FULLY AUTONOMOUS SOLAR PANELS CLEANING ROBOT

A Thesis submitted in partial fulfilment of the requirements of

Bachelor of Mechanical Engineering

(Mechatronics Engineering Department)

By

Ahmed Abdelaal Saleh

Ahmed Essam Sayed

Aya Ashraf Farag

Ali Noureldin Abdelaty

Hafsa Mohamed Taha

Mohamed Nasser Mohamed

Amal Samir Ismail

Maha Yehia Mostafa

Supervised By

Dr. Rowida Meligy

Cairo, 2024

FULLY AUTONOMOUS SOLAR PANELS CLEANING ROBOT

A Thesis submitted in partial fulfilment of the requirements of

Bachelor of Mechanical Engineering

(Mechatronics Engineering Department)

By

Ahmed Abdelaal Saleh

Ahmed Essam Sayed

Aya Ashraf Farag

Ali Noureldin Abdelaty

Hafsa Mohamed Taha

Mohamed Nasser Mohamed

Amal Samir Ismail

Maha Yehia Mostafa

Supervised By

Dr. Rowida Meligy

Cairo, 2024

Abstract

This project presents the design and development of a fully automated solar panel cleaning robot aimed at enhancing the efficiency and longevity of solar power systems. Solar panels are crucial for sustainable energy production, but their performance can be significantly hindered by dust and debris accumulation. Regular cleaning is essential to maintain optimal efficiency, but manual cleaning is labor-intensive, time-consuming, and often impractical for large installations.

The proposed robot addresses these challenges with a robust and innovative design. The robot features a steel chassis and a tracked motion system with a specialized suspension to navigate various terrains and panel surfaces. The cleaning mechanism consists of an arm equipped with a brush, controlled by linear motors to ensure precise and effective cleaning.

Key components of the robot include:

- Mechanical System: A steel chassis provides durability and stability, while a tracked motion system and advanced suspension ensure smooth navigation and cleaning efficiency.
- Control System: The robot is powered by the Robot Operating System (ROS), employing LIDAR for environmental mapping, and utilizing SLAM for autonomous navigation. Motor control is managed through PID controllers, with encoders for speed monitoring and an IMU for orientation and location adjustments.
- Cleaning Mechanism: The arm's movements are precisely controlled by linear motors, with potentiometers and reed sensors for angle adjustment, ensuring thorough cleaning of the solar panels.

The development of this robot is driven by the need for an efficient, reliable, and autonomous solution to maintain solar panel efficiency. By automating the cleaning process, the robot reduces labor costs, minimizes downtime, and enhances the overall productivity of solar power systems. This project not only contributes to the advancement of renewable energy technologies but also paves the way for future innovations in automated maintenance systems.

Table of Contents

List of tables

Table of Figures

LIST OF APPENDICES

Chapter 1

INTRODUCTION

1.1 Overview

Solar energy is a cornerstone of sustainable development, offering a renewable and clean source of power. Solar panels are pivotal in harnessing this energy, converting sunlight into electricity efficiently. However, the efficiency of solar panels can be significantly impacted by dust, dirt, and other environmental contaminants. Regular cleaning is essential to maintain optimal performance, yet manual cleaning methods are labor-intensive and impractical for large installations. This project addresses this challenge through the design and development of an autonomous solar panel cleaning robot. Our robot leverages advanced mechanical design and sophisticated control systems to ensure efficient, reliable, and autonomous cleaning.

Figure 1.1.1: Aerial view of a solar farm highlighting clean vs. dirty panels

1.2 The Importance of Solar Panels

Solar panels play a crucial role in the global shift towards renewable energy. By converting sunlight into electricity, they provide a sustainable and eco-friendly energy source, reducing reliance on fossil fuels and lowering greenhouse gas emissions. The widespread adoption of solar panels is essential for meeting global energy demands sustainably and mitigating the impacts of climate change.

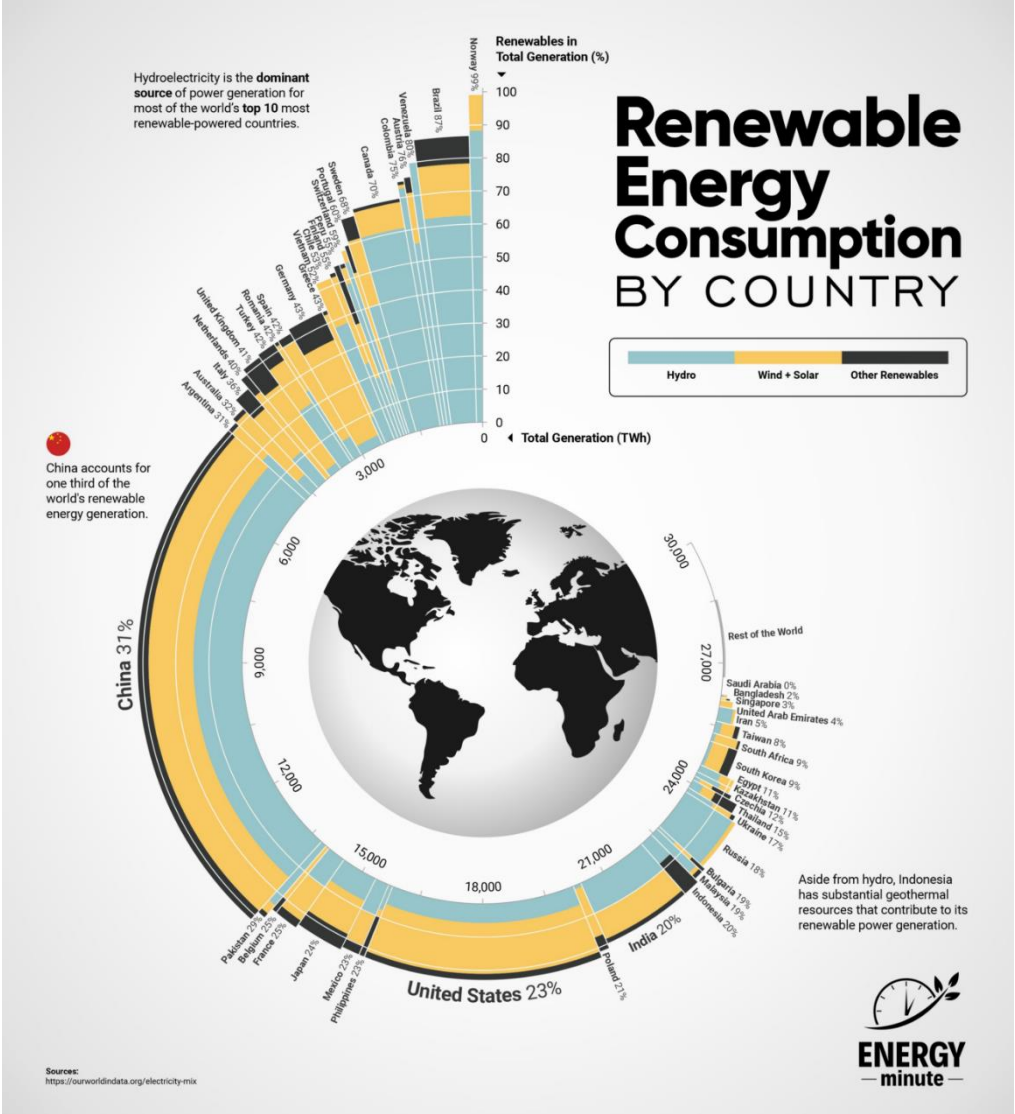


Figure 1.2.1: Infographic showing the growth of solar energy consumption worldwide

1.3 The Problem: Efficiency Loss Due to Contamination

One of the major challenges in maintaining solar panel efficiency is the accumulation of dirt, dust, bird droppings, and other contaminants on the surface of the panels. These obstructions can significantly reduce the panels' ability to absorb sunlight and convert it into electricity. Studies have shown that even a small amount of dirt can lead to substantial efficiency losses. Manual cleaning, though effective, is not feasible for large solar farms due to the high labor costs and logistical challenges.

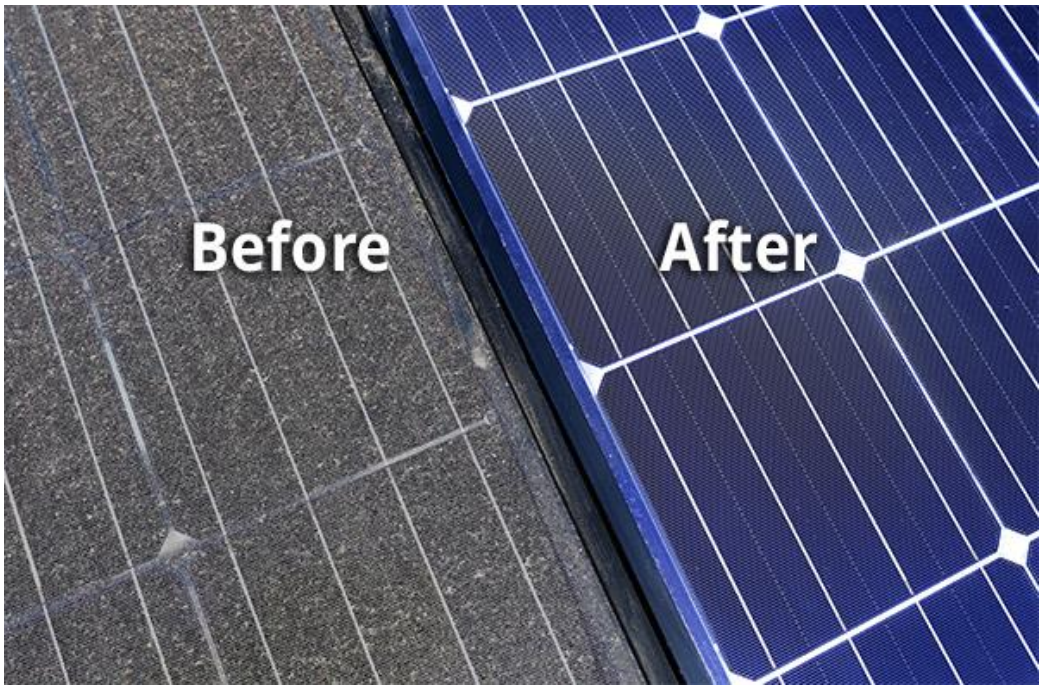


Figure 1.3.1: Close-up image of dirty vs. clean solar panels

1.4 The Solution: Autonomous Cleaning Robot

To address this problem, we have developed an autonomous robot designed specifically for cleaning solar panels. This robot combines advanced mechanical engineering, control systems, and robotics to provide a practical and efficient solution. The key features and capabilities of the robot include:

- **Autonomous Navigation:** The robot can navigate large solar farms independently, using advanced sensors and control systems.
- **Efficient Cleaning Mechanism:** Equipped with a long arm and brush, the robot ensures thorough cleaning of solar panels.
- **Adaptability:** The robot can adjust its cleaning mechanism to accommodate different panel orientations and environmental conditions.
- **Durability:** Designed to withstand harsh environmental conditions, the robot ensures reliable long-term operation.

1.5 Project Objectives

The primary objective of this project is to design, manufacture, and implement an autonomous robot capable of effectively cleaning solar panels. The robot is designed to:

1. Navigate various terrains autonomously.
2. Adjust its cleaning mechanism to accommodate different panel orientations.
3. Operate with minimal human intervention.
4. Ensure durability and reliability in harsh environmental conditions.

1.6 Motivation

The motivation for this project arises from the growing reliance on solar energy and the associated need for efficient maintenance solutions. Clean solar panels are crucial for maximizing energy output, as even minor obstructions can lead to significant efficiency losses. Manual cleaning methods are not only labor-intensive but also pose logistical challenges, especially for large solar farms. An autonomous cleaning robot offers a scalable and innovative solution, ensuring consistent maintenance and optimal panel performance.

Figure 1.6.1: Labors manually clean the solar panels

1.7 Literature Review

Existing approaches to solar panel cleaning range from manual methods to semi-automated systems. However, fully autonomous robots are relatively new and present unique challenges and opportunities. Studies emphasize the importance of integrating advanced sensors and control systems for reliable autonomy. Current solutions often lack the robustness and adaptability needed for varied environmental conditions, highlighting the necessity for a more comprehensive design.

1.8 Design and Development

Our robot's design focuses on robustness, efficiency, and adaptability. Key components include:

- **Tracked Motion System:** Provides stable and efficient movement across various surfaces, ensuring the robot can navigate different terrains encountered in solar farms.
- **Suspension System:** Enhances the robot's ability to traverse uneven terrain while maintaining contact with the solar panels, ensuring consistent cleaning.
- **Steel Chassis:** Offers durability and strength to withstand harsh environmental conditions, ensuring the robot's longevity.

The cleaning mechanism is designed to be efficient and adaptable, featuring a long arm equipped with a brush. The arm is driven by linear motors, allowing for precise adjustments to ensure thorough cleaning.

1.9 Control Systems and Autonomy

The robot's autonomy is achieved through the integration of the Robot Operating System (ROS) and various sensors and control systems:

- **LIDAR:** Used for environmental mapping and obstacle detection, enabling the robot to navigate complex environments.
- **PID Controllers:** Ensure precise control of the motors, enhancing the robot's stability and responsiveness.
- **Encoders:** Measure the speed of the robot, facilitating accurate motion control.

- **IMU (Inertial Measurement Unit):** Provides data on the robot's orientation and location, crucial for navigation and positioning.

These components are integrated to implement a Simultaneous Localization and Mapping **(SLAM)** model, which allows the robot to map its environment in real-time, adapt to changes, and perform cleaning tasks with high precision. The SLAM model enables the robot to navigate autonomously, avoiding obstacles and efficiently covering the area to be cleaned.

1.10 Cleaning Mechanism

The cleaning arm is a critical component designed for efficiency and adaptability. Key features include:

- **Linear Motors:** Drive the arm's movements, allowing for precise adjustments to reach various panel orientations.
- **Potentiometers and Reed Sensors:** Control the arm's angle, ensuring optimal cleaning contact with the panels. These sensors provide feedback to adjust the arm's position dynamically, ensuring thorough and consistent cleaning.

The arm's design ensures it can adapt to different panel configurations and orientations, maximizing cleaning efficiency.

1.11 Challenges and Solutions

Several challenges were encountered during the design and development process, each addressed with innovative engineering solutions:

- **Terrain Navigation:** The tracked motion system and advanced suspension design allow the robot to navigate uneven and varied terrains commonly found in solar farms.
- **Autonomous Operation:** Integration of ROS, LIDAR, and SLAM technologies ensures reliable and precise autonomous operation.
- **Efficient Cleaning:** The adjustable arm, driven by linear motors and controlled by potentiometers and reed sensors, ensures thorough cleaning of solar panels.

These solutions collectively contribute to the robot's robustness, efficiency, and autonomy.

1.12 Conclusion

This project represents a significant advancement in solar panel maintenance technology. By integrating advanced mechanical design with sophisticated control systems, we have developed an autonomous robot capable of reliable and efficient cleaning. The use of ROS and SLAM technologies ensures the robot can navigate and operate autonomously, making it a practical solution for large-scale solar farms. This project not only addresses a critical need in the solar energy sector but also demonstrates the potential of robotics and automation in solving complex real-world problems.

Overall, the successful development of this autonomous solar panel cleaning robot highlights the potential for innovative engineering solutions to enhance the efficiency and sustainability of renewable energy systems. The robot's ability to maintain optimal solar panel performance with minimal human intervention underscores the importance of automation in advancing renewable energy technologies.

Chapter 2

MECHANICAL DESIGN AND MANUFACTURING

OF THE CHASSIS AND THE MOTION SYSTEM

2.1 Mechanical designing and manufacturing

FIRST: MECHANICAL DESIGNING

The mechanical design of a solar panel cleaning robot is a multifaceted and intricate process, essential for ensuring the robot's efficiency and reliability. This design encompasses three main phases: the chassis, the motion system, and the arm carrying the solar panel cleaning brush. Each phase requires meticulous planning and integration of various mechanical components and systems to achieve optimal performance. This introduction provides an in-depth overview of each phase and the key considerations involved in the mechanical design of a solar panel cleaning robot.

Figure 2.1.1: Final robot design

Figure 2.1.2: Full robot after manufacturing

2.2 Design of the CHASSIS

The chassis serves as the foundational structure of our autonomous robot. It is the backbone that supports and integrates various components, ensuring the overall stability, durability, and functionality of the system.

In essence, the chassis plays a pivotal role in determining the performance and safety of the robot and the system overall.

In our project we agreed to maintain the chassis as simple as possible to keep the manufacturing process as easy as possible achieving the required functionality.

Choosing of this material with these specs and dimensions was after several stress analysis tests on different dimensions and the choice of the material was according to the most available material in the market to be sure of the availability of this material in the future.

We chose steel tubes with the following specifications:

Type	Material	Dimensions	Thickness	Yield strength	Tensile strength
Square hollow tube	Steel (37)	30 x 30 (mm)	1.5 (mm)	235 MPa	360 MPa

The chosen dimensions were confirmed after deciding the dimensions of the holes that we need to drill in the bar and also to maintain the costs as much as possible.

After that we have gone through many stresses analysis on the chassis overall to ensure that it will be reliable and able to withstand the loads, impacts and to be rigid.

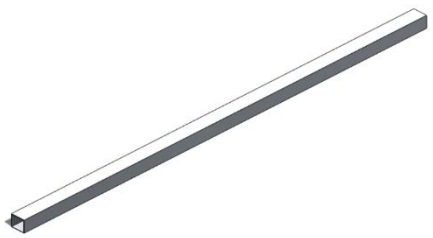


Figure 2.2.1: Metal bar used in the chassis

The following figure shows our chassis design.

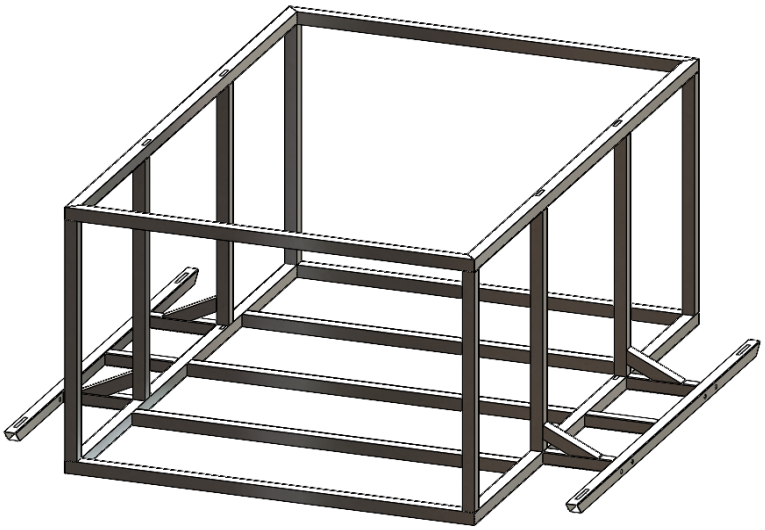


Figure 2.2.2: Complete chassis shape

- We have designed this chassis according to our needs, it consists of many square tubes welded together.
- We designed **two wings** coming out from the chassis to carry the tracked chains.
- We made many **Stresses analysis** on the wings to ensure that it can carry the tracked chains efficiently.
- By applying a total of **(140 Kgf)** on each wing which is way more than what we expect to be loaded on the wing and even the chassis, so we found these following results:

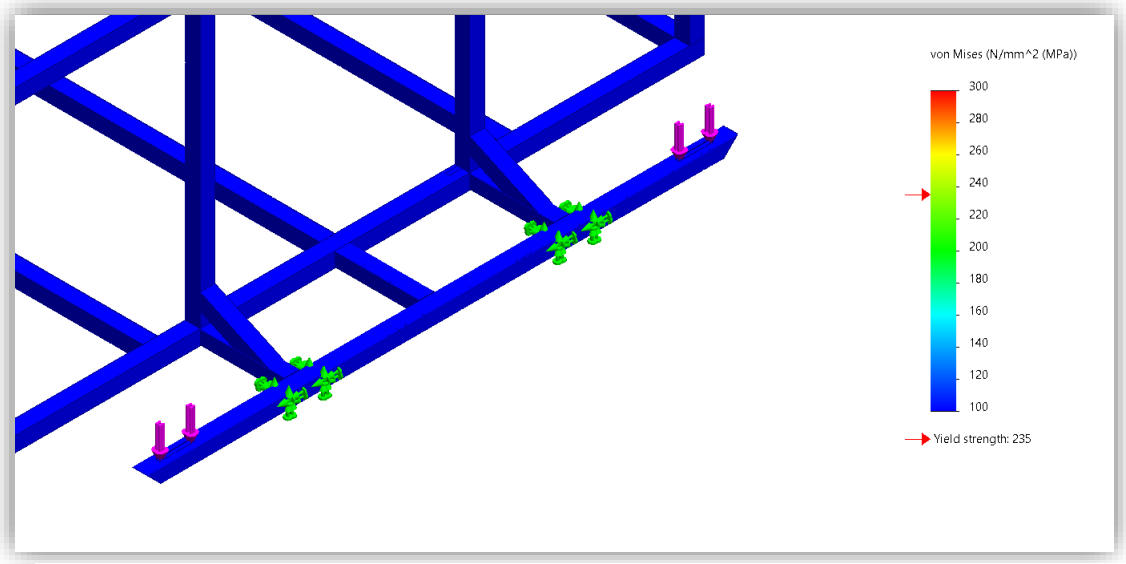


Figure 2.2.3: Stress on the chassis wing

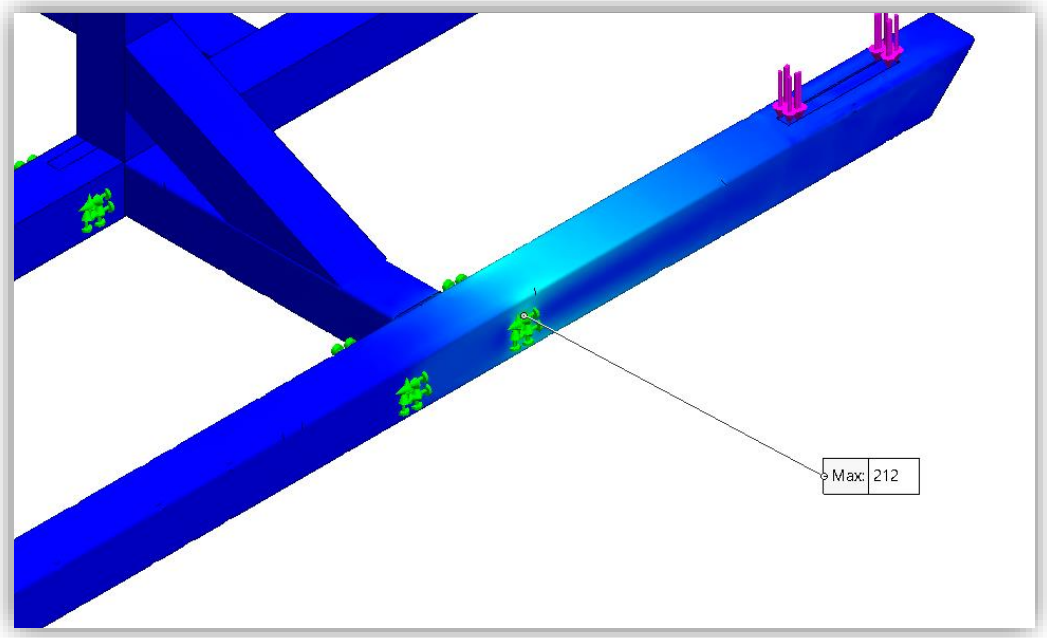


Figure 2.2.4: Maximum stress applied on the wing

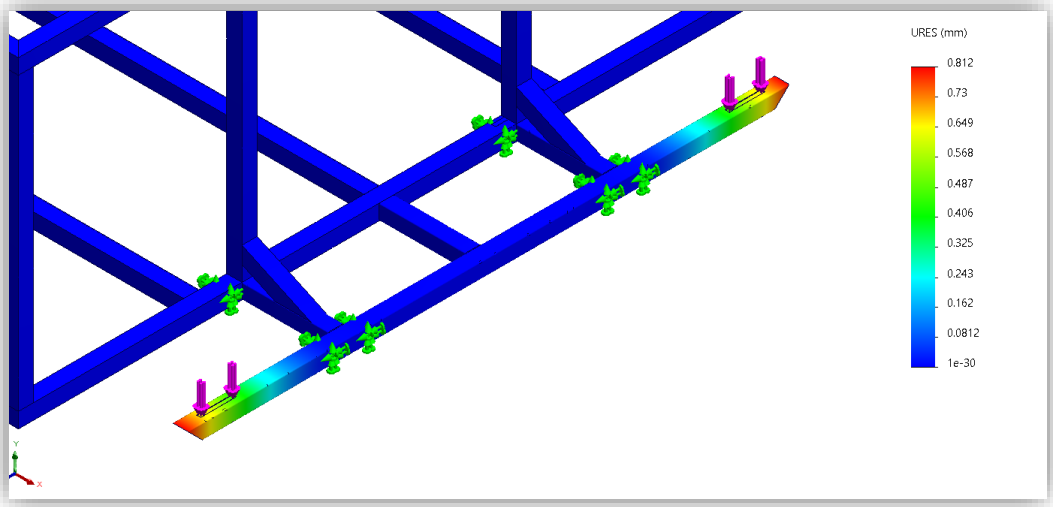


Figure 2.2.5: Deformation of the wing

These results show that:

- **Maximum stress** applied on the wing is **(212 MPa)** which is a number lower than the yield strength of the used material, so it is considered safe.
- **Maximum deformation** is only **(0.812 mm)** which is a very negligible number and doesn't affect the structure of the chassis at all.

Then we made another analysis on all parts of the chassis that we think will encounter stresses and loads and these are the results: The applied force is total (**400 Kgf**) on the chassis and (**280 Kgf**) on both wings

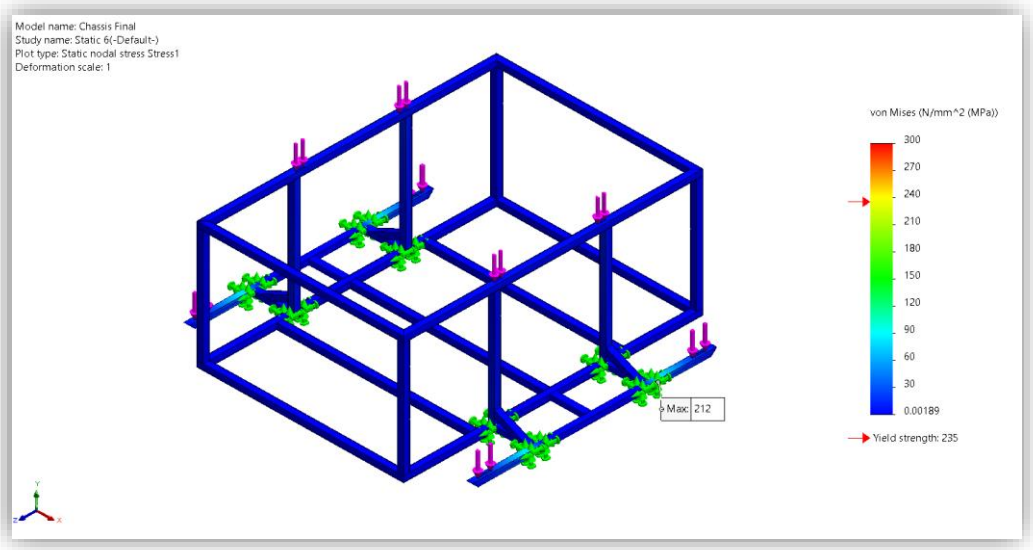


Figure 2.2.6: Stress applied on the chassis

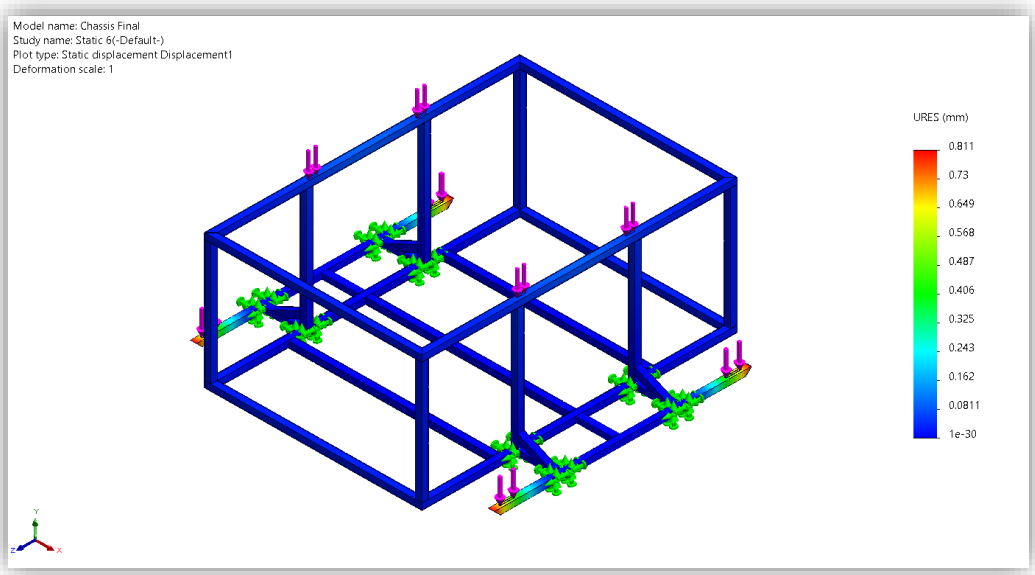


Figure 2.2.7: Deformation of the chassis

These results show that:

- **The maximum stress** applied on the chassis is **(212 MPa),** which is a number lower than the yield strength of the used material, so it is considered safe.
- **Maximum deformation** is only **(0.811 mm)** which is a very negligible number and doesn't affect the structure of the chassis at all.

2.3 Design of the MOTION SYSTEM

The motion system in our robot consists of 3 main parts which are:

1- Motors and load calculations
2- Driving gears, chains and shafts
3- Track chain

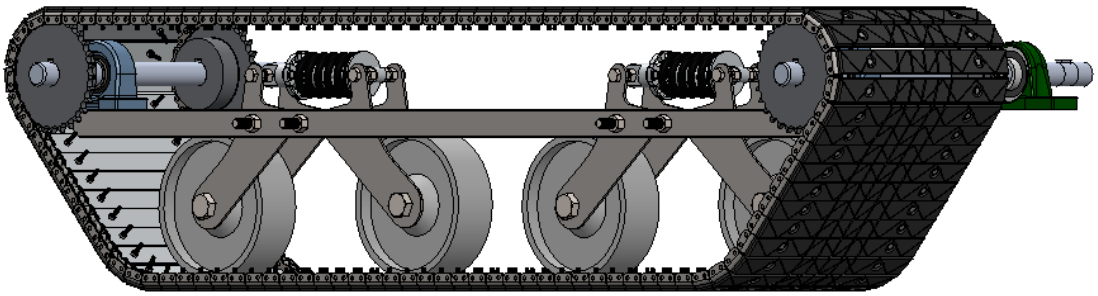


Figure 2.3.1: Motion system final shape

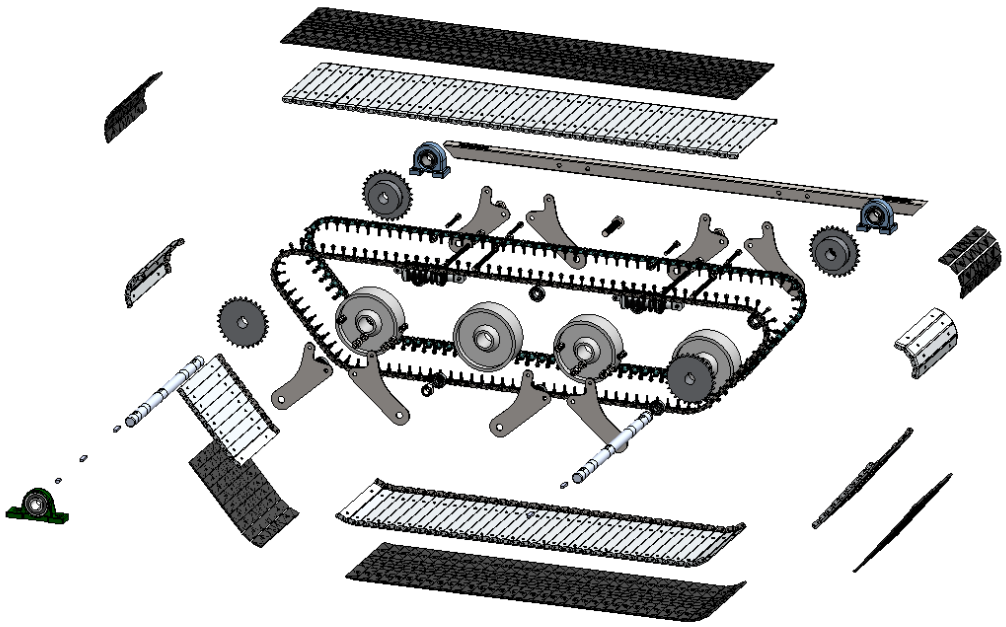


Figure 2.3.2: Motion system components

2.3.1 MOTORS AND LOAD CALCULATIONS

The efficiency and effectiveness of a solar panel cleaning vehicle are highly dependent on the selection of the appropriate motor. The motor is the heart of the vehicle's propulsion system, responsible for driving the tracks and powering other essential components. Selecting the right motor requires careful consideration of various factors, including the vehicle's operational requirements, the terrain of the solar farm, and the specific needs of the cleaning mechanism.

The process of motor selection involves determining the power and torque requirements based on the vehicle's weight, the type of terrain it will navigate, and the desired speed of operation. Additionally, the motor must be compatible with the vehicle's power supply, whether it be electric, hydraulic, or another type. Other considerations include the motor's efficiency, reliability, and ease of maintenance, all of which contribute to the overall performance and longevity of the vehicle.

Calculating the appropriate motor specifications involves a series of engineering analyses and mathematical computations. These calculations ensure that the motor can deliver the necessary power and torque to overcome various operational challenges, such as inclines, obstacles, and the resistance of the cleaning mechanism itself. By accurately determining these requirements, engineers can select a motor that ensures the vehicle operates smoothly, efficiently, and reliably in the demanding environment of a solar farm.

2.3.1.1 Weight assumption

The weight of the robot is assumed to be around **150 kg** so, the calculations were made on a weight of **200 kg** to keep safety factor. This weight influences various design aspects, such as the suspension system, motor selection, and overall structural integrity. The robot's weight ensures stability and efficient cleaning performance, balancing between being heavy enough to maintain traction and light enough for efficient mobility. Additionally, the chosen weight supports the durability and robustness needed for operating in different environmental conditions. Therefore, the 200 kg weight assumption is a pivotal factor in achieving the desired functionality and reliability of the robot.

2.3.1.2 Forces calculations

Determining the required power to move this weight.

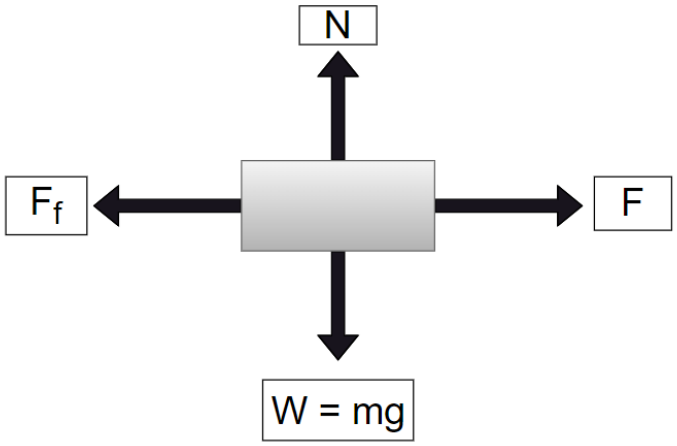


Figure 2.3.3: Forces acting on the robot

- Weight (W) = mg = 200 * 9.8 = 1960 N
- N = W = 1960 N
- Friction force (F_f) = μ N
- Let μ = 0.8. (Coefficient of friction between tires and roads)
- F_f = 0.8 * 1960 = 1568 N
- Force (F) = F_f = 1568 N (But let F = 1600 N)

The weight is to be Moved by 2 driving track chains so, as the weight is in the center of the robot.

Then, a force (F_{TC}) of (1600/2 = 800 N) is to be applied on each driving track chain.

In our case, the only available motors are with the following specifications:

- Power = 500 watts.
- Speed (n) = 200 RPM.
- Voltage = 24v.
- This motor has an output torque: $T_m = \frac{P}{\omega} = \frac{P}{\frac{2\pi(n)}{60}} = \frac{500}{\frac{2\pi(200)}{60}} \approx 24\ N.M$

Assume that our sprocket will have a radius (r): r = 0.06 m

Torque required to move each driving chain:

- $T_{TC} = F_{TC} * r$ = 800 N * 0.06 m = 48 N.M.

As we only have **1** motor for each driving chain then, the output torque of the motor will not be enough to withstand the required torque. So, we must do some speed reduction to increase the motor torque to be able to move our robot.

Achieving this torque requires a speed reduction system with a new gear ratio.

- The required gear ratio for each chain: $i = \frac{T_r}{T_m} = \frac{48}{24} = 2$
- Speed will be reduced to be: $n_2 = \frac{n1}{i} = \frac{200}{2} = 100\ RPM$

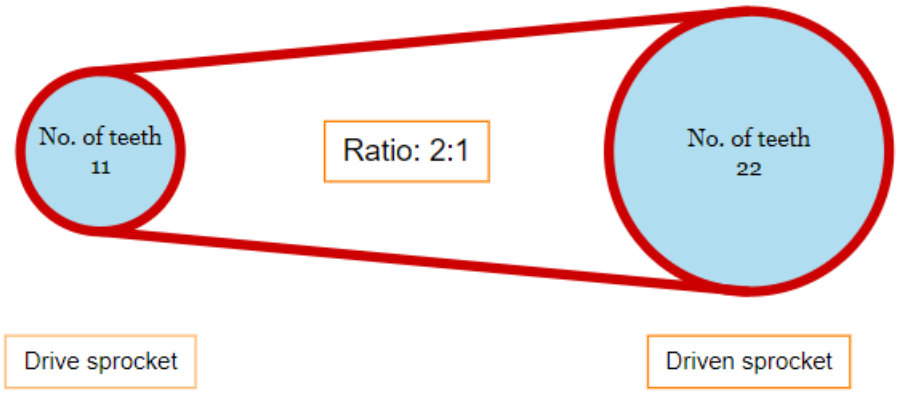


Figure 2.3.4: Gear ratio

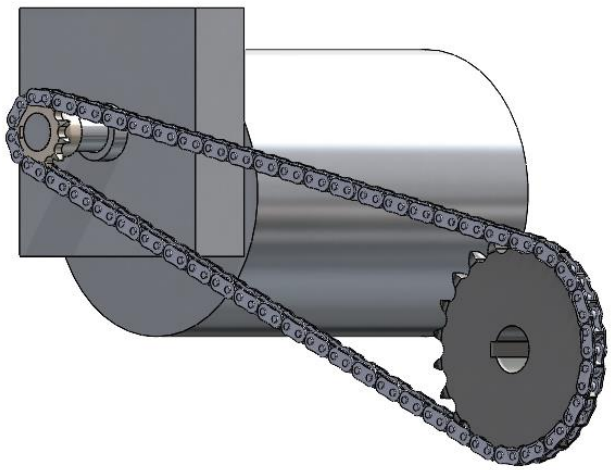


Figure 2.3.5:Gear ratio applied on the design

2.3.2 Driving sprockets, chains and shafts

2.3.2.1 First: Designing of the gear ratio system's sprockets and chains

- The planned working time is only one shift = 8hrs
- The design of the reduction system requires a design of driving and driven sprockets and their drive chains.
- Achieving the right design requires a formula of the pitch of the drive chain (t):

$$t = 2.8\sqrt{\frac{T}{Z_1 * m} * K}$$

No. of teeth of each sprocket:

- $Z_1 = 11$
- $Z_2 = 22$

Pitch of the chain:

- $t = 2.8\sqrt{\frac{23.9}{28*1} * K} = 15.875 = 0.625\ inch$

*K: is working conditions (no. of working shifts) divided by Pressure on the chains.

- Pitch diameter (D_i): $D_i = \frac{t}{\sin(\frac{180}{z})}$
- $D_1 = \frac{15.875}{\sin(\frac{180}{11})} = 56.347mm$ | $D_2 = \frac{15.875}{\sin(\frac{180}{22})} = 111.548mm$
- $r_2 = 28.17\ mm$ $r_2 = 55.77\ mm$
- Force applied on the driven sprocket: $(F_2) = \frac{T}{r} = \frac{48}{55.77/1000} = 860.6777$

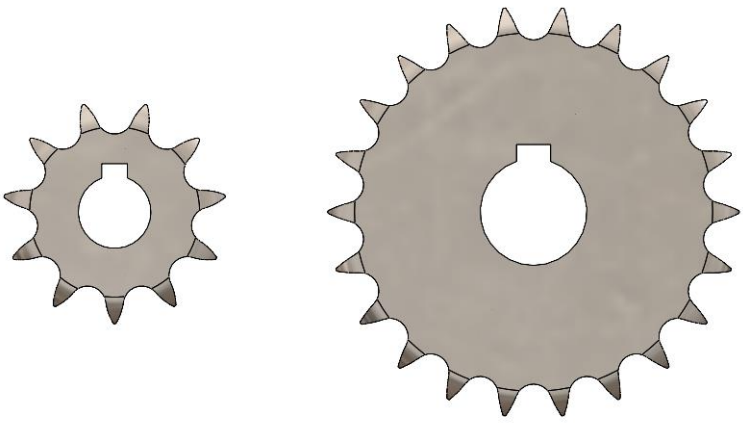


Figure 2.3.6: Gear ratio sprockets

2.3.2.2 Second: Designing of the driving chain system:

Each drive side will consist of 2 chains and every chain will have 4 sprockets.

Each drive chain will be carried on 2 shafts and each shaft will carry 2 sprockets.

The gear ratio of all sprockets in the system will be: $(i = 1)$

A power of $\frac{500}{2} = 250\ watts$ will be applied on each sprocket.

$Z_1 = Z_2 = 31 - 2i = 31 - 2 * 1 = 29\ Teeth$

Torque applied on each sprocket (T_s) will be the torque of the motor divided by 2.

- $T_s = \frac{P}{\omega} = \frac{250}{\frac{2\pi(100)}{60}} \approx 24\ N.M$
- $t = 2.8\sqrt{\frac{24}{29*1} * K} = 12.7\ cm = 0.5\ inch$

Pitch diameter of each sprocket:

- $D_i = \frac{t}{\sin(\frac{180}{z})} = \frac{12.7}{\sin(\frac{180}{29})} = 117.46\ mm$
- $Ri\ =\ 117.46\ /\ 2\ =\ 58.73\ mm\ =\ 0.05873\ m$

- Force applied on each sprocket Fs = Ts / (Di/2) $\frac{Ts}{Ri} = \frac{24}{0.05873} = 408.6497\ N$

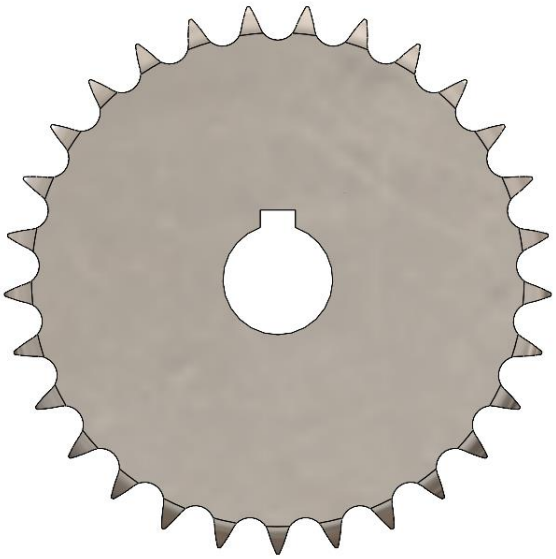


Figure 2.3.7: Sprocket carrying the tracked chain

a) Shaft design

As it is a designing phase, the locations of the bearings and gears on the shaft are unknown so it will be determined according to the design needs.

The following diagrams are the bending moment and torque diagrams, these diagrams shows the bending moments and torques at various points according to the applied forces and reactions from the sprockets and the bearings.

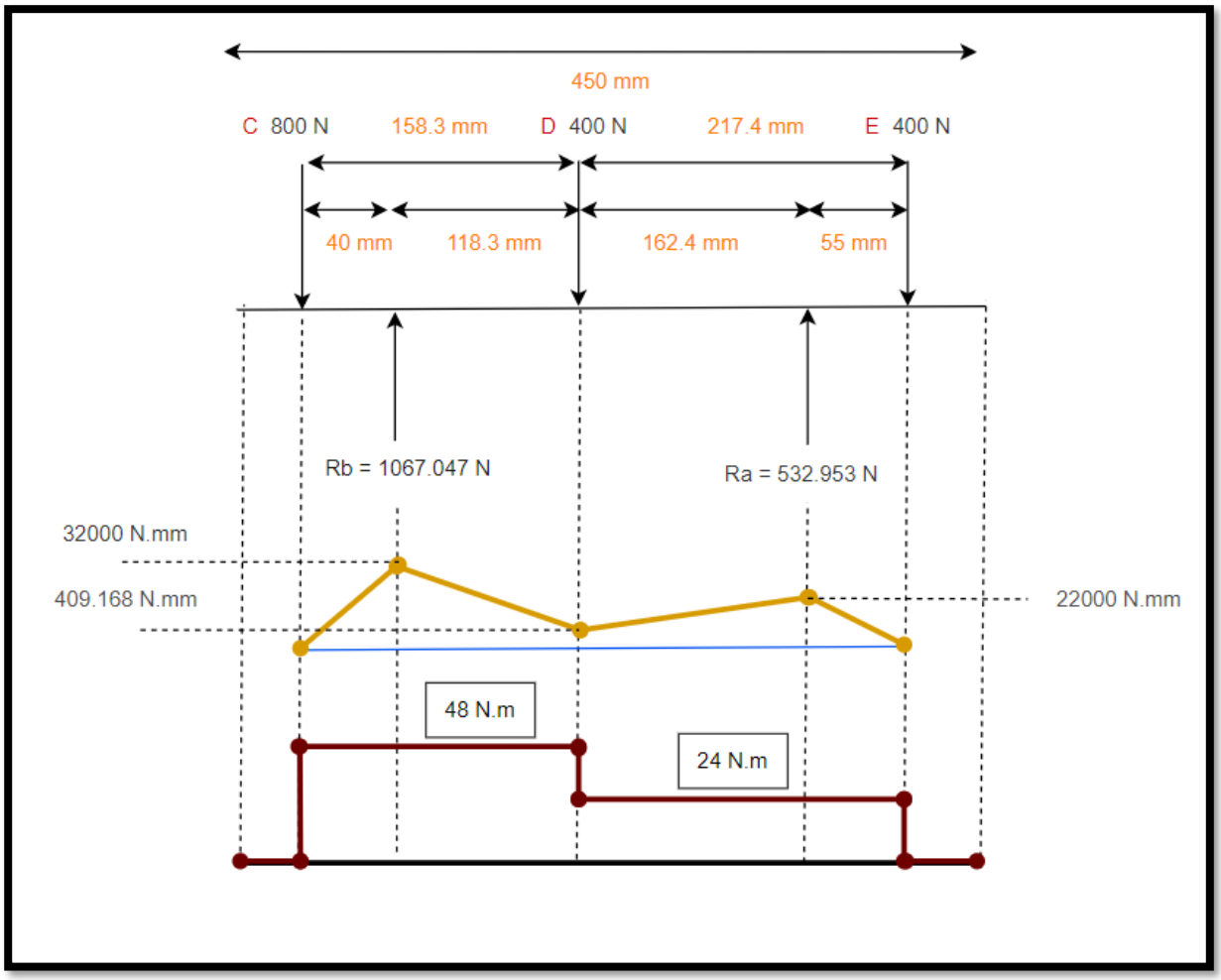


Figure 2.3.8: Bending moment diagram

BMD)$_A$ = 400* 55 = 22000 N.mm

BMD)$_B$ = 800 * 40 = 32000 N.mm

BMD)$_C$ = 0

BMD)$_D$ = (800 * 158.3) – (1067.047 * 118.3) = 408.3399 N.mm

BMD)$_E$ = 0

Design the shaft according to (Max. shear stress):

$$\tau_{all.} = \frac{16\,T_e}{\pi d^3}$$

$$T_e = \sqrt{(\max torque)^2 + (\max \text{bending moment})^2}$$

$$T_e = \sqrt{(48 * 1000)^2 + (32000)^2} = 57688.82041\ N.mm$$

Assume material of shaft is (Steel 37)

- $\boldsymbol{\sigma_\gamma = 235\ Mpa}$ $\boldsymbol{\sigma_\mu = 360\ Mpa}$
- $\tau_{all.} = \mathbf{0.30 * 0.75 * 235 = 52.875\ Mpa}$
- $\tau_{all.} = \mathbf{0.18 * 0.75 * 360 = 48.6\ Mpa}$

Select the smaller value $\tau_{all.} = \mathbf{48.6\ Mpa}$

$$\because \tau_{all.} = \frac{16 * T_e}{\pi {d_{sh.}}^3} \qquad \therefore 48.6 = \frac{16 * 57688.82041}{\pi {d_{sh.}}^3}$$

$\therefore d^3 = 6045.40$ $\therefore d = 18.34\ mm$ $(preferred: d = 20\ mm)$

**But we will open a keyway, so the chosen shaft is with (d = 25 mm)

In our design we have 4 shafts, each 2 shafts are similar and different from the other 2 shafts.

- 2 shafts carrying the 2 sprockets of the tracked chain and the sprocket of the motor.
- 2 shafts carrying only the 2 sprockets of the tracked chain.

Figure 2.3.9: Shaft final shape

Stress analysis on the shaft: as our shaft encounters torsion and bending so we did torsion and bending analysis on it, and these are the results:

- Applying torsion = double of the expected value = 100 N.m

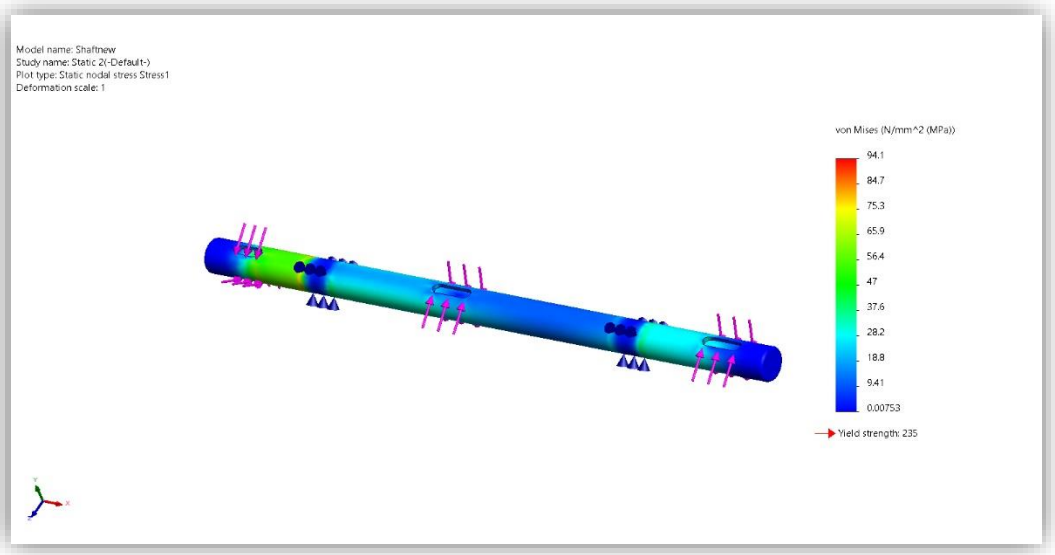


Figure 2.3.10: Stress applied on the shaft

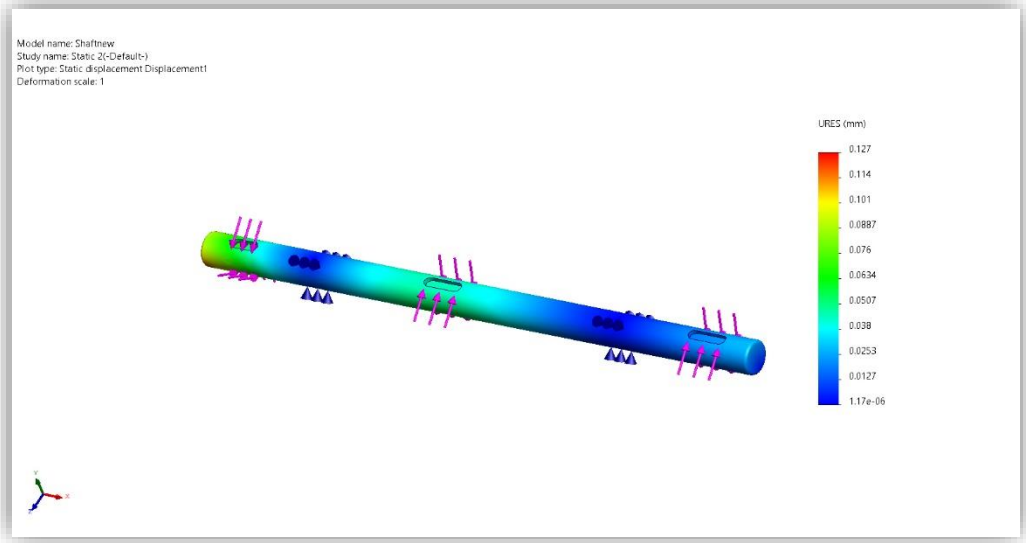


Figure 2.3.11: Deformation of the shaft

b) Key design:

$d_{sh.} = 20 + 5 = 25\ mm \qquad F.S = 1.8$

Assume key material (AISI 1006)

$\boldsymbol{\sigma_\gamma = 285\ Mpa \qquad \sigma_\mu = 320\ Mpa}$

$$F_{key} = \frac{2 * T_{max}}{d_{sh.}} = \frac{2 * 48 * 1000}{25} = 3840\ N$$

$$Key\ width\ W = \frac{d_{sh.}}{4} = \frac{25}{4} = 6.25\ mm$$

Mode of failure:

Shear mode

$$\boldsymbol{\tau = \frac{F_{key}}{n * l * w} \leq \frac{0.5\sigma_\gamma}{F.s}}$$

$$\frac{3840}{\mathbf{1 * l * 6.25}} \leq \frac{\mathbf{0.5 * 285}}{\mathbf{1.8}}$$

$$\boldsymbol{\therefore l \geq 7.760\ mm}$$

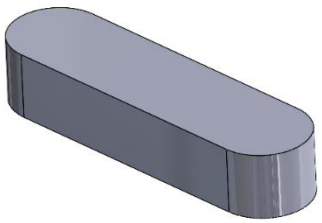


Figure 2.3.12: Key

Crushing stress

$$\boldsymbol{\sigma_C = \frac{F_{key}}{n(h/2 * l)} \leq \frac{1.5 * \sigma_\mu}{F.s}}$$

$$\boldsymbol{\sigma_C = \frac{3840}{1(h/2 * 6.0793)} \leq \frac{1.5 * 320}{1.8}}$$

$$\boldsymbol{\therefore h \geq 3.71mm}$$

Standard according to shaft diameter:

$\boldsymbol{W\ x\ h\ x\ l}$	$\boldsymbol{W\ x\ h\ x\ l}$
$\boldsymbol{6.25\ x\ 3.71\ x\ 7.760}$	$\boldsymbol{8\ x\ 7\ x\ 30}$

c) Bearing selection

According to our shaft diameter we are selecting the right bearing to withstand the forces applied on the shaft and to support the shaft.

We will go through the following steps to get the right and safest bearing.

1- Select a general bearing according to the shaft diameter:
   - Selected bearing no.: 205
   - Static load capacity (Co) = 6965 N
   - Dynamic load capacity (C_{10}) = 10690 N

2- Calculating the ratio (e'): $e' = \frac{fa}{V.\ Fr}$
   - Fa = neglected = 0

** Radial force applied on the bearings (Fr): we have 2 bearings in our design one has a force of **(1067.047 N)** while the other one has a force of **(532.953 N)** So we decided to design on the highest force which is **(1067.047 N)** to keep it safe.

