Misr University of Science and Technology
College of Engineering and Technology
Department of Mechatronics Engineering

B. Eng. Final Year Project

(Seed Sowing Robot)

By: Group (5)

NAME	ID
Assem Yasser Mohamed Elsharbeiny	80681
Khaled Abdelnasser Elshamndy	80768
Omar Ashraf Abdelmawgoud	80601
Mohamed Ahmed Mohamed Khater	79573
Mohamed Kamal Ali Ahmed	80651

Supervised By:

Supervisor(s)

Dr. Bahaa Nasser	Head of Department
Prof. Dr. Mohamed Hamdy	Department professor
Prof. Dr. Mohamed Ibrahim	Menofia University Professor
Prof. Dr. Khaled Mohamed Kamal	Vice-dean of the High Institute for Engineering & Technology
Assist. Prof. Dr. Tahani Wileam	Beni-suef University Assist. Professor
Dr. Alaa Zakria Nasser	Military Technical College
Dr. Bekhet Mohamed	Japanese University

Spring 2023

Misr University of Science and Technology
College of Engineering and Technology
Department of Mechatronics Engineering

B. Eng. Final Year Project

(Seed Sowing Robot)

By: Group (5)

NAME	ID
Assem Yasser Mohamed Elsharbeiny	80681
Khaled Abdelnasser Elshamndy	80768
Omar Ashraf Abdelmawgoud	80601
Mohamed Ahmed Mohamed Khater	79573
Mohamed Kamal Ali Ahmed	80651

Supervised By:

Supervisor(s)

Dr. Bahaa Nasser	Head of Department
Prof. Dr. Mohamed Hamdy	Department professor
Prof. Dr. Mohamed Ibrahim	Menofia University Professor
Prof. Dr. Khaled Mohamed Kamal	Vice-dean of the High Institute for Engineering & Technology
Assist. Prof. Dr. Tahani Wileam	Beni-suef University Assist. Professor
Dr. Alaa Zakria Nasser	Military Technical College
Dr. Bekhet Mohamed	Japanese University

Spring 2023

Abstract

Access to Agriculture is a major first step towards a better life. Agricultural robotics has been a popular subject in recent years from an academic as well as a commercial point of view. This is because agricultural robotics addresses critical issues such as seasonal shortages in manual labor, for example, during harvest, as well as the increasing concern regarding environmentally friendly practices. Much agricultural work is, by its nature, physically demanding. The risk of accidents is increased by fatigue, poorly designed tools, difficult terrain, exposure to extreme weather conditions, and poor general health, associated with working and living in rural communities. These problems are compounded by the fact that working and living conditions are interwoven. This project focuses on implementation a system that decreases operating costs, reduces the time required to dig, consume less water, and the performance of seed sowing using agribot. This reduces the dependence on staff. The robot for sowing seeds and digging will travel through various rows of soil and perform digging, sowing seeds and covering the soil by covering it and its integrated with irrigation system. This paper describes the complete installation of agribot including hardware and software facet.

Acknowledgment

We would like to express our sincere gratitude to all who contributed to the success of this project.

First and foremost, we would like to thank our project advisors and mentors for their invaluable guidance, support, and expertise throughout the project. Their insights and feedback were critical to the success of the project, and we are grateful for their time and dedication.

We would also like to express our appreciation to our team members who have worked tirelessly to design, develop and test the seed sowing car prototype and simulator. Their hard work, enthusiasm and dedication was instrumental in bringing the project to fruition.

List of Abbreviations

ROS	Robot Operating System
LIDAR	light detection and ranging
TFT	thin-film-transistor
LCD	liquid-crystal display
AMCL	Adaptive monto carlo localization
EKF	Extended Kalman filter
GUI	Graphic user interface
GPIO	General purpose I/O
IMU	inertial measurement unit
PWM	Pulse width Modulation
PGM	Portable gray map
SLAM	Simultaneous localization and mapping
TF	Transformation
URDF	Unified robot description format
XML	Extensible markup language

Table of Contents

List of Figures

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

List of Tables

Chapter 2

Chapter 3

Chapter 1: Introduction and Literature review

Introduction

Agriculture has been the backbone of the Indian economy and it will continue to remain so for a long time. It has to support almost 17 percent of the world population from 2.3 percent of the world's geographical area and 4.2 percent of the world's water resources. The net sown area is 142 Million hectares. The basic objective of the sowing operation is to put the seed and fertilizer in rows at desired depth and spacing, cover the seeds with soil and provide proper compaction over the seed. The recommended row to row spacing, seed rate, seed to seed spacing, and depth of seed placement vary from crop to crop and for different agricultural and climatic conditions to achieve optimum yields and an efficient sowing machine should attempt to fulfill these requirements. Also, saving in cost of the operation time, labor and energy are other advantages to be derived from the use of improved machinery for such operations. A traditional method of seed sowing has many disadvantages. This paper is about a robot which do operations like seedsowing, ploughing and levelling and its was controlled by ROS and Embedded system.

1.1 Background of the Study

Agribot, generally known as an agricultural robot, is a machine used in agriculture. Furthermore, the new concept of the paper makes it possible to dig and plant seed seeds and cover the earth automatically so that human efforts will be reduced to 90 percent.

Figure 1.1: Latest generation of seed sowing robots (The Fendt Xaver comes of age: Report, 2020)

1.2 Problem Definition

Introducing to increase the human efforts as it requires less amount of manmade labour and can be handle efficiently without a skilled operator. Moreover, the use of robotics in agriculture enhances worker and farmer productivity and working conditions. In addition, Precision farming is increasingly being driven by intelligent systems. Many agricultural procedures are currently carried out automatically.

1.3 Project Aim and Objectives

The aim of the project is to design, simulate and implement Agricultural Robotic to help the sector in its efficiency and in the profitability of the processes. In other words, mobile robotics works in the agricultural sector to improve productivity, specialization and environmental sustainability.

A well-defined project objectives are stated below :

- Surveying the previous work in agriculture robotics system to choose the best model.
- Investigating different agriculture robotics problems to identify the main agriculture robotics problems.
- Summarizing key findings of the reviewed previous work to choose a suitable model.
- Designing an agriculture robotics system.
- Manufacturing a smart agriculture robot system.
- Testing and verification of the manufactured model.
- Cost Study.

1.4 Proposed Solution

The project attends to solve the shortcomings by creating a machine that performs multiple operations and automates it to extend the yield on an outsized scale and reduce the manpower cost in addition to design and implement the device itself moreover to simulate and test it. Also, the design and simulation of the hardware will be done using Solid works to assure that the design is completely efficient. Then the electric circuit design will be simulated and tested in order to achieve the highest response for closed loop fully controlled system. Finally, will be the implementation and real-life examination.

1.5 Introduction to ROS

ROS stands for Robot Operating Systems which is used to control robotic components from a PC. ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex robotic projects. It is comprised of several nodes each communicate with other nodes using publish/subscribe messaging model. It Controls only in the form of an imaginary environment through some commands and codes.

1.5.1 Why ROS?

The nodes in ROS do not have to be on the same system or even of the same architecture, you could have an Arduino publishing messages, a laptop subscribing to them, and an Android phone driving motors. This makes ROS flexible and adaptable to the needs of the user.ROS is also open source, maintained by many people.

Figure 1.2 meaning ROS

1.6 Introduction to Embedded system

is a computer system a combination of a computer processor, computer memory, and input/output peripheral devices that has a dedicated function within a larger mechanical or electronic system. It is embedded as part of a complete device often including electrical or electronic hardware and mechanical parts. Because an embedded system typically controls physical operations of the machine that it is embedded within, it often has real-time computing constraints. Embedded systems control many devices in common use. In 2009, it was estimated that ninety-eight percent of all microprocessors manufactured were used in embedded systems.

Literature review
1.7 Manual operated seed sowing machine

Figure 1.3: Manual operated seed sowing machine, (Kyada A et al, 2010).

The design and development of a manually operated seed sowing machine are presented in this research article. In this paper, they discuss the goal of the design of the seed sowing machine, elements influencing seed emergence, and certain mechanisms. The main goal of sowing operations is to arrange the seed in rows at the correct depth and seed-to-seed spacing, cover the seeds with soil, and apply proper compaction over the seed. To produce the best yields, different crops and agro climate circumstances require varying seed-to-seed spacing and depth recommendations. From this, they can infer that mechanical parameter, such as regularity of seed placement and dispersion along rows, have an impact on seed germination. This power transmission system has plunger and seed meter devices. A mechanism is employed. When the machine is pushed, the power wheel rotates and transfers energy to the plunger via a chain and sprocket mechanism. Currently, a cam mounted on a sprocket shaft pushes the plunger downward. When the plunger has penetrated the earth, the flapper is opened during the backward stroke, causing the seed to be separated from the plunger and placed into the hole. This gives

them the impression that using a belt with small holes of a specific thickness will be advantageous for their project. Given that their automatic seed feeder can only handle little seeds, a conveyor belt with a motor is a good idea, (Kyada A et al, 2010).
According to Adisa and Braide (2012), the fundamental criteria for small-scale cropping machines include: being appropriate for small farms, simple in design, increasing planting efficiency, and decreasing the

laborintensiveness of manual planting methods. They created a row planter with 88% field effectiveness. The primary duties of planters include opening the seed furrow to the correct depth, dosing the seeds, depositing them in the furrow, and covering the seeds to the appropriate level for the type of crop being planted. Hopper, metering system, and furrow opener are the three basic parts of the planter.

Ikechukwu et al. (2014) designed and constructed a manually operated single row maize planter capable of accurately delivering seeds in a straight line with uniform depth in the furrow and with uniform spacing between the seeds. The results showed that the planter's capacity was 0.0486 hectare/h, with a field efficiency of 88%.

This paper focused on the metering mechanism, which comes in a variety of forms, is the brain of the sowing machine. The metering system works by releasing the seeds at the correct pace, assisting in creating the precise spacing required for good yields, and preventing skipping, doubling, and injuring the seeds during the planting process (Ani et al., 2016).

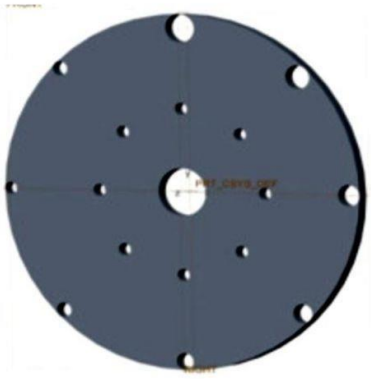


Fig. 1. View of a plate type seed-metering device for the planter

Fig. 2. Photograph of seed-metering of maize planter

Figure 1.4: the metering mechanism (Ani et al., 2016)

1.8 Remotely controlled seed sowing machine

This study report provides information on globalization; different technologies are being updated to reflect a new development based on automation that operates extremely firmly, highly efficiently, and quickly. Growing consumer demand for high-quality agricultural products and dwindling labor supply in rural farming areas make the advancement of agricultural system innovation a crucial task. The planned technology uses a microcontroller to seed agricultural robots. The system's goal is to assess the soil's PH, temperature, moisture, and humidity while also sowing. The robot is remote-controlled. The mechanism that was built allows for remote control of the robot's navigation. Through an internet system, the robot and remote system are linked. DC motors are employed by the robot to move around. A controller is used to control the motor's speed. Seeding is managed by the solenoid. This essay provides information about belt conveyor automation and motor use (Keshav Pureha, 2015).

This paper focused on developing and improving the manufacturing of agriculture robot. This robot's primary use in agriculture is at the seeding stage of harvesting. This robot reduces the need for human labor. This study illustrates a robot that is equipped to

automatically distribute seeds and apply pesticides. The entire operation is under the microcontroller's supervision. The manual control robot employs a remote controller to operate the apparatus and aids in guiding the robot on the playing field (Shriyash Thawali et. Al, 2017).

The aim of this paper was to seed at a specific d'pth, Vinay Kumar Tiwari et al. (2017) concentrated on all the basic automatic seeding mechanisms for moving them forward and backward as well as for raising and lowering the plough. This autonomous robot prefers to use systems that use relay switches, microcontrollers, and other devices to sow seeds and steer a vehicle. As an alternative to traditional farming, the seeding method is now employed.

1.9 Autonomous operated seed sowing machine

In this paper, a high-speed system for an advanced agricultural process that includes growing on a robotic platform is presented. The robotic system is an artificial agent that conveys a sense of autonomy and is driven by a four-wheeled DC motor. Machines cultivate the farm according to the crop, taking precise rows and columns into consideration. The infrared sensor senses the turning position of the vehicle at the end of the land and obstructions in the path. Water pressure can be used to find and resolve the seed block. Remote operation of the machine is possible, and a solar panel is utilized to charge a DC battery. In addition, the microcontrollers are programmed in assembly language. The microprocessor is with the assistance of a DC motor, the microcontroller is employed to regulate and track the system motion of the vehicle. The finished product of the device is also shown. In conclusion: The prerequisites and advancements achieved toward a precise autonomous farming system were discussed in this research. Less electricity is needed because the assembly is designed to cultivate ploughed area automatically. With the aid of water pressure, the seed blockage issue is solved.

Therefore, this project improves accuracy and efficiency. The project uses two unique mechanisms (Joshi et al, 2018).

1.10 Autonomous Operated Seed Sowing Machine Advantages

- It not required high operational costs and productivity increased.
- Saves money and reducing manual labour.
- Saves time and effort of the farmers.

1.11 Autonomous Operated Seed Sowing Machine Drawbacks

- Higher investment is needed just once.
- While agricultural robots and other equipment are complicated, farmers must receive training before using them.
- Complex algorithms are needed to design agricultural robots so they can perform any task.
- Before being used on any job site, agricultural robots must receive extensive testing.

- **The contents of the following chapters :**

Chapter 2: Design and Modeling of the Agriculture Robot

This chapter consists of the design and modeling of the agriculture robot are discussed. It begins with an overview of the topic, followed by static and fatigue stress analysis for the chassis and motor shaft. Power screw analysis and bolt selection are examined for optimal performance. The design of the drill is addressed, along with force analysis and motor sizing considerations. The chapter also covers analytical motor sizing calculations and specific design considerations for auger motors. Mechanical design aspects, including the creation of a 3D model and design drawings, are explained. The materials used and assumptions made during the design process are outlined, and the chapter concludes with a discussion on weights.

Chapter 3: Electronics Components & Circuit

This chapter consists of the electronics components and circuitry of the agriculture robot. It provides an overview of the chapter's contents, followed by a detailed description of various electronic components such as the Arduino Mega 2560, DC Motor Driver, LIDAR, Raspberry Pi 4 Model B, and more. The chapter also discusses the electrical circuit used in the robot, highlighting the connections and interactions between the different components.

Chapter 4: Robot Operating System

This chapter consists of the Robot Operating System (ROS) takes center stage in Chapter 5. The chapter begins with an overview and introduction to ROS, followed by an explanation of the objectives of using ROS in the agriculture robot. The steps for software

implementation are discussed, starting with the installation of Ubuntu and ROS. The chapter also covers the creation and initialization of a workspace folder and introduces key ROS terminologies such as nodes, packages, messages, topics, services, actions, and parameters. The chapter concludes with an exploration of ROS tools like the 3D visualization tool (Rviz) and the ROS GUI tool (rgt).

Chapter 5: Software

This chapter consists of the software aspects of the agriculture robot. It starts with an overview of the system architecture, providing an understanding of the software's structure and organization. The chapter then explores software classification and the integration of hardware and software components. Key topics such as Rosserial, SLAM (Simultaneous Localization and Mapping), transform configuration, and sensor fusion are discussed. The chapter also covers the navigation stack, including Move Base, global and local planners, costmap configuration, and AMCL (Adaptive Monte Carlo Localization). Additionally, it touches on the installation of required packages, Arduino pin configurations, simulation and testing procedures, block diagrams, user manuals, flowcharts, and algorithms used in the software implementation.

References

The chapter concludes with a list of references used throughout the document, providing readers with additional sources of information for further study and exploration.

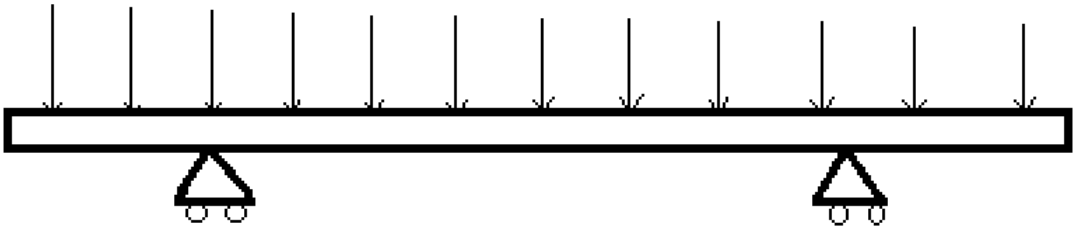
Chapter 2: Design and Modeling of the Agriculture Robot

Overview

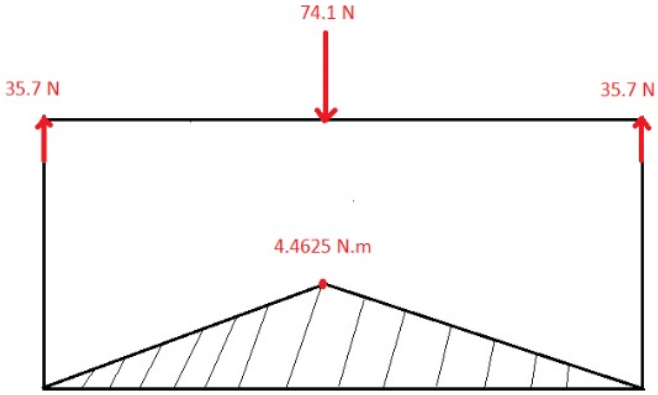
In this Section we will go through the Hardware design, drawings and the materials used to achieve the desired strength, and information about the material used, With stress, strain, displacement analysis of the project. As well as Motor sizing and wire sizing calculations in order to Choose and buy the required motor and wires for our project. And at the end there will be the list of components used with a brief description.

2.1 Static and fatigue stress analysis for chassis

Length = 0.51 m
The force applied on chassis = 140 N



$$\sum M_c = 0$$
$$R_A \times 0.08593 - 71.4(0.21093) + R_B(0.33593) = 0$$
$$\sum M_D = 0$$
$$-R_A(0.42163) + 71.4(0.29663) - R_B(0.17163) = 0$$
$$\sum F_y = 0$$
$$R_B - 71.4 + R_A = 0$$
$$\therefore\ R_B = 71.4 - R_A$$
$$R_A = 35.7\ N$$
$$R_B = 35.7\ N$$



$$M = 21.2\ N.m$$
$$c = 3 \times 10^{-3}\ m$$
$$I = \frac{b\,h^3}{12} = 3.816 \times 10^{-8}\ m^4$$

$$\sigma = \frac{M\,C}{I} = \frac{21.2\,(3\times10^{-3})}{3.816\times10^{-8}} = 1.9\ MPa < (\sigma_u) = 310\ \text{MPa}$$

Area of chassis = $0.37488672 m^2$
Ultimate strength of aluminum-5754 (σ_u) = 310 MPa
Yield strength of aluminum-5754 (σ_y) = 276 MPa
$$\sigma_r = \frac{\sigma_u}{2} = 155\ MPa$$

A = 0.7
B = 1
C=1
$k_t = 1$
k_f =1.5

$$\sigma_{max} = k_t \frac{F_{max}}{A} = \frac{140}{0.37488672} = 373\ Pa$$

$$\sigma_{min} = k_t \frac{F_{min}}{A} = \frac{70}{0.37488672} = 187\ Pa$$

$$\sigma_m = \frac{\sigma_{max} + \sigma_{min}}{2} = \frac{784.3 + 392.2}{2} = 280\ Pa$$

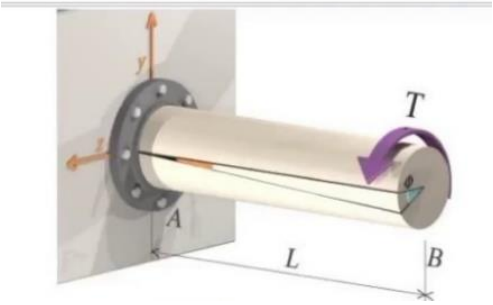
$$\sigma_v = \frac{\sigma_{max} - \sigma_{min}}{2} = \frac{784.3 - 392.2}{2} = 93\ Pa$$

$$\sigma_{en} = \sigma_m + (\frac{\sigma_y}{\sigma_r}) \times \frac{k_f \sigma_v}{ABC}$$

$$= (588.3) + \left(\frac{276}{155}\right) \times \frac{1.5(196.1)}{0.7} = 1336.6\ Pa$$

2.2 Static and fatigue stress analysis for motor shaft

$l = 10\ mm$
$k_b = 1.5$
$k_t = 1$
$M_t = 1\ N.m$
Ultimate strength of steel = 420 MPa
Yield strength of steel = 350 MPa
Shear stress (σ_s) = 0.18(σ_u) = $0.18 \times 420 = 75.6\ MPa$
$A = 0.6\ , B = 1\ , C = 1\ , N = 2$
$\sigma_{ys} = 0.6\sigma_y = 210\ MPa$
$\sigma_r = 0.5\sigma_u = 0.5 \times 420 = 210\ MPa$
$T_m = 1.5\ N.m$
$T_v = 1\ N.m$



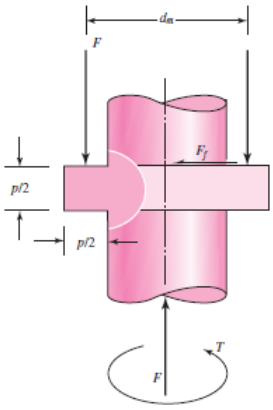
$$\sigma_{ms} = \frac{16T_m}{\pi d^3} = \frac{16 \times 1.5}{\pi d^3}$$

$$\sigma_{vs} = \frac{16T_v}{\pi d^3} = \frac{16 \times 1}{\pi d^3}$$

$$\frac{1}{N} = \frac{\sigma_{ms}}{\sigma_{ys}} + \frac{k_f \sigma_{vs}}{\sigma_r ABC}$$

$$\frac{1}{2} = \frac{24}{\pi d^3(210 \times 10^6)} + \frac{16}{(210 \times 10^6)(\pi d^3)(0.6)}$$
$$d = 5.356 \times 10^{-3} m$$

2.3 Power screw analysis



$d_m = d - \frac{p}{2} = 4.110$ mm
$d_r = d - p = 3.4$ mm
$d_c = 4.2$ mm
$l = np = 1.6$ mm

the torque required to turn the screw against the load is

$$T_R = \frac{Fd_m}{2}\left(\frac{l + \pi f d_m}{\pi d_m + fl}\right) + \frac{Ff_c d_c}{2}$$
$$= \frac{(2.992\times10^{-3})4.110}{2}\left(\frac{1.6+\pi\times0.20\times4.110}{(\pi\times4.110)-(0.20\times1.6)}\right) + \frac{(2.992\times10^{-3})0.2\times4.2}{2} = 2.439\times 10^{-3} \text{ Nm}$$

the load-lowering torque is

$$T_L = \frac{Fd_m}{2}\left(\frac{\pi f d_m - l}{\pi d_m + fl}\right) + \frac{Ff_c d_c}{2}$$
$$= \frac{(2.992\times10^{-3})4.110}{2}\left(\frac{\pi\times0.2\times4.110-1.6}{(\pi\times4.110)+(0.2\times1.6)}\right) + \frac{(2.992\times10^{-3})0.2\times4.2}{2} = 0.7298\times 10^{-3} \text{ Nm}$$

The overall efficiency in raising the load is

$$e = \frac{Fl}{2\pi T_R} = \frac{(2.992 \times 10^{-3}) \times 1.6}{2\pi(2.43910^{-3})} = 0.3124$$

The body shear stress τ due to torsional moment *TR* at the outside of the screw body is

$$\tau = \frac{16T_R}{\pi d_r^3} = \frac{16(2.439)}{\pi(3.4^3)} = 0.316\ MPa$$

The axial nominal normal stress σ is

$$\sigma = -\frac{4F}{\pi d_r^2} = -\frac{4(0.7298)}{\pi(3.4^2)} = -0.0804\ MPa$$

The bearing stress $\sigma_B$is, with one thread carrying 0.38*F*,

$$\sigma_B = -\frac{2(0.38F)}{\pi d_m(1)p} = -\frac{2(0.38 \times 2.992)}{\pi(4.110)(1)(1.6)} = -0.110069\ MPa$$

The thread-root bending stress σ_b with one thread carrying $0.38F$ is

$$\sigma_b = \frac{6(0.38F)}{\pi d_r(1)p} = \frac{6(0.38 \times 2.992)}{\pi(3.4)(1)(1.6)} = 0.39916\ MPa$$

$$\sigma_x = 0.39916\ MPa \qquad \tau_{xy} = 0$$
$$\sigma_y = -0.0804\ MPa \qquad \tau_{yz} = 0.316\ MPa$$
$$\sigma_z = 0 \qquad \tau_{zx} = 0$$

the von Mises stress is

$$\sigma' = \frac{1}{\sqrt{2}}\left[(\sigma_x - \sigma_y)^2 + (\sigma_y - \sigma_z)^2 + (\sigma_z - \sigma_x)^2 + 6(\tau_{xy}^2 + \tau_{yz}^2 + \tau_{zx}^2)\right]^{\frac{1}{2}}$$

$$= \frac{1}{\sqrt{2}}[(0.39916 - (-0.0804))^2 + ((-0.0804) - 0)^2 + (0 - 0.39916)^2 + 6(0 + 0.316^2 + 0)]^{\frac{1}{2}} = 0.705\ MPa$$

The principal stresses

$$\sigma_2, \sigma_3 = \frac{\sigma_y}{2} \pm \sqrt{(\frac{\sigma_y}{2})^2 + \tau_{yz}^2}$$

$$= \frac{-0.0804}{2} \pm \sqrt{(\frac{-0.0804}{2})^2 + 0.316^2} = 0.278, -0.359\ MPa$$

Ordering the principal stresses gives $\sigma_1, \sigma_2, \sigma_3 = 0.39916, 0.278, -0.359\ MPa$

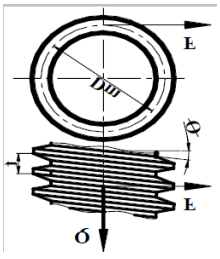
$$\sigma' = [\frac{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2}{2}]^{1/2}$$

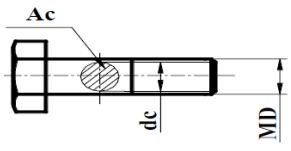
$$= [\frac{(0.39916 - 0.278)^2 + (0.278 - (-0.359))^2 + ((-0.359) - 0.39916)^2}{2}]^{1/2} = 0.705\ MPa$$

The maximum shear stress

$$\tau_{max} = \frac{\sigma_1 - \sigma_3}{2} = \frac{0.39916 - (-0.359)}{2} = 0.37908\ MPa$$

2.4 Bolts selection





Torque to overcome friction between threads (Nut and Bolt)

$Q = 0.2 \times 9.81 = 1.962\ N$

$D_m = 5.5 \times 10^{-3}\ m$

$\rho = tan^{-1}(f) = tan^{-1}(0.13) = 7.41°$

$\emptyset = 3.04°$

$$T_{ft.thr} = Q\tan(\emptyset + \rho) + \frac{D_m}{2} = (1.9620\tan(7.41 + 3.04) + \frac{5.5 \times 10^{-3}}{2} = 0.365\ N.m$$

Torque to overcome friction between Nut and tightening surface:

$f_s = 0.9$

$D = 5.5 \times 10^{-3}\ m$

$$T_{fr.s} = Qf_s\frac{(1.1D + \frac{1.5D}{2})}{2} = (1.962)(0.9)\frac{\left(1.1(5.5 \times 10^{-3}) + \frac{1.5(5.5 \times 10^{-3})}{2}\right)}{2} = 8.98 \times 10^{-3}\ N.m$$

$$T_{total} = T_{ft.thr} + T_{fr.s} = 0.374\ N.m$$

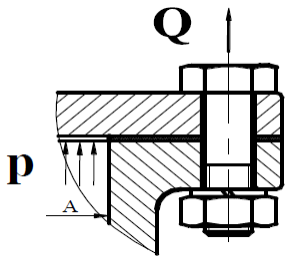
Efficiency of thread

$$\varphi = \frac{\tan(\emptyset)}{\tan(\emptyset + \rho)} = 0.29$$

Subject to an External Force

$$\sigma_t = \frac{2.3\ Q}{\left(\frac{\pi}{4}\right).{d_c}^2} = 0.638\ MPa$$

Lateral force = 30.4 N



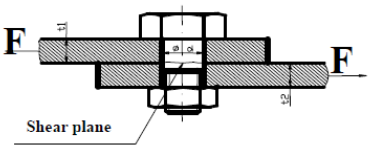
$$A_{share} = \frac{\pi d^2}{4} = 7.06 \times 10^{-3}\ m^2$$

$$\tau_{sh} = \frac{30.4}{4(7.06 \times 10^{-6})} = 1.076\ MPa$$

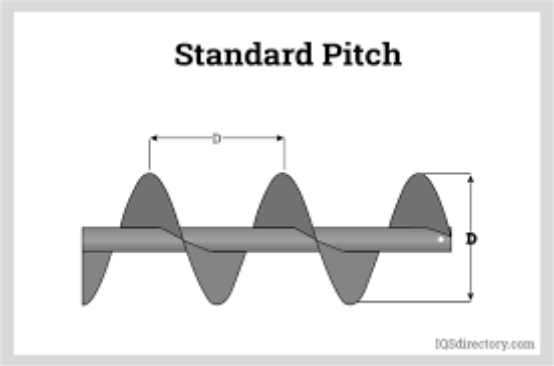
$$\leq |\tau_{sh}|_{Bolt} = 200\ MPa$$

$$A_{crush} = 1.2 \times 10^{-5}\ m^2$$

$$\sigma_{crush} = 634\ KPa \leq |\sigma_{crush}| = 260\ MPa$$

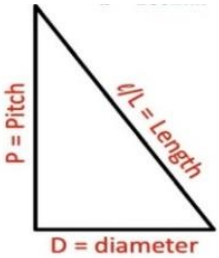


2.5 Design of drill

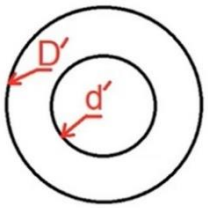


Type of pitch is standard.
Type of flight is standard.

The auger has speed range 50-150 rpm
Pitch = 10 mm
d = 10 mm
D = 30 mm



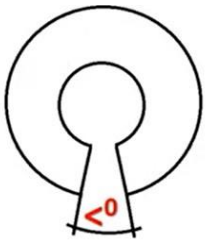
$$l = \sqrt{(d \times \pi)^2 + (p)^2}$$
$$= \sqrt{(10 \times \pi)^2 + (10)^2} = 32.97\ mm$$
$$L = \sqrt{(D \times \pi)^2 + (p)^2}$$
$$= \sqrt{(30 \times \pi)^2 + (10)^2} = 94.77\ mm$$
$$d' = \frac{D - d}{\left(\frac{L}{l}\right) - 1} = \frac{30 - 10}{\left(\frac{94.77}{32.97}\right) - 1} = 10.67\ mm$$



$$D' = (D - d) + d' = (30 - 10) + 10.67 = 30.67\ mm$$

Cutting angle

$$\theta = \frac{L}{\frac{\pi D}{360}} = \frac{94.77}{\frac{\pi \times 30.67}{360}} = 354°$$



θ =360 – 354 =5.9°

Screw discharge (capacity)

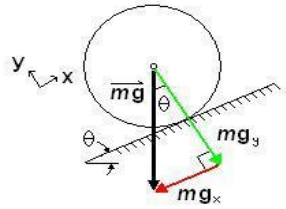
$$c_v = \frac{\pi}{4}(D^2 - d^2) \times p \times \frac{rpm}{60} \times k \times 3600 \times \rho$$
$$= \frac{\pi}{4}(0.03^2 - 0.01^2) \times 0.01 \times \frac{100}{60} \times 0.8 \times 3600 \times 1680 = 50\ kg/h$$

2.6 Force analysis

Mass of robot without load = 12.4 kg
The tank can contain 1.5 L
The bate can contain 0.1 kg
Payload = 8 kg
Total mass = 15 kg
Fraction = μ * FN = μ * mg = 0.8 * (15*9.81) = 118 N
The force on each wheel = 35 N
We considered that acceleration will be 0.2 m/s^2according to the time that we need
We consider that RPM of each motor will be 125 so :
w = $\frac{RPM}{60}$ *** 2π =** 13.1 rad/s
Velocity = R ***w** = 0.065 *13.1 = 0.9 m/s
So, the acceleration (Δt) = $\frac{a}{\Delta v}$ =0.2/0.9 =0.2 s

2.7 Motor Sizing

STEP 1 REQUIREMENTS
To calculate the required torque, power, current, and battery pack for a wheeled mobile robot: the idea of vectors Power; Current and Voltage; 2D Force Balance.
A wheeled robot's motors must create enough torque to overcome any irregularities in the surface or wheels, as well as friction in the motor itself, in order to roll on a horizontal surface. As a result, a robot (small or large) does not require a lot of torque to move horizontally [20°].
A huge robot will obviously have more friction and resistance than a
tiny one. In order to a robot to roll up an incline at a constant velocity (no acceleration or deceleration) it must produce enough torque to "counter act" the effect of gravity, which would otherwise cause it to roll down the incline. On an inclined surface (at an angle theta) however, only one component of its
weight (mgx parallel to the surface) causes the robot to move downwards. The other component mgy is balanced by the normal force the surface exerts on the wheels.



STEP 2 INCLINE

Friction between the wheel and the surface is required for the robot not to slide down the hill. A hefty truck's engine may be capable of producing 250 horsepower and tremendous torque, but we've all seen enormous trucks spinning their wheels as they fall backwards on an ice street (in person or on video). The torque is "produced" by friction (f).

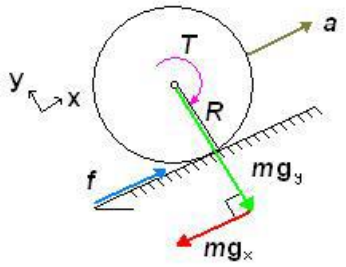
mgx = mg * sin (ɵ) mgy = mg * cos (ɵ)

STEP 3 TORQUE

The torque (**T**) required is:

$$\boldsymbol{T = F * R}$$

To select the proper motor, we must consider the "worst case scenario", where the robot is not only on an incline, but accelerating up it.



Note now that all forces (**F**) are along the x and y axes. We balance the forces in the x-direction:

$\sum F_x = M * a = F - m * g_x$

Inserting the equation for torque above, and the equation for mgx, we obtain:

$$M * a = \frac{T}{R} - M * g * \sin(\theta)$$

Rearrange the equation to isolate T:

$$\boldsymbol{T = R * M * (a + g * sin(\theta))}$$

This torque value represents the total torque required to accelerate the robot up an incline. However, this value must be divided by the total number (**N**) of drive wheels to obtain the torque needed for each drive motor. Note that we do not consider the total number of passive wheels as they have no effect on the torque required to move the object aside from adding weight.

$$T = \frac{(a + g * \boldsymbol{sin}(\theta)) * M * R}{N}$$

The final point to consider is the efficiency (e) in the motor, gearing and wheel (slip).

$$T = \left(\frac{100}{e}\right) * \frac{(a + g * \boldsymbol{sin}(\theta)) * M * R}{N}$$

STEP 4 POWER
Total power (**P**) per motor can be calculated using the following relation:

$$P = \mathrm{T} * \omega$$

STEP 5 CURRENT
T is known from above and the angular velocity (w) is specified by the builder. It is best to select the maximum angular velocity to be able to find the corresponding maximum power. Knowing the maximum power and the supply voltage (V) which the builder chooses, we can find an idea of the maximum current (I) requirements:

$$P = I * V$$

STEP 6 CAPACITY
Finally, the capacity (c) of battery pack required can be estimated using the equation:

$$c = I * t$$

You may wonder why such a large value is needed. This is because when choosing a battery pack, the rated amp hours are not an accurate indicate of the maximum current the pack can produce for extended periods of time. Also, the total charge is rarely retained over time. This way you will ensure the battery pack you select will be capable of producing the current your motors require, for the time you require and with the inefficiencies inherent in recharging battery packs. Note: This is the battery required PER MOTOR. To obtain a total battery pack required for the robot, multiply this value by the number of drive motors.

2.8 Analytical Motor Sizing Calculation

First, we will be going to calculate the torque of the motor needed using the equation below.

$$T = \left(\frac{100}{e}\right) \times \frac{(a + g\sin(\theta))M\,R}{N}$$

Where: a = Acceleration, g = Acceleration due to gravity, e = Efficiency, M = Mass,
R = Radius of the wheel, N = Number of motors, θ = angle of inclination

The value for efficiency here represents the total efficiency of the system and can be estimated as follows:
- Battery -> Motor Controller: 93% efficient
- Motor Controller -> Motors: 95% efficient
- Brushed DC motor: 91% efficient

Each step reduces the total efficiency by their respective percentage, so the total efficiency in this example would be:
93% x 95% x 91% = 80%
0.93 x 0.95 x 0.91 =0.80

By substituting the values in the above equation:
Efficiency (e) = 80%
Acceleration (a) = 0.2 $m/\ s^2$
Gravity (g) = 9.81m/s^2
Degree of incline (θ) = 20°
Mass (M) = 15 kg
Radius of drive wheel (R) = 0.065 m
Number of drive motors (N) = 4

T = $\frac{100}{80} * \frac{(0.2+9.81\sin(20))*15*0.065}{4}$ = 1 Nm

Total power (**P**) per motor can be calculated using the following relation:
P = T * **w = 1 ***41.89 **=** 41.89 **W**

maximum current (**I**) requirements:
P= I* V

Supply voltage = 12v
I = P/V =41.89 /12 =3.5 A

The capacity (c) of battery pack required can be estimated using the equation:
$c = I * t * N$

Desired operating time (t) = 1 hour
C = 3.5 * 1 * 4 = 14 Ah

Input

Total mass:

15

Kg

Number of drive motors:

4

[#]

Radius of drive wheel:

0.065

m

Robot Velocity:

0.9

m/s

Maximum incline:

20

[deg]

Supply voltage:

12

M

Desired acceleration:

0.2

m/s2

Desired operating time:

1

hs

Total efficiency:

80

[%]

Output (for each drive motor)

Angular Velocity:

13.846

rad/s

Torque*:

1.0832

Nm

Total Power:

14.999

W

Maximum current:

1.2499

[A]

Battery Pack

4.9995

[Ah]

2.9 For auger motors

We need pressure on the auger = half weight of the robot
The weight of the robot = 15*9.81=147.15 N
So, we need 74 N pressure to do the digging and it can't flip the robot.
Diameter of screw(r) = 5 mm

The torque for the First motor that move drill up and down = F*r =74*0.0025 =0.184 Nm
The time that takes to reach the maximum depth = 10s
We need to calculate the work it takes to reach the maximum depth.

maximum depth (d) =0.09 m
W= F*d = 74 *0.09 =6.66 J
Power = W/t = 6.66/10 = 0.666 W
HP = $\frac{0.666}{746} = 8.92x10^{-4}$ hp
RPM = $\frac{HP \times 9550}{T} = \frac{(8.92x10^{-4})(9550)}{0.145} = 59$ rpm

$$\omega = \frac{RPM}{60} \times 2\pi = \frac{59}{60} \times 2\pi = 6.18\, rev/s$$

