MISR UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING
DEPARTMENT OF MECHATRONICS

AMR(Autonomous Mobile Robot in Warehouse)

by Group(3)

Nathan Romany 91500
Mahmoud Tawfik 95933
Remon Lamy 95893
Ammar Abdel Naser 35243
Abdullah Sayed 91645

Supervised by

Dr. Bahaa Naser
Prof. Dr. Mohamed Hamdy
Prof. Dr. Mohamed Ibrahim
Dr. Alaa Zakaria Nasser
Dr. Ahmed Sabri
Ass. Prof. Tahani Wileam
Dr. Bekhet Moha

June 2024

Abstract

Abstract this book is containing a fully represented methodology for an AMR (Autonomous mobile robot) which is big jump forward in 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 splits in chapters for each designing phase. Phase one : investigate the market and choose the application which our robot will serve and know more about the cost and previously associated projects, Phase two : start our customized design for the robot function requirements and choosing suitable Material and start implement the theoretical work through cad models and doing the appropriate test using computer aided tools, Phase three : identify our electrical component and 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.

Contents

List of Figures

List of Tables

Chapter 1

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.

1.1 Introduction to AMR

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 in a diverse range of industries

1.2 Uses of AMR

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 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.

Figure 1.2: AMR in market

Figure 1.3: AMR in restaurant

Figure 1.4: AMR in hospital

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.

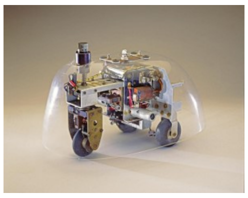


Figure 1.5: Elijah mobile robot 1948

1.3 History of AMRs

1.3.1 Autonomus vacuum cleaner

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 help 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)

1.3.2 Autonomus warehouses robots

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 1.6: Roomba autonomous

1.3.3 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.

Figure 1.7: collaborative robot

1.4 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

1.5 Design Consideration

After defining our robot commercially now here it is the technical definition first we have the robot features: 1. Moving speed up to 1.3 m/s 2. Mapping and localization 3. Tracking destination dynamically 4. Carrying backed products n 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: 1. Organizational structuring within the team and assembly methodology. 2. The scientific foundation of the design and its technical stages.

1.5.1 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.

1.6 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

1.7 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 Implementation

2.1 Mechanical Design

2.1.1 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.

2.1.2 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

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).

Requirements

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

Cost Effectiveness

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

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.

Functionality

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

Simplicity

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

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.

Aesthetics

-quality look
- safety
- Handling
- theme
- treatment or coating
- impression matter.

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.

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.

2.1.3 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	$5.56 \times 10^{-1} mm$
Steel	3 mm	21.5 Kg	350 Kg	$7.5 \times 10^{-1} mm$
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 Benefits: High strength-to-weight ratio Excellent fatigue strength Good ductility Good machinability

2.1.4 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	10×48
Ball screw system	1	3	1.2×20(for the screw)

2.1.5 Motor Sizing

For the DC Motors:
Wheel Rim = 5 cm
Car weight = $16 + 3 + 3 + 1.5 + 0.5 = 24$ Kg
Desired = 98 Kg
Total weight = 122Kg = 1197 N
Assuming the equal distribution of load $\frac{1197}{4} = 299.2$ N
MOTOR POWER = $\frac{F \times N \times R}{9550}$
F : Load
N : No. of revolution = 180 rpm
R : radius of the rim = 0.025 m
MOTOR POWER = $\frac{299.2 \times 180 \times 0.025}{9550}$
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 $\approx$ 0.227 hp

2.1.6 Shaft Design

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

Material for the shaft is steel Grade 40C8 that has a yield strength= 320 MPa
Take factor of safety = 4
Ultimate tensile strength = 560 MPa
Find allowable stress and shear:
$\sigma_b = \frac{S_yt}{f.s} = \frac{320}{40} = 80$ MPa
$\tau = \frac{0.5 \times S_yt}{f.s} = \frac{0.5 \times 320}{4} = 40$ MPa
Power of motor = 170 watt
N = 180 rpm
$Torque = \frac{power \times 60}{2\pi N}$ $Torque = \frac{170 \times 60}{2\pi \times 180} = 9.01$ N.m $= 9.01 \times 10^3$ N.mm
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 $= \frac{122Kg}{4} = 30.5$ Kg , W $=30.5/times9.81 = 299.2$ N
M: Bending moment
M $= WL = 299.2 \times 37 = 11070.4$ N.mm

T_e: equivalent twisting moment
M_e: equivalent bending moment
$T_e = \sqrt{(K_m \times M)^2 + (K_t \times T)^2}$
$T_e = \sqrt{(K_m \times M)^2 + (K_t \times T)^2} = 18892.48$ N.mm
$M_e = \frac{1}{2}[K_m \times M + \sqrt{(K_m \times M)^2 + (K_t \times T)^2}]$ $M_e = \frac{1}{2}[K_m \times M + \sqrt{(K_m \times M)^2 + (K_t \times T)^2}\ ] = 17749.04$ N.mm
$T_e = \frac{\pi \tau d^3}{16}$, $d = 13.39$ mm
$d = 15$ mm approx.
$M_e = \frac{\pi \sigma_b d^3}{32}$, $d = 13.12$ mm
$d = 15$ mm approx.

