Alexandria Higher Institute of Engineering &Technology (AIET)

Department of Mechatronics Engineering (EME)

Final Year Project Report (2022-2023)

Project Title AMR (Autonomous Mobile Robot)

No	Student name	Department
1	Ahmed Mosaad Farouk El Qelawi	EME
2	Mohamed Ibrahim Mohamed Yakout	EME
3	Mohamed Ahmed Ibrahim Ahmed	EME
4	Mohamed Ahmed Hassan Ali	EME
5	Mohamed Ashraf Mohamed Mahmoud	EME
6	Mohamed Bahgat Mustafa Mohamed	EME
7	Mohamed Alsawy Morsy Alsawy	EME
8	Mennaallah Sliem Abdel Nasser Fikry	EME
9	Moustafa Mohammed Hassan Mohmed	CE

A report submitted in part fulfilment of the degree of

BSc in Department of Mechatronics Engineering (EME)

Supervisors: DR. Amr Abdelfattah, DR. Amr Nasr & DR. Lamiaa Abdallah

Head of Department: DR Hassan Al-Shobashy

June 15, 2023

The main goals expected to be achieved from SDGs in the present project

Here are the goals expected to be achieved from the SDGs through the graduation project 2022-2023:

1- Decent Work and Economic Growth

- Shift of work force.
- Minimize human errors and suffering.
- Help factories and companies in more efficient and accelerated production.

2- Industry, Innovation and Infrastructure

- This project can be upgraded and include many projects and many fields.

3- Reduced Inequality Among Countries

- The use of modernized robots in industry in Egypt and other developing countries will improve its reputation and position among other countries.

رؤية ورسالة المعهد

Vision:

The Alexandria Higher Institute of Engineering and Technology, as an educational and research institution, aspires to achieve excellence and leadership in engineering education at all its local, regional, and international levels. This is to keep pace with engineering developments and develop the educational and professional capabilities of its students for community service and environmental development.

رؤية المعهد:

يتطلع معهد الإسكندرية العالى للهندسة والتكنولوجيا كمؤسسة تعليمية وبحثية إلى تحقيق التميز والريادة فى التعليم الهندسى على كافة مستوياتها المحلية والإقليمية والدولية لمواكبة التطورات الهندسية وتنمية قدرات طلابها التعليمية والمهنية من أجل خدمة المجتمع وتنمية البيئة.

Mission:

The Alexandria Higher Institute of Engineering and Technology seeks to prepare distinguished engineers capable of innovation, technological creativity, and competition in the job market by offering distinguished programs in engineering specializations. This keeps pace with scientific and research progress and upgrading professional performance in line with the developments of the times to achieve sustainable development for community service and environmental development.

رسالة المعهد:

يسعى معهد الاسكندريه العالي للهندسة والتكنولوجيا الي إعداد مهندسين متميزين قادرين علي الإبتكار والإبداع التكنولوجي والمنافسة في سوق العمل من خلال تقديم برامج متميزة في التخصصات الهندسية التي تتواكب مع التقدم العلمي والبحثي والإرتقاء بالأداء المهني بما يتماشي مع مستجدات العصر لتحقيق التنمية المستدامة لخدمة المجتمع وتنمية البيئة.

رؤيـــة ورسالة قسم هـندســة الميكاترونيات

Vision:

The department's vision is represented in the absolute practical and scientific leadership of the department's graduate by keeping pace with modern technical developments in the fields of mechanical, electrical, electronic and computer engineering to have an effective role in community service and environmental development.

رؤية قسم الميكاترونيات :

تتمثل رؤية القسم فى الريادة العملية والعلمية المطلقه لخريج القسم من خلال مواكبة التطورات التقنيه الحديثه فى مجالات الهندسة الميكانيكية والكهربيه والإلكترونيه والحاسوب ليكون له دور فعال فى خدمة المجتمع وتنمية البيئة

Mission:

The mission of the Department of Mechatronics Engineering is to prepare engineers capable of innovation and technological creativity and to play an effective role in building the national knowledge society by providing a scientific environment that keeps pace with the continuous development of mechatronics engineering sciences, which is concerned with methods of designing and building modern mechatronics systems, both mechanical, electrical and electronic, including computer control parts and systems. Micro electromechanical.

رسالة قسم الميكاترونيات :

تتمثل رسالة قسم هندسة الميكاترونيات فى إعداد مهندسين قادرين على الإبتكار والإبداع التكنولوجى ولعب دور فعال فى بناء مجتمع المعرفه الوطنى من خلال توفير بيئة علمية تواكب التطور المستمر لعلوم هندسة الميكاترونيات والتى تعنى بطرق تصميم وبناء نظم الميكاترونيات الحديثه بشقيها الميكانيكى والكهربائى والإلكترونى بما فيها أجزاء التحكم الحاسوبيه والنظم الكهروميكانيكية الدقيقه.

Declaration

This report has been prepared based on our own project. Where other published and unpublished source materials have been used, these have been acknowledged.

نقر نحن طلاب مشروع (AMR) الموقعين أدناه أنه تم اتباع قواعد وقوانين الملكية الفكرية

Student Name: **Signature:**

1- ……...

2- ……...

3- ……...

4- ……...

5- ……...

6- ……...

7- ……...

8- ……...

9- ……...

Date of Submission:

Acknowledgements

We would like to express our profound gratitude to Allah for granting us the ability to successfully complete this project. We recognize and appreciate the guidance and blessings received throughout our journey.

We extend our heartfelt thanks to **the Academy of Scientific Research and Technology (ASRT)** for their generous financial support. Their contribution has been instrumental in bringing this project to fruition and has enabled us to pursue our research goals.

Furthermore, we sincerely appreciate the guidance and mentorship of our supervisors, Dr. Amr Abdelfattah, Dr. Amr Nasr, and Dr. Lamiaa Abdallah. Their unwavering belief in our capabilities has shaped this book, and we are grateful for the opportunities they provided us to evolve and grow as researchers.

We would also like to acknowledge the contributions of all individuals who have supported us directly or indirectly. Their assistance, insights, and encouragement have significantly enriched the content of this book.

Lastly, we express our heartfelt gratitude to our families and friends. Their unwavering support, understanding, and patience have been crucial to our pursuit of this project. We are grateful for their love, encouragement, and belief in our abilities.

To all those who have played a part in making this book possible, we extend our sincerest appreciation. Your support and involvement have been integral to our success, and we are truly grateful for your contributions.

Contents

List Of Figures:

List of Tables:

Abstract

An Autonomous Mobile Robot (AMR) designed to deliver packages in indoor warehouses with a focus on modularity and human-robot interaction. The AMR is equipped with a versatile mechanical design that matches the specific requirements of the environment and considers budget constraints. It incorporates a range of sensors, including a camera with depth sensors, to perceive and map the surrounding environment.

The key features of this AMR include:

1. Perception:
    - Mapping and scanning of the environment to create a comprehensive 2D map.
    - Accurate localization of the robot and packages using sensor data and various algorithms.
    - Analysis of gathered data to facilitate efficient navigation and object avoidance.
2. Decision Making:
    - Intelligent path planning and decision-making capabilities.
    - Navigation system designed to ensure safe and obstacle-free movement.
    - Consideration of modularity to accommodate different operational requirements.
3. Navigation and Warehouse Operation:

    Versatile functionality across various applications, including logistics services, electronic commerce, storage, production, manufacturing, healthcare systems, and biotechnology.

    - The robot can adapt to different environments and effectively navigate within any facility.
4. Human-Robot Interaction and Error Handling:

    The AMR incorporates a user-friendly screen, allowing for intuitive human-robot interaction. Users can interact with the robot, input commands, and receive updates on the robot's status and progress.

    - The AMR's software includes error detection and handling mechanisms to ensure smooth operation and efficient performance. In the event of an error or unexpected situation, the robot is capable of detecting the issue and taking appropriate recover actions.
    - The error handling system is designed to be robust, allowing the robot to autonomously recover from common errors or failures. It can reevaluate its surroundings, replan its path, or request assistance from a human operator if necessary.

Project Specification

Objective:

Design and develop an autonomous mobile robot capable of Streamline operations, improve efficiency, and enhance safety in target environments.

Enable the robot to perform tasks such as mapping, navigation, object detection, and human-robot interaction.

1- Target Environments:

Warehouses: Facilitate package delivery, inventory management, and navigation within complex layouts.

1- Mapping: Generate accurate and up-to-date maps of the environment using sensor data and mapping algorithms.

2- Localization: Implement robust localization techniques to determine the robot's position within the environment.

3- Navigation: Plan and execute optimal paths to navigate through the environment while avoiding obstacles.

4- Human-Robot Interaction: Enable seamless interaction between the robot and human operators through intuitive interfaces.

5- Error Handling and Recovery: Implement mechanisms to detect errors and autonomously recover from failures.

2- Hardware Components:

1- Camera with depth sensors (e.g., Kinect 360): Provide visual and depth information for mapping, localization, and object detection.

2- Raspberry pi 4: Act as the secondary controller for sensor integration, data processing, and motor control.

3- Arduino Mega: Act as the secondary controller for IMU

4- Hoverboard motors and controllers: Drive the robot's movement and ensure smooth navigation.

5- Display screen: Facilitate human-robot interaction and provide real-time feedback.

3- Software Components:

- ROS (Robot Operating System): Framework for integrating and controlling various software modules.
    1- Mapping Algorithm (e.g., 2D Octomap): Create accurate and detailed maps of the environment.

    2- Localization Algorithm (e.g., Adaptive Monte Carlo Localization - AMCL): Estimate the robot's pose in the environment.

    3- Navigation Stack: Plan and execute optimal paths based on the map and robot's current position.

4- Safety Considerations:
    1- Implement collision avoidance mechanisms to ensure the robot operates safely in dynamic environments.

    2- Adhere to industry safety standards and guidelines for electrical systems, mechanical design, and software development.

    3- Regularly test and validate the robot's functionality and performance to identify and address any potential safety risks.

5- Project Constraints:
    1- Budget: Optimize the design and material selection to fit within the allocated budget.

    2- Timeframe: Complete the project within the specified timeline, accounting for design, development, testing, and documentation.

    3- Resource Availability: Consider the availability of required components, tools, and expertise during the project's execution.

Introduction

1.1 Preface

Welcome to this academic publication, which presents an in-depth documentation of our graduation project—an autonomous mobile robot designed for various applications, including warehouse operations and healthcare facilities. This book serves as a comprehensive resource, meticulously detailing the mechanical design, electrical system implementation, and software development process. It aims to provide academics, researchers, and professionals in the field of autonomous mobile robotics with a thorough understanding of the project's journey, challenges faced, and innovative solutions devised. By adhering to academic standards, this book contributes to the scholarly discourse surrounding autonomous mobile robotics and its diverse applications in different industries.

1.2 Project Motivation

The motivation behind our project stems from two primary factors: the increasing demand for automation in warehouse operations and the urgent need for contactless solutions in healthcare facilities, particularly during the COVID-19 era.

Warehouses have emerged as crucial components of modern supply chains, necessitating the development of efficient and autonomous solutions to streamline operations. Our autonomous mobile robot plays a pivotal role in enhancing warehouse efficiency, ensuring accurate inventory management, and optimizing package transportation. By automating material handling tasks, our robot offers increased productivity, reduced labor costs, and improved safety in warehouse environments.

In addition to warehouse applications, the ongoing pandemic has underscored the significance of contactless solutions in healthcare settings. Our autonomous mobile robot extends its capabilities to assist healthcare professionals in hospitals, clinics, and other medical facilities. It contributes to infection control measures by providing contactless delivery of medications, medical supplies, and laboratory samples. Furthermore, the robot's remote monitoring capabilities enable healthcare providers to interact with patients while maintaining physical distance, thus reducing the risk of viral transmission.

1.3 Project Advantages

Our autonomous mobile robot project possesses several advantages that make it valuable in both warehouse operations and healthcare facilities, particularly within the context of the COVID-19 era:

- Contactless Operations: The robot's autonomous navigation and delivery capabilities ensure safe and contactless operations, reducing the risk of viral transmission in both warehouse and healthcare environments.
- Efficiency and Accuracy: By leveraging advanced algorithms and sensing capabilities, our robot optimizes workflow, enabling efficient inventory management, precise package transportation, and timely delivery of essential items.
- Labor Optimization: Automation of routine tasks allows human workers to focus on higher-level activities, promoting productivity and reducing labor costs in warehouses. In healthcare settings, it enables healthcare professionals to allocate their time more effectively, prioritizing patient care and critical tasks.

- Enhanced Safety Measures: Our robot incorporates safety features such as obstacle detection and avoidance, promoting a safe working environment. In healthcare settings, it aids in maintaining social distancing protocols and reduces the need for unnecessary physical contact.
- Adaptability and Scalability: Designed with modularity in mind, our robot can be easily adapted and scaled to meet specific requirements in different warehouse and healthcare scenarios. It offers flexibility to accommodate changing needs and future advancements.

1.4 Project Methodology:

Throughout the project's lifecycle, we adhered to a systematic approach, aligning our methodology with academic standards, and addressing the unique challenges posed by the COVID-19 era. We prioritized safety, collaboration, and flexibility in our methodology, ensuring that our robot meets the specific requirements of both warehouse operations settings.

In conclusion, this academic publication chronicles the development journey of our autonomous mobile robot project, designed for warehouse operations and healthcare facilities. It explores the motivations driving our project, including the increasing demand for automation in warehouses and the need for contactless solutions in healthcare settings, particularly during the COVID-19 era. Additionally, it highlights the distinct advantages our robot offers in both warehouses .

emphasizing its potential to enhance operational efficiency, optimize labor utilization, and promote safety. By adhering to academic standards, this book presents a rigorous analysis of the mechanical design, electrical system implementation, and software development of our project. Moreover, it delves into the significance of our project in addressing the specific challenges posed by the COVID-19 era, such as the need for contactless operations and enhanced safety measures in warehouses.

By combining our expertise in mechanical engineering, electrical systems, and software development, we have successfully created an autonomous mobile robot that not only streamlines warehouse operations but also contributes to the safety and efficiency of healthcare facilities during these unprecedented times. The integration of state-of-the-art technologies, careful material selection, and innovative problem-solving approaches have enabled us to develop a robust and adaptable solution that can be customized to fit various industrial settings.

This academic publication incorporates a comprehensive list of references at the end of the book, providing readers with the opportunity to explore the relevant scholarly works and research that informed our project. These references contribute to the academic rigor of this publication and facilitate further exploration and advancement in the field of autonomous mobile robotics.

Lastly, we would like to thank our team members for their dedication, perseverance, and collaborative spirit. This project would not have been possible without their hard work and commitment to excellence. We hope that the knowledge and experiences documented in this book inspire future researchers and practitioners to further explore the potential of autonomous mobile robotics in addressing the challenges of our rapidly evolving industrial and healthcare landscape.

Together, let us embrace the possibilities offered by autonomous mobile robotics and work towards a future where technology enables safer, more efficient, and more sustainable operations in warehouses .

Chapter 2: **Mechanical Design and Implementation**

2.1 **Introduction:**

The physical elements that allow autonomous mobile robots to move, interact with their surroundings, and carry out activities are referred to as mechanical systems. An autonomous mobile robot's mechanical system oversees translating electrical signals into motion. To operate the robot's wheels or other parts, for instance, the motor system transforms electrical energy into rotational movement.

The design of the mechanical system is crucial for the performance and functionality of the robot. The size, weight, and power requirements of the components must be carefully considered to ensure that the robot is able to move effectively and efficiently. The mechanical system must also be durable and robust to withstand the stresses and strains of operation.

Some specific components of the mechanical system in autonomous mobile robots includes:

1. **Material selection:**

   Material selection is a critical aspect of designing mechanical systems for warehouse robots. The materials used in the construction of a robot must be strong and durable, so we also use aluminum chassis to strengthen warehouse robot.