V = ω * r = 6.18 * 0.0025 =0.015 m/s

2.10 Mechanical Design

2.10.1 Design Assembly 3D Model

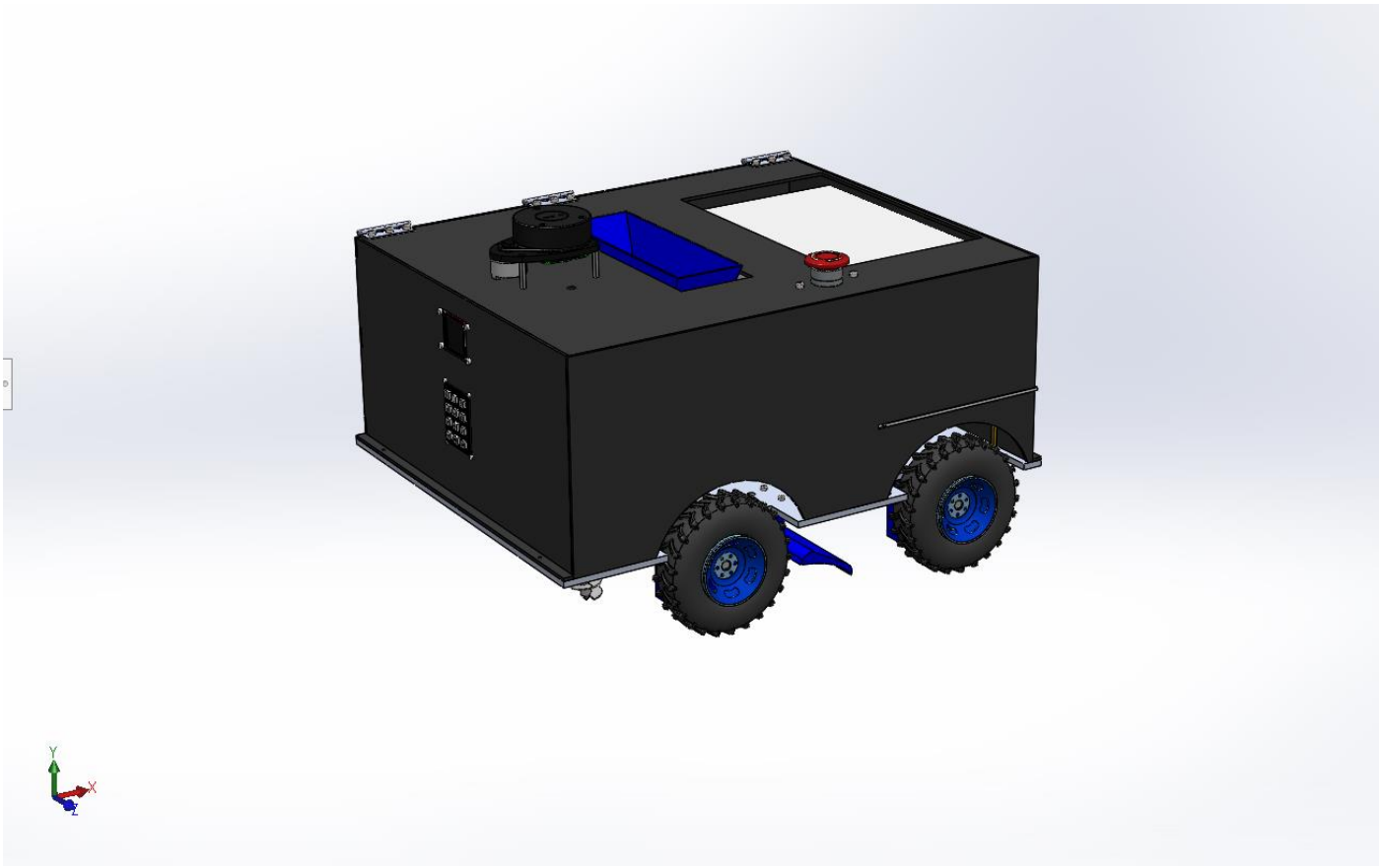


Figure 2.1: 3D Model

2.10.2 Design Drawings and Assembly

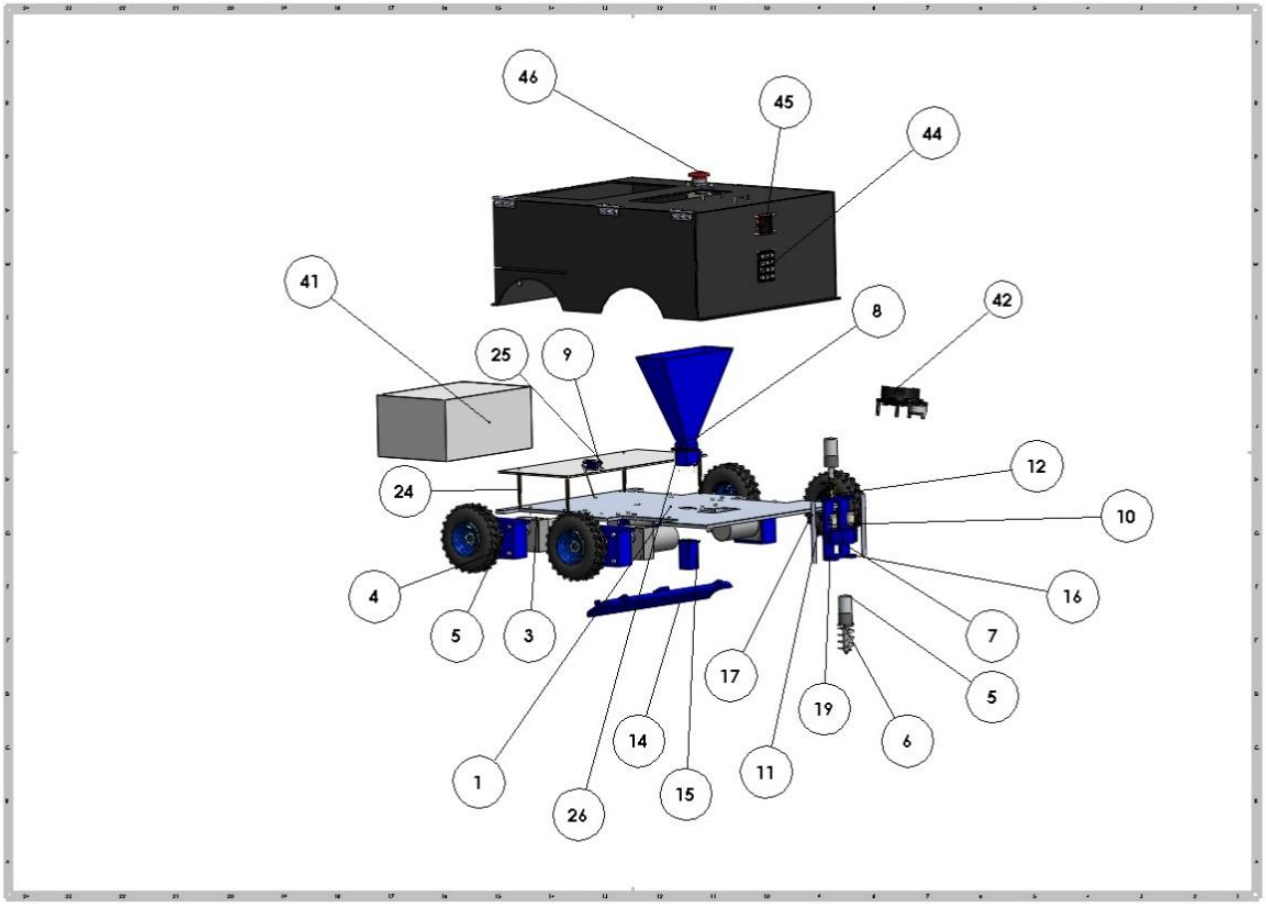
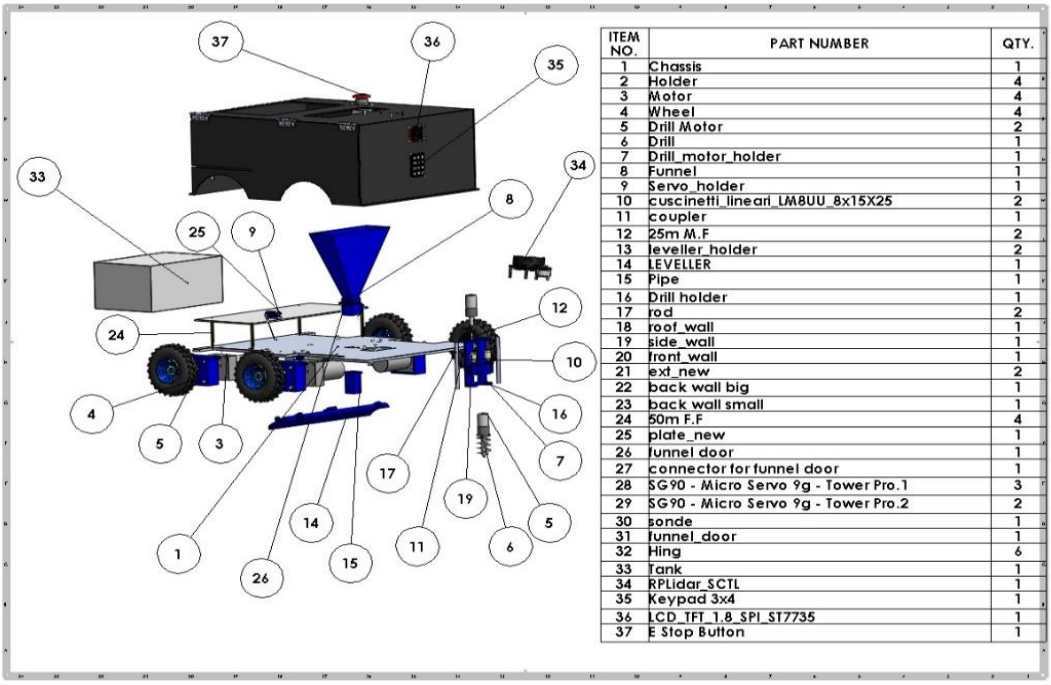


Figure 2.2: Exploded view



ITEM NO.	PART NUMBER	QTY.
1	Chassis	1
2	Holder	4
3	Motor	4
4	Wheel	4
5	Drill Motor	2
6	Drill	1
7	Drill_motor_holder	1
8	Funnel	1
9	Servo_holder	1
10	cuscinetti_lineari_LM8UU_8x15X25	2
11	coupler	1
12	25m M.F	2
13	leveller_holder	2
14	LEVELLER	1
15	Pipe	1
16	Drill holder	1
17	rod	2
18	roof_wall	1
19	side_wall	1
20	front_wall	1
21	ext_new	2
22	back wall big	1
23	back wall small	1
24	50m F.F	4
25	plate_new	1
26	funnel door	1
27	connector for funnel door	1
28	SG90 - Micro Servo 9g - Tower Pro.1	3
29	SG90 - Micro Servo 9g - Tower Pro.2	2
30	sonde	1
31	funnel_door	1
32	Hing	6
33	Tank	1
34	RPLidar_SCTL	1
35	Keypad 3x4	1
36	LCD_TFT_1.8_SPI_ST7735	1
37	E Stop Button	1

Figure 2.3 Exploded view with table of parts

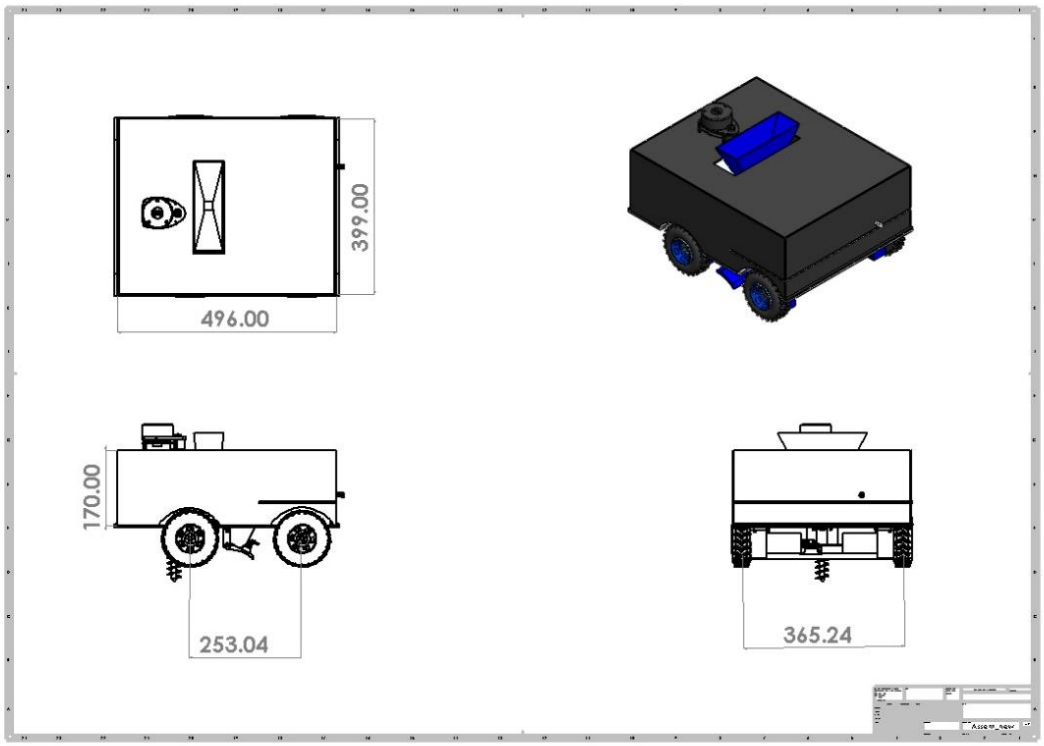


Figure 2.4: Assembly dimensions

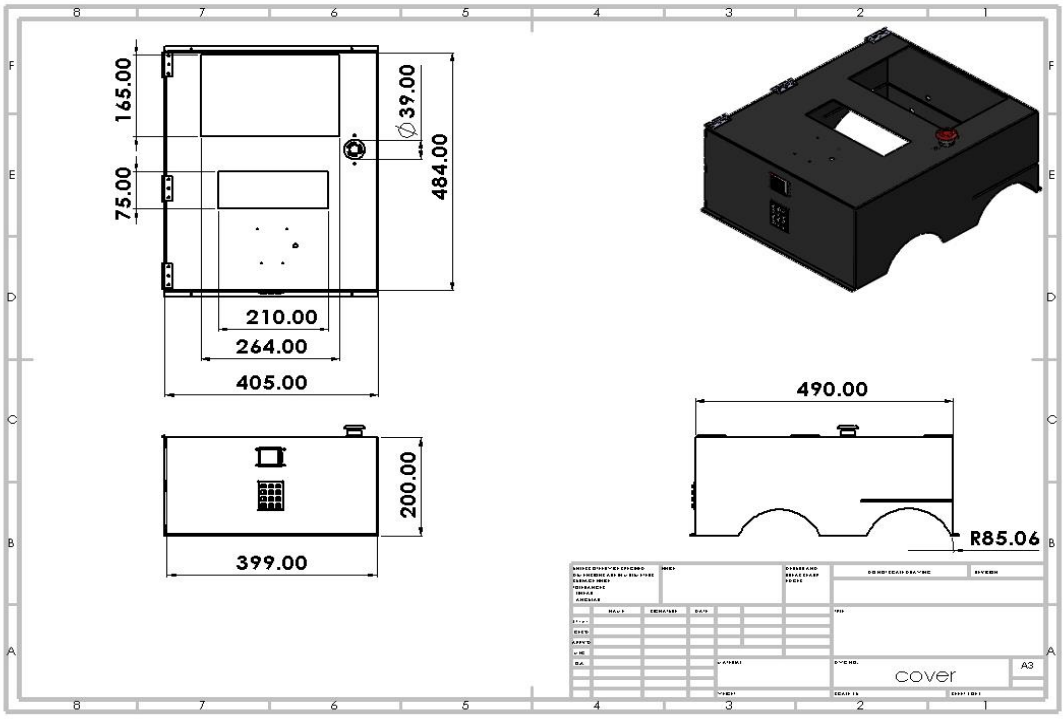


Figure 2.5: Cover dimensions

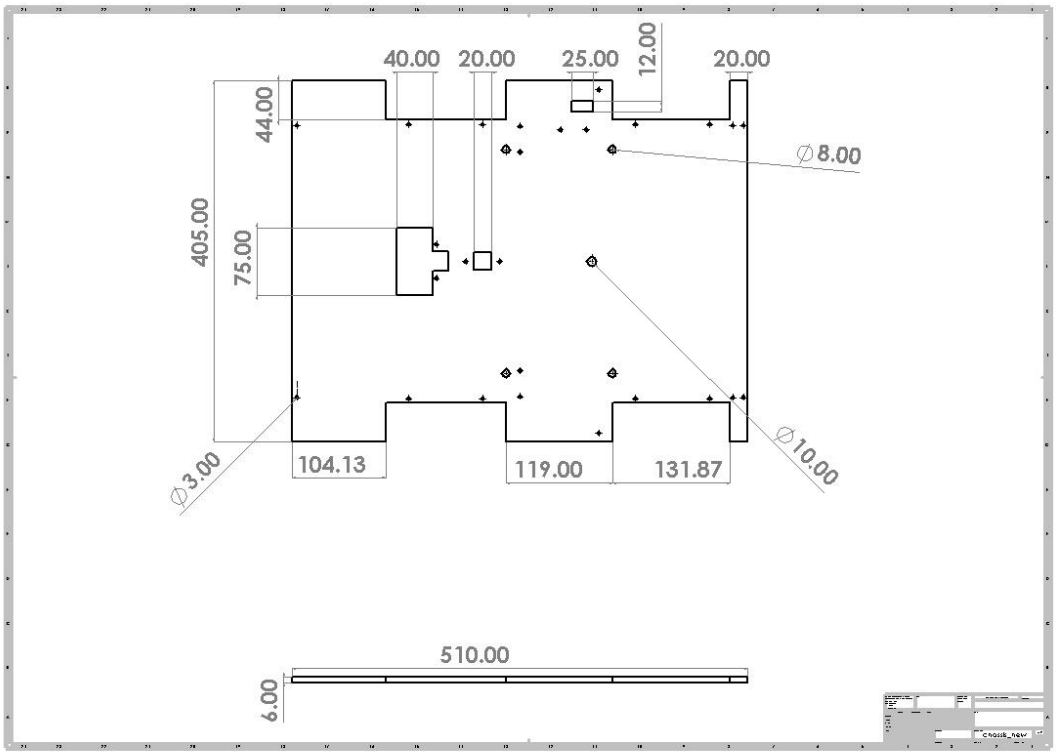


Figure 2.6: Chassis dimensions

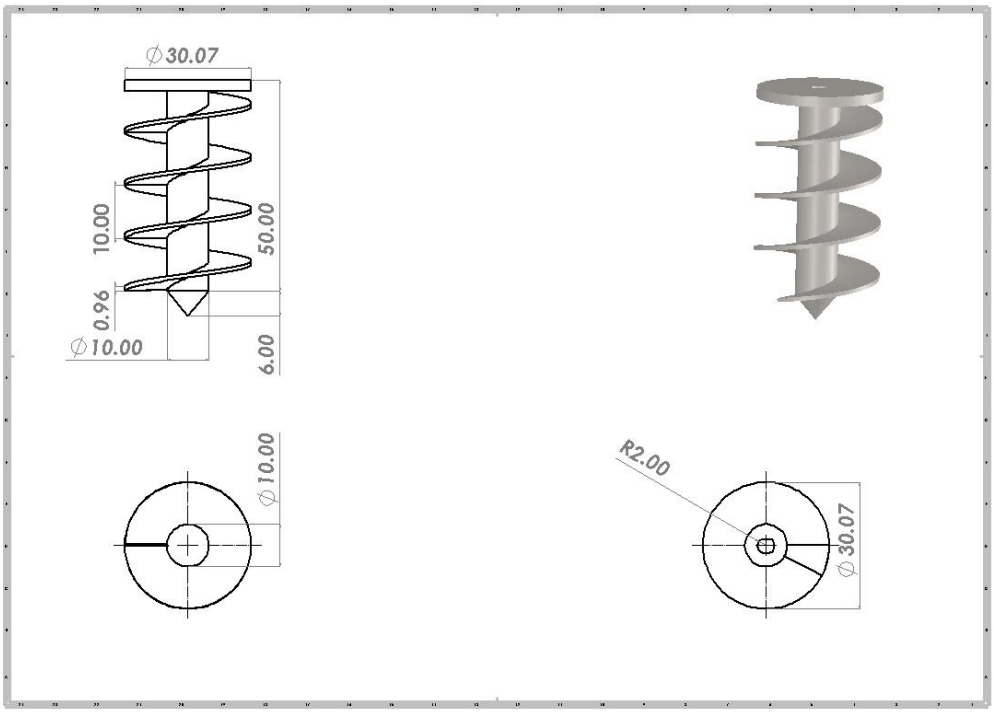


figure 2.7: Drill dimensions

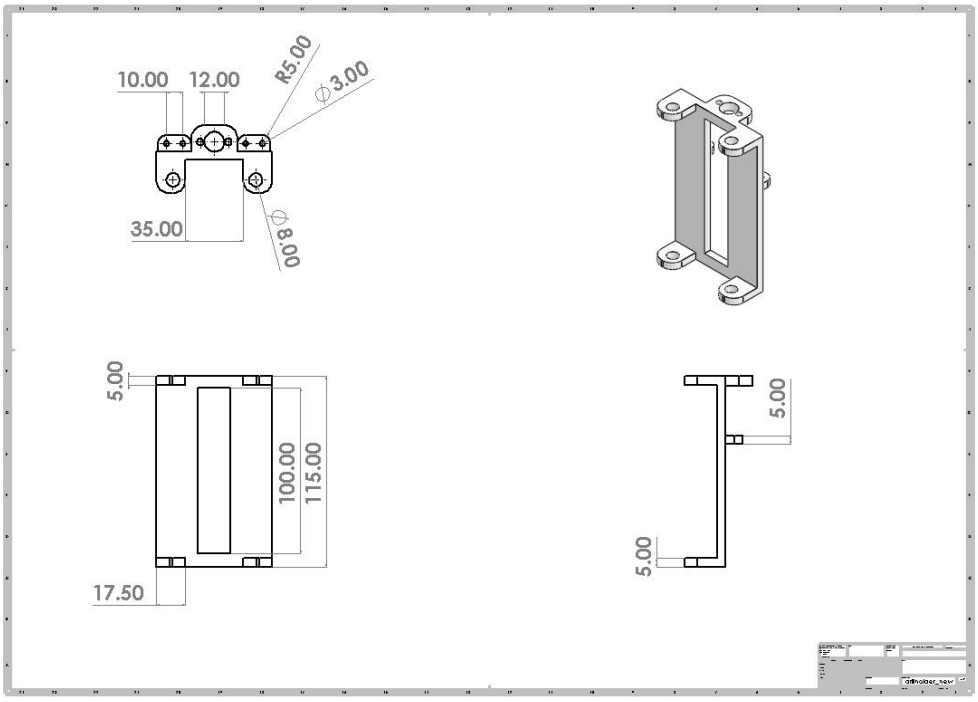


Figure 2.8: Drill Holder dimensions

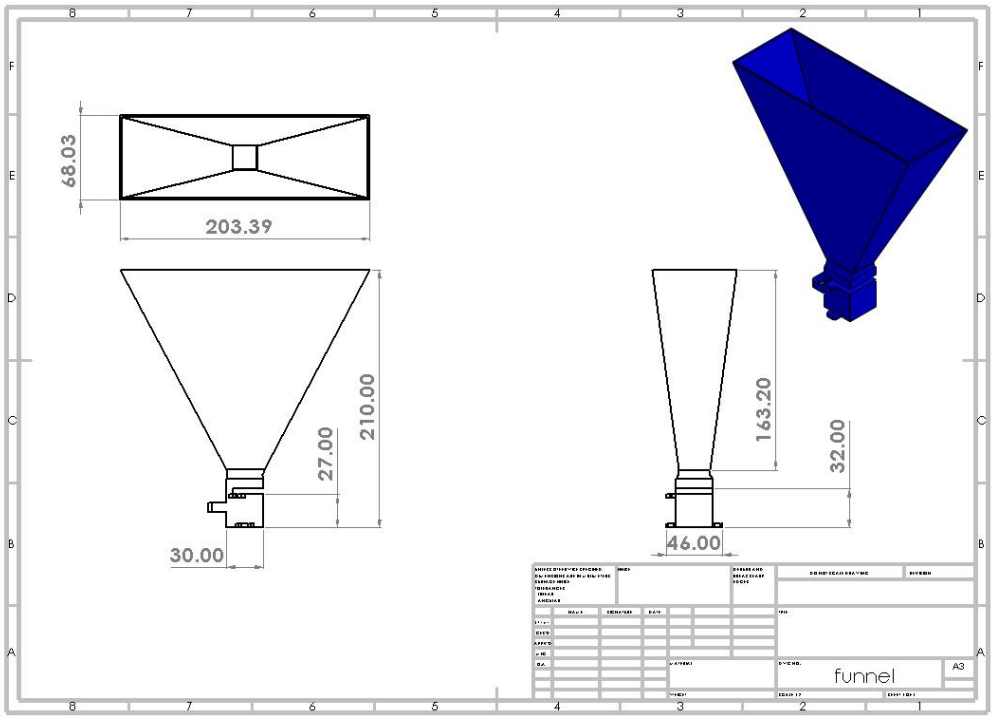


Figure 2.9: Funnel dimensions

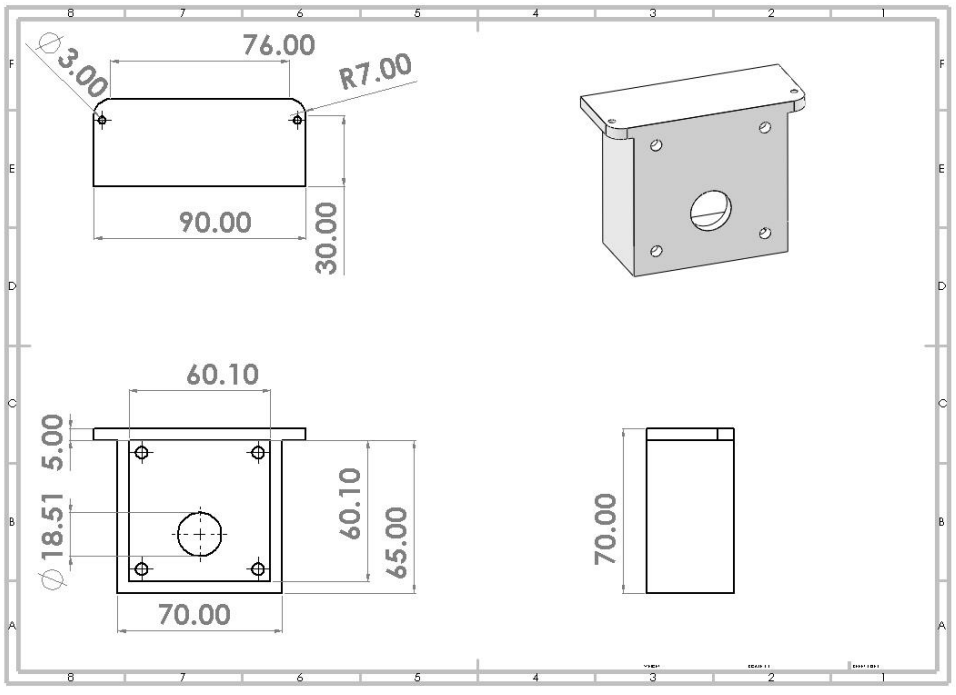


Figure 2.10: Motor Holder dimensions

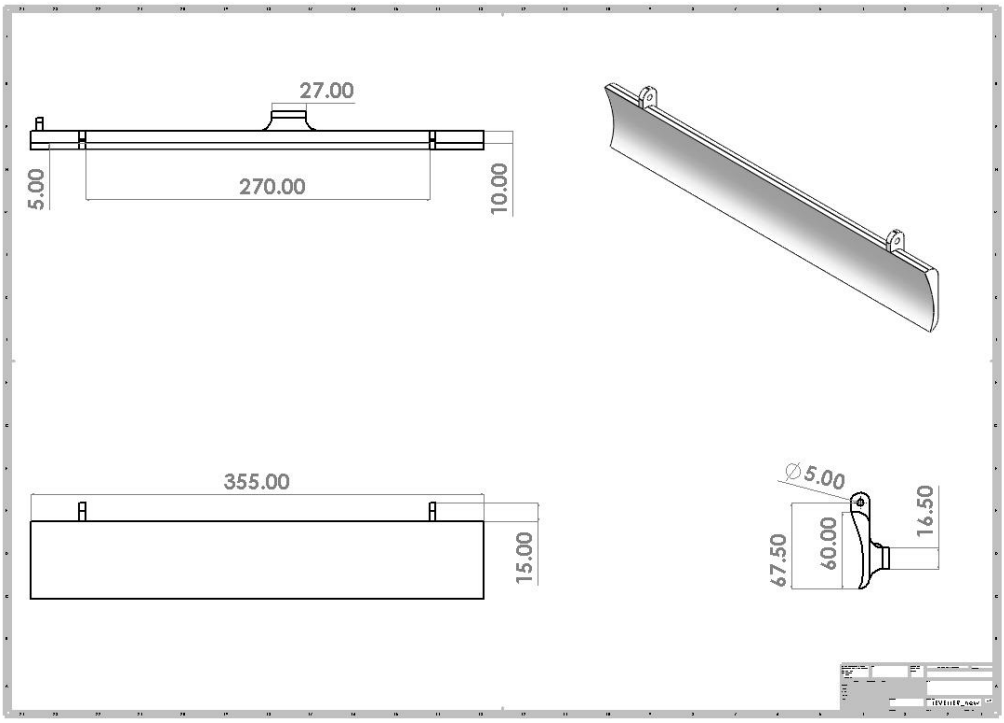


Figure 2.11: Leveler dimensions

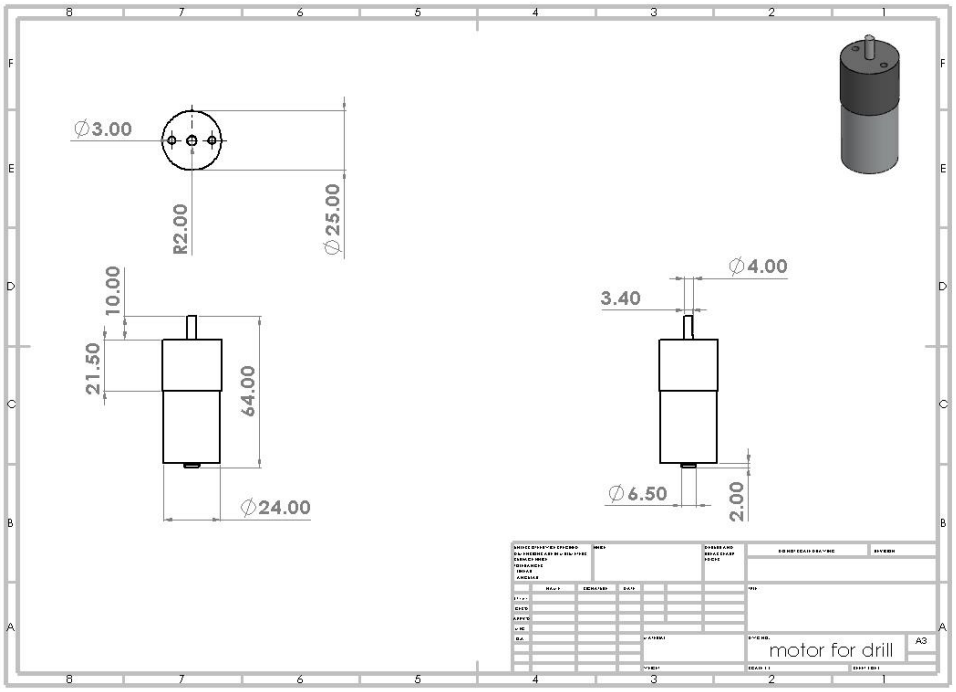


Figure 2.12: Motor for Drill dimensions

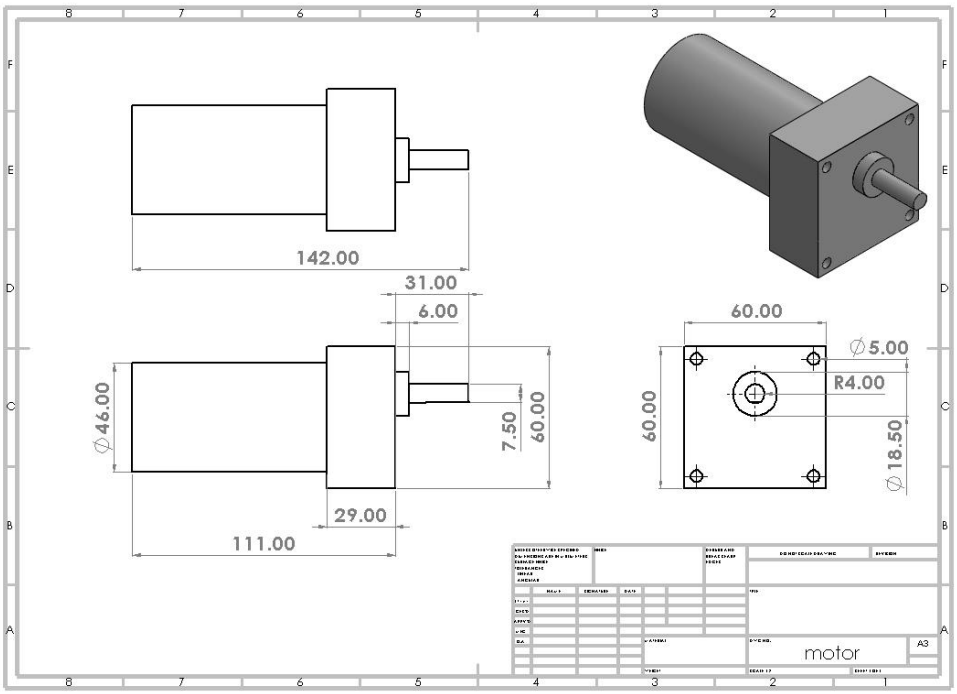


Figure 2.13: Motor dimensions

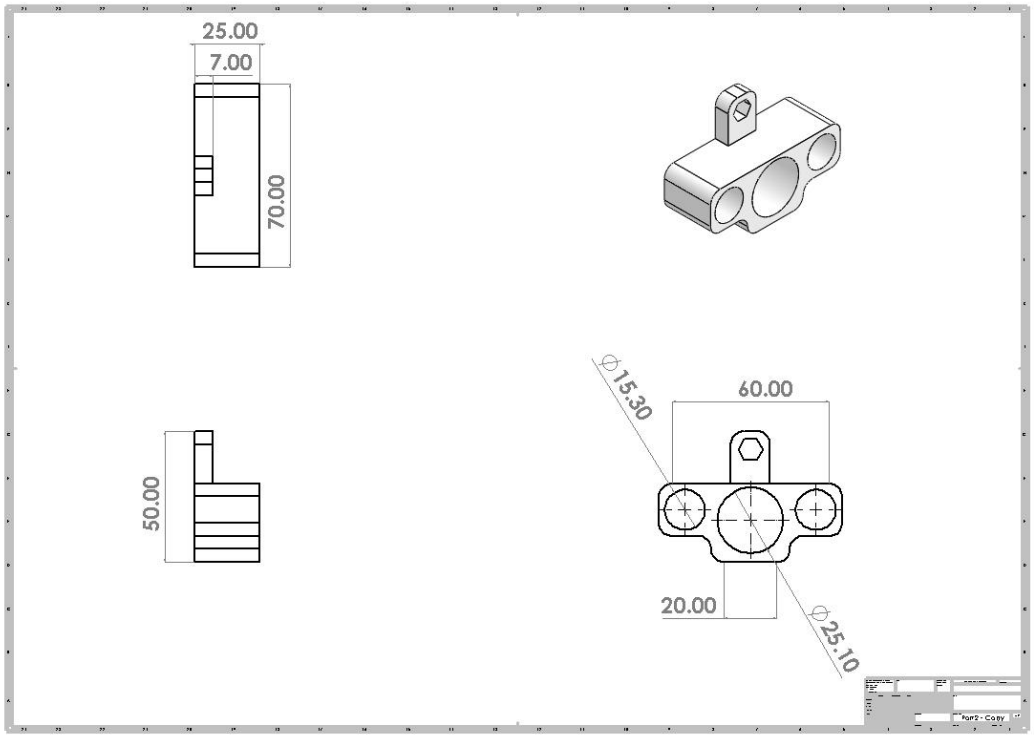


Figure 2.14: Drill Holder dimensions

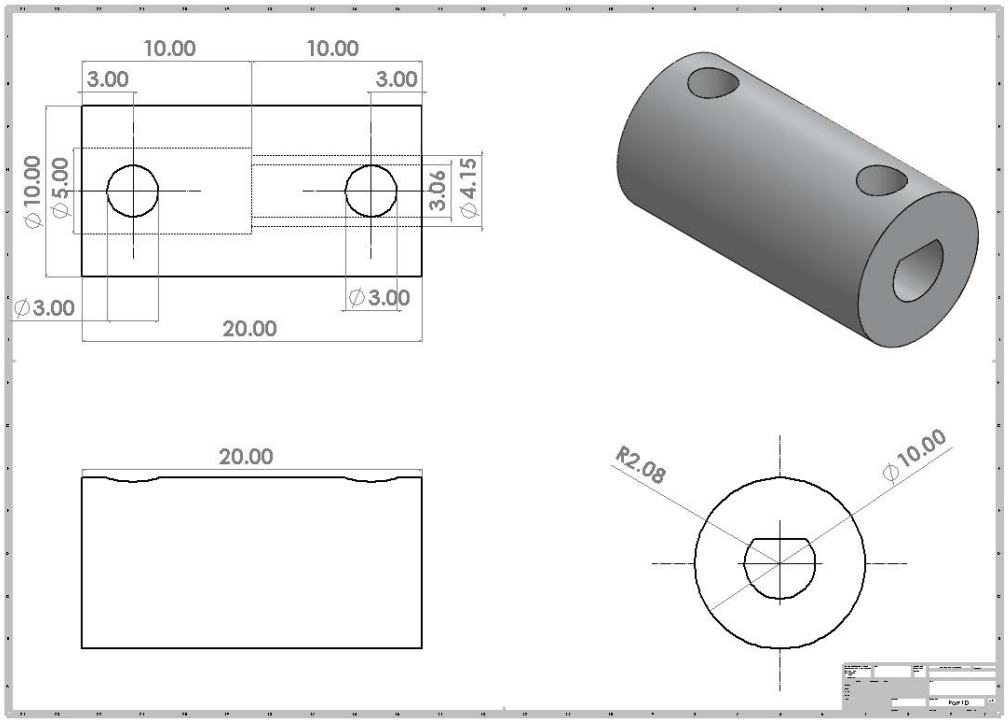


Figure 2.15: Drill coupling dimensions

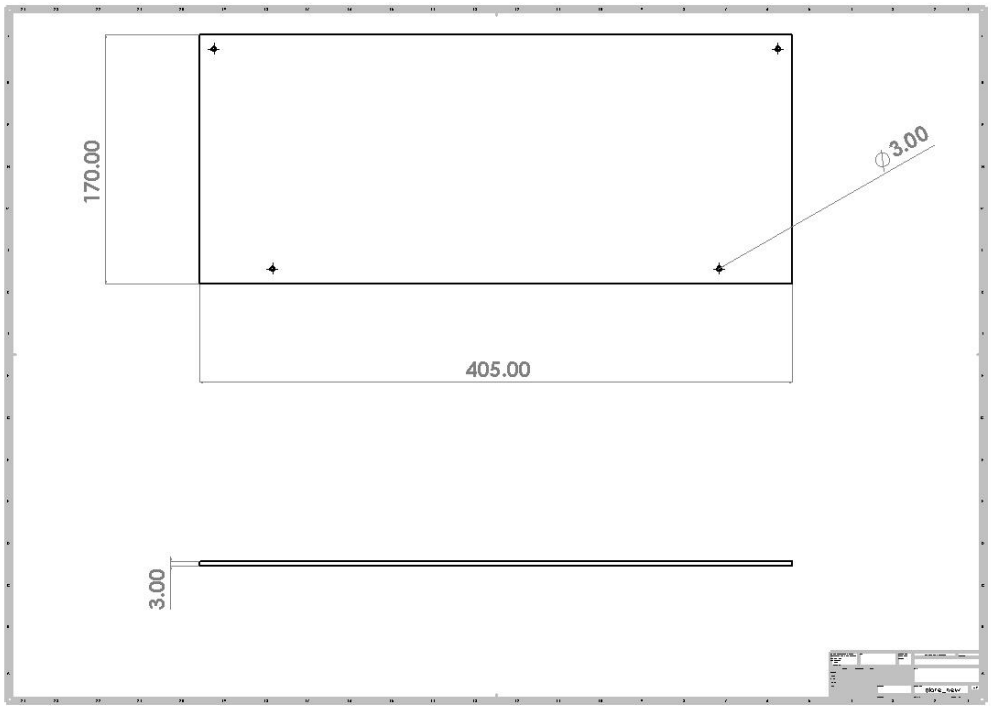


Figure 2.16: Plate dimensions

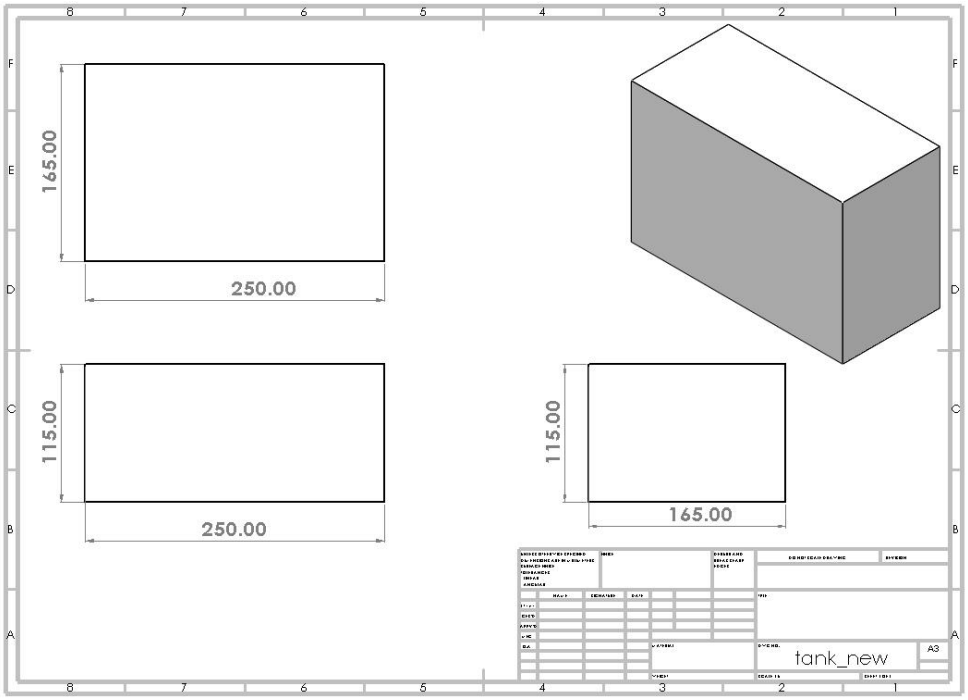


Figure 2.17: Tank dimensions

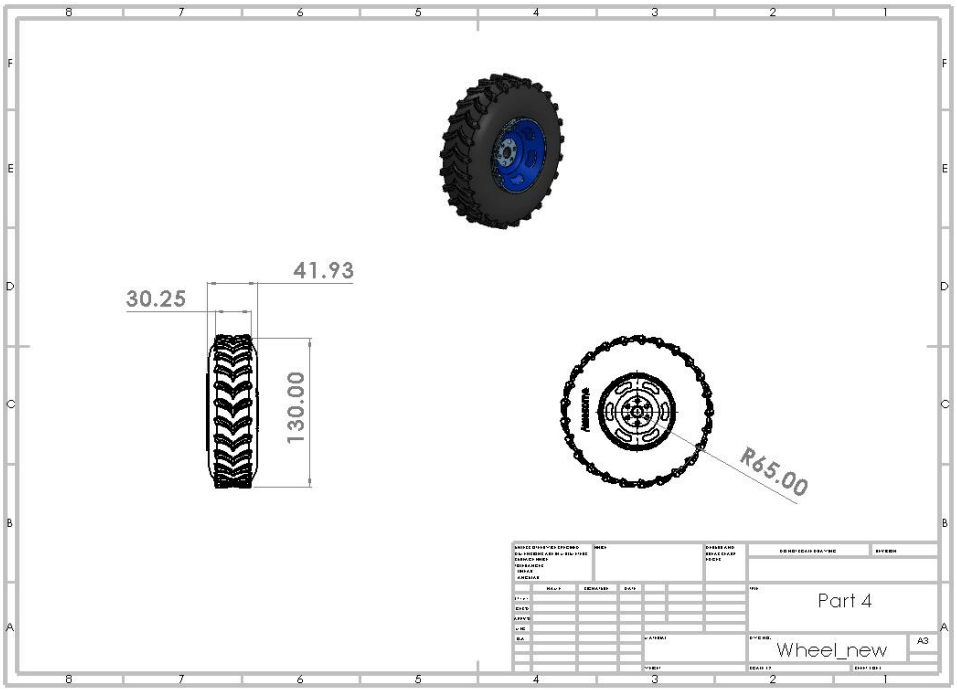


Figure 2.18: Wheel dimensions

Resultant Forces

Reaction forces

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N	0.0061617	139.994	-0.00623113	139.994

Reaction Moments

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N.m	0	0	0	0

Free body forces

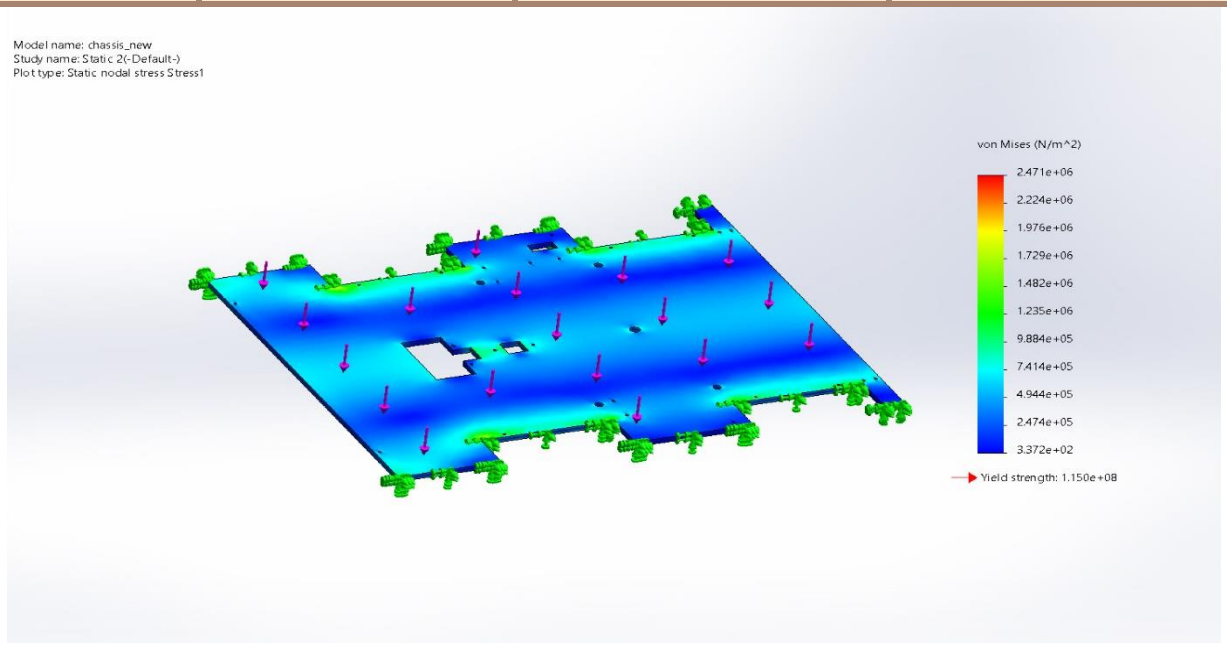
Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N	0.0717881	0.116156	-0.0251801	0.138851