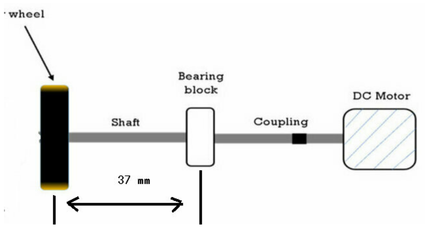


Figure 2.1: Powertrain

The Endurance Limit for shaft

What Does Endurance Limit (S_e) Mean? The endurance limit (S_e) 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.
$S_e = K_aK_bK_cK_dK_eK_fS_e^{'}$ 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
F = Fatigue strength fraction
$K_a = aS_{ub}^b$
Get a and b from Table 2.4

Table 2.4: value of factor a and b for K_a

	Factor a		
Surface Finish	S_{ub} kpsi	S_{ub} MPA	e^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

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

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

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

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

we can calculate K_d by using the following equation:
$K_d = 0.975 + 0.432 \times 10^{-3}T_F - 0.115 \times 10^{-5}T_F^2 + 0.104 \times 10^{-8}T_F^3 - 0.595 \times 10^{-12}T_F4$
Our robot will work in ware house at room temperature = 25 °C which =77 °F
So $K_d = 1.0018$

Table 2.5: Reliability Factor, K_e

Reliability, %	Transformation Variate, z_0	Reliability Factor, K_e
50	0	1.000
90	1.288	0.897
95	1.645	0.868

Get K_e value from table

Take the reliability $=90 K_e = 0.897$
Take $K_f = 1$
$S_e^{'} = 0.5 S_{ut} at 10^6 cycles, S_e^{'} = 280 MPa$
$S_e = (0.6137)(0.928)(1)(1.0018)(0.897)(1)(280) = 143.26 MPa$ For our material the Minimum tensile strength = 560 MPa
Which equal to 81.22 Kpsi (kilo pound force per square inch)
1 Kpsi = 6.895 MPa

Fatigue strength fraction (f) of (S_{ut}) at 10^3 cycles for $S_e = 0.5 S_{ut} at 10^6 cycles.$
F= Fatigue 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 F from Figure 2.2 $F \approx 0.875$
$N = 10^3$
$a = \frac{(FS_{ut})^2}{S_e} = \frac{(0.875 \times 560)^2}{143.26} = 1675.97 MPa$
$b = -\frac{1}{3} \log(\frac{FS_ut}{S_e}) = -0.1780$
$S_f = aN^b$
$S_f = 1675.97 \times (10^3)^{-0.1780} = 490.079 MPa$

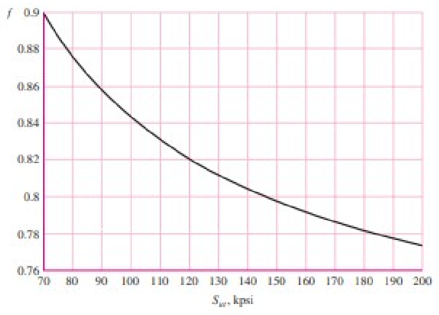


Figure 2.2: Fatigue strength fraction

2.1.7 Bearing Calculations

life (L) in revolutions can be expressed as
$L_D = L_{10} = l \times n \times 60$
Where:
L_D: Desired life in revolutions
L_R: Rating life in revolutions = 10^6
l: life (hours)
n: speed (rpm)
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
$X_D = \frac{L_D}{L_R} = \frac{54 \times 10^6}{10^6} = 54$
$C_{10} = a_f F_D [\frac{x_D}{x_o + (\theta - x_o)(1 - R_D)^{\frac{1}{b}}}]^{\frac{1}{a}}$ $R \geq 0.90$
Where:
R: Reliability = 0.90
F_D = 299.2 N
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
$C_{10} = 1.25 \times 299.2[\frac{54}{0.02+(4.439)(1-0.90)^{\frac{1}{1.483}}}]^{\frac{1}{3}} = 1375.83N = 1.375KN$
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

Table 2.6: Bearing Specifications

Designation	**Principal Dimensions**			**Basic Load Ratings**		**Speed Ratings**	
				dynamic	**static**	**dynamic**	**static**
	d [mm]	**D [mm]**	**B [mm]**	**C [kN]**	**Co [kN]**	**Reference Speed [r/min]**	**Limiting Speed [r/min]**
6002	15	32	9	5.85	2.85	50000	32000
6002-2RSH	15	32	9	5.85	2.85	-	14000
6002-2RSH/VA947	15	32	9	5.85	2.85	-	14000
6002-2RSL	15	32	9	5.85	2.85	50000	26000
6002-22	15	32	9	5.85	2.85	50000	26000
6002-RSH	15	32	9	5.85	2.85	-	14000
6002-RSL	15	32	9	5.85	2.85	50000	32000
6002-Z	15	32	9	5.85	2.85	50000	32000
61802	15	24	5	1.9	1.1	60000	38000

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

2.1.9 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
$\mu_s = \frac{F_r}{mg}$, $\mu_k = \frac{F_r}{mg}$
R_T= raduis of tire = 0.065 m
T_{FS}: Torque at static friction
T_{FK}: Torque for kinetic friction

F_r: Force to move object at constant speed

For static

$F_{rs} = m \times g \times \mu_k = 299.2 \times 0.6 = 179.52N$
$T_{FS} = F_{rs} \times R_T = 180 \times 0.065 = 11.7N.m$
$T_{FS} > T$ driving shaft
For the car to move without slipping

For kinetic

$F_{rk} = m \times g \times \mu_k = 299.2 \times 0.4 = 119.68N$
$T_{FK} = F_{rk} \times R_T = 120 \times 0.065 = 7.8N.m$

2.1.10 Friction between Two Tires and Floor:

F_{ms}: Max friction at static= $2F_{rs}$= 360 N
F_{Mk}: Max friction at kinetic= $2F_{rk}$= 240N
a: Acceleration
m: mass of the car
$a = \frac{F_{Ms}}{m} = \frac{360}{122} = 2.9m/s^2$
This the max acceleration that the robot can provide during full load with no slip.