2. **Design:**

   1- **Body design:**

      The design of a warehouse robot must consider factors such as size, weight, mobility, and payload capacity, as well as the need to operate safely around people and other equipment.

   2- **Wheels design:**

      The choice of wheels depends on the terrain the robot will operate on. For example, a robot designed for indoor use may have small wheels, while a robot designed for outdoor use may have larger, more robust wheels.

2.2 **Material selection**

Material selection is an important aspect of product design, as the properties of the materials used can have a significant impact on the performance, durability, and overall quality of the product. When selecting materials, it is important to consider the following properties:

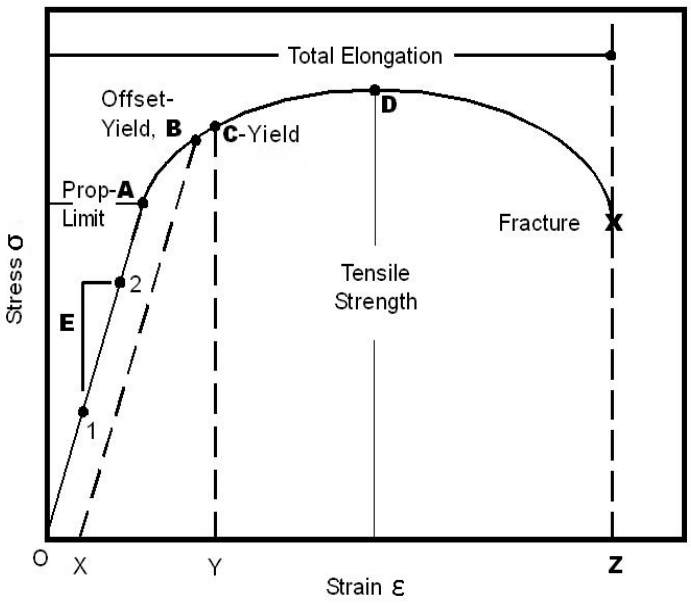

Figure 1 Picture of stress-strain curve

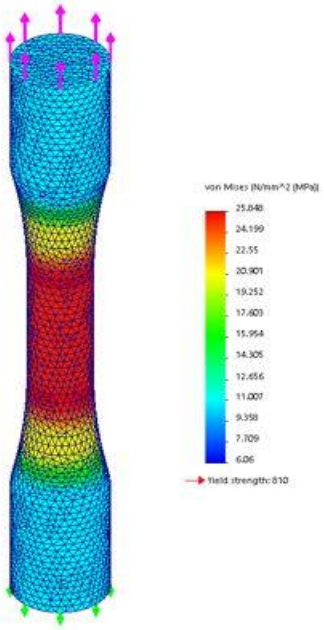

Figure 2 simulation of material stiffness

1. **Strength**: The strength of a material refers to its ability to withstand stress and deformation without breaking or failing. This is an important property to consider in applications where the material will be subjected to high loads or stresses.

2. **Durability**: The durability of a material refers to its ability to withstand wear and tear over time, without degrading or deteriorating. This is an important property to consider in applications where the material will be exposed to harsh environmental conditions, such as extreme temperatures or moisture.

3. **Density**: The density of a material refers to its mass per unit volume. This is an important property to consider in applications where weight is a concern, such as in the aerospace or automotive industries.

4. **Thermal conductivity**: The thermal conductivity of a material refers to its ability to conduct heat. This is an important property to consider in applications where heat transfer is a concern, such as in electronics or thermal management systems.

5. **Electrical conductivity**: The electrical conductivity of a material refers to its ability to conduct electricity. This is an important property to consider in applications where electrical conductivity is required, such as in the electronics or telecommunications industries.

6. **Corrosion resistance**: The corrosion resistance of a material refers to its ability to resist degradation due to exposure to the environment, such as rust or other forms of oxidation. This is an important property to consider in applications where the material will be exposed to moisture or corrosive substances.

Overall, when selecting materials for a product, it is important to consider the specific requirements and properties of the application, as well as the trade-offs between different materials and their properties.

2.2.1 Material available options:

2.2.1.1 Artelon Plastic (POLYAMIDE)

Artelon is a composite material composed of polyethylene fibers with ultra-high molecular weight and a bioabsorbable polymer matrix. Artelon's distinctive features are due to this combination of materials, which includes great strength, flexibility, and resilience to wear and tear. In recent years, Artelon has also gained popularity as a material for use in robotics. Its unique properties make it well-suited to the stresses and strains of a warehouse environment, where robots are required to lift, move, and transport heavy items.

Figure 3 Artelon Board

Physical Properties of Artelon:

- **Flexibility**: Despite its high strength, Artelon is also highly flexible, which makes it an ideal material for use in applications that require a degree of flexibility.
- **Water-resistant**: Artelon is highly resistant to water, which makes it an ideal material for use in applications that will be exposed to moisture or other fluids.
- **Low friction**: Artelon has a low coefficient of friction, which means it can slide easily against other materials without causing damage.
- **Corrosion resistance**: Artelon can still be affected by environmental factors such as UV radiation, temperature, and humidity. These factors can cause the material to degrade over time, leading to a loss of mechanical properties and potential failure of the device in which it is being used.
- **Ductility**: Artelon's ductility is limited compared to metals like steel or aluminum, which can undergo significant deformation without breaking. In some applications, this may limit Artelon's suitability, and other materials may be preferred.

Mechanical Properties of Artelon:

- **Elasticity**: Artelon has high elasticity, which allows it to deform and return to its original shape without permanent deformation.
- **High compressive strength**: Artelon has high compressive strength, meaning it can withstand compressive forces without breaking or deforming.
- **High tensile strength**: Artelon is a highly durable material that can withstand significant amounts of stress and strain without breaking or degrading. This makes it ideal for use in applications where high strength is required.
- **Elongation**: Artelon plastic has high elongation at break, which means it can stretch or deform significantly before breaking. The exact elongation value depends on the specific formulation of Artelon, but typical values are around 100-200%.
- **Fatigue resistance**: Artelon has excellent fatigue resistance, meaning it can withstand repeated loading and unloading cycles without breaking or degrading.

Advantages:

- **Lightweight**: Artelon is a lightweight material, which can be beneficial for robots, as it can reduce the overall weight of the robot and help increase its speed.
- **Flexible**: Artelon is a flexible material that can bend and flex without breaking, which can be useful for robots that require a certain level of flexibility in their movements.
- **Durable**: Artelon is a strong and durable material that can withstand wear and tear over time, making it a good choice for robots that are expected to perform repetitive tasks or operate in harsh environments.
- **Biocompatible**: Artelon is biocompatible, meaning that it is not harmful to living tissue, which can be important for robots that are designed to interact with humans or other living organisms.

Disadvantages:

- **Limited temperature range**: Artelon has a limited temperature range and may not be suitable for robots that operate in extreme temperatures.
- **Limited stiffness**: Artelon has a lower stiffness compared to some other materials, which may limit its use for robots that require high stiffness or rigidity.
- **Cost**: Artelon can be more expensive than some other materials, which may increase the cost of manufacturing robots that use it.

2.2.1.2 Sheet-metal

Sheet metal is a type of metal that is formed into thin, flat sheets or coils. It is a popular material for a wide range of applications, including construction, automotive, aerospace, and manufacturing. sheet metal is a versatile material that offers a range of properties suitable for a wide variety of applications. Here are some of the key properties of sheet metal:

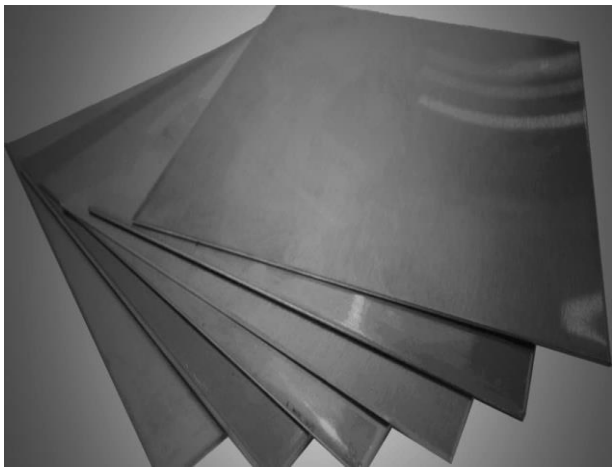

Figure 4 Sheet metal

Physical properties of sheet-metal:

- Density: The density of sheet metal varies depending on the type of metal used, but it is generally higher than that of other materials such as plastics or composites.
- Ductility: Sheet metal is ductile, which means it can be easily bent, stretched, or formed into various shapes without cracking or breaking. This property is important for manufacturing processes that require sheet metal to be shaped into complex geometries.
- Corrosion resistance: Many sheet metal materials, such as stainless steel, aluminum, and copper, are highly resistant to corrosion. This makes them ideal for outdoor or corrosive environments.
- Conductivity: Some sheet metal materials, such as copper and aluminum, are highly conductive to electricity and heat. This property makes them useful in applications such as electrical wiring.
- Appearance: Sheet metal can be finished in a variety of ways to achieve different appearances and textures. It can be painted, polished, or coated to provide a specific finish or color.

Mechanical properties of sheet-metal:

- **Strength**: Sheet metal is strong and can withstand high levels of stress and strain without breaking or deforming permanently. The strength of sheet metal can be improved through various processes such as heat treatment, cold working, and alloying.
- **Elastic modulus**: The elastic modulus of sheet metal is a measure of its stiffness or resistance to deformation. This property is important in applications where dimensional stability is critical.
- **Fatigue strength**: Fatigue strength is the ability of sheet metal to resist failure under repeated or cyclic loading. This property is important in applications that involve dynamic loading, such as in machinery or vehicles.
- **Toughness**: Toughness is the ability of sheet metal to absorb energy without breaking. This property is important in applications that require the material to withstand impacts or sudden shocks.

Advantages:

- **Availability**: Sheet metal is widely available and can be sourced from many suppliers, making it easy to obtain and use in manufacturing.
- **Cost**: Sheet metal is generally less expensive than other materials such as composites or plastics, making it a cost-effective choice for manufacturing robots.
- **Customization**: Sheet metal can be easily cut and shaped into custom shapes and sizes, making it suitable for creating robot components.

Disadvantages:

- **Weight**: Sheet metal can be heavy, which can increase the weight of the robot and limit its speed and manoeuvrability.
- **Corrosion**: Some sheet metal materials, such as iron and steel, are prone to corrosion, which can be problematic in certain environments.
- **Manufacturing complexity**: Sheet metal components can be more complex to manufacture than other materials, requiring specialized equipment and skilled labour.

2.2.1.3 Wood

Wood is a natural material that comes from trees and woody plants. It is composed mainly of cellulose fibres, which give it its strength and stiffness, and lignin, which binds the fibers together. Wood is a renewable resource and has been used for thousands of years for a wide variety of purposes, including building construction, furniture making, paper production, and fuel for heating and cooking. There are many different species of trees that produce wood with a wide range of properties and characteristics, making wood a versatile and valuable material. Wood can be used in the construction of certain types of robots, particularly those that are designed to be lightweight and have a natural appearance. For example, wood can be used to create the body or casing of a robot, as well as other parts such as arms, legs, and joints.

Figure 5 wood paneling

Properties of wood:

- **Durability**: The durability of wood depends on the types and the conditions it is exposed to. Some woods are more resistant to decay and insect damage than others.
- **Moisture content**: It can absorb and release moisture. The moisture content of wood affects its properties, such as strength, stiffness, and dimensional stability.
- Workability: Wood is a relatively easy material to work with using hand or power tools.
- **Density**: The density of wood varies depending on the species, but it ranges from 300 to 900 kg/m3.
- **Hardness:** Hardness is a measure of how resistant a material is to indentation or scratching. The hardness of wood can vary depending on the type. It is softer than most metals.
- **Elasticity**: Elasticity is a measure of how much a material will deform under stress and then return to its original shape when the stress is removed. Wood has high elasticity.
- **Thermal conductivity**: Wood is a good insulator and has low thermal conductivity. Its thermal properties depend on the types, moisture content, and other factors.
- **Fire resistance**: Wood is combustible and can burn, but some types of wood are more fire-resistant than others.

Advantages:

- **Lightweight**: Wood is a lightweight material compared to metals and some other materials, which makes it ideal for constructing lightweight robots that can move quickly and efficiently.
- **Renewable**: Wood is a renewable resource, which means it can be used to construct robots without depleting finite resources.
- **Low cost**: Wood is less expensive than many other materials, which can make it an attractive option for constructing robots on a budget.
- **Easy to work with**: Wood is an easy material to machine and work with using hand or power tools, which can make it easier to construct custom robot parts.

Disadvantages:

- **limited strength**: Wood is not as strong or durable as other materials commonly used in warehouse robots, such as metal or plastic, which could limit its use in applications that require high strength or resistance to wear and tear.
- **Fire hazard**: Wood is combustible and may be a fire hazard in certain applications, such as in a warehouse where flammable materials are.

Overcoming wood defects:

- **Surface treatments**: such as coloring, paint, or varnish, can help protect wood from moisture, UV damage, and wear and tear by spraying the surface of the external robot with insulating material to protect wood, give the robot an aesthetic appearance, and make it more suitable for specific applications.
- **Reinforcement techniques**: When it comes to building a robot, having a sturdy and reliable chassis is crucial. The chassis serves as the foundation of the robot, supporting all its components and providing a stable platform for movement. While there are many varied materials that can be used to construct a robot chassis, aluminum has emerged as a popular choice for many robotics applications. In this article, we will explore the benefits and considerations of using an aluminum chassis in robot installation.

2.2.2 Comparison

	WOOD	SHEET-METAL
WEIGHT	Less weight	Heavy
Formation & Modification	Easy to modify	More complex
Cost	Less cost	Expensive
Effect with load	withstand High load.	High load cause bending
Collision and friction	Long lifetime	Less time

Table 1 material selection comparison

According to the above we selected wood.

2.3 Design

2.3.1 Chassis

2.3.1.1 Load on chassis

In a robot, the load on the chassis refers to the weight or force applied to the robot's framework. The chassis is a critical component of the robot, providing structural support for all its components and enabling it to move and operate effectively.

The amount of load that a robot chassis can manage depends on its design, materials used, and intended application. For example, a small, lightweight robot may have a less robust chassis than a larger, heavier robot that needs to carry more weight or operate in more challenging environments.

Overloading a robot chassis can lead to a variety of problems, including reduced performance, decreased maneuverability, and potential damage to the robot's components. Additionally, excessive loads can cause the robot to become unstable or tip over, potentially causing injury or damage to surround objects.

To avoid overloading a robot chassis, it is important to carefully consider the weight and distribution of the components that will be mounted on the chassis. This includes the weight of the robot's body, motors, batteries, and any additional payloads or attachments. It is also important to consider the robot's operating environment, such as the terrain it will be traversing or the obstacles it may encounter.

Because aluminum fits requirements, we used it for chassis.

Benefits of Aluminum Chassis in Robot Installation:

1. **Lightweight**: One of the most significant benefits of aluminum chassis is its low weight. This makes it an excellent choice for robots that need to be agile and fast, as the reduced weight allows for more rapid acceleration and maneuverability.

2. **Strength**: Despite its low weight, aluminum is an extraordinarily strong and durable material. This makes it an ideal choice for robots that need to withstand rough terrain or harsh environmental conditions.

3. **Corrosion resistance**: Aluminum is highly resistant to corrosion, making it an excellent choice for robots that will be operating in wet or humid environments.

4. **Easy to machine**: Aluminum is a soft metal, which makes it easy to machine and fabricate. This means that it is straightforward to create custom chassis designs or modify an existing design to fit specific requirements.

5. **Aesthetic appeal**: Aluminum has a sleek and modern appearance, which can be an important consideration for robots that are intended for use in public-facing environments.