Free body moments

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N.m	0	0	0	1e-33

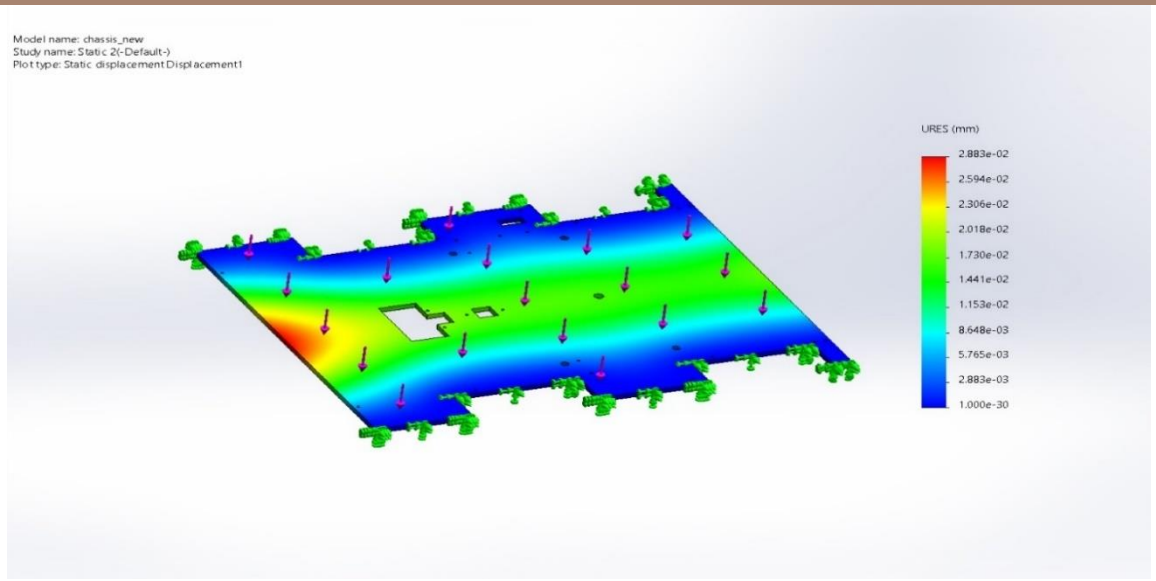
Study Results

Name	Type	Min	Max
Stress1	VON: von Mises Stress	3.372e+02N/m^2 Node: 14763	2.471e+06N/m^2 Node: 30686



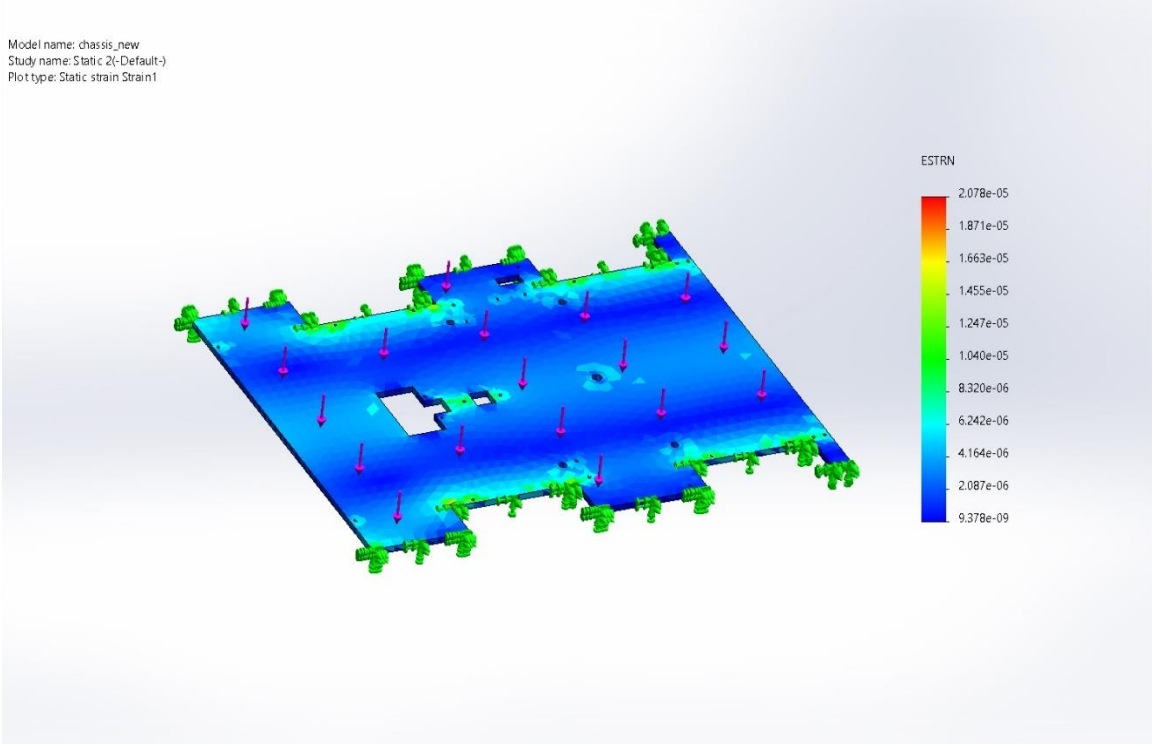
Chassis Stress

Name	Type	Min	Max
Displacement1	URES: Resultant Displacement	0.000e+00mm Node: 588	2.883e-02mm Node: 27301



Chassis Displacement

Name	Type	Min	Max
Strain1	ESTRN: Equivalent Strain	9.378e-09 Element: 4856	2.078e-05 Element: 1311



Chassis Strain

Resultant Forces

Reaction forces

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N	1	1	8.77772e-08	1.41421

Reaction Moments

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N.m	0	0	0	1e-33

Free body forces

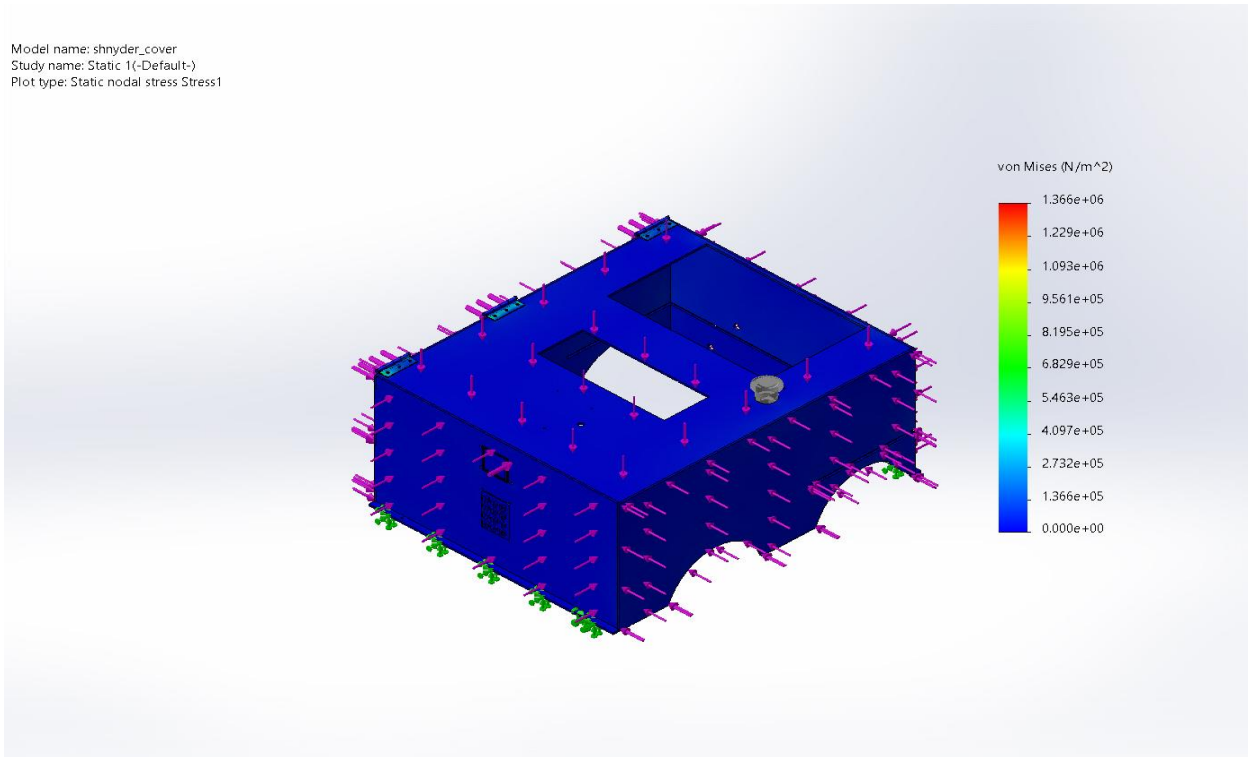
Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N	-1.32946e-06	-5.15209e-05	-7.81729e-05	9.36332e-05

Free body moments

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N.m	0	0	0	1e-33

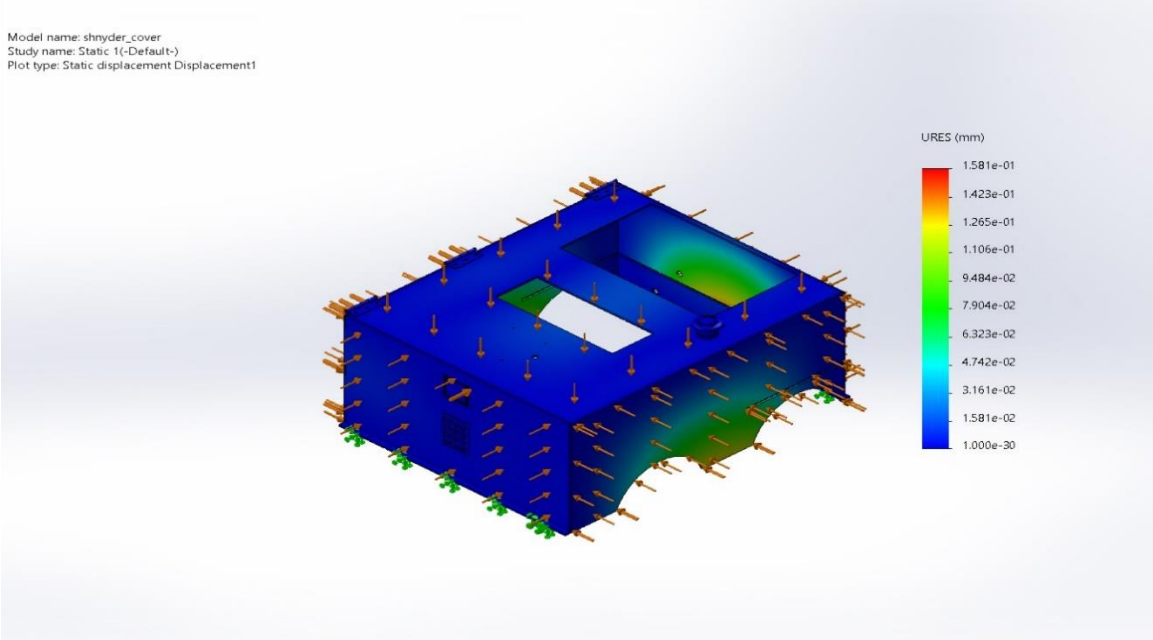
Study Results

Name	Type	Min	Max
Stress1	VON: von Mises Stress	0.000e+00N/m^2 Node: 1	1.366e+06N/m^2 Node: 156318



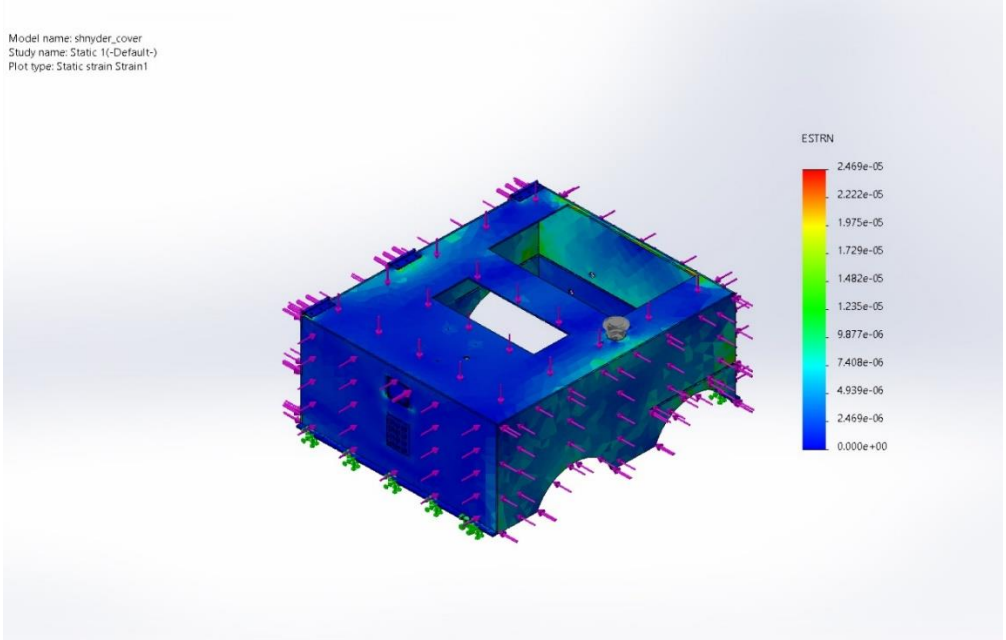
Cover Stress

Name	Type	Min	Max
Displacement1	URES: Resultant Displacement	0.000e+00mm Node: 1	1.581e-01mm Node: 248275



Cover Displacement

Name	Type	Min	Max
Strain1	ESTRN: Equivalent Strain	0.000e+00 Element: 1	2.469e-05 Element: 138086



Cover Strain

Resultant Forces

Reaction forces

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N	0.000556761	-0.00608707	45.6132	45.6132

Reaction Moments

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N.m	0	0	0	0

Free body forces

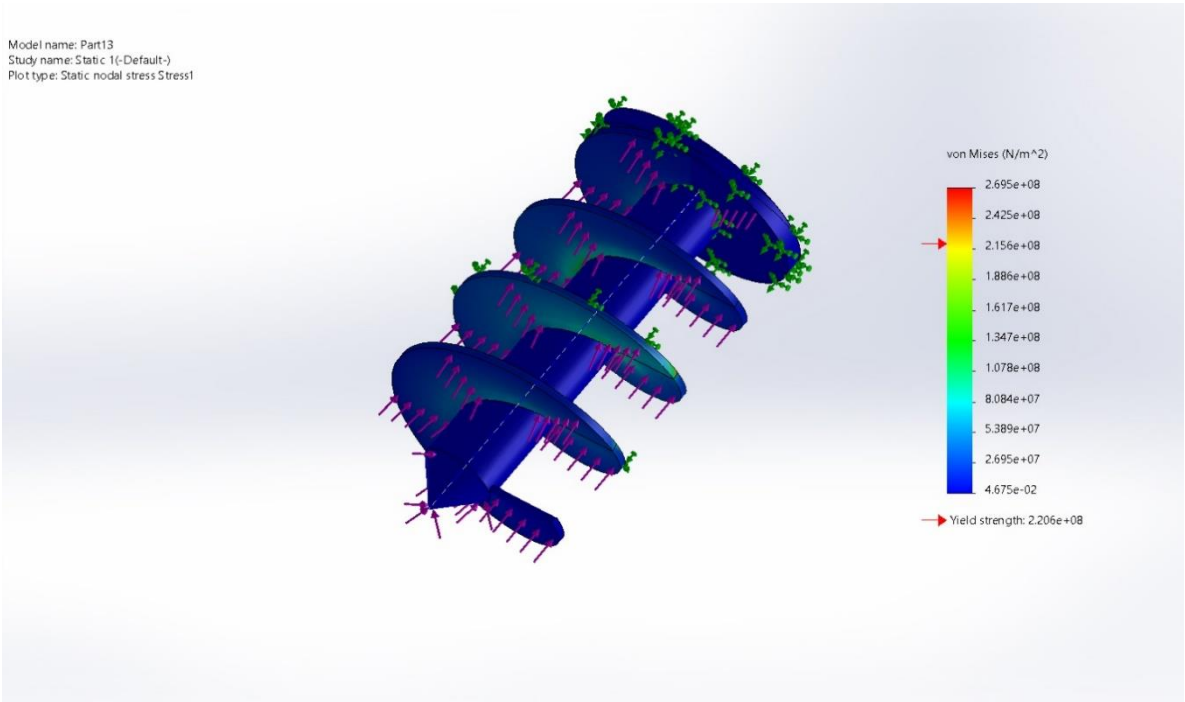
Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N	-0.095853	-0.0213076	0.0895516	0.132896

Free body moments

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N.m	0	0	0	1e-33

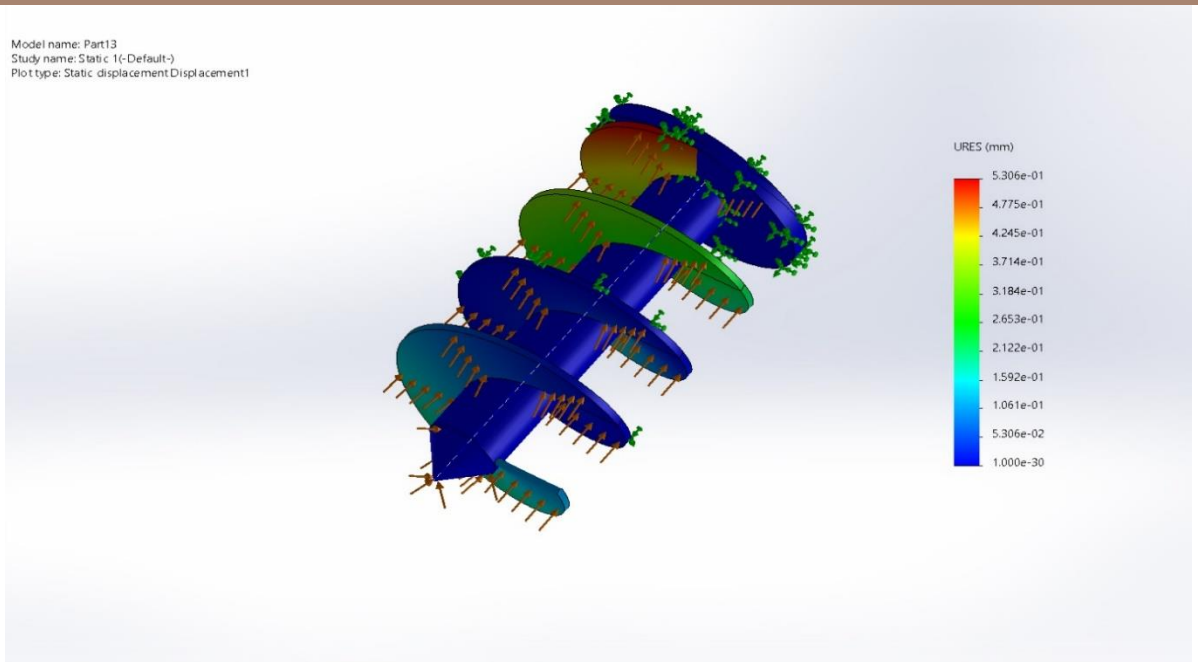
Study Results

Name	Type	Min	Max
Stress1	VON: von Mises Stress	4.675e-02N/m^2 Node: 18925	2.695e+08N/m^2 Node: 7924



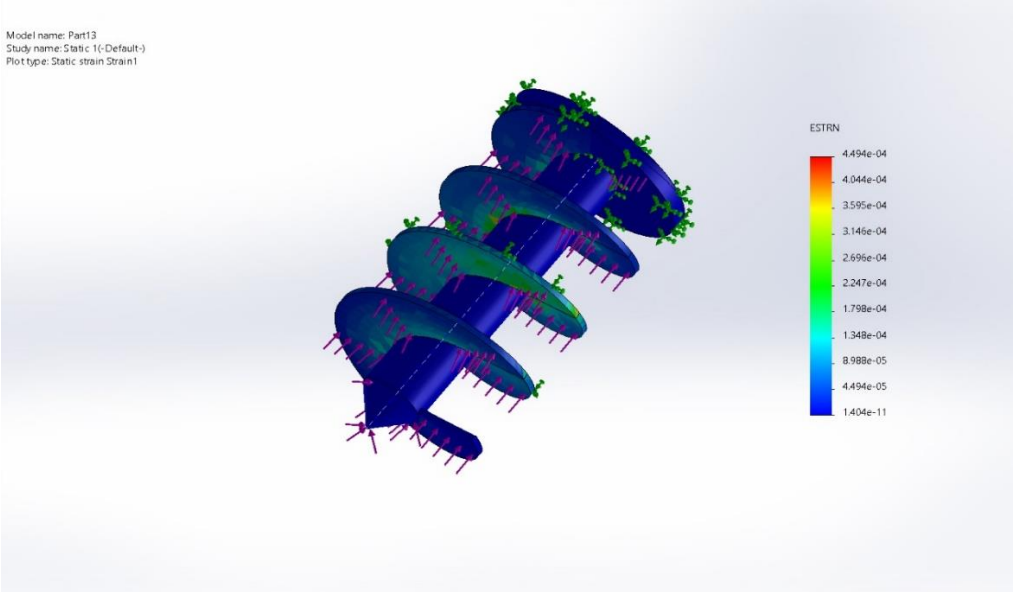
Drill Stress

Name	Type	Min	Max
Displacement1	URES: Resultant Displacement	0.000e+00mm Node: 7907	5.306e-01mm Node: 5010



Drill Displacement

Name	Type	Min	Max
Strain1	ESTRN: Equivalent Strain	1.404e-11 Element: 4916	4.494e-04 Element: 4618



Drill Strain

Resultant Forces

Reaction forces

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N	-0.010034	-4.82913	48.3137	48.5544

Reaction Moments

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N.m	0	0	0	0

Free body forces

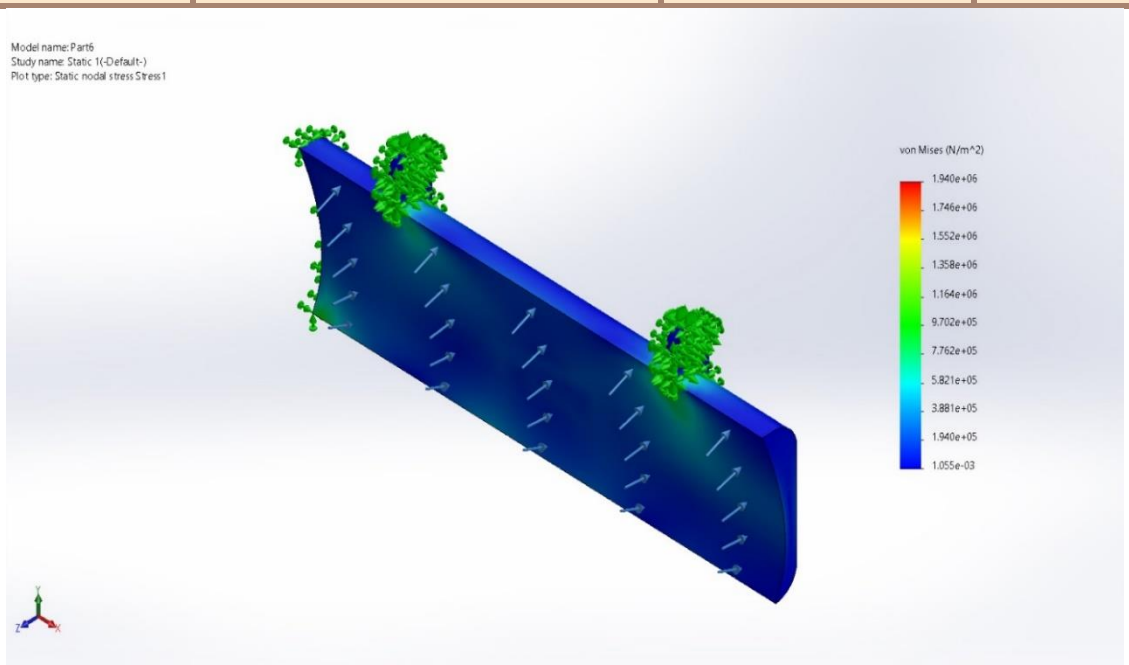
Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N	0.0101893	-0.0471217	0.0493479	0.068989

Free body moments

Selection set	Units	Sum X	Sum Y	Sum Z	Resultant
Entire Model	N.m	0	0	0	1e-33

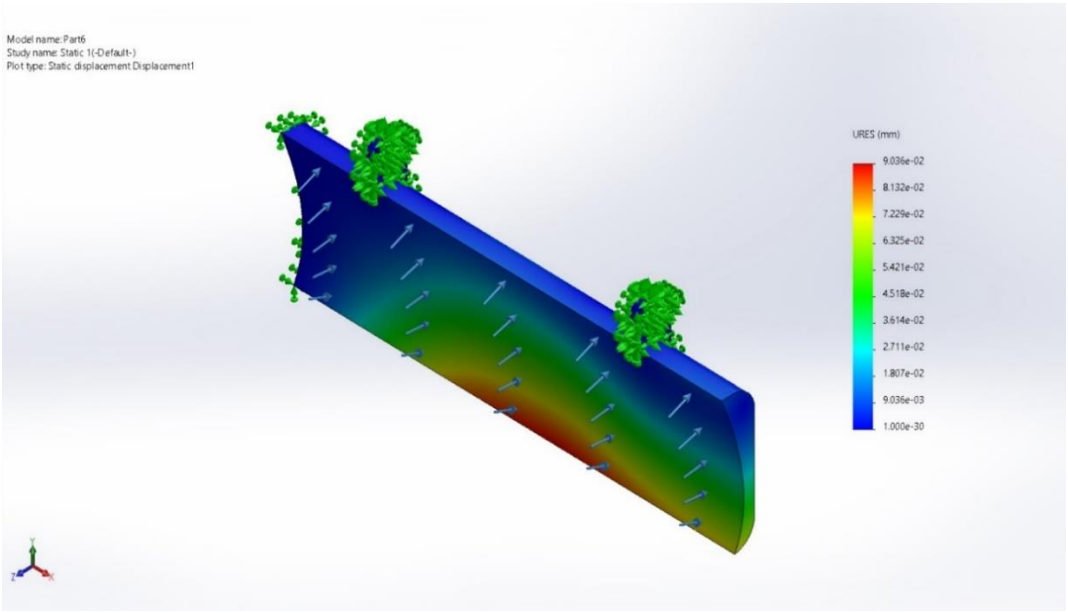
Study Results

Name	Type	Min	Max
Stress1	VON: von Mises Stress	1.055e-03N/m^2 Node: 6163	1.940e+06N/m^2 Node: 13547



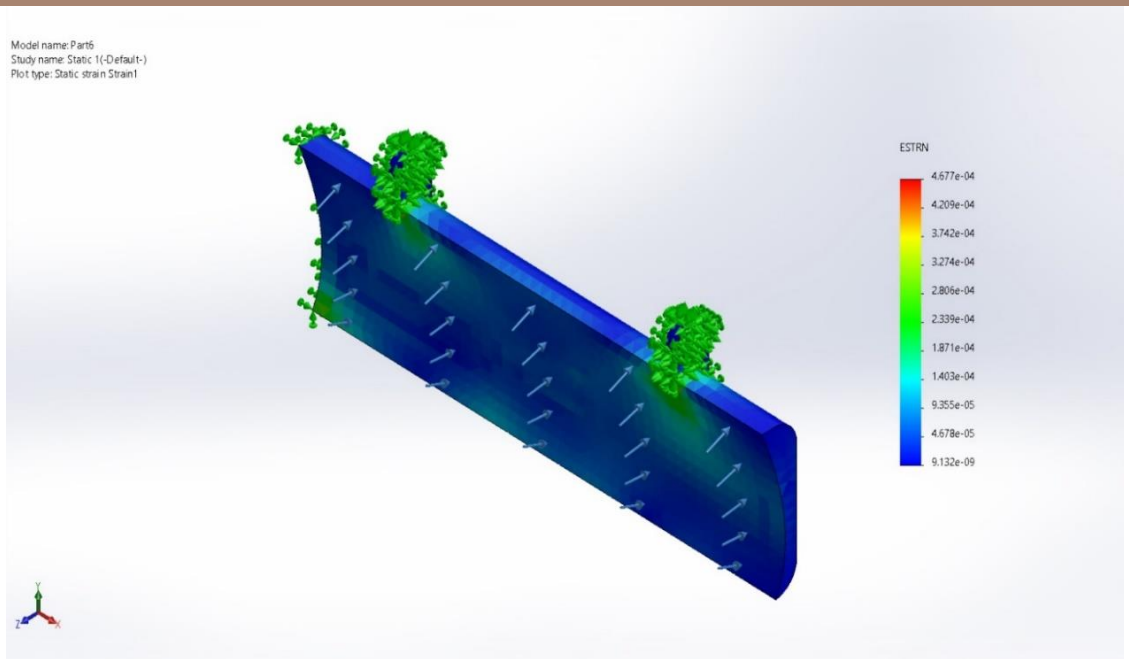
Leveler Stress

Name	Type	Min	Max
Displacement1	URES: Resultant Displacement	0.000e+00mm Node: 24	9.036e-02mm Node: 10373



Leveler Displacement

Name	Type	Min	Max
Strain1	ESTRN: Equivalent Strain	9.132e-09 Element: 1590	4.677e-04 Element: 4894



Leveler Strain

2.11 Materials used and Assumptions.

the materials used and assumptions made for an autonomous seed sowing robot. The focus is on selecting appropriate materials for the chassis, 3D printed parts, components, and water tank. Aluminum is chosen for the chassis due to its strength-to-weight ratio. PLA is selected for 3D printed parts, while alloy steel is used for specific components. Plastic is chosen for the water tank, and acrylic is used for the cover. Assumptions are made regarding load distribution and material properties to ensure the robot's overall functionality and performance.

Designing an autonomous seed sowing robot necessitates careful material selection to meet structural, weight, and functional requirements.

1. Chassis Material: Aluminum The chassis supports various components, including motors, batteries, controllers, and the water tank. To accommodate a weight of approximately 15 kg, a material with a high strength-to-weight ratio is necessary. Aluminum is selected for its favorable characteristics, including excellent strength, lightweight properties, and resistance to corrosion.

2. 3D Printed Parts Material: PLA Parts such as the funnel, drill holder, plate, pipe, leveler, and others are fabricated using 3D printing technology. PLA (Polylactic Acid) is chosen for its ease of printing, biodegradability, and mechanical strength. PLA provides adequate stiffness and durability for the required components while being environmentally friendly.

3. Component Material: Alloy Steel Components such as wheel motor holders, screws, and nuts are vital for ensuring stability and longevity. Alloy steel, known for its exceptional mechanical properties, including high strength, hardness, and resistance to wear, is selected for these critical components. The use of alloy steel guarantees reliable operation under varying loads and environmental conditions.

4. Water Tank Material: Plastic The water tank stores the necessary water supply for the seed sowing operation. Plastic, such as high-density polyethylene (HDPE) or polypropylene

(PP), is suitable for its lightweight nature, corrosion resistance, and ease of manufacturing. Plastic offers an efficient and cost-effective solution for the water tank, ensuring sufficient capacity and durability.

5. Cover Material: Acrylic The cover protects internal components from environmental factors and potential damage.

Acrylic (Polymethyl Methacrylate or PMMA) is chosen for its transparency, impact resistance, and ease of fabrication. Acrylic provides visibility of the internal components while safeguarding them from dust, debris, and potential impacts.

6. Assumptions: The following assumptions were made during material selection:

- The weight distribution and load-bearing requirements were considered when choosing aluminum for the chassis.
- PLA parts were designed with suitable wall thicknesses and structural reinforcements to meet mechanical strength requirements.
- Alloy steel components were selected based on typical material properties, ensuring compatibility with the robot's mechanical design and load requirements.
- Plastic material for the water tank was chosen based on its lightweight and corrosion-resistant properties.
- Acrylic cover material was assumed to provide sufficient protection and durability for the internal components.

Material selection is crucial for the successful development of an autonomous seed sowing robot. The chosen materials, including aluminum for the chassis, PLA for 3D printed parts, alloy steel for specific components, plastic for the water tank, and acrylic for the cover, provide a balanced approach to meet the functional and structural requirements. However, further analysis, testing,

and validation are recommended to ensure the chosen materials align with the specific needs of the robot.

Material	Density (Imperial) (lb/in^3)	Density (Metric) (g/cm^3)	Young's Modulus (Imperial) (ksi)	Young's Modulus (Metric) (GPa)	Yield Strength (Imperial) (ksi)	Yield Strength (Metric) (MPa)	Ultimate Tensile Strength (Imperial) (ksi)	Ultimate Tensile Strength (Metric) (MPa)
Aluminum	0.098	2.70	10.0	69.0	30.0	207.0	45.0	310.0
Plain Carbon Steel	0.283	7.85	29.0	200.0	53.0	365.0	63.0	435.0
Acrylic	-	1.18	-	3.2	-	70.0	-	48.0
PLA	-	1.24	-	3.4	-	44.0	-	303.0

Table 2.1 comparing the mechanical properties.

2.12 Weights

File Name	Quantity	Volume(mm^3)	Mass(g)	Total Weight(g)
aluminum wire 1	1	12.18	0.1	0.1
socket set screw cone point_iso(ISO 4027 - M3 x 4-S)	2	14.43	0.01	0.03
aluminum wire 2	1	17.28	0.14	0.14
hex nut style 1 gradeab_iso(ISO - 4032 - M3 - W - S)	19	48	0.05	0.91
socket head cap screw_iso(ISO 4762 M3 x 10 - 10S)	25	124.63	0.12	3.12
SG90 - Micro Servo 9g - Tower Pro.2	3	155.43	0.2	0.61
socket head cap screw_iso(ISO 4762 M3 x 16 - 16S)	4	161.2	0.16	0.64
hex nut gradec_iso(ISO - 4034 - M5 - S)	3	220.39	0.22	0.66
25m M.F	2	399.89	0.4	0.8
socket head cap screw_iso(ISO 4762 M5 x 20 - 20S)	2	588.41	0.59	1.18
50m F.F	4	729.1	6.2	24.79
socket head cap screw_iso(ISO 4762 M5 x 50 - 50S)	1	1099.61	1.1	1.1
power screw coupling	1	1210.34	1.21	1.21
sonde	1	1520.1	1.52	1.52
servo holder	1	2407.67	3.13	3.13
cuscinetti_lineari_LM8UU_8x15X25	2	3030.63	3.94	7.88
funnel gate	1	4157.22	11.22	11.22
leveller mounting	2	4216.57	5.48	10.96
M8 x M90 rod	2	4523.89	4.52	9.05
SG90 - Micro Servo 9g - Tower Pro.1	3	7044.98	9.16	27.48
soil drill	1	7542.51	60.54	60.54
motor mounting	4	10767.94	84.61	338.46
seed dispense pipe	1	11472.2	14.91	14.91
motor mounting	1	22017.85	28.62	28.62
motor	6	24388.78	191.65	1149.88
drill holder	1	32122.07	41.76	41.76
soil leveller	1	90921.49	118.2	118.2
funnel	1	103431.2	134.46	134.46
Wheel	4	113975.71	132.12	528.48
tank holding plate	1	169886.9	203.86	203.86
chassis	1	328681.23	394.42	394.42
water tank	1	1162446.32	1185.7	1185.7
Payload:				8000
Total:				12305.82

Table 2.2 Weights from SolidWorks

1. Cost: The cost evaluation of the seed sowing robot project is essential to assess its financial feasibility. The total cost of the project was 30,000, which includes expenses related to materials, components, electronics, software development, and labor. A breakdown of the costs incurred during the design and implementation phases was recorded. This evaluation provides insights into the project's affordability and allows for comparisons with alternative solutions in the market.
2. Environmental Impact: Considering the environmental impact of the seed sowing robot project is crucial in promoting sustainable practices. Throughout the design process, environmentally friendly materials were prioritized, and energy-efficient components were utilized wherever possible. By automating the seed sowing process, the project aims to reduce the use of traditional agricultural practices, such as manual labor or large machinery, which can have negative environmental implications. Additionally, the project's potential to optimize seed distribution and reduce wastage contributes to sustainable agricultural practices.
3. Manufacturability: The evaluation of manufacturability in the seed sowing robot project assesses the feasibility of mass production and scalability. During the design phase, considerations were made to ensure that the robot's components and assembly processes are compatible with existing manufacturing techniques. Modular design principles were implemented, enabling efficient manufacturing, maintenance, and potential future upgrades. This evaluation provides insights into the project's ability to be reproduced cost-effectively and efficiently on a larger scale.
4. Ethics: Ethical considerations play a significant role in the development of the seed sowing robot project. Care was taken to ensure that the project adheres to ethical standards and does not infringe upon any regulations or guidelines related to robotics and agriculture. The use of the robot aims to enhance agricultural practices while minimizing the negative impact on human labor and ensuring the welfare of plants. Ethical concerns regarding data

privacy and the responsible use of autonomous systems were also addressed during the project's implementation.

5. Social and Economic Impact: The evaluation of the social and economic impact of the seed sowing robot project analyzes the potential benefits and implications for society and the economy. The robot's automation capabilities can contribute to increased efficiency and productivity in agriculture, benefiting farmers by reducing labor costs and improving crop yields. Furthermore, by streamlining the sowing process, the project aims to alleviate the burden on agricultural workers, potentially leading to improved work conditions. The socio-economic impact evaluation takes into account both the short-term and long-term effects of implementing the seed sowing robot technology.
6. Health and Safety: The health and safety evaluation of the seed sowing robot project focuses on ensuring the well-being of operators, farmers, and the general public. Risk assessments were conducted during the design and implementation phases to identify potential hazards and implement necessary safety measures. Safety features such as emergency stop buttons, obstacle detection, and proper guarding were incorporated into the robot's design. Additionally, user-friendly interfaces and clear operating instructions were developed to minimize the risk of accidents or injuries.
7. Sustainability: Sustainability considerations were integrated into the seed sowing robot project to promote long-term viability and environmental consciousness. The use of efficient energy sources, such as rechargeable batteries or solar power, was explored to reduce reliance on non-renewable resources. The robot's design also allows for easy maintenance and repair, ensuring a longer lifespan and reducing waste generation. By encouraging sustainable agricultural practices through precise seed distribution and optimized resource utilization, the project aims to contribute to a more sustainable and environmentally friendly farming ecosystem.

Chapter 3: Electronics Components & Circuit

3.1 Electronics Components

3.1.1 Arduino Mega 2560

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega 2560 board is compatible with most shields designed for the Uno and the former boards Duemilanove or Diecimila.

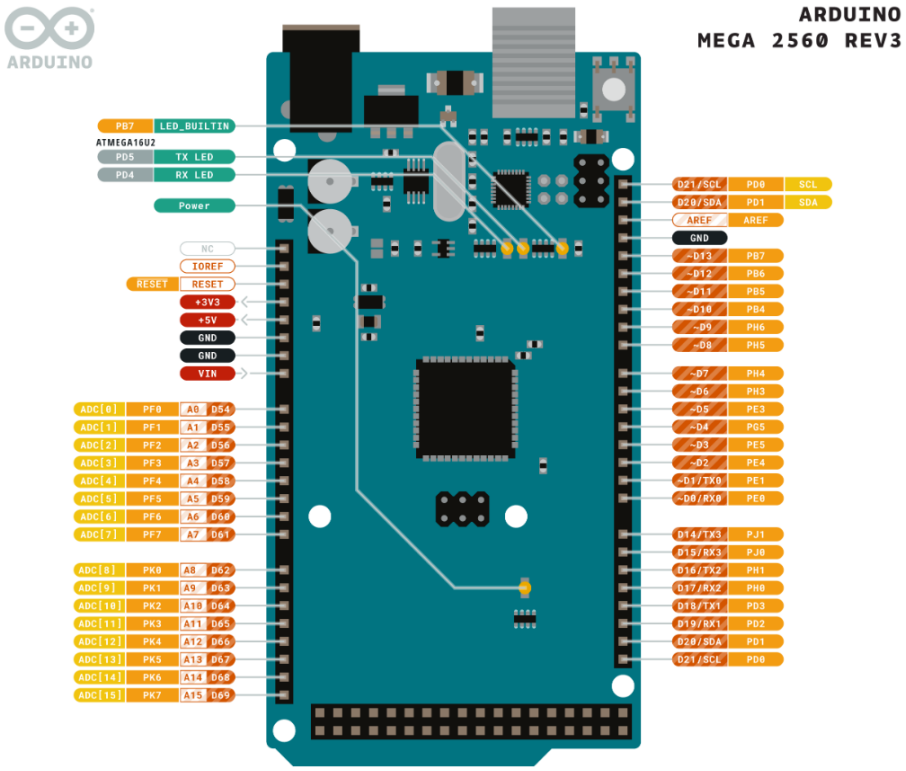


Figure 3.1 Pinout Diagram Arduino Mega 2560

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Table 3.1 specifications of Arduino Mega 2560

3.1.2 DC Motor Driver (Cytron MDD 10A)

MDD10A shipped after Jan 2017 is in Rev2.0 which Vmotor supports up to 30VDC.

DC brushed motor is the most commonly used and widely available motor in the market. Getting the motor to rotate is fairly easy, just connect the two terminals to power source and it will start spinning, that's the beauty of DC brushed motor. Yet, if you want to control the speed, direction, activation and automate all these functions, check out this tutorial: 5 easiest ways to control a DC motor. And using a motor driver is one of the ways. MDD10A is one of Cytron's motor driver series which offers easy-to-use features, and it can drive two independent motors.

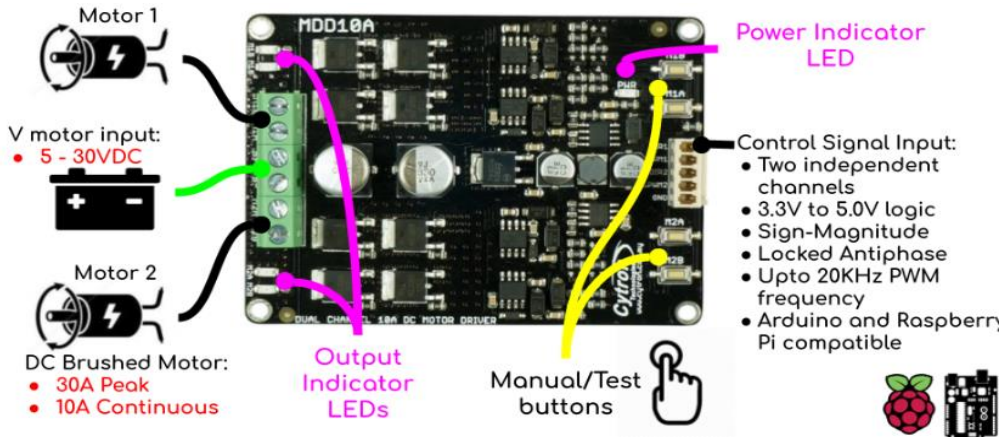


Figure 3.2 Dual Channel 10Amp DC Motor Driver

Operating Condition	
Motor Channel	2
Operating Voltage (VDC)	5 to 25 (Rev 1.0) 5 to 30 (Rev 2.0)
Peak Current (A)	30 (10 seconds)
Continuous Current (A)	10
Operating Modes	
Arduino Shield	No- can be used with Wire connection
PWM (Lock-Antiphase and Sign-Magnitude)	Yes
UART	NO
Analog	No
RC Servo Signal	NO
Special Protection	
Polarity Protection	NO
Appearance	
LEDs Indicator	Yes
Test/Manual Button	Yes
Cooling Fan	NO
Other Features	
others	No heat sink is required

Table 3.2 Specifications of Cytron MDD 10A

3.1.3 LIDAR

lidar is currently the most popular and cost-effective lidar in the open-source hardware field. It possesses small size and excellent quality, support 360° scanning and distance measurement, can be used on a personal computer or Jetson NANO Raspberry Pi and other micro-controllers by a USB data cable. We will provide many tutorial materials and codes, and ROS robot learning materials. These lidar are divided into 6 different models, each of which has a different shape and performance, and users can choose according to their requirements.