Static at no load

m_e: car weight no load= 22 Kg
$F_{rs} = mg\mu_s = 5.5 \times 9.81 \times 0.6 = 33N$
$T_{Fs} = F_{rs}R_T = 33 \times 0.065 = 2.145N.m$
$a = \frac{F_{rs}}{m_e} = \frac{2\times 33}{22} = 3m/s^2$

Kinetic at no load

$F_{rk} = mg\mu_k = 5.5 \times 9.81 \times 0.4 = 22N$
$T_{Fk} = F_{rk}R_T = 22 \times 0.065 = 1.43N.m$
$a = \frac{F_{rk}}{m_e} = \frac{2\times 22}{22} = 2m/s^2$
[1] F_{cr}: climbing and downgrade force
$F_{cr} = \pm mg(\sin\alpha$
$F_{cr} = \pm mg(\sin\alpha = \pm 25 \times 9.81 \times \sin 18$ at no load
$F_{cr} = \pm mg(\sin\alpha = \pm 140 \times 9.81 \times \sin 18$ Max. load
$F_{cr} = 75.8N$ at no load
$F_{cr} = 424.4N$ Max. load
$T_{cr} = F_{cr}R_T = 75.8 \times 0.065 = 4.927N.m$ No load
$T_{cr} = F_{cr}R_T = 687 \times 0.065 = 27.586N.m$ Max. load
$F_T = F_{rs} + F_{cr} = 179.52 + 424.4 = 603.92N$ Max. load

[1] F_D Drag friction force
We element drag force due to work in door friction force for climbing and downgrade

$T_{Fr} = 7.8 + 4.927 = 12.727 N.m$ no load
$T_{Fr} = 11.7 + 27.586 = 39.286 N.m$ Max. load

2.1.11 Trapezoidal Curve for (Actual N75. . . .)

Using our Motor

Move 10 meters in a straight line
At a=0.25 m/s^2
$V_m = 0.5 m/s$
$t_a = t_d = \frac{0.5}{0.25} = 2s$ where t_d: deceleration time
where t_a: acceleration time
$s_a = \frac{1}{2} \times 0.5 \times 2 = 0.5$ where s_a: Displacement covered during acceleration
$s_d = 0.5m$ where s_d: displacement covered during deceleration
$s_m = 9m$ where s_m: distance covered during motion
$J_{snoload}$: inertia of the system at no load

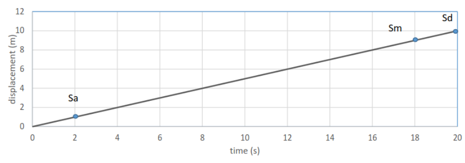


Figure 2.3: Displacement - Time Graph

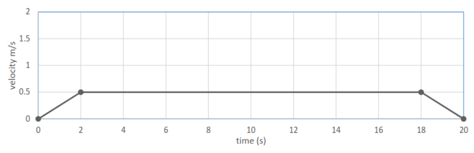


Figure 2.4: Velocity - Time Graph

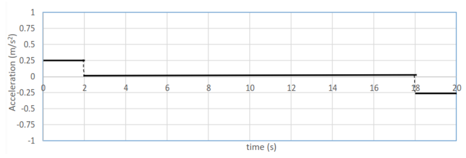


Figure 2.5: Acceleration - Time Graph

J_{sload}: inertia of the system at full load
T_{acc}: acceleration torque
T_{total}: Total torque

$J_{snoload} = 0.72 Kg.m^2$
$J_{sload} at 30 Kg = 3 + 2.8 + 2.2 = 8 Kg.m^2$
$J_{sload} at 60 Kg = 3.3 + 4.5 + 4.2 = 12 Kg.m^2$
No load:
$T_{acc} = 0.56 N.m$ $T_{total} = T_l + T_{acc} = 2.87 + 0.56 = 3.43 N.m$

Loaded at 30 Kg:
$T_{acc} = 1.2N.m$ $T_{total} = T_l + T_{acc} = 5 + 1.12 = 6.2N.m$
Loaded at 60 Kg:
$T_{acc} = 1.5N.m$ $T_{total} = T_l + T_{acc} = 8.5 + 1.5 = 10N.m$

2.1.12 Trapezoidal Curve (Theory)

Move 10 meters in a straight line
At $a_{max} = 2\frac{mV}{s^2} = 1.5m/s^2$
$t_a = t_d = \frac{1.5}{2} = 0.75s$ where t_d: deceleration time
where t_a: acceleration time
$s_a = \frac{1}{2} \times 1.5 \times 0.75 = 0.6$ where s_a: Displacement covered during acceleration
$s_d = 0.6m$ where s_d: displacement covered during deceleration
$s_m = 8.8m$ where s_m: distance covered during motion

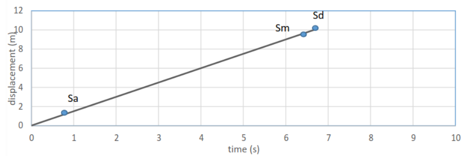


Figure 2.6: Displacement - Time Graph

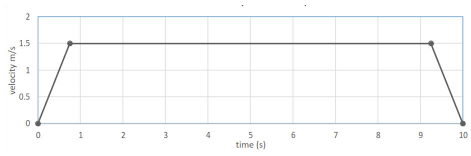


Figure 2.7: Velocity - Time Graph

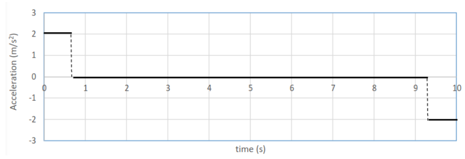


Figure 2.8: Acceleration - Time Graph

a: Angular acceleration
I_{xx}:component of inertia a $\in$ x axis
I_{yy}:component of inertia a $\in$ y axis
I_{zz}:component of inertia a $\in$ z axis
h:height
d:length
w:width
$T_{acc} = J_s a$
$I_{xx} = \frac{1}{12}m(h^2 + d^2)$
$I_{yy} = \frac{1}{12}m(d^2 + w^2)$

