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

B. Eng. Final Year Project

AMR(Autonomous mobile robot in warehouse)

By:

GROUP (3)

NAME	*ID*
Ammar Abdel Naser	*38243*
Mahmoud Tawfik	*95933*
Remon Lamy	*95893*
Abdullah Sayed	*91645*
Nathan Romany	*91500*

Supervised by:

Supervisor(s)

Dr. Bahaa Nasser	*Head of Department*
Prof. Dr. Mohamed Hamdy	*Department professor*
Prof. Dr. Mohamed Ibrahim	*Menofia University Professor*
Dr. Alaa Zakria Nasser	*Department professor*
Dr. Ahmed Sabri	*Department professor*
Ass. Prof. Tahani Wileam	*Beni-suef University*
Dr. Bekhet Mohamed	*Japanese University*

List of Content Page

List of Tables

List of Figures

Abstract

This book contains a fully represented methodology for an AMR (autonomous mobile robot), which is a big jump forward in the industrial field for its importance in boosting efficiency. AMRs automate repetitive tasks, freeing humans for higher-level work. Enhancing Productivity: Operating tirelessly and precisely, adapting to the future: AMRs offer flexible automation, adapting to changing demands and environments. The designing process is split into chapters for each designing phase.

Phase one: investigate the market, choose the application that our robot will serve, and learn more about the cost and previously associated projects. Phase two: start our customized design for the robot function requirements, choose suitable materials, and start implementing the theoretical work through CAD models and doing the appropriate tests using computer-aided tools. Phase three: identify our electrical component, analyze the power circuit for battery size selection, and test our circuit through suitable cad models. Phase four: identify the most powerful software tool to serve our system goal and start to simulate our system software modules through a simulated robotics environment.

Introduction

AMR stands for "Autonomous Mobile Robot." These are robots equipped with sensors, navigation systems, and advanced algorithms that enable them to move and operate autonomously in dynamic environments without human intervention. AMRs are designed to perform various tasks, such as transporting goods, navigating through warehouses, assisting in manufacturing processes, and conducting surveillance in security applications. They play a crucial role in industrial automation, logistics, healthcare, and other sectors where mobility, flexibility, and efficiency are paramount.

Autonomous Mobile Robots (AMRs) are cutting-edge transport robots specifically designed to move loads autonomously in a diverse range of industries, from automotive to logistics to consumer goods and other industrial processes.

(Figure 1.1 AMRs transport autonomously in a diverse range of industries)

We can use AMR in many different use cases forecast project the biggest uptakes for AMR will be in manufacturing and warehousing this is because AMR is are particularly useful for material picking and moving in Brick and Mortar retail AMRs can traverse the whole store and check shelves to make sure that they're fully replenished in a restaurant AMR can assist staff in delivering food as soon as it's ready in banking robots can help customers identify the forms of the need to fill out in malls customers to the location that they desire to get to in hotels if a customer orders a toothbrush was something else for the room AMRs are able to navigate and deliver the goods in hospitals they can deliver food water

(Figure 1.3 AMR in restaurant)

(Figure 1.4 AMR in hospital)

(Figure 1.2 AMR in market)

medications personal protective equipment they can sanitize environments with ultraviolet rays, moving it to the future AMRs can perform a generalized and specialized tasks this includes tasks that are not safe for humans to perform AMRs can inspect pipes from predictive maintenance gather a lot of data and send it back to the edge of the cloud without putting humans at risk, no one can forget the robots roll in the great disaster of Chernobyl nuclear power plant

So no doubt AMRs can take the industrial production to a new area They're really good at making things faster and smoother. By working alongside people, they help businesses keep up with what customers need. AMRs are like super helpful teammates, making sure everything runs smoothly and efficiently.

In historical point of view The history of AMRs (Autonomous Mobile Robots) is surprisingly long and rich, stretching back further than you might think from 1948 :William Grey Walter creates "Elijah" and "Elsie," two mobile robots exhibiting autonomous behavior, sparking early research Present: Continuous research and development lead to increasingly sophisticated and adaptable AMRs, capable of complex tasks and collaborating with humans.

(Figure 1.5 Elijah the first mobile robot 1948)

Key Milestones: 1994 The first autonomous vacuum cleaner, Roomba, becomes a commercial success. Roomba is a series of autonomous robotic vacuum cleaners made by the company iRobot. Introduced in September 2002, [1] they have a set of sensors used to them navigate the floor area of a home. These sensors can detect the presence of obstacles and steep drops (e.g., to avoid falling downstairs)

help

(Figure 1.6 Roomba The first autonomous vacuum clean)

2012: Amazon acquires Kiva Systems, further validating the potential of AMRs in logistics.

2016: A cobot, or collaborative robot, is a robot intended for direct human-robot interaction within a shared space, or where humans and robots are in close proximity. Cobot applications contrast with traditional industrial robot applications in which robots are isolated from human contact or the humans are protected by robotic tech vests. Cobot safety may rely on lightweight construction materials, rounded edges, and inherent limitation of speed and force, or on sensors and software that ensure safe behavior.

(Figure 1.7 collaborative robot)

OUR AMR

Navigation system:2D Rp Lidar 8m to scan the horizontal area and by using SLAM algorithm we build our map depending on this data

Lifting system: ball screw mechanism designed to left the stand 30mm with separated stepper motor.

Driving System: four DC motor controlled independently using four-wheel deferential steering system Power.

Power system: 40A Li ion battery to supplying the onboard component and the other systems.

AMR definition

Our project objective is to help in E-commerce field as warehouse application. There are two types of mobile robots used in this field AMR and AGV. AMR stands for Autonomous Mobile Robot, while AGV stands for Automated Guided Vehicle. Both types of robots are used to automate material handling and transportation tasks in industrial and commercial settings. However, there are some key differences between the two technologies. AGVs are typically programmed to follow a fixed path or route, which is often defined by wires, magnetic tape, or other physical guides. AGVs are typically less expensive than AMRs, but they are also less flexible and adaptable

AMRs, on the other hand, are able to navigate their environment autonomously, without the need for physical guides. This makes AMRs more flexible and adaptable than AGVs, and it also allows them to be deployed in more complex and dynamic environments. However, AMRs are typically more expensive than AGVs. Here is summarizes the key differences between AMRs and AGVs:

(Table 1.1 differences between AMRs and AGVs)

Characteristic	AMR	AGV
Navigation	Autonomous	Guided by physical guides
Flexibility and adaptability	High	Low
Cost	High	Low

Design Considerations

After defining our robot commercially now here it is the technical definition first we have the robot features: -

- moving speed up to 1.3 m /s
- mapping and localization
- dynamic tracking destination
- carrying backed products

In order to begin the transition to technical details, we need to start with VDI 2206. This is our primary reference for design as mechatronics engineers.

The project design is divided into two fundamental elements:

- Organizational structuring within the team and assembly methodology.
- The scientific foundation of the design and its technical stages.

*Organizational structuring within the team and assembly methodology.

This element separated to 2 stages:

a. Modularization: Modularization refers to the process of breaking down a complex system or project into smaller, self-contained modules or components. Each module serves a specific function or task and can be developed or modified independently. This approach simplifies design, development, and maintenance, making it more efficient and adaptable.

b. Hierarchization: Hierarchization involves structuring a project or system in a hierarchical manner, where components or modules are organized in a clear and ordered hierarchy. This hierarchy defines the relationships and dependencies between different parts of the system, allowing for better control, communication, and management of the project's complexity.

Automatic Control

In these part we will use PID controller among several types of controlling methods like:

1. Proportional-Integral (PI) Controller

2. Proportional-Derivative (PD) Controller

3. Proportional-Integral-Derivative-Filter (PIDF) Controller

4. Adaptive Controllers

5. Model Predictive Controller (MPC)

6. Fuzzy Logic Controller (FLC)

PID controllers are widely used in various control systems due to several reasons:

1. Simplicity

2. Versatility

3. Stability and Robustness

4. Tuning and Adaptability

5. Industry Standard

Information Technology

where the true value of the project lies. Each feature added in this branch enhances the product's worth. Therefore, our aim is to use one of the most commonly used microprocessors, either AVR or ARM, to optimize the robot's responsiveness as much as possible. We will create suitable drivers for each component of the robot. Additionally, for embedded systems, we intend to use ROS Noetic as the robot's operating system and incorporate simulation systems like Gazebo or Webot to the extent possible. These will serve as monitors for the robot's interaction with the surrounding environment. Furthermore, if time permits, we may explore other areas in greater depth

Chapter.2

MECHANICAL DESIGN: -

2.1 Introduction on autonomous Mobile robots: -

In the mechanical design we wanted to achieve the best results possible with the lowest cost possible so we followed a number of steps that is well known for mechanical design engineers we will discuss each step and then we will go throw the solid works and calculations.

2.2 Design steps: -

1-industry comparison

2- requirements

3- cost effectiveness

4- follow the loads

5- functionality

6- simplicity

7- Material environmental /usage

8-Aesthetics

9-Manufacturability

10- installation sequencing

1- Industry comparison

Comparing what we are going to design with what is already available for the clients and this comparison is made on cost, effectiveness and what's new what do I offer that is not available. (the comparison is available in the survey).

2- Requirements

During this point we studied all our requirements from raw materials, machining and human resources

3-Cost Effectiveness

We aimed at achieving the best results while maintaining the lowest cost possible.

4-Follow the loads

The primary consideration of a mechanical design is the function. Thus the loads are the primary consideration of the shape size and material.

5- Functionality

- Getting feedback from operators
- Making a cad simulation
- Check for safety requirements.

6- Simplicity

Keeping the design as simple as possible while getting the required function done this saves money and simplifies the programing and control part.

7- Material environmental usage

Studying the environment that the robot will work in to know the limitations of the material and the required specification of the material or the coating needed.

8-Aesthetics

-quality look

- safety

- Handling

- theme

- treatment or coating

- impression matter.

9- Manufacturability

Studying the manufacturability of the project and making sure that the required equipment is available and also the skilled operators to work them for you to be able to manufacture it the way you designed it.

10- Installation Sequencing Planning out the:

- Vendors
- Sourced materials
- Assembly
- Installation
- Maintenance

Those are the design steps that we tried to follow during the design process as much as possible.

2.3 First the material selection: -

Many different materials are used in the manufacture of autonomous mobile robot bodies, and the most commonly used materials nowadays are steel, aluminum, and stainless steel.

(Table 2.1 Deformation on different materials)

Material	Thickness	Weight Approx.	Load	Deformation On top part
Steel	2 mm	16 Kg	250 Kg	5.56×10^{-1} mm
Steel	3 mm	21.5 Kg	350 Kg	7.5×10^{-1} mm
Aluminum	2 mm	5.5 Kg	250 Kg	1.5 mm
Aluminum	3 mm	8 Kg	250 kg	1.22 mm

This (table 1.2) shows the deformation that occurs in a rectangular plate made of steel and aluminum as a result of effect on it with a load. The table shows the thickness, the approximate weight, the load, and the deformation resulting from the load.

In this table we studied the available material and concluded to use the (2mm) steel**.** We used AISI 4130 material for the body of our robot

Advantages of AISI 4130:

1-High strength-to-weight ratio
2- Excellent fatigue strength
3-Good ductility
4-Good machinability

2.4 Mechanical parts: -

(Table 2.2 Mechanical parts weight)

Part name	Number	Weight(Kg)	Parameters(cm)
AMR top	1	8	50*45
AMR bottom	1	8	50*45
Shafts	4	0.062	1.2*7
Tiers and coupling	4	1.5	13*6
Motors	5	3	10*4.8
Ball screw system	1	3	1.2*20(for the screw)

2.5 Motor Sizing:

A. For the DC Motors:

Wheel Rim = 5 cm

Car weight = 16+3+3+1.5+0.5= 24 Kg

Desired load = 98 Kg

Total weight = 122 KG = 1197 N

Assuming the equal distribution of load $\frac{1197}{4} = 299.2\ N$ for each Tire

$$MOTOR\ POWER = \frac{F \times N \times R}{9550} \qquad 2\text{-}1$$

$F:\ Load$

$N:\ No.of\ revolution = 180\ rpm$

$R:\ radius\ of\ the\ rim\ \ = 0.025\ m$

$$MOTOR\ POWER = \frac{299.2 \times 180\ \times 0.025}{9550} = 0.1409\ KW = 140.9\ watt$$

$1\ hp = 746\ watt$

Take factor of safety = 1.2

So The $MOTOR\ POWER = 140.9\ watt\ \times 1.2 = 169.1\ watt$

$$MOTOR\ POWER\ in\ hourse\ power \approx 170\ watt = 0.227\ hp$$

2.6 Shaft Design:

Material for the shaft is steel Grade 40C8 that has a $yeild\ strength = 320\ MPa$

Take factor of safety = 4 , $Ultimate\ tensile\ strength = 560\ MPa$

Find allowable stress and shear:

(Table 2.3 Combined shock and fatigue factor for bending and torsion)

Nature of load	K_m	K_t
1. Stationary shafts		
(a) Gradually applied load	1.0	1.0
(b) Suddenly applied load	1.5 to 2.0	1.5 to 2.0
2. Rotating shafts		
(a) Gradually applied or steady load	1.5	1.0
(b) Suddenly applied load with minor shocks only	1.5 to 2.0	1.5 to 2.0
(c) Suddenly applied load with heavy shocks	2.0 to 3.0	1.5 to 3.0

$$\sigma_b = \frac{Syt}{f.s} = \frac{320}{4} = 80\ MPa \quad (2-2)$$

$$\tau = \frac{0.5 * Syt}{f.s} = \frac{0.5 * 320}{4} = 40\ MPa$$

$Power\ of\ motor\ = 170\ watt$,

$N = 180\ rpm$

$$Torque\ = \frac{power*60}{2*\pi*N} = \frac{170*60}{2*\pi*180} = 9.01\ N.m\ = 9.01 \times 10^3\ N.mm \quad (2-3)$$

K_m: Combined shock and fatigue factor for bending.

K_t: Combined shock and fatigue factor for torsion.

Assuming the equal distribution of load, the shaft carries 299.2 N

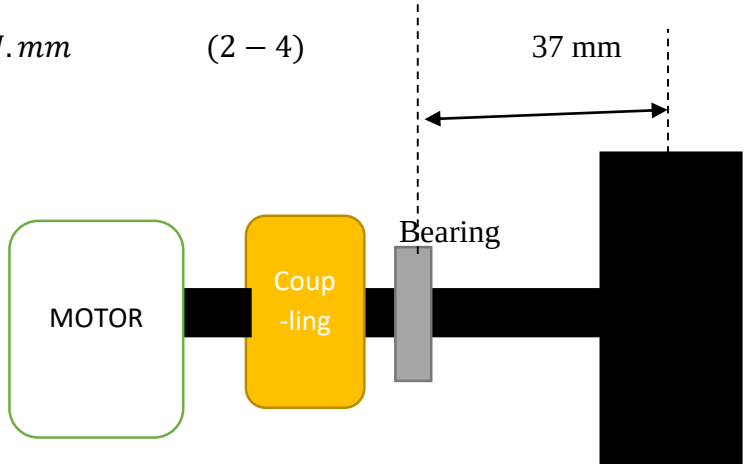
$$load = \frac{122\ Kg}{4} = 30.5\ Kg\ ,\quad W = 30.5 * 9.81 = 299.2\ N$$

$M: Bending\ moment$

$$M = W * L = 299.2 * 37 = 11070.4\ N.mm \quad (2-4)$$

T_e: $equivalent\ twisting\ moment$

M_e: $equivalent\ bending\ moment$



(Figure 2.1 displacement between bearing and wheel)

$$T_e = \sqrt{(K_m \times M)^2 + (K_t \times T)^2} \quad (2-5)$$

$$T_e = \sqrt{(K_m \times M)^2 + (K_t \times T)^2} = 18892.48\ N.mm$$

$$M_e = \frac{1}{2}\left[K_m \times M + \sqrt{(K_m \times M)^2 + (K_t \times T)^2}\ \right] \qquad (2-6)$$

$$M_e = \tfrac{1}{2}\left[K_m \times M + \sqrt{(K_m \times M)^2 + (K_t \times T)^2}\ \right] = 17749.04\ N.mm$$

$$T_e = \frac{\pi}{16} \times \tau \times d^3 \ \ , \ \ d = 13.39\ mm \qquad d = 15\ mm\ \ approx.$$

$$M_e = \tfrac{\pi}{32} \times \sigma_b \times d^3 \ \ , \ \ d = 13.12\ mm \qquad d = 15\ mm\ \ approx.$$

The Endurance Limit for shaft [5]

What Does Endurance Limit (Se) Mean?

The endurance limit (Se) of a material is defined as the stress below which a material can endure an infinite number of repeated load cycles without exhibiting failure. In other words, when a material is subjected to a stress that is lower than its endurance limit, it should theoretically be able to withstand an indefinite amount of load cycles.

$$Se\ = K_a K_b K_c K_d K_e K_f S'_e \qquad (2-7)$$

Where:

$K_a =$ surface condition modification factor

$K_b =$ size modification factor

$K_c =$ load modification factor

$K_d =$ temperature modification factor

$K_e =$ reliability factor

$K_f =$ miscellaneous − effects modification factor

$S'_e =$ rotary − beam test specimen endurance limit

$S_e =$ endurance limit at the critical location of a machine part in the geometry and condition of use

$S_{ut} = Minimum\ tensile\ strength$

$\mathcal{F} = F$atigue strength fraction

$K_a = aS_{ut}^b = 57.7(560)^{-0.718} = 0.61372 \qquad (2\text{-}8)$

Get (a) and (b) from Table 2.4

Surface Finish	Factor a S_{ut}, kpsi	Factor a S_{ut}, MPa	Exponent b
Ground	1.34	1.58	−0.085
Machined or cold-drawn	2.70	4.51	−0.265
Hot-rolled	14.4	57.7	−0.718
As-forged	39.9	272.	−0.995

(Table 2.4 value of factor a and b for K_a)

from the calculation we find the shaft diameter d = 15 mm

so $K_b = (\frac{15}{7.62})^{-0.107} = 0.9280$

$$k_b = \begin{cases} (d/0.3)^{-0.107} = 0.879d^{-0.107} & 0.11 \le d \le 2 \text{ in} \\ 0.91d^{-0.157} & 2 < d \le 10 \text{ in} \\ (d/7.62)^{-0.107} = 1.24d^{-0.107} & 2.79 \le d \le 51 \text{ mm} \\ 1.51d^{-0.157} & 51 < d \le 254 \text{ mm} \end{cases}$$

(Equations 2-9 size modification factor)

There is a bending force on the shaft

Kc = 1

$$k_c = \begin{cases} 1 & \text{bending} \\ 0.85 & \text{axial} \\ 0.59 & \text{torsion}^{17} \end{cases}$$

(Equation 2-10 load modification factor)

K_d

We can calculate K_d by using next Equation :

$K_d = 0.975 + 0.432(10^{-3})T_F - 0.115(10^{-5})T_F^2 + 0.104(10^{-8})T_F^3 - 0.595(10^{-12})T_F^4$ (2-11)

Our robot will work in ware house at room temperature = 25 ℃ which = 77 °F

Fide $K_d = 1.0018$

Get K_e value from Table

Take the reliability =90%

$K_e = 0.897$

Reliability, %	Transformation Variate z_a	Reliability Factor k_e
50	0	1.000
90	1.288	0.897
95	1.645	0.868

(Table 2.5 reliability factor)

Take $K_f = 1$

$S'_e = 0.5S_{ut}$ at 10^6 cycles , $S'_e = 280\ MPa$

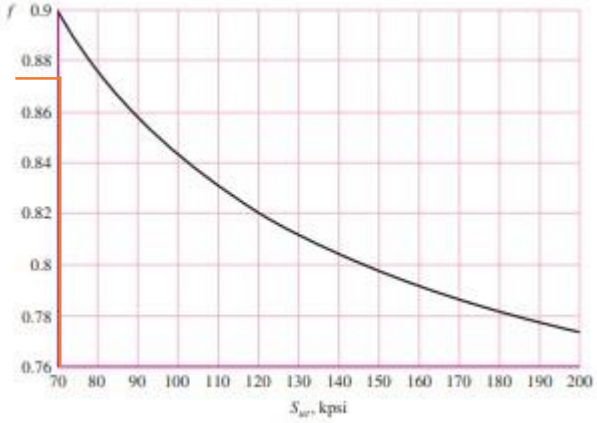
$$Se = (0.6137)(0.928)(1)(1.0018)(0.897)(1)(280) = 143.26\ MPa$$

For our material the $Minimum\ tensile\ strength = 560\ MPa$

Which equal to 81.22 Kpsi (kilo pound force per square inch)

$1Kpsi = 6.895\ MPa$

Fatigue strength fraction, f, of Sut at 10^3 cycles for Se = S'e = 0.5Sut at 10^6 cycles.



$\mathcal{F} = F$atigue strength fraction

$S_f =$ fatigue strength

where N is cycles to failure and the constants a and b are defined by the points

Find value of $\mathcal{F}$ from figure 2.2

$\mathcal{F} \cong 0.875$

(Figure 2.2 *F*atigue strength fraction – Kpsi Graph)

$N = 10^3$

$S_f = aN^b$

$$a = \frac{(\mathcal{F}S_{ut})^2}{S_e} = \frac{(0.875 \times 560)^2}{143.26} = 1675.97\ MPa \qquad (2-12)$$

$$b = -\frac{1}{3}\log\left(\frac{\mathcal{F}S_{ut}}{S_e}\right) = -0.1780 \qquad (2-13)$$

$$S_f = aN^b = 1675.97 \times (10^3)^{-0.1780} = 490.079\ MPa \qquad (2-14)$$

Shaft output:

After doing the calculations to find the shaft diameter based on the load and power required, we **found that the appropriate shaft diameter is 15 mm**. Made of steel Grade 40C8.

2.7 Bearing calculation:

life (L) in revolutions can be expressed as

$$L_D = L_{10} = l \times n \times 60 \qquad (2-15)$$

Where :

L_D : Desired life in revolutions

L_R : Rating life in revolutions = 10^6

l : life , hours

n : speed , rev/min

X_D : dimensionless multiple of rating life.

Take the desire life = 5000h and N = 180 rpm

$$L_D = L_{10} = 5000 \times 180 \times 60 = 54 \times 10^6 \ \ rev$$

$$X_D = \frac{L_D}{L_R} = \frac{54 \times 10^6}{10^6} = 54 \qquad (2-16)$$

Where:

R : Reliability = 0.90

$F_D = 299.2\ N$

$$C_{10} \doteq a_f F_D \left[\frac{x_D}{x_0 + (\theta - x_0)(1 - R_D)^{1/b}}\right]^{1/a} \qquad R \geq 0.90$$

(Figure 2.3 Dynamic load rating)

Weibull parameters:

$x_o = 0.02$

$(\theta - x_o) = 4.439$

a_f : $application\ factor = 1.25$

At our case the effected load on bearing is vertical (radial) so we will use radial ball bearing.

*For ball bearings in SKF table it has Designation number (6)

for ball bearing:

a = 3

b = 1.483

$$C_{10} = 1.25 \times 299.2 \times \left[\frac{54}{0.02 + (4.439)(1 - 0.90)^{\frac{1}{1.483}}}\right]^{\frac{1}{3}} = 1375.83\ N = 1.375\ K_N$$

(2-17)

After checking the value of **C** dynamic in SKF table is bigger than calculations

From SKF table we select (6002 ball bearing)

bore diameter = 15 mm , outer diameter = 32 mm , width (B) = 9 mm

Designation	Principal dimensions			Basic load ratings		Speed ratings	
				dynamic	static	Reference speed	Limiting speed
	d [mm]	D [mm]	B [mm]	C [kN]	C_0 [kN]	[r/min]	[r/min]
6002	15	32	9	5.85	2.85	50 000	32 000
6002-2RSH	15	32	9	5.85	2.85		14 000
6002-2RSH/VA947	15	32	9	5.85	2.85		14 000
6002-2RSL	15	32	9	5.85	2.85	50 000	26 000
6002-2Z	15	32	9	5.85	2.85	50 000	26 000
6002-RSH	15	32	9	5.85	2.85		14 000
6002-RSL	15	32	9	5.85	2.85	50 000	32 000
6002-Z	15	32	9	5.85	2.85	50 000	32 000
61802	15	24	5	1.9	1.1	60 000	38 000

(Figure 2.4 SKF ball bearing Catalog)

Motor Available in the market:

(SG-77551250000-60k)

That has a Maximum $rpm = 83$ ($no\ load$)

$load\ rpm\ = 63\ rpm$

$Load\ T = 2.79\ N.m$

$At\ maximum\ efficiency\ Torque = 8.53\ \ N.m$

$stall\ Torque = 11.16\ \ N.m$

This motor can carry up to 16 KG at $Torque = 4\ N.m\ \ and\ \ N = 42$, $rpm = 0.7$,

$$Rps = 0.3\frac{m}{s}$$

Maximum speed can be reached at $Torque = 2.79\ N.m$

Carrying Maximum = 11.4 KG per Motor

$N = 63\ rpm = 0.5\ m/s$

Lowest speed at Maximum Load $Torque\ = 8.53\ N.m$

$N\ = 20\ rpm = 0.13\ m/s$

Carrying 35 KG per Motor

Force of Friction on the floor:

At the worst case of concrete floor, the Static Coefficient of friction = 0.6

And for the Kinetic Coefficient of friction = 0.4

μ_s : Static Coefficient of friction

μ_k : Kinetic Coefficient of friction

$$\mu_S = \frac{F_r}{mg} \qquad (2-18) \quad , \quad \mu_K = \frac{F_r}{mg} \qquad (2-19)$$

$R_T\ = raduis\ of\ tire = 0.065m$

T_{Fs} : $Torque\ at\ static$ friction

T_{Fk} : $Torque\ at$ Kinetic friction

F_r : $Force\ to\ move\ object\ at\ constant\ speed$

For Static:

$F_{rs} = m * g * \mu_s = 299.2 * 0.6 = 179.52\ N$ $(2-20)$

$T_{Fs} = F_{rs} * R_T = 180 * 0.065 = 11.7\ N.m$ $(2-21)$

$T_{Fs} > T\ driving\ Shaft$

For the car to move without slipping

For Kinetic:

$F_{rk} = m * g * \mu_k = 299.2 * 0.4 = 119.68\ N$ $(2-22)$

$T_{Fk} = F_{rk} * R_T = 120 * 0.065 = 7.8\ N.m$ $(2-23)$

Friction between Two Tires and Floor:

F_{Ms} : $Max\ friction\ force\ at\ static = 2 * F_{rs} = 360\ N$ $(2-24)$

F_{Mk} : $Max\ friction\ force\ at\ Kinetic = 2 * F_{rk} = 240\ N$ $(2-25)$

a : Acceleration

m : $mass\ of\ the\ car$

$$a = \frac{F_{Ms}}{m} = \frac{360}{122} = 2.9\frac{m}{s^2} \quad (2-26)$$

This the max acceleration that the robot can provide during full load with no slip.

At no load

Static:

m_e : $car\ Wieght\ no\ load = 22KG$

$F_{rs} = m * g * \mu_s = 5.5 * 9.81 * 0.6 = 33\ N$

$T_{Fs} = F_{rs} * R_T = 33 * 0.065 = 2.145\ N.m$

$$a = \frac{F_{rs}}{m_e} = \frac{2 * 33}{22} = 3\ m/s^2$$

Kinetic:

$F_{rk} = m * g * \mu_k = 5.5 * 9.81 * 0.4 = 22\ N$

$T_{Fk} = F_{rk} * R_T = 22 * 0.065 = 1.43\ N.m$

$a = \frac{F_{rk}}{m_e} = \frac{2 * 22}{22} = 2\ m/s^2$

****NOTE : F_D Drag friction force**

We element drag force due to work in door

Friction force for climbing and downgrade:

F_{cr}: *climping and downgrade force*

$F_{cr} = \pm mg(\sin\alpha)$ $(2-27)$

$F_{cr} = \pm mg(\sin\alpha) = \pm 25 \times 9.81 \times \sin 18$ at no load

$F_{cr} = \pm mg(\sin\alpha) = = \pm 140 \times 9.81 \times \sin 18$ Max load

$F_{cr} = 75.8\ N$ *at no load*

$F_{cr} = 424.4\ N$ *Max load*

$T_{cr} = F_{cr} * R_T = 75.8 * 0.065 = 4.927\ N.m$ *No load*

$T_{cr} = F_{cr} * R_T = 687 * 0.065 = 27.586\ N.m$ *Max load*

$F_T = F_{rs} + F_{cr} = 179.52 + 424.4 = 603.92\ N$ *Max load*

$T_{Fr} = 11.7 + 27.586 = 39.286\ N.m$ *at Max load*

$T_{Fr} = 7.8 + 4.927 = 12.727\ N.m$ *at No load*

2.8 Trapezoidal Curve (Theory)

A trapezoidal move profile is used when the application needs to accelerate to a maximum velocity and then travel at that velocity for a specified time or distance. Some common processes that use trapezoidal move profiles are machining, dispensing, and painting.

The simplest form of trapezoidal move profile, and the one used in the examples below, is the 1/3, 1/3, 1/3 profile. In this case, 1/3 of the time is used for accelerating, 1/3 is used for constant velocity, and 1/3 is used for decelerating. But if you understand the geometry of the move profile, which is essentially two triangles and a rectangle, you can calculate the necessary parameters regardless of whether the time segments are equal or not

<u>move 10 m in straight line</u>

at $a_{max} = 2\frac{m}{s^2}$ $V = 1.5\ \frac{m}{s}$

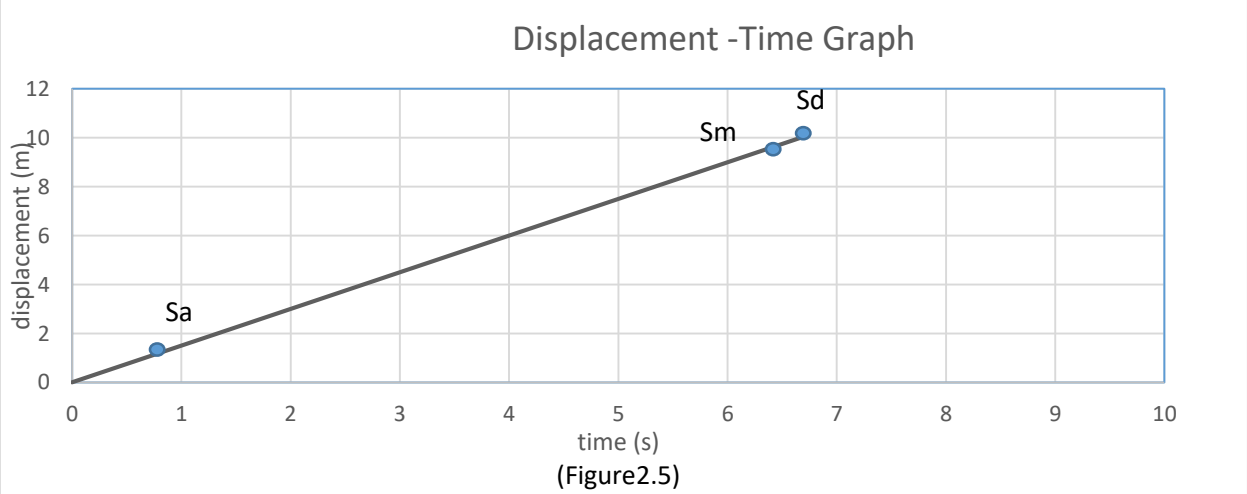
where $t_a = t_d = \frac{V}{a} = \frac{1.5}{2} = 0.75\ sec$ where t_d: $deceleration\ time$

$s_a = \frac{1}{2} \times 1.5 \times 0.75 = 0.6m$ where t_a: $acceleration\ time$

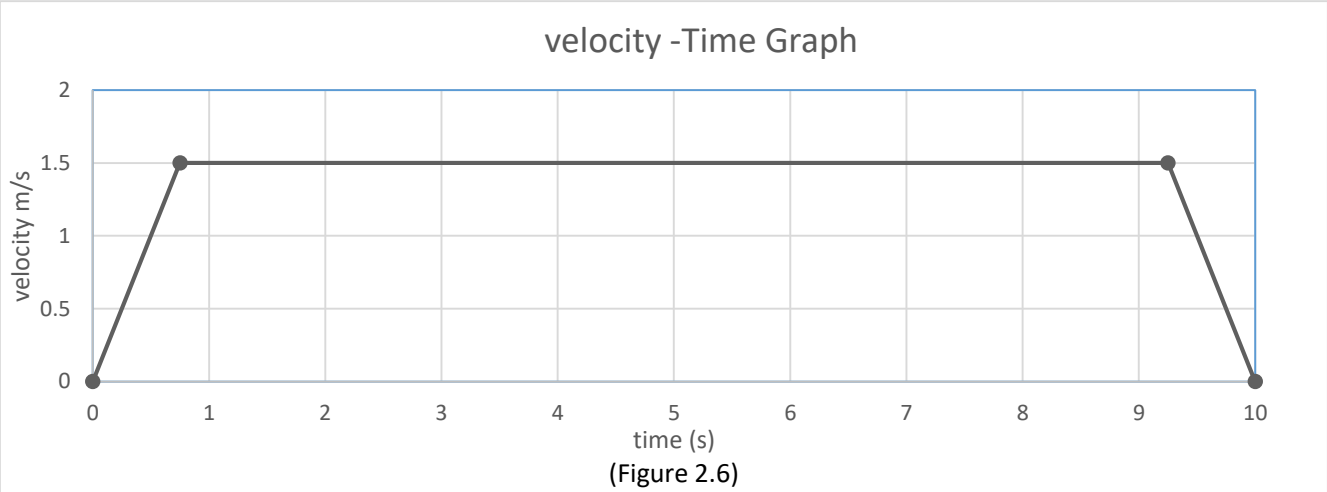
$s_d = 0.6\ m$ where s_a: $Displacement\ covered\ during\ acceleration$

$s_m = 8.8\ m$ where s_d: $displacment\ covered\ during\ deceleration$

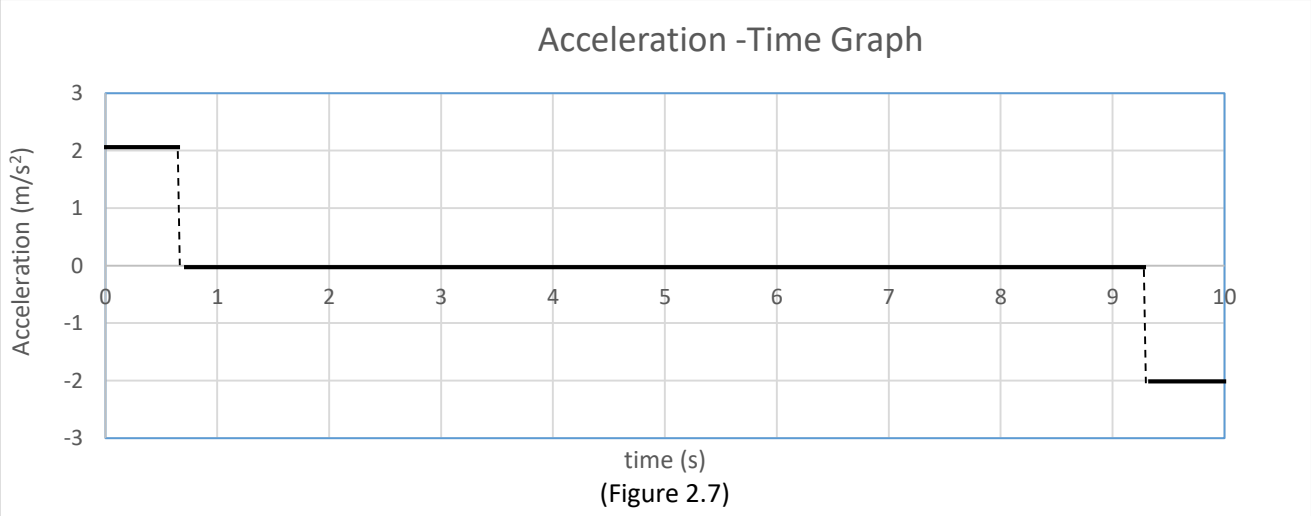
where s_m: $distance\ covered\ during\ motion$



(Figure2.5)



(Figure 2.6)



(Figure 2.7)

a: $Angular\ acceleration$

I_{xx}: $component\ of\ inertia\ in\ x\ axis$

I_{yy}: $component\ of\ inertia\ in\ y\ axis$

I_{zz}: $component\ of\ inertia\ in\ z\ axis$

h: $height$

d: $length$

w: $width$

$$T_{acc} = J_s \times a$$

$$I_{xx} = \frac{1}{12} m(h^2 + d^2)$$

$$I_{yy} = \frac{1}{12} m(d^2 + w^2)$$

$$I_{zz} = \frac{1}{12} m(h^2 + w^2)$$

$$J_{s\ no\ load} = 0.195 + 0.3 + 0.227 = 0.72\ Kg.m^2$$

$$J_{s\ load} = 4.75 + 5 + 3.7 = 13.45\ Kg.m^2$$

$$T_{acc\ (no\ load)} = 3.9\ N.m$$

$$T_{acc\ (load\ 100\ Kg)} = 13\ N.m$$

$$T_{acc\ (load\ 60\ Kg)} = 7.8\ N.m$$

2.9 Trapezoidal Curve for (Actual N75….)

Using our Motor

Move 10 Meter in straight line

At $a = 0.25\ m/s^2$ where t_d: *deceleration time*

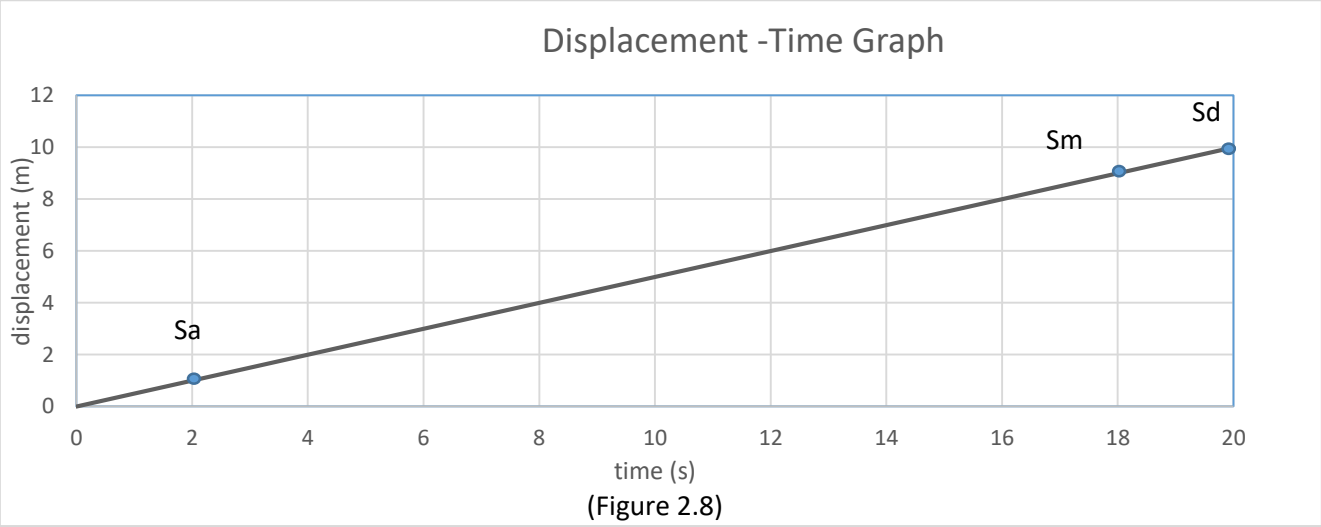
$V_m = 0.5\frac{m}{s}$ where t_a: *acceleration time*