Figure 3.3 light detection and ranging

Series	Triangular Ranging					TOF ranging		
Model	A1M8		A2M12	A3M1		S1M1	S2/S2L	M2M2
	A1 up ver	A1		Enhanced mode	Outdoor mode			
								
Recommended Applications	Smart sweeper, household robot (indoor)		Commercial or consumer robot 3D modeling (indoor)	High performance (indoor)	Stable performance, strong ability to resist sunlight (indoor/outdoor)	Strong ability to resist sunlight (indoor/outdoor)	Strong ability to resist sunlight (indoor/outdoor)	Commercial robot environmental mapping, hand-held measurement (indoor/outdoor)
Measuring radius	0.15m - 12m		0.2m - 16m	White object: 25m Black object: 10m	White object: 20m Black object:	White object: 40m Black object: 10m	White object: 0.05~30m Black object: 0.05~10m	0.1m~40m
Measurement dead zone	No reference value		No reference value	0.2m		0.1m	0.05m	No reference value
Communication rate	115200bps			256000bps			1M	
Sampling frequency	8K			16K	10K	9.2K	32K	9.2K
Scanning frequency	5.5Hz-10Hz		5Hz-15Hz	15Hz (10Hz-20Hz Adjustable)		8Hz-15Hz		
Angular resolution	≤1°		0.9°	0.225°		0.391°	0.12°	0.391°
Mechanical dimensions (unit: mm)	96.8*70.3*55		⌀76*41	⌀76*41		55.5*55.5*51	77.1*77*38.85	77.1*57*74.9
Supply current	100mA		450mA - 600mA			400mA		750mA - 1300mA
Power consumption	0.5W		2.25W-3W			> 2W		3.75W-6.5W
Output	UART serial port (3.3V level)							Ethernet/WiFi
Operating temperature	0℃~40℃					(-5℃-45℃)	(-10℃~50℃)	(-5℃-45℃)
Ranging accuracy	Actual distance 1% (≤3 m) Actual distance 2% (3-5 m) Actual distance 2.5% (>5m)					±5cm	±3cm	≤5cm (Within the range)
Supply voltage: 5V Scanning Range: 360°								

Table 3.3 LIDAR Parameter Comparison

3.1.4 DC Motor 12V witt Gearbox

We use 12V DC motor and 4.0amp rated current, which is powerful.

motor that gives us the needed torque (28.5 Kg.cm) and suitable speed of 83r/min with a 1:60 metal gearbox. The gears are all steel and the output shaft is 8mm diameter. The motor has extra rear shaft (on the high-speed side before the gearbox) which allow you to mount motor encoder for feedback of motor speed.

Specifications

- Rated voltage : 12VDC
- No load speed : 83 r/min
- Rated speed : 63 r/min
- Load torque current : 4 amp
- Rated torque : 28.5 kgf.cm
- Stall current :14.0 amp

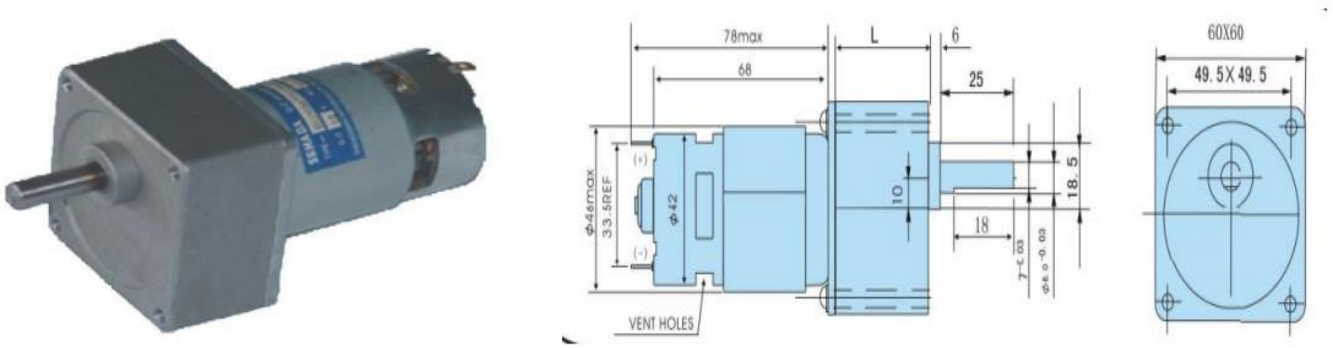


Figure 3.4 DC Geared Motor with Dimensions

3.1.5 Encoder

Calculating the number of revolutions per minute for each motor is one of the most common ways to track the robot's position. Odometry feedback refers to this method. Each encoder generate number of known

pulses per one revolution of wheel which it gives us indication of how much robot is moved (distance).We used 7 PPR encoder which generate 7 pulse per revolution in input shaft but our motor is attached to gearbox of 1:60 ratio so one revolution of output shaft will generate 420 pulse.Knowing that encoder has 2 channel with phase shift of 90 degree.

Specifications

- Encoder type : hall effect encoder
- supply voltage: 4.5~24v
- operation current :10 micro amp

Figure 3.5 Incremental Encoder

3.1.6 Raspberry Pi 4 Model B

The Raspberry Pi 4 offers ground-breaking increases in processor speed, multimedia performance, memory, and connectivity compared to the prior-generation boards, while retaining backwards compatibility and similar power consumption. The Raspberry Pi 4 provides desktop performance comparable to entry-level x86 PC systems. The Raspberry Pi 4 comes in three on-board RAM options for even further performance benefits: 2GB, 4GB and 8GB.

Features of Raspberry Pi computer, we can note:

- A high-performance 64-bit quad-core processor
- Dual display support with resolutions up to 4K via a pair of micro-HDMI ports
- Hardware video decoding up to 4Kp60.
- 4 GB of RAM
- A connection to the dual-band wireless local area network 2.4/5.0 GHz
- Bluetooth 5.0 / Gigabit Ethernet / USB 3.0 / PoE features (via a separate HAT PoE add-on module)

Specifications

- SoC Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit at 1.5GHz
- SDRAM 4 GB LPDDR4-2400
- Wireless LAN 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac, Bluetooth 5.0, BLE
- True Gigabit Ethernet
- 2 USB 3.0 ports, 2 USB 2.0 ports
- Fully backward compatible 40-pin GPIO connector
- 2 HDMI micro ports supporting video resolution up to 4K 60Hz
- 2-way MIPI DSI DSI/CSI ports for camera and display
- Stereo audio output and composite video port, 4-pole

- Slot for Micro SD card, for operating system and data storage
- Requires 5.1V, 3A power supply via USB-C or GPIO
- PoE (Power over Ethernet) enabled (requires PoE HAT)

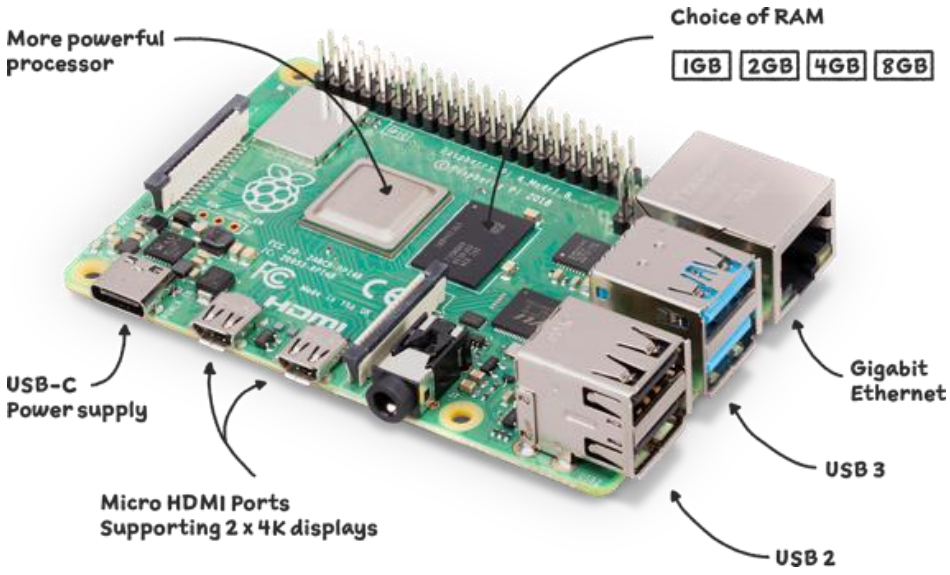


Figure 3.6 Raspberry Pi 4 Model B

3.1.7 ESP32

ESP32 is created by Espressif Systems with a series of SoC (System on a Chip) and modules which are low cost with low power consumption.

This new ESP32 is the successor to the well-known ESP8266(became very popular with its inbuilt WiFi). ESP32 not only has Built in WiFi but also has Bluetooth and Bluetooth Low Energy. In other words we can define ESP32 as “ESP8266 on Steroids”.

ESP32 chip ESP32-D0WDQ6 is based on a Tensilica Xtensa LX6 dual core microprocessor with an operating frequency of up to 240 MHz.

The small ESP32 package has a high level of integrations such as:

- Antenna switches
- Balun to control RF
- Power amplifier
- Low noise reception amplifier
- Filters and power management modules

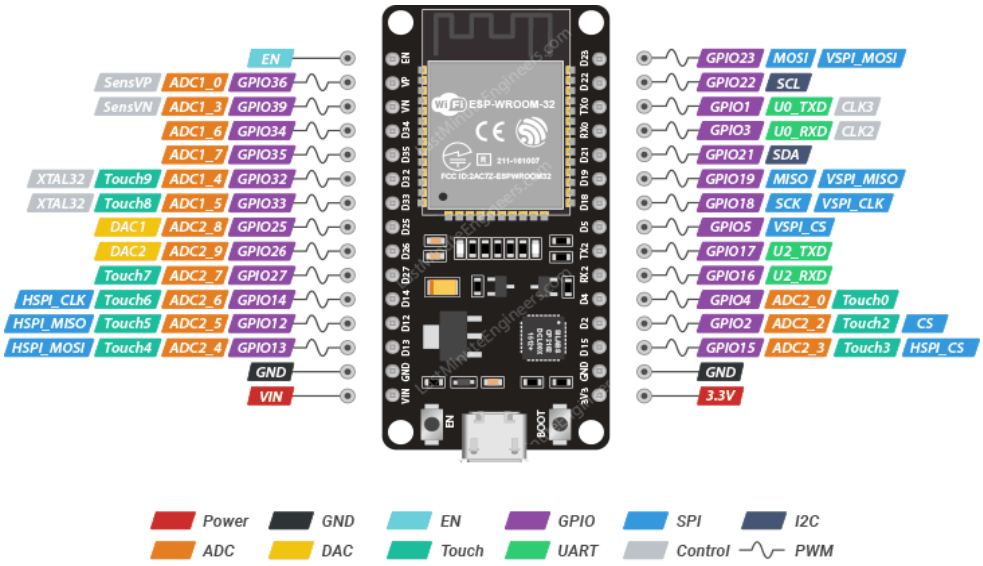


Figure 3.7 ESP32 Pinout

Specifications

Categories	Items	Specifications
Certification	RF certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Bluetooth certification	BQB
	Green certification	RoHS, REACH
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I^2C, LED PWM, Motor PWM, I^2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Operating voltage/Power supply	2.7 ~ 3.6V
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range	–40°C ~ 65°C
	Package size	(18.00±0.10) mm x (31.40±0.10) mm x (3.30±0.10) mm

Table 3.4 Specifications of ESP32

3.1.8 SERVO 90SG

Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds

but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a

motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

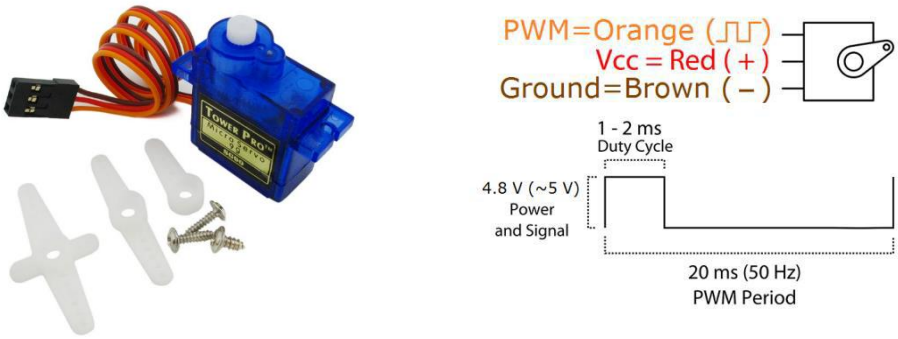


Figure 3.8 Servo motor SG90

Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Table 3.5 Dimensions & Specifications of servo motor

3.1.9 Keypad 4x3

Keypads are one of the most popular components that are widely used in electronics. Everybody can communicate with the system by switches. Normally, every key occupies one digital pin of the microcontroller. But by using a 3×4 keypad you can reduce the occupied pins. With this module, you can use all 12 switches by occupying only 7 pins of the microcontroller.

How it works

The Keypad 4×3 features a total of 12 buttons in Matrix form. Consider 4 rows as input and 3 columns as output. Each switch is connected from one side to a row and from the other side to a column. For example, if we press switch number 1, the input of this row is saved at the output of its column.

Figure 3.9 Keypad 4x3

3.1.10 DC Motor with encoder 12V

DC motor encoders are used for speed control feedback in DC motors where an armature or rotor with wound wires rotates inside a magnetic field created by a stator. The DC motor encoder provides a mechanism to measure the speed of the rotor and provide closed-loop feedback to the drive for precise speed control.

Figure 3.10 DC Motor with encoder 12V

Specification

- Gear ratio: 43.8:1
- No-load speed: 251 + 10% RPM
- No-load current: 350 mA
- Start Voltage: 1.0 V
- Stall Torque: 18 Kg.com
- Stall Current: 7 A
- Insulation resistance: 20 M Ω
- Encoder Operating Voltage: 5 V
- Encoder type: Hall
- Encoder Resolution: 16CPR (motor shaft)/700 CPR (gearbox shaft)
- Weight: 205g

3.1.11 Moisture sensor

can be used to detect the moisture of the soil. When the soil is dry, the module outputs a high level , whereas the output low. Using this sensor makes an automatic watering system, so that your garden plants need people to manage.

Instructions

- A soil moisture and humidity module most sensitive to the environment, usually used to detect soil moisture.
- Modules in the soil moisture reach the set threshold , DO port output high when soil moisture exceeds a set threshold, the module D0 output low.
- Digital outputs D0 small plates can be directly connected with the microcontroller through the microcontroller to detect high and low , thereby detecting soil moisture.

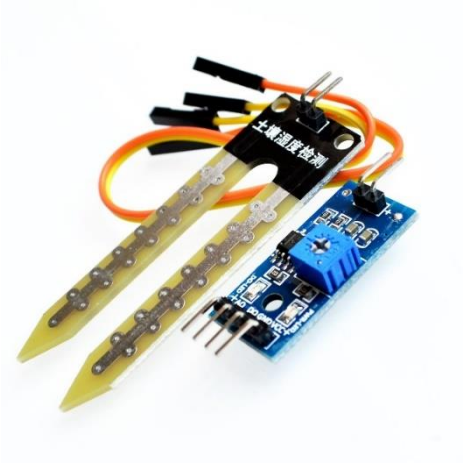


Figure 3.11 Moisture sensor

Specifications

- Operating Voltage 3.3V-5V.
- Module Dual Output mode , a simple digital output , analog output more accurate.
- Small PCB board size: 3cm * 1.6cm.
- VCC external 3.3V-5V
- GND GND External
- DO small board digital output interfaces (0 and 1)
- AO small board analog output interface

3.1.12 Current Sensor

A device that is used to detect & also change current to assessable output voltage is known as a current sensor. This output voltage is simply proportional to the current flow throughout the measured path. After that, this output voltage signal is used to display the current measured within an ammeter, for controlling purposes or simply stored for more analysis within a data acquisition system. So this is the function of a current sensor. The working principle of the current sensor is; once current is supplied throughout a circuit or a wire then a voltage drop takes place and also magnetic field will be generated nearby the current-carrying conductor. So, there are two kinds of current sensing direct current sensing & indirect current sensing.

Specifications

Measuring Range	5 A
Response Time	<5 μs
Power consumption	10 mA
Bandwidth	80 kHz
Sensitivity	185 mV/A
Linearity	1.5%
Accuracy at −40 to 25 °C	0.054 mV/A/°C
Accuracy at 25 to 150 °C	−0.008 mV/A/°C

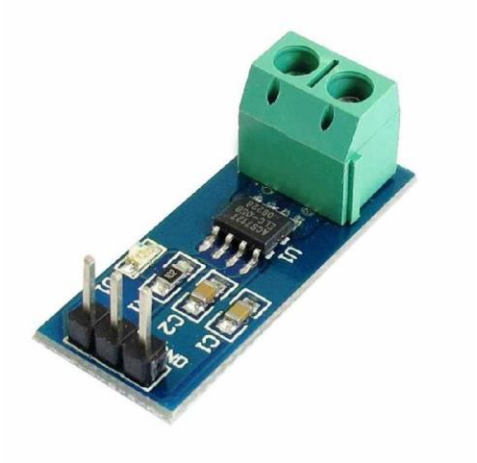


Figure 3.12 Current Sensor

3.1.13 MPU9250

The MPU-9250 is a 9-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer and a Digital motion processor. Based on the IC MPU-9250, which is the world's smallest 9-axis MotionTracking device and incorporates the latest InvenSense design innovations, this module features three 16-bit ADC for digitizing the gyroscope outputs and three 16-bit ADCs for digitizing the accelerometer outputs and three 16-bit ADCs for digitizing the magnetometer outputs. low power, low cost, and high performance make it suitable for consumer electronics equipment including smartphones, tablets and wearable sensors.

Features

- Three 16-bit analog-to-digital converters (ADCs) for digitizing the gyroscope outputs
- Three 16-bit ADCs for digitizing the accelerometer outputs.
- Three 16-bit ADCs for digitizing the magnetometer outputs.
- Gyroscope full-scale range of ±250, ±500, ±1000, and ±2000°/sec (dps)
- Accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g
- Magnetometer full-scale range of ±4800μT
- I2C and SPI serial interfaces
- Internal Digital Motion Processing engine supports advanced MotionProcessing and low power functions such as gesture recognition using programmable interrupts.
- Power supply: 3-5v

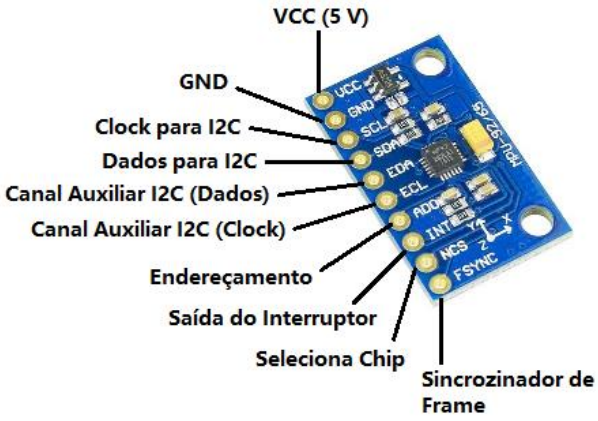


Figure 3.13 MPU9250

3.1.14 Level Sensor

The sensor generates an output voltage proportional to the resistance; by measuring this voltage, the water level can be determined.

S (Signal) is an analog output pin that will be connected to one of your Arduino's analog inputs.

+ (VCC) pin provides power to the sensor. It is recommended that the sensor be powered from 3.3V to 5V. Please keep in mind that the analog output will vary depending on the voltage supplied to the sensor.

– (GND) is the ground pin.

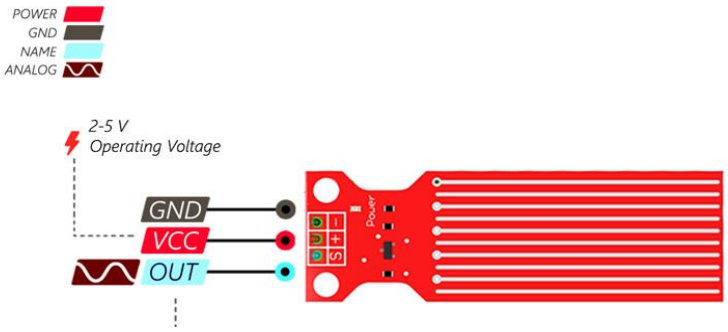


Figure 3.14 Level Sensor

Features

- The more water the sensor is immersed in, the better the conductivity and the lower the resistance.
- The less water the sensor is immersed in, the poorer the conductivity and the higher the resistance.

3.1.15 USB HUB (BYEASY)

The BYEASY Powered USB Hub is a popular USB hub that provides additional USB ports for connecting multiple devices to a computer or laptop. While I don't have the most up-to-date information on the specific model you're referring to, here are some common specifications and features you can expect from a typical BYEASY Powered USB Hub:

Number of Ports: BYEASY hubs usually come in various configurations, typically ranging from 4 to 10 USB ports. Some models may also include a mix of USB 2.0 and USB 3.0 ports.

USB Standards: The hub is likely to support USB 2.0 and/or USB 3.0 standards, with backward compatibility for older USB devices.

Power Supply: A powered USB hub means it comes with an external power adapter, allowing it to provide sufficient power to connected devices. This is particularly useful for high-power devices like external hard drives or charging smartphones and tablets.

Data Transfer Speed: USB 2.0 supports a maximum data transfer rate of 480 Mbps, while USB 3.0 supports up to 5 Gbps. The specific speeds of the hub's ports will depend on the version of USB supported.

Figure 3.15 USB HUB

3.1.16 Wire connector 2 in 8

- Safety and reliability: DIN rail terminal blocks are made from insulated material, flame-retardant, corrosion resistance, aging resistance, high pressure resistance. Strong structure firmly locked to DIN guide, long service life.
- Product feature: Wire splicers instead of the ordinary rail connector, reduce the wiring time by 50~70% with our new design. The wire connectors support a variety of installation methods (guide rail and screw mount). Make the connection easier faster and lower maintenance cost.
- Widely applicable: Lever wire connectors can be used for home appliance, lighting, electrical control, power supply, switches, machinery and other wire connection parts. Every junction independent shrapnel, different wire managed separately, improving safety performance.

Figure 3.16 Wire connector

3.1.17 Emergency button

Emergency stop push buttons are generally used in industrial sectors, especially in factories and manufacturing facilities. The emergency stop push button is a safety device connected to heavy-duty machinery. They serve as fail-safe control switches meant for the safety of any high-load industrial machinery and its users. During emergencies, they are used when the machinery's master switch cannot be accessed on time. Once pressed, the emergency stop push button disrupts any power supply to the machinery immediately and is also known as the kill switch or Estop switch. The emergency stop push buttons are usually coated in red paint to be spotted easily during any emergency.

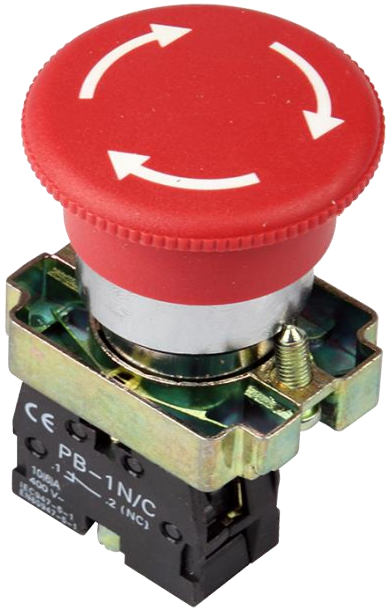


Figure 3.17 Emergency button

3.1.18 TFT LCD

is a variant of a liquid-crystal display that uses thin-film-transistor technology[1] to improve image qualities such as addressability and contrast. A TFT LCD is an active matrix LCD, in contrast to passive matrix LCDs or simple, direct-driven (i.e. with segments directly connected to electronics outside the LCD) LCDs with a few segments.

TFT LCDs are used in appliances including television sets, computer monitors, mobile phones, handheld devices, video game systems, personal digital assistants, navigation systems, projectors, and dashboards in some automobiles and in medium to high end motorcycles.

Technical Parameters

- Display color: 16BIT RGB 65K color
- Size: 1.8 (inch)
- Type: TFT
- Driver chip: ST7735S
- Resolution: 128 * 160 (Pixel)
- Module interface: 4-wire SPI interface
- Backlight: 2 White Led
- Effective display area: 28.03x35.04 (mm)
- Module PCB size: 38.30x62.48 (mm)
- Working temperature: -20 ℃ ~ 60 ℃
- Storage temperature: -30 ℃ ~ 70 ℃
- Working voltage: 5V / 3.3V
- Weight (including packaging): 18 (g)

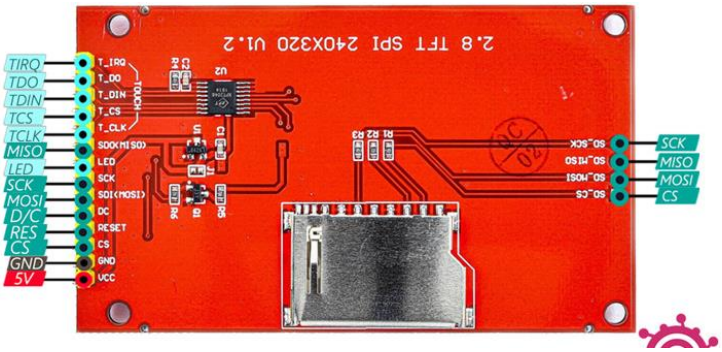


Figure 3.18 TFT LCD

3.1.19 XL4015 DC-DC Buck Converter Module

one such application is charging a lithium battery at a constant current using this module with ease. It's also a good idea to use a constant current when you're testing a circuit for the first time, so that any mistakes you might have made in the build process can be minimised. The device's PWM can reach 100% duty cycle and 180 kHz operating frequency at full load. The module has a 5A output current and can operate at up to 96 percent efficiency. Thermal shutdown, short circuit protection, and current limit are all included in the device's safety features.

Specification

- Output voltage: 0.8V-30V
- Input voltage: 5V-32V
- Output current: adjustable maximum 5A
- Size: 51 * 26.3 * 14 (L * W * H) (mm)
- Operating Temperature: -40 â„ƒ to +85 â„ƒ
- Voltage Regulation: ± 2.5%
- Load Regulation: ± 0.5%
- Switching frequency: 300KHz
- Conversion efficiency: 95% (the highest)

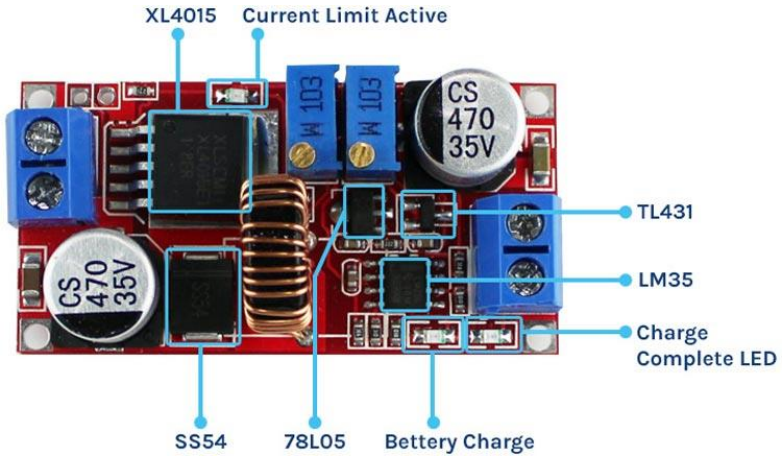


Figure 3.19 Buck Converter Module

3.1.20 lithium-ion battery

A lithium-ion or Li-ion battery is a type of rechargeable battery which uses the reversible reduction of lithium ions to store energy. The negative electrode of a conventional lithium-ion cell is typically graphite, a form of carbon. This negative electrode is sometimes called the anode as it acts as an anode *during discharge*. The positive electrode is typically a metal oxide; the positive electrode is sometimes called the cathode as it acts as a cathode *during discharge*. Positive and negative electrodes remain positive and negative in normal use whether charging or discharging and are therefore clearer terms to use than anode and cathode which are reversed during charging.

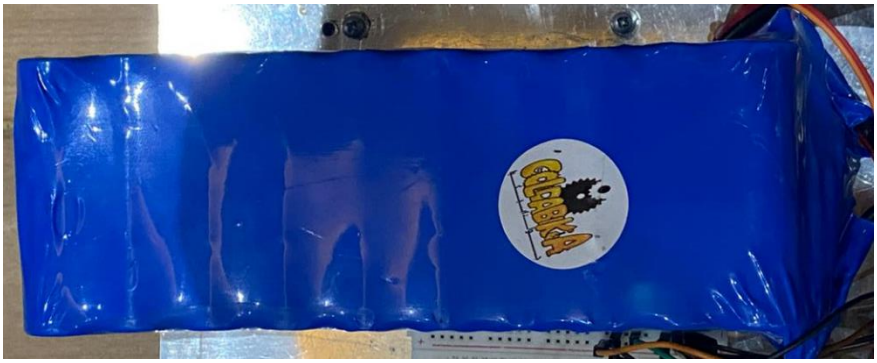


Figure 3.20 lithium-ion Battery

\

3.2 Electrical Circuit

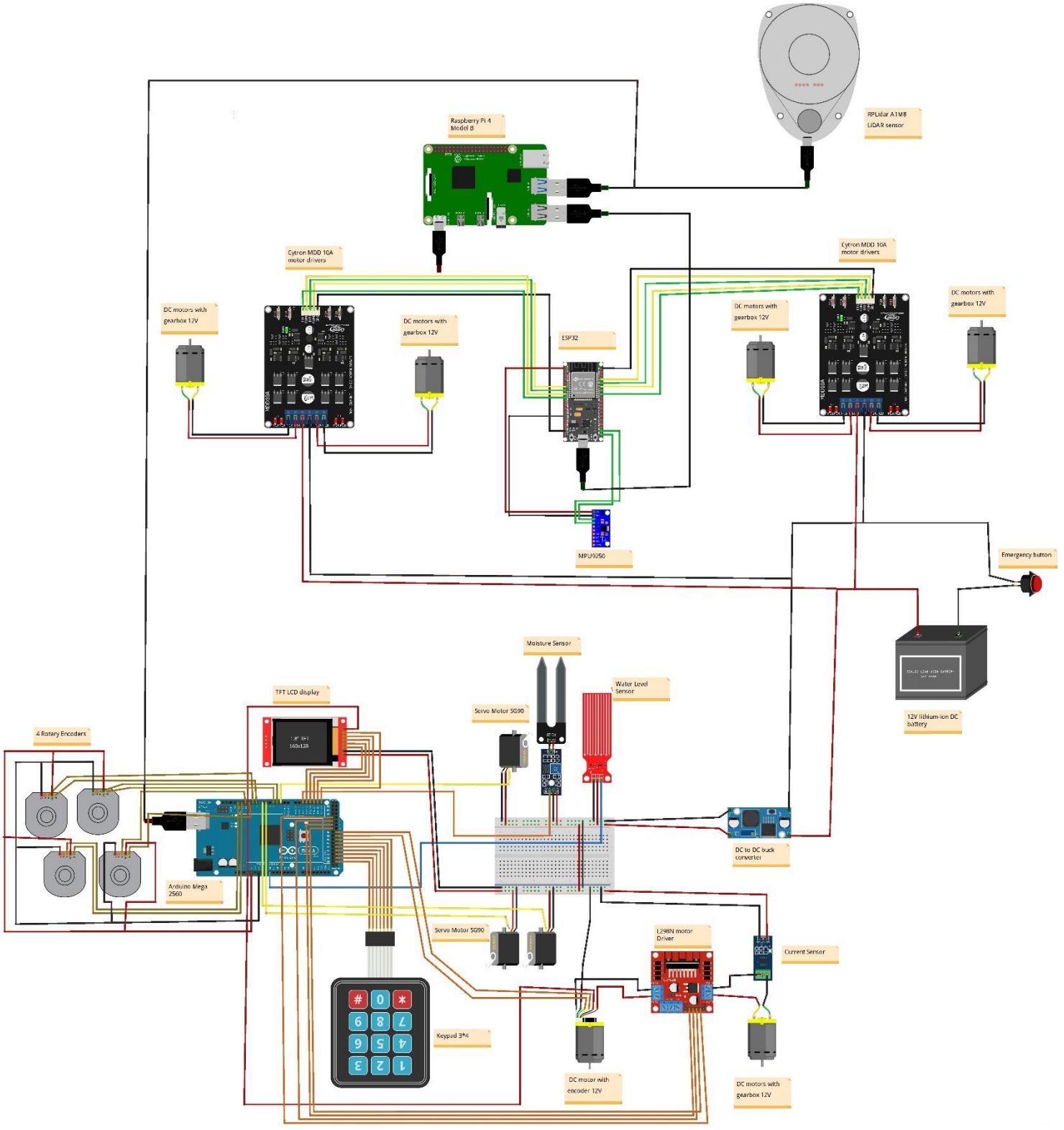


Figure 3.21 Electrical Circuit

Chapter 4: Robot Operating System

Overview

In this Chapter we will be going through and introduction about ROS, its objectives, terminologies used in it and how to create and initialize a workspace.

We will go step by step in the installation of ubuntu 20.04, and ROS to begin creating the packages needed later on with the project.

Explain how ROS communicates, and how nodes, topic, services work.

As well as the most used tools that are used while testing a package in ROS to visualize the data coming from simulation or hardware.

4.1 Introduction

Robots are made up of numerous functioning components and require specific knowledge in a variety of domains. As a result, several technical
constraints must be addressed, and extra study is required before robots may be employed in everyday life. Not just professionals, but also companies in adjacent industries and common users, must work together to achieve this. We require a platform for collaboration and technical progress in addition to the implementation and use of robots. ROS has various elements for spreading and lowering entry barriers to technology.

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.[21]

ROS stands for Robot operating system, those who are unfamiliar with ROS may mistakenly believe it is identical to the operating systems like windows or iOS.

A more appropriate description of ROS is that it is a Meta-Operating System. Although the word "meta-operating system" is not defined, it refers to a system that uses a virtualization layer between applications and distributed computing resources to accomplish tasks such as scheduling, loading, monitoring, and error handling.

4.2 Objective of ROS

The goal of ROS is to “build a development environment that allow robotic software development to collaborate on a global level”. In order to maintain a large code base which can be unmanageable in some cases in the robotics field, as well as an extremely tight coupling between different areas of the system’s functionality. Since there are many interfaces and components in our system that would raise integration complexity, ROS was

the appropriate choice because we wanted to avoid writing an unwieldy program which would be hard to maintain.

ROS sits on top of Linux and is used as a collection of software frameworks for robotic control and communication. Packages are a unit of organization of ROS code that contains source code for ROS nodes. Nodes are executables that run the software for the system and have the ability to communicate with other nodes; To communicate between nodes, ROS uses a publisher/subscriber model. Although ROS supports a variety of opera ting systems, since Ubuntu is the most widely used among ROS users.

4.3 Steps for software implementation

4.3.1 Ubuntu installation

- Download Ubuntu 20.04 LTS ISO File

  Download the ISO file from the link below

  https://releases.ubuntu.com/focal/

- Create a Bootable Disk

  next step is to burn the downloaded ISO image into the USB/DVD or flash drive to boot the computer from that drive, using Rufus or any similar apps. balenaEtcher - Flash OS images to SD cards & USB drives

  Change the boot sequence from the BIOS in order to boot from the flash drive.

- Boot from USB

  Once booted you will be presented with the following page with options included “try Ubuntu” and “install Ubuntu”.

  Click on “install Ubuntu” to continue the installation.

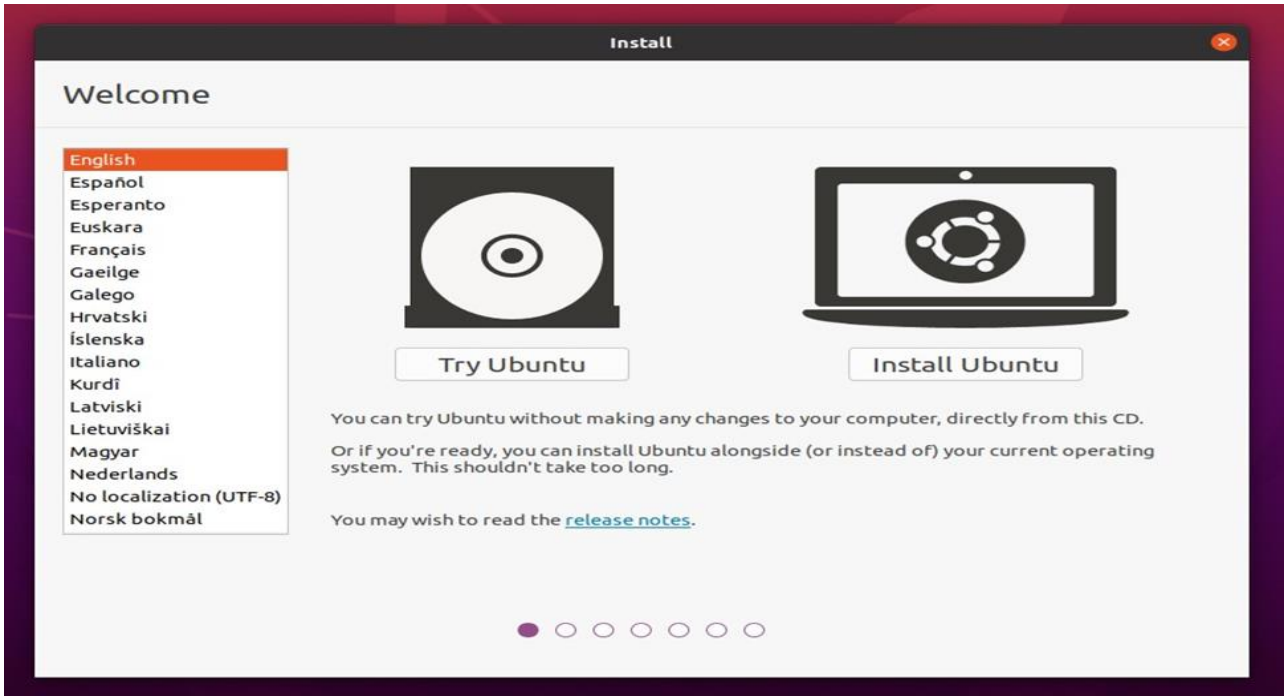


Figure 4.1 Installation Option

You will be asked to select your keyboard layout. Once you've chosen one, click Continue

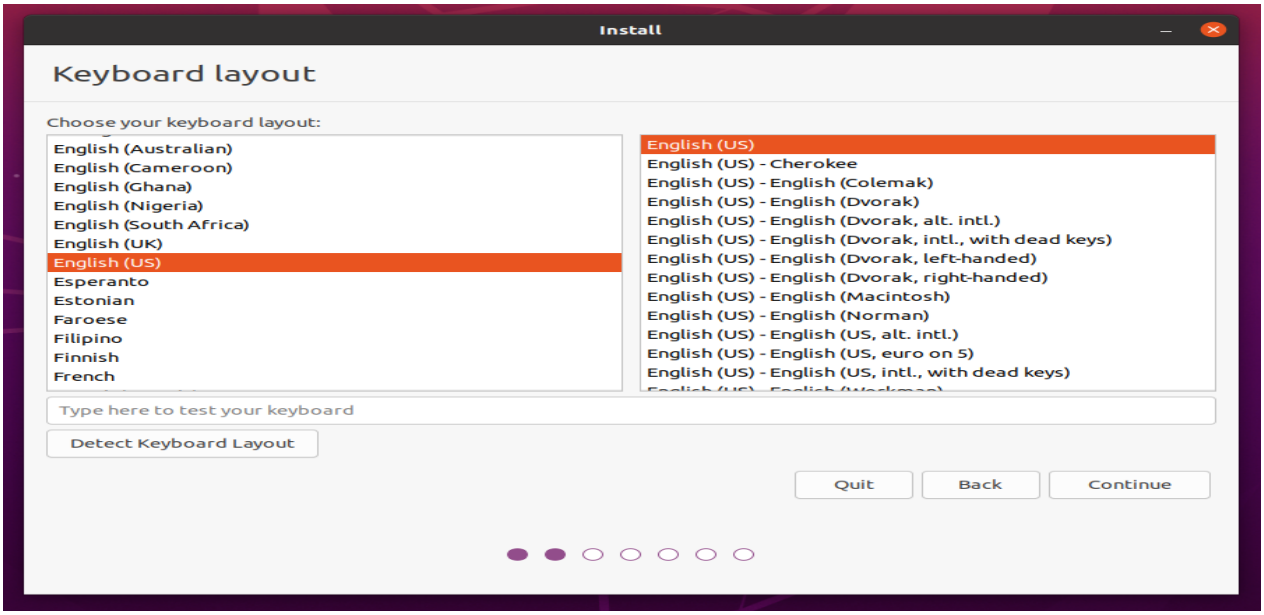


Figure 4.2 keyboard layout

- Preparing to install Ubuntu and other software.

  Install third party software for graphics and Wi-Fi hardware, MP3 and additional media formats Select this option if your system has internet connectivity)

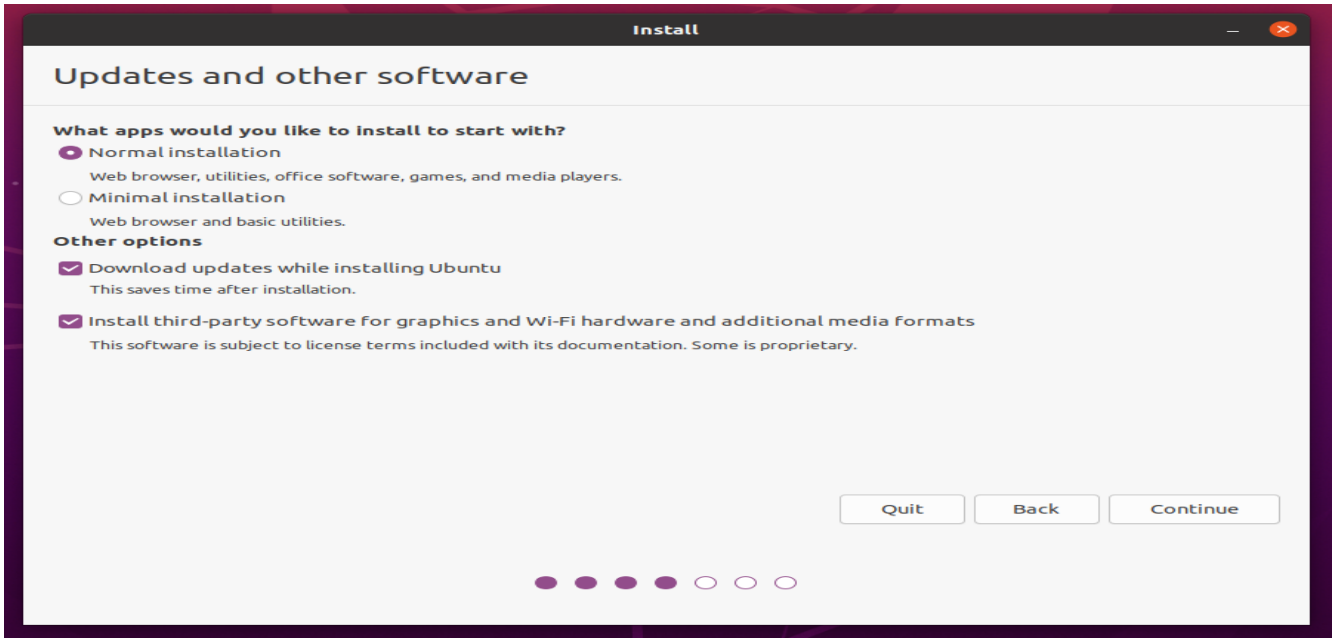


Figure 4.3 Updates and software

- Select the appropriate installation type

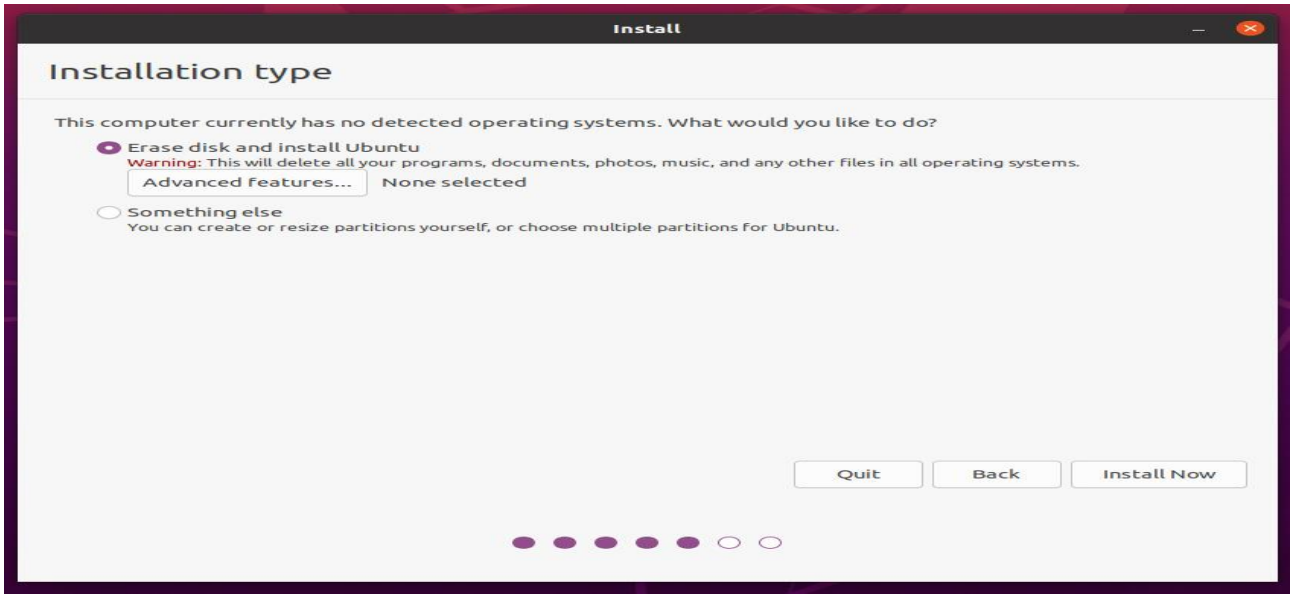


Figure 4.4 Installation type

Encrypt the new Ubuntu installation for security – Choose this option if you are looking for extended security for your disks as your disks will be completely encrypted. If you are beginner, then it is better not to worry about this option.