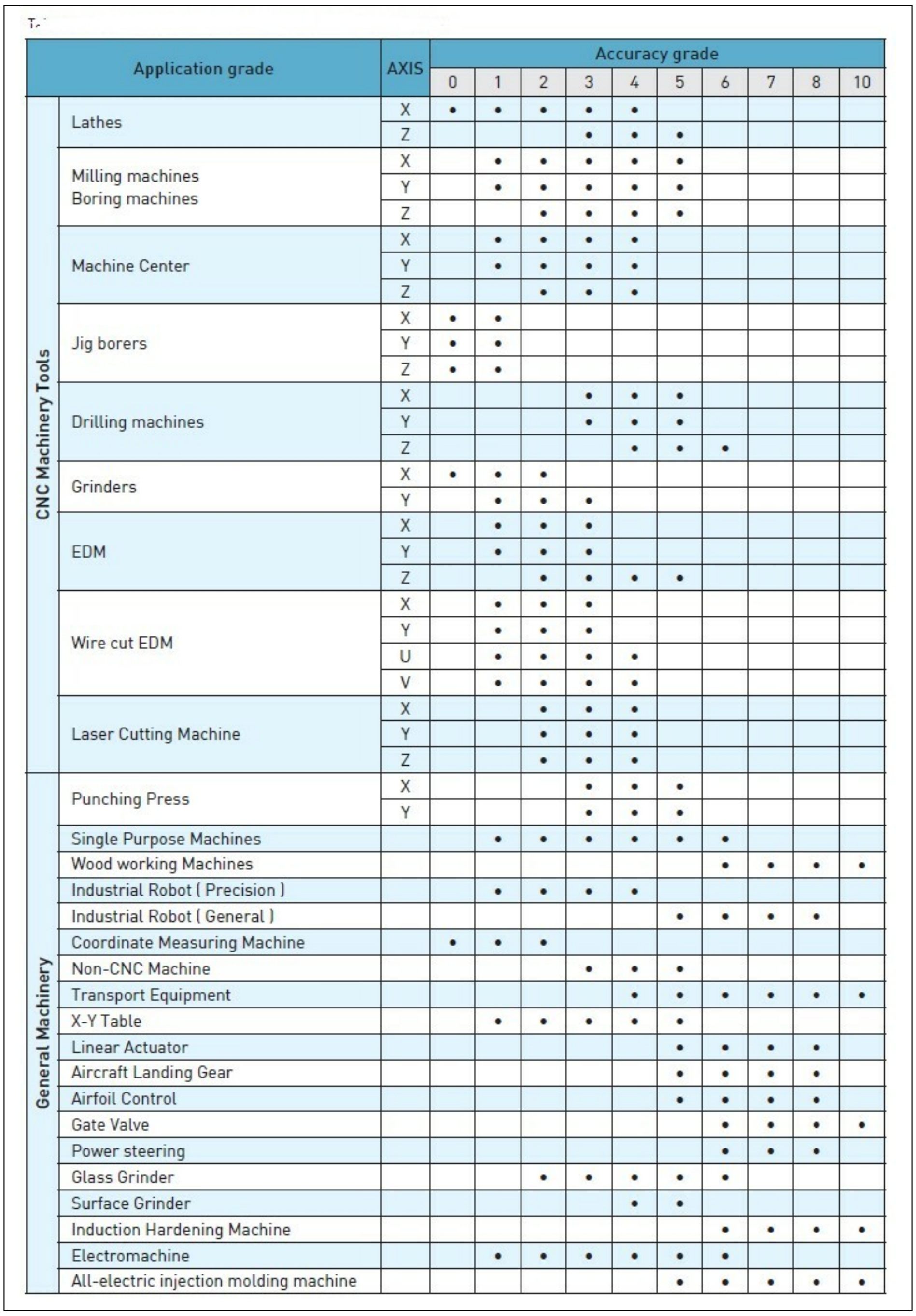
$I_{zz} = \frac{1}{12}m(h^2 + w^2)$

$J_{snoload} = 0.195 + 0.3 + 0.227 = 0.72Kg.m^2$
$J_{sload} = 4.75 + 5 + 3.7 = 13.45Kg.m^2$
No load:
$T_{acc} = 3.9N.m$
Loaded at 60 Kg:
$T_{acc} = 7.8N.m$
Loaded at 100 Kg:
$T_{acc} = 13N.m$

2.1.13 Ball Screw system

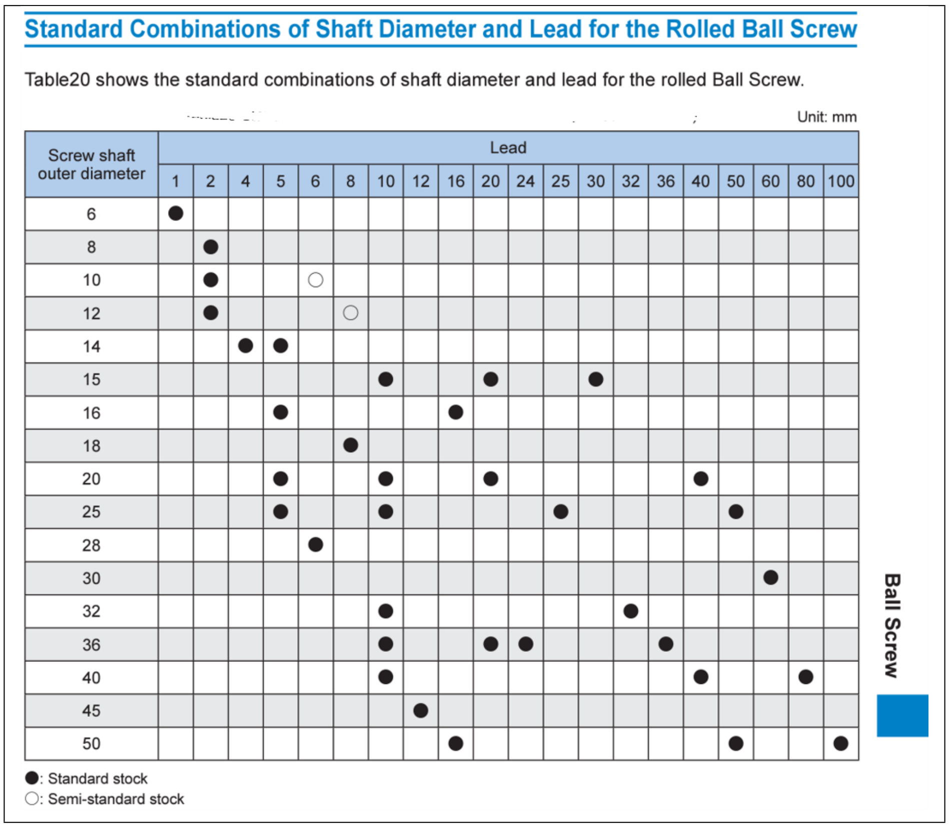
We selected the accuracy grade from the table (4) of lead angle accuracy, selected C7 grade Because it is the most popular and is in the table of accuracy grade select for linear actuators The shaft diameter = 12 mm (for availability) and shaft length 20cm from table (5) Take the lead = 2 mm from table (5) which for every full rotation the nut will move 2 mm linearly.
[2] Stepper motor (NEMA23) which has N= 18 rpm $(57H, 1.8°)$
Axial load= $220N^{thelead} = 2mm$
Torque=$\frac{Axialload \times lead}{2000\pi(0.8)}$
0.8 refer to effeciency of ball screw
Torque=$\frac{220\times2}{200\pi(0.8)} = 0.08N.m \approx 0.1N.m$
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.