- Fr = 1067.047 N
- $e' = \frac{Fa}{V * Fr} = \frac{0}{1067.047} = 0$ ** v = 1: because the inner ring rotates
- $e = \frac{Fa}{Co} = \frac{0}{6965} = 0$

∴ According to these ratios we should get some parameters from the table which are:

- x = 1 y = 0
- $(X * V * Wr + Y * Wa) = (1 * 1 * 1067.047 + 0) = 1067.047$
- Fe $= 1067.047$ N
- $Ln = Lh\,(60 * n) = 20000 * 60 * 100 = 120000000$
- $Ln = \left(\frac{C_{10}}{F_e}\right)^K * 10^6$

**K = 3 : Because it is a ball bearing not roller

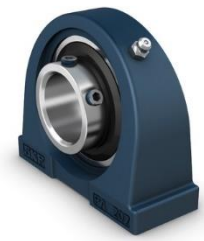
We should get the new Dynamic load capacity to compare it with the previous selected to ensure that the bearing is safe or not.

$$C_{10\,New} = \sqrt[3]{\frac{L_n}{10^6}} * F_e = \sqrt[3]{\frac{120000000}{10^6}} * 1067.047 = 5263.128 <$$

$$C_{10\,Old} = 10690 \qquad Then\ our\ bearing\ selection\ is\ safe.$$

Specs of the used Bearings:

We have used two types of bearings housings on our shafts to be suitable with our design but with the same ball bearings inside the housing.

Name:	UCPA 205	UCP 205
Photo		
Dynamic load rating	14 kN	14 kN
Static load rating	7.8 kN	7.8 kN
Limiting speed	5850 r/min	5850 r/min
Shaft tolerance	h6	h6

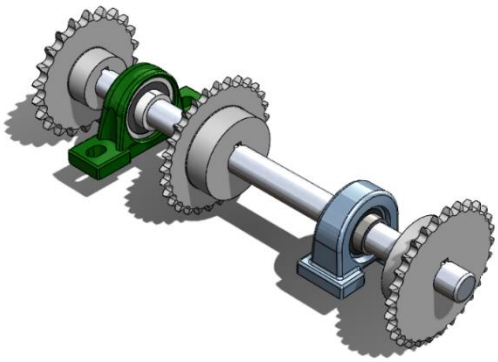


Figure 2.3.13: Full assembly of the shaft components

2.3.3 Designing of the Track chain system

The rapid expansion of solar energy has led to the establishment of vast solar farms, where the efficiency and cleanliness of solar panels are paramount. Cleaning these panels manually is labor-intensive and inefficient, especially given the extensive area they cover. Consequently, automated solutions like tracked vehicles have been developed to address this need. Central to the performance and reliability of these vehicles is the track chain system. This system is crucial not only for mobility and stability but also for ensuring the vehicle can operate effectively across diverse and often challenging terrains without damaging the solar panels or infrastructure.

Designing a track chain system for a solar panel cleaning vehicle involves several key considerations to ensure optimal performance. The system must balance stability, traction, durability, and ease of maintenance. It must also adapt to the unique requirements of solar farms, which include navigating soft ground, rough terrain, and ensuring minimal disturbance to the environment. This introduction provides an overview of the essential components and design principles involved in creating an effective track chain system for solar panel cleaning vehicles.

Designing of this part was the most challenging part as it consists of many critical points it will be divided to some parts as follows:

1- Track core metal bars
2- Suspension system

2.3.3.1 Track core metal plates

This was the most critical part of the track system design as the whole robot will be moving on these metal parts.

These plates were made of sheet metal with the following specs:

Material	Thickness	Length	width
Steel 37 (sheet)	2 mm	277.44 mm	24.7 mm

In our design this part is designed to be a little flexible but not exceeding the yield limit and keep the material in its original shape after a little bending.

The reason for this bending is the cost of the material so, we tried to keep the balancing between the cost and the efficiency, But the right thing is to use a higher material of a thicker sheet metal.

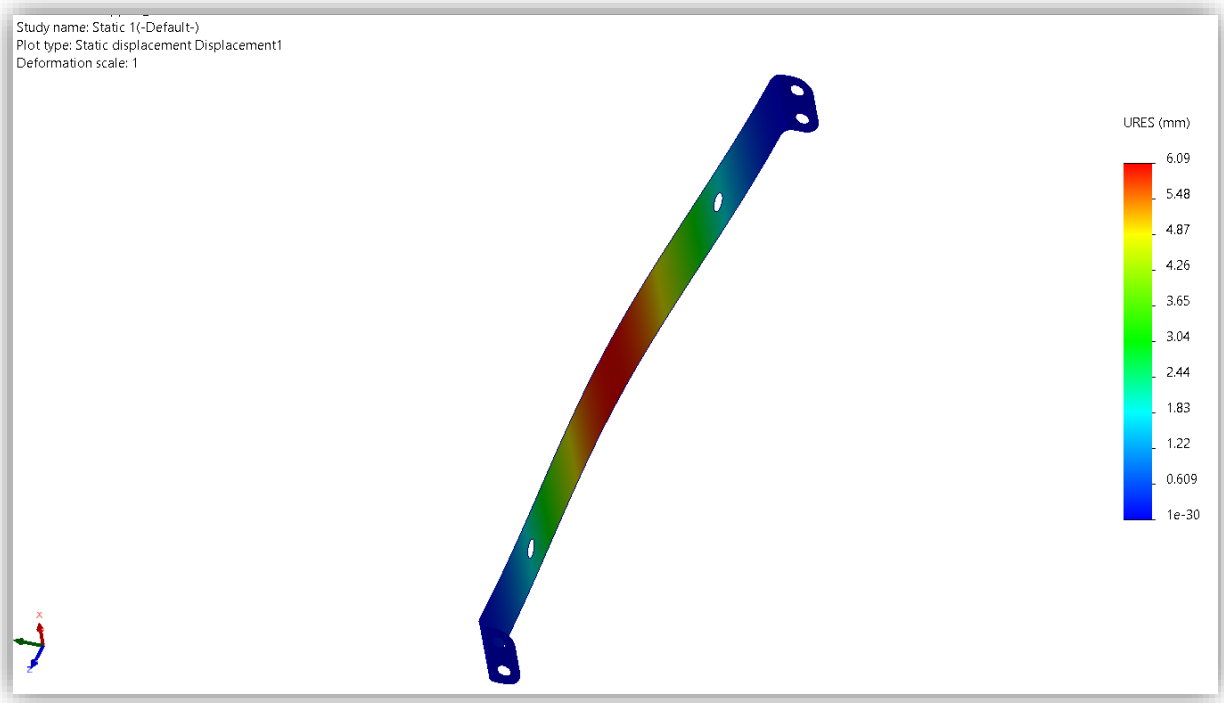


Figure 2.3.14: Deformation of the chain sheet part

We designed those parts to be cut from sheet metal then to be bent to achieve the required shape.

This process gone as follows:

1- The sheet metal is cut on a laser cutter.
2- Each part got bent from both ends.
3- Each part is welded to the chain as follows:

a. The chain consists of inner parts and outer parts like these:

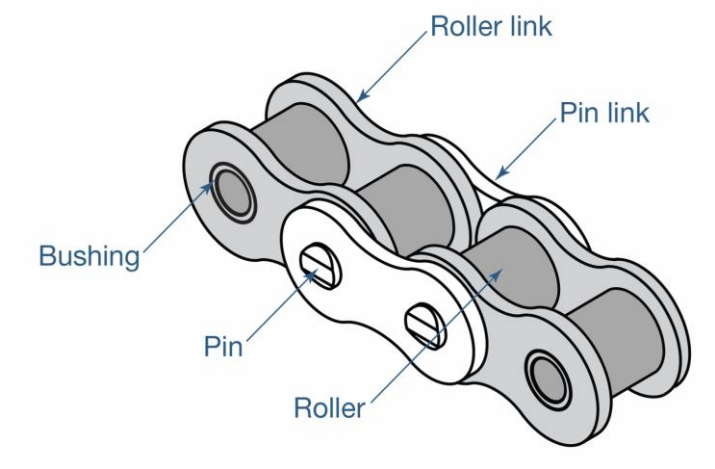


Figure 2.3.15: Construction of the chain

b. The metal part is to be welded to the chain as each pin of the chain has a hole in the metal part like this:

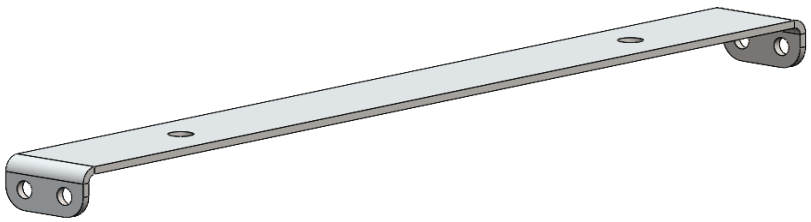


Figure 2.3.16: Chain's sheet part

c. Each pin goes in its hole tightly then got welded using to form a shape like this from both ends on 2 chains:

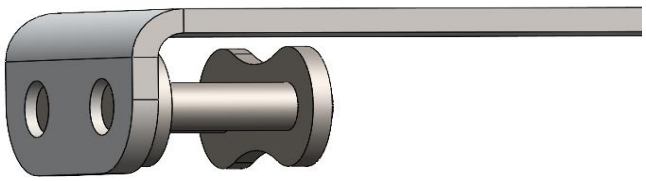


Figure 2.3.17: Assemble of the chain in the sheet part

d. Finally the track chain got its new shape after being welded:

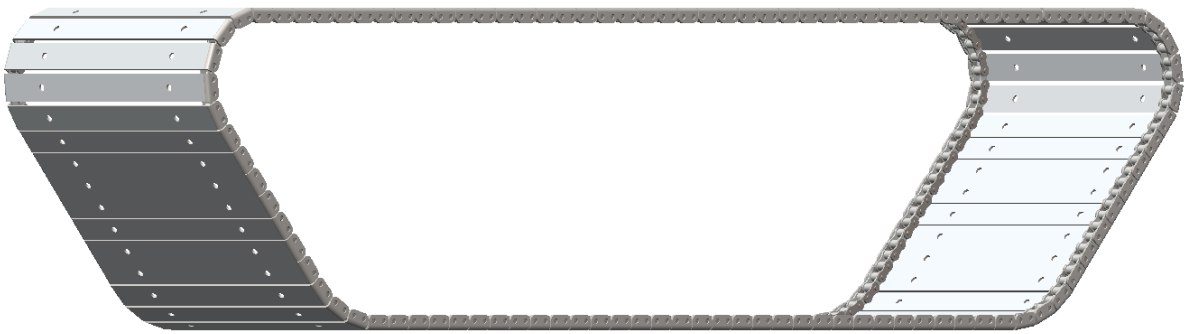


Figure 2.3.18: Fully assembled tracked chain

2.4 Suspension system

2.4.1 Introduction

The solar energy industry is experiencing rapid growth, making the efficiency and cleanliness of solar panels a priority for maximizing energy output. Large-scale solar farms, often situated in diverse and challenging terrains, necessitate innovative maintenance solutions. One effective approach involves using tracked vehicles designed specifically for cleaning solar panels. These vehicles must navigate various terrains, including uneven and soft ground, without causing damage to the panels or the surrounding infrastructure. Therefore, a robust and well-engineered suspension system is critical to their operation. This section explores the essential components and design considerations for developing an effective suspension system for such vehicles.

Track System

Tracks vs. Wheels Tracks are preferred over wheels for several reasons:

- **Stability**: Tracks provide a larger contact area with the ground, distributing the vehicle's weight more evenly. This results in improved stability, particularly on soft or uneven ground, which is common in solar farms.
- **Traction**: The extensive surface area of tracks ensures superior traction, enabling the vehicle to traverse slippery or loose surfaces, such as gravel or sand, more effectively. This traction is crucial for maintaining consistent movement and avoiding slips.
- **Ground Pressure**: Tracks exert less ground pressure compared to wheels. By spreading the vehicle’s weight over a larger area, tracks minimize the risk of sinking into soft soil and reduce potential damage to the ground surface and underlying infrastructure.

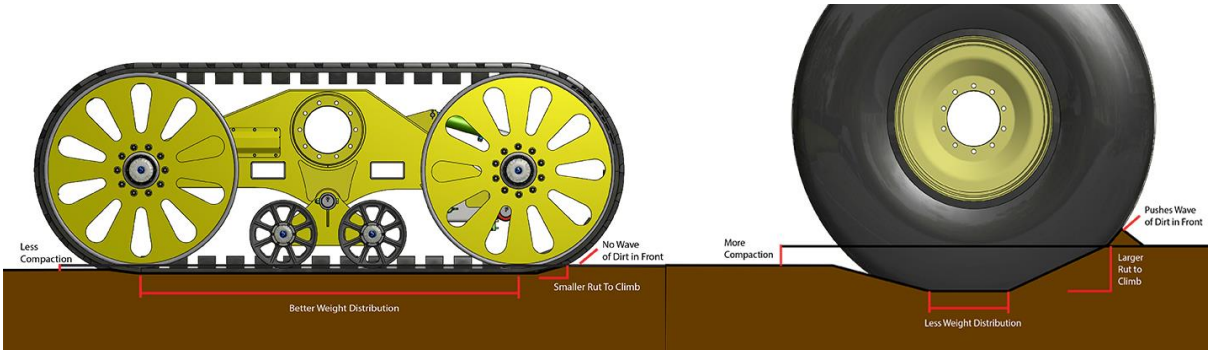


Figure 2.4.1: Tracked chain vs tire

Material The tracks must be constructed from materials that can endure harsh outdoor conditions while maintaining functionality:

- **Reinforced Rubber**: This material offers flexibility and durability, making it suitable for navigating varied terrains. Its resilience against wear and tear ensures a long operational life.

Suspension Design

Bogies and Rollers Incorporating bogies and rollers into the suspension system enhances weight distribution and ground contact:

- **Bogies**: These pivoting wheel assemblies allow the tracks to move independently over obstacles, maintaining consistent ground contact and providing a smoother ride. Bogies help distribute weight evenly, reducing stress on the vehicle's frame and suspension components.
- **Rollers**: Positioned along the length of the tracks, rollers facilitate the even distribution of the vehicle's weight across the entire track surface. This even distribution is essential for minimizing ground pressure and improving overall stability.

Spring and Damper System A combination of springs and dampers is vital for absorbing shocks and vibrations from rough terrain:

- **Springs**: These components absorb energy from impacts, preventing excessive jolts and protecting both the vehicle and the solar panels from damage. Springs are designed to compress and expand in response to terrain variations, maintaining ride comfort and stability.
- **Dampers (Shock Absorbers)**: Dampers dissipate the energy absorbed by the springs, ensuring a smoother ride and reducing wear on the suspension system. By controlling the rate of spring movement, dampers prevent excessive bouncing and improve vehicle control, especially on uneven surfaces.

Load Distribution Effective load distribution is critical for maintaining vehicle stability and protecting the terrain:

- **Even Weight Distribution**: Ensuring that the vehicle's weight is evenly distributed across the tracks prevents any part from sinking into soft ground. Even distribution reduces localized ground pressure, minimizing the risk of soil compaction and damage to solar farm infrastructure.

Durability

Robust Construction The suspension system must be designed for long-term durability to withstand continuous use in outdoor environments:

- **Weather Resistance**: Materials should be resistant to dust, moisture, and temperature fluctuations. Components exposed to the elements must maintain their functionality and structural integrity over time.
- **Strength**: Suspension components should be robust enough to endure the physical demands of navigating rough terrain. The use of high-strength materials and reinforced designs ensures the suspension system can handle the stresses of regular operation without failure.

Ease of Maintenance Designing the suspension system for easy maintenance and part replacement is crucial for minimizing downtime and ensuring the vehicle remains operational:

- **Modular Components**: Using modular components allows for quick repairs and replacements. If a part fails, it can be swiftly swapped out without requiring extensive disassembly, reducing the time the vehicle is out of service.
- **Accessible Design**: The suspension system should be designed for easy access to critical parts. Simplifying access to components such as springs, dampers, and bogies facilitates routine maintenance and reduces repair times, ensuring the vehicle can return to operation promptly.

2.4.2 Key Components

a. **Road Wheels:** These are the primary wheels that meet the ground and support the weight of the vehicle. They are mounted on bogies and connected to the suspension system to absorb shocks from the terrain.
b. **Sprockets:** These sprockets transfer motion coming from the motor on the shafts to the track itself and ensure that the track itself is balanced, preventing it from slipping and ensuring that the track itself is tensioned well.
c. **Suspension mechanism:** This is the mechanism of the connection between the shock absorbing module and the wheels.

   Our mechanism is inspired from a real suspension mechanism called "Horizontal Volute Spring System (HVSS) suspension", this system is used in the war tanks.

This system consists of:

1- Links with special design.
2- Shock absorbing module (spring + shock absorber).
3- Wheels.

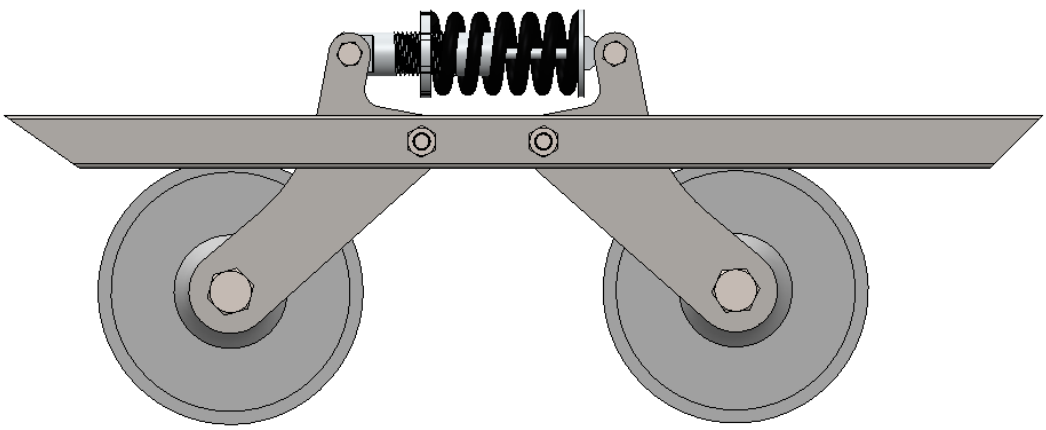


Figure 2.4.2: Fully assembled suspension mechanism

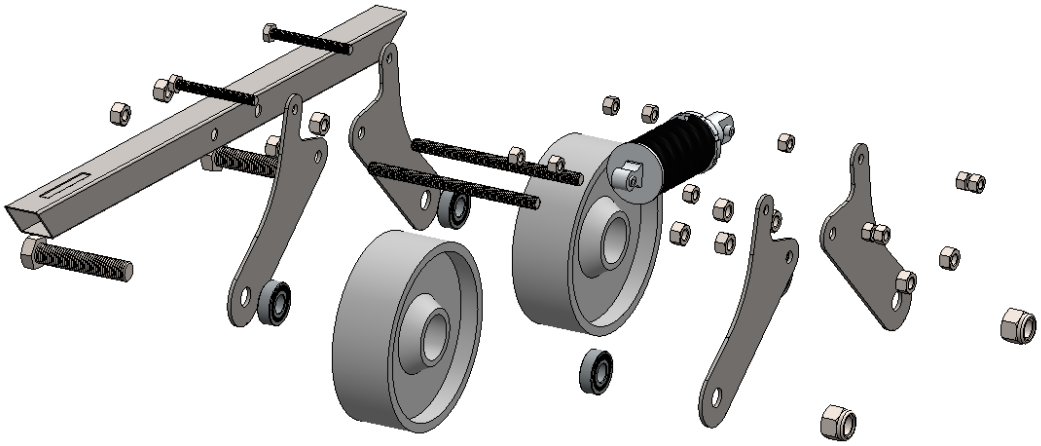


Figure 2.4.3: Components of the suspension mechanism

To ensure that this mechanism is efficient and able to perform its required function, we did some analysis and tests to get the best out of our idea.

2.4.2.1 The suspension mechanism

Mechanism links analysis and motion paths.

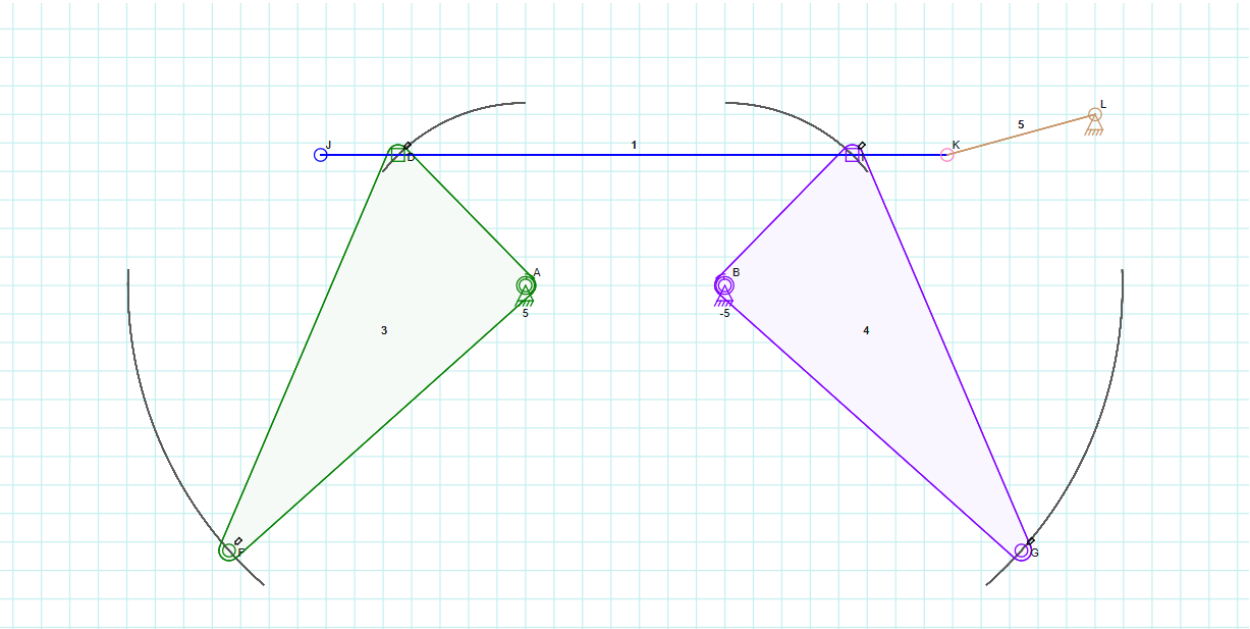


Figure 2.4.4: Mechanism of suspension system

- This test is done on an application called linkage.
- This application simulates the motion of the links together to ensure that all links are moving smoothly without any obstacles that limits the movement of any link during its function or the breaking of any link (ensuring that each link moves in its path without interfering with any other link).
- The shown curves on the figure are the paths of the links while performing its duty function.

Stress and deformation analysis on the links itself

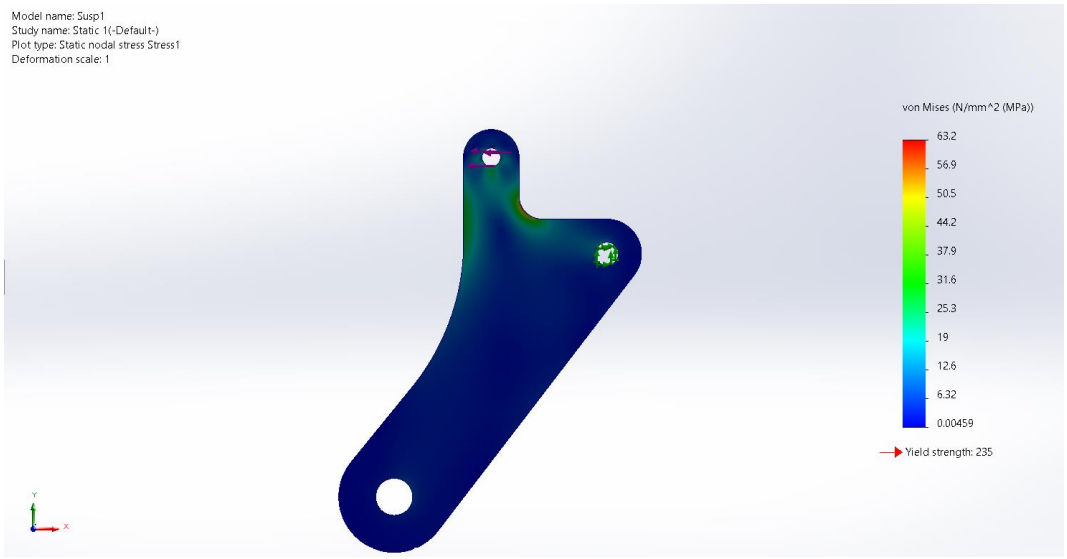


Figure 2.4.5: Stress on the suspension link

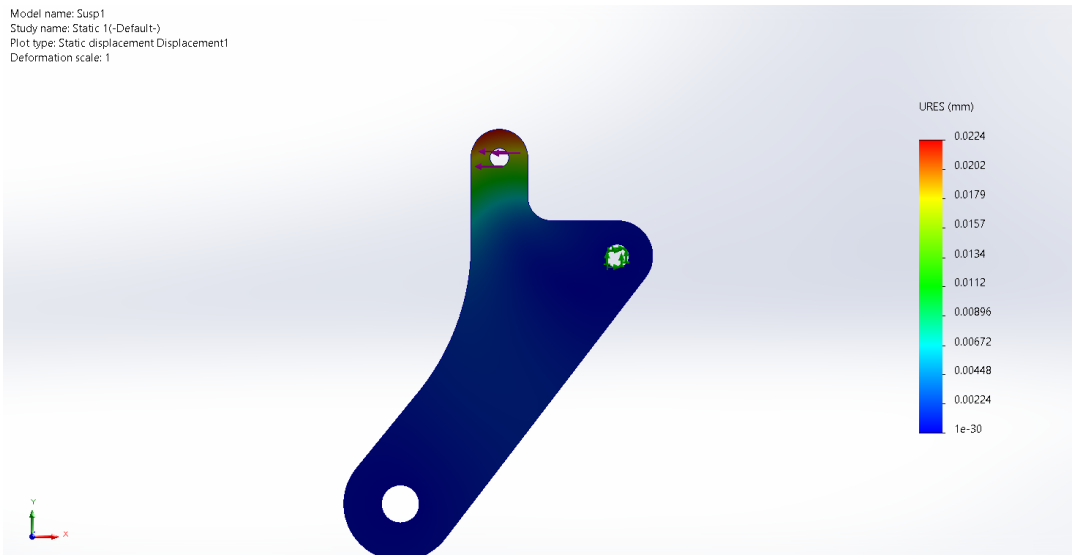


Figure 2.4.6: Deformation of the suspension link

Stress and deformation analysis on the screw carrying the mechanism parts

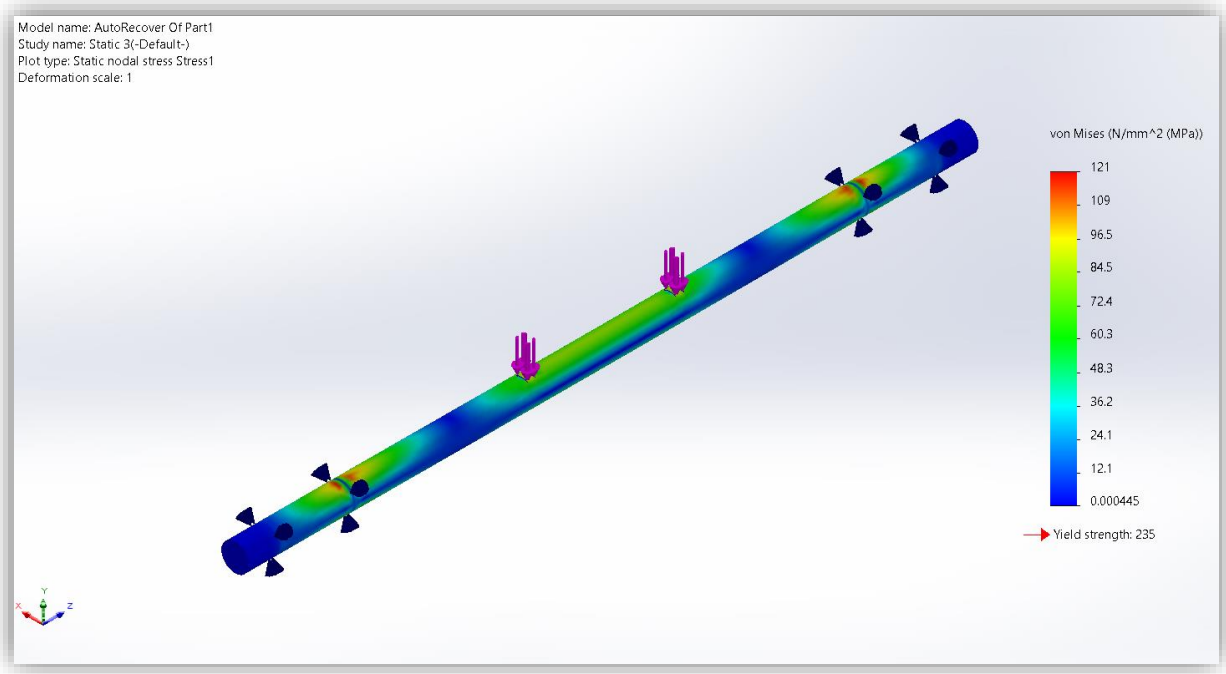


Figure 2.4.7: Stress on the suspension mechanism screw

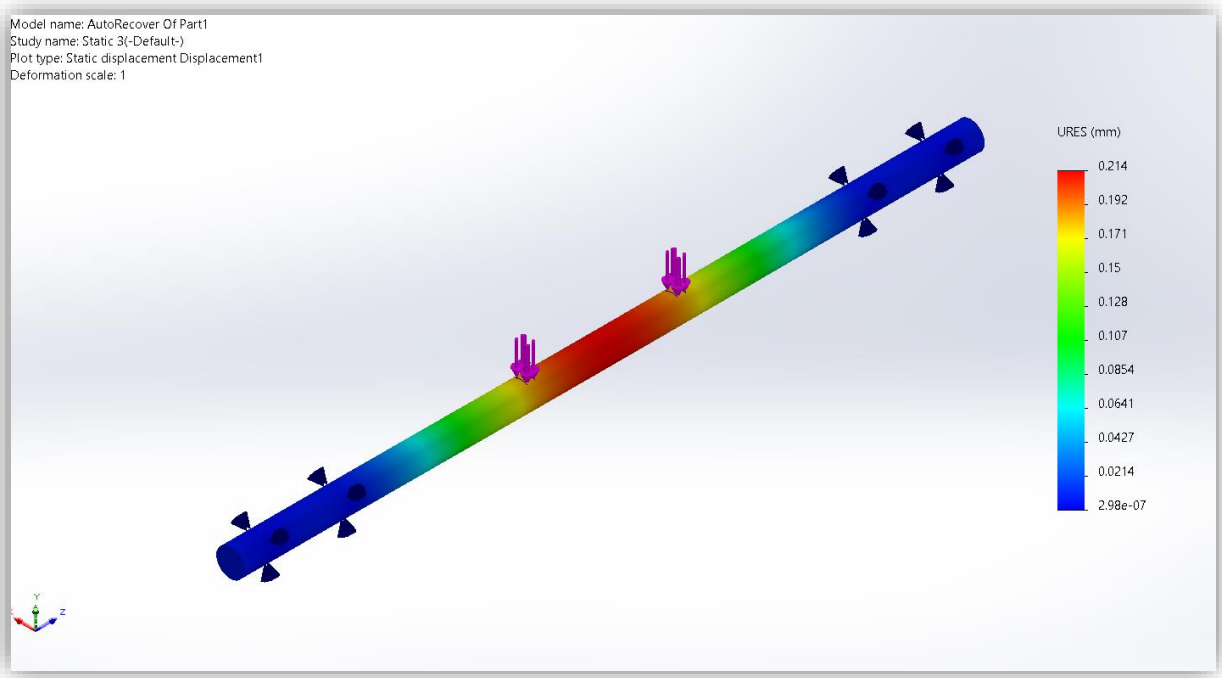


Figure 2.4.8: Deformation on the suspension mechanism screw

2.4.2.2 Shock absorbing module (spring + shock absorber)

This part is a critical part where it is responsible for carrying all the weight, maintaining stability and keeping the whole robot reacting well with any kind of terrain

Figure 2.4.9: Suspension Spring module

During this step we were confused between many choices, because this part is one of the main parts that is responsible for the stability and maintaining the vibrations during the motion of the robot.

So, we decided to choose this part from the available parts in the market.

After many searches and inspection of many parts we came up with this part.

This part has a compression ratio of:

- (750LBS / in) = (133.9 kg / cm)

But our design requires 4 parts, so it will be stiffer than it should be.

We then continued searching for a more flexible one, but we didn't find a replacement, so we decided to continue the manufacturing process with this part.

We have developed a simscape model to simulate the vibrations of the suspension spring when the robot is moving on the uneven ground.

The results were stiffer than we needed as we mentioned earlier so these results validate that we should have used a less stiff spring than this one.

The spring model on simscape:

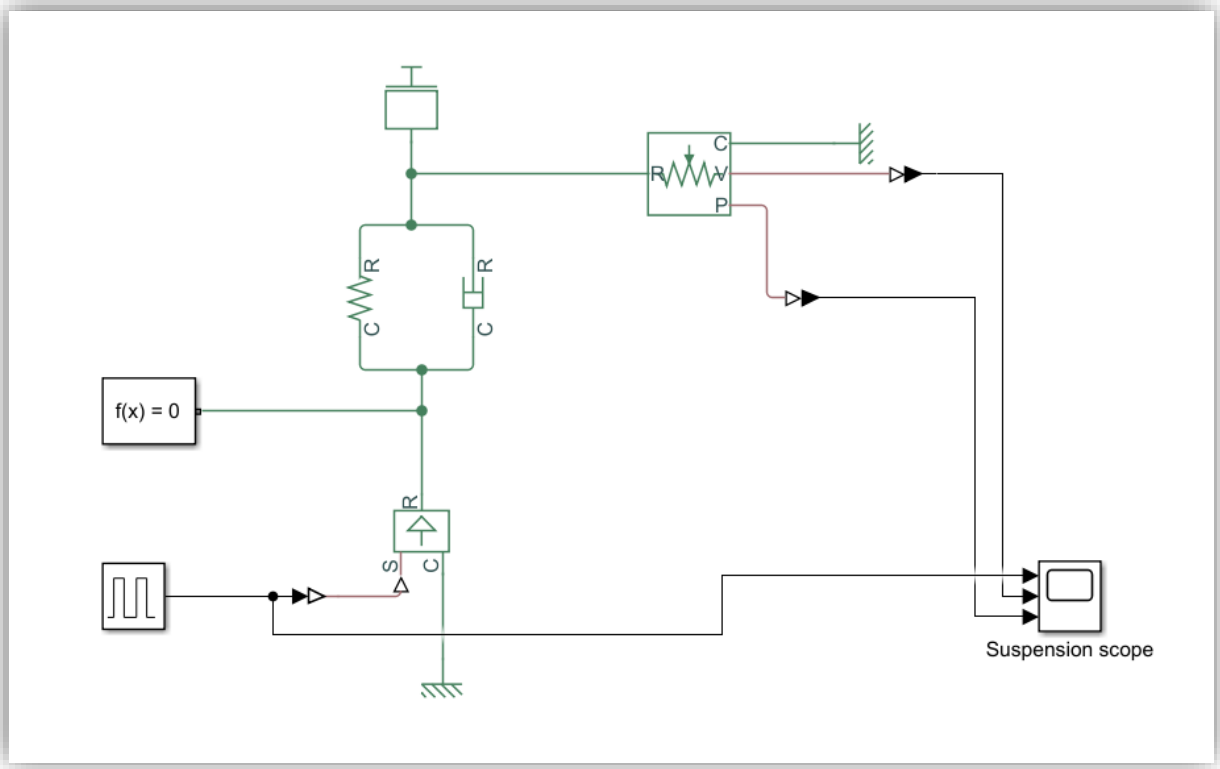


Figure 2.4.10: Suspension modelling on simscape

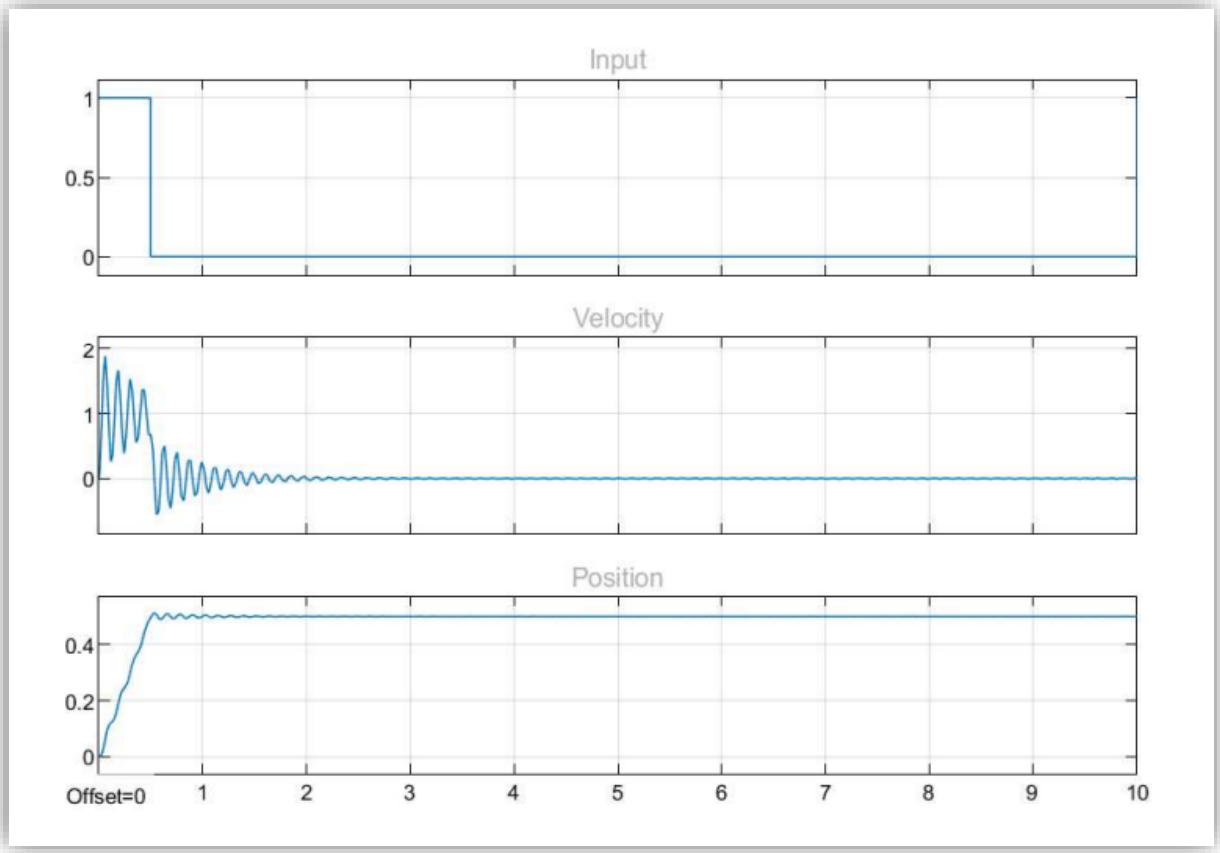


Figure 2.4.11: Results of the modelling

2.4.2.3 Wheels

The selection of the wheels was based on the width of the track chain, as a wider wheel will give more stability and keep the chain flatten as much as possible.

We have a 25 cm wide track chain, the wider wheels that we found in the market were 5 cm wide, so we used 4 wheels of it on each side.

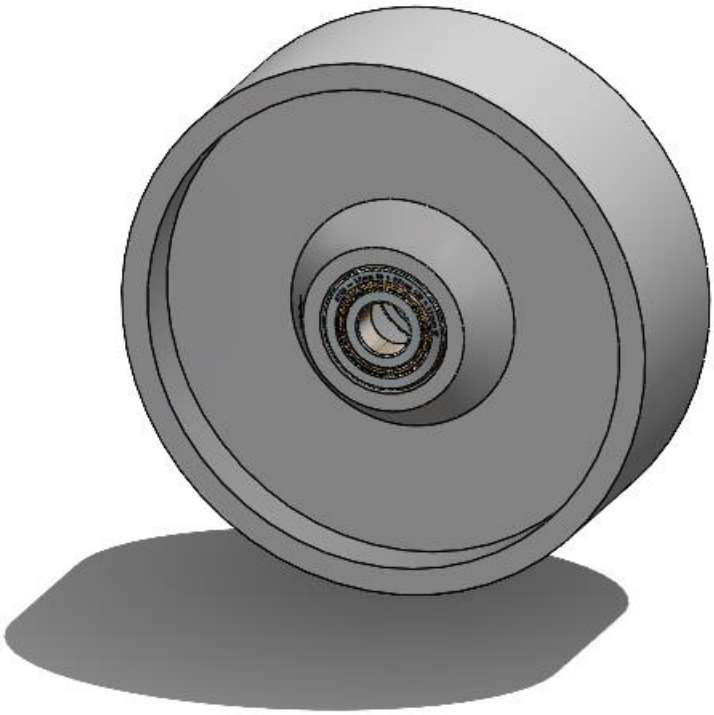


Figure 2.4.12: Wheel

These wheels include bearings from both sides to ensure the smoothness of the motion and to align the fixing shaft in the right position.

2.5 MOBILE ROBOT KINEMATICS

Kinematics is fundamental in understanding the mechanical behavior of mobile robots, crucial for designing them effectively and developing control software. Unlike manipulator robots, which are more complex with multiple joints, early mobile robots like differential-drive machines were simpler in design.

The robotics community has extensively studied manipulator robot kinematics and dynamics, achieving a comprehensive understanding. Mobile robotics faces similar kinematic challenges, focusing on defining the robot's workspace and controllability. Dynamics, influenced by factors like mass and force, further constrain mobile robot movement, such as limiting turning radius due to high center of gravity.

Unlike manipulators, which have fixed endpoints for precise position measurement, mobile robots are autonomous, requiring integrated motion over time for accurate position estimation. Factors like slippage introduce challenges, making precise position measurement difficult.

Understanding mobile robot motion begins by analyzing each wheel's contribution and constraints on motion, such as preventing lateral skidding. By establishing kinematic models and constraints for individual wheels, overall motion capabilities and constraints of the robot are defined, facilitating path planning and trajectory evaluation.

Different Kinematic Models for Mobile Robots:

1. **Ackermann Steering Model:**
   The Ackermann steering model, inspired by the steering mechanism in cars, uses a pair of front wheels that steer and a pair of rear wheels that provide power. This model ensures that all wheels follow circular paths centered on a common point, reducing tire slippage and wear. The Ackermann steering mechanism is ideal for high-speed applications due to its smooth and efficient turning capabilities. However, it is mechanically complex and expensive to implement. Additionally, it requires more space to turn compared to differential drive systems and cannot turn in place, limiting its maneuverability in confined environments.

2. **Omni-Directional Drive Model:**
   The omni-directional drive model uses specialized wheels, such as mecanum or omni wheels, which allow the robot to move in any direction without changing its orientation. This model provides full mobility and precise movement, making it ideal for complex environments that require agile navigation. The ability to move laterally as well as forward and backward simplifies path planning and enhances the robot's versatility. Despite these advantages, omni-directional drive systems are complex to control and typically more expensive due to the

specialized wheels and control algorithms required. They may also have limited load-carrying capacity and reduced traction on certain surfaces.

3. **Unicycle Model:**
   The unicycle model is a simplified representation of a mobile robot, abstracting its motion to that of a unicycle with a single wheel. The robot's state is defined by its position and orientation, and its movement is governed by linear and angular velocities. This model is mathematically simple and easy to implement, making it popular for theoretical studies and basic control applications. However, it lacks realism, as it does not account for physical constraints like wheel dynamics and friction. The unicycle model is limited to 2D motion and can encounter singularity issues at zero linear velocity, complicating control in some scenarios.

4. **Skid Steering Model**
   The skid steering model is commonly used in tracked vehicles and some wheeled robots. It achieves turning by varying the speed of tracks or wheels on each side, causing the robot to skid. This model is robust and well-suited for rough and uneven terrain, making it ideal for outdoor applications. Skid-steered robots can turn in place, offering excellent maneuverability. However, the increased friction from skidding leads to faster wear and tear on tracks or wheels, and higher energy consumption. The dynamics of skid-steered vehicles are also complex, making accurate modeling and control more challenging, particularly on slippery surfaces where traction can be reduced.
5. **Differential drive:**

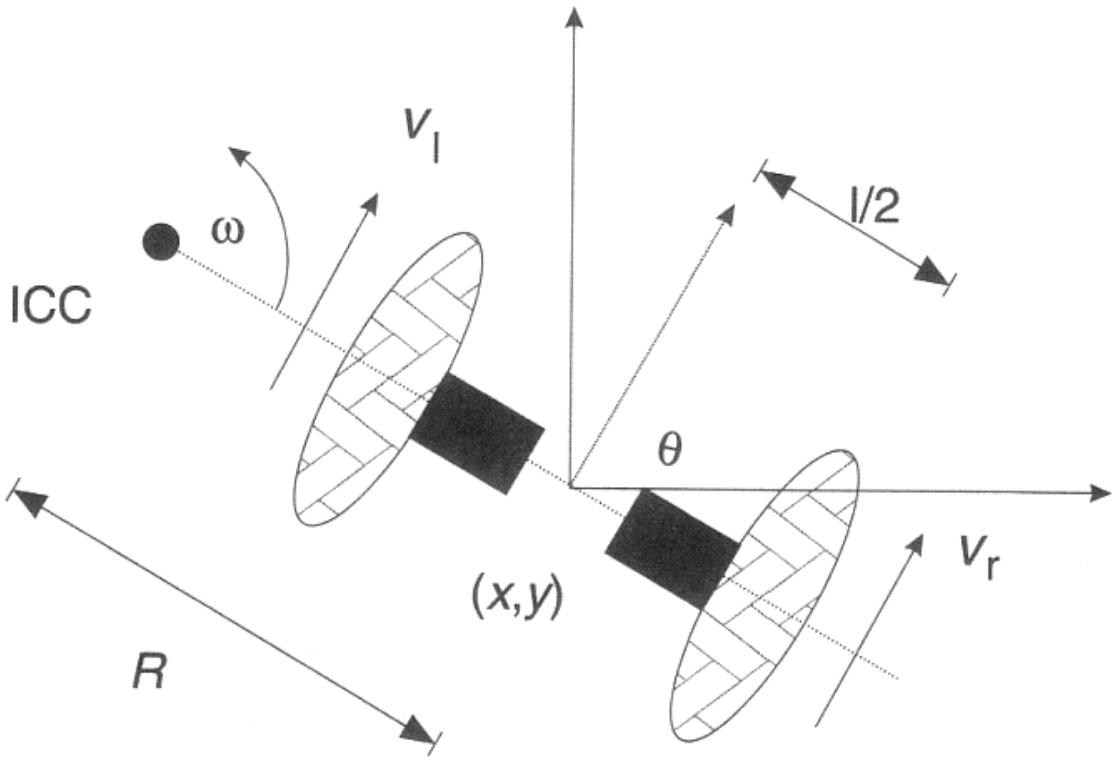
   Differential drive is a fundamental locomotion mechanism in mobile robotics, characterized by two independently driven wheels. This configuration allows the robot to move forward, backward, and rotate in place by varying the speeds and directions of the wheels. Unlike more complex drive systems, differential drive offers simplicity in control and mechanical design, making it popular for various robotic applications.

   1. **Mechanical Design**: Consists of two wheels mounted on a common axis, each driven by independent motors. Steering is achieved by driving the wheels at different speeds: moving both wheels forward makes the robot go straight, while differential speeds enable turning.
   2. **Control and Maneuverability**: Differential drive robots are highly maneuverable, capable of navigating tight spaces and executing precise movements. This maneuverability is crucial for tasks ranging from indoor exploration to outdoor traversal over uneven terrain.

3. **Applications**: Widely used in robotics research and industry, differential drive robots are employed in areas such as surveillance, logistics, and search-and-rescue operations. Their simplicity and effectiveness in various environments make them a versatile choice for autonomous systems.

Differential drive systems continue to evolve with advancements in sensor technologies and control algorithms, enhancing the capabilities and efficiency of mobile robotic systems.

By varying the velocities of the two wheels, we can vary the trajectories that the robot takes. Because the rate of rotation ω about the ICC (Instantaneous Center of Curvature) must be the same for both wheels, we can write the



following equations:

$$\omega \left(R + \frac{l}{2}\right) = V_r \qquad (1)$$

$$\omega \left(R - \frac{l}{2}\right) = V_l \qquad (2)$$

where l is the distance between the centers of the two wheels, V_r , V_l are the right and left wheel velocities along the ground, and R is the signed distance from the ICC to the midpoint between the wheels. At any instance in time, we can solve for R and ω:

$$R = \frac{l}{2}\frac{V_l + V_r}{V_r - V_l} \; ; \; \omega = \frac{V_r - V_l}{l} \qquad (3)$$

There are three interesting cases with these kinds of drives.

1. If $V\_l = V\_r$
   then we have forward linear motion in a straight line. R becomes infinite, and there is effectively no rotation ω is zero.
2. If $V_l = -V_r$
   then $R = 0$, and we have rotation about the midpoint of the wheel axis, it rotate in place
3. If $V_l = 0$
   Then we have rotation about the left wheel. In this case $R = \frac{l}{2}$.
   Same is true if $V_r = 0$.

The differential drive robot cannot move in the direction along the axis, this is a singularity. Differential drive vehicles are very sensitive to slight changes in velocity in each of the wheels. Small errors in the relative velocities between the wheels can affect the robot trajectory. They are also very sensitive to small variations in the ground plane.

Why to choose differential drive model:

1. **Precise Navigation:**
   The ability to turn in place and navigate accurately is essential for maneuvering around and between solar panels, ensuring efficient and thorough cleaning.
2. **Simplicity and Cost-Effectiveness:** With budget and simplicity in mind, the differential drive with only two motors offers a practical solution without compromising on functionality.
3. **Enhanced Stability and Traction with Tracked Wheels:** The use of tracked wheels provides the necessary stability and traction needed to move smoothly on the potentially uneven surface of a solar panel array.

Additional part in the design

During our designing phases and the interaction with the control phases, we found that the speed of the motor should be measured and sensed using an encoder to get the optimum speed of the robot and use this measured readings to build a control module using **PID controller** to ensure that the motion of the robot is smooth and reliable and to determine the motion of the robot efficiently to be able to draw a map, localize and navigate using ROS so, we have designed a holder to hold this encoder in front of the motor shaft precisely and accurately.

The encoder consists of 2 parts:

- The sensor itself
- A magnet

So, we came up with this design to achieve stability, accuracy and reliability:

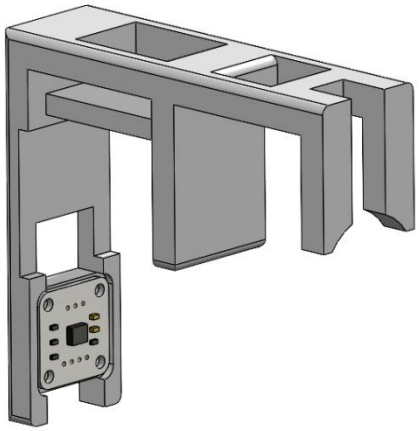


Figure 2.5.1: Speed encoder holder

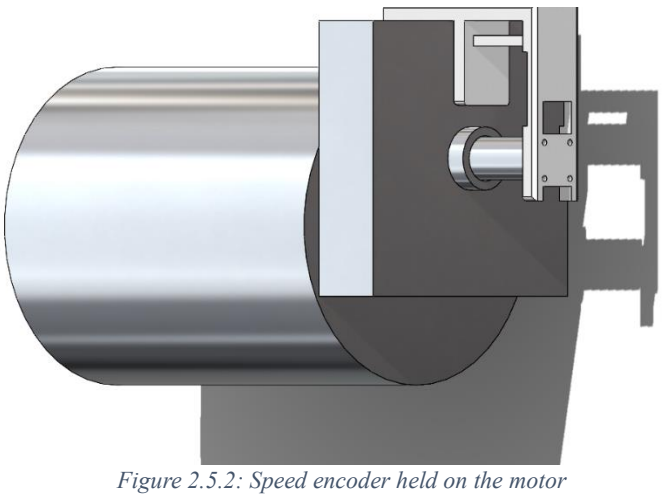


Figure 2.5.2: Speed encoder held on the motor

2.6 Manufacturing and Assembly

The manufacturing and assembly phase is a critical stage in the development of the autonomous solar panels cleaning robot. This phase involves transforming the design specifications and engineering drawings into a functional and operational robot. The process is meticulously planned and executed to ensure precision, durability, and efficiency in the final product.

This phase is divided into many parts to explain every process and every step in the process of manufacturing and assembly these processes and steps will be divided into the following points:

1- Material selection
2- Chassis manufacturing and assembly
3- Motion system manufacturing and assembly

2.6.1 Material Selection and Procurement:

The first step in the process is selecting and procuring the appropriate materials for the robot's components. Given the robot's requirement for durability and light weight, materials such as high-strength steel for the chassis and corrosion-resistant alloys for other structural elements are chosen and sourced from reliable suppliers.

The chosen materials were about the designed parameters and specifications that we mentioned above in the previous chapter.

These materials should handle and withstand all the loads and forces that will act on it during the

2.6.2 Chassis manufacturing and assembly

This step will be divided into some point as follows:

2.6.2.1 Cutting the metal bars (The core material of the chassis)

In this phase we bought the metal bars then cut them into many parts according to the designed dimensions and angles.

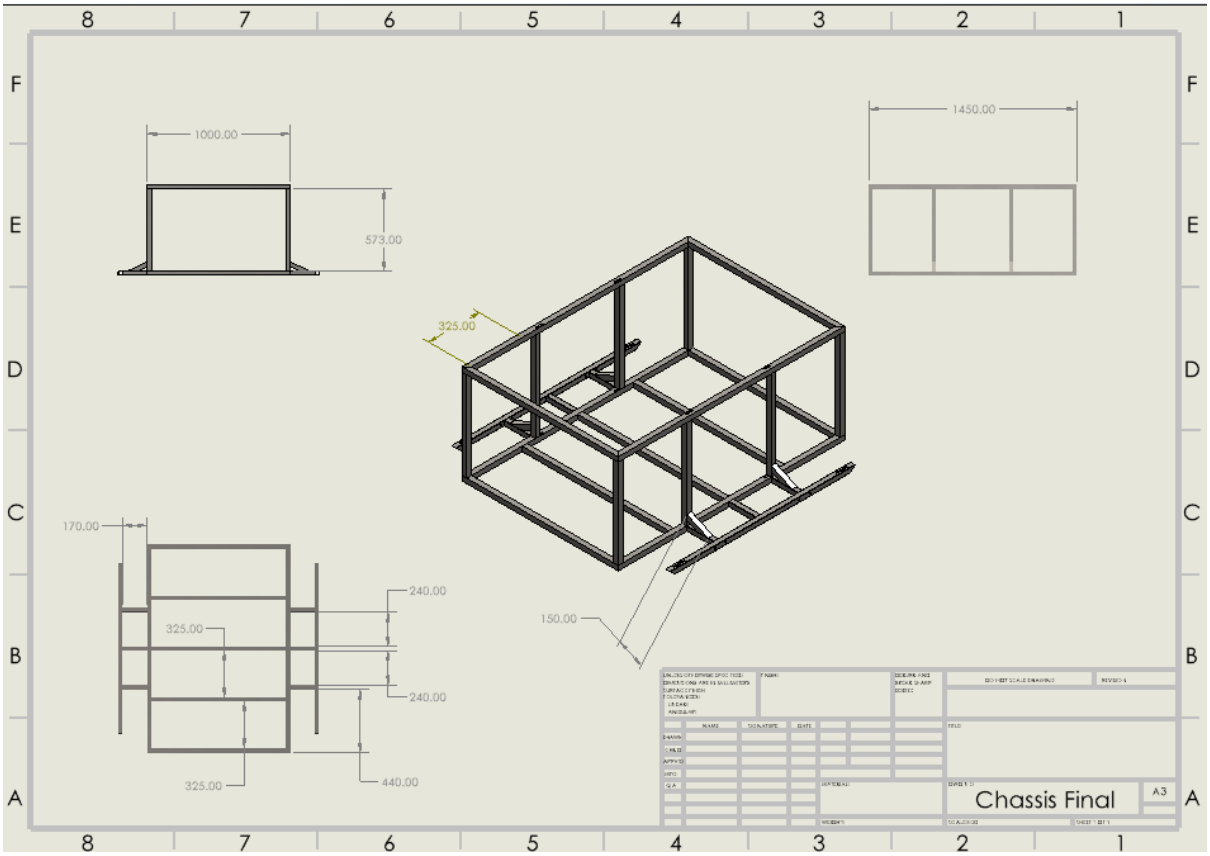


Figure 2.6.1: Dimensions of the chassis

2.6.2.2 Welding

In this phase we managed to weld the chassis parts together to make it strong and durable.