The following figure show simulation of load on chassis by solid works :

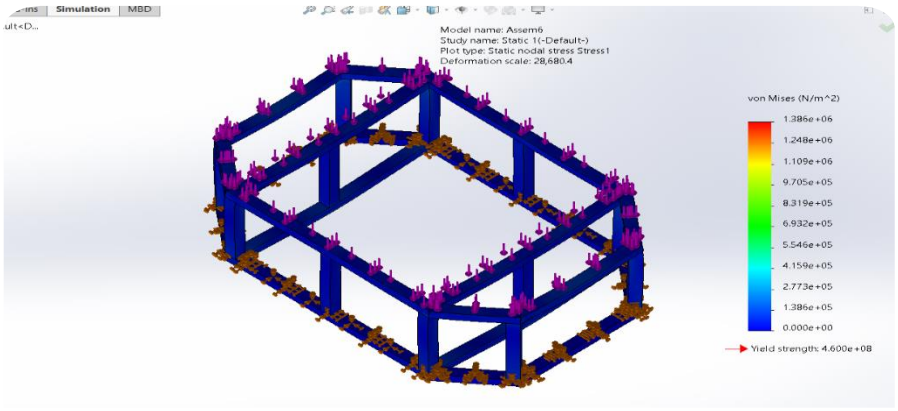

Figure 6 When load is 70 kg

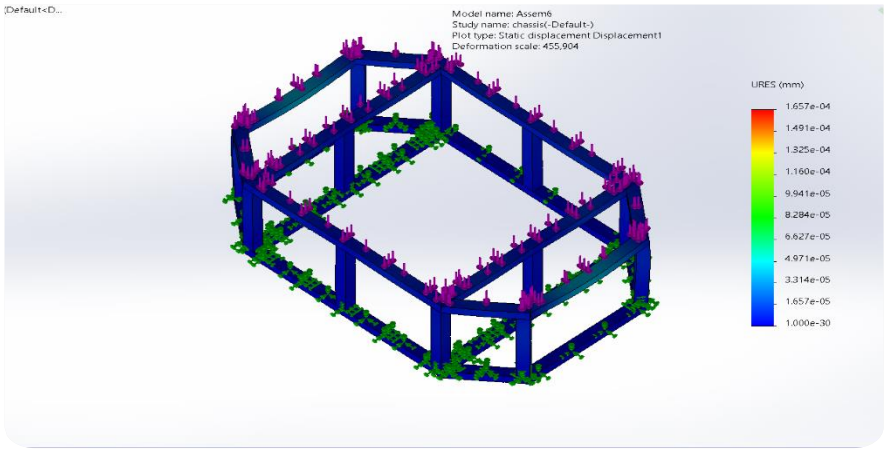

Figure 7 When load is 100 kg

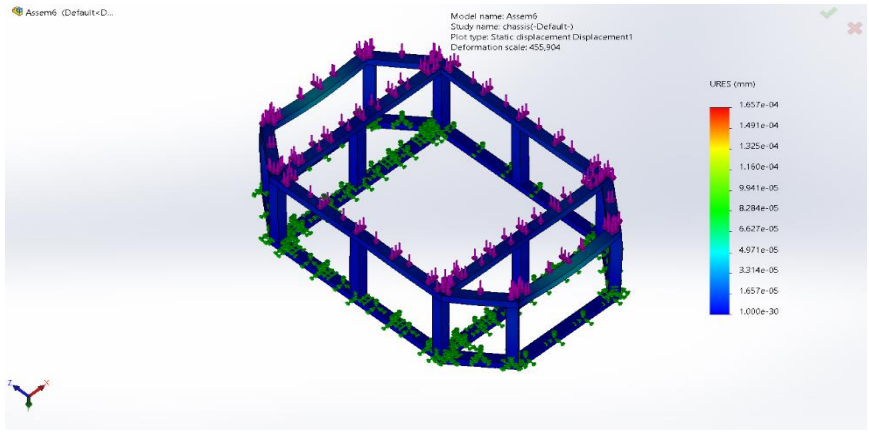

Figure 8 When load is 180 kg

2.3.1.2 Reinforcements option

1. Quad-shaped chassis:

A quad-shaped chassis is essentially a four-sided frame that is designed to provide maximum stability and strength. This type of chassis is commonly used in off-road vehicles, such as ATVs, UTVs, and dune buggies, as well as in industrial and agricultural machinery.

One of the key benefits of a quad-shaped chassis is its ability to distribute weight evenly across all four sides. This helps to prevent tipping or rolling, which can be a major safety concern in off-road vehicles that are designed for extreme terrain. By providing a stable and secure base, a quad-shaped chassis allows drivers to push the limits of their vehicles without worrying about losing control.

Another advantage of a quad-shaped chassis is its strength and durability. The four-sided design provides a rigid framework that can withstand the stresses of heavy use and rough terrain. This makes it an ideal choice for industrial and agricultural machinery that needs to be able to handle heavy loads and harsh environments.

2. Hexagonal chassis:

A hexagonal-shaped chassis is essentially a six-sided frame that is designed to provide maximum stability, strength, and versatility. Like the quad-shaped chassis, it is commonly used in off-road vehicles, industrial machinery, and other heavy-duty applications where stability and durability are essential.

One of the key advantages of a hexagonal-shaped chassis is its ability to distribute weight more evenly than a quad-shaped chassis. By adding two additional sides to the frame, a hexagonal chassis can provide even greater stability and support, reducing the risk of tipping or rolling even in the most extreme conditions.

Another benefit of a hexagonal-shaped chassis is its versatility. The six-sided design allows for greater flexibility in the placement of components, such as engines, transmissions, and suspension systems. This can lead to more efficient use of space and better overall performance.

In addition, a hexagonal-shaped chassis can be more aesthetically pleasing than a quad-shaped chassis. The six-sided design provides a unique and modern look that can help vehicles and machinery stand out from the competition.

According to previous features and provide it for better balance so select hexagonal chassis.

2.3.1.3 Chassis manufacturing

1. Aluminum Angles:

Aluminum angles are made from extruded aluminum, which is a process where raw aluminum is forced through a die to create a specific shape. The resulting angles are typically lightweight, yet strong and durable, making them an ideal choice for chassis manufacturing.

One of the key benefits of using aluminum angles in chassis manufacturing is their strength-to-weight ratio. Aluminum is a lightweight material, yet it is also incredibly strong and durable. This means that chassis made from aluminum angles can be strong and stable, while also being lightweight and easy to handle.

In addition, aluminum angles are highly corrosion-resistant, which is an important consideration for chassis manufacturing. Vehicles and machinery that operate in harsh environments, such as off-road vehicles or marine vessels, are exposed to corrosive elements that can cause rust and degradation over time. Using aluminum angles can help to prevent this, as aluminum is highly resistant to corrosion and will not rust.

Despite their many benefits, there are some limitations to using aluminum angles in chassis manufacturing. For example, they may not be suitable for applications where extreme strength or rigidity is required, as they may not be able to withstand the stresses involved. Additionally, aluminum angles may not be cost-effective for all applications, as they can be more expensive than other materials such as steel.

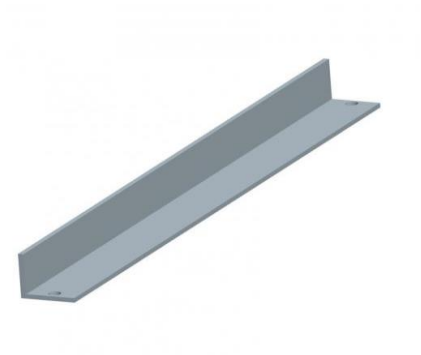

Figure 9 Aluminum Angles

2. Aluminum Square Tube:

One of the key advantages of using square aluminum tube in chassis manufacturing is its increased strength and rigidity. The square shape provides greater surface area for welding, which can increase the overall strength and stability of the chassis. This makes it a better choice for applications where extreme strength and rigidity are required, such as in heavy-duty machinery or high-performance vehicles.

Additionally, square aluminum tubes are highly versatile and can be used in a range of applications. It comes in a variety of sizes and thicknesses, making it easy to find the right dimensions for a specific application. This flexibility more than aluminum angles makes it an ideal choice for custom chassis manufacturing, as it can be tailored to meet the exact needs of the vehicle or machinery being built.

Another advantage of using square aluminum tube is its improved aesthetic appeal. While aluminum angles can be functional, they may not always provide the desired look or finish for a chassis. Aluminum square tube, on the other hand, can provide a clean and modern look more than aluminum angles, that is often preferred in high-end vehicles and machinery.

Figure 10 Aluminum Square Tube

According to the above we selected Aluminum Square Tube.

The following figure show chassis designed by solid works:

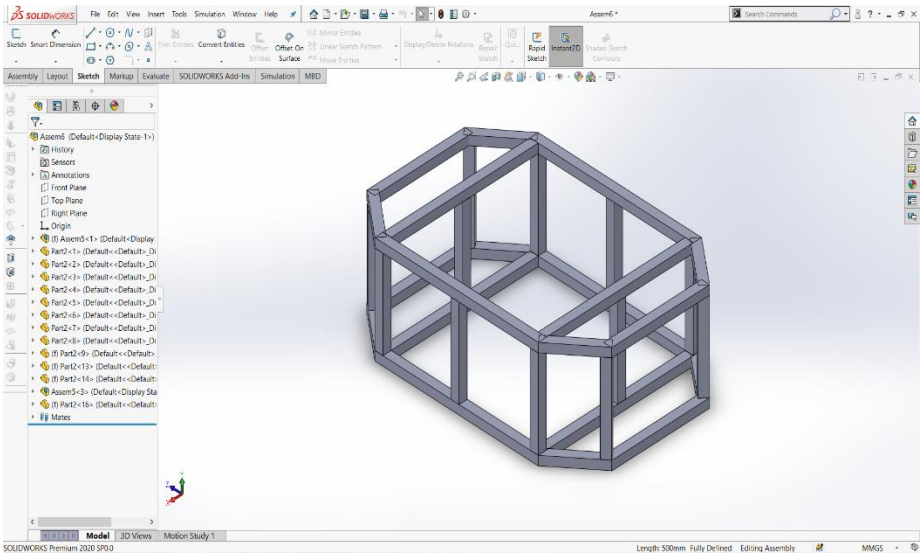

Figure 11 chassis side view

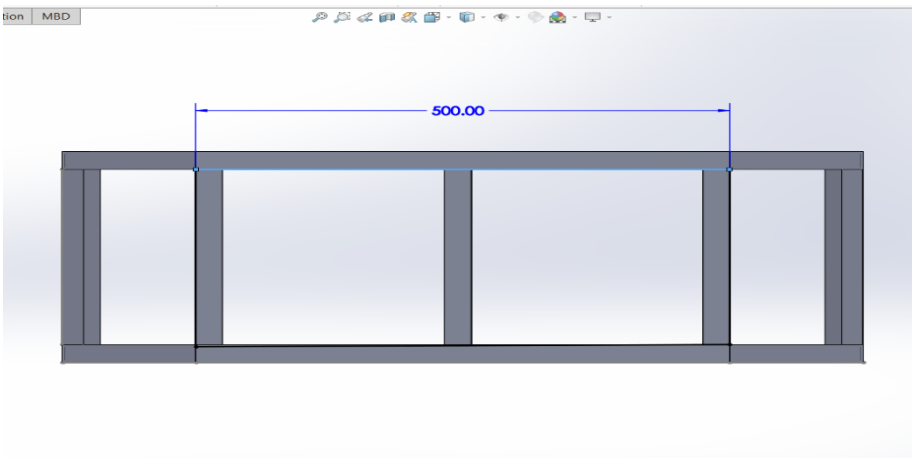

Figure 12 Mechanical system chassis

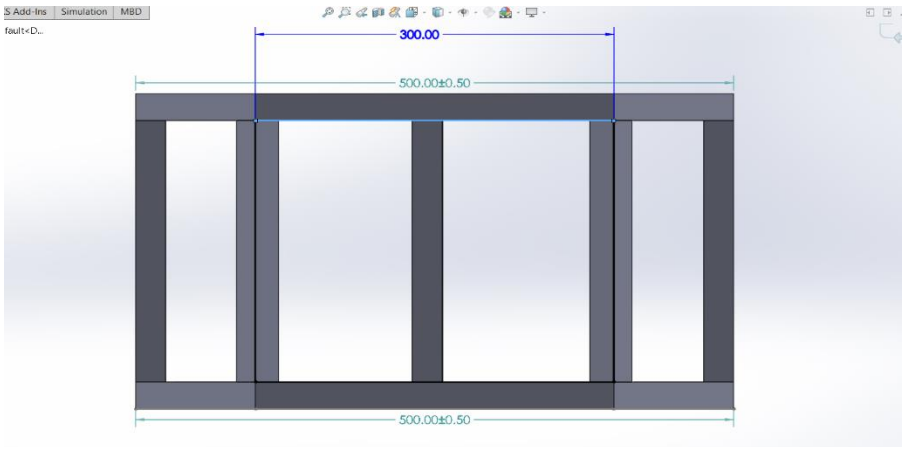

Figure 13 chassis plane view

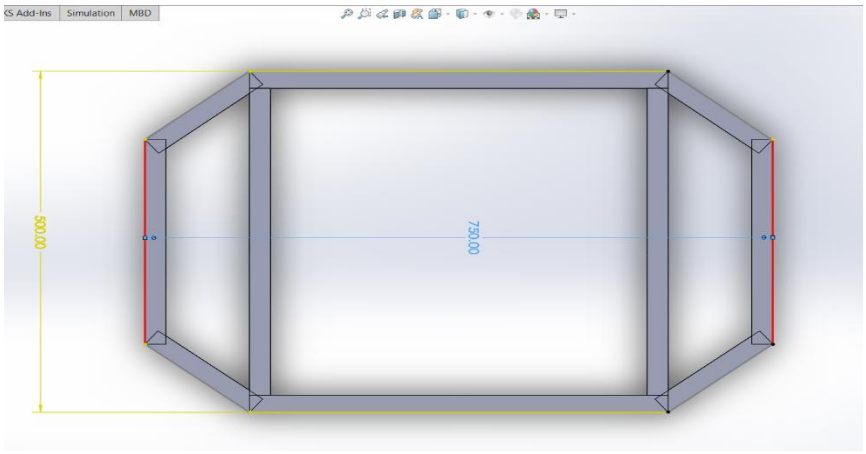

Figure 14 chassis front view

2.3.2 Wheel

Warehouse robots can use several types of wheels depending on their design and application. Here are some common wheel types used in warehouse robots:

2.3.2.1 Caster wheels:

These are small, swivelling wheels that are commonly used for low-speed applications where maneuverability is important. Caster wheels are often found on automated guided vehicles (AGVs) used for material handling in warehouses.

2.3.2.2 Omni-directional wheels:

These wheels allow a robot to move in any direction, making them useful for navigating tight spaces and avoiding obstacles. They are often used in AGVs and other warehouse robots.

2.3.2.3 Drive wheels:

These are the primary wheels that provide traction and movement to the robot. They are typically powered by motors and can be designed for specific applications, such as climbing inclines or rough terrain.

2.3.2.4 Hoverboard wheels:

Some warehouse robots use hoverboard wheels, which are like the wheels found on self-balancing hoverboards. These wheels use gyroscopic stabilization to provide smooth and stable movement, making them useful for applications where precision and stability are important. Also known as self-balancing scooter wheels, have several advantages for use in warehouse robots:

1. **Stability**: Hoverboard wheels use gyroscopic stabilization to maintain balance, which provides an elevated level of stability and control. This makes them useful for applications where precise movement and positioning are required.

2. **Maneuverability**: Hoverboard wheels allow for smooth and precise movement in any direction, which makes them ideal for use in tight spaces or around obstacles.