[2]Tables of these equations follow them



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: Recommended accuracy grade.jpg



Standard Combinations of Shaft Diameter and Lead for the Rolled Ball Screw

Table20 shows the standard combinations of shaft diameter and lead for the rolled Ball Screw.

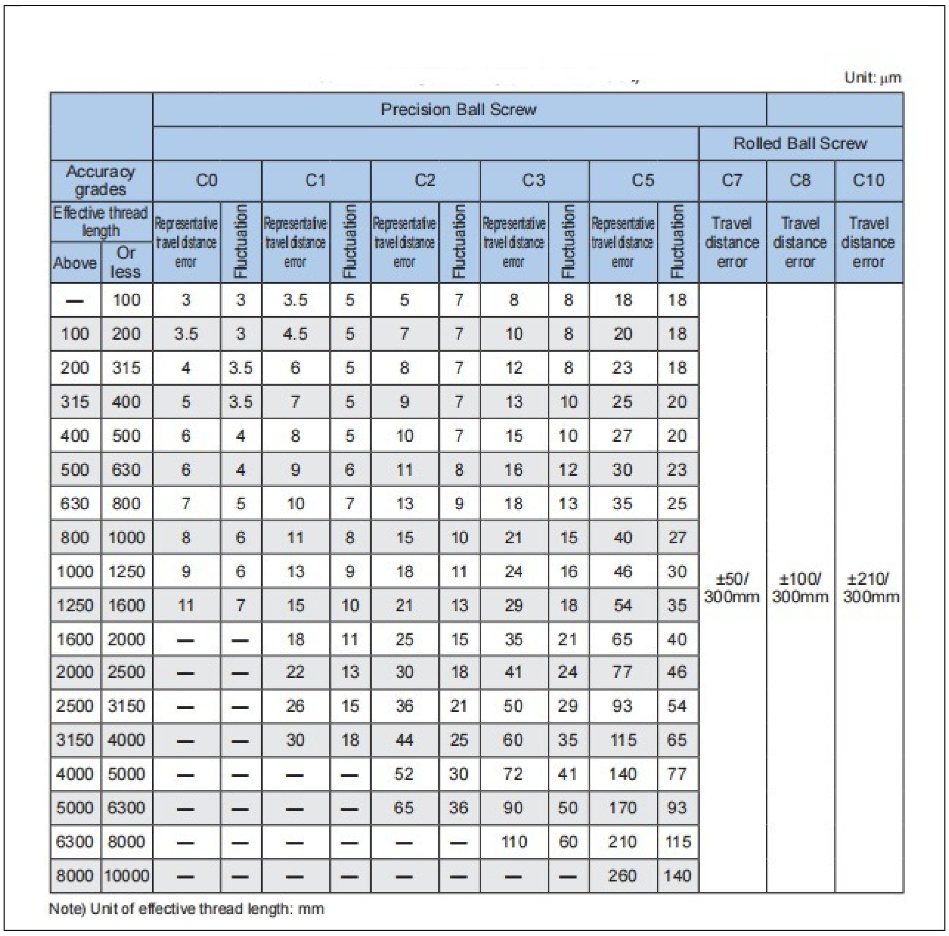
Unit: mm

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

Ball Screw

Table 2.8: standard of shaft diameter and lead for roller ball



Unit: μm

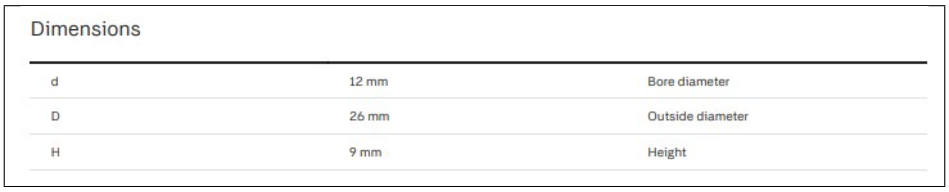
		Precision Ball Screw												
												Rolled Ball Screw		
Accuracy grades		C0		C1		C2		C3		C5		C7	C8	C10
Effective thread length		Representative travel distance error	Fluctuation	Representative travel distance error	Fluctuation	Representative travel distance error	Fluctuation	Representative travel 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: lead angle accuracy

2.1.14 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



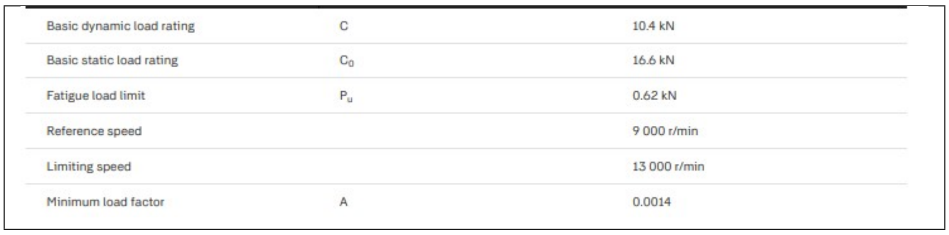
Dimensions

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

Table 2.10: 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)
• Applied Load over the bearing ¿ Minimum Requisite load From data sheet Kr = 0.0014, C = 10.4 KN
So Minimum Requisite load = $0.0014 \times 10.4 = 0.01456KN$