The used welding technology was argon welding. Argon welding, commonly known as Gas Tungsten Arc Welding **(GTAW)** or Tungsten Inert Gas **(TIG)** welding, is a process that uses argon gas as a shielding gas to protect the weld area from atmospheric contamination. This method is highly regarded in various industries for its precision and quality. Here are the primary advantages of the argon welding process:

High Weld Quality:

- Ensures strong, clean, and defect-free welds, crucial for the structural integrity of the chassis.

Precision:

- Provides precise control overheat input, minimizing distortion and ensuring accurate alignment of chassis components.

Material Versatility:

- Suitable for welding high-strength steel and various thicknesses used in your chassis, maintaining the material's properties and allowing for flexible design.

Aesthetic Finish:

- Produces smooth, visually appealing welds, reducing the need for extensive post-weld finishing.

Durability:

- Creates robust and corrosion-resistant joints, enhancing the longevity of the chassis.

Figure 2.6.2: Welding using Argon technique

Figure 2.6.3: Chassis during welding process

Figure 2.6.4: Final welded chassis

2.6.3 Motion system manufacturing and assembly

This was the most critical phase of the design, manufacturing and assembly. This phase consisted of many parts and points that will be discussed in the following topics:

2.6.3.1 Manufacturing and assembly of the driving system

The driving system in our robot consists of many parts as follows:

Component	Amount	Specifications
Shafts	4	• Length = 45 cm • Diameter = 25mm
Tracked chain sprockets	8	• Pitch diameter = 117.46 mm • Number of teeth = 29
Torque ratio sprockets	4	• Pitch diameter of the smaller sprockets = 56.347 mm • Number of teeth = 11 • Pitch diameter of the larger sprockets = 111.548 mm • Number of teeth = 22
Torque ratio chains	2	• Length of each one = 100 cm
Motors	2	----------
Bearings	8	• UCPA 205 (4 units) • UCP 205 (4 units)
Keys	10	• Dimensions are: 8 x 6 x 30 mm

Table 2.6-1: Motion system components

a. Machining of the shafts:

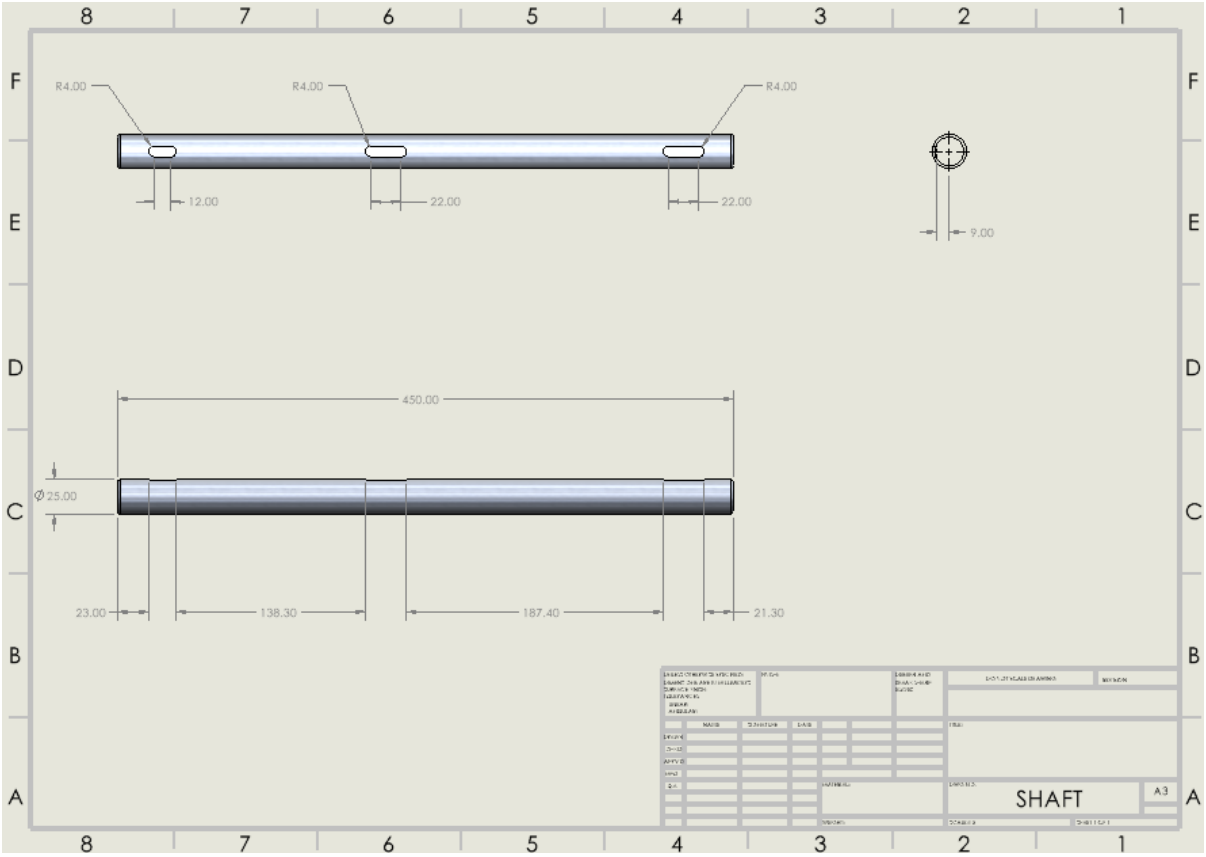


Figure 2.6.5: Dimensions of the shafts

- The shafts were bought as a long rod of metal of material (Steel 37).
- This rod got cut on a desk to be cut into 4 pieces with the mentioned specs.
- These pieces were then put on the turning machine to get cleaned, length adjusted and filleted from both ends.

Figure 2.6.6: Shafts after machining

- The last step was opening the slots of the keys on the milling machine.

Figure 2.6.7: Shafts after opening keys slots

Figure 2.6.8: Keys slot opening process

b. Machining of the sprockets

- The sprockets were bought with random shaft opening.

Figure 2.6.9: Sprocket before machining

- The sprockets hub was then reduced on the turning machine
- The shaft hole was adjusted on the turning machine to the required shaft diameter.

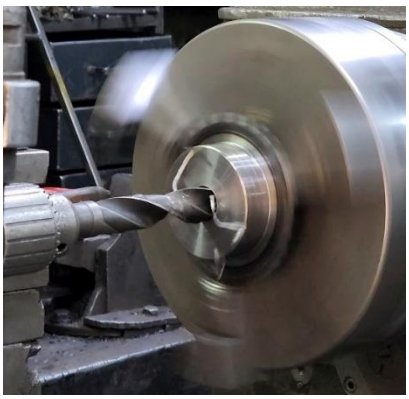


Figure 2.6.10: Sprocket's turning process

- Then on the milling machine the sprocket hub got drilled and threaded to make a hole for the screw that will fix the gear on the shaft without slipping.
- The last step was opening the keyhole using the shaper machine.

Figure 2.6.11: Sprocket's key opening process

c. Assembly of sprockets and shafts.

- After machining of the sprockets and shafts, the bearings got mounted on the chassis in the designed place using the required bolts and nuts.
- Then the shafts and sprockets got mounted on the bearings in the designed places.
- The sprockets got attached to the shafts using the keys and the screw that prevent it from slipping.

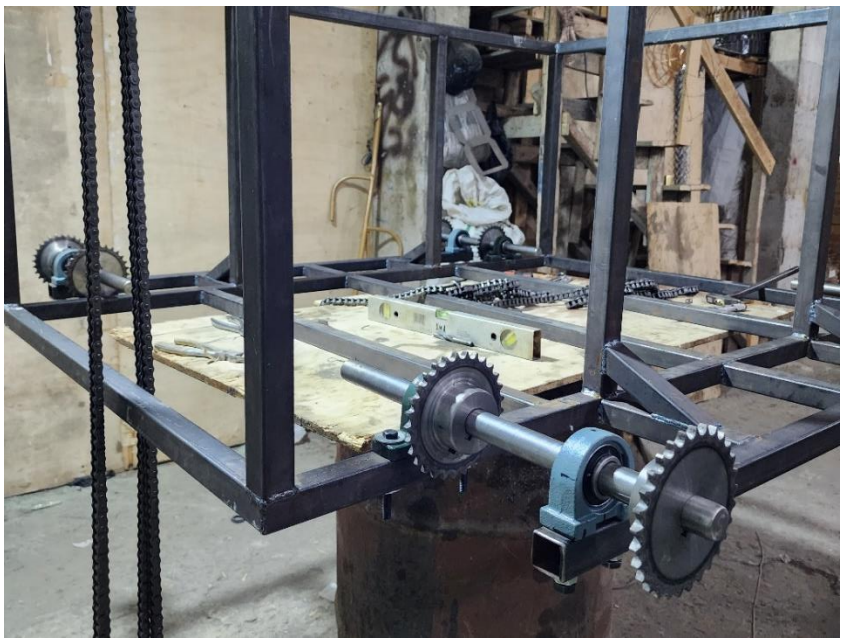


Figure 2.6.12: Final shape after mounting bearings, shafts and sprockets to the chassis

d. Mounting of the motors on the chassis

- The motors are mounted on two parallel steel bars using bolts and nuts, these bars are mounted firmly on the chassis using bolts and nuts.
- We have also designed and manufactured an encoder holder using a 3d printer with the specs. that we mentioned in the designing phase and its shape is as follows:

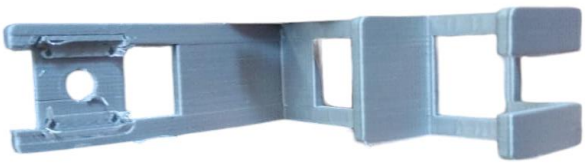


Figure 2.6.14: Encoder holder 3d printed

Figure 2.6.13:Encoder holder mounted on the motor

2.6.3.2 Manufacturing and assembly of suspension system mechanism

The suspension system mechanism was critical and hard to be designed and manufactured. This system is responsible for carrying all the robot's weight and responsible for the stability of it.

After designing all the parts, we moved on with these following steps:

1. We manufactured a testing part from the parts of the mechanism
2. After testing and ensuring that it is efficient and reliable, we managed to use this mechanism and go with it.

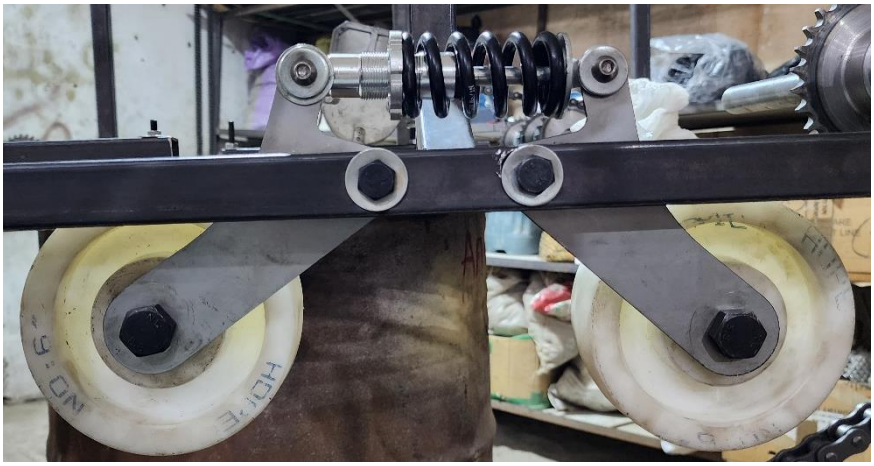


Figure 2.6.15: Suspension mechanism testing phase

3. We bought a sheet metal with thickness (3mm) and then cut it using a laser cutting machine to give us the exact dimensions and shape.

4. These parts are then got assembled on the chassis with the other components (**Wheels and suspension module**) to give us the final shape of the mechanism.

Figure 2.6.16: Suspension Spring module

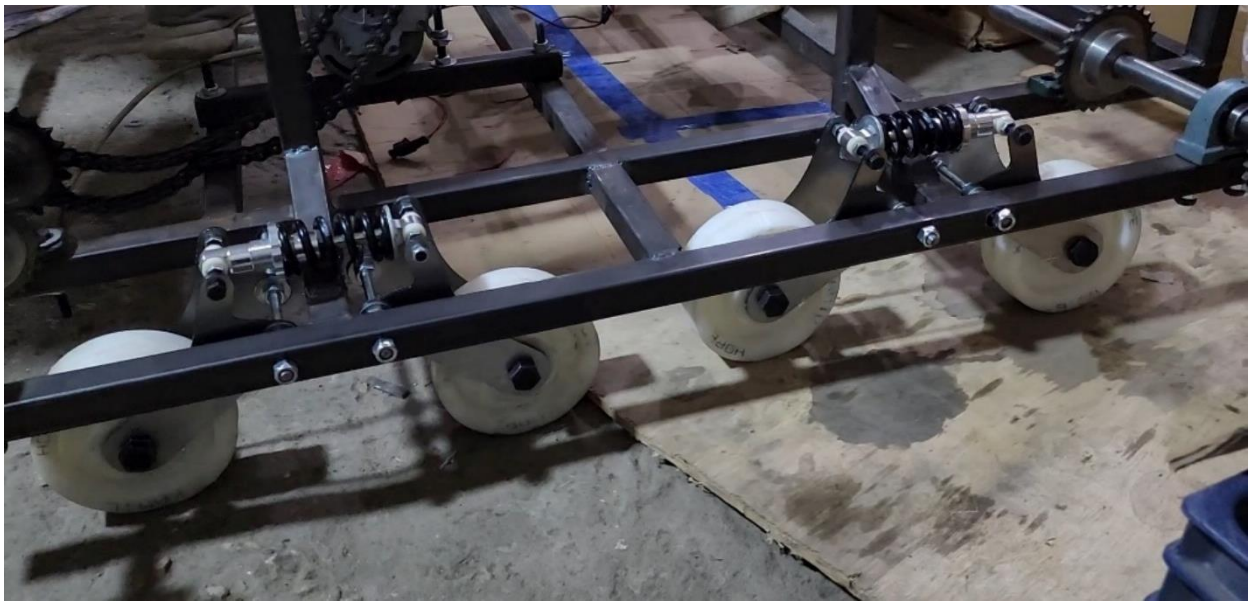


Figure 2.6.17: Final assembly of the suspension system

2.6.3.3 Manufacturing and assembly of the tracked chain

The tracked chain itself was very challenging as we found many methods varying greatly in prices, but we finally managed to go with a method of them which is welding the sheet metal parts directly to the chains and it goes through the following steps:

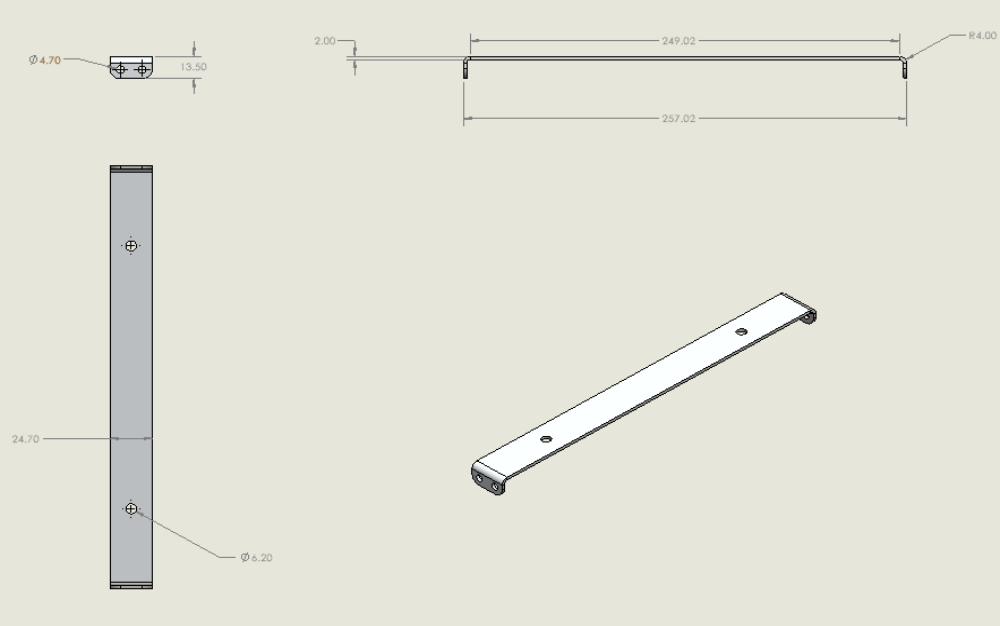


Figure 2.6.18:Dimensions of the chain's sheet part

1. We bought the chains with the required pitch and length

2. We bought sheet metal (2mm) and then got it cut using a laser cutter machine according to the designed dimensions.

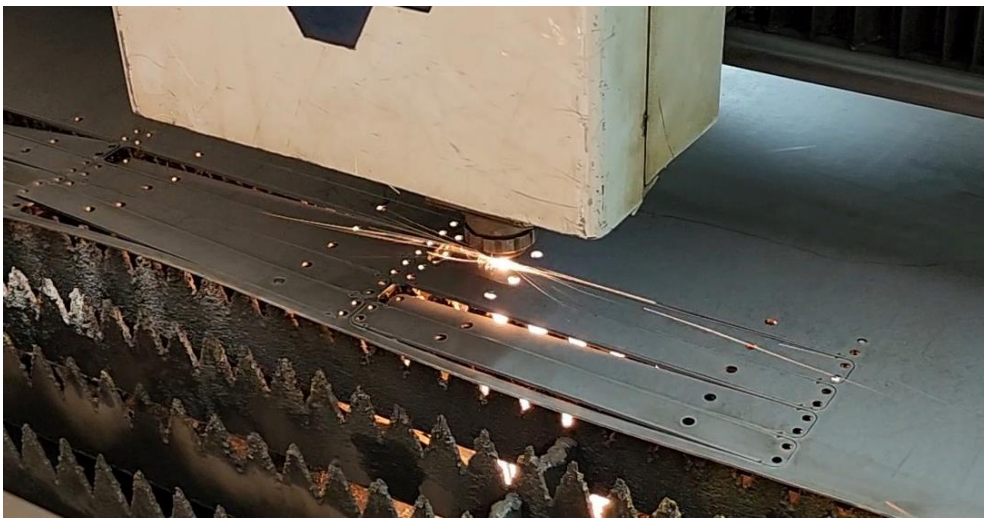


Figure 2.6.19: Cutting sheet metal on laser cutter

3. Those cut parts are then bent to give us the required shape.

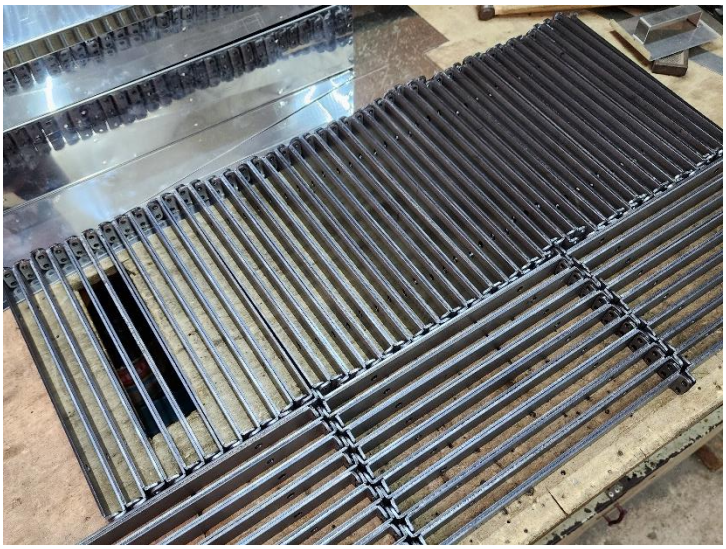


Figure 2.6.20: Sheet metal parts after being bent

4. We took all these part and chains to the welder to weld all the metal parts to the chains like this testing part:

Figure 2.6.21: Welding test of the sheet metal to the chain

Figure 2.6.22: Welding test of the sheet metal to the chain

5. After welding all the metal parts to the chains using (Argon welding) like the previous example this was the final shape of the chain.

Figure 2.6.23: Tracked chain after welding

6. The chain itself got placed on the sprockets and the shafts to give us the final shape of the motion system.

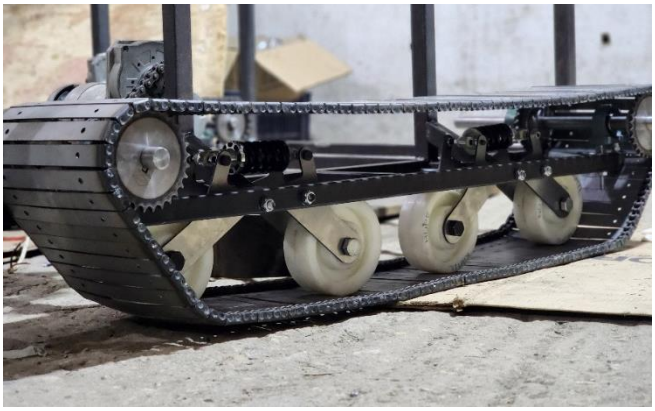


Figure 2.6.24: Final shape of the tracked chain after assembly on the chassis

Figure 2.6.25: Final shape after assembling the two chains

Figure 2.6.26: Final shape after assembly and covering

Chapter 3

DESIGNING AND MANUFACTURING OF THE ARM

3.1 Mechanical design of the robotic arm:

The robotic arm in the solar panel cleaning robot plays a vital role in ensuring the efficient and thorough cleaning of solar panels.

3.1.1 Construction of the Robotic Arm:

3.1.1.1 Structural Components

- **Base**: It's the foundation of the robotic arm, providing stability and support for the entire structure. It's constructed using 30x30 mm metal cubic beams.

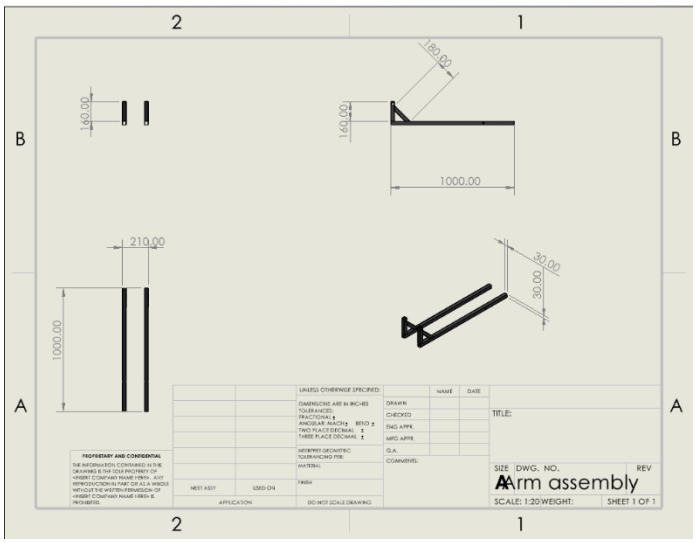


Figure 3.1.1: The Drawing of the robotic arm base

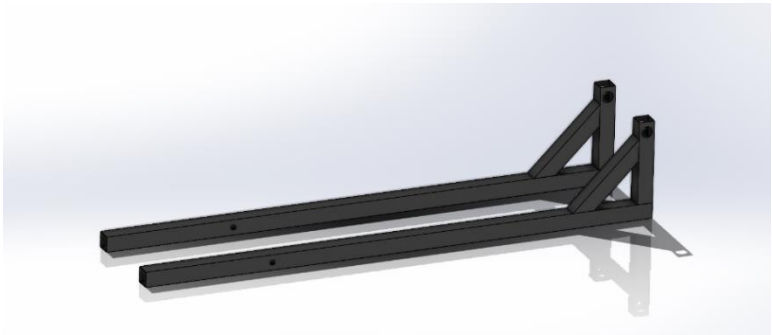


Figure 3.1.2. shows a the base of the robotic arm.

- **Joints**:

These are the connections between the arm's Links and segments, allowing for flexibility and a range of motion.

These joints are all rotational.

The joints are constructed using a ball screw and bearings to minimize friction.

Both the screw and the bearing bore have a diameter of 20 mm.

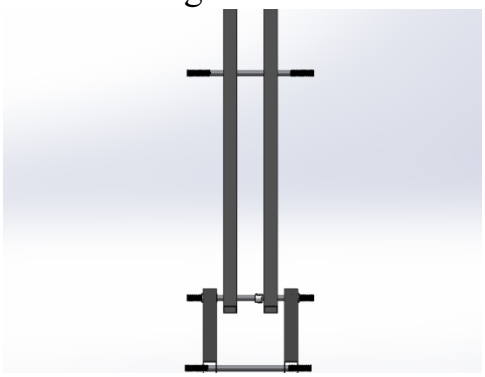


Figure 3.1.3: shows a front view of the base of the arm joined with the first link of the arm through a ball screw and some bearings

- **Links**:

The segments of the arm that connect the joints. These are designed to be strong enough to support the cleaning mechanism and withstand operational stresses.

The arm is constructed of two links, the Lower link (2000mm) and the upper link (1200mm).

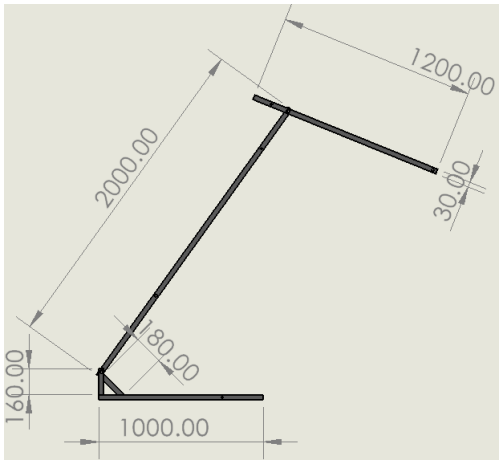


Figure 3.1.4: shows the dimensions of the arm links

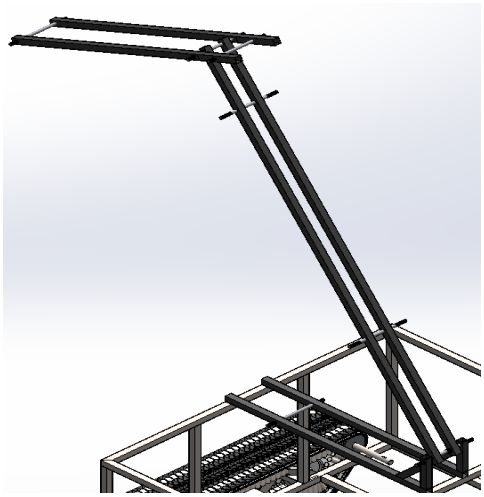


Figure 3.1.5. shows the arm of the robot constructed of two links and a base

- **End Effector (Brush)**:

The part of the arm that comes in contact with the solar panels.

The brush is securely mounted at the end of the motor using a holder made of structural steel bars, as specified in Table 2-1. The design process begins by fixing a PVC pipe to this structural steel holder, see figure 3-6. This PVC pipe is then connected to a cylindrical metal holder, which in turn is attached to a shaft. This shaft is essential as it connects the cylindrical metal holder to the motor, enabling the motor to rotate the brush.

Next, microfiber towels are to be manually fastened around the PVC pipe, forming the cleaning surface of the brush. This combination of materials ensures effective cleaning while being gentle on the solar panels.

To enhance the brush's performance, the structural steel holder is equipped with a suspension system at the top. This suspension system allows for better adjustment of the brush to the surface of the solar panel, accommodating slight variations in panel height. Additionally, it serves as a protective measure to prevent accidental collisions between the brush and the solar panel, ensuring safe and efficient operation.

We constructed the brush manually to make it cost-effective.

Figure 3.1.6: shows the steel holder of the brush

Figure 3.1.7: shows the design of the brush and the brush holder eith the suspension in reality

3.1.1.2 Materials

- **Structural Materials**: We Used structural steel DIN st-37 or S235JR according to EN standards, see Table2-1.
- **Brush Materials**: The brush is made from materials that are effective at cleaning but gentle enough not to scratch or damage the panels, such as Microfiber Mops.

3.1.1.3 Actuators

- **Motors**:

  We used a high-torque linear DC motor for the two links as an actuator and for the brush we also used a linear DC motor with a much smaller stroke length.

  The linear DC motor we used is *Super Jack Electric Linear Actuator DARL 3612-36 Vdc* with a Maximum stroke length of 450 mm for the upper and lower links.[1]

Figure 3.1.8: shows the Linear DC motor we used

The Actuator was fixed to the robot through metallic rings and the placement of the actuators on the arm was determined based on several key considerations:

We calculated the minimum and maximum heights required for the motors and performed angle calculations to determine the optimal distances. This process was carried out using SolidWorks, where we tested various configurations to find the best placement as shown in the figure below.

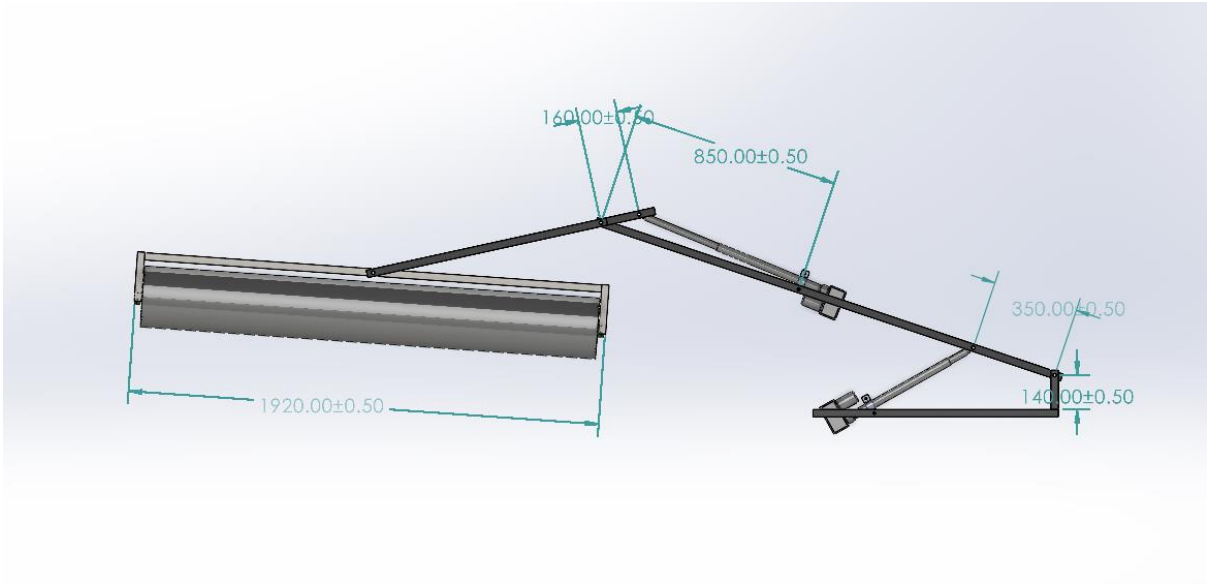


Figure 3.1.9:shows the dimensions of actuators placement to the arm of the robot

3.1.1.4 Sensors

- **Position Sensors**: Used to detect the position and angle of the solar panels and adjust the arm's movements accordingly.

  An ultrasonic sensor can be used for this purpose.

- **Force Sensors**:

  Ensure that the brush applies the correct amount of pressure to the panels during cleaning.

  Although force sensors would significantly enhance the cleaning efficiency and prevent potential damage to the panels, they were not included in the current design due to budget limitations.

- **Read Sensor:**

This sensor, embedded within the actuator, functions as a proximity switch, accurately detecting when the linear actuator is either fully retracted or fully extended

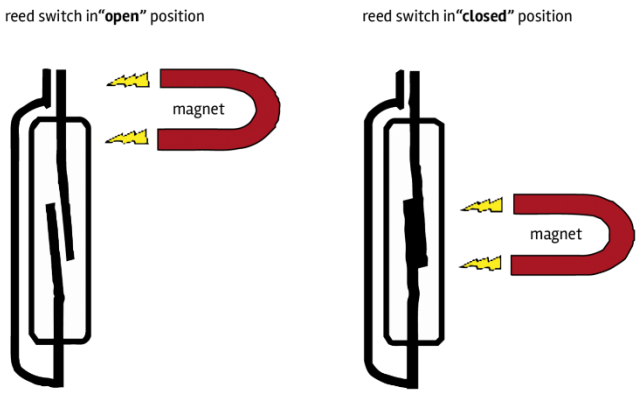


Figure 3.1.10: Reed sensor diagram

- **Potentiometer:**
  We used a potentiometer attached to the joint of the arm, fixed with a 3D-printed holder part that we designed on SolidWorks.

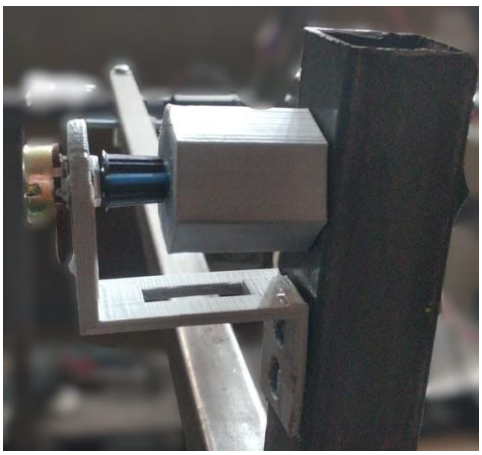


Figure 3.1.11: .shows a Potentiometer holder to hold the potentiometer to the level of the joint of the arm and a 3D-printer coupler to couple the potentiometer and the joint together.

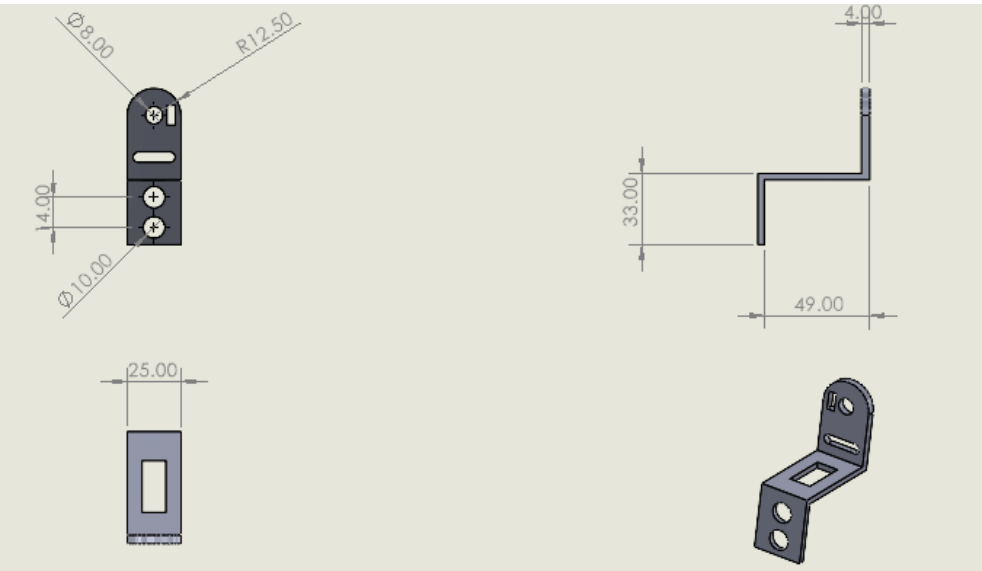


Figure 3.1.12: the working drawing of the potentiometer holder

Once the potentiometer is secured at the joint level using the holder illustrated in Figures 3-9 and 3-10, it is then coupled with the arm joint through a coupler attached to a knob, as depicted in the figure below.

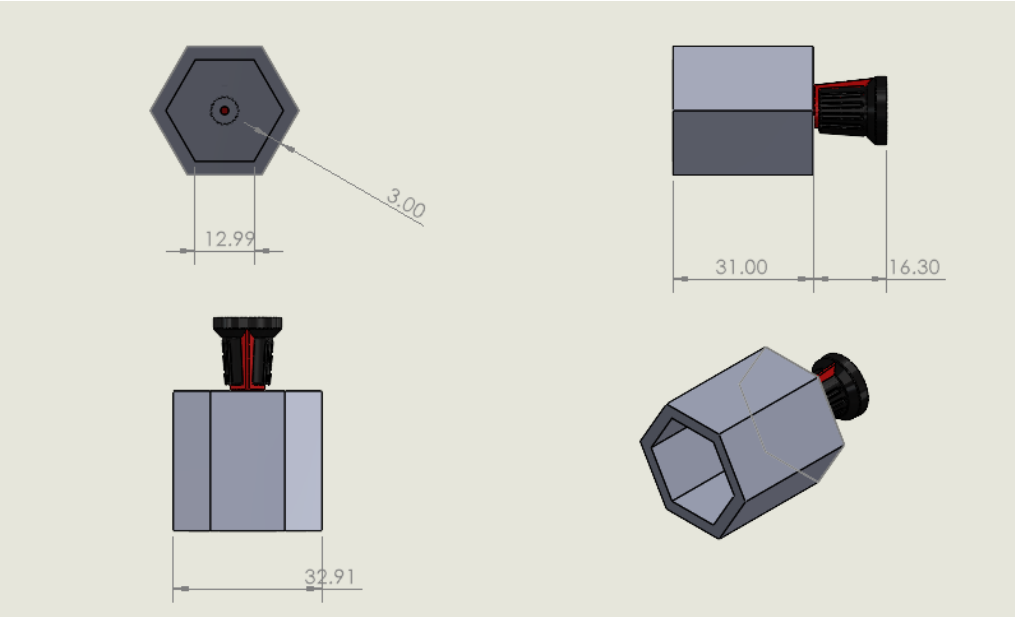


Figure 3.1.13: shows the potentiometer coupler attached to a knob

The potentiometer's main function is to provide feedback on the arm's position.

This feedback is then used to calculate the current angle of the arm, which allows us to adjust the arm to a desired angle as needed.

3.2 Kinematics of the arm:

The kinematics analysis on the manipulator is an important part of the robot study. The manipulator is composed of a serial of rigid links connected to each other with revolute or prismatic joints. Each joint location is usually defined relative to neighboring joint. Calculating the position and orientation of the end-effector of the manipulator by the given joint angles is known as forward kinematics. Inverse kinematics is the process of determining the joint configurations of a robotic manipulator to achieve a desired end-effector position and orientation in the operational space[2].

Kinematics is crucial because it helps us understand and predict the motion of robotic systems without considering the forces that cause this motion. It allows us to design and control robots accurately, ensuring they follow desired paths and perform tasks precisely.

3.2.1 Forward Kinematics:

- **Denavit-Hartenberg (D-H) Parameters:**

Denavit and Hartenberg had put forward a matrix method to build the attached coordinate system on each link in the joint chains of the robot for describing the relationship of translation or rotation between the contiguous links in 1955.[2]

Denavit-Hartenberg (D-H) parameters are used in robotics and mechanical engineering for standardizing and simplifying the description of joint relationships. They facilitate accurate kinematic modeling, streamline coordinate transformations between joints, support key algorithms like forward and inverse kinematics, and provide a physical basis for understanding robotic systems.

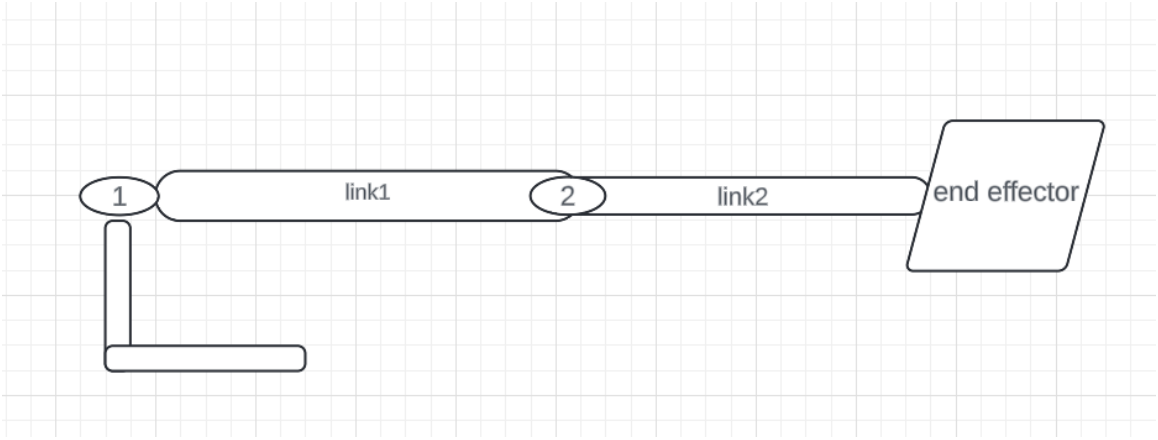
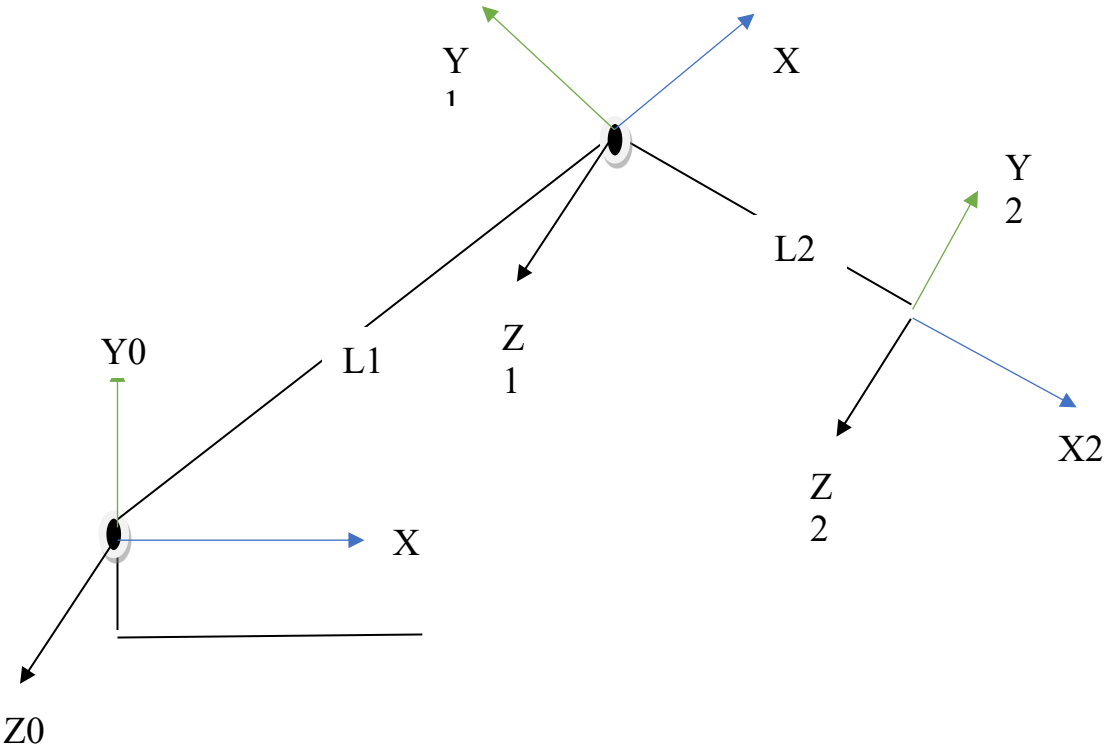


Figure 3.2.1: shows the schematic configuration of the arm



joint	type	theta	alpha	a	d
1	R	$\theta 1$	0	L1	0
2	R	$-\theta 2$	0	L2	0

Table 3.2-1: DH-Parameters

$$ {}^{0}_{1}T = \begin{bmatrix} c1 & -s1 & 0 & L1c1 \\ s1 & c1 & 0 & L1s1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}_{2}T = \begin{bmatrix} c2 & -s2 & 0 & L2c2 \\ s2 & c2 & 0 & L2s2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}_{2}T = {}^{0}_{1}T \,.\, {}^{1}_{2}T = $$

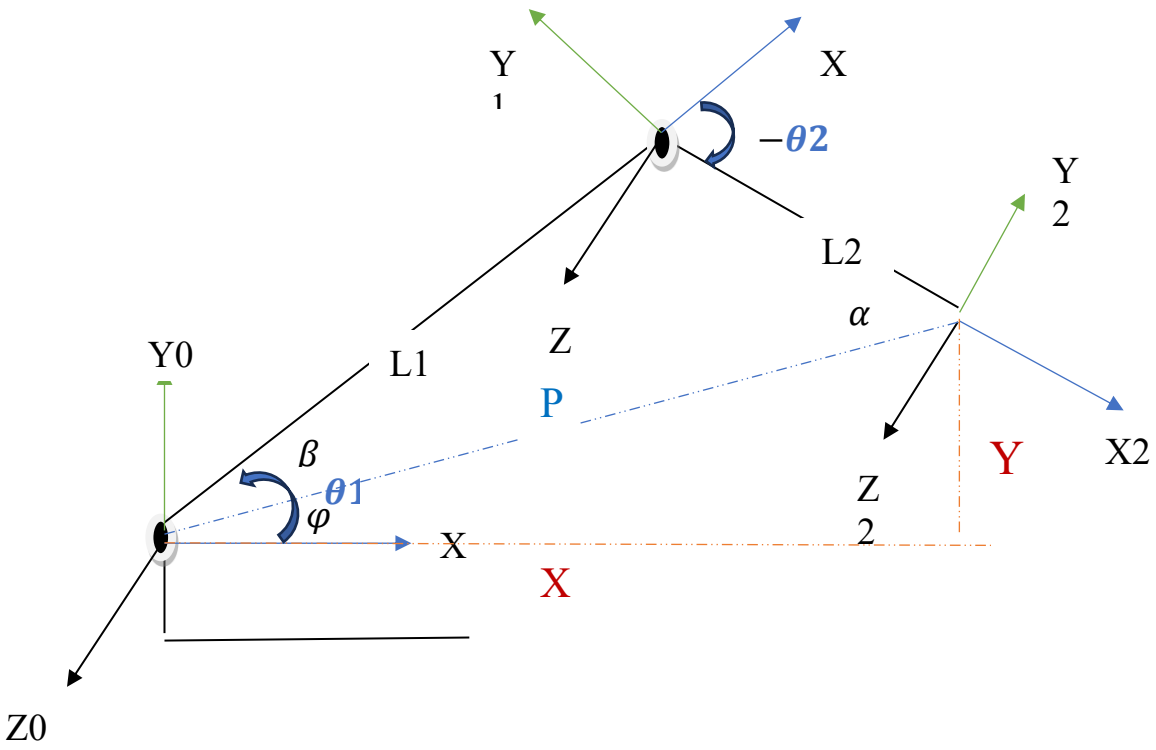
$$ \begin{bmatrix} c1 & -s1 & 0 & L1c1 \\ s1 & c1 & 0 & L1s1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} c2 & -s2 & 0 & L2c2 \\ s2 & c2 & 0 & L2s2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = $$

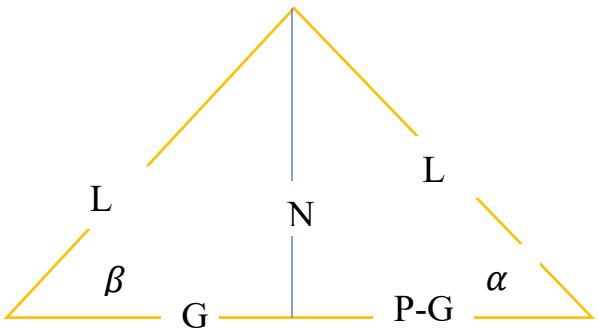
$$ \begin{bmatrix} c12 & -s12 & 0 & L1c1 + L2c12 \\ s12 & c12 & 0 & L1s1 + L2s12 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

3.2.2 Inverse kinematics:

Here, the goal is to find the joint angles or parameters that achieve a desired position or orientation of the end-effector. It's like solving the puzzle of how to position a robot's arm to reach a specific point in space, considering constraints like joint limits and reachability. Inverse kinematics is used to calculate the corresponding joint angles when the posture of the end effect is known. In this work, the graphical method is adopted.

The graphical method in inverse kinematics uses visual representations, like geometric shapes or coordinate systems, to depict the relationship between joint angles and the end-effector's position. It simplifies complex calculations, offering intuitive solutions through iterative adjustments of joint parameters. Advantages include visual clarity, simplicity in concept, and ease of implementation, making it valuable for both educational purposes and practical applications in robotics.





$\Theta_1 = \Phi + \beta$

$\varphi = tan^{-1}(\frac{Y}{X})$

$\alpha = tan^{-1}(\frac{N}{P-G})$

$L1^2 = N^2 + G^2$ (1)

$L2^2 = L1^2 - G^2 + P^2 + G^2 + 2PG$

$\Theta_2 = -(\alpha+\beta)$

$\beta = tan^{-1}(\frac{N}{G})$

$P = \sqrt{X^2 + y^2}$

$L1^2 - G^2 = N^2$ (2)

$x = (-L2^2 + L1^2 + P^2)/2P$

The main purpose of the arm is to position the brush, hanging from the end effector, onto the solar panel. For our calculations, we will select one model as a reference. The robot will work on a tracking panel that will be cleaned while in a rest mode (0 degrees). The panel has a height of 1.2 meters and a width of 2.33 meters, placing the center of the panel 1.165 meters from its edge. The base of the arm is located 1.55 meters from the edge of the panel and is 0.7 meters above the ground. Therefore, the center of the panel is 2.7 meters in the x-direction and 0.5 meters in the y-direction relative to the arm base.

Given these dimensions, there are many possibilities for the lengths of the arm's links. For this analysis, we will assume the first link length is 2 meters and the second link length is 1 meter. We will then verify whether these lengths are appropriate.

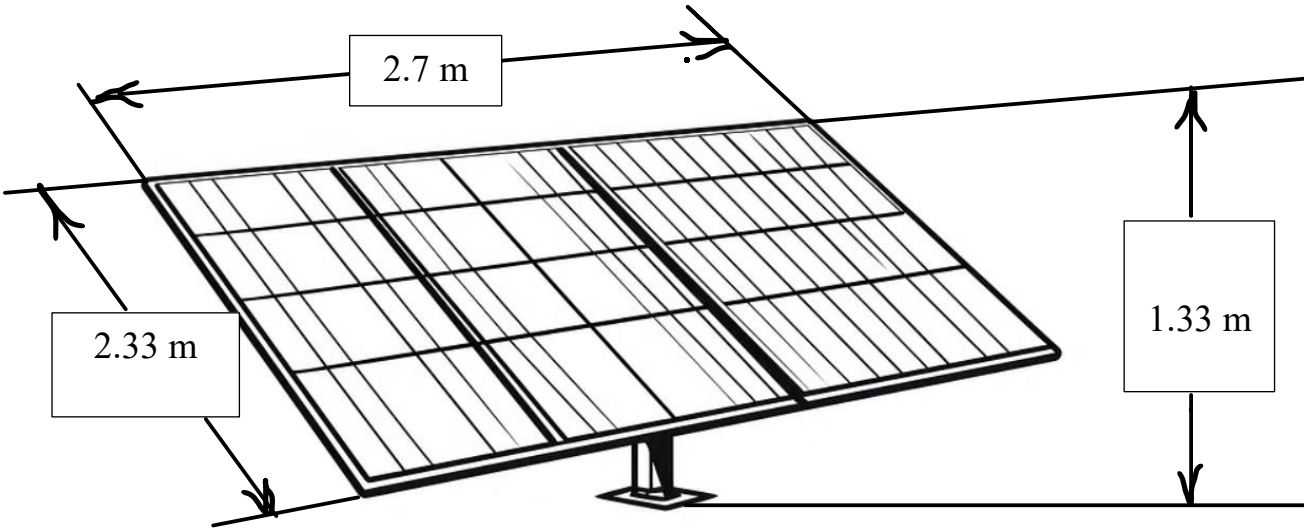


Figure 3.2.2: shows a schematic for the standard dimensions of the solar panels in Egypt in a BenBan Solar Farm

3.3 Motion study

After determining the lengths of the links in the previous section, we will now study the motion of the robotic arm. The arm's primary task is to move the brush and place it on the solar panel, which is a straightforward and repetitive task. Therefore, the robot's movement will involve navigating between specific, known points, repeatedly transitioning to and from the solar panels.

We can now select the previously mentioned point and use inverse kinematics equations to determine the angles of the joints required for each position.

point	Point description	X	Y	Theta 1	Theta 2
1: rest point start	The point where all the joints are at their minimum	0.8	0.7	20	- 20
2: via point	Point in the middle of the path between start and target for safety	1.4m	0.7m	56.0518	-23.8835
3: goal point	The point where the end effector is placed on the panel	2.6m	0.5m	29.9448	-19.9217

Table 3.3-1: The possible angles and positions of the arm

3.3.1 Workspace generation:

The goal of workspace generation is to find out the complete set of poses of the manipulator's end-effector in the Cartesian space when the manipulator runs through all possible configurations in the joint space [3].

The working space is a parameter that directly influences the robot arm's ability to perform tasks and its kinematic capabilities. It serves as an important indicator of the range of activities that the robot arm can effectively carry out within its operational capabilities

Robot arm and the sizes of workspace according to the design requirements, must reach the required working space, which is reasonable to judge about the structure, operability and flexibility of the robot arm.

Using the following MATLAB code, we can easily determine the workspace of the 2-DOF planner robot:

For L1=2; L2=1.

```
theta1_range = linspace (20, 90, 100);
theta2_range = linspace (20, 130, 100);
```

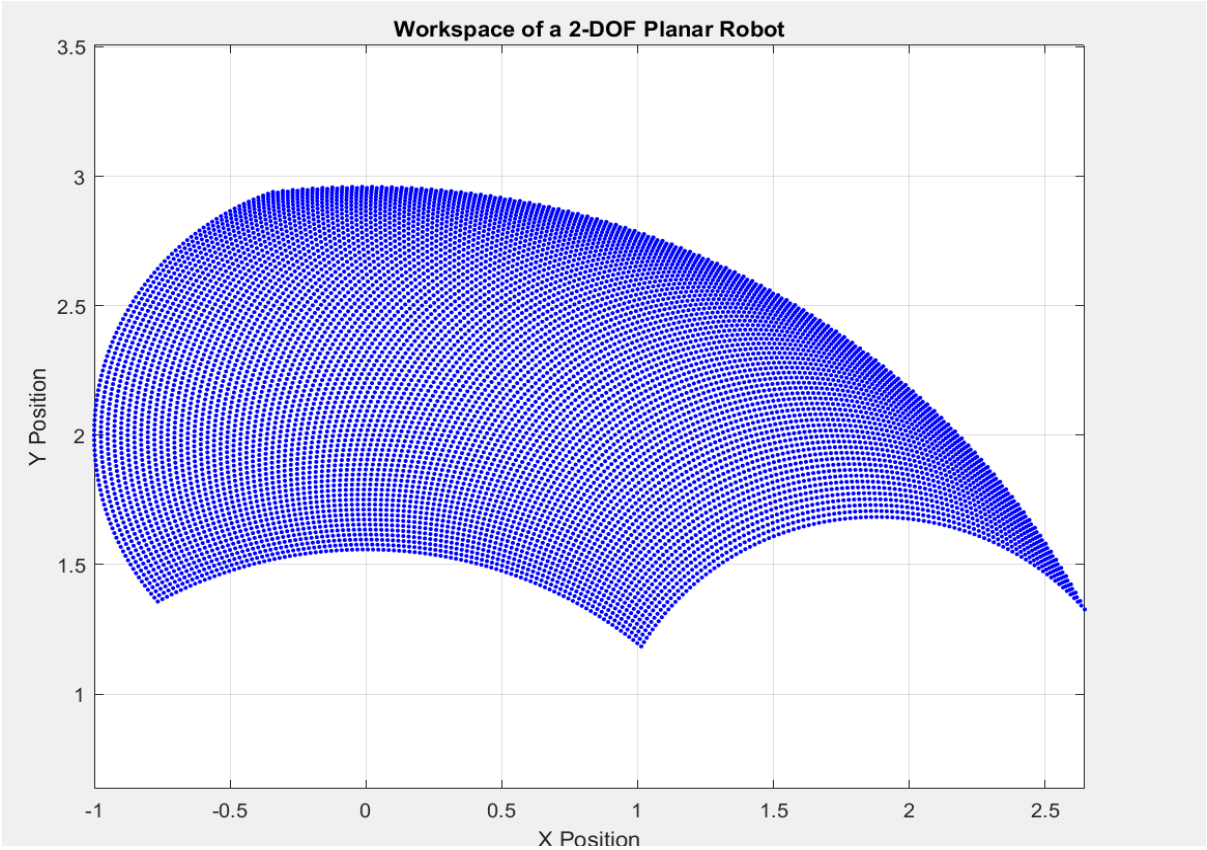


Figure 3.3.1: shows the workspace pf the 2-DOF planner robot on MATLAB

Therefore, from the above, we conclude that the lengths of links and the angles range are suitable for our application.