$t_a = t_d = \frac{0.5}{0.25} = 2\ s$ where s_a: *Displacement covered during acceleration*

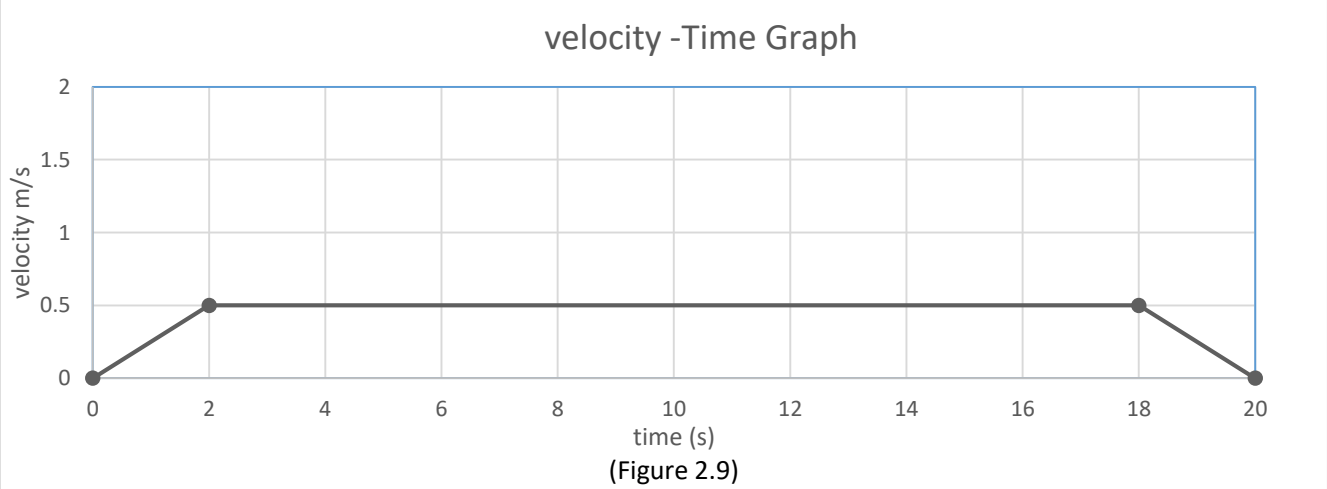
$s_a = \frac{1}{2} \times 0.5 \times 2 = 0.5$ where s_d: *displacment covered during deceleration*

$s_d = 0.5\ m$ where s_m: *distance covered during motion*

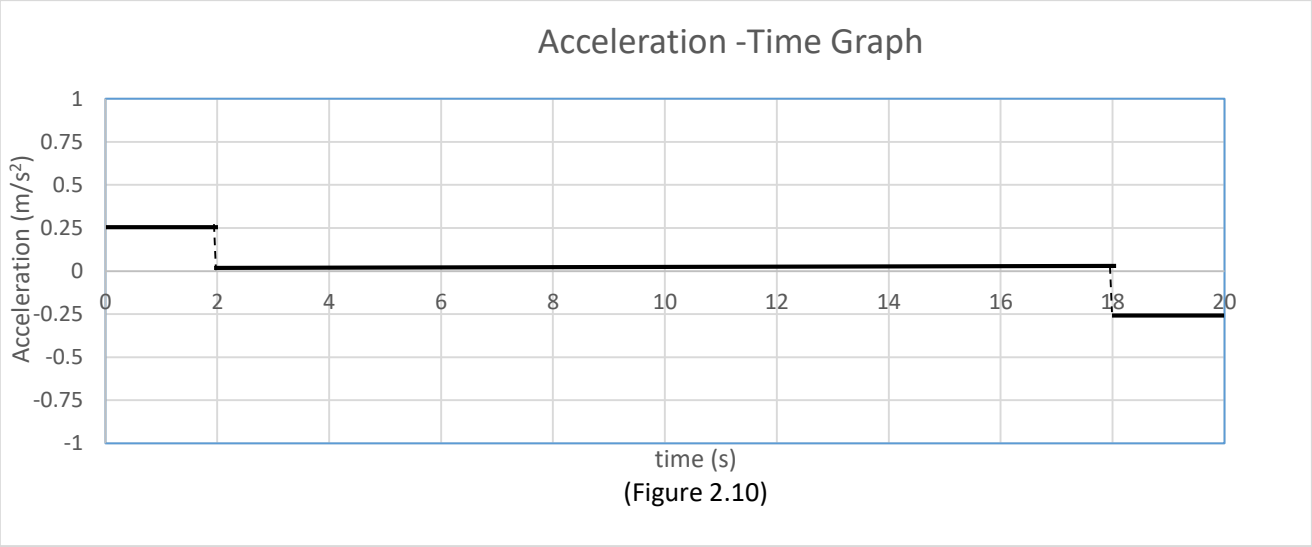
$s_m = 9\ m$



(Figure 2.8)



(Figure 2.9)



(Figure 2.10)

$Js_{no\ load}$: $inertia\ of\ the\ system\ at\ no\ load$

Js_{load}: $inertia\ of\ the\ system\ at\ full\ load$

$Tacc$: $acceleration\ torque$

T_{total}: $Total\ torque$

$Js_{no\ load} = 0.72\ Kg.m^2$

$Js_{load}\ at\ 30\ Kg = 3 + 2.8 + 2.2 = 8\ Kg.m^2$

$Js_{load}\ at\ 60\ Kg = 3.3 + 4.5 + 4.2 = 12\ Kg.m^2$

No load

$Tacc = 0.56\ N.m$ $\quad T_{total} = T_l + T_{acc} = 2.87 + 0.56 = 3.43\ N.m$

Loaded at 30 Kg

$Tacc = 1.2\ \ N.m$ $\quad T_{total} = 5 + 1.2 = 6.2\ \ N.m$

Loaded at 60 Kg

$Tacc = 1.5\ \ N.m$ $\quad T_{total} = 8.5 + 1.5 = 10\ N.m$

2.10 Ball Screw system:

We selected the accuracy grade from the table (2.6) of lead angle accuracy, selected C7 grade

Because it is the most popular and is in the table of accuracy grade select for linear actuators

The shaft diameter = 12 mm (for availability) and shaft length 20cm from table (5)

Take the lead = 2 mm from table (5) which for every full rotation the nut will move 2 mm linearly

(TABELS IN NEXT PAGES)

Stepper motor (NEMA23) which has $N = 18\ rpm$ $(57H, 1.8°)$

Axial load $= 220$ N $and\ the\ lead = 2\ mm$

$$Torque = \frac{Axial\ \ load * lead}{2000 * \pi * 0.8} \qquad (2-28)$$

$0.8\ refer\ to\ efficiency\ of\ ball\ screw$

$$Torque = \frac{220 * 2}{2000 * \pi * 0.8} = 0.08\ \ N.m\ \ approx.\ \ \ T = 0.1\ N.m$$

For the (NEMA 23) $Torque = 1.3\ N.m$ ($more\ than\ enough$)

One of the problem that faces BALL SCREW System is Backlash

Is an axial and angle play because of clearness between the ball-nut and screw, we can eliminate the backlash by pre-loading the ball-screw, by applying a fixed load on the nut so that the ball will not move in axial direction.

(Table 2.6 Recommended accuracy grade)

	Application grade	AXIS	Accuracy grade									
			0	1	2	3	4	5	6	7	8	10
CNC Machinery Tools	Lathes	X	•	•	•	•	•					
		Z				•	•	•				
	Milling machines Boring machines	X		•	•	•	•	•				
		Y		•	•	•	•	•				
		Z			•	•	•	•				
	Machine Center	X		•	•	•	•					
		Y		•	•	•	•					
		Z			•	•	•					
	Jig borers	X	•	•								
		Y	•	•								
		Z	•	•								
	Drilling machines	X				•	•	•				
		Y				•	•	•				
		Z					•	•	•			
	Grinders	X	•	•	•							
		Y		•	•	•						
	EDM	X		•	•	•						
		Y		•	•	•						
		Z			•	•	•	•				
	Wire cut EDM	X		•	•	•						
		Y		•	•	•						
		U		•	•	•	•					
		V		•	•	•	•					
	Laser Cutting Machine	X			•	•	•					
		Y			•	•	•					
		Z			•	•	•					
General Machinery	Punching Press	X				•	•	•				
		Y				•	•	•				
	Single Purpose Machines			•	•	•	•	•	•			
	Wood working Machines								•	•	•	•
	Industrial Robot (Precision)			•	•	•	•					
	Industrial Robot (General)							•	•	•	•	
	Coordinate Measuring Machine		•	•	•							
	Non-CNC Machine					•	•	•				
	Transport Equipment						•	•	•	•	•	•
	X-Y Table			•	•	•	•	•				
	Linear Actuator							•	•	•	•	
	Aircraft Landing Gear							•	•	•	•	
	Airfoil Control							•	•	•	•	
	Gate Valve								•	•	•	•
	Power steering								•	•	•	
	Glass Grinder				•	•	•	•	•			
	Surface Grinder						•	•				
	Induction Hardening Machine								•	•	•	•
	Electromachine			•	•	•	•	•	•			
	All-electric injection molding machine							•	•	•	•	•

(Table 2.7 standard of shaft diameter and lead for roller ball)

Screw shaft outer diameter	Lead																		
	1	2	4	5	6	8	10	12	16	20	24	25	30	32	36	40	50	60	80	100
6	●																			
8		●																		
10		●			○															
12		●				○														
14			●	●																
15							●			●			●							
16				●					●											
18						●														
20				●			●			●						●				
25				●			●					●					●			
28					●															
30																		●		
32							●							●						
36							●			●	●				●					
40							●									●			●	
45								●												
50									●								●			●

●: Standard stock
○: Semi-standard stock

(Table 2.8 lead angle accuracy)

		Precision Ball Screw												
												Rolled Ball Screw		
Accuracy grades		C0		C1		C2		C3		C5		C7	C8	C10
Effective thread length		Representative travel distance error	Fluctuation	Representative travel distance error	Fluctuation	Representative travel distance error	Fluctuation	Representative travel distance error	Fluctuation	Representative travel distance error	Fluctuation	Travel distance error	Travel distance error	Travel distance error
Above	Or less													
—	100	3	3	3.5	5	5	7	8	8	18	18			
100	200	3.5	3	4.5	5	7	7	10	8	20	18			
200	315	4	3.5	6	5	8	7	12	8	23	18			
315	400	5	3.5	7	5	9	7	13	10	25	20			
400	500	6	4	8	5	10	7	15	10	27	20			
500	630	6	4	9	6	11	8	16	12	30	23			
630	800	7	5	10	7	13	9	18	13	35	25			
800	1000	8	6	11	8	15	10	21	15	40	27			
1000	1250	9	6	13	9	18	11	24	16	46	30	±50/ 300mm	±100/ 300mm	±210/ 300mm
1250	1600	11	7	15	10	21	13	29	18	54	35			
1600	2000	—	—	18	11	25	15	35	21	65	40			
2000	2500	—	—	22	13	30	18	41	24	77	46			
2500	3150	—	—	26	15	36	21	50	29	93	54			
3150	4000	—	—	30	18	44	25	60	35	115	65			
4000	5000	—	—	—	—	52	30	72	41	140	77			
5000	6300	—	—	—	—	65	36	90	50	170	93			
6300	8000	—	—	—	—	—	—	110	60	210	115			
8000	10000	—	—	—	—	—	—	—	—	260	140			

Note) Unit of effective thread length: mm

(Table 2.9 Maximum length of the rolled ball screw by accuracy grade)

Unit: mm

Screw shaft outer diameter	Overall screw shaft length		
	C7	C8	C10
6 to 8	320	320	—
10 to 12	500	1000	—
14 to 15	1500	1500	1500
16 to 18	1500	1800	1800
20	2000	2200	2200
25	2000	3000	3000
28	3000	3000	3000
30	3000	3000	4000
32 to 36	3000	4000	4000
40	3000	5000	5000
45	3000	5500	5500
50	3000	6000	6000

Calculation for thrust bearing for ball screw system:

Thrust ball bearings, composed of bearing balls supported in a ring

Thrust ball bearings are designed to take axial (thrust) loads

In our ball screw system, the lead screw diameter is 12 mm, and the load Is pure Axial load so we used a Thrust ball bearing and will show the calculation in next pages

Basic Data for calculations:

1- Speed of the motor 180 rpm

2- System duty is 8 hours but will work intermittently

3- Load is Axial = 300 N

4- According to the lead screw diameter (12 mm) we choose -51101 Single direction thrust ball bearing

Dimensions

d	12 mm	Bore diameter
D	26 mm	Outside diameter
H	9 mm	Height

(Figure 2.11 dimensions of bearing)

Step 1 pre check:

*The Minimum Requisite load

*Speed limit

The Minimum Requisite load = Minimum load factor (Kr) x Basic Dynamic load rating(C)

(2-29)

* Applied Load over the bearing > Minimum Requisite load

From data sheet Kr = 0.0014, C = 10.4 KN

(Table 2.10 data of bearing)

Basic dynamic load rating	C	10.4 kN
Basic static load rating	C_0	16.6 kN
Fatigue load limit	P_u	0.62 kN
Reference speed		9 000 r/min
Limiting speed		13 000 r/min
Minimum load factor	A	0.0014

So Minimum Requisite load = 0.0014 x 10.4 = 0.01456 KN

Our load = 300 N, 0.3 KN > 0.01456 KN ✓

Speed limit:

Bearing max. speed should be lower than (Reference speed, Limiting speed)

our speed is 180 RPM < (9000 RPM, 13000 RPM) ✓

Step 2 Basic Rating life L10:

Where L10 is Basic Rating life $$L10 = \left(\frac{C}{P}\right)^{p} \qquad (2-30)$$

C is Basic Dynamic load rating

P is Equivalent dynamic load

p is = 3 for ball bearing, 3.33 for Roller bearing

$$P = XFr + YFa \qquad (2-31)$$

Where:

Fr Actual Radial load (KN)

Fa Actual Axial load (KN)

X Radial load factor

Y Axial load factor

*At our case load is pure Axial so $P = Fa = 300\ N = 0.3\ KN$

Step 2 Basic Rating life L10:

find Basic Rating life = $L10 = \left(\frac{10.4}{0.3}\right)^{3} = 41661.62\ M\ (rev)$

Now compare our result to the Recommended life of the bearing

- **Should meet or exceed the Recommended value of the recommended life**

Machine type	Specification life Operating hours
Household machines, agricultural machines, instruments, technical equipment for medical use	300 ... 3 000
Machines used for short periods or intermittently: electric hand tools, lifting tackle in workshops, construction equipment and machines	3000 ... 8 000
Machines used for short periods or intermittently where high operational reliability is required: lifts (elevators), cranes for packaged goods or slings of drums, etc.	8 000 ... 12 000

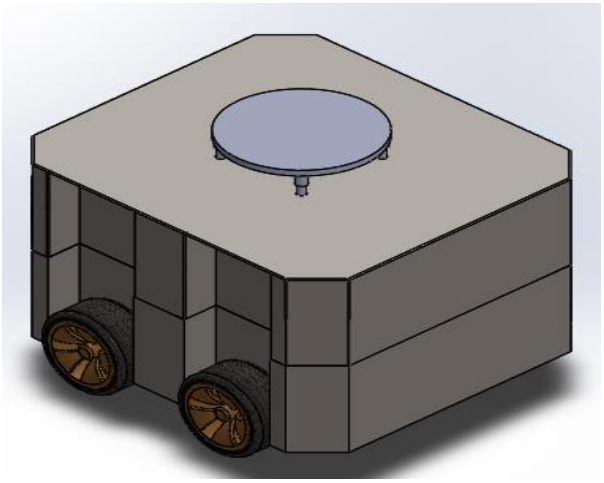
(Figure 2.12 specification life according to Machine type)

Find Operating hours = L10h $= \frac{10^6}{60\ x\ n}\ x\ L10,$ L10h $= \frac{10^6}{60\ x\ 180} x\ 41661 = 3857500\ Hrs.$

2.11 Parts Drawing and Assembly:

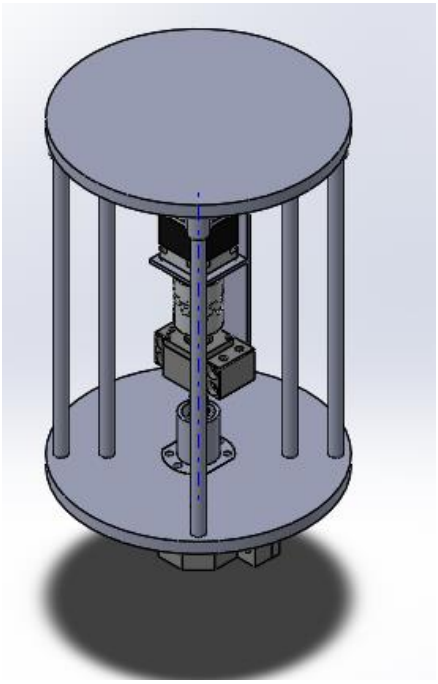
We used (Solid works) software to draw and Assembly parts

1- Full Assembly for Our AMR Robot



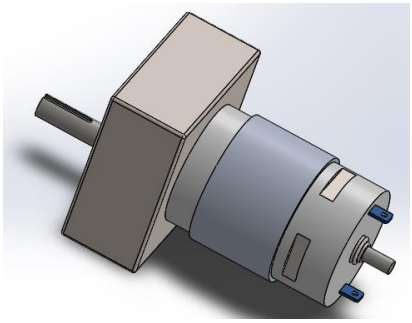
(Figure 2.13 Full Assembly for Our AMR Robot)

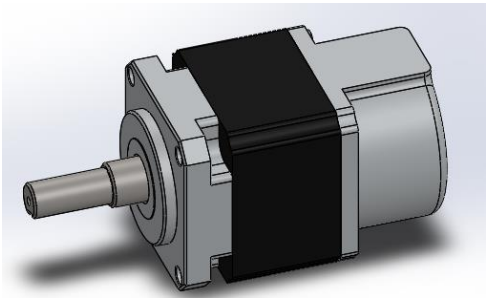
2- Ball Screw System showing its components: Ball Screw Shaft, The Nut, Stepper motor, Top plate, Base plate with 8 rods.



(Figure 2.14 Ball Screw System showing its components)

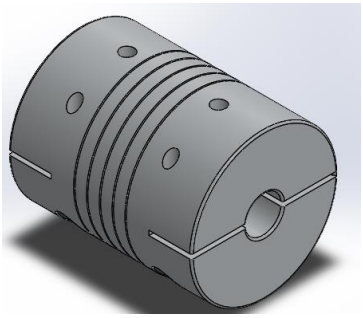
2- Dc Motor for Wheels and Stepper Motor for Ball Screw System.





(Figure 2.15 Dc Motor for Wheels and Stepper Motor for Ball Screw System)

3- Flexible Coupling which connects the dc Motor to the shaft attached to the wheel.



(Figure 2.16 Coupling which connects the dc Motor to the shaft)

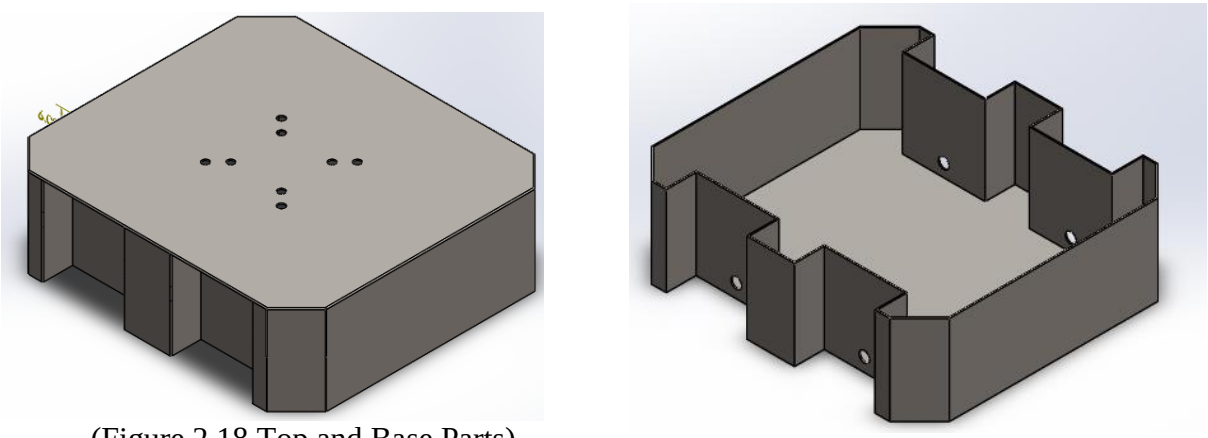
5- wheel It consists of two parts: the outer tire is made of rubber and the rim is usually a metal material

(Figure 2.17 wheel consists of two parts: the outer tire is made of rubber and the rim)

6- Top and Base Parts

At the Top Part there are 8 holes for the ball screw system to allow it to move up and down.

The Base part has a general shape that resembles a rectangle with four holes for the shaft that connects the wheel and the flexible coupling



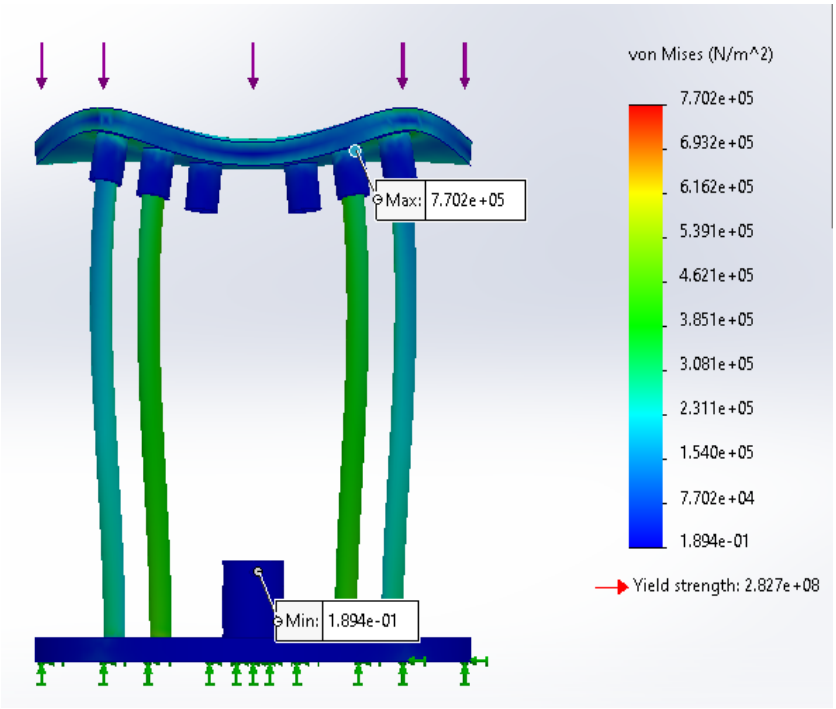
(Figure 2.18 Top and Base Parts)

Stress Analysis:

In figure 2.19 shows the maximum and minimum stress value when apply 30 kg load and the yield strength for our material. =2.827 × 10^8 N/m^2

Max = 7.702×10^5 N/m^2

Min= 1.894×10^{-1} N/m^2

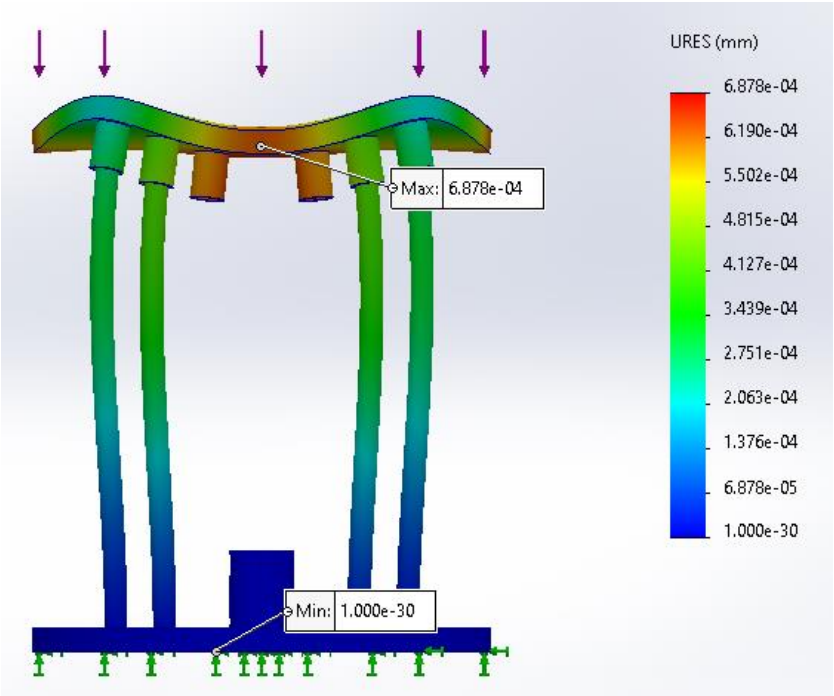


(Figure 2.19 stress analysis from SOLIDWORKS software)

In figure 2.20 shows the maximum and minimum Displacement after apply the load.

Max = 6.878×10^{-4} mm

Min= 1.000×10^{-30} mm



(Figure 2.20 displacement analysis from SOLIDWORKS software)

The mechanical design chapter shows the key considerations and processes involved in developing the physical hardware components of the project. The design ensures the overall structural strength and durability of the system to withstand expected loads, stresses, and environmental conditions. The mechanical structure enables modular assembly and disassembly to facilitate maintenance, upgrades, and debugging.

Chapter. 3

Electrical circuit: -

Introduction on the Electrical circuit and the component: -

3.1 Components:

1-raspberry pi 4 b module 8 Giga ram

2- Five DC-motors (SG775125000)

3- Two Cytron 10Amp 5V-30V DC Motor Driver (2 Channels) MDD10A (motor driver)

4- Two voltage step-down modules (12v-5v) (3A)

5- Camera Kinect 360

6- Delta2A 360 Degree LiDAR Sensor for ROS Robot Module Scanner Measuring Sensor Range 8m

7- Four encoders

8- IMU MPU

9- ESP 32

10-Battery

11- Battery management system

12- Arduino UNO for Ball screw system

3.2 Components description: -

1-raspberry pi 4 b module 8 Giga Ram

(Table 3.1 raspberry pi 4 b information)

General information	Product name	Raspberry Pi 4 Model B 8 GB
Processor	**CPU frequency**	**1.5 GHz**
	Number of cores	**4**

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 and 4 Gigabytes of LPDDR4 SDRAM.

(Figure 3.1 Raspberry Pi 4 Model B)

Positives: -

Vast Peripheral Support

1- Raspberry Pi comes with 26 GPIO Pins which are very useful indeed for embedded projects and interfacing hardware. These pins are really useful in learning about component interfacing. You can combine multiple digital sensors all together due to the good number of GPIO pins being given. It supports almost all the peripherals supported by Arduino.

It has a lot of accessories available for it in the market. You will find many Raspberry Pi cases with different designs, Raspberry pi HATs, Fans, Heat Sinks, etc... You will find a whole big community and support as well. It is said to be the most popular single-board computer of this era.

Multiple Sensors

2- As discussed in the above section that it comes with a lot of GPIO Pins, so it is obvious for it to support multiple sensors at once. You can connect various displays, modules, sensors, etc...to it. Unless it's not analog.

Supports all type of Codes

3- It's one of the best part of this board, if we compare it with Arduino you will know that Arduino only supports C, C++. While this board works as a single board computer. You get a Linux desktop environment in which you can code in almost any language, be it C, C++, C#, Ruby, Java , Python , etc...

This Support for all types of code makes this board famous, one of the main goal of the raspberry pi foundation was to provide cheap computing to people, so that they can learn programming. They have really reached their goal in providing cheap computing to people, so they can learn programming easily.

Moving towards the future, everything is turning digital, so we need more and more programmers. Raspberry Pi is really helping people who cannot spend much for desktop computers.

Faster Processor

4- Talking about the core. When we compare it with Arduino and other boards, you get a faster processor. Arduino comes with a controller, while here Raspberry Pi comes with a 1.6 GHz Processor in the 4B variant of Raspberry Pi.

Faster processor means good performance. The price to performance of raspberry pi board is really great. I bet you won't get that much performance on any board at that price.

Can be Used as a Portable Computer

5- You can do all sorts of stuff on the raspberry pi which are possible on a Linux distro. You will find many apps and packages that will help you do lots of tasks on the Pi, like photo editing, coding, etc. Many popular apps like Google Chrome and VLC are also available in Raspbian.

Negatives: -

Missing eMMC Internal Storage

1-Since the raspberry pi doesn't have any internal storage it requires a micro SD card to work as an internal storage. We all know that SD cards are not that fast. Even if we compare an class 10 High Speed micro SD card with an eMMC internal storage. It lacks performance, so this increases boot time of the board and read/write speed of the raspberry pi. Many board manufacturers like Beagle-bone and Asus Tinker board are now using eMMC internal storage for higher speed. They also give an option to expand internal storage with an external SD card. The boot time of such board is very less and super speedy. It think raspberry pi team should give a thought about this one in their next coming board.

Graphics Processor Missing

2-Well graphics process is a very crucial thing, if you're into photo editing, video editing and gaming. Without it your Computer is just a potato. Many of us need a graphics processor so we can do certain tasks. While the raspberry pi doesn't come with a GPU unit. The processor does all the task for it, which is inefficient.

Asus Tinker board comes with a graphics processor. You can play android games on the board while installing android OS easily. You can edit photos and videos faster.

Overheating

3- As the board doesn't come with any heat-sinks pre-applied or any cooling fan. As the raspberry pi 4 comes with a powerful processor and multiple features, it starts to heat up after sometime due to the same board size, the heat dissipation is not proper as expected. If you use it for continuous 6-7 hours without air-conditioning or heat-sink. It will heat up very much above 70 ° C if you are in south Asian region.

Not able to run Windows Operating system

4-Many people will argue about this, that it is able to run windows OS on it. But the fact is , that it is just a community made windows 10 port. Which is not an official release. It will crash a lot and there will be many bugs. Windows operating system is the most user friendly as we know, for gaming its the primary OS. You basically can't play games on Linux systems practically. Many apps are available for windows OS because of the ".exe" format support. We have alternative apps available on Linux, but many popular software developers use the .exe formats.

2-**DC-motors**

Our motors uses 12 volts and the current at no load is 0.5ampers And then the current ranges between 4A and 8A during working and has a maximum stall current of 14 A

3- **Cytron 10Amp 5V-30V DC Motor Driver (2 Channels) MDD10A (motor driver)**

The key aspect of selecting a suitable brushless DC motor driver lies in ensuring that the parameters of both the motor and the driver are compatible. This involves considering two aspects - the parameters of the DC motor and those of the driver itself.

Features:

- Bi-directional control for two brushed DC motors.
- Support motor voltage ranges from 5V to 30VDC (Rev2.0).
- No Reverse Polarity Protection at Motor, **please double-check the polarity before power-up**.
- Maximum current up to 10A continuous and 30A peak (10 seconds) for each channel.
- Support 3.3V and 5V logic level input (for PWM and DIR), compatible with Arduino and Raspberry Pi.
- Solid state components provide faster response time and eliminate the wear and tear of mechanical relay.
- Full NMOS H-Bridge for better efficiency and no heat sink is required.
- Regenerative Braking.
- Speed control PWM frequency up to 20KHz (output frequency is same and input frequency).
- Support both Locked-Antiphase and Sign-Magnitude PWM operation. ** Note this is not "RC PWM"
- 2 activation buttons for manual activation or fast test on each channel.
- Dimension: 84.5mm x 62mm

APPLICATIONS:

- DC motor driver
- 4WD high-power mobile robot platform
- Combat robots
- Pumps
- Electric fans
- Conveyors

4- Two voltage step-down modules (12v-5v) (3A)

We need two modules to step down the voltage from 12V to 5v/2A-3A

For the Lidar and the raspberry pi

5- Camera Kinect 360

The Kinect camera is an RGBD/ device with a resolution of 640×480 and a 24 bit color range (Red-Green- Blue channels). Working at a rate of 30 frame captures per second this camera, this camera is similar to the run of the mill webcam or the sensors in your digital camera and it is, in most regards, very common place. The depth and motion sensing technology at the core of the Kinect is enabled through its depth-sensing. The original Kinect for Xbox 360 used structured light for this: the unit used a near-infrared pattern projected across the space in front of the Kinect, while an infrared sensor captured the reflected light pattern. The depth and motion sensing technology at the core of the Kinect is enabled through its depth-sensing. The original Kinect for Xbox 360 used structured light for this: the unit used a near-infrared pattern projected across the space in front of the Kinect, while an infrared sensor captured the reflected light pattern.

6 - Delta2A 360 Degree LiDAR Sensor for ROS Robot Module Scanner Measuring Sensor Range 8m

Delta-2A is a new generation low cost, low power consumption LiDAR jointly developed by 3irobotics with Han's Laser. Equipped with the self-developed and patented wireless power transmission and communication technology, this product outreached the traditional LiDAR life limits and realized long time stable operation. Delta-2A LiDAR can perform 2D 360 degree scanning at frequency 2~5kHz within a 8 meter range. The generated 2D point cloud data can be used in mapping, localization, navigation and object/environment modeling.
The typical scanning frequency of Delta-2A is 4~10hz(360rpm), at which the angular resolution is 0.3~0.8°.
It performs excellent in all kinds of indoor environments and outdoors environment with no direct sunlight. Besides, each LiDAR pass through strict inspection to make sure the laser power conforms with FDA Class I human eye grade and is harmless to people and pets.

Modern LIDAR systems typically have frame rates between 10 and 30 frames per second

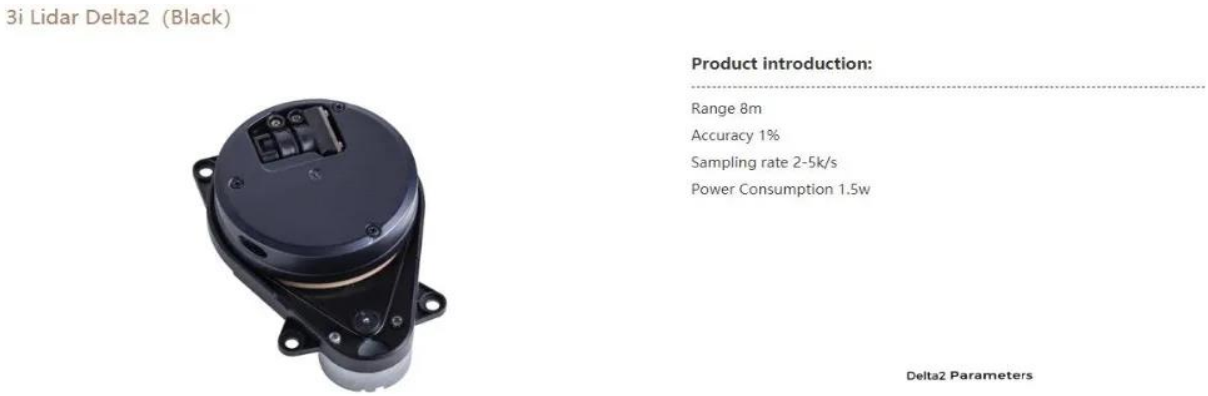
Product Parameters

Range	0.13m-8m (White Wall)	Sampling Rate	5K/s
scanning frequency	6.2Hz	Laser wavelength	780nm
Laser power	3mW (The most)	Accuracy	<1%@5m
Distance Variance Coefficient	DCV<0.2%	Communication Interface	RS232 (TTL)
Voltage	5V	Power consumption	1.5W
Baud rate	230400	Working current	500mA
weight	175g±2g	volume	107*76*53mm
Working temperature	0°C-45°C	Level	0°- 1°
Working environment humidity	<90%	Working environment illumination	<1000lux

Communication and interface

Operating mode	8-bit data, 1 stop bit, no parity		
Baud rate	230400	Output low level	<0.4
Output high level	2.9--3.5		

(Figure 3.2 LIDAR parameters)



(Figure 3.3 LIDAR parameters)

7- Four encoders magnetic

Two channel hall effect encoder. Operation
Voltage 4.5~24V Operation
Current 14~20mA
(7PPR)The encoder mainly installs on the motor of SG775125000 family (Available @ RAM Electronics).