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

Table 2.11: data of bearing

Our load = $300N, 0.3KN > 0.01456KN$ ✓
Speed limit:
Bearing max. speed should be lower than (Reference speed, Limiting speed)
our speed is 180 RPM ¡ (9000 RPM, 13000 RPM) ✓

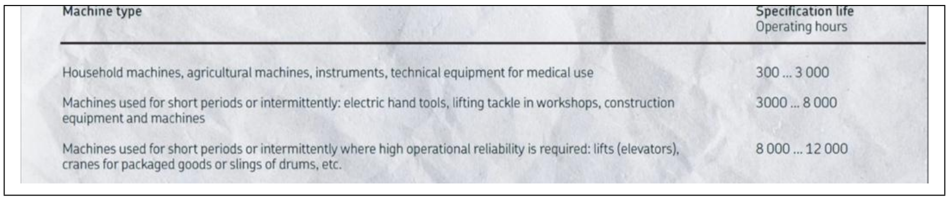
Step 2 Basic Rating life L10

$L10 = (\frac{C}{P})^p$
Where L10 is Basic Rating life

C is Basic Dynamic load rating
P is Equivalent dynamic
p is = 3 for ball bearing, 3.33 for Roller bearing
$P = XF_r + YF_A]a$
Where:—— F_r: Actual Radial load (KN)
F_a: Actual Axial load (KN)
X: Radial load factor
Y: Axial load factor
At our case load is pure Axial so $P = F_a = 300N = 0.3KN$

Step 3 Basic Rating life L10

find Basic Rating life $= L10 = (\frac{10.4}{0.3})^3 = 41661.62M(rev)$
Now compare our result to the Recommended life of the bearing • Should meet or exceed the Recommended value of the recommended life
Find Operating hours $= L10h = \frac{10^6}{60n} \times L10, \frac{10^6}{60 \times 180} \times 41661 = 3857500hr$



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

Table 2.12: specification life according to Machine type

2.2 Parts Drawing and Assembly

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

2.2.1 Full Assembly for Our AMR Robot

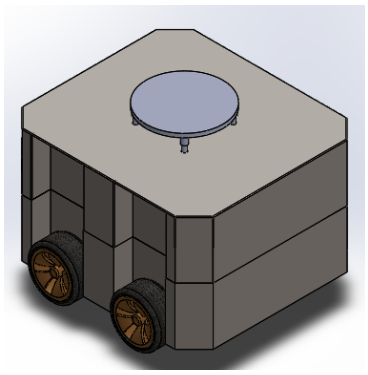


Figure 2.9: Full Assembly for Our AMR Robot

2.2.2 Ball Screw System showing its components

Ball Screw Shaft, The Nut, Stepper motor, Top plate, Base plate with 8 rods.

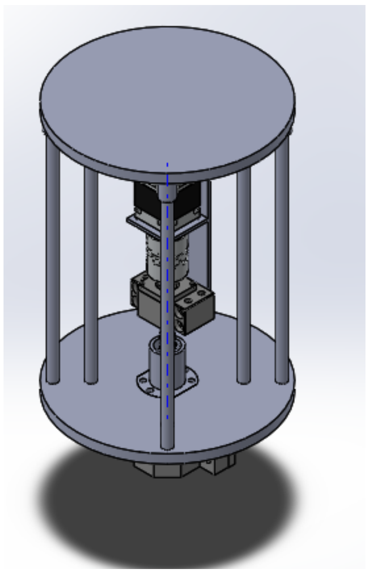


Figure 2.10: Ball Screw System showing its components

2.2.3 Dc Motor for Wheels and Stepper Motor for Ball Screw System

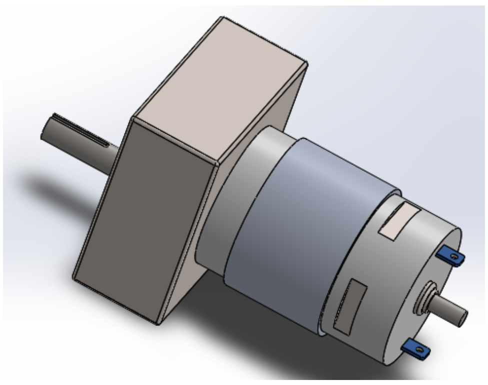


Figure 2.11: DC Motor for Wheels

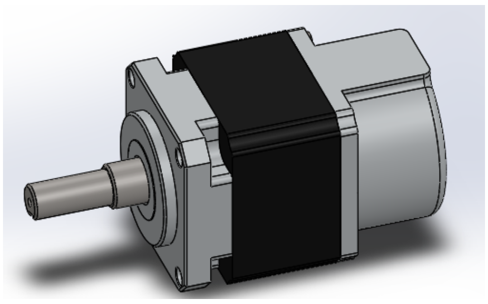


Figure 2.12: Stepper Motor for Ball Screw System

2.2.4 Coupling

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

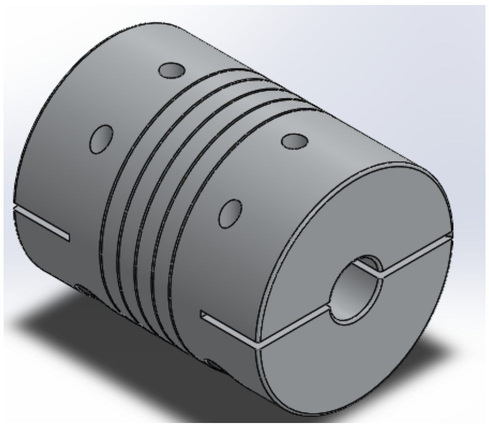


Figure 2.13: Coupling dc Motor to the shaft

2.2.5 Wheels

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

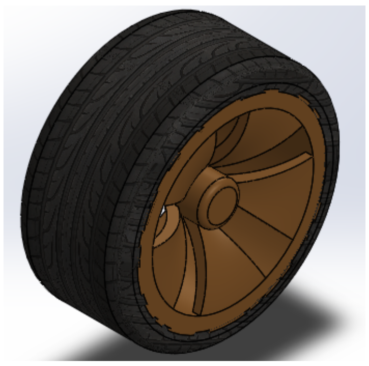


Figure 2.14: wheel consists of two parts: the outer tire is made of rubber and the rim