For a proper analysis of the performance of a robotic manipulator, computing the reachable workspace is not enough.[3]

So, the next steps will be evaluating the trajectory and dynamics of the arm.

3.4 Trajectory:

Trajectory planning is a crucial aspect of robotic control, as it determines the desired path, position, velocity, and acceleration of a robot's end-effector or joints over time. Effective trajectory planning is essential for ensuring smooth, efficient, and safe robot operation, allowing the robot to navigate its workspace while avoiding obstacles and achieving the desired task objectives.

In this case, we are planning the trajectory of a two-link planar robot using the relative angle between the two links, rather than the absolute joint angles. This approach offers several advantages, such as intuitive control, simplified trajectory planning, and the ability to maintain a specific end-effector orientation. By focusing on the relative angle, we can plan the trajectory more flexibly and intuitively, while still providing the necessary information to perform the dynamic calculations and control the robot's motion.[4]

Trajectory planning can be conducted either in joint-variable space or in Cartesian space. For joint-variable space planning, the time history of all joint variables and their first two-time derivatives are planned to describe the desired motion of the manipulators.

For the trajectory planning we represent the angle, and its first two derivatives using cubic polynomials.

cubic polynomials are often chosen in trajectory planning because they provide a good compromise between simplicity and smoothness, making them suitable for many applications where smooth and continuous motion is required.

3.5 Dynamics:

Robot arm dynamics deals with the mathematical formulations of the equations of robot arm motion. The dynamic equations of motion of a manipulator are a set of mathematical equations describing the dynamic behavior of the manipulator.

The dynamics problems are similar to the study of kinematic problems, with forward dynamics problems and inverse dynamics problems. The so-called forward dynamics problem is seeking the movement of the system, given the force acted on the system. The inverse dynamics problem is the opposite given the movement of the system, find the force acting on the system at this time. Specifically, for the robot, the forward dynamics problem is that given the driving torque τ, find position of joints q, velocities of joints $\dot{q}$, and the robot joint acceleration $\ddot{q}$. For an inverse dynamics problem, for a known set of the robot joint position q, joint speed $\dot{q}$, and joint acceleration $\ddot{q}$, find the driving force τ on the robot joints currently.[5]

There are two commonly used methods for formulating the dynamics , based on the specific geometric and inertial parameters of the robot: the Lagrange–Euler (L–E) formulation and the Recursive Newton–Euler (RN–E) method .

The Lagrange-Euler (L-E) formulation and the Recursive Newton-Euler (RN-E) method are both pivotal in understanding the dynamic behavior of robotic systems, yet they serve distinct purposes due to their methodological differences. The L-E formulation employs Lagrangian mechanics to derive exact solutions for robot dynamics, ensuring precision in modeling and analysis, albeit at the cost of computational complexity, particularly with higher degrees of freedom. In contrast, the RN-E method prioritizes computational efficiency by iteratively calculating joint torques and forces, making it ideal for real-time applications such as control algorithms and dynamic simulations. While both approaches are equivalent in describing robot motion dynamics, their suitability depends on the specific needs of the application, balancing between accuracy and computational feasibility in robotics research and development.

In our case, we choose to work with the Newton Euler method.

3.6 Integration with tracked robot body:

3.6.1 Mounting and Stability:

- **Secure Attachment**:

  The robotic arm is securely mounted on the tracked robot body to ensure stability during operation and withstand the dynamic forces encountered during movement and cleaning.

  The robot base is welded to the chassis of the robot, providing additional stability and strength to the overall structure.

- **Balanced Design**:

  The placement of the arm on the robot body is carefully considered to maintain balance and prevent tipping or instability, especially when the arm is fully extended.

  The robot arm system is mounted on the top center of the robot to ensure the stability of the robot's center of mass during movement.

Figure 3.6.1: the right view of the robot and the center of mass is nearly at the center of the robot

3.6.2 Maintenance and Accessibility

- **Ease of Access**: The design allows for easy access to both the robotic arm and the tracked base components for maintenance and repairs. This minimizes downtime and ensures the robot remains operational for longer periods.
- **Modular Components**: The use of modular components for both the arm and the base facilitates quick replacements and upgrades, enhancing the robot's longevity and adaptability to future improvements.

3.7 Validation of the design:

During the testing phase, we identified a significant vibration issue with the robotic arm. This vibration primarily stems from arm's length and insufficient material strength. Here are the key points related to this observation:

3.7.1 Vibration Issues:

- We observed noticeable vibrations in the arm, not directly from the actuator, but due to the arm's extended length.
- The materials used for the arm were not robust enough to handle the dynamic forces during operation.

3.7.2 Deflection Concerns:

- There is a minor deflection at the top end of the arm, which contributes to the vibration issue. While the arm appears rigid and robust in general, the deflection, albeit small, affects its stability.
- The deflection occurs primarily because the base of the arm is fixed too low, creating a larger unsupported length.

3.7.3 Potential Solutions:

- We could address this issue by using stronger materials to construct the arm, which would reduce vibrations and enhance overall stability.

- Another potential solution is to adjust the mounting point of the lower motor higher up the arm. This would limit the length of the unsupported section, thereby reducing both deflection and vibrations.

3.7.4 Lessons Learned and future improvements:

- The vibration issue highlighted the need for better material selection and mounting strategies. Initially, we were unsure of the optimal configuration, and the problem only became apparent post-construction.
- Additionally, the motor's mounting requires a stronger support structure to minimize vibrations further.

By implementing these improvements, we expect to significantly reduce the vibrations and enhance the performance of the robotic arm.

Chapter 4

Robot Control Architecture

4.1 INTRODUCTION

Robot control architecture consists of multiple control levels, each responsible for different aspects of the robot's functionality, from basic actuator control to complex navigation and mapping.

To have a better understand for these levels of control, we will take a top-down approach:

4.2 Control levels

Our application has three control levels, from down to top:

1. Navigation control (motion control).
2. Arm manipulating control
3. Actuator control.

We will discuss each of these levels in a particular chapter later, but for now let's first have a better idea about each component of our control system:

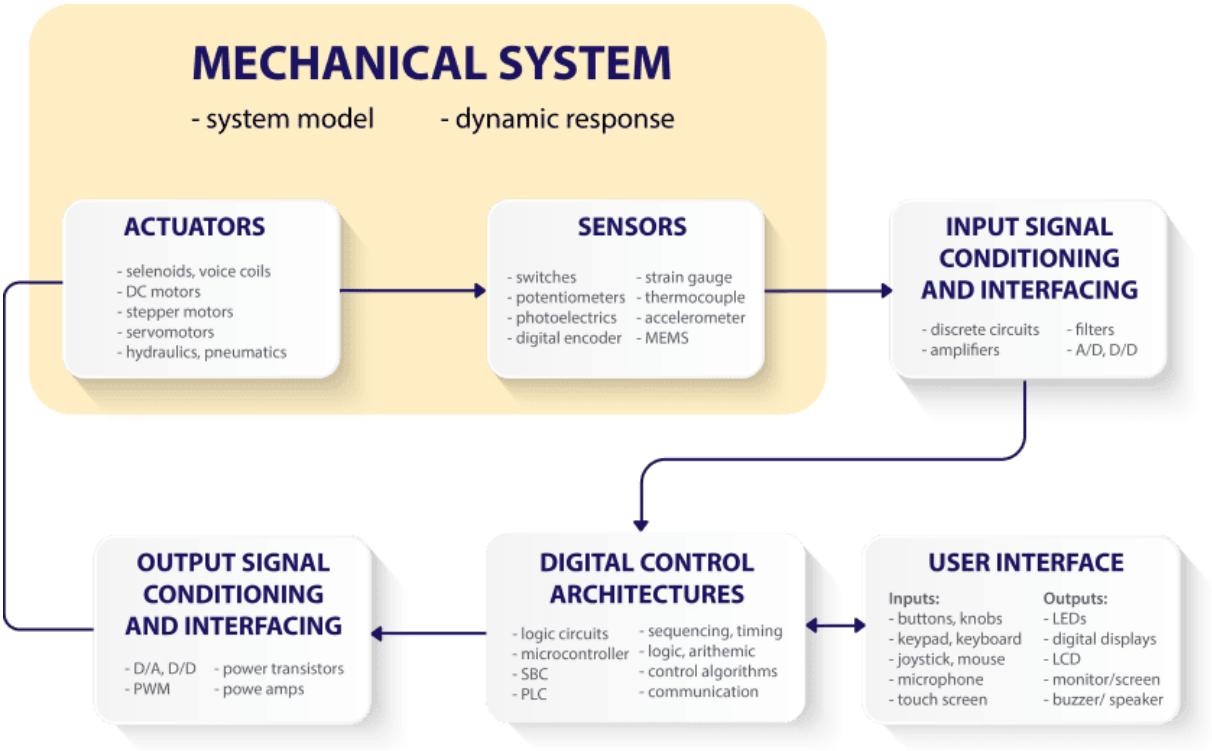


Figure 4.2.1: control system components

4.3 System Components

A mechatronics system integrates mechanical, electronic, and computer engineering to create complex and efficient systems. This system includes various components working together to achieve precise control and automation. Below are the main components of a mechatronics system**:**

1. mechanical system
2. Input Signal Conditioning and Interfacing
3. Digital Control Architectures
4. Output Signal Conditioning and Interfacing
5. User Interface

4.3.1 Mechanical system:

The mechanical system has 2 components to study,

4.3.1.1 Actuators**:**

actuators have been studied mechanically in chapter 2, but we now will discuss it again but from control point of view:

An actuator is a device that converts energy into motion. It is an essential component in various systems and machines, allowing them to perform physical tasks. Actuators receive control signals and use external energy sources to produce mechanical movement, which can be linear or rotary. They are commonly used in robotics, industrial automation, automotive systems, and consumer electronics.

Types of Actuators:

The types of actuators depend on the type of energy entering the actuator that will later be converted into movement.

1. **Electric Actuators**:
   - **Solenoids**: Generate linear motion using electromagnetic fields.
   - **Motors**: Provide rotary motion, with variations like DC motors, stepper motors, and servo motors.

2. **Hydraulic Actuators**:
   - Use pressurized hydraulic fluid to create motion.
   - Known for high force and precise control.
   - Common in heavy machinery and industrial applications.
3. **Pneumatic Actuators**:
   - Utilize compressed air to produce motion.
   - Often used in automation systems due to their simplicity and reliability.
   - Suitable for applications requiring fast and powerful movements.

4. **Thermal and Magnetic Actuators**:
   - Rely on temperature changes or magnetic fields to induce motion.
   - Examples include bimetallic strips (thermal) and voice coil actuators (magnetic).

Working Principle: The basic working principle of an actuator involves three main steps:

1. **Receiving Control Signals**: Actuators receive input signals from a controller, such as a microcontroller or PLC (Programmable Logic Controller). These signals can be electrical, hydraulic, pneumatic, or a combination.
2. **Energy Conversion**: The actuator converts the received energy into mechanical energy. The type of energy depends on the actuator (e.g., electrical energy for electric actuators, hydraulic fluid pressure for hydraulic actuators).
3. **Producing Motion**: The converted energy is used to produce the desired motion, whether linear or rotary. This motion is then used to perform specific tasks such as moving a robotic arm, opening a valve, or positioning a device.

Motor selection

There are three moving parts in our project needs to be attached to motors:

1. Motion motors attached to the chassis

These motors are responsible for the movement of the whole robot in the location and between rows of solar panels.

From studying the mechanical load specification that is mentioned before in the mechanical section, we have settled that the critical output of this motor is the Torque not the speed as the process of cleaning required low speed to be prepared efficiently.

So, we choose these motors based on this concept.

Motor specification:

- Voltage: 24 V
- Rated Speed: 200 rpm
- Rated Torque Max.: 24 n/m.
- No Load Current: <3 Amps
- Rated Wattage: 500 W (0.33 Horsepower)

Figure 4.3.1: chassis motor

2. Linear motors for arm movement.

These motors are responsible for the movement of the arm from and to the panel to be cleaned.

Figure 4.3.2: HARL3624+ linear motor

After studding the mechanical load of the arm that is mentioned before in the mechanical section, we have settled to choose the HARL3624+ to be our motor with these specifications:

▪ Input	36 V DC or 24 V DC
▪ Lifting power:	3,000 N
▪ Stroke length:	600
▪ Control:	ACME
▪ Speed:	4,2 mm/sec
▪ Duty Cycle:	20%
▪ Environment temperature:	-26°C ~65°C (-15°F~150°F)
▪ End switch:	Adjustable
▪ Sensor:	Reed Switch Sensor
▪ Static load:	4,500 N
▪ Maximum (recommended) size:	100
▪ Azimuth angle:	110º (70º R+º40 L or 40º R+70º L)
▪ Elevation angle:	20º ~ 80º (corrosion resistant

Table 4.3-1: Specs. of linear motor

3. **Motor for the rotary of the brush.**

After studying the mechanical load of the brush that is mentioned before in the mechanical section, we have settled to choose a 12V DC motor with the same BTS7960 Motor Driver.

Reasons for choosing these motors

1. **Powerful performance**
   - **Consistent Operation**: Brushed DC motors offer reliable and consistent performance, which is crucial in applications where dependable operation is necessary.
   - **Durability**: While the brushes and commutator do wear out over time, the overall durability of brushed DC motors can be high with proper maintenance.

2. **Simplicity and ease to control.**
   - **Simple Design**: The construction of brushed DC motors is straightforward, involving fewer components compared to other motor types. This simplicity makes them easy to understand, repair, and maintain.
   - **Direct Control**: Speed and direction can be easily controlled by adjusting the voltage and current. This makes brushed DC motors ideal for applications requiring simple and direct motor control

3. **Cost-Effectiveness.**
   - **Affordable**: Brushed DC motors are generally less expensive to produce, and purchase compared to other motor types, making them a cost-effective choice for many applications.
   - **Low Initial Investment**: The simple design and widespread availability contribute to lowering initial costs.

4. **High Starting Torque.**
   - **Immediate Torque**: Brushed DC motors can provide high starting torque, which is essential for applications that need to start under load or require quick acceleration.
   - **Load Handling**: This capability makes them suitable for applications like electric vehicles, cranes, and elevators, where significant torque is needed right from the start.

5. **High maintenance**
   - Readily Available Parts: Replacement parts, such as brushes and commutators, are readily available, making repairs straightforward and cost-effective.
   - Maintenance: Routine maintenance can significantly extend the lifespan of brushed DC motors, and the required maintenance is relatively simple.

4.3.1.2 Sensors:

In modern autonomous systems, sensors are essential for enabling precise and intelligent interactions with the environment. This section explores the various sensors integrated into our autonomous solar panel robot with an automated car, highlighting their functionalities, working principles, and contributions to the overall system.

The sensors used in this project include the:

- **Inertial Measurement Unit (IMU):** Analyzes the robot's orientation and movement by measuring acceleration and angular velocity.
- **Light Detection and Ranging (LiDAR):** Maps the environment by measuring distances to objects using laser pulses.
- **Encoders:** Provide precise position and speed feedback by converting motion into electrical signals.
- **Ultrasonic Sensors:** Detect nearby objects and measure distances using high-frequency sound waves.

Each of these sensors provides critical data that enhances the robot's performance, accuracy, and safety.

In the following sections, we will explore each of these sensors, discussing their working principles, integration into the autonomous system, and their specific roles in achieving the project's objectives.

A. Inertial Measurement Unit (IMU)

1. Introduction to the MPU9250 Sensor

The MPU9250 is a nine-axis motion tracking device designed by InvenSense, integrating a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis magnetometer. It is commonly used in applications like robotics, drones, and handheld devices for accurate motion and orientation measurement.

Figure 4.3.3: MPU 9250

2. Working Principle and Features

- **Gyroscope**:
  - **Principle**: Measures angular velocity or the rate of rotation around each axis.
  - **Mechanism**: Works based on the Coriolis effect. When the gyroscope experiences angular motion, a vibrating element within the sensor deflects due to this effect, allowing the gyroscope to detect rotational movement.
  - **Specifications**: Offers selectable ranges of ±250, ±500, ±1000, and ±2000 degrees per second (dps). This makes it useful for detecting precise changes in orientation and rotational speed.
- **Accelerometer**:
  - **Principle**: Measures linear acceleration along multiple axes.
  - **Mechanism**: Operates by sensing the force exerted on a mass when the object accelerates. The deflection or strain of the sensing element within the accelerometer is measured and converted into an electrical signal proportional to the acceleration.
  - **Specifications**: Provides selectable ranges of ±2g, ±4g, ±8g, and ±16g. This allows the accelerometer to detect changes in velocity and orientation, aiding in motion tracking and stabilization.
- **Magnetometer**:
  - **Principle**: Detects magnetic fields to determine orientation relative to the Earth's magnetic field.
  - **Mechanism**: Measures the strength and direction of the magnetic field using either magneto-resistive or Hall effect sensors. This helps in providing accurate heading information.
  - **Specifications**: Capable of measuring magnetic fields with a resolution of 4800 μT, making it essential for navigation and orientation purposes.
- **Digital Motion Processor (DMP)**:
  - **Function**: Offloads computation of motion processing algorithms from the host processor, reducing workload and power consumption.
  - **Benefit**: The DMP processes complex motion algorithms internally, providing fused sensor data directly, which simplifies integration and improves efficiency.
  -

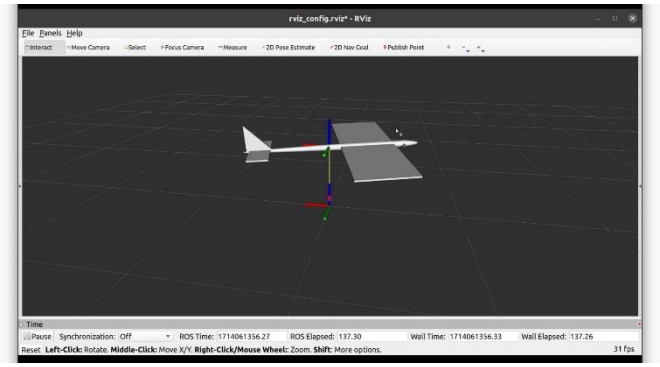
- **Communication Interface**:
  - **Supports**: I²C and SPI interfaces, which facilitate easy integration with various microcontrollers and processors.
  - **Flexibility**: Allows straightforward connection to different types of control systems, enhancing compatibility and ease of use.
- **Power Consumption**:
  - **Efficiency**: Designed to be energy-efficient with low power consumption, making it suitable for battery-operated devices. This is critical for applications requiring long-term operation without frequent battery replacement.

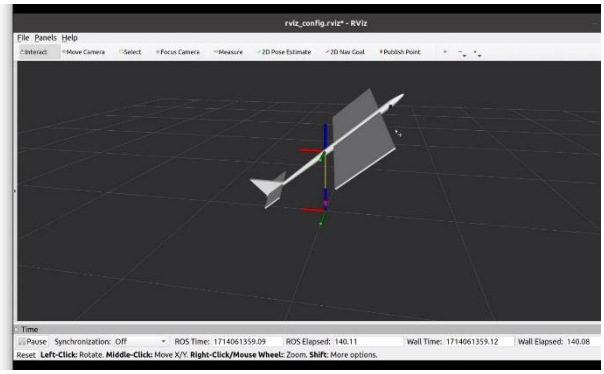
3. Integration into the Autonomous System

The MPU9250 sensor is essential for maintaining stable and accurate orientation and navigation in the autonomous system. By continuously measuring the robot's motion and orientation, the sensor data is used to control movements and ensure the robot follows the correct path.

4. Sensor Fusion Algorithms

To achieve precise motion tracking and orientation, raw data from the gyroscope, accelerometer, and magnetometer are combined using sensor fusion algorithms. These algorithms filter out noise and provide an accurate representation of the sensor's orientation and motion.





B. Light Detection and Ranging (LiDAR)

1. Introduction to the RPLIDAR Sensor

RPLIDAR is a low-cost, high-performance 360-degree 2D laser scanner (LIDAR) developed by SLAMTEC. It is widely used in applications such as robotic navigation, mapping, and obstacle avoidance due to its accuracy and efficiency in capturing the environment's spatial information.

Figure 4.3.4: rplidar A1

2. Working Principle

The RPLIDAR sensor works by emitting laser beams and measuring the time it takes for the beams to reflect off objects and return to the sensor. This time-of-flight (ToF) principle allows the sensor to calculate the distance to objects. The sensor continuously rotates, capturing a full 360-degree view of its surroundings.

- **Time-of-Flight (ToF)**: The RPLIDAR measures the time taken for emitted laser pulses to return after hitting an object. This time difference is used to calculate the distance to the object.
- **360-degree Scanning**: The sensor rotates to capture a complete 360-degree scan of the environment, providing comprehensive spatial data.

3. **Features and Specifications**

    - **High Accuracy**: Capable of measuring distances with millimeter precision.
    - **Long Range**: Can detect objects up to several meters away, depending on the specific model.
    - **High Resolution**: Produces high-resolution point cloud data, allowing detailed mapping and object detection.
    - **Compact and Lightweight**: Designed to be easily integrated into various systems, including small robots.
    - **Low Power Consumption**: Energy-efficient design suitable for battery-powered applications.
    - **Multiple Interfaces**: Supports UART and USB interfaces for easy integration with different control systems.

In this project, the RPLIDAR sensor is used for navigation and mapping. The sensor is mounted on the robot, continuously scanning the environment as the robot moves. The data collected by the RPLIDAR is processed to create a map of the surroundings, identify obstacles, and plan paths for the robot to follow.

The RPLIDAR sensor plays a critical role in the autonomous operation of the robot by providing real-time spatial data. This data is used for:

- **Obstacle Detection and Avoidance**: The sensor detects objects in the robot's path, allowing the control system to navigate around them.
- **Environment Mapping**: The collected data is used to build a map of the environment, which is crucial for tasks like path planning and localization.
- **Path Planning**: The sensor data helps the control system to determine the most efficient route to a target location, avoiding obstacles and navigating through complex environments.

Using the RPLIDAR sensor in this project offers several benefits:

- **Enhanced Navigation**: Provides accurate and detailed environmental data, improving the robot's ability to navigate and operate autonomously.
- **Improved Safety**: Detects obstacles in real-time, reducing the risk of collisions and enhancing the robot's ability to operate in dynamic environments.
- **Versatility**: Suitable for various applications beyond autonomous solar panel robots, including vacuum cleaners, drones, and industrial automation.

Advantages of LiDAR over Camera

For our outdoor project, we chose LiDAR over a camera due to several benefits:

- **All-Weather Operation:** LiDAR performs reliably in various weather conditions, such as bright sunlight, rain, or fog, where cameras may struggle.
- **Night-Time Operation:** LiDAR can operate effectively in low-light or night-time conditions without the need for external lighting.
- **Precision and Accuracy:** LiDAR provides highly accurate distance measurements and detailed 3D maps, essential for precise navigation and obstacle detection.
- **Less Data Processing:** LiDAR generates fewer complex data compared to image processing from cameras, leading to faster and more efficient real-time processing.

Typical specifications of a LiDAR sensor include:

- **Scanning Frequency**: 5.5 Hz (Adjustable from 5 to 10 Hz)
- **Scanning Range:** 0.15 meters to 12 meters
- **Angular Resolution:** ≤ 1° (Depends on the scanning frequency)
- **Angular Range**: 360°
- **Distance Resolution**: < 0.5 mm
- **Distance Accuracy**: < 1% of the distance (0.15m to 6m), < 2% of the distance (6m to 12m)
- **Sample Rate**: 8000 samples per second
- **Input Voltage**: 5 V ± 0.25 V (USB-powered)

- **Power Consumption**: 4 W (Typical)

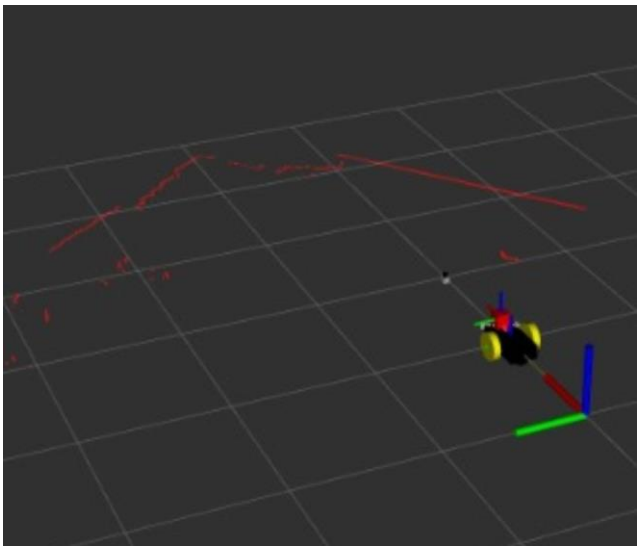
The LiDAR sensor is a vital component in our autonomous solar panel robot, enabling it to navigate and map its environment with high accuracy.

By providing real-time data on the robot's surroundings, LiDAR ensures safe and efficient operation, allowing the robot to perform tasks such as obstacle avoidance and precise positioning of solar panels.

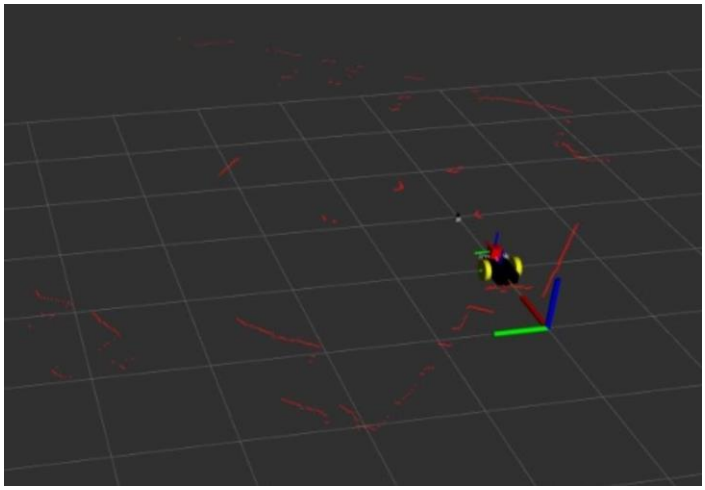
The decision to use LiDAR over a camera was driven by its superior performance in various outdoor conditions and its ability to deliver precise and reliable data for navigation and mapping.

The integration of LiDAR with the Raspberry Pi and Arduino ensures efficient data processing and control, enhancing the overall functionality of the robot.

4. Results



With filter



Without filter

C. AS5600 Encoder

The AS5600 is a contactless magnetic rotary position sensor, which means it can determine the angular position of a rotating object without physical contact. This encoder is widely used in applications requiring precise and reliable position measurement, such as robotics, automotive systems, and industrial automation.

The AS5600 encoder operates using a magnet placed near the sensor. The sensor detects the angle of the magnetic field generated by the magnet and converts this information into an electrical signal.

The key components and principles include:

- **Magnetic Sensing**: The AS5600 uses Hall effect technology to detect the position of the magnetic field.
- **Digital Output**: It provides a digital output representing the angular position of the magnet.
- **Resolution**: The encoder offers a 12-bit resolution, meaning it can distinguish 4096 unique positions within a 360-degree rotation.

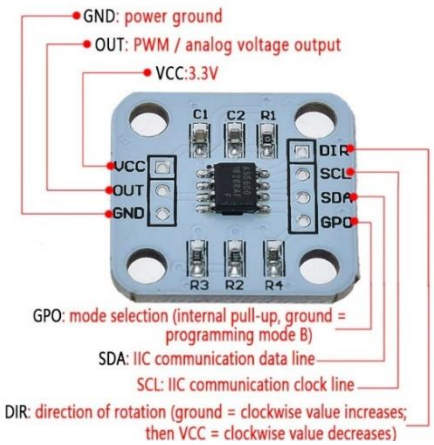


Figure 4.3.5: AS5600 encoder

Features and Specifications

- **Contactless Sensing**: Ensures durability and reliability by eliminating mechanical wear and tear.
- **High Resolution**: 12-bit resolution provides precise angular position measurement.
- **Low Power Consumption**: Designed to be energy-efficient, making it suitable for battery-powered applications.
- **Interfaces**: Supports multiple output interfaces, including analog output, PWM (Pulse Width Modulation), and I²C (Inter-Integrated Circuit).
- **Automatic Magnet Detection**: The sensor can automatically detect and compensate for the position of the magnet, simplifying the setup process.

The AS5600 encoder plays a crucial role in the PID control loop by providing accurate and real-time feedback on the motor's position. Here's how it contributes to each component of the PID controller:

- **Proportional Control**: The encoder feedback helps the PID controller adjust the motor's input to minimize the error between the desired and actual positions.
- **Integral Control**: The cumulative error is corrected over time, ensuring the motor reaches and maintains the desired position.
- **Derivative Control**: The rate of change of error وis used to predict future errors and adjust the motor input, accordingly, providing smoother control.

Using the AS5600 encoder in this project offers several benefits:

- **Precision**: High-resolution feedback ensures precise control of the motor's position.
- **Reliability**: The contactless design reduces the risk of mechanical failure.
- **Ease of Integration**: The automatic magnet detection and multiple interface options simplify the integration process.

In this project, the AS5600 encoder is integrated with the DC motors to provide precise feedback on their rotational position. The encoder is mounted in such a way that the rotating shaft of the motor is aligned with the magnet.

As the motor shaft rotates, the AS5600 measures the angle of the magnetic field and sends this information to the controller.

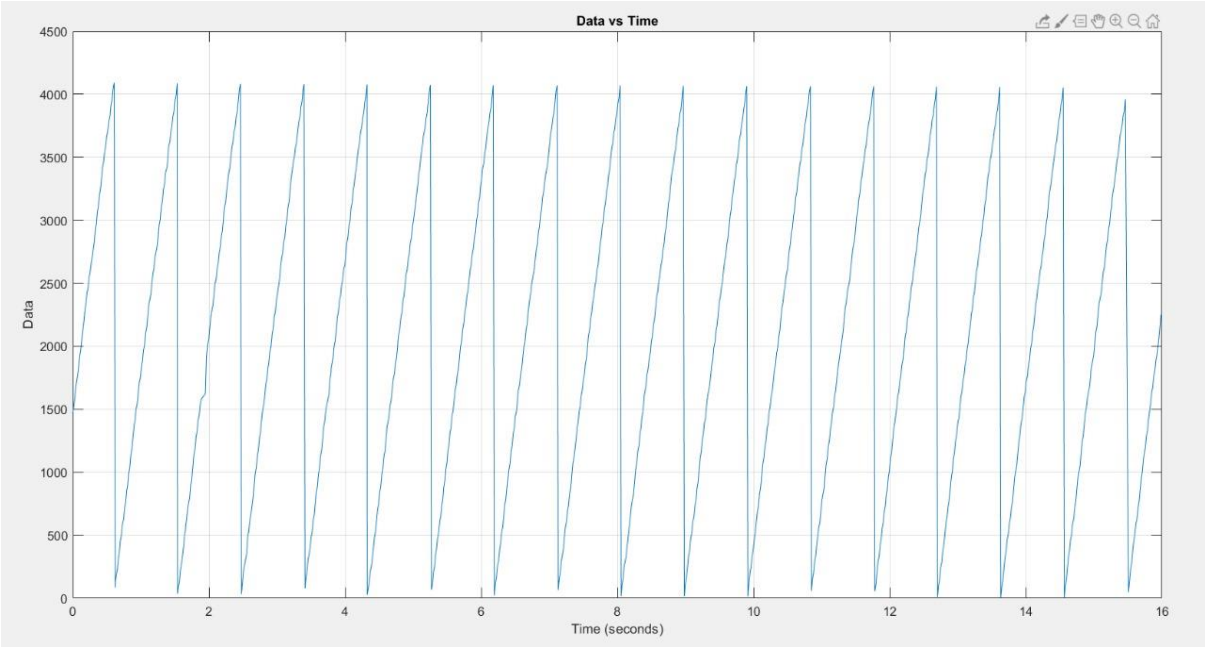
This section details the procedure for integrating and interfacing encoders via the I2C protocol within the ROS framework.

1. **Hardware Setup**
   - **Encoder Connection**: Ensure proper wiring of the encoder to the Raspberry Pi using the I2C interface. Refer to the encoder datasheet for pin configuration and connection details.
   - **Power Supply**: Provide adequate power to both the encoder and the control system to ensure reliable operation
2. **Software Configuration**
   - **ROS Installation**: Install and configure ROS on the Raspberry Pi or the microcontroller platform. Initialize the ROS core (roscore) to establish communication across nodes.
   - **I2C Setup**: Initialize the I2C communication protocol on the Raspberry Pi to facilitate data exchange with the encoder. Configure the I2C address and communication parameters as specified by the encoder manufacturer.
   - **ROS Node Development**

   - **Package Creation**: Develop a dedicated ROS package or extend an existing one to accommodate encoder data processing and publishing.
   - **Node Implementation**: Implement a ROS node responsible for reading encoder data via the I2C interface. Utilize ROS publisher functions to broadcast encoder readings to designated ROS topics.

- **Data Processing**
  - **I2C Data Retrieval**: Program the microcontroller/Raspberry Pi to fetch encoder data (e.g., counts, pulses) from the I2C bus using appropriate libraries.
  - **Data Conversion**: The AS5600 encoder generates digital output in the form of ticks or counts, representing the angular position of the encoder shaft. Here's how you can convert these ticks into meaningful metrics like angular position or velocity, ensuring compatibility with ROS message formats.

Results:



Collected data

D. Ultrasonic Sensors (HC-SR04)

Ultrasonic sensors are widely used in various applications for distance measurement and object detection. In the context of your autonomous solar panel cleaning robot, these sensors can play a crucial role in accurately positioning the brushes on the arm to ensure effective cleaning. Below are detailed points about ultrasonic sensors, their working principles, specifications, and how they can be integrated into your project.

Working Principle

- **Sound Wave Emission and Reception**:
  - Ultrasonic sensors operate by emitting ultrasonic waves (sound waves with a frequency higher than the audible range for humans, typically above 20 kHz) from a transmitter.
  - These waves travel through the air and reflect off objects in their path.
  - A receiver in the sensor then captures the reflected waves (echoes).

- **Time-of-Flight Calculation**:
  - The sensor measures the time it takes for the emitted wave to travel to the object and back to the receiver.
  - Using the speed of sound in air, the sensor calculates the distance to the object based on the time-of-flight of the ultrasonic waves.

- **Distance Measurement**:
  - Distance (D) is calculated using the formula:

$$D = \frac{(\text{TimeofFlight}) \times (\text{SpeedofSound})}{2}$$

  - The division by 2 accounts for the round-trip journey of the waves.

Integration into the Autonomous Solar Panel Cleaning Robot

- **Sensor Placement**:
  - Ultrasonic sensors should be strategically placed on the robotic arm to cover the area where the brushes will interact with the solar panel.
  - Multiple sensors can be used to ensure comprehensive coverage and accurate positioning.
- **Data Processing**:
  - The microcontroller (Arduino) collects distance data from the ultrasonic sensors.
  - The data is processed to determine the precise position of the solar panel surface relative to the brushes.
- **Control Algorithms**:
  - The processed distance data is used in control algorithms to adjust the position and pressure of the brushes.
  - Feedback from the sensors ensures that the brushes maintain optimal contact with the solar panel surface for effective cleaning without causing damage.
- **Obstacle Detection**:
  - Ultrasonic sensors can also be used to detect obstacles and prevent collisions, ensuring safe operation of the robotic arm.

Advantages

- **Non-contact Measurement**: Ultrasonic sensors measure distance without physical contact, reducing the risk of damage to the solar panel.
- **Versatility**: Can be used in various environmental conditions, including dust and dirt, which are common in solar panel installations.
- **Cost-effective**: Relatively inexpensive compared to other types of distance sensors.

Reliability: Provide consistent and reliable measurements.

HC-SR04 Ultrasonic Sensor:

- **Frequency**: 40 kHz
- **Range**: 2 cm to 400 cm
- **Resolution**: 3 mm
- **Accuracy**: ±3 mm
- **Beam Angle**: 15°
- **Power Supply**: 5V DC

Figure 4.3.6: HC-SR04 ultrasonic

E. Potentiometer

A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. It is commonly used to measure angular position and control mechanical movements in robotic systems.

- **Voltage Divider**: The potentiometer functions by dividing an input voltage based on the position of its wiper. As the wiper moves, the resistance changes proportionally, altering the output voltage.
- **Output Voltage**: The output voltage varies linearly with the position of the wiper. This output voltage can be read by an analog-to-digital converter (ADC) in a microcontroller to determine the wiper's position.

Rotary Potentiometers measure angular displacement. Suitable for applications requiring precise control over rotational movement.

Figure 4.3.7: rotary potentiometer

Specifications

- **Resistance Range**: Typically, 1 kΩ to 100 kΩ.
- **Linearity**: Indicates how closely the output voltage corresponds to the wiper position, usually within ±1%.
- **Rotation Angle**: Typically, 270° to 300° for rotary potentiometers.
- **Power Rating**: Determines the maximum power the potentiometer can handle, usually specified in watts.
- **Tolerance**: Indicates the accuracy of the resistance value, typically ±10% to ±20%.

Integration into the Robotic Arm

1. **Installation**:
    - Attach the rotary potentiometer to the joint of the robotic arm.
    - Ensure that the potentiometer's shaft is securely connected to the rotating part of the arm.
2. **Wiring**:
    - Connect the potentiometer's outer terminals to the power supply and ground.
    - Connect the wiper terminal to an analog input pin on the microcontroller.
3. **Reading the Position**:
    - Use the microcontroller to read the analog voltage output from the potentiometer.
    - Convert the analog voltage to a digital value using the ADC.
4. **Calculating the Angle**:
    - Map the digital value to the corresponding angle of rotation using the known range of the potentiometer.
5. **Controlling the Linear Motor**:
    - Use the calculated angle to control the linear motor's position.

- Implement a feedback loop where the potentiometer provides continuous position updates to the microcontroller, ensuring precise control over the arm's movement.

Advantages

- **Simplicity**: Easy to integrate and use with basic analog-to-digital conversion.
- **Cost-Effective**: Generally inexpensive compared to other position sensors.
- **Accuracy**: Provides reliable and accurate position measurements suitable for many applications

4.3.2 Input Signal Conditioning and Interfacing:

This component of the system is responsible for modifying the input signal to make it suitable for the controller and to make it valuable for the application we need it in.

4.3.2.1 Sensor Fusion

In the development of my autonomous solar panel robot, combining data from multiple sensors—known as sensor fusion—was a crucial aspect. This process enhances the robot's ability to navigate, detect obstacles, and operate efficiently by integrating information from the IMU, LiDAR, encoders, and ultrasonic sensors.

Implementing Sensor Fusion

Integrating data from these sensors was important for enhancing the robot's navigation capabilities. Here are some of the techniques have been used

1. Fusion of IMU Data Using Mahony and Madgwick Filters

Fusing data from the Inertial Measurement Unit (IMU) was crucial for accurately determining the robot's orientation and movement. The IMU, specifically the MPU9250, provides raw data on acceleration, angular velocity,

and magnetic field strength. To make this data useful for navigation and control, we implemented sensor fusion algorithms, focusing on the Mahony and Madgwick filters.

Madgwick Filter

The Madgwick filter is an orientation filter designed to efficiently fuse IMU data. It uses a gradient descent algorithm to estimate the orientation of the IMU by minimizing the error between the measured and estimated direction of the gravity vector and the magnetic field.

- **How It Works**: The filter leverages accelerometer and gyroscope data to calculate the orientation. It updates the quaternion representation of the orientation using the gyroscope data and corrects this estimate with the accelerometer and magnetometer data to account for drift.
- **Advantages**: The Madgwick filter is known for its good performance in real-time applications. It can provide accurate orientation estimates without requiring extensive computational resources [1].

Mahony Filter

The Mahony filter, another popular orientation filter, uses a complementary filter approach. It combines gyroscope integration for short-term orientation estimation with accelerometer and magnetometer data for long-term correction.

- **How It Works**: This filter updates the orientation quaternion by integrating gyroscope data and corrects it using feedback from accelerometer and magnetometer measurements. It applies a proportional and integral controller to reduce the orientation error.
- **Advantages**: The Mahony filter offers robust performance, especially in dynamic environments where the accelerometer and magnetometer data can be noisy. Its integral term helps in reducing steady-state errors, making it suitable for applications requiring high accuracy [2].

Why We Chose the Mahony Filter Over the Madgwick Filter

After testing both filters in our project, we opted to use the Mahony filter for several reasons:

1. **Robustness in Dynamic Environments**: The Mahony filter performed better in dynamic conditions with rapid movements and changes in orientation. Its ability to handle noisy data from the accelerometer and magnetometer made it more reliable.
2. **Reduced Steady-State Errors**: The integral term in the Mahony filter helped minimize steady-state errors, crucial for maintaining accurate long-term orientation and ensuring the stability and positioning of the solar panel.
3. **Performance Consistency**: During our tests, the Mahony filter consistently provided more stable and accurate orientation estimates compared to the Madgwick filter. This consistency was essential for the autonomous navigation and control algorithms used in our robot.

The output of the Mahony filter, as with the Madgwick filter, provides the robot's orientation in terms of **yaw**, **pitch**, and **roll**. These angles are critical for understanding and controlling the robot's orientation in three-dimensional space.

- **Yaw**: This is the rotation around the vertical axis. For our robot, yaw represents the direction the robot is facing relative to its initial orientation. Accurate yaw measurements are crucial for navigation and ensuring the robot follows the intended path.
- **Pitch**: This is the rotation around the side-to-side axis. Pitch measurements indicate the tilt of the robot forward or backward. In our project, maintaining proper pitch ensures the solar panel remains correctly angled to maximize sunlight capture.
- **Roll**: This is the rotation around the front-to-back axis. Roll measurements reflect the tilt of the robot from side to side. Monitoring roll is important for stability, especially when the robot moves over uneven terrain.

By accurately measuring yaw, pitch, and roll, the Mahony filter enables precise control of the robot’s orientation. This precision enhances the robot's ability to navigate autonomously and improves the overall efficiency and performance of the project.

2. **Fusion of IMU and Encoders Using Extended Kalman Filter (EKF)**

In our autonomous solar panel robot project, we integrated data from the Inertial Measurement Unit (IMU) and encoders using the Extended Kalman Filter (EKF). This fusion was essential for accurately tracking the robot's position and orientation, combining the strengths of both sensors.

The IMU, specifically the MPU9250, provides data on acceleration and angular velocity. It helps determine the robot's orientation and how it's moving in three-dimensional space.

Encoders, like the AS5600, measure the rotation of the robot's wheels. They provide information on how far each wheel has traveled, helping to calculate the robot's position and speed.

Inputs to EKF:

The IMU provides critical information about the robot's motion and orientation, Encoders mounted on the robot's wheels provide essential information about its movement like Wheel Odometry which tracks the rotation of each wheel, including changes in speed and direction.

This data helps in calculating the robot's instantaneous position and velocity, Distance Traveled which computes the distance each wheel has moved, essential for determining the robot's overall position and trajectory over time.

Outputs of EKF:

Position (x, y): The EKF estimates the robot's current coordinates in a defined coordinate system (often Cartesian coordinates).
Velocity: Calculated based on changes in position over time, derived from encoder data.
Orientation (Roll, Pitch, Yaw): Estimates the robot's orientation in three-dimensional space, crucial for navigation and control tasks.

The EKF is a mathematical algorithm that combines noisy measurements from multiple sensors to estimate the true state of a system. Here’s how it works in our project:

1. **Prediction Step**: Using data from the encoders, the EKF predicts the robot's position and orientation over time. It uses the known wheel movements (speed and direction) to estimate where the robot should be.
2. **Update Step**: The EKF then integrates data from the IMU, which provides real-time measurements of acceleration and angular velocity. These measurements refine the predicted position and orientation calculated from the encoders.

Why Use EKF for Fusion?

1. **Handling Sensor Noise**: Both the IMU and encoders can have noisy measurements. The EKF is effective because it uses statistical models to filter out this noise, providing a more accurate estimate of the robot's state.
2. **Integration of Data**: By combining IMU and encoder data, the EKF compensates for each sensor's limitations. For example, the encoders provide precise distance measurements but may drift over time, while the IMU offers real-time orientation but can be affected by acceleration changes.
3. **Improved Position and Orientation:** The EKF's ability to fuse these sensor inputs results in a more reliable calculation of the robot's position, speed, and orientation. This information is crucial for autonomous navigation tasks.

The EKF has numerous benefits, including

- **increased Robustness**: By integrating IMU and encoder data, the robot can maintain accurate position and orientation estimates even in dynamic environments with changes in terrain or speed.
- **Enhanced Navigation Precision**: The fused data from IMU and encoders allows the robot to navigate more precisely, following desired paths and adjusting its movements accordingly.
- **Real-time Performance**: The EKF operates efficiently in real-time, making it suitable for applications where timely updates on the robot's state are critical.

4.3.3 Controller

The main part of any control system and considered the brain of the system.

Firstly, we need to know what a microcontroller and its usage is and how to select a proper one for our project.

What is a microcontroller?

A microcontroller is a compact integrated circuit (ICs)designed to govern a specific operation in an embedded system. It is essentially a small computer on a single chip, consisting of a processor, memory, and input/output peripherals. Microcontrollers are widely used in various applications, from simple devices like remote controls and microwave ovens to complex systems like automotive engines and medical devices and autonomous robots.

Steps to Select a Microcontroller

- **Define Project Requirements**: List all the technical requirements of your project.
- **Research Microcontrollers**: Use resources like datasheets, manufacturer websites, and community forums to research microcontrollers that meet your requirements.
- **Evaluate Options**: Compare the shortlisted microcontrollers based on the factors mentioned above.
- **Prototype and Test**: If possible, get evaluation boards or development kits to prototype and test your project.
- **Final Selection**: Choose the microcontroller that best meets your project's requirements, budget, and availability.

Now, after taking into consideration all the previous issues, we have settled on choosing the Raspberry Pi 4 model B with 8Gb RAM and Arduino mega for some peripherals.

4.3.3.1 Raspberry pi 4 model B

The Raspberry Pi 4 Model B with 8GB of RAM is a powerful single-board computer designed for a wide range of applications, from educational projects to industrial automation and beyond. Here's an overview of its key features and specifications:

Figure 4.3.8: Raspberry Pi 4 model B

The Raspberry Pi 4 Model B (Pi4B) is the first of a new generation of Raspberry Pi computers supporting more RAM and with significantly enhanced CPU, GPU and I/O performance; all within a similar form factor, power envelope and cost as the previous generation Raspberry Pi 3B+.

The Pi4B is available with either 1, 2, 4 and 8 Gigabytes of LPDDR4 SDRAM.

Raspberry Pi 4 Hardware

- Quad core 64-bit ARM-Cortex A72 running at 1.5GHz
- 1, 2, 4 and 8 Gigabyte LPDDR4 RAM options
- H.265 (HEVC) hardware decode (up to 4Kp60)
- H.264 hardware decodes (up to 1080p60)
- VideoCore VI 3D Graphics
- Supports dual HDMI display output up to 4Kp60

Raspberry Pi 4 Software

- ARMv8 Instruction Set
- Mature Linux software stack
- Actively developed and maintained
- Recent Linux kernel support
- Many drivers up streamed
- Stable and well supported userland
- Availability of GPU functions using standard APIs

Raspberry Pi 4 Interfaces

- 802.11 b/g/n/ac Wireless LAN
- Bluetooth 5.0 with BLE
- 1x SD Card
- 2x micro-HDMI ports supporting dual displays up to 4Kp60 resolution
- 2x USB2 ports
- 2x USB3 ports
- 1x Gigabit Ethernet port (supports PoE with add-on PoE HAT)
- 1x Raspberry Pi camera port (2-lane MIPI CSI)
- 1x Raspberry Pi display port (2-lane MIPI DSI)

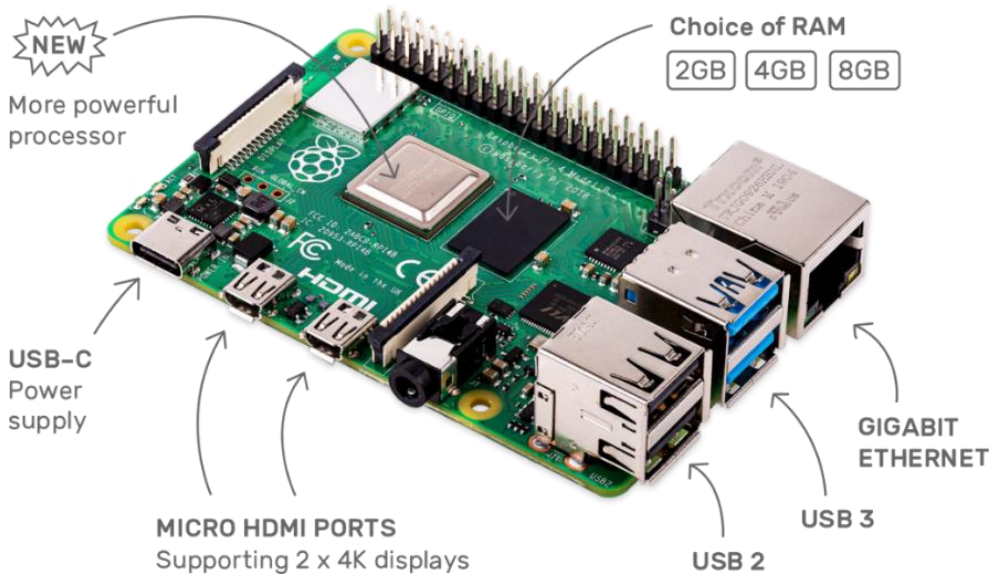


Figure 4.3.9: RPI ports

28x user GPIO supporting various interface options:

- Up to 6x UART
- Up to 6x I2C
- Up to 5x SPI
- 1x SDIO interface
- 1x DPI (Parallel RGB Display)
- 1x PCM
- Up to 2x PWM channels
- Up to 3x GPCLK outputs

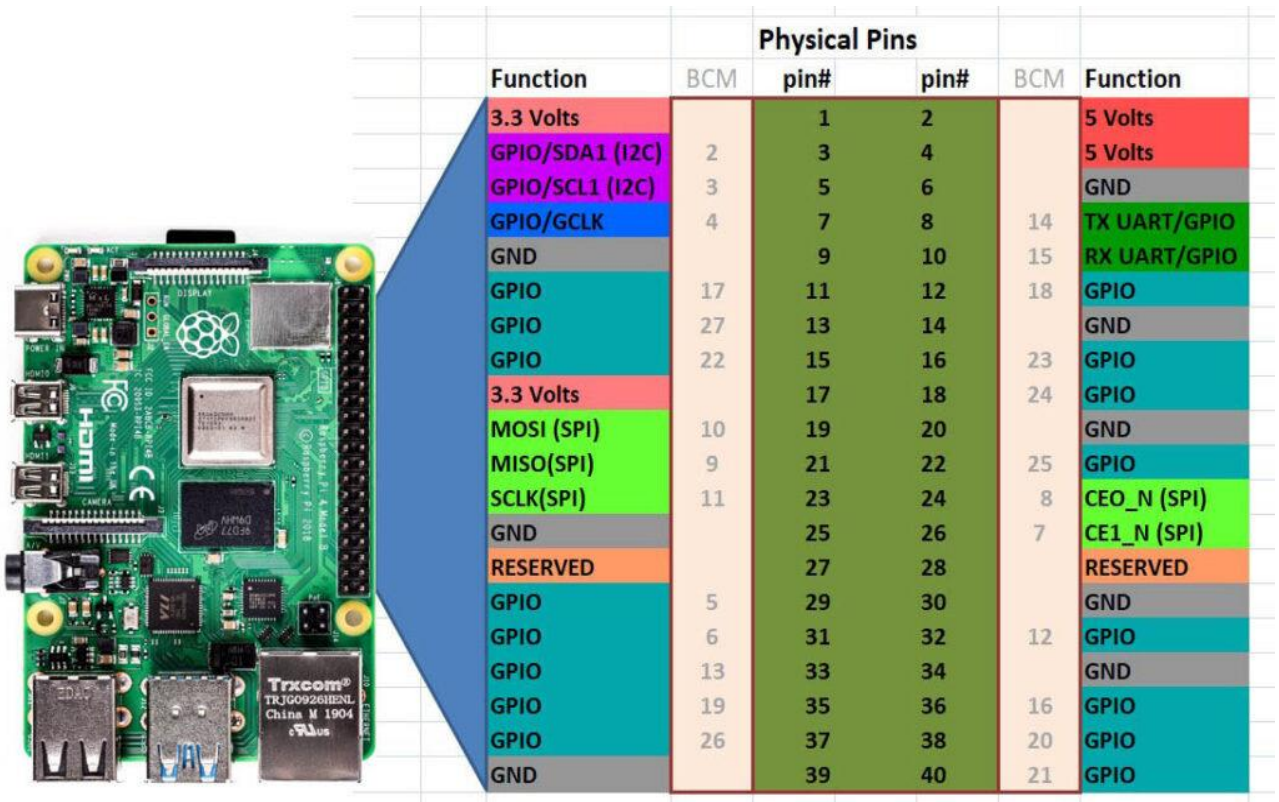


Figure 4.3.10: RPI pinout

Numbering system in raspberry pi 4

The Raspberry Pi 4 uses multiple numbering systems for its GPIO (General Purpose Input/Output) pins. Understanding these numbering systems is essential for correctly interfacing with the GPIO pins in your projects. The main numbering systems are:

1. **Broadcom (BCM) Numbering**:
    - This refers to the numbering of the GPIO pins based on the Broadcom SoC (System on Chip) used in the Raspberry Pi. Each GPIO pin is identified by its Broadcom GPIO number.
    - This numbering system is commonly used in software libraries like RPi.GPIO in Python.

2. **Physical (Board) Numbering**:
    - This refers to the physical pin numbers on the Raspberry Pi's 40-pin GPIO header.
    - Pins are numbered sequentially from 1 to 40, starting from the top left when looking at the GPIO header with the USB ports facing towards you
    - This system is often used in documentation and diagrams to make it easier to locate the physical pin on the board.

Raspberry Pi Power Requirements

The Pi4B requires a good quality USB-C power supply capable of delivering 5V at 3A. If attached downstream USB devices consume less than 500mA, a 5V, 2.5A supply may be used.

Reasons for choosing Raspberry Pi 4 for our project

1. **Performance**: The RPI 4 operates at a maximum CPU frequency of 1.5GHz and provides sufficient processing power to handle the diagnostic tasks effectively. The 64-bit ARM-Cortex A72 ensures efficient execution of instructions, enabling the system to perform measurements, calculations, and control operations swiftly where our project has lot of sensors and peripherals requires a speed and powerful processor.

2. **Memory**: The microcontroller has flexible Flash memory - SD card with 32 GB memory - for storing the program code and 8GB of SRAM for data storage. This memory capacity is adequate for accommodating the diagnostic system's firmware, variables, and data buffers.
3. **Extensive Software Support:** One of the most important reasons for choosing the Raspberry Pi supports a wide range of operating systems and programming languages, including Python, which is commonly used in robotics for its simplicity and extensive libraries. you can upload whatever OS you need to fit your project .in our project we uploaded Linux operating system with Ubuntu distribution in which we can use ROS Packages and libraries that facilitates the control, navigation and interfacing between the modules in our project.
4. **Peripherals:** The RPI offers a rich set of peripherals

- **GPIO Interface:** The Pi4B makes 28 BCM2711 GPIOs available via a standard Raspberry Pi 40-pin header. This header is backwards compatible with all previous Raspberry Pi boards with a 40-way header. As well as being able to be used as straightforward software-controlled input and output (with programmable pulls), GPIO pins can be switched (multiplexed) into various other modes backed by dedicated peripheral blocks such as I2C, UART and SPI. In addition to the standard peripheral options found on legacy Pis, extra I2C, UART and SPI peripherals have been added to the BCM2711 chip and are available as further mux options on the Pi4. This gives users much more flexibility when attaching add-on hardware as compared to older models.

These GPIO pins in our project are attached to the sensors as:

- RPLIDAR
- AS5600 Encoders

One of the challenges we faced with the Encoders is that the AS5600 Encoder we used in our project has only one address and we don't have the ability to modify this address for any one of the twice encoders we used. And we can't use the same address with the same i2c bus.

This problem hampers us from reading the data from the two encoders at the same time. After trying and searching we knew that the RPI allows us to make software i2c with different address that helps us to read both data from the encoders at the same time.

