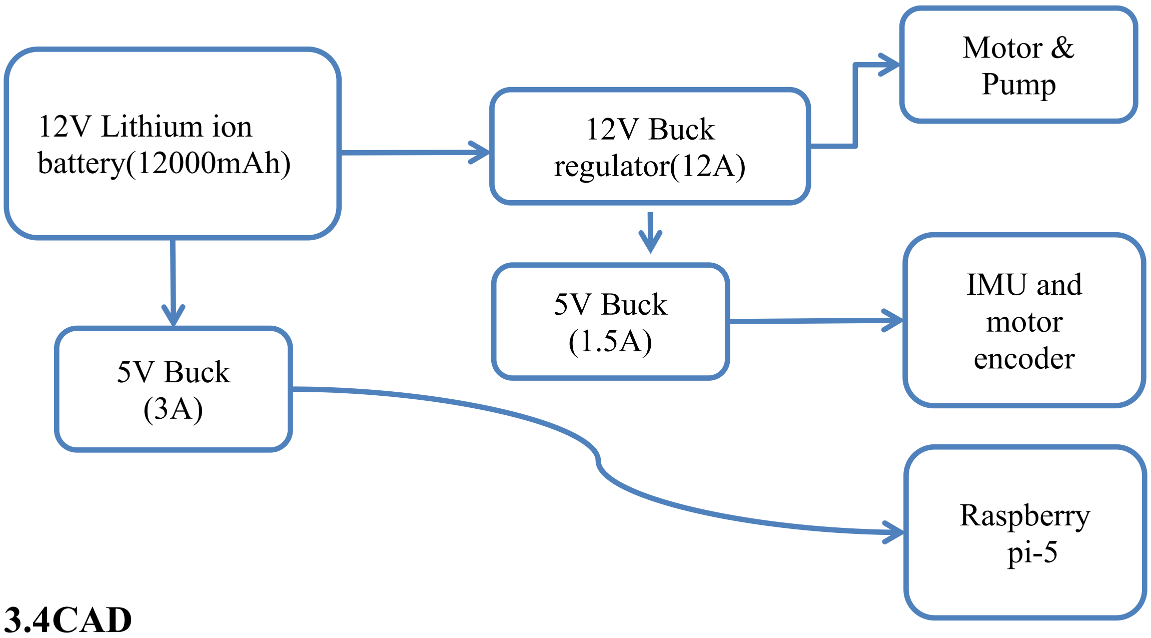
Fig: Generated waypoints from our custom waypoint_nv.py ROS node.

3.3 Circuit Diagram

Here ,Block diagram of various circuit component is given below:
Main Controller Unit :



Power Management:



3.4CAD

Before the physical construction of our robot ,we desigend a CAD for it, so that we can simulate it before practical deployment, this CAD is also necessary for the URDF generation.

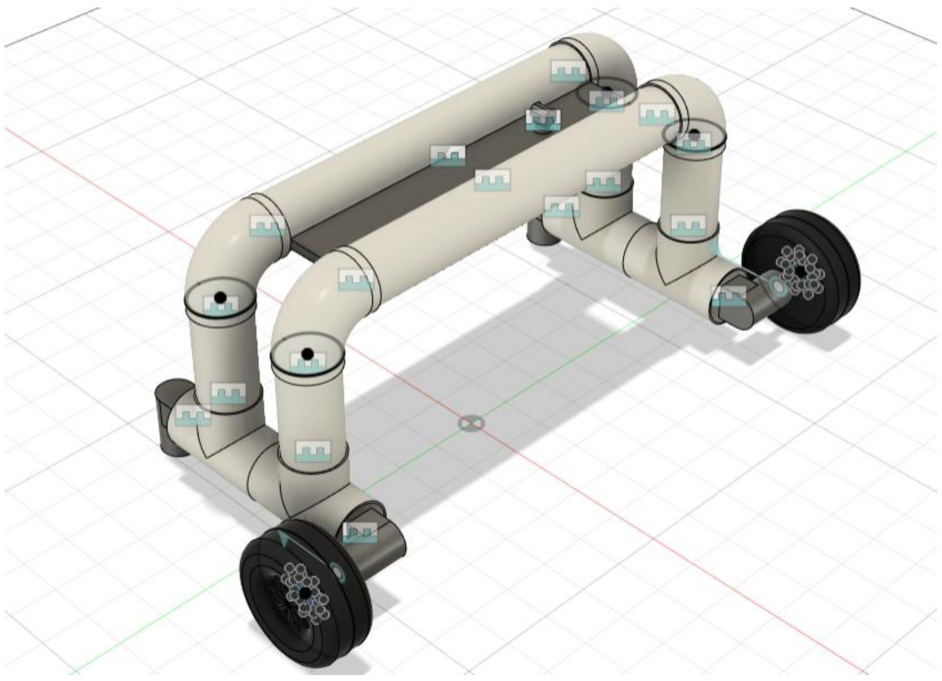


Fig :Technical CAD file containing all the Measurements and joints.

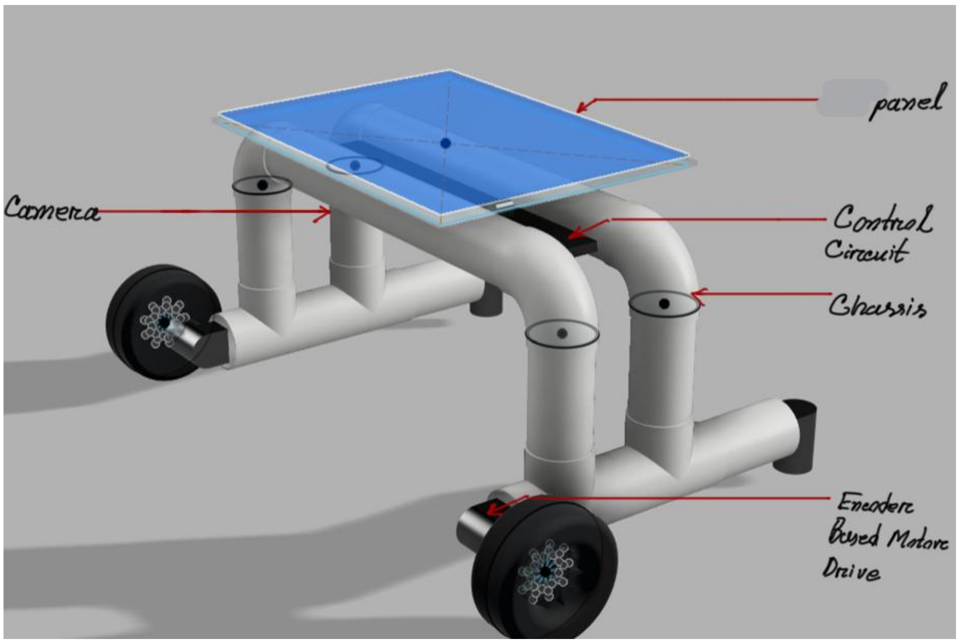


Fig : Rendered Grapahic for our AgroBot

This CAD file was used to generate URDF(Universal Robot Description Format).This .urdf file is required by the ROS parameter server to know the relative location of all its components with respect to its base-link.

A snippet of the generated URDF is given below:

```
<?xml version="1.0" ?>
<robot name="basic_structure" xmlns:xacro="http://www.ros.org/wiki/xacro">

<xacro:include filename="$(find basic_structure_description)/urdf/materials.xacro" />
<xacro:include filename="$(find
basic_structure_description)/urdf/basic_structure.trans" />
<xacro:include filename="$(find
basic_structure_description)/urdf/basic_structure.gazebo" />
<link name="base_link">
  <inertial>
    <origin xyz="1.1480419745044299e-14 0.0011587210328846216 0.30511587210328844"
rpy="0 0 0"/>
    <mass value="1.9060862219587476"/>
    <inertia ixx="0.025357" iyy="0.000584" izz="0.025904" ixy="-0.0" iyz="-2.2e-05"
ixz="0.0"/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://basic_structure_description/meshes/base_link.stl"
scale="0.001 0.001 0.001"/>
    </geometry>
    <material name="silver"/>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://basic_structure_description/meshes/base_link.stl"
scale="0.001 0.001 0.001"/>
    </geometry>
  </collision>
</link>
<link name="L11_1">
  <inertial>
    <origin xyz="0.029999999999999985 3.677692539541783e-17 5.551115123125783e-17"
rpy="0 0 0"/>
    <mass value="2.1698176210625393"/>
    <inertia ixx="0.029798" iyy="0.001734" izz="0.029798" ixy="-0.0" iyz="0.0"
ixz="0.0"/>
  </inertial>
  <visual>
    <origin xyz="-0.03 -0.0 -0.31" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://basic_structure_description/meshes/L11_1.stl"
scale="0.001 0.001 0.001"/>
    </geometry>
    <material name="silver"/>
  </visual>
  <collision>
    <origin xyz="-0.03 -0.0 -0.31" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://basic_structure_description/meshes/L11_1.stl"
scale="0.001 0.001 0.001"/>
    </geometry>
  </collision>
</link>
                       ........



<joint name="Rigid 33" type="fixed">
  <origin xyz="-0.06 0.0 -0.06" rpy="0 0 0"/>
  <parent link="L44_1"/>
  <child link="caster_L_1"/>
</joint>
<joint name="Rigid 34" type="fixed">
  <origin xyz="-0.06 0.0 -0.06" rpy="0 0 0"/>
  <parent link="L41_1"/>
  <child link="caster_R_1"/>
</joint>
<joint name="Rigid 35" type="fixed">
  <origin xyz="0.0 0.1 0.32" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="imu_v1_1"/>
</joint>
</robot>
```