2.2.6 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

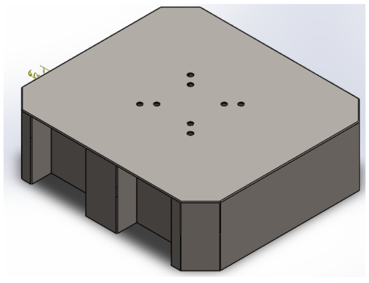


Figure 2.15: AMR's top

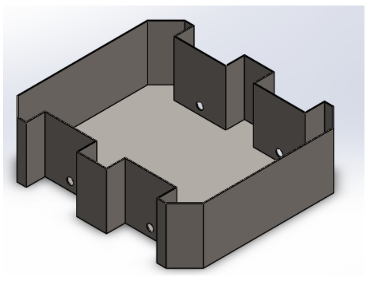


Figure 2.16: AMR's base

2.3 Stress Analysis

In figure 2.16 shows the maximum and minimum stress value when apply 30 kg load and the yield strength for
our material. $= 2.827 \times 10^8 N/m^2$
Max $= 7.702 \times 10^5 N/m^2$
Min $= 1.894 \times 10^{-1} N/m^2$

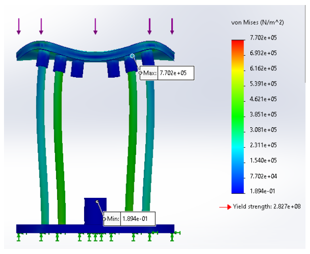


Figure 2.17: stress analysis from SOLIDWORKS software

In figure 2.17 shows the maximum and minimum Displacement after apply the load.
Max $= 6.878 \times 10^{-4} mm$
Max $= 1.0 \times 10^{-30} mm$

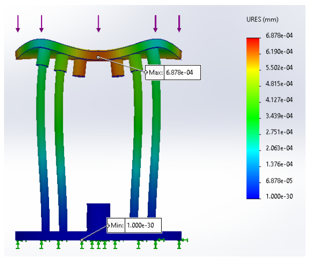


Figure 2.18: displacement analysis

Chapter 3

Electrical Implementation & Design

3.1 Components

1. Raspberry-pi 4b module 8gb 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. KinectV1 RGB-D camera
6. RPlidar A1 m8
7. Four encoders
8. MPU(9250)
9. Battery
10. Battery management system

3.2 Components description

3.2.1 Raspberry-pi 4b module 8gb ram

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

Table 3.1: Raspberry-pi Information

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: Fatigue strength fraction

Positives

1. Vast Peripheral Support:

Raspberry Pi comes with 26 GPIO Pins which are very use-
ful 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.

2. Multiple Sensors:

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.

3. Supports all type of Codes:

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, Cpp, Csh, Ruby, Java, Python and 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.

4. Faster Processor:

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.

5. Can be Used as a Portable Computer:

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

1. Missing eMMC Internal Storage

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.
2. Graphics Processor Missing
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.
3. Overheating:
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.
5. Not able to run Windows Operating system:
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.

3.2.2 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

3.2.3 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 [1]
• 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 × 62mm

Applications

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

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

We need module to step down the voltage from 12V to 5v/2A-3A for Raspberry pi.

3.2.5 KinectV1 RGB-D camera

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.

[1]please double-check the polarity before power-up.

3.2.6 RPlidar A1 m8

3 / 16 Copyright (c) 2009-2013 RoboPeak Team Copyright (c) 2013-2016 Shanghai Slamtec Co., Ltd. RPLIDAR A1 is a low cost 360 degree 2D laser scanner (LIDAR) solution developed by SLAMTEC. The system can perform 360degree scan within 6meter range. The produced 2D point cloud data can be used in mapping, localization and object/environment modeling. RPLIDAR A1's scanning frequency reached 5.5 hz when sampling 360 points each round. And it can be configured up to 10 hz maximum. RPLIDAR A1 is basically a laser triangulation measurement system. It can work excellent in all kinds of indoor environment and outdoor environment without sunlight.

Chapter 4

Real Robot Implementation

4.1 Embedded Layer

4.1.1 Parts

As we remember at at chapter 3, 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.

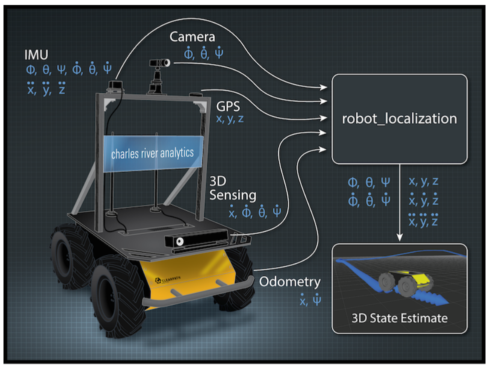
• ESP32 is one of oldest MCU's which has board manager at arduino IDE, that would provide us with many resources, libraries and community support.

• 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

4.1.2 MPU9250 and Magnetic rotary encoders

May some of readers get confused of mentioning 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. Odometery is the backbone of project. it is the feedback of robot location away from origin. We decided to depend on IMU and encoders of getting our odometery using kalmn filter.
I think now you could to understood why we mentioned MPU9250 and Encoders together.