- **Camera and Display Interfaces:** The Pi4B has a 1x Raspberry Pi 2-lane MIPI CSI Camera and 1x Raspberry Pi 2-lane MIPI DSI Display connector. These connectors are backwards compatible with legacy Raspberry Pi boards and support all the available Raspberry Pi camera and display peripherals.
- **HDMI:** The Pi4B has 2x micro-HDMI ports, both of which support CEC and HDMI 2.0 with resolutions up to 4Kp60**.**
- **Audio and Composite (TV Out):** The Pi4B supports near-CD-quality analogue audio output and composite TV-output via a 4-ring TRS 'A/V' jack. The analog audio output can drive 32 Ohm headphones directly.
- **USB:** The Pi4B has 2x USB2 and 2x USB3 type-A sockets. Downstream USB current is limited to approximately 1.1A in aggregate over the four sockets.

So, the raspberry pi is a Versatile Connectivity board that comes with multiple connectivity options including GPIO pins, USB ports, HDMI, Wi-Fi, and Bluetooth, allowing easy integration with various sensors, actuators, and communication modules.

5. **Compact Size:** The small form factor of the Raspberry Pi allows it to be easily integrated into compact and mobile robotic platforms.
6. **Flexibility :**The versatility of the Raspberry Pi allows it to be used in various roles within a robot, such as the main control unit, a vision processing node, or an interface for other microcontrollers as in our case we interfaced the RPI with Arduino mega.
7. **Energy Efficiency:** Raspberry Pi boards are energy-efficient, which is crucial for battery-operated autonomous robots that need to operate for extended periods.
8. **Affordability:** Raspberry Pi boards are cost-effective, making them an attractive option for hobbyists, students, and researchers working on budget-constrained projects.

These are the main reasons and factors that led to choosing the Raspberry Pi as the main controller for this project.

Our project includes a lot of sensors and actuators and has a lot of tasks working together. We don't want to overload the Raspberry Pi so it can operate more efficiently, and the number of pins is not enough for all these sensors.

Therefore, we will use the Arduino Mega alongside the Raspberry Pi to provide more pins and relieve the load on the Raspberry Pi.

The two controllers will interface through serial communication to exchange data. Let's learn a bit about the Arduino Mega, the peripherals we will connect to it in our project, and how the interfacing between it and the Raspberry Pi will take place.

4.3.3.2 Arduino Mega

The Arduino Mega 2560 is a popular open-source microcontroller board designed for complex projects that require more I/O pins and memory than standard Arduino boards. It is widely used in a variety of applications, including robotics, automation, and prototyping.

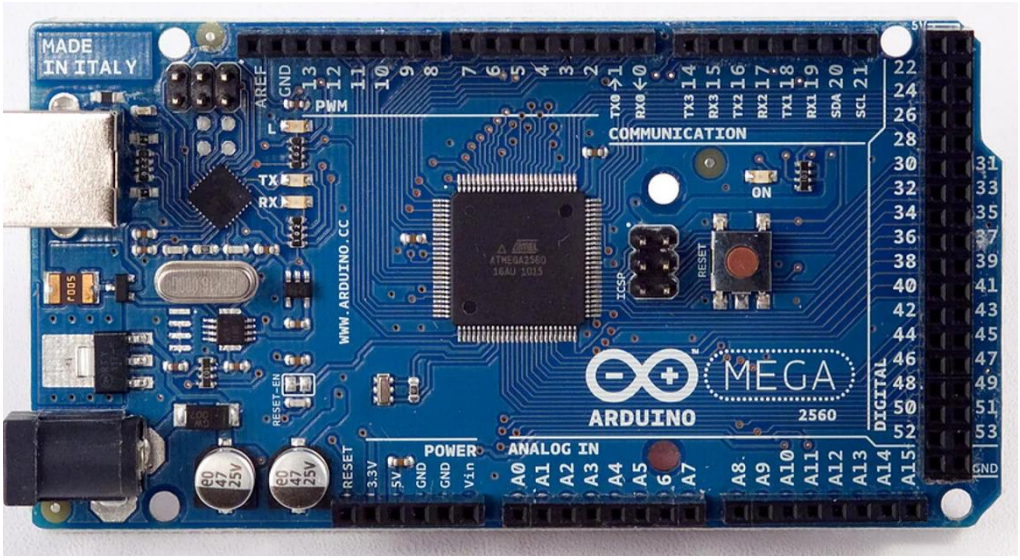


Figure 4.3.11: Arduino mega 2560

Specifications:

- **Operating Voltage:** 5V
- **Input Voltage (recommended)**: 7-12V
- **Input Voltage (limits)**: 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 (8 KB used by bootloader)
- **SRAM**: 8 KB
- **EEPROM**: 4 KB
- **Clock Speed**: 16 MHz
- **Communication**:
  - **Serial Ports (UART)**: 4 (hardware serial ports)
  - **I2C**: Yes (SDA and SCL pins)
  - **SPI**: Yes (on the ICSP header)
- **Other Features**:
  - **USB Connection**: for programming and communication
  - **Power Jack**: for external power supply
  - **Reset Button**

Why Choose Arduino Mega?

- Large Number of I/O Pins: Ideal for projects requiring multiple sensors, actuators, and interfaces.
- Memory Capacity: The 256 KB of flash memory and 8 KB of SRAM allow for larger programs and more complex tasks.
- Multiple Serial Ports: The four hardware UARTs facilitate communication with multiple devices simultaneously.
- Compatibility: Supports numerous shields and accessories available in the Arduino ecosystem.
- Ease of Use: The Arduino IDE is user-friendly, making it accessible for beginners and professionals alike.
- Community and Resources: Extensive community support, tutorials, and documentation are available.

In our project we use Arduino with:

- Motors
- Ultrasonic Sensors (HC-SR04)
- Relays
- IMU (MPU 9250)

Using the Arduino Mega alongside the Raspberry Pi allows you to leverage the strengths of both platforms, providing a robust solution for complex projects.

To exchange the data of these peripherals between the Raspberry pi and the Arduino there are a several methods. in our project we use the serial communication via UART communication protocol.

As our project is complicated and have a lot of modules so the Raspberry pi is subject to damage due to heat and high temperatures and any error in connection.so we must make protection circuits and keep the RPI in safe zone.

4.3.3.3 Raspberry Pi and Arduino Protection

1. **Enclosure/Case**:
   - **Use a Protective Case**: A good quality case can protect the Raspberry Pi from physical damage, dust, and static electricity. Cases are available in various materials, including plastic, aluminum, and acrylic.
   - **Consider a Case with Built-in Cooling**: Some cases come with built-in fans or heat sinks to help dissipate heat, which can prolong the life of your Raspberry Pi.

Figure 4.3.12: RPI Case

2. **Heat Management**:
   - **Heat Sinks**: Attach heat sinks to the CPU and other heat-generating components to help dissipate heat.
   - **Fans**: Use a small fan to increase airflow and keep the Raspberry Pi cool.
   - **Proper Ventilation**: Ensure that the Raspberry Pi is placed in a well-ventilated area to avoid overheating.

Figure 4.14 RPI Heat Sinks

3. **Static Electricity**:
   - **Handle with Care**: Always handle the Raspberry Pi by its edges and avoid touching the electronic components to prevent static discharge.
   - **Anti-Static Measures**: Use an anti-static mat or wrist strap when working on the Raspberry Pi to minimize the risk of static discharge.

4. **Environmental Protection**:
   - **Avoid Moisture and Dust**: Keep the Raspberry Pi away from moisture and excessive dust, which can cause corrosion or short circuits.
   - **Temperature Control**: Ensure the operating environment is within the recommended temperature range for the Raspberry Pi.

5. **Software Protection**:
   - **Regular Updates**: Keep the operating system and software up to date to protect against security vulnerabilities.
   - **Backup Data**: Regularly back up your data to prevent data loss in case of software corruption or hardware failure.
   - **Use Antivirus**: Consider using antivirus software to protect against malware.

6. **Safe Shutdown**:
   - **Proper Shutdown Procedures**: Always shut down the Raspberry Pi properly using the operating system's shutdown command to avoid corrupting the SD card.
   - **Uninterruptible Power Supply (UPS)**: Consider using a UPS to protect against unexpected power outages.

7. **Physical Security**:
   - **Secure Mounting**: If your Raspberry Pi is part of a larger project, ensure it is securely mounted to avoid accidental drops or disconnections.
   - **Locking Mechanism**: Use a case with a locking mechanism if the Raspberry Pi is in a public or shared space to prevent tampering.

By implementing these protections and best practices, you can safeguard your Raspberry Pi and Arduino devices, ensuring their longevity and reliability in various projects and applications.

As our project includes a lot of tasks working together, we need a control framework to handle all these tasks simultaneously and efficiently. Here comes the magic of ROS.

4.3.3.4 ROS (Robot Operating System)

ROS is an open-source framework designed to facilitate the development of robotic software. Despite its name, it is not an actual operating system; instead, it provides a set of tools, libraries, and conventions that simplify the process of creating complex and robust robot applications.

ROS runs on top of a traditional operating system (like Linux) and abstracts the hardware complexities, allowing developers to focus on higher-level functionalities such as perception, navigation, and control.

Features and Benefits of ROS:

- ROS promotes a modular approach to software design, where functionalities are divided into independent nodes. Nodes can be reused across different projects, accelerating development and reducing redundancy.
- It provides a hardware abstraction layer (HAL) that simplifies interfacing with various sensors, actuators, and robotic platforms. This abstraction allows developers to write code that is hardware-agnostic, making it easier to switch between different robot setups.
- ROS uses a publish-subscribe messaging system for communication between nodes. Nodes communicate by publishing messages to topics, allowing for asynchronous and decentralized data exchange. This design enhances scalability and fault tolerance in robotic systems.
- ROS includes powerful tools such as RViz for 3D visualization and Gazebo for robot simulation. These tools enable developers to visualize robot states, sensor data, and simulated environments, facilitating algorithm testing and validation.
- With a large and active community, ROS benefits from extensive community-contributed packages, libraries, and resources. This collaborative ecosystem supports knowledge sharing, best practices, and rapid advancement in robotic software development.

Usage of ROS:

ROS is widely used in academia for robotics research and education. It provides a flexible platform for experimenting with different algorithms and hardware configurations.

In industry, ROS facilitates the development of robotic systems for manufacturing, logistics, and automation. Its modular architecture and robust communication framework are well-suited for complex industrial environments.

ROS has been adopted by space agencies and defense organizations for tasks such as planetary exploration, satellite servicing, and unmanned aerial vehicle (UAV) operations.

Architecture of ROS:

ROS follows a distributed architecture that enables modular and scalable robotic software development. Understanding its architecture involves examining how various components interact and collaborate within a ROS-based system

1. **Nodes:**

   Nodes are fundamental units of computation in ROS. Each node performs a specific task, such as processing sensor data, controlling actuators, or executing algorithms.

   Nodes communicate with each other by publishing and subscribing to messages on topics or by using services. This communication follows a decentralized and asynchronous model, where nodes can operate independently yet collaborate effectively.

2. **Master:**

   The ROS Master is a centralized entity within a ROS network that manages the registration and discovery of nodes, topics, services, and the parameter server.

   It facilitates communication between nodes by providing information about available topics, services, and their respective publishers and subscribers. Nodes register with the Master upon startup to announce their capabilities and listen for announcements from other nodes.

3. **Topics:**

   Topics are named buses over which nodes exchange messages. They enable nodes to share data without needing to know the identity of the communicating nodes.

   Nodes can publish messages to topics to broadcast data, and other nodes can subscribe to topics to receive the published data. This model supports asynchronous communication, where nodes operate independently and consume data as needed.

4. **Messages:**

   Messages define the structure and content of data exchanged between nodes via topics or services. ROS provides predefined message types for common data formats (e.g., sensor readings, control commands) and allows users to define custom message types using the ROS message description language.

5. **Services:**

   Services enable nodes to perform remote procedure calls (RPCs) in a synchronous manner. A node can request a service from another node, which processes the request and sends back a response.

   Services are used for tasks that require direct interaction and immediate feedback between nodes, such as configuration updates, diagnostic checks, or complex calculations.

6. **Parameter Server:**

   Parameter Server is a shared, distributed storage system in ROS for managing configuration parameters.

   Nodes can store, retrieve, and update parameters dynamically during runtime. Parameters can include sensor calibration values, control parameters, and algorithmic constants, providing a centralized mechanism for parameter management across the ROS network.

4.3.4 Output Signal Conditioning and Interfacing

Output signal conditioning and interfacing involve preparing and transmitting control signals from the digital control system to the actuators. This step ensures that signals are in the correct form and strength to drive the actuators effectively. The main components involved in this stage are:

- **D/A, D/D Converters**: Digital-to-Analog (D/A) and Digital-to-Digital (D/D) converters translate digital control signals into analog or different digital formats required by the actuators.
- **PWM (Pulse Width Modulation)**: PWM is used to control the power supplied to actuators, especially motors, by varying the width of the pulses in a pulse train.
- **Power Transistors**: These act as switches or amplifiers to control the high power needed by actuators based on low-power control signals.
- **Power Amplifiers**: These devices amplify the control signals to the levels required by the actuators, ensuring they operate efficiently and effectively.

4.3.4.1 Motor Drivers

Motor drivers are essential components in mechatronics and robotics systems, serving as intermediaries between control systems (such as microcontrollers or microprocessors) and motors.

1. Chassis Motors' driver

For chassis motors, we should have used the Cytron MD10C R3 motor driver, but this IC is not available in the Egyptian Market and because of it is high cost out of Egypt we couldn't buy it and settled to use **BTS7960** 43A Motor driver. We controlled these motors by the BTS7960 43A Motor driver with Arduino.

This is a high-power H-bridge motor driver module that can control DC motors with a continuous current up to 43A and peak current up to 100A. It is widely

used in robotics, automation, and electric vehicle applications due to its high current handling capability, reliability, and ease of use.

Figure 4.3.13: BTS 7960

And because of a problem in volt that happened suddenly this driver was burned and then we used another driver which is **IR2104**, This IC is like the IR2184 but with small differences. The IR2104(S) are high voltage, high-speed power MOSFET and IGBT drivers with dependent high and low side referenced output channels. Turn-on rise time and Turn-off fall time for IR2184 are smaller than that of IR2104, so the switching speed of the Original Cytron IC is higher.

This collection (2× 24V Motors – 12V IR2104 Driver for each motor) controls the motion operation of our robot.

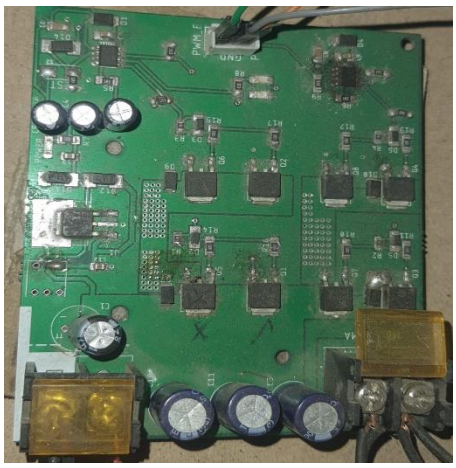


Figure 4.3.14: IR2104 Motor driver

2. Arm Motors Drivers:

We control arm linear motors by 30V electrical relay not by using a motor driver because we are only concerned to control the motor direction not the speed.

Relay

Electrical relays are versatile and essential components in controlling high-power circuits with low-power signals. They provide electrical isolation, high current handling, and simple control, making them suitable for various applications, including automotive systems, industrial automation, home appliances, communication systems, and power systems. Despite their mechanical wear and slower switching speeds compared to solid-state devices, their advantages make them a valuable choice in many scenarios, especially in ours.

Figure 4.3.15: Electrical array

3. Brush motor Driver:

For brush motor the BTS driver we discussed before is very sufficient for it.

4.3.5 User Interface

The user interface (UI) serves as the point of interaction between the user and the system, allowing users to input commands and receive feedback. A well-designed UI ensures that users can effectively control the system and monitor its status.

For this part we have the mobile application discussed in chapter 8, with which we can remotely monitor and control the robot.

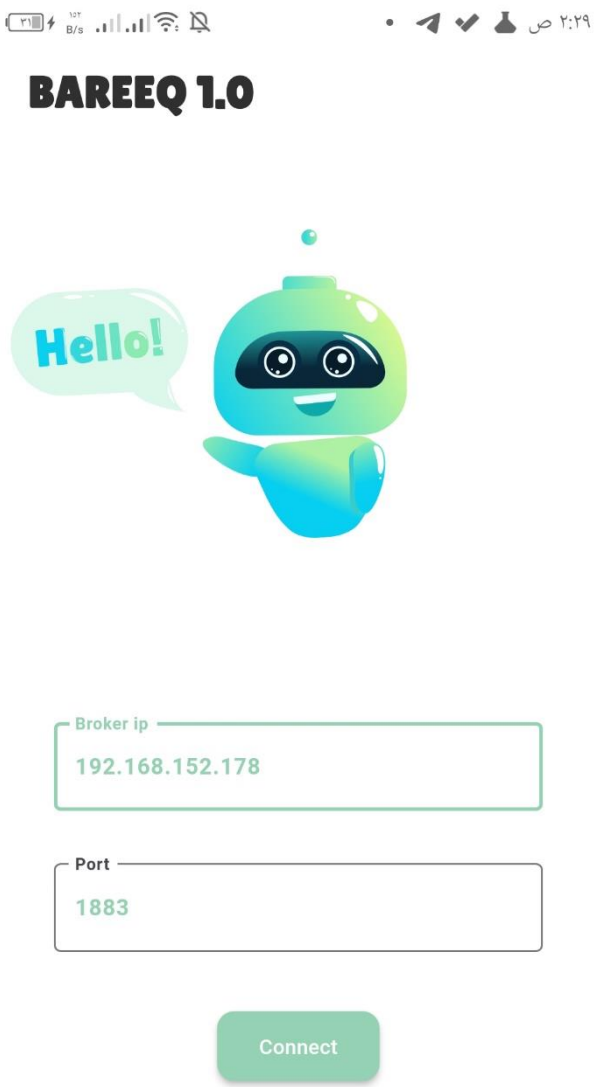


Figure 4.3.17: mobile application 1

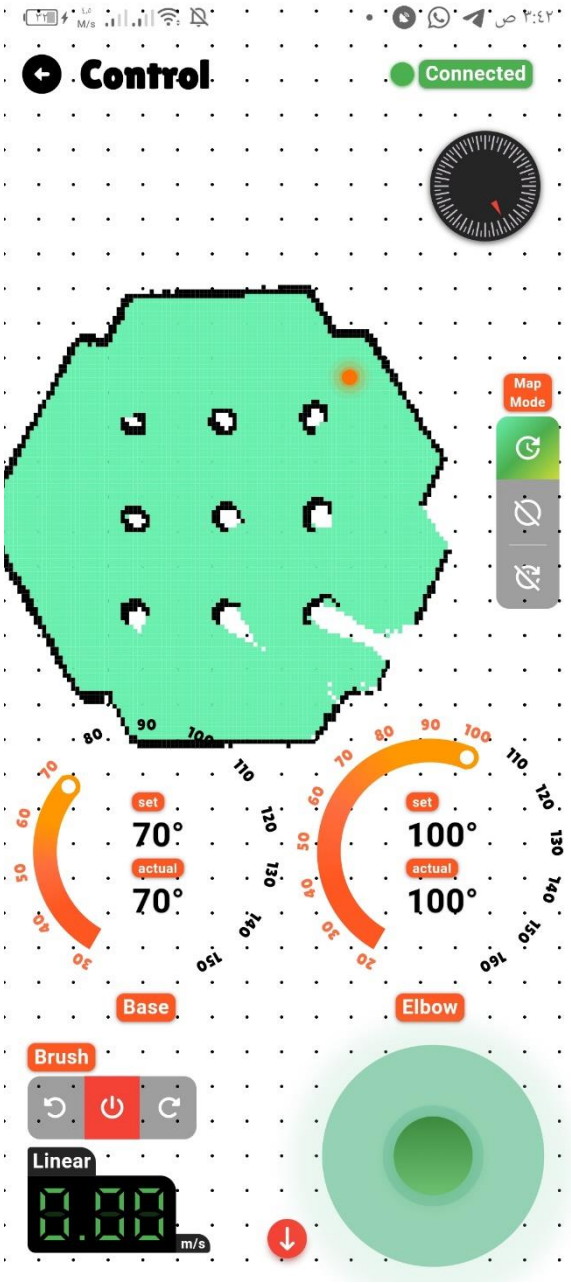


Figure 4.3.16:mobile application 2

Chapter 5

NAVIGATION CONTROL

5.1 INTRODUCTION

Navigation is the process of determining the position and orientation of an object or vehicle in each environment and planning a path to move from one location to another.

Navigation is an essential aspect of robotics, as it enables robots to move autonomously and interact with the environment effectively. Navigation in robotics involves several components, including perception, localization, path planning, and motion control.

For this level we have two approaches:

1. **Semi-Autonomous Navigation Control**
2. **Fully Autonomous Navigation Control**

We will discuss each of them in detail but let's first understand the concept of slam.

5.2 SLAM:

Simultaneous Localization and Mapping (SLAM) is a pivotal technology in the field of robotics and autonomous systems. It addresses the fundamental challenge of enabling a robot to navigate and understand an unknown environment without relying on pre-existing maps or external localization systems. By concurrently building a map of its surroundings and determining its position within that map, SLAM equips robots and autonomous vehicles with the capability to operate independently and adaptively in dynamic and unstructured environments.

The origins of SLAM trace back to early attempts at robotic navigation, where the need for robots to autonomously explore and interact with their surroundings became increasingly apparent. Traditional navigation methods, which relied heavily on pre-defined maps and external references, proved inadequate for real-world applications that demanded flexibility and real-time adaptability. SLAM emerged as a solution to these limitations, offering a method for robots to simultaneously chart their environment and pinpoint their location within it.

SLAM is not a single algorithm but a suite of techniques that work together to solve the localization and mapping problem. At its core, SLAM involves the integration of various sensors, such as cameras, lidar, and inertial measurement units, to gather data about the environment. This data is then processed using

sophisticated algorithms to create a map while continuously updating the robot's position and orientation.

5.2.1 Mapping:

Mapping is the process of creating a detailed representation of the environment using sensor data.

The problem of representing the environment in which the robot moves is a dual of the problem of representing the robot's possible position or positions. Decisions made regarding environmental representation can have an impact on the choices available for robot position representation. Often the fidelity of the position representation is bounded by the fidelity of the map.

Three fundamental relationships must be understood when choosing a particular map representation:

1. The precision of the map must appropriately match the precision with which the robot needs to achieve its goals.
2. The precision of the map and the type of features represented must match the precision and data types returned by the robot's sensors.
3. The complexity of the map representation has a direct impact on the computational complexity of reasoning about mapping, localization, and navigation.

Mapping algorithms:

5.2.1.1 GMapping:

GMapping algorithm is one of the solutions to SLAM problem which is based on particle filter. In using particle filter for SLAM, the pose of the robot and its uncertainty are approximated by a set of particles.

There are many variants of particle filter such. However, in GMapping, the improved Rao-Blackwell zed Particle Filter (RBPF) is implemented.

To better understand how GMapping is implemented in ROS, **Figure 5.1** gives the flowchart of how GMapping in ROS works. When a scan comes, GMapping checks whether there are an odometry pose that also comes with the scan or not

by looking up transform from “odom frame” to “base frame” in tf messages assuming that the LIDAR sensor is positioned at “base frame”. This is done by using message filters class. Note that this also means every odometry data which comes without scans is discarded. Also, the name of “odom frame” and “base frame” can be set as needed.

After timestamps checking is passed, the next procedure is motion filter. In GMapping, a motion filter is used to determine whether the information from LIDAR sensor needs to be processed or not. If there is enough linear movement or angular movement with respect to the previous odometry pose or if the timestamp of the recent data is old enough with respect to the previous timestamp, the information from LIDAR sensor will be processed. Then, after motion filter, the improved RBPF from is performed which includes the step from drawing a new pose for each particle i and adaptive resampling which depends on the number of effective particles.

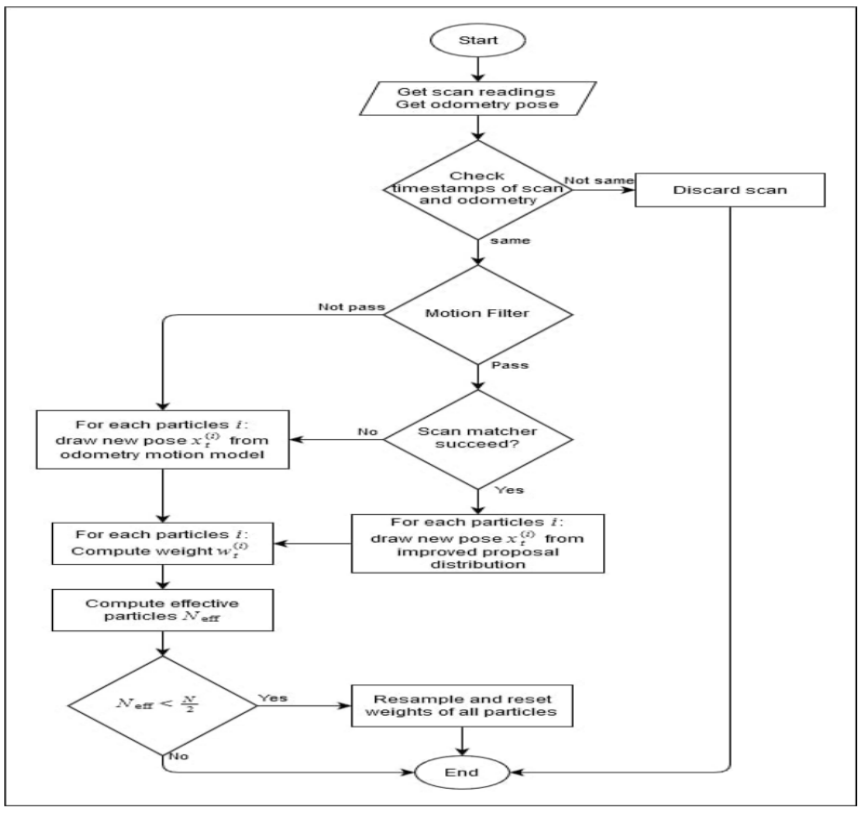


Figure 5.2.1: GMapping algorithm flow chart

Rao-Blackwell zed Particle Filter:

Rao-Blackwell zed particle filter relies on the factorization of the SLAM problem, that is to estimate the joint posterior $p(m,\ x_{1:t} \mid z_{1:t}\ ,\ u_{1:t-1})$ where m is the map, $x_{1:t}$ is the pose of the robot, $z_{1:t}$ is the measurements, $u_{1:t-1}$ is the actions performed by the robot. The factorization separates the estimation of the trajectory and the computation of the map given that trajectory

$$p(m,\ x_{1:t} \mid z_{1:t}\ ,\ u_{1:t-1}) = \ p(m_i \mid z_{1:t}\ ,\ x_{1:t}) \cdot \ p(\ x_{1:t} \mid z_{1:t}\ ,\ u_{1:t-1})$$

This factorization enables us to estimate the trajectory first then compute the map given the trajectory. The computation of the map has been discussed in Section 5.2.1.2, while the estimation of the trajectory is done using particle filter. To estimate the posterior $p(m,\ x_{1:t} \mid z_{1:t}\ ,\ u_{1:t-1})$, one of the particle filter algorithms which is quite common is the sampling importance resampling (SIR) filter. This SIR filter can be described as follows:

1. **Sampling:** The new set of particles $\{x_t^{(i)}\}$ are generated from the previous set of particles $\{x_{t-1}^{(i)}\}$ by sampling from a proposal distribution π. Usually, this proposal distribution π is an odometry motion model.
2. **Importance Weighting:** Each particle from the new set are assigned with a weight $w_t^{(i)}$. This weight can be computed as follows.

$$w_t^{(i)} \ = \ \frac{p\left(x_{1:t}^{(i)} \middle|\ z_{1:t},\ u_{1:t-1}\right)}{\pi(x_{1:t}^{(i)} \mid\ z_{1:t},\ u_{1:t-1})}$$

The weights consider the fact that the proposal distribution is in general not equal to the target distribution.

3. **Resampling:** Because a finite number of particles are used to approximate a continuous distribution, the particles with less importance weight get replaced by the particles with greater importance weight. Then, after resampling, all particles are assigned with the same weight. To compute efficiently the weights of each partiles, a recursive computation is formulated by assuming that the proposal distribution can be written in a recursive form as:

$$\pi\left(x_{1:t}^{(i)} \mid z_{1:t}\,,\, u_{1:t-1}\right)$$
$$= \pi\left(x_t^{(i)} \middle| x_{1:t-1}^{(i)}\,,\, z_{1:t}\,,\, u_{1:t-1}\right) \cdot \pi\left(x_{1:t-1}^{(i)} \middle| z_{1:t-1} - 1,\, u_{1:t-2}\right)$$

Also, the target distribution $p\left(x_{1:t}^{(i)} \mid z_{1:t}\,,\, u_{1:t-1}\right)$ is written in a recursive form as:

$$p\left(x_{1:t}^{(i)} \mid z_{1:t}\,,\, u_{1:t-1}\right) =$$
$$\eta p\left(z_t \middle| x_{1:t}^{(i)}\,,\, z_{1:t-1}\right).p\left(x_t^{(i)} \middle| x_{t-1}^{(i)}\,,\, u_{t-1}\right) \cdot$$
$$p(x_{1:t-1}^{(i)} \mid z_{1:t} - 1,\, u_{1:t-2})$$

Where $\eta = \frac{1}{p(\, z_t \mid x_{1:t-1},\, z_{1:t-1},\, u_{1:t-1})}$.

Then, the recursive computation of the weights is obtained by putting Equation.

$$w_t^{(i)} = \frac{\eta p\left(z_t \middle| x_{1:t}^{(i)}\,,\, z_{1:t-1}\right) \cdot p\left(x_t^{(i)} \middle| x_{t-1}^{(i)}\; u_{t-1}\right)}{\pi\left(x_t^{(i)} \middle| x_{1:t-1}^{(i)},\, z_{1:t}\,,\, u_{1:t-1}\right)} \cdot w_{t-1}^{(i)}$$

Now that the recursive equation has been formulated, the improved proposal distribution and when to perform the resampling step are discussed in the following sections.

Advantages:

- Efficient Particle Filter Implementation: Gmapping uses an efficient implementation of the particle filter, which allows it to handle many particles effectively. This leads to better accuracy in mapping and localization
- Compatibility with Different Sensors: Gmapping is compatible with various sensor types, making it versatile for different robotic platforms and environments.

- Adaptive Resampling: The algorithm employs adaptive resampling to maintain a diverse set of particles, improving the robustness of the SLAM process.
- Incremental Map Building: The algorithm incrementally builds the map as the robot explores the environment, providing real-time updates and adjustments.
- Effective in Cluttered Environments: Gmapping performs well in cluttered indoor environments, maintaining accuracy despite obstacles and dynamic elements.

Disadvantages

- Computational Complexity: Despite its efficiency, GMapping can still be computationally intensive, especially in large environments or when using a high number of particles.

- Dependency on Accurate Odometry: The algorithm's performance heavily relies on accurate odometry data. Any errors in odometry can significantly affect the quality of the generated map.
- Loop Closure Challenges: GMapping can face difficulties in detecting and closing loops, which is crucial for correcting accumulated errors over long trajectories.
- Limited Scalability: The performance of GMapping may degrade in larger environments, making it less suitable for extensive mapping tasks without sufficient computational resources

5.2.1.2 Google Cartographer:

Google Cartographer is a 2D SLAM algorithm that uses a graph-based approach instead of particle filters. It partitions maps into submaps, each containing several laser scans, and rasterizes them to create the final map. The system, implemented in ROS, uses both local and global approaches for SLAM.

The local approach matches the most recent scan against the current submap using non-linear optimization, while the global approach reduces drift error by matching scans against all submaps except the most recent one. Cartographer

employs a pose extrapolator to estimate the initial pose using odometry data and previous scan poses, enabling processing of scans without synchronized timestamps.

Pose extrapolation also unwarps scans to correct robot motion. Voxel and adaptive voxel filters down sample the measurements to reduce computational demand.

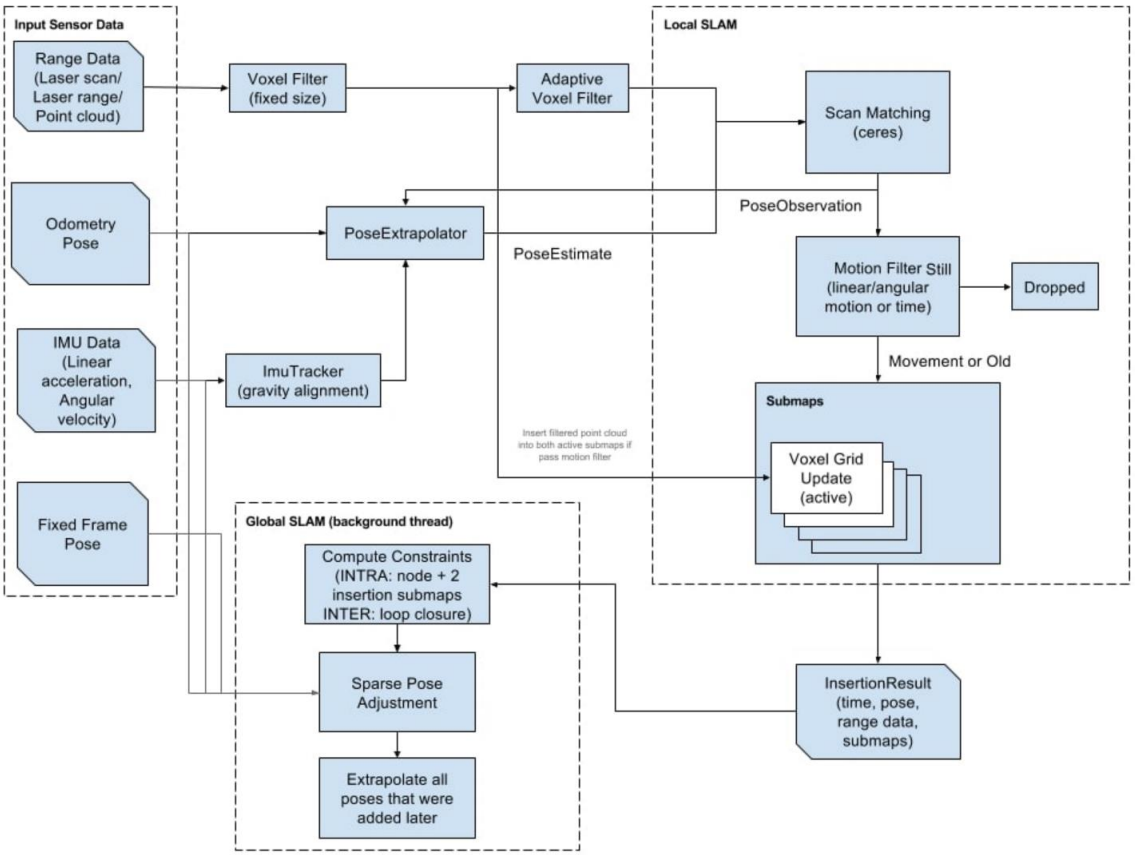


Figure 5.2.2: System Overview of Google Cartographer

The local SLAM system uses this preprocessed data to determine the current pose and submap. A motion filter ensures only significant or old enough scans are inserted into submaps.

Submaps have overlapping sections to maintain continuity, creating inter-submap constraints. The global approach performs pose-graph optimization, considering both inter-submap and intra-submap constraints, which are identified by the global scan matcher. Only a subset of scan nodes is matched to manage computational load, expanding the search window if no matches are found over time.

Advantages

- High Accuracy: Cartographer offers high accuracy in mapping and localization by using a combination of local and global optimization techniques.
- Real-Time Processing: The algorithm is capable of real-time processing, making it suitable for dynamic environments where quick updates are necessary.
- Multi-Sensor Fusion: Cartographer can integrate data from various sensors, such as LiDAR, IMU, and odometry, to enhance the robustness and accuracy of SLAM.
- Scalability: It scales well with both 2D and 3D environments, making it versatile for different types of mapping tasks.
- Efficient Loop Closure: The algorithm efficiently detects and handles loop closures, which helps in reducing accumulated drift and improving map consistency.
- ROS Integration: Cartographer is well-integrated with ROS, providing a robust framework for robotic applications and ease of use within the ROS ecosystem .

Disadvantages

- Computational Demands: The algorithm can be computationally intensive, requiring significant processing power and memory, which might be a limitation for low-powered devices.
- Complexity: The setup and tuning of Cartographer can be complex, requiring a good understanding of its parameters and configuration.
- Resource Intensive: It requires a lot of sensor data and resources to function optimally, which might not be feasible for all types of robots or applications.
- Sensitivity to Sensor Quality: The performance of Cartographer heavily relies on the quality and calibration of the sensors used. Poor sensor data can significantly affect mapping accuracy.

5.2.1.3 Hector SLAM:

As the paper stated, this system serves the local SLAM only or SLAM frontend only. Hence, it does not have any loop closure mechanism. Another feature that should be noted is this system relies only on range data. Information coming from odometry is not used at all. Also, although it is mentioned that the system provides 6-DOF pose estimate, the one implemented in ROS provides only 3-DOF pose estimate. The 3D state estimation is not implemented and what remains is the scan matching part.

In performing scan matching, Hector SLAM employs coarse-to-fine strategy, with manually adjustable number of resolution level. To be efficient, it maintains the occupancy grid maps on all resolutions. The scan will be inserted on all resolutions, making it different to the multi resolution employed in SLAM Toolbox whose cell in coarse resolution map contains the maximum occupancy of the finer cells inside. As for scan matching, it is performed on the coarsest level and the pose from the coarser level is used as initial estimate for matching in finer resolution. After obtaining the pose estimate from the finest resolution, the scan is inserted into the occupancy grid map at all levels. Prior inserting a scan, motion filtering is performed. If the new scan has moved far enough with respect to the pose of the last inserted scan, this new scan is inserted. The insertion is done on all resolutions of occupancy grid maps by using the pose estimate from scan matching at finest resolution.

Advantages

- Low Power Consumption: Efficient for small robots and systems with limited power resources.
- High-Frequency Data Processing: Capable of handling high update rates from laser sensors, improving accuracy in dynamic environments.

Disadvantages

- Dependency on Laser Data: Performance may degrade in environments where laser data quality is compromised.
- Sensor Noise: Requires robust sensor fusion techniques to mitigate noise effects on speed and position accuracy.

Hector SLAM is particularly effective for indoor environments where precise laser scanning is feasible, making it a valuable tool in the field of robotics and autonomous navigation.

a) Why GMapping:

- **Computational Efficiency**: GMapping is generally less computationally intensive compared to Cartographer. This makes it more suitable for platforms with limited processing power and memory.
- **Sensor Requirements**: GMapping primarily relies on laser scans and odometry, whereas Cartographer integrates multiple sensors like IMU, odometry, and LiDAR, which can be resource intensive.
- **Complexity and Ease of Use**: GMapping is simpler to set up and tune compared to Cartographer, which requires a more detailed understanding of its parameters and configuration.
- **Robustness in Outdoor Environments**: While Hector SLAM is effective indoors with high-frequency LiDAR data, it can be less robust in outdoor environments due to its lack of reliance on odometry. GMapping's use of odometry helps maintain better performance in varied terrains and larger outdoor spaces.
- **Handling Noisy Data**: GMapping is relatively tolerant to noisy sensor data compared to Cartographer, which heavily depends on high-quality sensor inputs for accurate mapping and localization.

5.2.2 Localization:

The localization problem is of utmost importance in the real world as this gives us a probabilistic estimate of the robot's current position and orientation. So, it is obvious that without this knowledge, the robot won't be able to take effective decisions and take sound actions if it doesn't know where it is located in the world. There are 3 different types of localization problems.

- **Local Localization:** This is the easiest localization problem. It is also known as position tracking. In this problem, the robot knows its initial pose and the localization challenge entails estimating the robot's pose as it moves out in the environment. This problem is not trivial as there is always some uncertainty in robot motion. However, the uncertainty is limited to regions surrounding the robot.
- **Global Localization:** This is a more complicated localization problem. In this case, the robot's initial pose is unknown, and the robot must determine its pose relative to the ground truth map. The amount of uncertainty is much higher.

- **The kidnapped robot problem:** This is the most challenging localization problem. This is just like the global localization problem, except that the robot may be kidnapped at any time and moved to a new location on the map.

Localization algorithms:

For the localization problem, a wide range of algorithms are available ranging from Monte Carlo Localization, Extended Kalman Filter to Markov and finally Grid Localization. The Monte Carlo Localization algorithm or MCL, is the most popular localization algorithm in robotics. After MCL is deployed, the robot will be navigating inside its known map and collect sensory information using RGB camera and range-finder sensors. MCL will use these sensor measurements to keep track of the robot's pose. MCL is often referred to as Particle Filter Localization, since it uses particles to localize the robot. These particles are virtual elements that resemble robot. Each particle has a position and orientation and it represents a guess where the robot might be located. These particles are re-sampled each time the robot moves and senses its environment. For this project, a modified version of this algorithm known as 'Adaptive Monte Carlo Localization' was used because this modified algorithm dynamically adjusts the number of particles over a period, as the robot navigates around the map, hence making the process more cost effective.

5.2.2.1 Kalman Filters:

The Kalman filter is an estimation algorithm that is very prominent in controls. It is used to estimate the value of variables in real time as the data is being collected. This variable can represent the position or velocity of the robot, or even the temperature of a process. The reason that the Kalman filter is so net worth is because it can take data with a lot of uncertainty or noise in the measurement and provide a very accurate estimate of the real values in a very short time. Unlike other estimation algorithms, it doesn't depend on a lot of data to calculate an accurate estimate. The Kalman filter was invented during the Apollo program.

It was used to help Apollo enter the orbit of the moon. Since its success with the Apollo program, Kalman filter has become one of the most practical algorithms in the field of control engineering.

The Kalman filter works cyclically between two steps. The Kalman filter produces an estimate of the state of the system as an average of the system's

predicted state and of the new measurement using a weighted average. The purpose of the weights is that values with better (i.e., smaller) estimated uncertainty are" trusted" more. The weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction of the system's state.

The result of the weighted average is a new state estimate that lies between the predicted and measured state and has a better estimated uncertainty than either alone. This process is repeated at every time step, with the new estimate and its covariance informing the prediction used in the following iteration. This means that the Kalman filter works recursively and requires only the last" best guess", rather than the entire history, of a system's state to calculate a new state.

The linear Kalman filter assumes that the output is proportional to the input and hence it can be only applied to linear systems. This limitation is overcome with Extended Kalman filters, which can be applied to non-linear systems, which is more applicable in robotics as real-world systems are more often non-linear than linear. Also, Linear Kalman filter assumes that both the prior and the posterior follows a unimodal Gaussian distribution. But in the real world, that is seldom the case. Extended Kalman filters overcome this problem by linear approximation of the posterior distribution after a non-linear transformation.

5.2.2.2 Particle Filters:

The Monte Carlo Localization or Particle Filters uses virtual particles to estimate a robot's pose. With MCL, particles are initially spread uniformly and randomly throughout the entire map. Just like the robot, each particle has an xy coordinate and an orientation vector. So, each of these particles represent the hypothesis of where the robot might be.

In addition to the 3d vector, particles are assigned a weight. The weight of a particle is the difference between the robot's actual pose and the particle's predicted pose. The importance of a particle is dependent on its weight. The bigger the particle, the more accurate it is. Particles with larger weights are more likely to survive during the re-sampling process.

After the re-sampling process, particles with significant weights are more likely to survive whereas others are more likely to die. Finally, after several iterations of the algorithm and after different stages of re-sampling, particles will converge and estimate the robot's pose. The MCL algorithm estimates the posterior distribution of a robot's position and orientation based on sensory information.

This process is known as Bayes Filter. Using a Bayes filtering approach, the state of a dynamical system can be estimated from the sensor measurements.

The MCL algorithm is composed of two main sections represented by two for-loops.

The first section is the motion and sensor update and the second one is the re-sampling process. Given a map, MCL is to determine the robot's pose represented by the belief (X_t). At each iteration, the algorithm takes the previous belief (Xt−1), the actuation command (u_t). and the sensor measurement (z_t). as input. Initially, the belief is obtained by randomly generating m particles. Then in the first loop, the hypothetical state is computed whenever the robot moves. Following, the particles' weight is computed using the latest sensor measurement. Now motion and measurements are both added to the previous state. In the second section of the MCL, a simple sampling process happens. Here, the particles with high probability survive and are re-drawn in the next iteration, while the others die. Finally, the algorithm outputs the new belief, and another cycle of iteration starts implementing the next motion by reading the new sensor measurements.

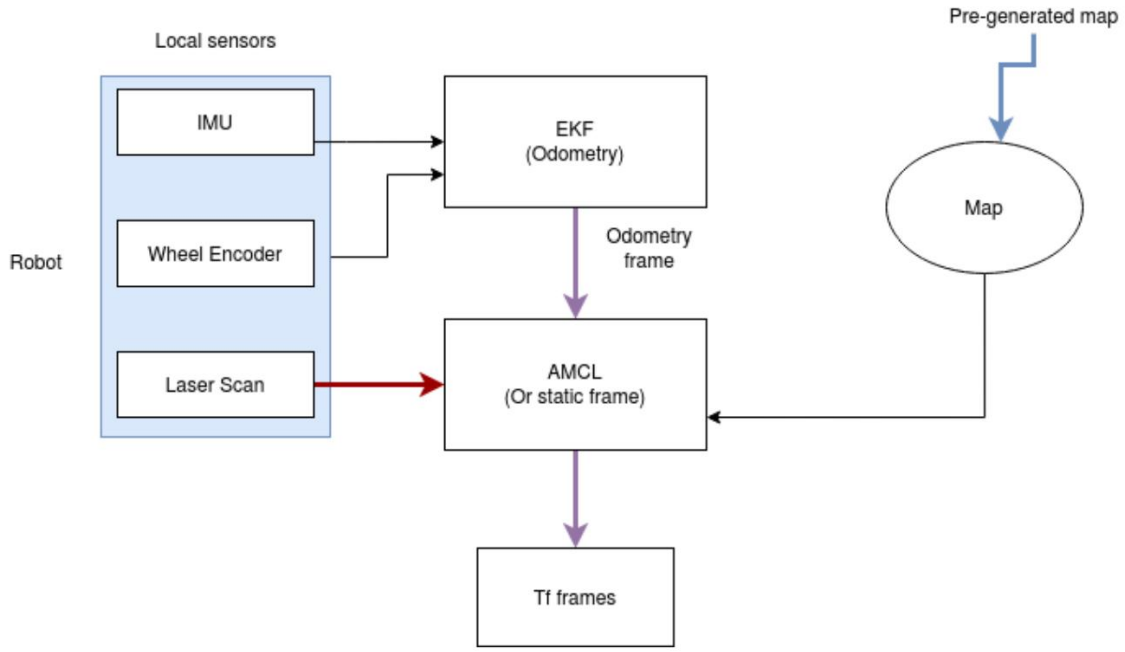


Figure 5.2.3: Localization data flow

5.2.2.3 Comparison:

There are certain significant benefits of Monte Carlo Localization over Extended Kalman Filter algorithm:

- MCL is easy to code.

- MCL represents non-Gaussian distribution and can approximate any other practical important distribution. This means MCL is unrestricted by a linear Gaussian state-based assumption as in the case of EKF. This allows MCL to model a much greater variety of environments, especially since the real world cannot be always modeled by Gaussian distributions.
- In MCL, the computational memory and the resolution of the solution can be controlled by changing the number of particles distributed uniformly and randomly throughout the map.

After understanding the concept and algorithms of mapping and localization, we now can get into the navigation control approaches:

5.3 Semi-Autonomous Control

By using the joystick on the remote mobile application, we move the vehicle in any direction.

Joystick

Joystick is a user friendly controller that takes x, and y axes values as an input related to the user thumb position on the joystick circle and map it to a unity circle as shown in figure 2.

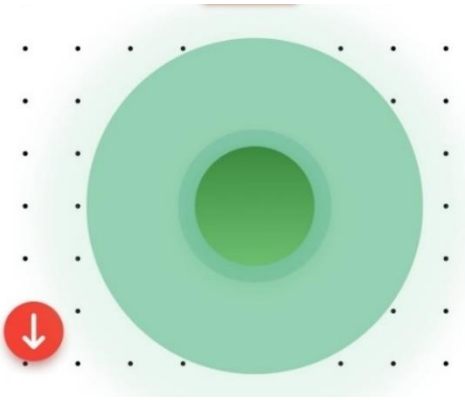


Figure 5.3.1: mobile application joystick

The goal is to move the robot towards the direction performed by the user thumb on the joystick. So, we take the input x, and y values and calculate vector magnitude and angle.

$$r = \sqrt{x^2 + y^2}$$

And $\emptyset = atan2(\frac{y}{x})$

That:

- r is the vector magnitude.
- $\emptyset$ is the vector absolute angle.

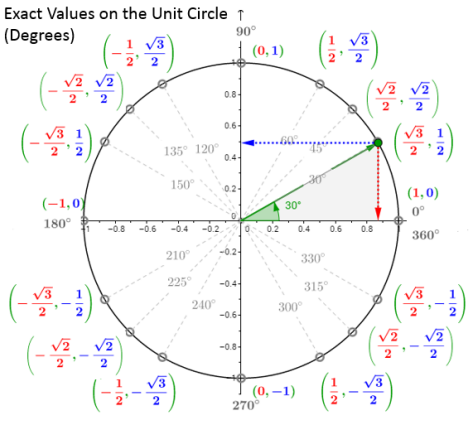


Figure 5.3.2: joystick unity circle values

Note that we use $atan2$ method instead of $atan$ when getting the angle of the vector because of different factors:

- $atan2(y, x)$ considers the signs of both x and y to determine the correct quadrant of the angle. But $\text{atan}(y, x)$ only returns values between $-\frac{\pi}{2}\ to \frac{\pi}{2}$, making it impossible to distinguish between vectors in opposite quadrants.
- $atan2(y, x)$ handles the case where x is zero without resulting in a division by zero error. While $\text{atan}(y, x)$ will cause a division by zero error if x is zero.

After calculating the magnitude and the angle of the vector on the mobile application, then we send these data to the robot to move like it.

Now we have two issues to handle:

1. Robot linear speed: and to map the r vector magnitude to the robot linear speed we just multiply the r vector by a scale which is equal to robot permissible maximum linear speed.

$$v = r * MAXIMUM\_LINEAR\_SPEED$$

2. Robot angular speed: we need to set the robot heading the same as the vector angle so, we must control the angular speed until reaches the required heading.

   This can be achieved by using a simple linear proportional controller by calculating the error between the required heading and the current heading.

   $e = \emptyset - \theta$

   That:

   - e is the heading error.
   - $\emptyset$ is the referenced heading.
   - θ is the current robot heading.

   And then multiply the error by the controller k:

   $\omega = k * e$

But the angular speed value should not exceed the maximum value, so that we must put a threshold that limits the value of ω within the permissible range.

$if\ \omega > \ MAXIMUM\_ANGULAR\_SPEED,$

$then\ \omega = MAXIMUM\_ANGULAR\_SPEED$

In this context we also should ensure that robot is going to rotate as least as possible but using atan2 introduces a small specific problem where the robot rotates in the longer direction inside of the smaller one.

For that, a special code was made to check for these corner cases to correct the direction of rotation. After trial and error, we found that there two cases that causes this problem:

a. when error > π , b. when error < - π.

So, we handled those to cases as the following conditions:

$if\ e > \ \pi\ ,$

$then\ e = \ -\ (\ 2\pi\ -\ error)$

$else\ if\ e < -\pi\ ,$

$then\ e = \ (\ 2\pi\ -\ error)$

5.4 Fully Autonomous Control

The second approach is the fully autonomous approach, in which the robot performs all steps of navigation by itself from building the map of the environment it navigates in, till the planning of the path it should take and avoidance of any obstacles in its way.

The steps of fully autonomous navigation control are:

- Mapping
- Localization
- Path planning

We have already talked about Mapping and Localization in the slam section. Now we just have path planning.

5.4.1 Path planning:

The path planning problem involves creating a path in an environment from the robot's initial pose to the goal position while ensuring the trajectory and planned motion align with the robot's kinematic constraints and avoid collisions.

Path planning is a crucial decision-making component of navigation, responsible for identifying the safest and quickest route to the destination.

Before executing any path planning, the robot's environment must be converted into a suitable discrete map. According to Siegwart, there are two primary strategies:

- **Graph Search**: This strategy involves constructing a graph of the robot's world, which is then searched to find the optimal path.

  Graph search algorithms utilize a graph data structure to map environments, where nodes represent potential robot positions and edges denote permissible movements. These algorithms navigate obstacles, consider robot dynamics, and optimize path traversal costs.

- **Potential Field Planning**: This strategy imposes a mathematical function directly onto the space, and the gradient of this function is followed to reach the goal.

  This approach is based on the concept of applying a virtual force field to the robot, where the robot is moved toward the target point due to the combined effects of attraction from the target and repulsion from obstacles.

Path planning can be categorized into two types based on environmental information levels:

1. **Global Path Planning**: This approach assumes the robot has complete knowledge of the environment and follows a predefined path. It is also known as offline or static path planning.

2. **Local Path Planning**: This approach operates with partial or no prior knowledge of the environment, requiring real-time monitoring and reaction to obstacles. It is also known as online or dynamic path planning.

Global path planning ensures an optimal route but necessitates a comprehensive map, while local path planning offers flexibility and real-time obstacle avoidance, albeit potentially yielding only locally optimal paths.

To balance these aspects, practical path planning strategies often integrate both global and local methods, combining the global optimal path with real-time obstacle avoidance to navigate effectively

5.4.1.1 Global path planning:

A. Dijkstra:

Dijkstra uses the roadmap approach to convert the problem into a graphic search method using the information of a grid cell map. This method starts with a set of candidate nodes where the vehicle can navigate (free space) assigning a cost value to each of them. From the starting point, this value is increased by the necessary number of nodes to passthrough to reach each node. For example, in Figure 5.4.1, the free space around the starting point takes the value of one unit and for the second generation of neighbors takes the value of two units and so on.

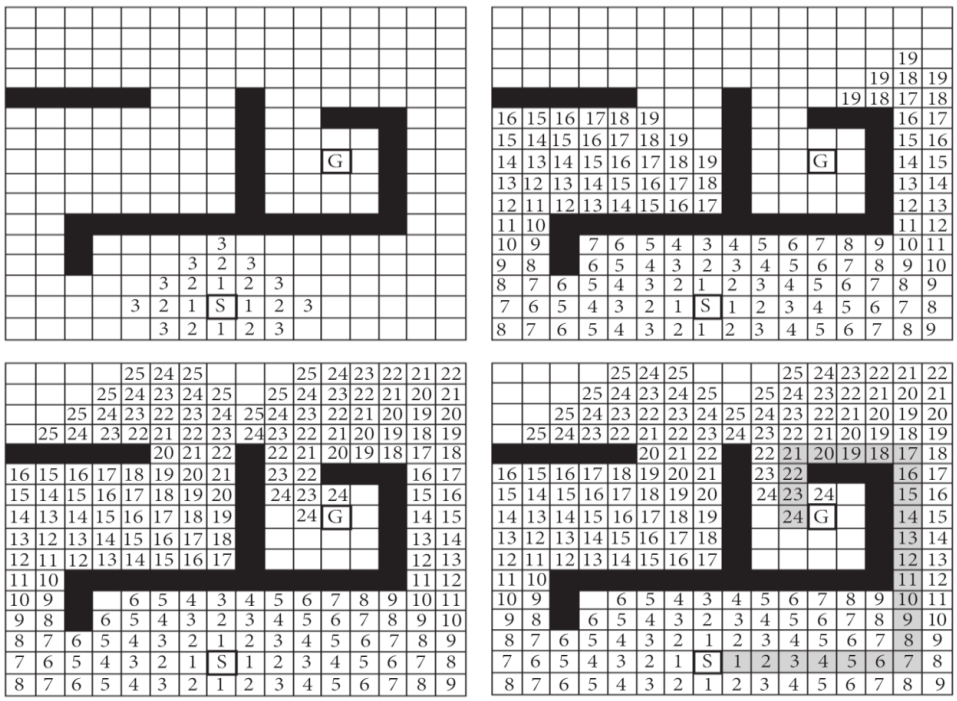


Figure 5.4.2: Dijkstra Algorithm

For each cell with value, a checked cell is assigned, and the method continues with the next generation of neighbors until the goal point is reached. The minimum value of the sum of all nodes from the starting point and the end point is the shortest path. After the success of finding the global path from the start to the goal, all the selected nodes are translated.