SEARCH KEY: DC.SG775

8- IMU MPU

The MPU-6050 measures acceleration over the X, Y, and Z-axis. Ideally, in a static object, the acceleration over the Z-axis is equal to the gravitational force, and it should be zero on the X and Y-axis. Using the accelerometer's values, it is possible to calculate the roll and pitch angles using trigonometry.

9- ESP 32

ESP32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth. The ESP32 series employs either a Tensilica Xtensa LX6 microprocessor in both dual-core and single-core variations, Xtensa LX7 dual-core microprocessor or a single-core RISC-V microprocessor and includes built-in antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power-management module

10-Battery

For each motor per hour between 4000 mA – 8000 mA and has a stall current of 14000 mA from data sheet of the motor.

$P = VI \quad where\ P\ is\ (power) \quad V\ is\ (voltage\ )\ \ I\ is\ (\ current)$
$P = 12 \times 4 = 48\ watt \qquad at\ current = 4000mA$
$P = 12\ \times 8 = 96\ watt \qquad at\ current = 8000mA$
Power for five motors in three cases according to load on the motor
$5\ \times 4 = 20A \quad at \quad 4000mA$
$5\ \times 6 = 30A \quad at \quad 6000mA$
$5\ \times 8 = 40\ A\ \ at \quad 8000mA$

For raspberry pi = 5 V, 2-3 A
Kinect camera = 12V, 1.01 – 1.5 A
Lidar = 5 V, 0.5 A

Total = 26 Ah in case $4000mA$
Take Battery capacitance = 32 Ah (Work for one hour)
The Robot consumes about 256 Ah for 8 hours' shift

Total = 36 Ah in case $6000mA$
Take Battery capacitance = 44 Ah (Work for one hour)
The Robot consumes about 352 Ah for 8 hours' shift

Total = 46 Ah in case $8000mA$
Take Battery capacitance = 55 Ah (Work for one hour)
The Robot consumes about 440 Ah for 8 hours' shift

At each state we used a safety of twenty percent in battery capacity so that the AMR would recharge at 20% of the battery and won't reach zero for the battery health and life time to be maintained

(Table 3.2 Battery capacity in different cases)

Case of the motor	Current in mA	consumption for 1 hour for all component	consumption for 8 hours for all component
1	$\boldsymbol{4000mA}$	32 Ah	256 Ah
2	$\boldsymbol{6000mA}$	44 Ah	352 Ah
3	$\boldsymbol{8000mA}$	55 Ah	440 Ah

11-Battery management system (BMS)

Battery management system (BMS) is technology dedicated to the oversight of a battery pack, which is an assembly of battery cells, electrically organized in a row x column matrix configuration to enable delivery of targeted range of voltage and current for a duration of time against expected load scenarios.

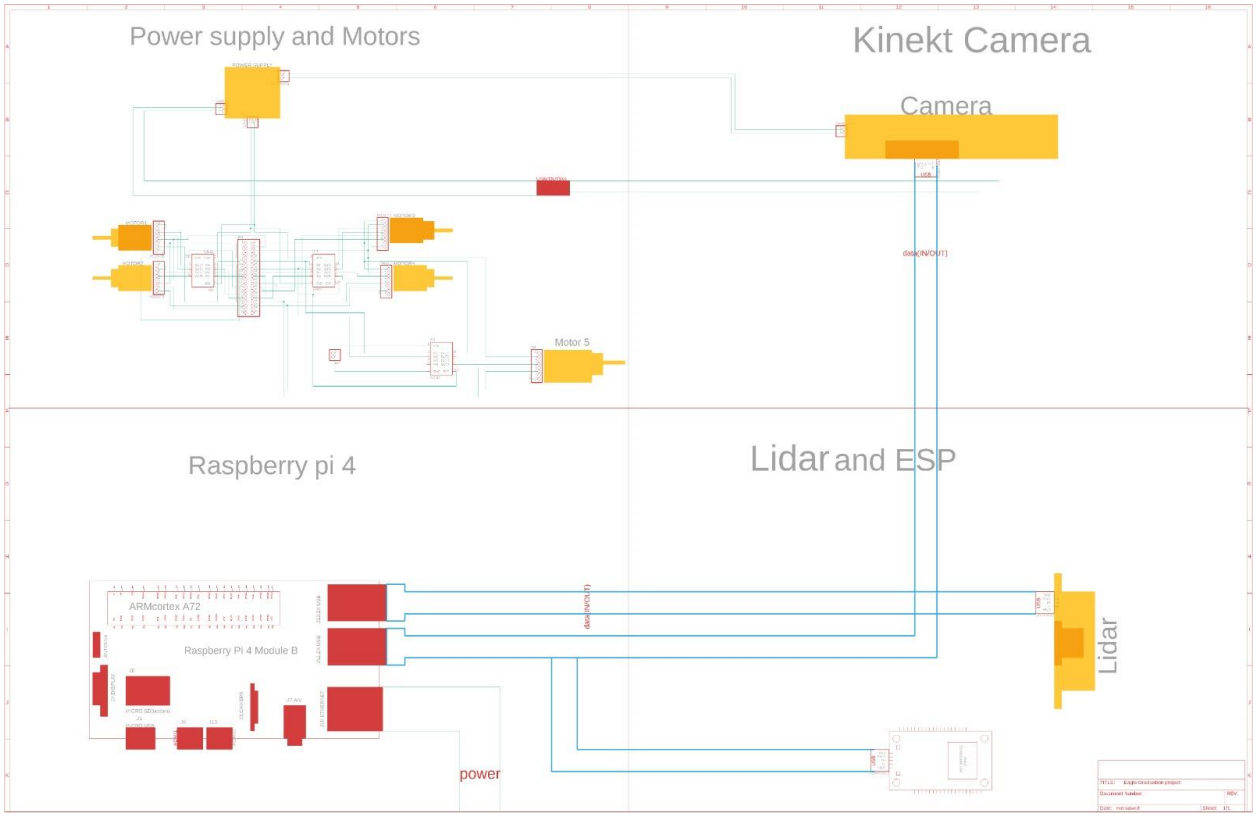
The oversight that a BMS provides usually includes:

- Monitoring the battery
- Providing battery protection
- Estimating the battery's operational state
- Continually optimizing battery performance
- Reporting operational status to external devices

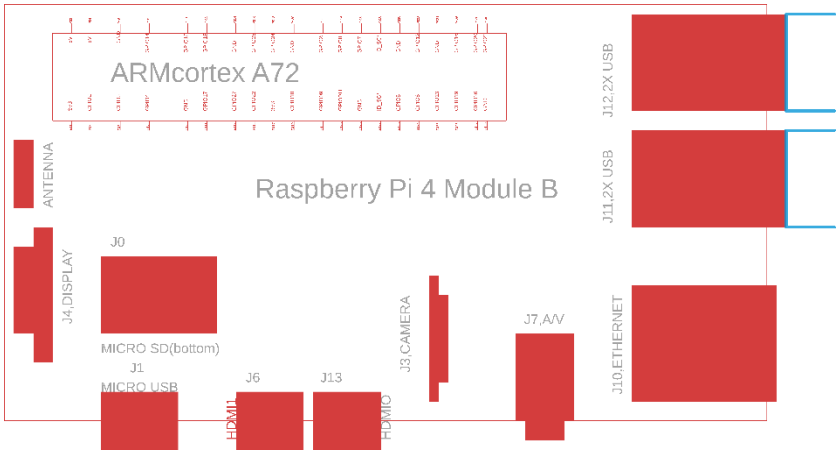
(Figure 3.4 example for Lithium Battery Charger Protection)

3.3 schematic diagram:

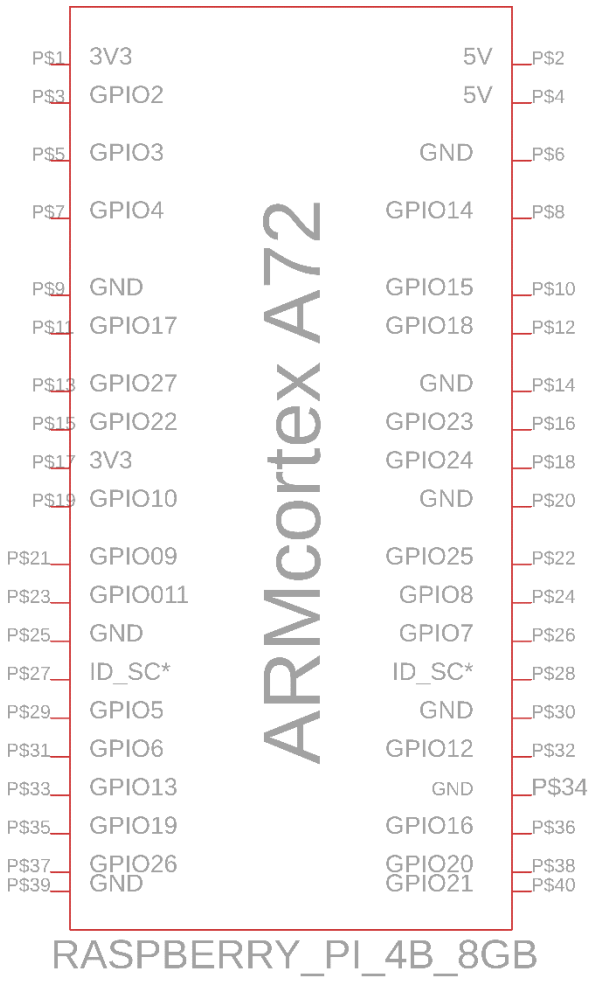
using EAGLE software



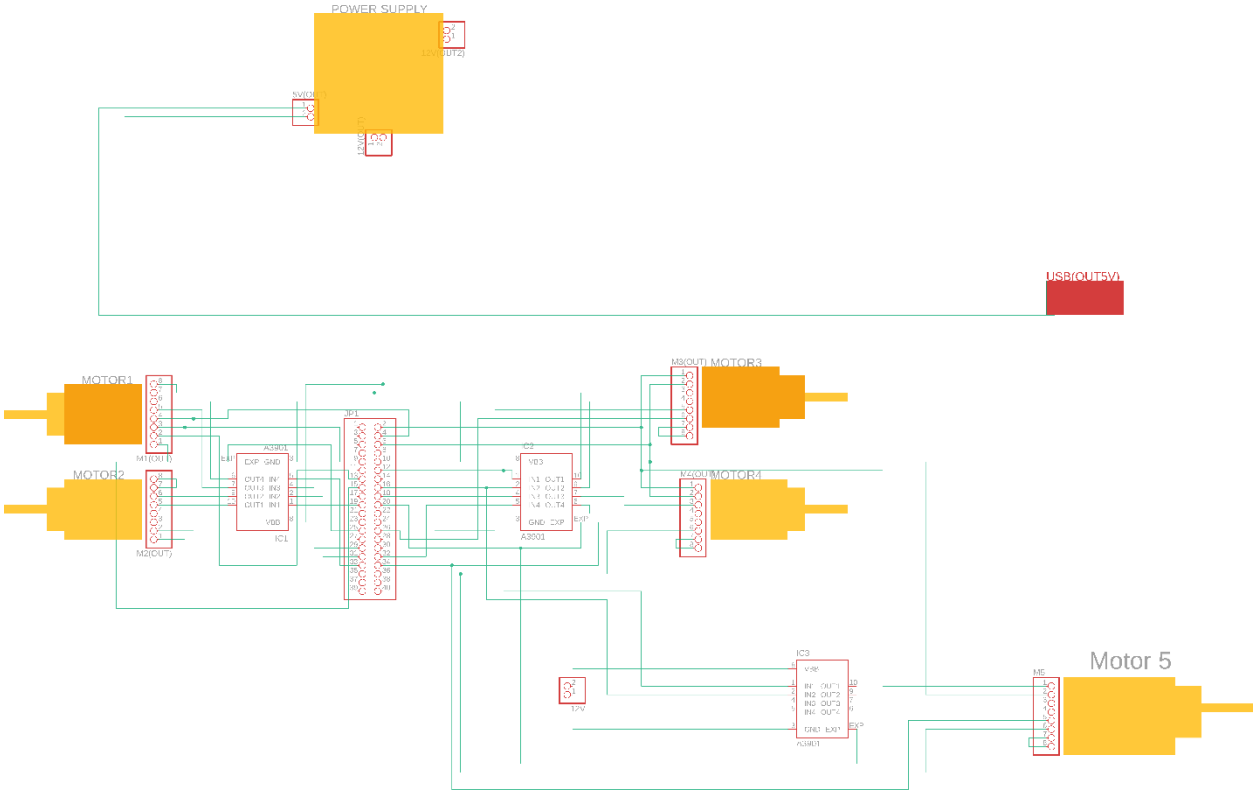
(Figure 3.5 schematic diagram on EAGLE)



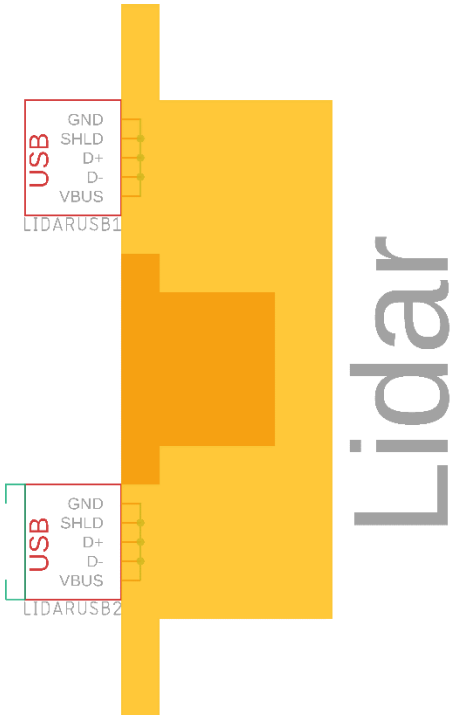
(Figure 3.6 schematic diagram of Raspberry Pi 4 Model B)



(Figure 3.7 Raspberry Pi 4 Model B pin out diagram)

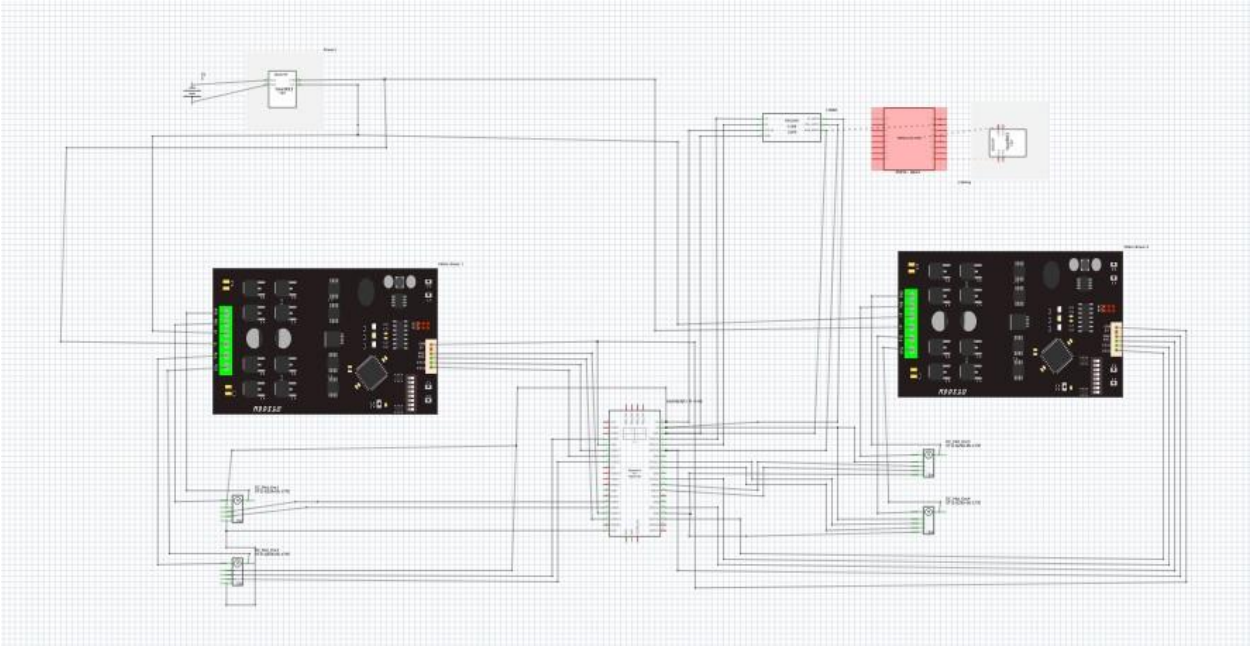


(Figure 3.8 Motor connection)



(Figure 3.9 Lidar pin out diagram)

Schematic on fritzing



(Figure 3.10 Schematic on FRITZING)

A schematic diagram was shown in the previous pictures, which shows the connection between the elements that we will work with. We have the Raspberry Pi4 & the camera & LIDAR sensor & Motor drivers, Battery and an ESP32. As for connecting the motor driver, each motor driver controls two motors, as both motors are connected to the ports designated for them. out 1 and out 2 for the first motor and out 3 and out 4 for the second motor. The motor driver is connected to the power source with two outputs. The first is Vcc pin for 12 volts and the second is GND pin. Then we have five pins that we connect to ESP32 for each motor there is five (PWM1, PWM2, Dir1, Dir2, GND)

Our motors come with encoder that has 4 pins two of them for voltage and ground from ESP32 and the other 2 pins Channel A and B.

Motor 1 Encoder A pin 5

Motor 1 Encoder B pin 23

Motor 1 PWM pin 27

Motor 1 Dir pin 33

Motor 2 Encoder A pin 12

Motor 2 Encoder B pin 13

Motor 2 PWM pin 26

Motor 2 Dir pin 32

Motor 3 Encoder A pin 16

Motor 3 Encoder B pin 17

Motor 3 PWM pin 14

Motor 3 Dir pin 4

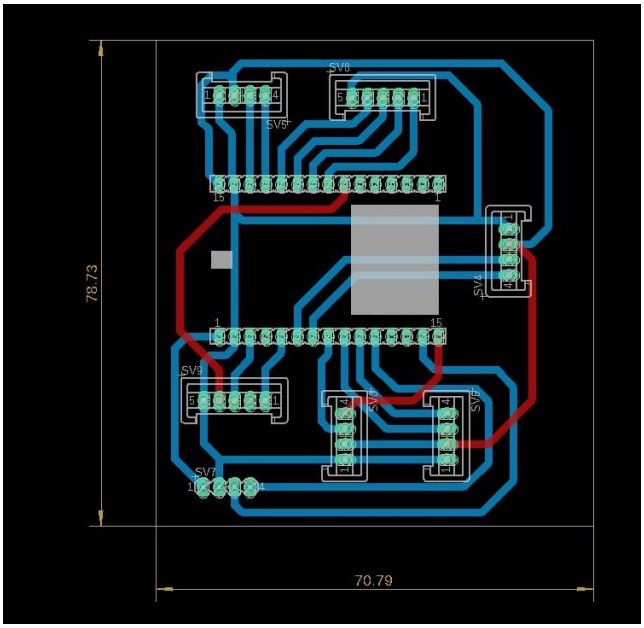
Motor 4 Encoder A pin 19

Motor 4 Encoder B pin 18

Motor 4 PWM pin 25

Motor 4 Dir pin 15

After connect the motors to motor driver and the driver to the ESP 32 connect ESP to Raspberry pi through USB cable. We used a PCB to make these connections



(Figure 3.11 PCB)

Advantages of PCB:

1. Compact Size
2. Easy to Diagnose and Repair
3. Time-Saving
5. Strong Connection

Chapter.4

Situable software

4 1.HOW TO CHOOSE ROBOT BRAIN: -

Key Factors of Selection

1-Processing power

2-Memory

3-Connectivity

4-Development Tools and Community Support

5-Cost

4.1.1 Processing Power and How to measure the processing power needed: -

1-Tasks :-Determining the required processing power for an autonomous mobile robot involves considering various factors related to its functionality and the complexity of its tasks like:

Localization and navigation in an indoor environment such as warehousing, Obstacle avoiding

2-Data processing requirement :-

For our used sensors the needed processing power depends on the amount of data they send each second that needs to be processed

- Lidar sample rate = 5KHz (data sheet)
- Kenict sample rate = 250 Mbs: according to the manufacturer the depth camera send two photos one for the color and the other for the depth data

Color photo size eq=resolution *3(RGB)*8

Depth photo size eq = resolution *16(average pixel size)

- Encoder sample rate = 1167 bit/s and it eq=7PPr * 2 channels * 5000rpm

3- Path planning algorithm :-

.the complexity of the navigation and path planning algorithms. More complex algorithms, especially in dynamic environments, may require more processing power.

4.1.2 Memory:-

Memory is another essential factor to consider when selecting a microprocessor. You'll need enough memory to store your robot's programming code and any data collected by its sensors.

- Types of memory and how they affect on our choice.

-Random Access Memory:- is used to store data that is actively being used or processed by a computer system

-Cache memory :- Faster access to frequently used data can significantly improve processing speed. Cache memory, located on or near the CPU, stores frequently accessed data for quick retrieval.

4.1. 3 Connectivity:-

Connectivity options such as Wi-Fi, Bluetooth, or USB can be critical to our robot's functionality, depending on its intended use. We'll want to choose a microprocessor that supports the connectivity options you need for your project.

- Connection types : -

In Our application, We Will need the following ways

1. Wi-Fi for downloading our OS and ROS
2. USB for connecting our sensors

4.1. 4 Development Tools and Community Support: -

- Community support:- A wide range of users and a very active community make it the best choice for fast development for beginners
- documentation :- Very well documented from more than one trustful source
- Open source :- Very easy to find open-source tools supported by it
- Troubleshooting :- Hard to stuck in a new problem with such a familiar KIT

4.1. 5 Final comparison: -

(It is clear that Pi 2 is out of the race without even a Wi-Fi module)

- **pi4 Vs pi3**
  Raspberry Pi 4 has a bigger processing power which according to :
  - Pi 3 got Just 1GB RAM
  - bigger cache memory improves the performance of pi4

- Higher clock frequency gives faster processing of data and low memory latency
- Connectivity in Pi4 helps to improve data transportation with 2 3.0 USB

◆ **8, 4, or 2 GB RAM**:-

We chose 8 GB for faster response as our system will use about 3GB of RAM

2 for our Ubuntu OS

1 for ROS + navigation and odometry packages and any other packages
as we will see later in this chapter that small algorithm can take to 80 percent from the CPU load(Gmapping) so it's necessary to get bigger ram for better performance

4.2 Software Tools:-

Key Factors of Selection:-

- **Why ROS**
- **ROS versions**
- **ROS distributions**
- **Simulator**

4.2 .1 Why ROS:

1- Global Community:- For over 10+ years the ROS project has produced a vast ecosystem of software for robotics by nurturing a global community

2- Proven in Use :- it's inside robots that are running in production all around the world today. In the autonomous mobile robot (AMR)

3- Shorten Time to Market :- ROS provides the tools, libraries, and capabilities that you need to develop your robotics applications, allowing you to spend more time on the work that is important for your business.

4- Multi-domain :- ROS is ready for use across a wide array of robotics applications, from indoor to outdoor, home to automotive

5- 100% Open-source:- ROS is and always will be open-source, ensuring that our global community has free and unfettered access to a high-quality, best-in-class, fully featured robotics SDK

6- Multi-platform :- ROS 2 is supported and tested on Linux, Windows, and macOS, as well as various embedded platforms

.

4.2. 2 ROS Versions :-

Why ROS 2 : Let's keep in mind the marketing point of view that will be an answer for a lot of questions in your mind.

(Table 4.1) compare between two Ros versions

	ROS 1	ROS 2	Comment
Security		✔	robots might need to communicate over insecure networks whether it is via ROS topics, services, or actions. This is an enormous concern for critical commercial applications and there were lots of attempts in the community to address this concern for ROS1
Master node	✔		In ROS1, you have something called ROS Master, which provides naming and registration services for ROS nodes. ROS master acts as the mediator for establishing connections between nodes If the ROS master dies then the communication between the existing node and any other new nodes will be impossible
Lossy Networking situations		✔	ROS1 performs well in reliable networking situations, as it is built using TCP protocol which is sufficient, however in unreliable networking situations, TCP/IP struggles to deliver reliable performance due to data re-transmits. Since ROS2 uses DDS no data re-transmits are needed
Multi-platform		✔	ROS1 is only supported on Linux, however, ROS2 is supported on Linux, Windows, and macOS. It is also easier to integrate ROS2 with cloud resources such as AWS.

4.2 .3 ROS distribution: -

- What is Distribution: -

-A ROS distribution is a versioned set of ROS packages

-The purpose of the ROS distributions is to let developers work against a relatively stable codebase until they are ready to roll everything forward.

Distributions types: -

- A ROS distribution has two types LTS and non-LTS
- LTS stands for long-time support which is more stable distribution and supported for 5 years and this gives your system long lifetime

4.2 .4 Simulator: -

1. Explanation of what is and why we use simulators:

In a very simple words simulators are where we test our system in almost all its variables and study its response as in real life

2. Robotics simulators:

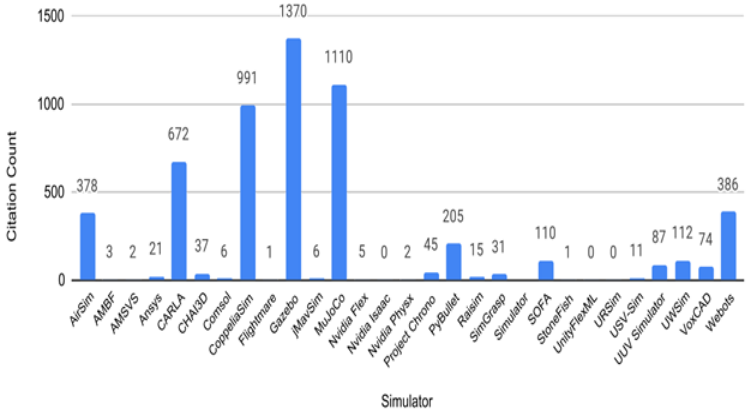
After searching between different software it has been cleared that they all provide almost the same features in a different way of implementation so we will choose the most used one for the best source of learning and here we talk about Gazebo

3. IEEE Access v9: -

Citation count from 2016 to 2020 for reviewed simulators. Citations were gathered from Google Scholar using either one or more of a simulators' research paper, reference manual or other citation type and then filtered for robotics keywords.

4. Benefits :-

- No cost needed just Knowledge
- Safe time and effort
- Very close to real system conditions



(Figure 4.1 Citation count for reviewed simulators)

Feature Comparison Between Popular Robotics Simulators Used for Mobile Ground Robotics

Simulator	GPS	Tracks	Wheels	Legs	Mecanum / Omni Wheels	Heightmap Import	OpenDrive / OpenStreetMap	Pathplanning	ROS Support	RGBD	LiDAR	Realistic Rendering
Gazebo	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
CoppeliaSim	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
Raisim	✗	✗	✓	✓	✗	✓	✗	✗	✗	✓	✓	✓, Unity
Webots	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
PyBullet	✗	✗	✓	✓	✗	✓	✗	✓	✗	✓	✓	✗
CARLA	✓	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓, Unreal
Project Chrono	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓, POV-Ray

(Figure 4.2 Comparison Between Popular Robotics Simulators)

4.3 System Flow chart and Algorithm :-

4.3 .1 Implementation of the Algorithm :-

- programs for the ROS platform were installed, and the package in charge of the operation of position sensors and visual sensors executes
- The package that captured the information obtained by the camera and converted it into a scan-type message to be used by the SLAM package was then executed
- Determined the mode mapping or Navigation
- In mapping the Mapping Package uses data from the visual sensor (camera & LiDAR) and the data from position sensors to generate the map and then save it in the database
- In the navigation mode the user sets the target location and the navigation package uses the sensor data to create path

4.3 .2 Pseudocode: -

The code begins by powering on a system and capturing sensor readings through its nodes. It then starts executing a SLAM (Simultaneous Localization and Mapping) algorithm package, initializing odometry data for position tracking. If the environment is detected as new, it clears the existing database, sets an initial point, and runs a mapping algorithm using the captured sensor data and odometry readings to generate a map. This map is then saved to the database, and a message is sent to the master system indicating that the map has been generated. If the environment is not new, the system reads the database to set a target position and executes a navigation stack. It uses the odometry data for position tracking and the sensor data to avoid obstacles, finally sending a message to the master system confirming it has reached the target position.

```
power On

data=sensors reading captured through its nodes
SLAM algorithm package start excuting
ododata =Odometry reading
locating mode detecation
if environment is_new
    (
    Clear data base
    set initial point
    excute Maping algorithm
      using data,ododata
    generate map
    save map in data base
send message to master (Map generated)
    )
else
    (
    Read database
    set target position
    excute navigation stack

    use Ododata For position tracking
    use data for avoid obestecal )
send message to master (in position)
```

(Figure 4.3 Pseudocode)

4.3 .3 Flowchart: -

breakdown of the flowchart:

1. Start
    - The process begins.
2. Sensors start
    - The system checks if the sensors have started.
    - If the sensors have not started, it loops back to wait until they do.
3. Execute sensors node
    - Once the sensors have started, the sensor node is executed.
4. Execute SLAM algorithm
    - The SLAM algorithm is executed to start the localization and mapping process.
5. Locating mode?
    - The system checks if it is in locating mode.
    - If Yes, proceed to activate incremental memory.
    - If No, proceed to deactivate incremental memory.

If in locating mode (Yes):

6. Active incremental memory
    - Incremental memory is activated to support real-time updates in mapping.
7. Clear database
    - The database is cleared to prepare for new data.
8. Obtain initial position
    - The initial position is obtained for starting the mapping process.
9. Execute mapping algorithm
    - The mapping algorithm is executed to create a map of the environment.
10. Save generated map
    - The generated map is saved.
11. End
    - The process ends.

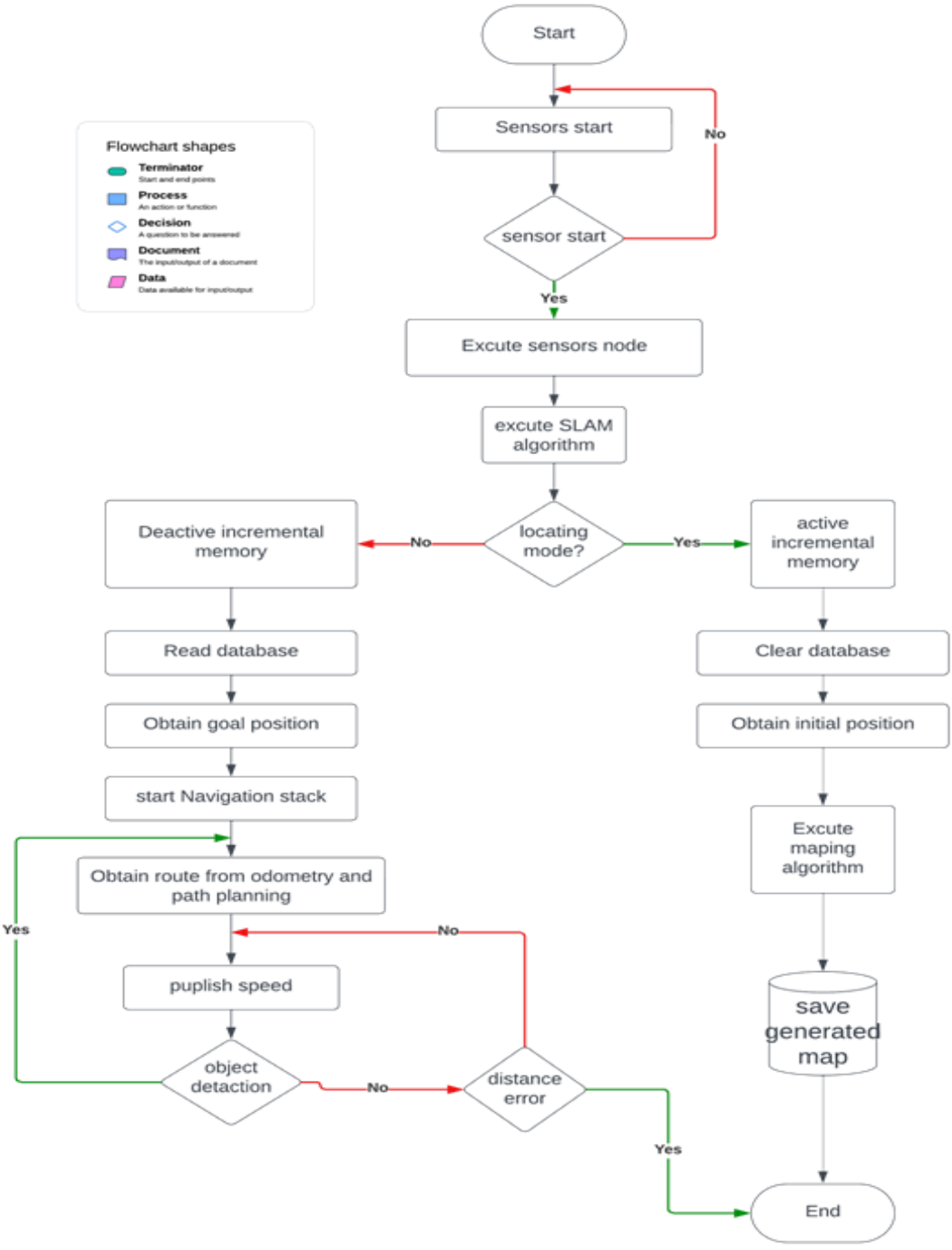
If not in locating mode (No):

6. Deactivate incremental memory
    - Incremental memory is deactivated to use existing data.
7. Read database
    - The system reads from the database to use previously stored data.
8. Obtain goal position
    - The goal position is obtained from the database.
9. Start Navigation stack
    - The navigation stack is started to begin the path planning and movement process.
10. Obtain route from odometry and path planning
    - The route is obtained using odometry and path planning algorithms.
11. Distance error?
    - The system checks for any distance error.
    - If Yes, the process ends.
    - If No, proceed to object detection.
12. Object detection
    - The system checks for object detection.
    - If an object is detected, it loops back to adjust the route.
13. Publish speed
    - The system publishes the speed for the robot to move.
14. End
    - The process ends.

Legend

- Terminator (Oval): Indicates the start and end points of the process.
- Process (Rectangle): Represents actions or steps taken in the process.
- Decision (Diamond): Represents decision points that branch the flow based on Yes/No answers.
- Document (Parallelogram): Represents documents or data that are referenced or produced.

- Data (Cylinder): Represents data storage or databases involved in the process.



(Figure 4.4 Flowchart)

Sensors section:

The sensors in our Autonomous Mobile Robot (AMR) form its perceptive eyes and ears,

enabling it to interact intelligently with its surroundings. From cameras capturing visual data to lidar sensing the environment in 360 degrees, our carefully selected sensors empower the AMR with the ability to navigate, avoid obstacles, and make informed decisions. This section dives into the technical details of each sensor, shedding light on their roles and contributions to the AMR's autonomy. Together, these sensors create a robust perception system, elevating our robot's capability to operate seamlessly and adapt to diverse environments.

Here they are our sensors and the use of each of them and its theory of work

3.4 Rp Lidar A1(slamtec)

A LiDAR is an electronic component and is part of the sensor family. More precisely, it is part of the Time Of Flight (ToF) sensor category. A sensor collects data about physical parameters such as temperature, humidity, light, weight, distance, etc. A LiDAR is measuring the distance to the nearest obstacle through a communication protocol.

- **Physics – How does Li-DAR work?** Li-DAR technology is a remote sensing technology that measures the distance between itself and a target. Light is emitted from the Li-DAR and travels to a target. It will reflect off of its surface and come back to its source. As the speed of light is a constant value, the Li-DAR is able to calculate the distance to the target, Distance = (Speed of Light x Time of Flight) / 2
*Knowing the position and orientation of the sensor***, the XYZ coordinate** *of the reflective surface can* be calculated**,** represented *by a* point

-The LiDAR vision system: There are 3 categories of LiDAR vision systems: **1D, 2D** or **3D**. They work in the same way, the difference is about using a point and shoot or a scanning mode system, and also the quantity of laser beams used.

**For a 2D Li-DAR, only one laser beam is necessary. Indeed, it will pulse based on a spin movement and collect horizontal distance to the targets to get data on X and Y axes.*

-Wavelength: The wavelength of the laser is an important parameter of the LiDAR. Indeed, the sunlight received on Earth's surface is spread out over a wide wavelength spectrum:

- 750 nm [selected]
- 940 nm
- 1125 nm
- 1400 nm
  Laser beams more powerful than the level 1 can be harmful to the human eye and damage the retina.

- LiDAR use the following wavelength:

    - Infrared (1500 –2000 nm) for meteorology/Doppler LiDAR – Scientific uses
    - Near-infrared (850 -940 nm) for terrestrial mapping
    - Blue-red (500 –750 nm) for bathymetry

- Ultraviolet (250 nm) for meteorology

- **Indoor/Outdoor**

All the LiDAR that comply to this technology standards can be used indoor. Only a few of them can be used outdoor. The following factors need to be taken into account:

- The wavelength: at 500 nm, the sunlight produces the highest level of disturbance
- The ambient light resistance (in Lux) : parameter which indicates how much light it can accept to work properly
- The surface type : transparent surface, smoke, fog, etc
- The environmental noise resistance ability: rain, snow, relief, etc
- The temperature range: operating temperature of the LiDAR
- Electromagnetic considerations: physics disturbance which can alter the sensor behaviour

Outdoor LiDAR are more expensive due to their higher performances.

Distance: LiDAR distance range go from 0.01m to 200m. Depending of the environment the LiDAR will be exposed to indoor, outdoor, rugged landscape, etc. A LiDAR with a suitable distance range needs to be picked.

Error: All LiDAR encounter two types of errors:

Systematic error : this type of error shifts all measurements in a systematic and predictable way. Systematic errors can’t be eliminated, but their influence can be minimized.