Fig :Snippet of URDF for our system

3.5Full Source Code of Firmware

Driver Code for **Main Control Unit**(esp32 dev board)

```
///####### Use the command below to validate 
Arduino node
#################################################
################
///rosrun rosserial_python serial_node.py 
_port:=/dev/ttyUSB0 _baud:=57600
///##############################################
#################################################
###########################

///## Board name = DOIT ESP32 DEVKIT V1
#################################################
#################################
#include <ESP32Servo.h>
#include <ESP32Encoder.h>
#include "pins_me.h"
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps612.h"
#if I2CDEV_IMPLEMENTATION == 
I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
#include <PID_v1.h>
#include <ros.h>
#include <std_msgs/String.h>
#include <sensor_msgs/Imu.h>
#include <geometry_msgs/Vector3Stamped.h>
#include <geometry_msgs/Twist.h>
#include <ros/time.h>



//initializing all the variables
// Shared Variables(Thread for Running the 
motor loop)
int sharedDirL = 0;
int sharedPwmValL = 0;
int sharedDirR = 0;
int sharedPwmValR = 0;
SemaphoreHandle_t xMutex; // Mutex to protect 
shared variables

TaskHandle_t motorTaskLHandle, 
motorTaskRHandle; // Task handles

     //Looptime in millisecond

const byte noCommLoopMax = 
10;                  //number of main loops the 
robot will execute without communication 
before stopping
unsigned int noCommLoops = 
0;                   //main loop without 
communication counter
unsigned int LOOPTIME = 100;  //100ms or 10 hz
```

```
double speed_cmd_left2 = 0;

const int PIN_ENCOD_A_MOTOR_LEFT = 
enc_left_y;             //A channel for 
encoder of left motor
const int PIN_ENCOD_B_MOTOR_LEFT = 
enc_left_g;             //B channel for 
encoder of left motor

const int PIN_ENCOD_A_MOTOR_RIGHT = 
enc_right_y;            //A channel for 
encoder of right motor
const int PIN_ENCOD_B_MOTOR_RIGHT = 
enc_right_g;            //B channel for 
encoder of right motor



//--- Robot-specific constants ---
##############################################
###########
unsigned long lastMilli = 0;
const double radius = 
0.0635;                 //Wheel radius, in m
const double wheelbase = 
0.64;             //Wheelbase, in m
const double encoder_cpr = 
4800;               //Encoder ticks or counts 
per rotation
const double speed_to_pwm_ratio = 
0.0041;     //  0.00235;   Ratio to convert 
speed (in m/s) to PWM value. It was obtained 
by plotting the wheel speed in relation to the 
PWM motor command (the value is the slope of 
the linear function).
const double min_speed_cmd = 
0.48;          //0.0882  (min_speed_cmd/speed_t
o_pwm_ratio) is the minimum command value 
needed for the motor to start moving.


double speed_req = 
0;                         //Desired linear 
speed for the robot, in m/s
double angular_speed_req = 
0;               //Desired angular speed for 
the robot, in rad/s

double speed_req_left = 
0;                    //Desired speed for left 
wheel in m/s
double speed_act_left = 
0;                    //Actual speed for left 
wheel in m/s
double speed_cmd_left = 
0;                    //Command speed for left 
wheel in m/s
```

```
double speed_req_right =
0;                  //Desired speed for right
wheel in m/s
double speed_act_right =
0;                  //Actual speed for right
wheel in m/s
double speed_cmd_right =
0;                  //Command speed for right
wheel in m/s

const double max_speed =
3.7;                // 0.4  Max speed in m/s

int PWM_leftMotor =
0;                    //PWM command for left
motor
int PWM_rightMotor =
0;                   //PWM command for right
motor


// PID Parameters
const double PID_left_param[] = { 0.9, 0,
0.1 }; //Respectively Kp, Ki and Kd for left
motor PID
const double PID_right_param[] = { 0.9, 0,
0.1 }; //Respectively Kp, Ki and Kd for right
motor PID

volatile float pos_left = 0;       //Left motor
encoder position
volatile float pos_right = 0;      //Right
motor encoder position
volatile float pos_left_prev = 0;       //Left
motor encoder position
volatile float pos_right_prev = 0;      //Right
motor encoder position
// MPU control/status vars
bool dmpReady = false;  // set true if DMP init
was successful
uint8_t mpuIntStatus;   // holds actual
interrupt status byte from MPU
uint8_t devStatus;      // return status after
each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet
size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes
currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y,
z]         quaternion container
VectorInt16 aa;         // [x, y,
z]            accel sensor measurements
VectorInt16 gy;         // [x, y,
z]            gyro sensor measurements
VectorInt16 aaReal;     // [x, y,
z]            gravity-free accel sensor
measurements
VectorInt16 aaWorld;    // [x, y,
z]            world-frame accel sensor
measurements
VectorFloat gravity;    // [x, y,
z]            gravity vector
float euler[3];         // [psi, theta,
phi]    Euler angle container
float ypr[3];           // [yaw, pitch,
roll]   yaw/pitch/roll container and gravity
vector
float
detect_x=70.0,detect_y=0.0,prediction_val=0.0;
float infer_pos_x = 90.0;



PID PID_leftMotor(&speed_act_left,
&speed_cmd_left, &speed_req_left,
PID_left_param[0], PID_left_param[1],
PID_left_param[2], DIRECT);          //Setting
up the PID for left motor
PID PID_rightMotor(&speed_act_right,
&speed_cmd_right, &speed_req_right,
PID_right_param[0], PID_right_param[1],
PID_right_param[2], DIRECT);   //Setting up the
PID for right motor
ESP32Encoder encoder;
ESP32Encoder encoder2;
MPU6050 mpu;
Servo myservo;
// Adafruit_MotorShield AFMS =
Adafruit_MotorShield();  // Create the motor
shield object with the default I2C address
// Adafruit_DCMotor *leftMotor =
AFMS.getMotor(1);      //Create left motor
object
// Adafruit_DCMotor *rightMotor =
AFMS.getMotor(2);     //Create right motor
object

ros::NodeHandle nh;

//function that will be called when receiving
command from host
void handle_cmd (const geometry_msgs::Twist&
cmd_vel) {
  noCommLoops =
0;
   //Reset the counter for number of main loops
without communication

  speed_req =
cmd_vel.linear.x;
      //Extract the commanded linear speed from
the message


  angular_speed_req =
cmd_vel.angular.z;                           //
Extract the commanded angular speed from the
message
```

```
  speed_req_right = speed_req -
angular_speed_req*(wheelbase/2);     //Calculat
e the required speed for the left motor to
comply with commanded linear and angular
speeds
  speed_req_left = speed_req +
angular_speed_req*(wheelbase/2);    //Calculate
the required speed for the right motor to
comply with commanded linear and angular
speeds
}
void handle_infer (const
geometry_msgs::Vector3Stamped& infer) {

  detect_x =
infer.vector.x ;

  detect_y = infer.vector.y ;
  prediction_val =
infer.vector.z ;
}
//
===============================================
==================
// ===                INTERRUPT DETECTION
ROUTINE                ===
//
===============================================
==================

volatile bool mpuInterrupt = false;     //
indicates whether MPU interrupt pin has gone
high
void dmpDataReady() {
  mpuInterrupt = true;
}



ros::Subscriber<geometry_msgs::Twist>
cmd_vel("cmd_vel", handle_cmd);   //create a
subscriber to ROS topic for velocity commands
(will execute "handle_cmd" function when
receiving data)
ros::Subscriber<geometry_msgs::Vector3Stamped>
infer("infer", handle_infer);
geometry_msgs::Vector3Stamped
speed_msg;                              //crea
te a "speed_msg" ROS message
ros::Publisher speed_pub("speed",
&speed_msg);                            //create a
publisher to ROS topic "speed" using the
"speed_msg" type
sensor_msgs::Imu imu_msg;