The minimum value of the sum of all nodes from the starting point and the end point is the shortest path. After the success of finding the global path from the start to the goal, all the selected nodes are translated into positions in the reference axes as the form $P_i = (x_i, y_i)^T$. The global planner divides the map into nodes for each free cell, but the outcome is not smooth, and some points are not compliant with the vehicle geometry and kinematics.

B. A*

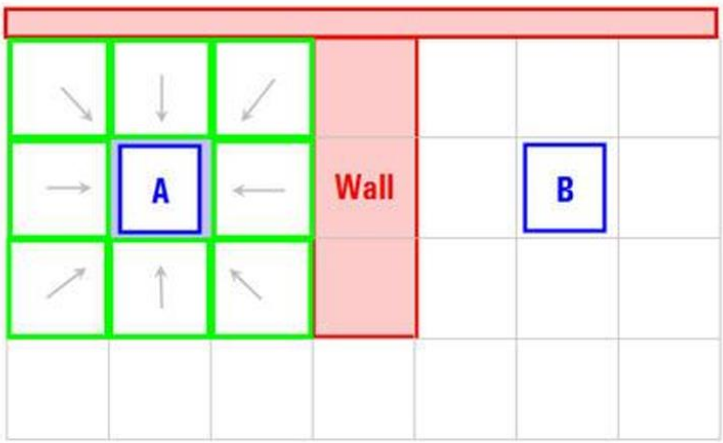


Figure 5.4.3: A algorithm explanation 1*

The A* algorithm is a graph search technique that finds the optimal path with the lowest cost from an initial node to any position in a previously defined graph. This method is like Dijkstra's algorithm but includes a heuristic function to try and reduce the number of nodes explored. Before the algorithm can be carried out, the robot world must be divided into a grid, which can be defined as a two-dimensional array. Every grid square must be classified as either a free space or an obstacle Figure 5.4.2.

All the squares (also referred to as cells or nodes) originally have their status set as unvisited or obstacle. The shortest path is created by evaluating the adjacent cells and calculating a cost function to choose the best next move.

The cost function $f(n)$ is defined as:

$$f(n) = g(n) + h(n)$$

The function $g(n)$ is the cost to move from the starting cell to any other position in the grid. The cost to move horizontally or vertically is given a value, and the cost to move diagonally is the square root of 2 times the cost of moving horizontally or vertically. The function $h(n)$ is the heuristic function which gives additional knowledge about the graph, making the algorithm more efficient.

The heuristic is the estimated cost to move from the current position to the goal node. The performance of the algorithm depends on the heuristic used, but the efficiency on average is much better than that of the Dijkstra algorithm.

If the heuristic function $h(n)$ is 0, the A* algorithm will behave the same as Dijkstra's and will be slower. If $h(n)$ has a very high value, the algorithm will run very quickly, however the path may not be the shortest possible route.

The heuristic used in this project is calculated estimating the Euclidian distance to target cell. This method estimates the absolute distance to the goal, ignoring any obstacles that might be in the way. In the first step of the algorithm, all the cells that are adjacent to the first cell are either added to an open list or a closed list. The open list contains all the cells that are possible candidates for the next move. The closed list contains all the cells that have already been evaluated or are obstacles. The value of the function $f(n)$ is calculated for all the adjacent cells that are in the open list. Supposing that the cost for horizontal and vertical moves is 10, for diagonal moves 14 and with A being the parent cell, the values for the adjacent cells in figure 5.4.2 would be as shown in Figure 5.4.3:

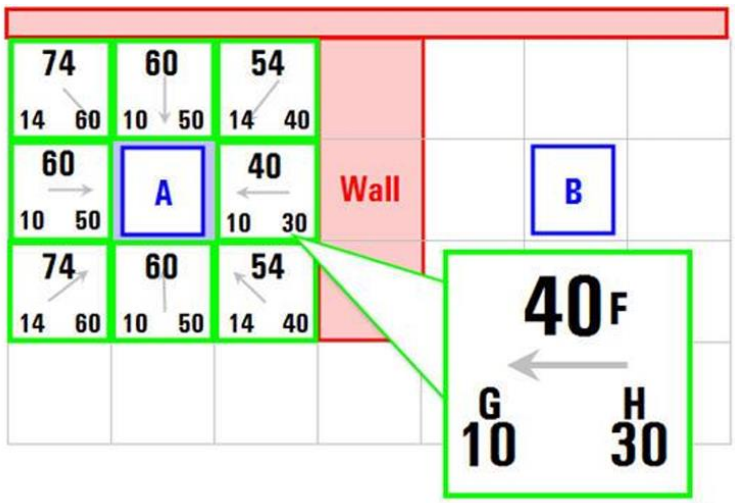


Figure 5.4.4: A algorithm explanation 2*

Once the cost for each adjacent cell has been calculated, the cell with the lowest value is chosen (in this case the cell with a total $f(n)$ score of 40) and the cells surrounding it that are not obstacles, or the previous cell are added to the open list and their new cost scores are calculated.

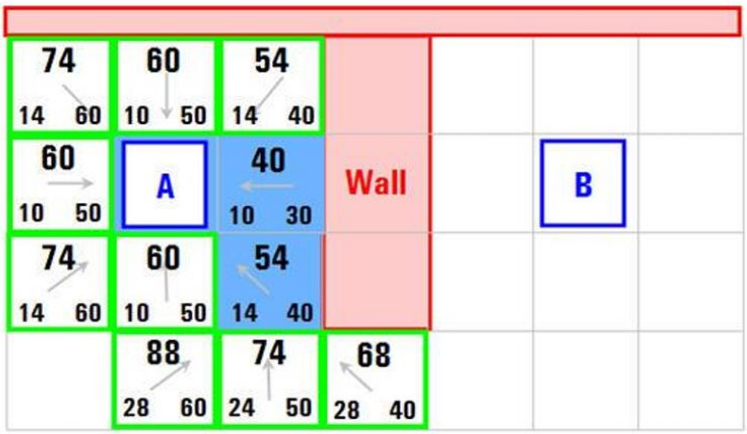


Figure 5.4.5: A algorithm explanation 3*

This process is repeated until the goal position is reached. To determine the path once at the target, the algorithm works backwards moving to the parent cell of the current cell (shown by the orange arrows in Figure 5.4.5). This is the path with the lowest cost from the starting position to the target cell.

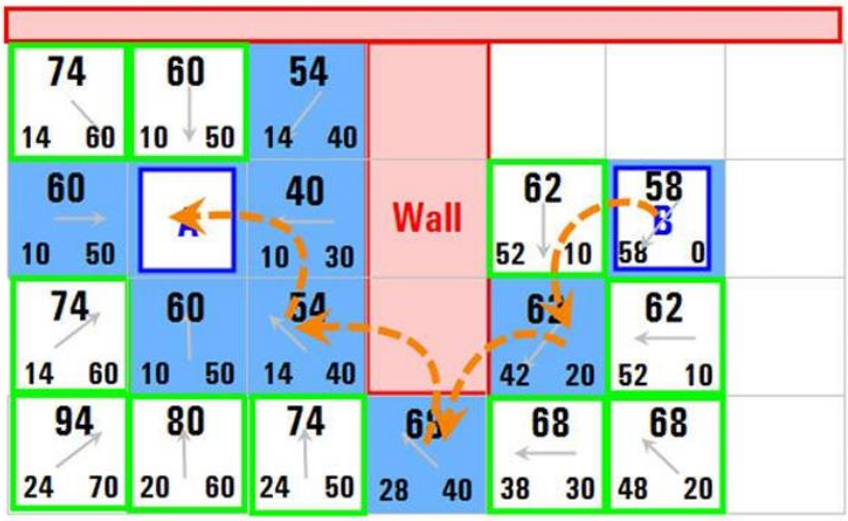


Figure 5.4.6: A algorithm explanation 4*

C. D*

The D* algorithm (Dynamic A*) is a real-time planning version of the A* algorithm which allows the path to be recalculated if unexpected obstacles are encountered.

If the environment map is incomplete or changes while the robot is moving, the path will need to be replanned. With the A*, the robot would have to wait for the path to be completely recalculated, which is slow and inefficient.

When the robot receives additional information about its environment, the D* algorithm can revise the path it calculated before and try to reduce the total cost of the trajectory. The D* algorithm start by planning a path using the A* method. After a certain amount of time, the robot might observe some changes in the environment with its onboard sensors, and a new trajectory must be computed.

The D* algorithm then generates a new path for the states that are affected by the new obstacle that was added or removed, without having to backtrack.

In addition to the open and closed lists used in the A* algorithm, this method has two more states for each node: Raise and lower. When a cell that was previously thought to be free space is an obstacle, the cost of that position increases. When a cell that contained an obstacle is free space, the state will be Lower because the cost of the path can be decreased. Analogous to the A* algorithm, a dynamic version also exists for the D* algorithm, which is known as Anytime D*.

Why A*:

- **Optimality**: A* guarantees finding the shortest path (with the lowest cost) in a graph when an admissible heuristic is used. This ensures efficiency in finding optimal routes, crucial for applications where minimizing travel time or energy consumption is essential.
- **Heuristic Function**: By incorporating a heuristic function, A* can focus on more promising paths, reducing the number of nodes explored compared to uninformed search algorithms like Dijkstra's. This makes it faster and more efficient, especially in large-scale environments.
- **Versatility**: A* can be applied to a wide range of grid-based or graph-based navigation problems. Its adaptability to different grid resolutions

and obstacle configurations makes it versatile for various real-world scenarios.

- **Real-Time Planning**: In dynamic environments or scenarios requiring real-time decision-making, A* performs well due to its ability to quickly adjust path calculations based on updated heuristic estimates and dynamic obstacle information.
- **Implementation Efficiency**: While A* is more computationally intensive than simpler algorithms like Dijkstra's, its efficiency in finding optimal solutions outweighs the additional computational cost, especially with modern computational capabilities.
- **Anytime Replanning**: A* can be extended to support anytime replanning, where initial paths are refined or adjusted over time based on changing environmental conditions or user preferences, ensuring adaptability and robustness

5.4.1.2 Local planner:

A local planner is a critical component of autonomous robot navigation systems, responsible for real-time path planning and obstacle avoidance. Unlike global planners that compute long-term paths from start to goal, local planners operate within shorter time horizons, typically influenced by immediate sensor data. Their primary function is to ensure the robot navigates safely around obstacles while adhering to the global path provided by higher-level planning algorithms.

A. Vector Field Histogram Plus:

Vector Field histogram Plus (some authors call it Enhanced Vector Field Histogram) is an improvement of Borenstein's and Koren's Vector Field Histogram (VFH). VFH employes a polar histogram grid for representation of robot's surrounding environment. This grid is built from a two-dimensional certainty grid updated from sensor readings taken with a ranging sensor (sonar, laser rangefinder). The polar histogram is a vector that moves with the robot.

Each element of the histogram corresponds to a circular sector for which information about amount and distance of obstacles in the form of a weighted sum is stored. The histogram after smoothening (by a simple low-pass filtering)

has typically “peeks”, i.e. sectors with high values and “valleys”, sectors with low values. Any valley containing sectors with values below a certain threshold can be called a candidate valley. If more than one candidate valley is detected, the best one that most closely matches the direction to the target is selected. Finally, the most suitable sector within the selected valley is chosen as the next goal and the robot is navigated towards it. The whole process is repeated whenever new sensor data are gathered (or in predefined time steps) until the final goal is reached. VFH+ enhances the original algorithm in several ways. First, threshold hysteresis is used to suppress alternating between several goals in narrow openings resulting in robot movement in the close vicinity of obstacles. Moreover, robot’s size is considered by enlarging obstacle cells by a robot diameter. Finally, the dynamics and kinematics of the robot were not taken into consideration. VFH+ uses a simple approximation of currently possible robot’s trajectories by a set of circular arcs with various curvatures assuming that forward and angular velocities are piecewise constant.

B. Smooth Nearness Diagram:

Another approach was presented by Minguez and Montano as a geometry-based implementation of the reactive navigation method design. The approach called Nearness Diagram (ND) navigation introduces “gaps” – discontinuities in the nearness of obstacles to the robot which indicate potential paths into occluded areas of the environment. By the pairs of consecutive gaps “regions” can defined, navigable regions are then “valleys”. After assembling all the valleys surrounding the robot, all the gaps are compared against the heading provided by the global planner. The next subgoal and control to it is determined based on position of two closes obstacles and the width of the valley containing the gap with the heading that best matches the heading to the goal. Smooth Nearness Diagram (SND) differs from ND in the way the next subgoal is determined. SND measures a threat possessed by each of the obstacles (an obstacle is considered a threat if it lies within the safety distance of the robot) – the treat measure increases as the obstacle gets closer to the robot.

Deflection from the desired heading is computed based on threat measurements of each obstacle. The experiment showed that oscillatory patterns were suppressed leading to method’s performance improvement in narrow corridors.

C. DWA:

The Dynamic Window Approach (DWA) is a crucial algorithm in mobile robotics for real-time path planning and obstacle avoidance. It enables autonomous robots to navigate dynamic and unpredictable environments effectively by considering their motion capabilities and the surrounding obstacles.

DWA is particularly effective in real-time applications where quick responses to changing conditions are essential. The DWA algorithm works by evaluating a range of potential velocities and selecting the best one based on several criteria, such as safety, feasibility, and efficiency. It generates trajectories in the velocity space and scores them according to their desirability. The chosen trajectory is the one that best satisfies the robot's objectives while ensuring safe navigation.

The key concepts and components of DWA include the robot's state representation, velocity space and dynamic window, trajectory generation, and trajectory evaluation. The robot's state is represented by its position and orientation. The velocity space is defined by the linear and angular velocities that the robot can achieve. The DWA explores the velocity of space by considering all possible combinations of linear and angular velocities within specified limits.

The dynamic window is a subset of this velocity space, constrained by the robot's current velocity and acceleration capabilities. For each combination of linear and angular velocities within the dynamic window, the algorithm predicts the resulting trajectory over a short time horizon. These trajectories are simulated based on the robot's motion model, which typically involves integrating the velocities to estimate future positions.

Each predicted trajectory is evaluated based on three primary criteria: collision risk, proximity to obstacles, and goal alignment. The trajectory's smoothness and feasibility, characterized by minimal changes in velocity, are also considered to ensure comfortable and feasible motion. The trajectories are scored using a weighted combination of the evaluation criteria.

The trajectory with the highest score is selected as the optimal path for the robot to follow. This involves sensor integration, velocity sampling, trajectory prediction, trajectory evaluation, optimization and execution. DWA requires real-time sensor data to accurately assess the environment. Sensors such as

LiDAR, cameras, and ultrasonic sensors provide information about obstacles and the robot's surroundings.

The algorithm iterates through possible linear and angular velocities within the dynamic window, generating candidate trajectories for each combination. Using the robot's motion model, the future positions and orientations are predicted for each velocity pair, creating a set of potential trajectories. Each trajectory is evaluated based on collision risk, proximity to obstacles, goal alignment, and smoothness. The scoring function combines these factors to rank the trajectories. The trajectory with the highest score is selected as the optimal path. The robot adjusts its velocity to follow this trajectory, continuously updating its path as new sensor data is received.

Why to use DWA:

The Dynamic Window Approach (DWA) is a powerful algorithm for local path planning and navigation in mobile robotics, offering several key advantages:

- **Real-Time Responsiveness:** DWA operates in real-time, continuously evaluating potential trajectories based on the latest sensor data. This allows robots to respond quickly to changing conditions and avoid collisions in dynamic environments, making it ideal for applications like autonomous vehicles and service robots.
- **Safety and Collision Avoidance:** DWA prioritizes safe navigation by evaluating each potential trajectory for collision risk. Using data from sensors like LiDAR and cameras, it ensures the robot maintains safe distances from obstacles, which is crucial for environments with humans or delicate equipment.
- **Feasibility and Smooth Motion:** The algorithm incorporates the robot's kinematic and dynamic constraints, ensuring feasible and realistic trajectories. It also prioritizes smooth, continuous motion, which improves stability, efficiency, and reduces wear on the robot's components.
- **Goal Alignment and Efficiency:** DWA balances immediate obstacle avoidance with long-term path efficiency, ensuring the robot stays aligned with its overall navigation goals. This dual focus makes it suitable for tasks were reaching specific destinations efficiently is important.
- **Versatility Across Applications:** DWA can be applied to various types of robots and environments, from indoor service robots to outdoor autonomous vehicles. Its adaptability to different sensor inputs and conditions makes it a flexible and reliable choice for many applications.

- **Integration with Robotic Systems:** DWA integrates well with robotic frameworks like the Robot Operating System (ROS), which simplifies implementation and enhances overall functionality. Its modularity allows for easy integration with other subsystems like perception, localization, and motion control.
- **Handling Dynamic Environments:** DWA excels in environments where obstacles move or appear unexpectedly. Its ability to update trajectory predictions in real-time ensures effective navigation even in dynamic settings.

5.5 Implementation:

5.5.1 Mapping:

1. **Semi-Automatic Navigation Control**
   - The robot is manually navigated through the environment.
   - The user uses the mobile application to control the robot's movement.
   - The user ensures the robot traverses all necessary areas of the solar plant.
   - The robot moves according to the commands from the mobile application.
   - It explores different paths and areas to gather comprehensive data.

2. **Sensor Data Collection**
   - The robot collects data from various sensors.
     - The LiDAR sensor continuously scans and measures distances to obstacles.
     - The IMU provides data on the robot’s orientation and motion.
     - Encoders track the movement of the robot’s wheels, providing odometry data.
   - The robot processes data from its LiDAR, IMU, and encoders.
   - It uses this data to understand its environment and position.

3. **GMapping Node Operation**
   - The robot creates a real-time map using collected sensor data.
   - The GMapping node processes incoming sensor data to construct the map.
   - The map is continuously updated as the robot moves and explores.
   - The robot integrates sensor data to build and refine a map of the environment.
   - It uses an algorithm to process and represent this data accurately.

4. **Complete Environment Coverage**
   - The robot ensures all areas of the solar plant are mapped.
   - The user guides the robot to navigate through all paths and spaces.
   - The robot continues to move and explore until it has covered the entire area.
   - The user ensures that no part of the solar plant is left unmapped.

5. **Save the Generated Map**
   - The generated map is saved for future use.
   - The map is stored as files (typically in .yaml and .pgm formats).
   - The robot provides the final map data to be saved.

6. **Load the Saved Map**
   - The saved map is loaded into the robot's system.
   - The map is integrated into the robot's navigation stack.
   - The robot accesses and uses the saved map for navigation.
   - It prepares to use the map data for autonomous operations.

7. **Autonomous Navigation with Saved Map**
   - The robot uses the saved map to navigate autonomously.
   - The robot's navigation stack utilizes the map for localization and path planning.
   - Algorithms such as AMCL (Adaptive Monte Carlo Localization) and path planning are employed.
   - The robot autonomously navigates the solar plant using the pre-saved map.
   - It localizes itself within the map and follows planned paths to move efficiently.

Map server:

The map_server in ROS is a vital component that facilitates the storage and retrieval of maps. It is responsible for managing map data and providing access to maps for various ROS nodes and applications.

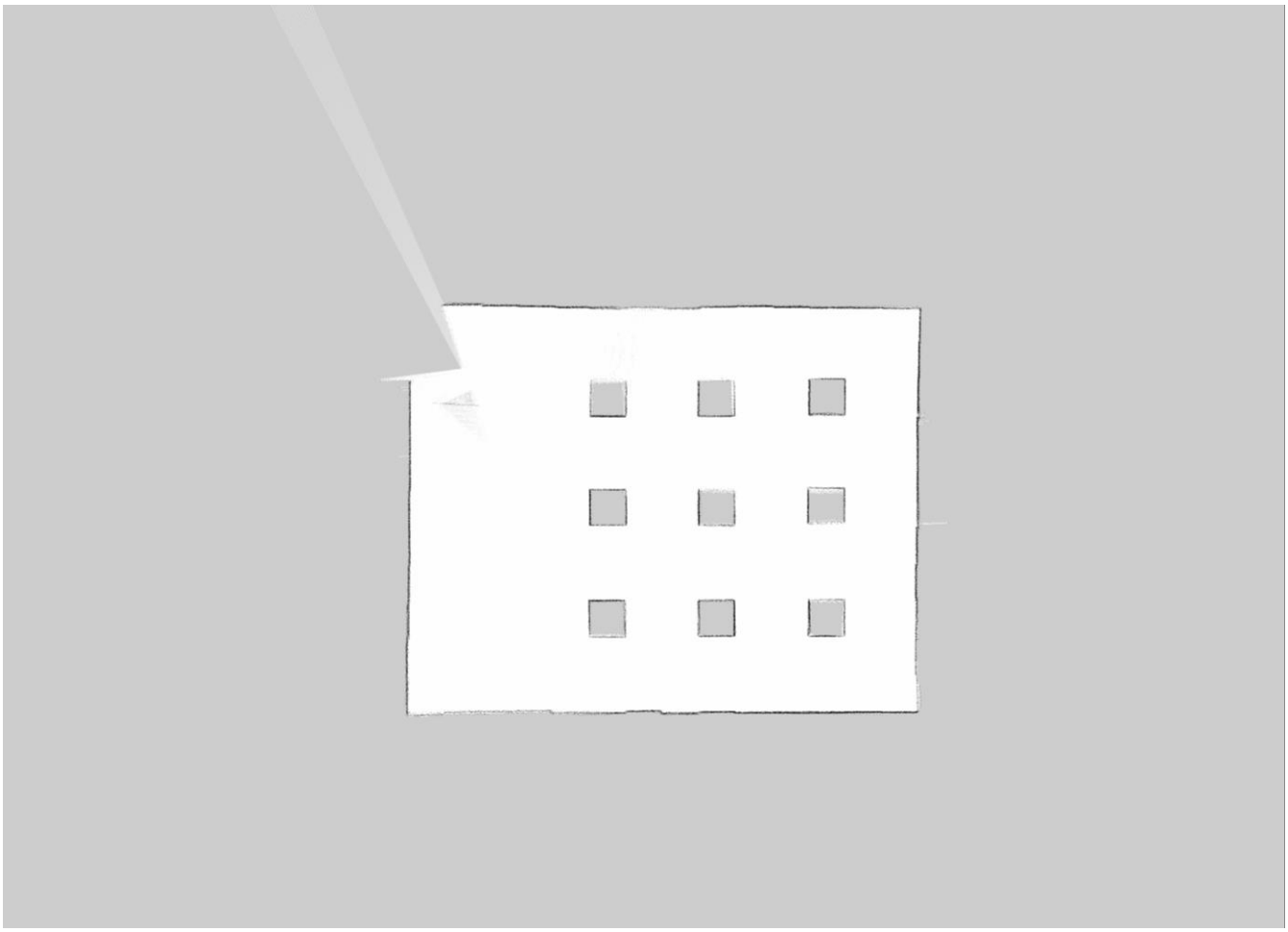
The map_server allows users to save maps generated by SLAM algorithms or other mapping techniques and publish them as a ROS service. When using the map_server, the user can load a pre-existing map from a file or generate a new map using SLAM or mapping algorithms. Once the map is loaded, the map_server provides a service interface that allows other ROS nodes to request the map data. This service enables nodes, such as navigation systems or visualization tools, to retrieve the map for their respective functionalities.

The map_server can publish the loaded map using the "map" topic, which allows other nodes to subscribe and receive updates whenever the map changes. This enables real-time map sharing and visualization. Additionally, the map_server can handle requests to save the map back to a file, allowing users to store the map data for future use or analysis. By utilizing the map_server in ROS, users can efficiently manage map data, share maps between different components, and persistently store maps for later use. This capability is essential in robotics applications that require reliable and consistent map storage and access. The map_server plays a crucial role in enabling seamless integration of map data within the ROS ecosystem and facilitating map-based functionalities, such as navigation, localization, and path planning.

Role of the Map Server in Navigation:

1. Map Loading:
    - The Map Server loads a pre-saved map file (typically .yaml and .pgm files) into the ROS environment.
    - This map represents the layout of the solar plant, including obstacles and free spaces.
2. Map Serving:
    - Once loaded, the Map Server publishes the map data to the ROS environment.
    - Other nodes, such as AMCL (Adaptive Monte Carlo Localization) and path planning nodes, can subscribe to the map topic provided by the Map Server.

- The map data is used for localization, navigation, and path planning.



Solar plant map

Integration into the Navigation Stack:

1. Localization:
   - AMCL uses the map data provided by the Map Server to localize the robot within the environment.
   - Accurate localization is essential for the robot to understand its position and orientation relative to the map.
2. Path Planning:
   - Path planning nodes utilize the map data to generate paths from the robot's current position to its target destination.
   - The map data helps in identifying obstacles and free spaces, ensuring safe and efficient path planning.
3. Autonomous Navigation:
   - With the map loaded and published by the Map Server, the robot can autonomously navigate the solar plant.
   - The navigation stack uses the map for localization and path planning, enabling the robot to move to desired locations while avoiding obstacles.

5.5.2 Navigation stack:

The navigation stack involves several interconnected components that work together to enable autonomous navigation. This section explains each block in the data flow graph, their roles, and how data flows through the system.

Components:

1. **Sensor Sources**

   LiDAR:

   - Scans the environment for obstacles and measures distances.
   - In ROS we use Rplidar package to interface with it.
   - The output is then fed to laser filters to control the angle of the sensor.

2. **Sensor transforms**
   - Transforms sensor data into a unified coordinate system.
   - Ensures that data from different sensors is accurately aligned and integrated.
   - Provides a consistent reference frame for all sensor data, enabling accurate mapping and localization.
3. **AMCL (Adaptive Monte Carlo Localization)**
   - Uses particle filters to estimate the robot's position and orientation within the map.
   - Processes data from sensor sources to maintain accurate localization.
   - Provides real-time localization, allowing the robot to understand its current position relative to the map.

4. **Map Server**
   - Loads and serves a pre-saved map to the ROS environment.
   - Publishes the map data, making it accessible to other nodes in the navigation stack.
   - Supplies the map used for localization and path planning, ensuring consistent navigation data.

5. **Odometry source**
   - **Wheel Encoders**: Wheel encoders measure the rotation of each wheel, providing data on the distance traveled. This data is crucial for estimating the robot's linear and angular velocities.
   - **IMU (Inertial Measurement Unit)**: IMUs provide orientation (roll, pitch, yaw) and angular velocity measurements. They help compensate for drift in wheel odometry, enhancing the accuracy of the robot's pose estimation.
   - **Sensor Fusion with EKF**: The Extended Kalman Filter (EKF) combines the non-linear data from wheel encoders and IMU to estimate the robot's position and orientation accurately.

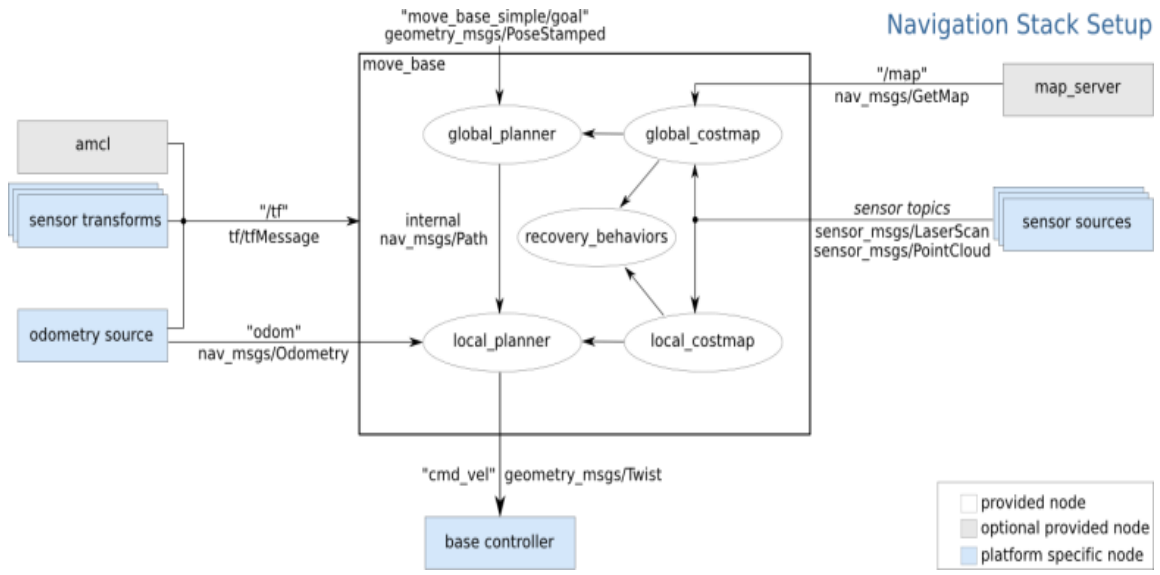


Figure 5.5.1: navigation stack block diagram

6. Move base:

Coordinates the planning and execution of navigation tasks, ensuring smooth and efficient movement towards goals.

- **Global Planner:** Computes the optimal path from the robot's current position to the target destination, considering the entire map.
- **Global cost map:** Represents the global view of the environment, incorporating static and dynamic obstacles.
- **Recovery Behaviors:** Implements strategies to handle navigation failures, such as obstacle avoidance and re-planning.
- **Local cost map:** Represents the local view of the environment, focusing on nearby obstacles and immediate navigation adjustments.
- **Local Planner:** Generates real-time trajectories for the robot to follow, adapting to dynamic changes in the environment.

7. Base Controller

- Receives commands from the Move Base and translates them into motor commands.
- Controls the robot's movement, ensuring it follows the planned paths and trajectories.
- Executes the navigation commands, physically moving the robot within the environment.

Differential Drive:

We use a differential drive system as the base controller for our robot. This setup is crucial for controlling the robot's movement precisely and efficiently.

- The differential drive system translates velocity commands into appropriate wheel velocities, ensuring the robot follows the planned paths accurately.
- The system involves several interconnected nodes, each playing a vital role in processing and executing these commands. Here is how it works, utilizing the key nodes from the differential_drive ROS package:

1. **diff_tf node:**

   Input: Output from the left and right wheel encoders.

   Output:

   - Publishes the odometry (the current pose and twist) of the robot.
   - Broadcasts the transform between the odometry frame and the robot's base_link.

2. **twist_to_motors node:**

`The node twist_to_motors` translate a twist message into velocity target messages for the two motors.

Input: The target twist for the robot.

Output:

- The target velocity for the left wheel
- The target velocity for the right wheel

3. pid_velocity node:

`The node pid_velocity` is a PID controller using feedback from a rotary encoder to the wheel power to control the velocity of the robot wheel. A typical differential drive setup will have 2 of these PID velocity controllers, one for each wheel.

Input:

- The output of the rotary encoder.
- The desired velocity in meters/second.

Output:

- The output of the PID controller, the power going to the motor.

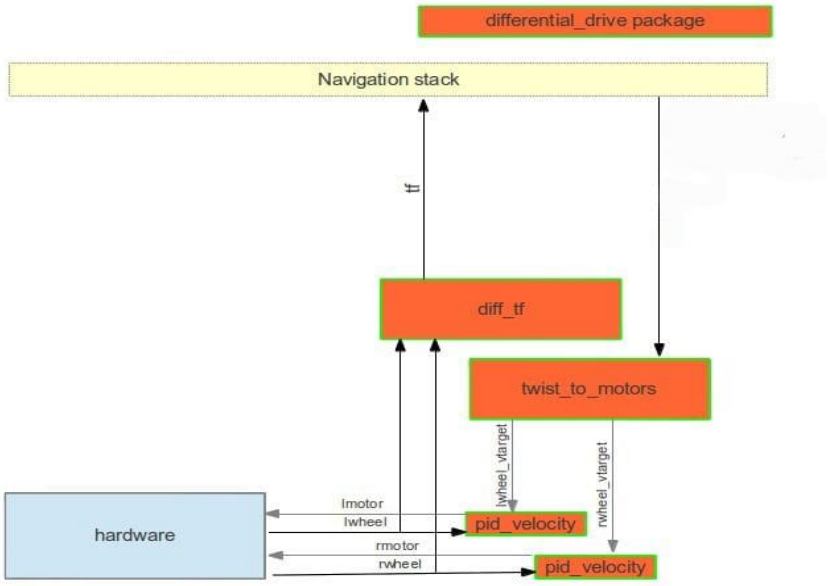


Figure 5.5.2: differential drive package block diagram

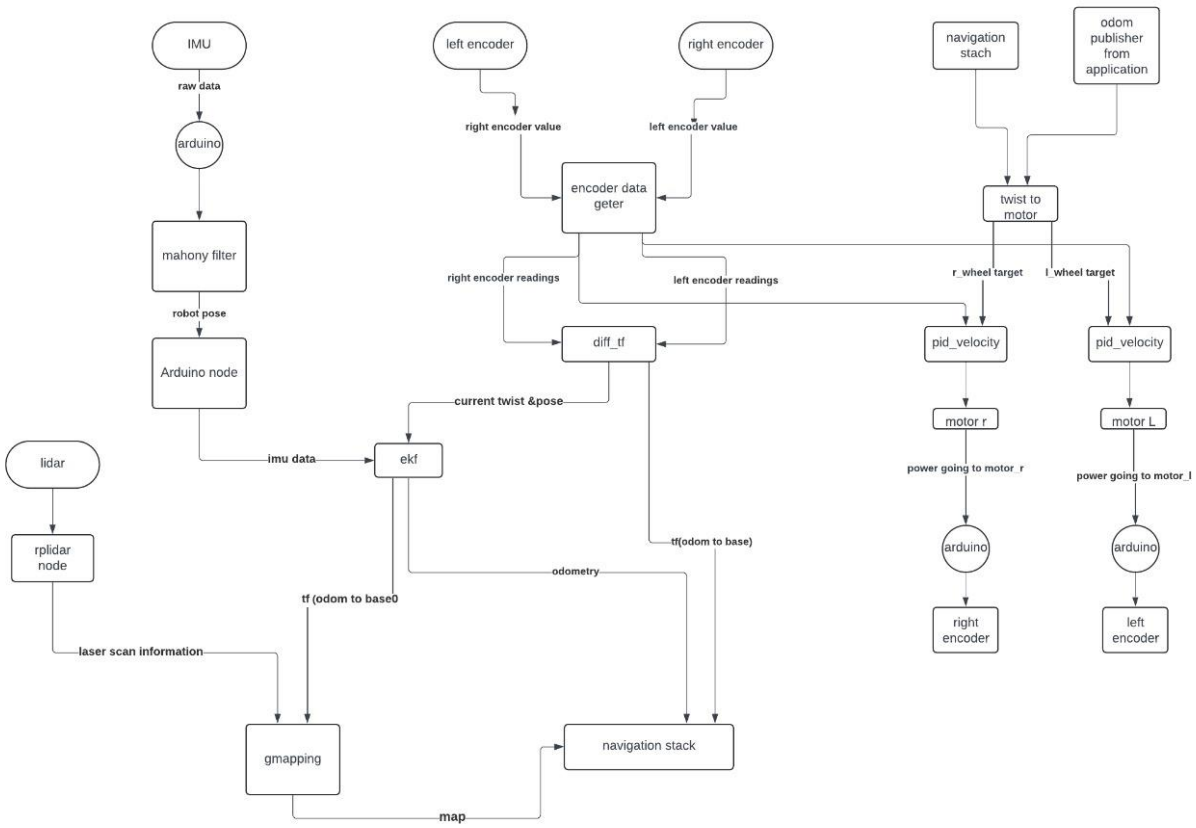


Figure 5.5.3: Navigation Flow diagram in ROS

5.6 Robot behavior:

In this section, we will discuss the performance of the solar panel cleaning robot, focusing on its path navigation and the coordinated operation of its end effector-the brush. The robot is designed to follow a predefined path, ensuring efficient cleaning of solar panels while avoiding obstacles.

Path Planning with Waypoints

The robot navigates using waypoints, which are predefined coordinates that guide its movement. These waypoints define the global path the robot follows during operation. The robot starts from an initial waypoint and proceeds through the sequence, ensuring it covers the entire solar panel array efficiently.

Operational Workflow

1. **Starting Position:**
   - When the robot reaches the start of a row, it stops.
   - The robot arm moves to a defined position, which is the initial position for starting the cleaning process.

2. **Cleaning Operation:**
   - The robot arm extends forward, maintaining a consistent distance from the solar panel surface to ensure thorough cleaning.
   - The brush begins its cleaning motion, synchronized with the robot's forward movement.
   - The robot continuously monitors its distance from the solar panels and any potential obstacles, adjusting its path as necessary to maintain safe operation.

3. **End of Row:**
   - Upon reaching the end of a row, the robot stops.
   - The robot arm moves to position one to avoid any contact with solar panels or obstacles during the transition to the next row.
   - The robot executes a zigzag maneuver to align with the next row.

4. **Transition and Repeat:**
   - The robot repeats the process for each subsequent row, ensuring the entire solar panel array is cleaned systematically.

Performance Metrics

To evaluate the performance of the robot, several metrics are considered:

4. **Cleaning Efficiency:**
   - Measure the time taken to clean each row and the entire array.
   - Assess the thoroughness of cleaning by inspecting the solar panels for any remaining debris.

2. **Navigation Accuracy:**
    - Monitor the robot's adherence to the predefined path using sensor data and feedback mechanisms.
    - Evaluate the robot's ability to maintain a safe distance from solar panels and obstacles.

3. **Operational Smoothness:**
    - Analyze the robot's transitions between rows, ensuring minimal disruption and smooth movement.
    - Evaluate the synchronization between the robot's navigation and the brush operation.

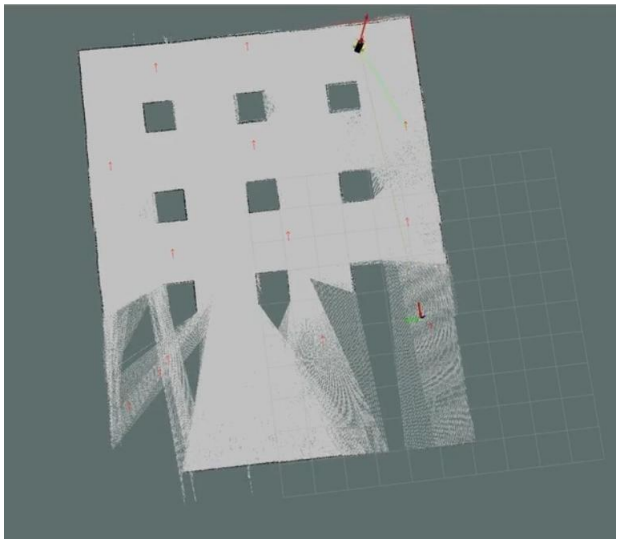
4. **Obstacle Avoidance:**
    - Assess the robot's ability to detect and navigate around obstacles without interrupting the cleaning process.
    - Measure the frequency and severity of any collisions or near-misses.

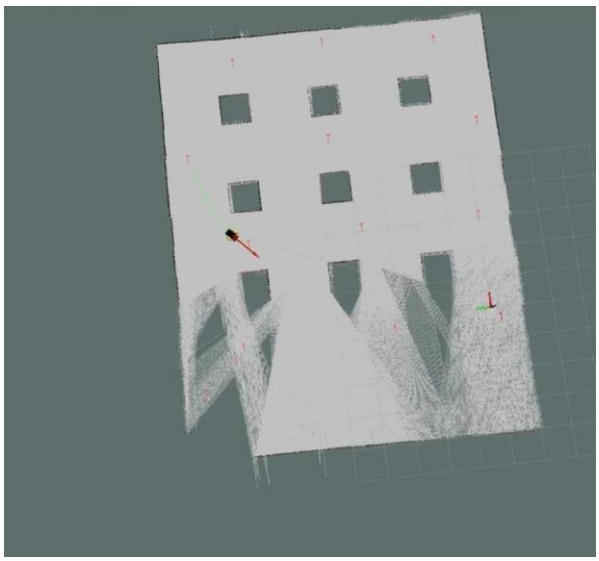
Integration with the Brush Operation

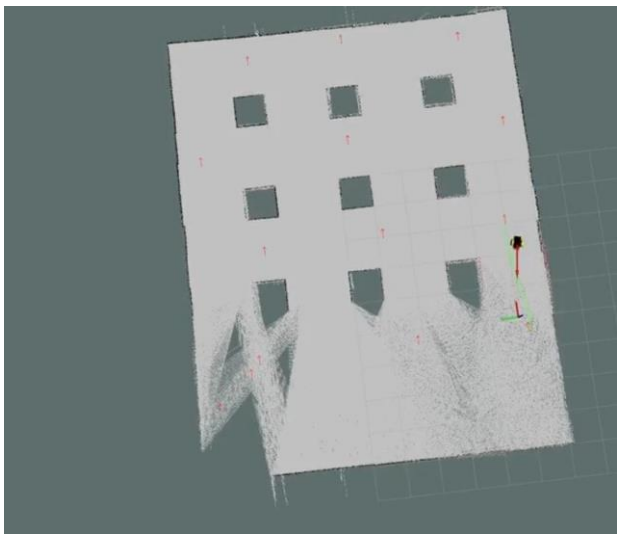
The brush, as the end effector of the robot arm, plays a crucial role in the cleaning process. Its operation is closely integrated with the robot's navigation system:

- The brush starts and stops cleaning based on the robot's position and motion.
- The robot arm's movement is coordinated to ensure the brush maintains optimal contact with the solar panel surface.
- The brush's speed and pressure can be adjusted based on the robot's speed and the condition of the solar panels.

GENERATED MAP







Chapter 6

ARM MANIPULATION CONTROL

6.1 INTRODUCTION

This chapter introduces the control of a two degrees-of-freedom (2-DOF) arm robot.

Robot arms are mechanical devices that mimic the movement and functionality of a human arm, allowing for precise control and movement in various directions.

Within the mobile vehicle system, the 2-DOF arm enhances functionality and flexibility. This introduction highlights the arm's role, design, and capabilities, showcasing its contribution to the project's broader objectives.

6.2 Main rule and components of the arm

Rule

The main rule of the arm is that it holds the cleaning brush frame. The arm is moving the brush to place it on the solar panel correctly and with suitable pressure &speed on the panel, so it doesn’t affect or damage the solar panel. Reaching the correct position using the movement of its joints.

Components of the arm

- Base
- Links
- Joint (revolute)
- End-Effector (brush)
- Actuators (linear actuator)
- Sensors (ultrasonic sensor, read sensor, potentiometer)

When talking about arm manipulation control, we have some factors to concern about:

1. Arm kinematics.
2. Arm dynamics.
3. Arm trajectory planning.

6.3 Kinematics

The kinematics analysis on the manipulator is an important part of the robot study. The manipulator is composed of a serial of rigid links connected to each other with revolute or prismatic joints. Each joint location is usually defined relative to neighboring joint.

Calculating the position and orientation of the end-effector of the manipulator by the given joint angles is known as **forward kinematics**.

Inverse kinematics is the process of determining the joint configurations of a robotic manipulator to achieve a desired end-effector position and orientation in the operational space.

Kinematics is crucial because it helps us understand and predict the motion of robotic systems without considering the forces that cause this motion. It allows us to design and control robots accurately, ensuring they follow desired paths and perform tasks precisely.

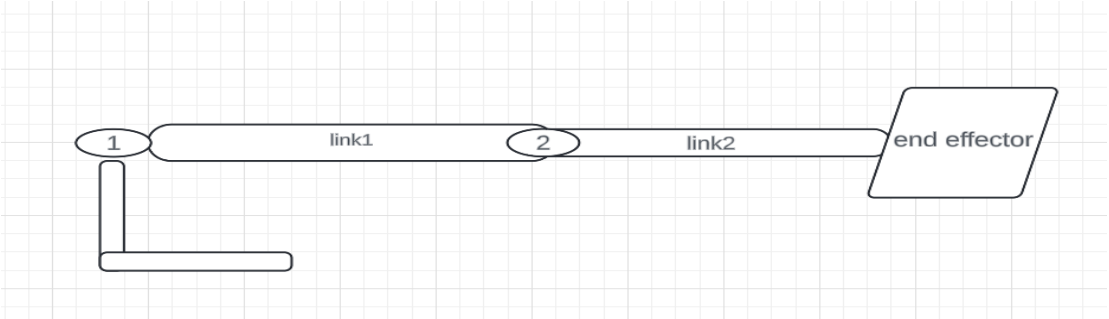


Figure 6.3.1: arm schematic diagram

6.3.1 Forward kinematics

Denavit Hartenberg (D-H) Parameters

Denavit and Hartenberg had put forward a matrix method to build the attached coordinate system on each link in the joint chains of the robot for describing the relationship of translation or rotation between the contiguous links in 1955. [1]

Denavit-Hartenberg (D-H) parameters are used in robotics and mechanical engineering for standardizing and simplifying the description of joint relationships. They facilitate accurate kinematic modeling, streamline coordinate transformations between joints, support key algorithms like forward

and inverse kinematics, and provide a physical basis for understanding robotic systems

6.3.1.1 Implementing D-H parameters for the arm

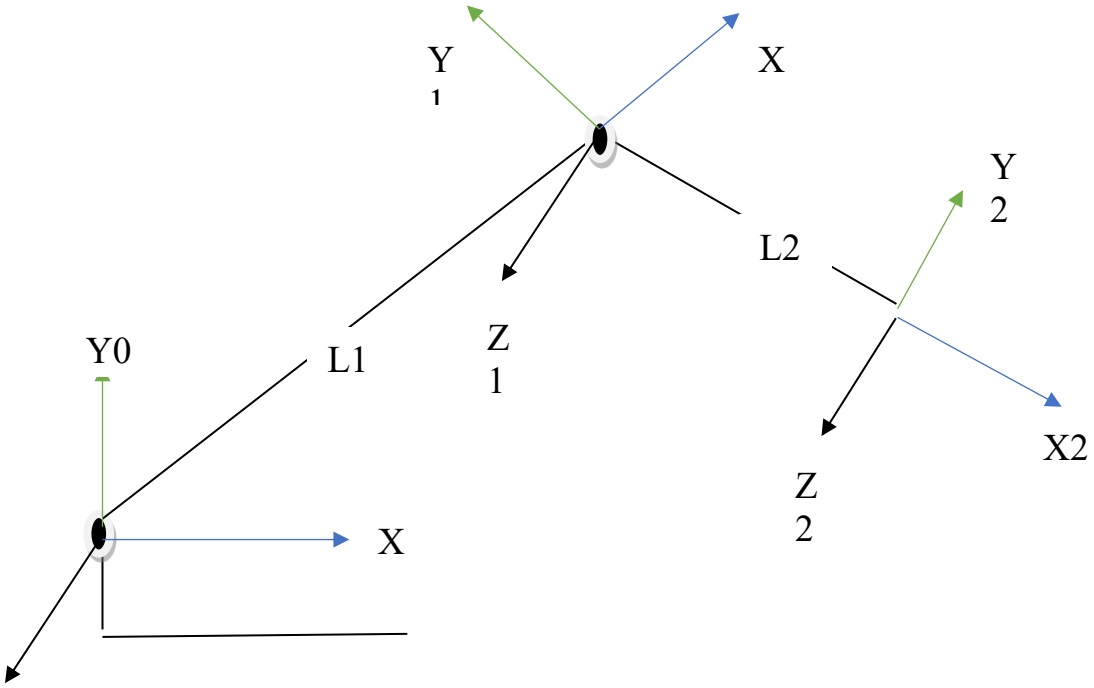


Figure 6.3.2: arm wire diagram

joint	type	theta	alpha	a	d
1	R	$\theta 1$	0	L1	0
2	R	$\theta 2$	0	L2	0

Table 6.1 D-H parameters

$$ {}^{0}_{1}T = \begin{bmatrix} c1 & -s1 & 0 & L1c1 \\ s1 & c1 & 0 & L1s1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} , \quad {}^{1}_{2}T = \begin{bmatrix} c2 & -s2 & 0 & L2c2 \\ s2 & c2 & 0 & L2s2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}_{2}T = {}^{0}_{1}T \cdot {}^{1}_{2}T = \begin{bmatrix} c1 & -s1 & 0 & L1c1 \\ s1 & c1 & 0 & L1s1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c2 & -s2 & 0 & L2c2 \\ s2 & c2 & 0 & L2s2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = $$

$$\begin{bmatrix} c12 & -s12 & 0 & L1c1 + L2c12 \\ s12 & c12 & 0 & L1s1 + L2s12 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.4 Inverse kinematics

Here, the goal is to find the joint angles or parameters that achieve a desired position or orientation of the end-effector. It's like solving the puzzle of how to position a robot's arm to reach a specific point in space, considering constraints like joint limits and reachability.

Inverse kinematics is used to calculate the corresponding joint angles when the posture of the end effect is known. In this work, the graphical method is adopted.

The graphical method in inverse kinematics uses visual representations, like geometric shapes or coordinate systems, to depict the relationship between joint angles and the end-effector's position. It simplifies complex calculations, offering intuitive solutions through iterative adjustments of joint parameters.

Advantages include visual clarity, simplicity in concept, and ease of implementation, making it valuable for both educational purposes and practical applications in robotics.

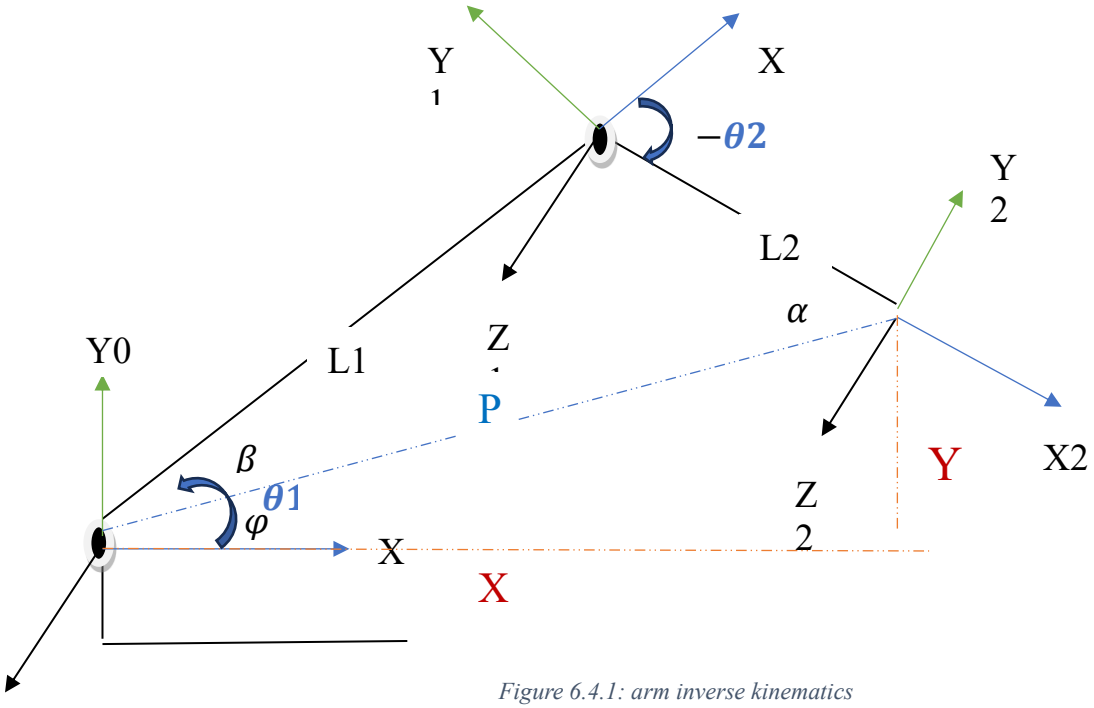
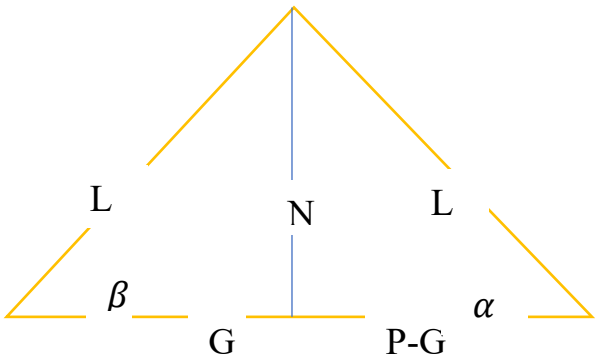


Figure 6.4.1: arm inverse kinematics



$\Theta_1 = \Phi + \beta$ $\Theta_2 = -(\alpha+\beta)$

$\varphi = tan^{-1}(\frac{Y}{X})$ $\beta = tan^{-1}(\frac{N}{G})$

$\alpha = tan^{-1}(\frac{N}{P-G})$ $P = \sqrt{X^2 + y^2}$

$L1^2 = N^2 + G^2$ (1) $L1^2 - G^2 = N^2$ (2)

$L2^2 = L1^2 - G^2 + P^2 + G^2$+2PG $x = (-L2^2 + L1^2 + P^2)/2P$

The main purpose of the arm is that it places the brush Hanging from the arm end effector on the solar panel .so we would select one model to do our calculations upon it.

Will be working on tracking panel which will be cleaned while in rest mode (0 degree) the height is 120 the width of the panel is 2.33 m, so the center of the panel Is at 1.165 cm from the edge of the panel.

The base of the arm is 1.55 m from the edge of the panel and the height of the arm base is 70 cm from the ground. given that the center of the panel is at the distance of 2.7m in x and 0.5m in y relative to the arm base.

There are many possibilities for the links lengths in this case so for now we will assume that the first link length is 2 meters, and the second link length is 1 meter. And we will check whether those lengths are valid or not.

But one can ask why we need forward kinematics in control,

Forward kinematics plays a crucial role in arm control process, by it we can calculate the actual end effector angle by taking the actual joints angles given by the potentiometers placed at every joint, and if the result of the forward

kinematics isn’t equal the reference angle value, then we control it until reaching the reference value.

6.5 Motion study

So, after determining the lengths of the links in the last section. will start the study of motion of the arm robot.

The arm has almost one task which is moving the brush and placing it on the panel. Which is not a complicated &repetitive task. So, the robot's movement will involve navigating between specific and known points, likely repetitive, to and from solar panels.

So now we can select the previously mentioned point, and using the inverse kinematics equations we can determine the angels of the joints required for each point.

Table 6.5-1: required points Inverse kinematics

point	Point description	X	Y	Theta 1	Theta 2
1: rest point start	The point where all the joints are at their minimum	0.8	0.7	20	-20
2: via point	Point in the middle of the path between start and target for safety	1.4m	0.7m	56.0518	-23.8835
3: goal point	The point where the end effector is placed on the panel	2.6m	0.5m	29.9448	-19.9217

6.6 Workspace generation

The goal of workspace generation is to find out the complete set of poses of the manipulator’s end-effector in the Cartesian space when the manipulator runs through all possible configurations in the joint space.

The working space is a parameter that directly influences the robot arm's ability to perform tasks and its kinematic capabilities. It serves as an important indicator

of the range of activities that the robot arm can effectively carry out within its operational capabilities

Robot arm and the sizes of workspace according to the design requirements, must reach the required working space, which is reasonable to judge about the structure, operability and flexibility of the robot arm.

For L1=2; L2=1;

theta1_range = linspace (20, 90, 100);

theta2_range = linspace (20, 130, 100);

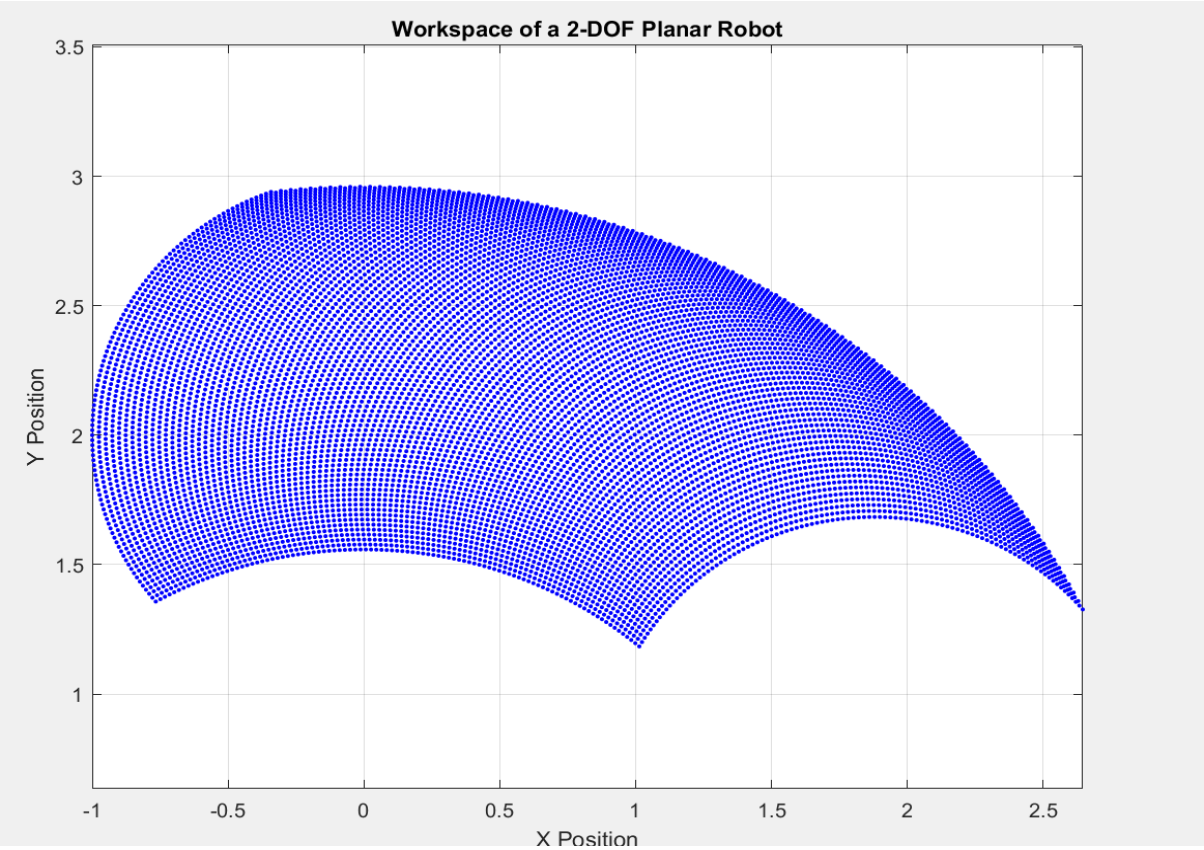


Figure 6.6.1: robot workspace

Therefore, from the above, we conclude that the lengths of links and the angles range are suitable for our application.