3. **Durability**: Hoverboard wheels are designed to withstand significant wear and tear, making them a reliable choice for use in warehouse robots that may be subjected to frequent use and rough terrain.

4. **Low profile**: Hoverboard wheels have a low profile, which allows for a lower overall robot height. This can be useful in applications where the robot needs to navigate through low-clearance areas or under obstacles.

5. **Reduced noise**: Hoverboard wheels produce less noise than some other types of wheels, which can be an advantage in applications where noise levels need to be minimized.

6. **Low maintenance**: Hoverboard wheels require minimal maintenance, which can reduce the overall cost of ownership for warehouse robots that use them.

The following figures show the component of hoverboard wheel:

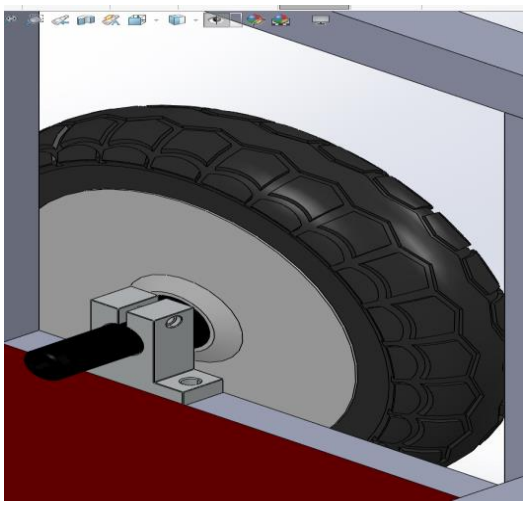

Figure 15 hub motor shaft bracket

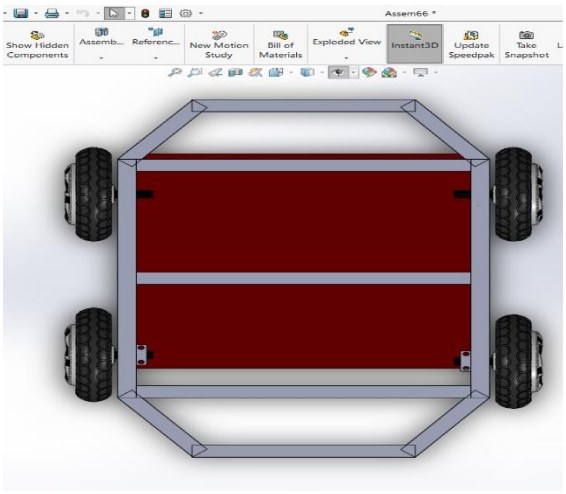

Figure 16 wheel hub

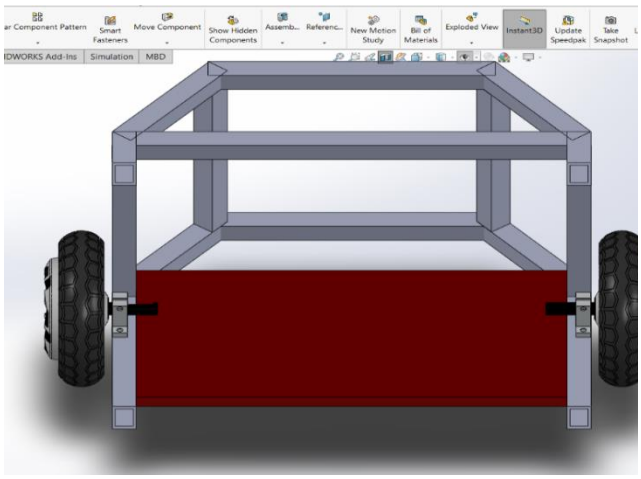

Figure 17 motor wheel

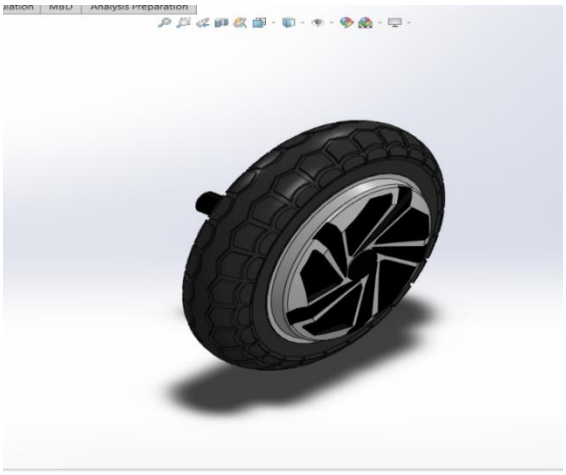

Figure 18 wheelbase assembly

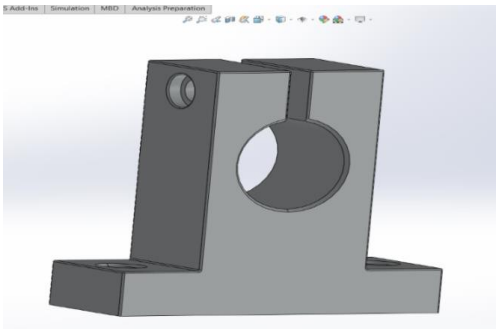

Figure 19 wheelbase plan view

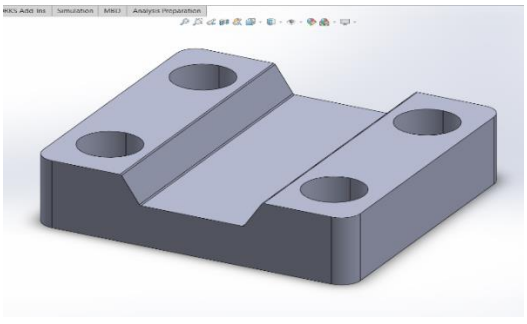

Figure 20 Wheel assembly parts

2.3.3 Screen holder

A 7-inch screen holder in a warehouse robot is typically used to hold a 7-inch screen that robot uses to navigate and communicate with the warehouse management system. The holder is usually mounted on the robot's body and is designed to securely hold the device in place while the robot moves around the warehouse.

The screen holder may be adjustable, allowing the operator to position the device at the optimal height and angle for viewing. It may also be designed to protect the device from damage due to vibrations or impacts during operation.

Having a screen holder on a warehouse robot allows the operator to monitor the robot's progress, receive alerts and notifications, and adjust the robot's route or task list as needed. This can help to improve efficiency and accuracy in the warehouse, as well as reduce the risk of errors or accidents.

2.3.3.1 Reason why you should have screen holder

There are several advantages of using a 7-inch screen holder, especially in the context of a warehouse robot. Here are some of the key advantages:

1. **Hands-free operation**: A 7-inch screen holder allows the operator to use the screen hands-free, as it is securely mounted on the robot's body. This allows the operator to focus on other tasks, such as monitoring the robot's progress or performing other warehouse tasks.

2. **Improved efficiency**: With a 7-inch screen holder, the mobile device is always within easy reach, allowing the operator to quickly access important information and adjust as needed. This can help to improve overall efficiency and productivity in the warehouse.

3. **Better accuracy**: By using a device with a 7-inch screen holder, the operator can access real-time information about the warehouse, such as inventory levels or item locations. This can help to improve accuracy in tasks such as picking and packing orders.

4. **Reduced risk of damage**: A 7-inch screen holder is designed to securely hold the mobile device in place, reducing the risk of damage due to drops or impacts. This can help to prolong the life of the screen and reduce repair costs.

5. **Flexibility**: A 7-inch screen holder may be adjustable, allowing the operator to position the screen at the optimal height and angle for viewing. This can help to reduce eyestrain and improve overall comfort for the operator.

•The following figures show the holder:

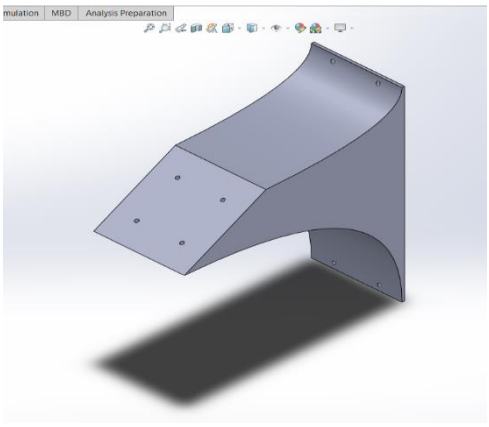

Figure 21 holder side view

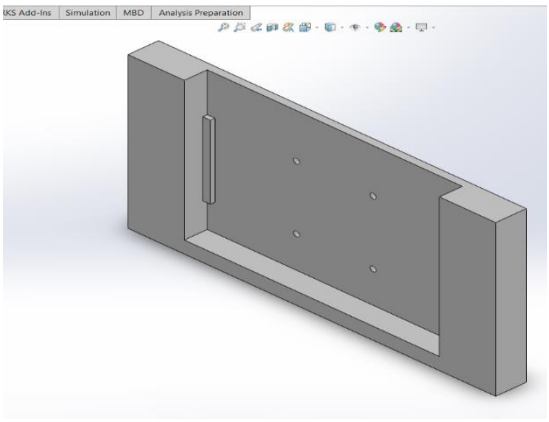

Figure 22 bracket

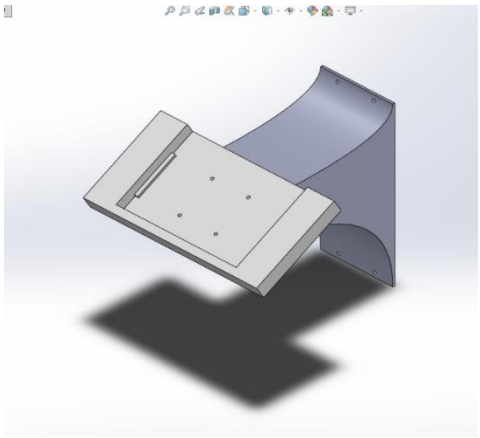

Figure 23 holder assembly

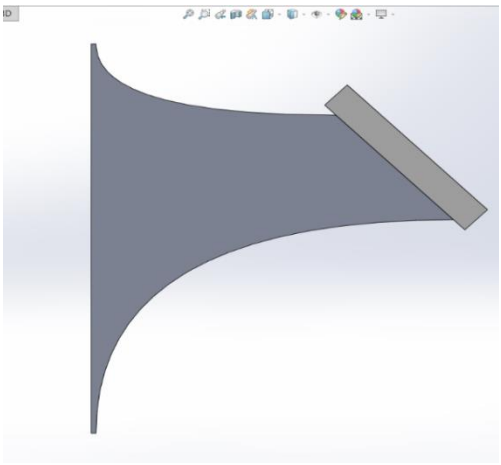

Figure 24 holder

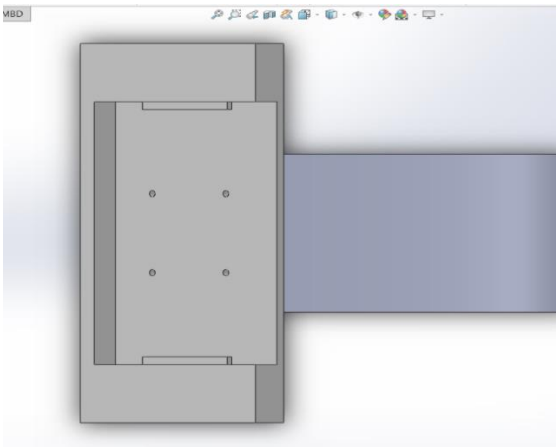

Figure 25 holder assembly

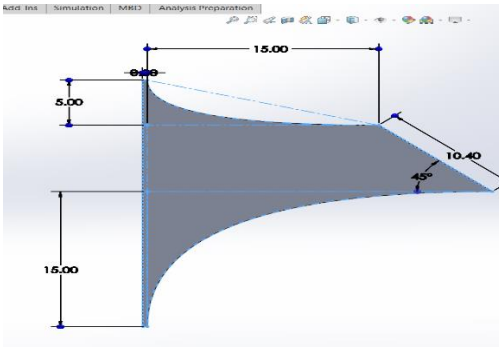

Figure 26 holder assembly with dimension

2.3.4 Robot design

2.3.4.1 Amazon warehouse robot

Kiva robots, which are also known as Amazon Robotics, are used to transport goods in Amazon's warehouses. These robots are designed to move around the warehouse floor and transport shelves of products to human workers for order fulfilment.

While these robots have been successful in improving efficiency and reducing labour costs, they are not without their challenges. Here are some potential defects or issues that could arise with the use of Kiva robots:

1. **Limited flexibility:** Kiva robots are designed to work in a specific environment and may not be easily adaptable to changes in the warehouse layout or inventory needs.

2. **High implementation costs**: Implementing Kiva robots can be expensive, as it requires significant investment in equipment, software, and infrastructure.

3. **Maintenance and repair costs**: Like any mechanical system, robots require regular maintenance and repair, which can be costly and time-consuming.

4. **Technological limitations**: Robots may not be able to replicate the decision-making and problem-solving capabilities of human workers, which could limit their effectiveness in certain tasks.

3.4.2 The proposed warehouse robot

The robot is more than just a machine that moves shelves around, and the worker is the one who decides what he wants. Through the use of the screen built into the design, the person working with the robot will be able to understand what the robot specifically wants, which will help to speed up the process and increase system productivity, which boosts overall efficiency. Next, we go further into our robot's features.

- **Robot specification:**
  - Robot dimension:

    Length 76.6cm * width 51.6cm * height 140 cm

  - Base dimension:

    Length 76.6cm * width 51.6cm * height 30 cm

  - Pole dimension:

    Length 76.6cm * width 51.6cm * height 110 cm

1. **Collaborative approach**: Our robot is designed to work collaboratively with human workers, creating a more flexible and adaptable workforce. This approach allows the robots to leverage the strengths of both humans and robots, resulting in a highly effective and efficient mechanical system.

2. **Flexibility**: Our robot is highly adaptable and can be easily integrated into existing warehouse systems. They can also be reprogrammed to perform new tasks or adjust to changes in the warehouse environment, making them highly flexible in the mechanical system.

3. **Safety features**: Our robot is designed with safety in mind, with advanced safety features such as obstacle detection and collision avoidance. This helps to reduce the risk of accidents or injuries in the warehouse, making them highly safe in the mechanical system.

4. **Scalability**: Our robot is designed to be scalable, allowing them to grow and adapt to changing warehouse needs. This makes them an ideal solution for companies that are looking to optimize their warehouse operations over the long term, making them highly effective in the mechanical system.

Defects in old design:

1. **large design**: This can be particularly problematic for smaller warehouses or businesses that are limited in their available space. Warehouse robots require dedicated areas for charging and maintenance, as well as clear pathways for navigation. This can lead to a reduction in the overall usable space in the warehouse, which can impact the efficiency of operations and limit the amount of inventory that can be stored.

2. **lack of cooling capabilities**: Many warehouse robots generate heat during operation, which can lead to overheating and damage to the robot's components. Without adequate cooling, warehouse robots may experience reduced performance, shortened lifespan, and increased maintenance costs. To address this issue, warehouse robot manufacturers may need to design robots with built-in cooling systems or explore alternative cooling solutions, such as fans or liquid cooling.

3. **Weight and balance**: Robot designs must consider the weight and balance of the robot, as they can impact the robot's stability and performance. If the robot is too heavy or unbalanced, it may be difficult to control or maneuver.

4. **Mechanical components**: The mechanical components of a robot, such as joints, linkages, and motors, must be designed to withstand the stresses of repeated use. If these components are not properly designed or manufactured, they may wear down or break prematurely.