ros::Publisher imu_pub("/imu/data_raw",
&imu_msg);
long sequence = 0;


//_____________________________________________
______________________________
```

```
void setup() {


    // join I2C bus (I2Cdev library doesn't do
this automatically)
  #if I2CDEV_IMPLEMENTATION ==
I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000); // 400kHz I2C clock.
Comment this line if having compilation
difficulties
  #elif I2CDEV_IMPLEMENTATION ==
I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
  #endif

  nh.initNode();                            //in
it ROS node
  nh.getHardware()-
>setBaud(57600);         //set baud for ROS
serial communication
  nh.subscribe(cmd_vel);                    //s
uscribe to ROS topic for velocity commands
  nh.subscribe(infer);                    //sus
cribe to ROS topic for infrance
  nh.advertise(speed_pub);                  //p
repare to publish speed in ROS topic
  nh.advertise(imu_pub);
  //AFMS.begin();
  // Thread
activation:###############################
###############################################
#
  // Create the FreeRTOS task
  // Create Mutex
  xMutex = xSemaphoreCreateMutex();

  // Create Task
  xTaskCreate(
    motorTaskL,          // Task function
    "Motor Control Left",// Task name
    1000,                // Stack size
    NULL,                // No initial
parameters
    1,                   // Task priority
    &motorTaskLHandle    // Task handle
  );
    xTaskCreate(
    motorTaskR,          // Task function
    "Motor Control Right",// Task name
    1000,                // Stack size
    NULL,                // No initial
parameters
    1,                   // Task priority
    &motorTaskRHandle    // Task handle
  );

  // Start with task suspended
  vTaskSuspend(motorTaskLHandle);
  vTaskSuspend(motorTaskRHandle);
  //##############################################
################################################
############



  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(EN_left, OUTPUT);

  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);
  pinMode(EN_right, OUTPUT);



  myservo.setPeriodHertz(50);    // standard 50
hz servo
  myservo.attach(servo_pin, 200, 2000); //
attaches the servo on pin 18 to the servo
object
  pinMode(pump_pin,OUTPUT);
  //setting motor speeds to zero
  // leftMotor->setSpeed(0);
  // leftMotor->run(BRAKE);
  // rightMotor->setSpeed(0);
  // rightMotor->run(BRAKE);

  //setting PID parameters
  PID_leftMotor.SetSampleTime(95);
  PID_rightMotor.SetSampleTime(95);
  PID_leftMotor.SetOutputLimits(-max_speed,
max_speed);
  PID_rightMotor.SetOutputLimits(-max_speed,
max_speed);
  PID_leftMotor.SetMode(AUTOMATIC);
  PID_rightMotor.SetMode(AUTOMATIC);

  ESP32Encoder::useInternalWeakPullResistors =
puType::up;
  encoder.attachFullQuad(PIN_ENCOD_A_MOTOR_LEFT,
PIN_ENCOD_B_MOTOR_LEFT);
  encoder2.attachFullQuad(PIN_ENCOD_A_MOTOR_RIG
HT, PIN_ENCOD_B_MOTOR_RIGHT);
  encoder.clearCount();
  encoder2.clearCount();


  //////////############### Initialize DMP
############################
  // initialize device
  Serial.println(F("Initializing I2C
devices..."));
  mpu.initialize();
  pinMode(GYRO_INTERRUPT_PIN, INPUT);

  // verify connection
  Serial.println(F("Testing device
connections..."));
  Serial.println(mpu.testConnection() ?
F("MPU6050 connection successful") :
F("MPU6050 connection failed"));



  // load and configure the DMP
  Serial.println(F("Initializing DMP..."));
  devStatus = mpu.dmpInitialize();

  // supply your own gyro offsets here, scaled
for min sensitivity
  mpu.setXGyroOffset(51);
  mpu.setYGyroOffset(8);
  mpu.setZGyroOffset(21);
  mpu.setXAccelOffset(1150);
  mpu.setYAccelOffset(-50);
  mpu.setZAccelOffset(1060);
  // make sure it worked (returns 0 if so)
  if (devStatus == 0) {
    // Calibration Time: generate offsets and
calibrate our MPU6050
    mpu.CalibrateAccel(6);
    mpu.CalibrateGyro(6);
    Serial.println();
    mpu.PrintActiveOffsets();
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.print(F("Enabling interrupt
detection (Arduino external interrupt "));
    Serial.print(digitalPinToInterrupt(GYRO_INT
ERRUPT_PIN));
    Serial.println(F(")..."));
    attachInterrupt(digitalPinToInterrupt(GYRO_
INTERRUPT_PIN), dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop()
function knows it's okay to use it
    Serial.println(F("DMP ready! Waiting for
first interrupt..."));
    dmpReady = true;

    // get expected DMP packet size for later
comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the
code will be 1)
    Serial.print(F("DMP Initialization failed
(code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
  }
  /////////#####################################
########################
  myservo.write(90);
  delay(100);
}

//_____________________________________________

_____________________________
unsigned long previousMillis1 = 0; // Stores
the last time for 1ms task
int miliTime  = 1;
int dir = 1;
void loop() {
  // unsigned long currentMillis = millis(); //
Get the current time

  // // Task for 1 millisecond interval
  // if (currentMillis - previousMillis1 >=
(miliTime%2 +1 )) {
  //   miliTime++;
  //   if(miliTime>10)
  //     miliTime = 1;
  //   previousMillis1 = currentMillis;
  //   dir = -1*dir;

  // }

  // Serial.println("Encoder count = " +
String((int32_t)encoder.getCount()) + " " +
String((int32_t)encoder2.getCount()));
  // delay(10);
  pos_left = encoder.getCount();
  pos_right = encoder2.getCount();

  nh.spinOnce();
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer))
{ // Get the Latest packet

  mpu.dmpGetQuaternion(&q, fifoBuffer);
  mpu.dmpGetGyro(&gy, fifoBuffer);
  //mpu.dmpGetAccel(&aa, fifoBuffer);
  mpu.dmpGetGravity(&gravity, &q);
  // mpu.dmpGetLinearAccel(&aaReal, &aa,
&gravity);
  // mpu.dmpGetLinearAccelInWorld(&aaWorld,
&aaReal, &q);
  //Header
  imu_msg.header.seq = sequence++;
  imu_msg.header.stamp = nh.now();
  imu_msg.header.frame_id  ="imu_v1_1";

  //Linear Acceleration
  imu_msg.linear_acceleration.x =
gravity.x*9.8 ;
  imu_msg.linear_acceleration.y =
gravity.y*9.8 ;
  imu_msg.linear_acceleration.z =
gravity.z*9.8 ;
  //   imu_msg.linear_acceleration.x =
gravity.x ;
  // imu_msg.linear_acceleration.y = gravity.y ;
  // imu_msg.linear_acceleration.z = gravity.z ;
  //Angular Velocity
  imu_msg.angular_velocity.x = gy.x ;
  imu_msg.angular_velocity.y = gy.y ;
  imu_msg.angular_velocity.z = gy.z ;

  //Orientation
  imu_msg.orientation.x = q.x ;
  imu_msg.orientation.y = q.y ;
  imu_msg.orientation.z = q.z ;
  imu_msg.orientation.w = q.w ;

  imu_pub.publish(&imu_msg);

  }
  if((millis()-lastMilli) >= LOOPTIME)
  {
    infer_pos_x =
map(detect_x,20,100,135,45);
    myservo.write(infer_pos_x);
    digitalWrite(pump_pin,LOW);
    if(detect_y>80)
    {
      digitalWrite(pump_pin,HIGH);
      detect_y = 0;
    }else{
      digitalWrite(pump_pin,LOW);
    }



                                  // enter timed loop
    lastMilli = millis();

    // if (!nh.connected()){
    //   analogWrite(PIN_SIDE_LIGHT_LED,
lightValueNoComm[lightInc]);
    //   lightInc=lightInc+1;
    //   if (lightInc >= 25){
    //     lightInc=0;
    //   }
    // }
    // else{
    //   analogWrite(PIN_SIDE_LIGHT_LED,
lightValue [lightInc]);
    //   lightT = 3000 -
((2625/max_speed)*((abs(speed_req_left)+abs(sp
eed_req_right))/2));
    //   lightInc=lightInc+(30/(lightT/LOOPTIME)
);
    //   if (lightInc >= lightIncNumber){
    //     lightInc=0;
    //   }
    // }
    // pos_left = encoder.getCount();
    // pos_right = encoder2.getCount();


    if (abs(pos_left) <
5){
     //Avoid taking in account small
disturbances
      speed_act_left = 0;
    }
    else {
      speed_act_left=(((pos_left-
pos_left_prev)/encoder_cpr)*2*PI)*(1000/LOOPTI
ME)*radius;           // calculate speed of
left wheel
    }

    if (abs(pos_right) <
5){