For a proper analysis of the performance of a robotic manipulator, computing the reachable workspace is not enough. So the next steps will be evaluating the trajectory and dynamics of the arm.

6.7 Trajectory

Trajectory planning is a crucial aspect of robotic control, as it determines the desired path, position, velocity, and acceleration of a robot's end-effector or joints over time. Effective trajectory planning is essential for ensuring smooth, efficient, and safe robot operation, allowing the robot to navigate its workspace while avoiding obstacles and achieving the desired task objectives.

In this case, we are planning the trajectory of a two-link planar robot using the relative angle between the two links, rather than the absolute joint angles. This approach offers several advantages, such as intuitive control, simplified trajectory planning, and the ability to maintain a specific end-effector orientation. By focusing on the relative angle, we can plan the trajectory in a more flexible and intuitive manner, while still providing the necessary information to perform the dynamic calculations and control the robot's motion.

Trajectory planning can be conducted either in joint-variable space or in Cartesian space. For joint-variable space planning, the time history of all joint variables and their first two-time derivatives are planned to describe the desired motion of the manipulators.

For the trajectory planning we represent the angle, and its first two derivatives using cubic polynomials.

Cubic polynomials are often chosen in trajectory planning because they provide a good compromise between simplicity and smoothness, making them suitable for many applications where smooth and continuous motion is required.

The next plot provides the trajectory planning of the arm providing required theta ,theta_dot , theta_ddot .over specific time .

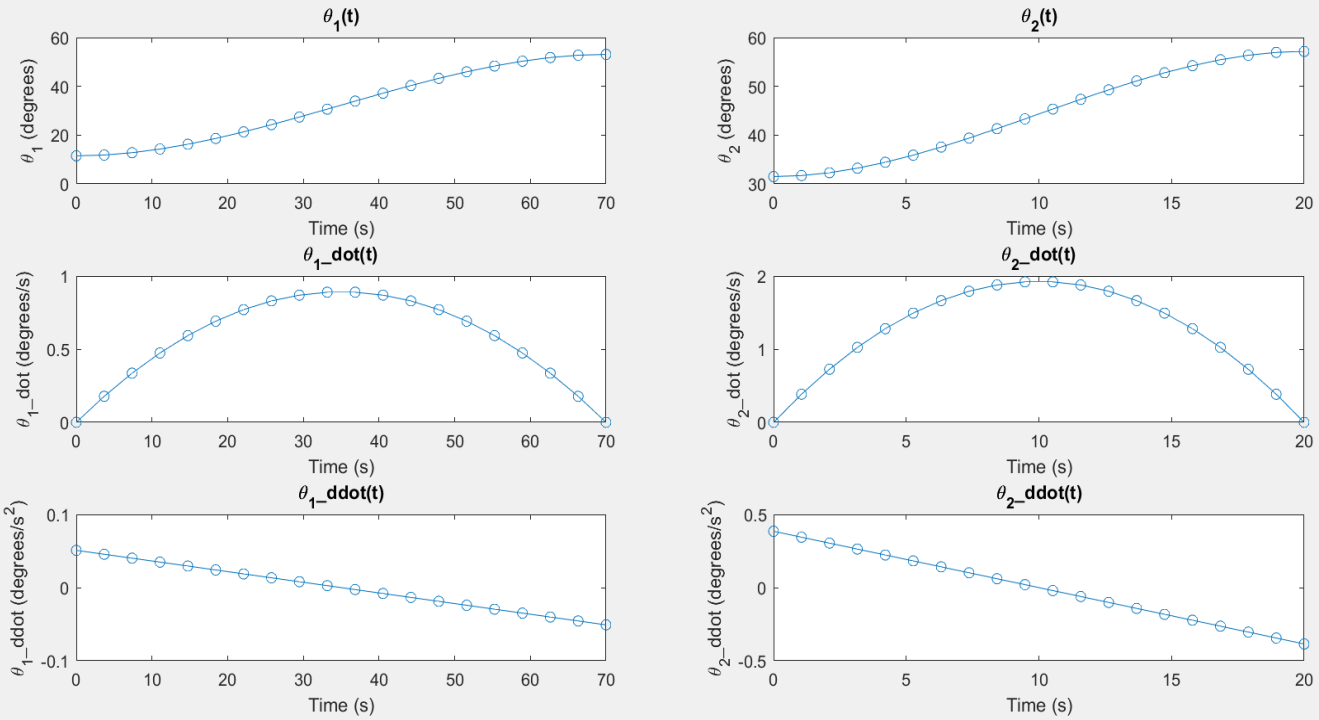
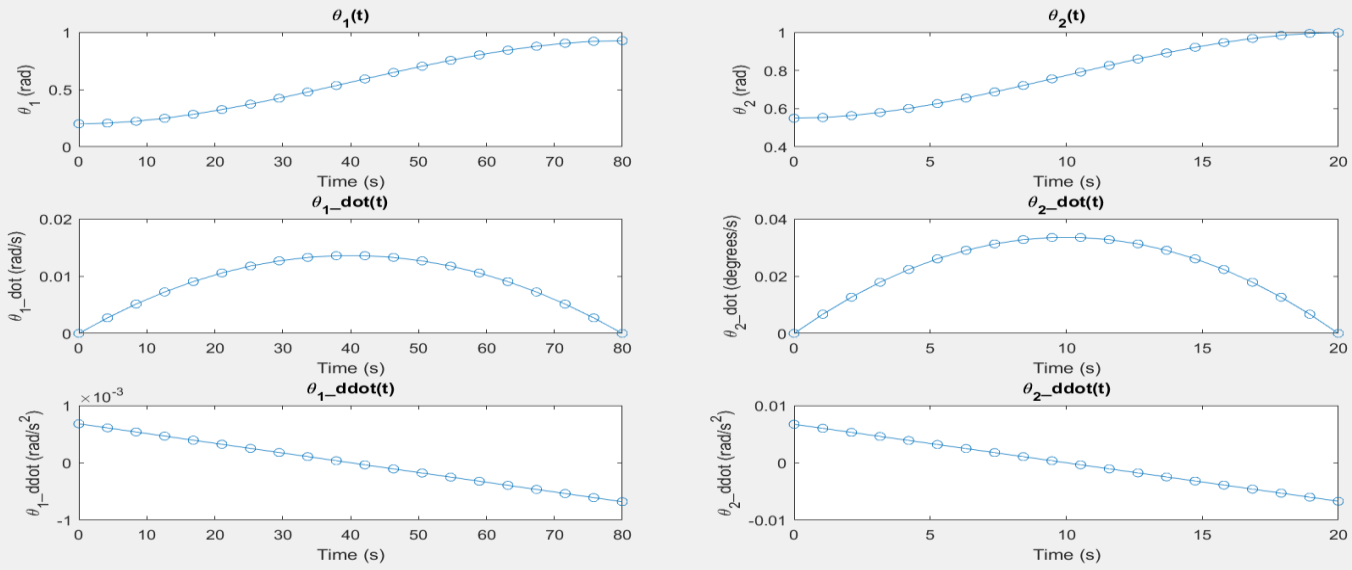


Figure 6.7.1: trajectory planning



6.8 Usage of Linear Motor in Controlling Arm Revolute Joints

In our project, we employ a linear motor to precisely control the movement of the arm's revolute joints. This section explains the mechanism and the mathematical relationship that governs this control system.

The linear motor is connected to the arm such that its linear displacement translates into rotational movement at the revolute joints. This translation is achieved through a linkage system that converts linear motion to angular displacement.

Mathematical Relationship

To understand the relationship between the linear displacement of the motor (denoted as d) and the angular displacement of the joint (denoted as θ), we derive the following relationship.

The linear motor is configured to exert a force along a line that translates to rotational motion at the joint. The displacement d of the linear motor results in an angular displacement θ of the joint. The exact relationship depends on the geometry of the linkage system.

Linkage Geometry:

Considering a simple pin-jointed linkage, the angle θ can be expressed as:

$$\theta = \frac{d}{r}$$

where r is the distance from the point of action of the linear motor to the pivot point of the joint. This relationship assumes small angular displacements where the linkage behaves almost linearly.

But as for larger angular displacements, the relationship becomes nonlinear. The cosine rule can be used to correct for larger angles:

$$\theta = cos^{-1}(\frac{a^2 + b^2 - c^2}{2 * b * a})$$

Where c is the variable length of the linear motor

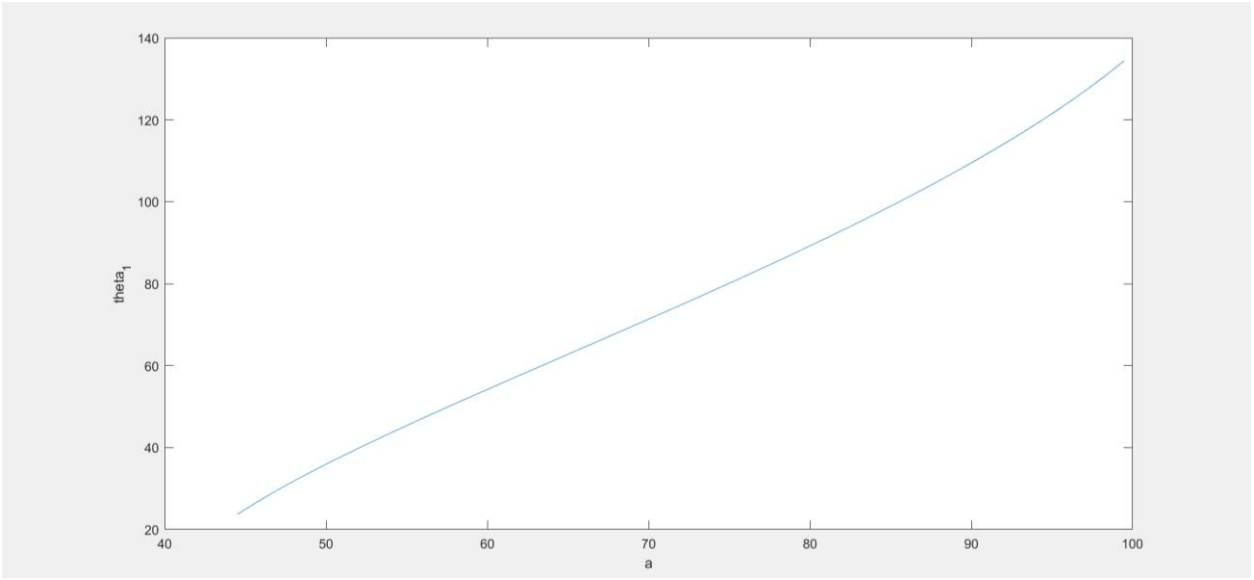


Figure 6.8.2: change of theta_1 over motor length

Will take the average of the difference between angels (angle(i)-angle(i-1)) to create a linear relation for the simplicity of calculations.

Dtheta1/dL1=2.188 deg/cm;

This cause error of 0.2

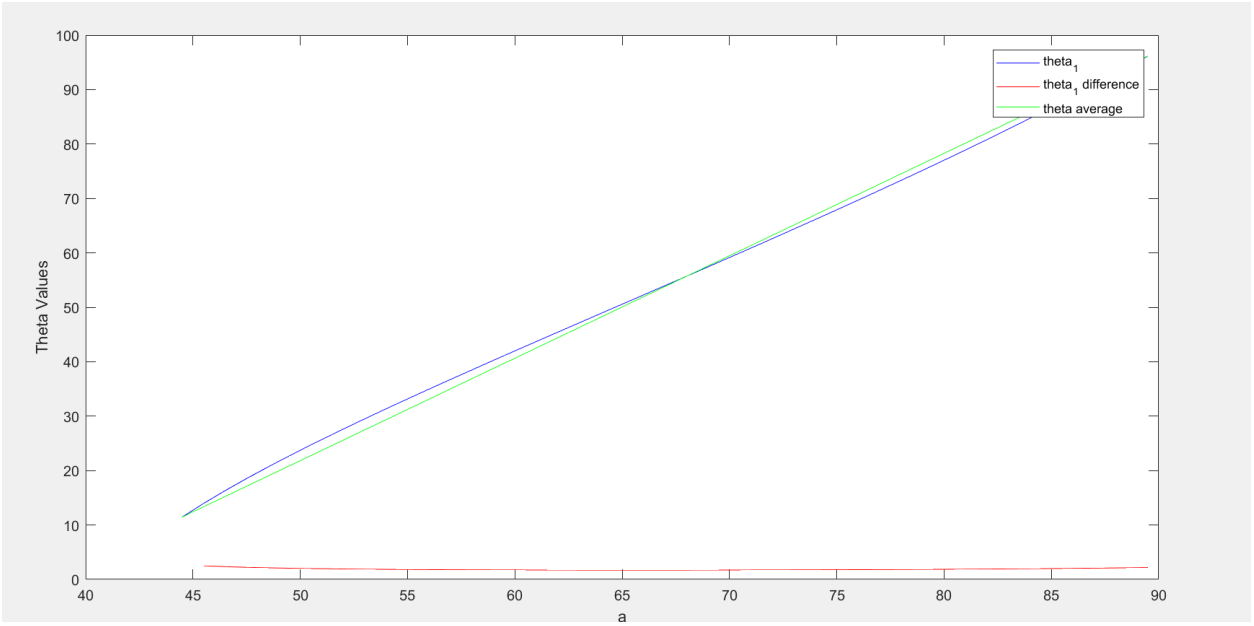


Figure 6.8.3: relation between change in the angle of the joint and the length of the linear motor for theta 1

Dtheta2/dL2=2.54 deg/cm.

This cause error of 0.22

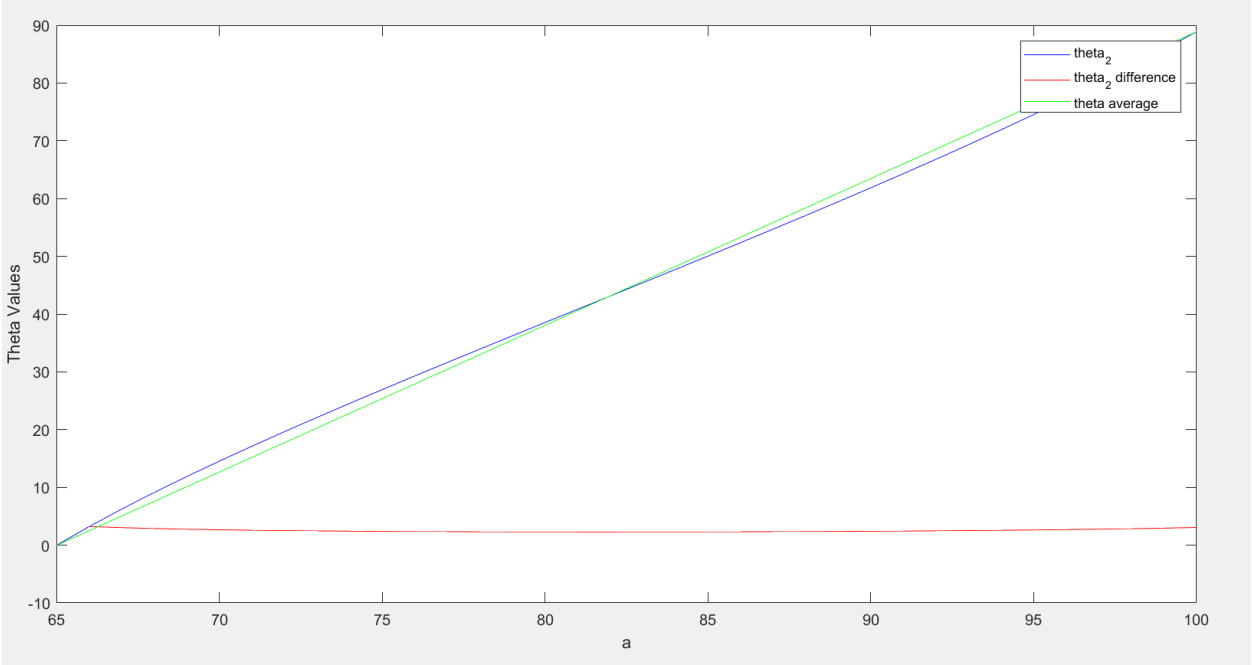


Figure 6.8.4: relation between change in the angle of the joint and the length of the linear motor for theta 2

Actual Linear motor speed &acceleration

As for the real speed and acceleration values we used visual inspection and observation. To calculate the change of the length of the linear motor. And then numerically calculate the first and second derivative.

So as for linear motor 1:

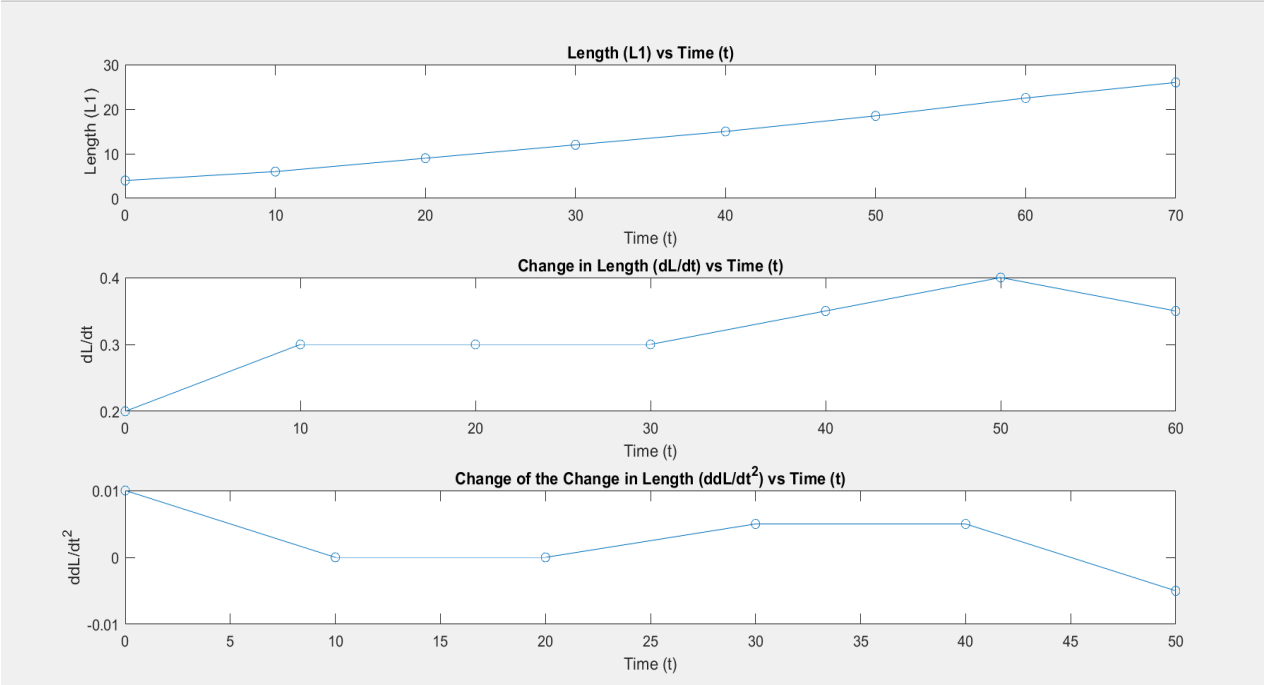


Figure 6.8.5: length, acceleration and velocity of the first motor

And as we calculated before that change of angle1 is equal to 2.188 deg/cm so:

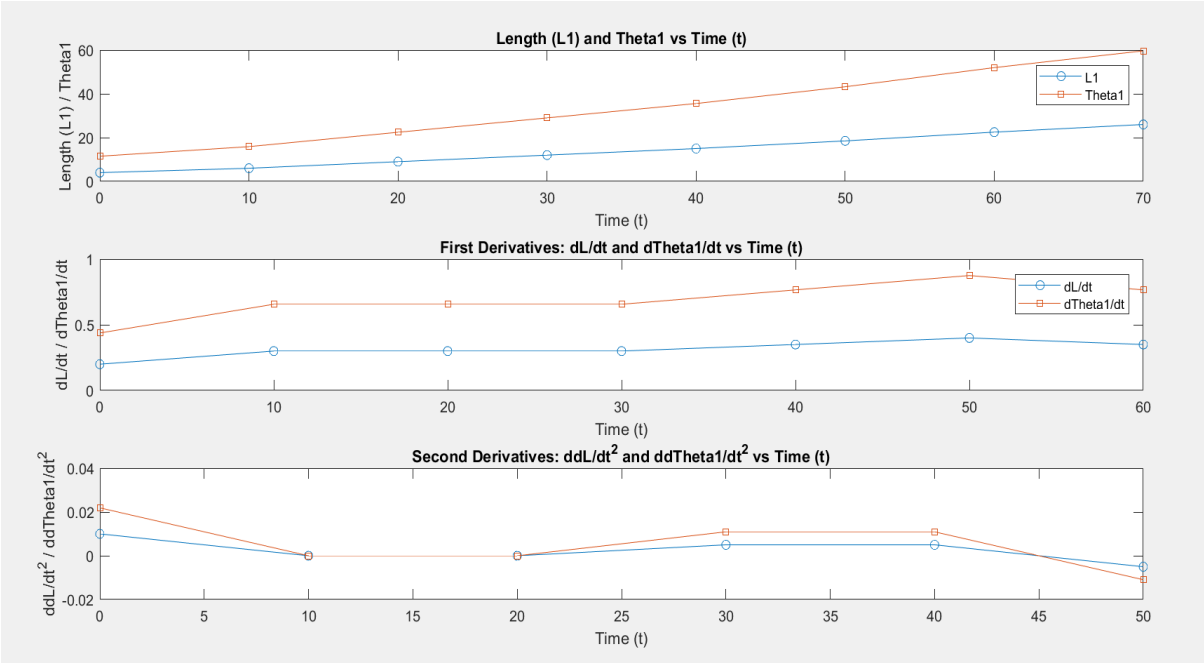


Figure 6.8.6: theta_1, dtheta_1 and ddtheta_1

6.9 Dynamics

Robot arm dynamics deals with the mathematical formulations of the equations of robot arm motion. The dynamic equations of motion of a manipulator are a set of mathematical equations describing the dynamic behavior of the manipulator.

The dynamics problems are like the study of kinematic problems, with forward dynamics problems and inverse dynamics problems. The so-called forward dynamics problem is seeking the movement of the system, given the force acted on the system. The inverse dynamics problem is the opposite given the movement of the system, find the force acting on the system at this time. Specifically, for the robot, the forward dynamics problem is that given the driving torque τ, find position of joints q, velocities of joints $\dot{q}$, and the robot joint acceleration $\ddot{q}$. For an inverse dynamics problem, for a known set of the robot joint position q, joint speed $\dot{q}$, and joint acceleration $\ddot{q}$, find the driving force τ on the robot joints currently.[4].

There are two commonly used methods for formulating dynamics, based on the specific geometric and inertial parameters of the robot: the Lagrange–Euler (L–E) formulation and the Recursive Newton–Euler (RN–E) method.

The Lagrange-Euler (L-E) formulation and the Recursive Newton-Euler (RN-E) method are both pivotal in understanding the dynamic behavior of robotic systems, yet they serve distinct purposes due to their methodological differences. The L-E formulation employs Lagrangian mechanics to derive exact solutions for robot dynamics, ensuring precision in modeling and analysis, albeit at the cost of computational complexity, particularly with higher degrees of freedom. In contrast, the RN-E method prioritizes computational efficiency by iteratively calculating joint torques and forces, making it ideal for real-time applications such as control algorithms and dynamic simulations. While both approaches are equivalent in describing robot motion dynamics, their suitability depends on the specific needs of the application, balancing accuracy and computational feasibility in robotics research and development.

In our case we choose to work with newton Euler method.

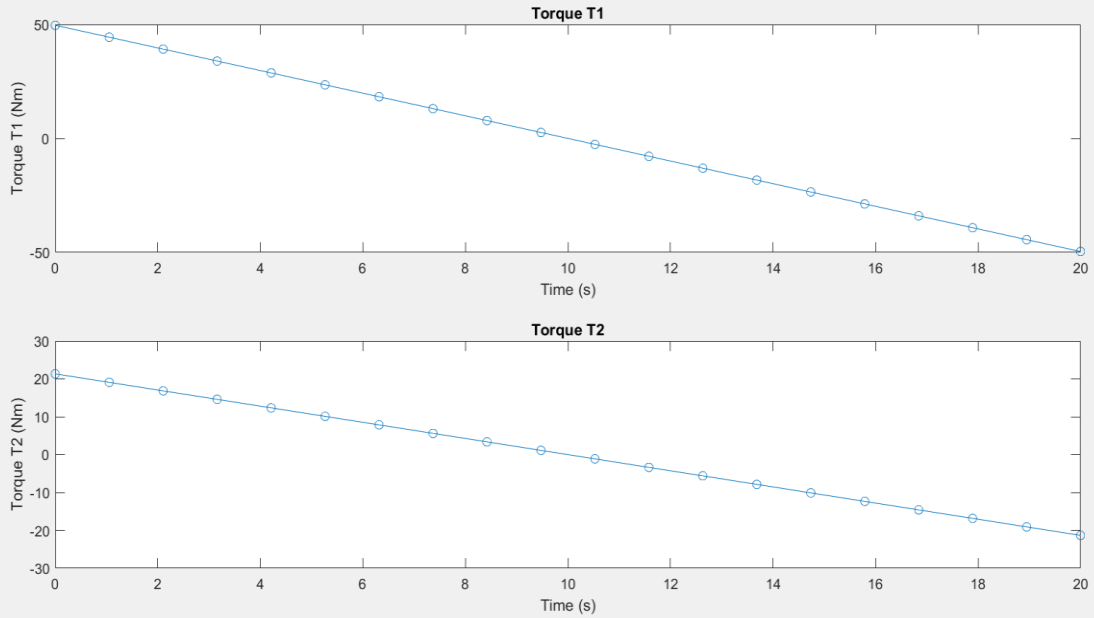


Figure 6.9.1: torque on joint1 and joint2

The previous plot represents the torque on the robot joints duo to values of theta, theta_dot, theta_ddot coming from trajectory planning

Chapter 7

ACTUATORS CONTROL

7.1 INTRODUCTION

The actuator control level is the foundational layer of the control architecture for the robot. This level is responsible for the direct manipulation and control of all mechanical actuators in the robot, ensuring precise and responsive movement.

It is crucial for executing basic tasks and enabling higher-level functionalities, such as arm positioning and navigation.

We have three types of actuators in our robot:

1. vehicle Motors,
2. arm linear motors,
3. and brush motor.

7.2 Vehicle Motors Controller:

Manage the speed and direction of the motors driving the robot's tracks. For achieving this task, we have various types of controllers, these are some of them:

- **Fuzzy Logic Controllers**

  Use fuzzy logic instead of binary logic to handle uncertainties and approximate reasoning, making decisions based on a set of rules. It is used for complex systems with uncertain dynamics, such as robotics and automotive applications.

  - **Advantages**: Robust to uncertainties, handles non-linear systems well.

- **Adaptive Controllers**

  Adjust motor parameters in real-time to adapt to changing conditions and maintain optimal performance. It is Ideal for systems with varying dynamics, such as aerospace and adaptive robotics.

  - **Advantages**: Self-tuning, handles changing environments.

- **Model Predictive Controllers (MPC)**

  Use a model of the system to predict future states and optimize control actions over a finite time horizon. Used usually for complex systems with constraints, such as process control and autonomous vehicles.

  - **Advantages**: Handles multivariable systems and constraints.

7.2.1 PID Controller

Introduction to PID Control

PID (Proportional-Integral-Derivative) control is a widely used feedback control technique in control systems. It combines three different control strategies - proportional, integral, and derivative - to adjust the control input in a manner that minimizes the error between a desired setpoint and the actual output of a system

Proportional (P) Control

Proportional control is the simplest form of control. The control signal is proportional to the current error value.

u(t)=k_pe(t)

- **Advantages**: Simple to implement, quick response to changes in error.
- **Disadvantages**: Cannot eliminate steady-state error; may cause oscillations if k_p is too high.

Integral (I) Control

Integral control sums the past errors to eliminate steady-state error. The control signal is proportional to the accumulated error over time.

u(t)=$k_i \int_0^t e(t)$dt

- **Advantages**: Eliminates steady-state error.
- **Disadvantages**: Can cause slow response; may introduce overshoot and oscillations if k_i is too high.

Derivative (D) Control

Derivative control predicts future error based on its rate of change. The control signal is proportional to the rate of change of the error.

$u(t)=k_d \frac{de(t)}{dt}$

- **Advantages**: Improves system stability and response; reduces overshoot.
- **Disadvantages**: Sensitive to noise; can amplify high-frequency noise in the signal.

PID Control

A PID controller combines all three strategies to leverage their individual strengths while mitigating their weaknesses.

$u(t)=k_p e(t) + k_i \int_0^t e(t)dt + k_d \frac{de(t)}{dt}$

- u(t) is the control signal.e(t) is the error (the difference between the desired setpoint and the actual output).
- k_P, k_I **and** $k_D$are the proportional, integral, and derivative gains,

PII fer:

ent

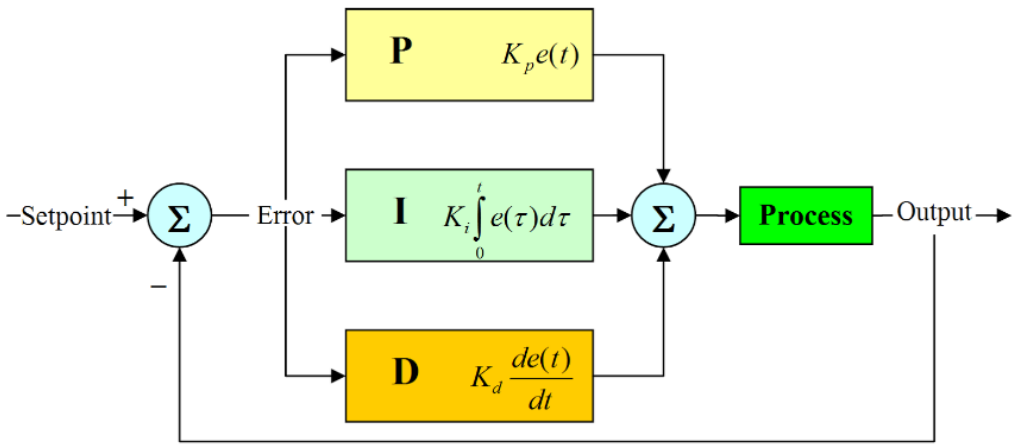


Figure 7.2.1: PID Block Diagram

2. **Versatility:** Applicable to a wide range of control problems, including those with time delays and non-linearities.
3. **Robustness:** Maintain stable and accurate performance despite disturbances and system changes.
4. **Elimination of Steady-State Error:** Integral term helps ensure the system output reaches and stays at the desired setpoint.
5. **Improved Stability and Response:** Derivative term predicts future errors, reducing overshot and enhancing stability.

- **Disadvantages**: More complex to tune; requires careful balance of and k_p, k_i and k_d

- **Challenges**

Lack of Motor Information: One significant challenge faced during this project is the absence of detailed specifications for the DC motors. Manufacturer-provided motor dynamics information, such as transfer functions or Bode plots, is typically essential for accurate PID controller tuning. Without this data, it becomes difficult to understand the motor's behavior under various operating conditions, leading to several issues:

- **Uncertain Dynamics**: The precise dynamic response of the motor is unknown, making it challenging to predict how the motor will react to control inputs.
- **Tuning Difficulties**: Accurate PID tuning requires knowledge of parameters like inertia, damping, and natural frequency, which are not readily available.
- **Risk of Instability**: Inadequate tuning might result in poor performance, such as overshooting, oscillations, or even instability in the control system.

7.2.2 Tuning the PID Parameters

Tuning a PID controller involves finding the optimal values for k_p, k_i and k_d . For this project, we used MATLAB's PID Tuner Toolbox and System Identification tool due to the unknown motor characteristics.

- **Tuning Process:**

1. **System Identification:** The process started with the identification of the motor's dynamic behavior using MATLAB's System Identification Toolbox. Given the motor's characteristics were unknown, a set of input-output data was collected from the motor.

   A. Data Collection Using ROS:

   - The motor's dynamic behavior data was collected using ROS (Robot Operating System).
   - The ROS master was initialized and connected to the appropriate IP address.
   - Data was collected from the '/rwheel_vel' and '/PWM' topics, which provided the right wheel velocity and motor command (PWM) information.
   - MATLAB provides an ROS Toolbox that facilitates interaction with ROS-enabled devices and systems. The ROS Toolbox in MATLAB allows users to:

   - **Connect to a ROS network:** Initialize the connection to a ROS master using the 'rosinit' function.
   - **Subscribe to topics:** Create subscribers to specific topics to receive messages. For example, the 'rossubscriber' function allows you to subscribe to topics like '/rwheel_vel' and '/PWM'.
   - **Collect data:** Use functions like receive to gather data from the subscribed topics. The data can then be stored for further analysis.
   - **Publish messages:** Send control commands or other information back to the ROS network using the 'rospublisher' and send functions.
   - **Visualize and process data:** Utilize MATLAB's powerful data visualization and processing capabilities to analyze the collected data.

B. System Identification:

- The data collected using ROS was used to fit a mathematical model to the motor's dynamics.
- The collected data was converted into an ‘iddata’ object for system identification.
- A transfer function model was estimated from the data using ‘tfest’.
- Initially, first-order then a second-order model was considered, but ultimately a third-order model was chosen based on better fit and analysis.
- The PID Tuner was used to tune the PID controller for the estimated system.

By following these steps, we effectively collected, analyzed, and used the data to model and control the motor system

C. Model Structure:

The data collected using ROS was then used to fit a mathematical model to the motor's dynamics. From the system identification process, we obtained an underdamped third-order transfer function by these steps .

i. .Comparing the Model with Data:

The estimated model was compared with the collected data to validate its accuracy.

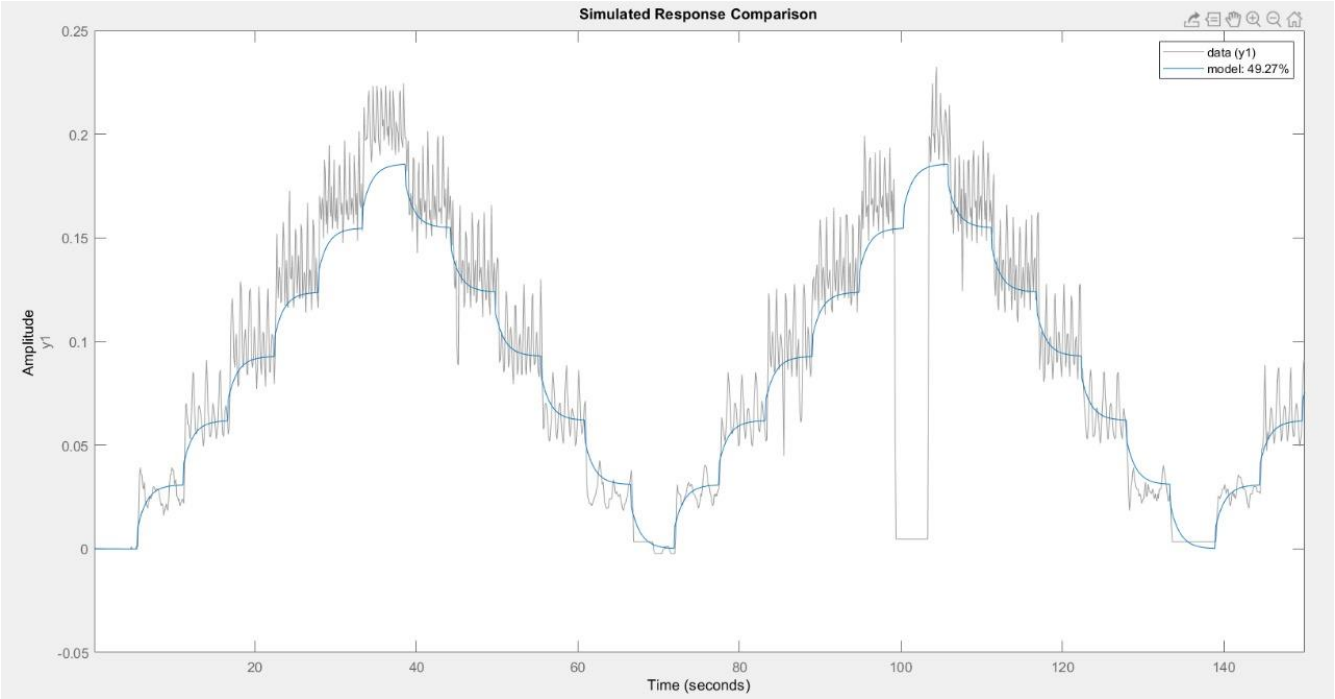


Figure 7.2 first order model data comparison

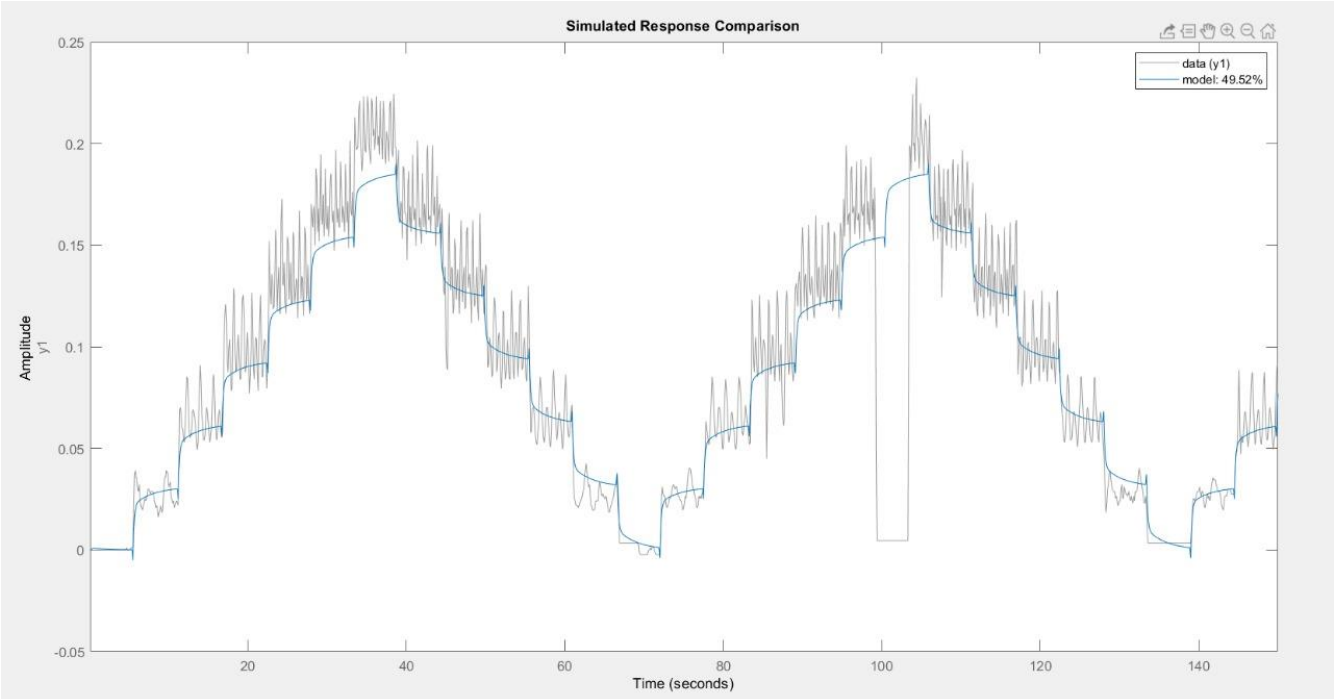


Figure 7.3 second order model data comparison

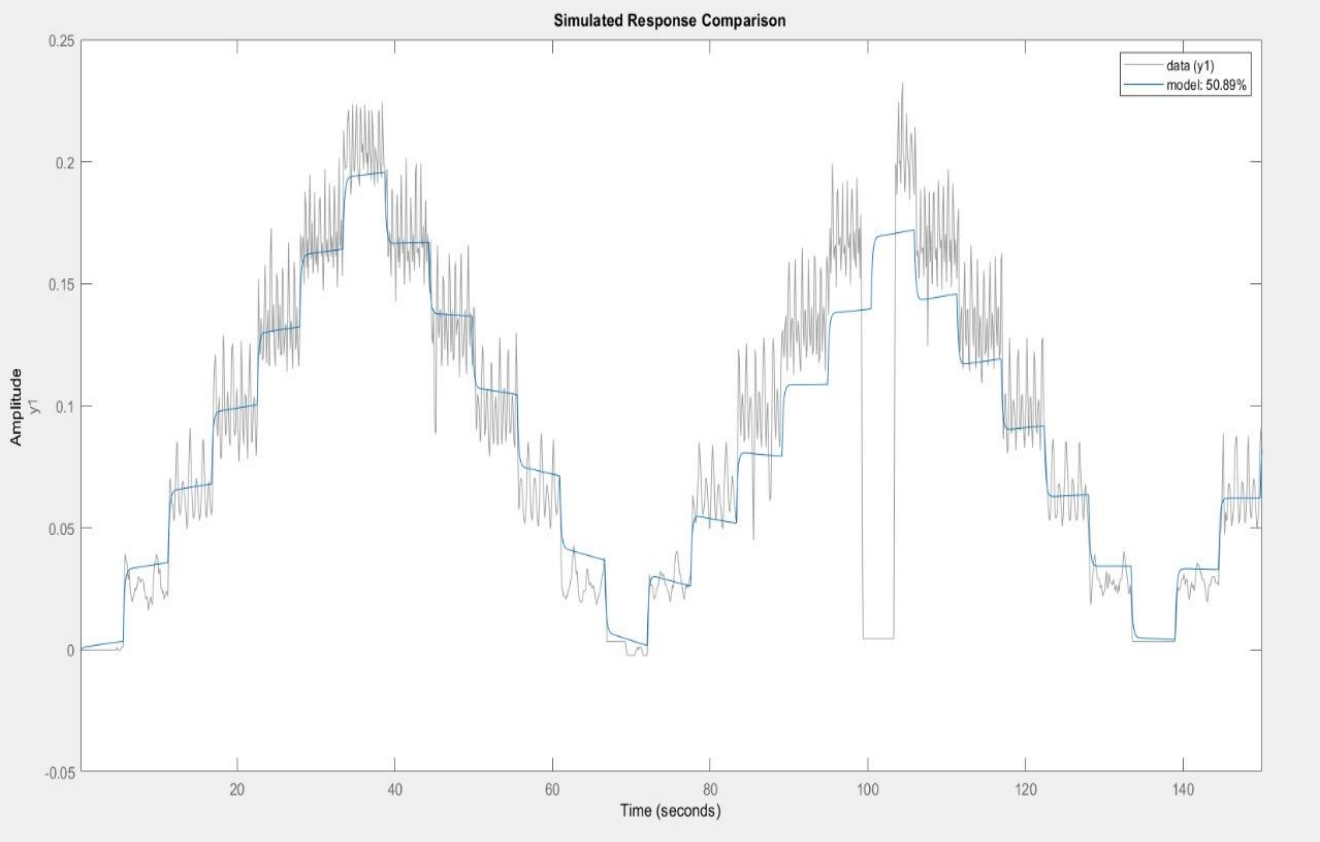


Figure 7.4 third order model data comparison

2. Analysis of the System:

   After estimating and comparing both second order and third-order models, the third-order model was chosen due to its better representation of the system's dynamics.

   The transfer function of the third-order model is:

```
tf_model =

  From input "u1" to output "y1":
  0.006696 s^2 + 0.0001367 s + 1.673e-05
  ---------------------------------------
  s^3 + 5.738 s^2 + 0.09438 s + 0.01367
```

2. **PID Tuner Toolbox:**
   - With the identified plant model, MATLAB's PID Tuner Toolbox was used for tuning the PID controller.
   - The toolbox provides a user-friendly interface to tune the PID parameters by adjusting sliders or entering values directly for k_p, k_i and k_d
   - The PID Tuner automatically optimizes these parameters to achieve the desired balance between responsiveness and stability.
3. **Tuning Results:**
   - The step response plot shows how the tuned PID controller tracks the reference input. The graph demonstrates an initially rapid response followed by slight oscillations and eventual steady-state achievement.
   - The tuned response exhibits underdamped behavior, characterized by a faster response and minimal overshoot.
   - The PID parameters obtained were:

k_p= 224.4, k_i=2570 , k_d=4.899

S

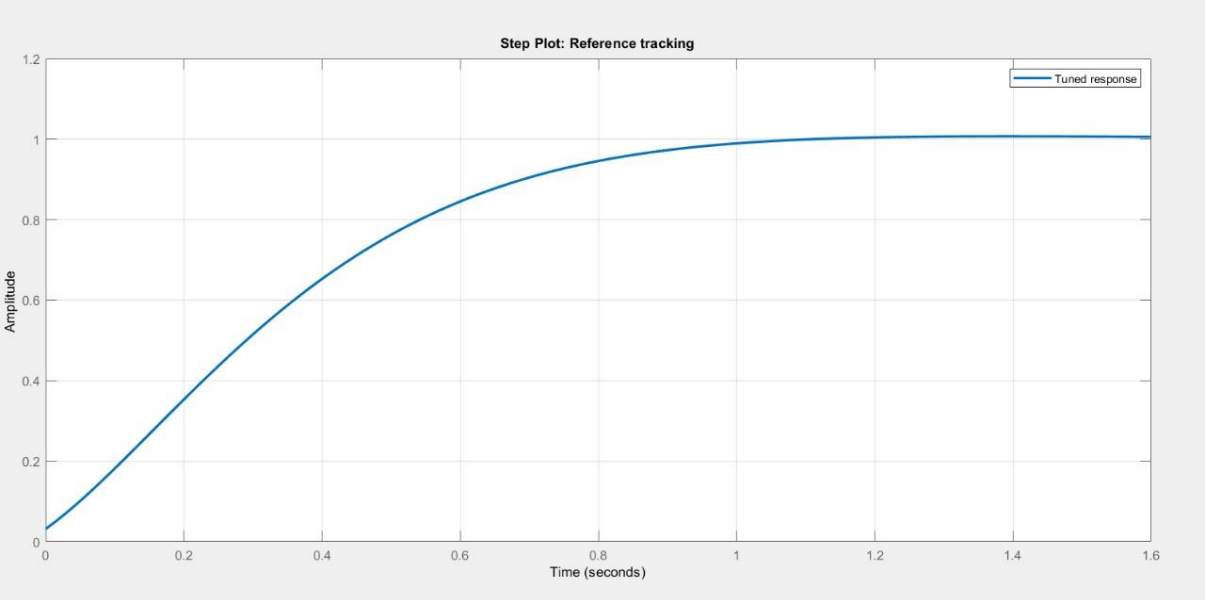


Figure 7.5 plant response after tuning

With an input of 125 PWM, the system reaches the steady state faster, taking approximately 1.2 seconds to settle. This improved response time demonstrates the effectiveness of the tuning process and the system's ability to quickly adapt to changes in input.

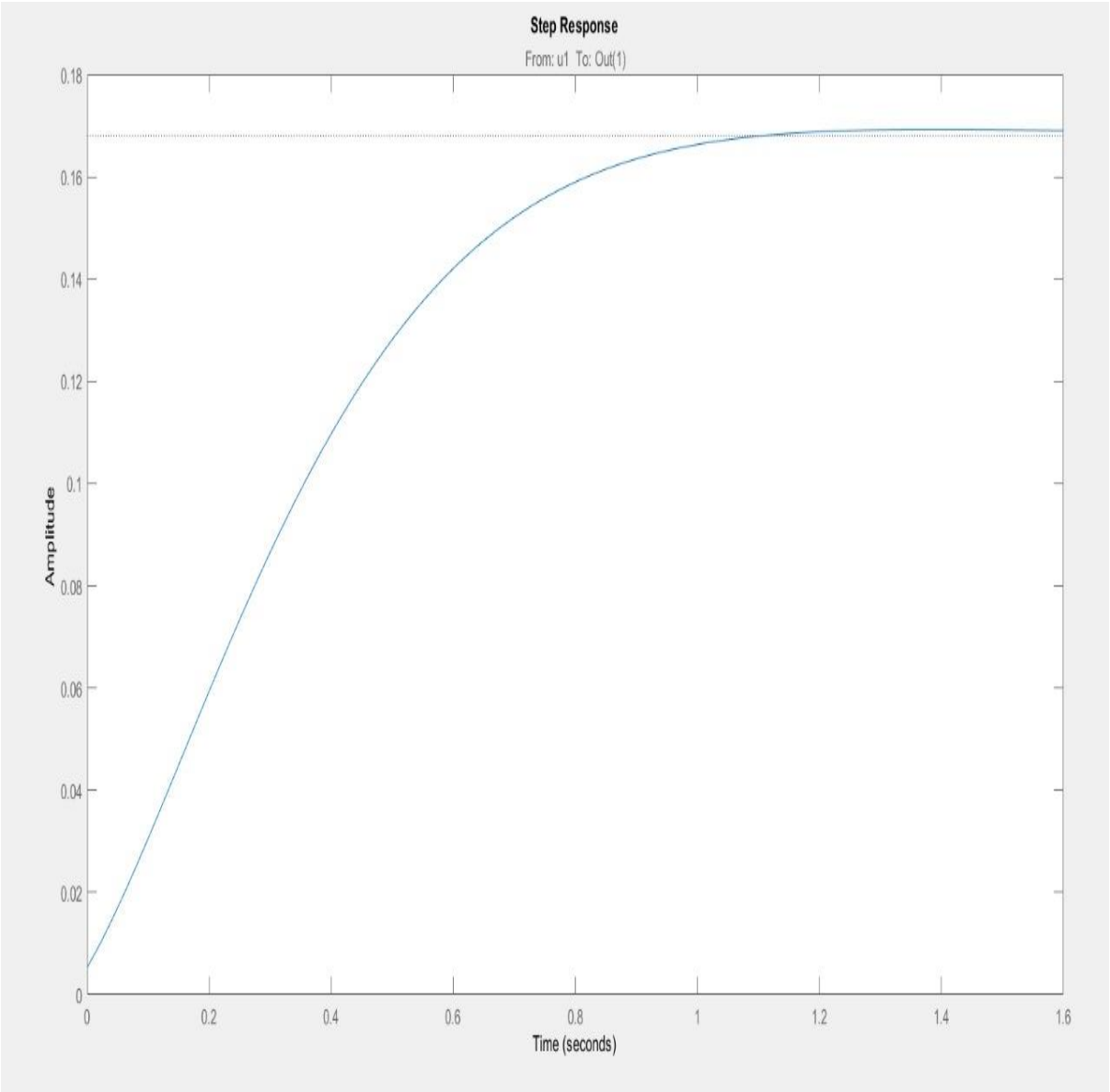


Figure 7.7 Step Response with 125 PWM Input

We applied the same procedure to the second motor, resulting in a mathematical model and a tuned system. The parameters of the PID controller for the second motor are as follows:

k_p= 224.4, k_i=2570 , k_d=4.899

```
sys =

  From input "u1" to output "y1":
   0.00242 s + 0.000585
  ----------------------
  s^2 + 2.261 s + 0.5664
```

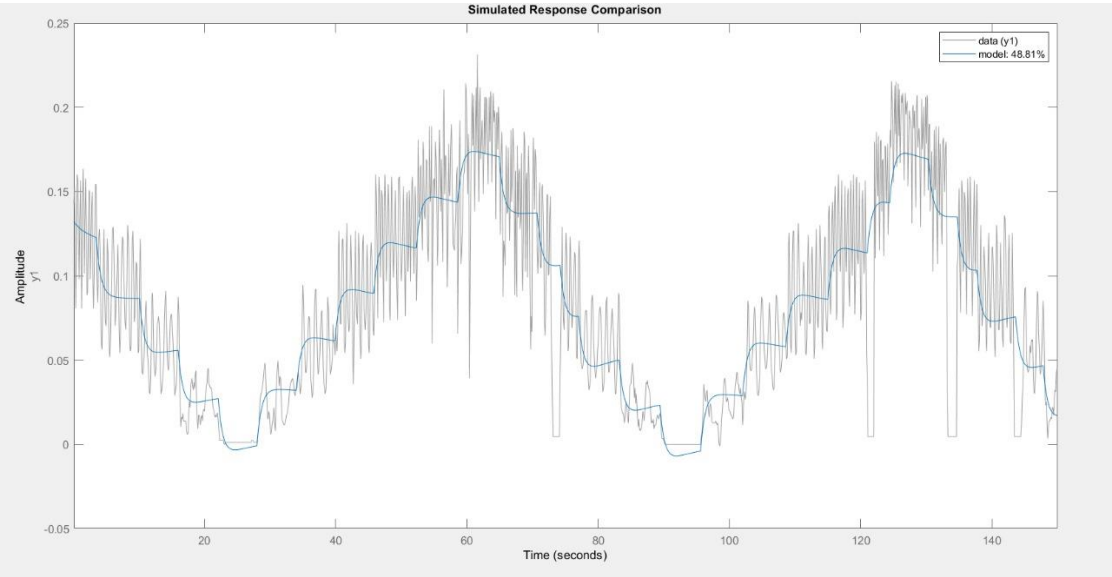


Figure 7.8 third order model data comparison

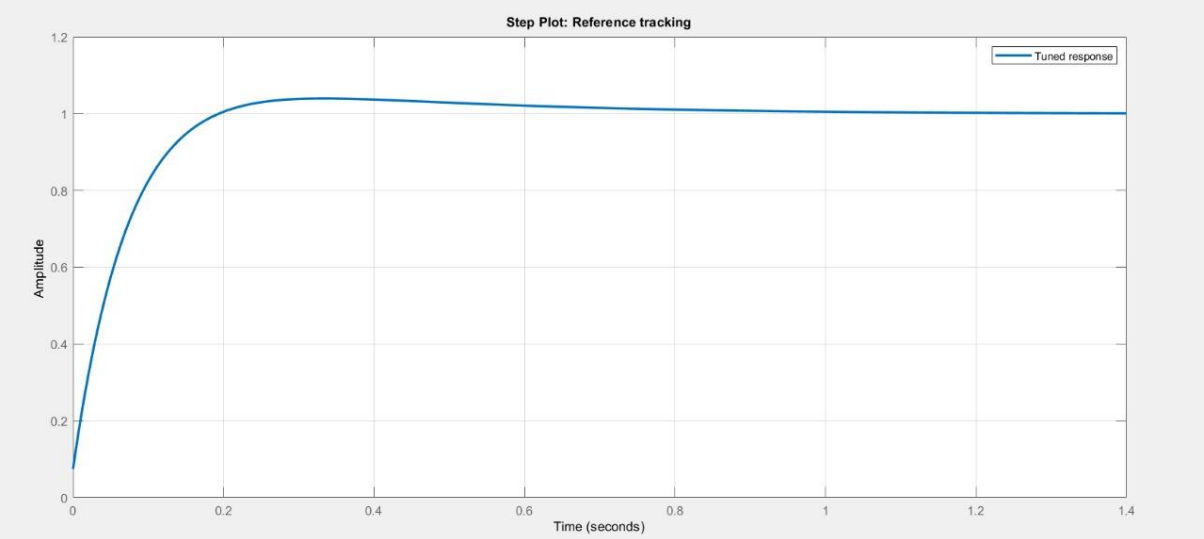


Figure 7.9 Tuned response

7.3 Arm linear motors:

Control the angle of joints used in the robotic arm and ensure precise and responsive control of the linear motors.