Use LVM with the new Ubuntu installation – Choose this option if you want to use LVM based file systems.

Something Else – Choose this option if you are advanced user and you want to manually create your own partitions and want to install Ubuntu along with existing OS (May be Windows or other Linux Flavor)

- Time zone and user credentials

Figure 4.5 Time zone

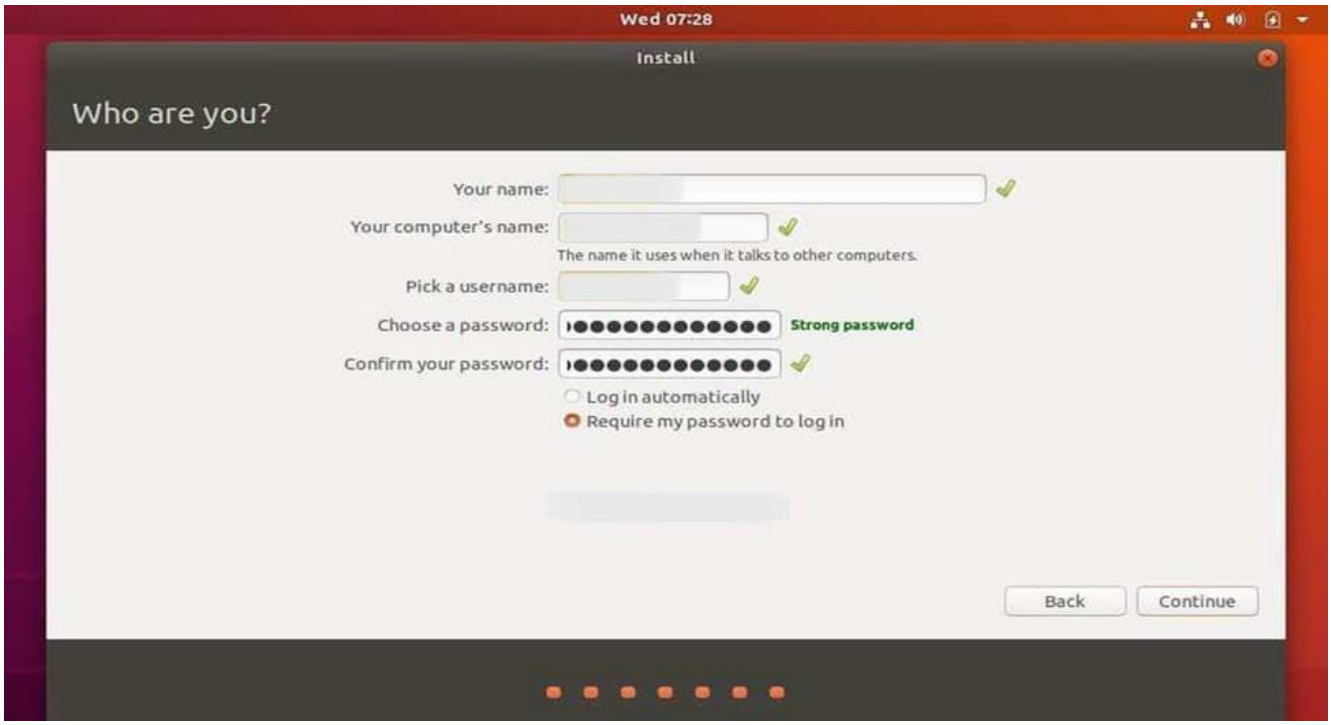


Figure 4.6 User Credentials

- Start installing ubuntu 20.04

  The installation of Ubuntu 20.04 LTS starts now and will take around 5-10 mins depending on the speed of your computer.

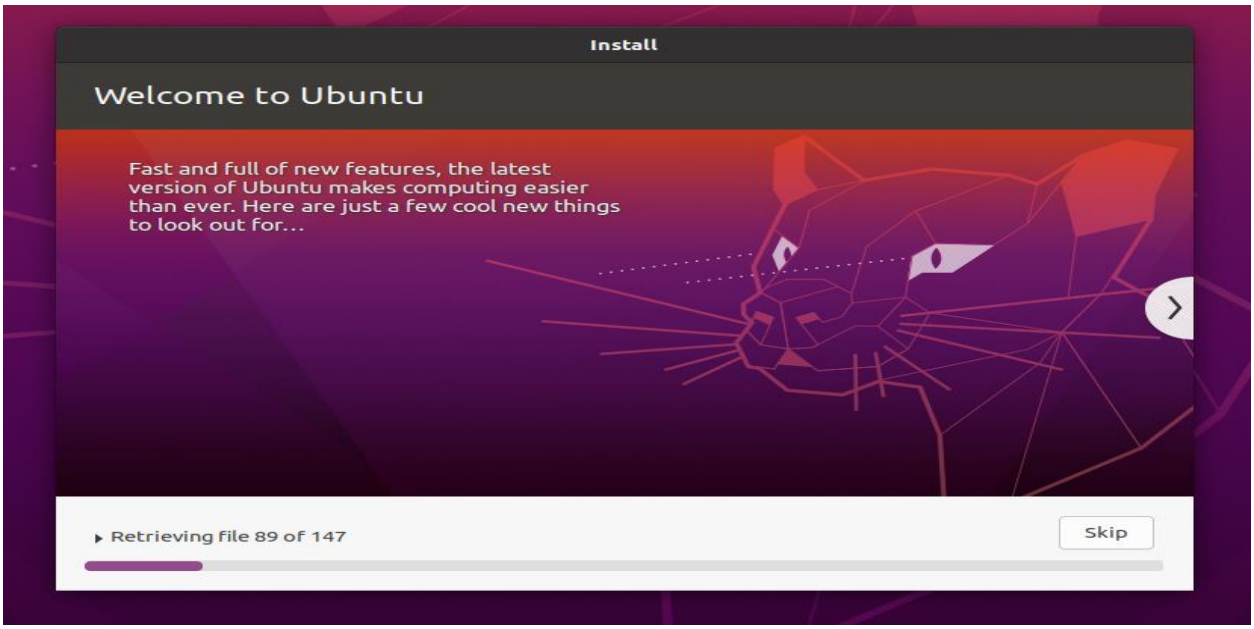


Figure 4.7 Installation

- Restart your system

Once the installation is completed, remove the USB/DVD from the drive and Click "Restart Now" to restart your system.

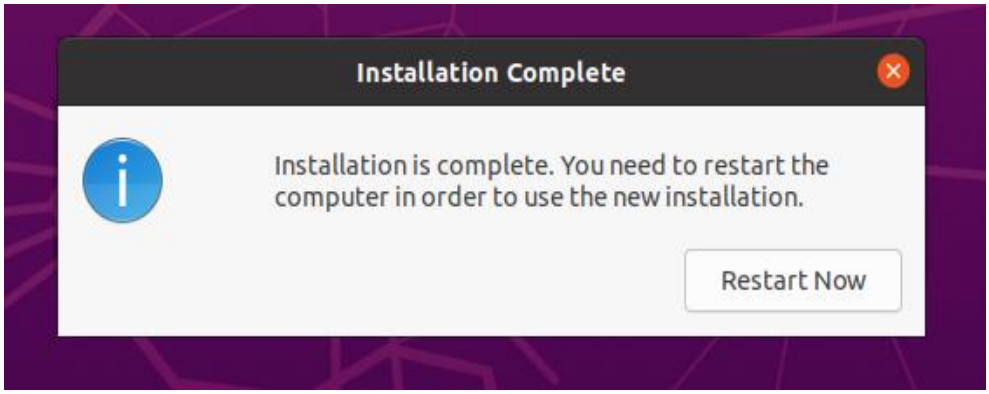


Figure 4.8 Restart Syst

- Login to your Desktop

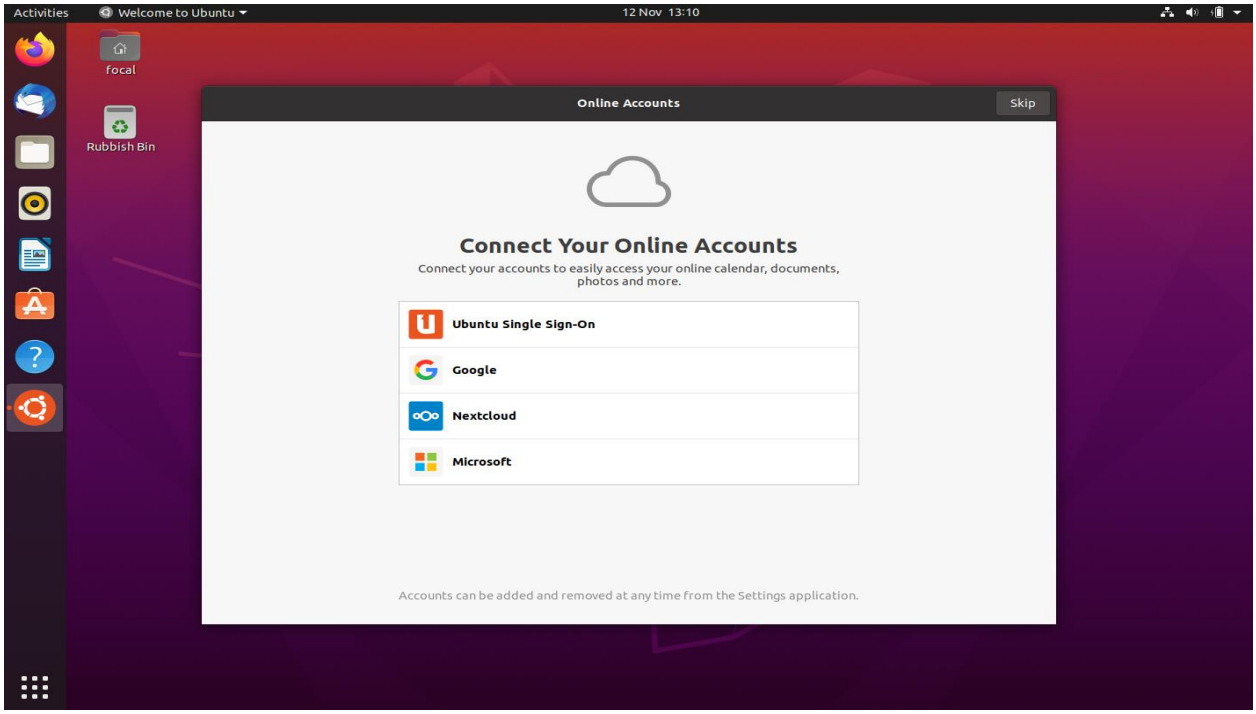


Figure 4.9 Ubuntu Window

4.3.2 ROS Installation

In this section we will go through ROS installation noetic version.

Setup your sources. List

Setup your computer to accept software from packages.ros.org.

```
• sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

Set up your keys

```
• sudo apt install curl # if you haven't already installed curl
• curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

Installation

First, make sure your Debian package index is up to date:

```
• sudo apt update
```

Now pick how much of ROS you would like to install.

- **Desktop-Full Install: (Recommended)** : Everything in **Desktop** plus 2D/3D simulators and 2D/3D perception packages

  ```
  o  sudo apt install ros-noetic-desktop-full
  ```

- **Desktop Install:** Everything in **ROS-Base** plus tools like rqt and rviz

  ```
  o  sudo apt install ros-noetic-desktop
  ```

- **ROS-Base: (Bare Bones)** ROS packaging, build, and communication libraries. No GUI tools.

  ```
  o  sudo apt install ros-noetic-ros-base
  ```

There are even more packages available in ROS. You can always install a specific package directly.

- `sudo apt install ros-noetic-PACKAGE`

example:

```
sudo apt install ros-noetic-slam-gmapping
```

To find available packages use:

```
apt search ros-noetic
```

Environment setup
You must source this script in every **bash** terminal you use ROS in.

```
source /opt/ros/noetic/setup.bash
```

It can be convenient to automatically source this script every time a new shell is launched.

These commands will do that for you.

If you have more than one ROS distribution installed, ~/.bashrc must only source

the setup.bash for the version you are currently using.

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

zsh

```
echo "source /opt/ros/noetic/setup.zsh" >> ~/.zshrc
source ~/.zshrc
```

Dependencies for building packages
Up to now you have installed what you need to run the core ROS packages. To create and manage your own ROS workspaces, there are various tools and requirements that are distributed separately. For example, rosinstall is a frequently used command-line tool that enables you to easily download many source trees for ROS packages with one command.

To install this tool and other dependencies for building ROS packages, run:

```
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
```

Initialize rosdep

Before you can use many ROS tools, you will need to initialize rosdep. rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS. If you have not yet installed rosdep, do so as follows.

```
sudo apt install python3-rosdep
```

With the following, you can initialize rosdep.

```
sudo rosdep init
rosdep update
```

For more information visit ROS Wiki kinetic installation.

4.4 Creating and Initializing a Workspace Folder

Noetic, a specific build system for ROS, is used. To utilize it, create and

establish a catkin workspace folder as shown below. this configuration only needs to be done once.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Now that we have created the catkin workspace folder, let's build it.

```
$ cd ~/catkin_ws/
$ catkin_make
```

Once you have finished building without errors, 'build' and 'devel' folders have been created. The build related files for the catkin build system are saved in the 'build' folder, and the execution related files are saved in the 'devel' folder.

Lastly, we will import the setting file associated with the catkin build system.

```
$ source ~/catkin_ws/devel/setup.bash
```

4.5 ROS Terminologies

This section explains the most frequently used ROS terms. It's preferable to be familiar with these concepts in order to know what exactly you are working with. You will become more familiar with the concepts as you engage with more and more practices.

ROS(Master , Core) :

ROS (Robot Operating System) master and roscore are closely related but they are not the same thing.

roscore is a command-line tool that initializes the ROS environment by starting up several core ROS nodes including the ROS Master, Parameter Server, and the rosout logging node. When you run the roscore command, it starts the ROS Master node as well as other essential nodes required for ROS communication.

The ROS Master is a crucial component of the ROS communication system. It is responsible for maintaining a directory of available ROS nodes, topics, and services, as well as providing a mechanism for nodes to find each other and communicate with each other.

The ROS Master keeps track of all the active ROS nodes in the system, and it also provides a central point for resolving naming conflicts. Whenever a new node is started, it registers with the ROS Master, which then adds the node to its registry of active nodes. The ROS Master also provides a centralized namespace for all nodes, topics, and services in the ROS system, which helps prevent naming conflicts between different nodes.

In summary, roscore is a tool used to start the ROS environment, which includes the ROS Master, Parameter Server, and the rosout logging node. The ROS Master, on the other hand, is a

critical component of the ROS communication system that maintains a directory of available ROS nodes, topics, and services and provides a mechanism for nodes to find and communicate with each other

4.5.1 Node

The smallest processor unit in ROS is referred to as a node. Consider it a single executable application. It is recommended by ROS to create a single node for each purpose and to develop for easy reusability. In the case of mobile robots, for example, the operating program is divided down into specific functions. Sensor drive, sensor data conversion, obstacle detection, motor drive, encoder input, and navigation are all handled by specialized nodes.

Upon startup, a node registers information such as name, message type, URI address and port number of the node. The registered node can act as a publisher, subscriber, service server or service client based on the registered information, and nodes can exchange messages using topics and services.

4.5.2 Package

A package is the basic unit of ROS. The ROS program is built as a set of packages, each of which contains either a configuration file for launching other packages or a set of nodes. All of the files required to operate the package are also included, including ROS dependent libraries for running various processes, datasets, and a configuration file.

4.5.3 Metapackage

A metapackage is a set of packages that have a common purpose. For example, the Navigation metapackage consists of 10 packages including AMCL, DWA, EKF, and map server.

4.5.4 Message

A node sends or receives data between nodes via a message. Messages are variables such as integer, floating point, and boolean. The message can contain a nested message structure that contains other messages or an array of messages. Messages are delivered via the TCPROS and

UDPROS communication protocols. Topic is used in unidirectional message delivery while service is used in bidirectional message delivery that request and response are involved.

4.5.5 Topic

The topic is similar to a conversation topic. The publisher node registers its topic with the master before attempting to publish messages on that topic. Subscriber nodes which want to receive the topic send a request to the publisher node with the name of the topic registered in the master.

The subscriber node connects directly to the publisher node using this information to exchange messages as a topic. For example, if the current position of the robot is generated in the form of odometry information by calculating the encoder values of both wheels of the mobile robot, the asynchronous odometry information can be continuously transmitted in unidirectional flow using a topic message (x, y,i).

4.5.6 Publish and Publisher

The action of sending relative messages corresponding to the topic is referred to as "publishing". The publisher node registers its own information and topic with the master and delivers a message to connected subscriber nodes which are interested in the same topic. The publisher is mentioned in the node, and it can be mentioned many times in the same node.

4.5.7 Subscribe and subscriber

The term ‘subscribe’ stands for the action of receiving relative messages corresponding to the topic. The subscriber node connects to the publisher node directly and receives messages from the associated publisher node. A subscriber is defined in the node and can appear numerous times in the same node.

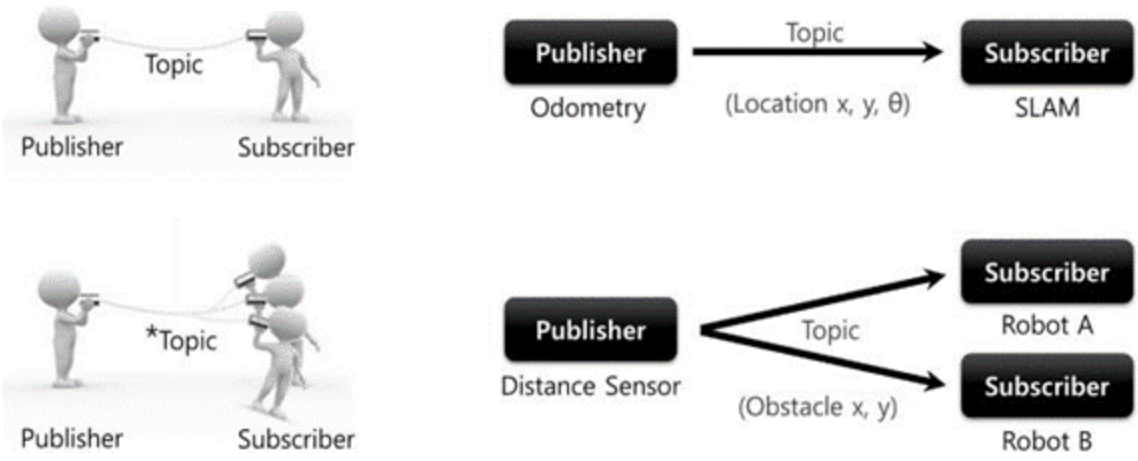


Figure 4.10 Topic Message Communication

4.5.8 Service

The service is synchronous bidirectional communication between the service client, which asks a service for a certain task, and the service server, which responds to requests.

4.5.9 Service Server

The ‘service server’ is a server in the service message communication that receives a request as an input and transmits a response as an output. Both request and response are in the form of messages.

4.5.10 Service Client

The ‘service client’ is a client in the service message communication that requests service to the server and receives a response as an input.

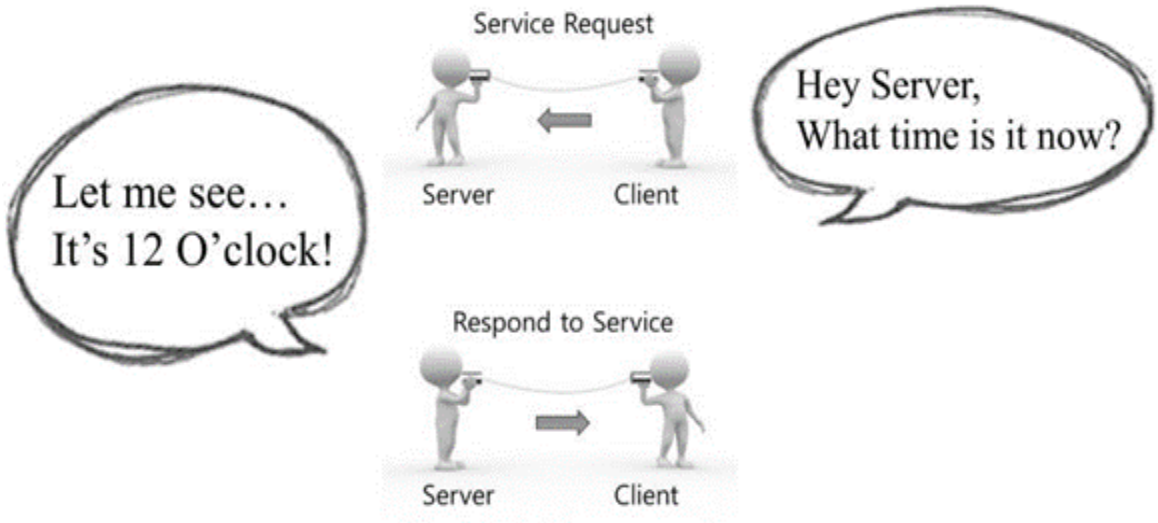


Figure 4.11 Service Message Communication

4.5.11 Action

Another message communication mechanism for asynchronous bidirectional communication is action. When responding to a request takes longer than expected and intermediate answers are required before the result is returned, action is utilized. Action files have a similar structure to service files. However, along with the goal and outcome data sections, which are represented as request and response in service, a feedback data component for intermediate responses has been included. The action client specifies the action's goal, and the action server executes the goal's action and delivers feedback and results to the action client.

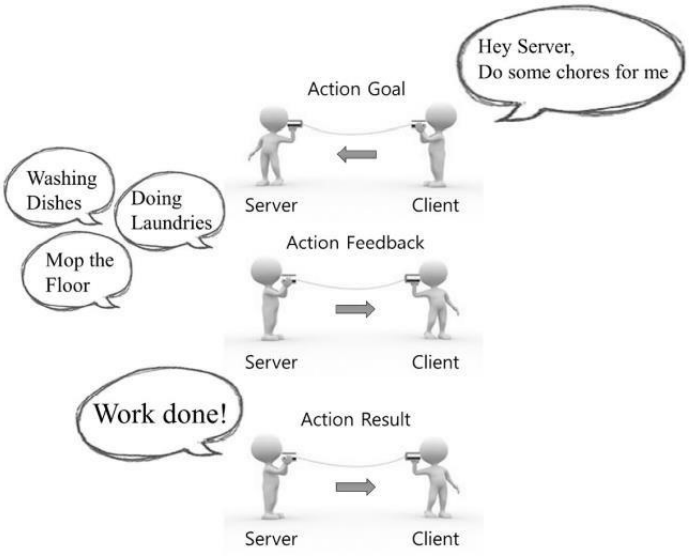


Figure 4.12 Action Message Communication

4.5.12 Parameter

It refers to parameter used in a node. You can think of it as a *.ini configuration file in windows programs. It has default values that can be read or written if necessary.

4.5.13 Catkin

Refers to the build system of ROS. The build system uses CMake (Cross Platform Make), and the build environment is described in the CMakeLists.txt file in the package folder.

4.5.14 Roscore

It is the command that runs the ROS master. It can be executed from another computer on the network if many computers are connected to it. Only one roscore should be running in the network, with the exception of special cases that support multiple roscore. The URI address and port number assigned to the ROS MASTER URI environment variables are used while ROS master is executing. If the user does not change the environment variable, the current local IP address is used as the URI address, and port 11311 is used as the master's default port.

- **Rosrun:**

It is referred to as the basic execution command for ROS. It is used to run a single node in the package.

- **Roslaunch:**

Compared to rosrun, roslaunch is used to run multiple nodes. roslaunch uses the '*.launch' file to define which nodes to be executed. The file is based on XML (Extensible Markup Language) and offers a variety of options in the form of XML tags.

- **ROS Wiki:**

Explains each package and the features provided by ROS.

4.6 Message Communication

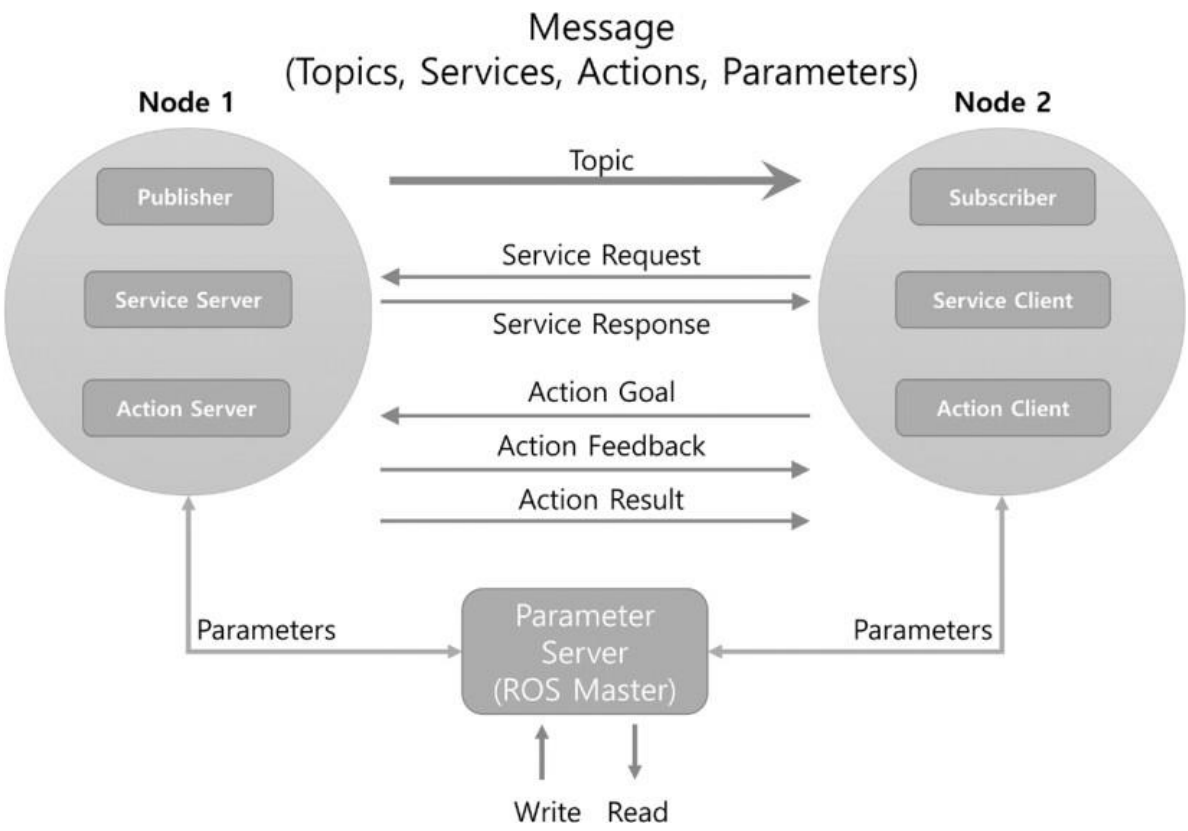


Figure 4.13 Message Communication Nodes

The publisher, the subscriber, the service server, the service client, the action server, and the action client can be implemented in separate nodes. The link between these nodes must first be created with the help of a master in order to exchange messages. A master functions similarly to a name server by storing node, topic, service, and action names, as well as the URI address, port number, and parameters. In other words, when nodes launch, they register their own information with the master and the master acquires relative information from other nodes. Then, for message communication, each node links directly to the others.

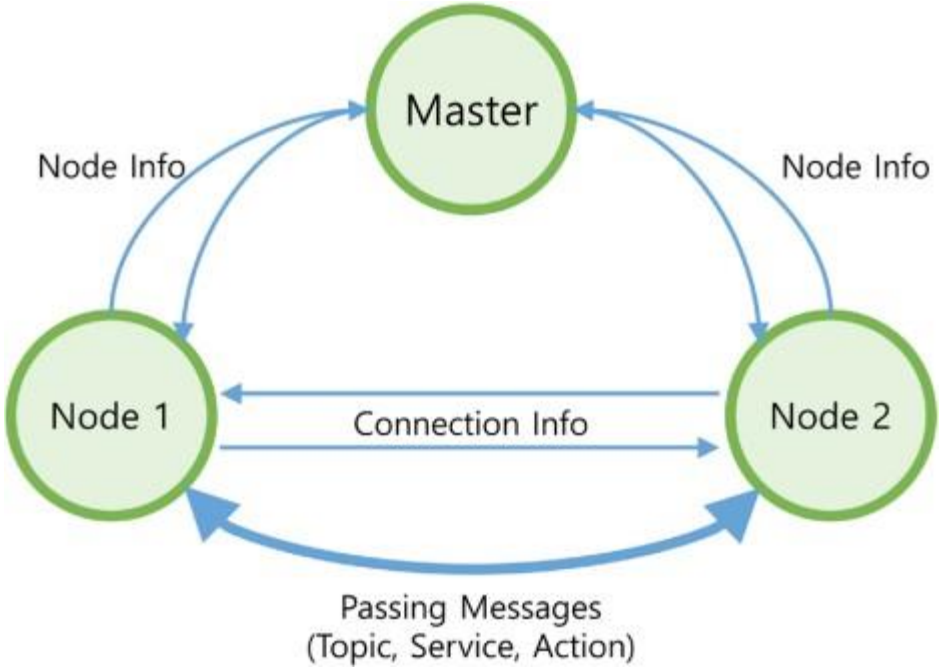


Figure 4.14 Message Communication

4.7 ROS Tools

There are various tools that can helps us using ROS. It is usually installed with ROS-full. The tools we are going to go through do not directly process a function in the ROS, but they are greatly useful supplementary tools for programming with ROS.

- Rviz
- Rqt_graph

4.7.1 3D Visualization Tool (Rviz)

RVIZ is a tool that allows you to visualize *Images, PointClouds, Lasers, Kinematic Transformations, RobotModels*...The list is endless. You even can define your own markers. It's one of the reasons why ROS got such a great acceptance. Before RVIZ, it was really difficult to know what the Robot was perceiving. And that's the main concept:

RVIZ is **NOT** a simulation. I repeat: It's **NOT** a simulation.
RVIZ is a representation of what is being published in the topics, by the simulation or the real robot.

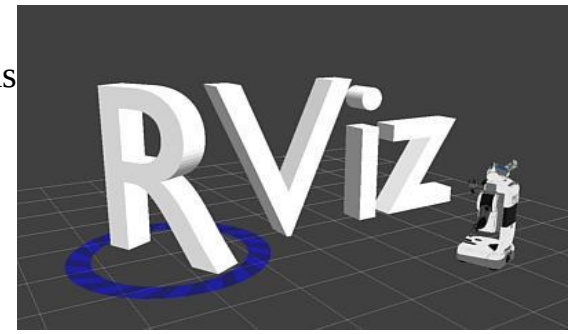


Figure 4.15 RViz

how to run rviz :

```
rosrun rviz rviz
```

you will see a window like this.

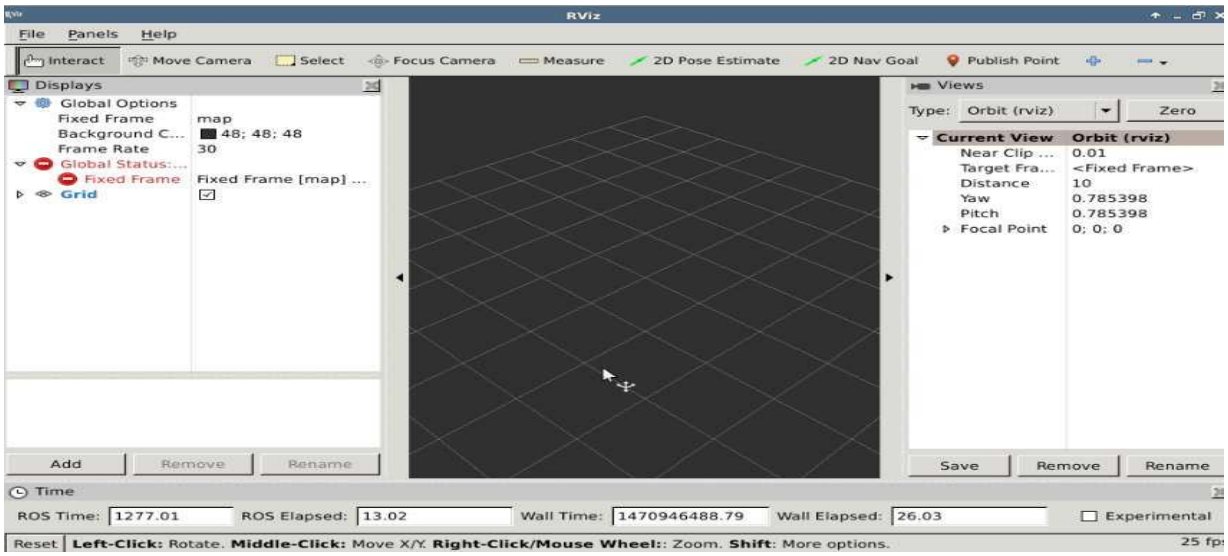


Figure 4.16 RViz example

4.7.2 ROS GUI Tool (rqt)

ROS provides various GUI tools for robot development. For example, there is a graphical tool that shows the hierarchy of each node as a diagram thereby showing the status of the current node and topic.

Rqt_graph is a tool that shows the correlation among active nodes and messages being transmitted on the ROS network as a diagram. This is very useful for understanding the current structure of the ROS network.
As an example, for the purpose of checking the nodes, let's run 'turtlesim_node' and 'turtle_teleop_key' in the 'turtlesim' package, and the 'uvc_camera_node' in the 'uvc_camera' package.

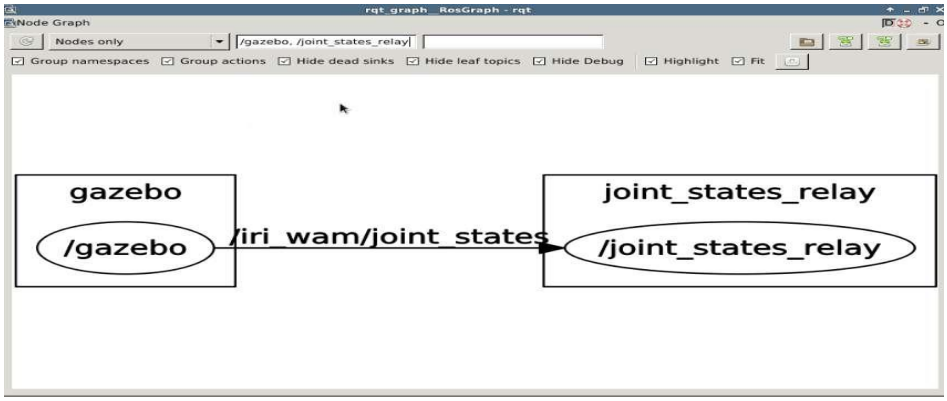


Figure 4.17 rqt_graph example

Chapter 5: Software

Overview

In this chapter, we will go through the circuit diagram and system architecture by explaining the different parts in the system and their classification. The system is classified into two parts depending on the task done.

System integration of Raspberry Pi 4 ,ESP32 and Arduinomega2560 and how they communicate by adjusting the connection between them by using rosserial.

Then, Explanation of the different packages that will be used like SLAM, navigation stack. Going through how Occupancy grid works and how gmapping creates a map.

After knowing the packages that will be used and how they work, we will go through the simulation and testing of the algorithms and tuning the parameters to achieve the best results.

5.1 System Architecture

This section describes the robot system architecture, visually.
presenting the connections between components, both hardware and software.

5.1.1 Introduction

a Raspberry Pi that calculates a route to a waypoint selected by a user and uses data read by the LIDAR for obstacle avoidance, and ESP32 that sends the appropriate power to the motors to follow the path generated by the Raspberry Pi and send IMU sensor and encoders readings to the Raspberry Pi for navigation correction. ArduinoMega2560 will be connected to other sensor's essential for the seed-sowing process and will send appropriate powers to the two motors responsible for the rotation and vertical motion of the drill.

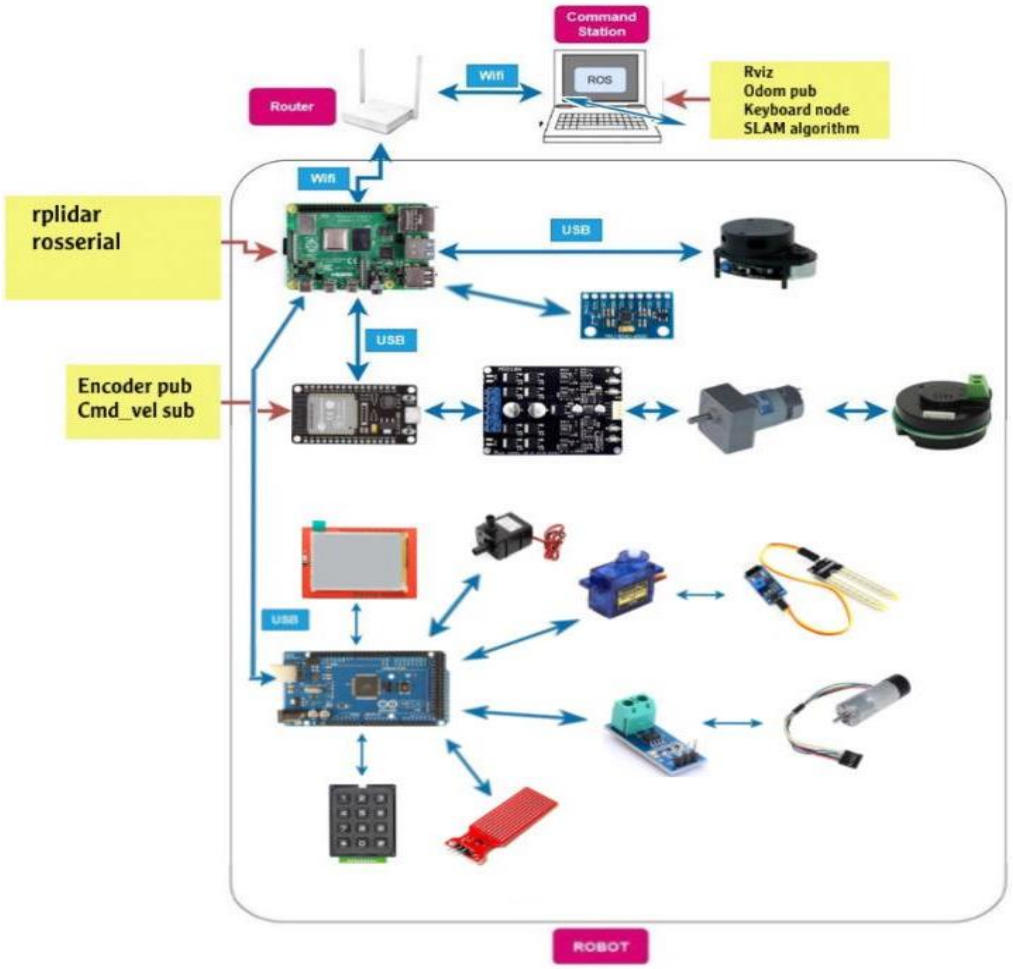


Figure 5.1 System configuration

5.2 Software classification

The Software can be split into two groups depending on the task carried out.

1. High Level tasks, such as Navigation and localization,
2. Low Level tasks, such as sending data to motors and receive sensor data.

As particle filtering and similar localization techniques are computationally intensive, the low-level task was moved into a separate microcontroller This creates two separate codebases: the low level running on ESP32 and ArduinoMega2560 microcontroller, and the high level running on personal computer.

The high level interacts with the user and runs the algorithms for mapping, navigation and localization. The low level receives sensor readings from motor encoders and IMU which controls motor actuation based on commands sent from the high level.

The high level codebase consists of ROS nodes that handles different functionalities of the autonomous system. Each node is a separate system process which communicate to each other.

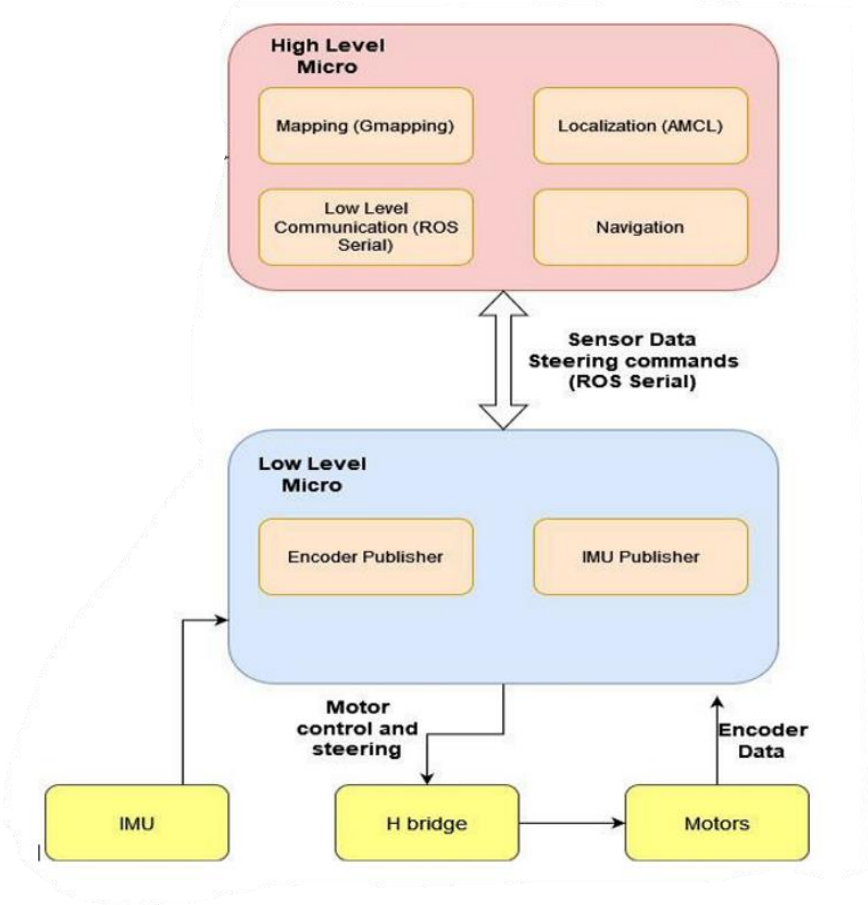


Figure 5.2 System Diagram

5.3 Hardware software integration

The Raspberry Pi communicates with the sensors through a ESP32 and ArduinoMega2650. It handles calculating the path using ROS navigation stack, while the microcontrollers handles supplying the motor with the pwm needed to follow that path and perform other opertations. The navigation stack containing the current robot velocity is converted into left and right motor target velocities and then publish the target velocities into the respected ROS topics for left and right wheels.

The ESP32 uses subscription template functions from the ROS library that reads the data published to the left and right wheels topics. When the subscription functions detect a message, they trigger a callback function that applies the motor pwm and direction. Using the encoders on the motors, the ESP32 reads and publishes the number of encoder pulses from the start of tracking to the right and left motor topics. The PID velocity node subscribes to the right and left motors topic and uses the encoder pulse values to correct the power output back to the motors. The ArduinoMega2650 done the same thing with the motors that are controlling the drill but in a different way as after the robot reaches the specific location the Arduino will send appropriate power signals to the motors to done the drilling process and stops the operation when overcurrent is detected.