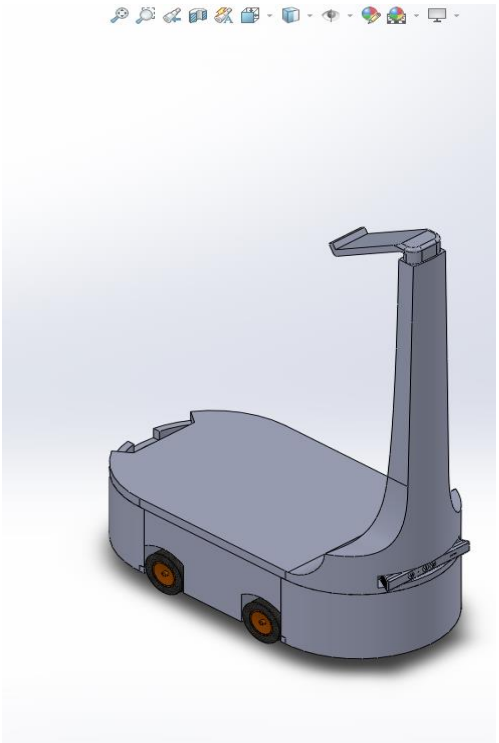

Figure 27 old robot design side view

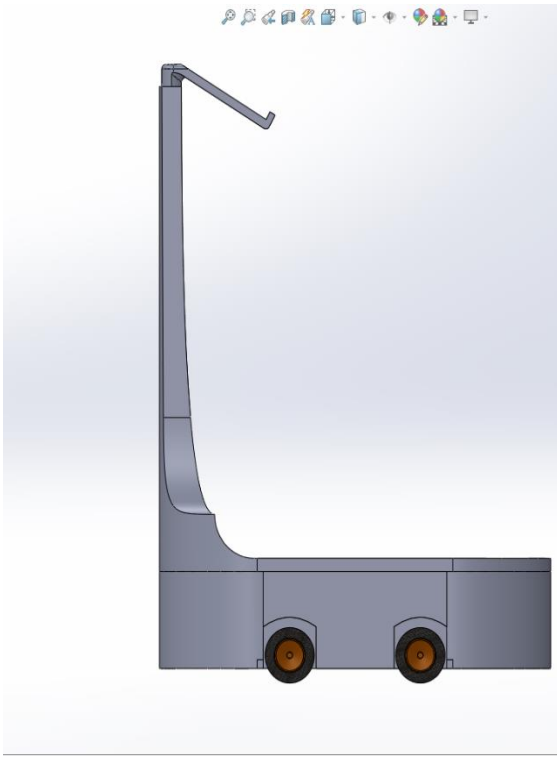

Figure 28 old design robot assembly

How to overcome defects that in old design:

1. **Overcome large design:** One potential solution for overcoming the challenges posed by the large design of warehouse robots is to decrease the size and design of the robot to better fit the warehouse environment. This can be achieved through a number of strategies, including:

I. **Customization**: Warehouse robots can be customized to fit the specific needs of a business and the dimensions of the warehouse. By working with a manufacturer to customize the design of the robot, businesses can create a system that fits their unique requirements and minimizes the impact on their available space.

II. **Streamlined design**: Some warehouse robot manufacturers offer robots with a more streamlined design that takes up less space in the warehouse. By choosing robots with a smaller and more compact design, businesses can reduce the overall footprint of their robot system and create more usable space in the warehouse.

III. **Modular design**: A modular design allows businesses to add or remove components from the robot, making it easier to customize the robot's size and design to fit the warehouse environment. For example, a business may add or remove wheels or other components to adjust the height or width of the robot and make it better suited to the warehouse space.

IV. **Hybrid workforce**: A hybrid workforce that combines human workers and robots can help to reduce the overall number of robots needed and create more space in the warehouse. By leveraging the strengths of both human workers and robots, businesses can create a more efficient and effective workforce that is better suited to the available space

2. Overcome lack of cooling

Creating holes in the design of warehouse robots can be a potential solution for addressing the lack of cooling capabilities. By creating ventilation holes in the robot's design, businesses can improve the airflow and help to dissipate heat generated during operation.

There are several considerations to keep in mind when creating ventilation holes in warehouse robots.

First, the holes must be strategically placed to ensure that they do not compromise the structural integrity of the robot or impact its performance. The holes should be placed in areas where heat is generated, such as near the robot's motor or battery, to improve the airflow and cooling.

Second, the size and shape of the holes must be carefully considered. The holes should be large enough to allow for adequate airflow, but not so large that they compromise the robot's design or allow debris or dust to enter the robot's internal components.

Finally, businesses should consider the impact of creating ventilation holes on the robot's safety features. For example, creating holes near the robot's sensors or cameras may impact its ability to detect obstacles or avoid collisions.

3. Overcome Weight and balance defect:

Weight and balance defects in warehouse robots can be a significant challenge, as they can impact the robot's ability to move and navigate through the warehouse. To overcome this challenge, businesses can explore several strategies, including:

I. **Reducing weight**: One potential solution is to reduce the weight of the robot by using lighter materials or by removing unnecessary components. By reducing the weight of the robot, businesses can improve its balance and maneuverability, making it easier to navigate through the warehouse.

II. **Reconfiguring the robot**: Another potential solution is to reconfigure the design of the robot to improve its weight distribution. For example, businesses may adjust the placement of the robot's battery or other components to improve its balance and reduce the risk of tipping.

4. Overcome mechanical component defects:

Mechanical component defects in warehouse robots can be a significant challenge, as they can impact the robot's ability to operate effectively in the warehouse. To overcome this challenge, businesses can explore several strategies, including:

I. **Regular maintenance and inspections**: Regular maintenance and inspections can help to identify potential mechanical component defects before they become a problem. By scheduling regular maintenance and inspections, businesses can ensure that their robots are operating at peak performance and address any issues with mechanical components as they arise.

II. **Upgrading components**: Upgrading components can help to improve the overall performance and reliability of the robot. For example, businesses may choose to upgrade the robot's motors or sensors to improve their efficiency or accuracy.

III. **Implementing a fleet management system**: A fleet management system can help to optimize the use and movement of warehouse robots, reducing the impact of mechanical component defects. This may include using algorithms to optimize robot routes and tasks, as well as scheduling robots for maintenance and repairs to ensure that they are operating at peak performance.

IV. **Using predictive maintenance**: Predictive maintenance involves using data and analytics to predict when mechanical component defects are likely to occur, allowing businesses to take preventative measures before they become a problem. This may include monitoring the robot's performance in real-time and using machine learning algorithms to predict when specific components may require maintenance or replacement.

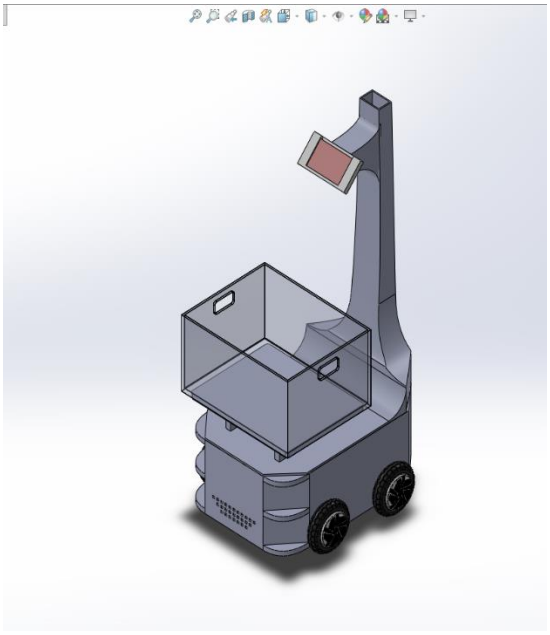

Figure 29 new robot design plan view

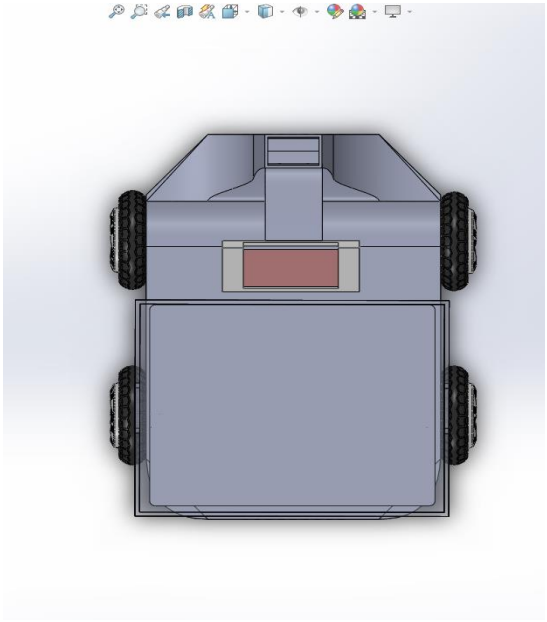

Figure 30 new robot design assembly

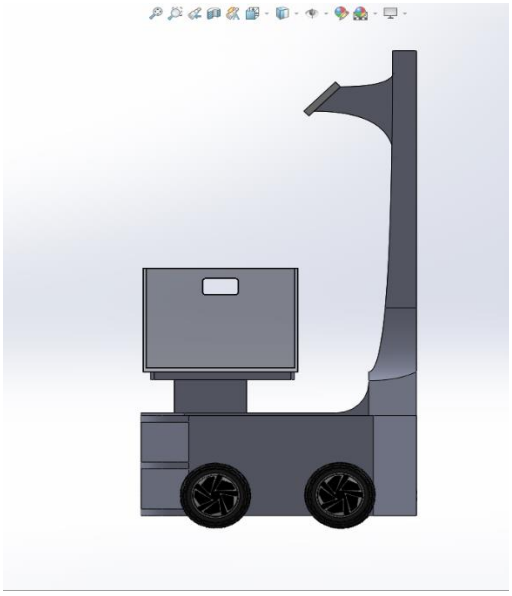

Figure 31 new robot design side view

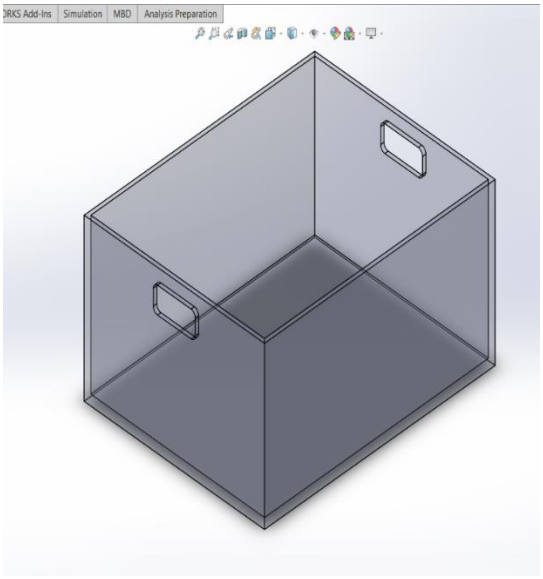

Figure 32 Basket

2.4 Manufacturing

2.4.1 Inceptive Overview

Manufacturing of robots has become increasingly important in recent years, as the demand for automation and robotics in various industries has grown. The process of manufacturing a robot involves many different steps, from design and prototyping to production and testing. In this article, we will explore the manufacturing process for robots and the key considerations that manufacturers must take into account to produce effective and reliable robotic systems.

The first step in manufacturing a robot is the design and prototyping phase. This involves designing the mechanical and electrical components of the robot, as well as the software and control systems that will enable it to function. During this phase, manufacturers must consider factors such as the intended use of the robot, the environment in which it will operate, and the necessary safety features to ensure safe operation.

Once the design is complete, the next step is to produce the physical components of the robot. This involves manufacturing the various mechanical components such as the chassis, arms, and joints, as well as the electrical components such as motors, sensors, and control systems. Manufacturers must ensure that each component is produced to the appropriate specifications and tolerances to ensure proper operation and reliability.

Assembling the robot is the next step in the manufacturing process. This involves connecting the various components of the robot and ensuring that they work together seamlessly. Testing is an essential part of this process, as manufacturers must ensure that the robot operates as intended and that all safety features are functioning properly.

2.4.2 Wood manufacturing in details

2.4.2.1 Manufacturing process limitations

Wood manufacturing is a complex and dynamic industry that plays a critical role in construction, and other related industries. However, as with any manufacturing process, there are limitations and challenges that manufacturers must face to ensure the quality and consistency of their products. In this part of book, we will explore some of the limitations that manufacturers may face during wood manufacturing and discuss strategies for overcoming these challenges.

One of the primary limitations in wood manufacturing is the availability and quality of raw materials. Wood is a natural resource, and the availability and quality of different wood species can vary greatly depending on factors such as climate, location, and harvesting practices. Manufacturers must work closely with suppliers to ensure that they have access to high-quality raw materials that meet their specific requirements.

Another limitation in wood manufacturing is the variability of wood itself. Wood is a natural material, and its properties can vary depending on factors such as species, age, and growth conditions. Manufacturers must account for this variability in their manufacturing processes to ensure that their products are consistent in quality and performance.

Wood manufacturing also poses challenges in terms of machining and finishing. Wood is a relatively soft material, and it can be prone to chipping, splintering, or warping if not handled properly. Manufacturers must use specialized tools and techniques to machine and finish the wood without damaging or distorting it.

Finally, environmental concerns are another limitation that manufacturers face in wood manufacturing. Wood manufacturing can be resource-intensive and can generate waste and emissions that can harm the environment. Manufacturers must implement sustainable practices to minimize their environmental impact and ensure that their products are produced in a responsible and ethical manner.

2.4.2.2 Overcoming Manufacturing process limitations

To overcome the limitations of wood manufacturing, manufacturers can implement several strategies and best practices. Here are some ways to address the limitations discussed in the previous point:

I. **Raw materials**: They can also explore alternative sources of wood, such as reclaimed or recycled wood, to reduce their reliance on virgin materials.

II. **Variability of wood**: To account for the variability of wood, manufacturers can implement quality control measures throughout the manufacturing process. This can include inspecting raw materials for defects, using specialized tools and techniques to machine and finish the wood, and testing finished products to ensure that they meet quality and performance standards.

III. **Machining and finishing**: To overcome the limitations of machining and finishing wood, manufacturers can use specialized tools and equipment that are designed for working with wood. They can also invest in training and education for their employees to ensure **Environmental concerns** that they have the necessary skills and knowledge to handle wood properly.

IV. **Environmental concerns**: To address environmental concerns, manufacturers can implement sustainable practices throughout the manufacturing process. This can include using renewable energy sources, reducing waste and emissions, and using environmentally friendly materials and processes.

2.4.2.3 Final Design after Manufacturing, Assembly and Finishing

Figure 33 Final design after mechanical assembly

Figure 34 Final design after mechanical assembly and electrical components

2.5 Overall challenges in project

There can be challenges in the development and implementation of mechanical systems for warehouse robots related to materials, mechanical components, and specific design features such as aluminum brackets in the chassis. Here are some potential challenges to consider:

I. **Material selection**: Choosing the right materials for components in a warehouse robot can be challenging. Components need to be strong and durable, but also lightweight to minimize the overall weight of the robot. Balancing these requirements can be difficult, and there may be trade-offs between strength, weight, and cost.

II. **Aluminum bracket design**: In the context of a warehouse robot, aluminum brackets in the chassis may present specific challenges related to strength, durability, and weight. The brackets may need to be designed to withstand repeated use without failure, while also minimizing the overall weight of the robot.

III. **Mechanical component design**: The design of mechanical components can also present challenges, particularly when it comes to balancing strength and durability with size and weight. Components may need to be designed to fit into tight spaces or to withstand repeated use without failure.

IV. **Manufacturing and assembly**: The manufacturing and assembly of mechanical systems can be complex, particularly when it comes to ensuring that components are assembled correctly and in the right order. This can be particularly challenging when components are small or intricate, or when assembly is required in a confined space.

V. **Maintenance and repair**: As with any mechanical system, maintenance and repair of a warehouse robot can be challenging. Components may be difficult to access, and repairs may require specialized tools or expertise.

All of the above challenges we faced in the mechanical system part were mentioned how we overcome them earlier in the book.