Statistical error: additional errors, due to the environment and physics parameters (refractions, diffraction, etc), can also occur. A statistical error happens when the exact same measurements done by the LiDAR will display different values. LIDAR range distance error change between ±10mm to ±200mm.

Power supply: All LiDAR require a power supply of some sort. Depending of the voltage and the current consumption, power consumption can be calculating. Using a battery, this parameter has a real importance, indeed, a LiDAR which is consuming a lot of power will shorten the battery cycle.

Interface

The **interface, controller and communication protocol** that will be used with the LiDAR must be able to keep up with the data rate measurement (I2C, PWM, SPI, serial, etc), in order not to lose any information.

You need to be sure to have the same communication speed between your LiDAR and your board to process data accurately to get the expected behavior for your robot.

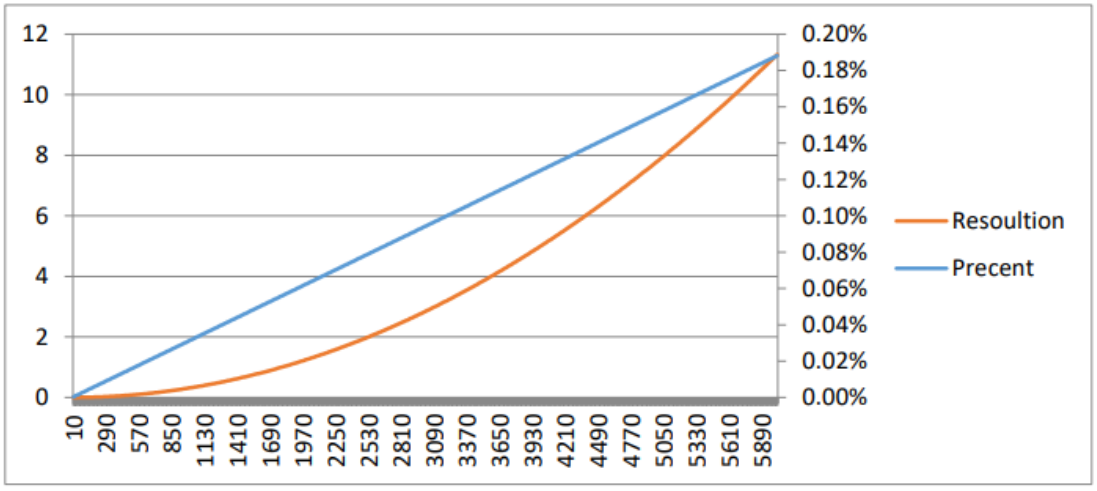
Conclusion – LiDAR pros and cons

- LiDAR pros
    - Data can be collected quickly and with high accuracy
    - LiDAR can be integrated with other sensors : sonar, camera, IMU, GPS, ToF sensors
    - The LiDAR technology can be used in daylight or in the dark, thanks to an active illumination sensor
    - Can be used to collect data about unreachable places
    - LiDAR are fast and highly accurate. It is a great tool to collect data about vast areas of land
    - Once properly set up, a LiDAR is an autonomous piece of technology and can pretty much run by itself
- LiDAR cons
    - LiDAR can be expensive depending on the specifications required by the project
    - LiDAR is ineffective in heavy rain, low-hanging clouds, if your environment has a fog or smoke atmosphere, or using transparent obstacles
    - Analyzing the massive quantity of data collected can consume time and resources
    - Powerful laser beams used in some LiDAR can damage the human eye
    - Difficulties to penetrate highly dense matter

Measurement Performance

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter(m)	TBD	0.15 - 6	TBD	White objects
Angular Range	Degree	n/a	0-360	n/a	
Distance Resolution	mm	n/a	<0.5 <1% of the distance	n/a	<1.5 meters All distance range*
Angular Resolution	Degree	n/a	≤1	n/a	5.5Hz scan rate
Sample Duration	Millisecond(ms)	n/a	0.5	n/a	
Sample Frequency	Hz	n/a	≥2000	2010	
Scan Rate	Hz	1	5.5	10	Typical value is measured when RPLIDAR A1 takes 360 samples per scan

(Figure 4.5 Lidar Measurement Performance)



(Figure 4.6 Lidar resolution and precent)

Laser Power Specification

Item	Unit	Min	Typical	Max	Comments
Laser wavelength	Nanometer(nm)	775	785	795	Infrared Light Band
Laser power	Milliwatt (mW)	TBD	3	5	Peak power
Pulse length	Microsecond(us)	TBD	110	300	

(Figure 4.7 Lidar Laser Power Specification)

Communication interface

Item	Unit	Min	Typical	Max	Comments
Band rate	bps	-	115200	-	
Working mode	-	-	8N1	-	8n1
Output high voltage	Volt (V)	2.9	-	3.5	Logic High
Output low voltage	Volt (V)	-	-	0.4	Logic Low
Input high voltage	Volt (V)	1.6*	-	3.5	Logic High
Input low voltage	Volt (V)	-0.3	-	0.4	Logic Low

(Figure 4.8 Lidar Communication interface)

MISC

Item	Unit	Min	Typical	Max	Comments
Weight	Gram (g)	TBD	190	TBD	
Temperature range	Degree Celsius (°C)	0	TBD	45	

(Figure 4.9 Lidar MISC)

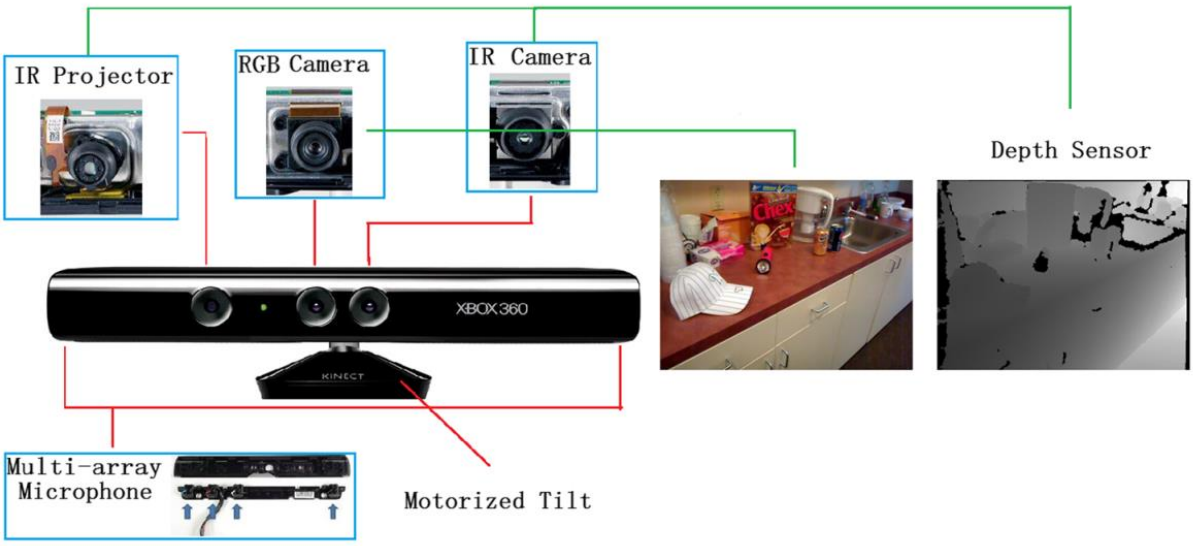
3.5 Kinect RGB_D camera

A brief review of Kinect

In the past years, as a new type of scene representation, RGB-D data acquired by the consumer-level Kinect sensor has shown the potential to solve challenging problems for computer vision. The hardware sensor as well as the software package are released by Microsoft in November 2010 and have a vast of sales until now. At the beginning, Kinect acts as an Xbox accessory, enabling players to interact with the Xbox 360 through body language or voice instead of the usage of an intermediary device, such as a controller. Later on, due to its capability of providing accurate depth information with relatively low cost, the usage of Kinect goes beyond gaming, and is extended to the computer vision field. This device equipped with intelligent algorithms is contributing to various applications, such as 3D-simultaneous localization and mapping (SLAM) [39, 54], people tracking [69], object recognition [11] and human activity analysis [13, 57], etc. In this section, we introduce Kinect from two perspectives: hardware configuration and software tools.

Kinect hardware configuration

Generally, the basic version of Microsoft Kinect consists of a RGB camera, an infrared camera, an IR projector, a multi-array microphone [49] and a motorized tilt. Figure 1 shows the components of Kinect and two example images captured by RGB and depth sensors, respectively. The distance between objects and the camera is ranging from 1.2 meters to 3.5 meters. Here, RGB camera is able to provide the image with the resolution of 640 × 480 pixels at 30 Hz. This RGB camera also has option to produce higher resolution images (1280×1024 pixels), running at 10 Hz . The angular field of view is 62 degrees horizontally and 48.6◦ vertically. Kinect's 3D depth sensor (infrared camera and IR projector) can provide depth images with the resolution of 640 × 480 pixels at 30 Hz. The angular field of this sensor is slightly different with that of the RGB camera, which is 58.5 degrees horizontally and 46.6 degrees vertically. In the application such as NUI (Natural User Interface), the multi-array microphone can be available for a live communication through acoustic source localization of Xbox 360. This microphone array actually consists of four microphones, and the channels of which can process up to 16-bit audio signals at a sample rate of 16 kHz. Following



(Figure 4.10 Kinect RGB_D camera)

(Fig.4. 10) Illustration of the structure and internal components of the Kinect sensor. Two example images from RGB and depth sensors are also displayed to show their differences

In general, the technology used for generating the depth map is based on analyzing the speckle patterns of infrared laser light. The method is patented by PrimeSense.

1 Comparison between Kinect v1 and Kinect v2

		Kinect for windows v1	Kinect for windows v2
Color	Resolution	640×480	1920×1080
	fps	30fps	30fps
Depth	Resolution	640×480	512×424
	fps	30fps	30fps
Sensor		Structured light	Time of flight
Range		1.2 ~ 3.5*m*	0.5 ~ 4.5*m*
Joint		20 joint / people	25 joint / people
Hand state		Open / closed	Open / closed / Lasso
Number of Apps		Single	Multiple
Body Tracking		2 people	6 people
Body Index		6 people	6 people
Angle of View	Horizontal	62 degree	70 degree
	Vertical	48.6 degree	60 degree
Tilt Motor		Yes	No
Aspect Ratio		4:3	6:5
Supported OS		Win 7, Win 8	Win 8
USB Standard		2.0	3.0

(Figure 4.11 Comparison between Kinect v1 and v2)

applications. It is not specifically built for Kinect, and it can support multiple PrimeSense 3D sensors. Normally, users need to install SensorKinect, NITE, and OpenNI to control the Kinect sensor, where SensorKinect is the driver of Kinect and NITE is the middleware provided by PrimeSense . The latest version of OpenNI is the version 2.2.0.33 until June 2015. The Point Cloud Library (PCL) is a standalone open source library which provides SLAM-related tools such as surface reconstruction, sample consensus, feature extraction, and visualization for RGB-D SLAM. It is licensed by Berkeley Software Distribution (BSD). More details and publications about PCL can be found in . The official version of Kinect for Windows SDK8 was released in July 2011, which provides a straightforward access to Kinect data: depth, color and disparity. The newest version is the SDK 2.0. It can be applied for Windows 7, Windows 8, Windows 8.1 and Windows Embedded 8 with C++, C# or VB.NET. The development environment uses Visual Studio 2010 or higher versions. Regarding the software tool, it mainly contains skeletal tracking, higher depth fidelity, audio processing and so on. The comparison of Kinect Windows SDK and unofficial SDK, e.g., OpenNI, can be summarized below. The detailed same and difference between the Kinect Windows SDK and unofficial SDK can be seen in Table 2. Kinect Windows SDK:

1) It supports audio signal processing and allows to adjust the motor angle.

2) It provides a full-body tracker including head, feet, hands and clavicles. Meanwhile, some details such as occluded joints are processed meticulously.

3) Multiple Kinect sensors can be supported.

OpenNI/NITE library:

1) Commercial use of OpenNI is allowed.

2) Frameworks for hand tracking and hand-gesture recognition are included in OpenNI. Moreover, it automatically aligns the depth image and the color image.

3) It consumes less CPU power than that of Kinect Windows SDK.

4) It supports Windows, Linux and Mac OSX. In addition, streaming the raw Infrared video data becomes possible. In conclusion, the most attractive advantage of OpenNI is the feasibility for multiple operational platforms. Besides it, using OpenNI is more convenient and can obtain better results for the research of colored point clouds. However, in terms of collection quality of the original image and the technology for pre-processing, Kinect for Windows SDK seems to be more stable. Moreover, Kinect for Windows SDK is more advantageous when requiring skeletal tracking and audio processing.

2 Comparison between the Kinect Windows SDK and unofficial SDK

	Kinect windows SDK	Unofficial SDK
Supported OS	Windows 7×86/×64 Windows 8, Windows 8.1 and Windows Embedded 8	Windows XP/Vista/7×86/×64 Windows 8, Windows 8.1 and Windows Embedded 8 LinuxUbuntu×86/×64 Mac OS Android
Development language	C++, C#	C, C++, C#, Java
Commercial use	No	Yes
Supports for audio and motor/tilt	Yes	No
Supports multiple sensors	Yes	No
Consumption of CPU power	More	Less
Full body tracking	Includes head, hands, feet, clavicles Calculates positions for the joints, but not rotations Only tracks the full body, no hands only mode	No head, hands, feet, clavicles Calculates both positions and rotations for the joints Supports for hands only mode
Supports for Unity3D game engine	No	Yes
Supports for record/ playback to disk	No	Yes
Supports to stream the raw InfraRed video data	No	Yes

(Figure 4.12 Comparison between Kinect windows SDK and unofficial SDK)

3.6 Encoders

In the case of the drive motors, we want motors with integrated magnetic encoders. A Hall sensor is used to generate pulses every time 1 or more magnets on the motor shaft rotates past the sensor. You send those pulses to a micro-controller, which counts them and computes the actual wheel speed. (To do this, it needs to know the gear ratio and some other specifics — more on that later)

Our encoder is MY-775 in its specifications

So to compute the motor speed using our micro-controller

We use timer peripheral to set an interrupt each one second and use counter to count the pulses through this second and through the next Eq we obtain our angular velocity ω=(Number of Pulses/Counts per Revolution)$\times 2\pi$ = rps

So now the micro-controller can both control the speed of the motor, by varying the PWM pulse width, and read the actual speed of the shaft.

3.7 Software algorithms

as represented in our flowchart the second step after powering our system is to start the SLAM algorithm in the few following lines we will compare between slam algorithms and explain on which base we categorize them

Simultaneous Localization and Mapping (SLAM) refers to creating a map of an unknown environment while simultaneously determining location. SLAM is commonly associated with developing autonomous vehicles and robotics. Other application areas of SLAM

SLAM can be categories based on **the application environment into**:

SLAM Feature-Based:

In feature-based SLAM (Simultaneous Localization and Mapping), the focus is on extracting distinctive features from the environment, such as corners or key points, to track the robot's movement and create a map. These features serve as landmarks for localization. Algorithms like the Extended Kalman Filter (EKF) or FastSLAM are commonly used in feature-based SLAM. This approach is efficient in handling complex environments and can work well with various sensor modalities, including cameras and lidar.

SLAM Mark-Based (Landmark-Based):

SLAM mark-based, or landmark-based SLAM, is a variant of SLAM where the system explicitly represents and tracks individual landmarks in the environment. Landmarks are specific points or features that are detected and tracked to create a map and estimate the robot's pose. This approach is often associated with grid-based mapping techniques, where the environment is discretized into a grid, and each cell may represent a landmark or obstacle. Particle filters, such as those used in the FastSLAM algorithm, are commonly applied in landmark-based SLAM.

Grid-Based SLAM:

Description: Grid-based SLAM represents the environment as a grid map, where each cell in the grid contains information about the occupancy or features of that part of the space. This approach discretizes the environment into a grid.

- The sensors type

Visual SLAM (Simultaneous Localization and Mapping):

Visual SLAM refers to a class of SLAM methods that primarily use visual information from cameras to navigate an environment, simultaneously creating a map of that environment. Visual SLAM systems typically extract features from camera images, track these features over time, and use the information to estimate the robot's position and build a map. Visual SLAM can be effective in environments with sufficient visual features, making it suitable for applications such as robotics, augmented reality, and autonomous vehicles.
(ORB-SLAM, monocular DPPTAM, stereo ZedFu and RTAB-Map)

Lidar SLAM:

Lidar SLAM involves the use of Lidar (Light Detection and Ranging) sensors to perform simultaneous localization and mapping. Lidar sensors emit laser beams and measure the time it takes for the laser beams to return after reflecting off objects in the environment. This information is used to create a detailed 3D map of the surroundings. Lidar SLAM is known for its accuracy in mapping and is often used in robotics, autonomous vehicles, and other applications where precise 3D mapping is crucial.
(HectorSLAM, GMapping, cartographer, SLAM_toolbox)

For Now we still developing our 2D Mapping algorithm then we will upgrade it to a suitable 3D Mapping

1: Popular ROS-compatible lidar and visual SLAM approaches with their supported inputs and online outputs.

	Inputs							Online Outputs			
	Camera				Lidar		Odom	Pose	Occupancy		Point
	Stereo	RGB-D	Multi	IMU	2D	3D			2D	3D	Cloud
GMapping					✓		✓	✓	✓		
TinySLAM					✓		✓	✓	✓		
Hector SLAM					✓			✓	✓		
ETHZASL-ICP					✓	✓	✓	✓	✓		Dense
Karto SLAM					✓		✓	✓	✓		
Lago SLAM					✓		✓	✓	✓		
Cartographer					✓	✓	✓	✓	✓		Dense
BLAM						✓		✓			Dense
SegMatch						✓					Dense
VINS-Mono				✓				✓			
ORB-SLAM2	✓	✓									
S-PTAM	✓							✓			Sparse
DVO-SLAM		✓						✓			
RGBiD-SLAM		✓									
MCPTAM	✓		✓					✓			Sparse
RGBDSLAMv2		✓					✓	✓		✓	Dense
RTAB-Map	✓	✓	✓		✓	✓	✓	✓	✓	✓	Dense

(Figure 4.13 ROS compatible lidar and visual SLAM)

Next we will show few differences between group of the most commonly used 2D Slamming

HECTOR_SLAM:decent option if you have a lidar-only system (and low vibrations), definitely much better maps when reliable odometry is also used.

This is an interesting SLAM package because it works both with and without odometry info. It also has a neat hector_trajectory_server node that makes the trajectory data available via a topic which can then be used to visualize the robot's path using Rviz or Foxglove. hector_slam using only the lidar data was not quite a success. Due to the robot's fast movements and jerky motion at times, you could only get good maps when you drove really slow and did not make any rotations. However, with the odometry from any visual sensor, things got much better you could now drive faster and make sharp turns. But now, the SLAM algorithm seemed to be heavily dependent on the odometry, and the visual sensor is not always reliable. Now, you will be able to get really good maps, odometry did not fail (which it usually does when I either block vision or accelerate quickly.

GMAPPING:Great maps for small spaces, perfect starter SLAM package

Its the standard SLAM package in ROS, it provides really good maps, certainly much better than hector_slam. Unfortunately, it needs an odometry source to work well, so this cannot be used in a lidar-only system. Luckily, GMapping doesn't seem to work well in really large spaces (like warehouses), so while its really good for a home/studio environment, there is room for improvement when in bigger spaces (using slam_toolbox is an alternative)

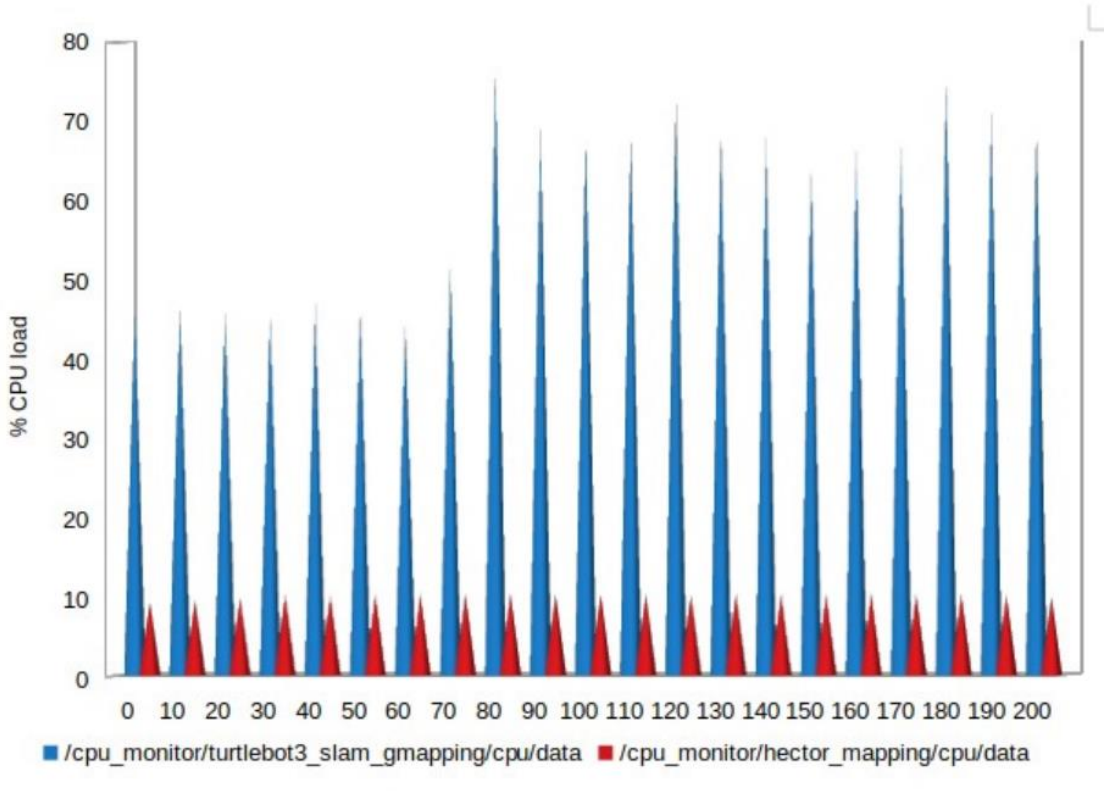
KARTO:Final conclusion: Another great starter SLAM package for ROS learners, I haven't personally tested this in larger spaces, but provides Gmapping-like results in small spaces. This was not part of my plans but from everything I've read, slam_karto is another great SLAM package to start off with. Just like GMapping, it provides really good maps, but uses odometry and unlike hector_slam, does not fail when odometry fails from time to time (the odometry eventually recovers, but while its in the failure state, the SLAM pose does not drift). slam_karto also provides a visual feedback of the traversed path of the robot, which can be visualized in Rviz or Foxglove.

SLAM_TOOLBOX: This package has the most options compared to the other methods - online/offline configurations, lifelong mapping and localization modes. In small spaces, the generated maps are just as good as the gmapping maps but slam_toolbox is more reliable.

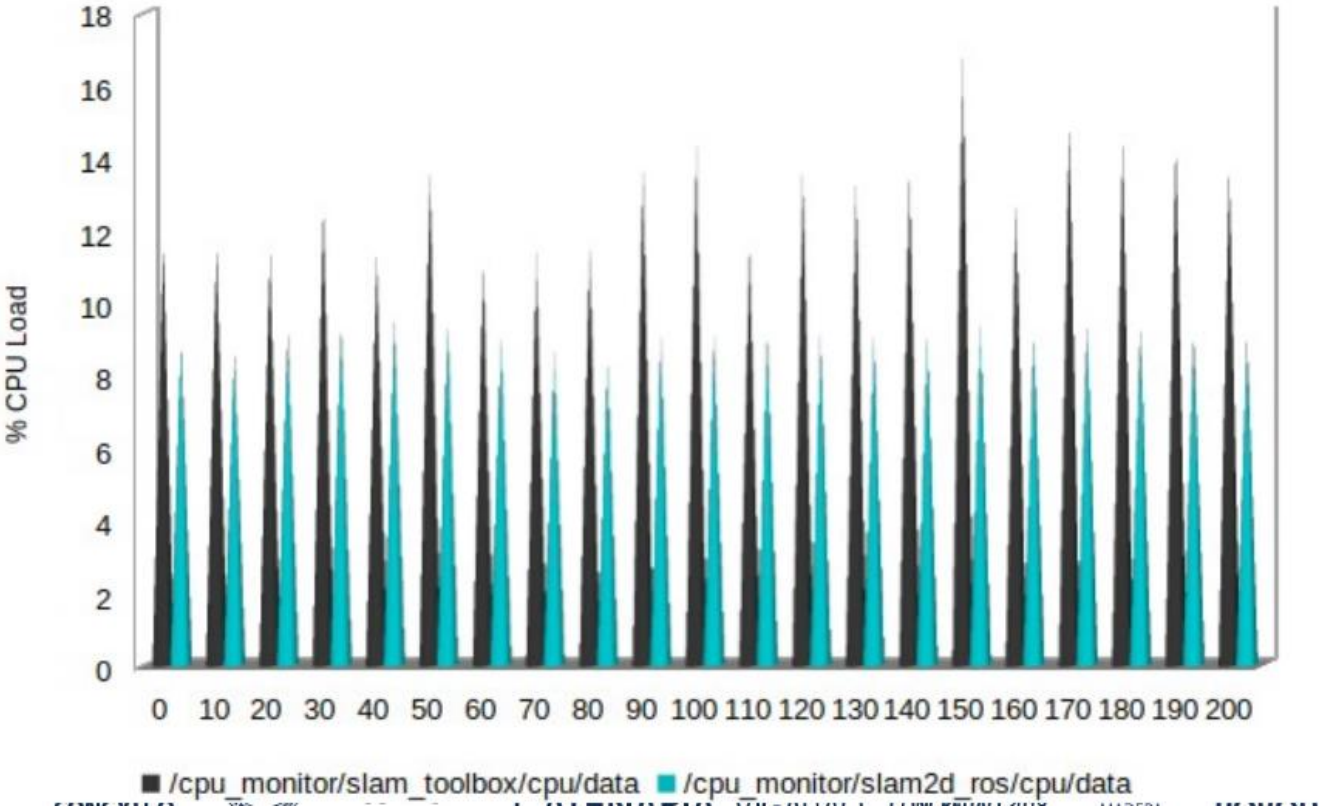
This package provides a lot more options compared to the other methods - synchronous/asynchronomous mapping, lifelong mapping, offline mapping, map-merging tools, an interactive mode. A lot more information can be seen on their github repo. For now, the generated maps are comparable to the maps created with gmapping, but slam_toolbox is in general quite reliable. so we plan on continuing to use the slam_toolbox package.

For us, we found slam_toolbox to be the most reliable out of the four methods we tested. It also implemented a lot of different use-cases, and provided tools, all optimized for large scale mapping. Also looking at anecdotal experience documented online, we definitely think slam_toolbox is the right choice for 2d SLAM. However, for small places, gmapping and slam_karto provide similar results, sometimes gmapping is much better as well.

Another vital reason for deferring an algorithm to another is CPU loads the next graphs show the representation of CPU load in three cases of the previous methods which are the most commonly used this data is taken from AMD Radeon R4 8GB RAM GPU Clock800 MHz



(Figure 4.14 CPU loads graph)



(Figure 4.15 CPU loads graph)

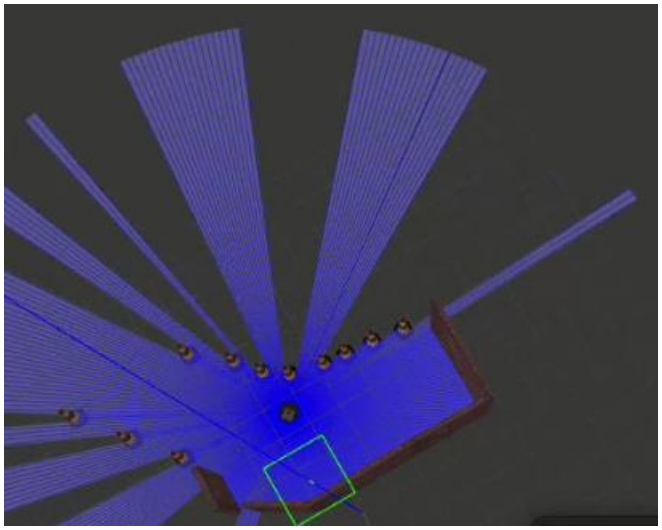
(Table 4.2 Mapping Algorithms)

Algorithms	odometry	Big maps	CPU laod
Slam_toolbox	A need	Good	Fairly low
Gmapping	A need	Bad	Very high
Hector_slam	Recommended	Bad	low

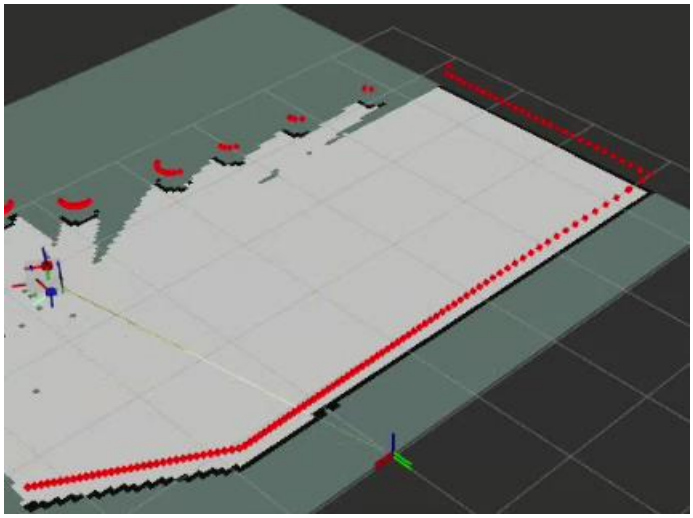
Although Slam_toolbox is not the lowest recorded data it seems to be a good deal with what it provides like reliability and features and good accuracy to

so Now let’s show our progress on such a system
our tools as explained previously ROS2, Gazebo, Rviz,
The Packages used : slam_toolbox | rplidar_ros2
these packages provide a punch of functions like online, lifelong , and offline mapping
Let’s explain through the following graphs how it works
Step 1: After creating URDF which represents the robot hardware to the simulator separated into parts we use a tool called Xacro to build and process these files together into one processed file This is passed to robot_state_publisher which makes the data available on the /robot_description topic, and also broadcasts the appropriate transforms.

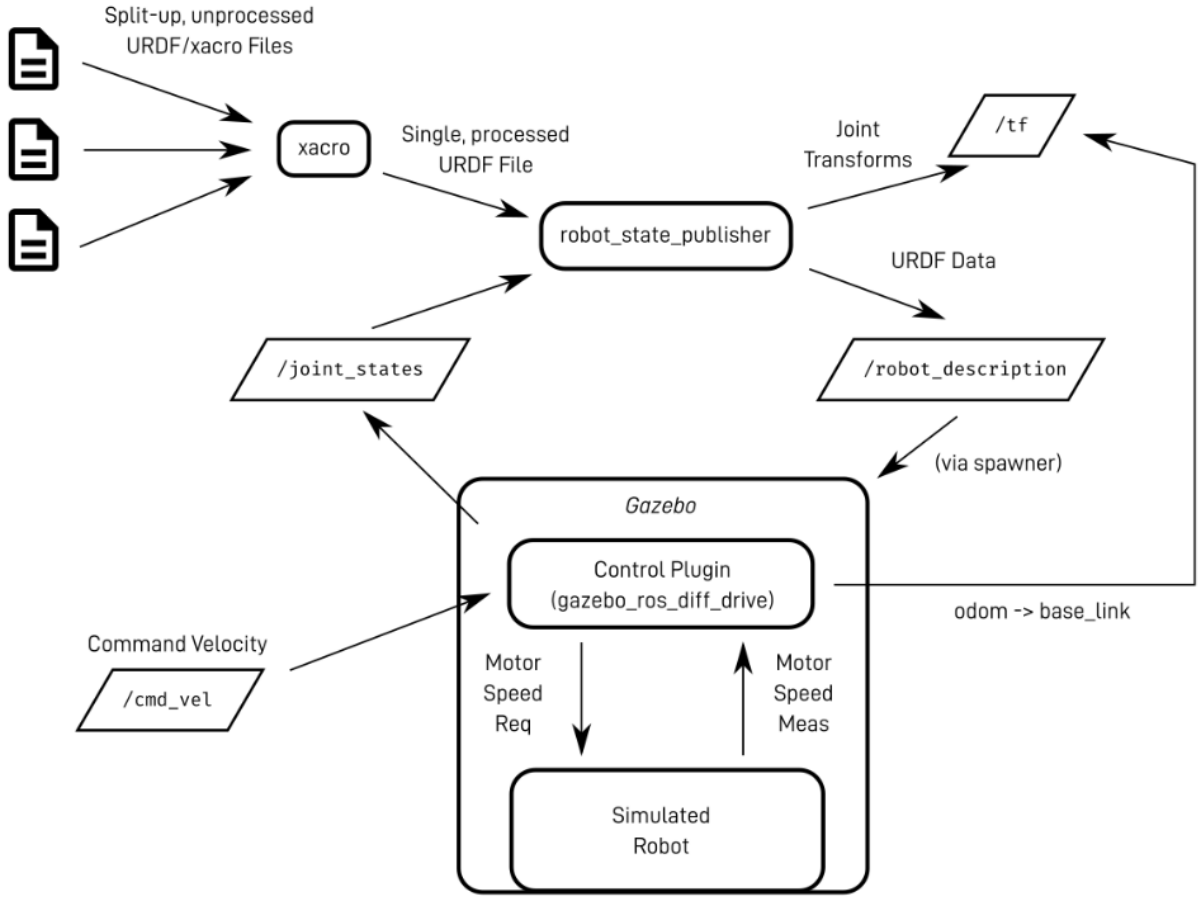
Step 2: the Gazebo robot is spawned from /robot_description, and the joint_states are published by the control plugin. The plugin also broadcasts a transform from a new frame called Odom (which is like the world origin, the robot's start position), to base_link, which lets any other code know the current position estimate for our robot.
Step3: provide a controller to control the robot's motion so that we can navigate through our environment to create our map
Step 4: provide lidar urdf which publishing data to be used in the mapping phase on /scan topic in laser_frame
Step5: launch our lidar node passing the ID frame of our urdf lidar to it as a parameter so it can publish its data to our virtual lidar and then publish it again to /scan
Step6:launch our wanted mode file from the slam toolbox
Step7:launch Rviz to visualize our mab and save it after finishing
here is our generated map and our environment now



(Figure 4.16 generated map)



(Figure 4.17 generated map)



(Figure 4.18 Robot state updates)

Tips:

online_sync:Description: This launch file is likely designed for online synchronous SLAM, where sensor data is processed in real-time, and the SLAM algorithm produces estimates of the robot's pose and the map synchronously with the sensor data.

online_async:Description: Similar to online_sync, but in this case, the SLAM algorithm operates asynchronously. This means that sensor data may be processed independently of real-time constraints, which can be beneficial in scenarios where occasional delays in processing are acceptable.

offline:Description: This launch file might be intended for offline SLAM processing, where sensor data collected during a robot's mission is processed after the mission is complete. This can be useful for generating maps or analyzing data post-mission.

lifelong:Description: This launch file may be designed for lifelong SLAM scenarios, where a robot continuously operates over an extended period, incrementally updating its map and localization estimates over time.

merge_map_kinematics:Description: This launch file might be intended for scenarios where maps from different sources or sensors need to be merged. It may also include kinematic information to enhance map alignmen

ROS Visualization

Robot Operating System (ROS) Visualization plays a pivotal role in enhancing the capabilities of robots by providing intuitive ways to perceive, analyze, and interact with their surroundings. As robots become more sophisticated, the need for effective visualization tools becomes increasingly crucial for researchers, developers, and operators.

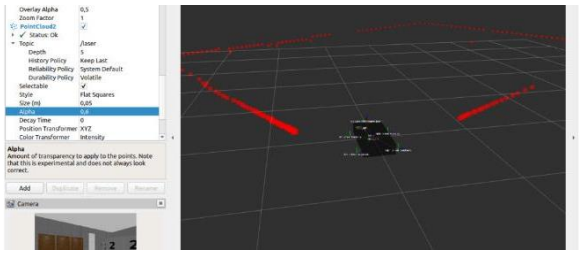
1. RViz:

3D Robot Visualizer: RViz is the go-to tool for 3D visualization in ROS. It allows you to:

Display your robot's model: Load CAD models of your robot and its sensors to see their positions and movements in real-time.

Visualize sensor data: Overlay sensor data like LiDAR points, camera images, and IMU readings onto the robot model or in separate displays.

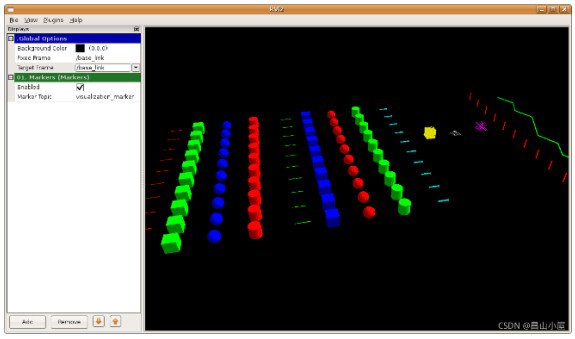
Debug and monitor: Track robot state, joint angles, and other variables to identify issues and optimize performance.



(Figure 4.19 robot's model in RViz)

2. Markers:

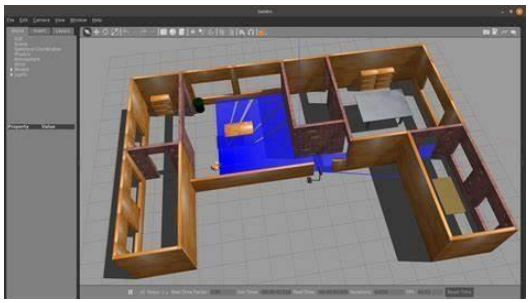
Customizable 3D Objects: Markers are lightweight objects you can publish to RViz from your ROS nodes. They come in various shapes (spheres, cubes, arrows, etc.) and can be colored and sized to represent different data points or robot states.