    //Avoid taking in account small
disturbances
      speed_act_right = 0;
    }
    else {
      speed_act_right=(((pos_right-
pos_right_prev)/encoder_cpr)*2*PI)*(1000/LOOPT
IME)*radius;          // calculate speed of
right wheel
    }
    encoder.clearCount();
    encoder2.clearCount();
    // pos_left = 0;
    // pos_right = 0;

    speed_cmd_left = constrain(speed_cmd_left,
-max_speed, max_speed);
    PID_leftMotor.Compute();

    // compute PWM value for left motor. Check
constant definition comments for more
information.
    PWM_leftMotor =
constrain(((speed_req_left+sgn(speed_req_left)
*min_speed_cmd+0.1)/speed_to_pwm_ratio) +
(speed_cmd_left/speed_to_pwm_ratio), -255,
255); //

    if (noCommLoops >= noCommLoopMax)
{                   //Stopping if too much time
without command
      // leftMotor->setSpeed(0);
      // leftMotor->run(BRAKE);
      vTaskSuspend(motorTaskLHandle);
      xSemaphoreTake(xMutex, portMAX_DELAY);
      sharedDirL = 1;          // Set direction
to forward
      sharedPwmValL = 0;    // Set PWM value
      xSemaphoreGive(xMutex);
      vTaskResume(motorTaskLHandle); //
Activate task
      //setMotor(1,0,EN_left,IN_1,IN_2);
//IN_1,2 (left)
    }
    else if (speed_req_left ==
0){                            //Stopping
      // leftMotor->setSpeed(0);
      // leftMotor->run(BRAKE);
      vTaskSuspend(motorTaskLHandle);
      xSemaphoreTake(xMutex, portMAX_DELAY);
      sharedDirL = 1;          // Set direction
to forward
      sharedPwmValL = 0;    // Set PWM value
      xSemaphoreGive(xMutex);
      vTaskResume(motorTaskLHandle); //
Activate task
      //setMotor(1,0,EN_left,IN_1,IN_2);
//IN_1,2 (left)
    }
    else if (PWM_leftMotor >
0){                           //Going forward
      // leftMotor-
>setSpeed(abs(PWM_leftMotor));
      // leftMotor->run(BACKWARD);
      vTaskSuspend(motorTaskLHandle);
      xSemaphoreTake(xMutex, portMAX_DELAY);
      sharedDirL = 1;          // Set direction
to forward
      sharedPwmValL = PWM_leftMotor;    // Set
PWM value
      xSemaphoreGive(xMutex);
      vTaskResume(motorTaskLHandle); //
Activate task
      //setMotor(dir,abs(PWM_leftMotor),EN_left,
IN_1,IN_2);

    }
    else
{                                            /
/Going backward
      // leftMotor-
>setSpeed(abs(PWM_leftMotor));
      // leftMotor->run(FORWARD);

      vTaskSuspend(motorTaskLHandle);
      xSemaphoreTake(xMutex, portMAX_DELAY);
      sharedDirL = -1;          // Set direction
to forward
      sharedPwmValL = PWM_leftMotor;    // Set
PWM value
      xSemaphoreGive(xMutex);
      vTaskResume(motorTaskLHandle); //
Activate task
      //setMotor(-
1*dir,abs(PWM_leftMotor),EN_left,IN_1,IN_2);

    }

    speed_cmd_right = constrain(speed_cmd_right,
-max_speed, max_speed);
    PID_rightMotor.Compute();

    // compute PWM value for right motor. Check
constant definition comments for more
information.
    PWM_rightMotor =
constrain(((speed_req_right+sgn(speed_req_righ
t)*min_speed_cmd)/speed_to_pwm_ratio) +
(speed_cmd_right/speed_to_pwm_ratio), -255,
255); //
```

```
    if (noCommLoops >= noCommLoopMax)
{                   //Stopping if too much time
without command
      // rightMotor->setSpeed(0);
      // rightMotor->run(BRAKE);
      vTaskSuspend(motorTaskRHandle);
      xSemaphoreTake(xMutex, portMAX_DELAY);
      sharedDirR = 1;         // Set direction
to forward
      sharedPwmValR = 0;    // Set PWM value
      xSemaphoreGive(xMutex);
      vTaskResume(motorTaskRHandle); //
Activate task
      //setMotor(1,0,EN_right,IN_3,IN_4);
    }
    else if (speed_req_right ==
0){                         //Stopping
      // rightMotor->setSpeed(0);
      // rightMotor->run(BRAKE);
      vTaskSuspend(motorTaskRHandle);
      xSemaphoreTake(xMutex, portMAX_DELAY);
      sharedDirR = 1;         // Set direction
to forward
      sharedPwmValR = 0;    // Set PWM value
      xSemaphoreGive(xMutex);
      vTaskResume(motorTaskRHandle); //
Activate task
      //setMotor(1,0,EN_right,IN_3,IN_4);
    }
    else if (PWM_rightMotor >
0){                           //Going forward
      // rightMotor-
>setSpeed(abs(PWM_rightMotor));
      // rightMotor->run(FORWARD);
      vTaskSuspend(motorTaskRHandle);
      xSemaphoreTake(xMutex, portMAX_DELAY);
      sharedDirR = 1;         // Set direction
to forward
      sharedPwmValR = PWM_rightMotor;    // Set
PWM value
      xSemaphoreGive(xMutex);
      vTaskResume(motorTaskRHandle); //
Activate task

      //setMotor(dir,abs(PWM_rightMotor)-
10,EN_right,IN_3,IN_4);


    }
    else
{
//Going backward
      // rightMotor-
>setSpeed(abs(PWM_rightMotor));
      // rightMotor->run(BACKWARD);
      vTaskSuspend(motorTaskRHandle);
      xSemaphoreTake(xMutex, portMAX_DELAY);
      sharedDirR = -1;         // Set direction
to forward
      sharedPwmValR = PWM_rightMotor;    // Set
PWM value
      xSemaphoreGive(xMutex);
      vTaskResume(motorTaskRHandle); //
Activate task

      //setMotor(-1*dir,abs(PWM_rightMotor)-
10,EN_right,IN_3,IN_4);


    }

    if((millis()-lastMilli) >=
LOOPTIME){         //write an error if
execution time of the loop in longer than the
specified looptime
      Serial.println(" TOO LONG ");
    }

    noCommLoops++;
    if (noCommLoops == 65535){
      noCommLoops = noCommLoopMax;
    }

    publishSpeed(LOOPTIME);   //Publish
odometry on ROS topic
    // pos_left_prev = pos_left;
    // pos_right_prev = pos_right;
  }
 }

//Publish function for odometry, uses a vector
type message to send the data (message type is
not meant for that but that's easier than
creating a specific message type)
void publishSpeed(double time) {
  speed_msg.header.stamp =
nh.now();      //timestamp for odometry data
  speed_msg.vector.x =
speed_act_left;    //left wheel speed (in m/s)
  speed_msg.vector.y =
speed_act_right;   //right wheel speed (in m/s)
  speed_msg.vector.z =
time/1000;         //looptime, should be the
same as specified in LOOPTIME (in s)
  speed_pub.publish(&speed_msg);
  nh.spinOnce();
  nh.loginfo("Publishing topic: speed");
}

//Left motor encoder counter
// void encoderLeftMotor() {
//   if (digitalRead(PIN_ENCOD_A_MOTOR_LEFT) ==
digitalRead(PIN_ENCOD_B_MOTOR_LEFT))
pos_left++;
//   else pos_left--;
//   // Serial.print("Left_mot :");
//   // Serial.println(pos_left);
// }

// //Right motor encoder counter
// void encoderRightMotor() {
```

```
//   if (digitalRead(PIN_ENCOD_A_MOTOR_RIGHT)
== digitalRead(PIN_ENCOD_B_MOTOR_RIGHT))
pos_right--;
//   else pos_right++;
//   // Serial.print("Right_mot :");
//   // Serial.println(pos_right);
// }

template <typename T> int sgn(T val) {
    return (T(0) < val) - (val < T(0));
}
void setMotor(int dir,int pwmVal, int enx ,int
in1,int in2)
{
//dir = 1(forward)
//dir = -1(reverse)
  //ledcWrite(enx, pwmVal);
  analogWrite(enx, pwmVal);
  if(dir == -1)
  {
      digitalWrite(in1,HIGH);
      digitalWrite(in2,LOW);

  }
  else if(dir == 1)
  {
      digitalWrite(in1,LOW);
      digitalWrite(in2,HIGH);

  }
  else
  {
      digitalWrite(in1,LOW);
      digitalWrite(in2,LOW);
  }
}
```

```
void motorTaskL(void *parameter) {
  while (true) {
    // Read shared parameters safely
    xSemaphoreTake(xMutex, portMAX_DELAY);
    int dir = sharedDirL;
    int pwmVal = sharedPwmValL;
    xSemaphoreGive(xMutex);

    // Perform motor control
    setMotor(-
1*dir,abs(pwmVal),EN_left,IN_1,IN_2);
    delay(7.5);
    setMotor(1*dir,abs(pwmVal),EN_left,IN_1,IN_
2);
    delay(15.5);

  }
}
void motorTaskR(void *parameter) {
  while (true) {
    // Read shared parameters safely
    xSemaphoreTake(xMutex, portMAX_DELAY);
    int dir = sharedDirR;
    int pwmVal = sharedPwmValR;
    xSemaphoreGive(xMutex);

    // Perform motor control
    setMotor(-
1*dir,abs(pwmVal),EN_right,IN_3,IN_4);
    delay(7.5);
    setMotor(1*dir,abs(pwmVal),EN_right,IN_3,IN
_4);
    delay(15.5);

  }
}
```