5.4 Rosserial

Rosserial is a ros communication protocol which is a point to point protocol for wrapping standard ROS serialized messages and multiplexing multiple topics and services over. It is primarily used for integrating microcontrollers (Arduino) into ros. Ros serial consists of libraries for use with microcontrollers and nodes for the PC side. We used rosserial to connect the ArduinoMega2650 and ESP32.

5.5 SLAM (simultaneous localization and mapping)

SLAM is a straightforward and common problem which is spatial exploration. You enter an unknown space, study it, and move around within it; you create a model of it, and you know where you are in it.

Then you can figure out how to go to any area of the space, as well as how to exit it.

It is a technology for autonomous vehicles that allows you to create a map while also locating your vehicle within that map. The car can map out unknown environments using SLAM methods. Engineers use the map data to perform activities like route planning and obstacle avoidance.

Consider a robotic vacuum cleaner for your home. It will simply travel around a room at random without SLAM and may not be able to clean the full floor surface.[24] Furthermore, because this method consumes a lot of power, the battery will drain faster. Robots with SLAM, on the other hand, can determine the amount of movement required based on information such as the number of wheel revolutions and data from cameras and other imaging sensors. This is referred to as localization. The robot can also use the camera and other sensors to develop a map of the obstacles in its environment so that it doesn't have to clean the same spot twice.

This is referred to as mapping.

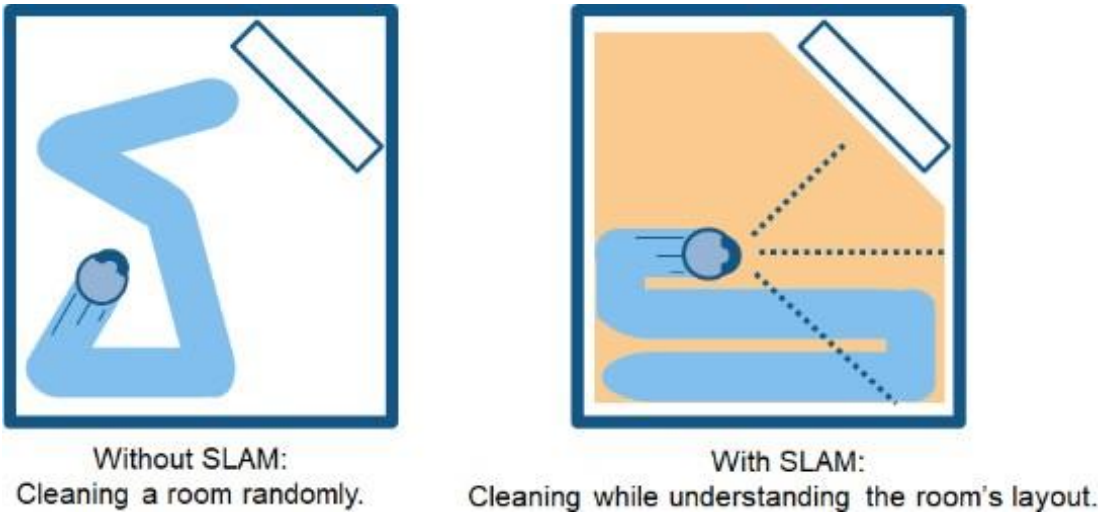


Figure 5.3 Benefit of SLAM

How it Works

To achieve SLAM, two sorts of technological components are utilized. The first is sensor signal processing, which includes front-end processing and is heavily reliant on the sensors employed. Pose-graph optimization, which includes sensor-agnostic back-end processing, is the second type.

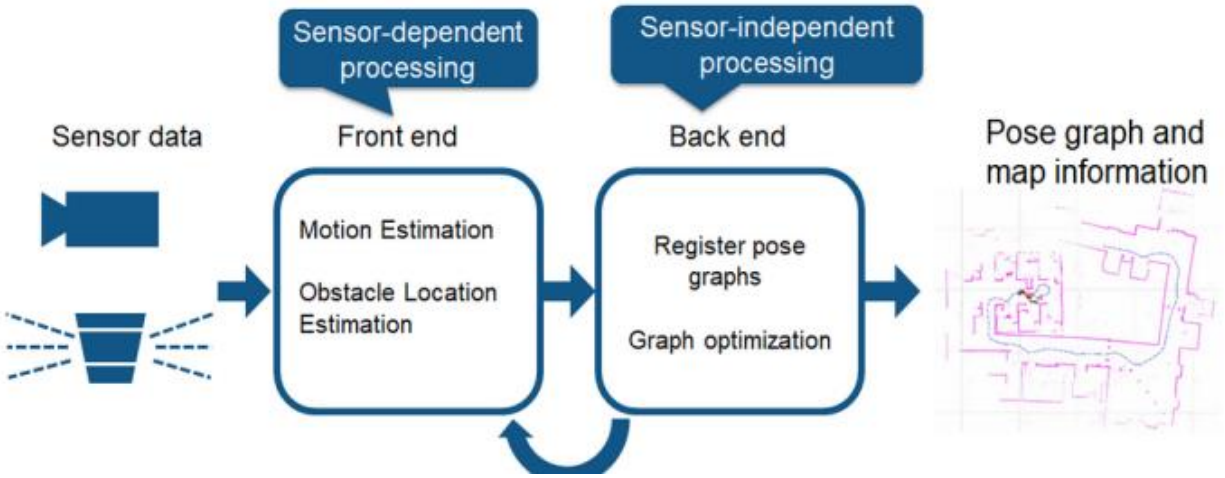


Figure 5.4 SLAM Process Flow

We will be using a package called Gmapping, which is based on RaoBlackwellized particle filter to learn grid maps from laser range data. It has an effective means to solve the simultaneous localization and mapping (SLAM) problem as it was proposed as a viable solution to SLAM problems.

This method employs a particle filter, with each particle carrying its own map of the environment. Each particle in a RBPF represents a possible robot trajectory and a map.

In a Rao-Blackwellized particle filter for learning grid maps, it is adaptive strategies for reducing the number of particles where it is a method for calculating an accurate proposal distribution that considers both the robot's movement and the most recent observation. This reduces the uncertainty about the robot's pose during the filter's prediction step. Then a method for selectively performing re-sampling procedures that significantly minimizes the number of operations required.

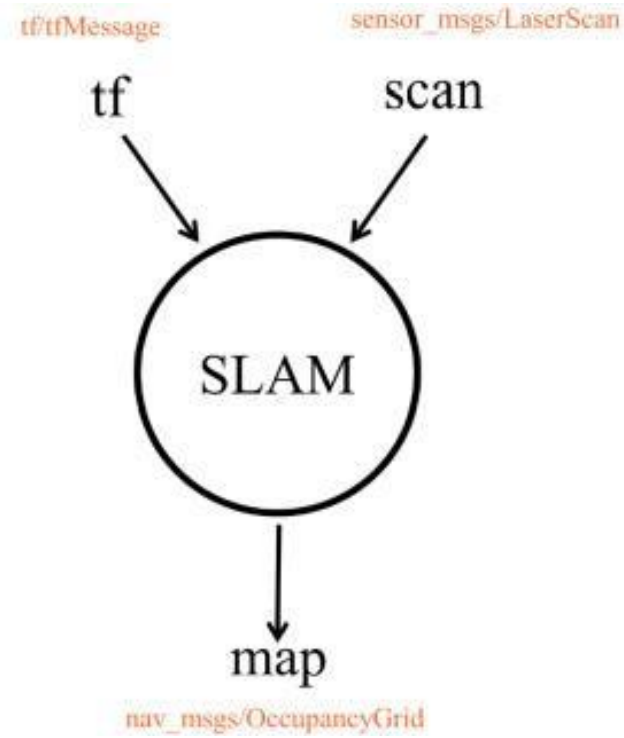


Figure 5.5 laser effect on OGM

The map made by Gmapping is grid based which is called occupancy grid map, The basic idea of the occupancy grid is to represent a map of the environment as an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment. Occupancy grid algorithms compute approximate posterior estimates for these random variables.

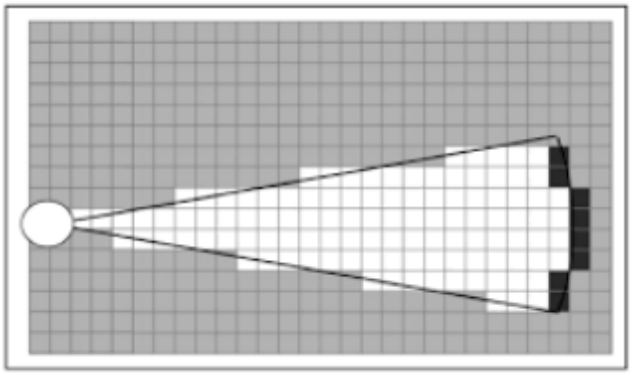


Figure 5.6 Occupancy Grid

5.6 Transform Configuration

Many ROS packages require the tf software library to be used to publish a robot's transform tree. A transform tree defines offsets in terms of translation and rotation between distinct coordinate frames.

Consider the case of a simple robot with a mobile base and a single laser positioned on top of it. There are two coordinate frames in relation to this robot, one for the center of the body(base link) and other for the laser positioned at the top of the body(base laser).

other for the laser positioned at the top of the body(base_laser).

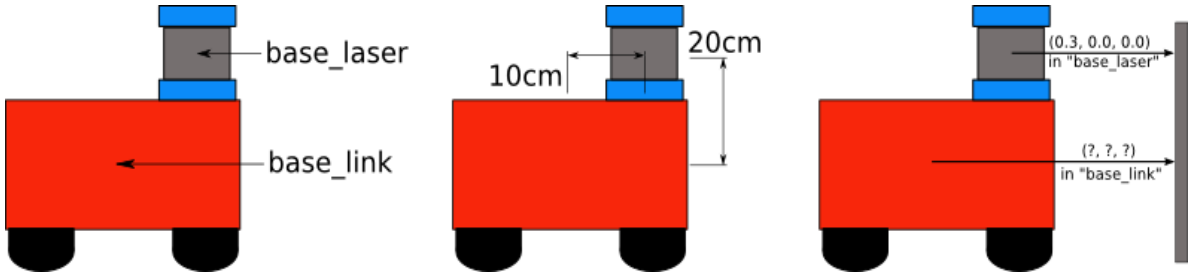


Figure 5.7 TF Example

At this point, let's assume that we have some data in the "base_laser" coordinate frame. In order to take this data and use it to help the mobile base avoid obstacles in the world. we need a way of transferring the laser scan we've received from the "base_laser" frame to the "base_link" frame. In essence, we need to define a relationship between the "base_laser" and "base_link" coordinate frames.[30]

When specifying this connection, assume the laser is set 10cm forward and 20cm above the mobile base's center point. The "base link" frame is now linked to the "base laser" frame by a translational offset.[30] We know that to receive data from the "base link" frame to the "base laser" frame, we must apply a translation of (x: 0.1m, y: 0.0m, z: 0.2m), and the reverse to get data from the "base laser" frame to the "base link" frame (x: -0.1m, y: 0.0m, z: -0.20m).

We could take care of this relationship by storing and applying the necessary translations across the frames as needed, but as the number of coordinate frames grows, this becomes a significant nuisance.

Fortunately, we won't have to undertake any of this labour ourselves. Instead, we'll use tf to establish the relationship between "base link" and "base laser" and let it take care of the transformation between the two coordinate frames.

To define and store the relationship between the "base_link" and "base_laser" frames using tf, we need to add them to a transform tree. Tf uses a tree structure to guarantee that there is only a single traversal that links any two coordinate frames together and assumes that all edges in the tree are directed from parent to child nodes

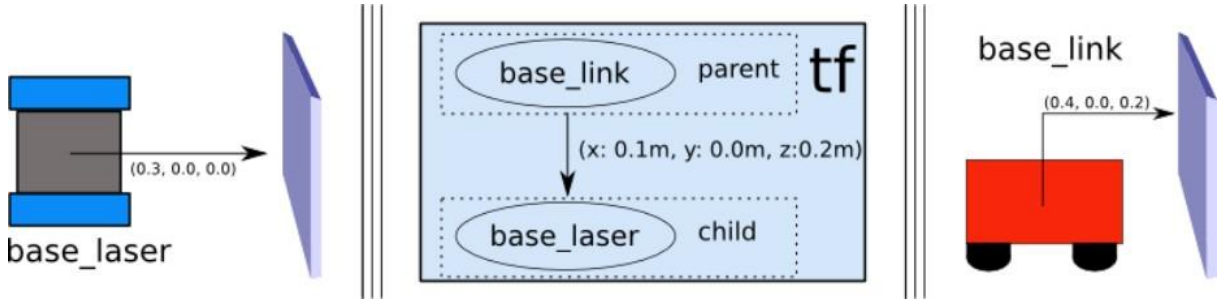


Figure 5.8 TF Tree illustration

In this example, there are only two nodes, one for the "base_link" frame and one for the "base_laser" frame. We must first pick which node will be the parent and which will be the child in order to establish the edge between them. Remember that tf expects that all transformations flow from parent to child, thus this distinction is critical.

5.7 Publishing odometry information

In this section we will provide an example of publishing odometry information in order to be used in the navigation stack. It will cover both publishing the nav_msgs/Odometry message over ROS, and a transform from a "odom" coordinate frame to a "base_link" coordinate frame over tf.

The navigation stack makes use of tf to figure out where the robot is in the world and how sensor data is related to a static map. Tf on the other hand, does not provide any information regarding the robot's velocity. As a result, according to the navigation stack, any odometry

source must publish both a transform and a nav_msgs/Odometry message over ROS that contains velocity information.

5.7.1 Nav_msgs/Odometry Message

It stores an estimate of the position and velocity of a robot in free space.

- The pose in this message should be specified in the coordinate frame given by header. frame_id.
- The twist in this message should be specified in the coordinate frame given by the child_frame_id

The pose in this message corresponds to the estimated position of the robot in the odometry frame along with an optional covariance for the certainty of that pose estimate. The twist in this message corresponds to the robot's velocity in the child frame, normally the coordinate frame of the mobile base, along with an optional covariance for the certainty of that velocity estimate.

5.7.2 Using tf to publish an odometry transform.

In a transform tree, the "tf" software library is in charge of managing the connections between coordinate frames relevant to the robot. As a result, any odometry source that handles a coordinate frame must publish data about it. To View the frames of your current transform tree.

```
$ rosrun tf2_tools view_frames.py
```

Which creates a PDF graph of your TF.

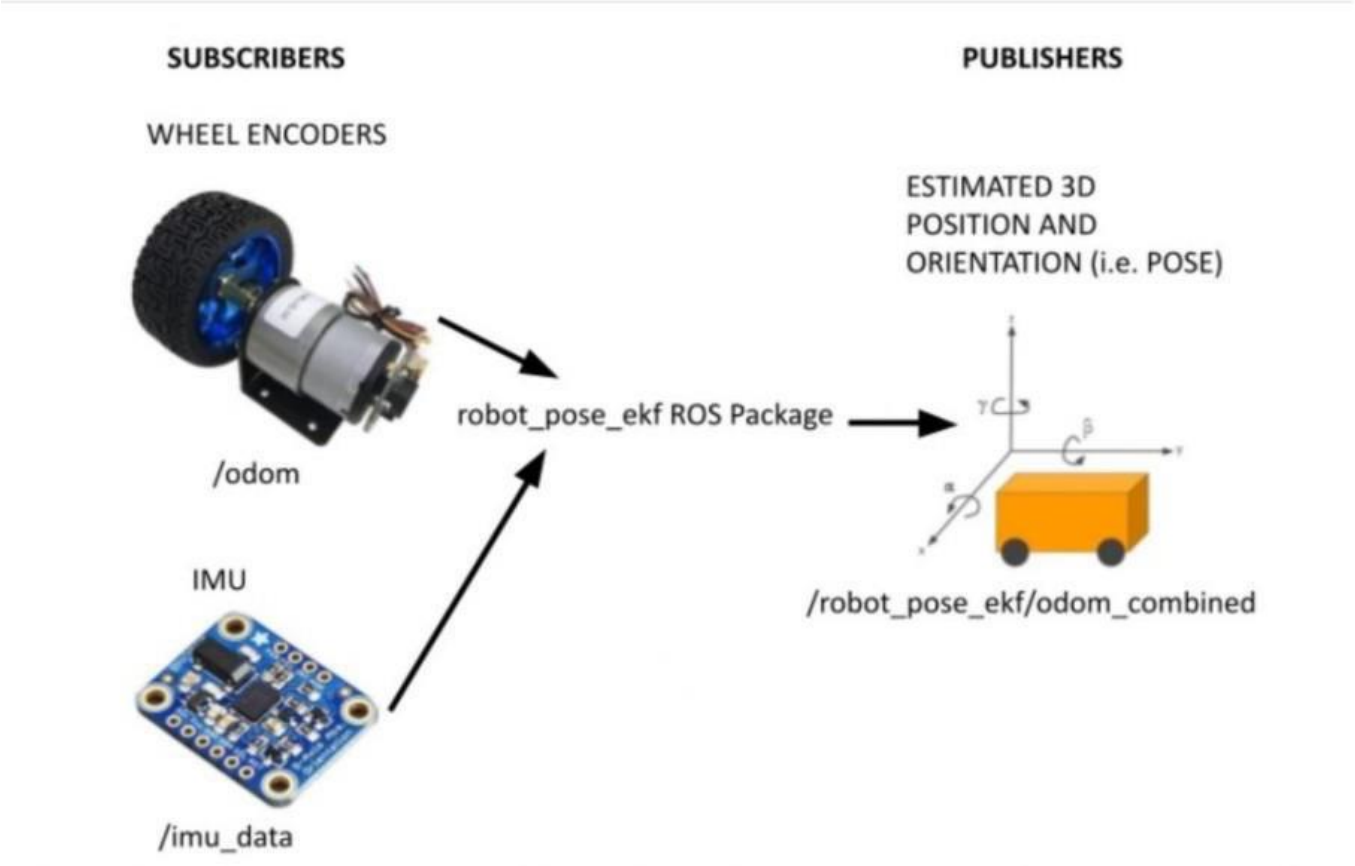
5.8 Sensor fusion

Each sensor in the world has its strengths and weaknesses, so basically sensor fusion is a process of combining multiple sensor readings to bring out the best estimation other than using a single sensor that can be subjected to noise and errors while working due to environment. The resulting model from the sensors used together will be more accurate because it balances the strengths of the sensors used. A vehicle could use sensor fusion to fuse information from multiple sensors of the same type as well. This improves

perception by taking advantage of partially overlapping fields of view.

In our project we will be using an incremental encoder attached to the motor's shaft that sends a feedback signal that can be used to determine position, count, speed, or direction, as well as an IMU that is used to measure acceleration, angular velocity and determine orientation in real time.

The imu measures the position by taking the double integral of the acceleration of over the course of time ($x = \iint a\, dt$). The readings of the imu contain noise which affect the accuracy of the estimation which can accumulate the error over time. The encoder measures the position by calculating the distance moved from the prior position by taking the integral of the velocity ($x = \int v\, dt$), but the readings are subjected to error



from slippage or the wheels being stuck.

Figure 5.9 Sensor fusion

Kalman filter is an algorithm that uses a series of measurements observed over real time (containing noise and other inaccuracies) and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone. The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error) is observed, the next step is the correction of these estimates, which are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

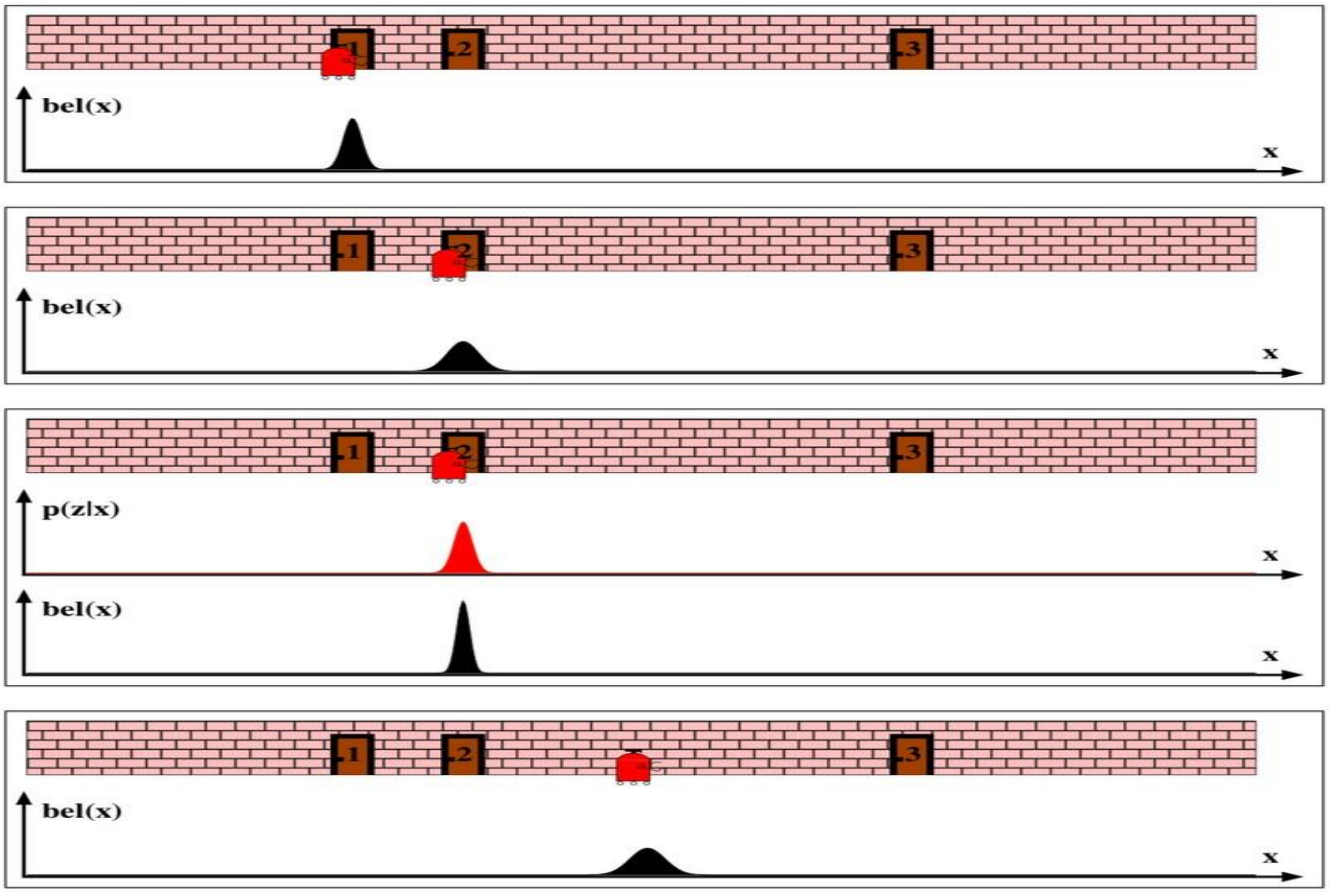


Figure 5.10 EKF example

At time:

- **t=1**, Initial belief represented by a Gaussian distribution around the first door.
- **t=2**, Motion is updated and the new belief is represented by a shifted Gaussian of increased weight.
- **t=3**, Measurement is updated and the robot is more certain of its location. The new posterior is represented by a Gaussian with a small variance.
- **t=4**, Motion is updated and the uncertainty increases.

5.9 Navigation Stack

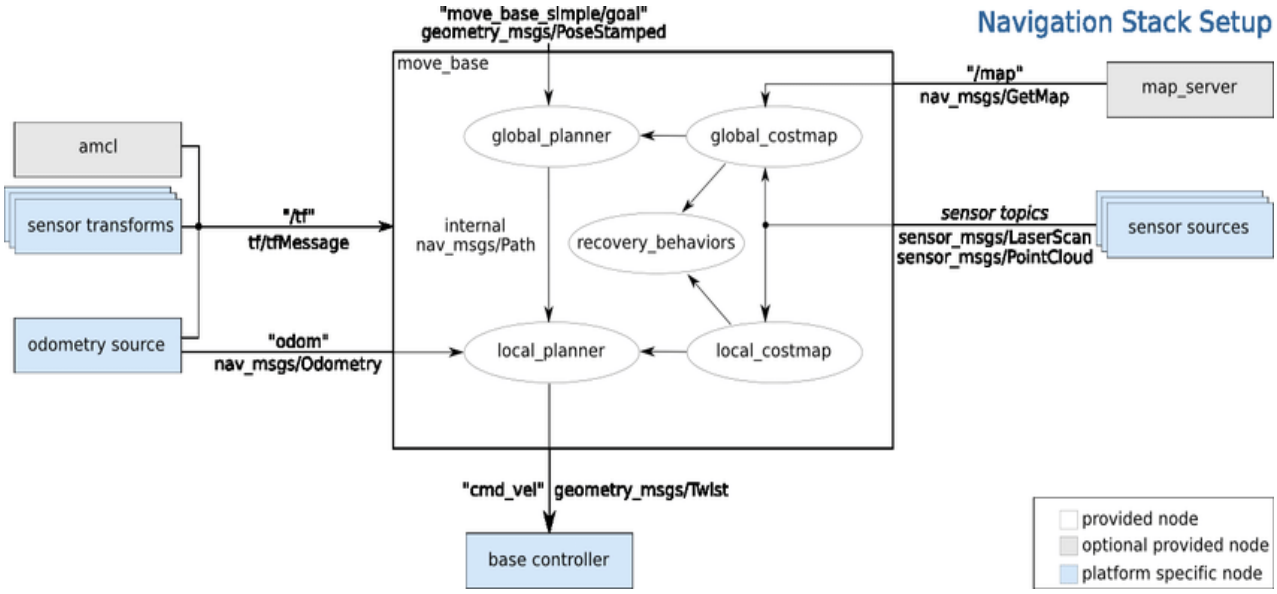


Figure 5.11 Navigation Stack Layout

Navigation in robotics is inseparable and essential. Navigation is the movement of the robot to a defined destination. But it is important to know where the robot itself is located and to have a map of the given environment. It is also important to find an optimized and short path among the various routing options, and to avoid obstacle. It is not an easy mission, luckily the navigation stack is present and makes things easier. The navigation stack assumes that the robot is set to run in a specific way. This setup is depicted in the diagram above. The white components are already implemented required components, the grey components are already

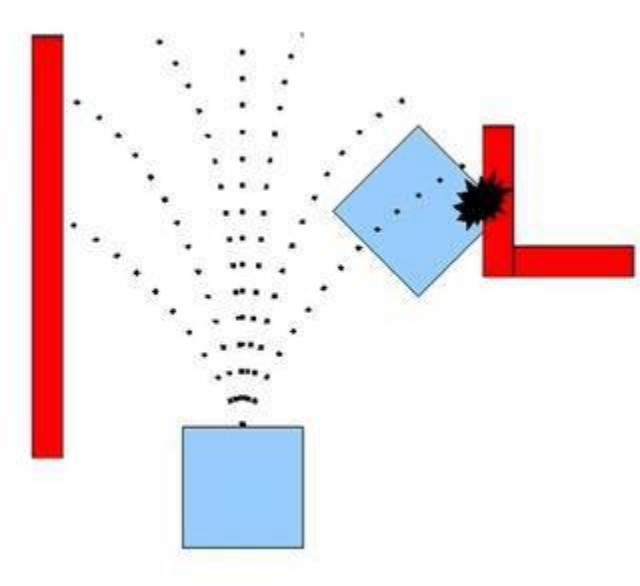
implemented optional components, and the blue components must be constructed for each robot platform.

The requirements of Navigation stack are Odometry (pose + Orientation) and sensor readings in order to autonomously navigate in the given map and avoid obstacles.

5.9.1 Move Base

Move_base node is the most important thing in the stack, where it is based on Planning. Which is basically creating a path and then following it. It is responsible for planning a collision-free path from a starting location to a goal location for amobile robot.

To generate a path and avoidobstacles, It first deduces several possibilities and gives them scores, and based on those scores it chooses the optimal path to the goal, then it repeats this operation over and over again.



There are two main planners in the navigation stack which are:

- Global Planner
- Local Planner

Figure 5.12 Robot Planning

5.9.2 Global Planner

Generally, a pathfinding

method examines a graph by starting at one location and examining nearby points until the destinations are reached, with the goal of finding the cheapest route.

Although approaches such as breadth-first search would locate a route if given enough time, there are other methods that would get to the goal sooner. Pathfinding has two fundamental problems. The first is to discover a path between two nodes in an environment. The second problem is the shortest path problem, which entails determining the shortest path. Compared to algorithms like breath-first search which tends to search for every possibility to reach the goal node which is a highly iterative and intensive operation so, algorithms such as A* and Dijkstra tactically eliminate paths, either by heuristics or through dynamic programming.

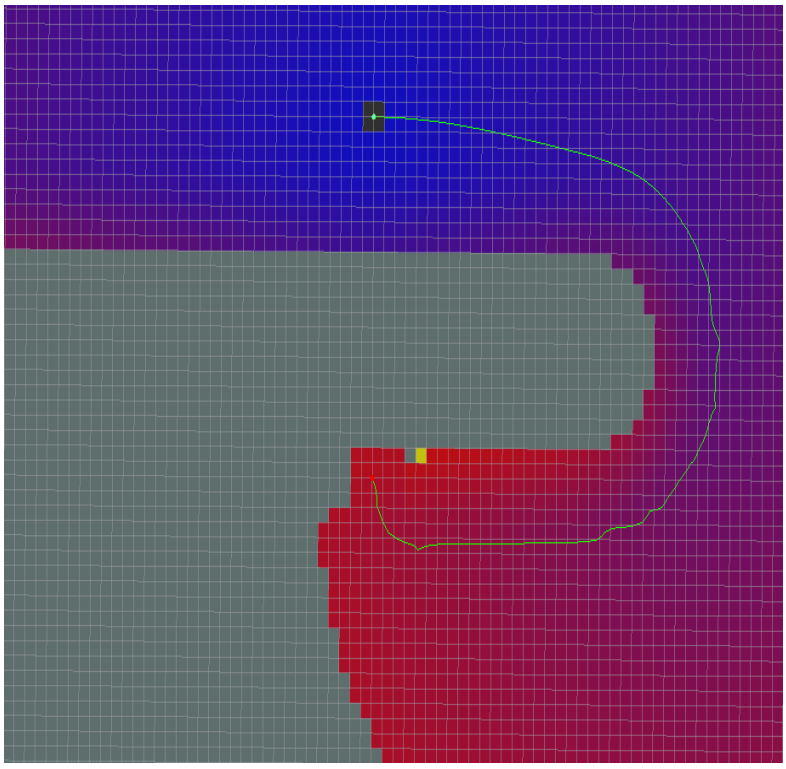


Figure 5.13 Path Planning

5.9.2.1 Dijkstra's Algorithm

It is an example of a graph-based pathfinding, It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined.

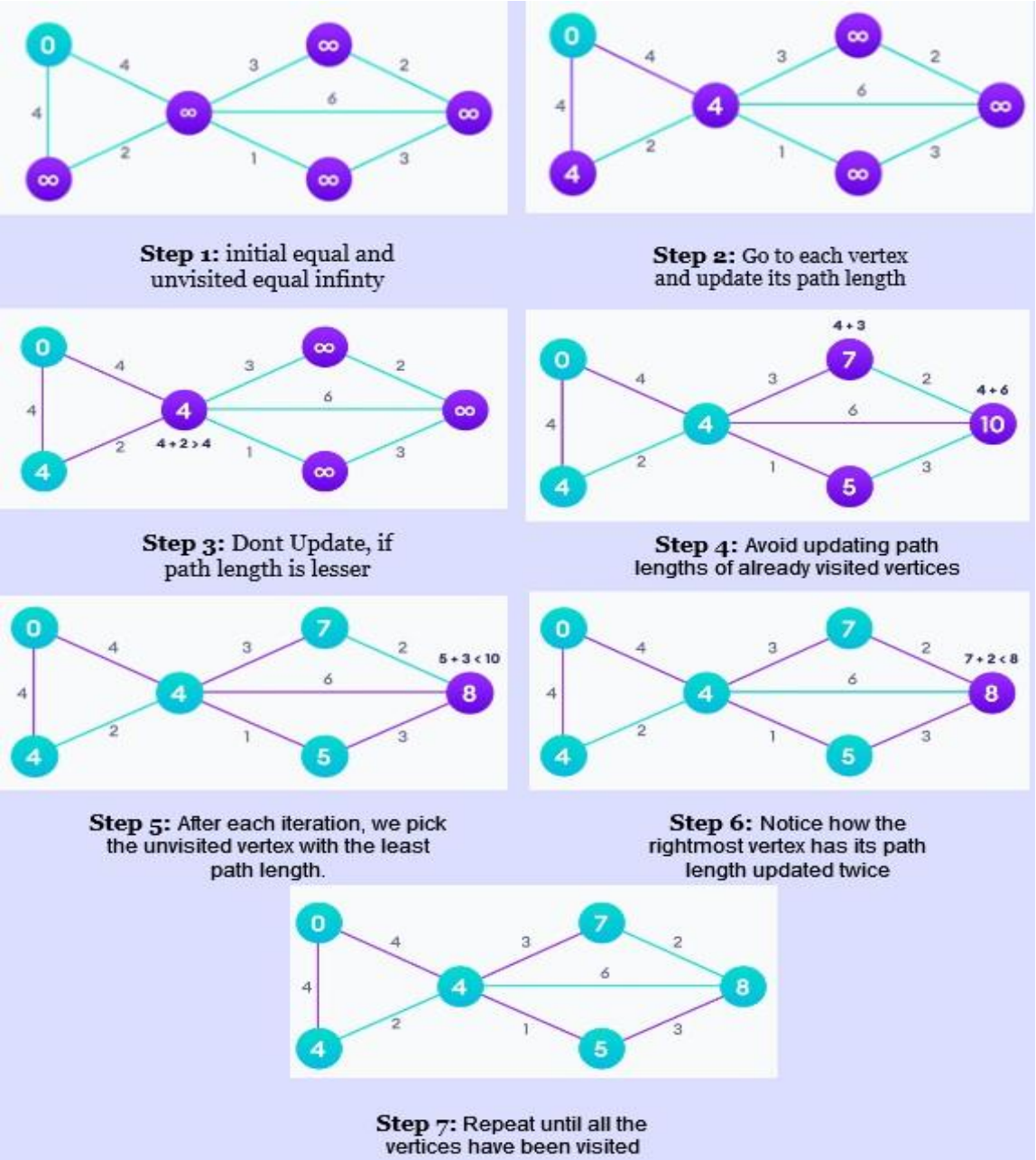


Figure 5.14 Steps for path panning using Dijkstra's Algorithm

Assume you want to discover the shortest route between two city map intersections: a starting point and a destination. The distance (from the beginning location) to every other node on the map is initially marked with infinity by Dijkstra's algorithm. This is done to indicate that certain nodes have not yet been visited, not to imply that there is an infinite distance. The current node will be the beginning point for the first iteration, and the distance to it will be zero.

The current node will be the closest unvisited node to the beginning point in following cycles.

Update the distance to every unvisited node that is directly connected to the current node.

This is accomplished by calculating the sum of the distance between an unvisited node and the current node's value, and then relabeling the unvisited intersection with this value (the sum) if it is less than the current value of the unvisited node. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths.

After you have updated the distances to each neighboring node, mark the current node as visited and select an unvisited node with minimal distance (from the starting point) as the current node. nodes marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned to.

Continue updating the nearby nodes with the shortest distances, marking the present node as visited, and moving on to the next nearest unvisited node until the destination is marked as visited.

Once you have marked the destination as visited, you have determined the shortest path to it from the starting point and can trace your way back following the arrows in reverse.

This algorithm makes no attempt of direct "exploration" towards the destination as one might expect. Rather, the sole consideration in determining the next "current" intersection is its distance from the starting point. This algorithm therefore expands outward from the starting point, interactively considering every node that is closer in terms of shortest path distance until it reaches the destination. When understood in this way, it is clear how the algorithm necessarily finds the shortest path. However, it may also reveal one of the algorithm's weaknesses: its relative slowness in some topologies.

5.9.3 Local Planner

Once the global planner has calculated the path to follow, this path is sent to the local planner. The local planner, then, will execute each segment of the global plan (let's imagine the local plan as a smaller part of the global plan). So, given a plan to follow (provided by the global planner) and a map, the local planner will provide velocity commands in order to move the robot.

Unlike the global planner, the local planner monitors the odometry and the laser data, and chooses a collision-free local plan (let's imagine the local plan as a smaller part of the global plan) for the robot. So, the local planner can recompute the robot's path on the fly in order to keep the robot from striking objects, yet still allowing it to reach its destination.

Once the local plan is calculated, it is published into a topic named /local_plan. The local planner also publishes the portion of the global plan that it is attemting to follow into the topic /global_plan

The basic idea of the Dynamic Window Approach (DWA) algorithm is as follows:

1. Discretely sample in the robot's control space (dx,dy,dtheta)
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).
4. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
5. repeat.

5.9.4 Costmap configuration

In the navigation stack, the obstacles in the environment are stored inside to map called as costmaps, one for the global planning which is used for creating long-term plans of the environment and the other is local planning which is used for obstacle avoidance. The way of configuration will go as follows first a general configuration for both the costmaps, a configuration specific for the global costmap and local costmap individually Therefore, there are three sections for costmap configuration: common configuration options, global configuration options, and local configuration options.

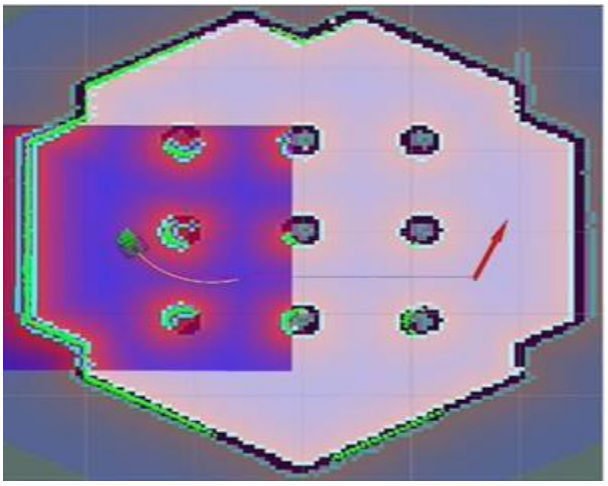


Figure 5.15: Global and Local costmaps

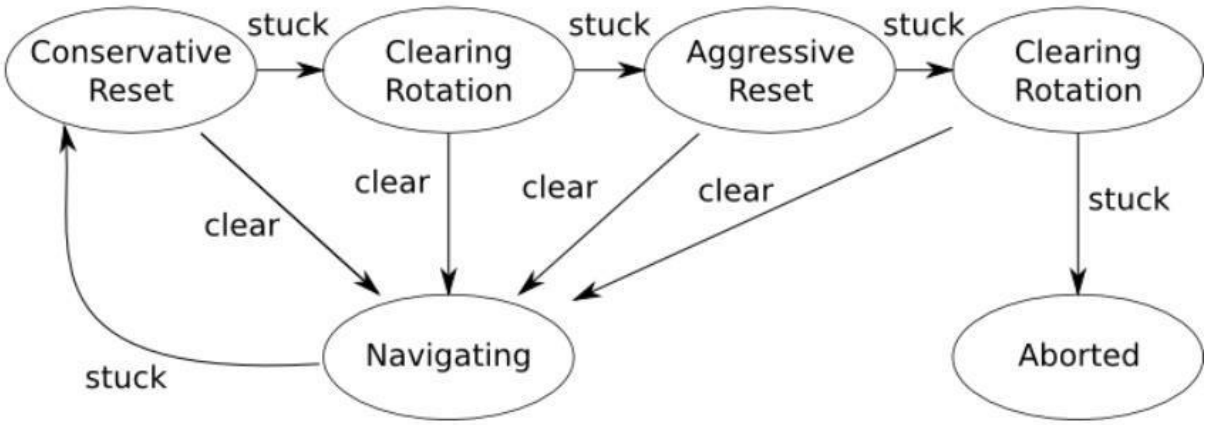


Figure 5.16 Recovery behavior

When the robot believes itself to be stuck, the move base node may undertake optional recovery behaviors. To clear out space, the move base node will do the following steps by default:
At the start, the robot's map will be cleansed of obstacles outside of a user-specified region. The robot will then, if possible complete an in- place revolution to clear space. If it fails, it will attempt to clear the map even more aggressively by removing all the obstacles outside the rectangular region where it may rotate in position, then it will attempt another in place rotation. If all fails the robot will declare that it's impossible to reach the user specified goal. The recovery behavior can specifically be configured or disabled using the recovery behavior enabled parameter.

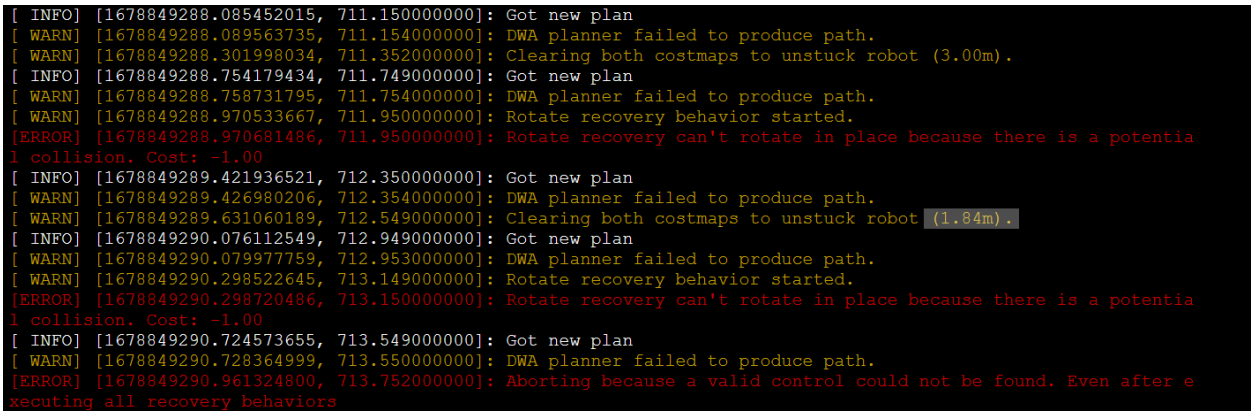


Figure 5.17 Recovery behavior error

5.9.5 AMCL (Adaptive Monto Carlo Localization)

Amcl is a ROS package that deals with robot localization. It is also known as particle filter localization. The algorithm uses a particles to represent the distribution of likely states, A particle can be considered as a virtual element that resembles the robot, i.e., a hypothesis of where the robot is. The algorithm estimates the pose of a robot as it moves and senses the environment. The algorithm typically starts with a uniform random distribution of particles over the configuration space, implying that the robot has no information about its location and assumes it is equally likely to be anywhere in space. Each sample has a position and orientation data representing the likelihood of the robot's pose.

Particles are all sampled randomly initially. The robot updates its belief as it moves and detects objects in the environment the particles are resampled using recursive Bayesian estimation as a result it converges the particles together as it predicted the pose a bit better, and as the process continues and the robot keeps moving through the environment the particles start to accurately estimate the actual pose of the robot, In addition for each particle estimated it has a pose estimation to the actual robot pose, as well as it is assigned with a weight determining how close the estimated pose is to the true state. At its most basic level, the AMCL software keeps track of a probability distribution across all potential robot positions and updates it with data from odometry and laser scans. The package depthimage to laserscan, which takes in a depth stream and broadcasts a laser scan on sensor msgs/LaserScan, is used to generate these 2D laser scans. The filter is "adaptive" because the number of particles in the filter is dynamically adjusted: when the robot's pose is very uncertain, the number of particles is increased; when the robot's pose is well known, the number of particles is reduced. This allows the robot to choose between processing speed and precision in localization. In the case of a mobile robot, each particle is represented as a particle = pose (x, y, i), weight, and each particle is an arbitrary small particle representing the estimated position and orientation of the robot expressed by x, y, and i of the robot and the weight of each particle. Even though the AMCL package works well out of the box, there are a number of parameters that may be tweaked dependent on the platform and sensors used. Configuring these parameters can improve the AMCL package's performance and accuracy while reducing the number of recovery rotations the robot makes while navigating.