(Figure 4.20 Example for Markers)

3. Gazebo Simulation Environment:

Gazebo is a popular simulation environment in ROS that provides realistic 3D simulation for robots. It allows users to visualize and test robot behaviors in a simulated world before deploying them in the real environment, aiding in development and debugging.



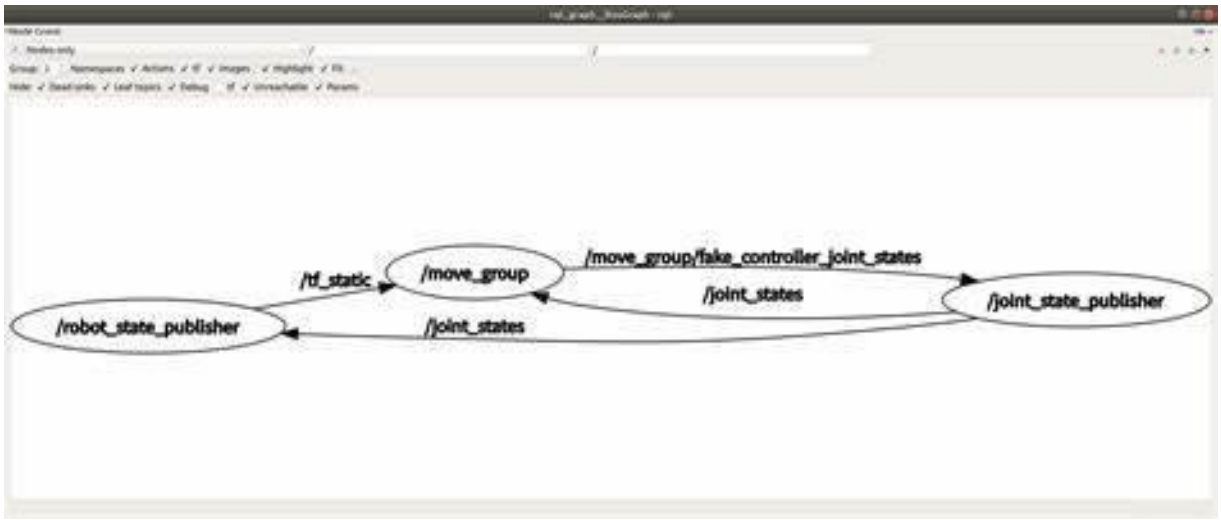
(Figure 4.21 Simulation Environment)

Source: Gazebo

4. RQT (ROS Qt-based GUI Framework):

RQT is a framework that allows users to create custom graphical user interfaces (GUIs) to visualize and interact with ROS data. It provides a flexible way to design custom tools and dashboards tailored to specific robotic applications.

Source: ROS Wiki – RQT



(Figure 4.22 RQT Framework)

5. ROS Web Visualization with Webviz:

Webviz is a web-based visualization tool for ROS, enabling users to visualize and interact with ROS data through a web browser. This facilitates remote monitoring and control of robotic systems, providing accessibility from anywhere with an internet connection.

Source: Webviz

(Figure 4.23 Visualization with Webviz)

In summary, ROS Visualization tools like RViz, Interactive Markers, Gazebo, RQT, and Webviz play a crucial role in enhancing the development, testing, and operation of robotic systems. These tools provide intuitive and dynamic interfaces for users to gain insights into the robot's perception, debug algorithms, and optimize performance, ultimately contributing to the advancement of robotics research and applications.

What we have done.

we created ros pkg for visualization, and it launch our rviz simulation and gazebo simulation, and we would implement it to apply the function of our robot to be achieved on the simulators as same as real life. To achieve that we had to do three elements the robot model, launch files and the world.

1. **URDF:**

URDF defines a robot's structure by specifying its links, joints, sensors, and visual/physical properties. Links represent individual parts of the robot, and joints define how these links are connected. Visual andcollision properties describe the robot's appearance and interaction with the environment.

Links:

```
       <link
name="base">
  <inertial>
<mass value="8.34" />
<origin xyz="0 0 0.1" rpy="0 0 0" />
<inertia ixx="0.002386469" iyy="0.004827723" izz="0.003252219" ixy="0.0"
ixz="0.0" iyz="0.0" />
</inertial>
<visual>
<geometry>
<mesh filename="package://my_amr/meshes/Base_final.stl"/>
</geometry>
<material name="gray">
<color rgba=".7 .7 .7 1" />
</material>
</visual>
<collision>
<geometry>
<mesh filename="package://my_amr/meshes/Base_final.stl"/>
</geometry>
<contact_coefficients mu="1" mu2="1" kp="1e+13.0" kd="1.0"/>
```

```
</collision>
</link>
```

- `<link name="base">`: This defines a link in the robot model with the name "base." In URDF, a linkrepresents a physical part of the robot.
- `<inertial>`: This block provides information about the inertial properties of the link, such as massand inertia. In this case:
  - `mass value="8.34"`: Specifies the mass of the link.

- `origin xyz="0 0 0.1" rpy="0 0 0"`: Defines the origin of the inertial frame, specifying theposition (xyz) and orientation (rpy) relative to the link.
- `inertia ixx="0.002386469" iyy="0.004827723" izz="0.003252219" ixy="0.0" ixz="0.0" iyz="0.0":`

Describes the inertia matrix of the link.

- `<visual>` : This block defines the visual representation of the link for visualization tools like RViz.
  - `<geometry>`: Specifies the geometry of the visual representation.
  - `<mesh filename="package://my_amr/meshes/Base_final.stl"/>` : Indicates that the visual geometry is represented by an STL mesh file located in the specified package.
  - `<material name="gray">` : Defines a material named "gray" for the link's visual appearance.
  - `<color rgba=".7 .7 .7 1" />`: Specifies the RGBA color of the material, where ".7 .7 .7"represents a shade of gray, and "1" is the alpha (transparency) value.
- `<collision>`: This block defines the collision properties of the link, used for physics simulation andcollision detection.
  - `<geometry>` : Specifies the collision geometry.
  - `<mesh filename="package://my_amr/meshes/Base_final.stl"/>:` Similar to the visual geometry, this indicates the link's collision is represented by the same STL mesh file.
  - `<contact_coefficients mu="1" mu2="1" kp="1e+13.0" kd="1.0"/>`: Sets contact coefficientsfor physics simulation, including friction (mu), second-order friction (mu2), stiffness (kp), and damping (kd).

Joints:

```
<joint name="rotor_right_back_joint" type="continuous">
<origin xyz="-0.108 0.19 -0.028  " rpy="0 0.0 0.0"/>
<parent link="base"/>
<child link="shaft_right_back"/>
<axis xyz="0.0 1 0"/>
</joint>\


<joint name="inner_right_back_joint" type="fixed">
<origin xyz="-0.0 0.022 0.0  " rpy="0 0.0 0.0"/>
<parent link="shaft_right_back"/>
<child link="inner_right_back"/>
<axis xyz="0.0 0.0 0.0"/>
</joint>


<joint name="outer_right_back_joint" type="fixed">
<origin xyz="0.0 0.0 0.0  " rpy="0 0.0 0.0"/>
```

```
<parent link="inner_right_back"/>
<child link="outer_right_back"/>
<axis xyz="0.0 0.0 0.0"/>
</joint>
```

Rotor Right Back Joint (rotor_right_back_joint**)**:

- name="rotor_right_back_joint": Specifies the name of the joint.
- type="continuous": Indicates that the joint is continuous, meaning it can rotate indefinitely.
- <origin> : Defines the position and orientation of the joint relative to its parent link (base in this case).
- <parent link="base"/>: Specifies the parent link connected to this joint.
- <child link="shaft_right_back"/>: Specifies the child link connected to this joint.
- <axis xyz="0.0 1 0"/> : Defines the rotational axis of the joint.

Inner Right Back Joint

(inner_right_back_joint)**:**

- name="inner_right_back_joint" : Specifies the name of the joint.
- type="fixed": Indicates that the joint is fixed, meaning it does not allow relative motion between theparent and child links.
- <origin>: Defines the position and orientation of the joint relative to its parent link (shaft_right_backin this case).
- <parent link="shaft_right_back"/>: Specifies the parent link connected to this joint.
- <child link="inner_right_back"/>: Specifies the child link connected to this joint.
- <axis xyz="0.0 0.0 0.0"/>: Since the joint is fixed, the axis values are set to zero.

Outer Right Back Joint (outer_right_back_joint**)**:

- name="outer_right_back_joint": Specifies the name of the joint.
- type="fixed": Indicates that the joint is fixed.
- <origin>: Defines the position and orientation of the joint relative to its parent link (inner_right_backin this case).
- <parent link="inner_right_back"/>: Specifies the parent link connected to this joint.
- <child link="outer_right_back"/>: Specifies the child link connected to this joint.
- <axis xyz="0.0 0.0 0.0"/>: Since the joint is fixed, the axis values are set to zero.

Gazebo plugins:

```
<gazebo>
<plugin
filename="libgazebo_ros_diff_drive.so"
name="gazebo_base_controller">
<odometry_frame>odom</odometry_frame>
<commandTopic>cmd_vel</commandTopic>
<publish_odom>true</publish_odom>
<publish_odom_tf>true</publish_odom_tf>
```

```
<update_rate>15.0</update_rate>
<left_joint>rotor_left_back_joint</left_joint>
<right_joint>rotor_right_back_joint</right_joint>
<wheel_separation>0.4</wheel_separation>
<wheel_diameter>0.13</wheel_diameter>
<max_wheel_acceleration>0.25</max_wheel_acceleration>
<robotBaseFrame>base</robotBaseFrame>


</plugin>
</gazebo>
```

- **<plugin>**: This tag defines a Gazebo plugin. The specified plugin, **libgazebo_ros_diff_drive.so** , is a library that provides a differential drive controller interface for ROS.
  - **filename="libgazebo_ros_diff_drive.so"**: Specifies the filename of the Gazebo plugin library.
  - name="**gazebo_base_controller**": Assigns a name to the plugin instance.
  - <**odometry_frame**>: Specifies the frame ID for odometry data.
  - <**commandTopic**>: Defines the ROS topic on which the simulated robot receives velocity commands.
  - <**publish_odom**>: Indicates whether to publish odometry information. In this case, it's set to true.
  - <**publish_odom_tf**>: Indicates whether to publish odometry transformation (TF). It's set to true.
  - <**update_rate**>: Specifies the rate at which the plugin updates the simulation, in Hertz. Here, it's set to 15.0 Hz.
  - <**left_joint**> and <**right_joint**>: Specify the joint names for the left and right wheels, respectively. These joints are used by the differential drive controller to simulate the motion of the robot.
- **<wheel_separation>**: Defines the separation between the left and right wheels.
- **<wheel_diameter>**: Specifies the diameter of the wheels.
- **<max_wheel_acceleration>** : Sets the maximum allowed acceleration for the wheels.
- **<robotBaseFrame>:** Specifies the base frame of the robot.

2. Launch files:

We have three launch files rviz launch file, start world launch file and gazebo launch file.

- Rviz launch file:

```
#!/usr/bin/python3
# -*- coding:
utf-8 -*-
import os
from ament_index_python.packages import
get_package_share_directory from launch
import LaunchDescription
from
launch_ros.actions
import Node from
scripts import
GazeboRosPaths def
generate_launch_descr
iption():
package_share_dir =
get_package_share_directory("my_amr")
urdf =
os.path.join(package_share_dir,
"urdf", "amr.urdf")
model_path, plugin_path, media_path = GazeboRosPaths.get_paths()
```

```
env = {
        "GAZEBO_MODEL_PATH": model_path,
    }

return
LaunchDescription(
[ Node(
package='robot_state_publish
er',
executable='robot_state_publ
isher',
name='robot_state_publisher'
, output='screen',
arguments=[urdf]),
Node(
package='joint_state_publisher_gu
i',
executable='joint_state_publisher
_gui',
name='joint_state_publisher_gui',
arguments=[urdf]),
Node(
package='rviz2',
executable='rviz
2',
name='rviz2',
output='screen'
),
])
```

- Shebang and Encoding Declarations:

These lines are typical Python script header lines. The shebang (`#!/usr/bin/python3`) specifies the path to the Python interpreter. The encoding declaration (`# -*- coding: utf-8 -*-`) specifies the character

- Import Statements:
  - `os`: Provides a way to interact with the operating system.
  - `ament_index_python.packages`: Used to get the package share directory.
  - `launch`: The main module for the ROS 2 system.
  - `launch_ros.actions`: Contains ROS-specific actions for the system.
  - `scripts.GazeboRosPaths`: A custom script or module (not provided in the code snippet) thatappears to define paths for Gazebo simulation.
- Function Definition:
  - This function defines the launch description. It is called when the launch file is executed.
- Package Share Directory:
  - Retrieves the share directory path for the ROS package named "my_amr."
- File Paths:
  - Constructs the path to the URDF (Unified Robot Description Format) file for the robot model.
- Gazebo Paths:
  - Obtains paths related to Gazebo simulation. The specific details depend on the implementation ofthe `GazeboRosPaths` module, which is not provided in the code snippet.
- Environment Variables:
  - Sets an environment variable (`GAZEBO_MODEL_PATH`) to the Gazebo model path.
- Launch Description:
  - Creates and returns a `LaunchDescription` object, which represents the set of nodes to belaunched.
- Node Configurations:

The script launches three nodes:

- `robot_state_publisher`: Publishes the state of the robot to the ROS Parameter Server.
- `joint_state_publisher_gui`: Provides a GUI for interactive control of joint states.
- `rviz2`: Launches RViz, a 3D visualization tool for ROS.

Each `Node`instance is configured with package name, executable, node name, output behavior, and

arguments (e.g., the URDF file for the robot state publisher).

Overall, this script sets up a basic ROS 2 simulation environment for a robot using Gazebo and RViz. The specific details may vary depending on the actual content of the **GazeboRosPaths** module and the package structure.

- Star world Launch file:

```
#!/usr/bin/python3
# -*-
coding: utf-
8 -*- import
os
from ament_index_python.packages
import
get_package_share_directory from
launch import LaunchDescription
from launch.actions
import
ExecuteProcess from
launch_ros.actions
import Node
from scripts import GazeboRosPaths


def generate_launch_description():
package_share_dir =
get_package_share_directory("my_a
mr") urdf_file =
os.path.join(package_share_dir,
"urdf", "amr.urdf") model_path,
plugin_path, media_path =
GazeboRosPaths.get_paths() env = {
"GAZEBO_MODEL_PATH": model_path, # as we only to add
amr(model) into gazebo models path
"GAZEBO_PLUGIN_PATH": plugin_path,
"GAZEBO_RESOURCE_PATH": media_path,
}
return
LaunchDescripti
on( [
ExecuteProcess(
cmd=["gazebo","-
s","libgazebo_ros_factory.so",],
output="screen",
additional_env=env,
```

```
),
Node(
package="gazebo_ros",
executable="spawn_entity.p
y",
arguments=["-entity","maze_bot","-b","-file", urdf_file,
],
),
Node(
package="robot_state_publ
isher",
executable="robot_state_pu
blisher", output="screen",
arguments=[urdf_file],
),
]
)
```

This is another Python script for launching a set of ROS 2 nodes to simulate a robot using Gazebo. Let's break down the code:

- Shebang and Encoding Declarations:

Similar to the previous script, these lines specify the Python interpreter and the character encoding of thesource file.

ExecuteProcess

- Import Statements:

The same imports as in the previous script. Notably, it imports from launch.actions, which is used to execute external processes as part of the launch.

"my_amr."

- Function Definition:

This function defines the launch description.

- Package Share Directory:

Obtains paths related to Gazebo simulation.

- Environment Variables:

Sets environment variables for Gazebo, specifying the model path, plugin path, and resource path.

`LaunchDescription`

- Launch Description:

Creates and returns a object, which represents the set of nodes and processes to belaunched.

`libgazebo_ros_factory.so`

- Node for Spawning Entity:

This node is responsible for spawning the robot entity in Gazebo. It uses `spawn_entity.py` script from the `gazebo_ros` package and specifies arguments like the entity name, a flag to prevent ROS-relatednamespaces, and the URDF file.

- Node for Robot State Publisher: `robot_state_publisher`

This node publishes the robot state to the ROS Parameter Server. It uses the

- **Gazebo launch file**:

```
#!/usr/bin/python3
#-*-
coding:
utf-8 -*-
import os
from
ament_index_python.packages
import
get_package_share_directory from
launch import LaunchDescription
from launch.actions
import
ExecuteProcess from
launch_ros.actions
import Node
from scripts import GazeboRosPaths

def generate_launch_description():
package_share_dir =
```

```
get_package_share_directory("my
_amr") urdf_file =
os.path.join(package_share_dir,
"urdf", "amr.urdf") model_path,
plugin_path, media_path =
GazeboRosPaths.get_paths() env =
{
"GAZEBO_MODEL_PATH": model_path, # as we only to add
maze_bot(model) into gazebo models path
"GAZEBO_PLUGIN_PATH": plugin_path,
"GAZEBO_RESOURCE_PATH": media_path,
}
return
LaunchDescrip
tion( [
ExecuteProcess(
cmd=["gazebo","-
s","libgazebo_ros_factory.so"
,], output="screen",
additional_env=env,
),
Node(
package="gazebo_ros"
,
executable="spawn_en
tity.py",
arguments=["-entity","amr_description","-b","-file",
urdf_file,
],
),
Node(
package="robot_state_pu
blisher",
executable="robot_state
_publisher",
output="screen",
arguments=[urdf_file],
),
])
```

- **Shebang and Encoding Declarations:**

These lines are standard Python script header lines, specifying the Python interpreter and the character encoding of the source file.

- Import Statements:

Similar to the previous scripts, it imports necessary modules for interacting with the operating system, defining launch configurations, and dealing with Gazebo-specific ROS actions.

- Function Definition:

This function defines the launch description.

- Package Share Directory:

Retrieves the share directory path for the ROS package named "my_amr."

- File Paths:

Constructs the path to the URDF file for the robot model.

- Gazebo Paths:

Obtains paths related to Gazebo simulation.

- Environment Variables:

Sets environment variables for Gazebo, specifying the model path, plugin path, and resource path.

- Launch Description:

Creates and returns a `LaunchDescription` object, which represents the set of nodes and processes to be launched.

- ExecuteProcess for Gazebo:

This launches Gazebo with the specified shared library and additional environment variables. It uses `libgazebo_ros_factory.so` as the server plugin.

- Node for Spawning Entity:

This node is responsible for spawning the robot entity in Gazebo. It uses the `spawn_entity.py` script from the `gazebo_ros` package and specifies arguments like the entity name, a flag to prevent ROS-related namespaces, and the URDF file.

- Node for Robot State Publisher:

This node publishes the robot state to the ROS Parameter Server. It uses the `robot_state_publisher` executable and provides the URDF file as an argument.

```
<?xml version="1.0" ?>
<sdf version="1.6">
<world name="default">
<static>true</static>
<include> <uri>model://sun</uri> </include>
<include> <uri>model://ground_plane</uri> </include>
</world>
</sdf>
```

3. World

This code is written in XML and represents a description of a world in the context of a simulation usingthe Simulation Description Format (SDF). SDF is commonly used in robotics and simulation environments, such as the Robot Operating System (ROS) and Gazebo.

`<?xml version="1.0" ?>:`

This line indicates that the document is an XML file with version 1.0.

`<sdf version="1.6">:`

The `<sdf>` element is the root element for the SDF document, and it specifies the version of the SDF being used. In this case, version 1.6.

`<world name="default">:`

Inside the `<sdf>` element, there is a `<world>` element. It defines a world within the simulation. The `name`attribute is set to "default" for this world.

`<static>true</static>:`

Within the `<world>` element, there is a `<static>` element. This element indicates whether the objects in the world are static or not. Here, it is set to `true`, meaning the objects are static.

`<include><uri>model://sun</uri></include>:`

The `<include>` element is used to include other elements or models in the world. Here, it includes a model with the URI (Uniform Resource Identifier) "`model://sun`". This could be a representation of a sun in the simulation.

`<include><uri>model://ground_plane</uri></include>:`

Similarly, another `<include>`element is used to include a model with the URI "model://ground_

plane".
This could represent a ground plane in the simulation.

`</world>:`

Closing tag for the `<world>` element, indicating the end of the world description.

`</sdf>:`

Closing tag for the `<sdf>` element, indicating the end of the SDF document.

In summary, this XML code describes a simulation world using the Simulation Description Format. The worldis named "default," and it includes static objects such as a sun and a ground plane. The specific details of these objects are likely defined in separate files with the URIs "`model://sun`" and "`model://ground_plane`."

Chapter. 5

Kinematics and Control: -

Wheel selection: -

Types of wheels:

1- Conventional wheels

-Fixed -Non-Fixed

(Figure 5.1 Fixed wheel)

2- Non-Conventional Wheels

-Mecanum Wheel -Omni wheel

(Figure 5.2 Mecanum Wheel)

(Figure 5.3 Omni Wheel)

We selected the fixed conventional wheel for its grip on the ground and its ability to withstand pressure and mostly its availability and cost.

-Steering systems:

Types of steering systems:

1- Ackerman steering system

2- Differential steering
3- Mecnum steering
4- Omni steering
5- Skid steering

- Suitable steering system

-Differential steering system

Because the differential steering has four cases where the robot can move forward and backward in a straight line third it can rotate around its center with an arc equal to the ratio between the velocity of the right side and the left side and the last case and the most crucial for our application is that it can rotate with zero radius around its axis clock wise and counter clock wise.

Finding if our robot is holonomic or Non-holonomic:

Based on our wheel configuration our robot is a non-holonomic robot because it can't move in the lateral direction so it can be integrated in the X-direction and the rotation about Z-axis and it moves in the XY-plane

Finding Kinematics: -

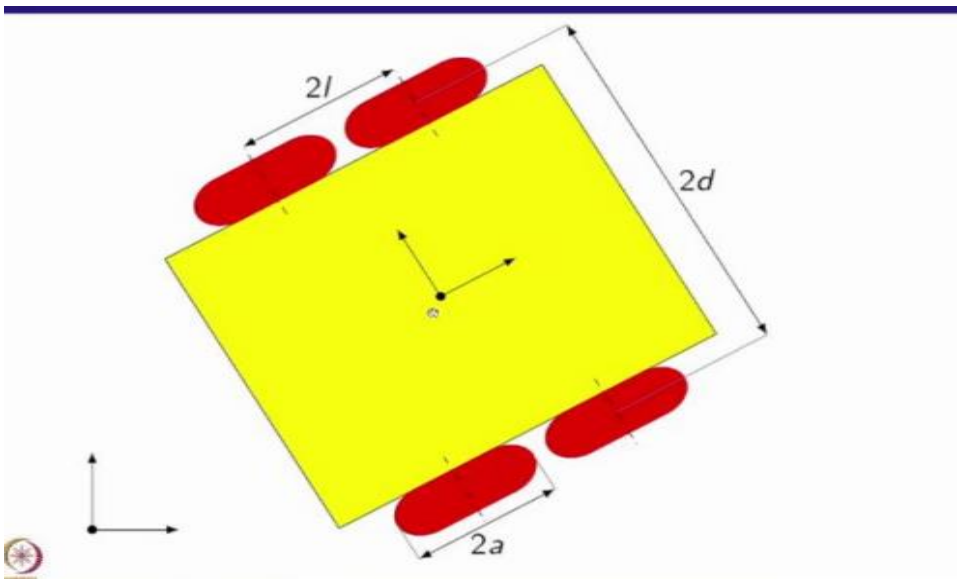
the branch of mechanics concerned with the motion of objects without reference to the forces which cause the motion.

Wheel configuration matrix: General equation

$$\omega_i = \begin{bmatrix} \frac{1}{a_i} & \frac{1}{a_i}\tan\phi_i \end{bmatrix} \begin{bmatrix} \cos\theta_{Bi} & \sin\theta_{Bi} \\ -\sin\theta_{Bi} & \cos\theta_{Bi} \end{bmatrix} \begin{bmatrix} 1 & 0 & -d_{yi} \\ 0 & 1 & d_{xi} \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix}$$

ω_i - Angular velocity of the i^{th} wheel.
a_i - Radius of the i^{th} wheel.
θ_{Bi} - Angle between the vehicle frame (B) to the wheel frame (c_i).
d_{xi} and d_{yi} are the position coordinates of c_i with reference to B.
ϕ_i - Angle between roller axis to the x_{ci} axis.
u: Forward velocity of the mobile robot w.r.t. frame B.
v: Lateral velocity of the mobile robot w.r.t. frame B.
r: Angular velocity of the mobile robot w.r.t. frame B.

(figure 5.4 Wheel configuration matrix)



(Figure 5.5 Top view for our robot)

The (d) is the distance from the robot center to the wheel in the y-direction and the (l) is the the distance from the center in the x-direction

Wheel configuration for our robot :-

$$\omega_1 = \begin{bmatrix} \frac{1}{a} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -d \\ 0 & 1 & l \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \quad (5\text{-}1)$$

$$\omega_2 = \begin{bmatrix} \frac{1}{a} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -d \\ 0 & 1 & -l \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \quad (5\text{-}2)$$

$$\omega_3 = \begin{bmatrix} \frac{1}{a} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & d \\ 0 & 1 & l \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \quad (5\text{-}3)$$

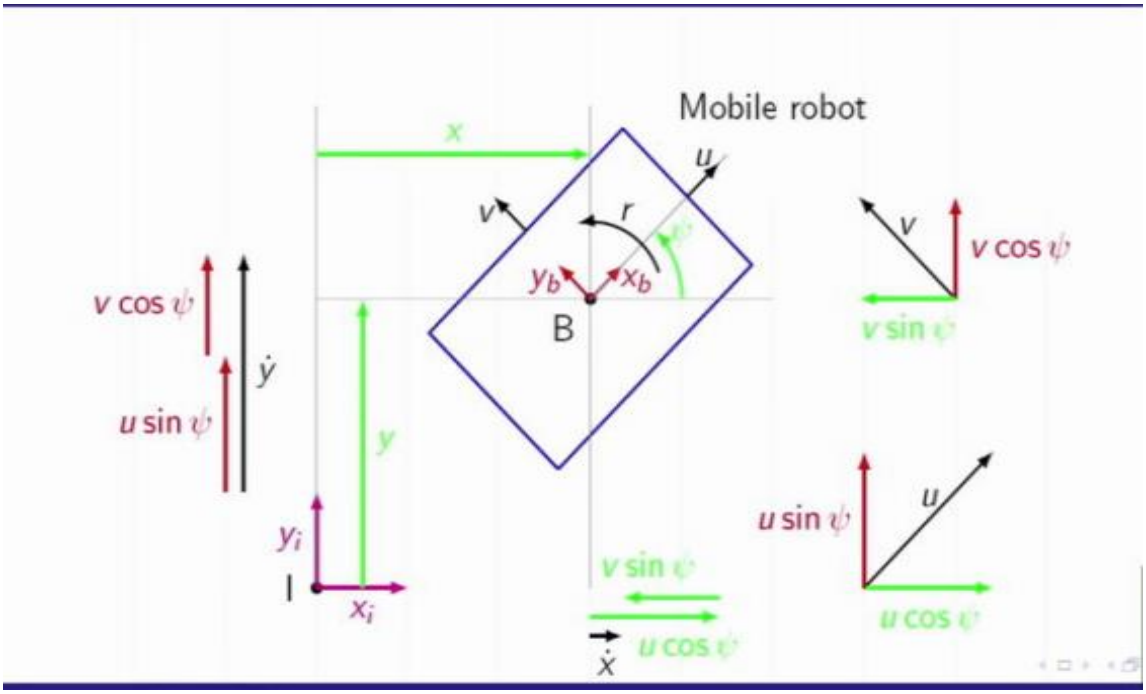
$$\omega_4 = \begin{bmatrix} \frac{1}{a} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & d \\ 0 & 1 & -l \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix} \quad (5\text{-}4)$$

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{a} & \frac{-d}{a} \\ \frac{1}{a} & \frac{-d}{a} \\ \frac{1}{a} & \frac{d}{a} \\ \frac{1}{a} & \frac{d}{a} \end{bmatrix} \begin{bmatrix} u \\ r \end{bmatrix} \quad (5\text{-}5)$$

$$\begin{bmatrix} u \\ r \end{bmatrix} = \begin{bmatrix} \frac{a}{4} & \frac{a}{4} & \frac{a}{4} & \frac{a}{4} \\ \frac{-a}{4d} & \frac{-a}{4d} & \frac{a}{4d} & \frac{a}{4d} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \quad (5\text{-}6)$$

$$\text{Zeta=W*omega=}\begin{bmatrix} u \\ v \\ r \end{bmatrix} = \begin{bmatrix} \frac{a}{4} & \frac{a}{4} & \frac{a}{4} & \frac{a}{4} \\ 0 & 0 & 0 & 0 \\ \frac{-a}{4d} & \frac{-a}{4d} & \frac{a}{4d} & \frac{a}{4d} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \qquad (5\text{-}7)$$

The jacopian for the robot:



(Figure 5.6 jacopian for the robot)

This figure shows the body frame position w.r.t the inertial fram so we deduce the jacopian as the rotation matrix around z-axis of the body fram w.r,t the inertial frame

$$\text{J(psi)=}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \Psi \end{bmatrix} = \begin{bmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ r \end{bmatrix}$$

- Eta-dot = J(psi)*zeta (Forward kinematics)

Where the forward kinematics shows the effect of the velocity of each wheel on the velocity vector where the eta-dot is the velocity vector and the zeta is the velocity input command.

- Zeta = Eta-dot*inv(J(psi)) (inverse kinematics)

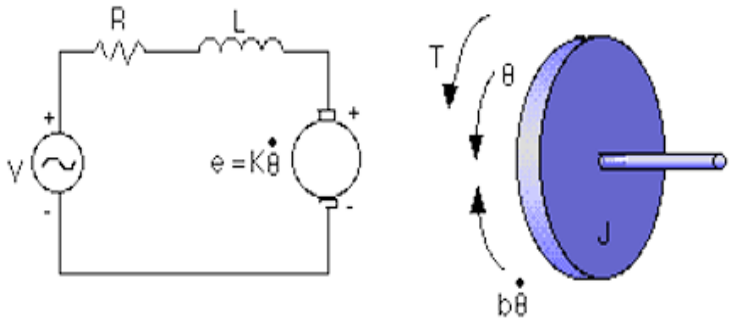
While inverse kinematics shows the required speed to reach a desired postion.

CONTROL

1- DC-Motor modeling

5.1- DC-Motor Parameters:

A common actuator in control systems is the DC motor. It directly provides rotary motion and it can be easily connected to wheels, gears, etc. The dc-motor can be represented by a simple electrical circuit and a free body diagram of a rotor as shown in (Figure 5.7)



(Figure 5.7 dc-motor)

In this model the input volt (V) and the output is the speed of the Shaft $\boldsymbol{\omega}$. By taking into consideration the all parameters of the motor like friction and other we can say that the psychical parameters of the motor are:

- V: voltage applied to the motor [volt]
- i: The current of the motor [Ampere]
- **θ**: The angel of the motor [Rad]
- θ: The speed of the motor [Rad/sec]
- T: Motor torque [N.m]
- J: Moment of inertia of the rotor [kg.m^2]
- b: Motor viscous friction constant [N.m.s]
- Ke: Electromotive force constant [V/rad/sec]
- Kt: Motor torque constant [N.m/Amp]
- R: Electric resistance [Ohm]
- L: Electric inductance [H]

5.2- DC-Motor Equations:

Generally, the Torque of the motor (T) is directly proportional with the Current of the motor and the strength of the magnetic field. since the magnetic field is constant the torque is proportional with the current only by a constant ($\boldsymbol{Kt}$). The torque constant ($\boldsymbol{Kt}$) is specific to motor's design, including its magnetic strength, number of wire turns, and armature length. The slope of the motor's torque-current curve is determined by the torque constant.

$$\mathbf{T}=\boldsymbol{Kt}*\boldsymbol{i} \quad \textbf{(5-8)}$$

The back emf is directly proportional to the speed of the motor shaft by a constant ($\boldsymbol{Ke}$).

$$\mathbf{e}=\boldsymbol{Ke}*\boldsymbol{\theta} \quad \textbf{(5-9)}$$

In SI units, the motor torque and back emf constants are equal, that is, $\boldsymbol{Ke}=\boldsymbol{Kt}$, we will use ($\boldsymbol{K}$) to represent both the motor torque constant and the back emf constant.

By applying Kirchhoff's voltage law in the circuit.

$$\mathbf{l}*\boldsymbol{di}/\boldsymbol{dt}+\boldsymbol{R}*\boldsymbol{i}=\boldsymbol{V}-\boldsymbol{e} \quad \textbf{(5-10)}$$

By applying newton second law

$$\mathbf{T}=\boldsymbol{J}*\boldsymbol{\theta}+\boldsymbol{b}*\boldsymbol{\theta} \quad \textbf{(5-11)}$$

By substation of equation (5-9) in (5-10)

$$\mathbf{l}*\boldsymbol{di}\,\boldsymbol{dt}+\boldsymbol{R}*\boldsymbol{i}=\boldsymbol{V}-\boldsymbol{K}*\boldsymbol{\theta} \quad \textbf{(5-11)}$$

By substation of equation (5-8) in (5-11)

$$\mathbf{K}*\boldsymbol{i}=\boldsymbol{J}*\boldsymbol{\theta}+\boldsymbol{b}*\boldsymbol{\theta} \quad \textbf{(5-12)}$$

By applying Laplace transform to equation (5-11) & (5-12)

$$(\boldsymbol{L}*\boldsymbol{s}+\boldsymbol{R})\boldsymbol{i}(\boldsymbol{s})=\boldsymbol{V}(\boldsymbol{s})-\boldsymbol{k}*\boldsymbol{s}*\boldsymbol{\theta}(\boldsymbol{s}) \quad \textbf{(5-13)}$$

$$\mathbf{K}*\boldsymbol{i}(\boldsymbol{s})=\boldsymbol{s}(\boldsymbol{J}*\boldsymbol{s}+\boldsymbol{b}*\boldsymbol{s})\boldsymbol{\theta}(\boldsymbol{s}) \quad \textbf{(5-14)}$$

By subtitling $\boldsymbol{i}(\boldsymbol{s})$ from equation (5-14) into equation (5-13)

$$(\ (\boldsymbol{L}*\boldsymbol{s}+\boldsymbol{R})(\boldsymbol{J}*\boldsymbol{s}+\boldsymbol{b}*\boldsymbol{s})+\mathrm{k}2/\ \boldsymbol{k}\ )\boldsymbol{s}\boldsymbol{\theta}(\boldsymbol{s})=\boldsymbol{V}(\boldsymbol{s}) \quad \textbf{(5-15)}$$

$$((\ (\boldsymbol{L}*\boldsymbol{s}+\boldsymbol{R})(\boldsymbol{J}*\boldsymbol{s}+\boldsymbol{b}*\boldsymbol{s})+\boldsymbol{K}\mathbf{2})\ /\boldsymbol{k}\ )=\boldsymbol{V}(\boldsymbol{s})\ \boldsymbol{\theta}(\boldsymbol{s}) \quad \textbf{(5-16)}$$

$\boldsymbol{\theta}(\boldsymbol{s})\ /\boldsymbol{V}(\boldsymbol{s})=\boldsymbol{k}/\ ((\boldsymbol{L}*\boldsymbol{s}+\boldsymbol{R})(\boldsymbol{J}*\boldsymbol{s}+\boldsymbol{b}*\boldsymbol{s})+\boldsymbol{K}\mathbf{2}\ )$(11) SPEED VOLT EQUATION

$\boldsymbol{\theta}(\boldsymbol{s})/\ \boldsymbol{V}(\boldsymbol{s})=\boldsymbol{k}/\ (\boldsymbol{s}*((\boldsymbol{L}*\boldsymbol{s}+\boldsymbol{R})(\boldsymbol{J}*\boldsymbol{s}+\boldsymbol{b}*\boldsymbol{s})+\boldsymbol{K}\mathbf{2}))$ (12)POSITION VOLT EQUATION

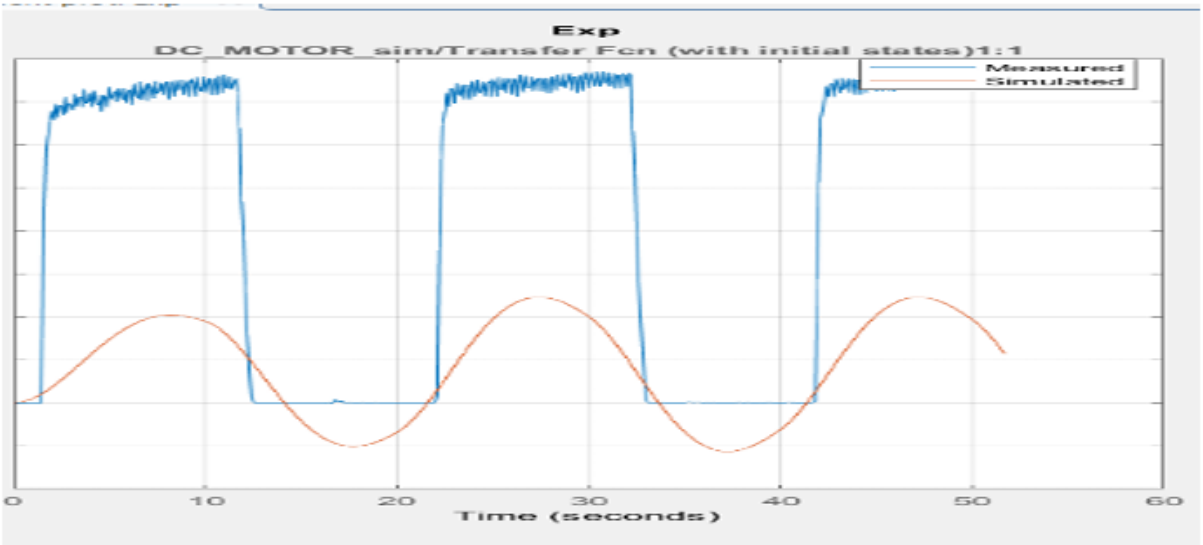
5.3- Parameters acquiring:

In normal cases most of the parameters can be acquired form the motor datasheet but since the motor we are working with is of an unknown origin and the datasheet could not be acquired so other methods are used to acquire those parameters the method that we use is curve fitting

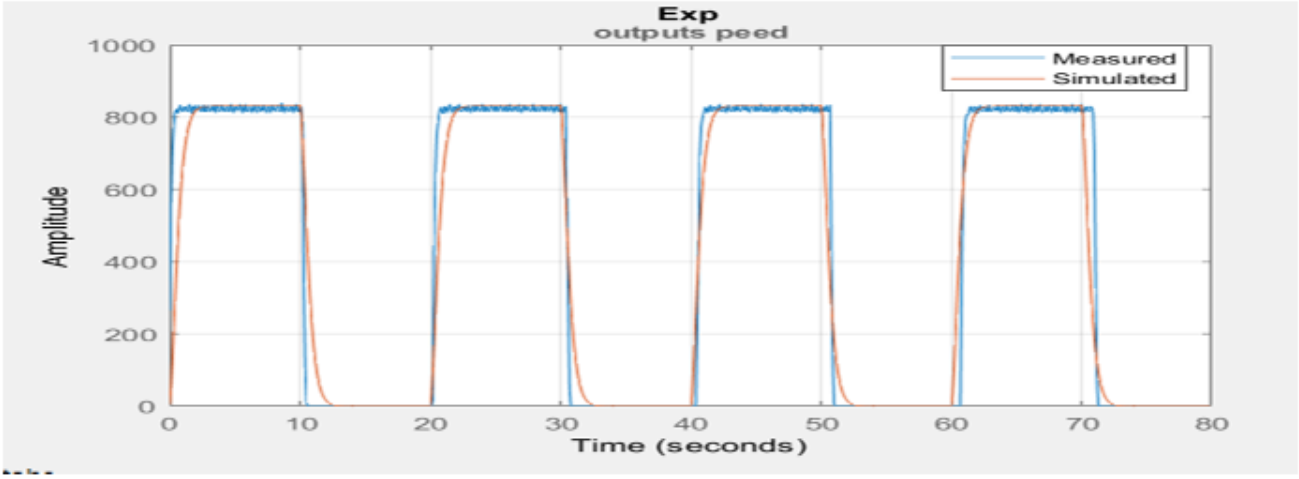
method.

Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, those series of data is acquired by applying a known input to the system (volt) and recording the output of the system (speed or angle). Curve fitting can involve either interpolation, where an exact fit to the data is required, or smoothing, in which a "smooth" function is constructed that approximately fits the data.

This process is done using the MATLAB Simulink parameter estimation tool were the input (volt) and the output (speed) is applied then a series of iteration is done to find the best values of the parameters to fit the output as seen in figure 4-8 and 4-9.



(Figure 5.8 values of the parameters to fit the output)



(Figure 5.9 values of the parameters to fit the output)

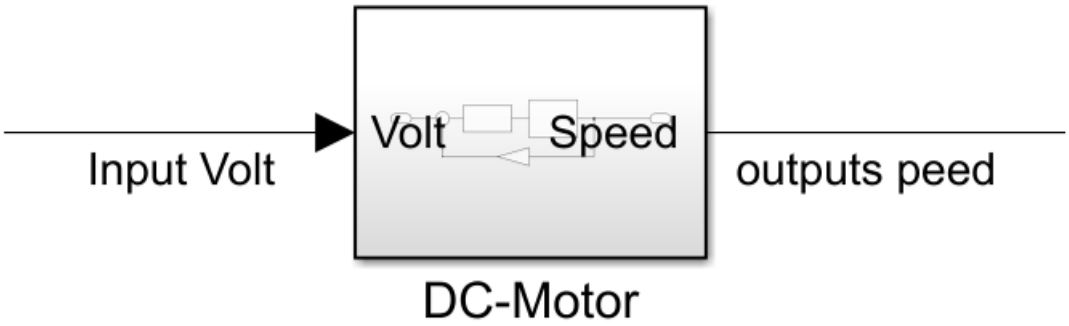
The result of these operation is that we were able to acquire the parameters of the dc motor without the datasheet but by logging the input and the output of the motor and finding the best parameters that can make the model respond similarly.

(Table 5.1 Value of Parameters)

Name of parameter	*Value*
B	*0.2055*
J	*0.0105*
K	*0.9070*
L	*0.3943*
R	*7.7359*

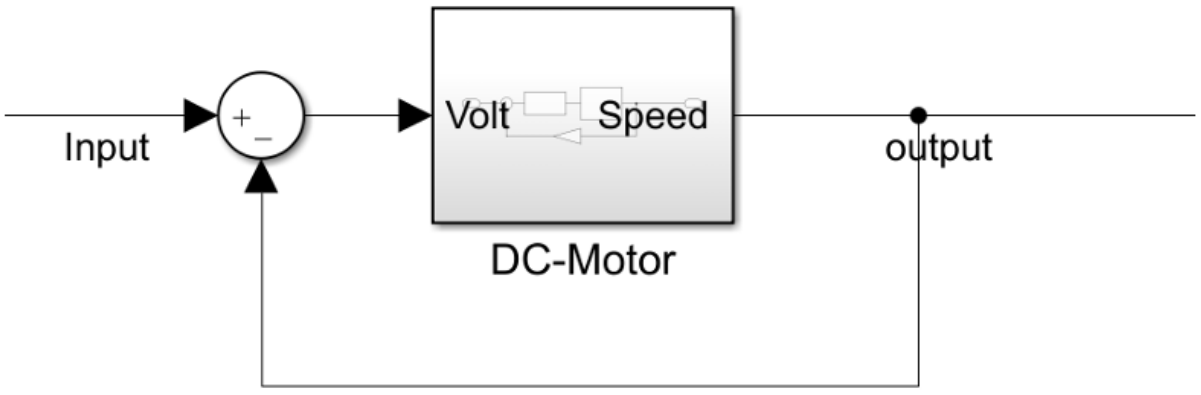
Why use a feedback control?

in the last part we found a relation between the speed and the volt we set a specific volt to get a speed but there is no way of knowing if the motor has reached the required speed that is called an Open loop system where the output of the model does not affect the input.



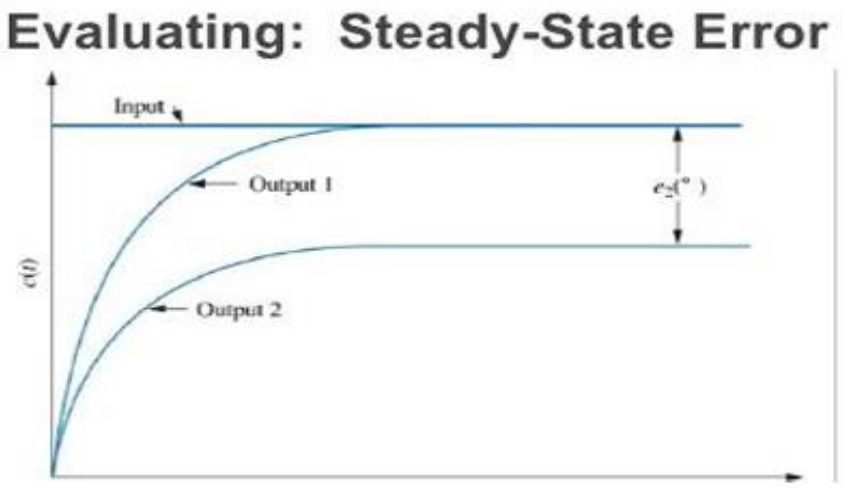
(Figure 5.10 open loop system)

So, by adding any disturbance to the system the output well change therefore we use a feedback from the output to be affect the inputs and that is called a Close loop system.



(Figure 5.11 closed loop system)

What does the feedback do? By which a measure of the real output of the system that includes the effect of the disturbances and compare it by the input. The comparison between the input (the demand value) and the output (the obtained value) give the system error that depends on all the system imperfection effects including the disturbances. The error signal can be obtained by means of mathematical subtraction that normally obtained by a device called comparator. The feedback will not solve the problem interlay there will be a SteadyState error



(Figure 5.12 Steady –state Error)

So, know we have the steady state problem and we have another problem that in this current application we want to be able to insert a required speed not a volt. The beat solution to these problems is to use a controller and the controller we will be using is an PI controller.

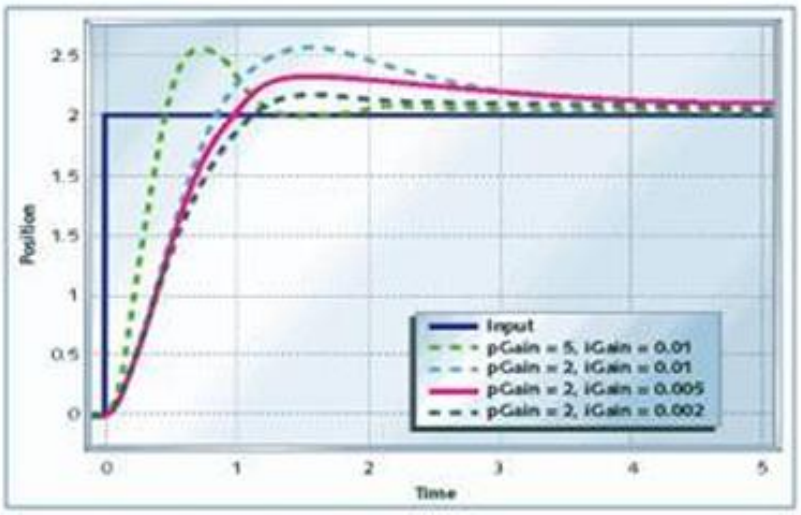
5.4- PID Controller: Proportional–integral–derivative controller (PID controller) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value as the difference between a desired set point (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively) as shown in figure 5-12.

let first discus each term of the controller alone.

P- Controller: Proportional or P- controller gives output which is proportional to current error e (t). It compares desired or set point with actual value or feedback process value. The resulting error is multiplied with proportional constant to get the output. If the error value is zero, then this controller output is zero. This controller never reaches the steady state condition. It provides stable operation but always maintains the steady state error.

PI- Controller: Due to limitation of p-controller where there always exists an offset between the process variable and set point, I-controller is needed, which provides necessary action to eliminate the steady state error. It integrates the error over a period of time until error value reaches to zero. It holds the value to final control device at which error becomes zero. Integral

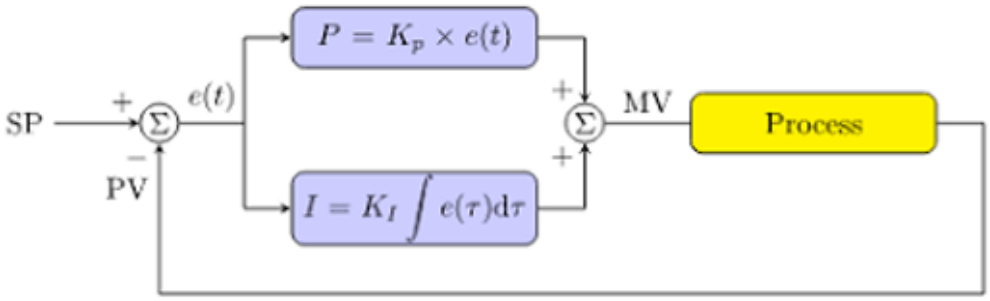
control decreases its output when negative error takes place. It limits the speed of response and affects stability of the system. Speed of the response is increased by decreasing integral gain Ki.



(Figure 5.13 PI control response)

In figure 5-11, as the gain of the I-controller decreases, steady state error also goes on decreasing. For most of the cases, PI controller is used particularly where high speed response is not required.

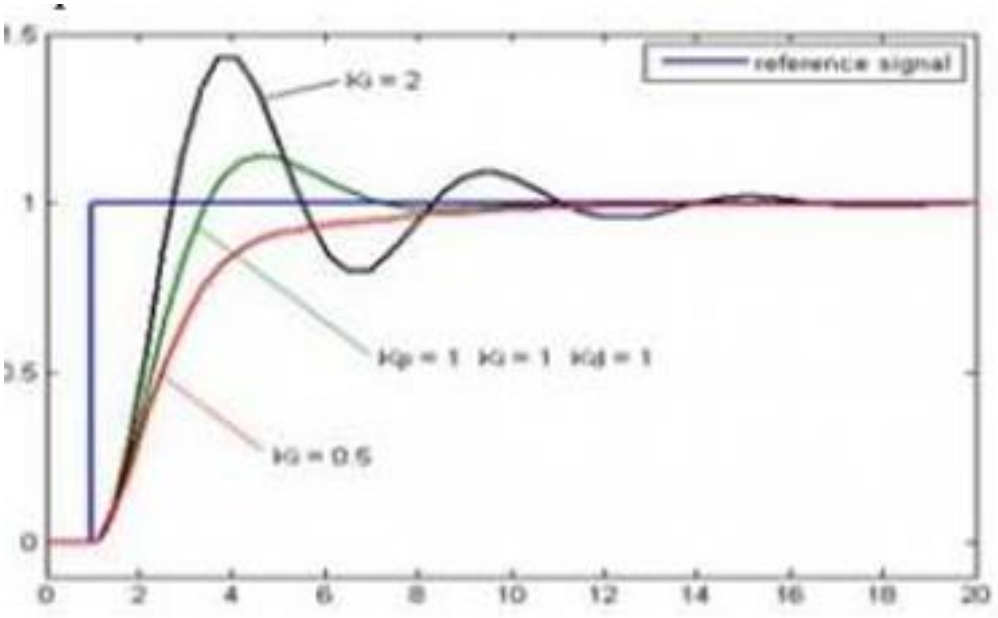
While using the PI controller, I-controller output is limited to somewhat range to overcome the integral wind up conditions where integral output goes on increasing even at zero error state, due to nonlinearities in the plant.



(Figure 5.14 PI Controller)

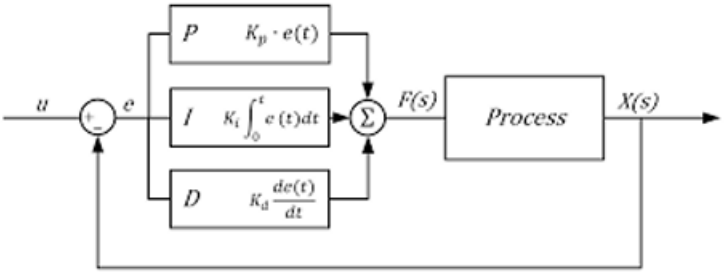
PID- Controller:

I-controller doesn't have the capability to predict the future behavior of error. So, it reacts normally once the set point is changed. D-controller overcomes this problem by anticipating future behavior of the error. Its output depends on rate of change of error with respect to time, multiplied by derivative constant. It gives the kick start for the output thereby increasing system response.



(Figure 5.15 PID Controller response)

In the above figure response of D controller is more, compared to PI controller and also settling time of output is decreased. It improves the stability of system. Increasing the derivative gain increases speed of response. But the major drawback of the D-controller is that any spike in the data or bid disturbance will make the controller behave badly so we need a filter on the derivative term to make it behave normally.



(Figure 5.16 PID Controller)

5.5- Choosing a controller:

Since the loaded motor response is slow so will be applying a PI-Controller and the equation representing the controller is

e(t)=$setpoint-processvalue$ (5-17)

u(t)=$kp*e(t)+ki*\int e(t)dt$ (5-18)

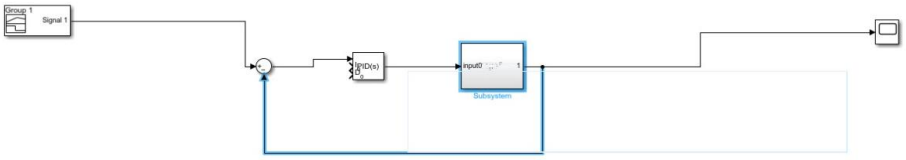
But since the motor is being controller by a discrete system it is needed to transfer this function to the z-domain

U(z)=$kp*E(z)+ki*\frac{z+1}{z-1}*Ts*E(s)$ (5-19)

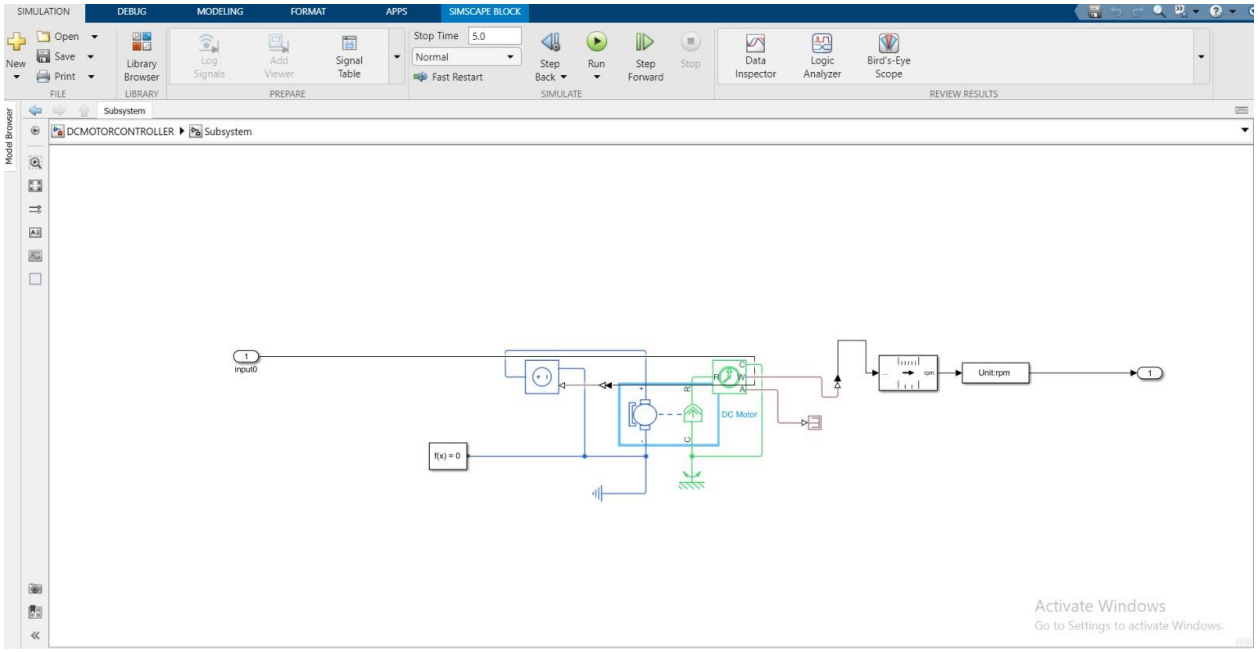
W here Ts: is the sampling time

5.6- Tuning the controller:

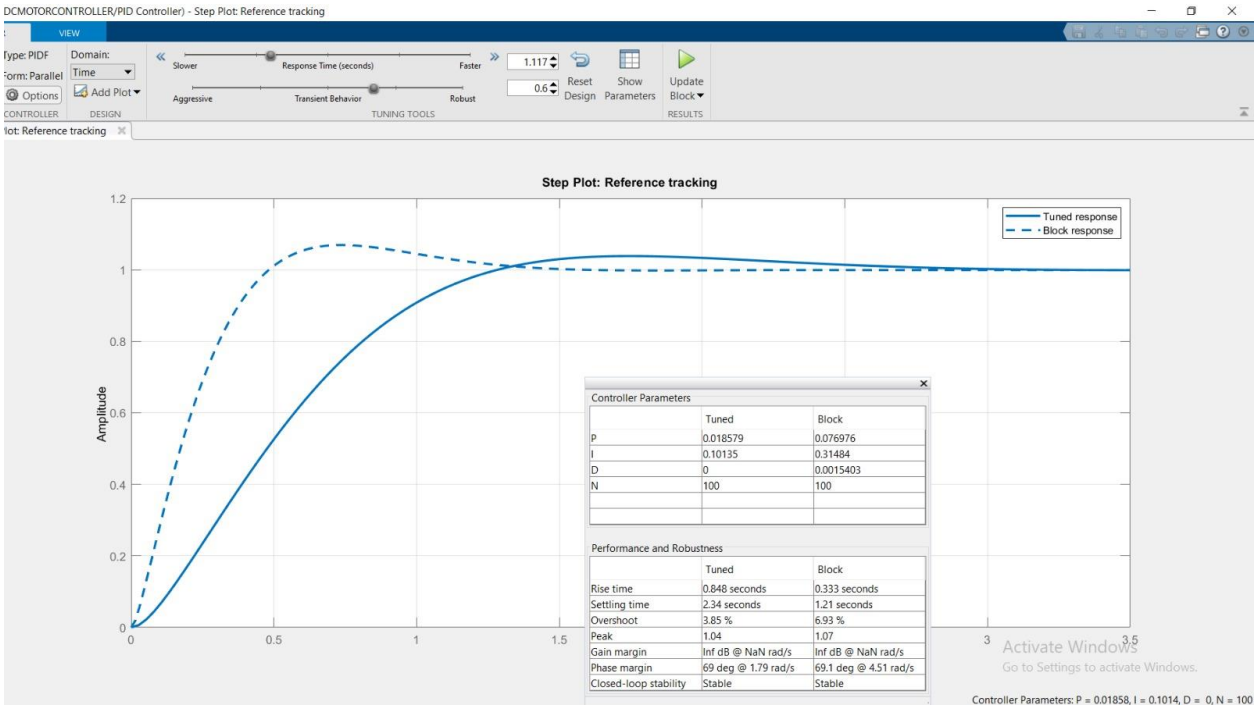
Tuning is the process in which we a acquire the values of kp and ki there are many different way and tools by which we can acquire those variables like trial and error but since we have a model of the dc motor we can easily get the variables using MATLAB SIMULINK



(Figure 5.17 pi-controller using MATLAB SIMULINK)



(Figure 5.18 DC-motor model)



(Figure 5-19)

By this tool we can easily change the Parameters of system like Overshoot, settling time.

Conclusion:

By acquiring a relation between the motor input (volt) and the output (speed or angle) and get all the parameters we can construct a controller to be able set a required speed or angle and overcome any disturbance

Chapter 6

Practical work

In the practical implementation of the project, we performed three operations to be able to assemble the robot's body.

- The first was to use laser cutting on the metal plate.
- The second was the bending machine to form the sides of the car for the upper and lower parts.
- The third was argon welding to assemble the sides of the upper part with the upper metal plate and the sides of the lower part with the Lower metal plate.

As for the bearings, we made a metal cover for it and this cover was attached to the robot's body, and the bearings were installed and removed using a tool called a claw puller.

We also have motor holders. We used a laser machine to cut the metal plate and make the necessary holes for the motor shaft, the motor mounting screws, and the holes for fixing the holder in the robot's body. After cutting the laser, we used a bending machine to make the holder at a right angle.

All parts were manufactured based on the dimensions and sizes in the design and drawing sheets that we made in the first part of the project.

Process sheets :

For the Top part

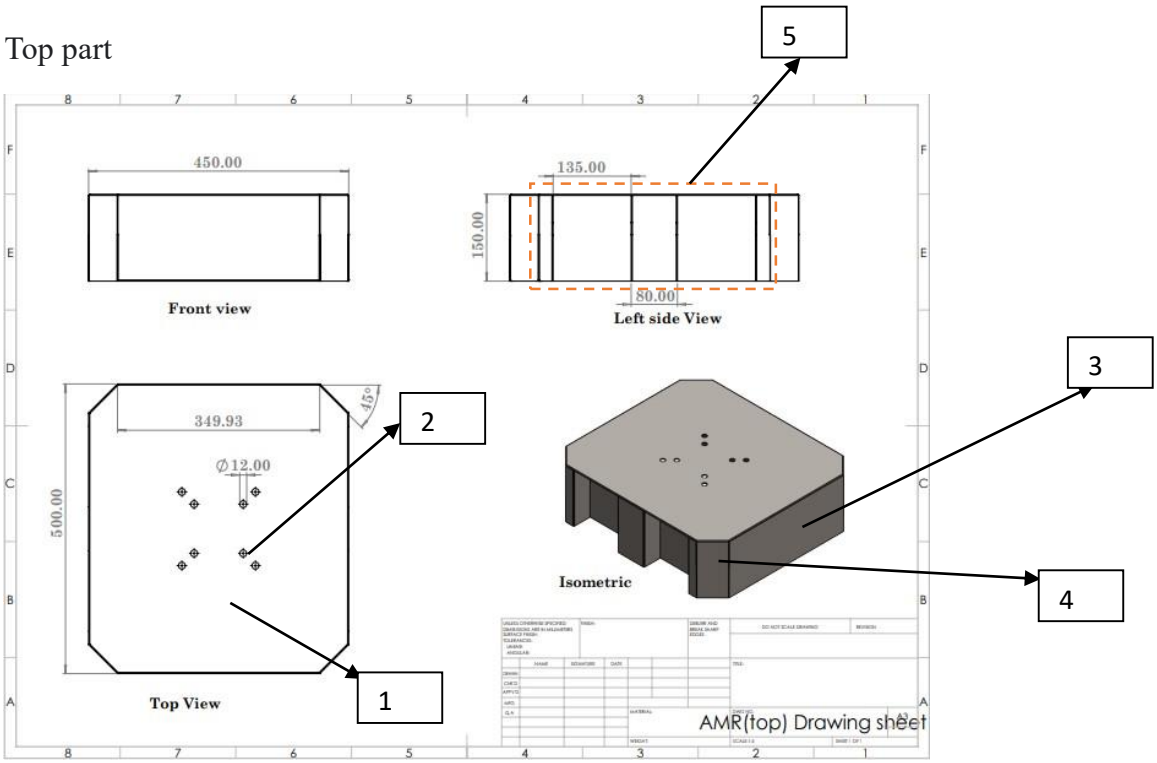


Table: process sheet

Part name (Top) Material: AISI316 Alloy steel

Quantity 1 Cycle time: min

(Table 6.1 process sheet Part name (Top))

Machine	Operations	Tools & gauge	Time
- Laser cutting	Check the metal plate size Fix the sheet metal and Calibration	Vernier and Tape measuring	
-	Laser machine take the drawing for the part in Auto cad file format	Computer connected to the machine	
-	Machine start cutting outer boundary 500*450 mm of upper sheet metal (1)	Laser beam	
-	Then after finish (1) it's start to cutting 8 holes ϕ12mm (2)	Laser beam	
Laser cutting	Then after finish (2) it's start to cutting (3) 350*150 mm (two times)	Laser beam	
Laser cutting	Then start cutting (4) 70*150mm (four times)	Laser beam	
Bending	Check size of the Sides (5) of the Top part 150*1000 mm use drawing sheet to follow the shape	Vernier and Tape measuring	
- - - - - - - - -	After 25.6 mm in length After 48 mm in length After 135 mm in length After 48 mm in length After 80 mm in length After 48 mm in length After 135 mm in length After 48 mm in length After 25.6 mm in length	Right angle die - - - - - - - -	
Argon welding	Assembly the Sides no.5 and no.4 and no.3 with no.1	Electrode	

Process sheets :

For the Base part

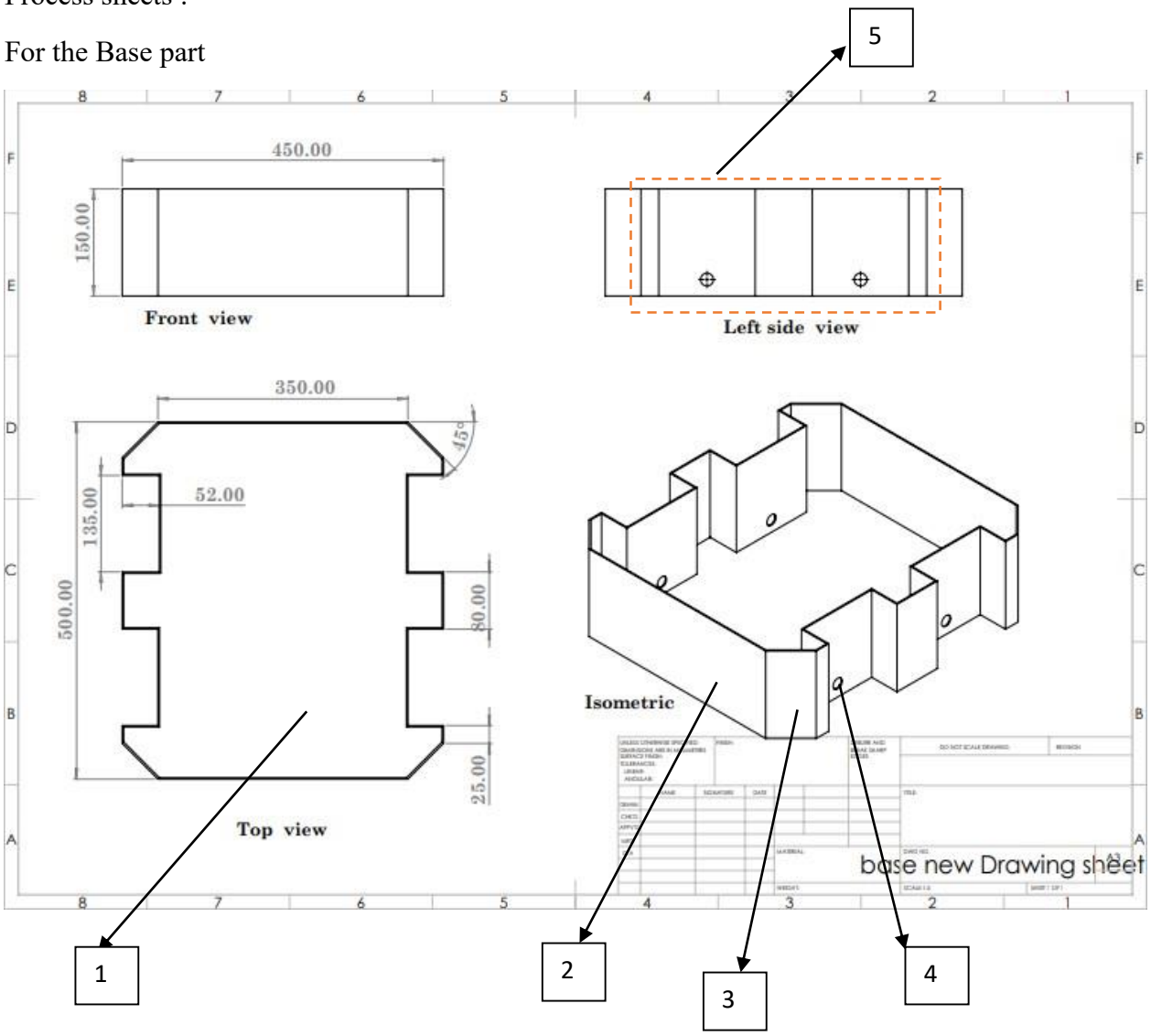


Table: process sheet

Part name (Base) Material: AISI316 Alloy steel

Quantity 1 Cycle time: min

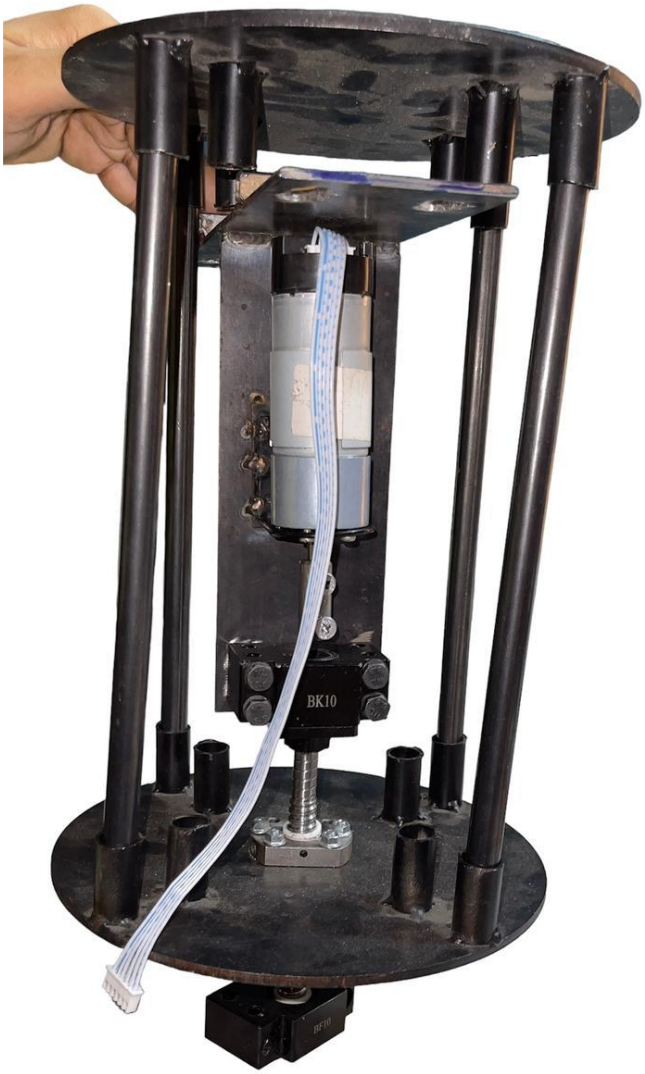
(Table 6.2 process sheet Part name (Base))

Machine	Operations	Tools & gauge	Time
- Laser cutting	Check the metal plate size Fix the sheet metal and Calibration	Vernier and Tape measuring	
-	Laser machine take the drawing for the part in Auto cad file format	Computer connected to the machine	
-	Machine start cutting outer boundary 500*450 mm of upper sheet metal (1)	Laser beam	
Laser cutting	Then after finish (1) it's start to cutting (2) 350*150 mm (two times)	Laser beam	
Laser cutting	Then start cutting (3) 70*150mm (four times)	Laser beam	
Laser cutting	Then after finish (3) it's start to cutting 4 holes ϕ15mm (4)	Laser beam	
Bending	Check size of the Sides (5) of the Top part 150*1000 mm use drawing sheet to follow the shape	Vernier and Tape measuring	
- - - - - - - - - -	After 25.6 mm in length After 48 mm in length After 135 mm in length After 48 mm in length After 80 mm in length After 48 mm in length After 135 mm in length After 48 mm in length After 25.6 mm in length	Right angle die - - - - - - - -	
Argon welding	Assembly the Sides no.5 and no.2 and no.3 with no.1	Electrode	

AMR Robot Final Look:

(Figure 6.1 AMR Final form)

(Figure 6.2 AMR Final form)



(Figure 6.3 ball Screw system)

Chapter 7

Real Robot Implementation

7.1- Embedded Layer

7.1.1- Parts

As we remember at chapter 4, we have talk about our project different components:

1. ESP32

2. MPU 9250

3. Magnetic Rotary Encoders

4. Motor Drivers (cytron MDD10A)

ESP32

Why ESP32? • After searching we saw that we can't depend totally on Raspberry pi through all tasks. Because that's design principle has many problem:

1. Complexity of accessing bare metal layer at Raspberry pi.
2. Leakage of components libraries for Raspberry pi exactly those which in cpp[1]
3. Power Feedback mistakes which could may damage Raspberry in just a seconds.

- ESP32 is one of oldest MCU's which has board manager at arduino IDE,

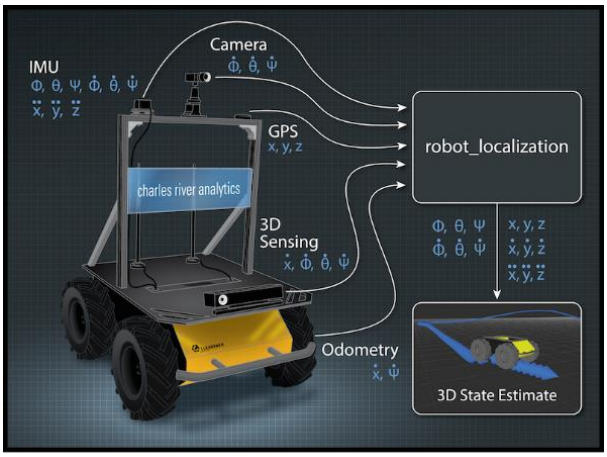
that would provide us with many resources, libraries and community sup-

port.

- ESP32 is high performance because of it's multi-core and low cost. So it's reliable at variety of task especially those which in Real-time and No fear of damaging it while learning of wrong wiring because of its low cost.

[1]we have to use Cpp in different applications like MPU, because it provide more accuracy for those measurements

7.1.2- MPU9250 and Magnetic rotary encoders



May some of readers get confused of men- tioning MPU9250 and encoders at the same section. After our robotics studying we gain information that many routes could get the same result by simpler way ”All roads lead to Rome”.

Rome of our project is odometery. Odome- tery is the backbone of project. it is the feedback of robot location away from origen. We decided to depend on IMU and encoders of getting our odometery using kalmn filter (Figure 7.1: robot localization pack- age inputs ability)

I think now you could to understood why we mentioned MPU9250 and Encoders together.

7.1.3 Motor Drivers (cytron MDD10A)

As we mentioned previously what is the usage of motor drivers and how it works.

7.2 Embedded layer codes algorithm

7.2.1 Preparation

ESP32 board manager

Arduino IDE is open source IDE which allow developers to add their libraries and board managers.

What is board manager?

Board manager is combination of peripheral libraries give the user access

on pointers using simple commands like: digital Read (), digital Write () and ...etc.

Arduino IDE at its earliest version it was made for Arduino boards only. But after several years and several version developers started to feel very comfort with Arduino language, so they started to develop more affordable boards to be usable Arduino ide and its language. From here we decided to install ESP32 board manager, we checked it by running simple examples like blinking LEDs.

7.2.2 Magnetic Rotary Encoders: Working Principle

Magnetic rotary encoders are devices used to measure the angular position or motion of a rotating shaft. Here's a summary of their working principle:

Components

•Magnet: Typically, a small, cylindrical magnet attached to the rotating shaft.