***Table**: Driver Code for the main controller*

Source Code for Odometry publsiher (.cpp)

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/Vector3.h>
#include <stdio.h>
#include <cmath>

double radius =
0.0635;                          //Wheel
radius, in m
```

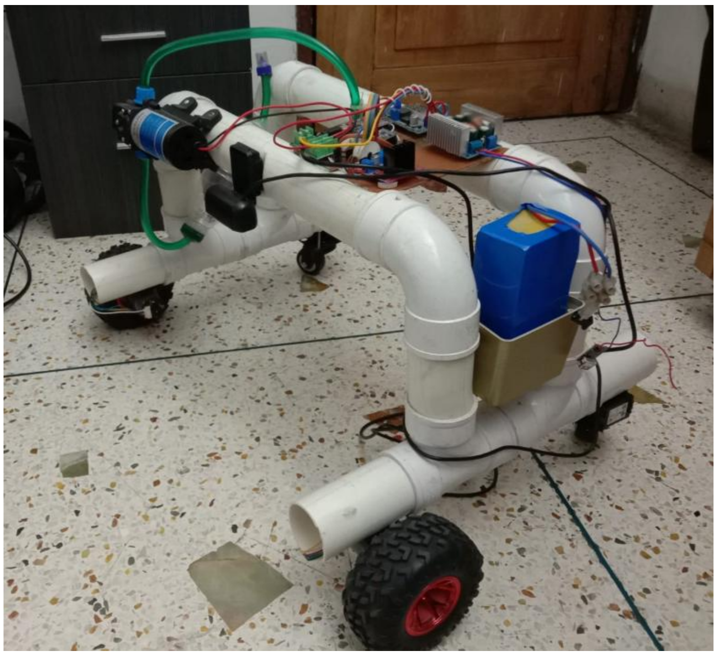
```
double wheelbase =
0.64;                         //Wheelbase, in m
double two_pi = 6.28319;
double speed_act_left = 0.0;
double speed_act_right = 0.0;
double speed_req1 = 0.0;
double speed_req2 = 0.0;
double speed_dt = 0.0;
double x_pos = 0.0;
double y_pos = 0.0;
double theta = 0.0;
ros::Time current_time;
ros::Time speed_time(0.0);

void handle_speed( const
geometry_msgs::Vector3Stamped& speed) {
  speed_act_left =
trunc(speed.vector.x*100)/100;
  ROS_INFO("speed left : %f", speed_act_left);
  speed_act_right =
trunc(speed.vector.y*100)/100;
  ROS_INFO("speed right : %f", speed_act_right);
  speed_dt = speed.vector.z;
  speed_time = speed.header.stamp;
}

int main(int argc, char** argv){
  ros::init(argc, argv, "nox_controller");

  ros::NodeHandle n;
  ros::NodeHandle nh_private_("~");
  ros::Subscriber sub = n.subscribe("speed", 50,
handle_speed);
  ros::Publisher odom_pub =
n.advertise<nav_msgs::Odometry>("odom", 50);
  tf::TransformBroadcaster broadcaster;

  double rate = 10.0;
  double linear_scale_positive = 1.0;
  double linear_scale_negative = 1.0;
  double angular_scale_positive = 1.0;
  double angular_scale_negative = 1.0;
  bool publish_tf = true;
  double dt = 0.0;
  double dx = 0.0;
  double dy = 0.0;
  double dth = 0.0;
  double dxy = 0.0;
  double vx = 0.0;
  double vy = 0.0;
  double vth = 0.0;
  char base_link[] = "base_link";
  char odom[] = "odom";
  // char kinect[] = "/kinect";
  // char camera_link[] = "/camera_link";
  ros::Duration d(1.0);
  nh_private_.getParam("publish_rate", rate);
  nh_private_.getParam("publish_tf",
publish_tf);
  nh_private_.getParam("linear_scale_positive",
linear_scale_positive);
  nh_private_.getParam("linear_scale_negative",
linear_scale_negative);
  nh_private_.getParam("angular_scale_positive",
angular_scale_positive);
  nh_private_.getParam("angular_scale_negative",
angular_scale_negative);

  ros::Rate r(rate);
  while(n.ok()){
    ros::spinOnce();
    current_time = speed_time;
    dt = speed_dt;          //Time in s
    ROS_INFO("dt : %f", dt);
    dxy = (speed_act_left+speed_act_right)*dt/2;
    ROS_INFO("dxy : %f", dxy);
    dth = ((speed_act_right-
speed_act_left)*dt)/wheelbase;

    if (dth > 0) dth *= angular_scale_positive;
    if (dth < 0) dth *= angular_scale_negative;
    if (dxy > 0) dxy *= linear_scale_positive;
    if (dxy < 0) dxy *= linear_scale_negative;

    dx = cos(dth) * dxy;
    dy = sin(dth) * dxy;

    x_pos += (cos(theta) * dx - sin(theta) *
dy);
    y_pos += (sin(theta) * dx + cos(theta) *
dy);
    theta += dth;

    if(theta >= two_pi) theta -= two_pi;
    if(theta <= -two_pi) theta += two_pi;

    geometry_msgs::Quaternion odom_quat =
tf::createQuaternionMsgFromYaw(theta);
    geometry_msgs::Quaternion empty_quat =
tf::createQuaternionMsgFromYaw(0);

    if(publish_tf) {
      geometry_msgs::TransformStamped t;
      //geometry_msgs::TransformStamped k;

      t.header.frame_id = odom;
      t.child_frame_id = base_link;
      t.transform.translation.x = x_pos;
      t.transform.translation.y = y_pos;
      t.transform.translation.z = 0.0;
      t.transform.rotation = odom_quat;
      t.header.stamp = current_time;