Chapter 3: **Electric system and control**

3.1 **Preliminary Study**

The electric section of our autonomous mobile robot project plays a crucial role in enabling the seamless operation of the robot, facilitating its autonomous navigation and efficient package transportation. This section encompasses a diverse range of electrical components and systems, including sensors, motor control mechanisms, power management systems, and control units. In this introduction, we will explore the key components utilized, including the MPU6050 sensor, Kinect v1 sensor, voltage sensor, charging system, hover board motor, its controller, Raspberry Pi 4, and Arduino Mega. Sensors are used to detect the robot's surroundings and provide feedback to the control system, which processes the information and generates commands for the motors and other components of the robot to perform specific tasks.

One of the fundamental aspects of our autonomous mobile robot is its ability to perceive and interact with its environment. To achieve this, we employ a combination of sensors. The MPU6050 sensor, integrating gyroscope and accelerometer functionality, allows for accurate motion sensing and orientation tracking. This enables the robot to maintain stability and adjust its movements accordingly. Additionally, the Kinect v1 sensor, utilizing structured light technology, captures detailed 3D information of the surroundings, enhancing the robot's perception and facilitating obstacle detection and avoidance.

To achieve agile and controlled movement, our autonomous robot employs a hover board motor as the primary propulsion mechanism. The hover board motor offers a compact and efficient solution, providing reliable and powerful motion capabilities. It is accompanied by a dedicated motor controller, which regulates the motor's speed, direction, and torque. This combination ensures precise and responsive movement, allowing the robot to navigate through its environment and transport packages with ease.

The intelligence and decision-making capabilities of our autonomous mobile robot are facilitated by the integration of the Raspberry Pi 4 and Arduino Mega control units. The Raspberry Pi 4 handles high-level processing tasks, including sensor data analysis and decision-making algorithms, while the Arduino Mega enables real-time control and interfaces with various sensors and actuators. This combination of control units provides a powerful and flexible platform for executing intelligent control strategies, enabling autonomous navigation and efficient package transportation.

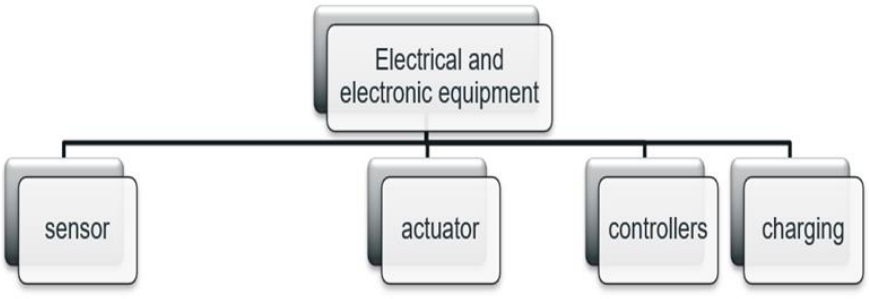

Figure 35: Roadmap for electric section

3.2 Sensors and Perception

A sensor is a device that detects or measures physical properties of the environment or objects and converts them into signals that can be interpreted by humans or other systems. Sensors can detect a wide range of physical properties such as temperature, pressure, humidity, light, sound, motion, position, and many others.

Sensors are used in many applications such as environmental monitoring, industrial automation, medical devices, consumer electronics, and more. For example, a temperature sensor can be used to monitor the temperature of a room or an object, while a motion sensor can detect the movement of people or objects in a specific area. Sensors can be found in everyday devices such as smartphones, cars, and home appliances.

Sensors come in many different types and forms, depending on the physical property they are designed to detect and the specific application they are used for. The graduation project uses many types of sensors, which are explained below.

3.2.1 Depth sensors

They are designed to capture and measure the distance or depth information of objects or surfaces within its field of view. It provides quantitative data about the spatial relationships and distances between the sensor and the objects in the environment. They typically use various techniques to determine the depth or distance to objects. Some common examples include:

- **Time-of-Flight (ToF) Sensors**: ToF sensors emit light, usually infrared, and measure the time it takes for the light to travel to the object and back. By calculating the round-trip time, the sensor can estimate the depth or distance. **The Microsoft Kinect v2 sensor** is a popular ToF sensor used in depth sensing applications.

- **Structured Light Sensors**: Structured light sensors project a known pattern of light, such as grids or dots, onto the scene. The deformation of the pattern on the objects is captured by the sensor, and depth information is calculated based on the distortion. **The Microsoft Kinect v1 sensor**, also known as Kinect for Xbox 360, is an example of a structured light sensor.

- **Stereo Vision Systems**: Stereo vision systems use a pair of cameras, mimicking **human binocular vision**, to capture two slightly offset images of the same scene. By comparing the disparities between corresponding pixels in the two images, depth information can be inferred. Examples of stereo vision systems include the ZED cameras.

- **LIDAR Sensor**: LIDAR sensors use laser beams to measure distances. They emit laser pulses and measure the time it takes for the pulses to return after hitting objects. By scanning the laser beams across the scene, LIDAR sensors generate a detailed 3D point cloud representation that includes depth information.

LIDAR technology systems are based on the number of light beams that can be focused on a surface and the surface scanning mode. Either way, LIDAR systems can be 1D, 2D, or 3D. 2D Lidar will detect an object. 3D Lidar will produce a more detailed point cloud which can be used to determine the shape and depth of the object. 2D sensors are most suitable for performing detection and ranging tasks. 3D LIDAR is most suitable for mapping and detailed analysis.

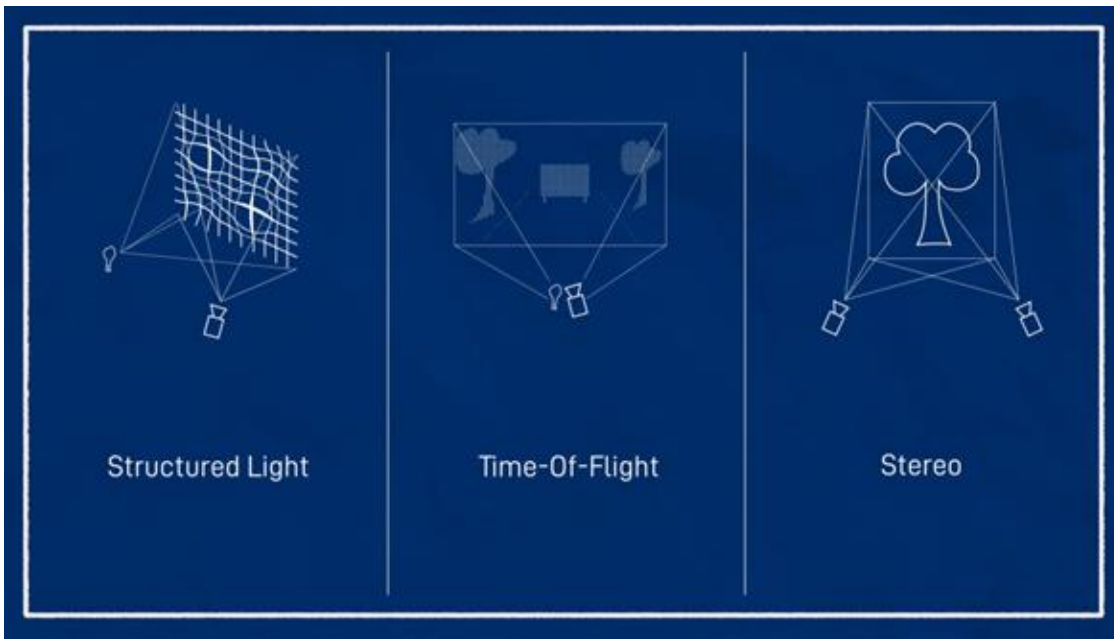

Figure 36: Different types of depth sensor techniques

According to the previous information and the limitations of the budget and the processing power, the **Kinect v1 sensor** seemed to be the best option for the graduation project.

3.2.1.1 Kinect v1

The Kinect v1 sensor, also known as Kinect for Xbox 360 or Kinect for Windows, is a motion-sensing input device developed by Microsoft. It was originally released in 2010 and was primarily designed for gaming purposes. However, it gained popularity in various fields beyond gaming due to its capabilities for depth sensing, skeleton tracking, and voice recognition.

Key features of the Kinect v1 sensor include:

1. **Depth Sensing:** The sensor uses an infrared projector and a depth sensor to capture a 3D view of the environment. This allows it to measure the distance of objects from the sensor and create a depth map.
2. **Skeleton Tracking:** The Kinect v1 sensor has the ability to track up to six people simultaneously. It can detect and track the position of joints in the human body, enabling motion capture and gesture recognition.
3. **RGB Camera:** The sensor includes a standard RGB camera that captures color images of the scene.
4. **Voice Recognition:** The Kinect v1 sensor has built-in microphones and supports voice recognition capabilities. It can interpret voice commands and perform actions based on the recognized commands.
5. **SDK and Development**: Microsoft released a software development kit (SDK) for the Kinect v1 sensor, allowing developers to create applications and experiences using the sensor's capabilities. The SDK provided APIs and tools for accessing depth data, skeletal tracking, and other features.

3.2.1.2 The Operating Principle of The Kinect v1

The Kinect v1 sensor operates based on a technology known as structured light. The work principle of the Kinect v1 involves projecting a pattern of infrared (IR) light onto the surrounding environment and analysing the reflected light to extract depth and 3D information.

The steps of operation of the Kinect v1 include:

1. **IR Pattern Projection:** The Kinect v1 sensor emits a structured pattern of infrared light onto the scene in front of it. This pattern consists of a grid or a series of dots, typically invisible to the human eye. The IR pattern serves as a reference for depth calculation and allows the sensor to gather information about the environment's geometry.

2. **Depth Measurement**: As the IR pattern is projected, it interacts with the objects and surfaces in the scene. The depth sensor in the Kinect v1 measures the distortion and displacement of the projected pattern caused by these objects. By analyzing the distortion of the pattern, the sensor can estimate the depth or distance of each point in the field of view.

3. **Infrared Imaging:** In addition to depth measurement, the Kinect v1 sensor captures an infrared image of the scene. This image is obtained by recording the reflected infrared light from the environment. It provides additional visual information that can be used for various computer vision tasks, such as object recognition and tracking.

4. **Depth Calculation:** Using the information obtained from the depth measurements and the infrared image, the Kinect v1 sensor calculates the 3D position of objects in the scene. By correlating the depth values with the corresponding pixels in the infrared image, the sensor creates a depth map, which represents the distance of each point from the sensor's viewpoint.

5. **Data Processing:** The depth and infrared data captured by the Kinect v1 sensor are then processed and transmitted to a connected computer or device. The sensor may employ various algorithms and techniques, such as point cloud generation or skeleton tracking, to extract meaningful information from the raw data. This processed data can be utilized for applications such as gesture recognition, augmented reality, robotics, and more.

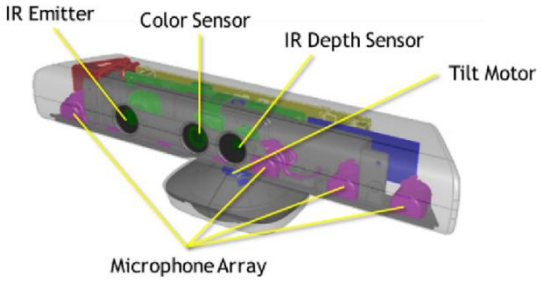

Figure 37: Kinect structure

Figure 38: Xbox 360 Kinect

3.2.1.3 Why Kinect v1

The Kinect v1 sensor has an active developer community with a wealth of resources, tutorials, and open-source libraries available.

Some of the advantages of the Kinect v1 include:

1. **Cost:** The Kinect v1 sensor is generally more affordable compared to the Kinect v2 sensor.

2. **Compatibility:** The Kinect v1 sensor is compatible with a wider range of platforms and operating systems.

3. **Availability:** The Kinect v1 sensor has been on the market for a longer time and is widely available.

In the graduation project, we are utilizing the Kinect v1 sensor to capture data from the external world. The Kinect v1 sensor is a depth-sensing camera that combines RGB imaging and infrared depth mapping to create a detailed understanding of the robot's surroundings. This sensor provides us with valuable information about the environment, including depth perception, object detection, and motion tracking.

The Kinect v1 sensor captures data in the form of a depth map, which represents the distance of objects from the sensor. This depth data is combined with RGB imagery to create a comprehensive representation of the robot's surroundings. By using the Kinect v1 sensor, we can gather essential information about the environment, such as the location of obstacles, the position of objects, and even the movement of people or other entities.

3.2.1.4 Specifications:

- **Voltage:** The Kinect v1 sensor typically operates with a voltage supply of 12 volts (V). This voltage is required to power the sensor and its internal components.

- **Current:** The Kinect v1 sensor has a maximum current consumption of around 1 ampere (A) during normal operation.

- **Power Consumption:** The Kinect v1 sensor has a typical power consumption of around 12 watts (W) during normal operation.

3.2.2 IMU sensor

An Inertial Measurement Unit (IMU) sensor is a device that measures and reports on the orientation, acceleration, and angular velocity of an object. IMUs are commonly used in robotics, drones, virtual reality systems, and other applications that require accurate motion sensing.

An IMU typically contains three types of sensors: accelerometers, gyroscopes, and magnetometers. Accelerometers measure linear acceleration, while gyroscopes measure rotational velocity. Magnetometers measure the Earth's magnetic field and can be used to determine the orientation of the sensor in relation to the Earth's magnetic field.

By combining the data from these sensors, an IMU can provide real-time information about the movement and orientation of an object in three-dimensional space. This information can be used to stabilize a drone, control a robot, or track the movement of a person in a virtual reality system.

There are several types of IMU sensors. Some of the most common types are:

1. Six Degrees of Freedom (6-DoF) IMUs.
2. Nine Degrees of Freedom (9-DoF) IMUs.
3. Ten Degrees of Freedom (10-DoF) IMUs.

MPU6050 is of the **Six Degrees of Freedom (6-DoF)** type, and it is the sensor we chose for the graduation project.

3.2.2.1 IMU vs GPS

Both GPS (Global Positioning System) and IMU (Inertial Measurement Unit) are sensor technologies commonly used for navigation and position estimation. While they serve similar purposes, there are some key differences and areas of overlap between them.

The robot must be able to measure and estimate its pose (position + orientation). In the case of an automobile, GPS (Global Positioning System) is used to estimate its pose. However, GPS cannot be used indoors, and even if it can be used, a GPS with large errors cannot be used for robots. Nowadays, high-precision system such as DGPS is used, but this is also useless indoors as well as being too marker recognition and indoor location estimation have been introduced. However, in terms of cost and accuracy, it is still insufficient for general use. Currently, the most widely used indoor pose estimation method for service robots is dead reckoning, which is a relative pose estimation, but it has been used for a long time and is composed of low-cost sensors and can obtain a certain level of accurate pose estimation result. The amount of movement of the robot is measured with the rotation and the actual travel distance. Therefore, the inertial information from IMU sensor can be used to reduce the error by compensating the error of position and orientation between calculated value and the actual value.

3.2.2.2 MPU6050 sensor

In this project, the MPU6050 plays a crucial role as a sensor for measuring motion and orientation.

Here the functions and capabilities of the MPU6050 in the project:

Motion Sensing: The accelerometer within the MPU6050 measures acceleration in three axes (X, Y, and Z), providing information about the robot's linear movements. This data helps in detecting changes in velocity, direction, and tilt.

Orientation Tracking: The gyroscope component of the MPU6050 measures angular velocity around the three axes. By integrating the gyroscope data over time, you can determine the robot's orientation and track its rotational movements.

Inertial Navigation: By combining the data from the accelerometer and gyroscope, the MPU6050 can provide inertial navigation information. This enables the estimation of the robot's position, velocity, and orientation based on its previous known state and the measured changes in motion.