•Sensor: Usually a Hall effect sensor or a magneto resistive sensor positioned near the magnet.

•Signal Processing Unit: Converts the raw sensor data into a read- able output.

Working Principle

1.Magnetic Field Generation: As the shaft rotates, the attached magnet also rotates, creating a varying magnetic field.

2.Detection by Sensor: The sensor, which is placed close to the rotating magnet, detects changes in the magnetic field. Hall effect sensors measure the voltage changes caused by the magnetic field, while magneto resistive sensors measure changes in resistance.

3.Signal Conversion: The sensor outputs an analog or digital signal corresponding to the position of the magnetic field. This signal is often sinusoidal or consists of pulses.

4.Interpolation and Signal Processing: The signal processing unit interpolates the raw signal from the sensor to provide a precise angular position. This may involve converting the analog signal to digital, filtering noise, and calibrating the output.

5.Output: The processed signal is output in a form suitable for the application, such as an angle reading in degrees, a pulse count, or a digital code. Common output formats include incremental (providing position relative to a reference point) or absolute (providing exact position).

Types of Magnetic Rotary Encoders

•Incremental Encoders: Provide relative position information by generating pulses as the shaft rotates. The number of pulses per revolution indicates the resolution.

•Absolute Encoders: Provide a unique position value for each shaft position within a single revolution, ensuring exact position tracking without needing a reference point.

Advantages

•Durability: They are robust and can operate in harsh environments (dust, moisture, etc.).

•Reliability: Non-contact operation reduces wear and tear.

•Versatility: Suitable for various applications, from industrial machinery to automotive systems.

Applications

•Motor control

•Robotics

•Industrial automation

•Automotive systems (e.g., steering angle sensors)

Encoder Interrupts:

Encoder interrupts are used to handle signals from encoders efficiently. When the encoder's sensor detects a change in position, it generates an interrupt signal. This signal triggers an interrupt service routine (ISR) in the system's microcontroller or processor. The ISR processes the encoder signal, updates the position counter, and performs any necessary calculations. Using interrupts allows the system to respond quickly to position changes without continuously polling the encoder, thus saving processing resources and increasing efficiency.

Half-Quad & Full-Quad

Half-quad and full-squad refer to different methods of decoding the output signals from incremental encoders to determine position.

•Half-Quad Decoding: In half-quad decoding, the system processes only one edge (rising or falling) of each pulse signal from the encoder. This method provides a basic level of resolution, counting one pulse per signal change.

•Full-Quad Decoding: Full-squad decoding processes both the rising and falling edges of each pulse signal, effectively doubling the resolution compared to half-quad decoding. This method counts two pulses per signal change, allowing for more precise position measurement.

Editing Encoder Pulse Counter Library for Getting RPM

To calculate the RPM (revolutions per minute) of a rotating shaft using an encoder, you need to modify the encoder pulse counter library. The following function calculates the RPM by measuring the number of encoder pulses over a specific time interval.

```
int AmrEncoder::getRPM() {
    long encoder_ticks = encoder_->getCount();

    // Get current time
    unsigned long current_time = millis();

    // Calculate delta time since last update
    unsigned long dt = current_time - prev_update_time_;

    // Convert time from milliseconds to
    minutesdouble dtm = (double)dt /
    60000;

    // Calculate delta ticks since last update
    double delta_ticks = encoder_ticks - prev_encoder_ticks_;

    // Calculate wheel's speed (RPM)

    // Update previous values for next
    calculationprev_update_time_ =
    current_time; prev_encoder_ticks_ =
    encoder_ticks;

    return (delta_ticks / counts_per_rev_) / dtm;
}
```

Here's a breakdown of how this function works:

•Get Encoder Ticks: The function starts by retrieving the current count of encoder pulses using encoder getCount().

•Get Current Time: It then gets the current time in milliseconds using millis().

•Calculate Delta Time: The time elapsed since the last update (dt) is calculated by subtracting prev update time from current time.

•Convert Time to Minutes: The elapsed time in milliseconds is converted to minutes by dividing dt by 60000.

•Calculate Delta Ticks: The function calculates the difference in encoder ticks since the last update (delta ticks) by subtracting prev encoder ticks from encoder ticks.

•Calculate RPM: The RPM is calculated by dividing the delta ticks by counts per rev(the number of encoder pulses per revolution) and then dividing by the elapsed time in minutes (dtm).

•Update Previous Values: Finally, the function updates prev update time and prev encoder ticks with the current values for use in the next calculation.

In summary, magnetic rotary encoders use a rotating magnet and a sen- sor to detect changes in the magnetic field, converting these changes into signals that represent the angular position of a shaft. Their non-contact nature and robustness make them ideal for a wide range of applications.

7.3 MPU9250 working principle

The MPU9250's working principle relies on the integration of three distinct types of sensors: a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis magnetometer. Each sensor provides crucial data for motion tracking and orientation determination.

7.3.1 Gyroscope

The gyroscope measures the angular velocity along the X, Y, and Z axes. It operates based on the Coriolis effect. When the device rotates, a vibrating structure inside the gyroscope experiences a force proportional to the angular velocity. This force causes a displacement that is detected by the sensor, converting it into an electrical signal proportional to the rate of rotation. The gyroscope is used to measure rotational motion and can detect changes in orientation.

7.3.2 Accelerometer

The accelerometer measures linear acceleration along the X, Y, and Z axes. It typically uses a microelectromechanical system (MEMS) structure com- posed of a small mass suspended by springs within a silicon frame. When the device accelerates, the mass moves due to inertia, causing a change in capacitance or resistance, depending on the design. This change is converted into an electrical signal, providing data on the linear acceleration of the device. The accelerometer is crucial for detecting motion, tilt, and gravitational force.

7.3.3 Magnetometer

The magnetometer measures the strength and direction of the magnetic field along the X, Y, and Z axes. It typically uses anisotropic magne-toresistive (AMR) technology, which changes resistance in response to the magnetic field. This change is measured and converted into an electrical signal, providing data on the magnetic field's direction and intensity. The magnetometer is essential for determining the device's heading relative to the Earth's magnetic field, acting as a digital compass.

7.3.4 Sensor Fusion

The MPU9250 includes a Digital Motion Processor (DMP) that performs sensor fusion algorithms. Sensor fusion combines data from the gyroscope, accelerometer, and magnetometer to provide a more accurate and stable estimate of the device's orientation and motion. The DMP processes the raw data from all three sensors, applying algorithms to filter out noise and correct for drift, resulting in precise measurements of orientation (pitch, roll, and yaw) and motion.

7.3.5 Interfaces and Data Output

The MPU9250 communicates with external devices via I2C or SPI inter- faces, allowing for easy integration into various systems. The processed data can be read from the device registers, providing real-time information on the device's motion and orientation. The sensor's high sensitivity and precision make it suitable for applications such as drone stabilization, robotic navigation, virtual reality, and motion capture systems.

subsection Using the Bolder Flight MPU9250 library to get measurements The Bolder Flight MPU9250 library provides a convenient way to

interface with the MPU9250 sensor and retrieve measurements. It abstracts away the complexities of communication protocols and sensor calibration, allowing users to focus on utilizing the sensor data for their specific applications. By utilizing this library, developers can easily integrate the MPU9250 into their projects and access reliable motion and orientation data with minimal effort.

To use the Bolder Flight MPU9250 library, you first need to include the library header file:

```
#include "mpu9250.h"
```

Next, you need to create an instance of the Mpu9250 class:

```
bfs::Mpu9250 imu;
```

In the setup() function, initialize the serial communication, start theI2C bus, configure the MPU9250 object, and initialize the sensor:

```
void setup() {

  /* Serial to display
  data */
  Serial.begin(115200);
  while(!Serial) {}

  /* Start the
  I2C bus */
  Wire.begin();
  Wire.setClock(
  400000);

  /* I2C bus,  0x68 address */

  imu.Config(&Wire,  bfs::Mpu9250::I2C_ADDR_PRIM);

  /* Initialize and
  configure IMU */if
  (!imu.Begin()) {

    Serial.println("Error initializing communication
    with IMU");while(1) {}

  }

  /* Set the sample
  rate divider */if
  (!imu.ConfigSrd(19))
  {

    Serial.println("Error
    configured SRD");while(1)
    {}

  }

}
```

In the loop() function, check if data is available from the sensor and then read and print the sensor data:

```
void loop() {

  /* Check if
  data read */
  if
  (imu.Read())
  {
```

```
Serial.print(imu.new_imu_data());
Serial.print("\t");
Serial.print(imu.new_mag_data());
Serial.print("\t");
```

```
Serial.print(imu.accel_x_mp
s2()); Serial.print("\t");
Serial.print(imu.accel_y_mp
s2()); Serial.print("\t");
Serial.print(imu.accel_z_mp
s2()); Serial.print("\t");
Serial.print(imu.gyro_x_rad
ps()); Serial.print("\t");
Serial.print(imu.gyro_y_rad
ps()); Serial.print("\t");
Serial.print(imu.gyro_z_ra
dps());Serial.print("\t");
Serial.print(imu.mag_x_u
t()); Serial.print("\t");
Serial.print(imu.mag_y_u
t()); Serial.print("\t");
Serial.print(imu.mag_z_u
t()); Serial.print("\t");
Serial.print(imu.die_temp
_c()); Serial.print("\n");
    }
  }
```

This code continuously reads data from the MPU9250 sensor and prints it to the serial monitor.

7.3.6 Summary

In essence, the MPU9250 works by measuring angular velocity, linear accel- eration, and magnetic field strength along three perpendicular axes using its gyroscope, accelerometer, and magnetometer. The integrated DMP performs complex computations to fuse this data, providing accurate and reliable information on the device's motion and orientation, which is cru- cial for a wide range of advanced applications.

7.4 Cytron MDD10A working principle and how to use it?

The Cytron MDD10A motor driver is a highly efficient and reliable device tailored for controlling DC motors in a wide range of applications. Its core working principle revolves around the use of an H-bridge circuit configuration. An H-bridge consists of four switching elements, typically transistors, arranged in a square with the motor placed in the center. By

controlling the state (on or off) of these transistors, the Cytron MDD10A can manipulate the direction and speed of the connected motor.

When the transistors on one side of the H-bridge are turned on while those on the opposite side are turned off, current flows from the power source through the motor in one direction, causing it to rotate clockwise. Conversely, when the transistors on the opposite side are activated, current flows in the opposite direction, leading to counterclockwise rotation. This bidirectional control capability makes the Cytron MDD10A versatile for various motor control applications.

Moreover, the Cytron MDD10A features Pulse Width Modulation (PWM) speed control. PWM works by rapidly switching the power supplied to the motor on and off at a varying duty cycle. By adjusting the duty cycle of the PWM signal, the average voltage applied to the motor can be controlled, thereby regulating its speed.This allows for precise speed adjustments and smooth acceleration and deceleration of the motor.

In addition to its fundamental motor control capabilities, the Cytron MDD10A is equipped with built-in protection features to safeguard both the motor and the driver itself. These protections include overcurrent and overheating protection mechanisms, which help prevent damage to the motor and the driver in case of excessive current draw or prolonged operation under high temperatures.

7.4.1 Creating library and how to use it

To utilize the Cytron MDD10A motor driver in Arduino projects, a C++ library can be created. Below is an example code snippet demonstrating how to create and use such a library:

This code snippet demonstrates the creation of a C++ class named

Controller, which encapsulates the functionality of the Cytron MDD10A motor driver. The class constructor initializes the necessary pins and con- figurations, while the spin method controls the motor’s speed and direc- tion based on the provided PWM value. This library can be included in Arduino sketches to easily control the Cytron MDD10A motor driver.

```
#include "AMR_Motor.h"
```

```
Controller::Controller( int pwm_pin, int motor_pinA, int motor_channel, int freq, int res):
pwm_pin_(pwm_pin),
motor_pinA_(motor_pinA), motor_channel_(motor_channel), freq_(freq),
res_(res)
{
pinMode(pwm_pin_, OUTPUT); pinMode(motor_pinA_, OUTPUT);
ledcSetup(motor_channel_ ,freq_ , res_);

ledcAttachPin(pwm_pin_,motor_channel_); ledcWrite(motor_channel_, abs(0));
}
void  Controller::spin(int  pwm)
{
if(pwm > 0)
{
digitalWrite(motor_pinA_, HIGH);
}
else if(pwm < 0)
{
digitalWrite(motor_pinA_,  LOW);
}
//digitalWrite(motor_pinA_,pwm      <  0);
//digitalWrite(motor_pinA_,pwm      > 0); ledcWrite(motor_channel_, abs(pwm));
}
```

7.4.2 AMRmotor Library

The provided code is part of the implementation of the ‘AMR Motor.h‘ library. This library likely provides functionalities for controlling motors used in the robot.

- Constructor:

– The Controller class constructor initializes the motor controller

object with the given parameters: pwm pin, motor pinA, motor channel, freq, and res.

– It sets up the PWM pin and the motor pin for output.

– ledcSetup function is used to configure the LEDC (LED Control) peripheral with the specified parameters such as frequency and resolution.

– ledcAttachPin is used to attach the PWM pin to the specified LEDC channel.

– Finally, ledcWrite is called to set the initial PWM value to 0.

- spin Function:

– This function is responsible for spinning the motor at a specified PWM value.

– Depending on the sign of the PWM value, the direction of rotation is set by toggling the motor pin (motor pinA) high or low.

– The absolute value of the PWM is written to the LEDC channel to control the motor speed.

– There are commented out lines using digitalWrite to set the motor direction directly based on the PWM value. However, these lines are not currently used in the code.

7.5 Linorobot Kinematics Library and Usage

The Linorobot Kinematics library is designed to assist with robotic motion calculations, particularly for differential-drive and four-wheel-drive robots.

7.5.1 Key Components

1.Initialization: The Kinematics class is initialized with parameters such as maximum motor RPM, wheel diameter, distances between wheels, and PWM resolution.

•Constructor:

Kinematics::Kinematics(int motor max rpm, float wheel diameter,

2.Forward Kinematics: The getRPM() function calculates the re- quired RPMs for each motor based on linear and angular velocities.

•Function:

Kinematics::output Kinematics::getRPM(float linear x, float line

3.PWM Conversion: The getPWM() function converts the calculated RPMs into PWM values.

•Function:

Kinematics::output Kinematics::getPWM(float linear x, float line

4.Inverse Kinematics: The getVelocities() functions calculate lin- ear and angular velocities based on motor RPMs.

•Functions:

Kinematics::velocities Kinematics::getVelocities(int motor1, in Kinematics::velocities Kinematics::getVelocities(int motor1, in

5.RPM to PWM Conversion: The rpmToPWM() function converts RPM values to PWM values.

•Function:

int Kinematics::rpmToPWM(int rpm)

7.5.2 Usage

1.Initialization:

Kinematics kinematics(motor max rpm, wheel diameter, fr wheels dist,

2.Forward Kinematics:

Kinematics::output rpm = kinematics.getRPM(linear x, linear y, angu

3.PWM Conversion:

Kinematics::output pwm = kinematics.getPWM(linear x, linear y, angu

4.Inverse Kinematics:

Kinematics::velocities vel = kinematics.getVelocities(motor1, motor

or

Kinematics::velocities vel = kinematics.getVelocities(motor1, motor

5.RPM to PWM Conversion:

int pwm value = kinematics.rpmToPWM(rpm value);

Ensure you provide appropriate inputs to these functions based on your robot's configuration and requirements.

This library abstracts the complex kinematics calculations, making it easier for you to focus on implementing control logic for your robot's mo- tion.

7.6 Rosserial node and robot base code

This code sets up a ROS node for controlling a robot's motion using ve- locity commands received over the 'cmd vel' topic. It reads IMU data and publishes it. Additionally, it provides functions for debugging and controlling the robot's motion based on the received commands.

```
#include "ros.h" #include "ros/time.h"

//header file for publishing velocities for odom #include "geometry_msgs/Pose.h"

//header file for cmd_subscribing to "cmd_vel" #include "geometry_msgs/Twist.h"

#include "sensor_msgs/Imu.h"


#include "AMR_Motor.h" #include "Kinematics.h" #include "AmrEncoder.h" #include "mpu9250.h"


#define AMR_BASE SKID_STEER #define DEBUG 1
```

```
//==================BIGGER   ROBOT   SPEC
(MDDA10)==============================

#define K_P 0.348607064985295 // P constant #define K_I 0.25750443968910 //  I constant
#define K_D 0.0 // D constant


// define your robot' specs here

#define MAX_RPM 83          // motor's maximum RPM

#define COUNTS_PER_REV 844     // wheel encoder's no of ticks per rev #define
WHEEL_DIAMETER 0.125           // wheel's diameter in meters

#define PWM_BITS 8          // PWM Resolution of  the  microcontroller #define
LR_WHEELS_DISTANCE 0.45       // distance between left and right  wheels #define
FR_WHEELS_DISTANCE  0.215              // distance between front and back wheels.

//Ignore this if you're on 2WD/ACKERMANN


#defineMOTOR1_ENCODER_A      5

#defineMOTOR1_ENCODER_B      23

#defineMOTOR2_ENCODER_A      12

#defineMOTOR2_ENCODER_B      13

#defineMOTOR3_ENCODER_A      16

#defineMOTOR3_ENCODER_B      17

#defineMOTOR4_ENCODER_A      19

#define MOTOR4_ENCODER_B 18


#define MOTOR1_PWM 27 // Changed to a valid PWM pin #define MOTOR1_IN_A 33

#define MOTOR1_CHANNEL 0

#define MOTOR2_PWM 26 // Changed to a valid PWM pin #define MOTOR2_IN_A 32
```

```
#define MOTOR2_CHANNEL 1

#define MOTOR3_PWM 14 // Changed to a valid PWM pin #define MOTOR3_IN_A 4

#define MOTOR3_CHANNEL 6

#define MOTOR4_PWM 25 // Changed to a valid PWM pin #define MOTOR4_IN_A 15

#define MOTOR4_CHANNEL 7

#define PWM_MAX (pow(2, PWM_BITS) - 1) #define PWM_MIN (-PWM_MAX)

#define FREQ  5000

#define  RES   8

bfs::Mpu9250 imu;

#define IMU_PUBLISH_RATE 20 //hz #define COMMAND_RATE 20 //hz #define
DEBUG_RATE 5


AmrEncoder motor1_encoder(MOTOR1_ENCODER_A, MOTOR1_ENCODER_B );
AmrEncoder motor2_encoder(MOTOR2_ENCODER_A, MOTOR2_ENCODER_B );
AmrEncoder motor3_encoder(MOTOR3_ENCODER_A, MOTOR3_ENCODER_B );
AmrEncoder motor4_encoder(MOTOR4_ENCODER_A, MOTOR4_ENCODER_B );


Controller motor1_controller(MOTOR1_PWM, MOTOR1_IN_A, MOTOR1_CHANNEL,
FREQ, RES); Controller motor2_controller(MOTOR2_PWM, MOTOR2_IN_A,
MOTOR2_CHANNEL, FREQ, RES); Controller motor3_controller(MOTOR3_PWM,
MOTOR3_IN_A, MOTOR3_CHANNEL, FREQ, RES); Controller
motor4_controller(MOTOR4_PWM, MOTOR4_IN_A, MOTOR4_CHANNEL, FREQ, RES);


Kinematics kinematics(MAX_RPM, WHEEL_DIAMETER, FR_WHEELS_DISTANCE,
LR_WHEELS_DISTANCE, PWM_BITS);

float g_req_linear_vel_x = 0; float g_req_linear_vel_y = 0; float g_req_angular_vel_z = 0;
```

```
unsigned long g_prev_command_time = 0;

void commandCallback(const  geometry_msgs::Twist&  cmd_msg);

ros::NodeHandle nh;


ros::Subscriber<geometry_msgs::Twist> cmd_sub("cmd_vel", commandCallback);
ros::Subscriber<control_msgs::PidState> pid_sub("pid", PIDCallback);


sensor_msgs::Imu imu_msg;

ros::Publisher imu_pub("imu",  &imu_msg);

geometry_msgs::Pose raw_vel_msg;

ros::Publisher raw_vel_pub("raw_vel", &raw_vel_msg);

void setup() {

nh.initNode(); nh.subscribe(pid_sub); nh.subscribe(cmd_sub); nh.advertise(raw_vel_pub);
nh.advertise(imu_pub); nh.loginfo("AMR CONNECTED");

Serial.begin(115200); while(!Serial) {}

/* Start the I2C bus */ Wire.begin(); Wire.setClock(400000);

/*  I2C  bus,    0x68 address */

imu.Config(&Wire,   bfs::Mpu9250::I2C_ADDR_PRIM);

/* Initialize and configure IMU */ if (!imu.Begin()) {

Serial.println("Error initializing communication with IMU"); while(1) {}

}

/* Set the sample rate divider */ if (!imu.ConfigSrd(19)) {

Serial.println("Error configured SRD"); while(1) {}

}

delay(5000);
```

```
}
void loop() {
static unsigned long prev_control_time = 0; static unsigned long prev_imu_time = 0; static unsigned long prev_debug_time = 0; static bool imu_is_initialized;
//this block drives the robot based on defined rate
if ((millis() - prev_control_time) >= (1000 / COMMAND_RATE))
{
moveBase();
prev_control_time = millis();
}
//this block stops the motor when no command is received if ((millis() - g_prev_command_time) >= 400)
{
stopBase();
}
//this block publishes the IMU data based on defined rate
if ((millis() - prev_imu_time) >= (1000 / IMU_PUBLISH_RATE))
{
//sanity check if the IMU is connected if (!imu_is_initialized)
{
imu_is_initialized = setupIMU();
if(imu_is_initialized) nh.loginfo("IMU Initialized");
else
nh.logfatal("IMU failed to initialize. Check your IMU connection.");
}
else
```

```
{
publishIMU();
}
prev_imu_time = millis();
}
//this block displays the encoder readings.
//change DEBUG to 0 if you don’t want to display if(DEBUG)
{
if ((millis() - prev_debug_time) >= (1000 / DEBUG_RATE))
{
printDebug(); prev_debug_time = millis();
}
}
//call all the callbacks waiting to be called nh.spinOnce();
}
void commandCallback(const geometry_msgs::Twist& cmd_msg)
{
//callback function  every  time
//linear and angular speed is received from ’cmd_vel’ topic
//this callback function
//receives cmd_msg object where linear and angular speed are stored g_req_linear_vel_x =
cmd_msg.linear.x;
g_req_linear_vel_y = cmd_msg.linear.y; g_req_angular_vel_z = cmd_msg.angular.z;
g_prev_command_time = millis();
}
```

```
void moveBase()
{
//get the required rpm for each motor based on required velocities, and base used
Kinematics::output req_pwm =
kinematics.getPWM(g_req_linear_vel_x, g_req_linear_vel_y, g_req_angular_vel_z);
//get the current speed of each motor
int current_rpm1 = motor1_encoder.getRPM(); int current_rpm2 = motor2_encoder.getRPM();
int current_rpm3 = motor3_encoder.getRPM(); int current_rpm4 = motor4_encoder.getRPM();


//the required rpm is capped
at -/+ MAX_RPM to prevent the PID from having too much error
//the PWM value sent to the motor driver is the
//calculated PID based on required RPM vs measured RPM
motor1_controller.spin(req_pwm.motor1); motor2_controller.spin(req_pwm.motor2);
motor3_controller.spin(req_pwm.motor3); motor4_controller.spin(req_pwm.motor4);
motor1_controller.spin(MAX_RPM); motor2_controller.spin(MAX_RPM);
motor3_controller.spin(MAX_RPM); motor4_controller.spin(MAX_RPM);
Kinematics::velocities current_vel;
current_vel = kinematics.getVelocities(current_rpm1, current_rpm2, current_rpm3,
current_rpm4);
        //pass velocities to publisher object
        raw_vel_msg.position.x =
        current_vel.linear_x;
        raw_vel_msg.position.y =
        current_vel.linear_y;
        raw_vel_msg.orientation.z =
        current_vel.angular_z;


        //publish raw_vel_msg
        raw_vel_pub.publish(&raw_vel_msg);

  }
```

```
void stopBase()
{
    g_req_linear_vel_x = 0;
    g_req_linear_vel_y = 0;
    g_req_angular_vel_z = 0;
}

void publishIMU()
{
    if( imu.Read())
      {
       imu.new
       _imu_da
       ta();
       imu.new
       _mag_da
       ta();
      pass accelerometer data to imu object
      imu_msg.linear_acceleration.x =
      imu.accel_x_mps2();
      imu_msg.linear_acceleration.y =
      imu.accel_y_mps2();
      imu_msg.linear_acceleration.z =
      imu.accel_z_mps2();
     //pass gyroscope data to imu object
      imu_msg.angular_velocity.x =
      imu.gyro_x_radps();
      imu_msg.angular_velocity.y =
      imu.gyro_y_radps();
      imu_msg.angular_velocity.z =
      imu.gyro_z_radps();

      //pass accelerometer data to
      imu object
      imu_msg.orientation.x =
      imu.mag_x_ut();
      imu_msg.orientation.y =
      imu.mag_y_ut();
      imu_msg.orientation.z =
      imu.mag_z_ut();

      //publish raw_imu_msg
      imu_pub.publish(&imu_msg);
```

```
        }
    }


void printDebug()
{
    char buffer[50];

    sprintf (buffer, "Encoder FrontLeft  : %ld",
    motor1_encoder.getRPM());nh.loginfo(buffer);
    sprintf (buffer, "Encoder FrontRight : %ld", motor2_encoder.getRPM());
     nh.loginfo(buffer);
     sprintf (buffer, "Encoder RearLeft       : %ld", motor3_encoder.getRPM()); nh.loginfo(buffer);
     sprintf (buffer, "Encoder RearRight      : %ld", motor4_encoder.getRPM()); nh.loginfo(buffer);
     }
```

7.6.1 Topics

•cmd vel: This topic is used to send velocity commands to the robot. Subscribers to this topic parse incoming Twist messages to determine the desired linear and angular velocities of the robot.

•pid: This topic is not defined in the provided code snippet. It is likely used for receiving PID control parameters.

•imu: IMU data is published on this topic. It includes accelerometer, gyroscope, and magnetometer readings.

•raw vel: This topic publishes the raw velocities of the robot.

7.6.2 Header Files

- #include "ros.h" and #include "ros/time.h": ROS-specific header files for essential ROS functionalities.
- Other header files like geometry msgs/Pose.h, geometry msgs/Twist.h, and sensor msgs/Imu.h are for message types defined in ROS.

7.6.3 Definitions and Constants

- Constants like PID constants (K P, K I, K D), physical properties of the robot (MAX RPM, WHEEL DIAMETER, etc.), and pin configurations for motors and encoders are defined.

7.6.4 Object Instantiation

– Objects for motor controllers, encoders, kinematics, and an MPU9250 sensor are instantiated. These objects are utilized for controlling the robot's motion and obtaining sensor data.

7.6.5 Global Variables

–Variables like g req linear vel x, g req linear vel y, g req angular vel

are declared to store the required linear and angular velocities of the robot.

7.6.6 ROS Node Setup

– ros::NodeHandle nh; initializes a ROS node. Subscribers (cmd sub and pid sub) and publishers (imu pub and raw vel pub) are ini- tialized.

7.6.7 Setup Function

– setup() function initializes the ROS node, sets up serial commu- nication, initializes the MPU9250 sensor, and configures it.

7.6.8 Loop Function

loop() function continuously runs the main logic, checking for new commands, publishing IMU data, displaying debug informa- tion, and calling callback functions. It uses nh.spinOnce() to handle ROS callbacks.

7.6.9 Callback Functions

– commandCallback(const geometry msgs::Twist& cmd msg): This function is called whenever a new velocity command is received

on the 'cmd vel' topic. It updates the global variables with the new velocity commands. There is another callback function PIDCallback, which is not defined in the provided code snippet.

7.6.10 Motion Control Functions

– moveBase(): Calculates the required PWM for each motor based on the desired velocities and controls the motors accordingly. It also publishes the current velocities of the robot.

– stopBase(): Stops the robot when no command is received for a certain duration.

7.6.11 IMU Data Publishing Function

– publishIMU(): Reads data from the MPU9250 sensor and pub- lishes it as IMU data. This function is called periodically to publish IMU data at a defined rate.

7.6.12 Debug Function

– printDebug(): Prints encoder readings for debugging purposes.

7.7 Robot Host

According to the previous sprint, we have done a lot. But till now we still have 2 problems:

• Uncalibrated IMU message: Bolder flight library supported us with measurement but didn't provide us with calibration.

• Insensitive Odometry due to depending on encoders only: After several tries of depending on encoders only for odometry, we found that it is almost impossible because of slipping, which is not detected by encoders.

7.7.1 Published Topics Integration

Imu calib

We searched for hours about achieving IMU calibration but we found that it is complicated mathematical equations and it would take a lot of time to achieve. Moreover, those equations change according to your application. So we tried to find someone who has a similar application and has faced this problem too. We were lucky because we found a developer on GitHub called Dan Koch who has created a ROS package for a similar task called Imu calib3.

Imu calib Nodes: Imu calib has two nodes:

• do calib: do calib is responsible for performing some steps in se- quence and finally generating a .yaml file which contains the needed calibration values.

• apply calib: apply calib is responsible for using the noisy Imu msg and the calibration values to stream a calibrated Imu msg.

robot localization

It was our second magical solution, which we used to improve odometry by using both raw odom and calibrated Imu msg to generate more accurate odometry.

Usage of Kalman Filter in the robot localization ROS Pack- age

The Kalman filter is a powerful mathematical tool used in the robot localization

ROS (Robot Operating System) package to provide accurate state esti- mation for mobile robots. This package utilizes different versions of the Kalman filter,

including the Extended Kalman Filter (EKF) and the Un- scented Kalman Filter (UKF), to fuse sensor data from multiple sources, such as IMUs (Inertial Measurement Units), GPS, and wheel encoders.

• State Estimation: The Kalman filter helps estimate the state of the robot, which includes its position, velocity, and orientation, even

3For more information about Imu calib, visit Github official repository.

when the measurements are noisy. It provides a best guess of the robot's state by predicting future states and updating the predictions with actual measurements.

• Sensor Fusion: The robot localization package uses the Kalman filter to integrate data from multiple sensors. Each sensor may pro- vide different types of information with varying accuracy. The filter combines these inputs to improve the overall state estimate.

• Error Minimization: By maintaining a probabilistic model of the state and measurement uncertainties, the Kalman filter continuously corrects the state estimates. It minimizes the estimation error by adjusting the predicted state with new measurements.

• Handling Nonlinearities: The EKF and UKF versions are specif- ically designed to handle the nonlinearities present in robot motion and sensor models. EKF linearizes the system around the current estimate, while UKF uses a set of sample points to better capture the nonlinear transformations.

• Configuration: The robot localization package allows extensive configuration through ROS parameters. Users can specify which sen- sors to use, how to handle different types of measurements, and how to set the initial state and noise characteristics.

In essence, the Kalman filter in the robot localization package en- hances the accuracy and reliability of the robot's state estimation, which is crucial for navigation and control tasks in various robotics applications.

7.8 Mapping & Navigation

Now we could say that we are ready to score our goal and reach the end of our task. Because we almost have all topics needed for mapping and navigation.

• 7.8.1 Mapping Preparation

: Mapping has several algorithms and techniques we have talked about before. But the common factor of 90% of them their need to laser msg which is provided from lidar.

Lidar ros Package Introduction

This package provides basic device handling for 2D Laser Scanner RPLIDAR A1/A2 and A3.

RPLIDAR is a low cost LIDAR sensor suitable for indoor robotic SLAM application. It provides 360 degree scan field, 5.5hz/10hz ro- tating frequency with guaranteed 8 meter ranger distance, current more than 16m for A2 and 25m for A3 . By means of the high speed image processing engine designed by RoboPeak, the whole cost are reduced greatly, RPLIDAR is the ideal sensor in cost sensitive areas like robots consumer and hardware hobbyist.

RPLIDAR A3 performs high speed distance measurement with more than 16K samples per second,RPLIDAR A2 performs high speed dis- tance measurement with more than 4K/8K samples per second,RPLIDAR A1 supports 2K/4K samples per second. For a scan requires 360 sam- ples per rotation, the 10hz scanning frequency can be achieved. Users can customized the scanning frequency from 2hz to 10hz freely by control the speed of the scanning motor. RPLIDAR will self-adapt the current scanning speed.

The driver publishes device-dependent sensor msgs/LaserScan data.4

Launching rplidar ros package

The provided launch file is an XML configuration file for ROS (Robot Operating System) that is used to start up a node associated with the rplidar ros package. Here is a breakdown of each part of the launch file:

<launch>

<node name="rplidar_node" pkg="rplidar_ros" type="rplidarNode" output="screen">

<param name="serial_port" type="string" value="/dev/ttyUSB2"/>

<param name="serial_baudrate" type="int" value="115200"/>

<param name="frame_id" type="string" value="laser"/>

<param name="inverted" type="bool" value="false"/>

<param name="angle_compensate" type="bool" value="true"/>

</node>

</launch>

4For more Information visit: ros wiki RPlidar ros

Explanation

– Launch Tag (<launch> ... </launch>): This is the root element of a ROS launch file. It encompasses all the nodes and parameters that need to be launched.

– Node Tag (<node name="rplidar node" pkg="rplidar ros" type="rplidarNode" output="screen">):

∗ name: Specifies the name of the node. In this case, the node is named rplidar node.

∗ pkg: Indicates the ROS package where the node executable is located. Here, it is rplidar ros.

∗ type: The type of the node, which usually corresponds to the executable file within the package. For this node, it is rplidarNode.

∗ output: Specifies where the node's output should be sent. Setting this to screen means that the output will be displayed on the console screen.

– Param Tags: These tags define parameters that are passed to the node when it is launched. Each parameter has a name, type, and value.

∗ <param name="serial port" type="string" value="/dev/ttyUSB2"/

· name: serial port

· type: string

· value: /dev/ttyUSB2

· This parameter specifies the serial port that the RPLIDAR is connected to. Here, it is set to /dev/ttyUSB2.

∗ <param name="serial baudrate" type="int" value="115200"/>

· name: serial baudrate

· type: int

· value: 115200

· This parameter sets the baud rate for the serial communi- cation. Here, it is set to 115200.

∗ <param name="frame id" type="string" value="laser"/>

· name: frame id

· type: string

· value: laser

· This parameter defines the frame ID for the laser scans. Here, it is set to laser.

∗ <param name="inverted" type="bool" value="false"/>

· name: inverted

· type: bool

· value: false

· This boolean parameter specifies whether the RPLIDAR data is inverted. Here, it is set to false.

∗ <param name="angle compensate" type="bool" value="true"/>

· name: angle compensate

· type: bool

· value: true

· This boolean parameter indicates whether angle compen- sation should be applied to the RPLIDAR data. Here, it is set to true.

Summary The launch file starts a node called rplidar node from the

rplidar ros package. It configures the node with several parameters:

– The serial port is set to /dev/ttyUSB2.

– The baud rate is set to 115200.

– The frame ID for the laser is laser.

– The data is not inverted (false).

– Angle compensation is enabled (true).

These parameters are essential for the proper communication and functioning of the RPLIDAR device in the ROS environment.

• 7.8.2 Cartographer

Cartographer is a system that provides real-time simultaneous local- ization and mapping (SLAM) in 2D and 3D across multiple platforms and sensor configurations.

Terminology

This documents a few common patterns that exist in the Cartographer codebase.
Frames

global map frame

This is the frame in which global SLAM results are expressed. It is the fixed map frame including all loop closure and optimization results. The transform between this frame and any other frame can jump when new optimization results are available. Its z-axis points upwards, i.e. the gravitational acceleration vector points in the -z direction, i.e. the gravitational component measured by an accelerometer is in the +z direction.

local map frame

This is the frame in which local SLAM results are expressed. It is the fixed map frame excluding loop closures and the pose graph opti- mization. For a given point in time, the transform between this and the global map frame may change, but the transform between this and all other frames does not change.

submap frame

Each submap has a separate fixed frame.

tracking frame

The frame in which sensor data is expressed. It is not fixed, i.e. it changes over time. It is also different for different trajectories.

gravity-aligned frame

Only used in 2D. A frame colocated with the tracking frame but with a different orientation that is approximately aligned with gravity, i.e. the gravitational acceleration vector points approximately in the - z direction. No assumption about yaw (rotation around the z axis between this and the tracking frame) should be made. A different gravity-aligned frame is used for different trajectory nodes, e.g. yaw can change arbitrarily between gravity-aligned frames of consecutive nodes.

Transforms

local pose Transforms data from the tracking frame (or a submap frame, depending on context) to the local map frame. global pose Transforms data from the tracking frame (or a submap frame, depend- ing on context) to the global map frame. local submap pose Trans-

forms data from a submap frame to the local map frame. global submap pose Transforms data from a submap frame to the global map frame.5

Launch Mapping

The provided launch file is an XML configuration file for ROS (Robot Operating System) used to start various nodes and set parameters for mapping purposes. Here is a breakdown of each part of the launch file:

<launch>

<!-- Load robot description and start state publisher-->

<param name="robot_description" textfile="$(find gbot_core)/urdf/amr.urdf"

<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot

<!-- Start Google Cartographer node with custom configuration file-->

<node name="cartographer_node" pkg="cartographer_ros" type="cartographer_n

-configuration_directory

$(find gbot_core)/configuration_files

-configuration_basename gbot_lidar_2d.lua" output="screen">

</node>

<!-- Additional node which converts Cartographer map into ROS occupancy gr Not used and can be skipped in this case -->

<node name="cartographer_occupancy_grid_node" pkg="cartographer_ros" type="cartographer_occupancy_grid_node" args="-resolution 0.05" />

</launch>

Explanation

– Launch Tag (<launch> ... </launch>): This is the root element of a ROS launch file. It encompasses all the nodes and parameters that need to be launched.

– Param Tag (

5For more information visit: ros wiki Cartographer documentation

<param name="robot_description" textfile="$(fi

):

∗ name: robot description

∗ textfile: Specifies the path to the URDF file containing the

robot description. Here, it is set to $(find gbot core)/urdf/amr.urdf.