      // k.header.frame_id = kinect;
      // k.child_frame_id = camera_link;
      // k.transform.translation.x = 0.0;
      // k.transform.translation.y = 0.0;
      // k.transform.translation.z = 0.0;
      // k.transform.rotation = empty_quat;
      // k.header.stamp = current_time;

      broadcaster.sendTransform(t);
      // broadcaster.sendTransform(k);
    }

    nav_msgs::Odometry odom_msg;
    odom_msg.header.stamp = current_time;
    odom_msg.header.frame_id = odom;
    odom_msg.pose.pose.position.x = x_pos;
    odom_msg.pose.pose.position.y = y_pos;
    odom_msg.pose.pose.position.z = 0.0;
    odom_msg.pose.pose.orientation = odom_quat;
    if (speed_act_left == 0 && speed_act_right
== 0){
      odom_msg.pose.covariance[0] = 1e-9;
      odom_msg.pose.covariance[7] = 1e-3;
      odom_msg.pose.covariance[8] = 1e-9;
      odom_msg.pose.covariance[14] = 1e6;
      odom_msg.pose.covariance[21] = 1e6;
      odom_msg.pose.covariance[28] = 1e6;
      odom_msg.pose.covariance[35] = 1e-9;
      odom_msg.twist.covariance[0] = 1e-9;
      odom_msg.twist.covariance[7] = 1e-3;
      odom_msg.twist.covariance[8] = 1e-9;
      odom_msg.twist.covariance[14] = 1e6;
      odom_msg.twist.covariance[21] = 1e6;
      odom_msg.twist.covariance[28] = 1e6;
      odom_msg.twist.covariance[35] = 1e-9;
    }
    else{
      odom_msg.pose.covariance[0] = 1e-3;
      odom_msg.pose.covariance[7] = 1e-3;
      odom_msg.pose.covariance[8] = 0.0;
      odom_msg.pose.covariance[14] = 1e6;
      odom_msg.pose.covariance[21] = 1e6;
      odom_msg.pose.covariance[28] = 1e6;
      odom_msg.pose.covariance[35] = 1e3;
      odom_msg.twist.covariance[0] = 1e-3;
      odom_msg.twist.covariance[7] = 1e-3;
      odom_msg.twist.covariance[8] = 0.0;
      odom_msg.twist.covariance[14] = 1e6;
      odom_msg.twist.covariance[21] = 1e6;
      odom_msg.twist.covariance[28] = 1e6;
      odom_msg.twist.covariance[35] = 1e3;
    }
    vx = (dt == 0)?  0 :
(speed_act_left+speed_act_right)/2;
    vth = (dt == 0)? 0 : (speed_act_right-
speed_act_left)/wheelbase;
    odom_msg.child_frame_id = base_link;
    odom_msg.twist.twist.linear.x = vx;
    odom_msg.twist.twist.linear.y = 0.0;
    odom_msg.twist.twist.angular.z = dth;