The control of arm motors is carried out in two ways.

- The first way is to semi autonomously adjust the angels of the arm using the mobile application. By adjusting the gauge of both base, and elbow joints angle until the actual value reaches the desired angles.

- The second one is used when the robot is in his automatic mode.

Figure 7.3.1: mobile application arm joints control

The arm will be suspended to a repetitive motion path through a predefined point. this approach will work as follows:

once the robot base (vehicle) has reached the required position Infront of the solar panel or solar panel row, then ROS will send a signal to Arduino to start its predefined path whether this signal comes from the application or from the path algorithm. Arduino starts to compute the required angle set for this path.

The motors are connected to power through a relay. A relay is an electrically operated switch used to control the flow of electrical power in a circuit.

It allows a low-power control signal to control a higher-power circuit. Each channel has 3 pins (NO, NC, Common). When the enable of the channel is high the relay switch positions the normally open is now closed and the closed one is open.

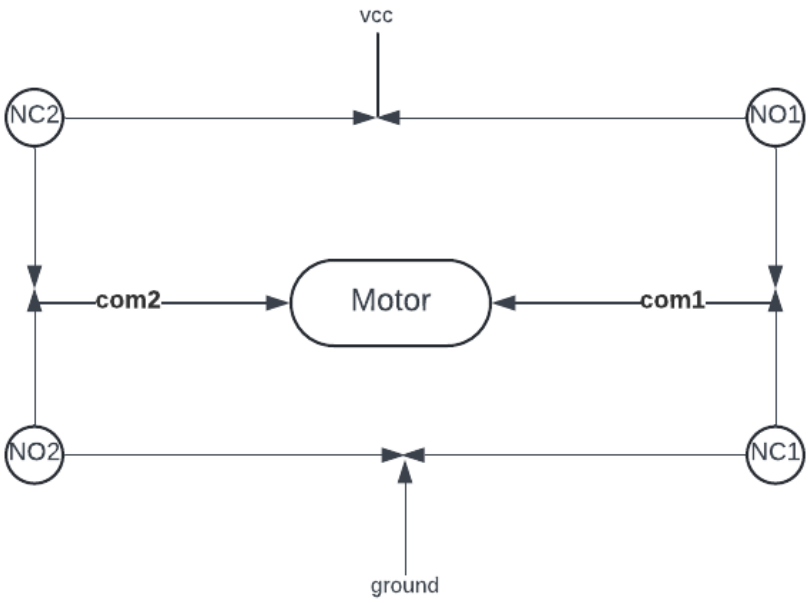


Figure 7.3.3: linear motor relay control

As shown in figure 7.3.3, the motor is connected in an H_Bridge circuit manner. The motor is controlled using two relays' channels, so if both are activated the motor is working forward. if both of it are deactivated the motor works backward.to stop the motor we need to activate one channel only.

7.3.1 Real-time Feedback Monitoring

Arm motors are activated using Arduino signals, so for Arduino after computing the desired angle needed for each point it starts to move the arm to reach the desired angle at manipulator. The work manner for this part is that the Arduino gives signal to the relay to power the motor for a certain amount of time that enough for the joint to reach the desired angle. and get feedback of the real-time angle using a potentiometer.

This work is like a servo motor control. The potentiometer is an adjustable resistance connected to the joints of the robot. Every time the joint rotates it changes the potentiometer value. so do the voltage passing through it.

So, we connect the potentiometer output to the Arduino to read the volte passing through and determine the current angle.

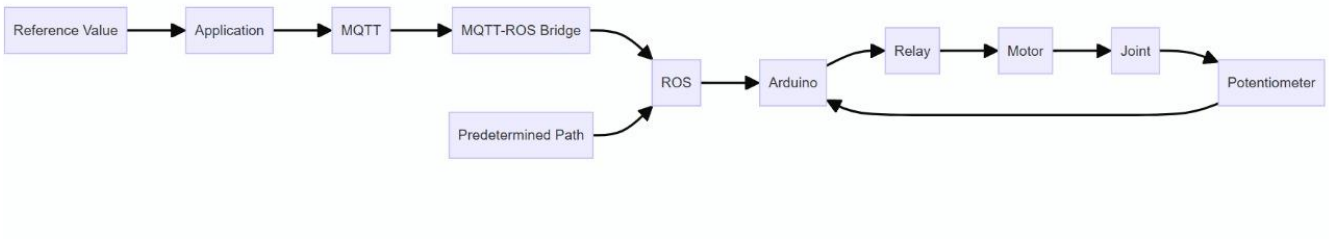


Figure 7.3.4: Arm control flow diagram

7.4 Pid control of arm robot

As is the case with the dc motors, the absence of detailed specifications and dynamic specifications for the DC linear motors. is a big challenge in the process of controlling the motor. So, we would just deal with a relation between output(angle)and input (length or position of linear actuator).

The relation between the angle and the actuator length is originally nonlinear.

But as clarified in the previous chapter we will linearize this relation for simplicity.

the goal is to achieve precise control of the actuator length (or position) based on the feedback from the potentiometer.

The PID output typically adjusts the voltage or current supplied to the DC linear motor through the motor driver.

Monitor the feedback from the potentiometer continuously to adjust the output and maintain the actuator at the desired position.

1. **Purpose and Function of Each PID Component (P, I, D)**

Proportional (P) Control:

- Purpose: Adjusts the motor output proportionally to the current error (difference between desired and actual position)

- Impact: Provides immediate response to error but can cause overshoot or oscillations if (Kp) is too high.

Integral (I) Control:

- Purpose: Integrates the error over time to eliminate steady-state error and improve system response.

- Impact: Corrects for long-term error accumulation, enhancing precision and stability.

Derivative (D) Control:

- Purpose: Predicts the system's future behavior based on the rate of change of error, dampening oscillations and improving stability.

- Impact: Provides damping effect, reducing overshoot and improving transient response.

2. Utilization of Feedback from Potentiometer

- Feedback Mechanism: Potentiometer provides continuous feedback on the current position of the DC linear motor.

- Implementation: The feedback signal is compared with the desired position (setpoint) to compute the error, which is then used as input to the PID controller.

- Role: Enables the PID controller to continuously adjust the motor output to maintain the actuator at the desired position despite disturbances or variations.

3. Linearization Process and Impact on Control Simplicity

- Original Nonlinearity: The relationship between angle and actuator length was originally nonlinear.

- Linearization Approach: Simplified this relation to a linear approximation within the operating range of the arm robot.

-Impact: Facilitates easier implementation and tuning of PID control.

- Advantages: Reduces complexity in controller design and tuning, making the system more robust and easier to understand.

4**. Challenges Encountered and Addressed During PID Tuning**

- Initial Overshoot: Addressed by reducing the proportional gain (Kp) and adjusting the derivative gain (Kd) to dampen oscillations.

- Steady-State Error: Tuned the integral gain (Ki) to eliminate remaining steady-state errors.

- Nonlinear Behavior: Mitigated by careful selection of the linearization range and validating the linear approximation through experimentation.

- Integration with Mechanical Dynamics: Considered the inertia and friction of the arm robot's joints, adjusting PID parameters to optimize performance.

7.5 Brush motor control:

Precise control for brush motor isn’t required for our application to reduce costs and computational resources we just perform open loop control for both speed and direction of it.

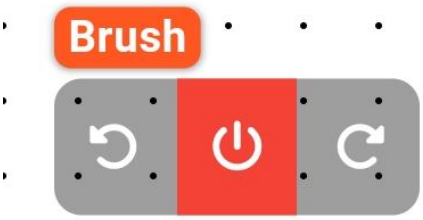


Figure 7.5.1: mobile application brush control

Like the arm motors, the brush motor control can be controlled semi autonomously through the mobile application by controlling the operating of it and the direction of the brush.

It can be controlled also fully autonomously when the ultra-sonic sensor placed on the brush detects the surface of the panel.

Chapter 8

REMOTE CONTROL AND MONITORING

8.1 INTRODUCTION

1. **Remote control**

As we mentioned in the control section, we have two approaches for motion and navigation control, first is fully autonomous navigation and second is semi-autonomous navigation.

We need the last approach for:

- building a map for the first time.
- making a specific motion or task.
- And other …

We need remote control not just for motion control but also for arm motion control and brush control.

2. **Remote monitoring**

For safety aspects and other aspects, we need to monitor various data.

By providing real-time data on the:

- Robot's speed.
- Robot location.
- and the environment map generated using the SLAM algorithm as we explained before.
- And other data like the surrounding temperature and robot state.

so that operators can make informed decisions and promptly address any issues that arise. This continuous flow of information allows for proactive management, reducing time and preventing potential damage to the solar panels or the robot itself.

8.2 IoT (Internet of Things)

IoT represents a technology paradigm where everyday objects are interconnected through the internet, enabling them to send and receive data. This connectivity facilitates advanced automation and control.

In the context of robotics, IoT allows for integration and communication between robots and other devices, improving their functionality, efficiency, and user interaction.

In our robot, we need IoT to achieve the remote control and monitoring mentioned above. With the aid of IoT, the robot can communicate with a mobile application, providing real-time data and receiving commands from users located anywhere in the world.

8.2.1 Different Network Protocols Used for IoT

Network protocols are the backbone of IoT communication, enabling devices to connect, share data, and interact with each other. Various network protocols are tailored to meet the specific requirements of different IoT applications, ranging from short-range, low-power communication to long-range, high-bandwidth connectivity.

- **Wi-Fi (Wireless Fidelity)**

High-speed wireless communication standard based on IEEE 802.11. Suitable for high-bandwidth applications like smart home devices, video streaming, and industrial automation.

  - **Advantages**: High data rates, widespread availability, easy integration with existing infrastructure, and strong security features.

- **Bluetooth**

Short-range wireless communication standard based on IEEE 802.15.1. It is ideal for personal area networks (PAN), such as wearable devices, health monitors, and smart home products.

  - **Advantages**: Low power consumption (especially with BLE - Bluetooth Low Energy), easy to use, ubiquitous, and capable of device-to-device communication.

- **Zigbee**

Low-power, low-data rate wireless communication protocol based on IEEE 802.15.4. Suitable for home automation, smart energy management, and industrial control systems.

  - **Advantages**: Low power consumption, mesh networking capabilities, secure communication, and scalability for large networks.

- **LoRaWAN (Long Range Wide Area Network)**

Low-power, long-range wireless communication protocol designed for wide-area networks. Ideal for applications requiring long-range communication and low power usage, such as smart agriculture, asset tracking, and environmental monitoring.

  - **Advantages**: Long-range connectivity (up to several kilometers), low power consumption, scalability, and support for many devices.

- **LPWAN (Low Power Wide Area Network)**

Technology standardized by 3GPP. Suitable for applications requiring deep indoor coverage and long battery life, such as smart metering, environmental monitoring, and asset tracking.

  - **Advantages**: Excellent coverage, low power consumption, high network capacity, and support for existing cellular infrastructure.

- **Sigfox**

Low-power, wide-area network technology designed for IoT. It is the best for low-bandwidth applications like smart city sensors, asset tracking, and environmental monitoring.

  - **Advantages**: Ultra-low power consumption, long-range communication, simple and cost-effective implementation, and global coverage.

- **Cellular (2G, 3G, 4G, 5G)**

Wide-area network technology using existing cellular networks. Suitable for applications requiring high mobility and wide-area coverage, such as connected vehicles, smart cities, and industrial IoT.

  - **Advantages**: Extensive coverage, high data rates (especially with 4G and 5G), reliable connectivity, and support for high mobility

In our application we use Wi-Fi to achieve relatively wide range and high data rate and for provide the possibility of streaming a video by a camera placed on the robot as a future feature.

8.2.2 Different Communication Protocols Used for IoT

There are numerous communication protocols to achieve IoT principles like:

- **MQTT (Message Queuing Telemetry Transport)**

Lightweight, publish-subscribe model. Suitable for applications requiring efficient data transmission with low bandwidth, such as remote monitoring and control.

  - **Advantages**: Low latency, scalability, support for Quality of Service (QoS) levels, secure communication, and ease of implementation.

- **CoAP (Constrained Application Protocol)**

A protocol designed for constrained devices. It is ideal for low-power, low-bandwidth environments, such as smart home devices and industrial sensors.

- **Advantages**: Efficient communication, support for multicast, simple to implement, and integrates well with HTTP-based systems.

- **Web Sockets**

Full-duplex communication channel over a single TCP connection. Used for real-time applications requiring low-latency communication, such as live data feeds and interactive applications.

- **Advantages**: Low overhead, real-time bidirectional communication, and compatibility with web browsers.

- **XMPP (Extensible Messaging and Presence Protocol)**

Open-standard communication protocol for message-oriented middleware. It is suitable for real-time communication, such as instant messaging and presence information.

- **Advantages**: Extensible, secure, supports various communication patterns, and provides real-time presence information.

In our application we use MQTT protocol.

8.3 MQTT (Message Queuing Telemetry Transport) Protocol

MQTT is a communication protocol designed for low-bandwidth, high-latency, or unreliable networks, MQTT is particularly well-suited for IoT applications where minimal overhead and real-time data transmission are essential. MQTT protocol operates on a publish-subscribe model, which is -by the way- the same communication pattern that ROS uses for manipulating data among nodes over topics.

MQTT has various advantages like:

- **Low Bandwidth Consumption**: MQTT is designed to minimize bandwidth usage, making it suitable for networks with limited capacity.
- **Low Latency**: MQTT provides low-latency communication, ensuring that messages are delivered promptly. This is essential for real-time applications, such as remote control and monitoring of the robot.

- **Scalability**: MQTT protocol can handle numerous clients simultaneously, making it highly scalable. This allows for easy expansion of the system, if there are more than one robot in the plant.
- **Quality of Service (QoS) Levels**: MQTT supports different QoS levels, ensuring reliable message delivery based on application requirements
- **Security Features**: MQTT can incorporate security measures, such as TLS/SSL encryption and authentication, to protect data transmission.
- **Cross-Platform Compatibility**: MQTT is platform-independent and can be implemented on various hardware and software platforms.

MQTT have 2 types of participant devices:

1. **The broker**

Which acts as the server of the protocol and handling the data transferred between clients over topics and responsible for the authentication of clients and achieve the chosen quality of service level.

The broker can be either:

- Offline: on the edge computer or even on the raspberry by itself using local brokers like Mosquitto and HiveMQ.
- Online: on a cloud server like cloud MQTT.

For avoiding internet-less cases in solar plants located far away in desert, we use local broker for our application which is Mosquitto because of its simplicity and because its free open-source solution.

2. **The clients**

Client devices are the devices to connect to each other over the MQTT broker. In our application we have 2 primary types of clients:

- First is the robot itself which subscribes to the topics to either publish data for monitoring or listen to data for taking commands. It can be more than one robot in large capacity solar plants and all of them are clients.
- Second is the mobile application which we use for achieving remote control and monitoring.

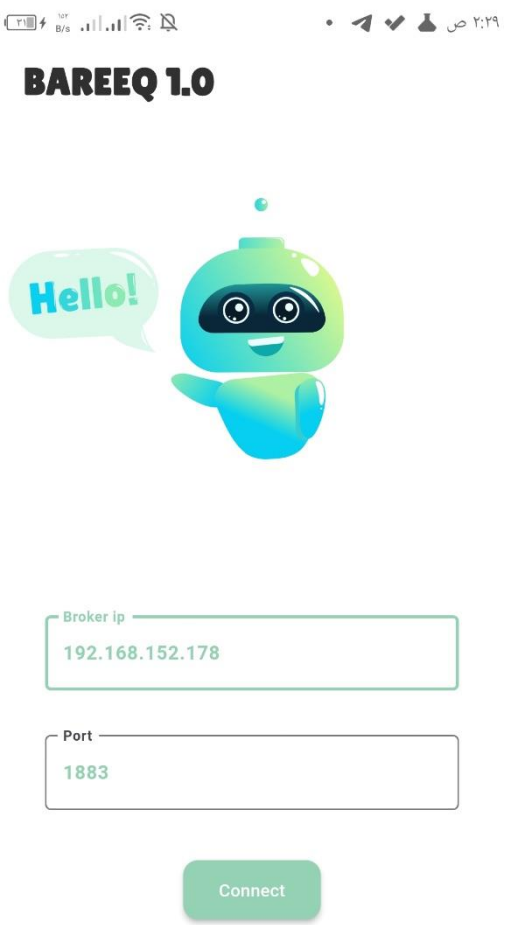


Figure 8.3.1:connect application to broker

8.4 Mobile Application

Using flutter framework, we create a user-friendly interface which the operator can use easily for controlling the robot or monitoring data.

Use of mobile application

1. Connect to broker by using the IP of the broker device and the port used for it.
2. After connecting successfully to the broker, we need to subscribe to the required MQTT topics for transferring data.
3. After connecting to the required topics then we can enter the control panel and control robot from it.

8.4.1 Flutter Framework:

Flutter is an open-source UI software development kit created by Google. It is designed to build natively compiled applications for mobile, web, and desktop from a single codebase.

Features:

1. **Single Codebase for Multiple Platforms**: Flutter allows us to write one codebase and deploy it on Android, iOS, web, and desktop platforms, significantly reducing development time and effort.
2. **Expressive and Flexible UI**: Flutter provides a rich set of pre-designed widgets that can be customized extensively. This flexibility allows developers to create complex and attractive UIs easily.
3. **Native Performance**: Flutter applications are compiled directly to native ARM code using Dart, the language it uses. This ensures that Flutter apps have high performance, similar to those written in native languages.
4. **Rich Ecosystem and Community**: Flutter boasts a vibrant community and a wide range of plugins and packages available through the Flutter and Dart package repositories. This ecosystem supports a variety of functionalities, from simple utilities to complex integrations.

8.4.2 Architecture

Flutter's architecture consists of three main layers:

1. **Framework**: The top layer contains the Flutter framework, which includes the rich set of pre-designed widgets, the rendering engine, and the Dart language. This layer provides the building blocks for creating applications.
2. **Engine**: The middle layer is the Flutter engine, written primarily in C++. It provides low-level rendering support using the Skia graphics library, as well as platform-specific code for iOS and Android.
3. **Embedder**: The bottom layer consists of platform-specific embedders. These embedders handle the interaction between the Flutter engine and the host platform (Android, iOS, web, etc.), including tasks like rendering surfaces, thread management, and input handling.

8.4.3 Architectural patterns:

Architectural patterns in Flutter provide a structured approach to managing the interactions between different components of an application, such as the user interface, business logic, and data management. Understanding and implementing these patterns is crucial for building robust and scalable applications.

8.4.4 Common Flutter Architectural Patterns

1. **MVC (Model-View-Controller):**
    - **Model**: Manages the data and business logic.
    - **View**: Handles the UI and presentation layer.
    - **Controller**: Acts as an intermediary between Model and View, handling user input and updating the Model.
2. **MVVM (Model-View-ViewModel):**
    - **Model**: Manages the data and business logic.
    - **View**: Displays the UI.
    - **ViewModel**: Manages the UI-related data and state, acting as a bridge between the Model and the View.

8.5 Why Use the MVC Pattern

The Model-View-Controller (MVC) pattern is a widely adopted software design pattern that helps us to organize and manage complex applications by separating concerns into three main components: Model, View, and Controller. Here are several reasons why using the MVC pattern is beneficial:

8.5.1 Separation of Concerns

- **Modularity**: MVC divides the application into three interconnected components, each responsible for specific tasks. This modularity makes the codebase easier to manage and understand.
- **Maintainability**: With distinct separation, changes in one component (e.g., UI updates in the View) do not directly affect other components (e.g., business logic in the Model), making the application easier to maintain and modify.

8.5.2 Reusability

- **Component Reuse**: Components in the MVC pattern can be reused across different parts of the application or even in other projects. For example, the same Model can be used with different Views or Controllers.
- **Code Reuse**: Business logic and data access code are centralized in the Model, reducing duplication and promoting reuse of code.

8.5.3 Testability

- **Unit Testing**: The separation of concerns facilitates unit testing. The Model can be tested independently of the UI, ensuring that business logic is correct without needing to interact with the user interface.
- **Automated Testing**: Automated tests can be written more easily for individual components, ensuring the reliability and robustness of the application.

8.5.4 Parallel Development

- **Team Collaboration**: Different developers or teams can work on different components simultaneously. For instance, one team can work on the Model and business logic, while another team focuses on the UI (View), and yet another handles user input and application flow (Controller).

8.5.5 Scalability

- **Application Growth**: MVC's structured approach makes it easier to scale applications. New features can be added with minimal impact on existing code, and the application can grow without becoming unmanageable.
- **Performance**: Efficient separation allows for optimized performance tuning. For example, Views can be optimized for rendering without affecting the underlying business logic.

8.5.6 Flexibility and Adaptability

- **UI Flexibility**: The View can be changed or replaced without altering the business logic in the Model. This is especially useful for applications that need to support multiple UIs (e.g., web, mobile, desktop).
- **Platform Independence**: The Model and Controller can remain unchanged when the application is ported to different platforms, requiring only changes to the View.

8.5.7 User Experience

- **Consistent UI/UX**: MVC helps in creating a consistent user experience by cleanly separating the user interface from the underlying logic and data management.
- **Responsive Design**: By managing the application flow and UI rendering separately, MVC facilitates the creation of responsive and dynamic user interfaces.

8.5.8 State Management

State management in Flutter is a core concept that involves handling the state of an application to ensure a responsive and dynamic user interface. The state refers to any data that can change over time in response to user actions, network requests, or other events.

Effective state management ensures that the UI accurately reflects the current state of the application and responds correctly to user interactions.

Why State Management is Important

- **Consistency**: Ensures the UI consistently represents the current state of the application.
- **Maintainability**: Simplifies the process of managing and updating the state, making the code easier to understand and maintain.
- **Scalability**: Helps manage complex state interactions and dependencies as the application grows.
- **Performance**: Optimizes the application's performance by efficiently managing state changes and updates to the UI.

State Management Patterns:

8.5.9 Scoped Model

Scoped Model is a pattern where state is encapsulated in models that notify listeners when changes occur. This pattern is often used to manage state within a specific scope, such as a widget subtree in Flutter. It helps in keeping state management localized, making the application more modular and easier to maintain.

8.5.10 Inherited Widget

Inherited Widget is a built-in Flutter widget that allows state to be efficiently propagated down the widget tree. This pattern is suitable for sharing state across many widgets without having to pass data through multiple widget constructors. It provides a way to make data available to the entire subtree of a widget.

8.5.11 Provider

Provider is a popular state management library for Flutter that simplifies state management by combining the power of InheritedWidget and the simplicity of a builder function. It provides a consistent way to access state from anywhere in the widget tree and is often used in conjunction with ChangeNotifier to handle state changes.

8.5.12 Riverpod

Riverpod is a modern state management library for Flutter that offers a more robust and safer alternative to Provider. It eliminates some of the boilerplate and limitations of Provider, providing a more straightforward and flexible API. Riverpod ensures better compile-time safety and improved performance.

8.5.13 Redux

Redux is a predictable state container for Dart applications, inspired by the Redux pattern used in JavaScript. It uses a single store to hold the entire state of the application and updates state through actions and reducers. This pattern promotes a unidirectional data flow, making the state changes predictable and easier to debug.

8.5.14 Bloc (Business Logic Component)

Bloc is a state management library that uses the concept of streams to manage state in a reactive way. It separates business logic from UI, making the code more modular and testable. Bloc manages state transitions based on events, ensuring that the state is consistent and that UI components react to state changes efficiently.

8.5.14.1 Key Features of Bloc:

- **Separation of Concerns**: Bloc separates business logic from the presentation layer, making the codebase cleaner and more maintainable.
- **Reusability**: Business logic components (Blocs) can be reused across different parts of the application.
- **Testability**: Bloc makes it easy to write unit tests for business logic, as the logic is decoupled from the UI.
- **Reactive Programming**: Bloc leverages streams to handle state changes, making it suitable for applications with complex and asynchronous interactions.

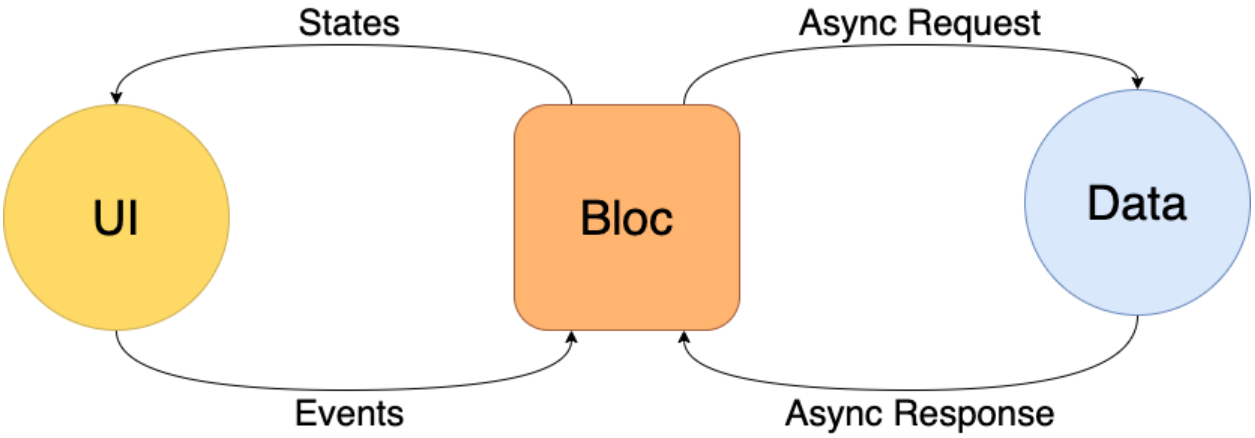


Figure 8.5.1: Block pattern flow diagram

Cubit

A cubit is a simplified version of a Bloc (Business Logic Component). While both Cubits and Blocs are used for managing state and business logic in Flutter applications, Cubits are designed to be more lightweight and straightforward, suitable for simpler use cases where the full power of Bloc might be unnecessary.

In Flutter's Bloc pattern, particularly when using Cubit, managing states is central to how the application responds to events and user interactions. Here's a detailed look at how states work within the context of Cubit:

States

States in Cubit are typically represented by a class that holds the data relevant to that state. This class is immutable and should represent a specific condition or snapshot of the application's state at a given moment.

- **Initial State:** Every Cubit starts with an initial state, defined when the Cubit is initialized. This initial state represents the application's state before any events or actions have been processed.
- **State Changes:**
  1. **Emitting States:** When an event or action occurs in the application, the Cubit emits a new state to reflect the updated condition of the application. This is done using the emit method provided by Cubit.
  2. **Immutability:** States are immutable, meaning once emitted, they cannot be changed directly. Instead, a new instance of the state class is created with the updated data.
- **State Transition:**
  1. **Transition Flow:** State transitions in Cubit follow a unidirectional flow. Events trigger state changes, and each emitted state represents a unique point in the application's lifecycle.
  2. **Deterministic:** The transition from one state to another is deterministic and predictable, based solely on the events and actions processed by the Cubit.

- **State Management:**
  1. **Single State Object:** Cubit manages the application's state using a single state object at any given time. This object encapsulates all the data needed to represent the application's current condition.
  2. **State Equality:** Equality of states is crucial for efficient UI updates in Flutter. Cubit ensures that only states with different instances trigger UI updates, optimizing performance.

Application Cubits:

In our mobile application, we have two screens:

1. **Connection screen:** shown in figure 8.1 from which we connect to the broker and subscribe to the required topics to transfer data from and to the robot.
   This screen is controlled by a cubit called **ConnectionScreenCubit** which has several states like:

   - **ConnectionInitState:** the initialization state which the screen starts with. (figure 8.1)
   - **ConnectionErrorState:** the state which the cubit emits when an error occurs during the connection.
   - **ConnectingState:** the state which the cubit emits when the application is trying to connect to the broker (figure 8.4).
   - **ConnectedState:** this state emitted when the application is connected successfully to the broker. (figure 8.2)
   - **SubscribedState:** when the application is subscribed to at least one topic, the cubit emits this state. (figure 8.3)
   - **NotConnectedState:** when the application failed to connect to the broker, then the cubit emits this state. (figure 8.5)

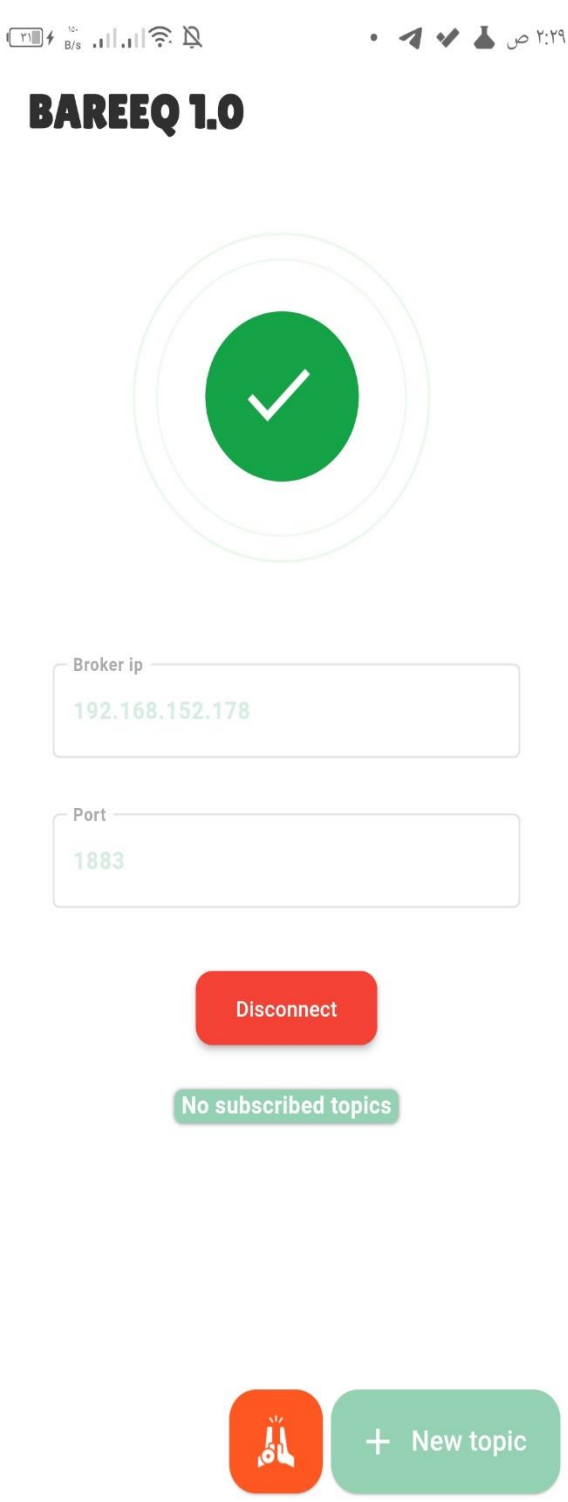


Figure 8.5.3: connected state

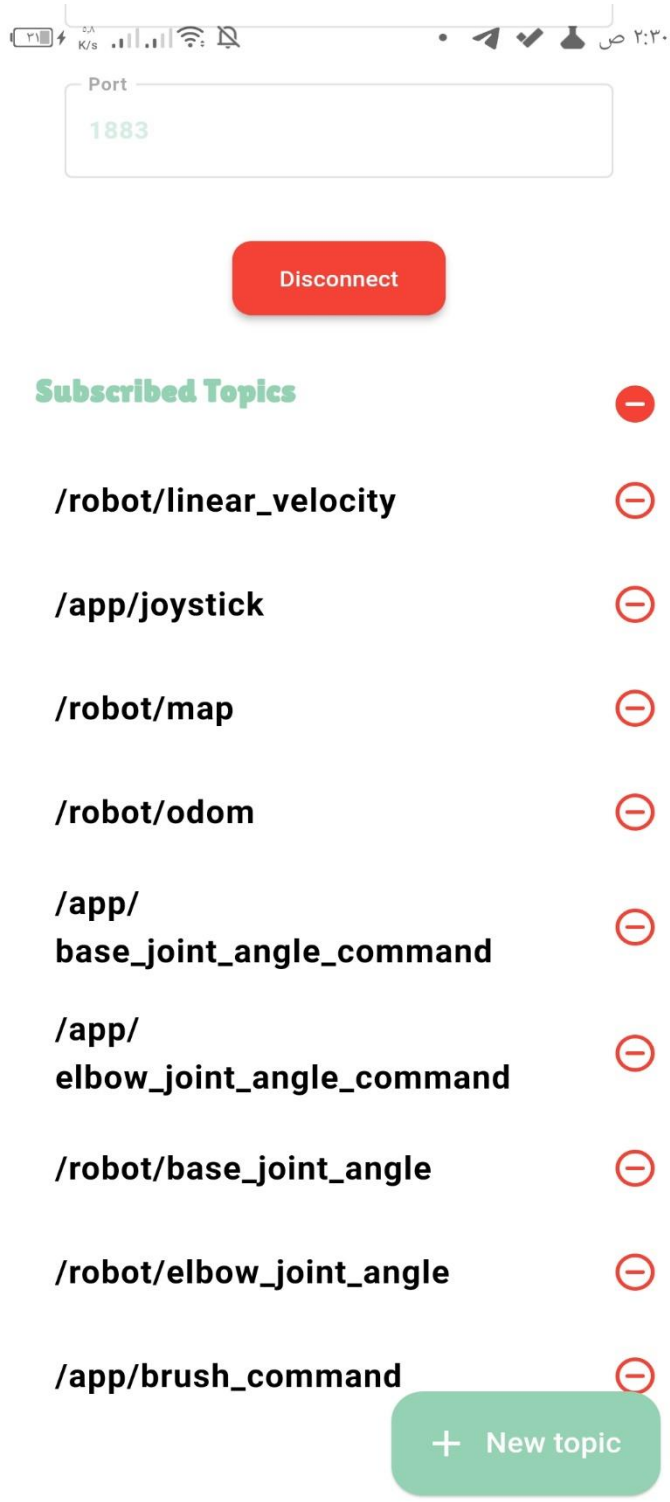


Figure 8.5.2: subscribed state

Figure 8.5.4: Connecting state

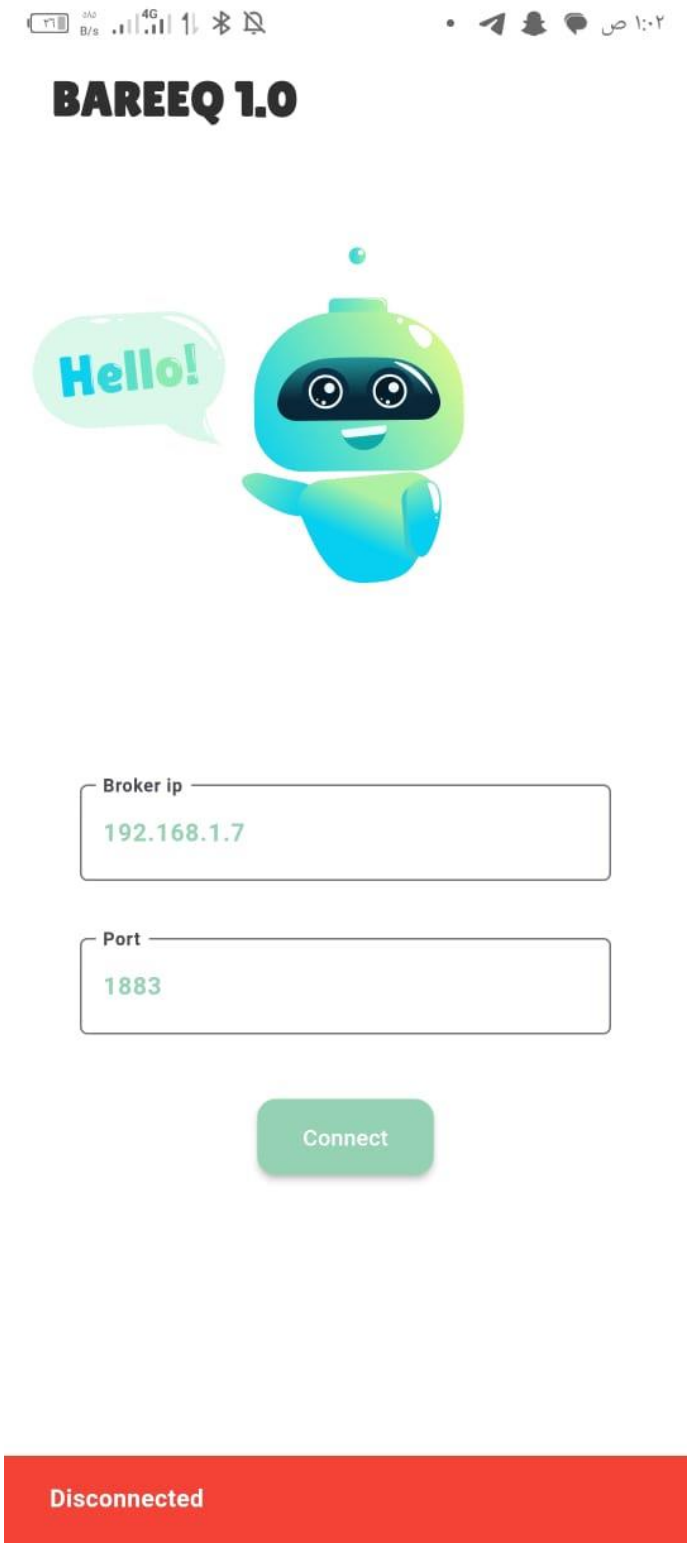


Figure 8.5.5: disconnect state

2. **Control screen:** the control panel of the robot in which we can send commands to the robot while working in semi-automatic mode and monitor data from the robot.
   Control panel screen is shown in (figure 8.11).

   We can't enter the control screen if the required topics aren't subscribed so we must subscribe to all required topics from the button shown in (figure 8.6).
   The required topics are shown in (figure 8.7).

Figure 8.5.6: subscribe to all required topics

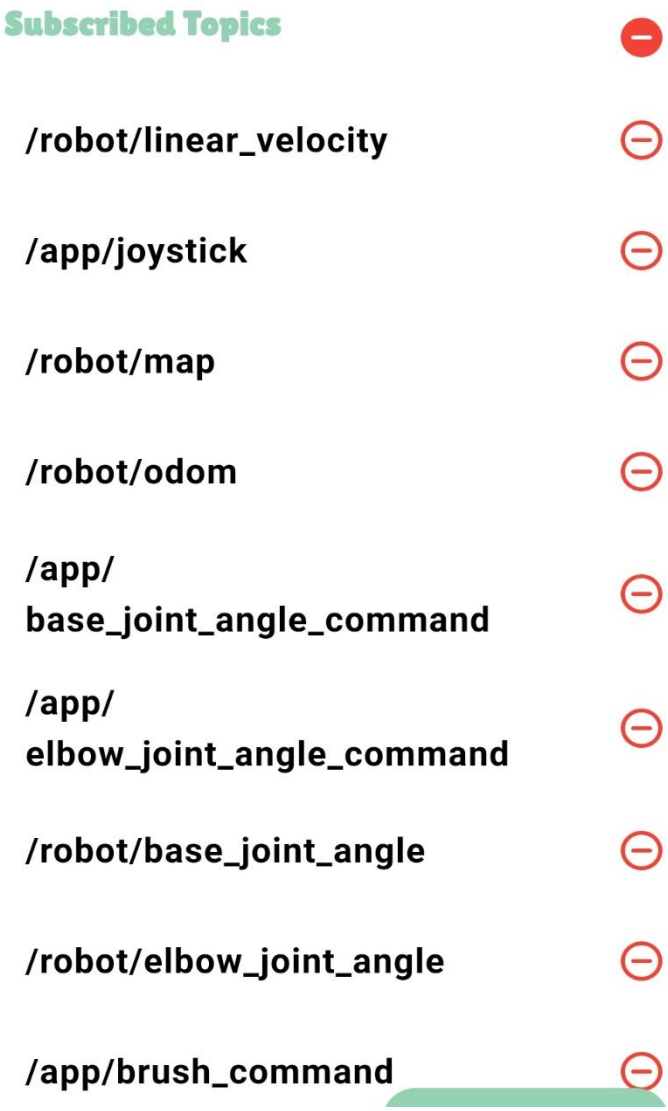


Figure 8.5.7: required topics

This screen is controlled by a cubit called **ControlScreenCubit** which has several states like:

- **ControlInitState:** the initialization state which the screen starts with.
- **ControlLoadingState:** the cubit emits this state if the screen is loading.
- **ControlErrorState:** the state which the cubit emits when an error occurs while controlling and monitoring the robot.
- **ControlConnectedState:** the state which the cubit emits if the required topics for control and monitoring are subscribed, and the clients are connected.
- **ControlNotConnectedState:** the state which the cubit emits if the required topics for control and monitoring aren't subscribed or any of the clients disconnected for any reason.
- **HideMapState:** the cubit emits this state if the user chose to hide map to reduce processing headache on the mobile.

  The user can activate this state by pressing this button in (figure 8.8)

Figure 8.5.8: hide map

- **DontUpdateMapState:** the cubit emits this state if the user chose to use the current map and doesn't update it as the robot moves around.

  This mode is very effective when the map is already generated.

  The user can activate this state by pressing this button in (figure 8.9)

Figure 8.5.9: don't update map

- **UpdateMapState:** the cubit emits this state if the user chooses to update map in time as the robot moves around. The user can activate this state by pressing this button in (figure 8.10)

Figure 8.5.10: update map

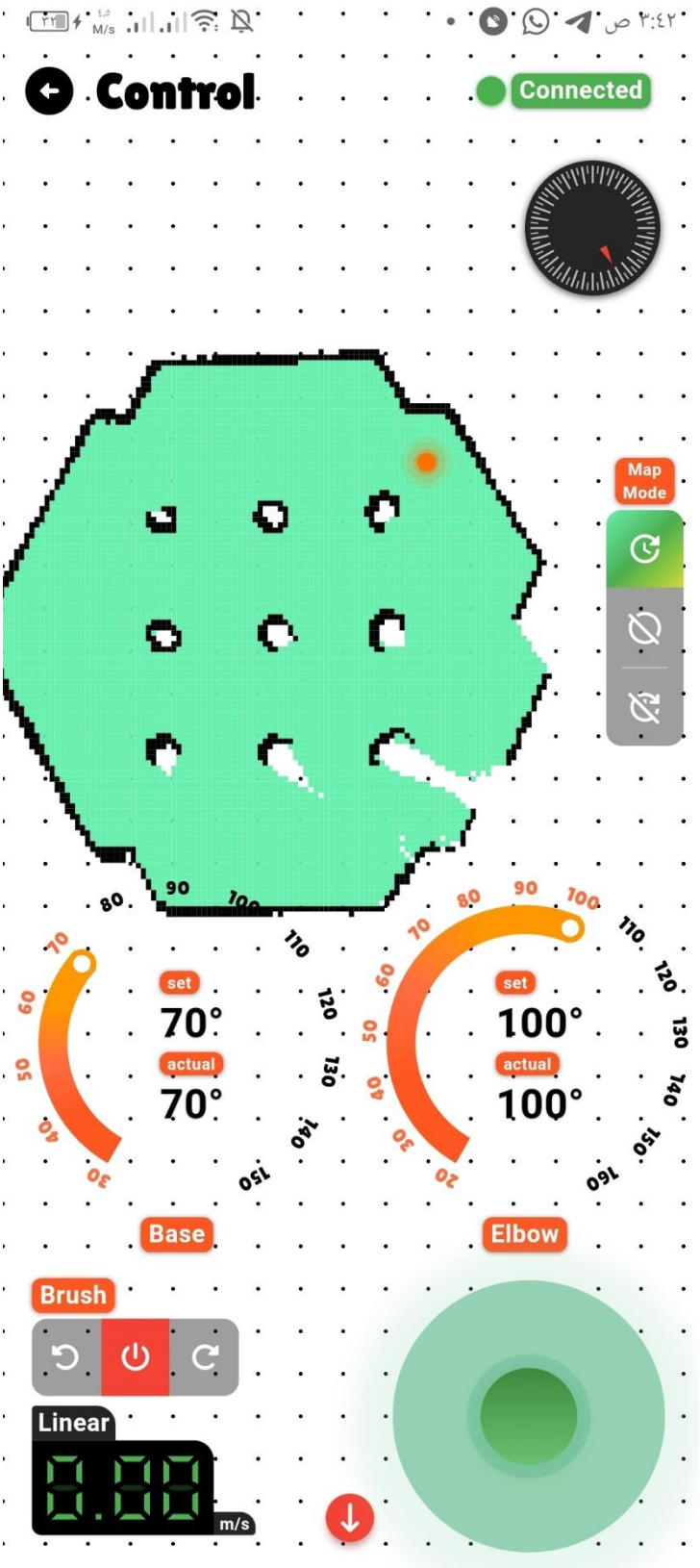


Figure 8.5.11: Control panel screen

8.5.15 MQTT Manager

MQTT Manager is the component responsible for interfacing flutter code with the MQTT infrastructure. We are using MQTT_Client package for achieving this role,

The Flutter MQTT_Client package is a library that enables Flutter applications to connect to MQTT brokers and communicate using the MQTT protocol

8.5.16 Manipulated Data

The mobile application publishes control commands to different MQTT topics, which the robot listens to. These commands include:

- joystick inputs for manual navigation,
- arm joints angles control,
- brush operating control.

On the other hand, the robot publishes its operating data and feedback to other topics used for monitoring which the mobile application listens to. These Data include:

- robot current actual speed,
- robot temperature,
- arm joints actual angles,
- map built by the robot,
- robot live pose within the map.

8.6 MQTT-ROS Bridge

The MQTT-ROS bridge enables seamless communication between ROS nodes and MQTT clients, facilitating data exchange and command control. The MQTT-ROS bridge serves as an intermediary that translates messages between the ROS and MQTT communication protocol.

The bridge also maps ROS topics to corresponding MQTT topics. This mapping ensures that data published on a ROS topic is correctly forwarded to the appropriate MQTT topic and vice versa

Acknowledgement

With the blessings and guidance of Allah, we are delighted to express our sincere appreciation and gratitude to all those who have supported us in completing this Autonomous Robot project.

We would like to extend our heartfelt thanks to our esteemed supervisors, Dr. Rowida Meligy, for their unwavering guidance, expert advice, and constant support throughout the various stages of this project. Their mentorship has been instrumental in the successful completion of this project.

We would also like to express our deepest appreciation to the Academy of Science and Research Technology (ASRT) for their generous financial support through their fund, which has enabled us to undertake this project. Their contribution has been invaluable in ensuring the success of this project.

Furthermore, we would like to express our sincere gratitude to all the examiners who have reviewed and provided feedback on our project. Your insights and suggestions have been invaluable in refining our work and enhancing its quality. We are grateful for your time and effort.

Finally, we would like to extend our gratitude to our families and friends for their unwavering support and encouragement throughout our academic journey. Their constant support has been a source of motivation and inspiration for us. May Allah bless all those who have contributed to this project and grant us the success and guidance in all our future endeavors.

Mechatronics Engineering

Department Faculty of Engineering

Helwan University

Cairo, Egypt 12 July, 2024

REFERENCES

1. Design on machine elements (Mechanical designing book of the mechanical engineering department, Helwan University)
2. Chain drive | Online Calculator (calcdevice.com)
3. https://www.usarollerchain.com/roller-chain-size-chart-s/4869.htm
4. https://productselect.skf.com/guidedflow/index.html#/start-page/
5. https://octaforce.com/sprocket/simplex-sprockets-2/
6. https://www.theworldmaterial.com/
7. https://www.mathworks.com/help/simulink/slref/automotive-suspension.html
8. https://bicyclepost.blogspot.com/2020/09/force-analysis-of-bicycle-chain-and.html
9. https://grabcad.com/
10. https://www.martinsprocket.com/docs/catalogs/engineering/engineering%20catalog/sprocket-engineering-data.pdf
11. The link of the page containing the datasheet of the linear DC Actuator: https://www.ebay.com/itm/294392697019
12. Kinematics Analysis and Workspace Calculation of a 3-DOF Manipulator. https://iopscience.iop.org/article/10.1088/1755-1315/170/4/042166/pdf).
13. WSRender: A Workspace Analysis and Visualization Toolbox for Robotic Manipulator Design and Verification.
14. (“Introduction to Robotics: Mechanics and Control" by John J. Craig).
15. Dynamics and Control of Robotic Manipulators with Contact and Friction.
16. [Introduction to Autonomous Mobile Robots second edition Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza]
17. [A Review of 2D SLAM Algorithms on ROS Supervisor: Prof. Matteo Matteucci Master Thesis by: Bayu Kanugrahan Luknanto].
18. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters Giorgio Grisetti∗† Cyrill Stachniss‡∗ Wolfram Burgard∗ ∗ University of Freiburg, Dept. of Computer Science, Georges-Kohler-Allee 79, D-79110 Freiburg, Germany ¨ † Dipartimento Informatica e Sistemistica, Universita “La Sapienza”, Via Salaria 113, I-00198 Rome, Italy ´ ‡ Eidgenossische Technische Hochschule Zurich (ETH), IRIS, ASL, 80 ¨ 92 Zurich, Switzerland
19. Liu, X., Lin, Y., Huang, H., Qiu, M. (2020). Comparative Analysis of Three Kinds of Laser SLAM Algorithms. In: Qiu, M. (eds) Algorithms and

Architectures for Parallel Processing. ICA3PP 2020. Lecture Notes in Computer Science (), vol 12453. Springer, Cham. https://doi.org/10.1007/978-3-030-60239-0_31
20. researchgate.net - Map Comparison of Lidar-based 2D SLAM Algorithms
21. researchgate.net - A Partitioned Approach for Efficient Graph-Based Place
22. researchgate.net - A life-long SLAM approach using adaptable local maps
23. nature.com - Finding the best hardware configuration for 2D SLAM in
24. wiley.com - Multiple-Robot Simultaneous Localization and Mapping
25. Robot localization in a mapped environment using Adaptive Monte Carlo algorithm
26. International Journal of Scientific & Engineering Research Volume 9, Issue 10, October-2018
27. Autonomous Navigation Algorithms for Mobile Robots based on LiDAR Technologies Autor: Leanne Rebecca Miller Director: Pedro Javier Navarro Lorente Codirector: Carlos Fernández Andrés Cartagena, Diciembre de 2017
28. Path planning techniques for mobile robots: Review and prospect Lixing Liu , Xu Wang , Xin Yang * , Hongjie Liu , Jianping Li , Pengfei Wang College of Mechanical and Electrical Engineering, Hebei Agricultural University, Baoding 071000, China
29. Marín, Pablo & Hussein, Ahmed & Martín Gómez, David & de la Escalera, Arturo. (2018). Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles. Journal of Advanced Transportation. 2018. 1-10. 10.1155/2018/6392697.
30. Autonomous Navigation Algorithms for Mobile Robots based on LiDAR Technologies Autor: Leanne Rebecca Miller Director: Pedro Javier Navarro Lorente Codirector: Carlos Fernández Andrés Cartagena, Diciembre de 2017
31. Comparison of local planning algorithms for mobile robots Miroslav Kulich, Viktor Koz´ak, and Libor P˘reu˘cil
32. [1]. (Kinematics Analysis and Workspace Calculation of a 3-DOF Manipulator. https://iopscience.iop.org/article/10.1088/1755-1315/170/4/042166/pdf)
33. [2] WSRender: A Workspace Analysis and Visualization Toolbox for Robotic Manipulator Design and Verification
34. [3] (“Introduction to Robotics: Mechanics and Control" by John J. Craig)
35. [4] Dynamics and Control of Robotic Manipulators with Contact and Friction
36. K.J. Åström and T. Hägglund, "PID Controllers: Theory, Design, and Tuning", Instrument Society of America, 1995.

37.K. Ogata, "Modern Control Engineering", Pearson Education, 2010.
38.G.F. Franklin, J.D. Powell, and A. Emami-Naeini, "Feedback Control of Dynamic Systems", Pearson Education, 2018.
39.R.C. Dorf and R.H. Bishop, "Modern Control Systems", Pearson Education, 2016.
40.N.S. Nise, "Control Systems Engineering", Wiley, 2015.
41.MathWorks Documentation on PID Tuner: PID Tuning in MATLAB.
42.MathWorks Documentation on System Identification Toolbox: System Identification Toolbox.
43."Feedback Control of Dynamic Systems" by G.F. Franklin, J.D. Powell, and A. Emami-Naeini, Pearson Education, 2018.
44.MATLAB Central: A community and resource hub for MATLAB users, where you can find discussions, examples, and scripts related to PID control and system identification.

APPENDICES

Appendix 1: Trajectory planner code

Code[1]

```
% Define symbolic variables
syms t_sym
t1_i = 0;
t1_f = 70;
t2_i = 0;
t2_f = 20;
theta1_i = 11.5;
theta2_i = 31.5;
theta1_f =53.1;
theta2_f = 57.17;
theta1_dot_i = 0;
theta2_dot_i = 0;
theta1_dot_f = 0;
theta2_dot_f = 0;

% theta1_i = deg2rad(th1_i);
% theta2_i = deg2rad(th2_i);
% theta1_f =deg2rad(th1_f);
% theta2_f = deg2rad(th2_f);


% Coefficients calculations
a1 = theta1_i;
a2 = theta2_i;
b1 = theta1_dot_i;
b2 = theta2_dot_i;
c1 = ((3/(t1_f^2))*(theta1_f-theta1_i)) - ((1/t1_f)*(theta1_dot_f + 2*theta1_dot_i));
c2 = ((3/(t2_f^2))*(theta2_f-theta2_i)) - ((1/t2_f)*(theta2_dot_f + 2*theta2_dot_i));
d1 = ((1/(t1_f^2))*(theta1_dot_f-theta1_dot_i)) - ((2/(t1_f^3))*(theta1_f-theta1_i));
d2 = ((1/(t2_f^2))*(theta2_dot_f-theta2_dot_i)) - ((2/(t2_f^3))*(theta2_f-theta2_i));
% Polynomial equations
theta1 = a1 + b1 * t_sym + c1 * t_sym.^2 + d1 * t_sym.^3;
theta2 = a2 + b2 * t_sym + c2 * t_sym.^2 + d2 * t_sym.^3;
theta1_dot = b1 + 2 * c1 * t_sym + 3 * d1 * t_sym.^2;
theta2_dot = b2 + 2 * c2 * t_sym + 3 * d2 * t_sym.^2;
theta1_ddot = 2 * c1 + 6 * d1 * t_sym;
theta2_ddot = 2 * c2 + 6 * d2 * t_sym;

% Define time vector for plotting
t1 = linspace(t1_i, t1_f, 20);
t2 = linspace(t2_i, t2_f, 20);
```

```
% Evaluate the expressions over time
theta1_vals = double(subs(theta1, t_sym, t1));
theta2_vals = double(subs(theta2, t_sym, t2));
theta1_dot_vals = double(subs(theta1_dot, t_sym, t1));
theta2_dot_vals = double(subs(theta2_dot, t_sym, t2));
theta1_ddot_vals = double(subs(theta1_ddot, t_sym, t1));
theta2_ddot_vals = double(subs(theta2_ddot, t_sym, t2));

% Plotting the results
figure;
subplot(3, 2, 1);
plot(t1, theta1_vals, '-o');
title('\theta_1(t)');
xlabel('Time (s)');
ylabel('\theta_1 (degrees)');

subplot(3, 2, 2);
plot(t2, theta2_vals, '-o');
title('\theta_2(t)');
xlabel('Time (s)');
ylabel('\theta_2 (degrees)');

subplot(3, 2, 3);
plot(t1, theta1_dot_vals, '-o');
title('\theta_1\_dot(t)');
xlabel('Time (s)');
ylabel('\theta_1\_dot (degrees/s)');

subplot(3, 2, 4);
plot(t2, theta2_dot_vals, '-o');
title('\theta_2\_dot(t)');
xlabel('Time (s)');
ylabel('\theta_2\_dot (degrees/s)');

subplot(3, 2, 5);
plot(t1, theta1_ddot_vals, '-o');
title('\theta_1\_ddot(t)');
xlabel('Time (s)');
ylabel('\theta_1\_ddot (degrees/s^2)');

subplot(3, 2, 6);
plot(t2, theta2_ddot_vals, '-o');
title('\theta_2\_ddot(t)');
xlabel('Time (s)');
ylabel('\theta_2\_ddot (degrees/s^2)');

max(theta1_ddot_vals)
max(theta2_ddot_vals)
max(theta1_dot_vals)
max(theta2_dot_vals)
```