– Node Tag for State Publisher (<node name="robot state publisher" pkg="robot state publisher" type="robot state publisher"

/>):

∗ name: Specifies the name of the node. In this case, the node is named robot state publisher.

∗ pkg: Indicates the ROS package where the node executable is located. Here, it is robot state publisher.

∗ type: The type of the node, which usually corresponds to the executable file within the package. For this node, it is robot state publisher.

– Node Tag for Cartographer (<node name="cartographer node" pkg="cartographer ros" type="cartographer node" args="-configura

$(find gbot core)/configuration files -configuration basename gbot lidar 2d.lua" output="screen">):

∗ name: Specifies the name of the node. In this case, the node is named cartographer node.

∗ pkg: Indicates the ROS package where the node executable is located. Here, it is cartographer ros.

∗ type: The type of the node, which usually corresponds to the executable file within the package. For this node, it is cartographer node.

∗ args: Specifies the arguments passed to the node. Here, it includes:

· -configuration directory: The directory containing the configuration files. It is set to $(find gbot core)/configuration fi

· -configuration basename: The base name of the con- figuration file. It is set to gbot lidar 2d.lua.

∗ output: Specifies where the node's output should be sent. Setting this to screen means that the output will be displayed on the console screen.

– Node Tag for Occupancy Grid (<node name="cartographer occupancy pkg="cartographer ros" type="cartographer occupancy grid node" args="-resolution 0.05" />):

∗ name: Specifies the name of the node. In this case, the node is named cartographer occupancy grid node.

∗ pkg: Indicates the ROS package where the node executable is located. Here, it is cartographer ros.

∗ type: The type of the node, which usually corresponds to the executable file within the package. For this node, it is cartographer occupancy grid node.

∗ args: Specifies the arguments passed to the node. Here, it sets the resolution to 0.05.

Summary The launch file sets up the following:

– Loads the robot description from a URDF file located at $(find gbot core)/urdf/amr.urdf.

– Starts the robot state publisher node to publish the state of the robot.

– Launches the cartographer node from the cartographer ros

package with a custom configuration file located in $(find gbot core)/conf

and named gbot lidar 2d.lua.

– (Optional) Starts the cartographer occupancy grid node to con- vert the Cartographer map into a ROS occupancy grid map with a resolution of 0.05.

• 7.8.3 AMCL Navigation

amcl is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.

Algorithms

Many of the algorithms and their parameters are well-described in the book Probabilistic Robotics, by Thrun, Burgard, and Fox. The user

is advised to check there for more detail. In particular, we use the fol- lowing algorithms from that book: sample motion model odometry,

beam range finder model, likelihood field range finder model, Augmented MCL, and KLD Sampling MCL.

As currently implemented, this node works only with laser scans and laser maps. It could be extended to work with other sensor data.

Nodes

amcl: amcl takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates. On startup, amcl initializes its particle filter according to the parameters provided. Note that, because of the defaults, if no parameters are set, the initial filter state will be a moderately sized particle cloud centered about (0,0,0).

• Subscribed Topics scan (sensor msgs/LaserScan) Laser scans.

tf (tf/tfMessage) Transforms.

initialpose (geometry msgs/PoseWithCovarianceStamped)

Mean and covariance with which to (re-)initialize the particle filter. map (nav msgs/OccupancyGrid)

When the use map topic parameter is set, AMCL subscribes to this topic to retrieve the map used for laser-based localization. New in navigation 1.4.2.

• Published Topics amcl pose (geometry msgs/PoseWithCovarianceStamped)

Robot's estimated pose in the map, with covariance. particlecloud (geometry msgs/PoseArray)

The set of pose estimates being maintained by the filter. tf (tf/tfMessage)

Publishes the transform from odom (which can be remapped via the odom frame id parameter) to map.

Launching AMCL

This XML code is a launch file for the Adaptive Monte Carlo Local- ization (AMCL) package in ROS (Robot Operating System). Let's break it down:

<launch>

<node pkg="amcl" type="amcl" name="amcl" output="screen">

<!-- Change this if you want to change your base frame id. -->

<param name="base_frame_id" value="base_footprint"/>

<!-- Maximum rate (Hz) at which scans and

paths are published for visualization, -1.0 to disable. -->

<param name="gui_publish_rate" value="10.0"/>

<param name="kld_err" value="0.05"/>

<param name="kld_z" value="0.99"/>

<param name="laser_lambda_short" value="0.1"/>

<param name="laser_likelihood_max_dist" value="2.0"/>

<param name="laser_max_beams" value="60"/>

<param name="laser_model_type" value="likelihood_field"/>

<param name="laser_sigma_hit" value="0.2"/>

<param name="laser_z_hit" value="0.5"/>

<param name="laser_z_short" value="0.05"/>

<param name="laser_z_max" value="0.05"/>

<param name="laser_z_rand"   value="0.5"/>

<param name="max_particles" value="2000"/>

<param name="min_particles" value="500"/>

<!-- Specifies the expected noise in odometry's rotation

estimate from the rotational component of the robot's motion. -->

<param name="odom_alpha1" value="0.25"/>

<!-- Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion. -->

<param name="odom_alpha2" value="0.25"/>

<!-- Specifies the expected noise in odometry's translation

estimate from the translational component of the robot's motion. -->

<param name="odom_alpha3" value="0.25"/>

<!-- Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion. -->

<param name="odom_alpha4" value="0.25"/>

<!-- Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion. -->

<param name="odom_alpha5" value="0.1"/>

<param name="odom_frame_id" value="odom"/>

<param   name="odom_model_type"   value="diff"/>

<!-- Exponential decay rate for the slow average weight filter, used in deciding when to recover by adding random poses. -->

<param name="recovery_alpha_slow" value="0.001"/>

<!-- Exponential decay rate for the fast average weight filter, used in deciding when to recover by adding random poses. -->

<param name="recovery_alpha_fast" value="0.1"/>

<!-- Number of filter updates required before resampling. -->

<param name="resample_interval" value="1"/>

<!-- Default 0.1; time with which to post-date the transform that is published to indicate that this transform is valid into the future. -->

<param name="transform_tolerance" value="1.25"/>

<!-- Rotational movement required before

performing a filter update. 0.1 represents 5.7 degrees -->

<param name="update_min_a" value="0.2"/>

<!-- Translational movement required before performing a filter update. -->

<param name="update_min_d" value="0.2"/>

</node>

</launch>

– base frame id: This parameter specifies the frame ID of the robot's base. The AMCL algorithm uses this frame as the refer- ence for localization.

– gui publish rate: This sets the rate at which scans and paths are published for visualization purposes in the graphical user in- terface (GUI).

– kld err and kld z: These parameters are used in the KLD sam- pling algorithm, which is employed by AMCL for particle resam- pling.

– laser lambda short: This parameter is related to the sensor model and affects the likelihood of short-range laser measure- ments.

– laser likelihood max dist: Specifies the maximum distance at which laser measurements are considered reliable for the sensor model.

– laser max beams: Sets the maximum number of laser beams to be used for localization.

– laser model type: Determines the type of laser sensor model used by AMCL. Here, it's set to use the likelihood field model.

– Parameters starting with laser sigma hit to laser z rand: These parameters fine-tune the sensor model, specifying characteristics like the standard deviation of hit probability, the probability of unexpected measurements, etc.

– max particles and min particles: These parameters specify the maximum and minimum number of particles used in the par- ticle filter algorithm, which AMCL employs for localization.

– Parameters odom alpha1 to odom alpha5: These parameters de- fine the expected noise characteristics of the odometry readings.

– odom frame id: Specifies the frame ID of the odometry data.

– odom model type: Defines the type of odometry model used by AMCL. Here, it's set to use a differential drive model.

– Parameters recovery alpha slow and recovery alpha fast: These parameters control the rate of decay for the average weight filter used in recovery behavior.

– resample interval: Specifies the number of filter updates re- quired before resampling the particles in the particle filter.

– transform tolerance: This parameter specifies the tolerance in seconds with which to post-date the transform that is published.

– Parameters update min a and update min d: These parameters define the minimum rotational and translational movements, re- spectively, required before performing a filter update.

In summary, each parameter in the AMCL launch file plays a cru- cial role in configuring the behavior and performance of the AMCL localization algorithm within the ROS environment, fine-tuning vari- ous aspects such as sensor models, odometry characteristics, particle filter settings, and recovery behavior.

End Effector:

Our end effector is the ball screw system. This code enables the control of a motor using Bluetooth commands and an encoder to track its rotation. It utilizes the `SoftwareSerial` library to manage Bluetooth communication on digital pins 6 and 7, while the motor is controlled using pins 10, 11, and 12. The encoder, connected to pins 2 and 3, has 1400 pulses per revolution and is used to track the motor's rotation.

In the `setup` function, pin modes are configured, serial communication is initialized, and interrupts are attached to the encoder pins to call the `updateEncoder` function whenever the encoder state changes. The initial motor state is set to stopped, and instructions are printed to the Serial Monitor.

In the `loop` function, the code continuously checks for Bluetooth commands. When a command is received, it is trimmed of any whitespace and compared to either "up" or "down". Depending on the command, the motor is instructed to rotate clockwise or counterclockwise by calling the `handleRotation` function.

The `handleRotation` function determines the target number of revolutions based on the total pulses counted so far and converts this to target pulses. It then calls `rotateMotor`, which sets the motor direction and starts the motor. The motor continues to run until the target pulse count is reached, at which point it stops.

The `updateEncoder` function, an interrupt service routine, updates the encoder count by reading the encoder pins and determining the direction of rotation. This ensures that the motor stops precisely when the target number of pulses is reached, allowing for accurate control of the motor's rotation based on the Bluetooth commands received.

```
#include <SoftwareSerial.h>

// Define pin numbers
const int enaPin = 10;
const int in1Pin = 11;
const int in2Pin = 12;
const int encoderPinA = 3;
const int encoderPinB = 2;

// Bluetooth module pins
const int bluetoothTx = 6; // TX pin of Bluetooth module connected to Arduino's
digital pin 6
```

```
const int bluetoothRx = 7; // RX pin of Bluetooth module connected to Arduino's
digital pin 7

// Motor parameters
const int motorSpeed = 255; // Maximum speed

// Encoder parameters
const int pulsesPerRevolution = 1400; // Change this to match your encoder

volatile long encoderCount = 0; // To store the encoder count
volatile int lastEncoded = 0; // Last encoder state
long totalPulses = 0; // To store the total pulses

SoftwareSerial bluetooth(bluetoothTx, bluetoothRx);

void setup() {
  // Set pin modes
  pinMode(enaPin, OUTPUT);
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);
  pinMode(encoderPinA, INPUT_PULLUP);
  pinMode(encoderPinB, INPUT_PULLUP);

  // Initialize Serial Monitor
  Serial.begin(9600);

  // Initialize Bluetooth communication
  bluetooth.begin(9600);

  // Attach interrupts to encoder pins
  attachInterrupt(digitalPinToInterrupt(encoderPinA), updateEncoder, CHANGE);
  attachInterrupt(digitalPinToInterrupt(encoderPinB), updateEncoder, CHANGE);

  // Initial motor state
  digitalWrite(in1Pin, LOW);
  digitalWrite(in2Pin, LOW);
  analogWrite(enaPin, 0);

  Serial.println("Bluetooth Motor Control");
  Serial.println("Send 'up' to rotate clockwise or 'down' to rotate
counterclockwise.");
}

void loop() {
  if (bluetooth.available()) {
    String command = bluetooth.readStringUntil('\n');
    command.trim(); // Remove any trailing whitespace
```

```
    if (command.equalsIgnoreCase("up")) {
      Serial.println("Starting clockwise rotation...");
      handleRotation(true);
    } else if (command.equalsIgnoreCase("down")) {
      Serial.println("Starting counterclockwise rotation...");
      handleRotation(false);
    } else {
      Serial.println("Invalid command. Send 'up' or 'down'.");
    }
  }
}

// Function to handle motor rotation logic based on total pulses
void handleRotation(bool clockwise) {
  int targetRevolutions;
  if (totalPulses >= pulsesPerRevolution) {
    targetRevolutions = clockwise ? 4 : 6;
  } else if (totalPulses <= -pulsesPerRevolution) {
    targetRevolutions = clockwise ? 6 : 4;
  } else {
    targetRevolutions = 5; // Default rotation if totalPulses is within +/- 1400
  }

  int targetPulses = targetRevolutions * pulsesPerRevolution;
  rotateMotor(clockwise, targetPulses);

  // Update the total pulses after rotation
  totalPulses += clockwise ? targetPulses : -targetPulses;

  // Display total pulses on Serial Monitor
  Serial.print("Total Pulses: ");
  Serial.println(totalPulses);
}

// Function to rotate the motor
void rotateMotor(bool clockwise, int pulses) {
  // Reset encoder count
  encoderCount = 0;

  if (clockwise) {
    digitalWrite(in1Pin, LOW);
    digitalWrite(in2Pin, HIGH);
  } else {
    digitalWrite(in1Pin, HIGH);
    digitalWrite(in2Pin, LOW);
  }

  analogWrite(enaPin, motorSpeed);
```

```
  // Wait until the target pulses are reached
  while (abs(encoderCount) < pulses) {
    Serial.print("Steps: ");
    Serial.println(encoderCount);
    delay(100); // Add a short delay to allow serial output to be readable
  }

  // Stop the motor
  analogWrite(enaPin, 0);
  digitalWrite(in1Pin, LOW);
  digitalWrite(in2Pin, LOW);

  Serial.println("Target reached. Motor stopped.");
}

// Interrupt service routine to update encoder count
void updateEncoder() {
  int MSB = digitalRead(encoderPinA); // MSB = most significant bit
  int LSB = digitalRead(encoderPinB); // LSB = least significant bit

  int encoded = (MSB << 1) | LSB; // Converting the 2 pin value to single number
  int sum = (lastEncoded << 2) | encoded; // Shifting previous encoded value to
the left and adding current

  if (sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) {
    encoderCount++;
  } else if (sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) {
    encoderCount--;
  }

  lastEncoded = encoded; // Store this value for next time
```

List of Symbol

Symbol	Meaning
AMR	Autonomous mobile robot in warehouse
AGV	Automated Guided Vehicle
F	$Load$
N	$No. of\ revolution$
R	$radius\ of\ the\ rim$
σ_b	allowable stress
τ	Allowable shear
T	$Torque$
K_m	Combined shock and fatigue factor for bending.
K_t	Combined shock and fatigue factor for torsion.
M	$Bending\ moment$
T_e	$equivalent\ twisting\ moment$
M_e	$equivalent\ bending\ moment$
K_a	surface condition modification factor
K_b	size modification factor
K_c	load modification factor
K_d	temperature modification factor
K_e	reliability factor
K_f	miscellaneous − effects modification factor
S'_e	rotary − beam test specimen endurance limit
S_e	endurance limit at the critical location of a machine part in the geometry and condition of use
S_{ut}	$Minimum\ tensile\ strength$
$\mathcal{F}$	Fatigue strength fraction
S_f	fatigue strength
L_D	Desired life in revolutions
L_R	Rating life in revolutions = 10^6
l	life , hours
X_D	dimensionless multiple of rating life
R	Reliability
a_f	$application\ factor$
μ_s	Static Coefficient of friction
μ_k	Kinetic Coefficient of friction
R_T	$raduis\ of\ tire = 0.065m$
T_{Fs}	$Torque\ at\ static$ friction
T_{Fk}	$Torque\ at$ Kinetic friction
F_r	$Force\ to\ move\ object\ at\ constant\ speed$
F_{Ms}	$Max\ friction\ force\ at\ static$
F_{Mk}	$Max\ friction\ force\ at\ Kinetic$
a	Acceleration
m	$mass\ of\ the\ car$
m_e	$car\ Wieght\ no\ load$
F_{cr}	$climping\ and\ downgrade\ force$
t_d	$deceleration\ time$

t_a	$acceleration\ time$
s_a	$Displacement\ covered\ during\ acceleration$
s_d	$displacment\ covered\ during\ deceleration$
s_m	$distance\ covered\ during\ motion$
$Js_{no\ load}$	$inertia\ of\ the\ system\ at\ no\ load$
Js_{load}	$inertia\ of\ the\ system\ at\ full\ load$
$Tacc$	$acceleration\ torque$
T_{total}	$Total\ torque$
a	$Angular\ acceleration$
I_{xx}	$component\ of\ inertia\ in\ x\ axis$
I_{yy}	$component\ of\ inertia\ in\ y\ axis$
I_{zz}	$component\ of\ inertia\ in\ z\ axis$
h	$height$
d	$length$
w	$width$
IMU	Inertial measurement unit
BMS	Battery management system
PWM	Pulse width modulation
V	$voltage$
I	$current$
ROS	Robot operating system
SLAM	Simulation Localization and mapping
RGB	Red –green – blue channel
RAM	Random Access Memory
LTS	long-time support
PCB	Printed Circuit Board

References:

VDI 2206. primary reference for design as mechatronics engineers. *https://www.sciencedirect.com/science/article/pii/S1474667017340351?via%3Dihub*

Articulated Robotics (Director). (2022). Creating a rough 3D model of our robot with URDF. *https://www.youtube.com/watch?v=BcjHyhV0kIs*

Bowling, J. (2022, September 28). How To Add A Motor Controller To Your ROS Robot. Exploring ROS Robotics. *https://medium.com/exploring-ros-robotics/how-to-add-a-motor-controller-to-your-ros-robot-fd17352cd5e3*

Budynas, R. G., & Nisbett, J. K. (2011). *Shigley's mechanical engineering design* (9. ed). McGraw-Hill.

Create URDF File and Model of Four-Wheeled Mobile Robot in ROS - ROS Robotics Tutorial—YouTube. (n.d.). Retrieved December 26, 2023, from https://www.youtube.com/watch?v=Eie-KoMQtxs&t=672s&ab_channel=AleksandarHaber

ed. (2023, May 5). ROS2 Nav2—Generate a Map with slam_toolbox. *The Robotics Back-End*. https://roboticsbackend.com/ros2-nav2-generate-a-map-with-slam_toolbox/

Electric Motor Power Measurement and Analysis | Yokogawa Electric Corporation. (n.d.). Retrieved December 26, 2023, from https://www.yokogawa.com/library/resources/media-publications/electric-motor-power-measurement-and-analysis/

Energy, N. C., & d6admin, teddy. (2020, August 10). *How to Calculate Your kWh Rate | Electricity Company in Texas*. NEC Co-Op Energy. https://neccoopenergy.com/how-to-calculate-your-kwh-rate/

Getting Ready for ROS Part 7: Describing a robot with URDF | Articulated Robotics. (n.d.). Retrieved December 26, 2023, from https://articulatedrobotics.xyz/ready-for-ros-7-urdf/

Introduction to URDF — Industrial Training documentation. (n.d.). Retrieved December 26, 2023, from https://industrial-training-master.readthedocs.io/en/melodic/_source/session3/Intro-to-URDF.html

Kamath, A. (2021, September 5). *Comparing different SLAM methods*. https://adityakamath.github.io/2021-09-05-comparing-slam-methods/

Ltd, R. P. (n.d.). *Raspberry Pi 4 Model B specifications*. Raspberry Pi. Retrieved December 26, 2023, from https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

Lynch, V. (2020, July 16). Battery calculator. *The Engineering Mindset*. https://theengineeringmindset.com/battery-life-calculator/

urdf/XML/model—ROS Wiki. (n.d.). Retrieved December 26, 2023, from https://wiki.ros.org/urdf/XML/model

Voss, W. (n.d.). *A Comprehensible Guide to Servo Motor Sizing*.

40 C 8 data sheet,
from *https://www.bhel.com/sites/default/files/aa10208-r07-1578116619.pdf*

AISI-SAE-4130-Product-Datasheet-D.A.Cooper-Sons-.pdf ,from *https://www.dacooper.co.uk/wp-content/uploads/2020/11/AISI-SAE-4130-Product-Datasheet-D.A.Cooper-Sons-.pdf*

A Textbook of Machine Design by R.S.KHURMI AND J.K.GUPTA [tortuka]_1490186411865.pdf. (2023, October 5). https://www.slideshare.net/AmarYasser13/a-textbook-of-machine-design-by-rskhurmi-and-jkgupta-tortuka1490186411865pdf

ball screw catalogue - HIWIN.pdf , from *https://hiwin.us/wp-content/uploads/ballscrews.pdf*

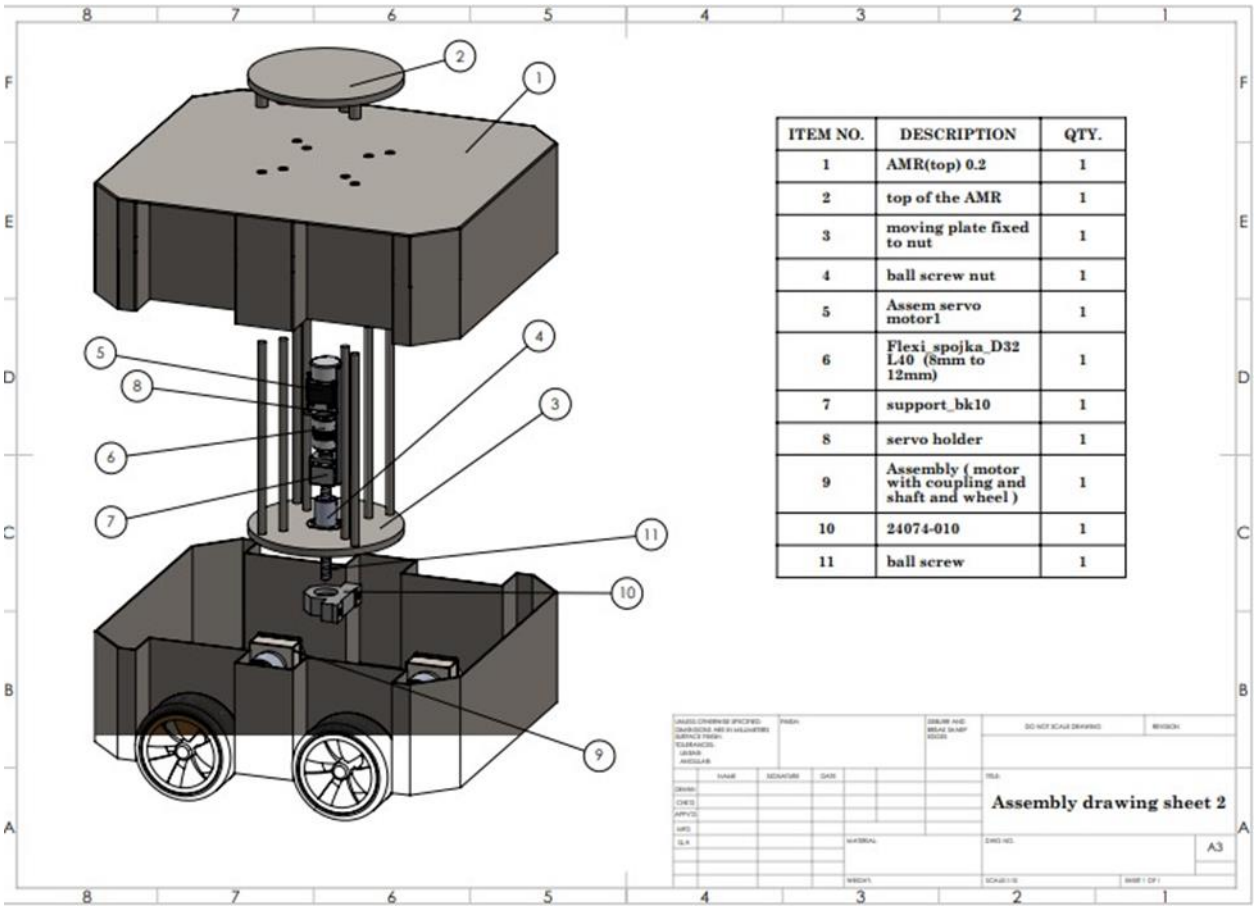
Ball screw catalogue - THK.pdf ,from *https://tech.thk.com/en/products/pdf_download.php?file=E_15_BallScrew.pdf*

Collision Free Navigation of a Multi-Robot Team for Intruder Interception by Marzoughi, Ali https://www.researchgate.net/publication/327549768_Collision_Free_Navigation_of_a_Multi-Robot_Team_for_Intruder_Interception

"I, Cobot: Future collaboration of man and machine". The Manufacturer. Retrieved 18 January 2024.

Appendix

Drawing sheets for our AMR

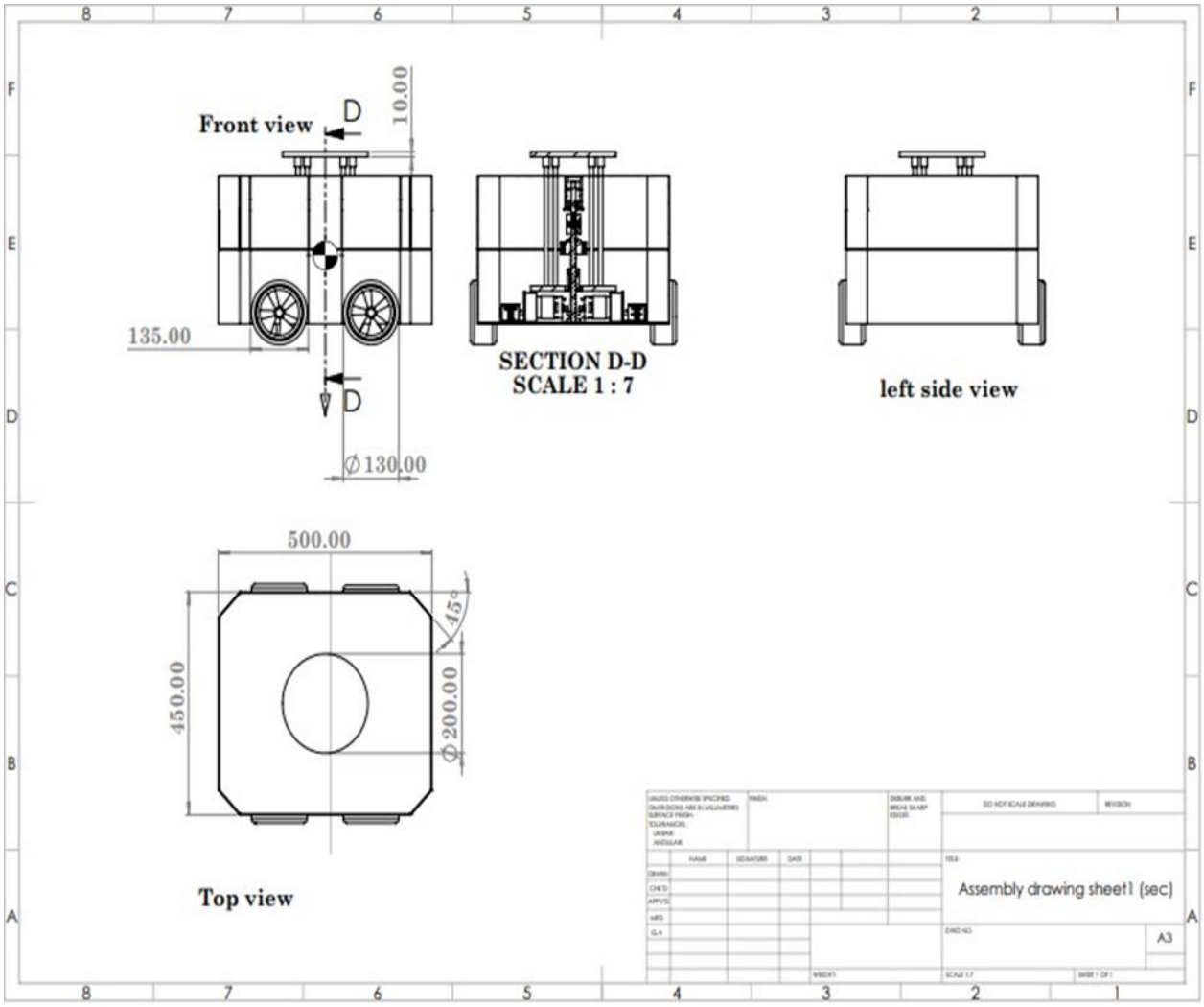


ITEM NO.	DESCRIPTION	QTY.
1	AMR(top) 0.2	1
2	top of the AMR	1
3	moving plate fixed to nut	1
4	ball screw nut	1
5	Assem servo motor1	1
6	Flexi_spojka_D32 L40 (8mm to 12mm)	1
7	support_bk10	1
8	servo holder	1
9	Assembly (motor with coupling and shaft and wheel)	1
10	24074-010	1
11	ball screw	1

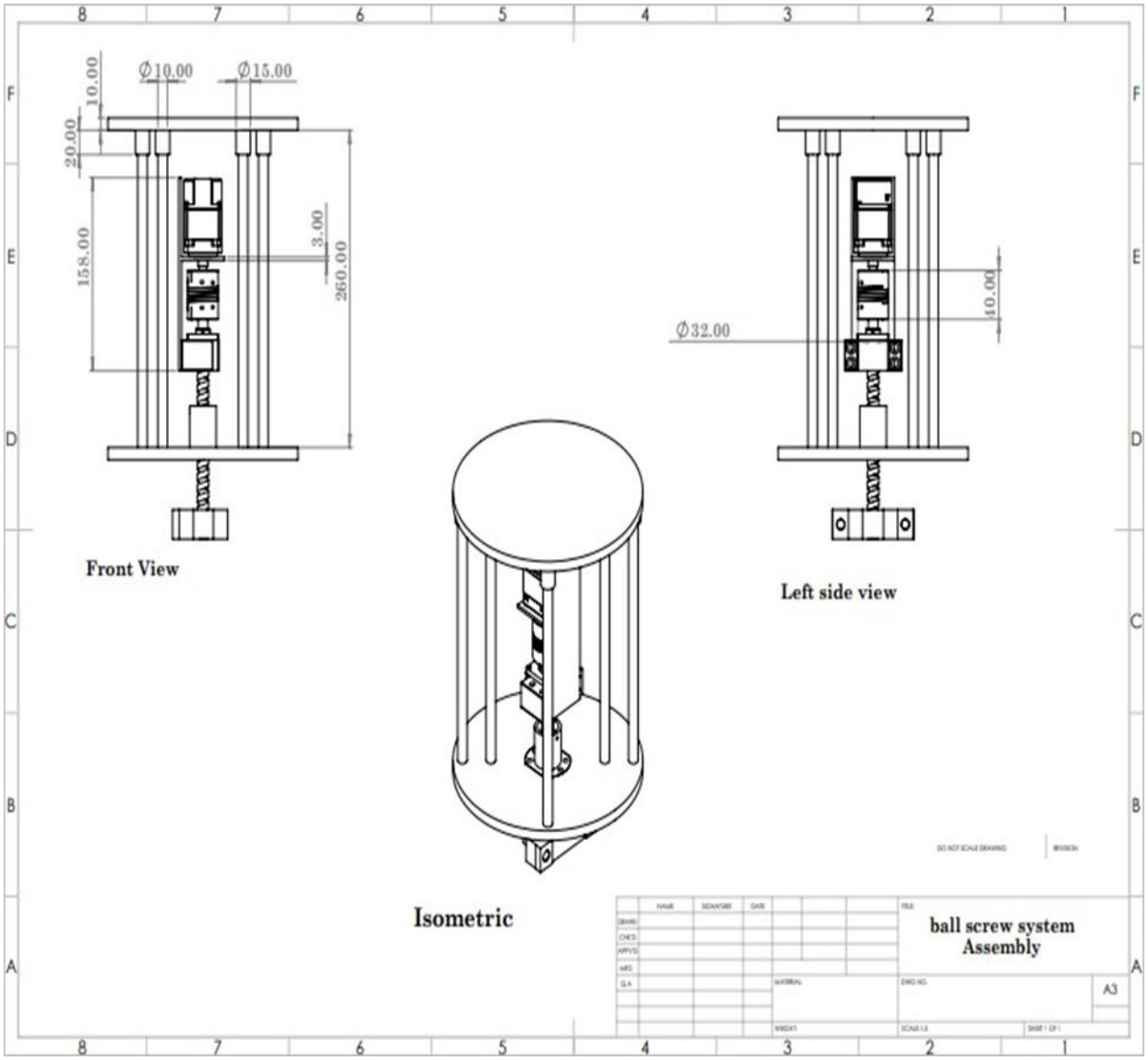
Assembly drawing sheet 2

A3

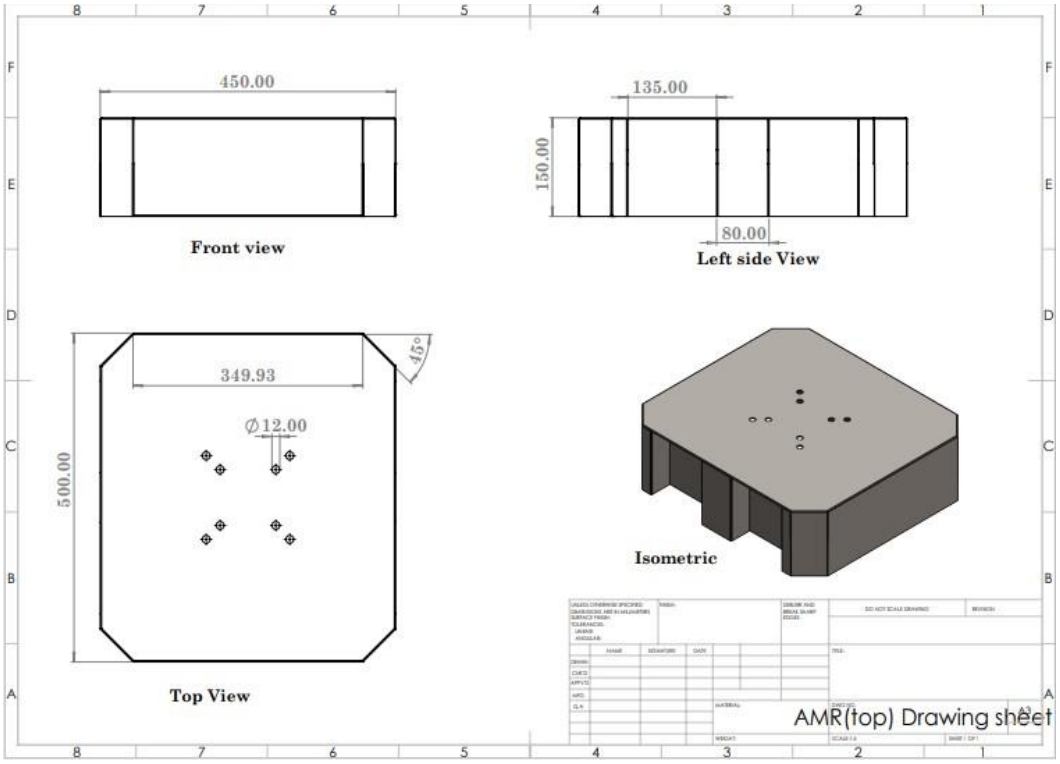
(Figure A-1 Assembly drawing sheet)



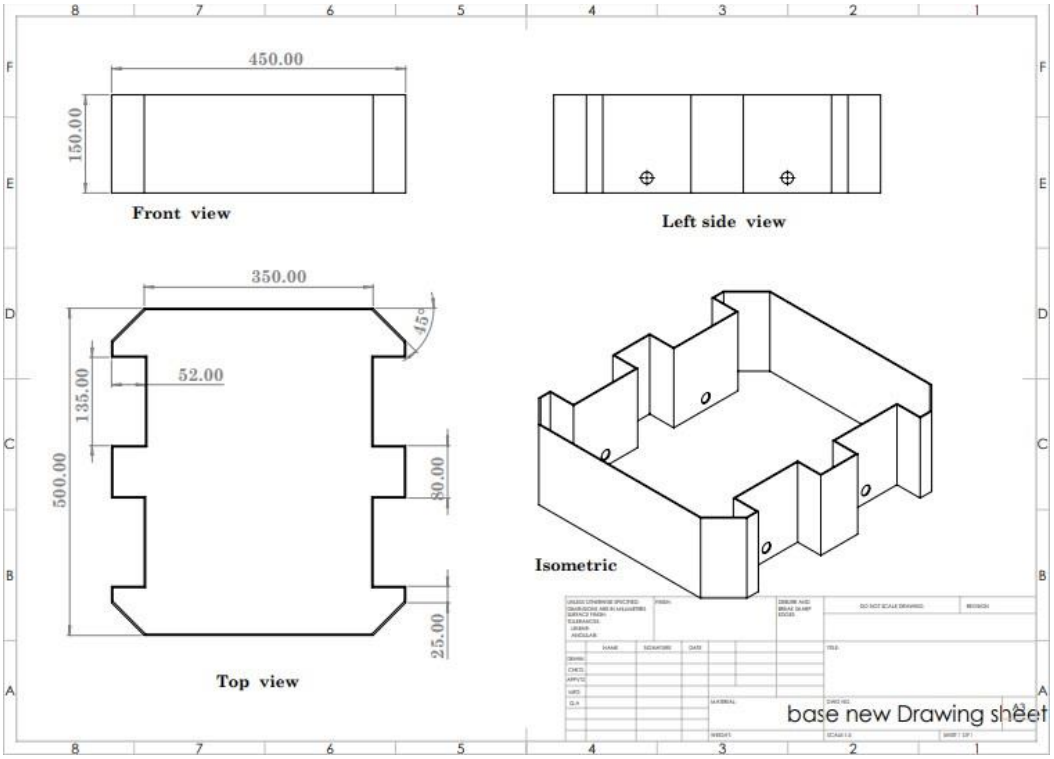
(Figure A-2 AMR drawing sheet)



(Figure A-3 ball screw system drawing sheet)



(Figure A-4 AMR top drawing sheet)



(Figure A-5 AMR base drawing sheet)

Hardware documentation:

https://drive.google.com/file/d/13SDHYEnfX5dmtx7m_uXHQYzaQlWOxfyh/view?usp=sharing

https://drive.google.com/file/d/1YcWV3u_n1QZP2ZASqxskPlA7FJJAOy2N/view?usp=sharing