    odom_pub.publish(odom_msg);
    r.sleep();
  }
}
```

***Table:** Source Code for odometry publisher.*

4 Implementation

4.1 Description

The Overall structure that we have constructed is given below:



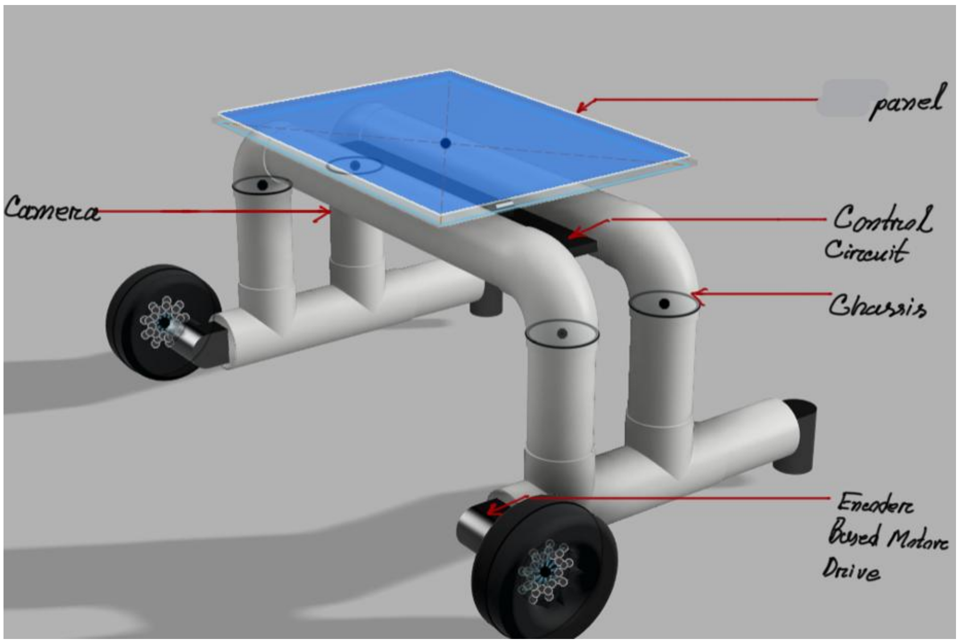


Figure:Comparison between CAD and actual constrution of the AgroBot

Main Controll unit:

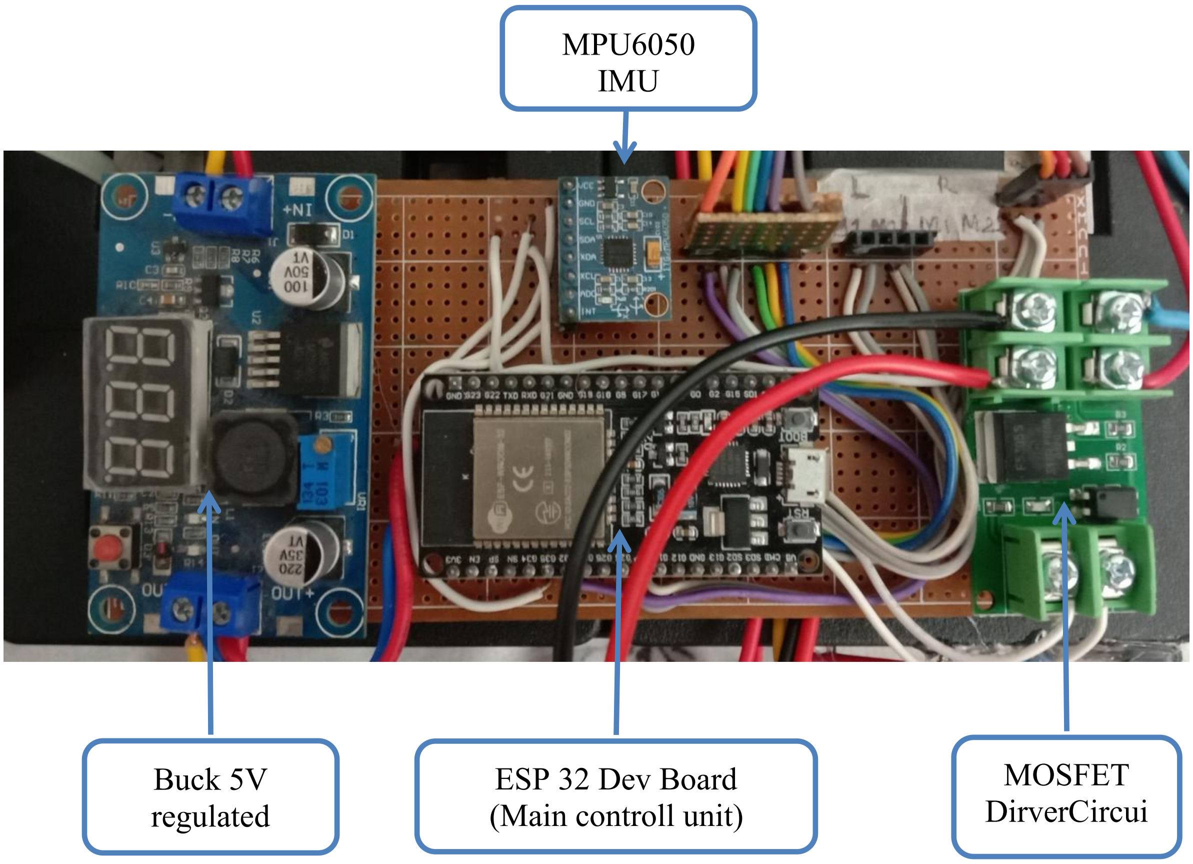


Figure: Main Controller unit(Hardware setup)

Power Management System:

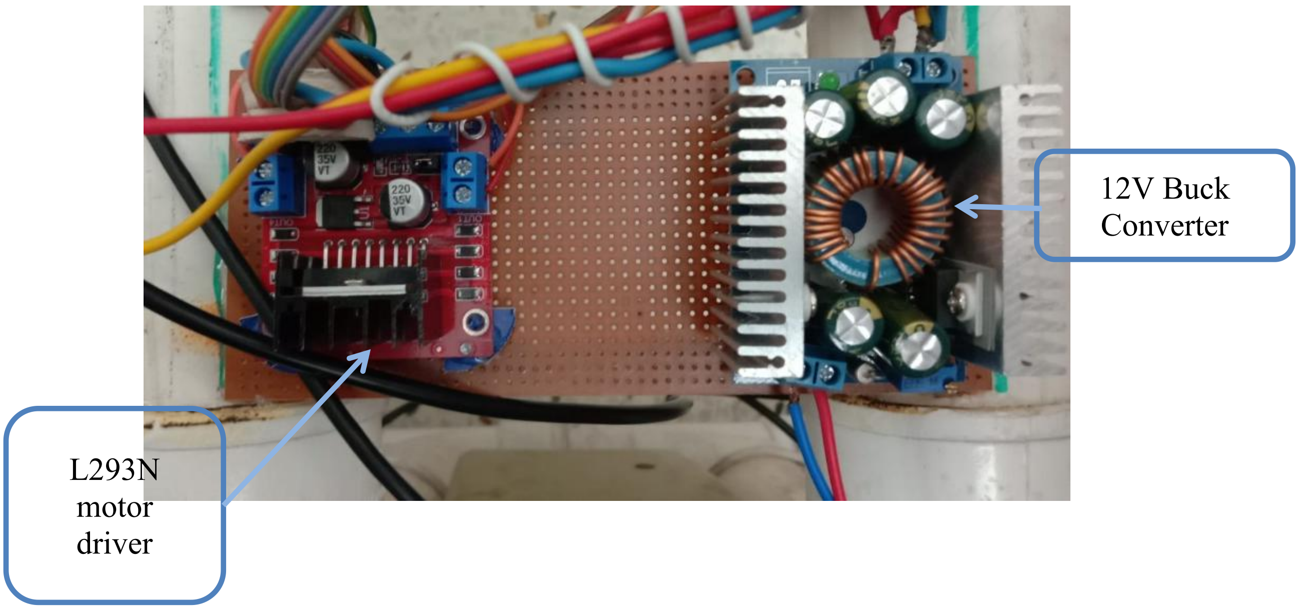


Figure: Power management unit(Hardware setup)

4.2 Experiment and Data Collection

For our Project we have experimented with vast amount of data.Finally we opted for a dataset that best suited our need and lighting condition. One important thing during the collection of dataset was to choose the proper angle that suited our physical construction of the robot.

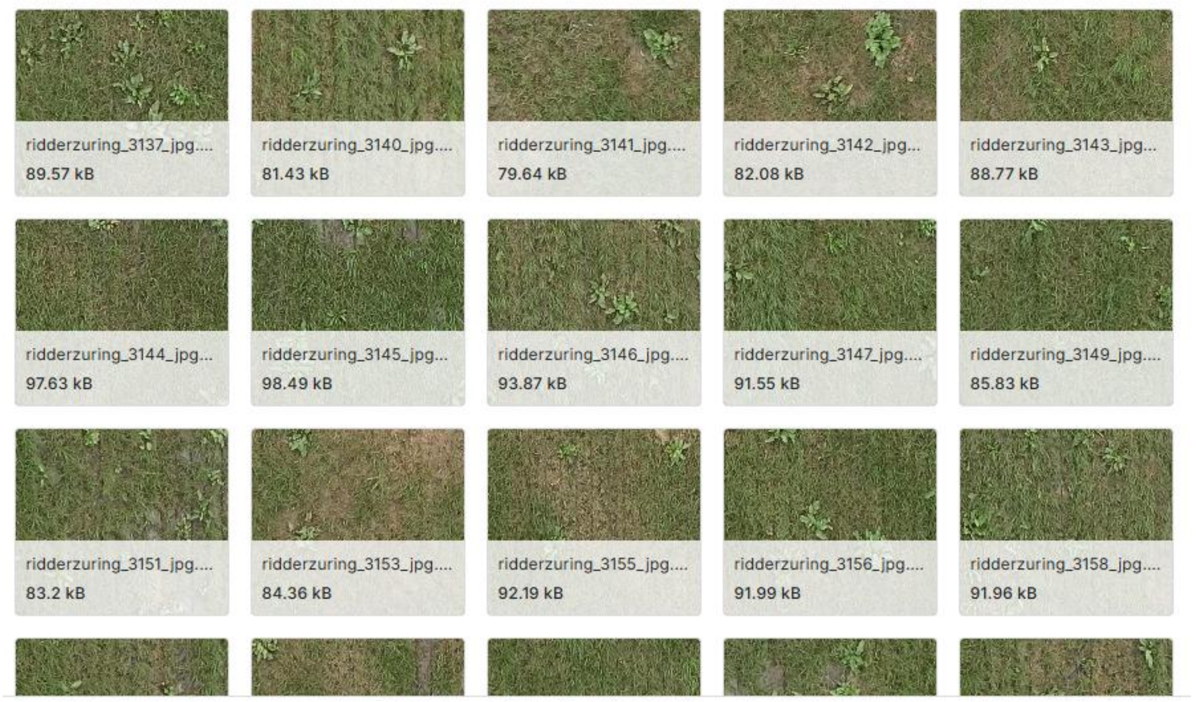

ridderzuring_3137_jpg.... 89.57 kB
ridderzuring_3140_jpg.... 81.43 kB
ridderzuring_3141_jpg.... 79.64 kB
ridderzuring_3142_jpg... 82.08 kB
ridderzuring_3143_jpg... 88.77 kB
ridderzuring_3144_jpg... 97.63 kB
ridderzuring_3145_jpg... 98.49 kB
ridderzuring_3146_jpg.... 93.87 kB
ridderzuring_3147_jpg.... 91.55 kB
ridderzuring_3149_jpg.... 85.83 kB
ridderzuring_3151_jpg.... 83.2 kB
ridderzuring_3153_jpg.... 84.36 kB
ridderzuring_3155_jpg.... 92.19 kB
ridderzuring_3156_jpg.... 91.99 kB
ridderzuring_3158_jpg.... 91.96 kB

Fig : Snapshot of our dataset.

4.3 Results

(i) Weed Detection: There are two significant things to consider while detecting weeds. The first one is accuracy and the second one is inference time. In order to measure the accuracy of object detection, mAP50 score is used.

mAP50 refers to Mean Average Precision at Intersection over Union (IoU) threshold of 0.50. IoU measures how much the predicted bounding box overlaps with the ground truth bounding box, and IoU of 0.50 means the predicted box overlaps at least 50% with the ground truth.mAP50 measures how well the model detects objects with at least 50% overlap and is a good indicator of how accurate the model is in detecting weeds.

Inference time refers to the time it takes for the model to process an image and make predictions (i.e., detecting and classifying objects). In real-world scenarios, especially in agriculture, it's important that weed detection models not only be accurate but also fast enough to be used in real-time applications, such as autonomous weeding robots or drones. Faster inference times mean more rapid decision-making and quicker action in the field.

Balancing **high mAP50 (accuracy)** with **low inference time** is key for developing effective weed detection systems. However it is difficult to maintain a good balance between these two as in order to decrease inference time, we have to sacrifice accuracy.

In order to have a decent inference time we used image of size (256,256,3). Our obtained score from test dataset is :

Model	***Weight***	***Precision***	***Recall***	***mAP50***	***Inference time***
YOLO v5	.onnx	0.758	0.731	0.763	0.2ms
	.pt	0.721	0.743	0.752	0.18ms

Following is an image of weed detection using ur model

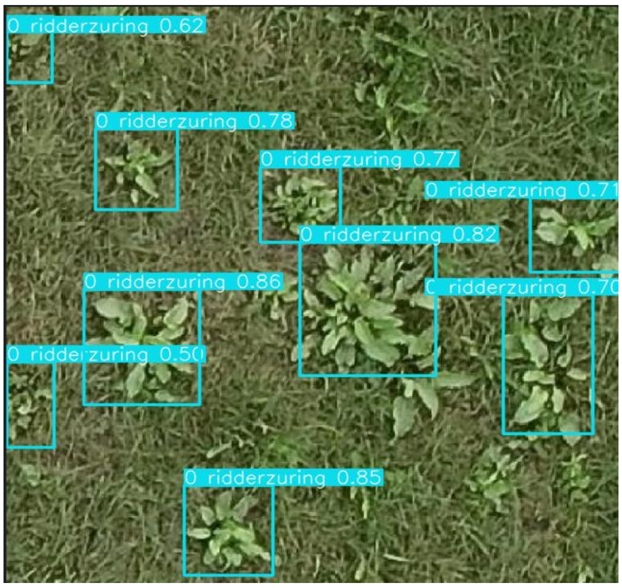


Fig : snapshot of live-inference

5 Design Analysis and Evaluation

5.1 Novelty

Some of recognisable novelty of our project is given below:

1. Here we have trained and hyper parameter tuned a vision model to detect particularly invasive vegetation(weeds) in a crop field.
2. We implemented and our whole model in ROS (noetic).
3. We have also developed a custom odometry based autonomous navigation system carter for our unique rover structure
4. .In order to make our ROS system OS independent ,we have implemented a custom Docker Image.Our Image contains all the necessary file to reproduce this project on different sysytem if necessary.

5.2 Design Considerations

5.2.1 Considerations to public health and safety

This project aims to enhance public health and safety by providing an automated solution that reduces manual labor and minimizes the exposure to harmful chemicals used in traditional weed control. By accurately targeting weeds and using less herbicide, the system mitigates health risks for farm-workers and consumers, contributing to safer agricultural practices.

5.2.2 Considerations to environment

Environmental impact is a key factor in the design of this system. By minimizing the use of chemical herbicides, the project reduces soil and water contamination. The system's precision spraying also limits the overuse of chemicals, promoting a healthier ecosystem and contributing to sustainable farming practices.

5.2.3 Considerations to cultural and societal needs

The project addresses societal needs by supporting small-scale farmers who often face labor shortages and high costs associated with weed control. By reducing dependence on chemical inputs and manual labor, this system can improve livelihoods, increase crop yields, and promote food security, especially in resource-limited agricultural regions.

5.3 Impact Assessment

5.3.1 Assessment of Societal and Cultural Issues

The introduction of this system to the agricultural sector can significantly benefit rural communities by lowering production costs and improving crop yields. It can also align with cultural practices of sustainable farming by encouraging environmentally friendly agricultural techniques.

5.3.2 Assessment of Health and Safety Issues

This project prioritizes health and safety by reducing the need for manual application of herbicides, thereby lowering the risk of exposure to hazardous chemicals. Additionally, it mitigates the physical strain associated with traditional weed removal methods, contributing to safer working conditions for farmers.

5.3.3 Assessment of Legal Issues

The system is designed in compliance with agricultural regulations regarding pesticide use and autonomous technology. By limiting chemical application and ensuring precise targeting, the project addresses legal concerns related to environmental and health standards in farming operations.

5.4 Sustainability and Environmental Impact Evaluation

This system enhances sustainability by significantly reducing the use of chemical herbicides, thus preserving soil quality and preventing pollution of water sources. Its energy-efficient design and targeted spraying mechanism contribute to sustainable agricultural practices, while the reduced environmental footprint aligns with global efforts to combat climate change.

5.5 Ethical Issues

The development of the AgroBot system incorporates several ethical considerations that guide its design, implementation, and operational use. Below are the key ethical challenges and the proactive measures taken to address them:

Privacy Concerns: The AgroBot uses advanced technologies such as computer vision for weed detection, which may raise concerns regarding privacy. Although the system is designed for agricultural fields, there is a potential for unintended data capture in certain settings, such as adjacent properties or public spaces. To mitigate this, the system is configured to focus solely on crop areas, ensuring that no extraneous data is collected. Moreover, any data stored is anonymized and used exclusively for improving the system's performance in agricultural environments.

Safety and Human Well-being: As the system operates autonomously in agricultural fields, ensuring human safety is paramount. The AgroBot is equipped with fail-safe mechanisms to prevent accidental harm to humans or animals. These include sensors to detect obstacles, automatic shutdown in unsafe situations, and remote override capabilities for manual intervention if necessary. These safety measures are critical to ensuring that the system operates without posing risks to farmworkers or the environment.

Data Security: As the AgroBot collects operational data to improve its performance, there are concerns regarding the security of this data. The system employs encryption and secure communication protocols to protect against data breaches or unauthorized access. Data collected is limited to agricultural metrics, ensuring that personal or sensitive information is not compromised.

By proactively addressing these ethical challenges, our project not only aligns with ethical principles but also enhances its credibility, trustworthiness, and overall positive impact on society, promoting a safer and more sustainable future.

6 Reflection on Individual and Team work

6.1 Individual Contribution of Each Member

1906027 : Chassis design & construction .
1906048 & 1906102 : Circuit design and implementation.
1906078 : Pump and spray system development
1906109 : YOLOv5 model training and testing .Improvement of the performance of inference.
1906112 : Full ROS system development, implementing autonomous movement and integrating it with Detection scheme.

6.2 Mode of TeamWork

We tried work synchronously to improve the efficiency of our team.Our main target was to utilize all the resources that we within this short period of time.At first we divide the project in 2 distinct segments , Machine learning and odometric control. Then each of our team members individually researched their designated topics. Few of our members were assigned to acquire market status of each individual components. .

6.3 Diversity Statement of Team

During the assignment of individual tasks we searched for particular expertise and previous experiences.All of our team mates communicated freely with each other and expressed their concerned regarding any approaching issue.

6.4 Log Book of Project Implementation

Date	Milestone achieved	Individual Role	Team Role	Comments
September 26	Data-set finalization and planning		Whole team	
September 28	Navigational system design	1906112		
October 2	Data synchronization between Between the imaging sensor & main control board was achieved		Whole team	Successfully achieved desired result
October 3	Power management system was developed	1906048 & 1906102		
October 5	The main rover body components finalized & building process initiated	1906027		
November 7	Sample image collection & training of YoloV5 model was done	1906109		
November 15	Ros Move Base integration and Extended Kalman filter integration	1906112		Was a somewhat iterative process
November 21	Various modules software code unified and synchronized	1906109 &1906112		
September 1	The water spray and pump system developed	1906078		
November 4	Final modification and imrovement in odometry of the rover body		Whole team	Met expectation

7 Communication

7.1 Executive Summary

Our project, AgroBot, introduces an intelligent, vision-guided solution to tackle weed detection and elimination in agriculture. By leveraging cutting-edge technologies such as computer vision, machine learning, and robotics, AgroBot autonomously identifies and targets weeds with precision. Equipped with a selective spraying mechanism, the system minimizes the use of chemical herbicides, thereby reducing environmental impact and preserving soil health.

This innovative approach addresses key challenges faced by farmers, including labor shortages and high production costs, while promoting sustainability in agriculture. Designed to be affordable and efficient, AgroBot is particularly suited for small-scale farming systems, making advanced agricultural technology accessible to economically disadvantaged communities. The project embodies a commitment to sustainability, public health, and ethical considerations, paving the way for a more efficient and environmentally friendly future in farming.

7.2 User Manual

The robot will autonomously detect and extinguish weed by spraying field selectively. Before this the robot must be initialized accordingly.

- Firstly the the container must be filled appropriate pesticide.
- Then the container must be attached and the air release valve must be opened.
- Then Power port must be connected.
- The rover body must be placed in a flat surface for at least 2 seconds.
- Then the rover will automatically starts its detection and execution routine.
- In order to turn it off ,just unplug its power port.

8 Project Management and Cost Analysis

8.1 Bill of Materials

Item	**Quantity**	**Cost per item(BDT)**	**Total cost(BDT)**
Raspberry pi 5(8gb RAM)	1	15500	15500
Web Camera	1	1450	1450
Servo	1	450	450
DC geared motor	2	1500	3000
Platform(chassis)	1	1000	1000
17mm wheel	2	120	240
Pump (DIAPHRAGM WATER PUMP)	1	1500	1500
Battery(Li-ion used)	1	2000	2000
Miscellaneous	-	1000	1000
Total			**25140**

9 Future Work

Future works include:

1. Increasing the and diversifying the dataset , to increase the detection accuracy.
2. Scale this project for more diverse task, such leaf decease detection and prevention.
3. Incorporate GPS RTK(Real Time Kinematics) for cm level accuracy while traversing the crop field.

4. Increase the amount of spraying nozzle to improve overall efficiency.
5. Incorporate Variable-radius wheel for better terrain traversing capability.

10 References

1. https://thesai.org/Downloads/Volume15No4/Paper_11-Automated_Weeding_Systems_for_Weed_Detection.pdf
2. https://www.mdpi.com/2624-7402/6/3/187
3. https://ecorobotix.com/en/avo/
4. https://www.earthsense.co/terramax
5. https://www.naio-technologies.com/en/ted/
6. https://carbonrobotics.com/laserweeding?gad_source=1&gclid=Cj0KCQiAvP-6BhDyARIsAJ3uv7YBoeZZkFQMO5X-Xgc0Goe7pLGckuMZl-VAGnOTaMav5fA1B8YTtxwaAk2bEALw_wcB
7. https://farmdroid.com/products/farmdroid-fd20/
8. https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2022.1091655/full