MCL maintains two probabilistic models, a motion model and a measurement model. In ROS amcl, the motion model corresponds to a model of the odometry, while the measurement model corresponds to a model of laser scans. Given a map of the environment, the goal of the algorithm is for the robot to determine its pose within the environment. At every time t the algorithm takes as input the previous belief Xt-1 = { xt-1 [1] , xt-1 [2] , ..., xt-1 [M]} , an actuation command ut, and data received from sensors zt; and the algorithm outputs the new belief Xt.

```
Algorithm MCL(X_{t-1}, u_t, z_t)
    x̄_t = X_t = ∅
    for m = 1 to M:
      x_t^[m] = motion_update(u_t, x_{t-1}^[m])
      w_t^[m] = sensor_update(z_t, x_t^[m])
      x̄_t = x̄_t + (x_t^[m], w_t^[m])
      endfor
      for m = 1 to M:
        draw x_t^[m] from x̄_t with probability α w_t^[m]
        X_t = X_t + x_t^[m]
      endfor
      return X_t
```

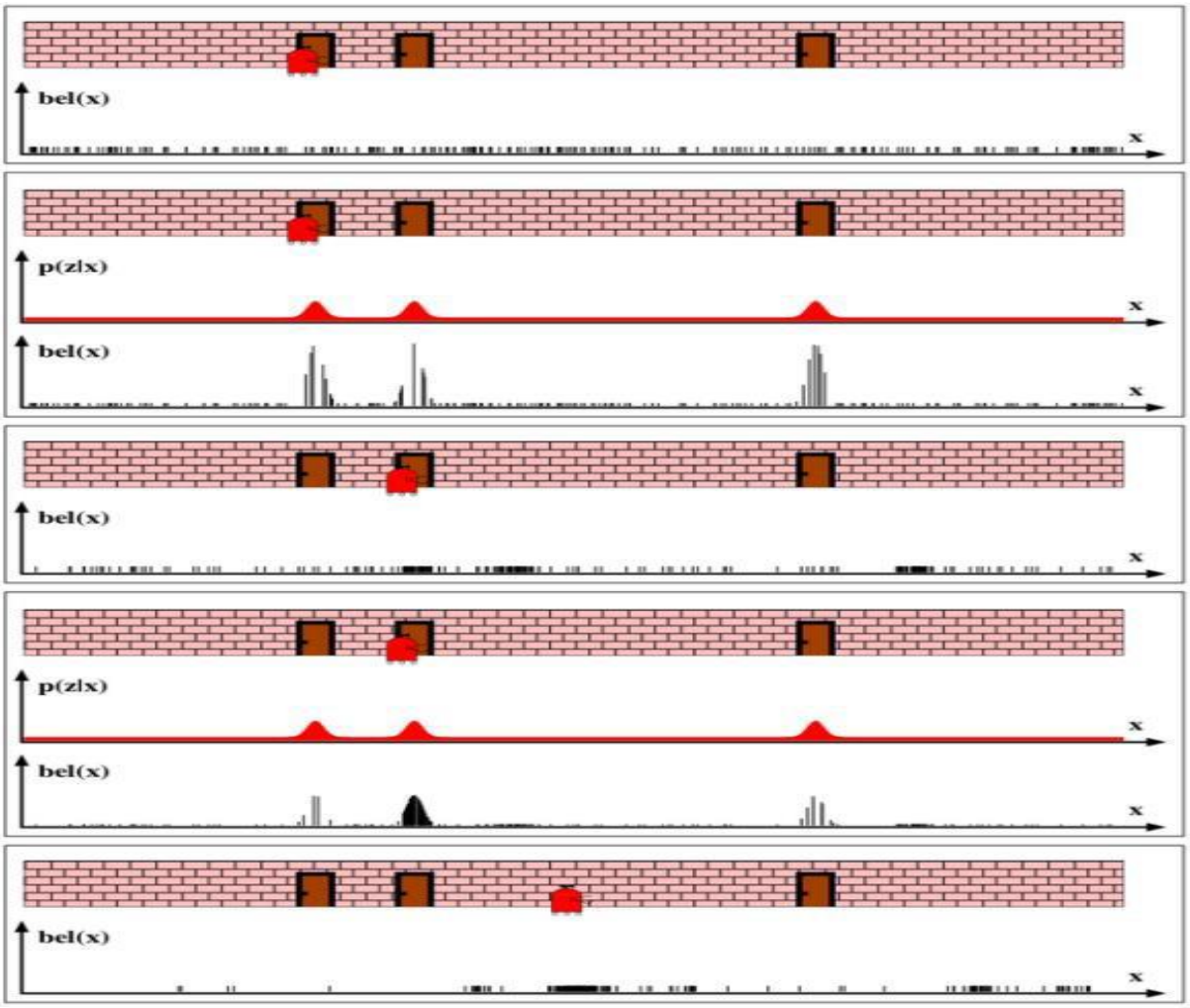


Figure 5.18 AMCL example

At time:

- **t=1**, Particles are drawn randomly and uniformly over the entire pose space.
- **t=2**, Measurement is updated and an importance weight is assigned to each particle.
- **t=3**, Motion is updated and a new particle set with uniform weights and high number of particles around the three most likely places is obtained in resampling.
- **t=4**, Measurement assigns non-uniform weight to the particle set.
- **t=5**, Motion is updated and a new resampling step is about to start.

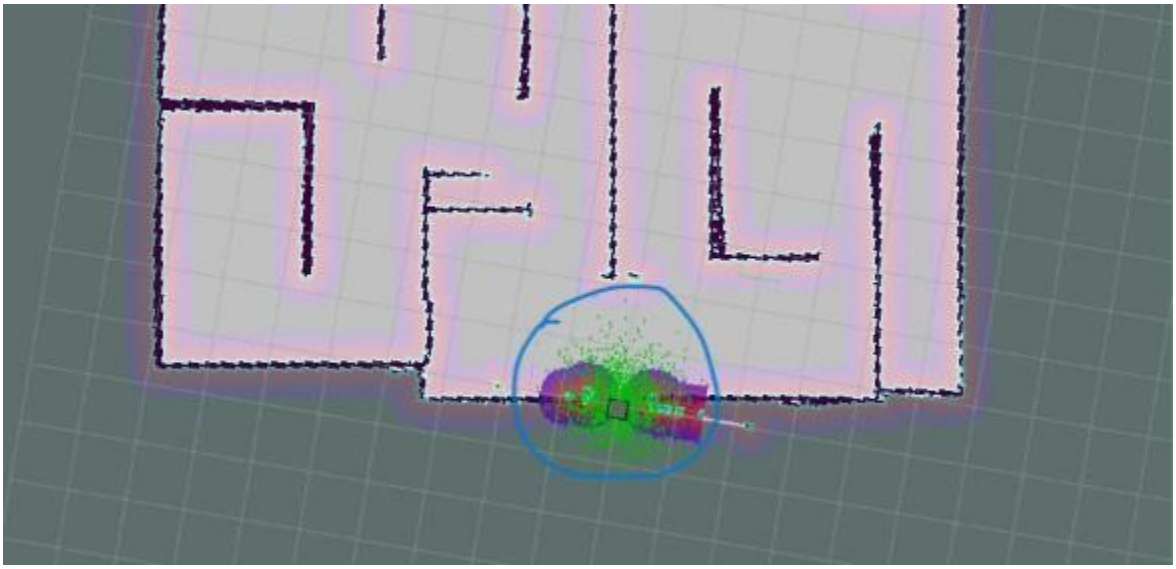


Figure 5.19 Pose Estimation

By now you should have a clear idea on the algorithms used, in the next section we will go through the installation of those packages and use them in a simulated environment before shifting to the real hardware

5.10 Installing the required Packages.

For the mapping operation there are multiple of packages that are needed to start.

first is Gmapping which is the main algorithm for the mapping mission.

```
sudo apt-get install ros-noetic-slam-gmapping
```

And for the keyboard control to move the robot manually

```
sudo apt-get install ros-noetic-teleop-twist-keyboard
```

And at the end by using the map-server package we can save the scanned map to be later used in the navigation and localization step.

```
sudo apt-get install ros-noetic-map-server
```

Next is to install the required package which is a collection of packages that help the robot to move from start location to goal location.

```
sudo apt-get install ros-noetic-navigation
```

Next up install rosserial packages in order for the ros device can communicate with the teensy topics.

```
sudo apt-get install ros-noetic-rosserial
```

5.11 Pins in Arduino

Arduino boards have multiple GPIO pins depending upon the board, some of the pins are analog which are connected to on board 10 bit-ADC (analog to digital converter). Analog pins can also be configured as digital ones. Arduino programming uses different functions to declare input output pins. Following is the function which is used to define pins in Arduino.

Vin: This is the input voltage pin of the Arduino board used to provide input supply from an external power source.

5V: This pin of the Arduino board is used as a regulated power supply voltage and it is used to give supply to the board as well as onboard components.

3.3V: This pin of the board is used to provide a supply of 3.3V which is generated from a voltage regulator on the board

GND: This pin of the board is used to ground the Arduino board.

Reset: This pin of the board is used to reset the microcontroller It is used to Resets the microcontroller.

Analog Pins: The pins A0 to A15 are used as an analog input and it is in the range of 0-5V. The analog pins on this board can be used as a digital Input or Output pins.

Serial pins: It is used for communication between the Arduino board and a computer or other devices.

The TXD and RXD are used to transmit & receive the serial data resp. It includes serial 0, Serial 1, serial 2, Serial 3 as follows:

1. Serial 0: It consists of Transmitter pin number 1 and receiver pin number 0
2. Serial 1: It consists of Transmitter pin number 18 and receiver pin number 19
3. serial 2: It consists of Transmitter pin number 16 and receiver pin number 17
4. Serial 3: It consists of Transmitter pin number 14 and receiver pin number 15

External Interrupts pins: This pin of the Arduino board is used to produce the External interrupt and it is done by the pin numbers 0,3,21,20,19,18.

I2C: This pin of the board is used for I2C communication.

1. Pin number 20 signifies Serial Data Line (SDA)and it is used for holding the data.
2. Pin number 21 signifies Serial Clock Line (SCL) and it is used for offering data synchronization among the devices.

SPI Pins: This is the Serial Peripheral Interface pin, it is used to maintainSPI communication with the help of the SPI library. SPI pins include:

1. MISO: Pin number 50 is used as a Master In Slave Out
2. MOSI: Pin number 51 is used as a Master Out Slave In
3. SCK: Pin number 52 is used as a Serial Clock
4. SS: Pin number 53 is used as a Slave Select

LED Pin: The board has an inbuilt LED using digital pin-13. The LED glows only when the digital pin becomes high.

AREF Pin: This is an analog reference pin of the Arduino board. It is used to provide a reference voltage from an external power supply.

To **define an Arduino pin** two ways can be used and those are:

- Using pinMode() function
- Using variables

Using pinMode() Function

The pinMode() function in Arduino is used to define pins. This function specified the given pin to either act as input or output. Pins on Arduino are default to set as input so we do not need to declare them separately as input using the pinMode() function.

In Arduino input pins can be triggered with a slight change in current inside the circuit. A Small amount of current can change the state of input pins from one to another. This also explains that pins configured as **pinMode(pin, INPUT)** can sense small changes and easily pick up electrical noises from the environment even when nothing or single wires is connected to them.
Below is the given syntax of pinMode() function:

```
pinMode(pin, mode)
```

Figure 5.20 Syntax of pinMode

Using Variables

Variables in programming are used to store data. Variable syntax consists of name, value, and a type. Variables can also be used for declaring pins in Arduino programming. We called it a declaration.

Here is a simple syntax of declaring pin 13 using an **int** variable:

```
int pin = 13;
```

Figure 5.21 Syntax of Variable

Here we created a variable whose name is **pin** having value **13,** and type is of **int.**
Once the pin is defined using a variable it's a lot easier to switch between the pins during the whole Arduino code, we just need to assign a new value to the variable pin and a new pin will be defined.

5.12 Simulation and Testing

Because there is uncertainty about the system's success in robotics, it is industry usual to run a simulation of the system before investing in development expenditures. We also wanted to check the software's viability and technique.

We used Gazebo to create a 3D model of the robot. In the simulation, a maze was built using gazebo's world builder to test the algorithms and the robot was placed Inside the room. A TOF was used to scan the walls of the maze and build a map, while encoder data were supplied automatically by gazebo publishing in the odometry topic.

Rviz was used to visualize the data from the simulation such as (map diagram, pose, interface to send waypoints). Rviz is also necessary with the final product to get pose on the map from sensor data on the robot.

The simulation's key benefit is that it gives the team a working prototype of the mapping, navigation, and localization algorithms before moving forward with the project. It also guided the team on the complexity and fine technical details of ROS and its packages, so that when the hardware (reality) arrived, the team was ready to test and implement.

Lastly the data obtained from hardware needs to be properly calibrated for the navigation software to achieve a functioning system.

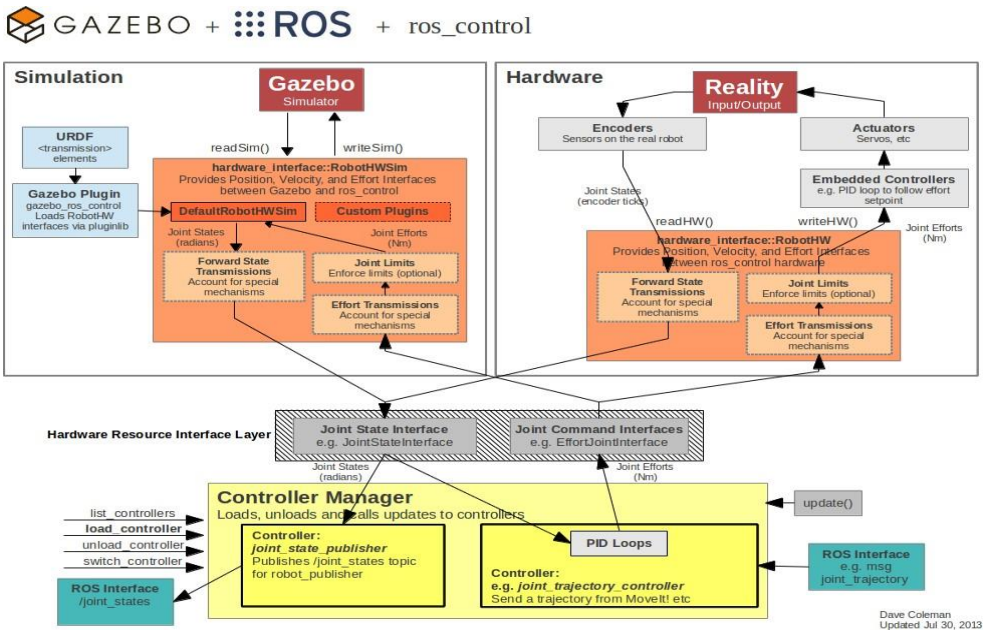


Figure 5.22 Gazebo Reality interface

5.13 Simulation Procedure

The simulation was run on Ubuntu 20.04. Main steps we went through in our simulation.

1. Creating the environment
2. Create a 2D Occupancy Grid of the environment (SLAM)
3. Navigate autonomously using the map generated from step 2

The first step was to create the 3D model of the robot to represent its dimensions it doesn't have to be detailed, using a Unified robot description format (URDF) model as shown in the figure below.

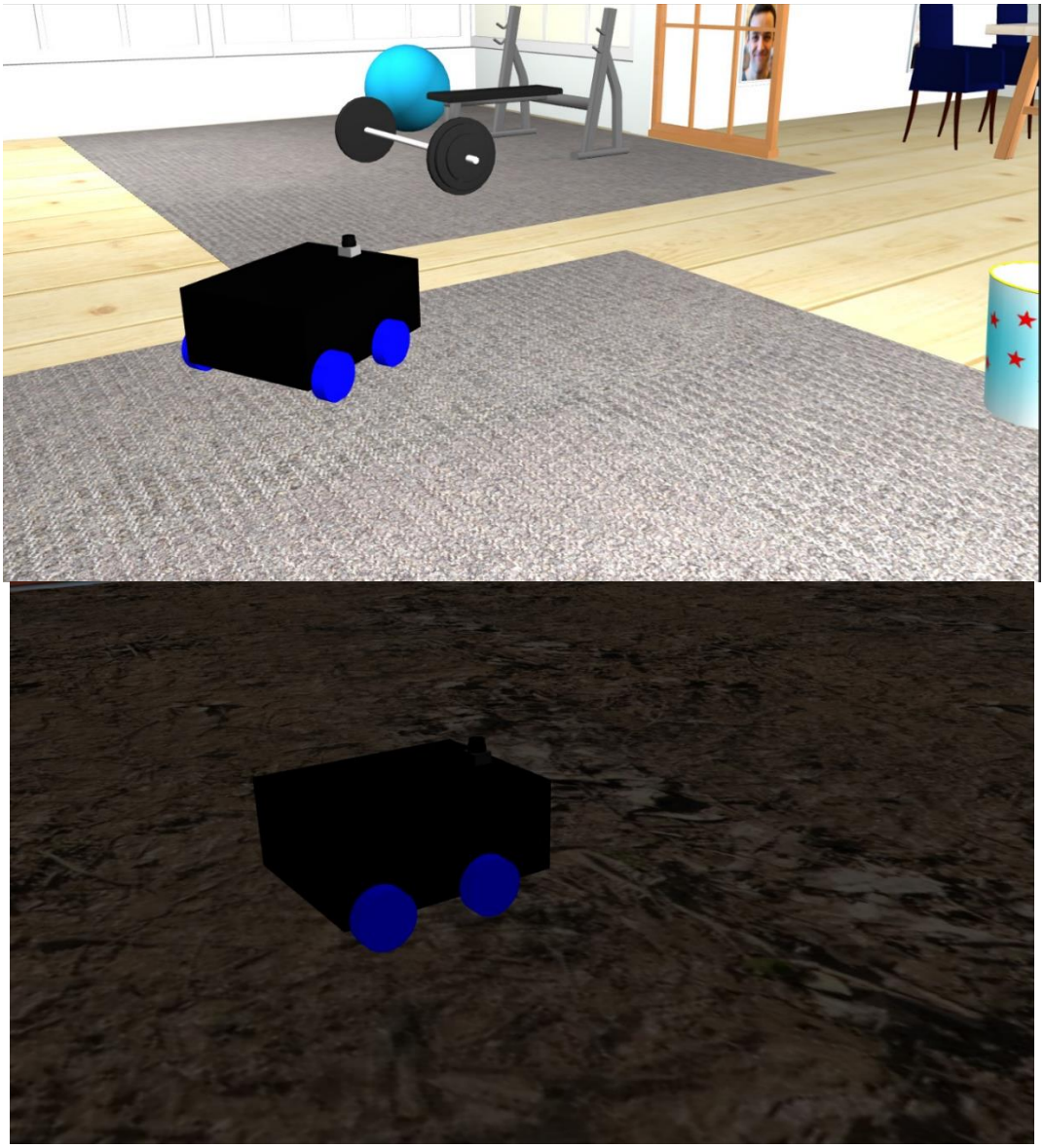


Figure 5.23 URDF Example

The second step is to build the environment and run SLAM to create the Occupancy grid map of the environment.

Figure 5.24 Simulated environments

Next up in Rviz with using the keyboard control to move the robot in the map it will start to generate the 2D Occupancy grid map.

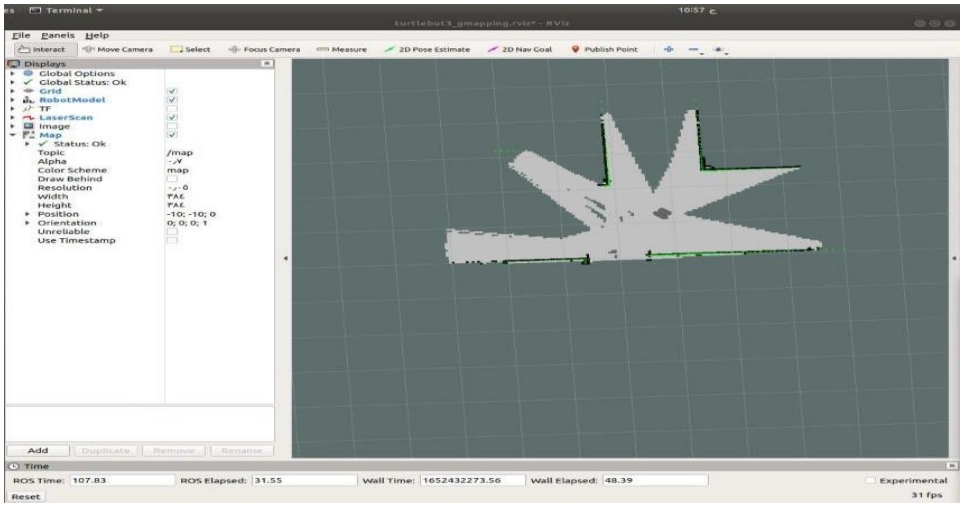
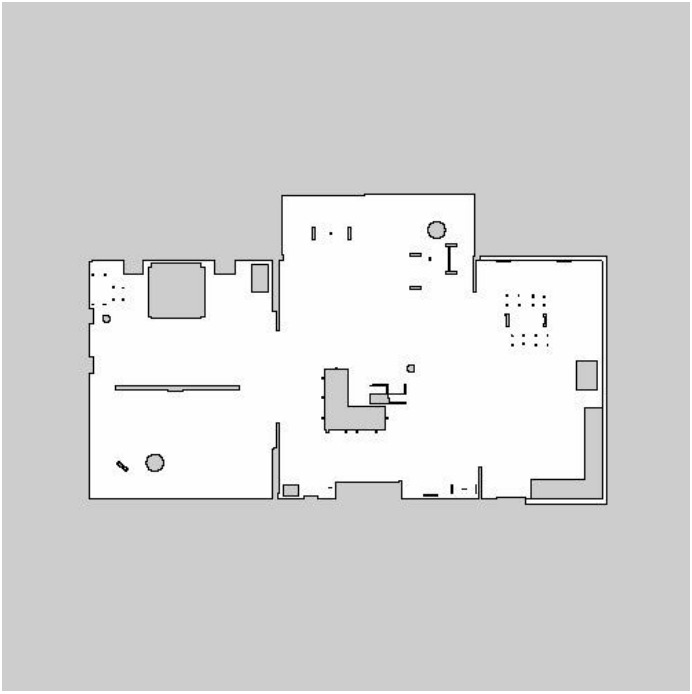


Figure 5.25 Mapping Operation

After reading the whole map its time to save the map using map_server package to be called in the navigation section.

The 2D Occupancy grid is stored as a .pgm (Portable Graymap) image and a .yaml file for metadata. Non white pixels in the image represent obstacles. Note



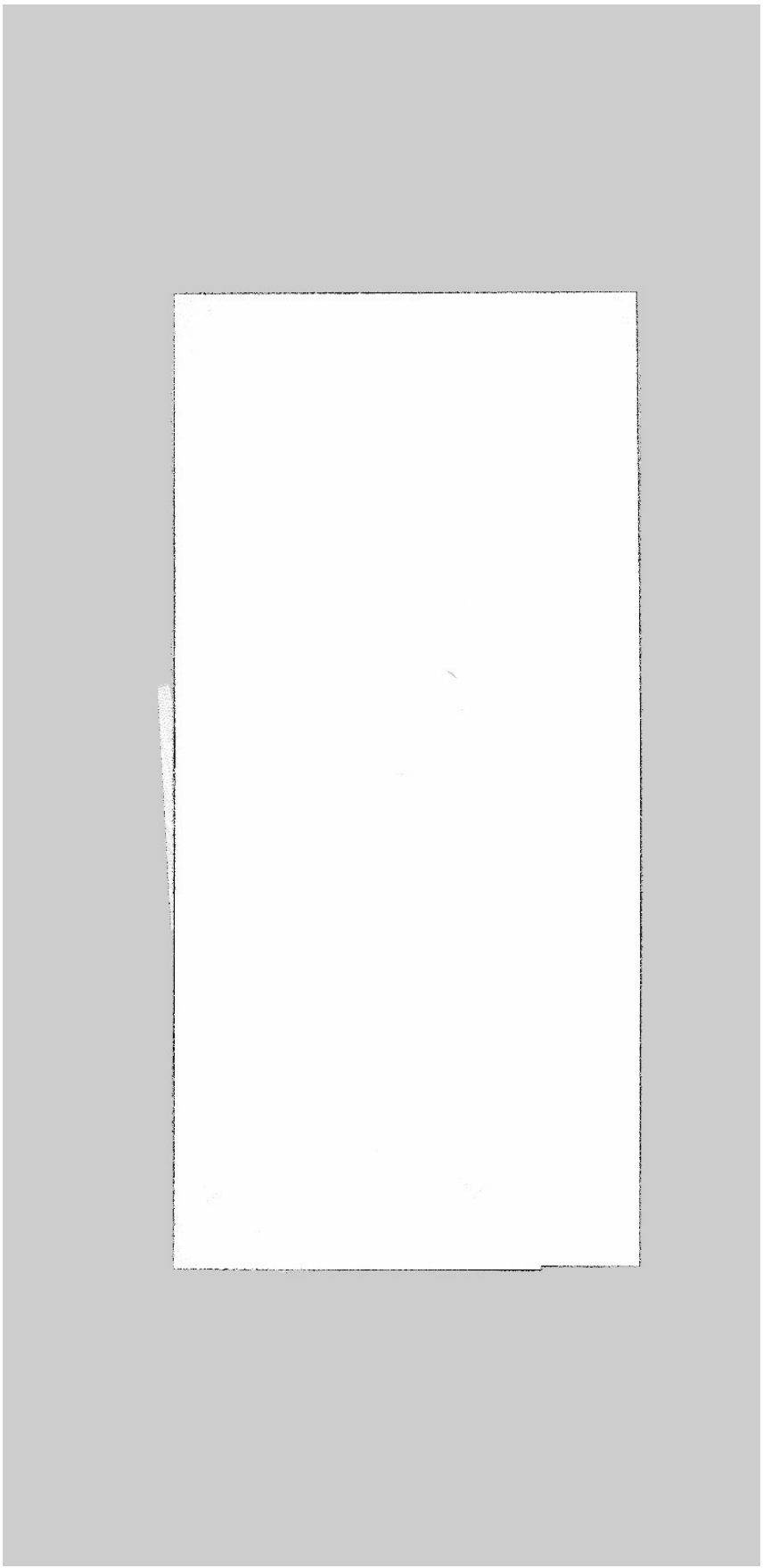
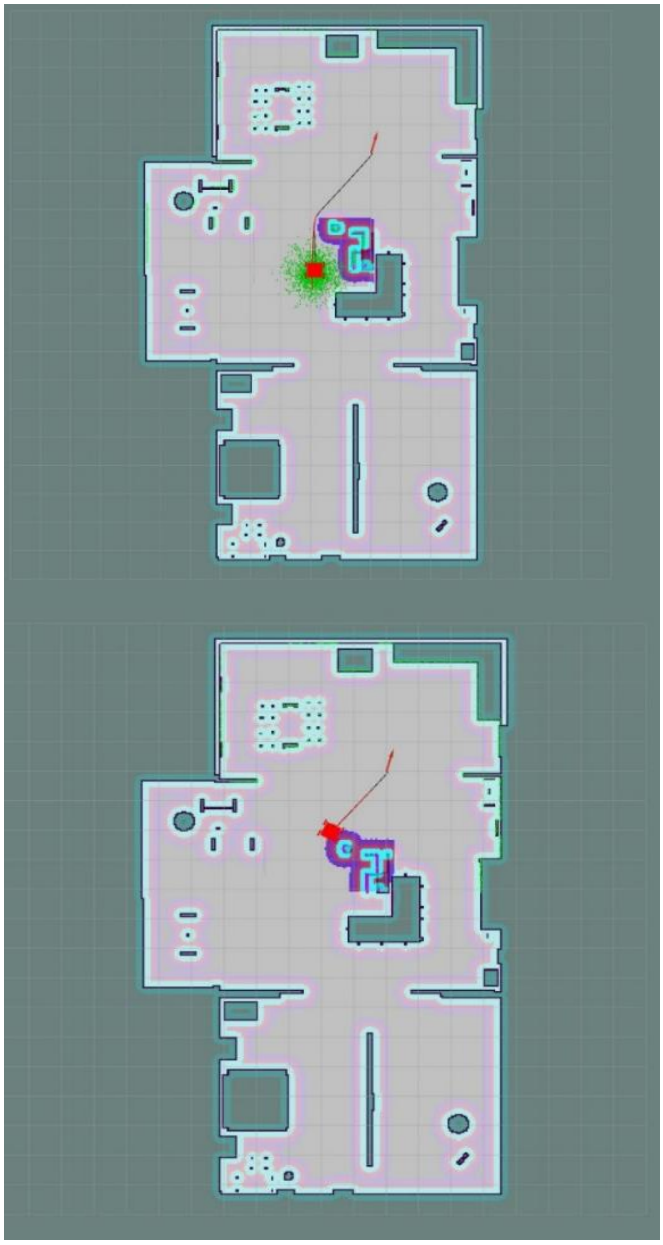


Figure 5.26 Saved Maps

The third step is to initialize the navigation packages Navigation launch package:

1. AMCL package
2. Move_base navigation package.

Now that the map has been created and saved, the navigation stack packages localize the robot in the environment using LiDAR, and the user can use the RViz GUI to select waypoints for the robot to navigate.



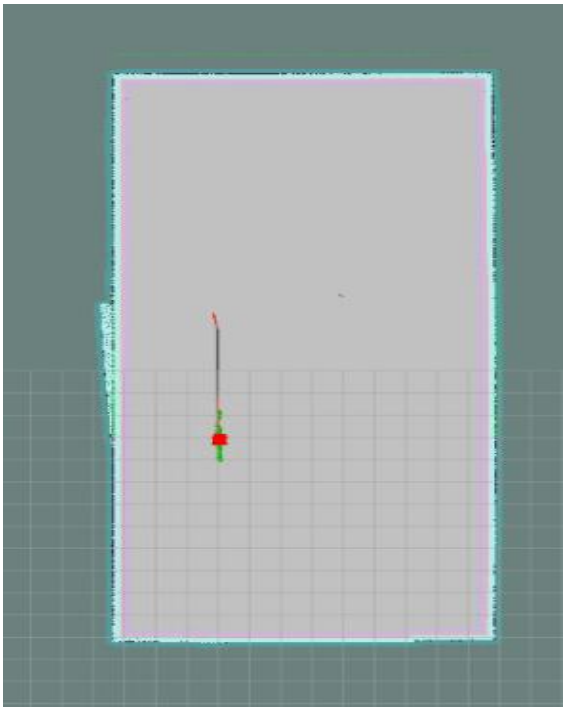


Figure 5.27 RViz Navigation GUI

By this point the robot is capable to navigate autonomously when given a goal waypoint by the user while avoiding obstacles in the simulation.

5.14 User Manual

View Odometry

On the developer computer, control the robot with keyboard (Tele-op mode), write the following in terminal, Launch base driver:

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

To Subscribe to /odom topic:

```
$ rostopic echo odom
```

- **Creating a Map**

On the developer computer write the following in terminal:

```
$ roslaunch gmapping slam_gmapping.launch
```

On the developer computer, to control the robot with keyboard to scan the map, write the following in terminal:

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

On your development computer, Run rviz:

```
rosrun rviz rviz
```

- **Saving the map**

After scanning the whole map now, we save it by running map_server on the robot's computer:
On your development computer, Run rviz:

```
$ rosrun map_server map_saver -f my_map
```

5.15 Flow chart

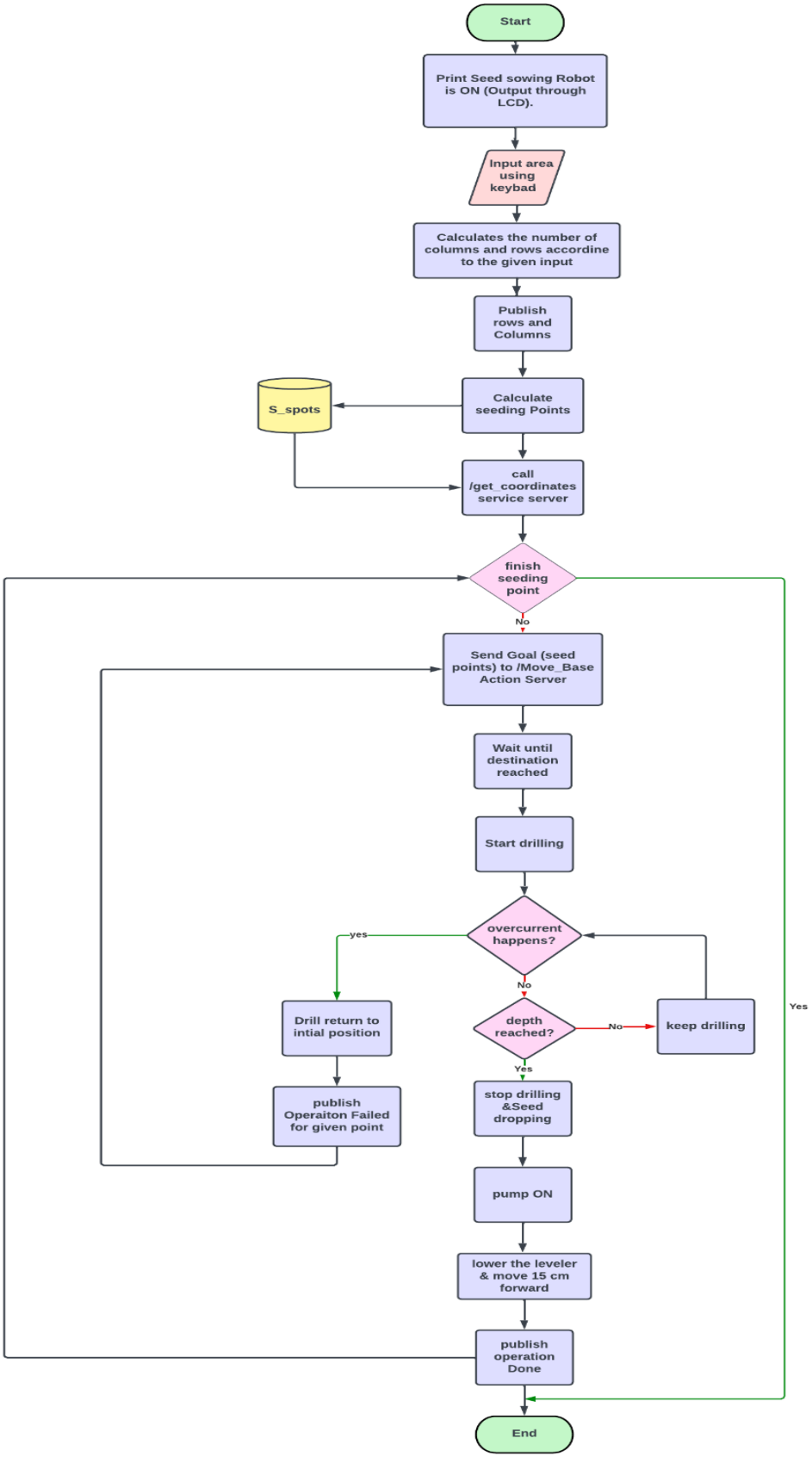


Figure 5.28 Flow Chart

Conclusion:

The development of a seed sowing robot with irrigation capabilities brings numerous benefits to the field of agriculture. By integrating the components mentioned earlier, including DC motors with gearboxes, wheels, sensors, microcontrollers, and various other components, the seed sowing robot can automate the sowing process and improve efficiency in agricultural operations. The inclusion of irrigation components such as a water pump and sensors enables the robot to provide precise and controlled watering, optimizing plant growth and resource utilization.

The seed sowing robot's ability to accurately distribute seeds and provide irrigation offers several advantages. It reduces the need for manual labor, leading to increased productivity and cost savings for farmers. The robot's precise seed distribution ensures optimal plant spacing, maximizing crop yield and minimizing seed wastage. Moreover, the incorporation of irrigation capabilities helps to maintain proper soil moisture levels, promoting healthier plant growth and reducing water consumption.

This robotic system also addresses various important factors, including cost, environmental impact, manufacturability, ethics, social and economic impact, health and safety, and sustainability. The cost evaluation of the project reveals its affordability within the specified budget. The use of sustainable materials, efficient energy sources, and optimized resource utilization contributes to a reduced environmental impact. The project's manufacturability considers compatibility with existing manufacturing techniques and allows for potential scalability. Ethical considerations ensure the responsible use of automation technology in agriculture, prioritizing human welfare and adherence to regulations. The social and economic impact evaluation highlights the potential benefits for farmers, workers, and the agricultural industry as a whole. Health and safety measures, including risk assessments and safety features, guarantee a secure operating environment. Lastly, the emphasis on sustainability ensures long-term viability and minimizes waste generation.

In conclusion, the seed sowing robot with irrigation combines advanced robotic technology with efficient seed distribution and precise watering, offering significant advantages to the agricultural sector. By integrating various components and addressing key evaluation factors, this innovative solution enhances productivity, resource utilization, and sustainability in farming practices.