Sensor Fusion: The MPU6050 allows for sensor fusion techniques, where data from multiple sensors (such as the accelerometer, gyroscope, and potentially magnetometer) are combined to obtain more accurate and robust measurements. Sensor fusion algorithms, such as Kalman filtering, can be applied to fuse the data from the MPU6050 with other sensors to improve the accuracy of pose estimation.

Communication Interface: The MPU6050 typically interfaces with Arduino Mega through standard communication protocols like I2C (Inter-Integrated Circuit). This allows for easy integration and data exchange between the MPU6050 and the Arduino

3.2.2.3 Specifications

- I2C Digital-output of 6 or 9-axis Motion Fusion data in rotation matrix, quaternion, Euler Angle, or raw data format
- Input Voltage: 2.3 – 3.4V.
- Tri-Axis angular rate sensor (gyro) with a sensitivity up to 131 LSBs/dps and a full-scale range of ±250, ±500, ±1000, and ±2000dps.
- Tri-Axis accelerometer with a programmable full-scale range of ±2g, ±4g, ±8g and ±16g
- Digital Motion Processing™ (DMP™) engine offloads complex Motion Fusion, sensor timing synchronization and gesture detection.
- Digital-output temperature sensor.

3.2.2.4 Connections

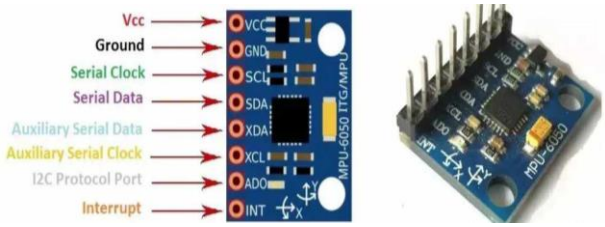

Figure 39: MPU6050

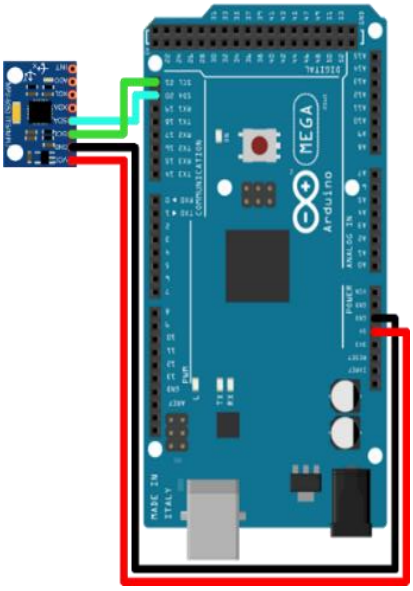

Figure 40: Connection with Arduino

3.2.3 Hall Effect Sensors

A Hall effect sensor is a type of magnetic sensor that detects the presence of a magnetic field. It works based on the principle of the Hall effect, which states that when a magnetic field is applied perpendicular to the direction of electron flow in a conductor, a voltage difference, known as the Hall voltage, is generated across the conductor.

Hall effect sensors consist of a thin sheet of semiconductor material with a small, rectangular strip of metal on top of it. When a magnetic field is applied perpendicular to the metal strip, it causes the electrons to accumulate on one side of the strip, generating a voltage difference between the two sides. This voltage can be measured and used to determine the strength and direction of the magnetic field.

There are generally three main types of Hall Effect sensors based on their output and construction:

Hall Effect Switches: Hall Effect switches are digital sensors that detect the presence or absence of a magnetic field. When a magnetic field is present, the Hall sensor output switches from one state to another, indicating the magnetic field's presence or absence. Hall Effect switches are commonly used in applications such as **proximity sensing.**

Hall Effect Linear Sensors: Hall Effect linear sensors provide a linear analog output that varies in proportion to the strength of the applied magnetic field. These sensors are used to measure linear or angular position, displacement, or fluid levels. Linear Hall Effect sensors are often used in automotive systems, industrial control, robotics, and consumer electronics.

Hall Effect Current Sensors: Hall Effect current sensors are designed to measure the current flowing through a conductor. They operate based on the Hall Effect principle, where the magnetic field produced by the current is measured to determine the current magnitude. Hall Effect current sensors are commonly used in power electronics, motor control, energy monitoring systems, and current measurement applications.

Hall Effect Switches are integrated within the hoverboard brushless DC motor, which is what we used in the project for motor control. This will be explained further in the "Actuators" section.

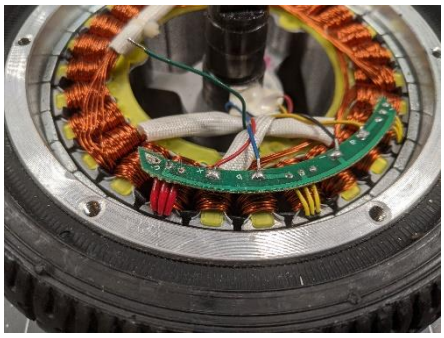

Figure 41: Hoverboard brushless DC motor

Figure 42: Hall effect switches connection within the hoverboard brushless DC motor

3.2.4 Voltage Sensors

A voltage sensor is an electronic device that measures the voltage level of an electrical circuit or system. Voltage sensors can be used in a variety of applications, including power systems, automotive systems, and electronic devices.

Voltage sensors typically use one of two measurement techniques: contact and non-contact. Contact voltage sensors are typically designed to measure the voltage across a specific component in a circuit, such as a resistor or capacitor. These sensors are usually connected directly to the component they are measuring, and the voltage level is determined by the voltage drop across the component.

Non-contact voltage sensors, on the other hand, measure the voltage level in a circuit without having to physically touch the circuit. These sensors typically use magnetic or electric fields to measure the voltage level, and they are often used in high-voltage applications where direct contact is not possible or safe.

The output of a voltage sensor is typically an electrical signal proportional to the measured voltage level. This signal can be in the form of an analog voltage or current, or it can be a digital signal that can be processed by a microcontroller or other electronic device.

The voltage sensor is very essential to track the level of the batteries, which is why the **voltage sensor** was chosen for the graduation project.

3.2.4.1 Specifications

- Supply voltage: 5V.
- Measuring range: from 0V to 25V.
- Output voltage: proportional to input in the range from 0V to 5V.

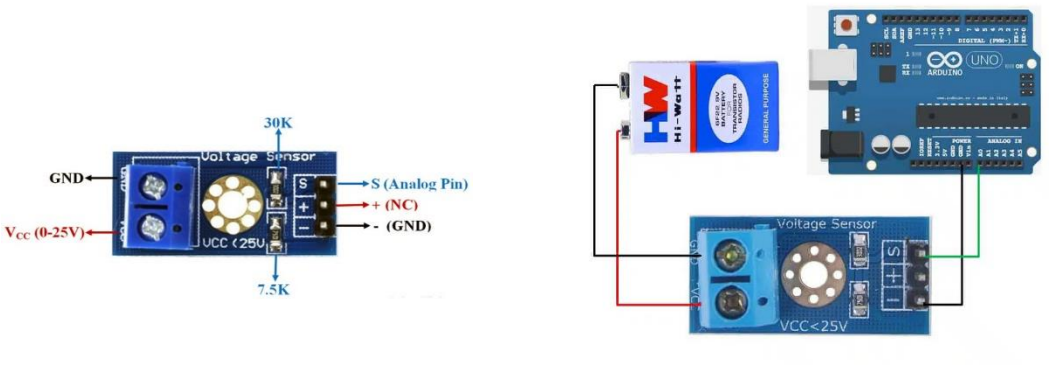

Figure 43: The voltage sensor

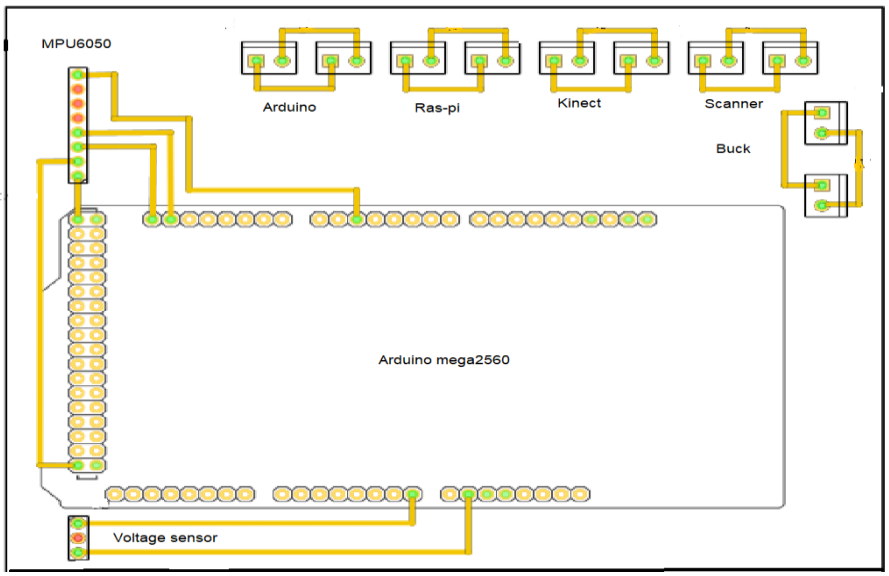

Figure 44: The connection of the voltage sensor within the Arduino Mega2560 shield

3.3 **Actuators**

An actuator is a mechanical or electromechanical device that is used to convert energy into motion. Actuators are used to control the motion of various systems and devices, such as valves, pumps**,** robots, and aircraft control surfaces. They are a critical component in many industrial, commercial, and consumer applications.

Actuators can be classified into several different categories based on their operating principle, one category of which is the **electric actuators**, which is the category that includes motors.

Electric actuators: These actuators use electrical energy to generate mechanical motion, and can be further classified as:

- DC Motors: These actuators use direct current (DC) to generate motion.
- AC Motors: These actuators use alternating current (AC) to generate motion

3.3.1 Calculating Force, Torque, and Power Requirements

These equations allow us to analyse the mechanical performance of our autonomous mobile robot. They help us understand the forces at play, determine the power requirements, and make informed decisions regarding the design, motor selection, and overall efficiency of the robot. By considering these equations during the design phase, we can optimize the robot's performance and ensure its successful operation in transporting packages autonomously.

- Maximum Tractive Force = 0.4 × Total Mass
- Maximum Tractive Force = 0.02 × Total Mass
- Drag Resistance Force = $1.17 \times (\frac{1.2 \times velocity^2}{2}) \times 9$
- Maximum Acceleration Force = Maximum Tractive Force + Maximum Tractive Force + Drag Resistance Force
- Torque (T) = Maximum Acceleration Force × Radius of Wheel
- Power (P) = Maximum Acceleration Force × Velocity

Project Specification and Calculations:

- Mass of chassis = 4 kg
- Mass of body = 14 kg
- Mass of battery = 30kg
- Load of backage = 20 kg
- Total Mass = 68 kg
- Radius of wheel = 4 inch = 0.1016 m
- Velocity of robot = 1.2 m/s
- Time needed = 2 sec
- $F_{\max tractvie} = 0.4 \times 68 = 27.2$ N
- $F_{rolling\ resistance} = 0.02 \times 68 = 1.36$ N
- $F_{drag\ resistance} = 1.17 \times \frac{1.2 \times 1.2^2}{2} \times 9 = 9.1$ N
- $F_{\max accerelation\ for\ the\ robot} = 27.2 + 1.36 + 9.1 = 37.66$ N
- $T = 37.66 \times 0.1016 = 3.83\ N.M$
- $P = 37.66 \times 1.2 = 45.9\ Watt$

3.3.2 Motor Selection

According to the given requirements, the use of a hoverboard motor is a suitable choice for our project. The hoverboard motor is known for its power and efficiency, making it capable of providing the necessary force to propel the robot and transport packages effectively.

The hoverboard motor is designed to deliver high torque, allowing it to handle the weight of the robot and the additional load of the packages. With a powerful motor like this, we can ensure that the robot has sufficient force to move smoothly, even on various terrains or inclines.

3.3.3 DC Motors

There are several types of DC (Direct Current) motors, which can be classified based on their construction, operating principle, and performance characteristics. Here are some of the common types of DC motors:

1. **Brushed DC Motor**: This is the most common type of DC motor, which uses brushes and a commutator to transfer power to the rotor. It has a simple construction and can provide high starting torque but may require periodic maintenance due to wear and tear of brushes.

2. **Brushless DC Motor:** This type of DC motor uses an electronic commutation system instead of brushes, which makes it more reliable and efficient. It is commonly used in applications that require high precision and low noise like **hoverboard motor** and Brushless Outrunner Motor

Features	Brushless(BLDC)	Brushed(BDC)
Cost	High	Low
Commutation	Electronic sensor	Brushed
maintenance	Less required	Periodic
Control	Expensive	Cheaper
Efficiency	High	Moderate
Speed range	Higher	Lower

Table 2: Comparison between brushed DC motor and brushless DC motor

3.3.4 The Brushless DC Motor of the Hoverboard

The hoverboard motor, also known as a brushless DC (BLDC) motor, operates based on the principle of electromagnetic induction and uses electronic commutation to keep it running. It consists of three main components: a stator, a rotor, and hall effect sensors.

The stator is the stationary part of the motor and contains three sets of windings, also known as phases, spaced evenly around its circumference. Each phase consists of multiple coils that are energized in a specific sequence to create a rotating magnetic field.

The rotor is the rotating part of the motor and consists of permanent magnets or electromagnets. The rotor is designed with a specific number of magnetic poles, typically an even number, to create a balanced magnetic field.

To determine the position of the rotor and control the motor's operation, Hall effect sensors are used. These sensors are typically positioned near the stator windings and detect the magnetic field produced by the rotor. The Hall effect sensors provide feedback to the motor controller about the position of the rotor, allowing it to determine when to energize the stator windings.

The motor controller receives input from the Hall effect sensors and uses this information to generate the appropriate signals for the stator windings. It determines which windings need to be energized and in what sequence to create a rotating magnetic field. By energizing the windings in a specific order, the magnetic field produced by the stator interacts with the permanent magnets or electromagnets on the rotor, causing it to rotate.

The motor controller continuously monitors the position of the rotor through the Hall effect sensors and adjusts the timing and sequence of the stator windings' energization accordingly. This precise control ensures that the motor maintains smooth and continuous rotation, allowing the hover board or any other device using a similar motor to operate effectively.

By using electronic commutation and the feedback from Hall effect sensors, the hover board motor eliminates the need for brushes and commutators found in traditional brushed motors. This results in improved efficiency, reduced maintenance, and longer lifespan.

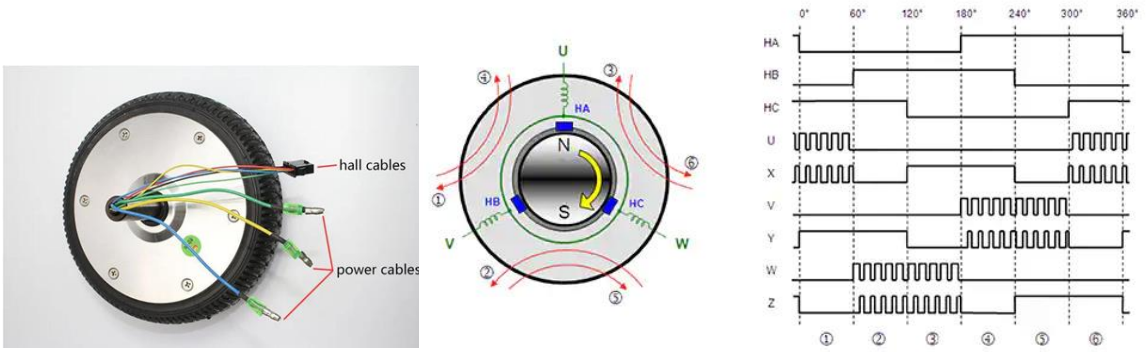

Figure 45: Brushless DC motor

3.3.5 Hoverboard Controller

The main circuit board controller for self-balancing hoverboards is a crucial component that manages the overall operation and functionality of the device.