IMU
$\Phi, \theta, \Psi, \dot{\Phi}, \dot{\theta}, \dot{\Psi}$
$\ddot{x}, \ddot{y}, \ddot{z}$
Camera
$\dot{\Phi}, \dot{\theta}, \dot{\Psi}$
GPS
x, y, z
charles river analytics
3D Sensing
$\dot{x}, \dot{\Phi}, \dot{\theta}, \dot{\Psi}$
Odometry
$\dot{x}, \dot{\Psi}$
robot_localization
Φ, θ, Ψ
$\dot{\Phi}, \dot{\theta}, \dot{\Psi}$
x, y, z
$\dot{x}, \dot{y}, \dot{z}$
$\ddot{x}, \ddot{y}, \ddot{z}$
3D State Estimate

Figure 4.1: robot localization package inputs ability

4.1.3 Motor Drivers (cytron MDD10A)

As we mentioned previously what is the usage of motor drivers and how it works. [2]

4.2 Embedded layer codes algorithm

4.2.1 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: digitalRead(), digitalWrite() 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

[2] Revise chapter 3 section 3.1.2

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.

4.2.2 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 magnetoresistive sensor positioned near the magnet.
- **Signal Processing Unit**: Converts the raw sensor data into a readable 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 magnetoresistive 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-quad 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-quad 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 minutes
    double 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 calculation
    prev_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 sensor 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.

4.3 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.

4.3.1 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.

4.3.2 Accelerometer

The accelerometer measures linear acceleration along the X, Y, and Z axes. It typically uses a microelectromechanical system (MEMS) structure composed 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.

4.3.3 Magnetometer

The magnetometer measures the strength and direction of the magnetic field along the X, Y, and Z axes. It typically uses anisotropic magnetoresistive (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.

4.3.4 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.

4.3.5 Interfaces and Data Output

The MPU9250 communicates with external devices via I2C or SPI interfaces, 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.

subsectionUsing 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 the I2C 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_mps2());
    Serial.print("\t");
    Serial.print(imu.accel_y_mps2());
    Serial.print("\t");
    Serial.print(imu.accel_z_mps2());
    Serial.print("\t");
    Serial.print(imu.gyro_x_radps());
    Serial.print("\t");
    Serial.print(imu.gyro_y_radps());
    Serial.print("\t");
    Serial.print(imu.gyro_z_radps());
    Serial.print("\t");
    Serial.print(imu.mag_x_ut());
    Serial.print("\t");
    Serial.print(imu.mag_y_ut());
    Serial.print("\t");
    Serial.print(imu.mag_z_ut());
    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.

4.3.6 Summary

In essence, the MPU9250 works by measuring angular velocity, linear acceleration, 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 crucial for a wide range of advanced applications.

4.4 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.

4.4.1 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 configurations, while the `spin` method controls the motor's speed and direction 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));


}
```

4.4.2 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.

4.5 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.

4.5.1 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 required 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 linear and angular velocities based on motor RPMs.

- Functions:

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

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

   - Function:

     ```
     int Kinematics::rpmToPWM(int rpm)
     ```

4.5.2 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, angul
   ```

3. **PWM Conversion**:

   ```
   Kinematics::output pwm = kinematics.getPWM(linear_x, linear_y, angul
   ```

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 motion.

4.6 rosserial_node and robot base code

This code sets up a ROS node for controlling a robot's motion using velocity 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

#define MOTOR1_ENCODER_A 5
#define MOTOR1_ENCODER_B 23

#define MOTOR2_ENCODER_A 12
#define MOTOR2_ENCODER_B 13

#define MOTOR3_ENCODER_A 16
#define MOTOR3_ENCODER_B 17

#define MOTOR4_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_data();
       imu.new_mag_data();
      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);
}
```

4.6.1 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.

- ### 4.6.2 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.

- ### 4.6.3 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.

- ### 4.6.4 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.

- **4.6.5 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.

- **4.6.6 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 initialized.

- **4.6.7 Setup Function**

  :

  – `setup()` function initializes the ROS node, sets up serial communication, initializes the MPU9250 sensor, and configures it.

- **4.6.8 Loop Function**

  :

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

- **4.6.9 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.

- **4.6.10 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.

- **4.6.11 IMU Data Publishing Function**

  :

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

- **4.6.12 Debug Function**

  :

  - `printDebug()`: Prints encoder readings for debugging purposes.

4.7 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.

4.7.1 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_calib[3].

Imu_calib Nodes: Imu_calib has two nodes:

- **do_calib**: do_calib is responsible for performing some steps in sequence 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 Package

The Kalman filter is a powerful mathematical tool used in the `robot_localization` ROS (Robot Operating System) package to provide accurate state estimation for mobile robots. This package utilizes different versions of the Kalman filter, including the Extended Kalman Filter (EKF) and the Unscented 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

[3]For 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 provide 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 specifically 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 sensors 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 enhances the accuracy and reliability of the robot's state estimation, which is crucial for navigation and control tasks in various robotics applications.

4.8 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.

- ### 4.8.1 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 rotating 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 distance measurement with more than 4K/8K samples per second,RPLIDAR A1 supports 2K/4K samples per second. For a scan requires 360 samples 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>
```

[4]For 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 communication. 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 compensation 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.

• 4.8.2 Cartographer

Cartographer is a system that provides real-time simultaneous localization 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 optimization. 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, depending on context) to the global map frame. local_submap_pose Transforms 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** (

[5]For 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 configuration 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)/confi` and named `gbot_lidar_2d.lua`.
- (Optional) Starts the `cartographer_occupancy_grid_node` to convert the Cartographer map into a ROS occupancy grid map with a resolution of `0.05`.

- ### 4.8.3 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 following 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 Localization (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 reference for localization.

– `gui_publish_rate`: This sets the rate at which scans and paths are published for visualization purposes in the graphical user interface (GUI).

– `kld_err` and `kld_z`: These parameters are used in the KLD sampling algorithm, which is employed by AMCL for particle resampling.

– `laser_lambda_short`: This parameter is related to the sensor model and affects the likelihood of short-range laser measurements.

– `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 particle filter algorithm, which AMCL employs for localization.
- Parameters `odom_alpha1` to `odom_alpha5`: These parameters define 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 required 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, respectively, required before performing a filter update.

In summary, each parameter in the AMCL launch file plays a crucial role in configuring the behavior and performance of the AMCL localization algorithm within the ROS environment, fine-tuning various aspects such as sensor models, odometry characteristics, particle filter settings, and recovery behavior.