Appendices

Appendix A – Program Code

- Arduino Code

```
#include <Arduino.h>
#include <ros.h>
#include <std_msgs/Int16.h>
#include <std_msgs/Float64.h>
#include <std_msgs/Int32.h>
#include <geometry_msgs/TwistStamped.h>
#include <std_msgs/String.h>
#include <TFT.h>
#include <SPI.h>
#include <Keypad.h>
#include <Servo.h>



#define TICKS_MEASURMENT_INTERVAL 50

// Encoder output to Arduino Interrupt pin. Tracks the tick count.
#define ENC_IN_LEFT_A 2
#define ENC_IN_RIGHT_A 3
#define ENC_IN_LEFT_A_f 19
#define ENC_IN_RIGHT_A_f 18

// Other encoder output to Arduino to keep track of wheel direction
// Tracks the direction of rotation.
#define ENC_IN_LEFT_B 22
#define ENC_IN_RIGHT_B 23
#define ENC_IN_LEFT_B_f 25
#define ENC_IN_RIGHT_B_f 24
//drill motor pins
#define IN1 26
#define IN2 27
#define IN3 28
#define IN4 29
```

```
#define cs 53
#define dc 12
#define rst 13

#define funnel_servo_pin 11
#define levelr_servo_pin 10
#define Relay_PIN 30
#define MAX_DRILL_CURRENT 2.3

Servo funnel_servo;
Servo levelr_servo;
Servo myservo;
const byte ROWS = 4 ;
const byte COLS = 3 ;

char hexakeys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
byte rowPins[ROWS] = {49,48,47,46} ;
byte colPins[COLS] = {45,44,43};
char string[20] ;
char string1[20];
bool area = false ;
int i = 0;
int j = 0;
int t = 0;
int ROW =0;
int COL=0;
char  waitkey = NO_KEY ;
int x = 0 ;
int y = 0 ;
```

```
Keypad myKeypad = Keypad(makeKeymap(hexakeys), rowPins, colPins,
ROWS, COLS);
// 100ms interval for measurements
long previousMillis = 0;
long currentMillis = 0;
long lcd_time = 0 ;
long lcd_ptime = 0 ;

//time for velocity measurments
long current_time = 0;
long prev_time = 0;
long time_interval =0;
//variables needed to compute velocity
float wheelbase = 0.4;
float wheel_radius = 0.13;
long prev_bl_ticks = 0;
long prev_br_ticks = 0;
long prev_fl_ticks = 0;
long prev_fr_ticks = 0;

//speed_test

double velLeftWheel = 0;
double velRightWheel = 0;
const double TICKS_PER_METER = 685.50;



// create an instance of the library
TFT TFTscreen = TFT(cs, dc, rst);
// Handles startup and shutdown of ROS
ros::NodeHandle nh;

volatile int pose = 0; //measure drill motor ticks
```

```
// True = Forward; False = Reverse
boolean Direction_left = true;
boolean Direction_right = true;

// Minumum and maximum values for 16-bit integers
const int encoder_minimum = -32768;
const int encoder_maximum = 32767;

// Keep track of the number of wheel ticks
std_msgs::String str_msg;
ros::Publisher chatter("a_chatter", &str_msg);

std_msgs::Int32 row;
ros::Publisher rows_pub("/rows",&row);

std_msgs::Int32 col;
ros::Publisher cols_pub("/columns",&col);

std_msgs::Int16 right_wheel_tick_count;
ros::Publisher rightPub("br_ticks", &right_wheel_tick_count);

std_msgs::Int16 left_wheel_tick_count;
ros::Publisher leftPub("bl_ticks", &left_wheel_tick_count);

std_msgs::Int16 right_wheel_tick_count_f;
ros::Publisher rightPub_1("fr_ticks", &right_wheel_tick_count_f);

std_msgs::Int16 left_wheel_tick_count_f;
ros::Publisher leftPub_1("fl_ticks", &left_wheel_tick_count_f);

std_msgs::Int16 right_wheel_tick_count_2;
ros::Publisher rightPub2("right_ticks", &right_wheel_tick_count_2);

std_msgs::Int16 left_wheel_tick_count_2;
ros::Publisher leftPub2("left_ticks", &left_wheel_tick_count_2);
```

```
std_msgs::Float64 right_wheel_vel;
ros::Publisher rightvel("right_vel", &right_wheel_vel);

std_msgs::Float64 left_wheel_vel;
ros::Publisher leftvel("left_vel", &left_wheel_vel);

void op_Callback(const std_msgs::String&op_msg );
ros::Subscriber<std_msgs::String> op_sub("/operation", op_Callback);

//geometry_msgs::TwistStamped measured_vel;
//ros::Publisher velocity_pub("v_w_velocity", &measured_vel);

// Increment the number of ticks

void right_wheel_tick() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_RIGHT_B);

  if(val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }

  if (Direction_right) {

    if (right_wheel_tick_count.data == encoder_maximum) {
      right_wheel_tick_count.data = encoder_minimum;
    }
    else {
      right_wheel_tick_count.data++;
    }
  }
```

```
else {
    if (right_wheel_tick_count.data == encoder_minimum) {
      right_wheel_tick_count.data = encoder_maximum;
    }
    else {
      right_wheel_tick_count.data--;
    }
  }
}

// Increment the number of ticks
void left_wheel_tick() {

  // Read the value for the encoder for the left wheel
  int val = digitalRead(ENC_IN_LEFT_B);

 if(val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count.data == encoder_maximum) {
      left_wheel_tick_count.data = encoder_minimum;
    }
    else {
      left_wheel_tick_count.data++;
    }
  }
  else {
    if (left_wheel_tick_count.data == encoder_minimum) {
      left_wheel_tick_count.data = encoder_maximum;
    }
```

```
else {
     left_wheel_tick_count.data--;
   }
  }
}

// Increment the number of ticks
void right_wheel_tick_f() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_RIGHT_B_f);

  if(val == LOW) {
    Direction_right = false; // Reverse
  }
  else {
    Direction_right = true; // Forward
  }
  if (Direction_right) {
    if (right_wheel_tick_count_f.data == encoder_maximum) {
      right_wheel_tick_count_f.data = encoder_minimum;
    }
    else {
      right_wheel_tick_count_f.data++;
    }
  }
  else {
    if (right_wheel_tick_count_f.data == encoder_minimum) {
      right_wheel_tick_count_f.data = encoder_maximum;
    }
    else {
      right_wheel_tick_count_f.data--;
    }
  }
```

```
// Increment the number of ticks
void left_wheel_tick_f() {

  // Read the value for the encoder for the right wheel
  int val = digitalRead(ENC_IN_LEFT_B_f);


 if(val == LOW) {
    Direction_left = true; // Reverse
  }
  else {
    Direction_left = false; // Forward
  }

  if (Direction_left) {
    if (left_wheel_tick_count_f.data == encoder_maximum) {
      left_wheel_tick_count_f.data = encoder_minimum;
    }
    else {
      left_wheel_tick_count_f.data++;
    }
  }
  else {
    if (left_wheel_tick_count_f.data == encoder_minimum) {
      left_wheel_tick_count_f.data = encoder_maximum;
    }
    else {
      left_wheel_tick_count_f.data--;
    }
  }
}      left_wheel_tick_count.data--;
    }
  }
```

```
int measure_current(){
 int adc = analogRead(A1);
  float voltage = adc*5/1023.0 ;
  float current = (voltage-2.5)/0.1;
  return current ;
}

void drill_down(){

  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);


}


void drill_up(){

  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);



void PUMP_ON(){

  digitalWrite(Relay_PIN,LOW);

}
```

```
void setup() {

  //initialize the library
  TFTscreen.begin();

  // clear the screen with a black background
  TFTscreen.background(0, 0, 0);
  //set the text size
  TFTscreen.setTextSize(2);

  TFTscreen.stroke(255, 255, 255);
  TFTscreen.text("seed sowing", 15,47);
  TFTscreen.text("robot is on", 15,70);
  delay(2000);
  TFTscreen.background(0, 0, 0);

  // Set pin states of the encoder
  pinMode(ENC_IN_LEFT_A , INPUT);
  pinMode(ENC_IN_LEFT_B , INPUT);
  pinMode(ENC_IN_RIGHT_A , INPUT);
  pinMode(ENC_IN_RIGHT_B , INPUT);
  pinMode(ENC_IN_LEFT_A_f , INPUT);
  pinMode(ENC_IN_LEFT_B_f , INPUT);
  pinMode(ENC_IN_RIGHT_A_f , INPUT);
  pinMode(ENC_IN_RIGHT_B_f , INPUT);
  // Set drill mechanism pins
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(Relay_PIN,OUTPUT);
  pinMode(21, INPUT_PULLUP);void PUMP_OFF(){
```

```
// Attach interrupt
  attachInterrupt(digitalPinToInterrupt(20), TIKS, RISING);
  // Every time the pin goes high, this is a tick
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A),
left_wheel_tick, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A),
right_wheel_tick, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_LEFT_A_f),
left_wheel_tick_f, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A_f),
right_wheel_tick_f, RISING);

  funnel_servo.attach(funnel_servo_pin);
  levelr_servo.attach(levelr_servo_pin);
  funnel_servo.write(180);
  levelr_servo.write(90);

  nh.getHardware()->setBaud(57600);
  nh.initNode();
  nh.advertise(rows_pub);
  nh.advertise(cols_pub);
  nh.advertise(rightPub);
  nh.advertise(leftPub);
  nh.advertise(rightPub_1);
  nh.advertise(leftPub_1);
  nh.advertise(rightPub2);
  nh.advertise(leftPub2);
  nh.advertise(rightvel);
  nh.advertise(leftvel);
  nh.advertise(chatter);
  nh.subscribe(op_sub);
}
```

```
//fuction to claculate the rows.
int calc_rows(int y){
  int r= y/0.25;
  return r;

  }
//fuction to claculate the columns.
int calc_cols(int x){
  int c= (x-0.5)/0.25;
  return c;
    }
void loop() {

  // put your main code here, to run repeatedly:
  nh.spinOnce();

  currentMillis = millis();

   if (currentMillis - previousMillis >
TICKS_MEASURMENT_INTERVAL) {
   previousMillis = currentMillis;
   publish_ticks();

   calc_vel_right_wheel();
   calc_vel_left_wheel();
   //measured_velocity();
   }

  if (area == false){
  TFTscreen.text("Enter Land ", 6,10);
  TFTscreen.text("size:", 6,30);
  TFTscreen.text("1-(x):", 6,50);
  TFTscreen.text("1-(y):", 6,90);
```

```
waitkey = myKeypad.getKey();
  if (waitkey != NO_KEY && t == 0){
    if(waitkey == '#'){
    sscanf(string, "%d", &x);
    ROW = calc_rows(x);
    TFTscreen.text(string, 100,50);
    row.data = ROW;
    rows_pub.publish(&row);
    // Serial.println(x);
    waitkey = 'NO_KEY';
    t = 1 ;
    i = 0 ;
  }
  else {
    // Serial.println(waitkey);
   string[i] = waitkey ;
   i++;
   waitkey = NO_KEY;
  }
  }

  else if (waitkey != NO_KEY && t == 1){
    if(waitkey == '#'){

    TFTscreen.text(string1, 100,90);
    sscanf(string1, "%d", &y);
    // Serial.println(y);
    waitkey = NO_KEY;
    area = true ;
    i = 0 ;
    t = 0 ;
    COL = calc_cols(y);
    col.data = COL;
    cols_pub.publish(&col);
```

```
else{
   string1[i] = waitkey ;
    // Serial.println(waitkey);
   i++;
   waitkey = 'NO_KEY';
  }
  }
  }

  if (area == true){

  waitkey = myKeypad.getKey();

  if (waitkey == '*'){
   TFTscreen.background(0, 0, 0);
   area = false;
  }
  }

}


void publish_ticks(){

  // Record the time
  //currentMillis = millis();

  // If 100ms have passed, print the number of ticks
  //if (currentMillis - previousMillis >
TICKS_MEASURMENT_INTERVAL) {

   // previousMillis = currentMillis;
```

```
  rightPub.publish( &right_wheel_tick_count );
    leftPub.publish( &left_wheel_tick_count );
    rightPub_1.publish( &right_wheel_tick_count_f );
    leftPub_1.publish( &left_wheel_tick_count_f );
    right_wheel_tick_count_2.data = (right_wheel_tick_count.data/2.0 +
right_wheel_tick_count_f.data/2.0);
    rightPub2.publish( &right_wheel_tick_count_2 );
    left_wheel_tick_count_2.data = (left_wheel_tick_count.data/2.0 +
left_wheel_tick_count_f.data/2.0);
    leftPub2.publish( &left_wheel_tick_count_2 );
//}
}

//velocity measurment
void calc_vel_left_wheel(){

  // Previous timestamp
  static double prevTime_l = 0;

  // Variable gets created and initialized the first time a function is called.
  static int prevLeftCount = 0;

  // Manage rollover and rollunder when we get outside the 16-bit integer
range
  int l_numOfTicks = (65535 + left_wheel_tick_count_2.data -
prevLeftCount) % 65535;

  // If we have had a big jump, it means the tick count has rolled over.
  if (l_numOfTicks > 10000) {
       l_numOfTicks = 0 - (65535 - l_numOfTicks);
  }
```

```
   // Calculate wheel velocity in meters per second
  velLeftWheel = l_numOfTicks/TICKS_PER_METER/((millis()/1000.0)-
prevTime_l);

  // Keep track of the previous tick count
  prevLeftCount = left_wheel_tick_count_2.data;

  left_wheel_vel.data = velLeftWheel;
  leftvel.publish(&left_wheel_vel);
  // Update the timestamp
  prevTime_l = (millis()/1000.0);

}


// Calculate the right wheel linear velocity in m/s every time a
// tick count message is published on the /right_ticks topic.
void calc_vel_right_wheel(){

  // Previous timestamp
  static double prevTime_R = 0;

  // Variable gets created and initialized the first time a function is called.
  static int prevRightCount = 0;
  //Serial.println("right_ticks:"+String(right_wheel_tick_count_2.data));
  //Serial.println("previos_right_ticks:"+String(prevRightCount));

// Manage rollover and rollunder when we get outside the 16-bit integer
range
  int R_numOfTicks = (65535 + right_wheel_tick_count_2.data -
prevRightCount) % 65535;
  //Serial.println("num0fTicks:"+String(numOfTicks));
```

```
if (R_numOfTicks > 10000) {
      R_numOfTicks = 0 - (65535 - R_numOfTicks);
  }

  // Calculate wheel velocity in meters per second
  // Calculate time interval in seconds
  //double deltaTime = (millis() / 1000.0) - prevTime_R;
  //Serial.println("deltatiime:"+String(deltaTime));
  velRightWheel =
R_numOfTicks/TICKS_PER_METER/((millis()/1000.0)-prevTime_R);
  //Serial.println("velocity:"+String(velRightWheel));

  prevRightCount = right_wheel_tick_count_2.data;

  right_wheel_vel.data = velRightWheel ;
  rightvel.publish(&right_wheel_vel) ;

  prevTime_R = (millis()/1000.0);

}




void TIKS() {
  // Read the value for the encoder for the right wheel
  int val = digitalRead(21);

  if (val == HIGH) {
    pose++;
  } else {
    pose--;
  }
}
```

```
void op_Callback( const std_msgs::String& op_msg){

  str_msg.data = "operation started";
  chatter.publish( &str_msg );
  while(pose <= 20000){
     str_msg.data = "drill going down";
     chatter.publish( &str_msg );
     drill_down();
     if(measure_current() < MAX_DRILL_CURRENT){

       break;
     }
   }

  while(pose >= 20000){
    str_msg.data = "drill going up";
    chatter.publish( &str_msg );
    drill_down();
  }

  funnel_servo.write(120);
  levelr_servo.write(180);
  delay(500);
  funnel_servo.write(180);

    PUMP_ON();
  str_msg.data = "PUMP ON";
  chatter.publish( &str_msg );
  delay(1000);
  void PUMP_OFF();

  str_msg.data = "Done";
  chatter.publish( &str_msg );

}
```

- ESP32 Code

```
#include "MPU9250.h"
#include <ros.h>
#include <String.h>
#include "CytronMotorDriver.h"
#include <std_msgs/Int16.h>
#include <std_msgs/Float64.h>
#include <geometry_msgs/Twist.h>
#include <std_msgs/String.h>
#include <sensor_msgs/Imu.h>
#include <sensor_msgs/MagneticField.h>


#define IMU_PUBLISH_RATE 50 //HZ

long sequence = 0;

// an MPU9250 object with the MPU-9250 sensor on I2C bus 0 with address 0x68
MPU9250 IMU(Wire,0x68);

int status;

//variable to measure time
static unsigned long prev_imu_time = 0;

// Motor A&B connections
const int enA =4;
const int in1 =14;
const int enB =17;
const int in2 =16;

// Motor C&D connections
const int enC = 19;
const int in3 =18;
const int enD = 13;
const int in4 = 23;
```

```
// Handles startup and shutdown of ROS
ros::NodeHandle nh;

// How much the PWM value can change each cycle
const int PWM_INCREMENT = 1;


// Proportional constant, which was measured by measuring the
// PWM-Linear Velocity relationship for the robot.
const int K_P = 255;

// Y-intercept for the PWM-Linear Velocity relationship for the robot
const int b = 5;

// Correction multiplier for drift. Chosen through experimentation.
const int DRIFT_MULTIPLIER = 5;

// Turning PWM output (0 = min, 255 = max for PWM values)
const int PWM_TURN = 120;

// Set maximum and minimum limits for the PWM values
const int PWM_MIN = 40; // about 0.35 m/s
const int PWM_MAX = 100; // about 0.39 m/s

// Set linear velocity and PWM variable values for each wheel
double velLeftWheel = 0;
double velRightWheel = 0;
double pwmLeftReq = 0;
double pwmRightReq = 0;
// Record the time that the last velocity command was received
double lastCmdVelReceived = 0;

// Time interval for measurements in milliseconds
const int interval = 50;
long previousMillis = 0;
long currentMillis = 0;

//variable to set the driving mode
String drive_mode = "";
```

```
CytronMD motor1(PWM_DIR, 4, 14); // PWM 1 = Pin 13, DIR 1 = Pin 22.
CytronMD motor2(PWM_DIR, 17, 16); // PWM 2 = Pin 12, DIR 2 = Pin
24.
CytronMD motor3(PWM_DIR, 13, 23); // PWM 3 = Pin 11, DIR 3 = Pin
26.
CytronMD motor4(PWM_DIR, 19, 18); // PWM 4 = Pin 10, DIR 4 = Pin
30.

//ROS subscribers and publishers

std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

std_msgs::Float64 deb;
ros::Publisher debug("debug", &deb);

std_msgs::Float64 deb1;
ros::Publisher debug1("debug1", &deb1);

void set_mode(const std_msgs::String&mode );
ros::Subscriber<std_msgs::String> mode("driving_mode", set_mode);

void cmdCallback(const std_msgs::String&str_msg_2 );
ros::Subscriber<std_msgs::String> sub("arduino_cmd", cmdCallback);

void right_vel_callback(const std_msgs::Float64&right_vel);
ros::Subscriber<std_msgs::Float64> RVelocity("right_vel",
right_vel_callback);
void left_vel_callback(const std_msgs::Float64&left_vel);
ros::Subscriber<std_msgs::Float64> LVelocity("left_vel",
left_vel_callback);

// Set up ROS subscriber to the velocity command
void calc_pwm_values(const geometry_msgs::Twist& cmdVel);
ros::Subscriber<geometry_msgs::Twist> subCmdVel("cmd_vel",
calc_pwm_values );

sensor_msgs::Imu imu_msg;
ros::Publisher imu_pub("/imu/data_raw", &imu_msg);
```

```
void setup() {


  nh.initNode();

  nh.subscribe(subCmdVel);
  nh.subscribe(sub);
  nh.subscribe(RVelocity);
  nh.subscribe(LVelocity);
  nh.subscribe(mode);
  nh.advertise(chatter);
  nh.advertise(debug);
  nh.advertise(debug1);
  nh.advertise(imu_pub);


  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(enC, OUTPUT);
  pinMode(enD, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  // Set the motor speed
  analogWrite(enA, 0);
  analogWrite(enB, 0);
  analogWrite(enC, 0);
  analogWrite(enD, 0);

    status = IMU.begin();
  while (status < 0) {
    status = IMU.begin();
  }
}
```

```
void loop() {


  //ensure that callbacks are called and messages are processed within the
ROS event loop.
  nh.spinOnce();
  // Record the time
  pub_imu();

  currentMillis = millis();

  // If the time interval has passed, publish the number of ticks,
  // and calculate the velocities.
  if (currentMillis - previousMillis > interval) {

    previousMillis = currentMillis;

  if(drive_mode == "teleop"){

  //do nothing

  }

  else{
 // Stop the car if there are no cmd_vel messages
  if((millis()/1000) - lastCmdVelReceived > 1) {
  pwmLeftReq = 0;
  pwmRightReq = 0;
   }
  set_pwm_values();
  }

  }

}

void set_mode(const std_msgs::String&mode ){
 drive_mode = mode.data ;

}
```

```
//custom made function for easier motor control
void cmdCallback( const std_msgs::String& str_msg_2) {
  String cmd_str(str_msg_2.data); // convert const char* to String
  cmd_str.trim();

  if (cmd_str == "forward"){
    str_msg.data = str_msg_2.data;
    chatter.publish( &str_msg );
    motor1.setSpeed(70);
    motor2.setSpeed(70);
    motor3.setSpeed(-70);
    motor4.setSpeed(-70);
  }

  else if (cmd_str == "backward") {
    str_msg.data = str_msg_2.data;
    chatter.publish( &str_msg );
  motor1.setSpeed(-70);
  motor2.setSpeed(-70);
  motor3.setSpeed(70);
  motor4.setSpeed(70);
  }

  else if (cmd_str == "left") {
    str_msg.data = str_msg_2.data;
    chatter.publish( &str_msg );
  motor1.setSpeed(110);
  motor2.setSpeed(110); .
  motor3.setSpeed(110);
  motor4.setSpeed(110);

}
else if (cmd_str == "right"){
  str_msg.data = str_msg_2.data;
  chatter.publish( &str_msg );
  motor1.setSpeed(-110);
  motor2.setSpeed(-110);
  motor3.setSpeed(-110);
  motor4.setSpeed(-110);
  }
```

```
else if (cmd_str == "stop"){
     str_msg.data = str_msg_2.data;
    chatter.publish( &str_msg );
  motor1.setSpeed(0);
  motor2.setSpeed(0);
  motor3.setSpeed(0);
  motor4.setSpeed(0);
   }

}

void right_vel_callback(const std_msgs::Float64&right_vel){
velRightWheel = right_vel.data;
}

void left_vel_callback(const std_msgs::Float64&left_vel){
velLeftWheel = left_vel.data;
}

// Take the velocity command as input and calculate the PWM values.
void calc_pwm_values(const geometry_msgs::Twist& cmdVel) {

  // Record timestamp of last velocity command received
  lastCmdVelReceived = (millis()/1000);

  // Calculate the PWM value given the desired velocity
  pwmLeftReq = K_P * cmdVel.linear.x + b;
  pwmRightReq = K_P * cmdVel.linear.x + b;
   deb1.data = pwmRightReq;
    debug1.publish(&deb1);
  // Check if we need to turn
  if (cmdVel.angular.z != 0.1) {

    // Turn left
    if (cmdVel.angular.z > 0.1) {
      pwmLeftReq = -PWM_TURN;
      pwmRightReq = PWM_TURN;
    }
```

```
    // Turn right
    else {
      pwmLeftReq = PWM_TURN;
      pwmRightReq = -PWM_TURN;
    }
  }
  // Go straight
  else {
     // Remove any differences in wheel velocities
    // to make sure the robot goes straight
    static double prevDiff = 0;
    static double prevPrevDiff = 0;
    double currDifference = velLeftWheel - velRightWheel;
    double avgDifference = (prevDiff+prevPrevDiff+currDifference)/3;
    prevPrevDiff = prevDiff;
    prevDiff = currDifference;
    // Correct PWM values of both wheels to make the vehicle go straight
    pwmLeftReq -= (int)(avgDifference * DRIFT_MULTIPLIER);
    pwmRightReq += (int)(avgDifference * DRIFT_MULTIPLIER);
  }

  // Handle low PWM values
  if (abs(pwmLeftReq) < PWM_MIN) {
    pwmLeftReq = 0;
  }
  if (abs(pwmRightReq) < PWM_MIN) {
    pwmRightReq = 0;
  }
}
void set_pwm_values() {

  // These variables will hold our desired PWM values
  static int pwmLeftOut = 0;
  static int pwmRightOut = 0;
```

```
// If the required PWM is of opposite sign as the output PWM, we want to
// stop the car before switching direction
static bool stopped = false;
if ((pwmLeftReq * velLeftWheel < 0 && pwmLeftOut != 0) ||
    (pwmRightReq * velRightWheel < 0 && pwmRightOut != 0)) {
  pwmLeftReq = 0;
  pwmRightReq = 0;
}

// Set the direction of the motors
if (pwmLeftReq > 0) { // Left wheel forward
  digitalWrite(in4, HIGH);
  digitalWrite(in3, HIGH);
}
else if (pwmLeftReq < 0) { // Left wheel reverse
  digitalWrite(in4, LOW);
  digitalWrite(in3, LOW);
}
else if (pwmLeftReq == 0 && pwmLeftOut == 0 ) { // Left wheel stop
  analogWrite(enC, 0);
  analogWrite(enD, 0);
}
else { // Left wheel stop
  analogWrite(enC, 0);
  analogWrite(enD, 0);
}

if (pwmRightReq > 0) { // Right wheel forward
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
}
else if(pwmRightReq < 0) { // Right wheel reverse
  digitalWrite(in1, HIGH);
  digitalWrite(in2, HIGH);
}

else if (pwmRightReq == 0 && pwmRightOut == 0) { // Right wheel stop
  analogWrite(enA, 0);
analogWrite(enB, 0);
}
```

```
else { // Right wheel stop
   analogWrite(enA, 0);
  analogWrite(enB, 0);
  }

  // Increase the required PWM if the robot is not moving
  if (pwmLeftReq != 0 && velLeftWheel == 0) {
   pwmLeftReq *= 1.5;
  }
  if (pwmRightReq != 0 && velRightWheel == 0) {
   pwmRightReq *= 1.5;
  }

  // Calculate the output PWM value by making slow changes to the current
value
  if (abs(pwmLeftReq) > pwmLeftOut) {
   pwmLeftOut += PWM_INCREMENT;
  }
  else if (abs(pwmLeftReq) < pwmLeftOut) {
   pwmLeftOut -= PWM_INCREMENT;
  }

  else{}

  if (abs(pwmRightReq) > pwmRightOut) {
   pwmRightOut += PWM_INCREMENT;
  }

  else if(abs(pwmRightReq) < pwmRightOut) {
   pwmRightOut -= PWM_INCREMENT;
  }
  else{}

  // Conditional operator to limit PWM output at the maximum
  pwmLeftOut = (pwmLeftOut > PWM_MAX) ? PWM_MAX :
pwmLeftOut;
  pwmRightOut = (pwmRightOut > PWM_MAX) ? PWM_MAX :
pwmRightOut;
```

```
// PWM output cannot be less than 0
  pwmLeftOut = (pwmLeftOut < 0) ? 0 : pwmLeftOut;
  pwmRightOut = (pwmRightOut < 0) ? 0 : pwmRightOut;

  // Set the PWM value on the pins
  deb.data = pwmLeftOut;
  debug.publish(&deb);
  analogWrite(enC, pwmLeftOut);
  analogWrite(enD, pwmLeftOut);
  analogWrite(enA, pwmRightOut);
  analogWrite(enB, pwmRightOut);

}
void pub_imu(){

    if ((millis() - prev_imu_time) >= (1000 / IMU_PUBLISH_RATE)) {
    IMU.readSensor();

    // Header
    imu_msg.header.seq = sequence++;
    imu_msg.header.stamp = nh.now();
    imu_msg.header.frame_id = "imu";

    // Linear Acceleration
    imu_msg.linear_acceleration.x = IMU.getAccelX_mss();
    imu_msg.linear_acceleration.y = IMU.getAccelY_mss();
    imu_msg.linear_acceleration.z = IMU.getAccelZ_mss();

    // Angular Velocity
    imu_msg.angular_velocity.x = IMU.getGyroX_rads();
    imu_msg.angular_velocity.y = IMU.getGyroY_rads();
    imu_msg.angular_velocity.z = IMU.getGyroZ_rads();
```

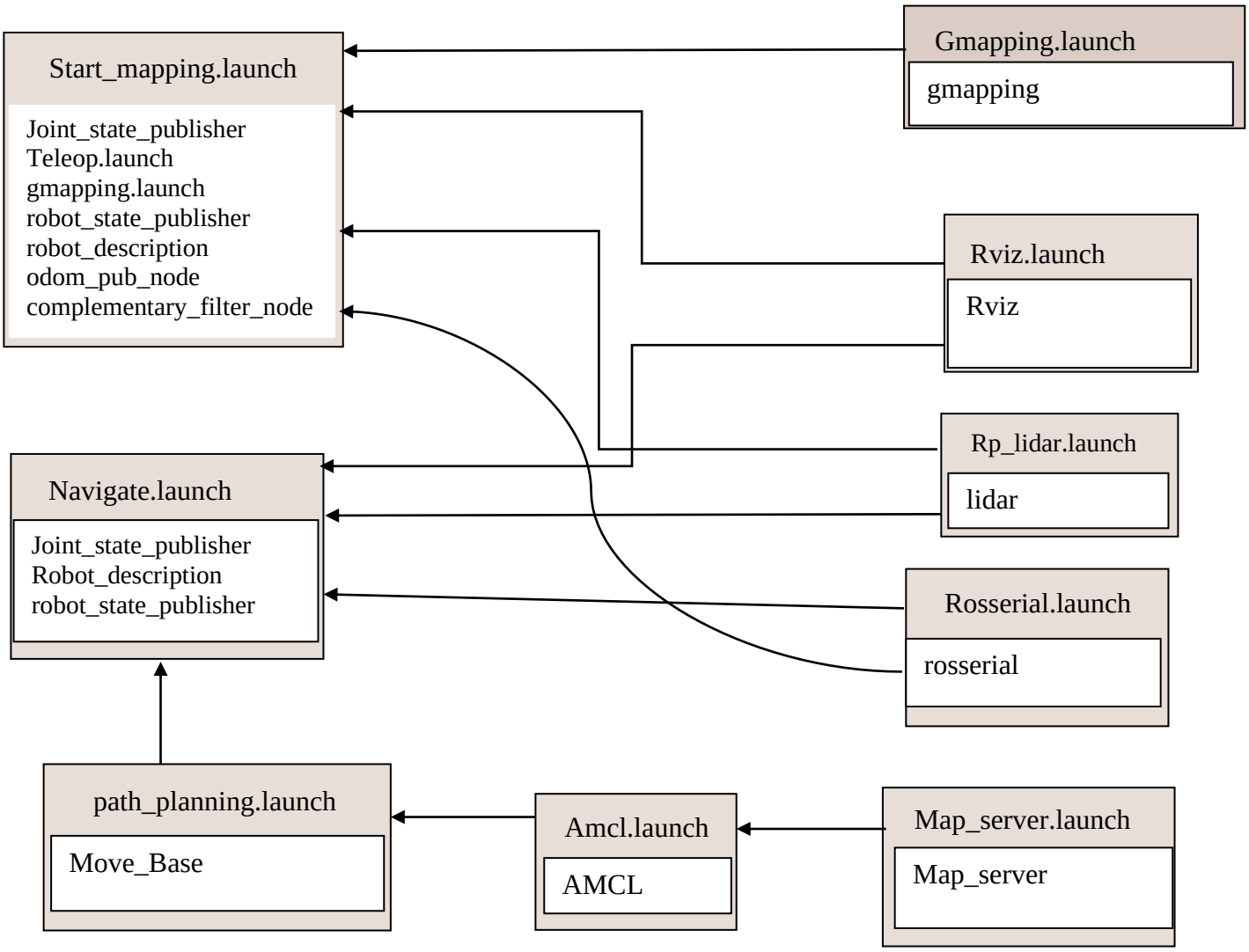
```
  // Orientation
    imu_msg.orientation.x = 0;
    imu_msg.orientation.y = 0;
    imu_msg.orientation.z = 0;
    imu_msg.orientation.w = 0;

    imu_pub.publish(&imu_msg);

    nh.spinOnce();

    prev_imu_time = millis();

  }

}
```

Appendix B - Project file Layout

```
├── build
├── devel
├── src
├── agribot
│  ├── config
│  │  ├── madgwick_filter_params.yaml
│  ├── launch
│  │  ├── Start_mapping.launch
│  │  └── Navigate.launch
│  ├── urdf
│  │  ├── agri.xacro
│  └── spots
│  │  └── n_spots.txt
│  │  └── s_spots.txt
│  ├── srv
│  │  └── AgriServiceMessage.srv
│  ├── src
│  │  └── compute_odom.py
│  │  └── get_coordinates_service_server.py
│  │  └── send_coordinate_action_client.py
│  │  └── spots_to_file.py
│  │  └── teleop.py
│  ├── README.md
│  ├── package.xml
│  ├── CMakeLists.txt
├── my_agri_navigation
│  └── config
│  │  └── costmap_common_params.yaml
```

```
│   │   └── global_costmap_params.yaml
│   │   └── local_costmap_params.yaml
│   │   └── move_base_params.yaml
│   ├── launch
│   │   └── amcl.launch
│   │   └── map_server.launch
│   │   └── path_planning.launch
│   │   └── gmapping.launch
│   └── map
│   │   └── map.pgm
│   │   └── map.yaml
│   └── param
│   │   └── params.yaml
│   ├── CMakeLists.txt
│   └── package.xml
├── rosserial
│   ├── launch
│   │   └── rosserial.launch
│   └── package.xml
│   └── CMakeLists.txt
├── rplidar_ros
│   │   ├── launch
│   └── rplidar.launch
│   ├── CMakeLists.txt
│   └── package.xml
│   ├── README.md
├── rviz_initial_pose
│   ├── launch
│   │   ├── rviz.launch
```

```
│ ├── src
│ │ ├── rviz_click_to_2d.cpp
```



Appendix C – lunch files Code

- Start_mapping.launch

```
<launch>
   <!-- send urdf to param server -->
   <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find
agribot)/urdf/omar.xacro'" />

   <!-- Send fake joint values-->
   <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher">
      <param name="use_gui" value="false"/>
   </node>

   <!-- Send robot states to tf -->
   <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" respawn="false" output="screen"/>
   <!--rosserial-->
   <include file="$(find rosserial)/launch/rosserial.launch" />
   <!--rviz-->
   <node name="rviz" pkg="rviz" type="rviz" />
   <!--start gmapping-->
   <include file="$(find my_agri_navigation)/launch/gmapping.launch" />
   <!--control the robot using keyboard-->
   <node pkg='agribot' name='teleop_keyboard' type='teleop.py' output='screen'/>
   <!--compute odometry-->
   <node pkg= 'agribot nam' = 'odom_pub_node type'='compute_odom.py output'= 'screen' />
   <!--imu_filter-->
   <node name="imu_filter_node_for_orientation" pkg="imu_complementary_filter"
type="complementary_filter_node" />
    <!--lidar-->
   <include file="$(find rplidar_ros)/launch/rplidar.launch" />
<remap from="odom" to="/odom" />
   <remap from="imu_data" to="imu/data" />
   <node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf">
   <param name="output_frame" value="odom"/>
   <param name="base_footprint_frame" value="base_footprint"/>
   <param name="freq" value="300.0"/>
   <param name="sensor_timeout" value="1.0"/>
   <param name="odom_used" value="true"/>
   <param name="imu_used" value="true"/>
   <param name="vo_used" value="false"/>
   <param name="gps_used" value="false"/>
   <param name="debug" value="false"/>
   <param name="self_diagnose" value="false"/>
   </node>

</launch>
```

- Navigate.launch

```
<launch>
   <!-- send urdf to param server -->
   <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find
agribot)/urdf/omar.xacro'" />

   <!-- Send fake joint values-->
   <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
      <param name="use_gui" value="false"/>
   </node>

   <!-- Send robot states to tf -->
   <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher"
respawn="false" output="screen"/>

   <!--rosserial-->
   <include file="$(find rosserial)/launch/rosserial.launch" />
   <!--compute odometry-->
   <include file="$(find odometery)/launch/odom.launch" />
   <!--imu_filter-->
   <node name="imu_filter_node_for_orientation" pkg="imu_complementary_filter"
type="complementary_filter_node" />
    <!--lidar-->
   <include file="$(find rplidar_ros)/launch/rplidar.launch" />

   <!--path_planning-->
   <include file="$(find my_agri_navigation)/launch/path_planning.launch" />

   <!--rviz-->
   <node name="rviz" pkg="rviz" type="rviz" />
<remap from="odom" to="/odom" />
   <remap from="imu_data" to="imu/data" />
   <node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf">
   <param name="output_frame" value="odom"/>
   <param name="base_footprint_frame" value="base_footprint"/>
   <param name="freq" value="300.0"/>
   <param name="sensor_timeout" value="1.0"/>
   <param name="odom_used" value="true"/>
   <param name="imu_used" value="true"/>
   <param name="vo_used" value="false"/>
   <param name="gps_used" value="false"/>
   <param name="debug" value="false"/>
   <param name="self_diagnose" value="false"/>
   </node>

</Launch>
```

- Amcl.launch

```
<launch>



  <arg name="use_map_topic" default="false"/>
  <arg name="scan_topic" default="scan"/>
  <include file="$(find my_agri_navigation )/launch/map_server.launch" />


  <node pkg="amcl" type="amcl" name="amcl" output="screen">
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha5" value="0.1"/>
    <param name="gui_publish_rate" value="10.0"/>
    <param name="laser_max_beams" value="30"/>
    <param name="min_particles" value="500"/>
    <param name="max_particles" value="5000"/>
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.99"/>
    <param name="odom_alpha1" value="0.2"/>
    <param name="odom_alpha2" value="0.2"/>
    <!-- translation std dev, m -->
    <param name="odom_alpha3" value="0.8"/>
    <param name="odom_alpha4" value="0.2"/>
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_model_type" value="likelihood_field"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d" value="0.2"/>
    <param name="update_min_a" value="0.5"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="resample_interval" value="1"/>
    <param name="transform_tolerance" value="0.1"/>
    <param name="recovery_alpha_slow" value="0.0"/>
    <param name="recovery_alpha_fast" value="0.0"/>
</node>


</launch
```

- Map_server.launch

```
<launch>
  <arg name='map_file' default="$(find my_agri_navigation)/map/map1.yaml" />
  <node pkg='map_server' name='map_server' type='map_server' args='$(arg map_file)' />
</launch>
```

- path_planning.launch

```
<launch>

  <!--AMCL and Map_server-->
  <include file="$(find my_agri_navigation)/launch/amcl.launch" />


  <!--path_planning-->
  <arg name="no_static_map" default="false"/>
  <arg name="base_global_planner" default="navfn/NavfnROS"/>
  <arg name="base_local_planner" default="dwa_local_planner/DWAPlannerROS"/>
    <param name="base_global_planner" value="$(arg base_global_planner)"/>
    <param name="base_local_planner" value="$(arg base_local_planner)"/>

    <rosparam file="$(find my_agri_navigation)/config/move_base_params.yaml"
command="load" />

    <rosparam file="$(find my_agri_navigation)/config/costmap_common_params.yaml"
command="load" ns="global_costmap" />

    <rosparam file="$(find my_agri_navigation)/config/costmap_common_params.yaml"
command="load" ns="local_costmap" />

    <rosparam file="$(find my_agri_navigation)/config/local_costmap_params.yaml"
command="load" ns="local_costmap" />

    <param name="local_costmap/width" value="3"/>

    <param name="local_costmap/height" value="3"/>

    <rosparam file="$(find my_agri_navigation)/config/global_costmap_params.yaml"
command="load" ns="global_costmap" unless="$(arg no_static_map)"/>


    <param name="controller_frequency" value="5.0" />

    <param name="controller_patience" value="15.0" />

  </node>

</launch>
```

- gmapping.launch

```
<launch>
  <arg name="scan_topic"  default="/scan" />
  <arg name="base_frame"  default="base_footprint"/>
  <arg name="odom_frame"  default="odom"/>
  <arg name="map_frame" default="map"/>
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="base_frame" value="$(arg base_frame)"/>
    <param name="odom_frame" value="$(arg odom_frame)"/>
    <param name="map_update_interval" value="5.0"/>    <!--Sets the time (in seconds) to wait
until update the map.-->
    <param name="maxUrange" value="3.0"/>             <!--The maximum range of the laser
scanner used in the mapping process.-->
    <param name="maxRange" value="5.0"/>              <!--The maximum range of the laser
scanner used in the laser likelihood field computation.-->
    <param name="sigma" value="0.05"/>                <!--The standard deviation of the noise in
the laser range measurements.-->
    <param name="kernelSize" value="1"/>              <!--The size of the kernel used for laser
likelihood field computation.-->
    <param name="lstep" value="0.05"/>                <!--The linear resolution used in the laser ray
casting process.-->
    <param name="astep" value="0.05"/>                <!--The angular resolution used in the laser
ray casting process.-->
    <param name="iterations" value="5"/>              <!--The number of iterations used for laser
ray casting.-->
    <param name="lsigma" value="0.075"/>              <!--The standard deviation of the noise in
the robot's motion estimates.-->
    <param name="ogain" value="3.0"/>                 <!--The gain used in the calculation of the
laser scan's likelihood.-->
    <param name="lskip" value="0"/>                   <!--The number of laser scans to skip before
processing.-->
    <param name="minimumScore" value="200"/>          <!--The minimum score required for a
laser scan to be considered valid.-->
    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>
    <param name="linearUpdate" value="0.1"/>          <!--Sets the linear distance that the robot
has to move in order to process a laser reading.-->
    <param name="angularUpdate" value="0.2"/>         <!--Sets the angular distance that the
robot has to move in order to process a laser reading.-->
    <param name="temporalUpdate" value="-1.0"/>       <!--Sets the time (in seconds) to wait
between laser readings. If this value is set to -1.0, then this function is turned off.-->
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="80"/>
```

```
    <param name="xmin" value="-5.0"/>
    <param name="ymin" value="-5.0"/>
    <param name="xmax" value="5.0"/>
    <param name="ymax" value="5.0"/>

    <param name="delta" value="0.1"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>
</launch>
```

- rosserial.launch

```
<launch>

        <node pkg="rosserial_python" type="serial_node.py" name="serial_node" output="screen">
          <param name="port" value="/dev/esp" />
          <param name="baud" value="57600" />
          <param name="buffersize" value="1024" />
        </node>



        <node pkg="rosserial_python" type="serial_node.py" name="serial_node_1"
output="screen">
          <param name="port" value="/dev/a_mega" />
          <param name="baud" value="57600" />
          <param name="buffersize" value="1024" />
        </node>


</launch>
```

- Rplidar.launch

```
<launch>
  <node name="rplidarNode"          pkg="rplidar_ros"  type="rplidarNode" output="screen">
  <param name="serial_port"         type="string" value="/dev/ttyUSB2"/>
  <param name="serial_baudrate"     type="int"    value="115200"/>
  <param name="frame_id"            type="string" value="laser"/>
  <param name="inverted"            type="bool"   value="false"/>
  <param name="angle_compensate"    type="bool"   value="true"/>
  </node>
</launch>
```

- Rviz.launch

```
<launch>
    <!--rviz-->
    <node name="rviz" pkg="rviz" type="rviz" />

</launch>
```

Appendix D – Components & Cost

List of components for your seed sowing robot project

1. DC motors with gearbox
2. 4 wheels
3. DC motor with encoder
4. 3D printing parts
5. 4 holders for motors
6. 2 servo motors
7. Chassis
8. ESP32 microcontroller
9. MPU9250 sensor (accelerometer and gyroscope)
10. RPLidar A1M8 LiDAR sensor
11. Raspberry Pi 4 (for higher-level processing)
12. Arduino Mega (for low-level motor control)
13. DC to DC buck converter
14. USB HUB (BYEASY)
15. Water pump
16. Emergency button
17. TFT LCD display
18. Keypad for user input
19. 12V lithium-ion DC battery
20. Wire connectors
21. Current sensor
22. Cytron MDD 10A motor drivers
23. L298N motor driver
24. Acrylic cover for protection
25. Steel auger

Cost Table

Component	Quantity	Cost
DC motors with gearbox	4	5400
Wheels	4	880
DC motors with encoder	2	750
3D printing parts	-	1500
Holders for motors	4	240
Servo motors	2	300
Chassis	1	1000
ESP32 microcontroller	1	300
MPU9250 sensor (accelerometer and gyroscope)	1	350
RPLidar A1M8 LiDAR sensor	1	4000
Raspberry Pi 4	1	5000
Arduino Mega	1	550
DC to DC buck converter	1	125
USB HUB (BYEASY)	1	400
Water pump	1	200
Emergency button	1	50
TFT LCD display	1	260
Keypad for user input	1	125
12V lithium-ion DC battery	1	850
Wire connectors	2	70
Current sensor	1	65
Cytron MDD 10A motor drivers	2	1700
L298N motor driver	1	150
Acrylic cover	1	600
Steel auger	1	250
Total Cost	-	**23915**

Appendix E - TimeTable Plan

PROJECT NAME	PROJECT DURATION	PROJECT START DATE	PROJECT END DATE
Seed Sowing Robot	227	Oct 22, 2022	Jun 06, 2023

Task ID	Task Description	Duration	Start Date	End Date
1	Draw Prototype for platform desig	4	Oct 22, 2022	Oct 26, 2022
2	Components Selection	4	Oct 27, 2022	Oct 31, 2022
3	Mechanical calculations	14	Nov 01, 2022	Nov 15, 2022
4	Drawing parts in Solidworks	17	Nov 16, 2022	Dec 03, 2022
5	Make analysis for each part	3	Dec 04, 2022	Dec 07, 2022
6	Select needed motors	3	Dec 08, 2022	Dec 11, 2022
7	Material Selection	7	Dec 12, 2022	Dec 19, 2022
8	Find manufacturing workshop	10	Dec 20, 2022	Dec 30, 2022
9	Search for selected motor in shop	12	Dec 31, 2022	Jan 12, 2023
10	Read more about Ubuntu , ROS	12	Jan 13, 2023	Jan 25, 2023
11	Installation Ubuntu & Ubuntu MA	5	Jan 26, 2023	Jan 31, 2023
12	Create Packages and nodes	8	Feb 01, 2023	Feb 09, 2023
13	Read more about Gazebo and Rvi	9	Feb 10, 2023	Feb 19, 2023

Task ID	Task Description	Duration	Start Date	End Date
1	Networking	5	Feb 18, 2023	Feb 23, 2023
2	Robot Kinematics	5	Feb 24, 2023	Mar 01, 2023
3	Simulation using Gazebo and	4	Mar 02, 2023	Mar 06, 2023
4	Odometry	15	Mar 07, 2023	Mar 22, 2023
5	Control	5	Mar 23, 2023	Mar 28, 2023
6	Wire Up	6	Mar 29, 2023	Apr 04, 2023
7	Sensor Fusion	15	Apr 05, 2023	Apr 20, 2023
8	Robot Transforms	10	Apr 21, 2023	May 01, 2023
9	SLAM	5	May 02, 2023	May 07, 2023
10	Drive Robot with teleop	2	May 08, 2023	May 10, 2023
11	Autonomous Navigation	10	May 11, 2023	May 21, 2023
12	Apply AMCL	2	May 22, 2023	May 24, 2023
13	Path Planning	5	May 25, 2023	May 30, 2023
14	Hardware Assemply	5	May 31, 2023	Jun 05, 2023
15	Hardware implemntation on	2	Jun 06, 2023	Jun 08, 2023
16	Finalization	3	Jun 09, 2023	Jun 12, 2023
17	Testing of final product	5	Jun 13, 2023	Jun 18, 2023

References

[1] F. Dong, O. Petzold, W. Heinemann, et al. Time-optimal guidance control for an agricultural robot with orientation constraints. Comput. Electron. Ag. 99(2013), 124-131.

[2] T. Bakker, K. A. Van, J. Bontsema, et al. Systematic design of an autonomous platform for robotic weeding. J. Terramechanics, 47(2010), 63-73.

[3] R. González, F. Rodríguez, J. Sánchezhermosilla, et al. Navigation techniques for mobile robots in greenhouses. Appl. Eng. Agric. 25(2009), 153-165.

[4] S. S. Mehta, T. F. Burks, W. E. Dixon, Vision-based localization of a wheeled mobile robot for greenhouse applications: a daisy-chaining approach. Comput. Electron. Ag. 63(2008), 28-37.

[5] S. Janos, I. Matijevics, Implementation of potential field method for mobile robot navigation in greenhouse environment with WSN support. International Symposium on Intelligent Systems and Informatics, New York, 2010, pp.319-323.

[6] P. Dario, G. Sandini, B. Allotta, et al. The AGROBOT projects for greenhouse automation. Acta Horticulturae, 361(1994), 85-92

[7] A. Mandow, J. M. Gómez-De-Gabriel, J. L. Martínez, et al. The autonomous mobile robot Aurora for greenhouse operation. IEEE Robot. Autom. Mag., 3(1996), 18-28.

[8] S. Singh, W. S. Lee, T. F. Burks, Autonomous robotic vehicle development for greenhouse spraying. Transactions of the ASAE, 48(2005), 2355-2361.

[9] E. J. V. Henten, J. Hemming, B. A. J. V. Tuijl, et al. An autonomous robot for harvesting cucumbers in greenhouses. Auton. Robot. 13(2002), 241-258.

[10] P. Rajendra, N. Kondo, K. Ninomiya, et al. Machine vision algorithm for robots to harvest strawberries in tabletop culture greenhouses. Engineering in Agriculture, Environment and Food, 2(2009), 24-30.

[11] S. Kitamura, K. Oka, Recognition and cutting system of sweet pepper for picking robot in greenhouse horticulture. IEEE International Conference on Mechatronics and Automation. New York, 2005, pp. 1807-1812.

[12] M. H. Ko, B. Ryuh, K. C. Kim, et al. Autonomous greenhouse mobile robot driving strategies from system integration perspective: review and application. IEEE-ASME T. Mech. 20(2014), 1-12.

[13] A. Sulakhe, M. N. Karanjkar, Design and operation of agricultural based pesticide spraying robot. International Journal of Science and Research, 4(2013), 1286-1289. 5

[14] M. Monta, N. Kondo, Y. Shibano, Agricultural robot in grape production system. IEEE International Conference on Robotics and Automation. New York, 1995, pp. 2504-2509.

[15] G. Belforte, R. Deboli, P. Gay, et al. Robot design and testing for greenhouse applications. Biosyst. Eng. 95(2006), 309-321.

[16] V. Sankaranarayanan, A. D. Mahindrakar, Switched control of a nonholonomic mobile robot. Commun. Nonlinear Sci. 14(2009), 2319-2327.

[17] Chen, J., Li, Y., & Li, X. (2019). Design and implementation of an autonomous mobile robot for seed sowing in field environments. Journal of Intelligent & Robotic Systems, 93(3-4), 529-546.

[18] Cheng, X., Zhang, Y., Wang, Q., Li, S., & Liu, Y. (2020). Research on localization technology for autonomous seed sowing robot. Transactions of the Chinese Society of Agricultural Engineering, 36(14), 171-179.

[19] Xu, X., Li, X., & Li, J. (2021). A Robust Localization System for Autonomous Seed Sowing Mobile Robots in Complex Field Environments. IEEE Transactions on Instrumentation and Measurement, 70, 1-11.