The differences between existing BLDC controllers are as follows:

BLDC controllers	otter control	Vedder ESC	Hoverboard Hardware
Shape			
Price	Expensive (>40€)	Expensive (>120€)	Expensive (15-20€ shipped)
Current rating	60A max	100A max	20 to 30 (A).
Communication interface	Lots of interfaces (CAN, RS485)	Lots of interfaces (CAN, RS485)	Interface must be hacked
Number of motors	Single motor only	Single motor only	Two BLDC controllers (Two motors)
Size	small	small	large

Table 3: Comparison between existing BLDC controllers

As stated above, **the hoverboard hardware** is the best option on a limited budget, and it is very versatile, given that it has two BLDC motors instead of one. This is the reason why we chose it for the project.

In our autonomous mobile robot project for package transportation, we have incorporated a hoverboard's original controller, which offers a reliable and efficient solution for controlling the two BLDC motors used in the robot's propulsion system. The hoverboard controller serves as a central hub for receiving input signals from various sensors and translating them into motor commands, allowing precise control over the robot's movement.

The hoverboard's original controller is specifically designed to handle the power requirements and operational characteristics of the BLDC motors. It offers advanced motor control algorithms, ensuring smooth acceleration, deceleration, and direction changes. The controller utilizes feedback mechanisms to maintain stability and balance, utilizing data from the MPU6050 sensor and other sensors to adjust motor speed and torque.

3.3.6 Hoverboard Components and Integration

The components on the main circuit board controller:

1. **STM32 Microcontroller:** The STM32 microcontroller serves as the main control unit, running the motor control algorithms and handling communication with other components. STM32 microcontrollers offer a range of options suitable for motor control applications, such as the STM32F4 or STM32F7 series.

2. **Gate Driver:** A gate driver is necessary to provide the high-power signals required to drive the power transistors or MOSFETs connected to the motor. The gate driver ensures efficient switching of the transistors based on the control signals from the STM32 microcontroller.

3. **Power Transistors/MOSFETs:** Power transistors or MOSFETs act as switches to control the current flow through the motor coils. These components should be selected based on the power requirements of the motor and the specific design considerations.

4. **Current Sensing Components:** Current sensing is crucial for motor control. You can integrate current sensing components, such as shunt resistors or current sensing amplifiers, to measure the motor's current and provide feedback to the STM32 microcontroller for accurate control and protection.

5. **Hall Effect Sensors:** the STM32 microcontroller should have inputs capable of receiving signals from these sensors. The microcontroller uses this information to determine the commutation sequence and control the motor accordingly.

6. **External Crystal/Clock Source:** The STM32 microcontroller requires a precise clock source for accurate timing and synchronization. An external crystal or clock source may be used to provide the required clock signal to the microcontroller.

7. **Power Supply:** A power supply circuit is needed to provide the necessary power to the motor controller system. This includes supplying power to the STM32 microcontroller, gate driver, and other components.

8. **Protection Circuitry:** Implementing various protection features is essential to ensure safe and reliable operation. These protections may include overcurrent protection, overvoltage protection, undervoltage protection, temperature monitoring, and short-circuit protection. Dedicated protection components such as fuses, overvoltage protection ICs, and thermal sensors can be integrated into the system.

To control the motor, we are using a combination of FOC (Field-Oriented Control) and Hall effect sensors to control a BLDC (Brushless DC) motor in a hoverboard.

FOC is a control technique that allows for precise control of the motor's torque and flux, resulting in improved efficiency and performance. It requires accurate information about the rotor position, which can be obtained through sensors like Hall effect sensors. Hall effect sensors detect the position of the rotor magnets and provide feedback to the motor controller.

Here's how the combination of FOC and Hall effect sensors can work together in a hoverboard motor control system:

1. **Rotor Position Detection:** The Hall effect sensors are strategically placed around the stator to detect the position of the rotor magnets as they pass by. The sensors generate electrical signals based on the magnetic field changes, indicating the rotor position.

2. **Commutation Sequence:** The rotor position feedback from the Hall effect sensors is used by the motor controller to determine the commutation sequence. The controller then activates the appropriate stator windings in the correct sequence to create a rotating magnetic field.

3. **FOC Control:** With the rotor position known, the motor controller applies FOC algorithms to precisely control the current and voltage in the stator windings. This control technique optimizes the motor's performance, efficiency, and torque output.

4. **Speed and Torque Control:** The motor controller adjusts the PWM (Pulse Width Modulation) signals based on the desired speed and torque requirements. By modulating the duty cycle of the PWM signals, the controller controls the motor's speed and torque output.

By combining FOC control with Hall effect sensors, hoverboard motor control systems can achieve smooth and efficient operation, accurate torque control, and improved overall performance.

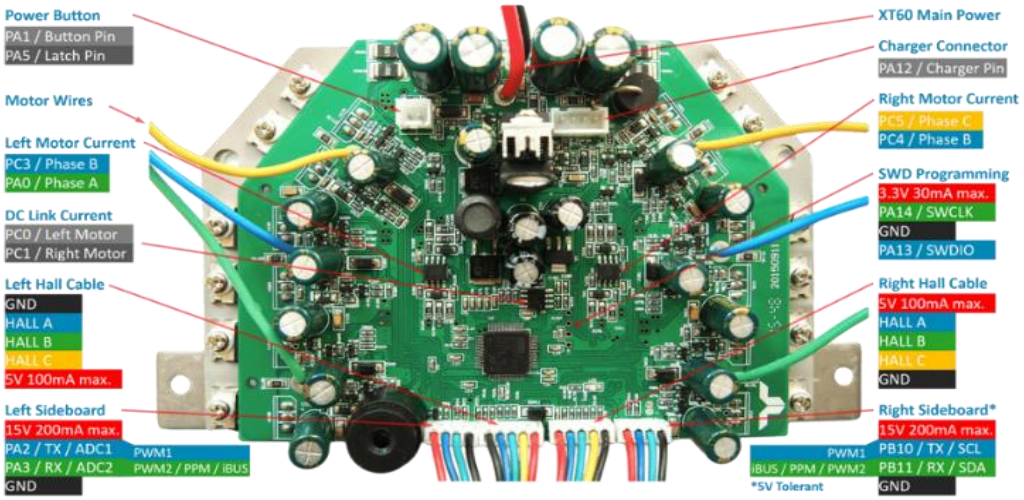

Figure 46: Hoverboard main circuit board controller

3.4 Control and Decision-Making Units

3.4.1 The Arduino

The Arduino is a widely used microcontroller platform that provides a simple and accessible way to create interactive electronic projects. It features various microcontroller boards, each with its own processor. It is an open-source electronics platform that consists of both hardware and software components**.**

We used one of the many Arduino boards in the project, and ultimately landed on the **Arduino Mega** board for its versatility and memory capacity.

The Arduino, in our project, serves as the central microcontroller responsible for the robot's control and coordination. Specifically, the Arduino Mega is employed to interface with the MPU6050 sensor. The MPU6050 is an inertial measurement unit (IMU) that combines a gyroscope and accelerometer. It provides critical motion and orientation data for the robot, enabling it to maintain balance and adjust its movements accordingly during package transportation.

The Arduino Mega's expanded input/output (I/O) capabilities and processing power make it an ideal choice for integrating multiple sensors and controlling motor systems. It efficiently reads data from the MPU6050 sensor and processes it in real-time. The Arduino Mega then acts as an intermediary between the sensor and Raspberry Pi 4. It sends the collected motion and orientation data to Raspberry Pi 4 for further analysis, decision-making, and higher-level processing.

By utilizing the Arduino Mega in this capacity, we ensure seamless communication between the MPU6050 sensor and the Raspberry Pi 4, enabling the robot to make intelligent decisions based on real-time data. This integration enhances the robot's ability to navigate its environment, avoid obstacles, and transport packages safely and efficiently.

3.4.1.1 Specifications

- Microcontroller: ATmega2560
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V.
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 54 (of which 15 provide PWM output)
- Analog Input Pins: 16
- DC Current per I/O Pin: 20 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 256 KB of which 8 KB is used by bootloader.
- SRAM: 8 KB
- EEPROM: 4 KB
- Clock Speed: 16 MHz

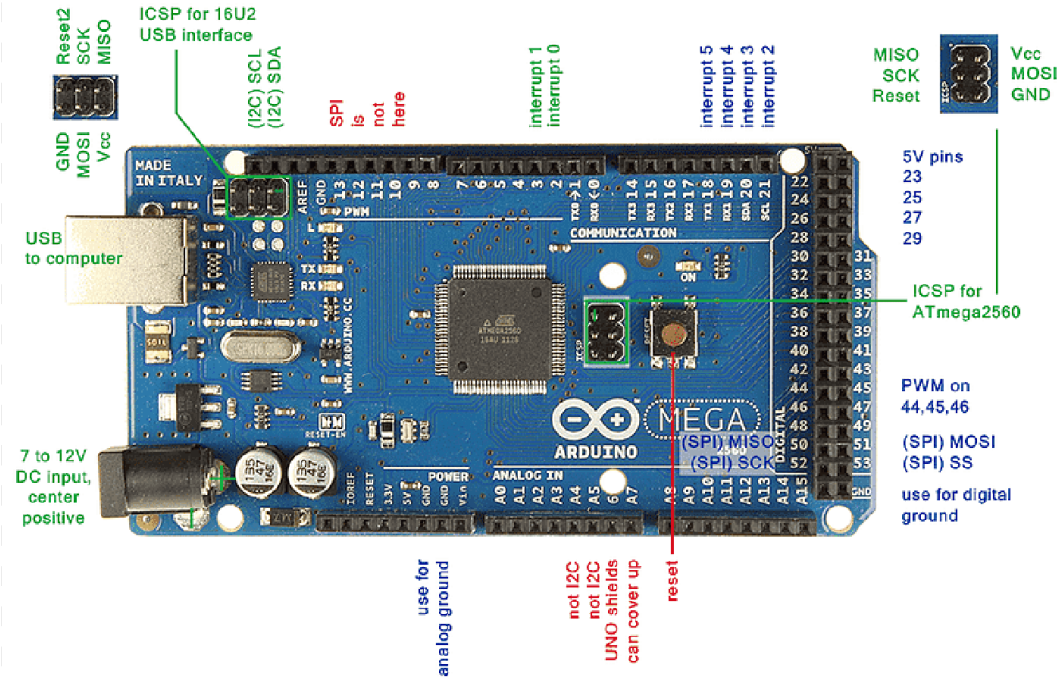

Figure 47: Arduino Mega

3.4.2 The Raspberry Pi

The Raspberry Pi is a single-board computer that integrates both a microcontroller and a microprocessor. It combines the functionality of a microcontroller and the power of a microprocessor.

In the graduation project, we are using **The Raspberry Pi 4** which is a single-board computer that serves as a versatile and powerful platform for various electronic projects, including robotics. It is the fourth generation of the Raspberry Pi series, offering significant improvements in performance and capabilities compared to its predecessors.

The Raspberry Pi 4 serves as a powerful computing platform in our project, responsible for reading data from the Kinect v1 sensor and orchestrating the robot's operations. The Kinect v1, a depth sensor, provides crucial 3D information about the environment, allowing the robot to perceive distances, detect obstacles, and enhance its navigation capabilities. The The Raspberry Pi 4 receives depth data from the Kinect v1, processes it using appropriate algorithms, and makes intelligent decisions based on this information.

To enhance the robot's stability and balance, we rely on the MPU6050 sensor. The MPU6050 combines a gyroscope and accelerometer, providing accurate motion and orientation data. This data is vital for the robot's motion control and adjustment during package transportation. To enable communication between the MPU6050 and The Raspberry Pi 4, we utilize the Arduino Mega. Through the ROS serial communication protocol, the Arduino Mega reads the sensor data from the MPU6050 and transmits it to The Raspberry Pi 4 in real-time. This seamless integration enables The Raspberry Pi 4 to analyze the motion and orientation data from the MPU6050 and utilize it for further decision-making and control.

By employing The Raspberry Pi 4 for Kinect data acquisition and the Arduino Mega for MPU6050 data acquisition, we ensure a comprehensive and synchronized sensing system for our autonomous mobile robot. The combination of depth information from the Kinect v1 and motion data from the MPU6050 empowers the robot to navigate its surroundings effectively, detect

obstacles, and transport packages with precision and efficiency. The integration of ROS serial communication facilitates seamless data transfer and enables advanced data analysis and decision-making capabilities.

3.4.2.1 Specifications

- **Processor:** The Raspberry Pi 4 is powered by a Broadcom BCM2711 quad-core Cortex-A72 (ARMv8) 64-bit processor, running at 1.5 GHz. This provides a substantial increase in processing power compared to previous Raspberry Pi models.

- **RAM:** The Raspberry Pi 4 is available with different RAM options, including 2GB, 4GB, and 8GB LPDDR4 SDRAM. The increased memory capacity allows for more demanding applications and multitasking.

- **GPU:** It features a Video Core VI graphics processing unit (GPU) with support for OpenGL ES 3.x and 4K video playback at 60 frames per second.

- **Connectivity:** The Raspberry Pi 4 offers enhanced connectivity options, including dual-band 2.4 GHz and 5 GHz 802.11ac wireless LAN (Wi-Fi), Bluetooth 5.0, and Gigabit Ethernet. It also has two USB 3.0 ports and two USB 2.0 ports for connecting peripherals.

- **Video and Display:** It supports dual-monitor configurations with two micro-HDMI ports capable of outputting up to 4K resolution. The Raspberry Pi 4 also has a MIPI CSI camera port and a DSI display port for connecting cameras and displays.

- **Storage:** The Raspberry Pi 4 features a microSD card slot for storage, allowing you to use a microSD card to boot and store the operating system and files.

- **GPIO:** It retains the 40-pin GPIO header found on previous models, providing digital I/O, analog inputs, and various communication interfaces, making it compatible with a wide range of add-on boards and accessories.

- **Operating System:** The Raspberry Pi 4 supports a variety of operating systems, including the official Raspberry Pi OS (formerly known as Raspbian), Ubuntu, and other Linux distributions.

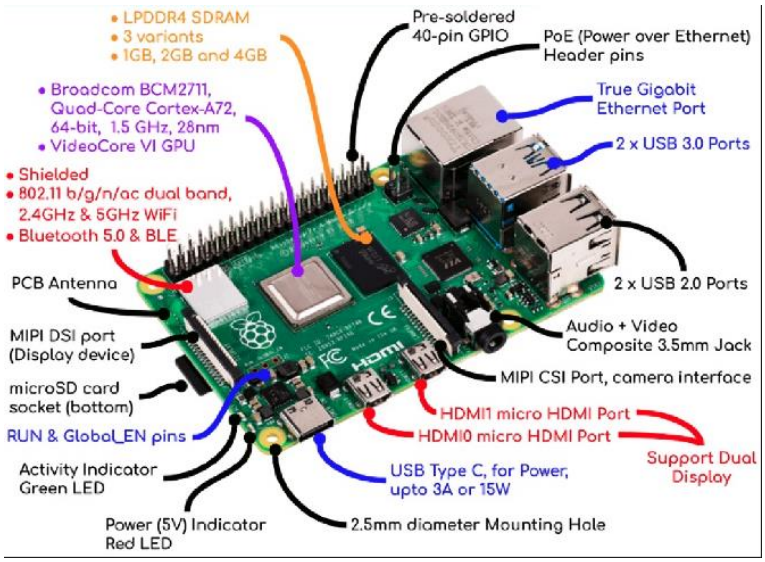

Figure 48: Raspberry Pi 4

3.5 Proposed system structure

- As shown in fig 47 the full system structure with all sensors used in this project.

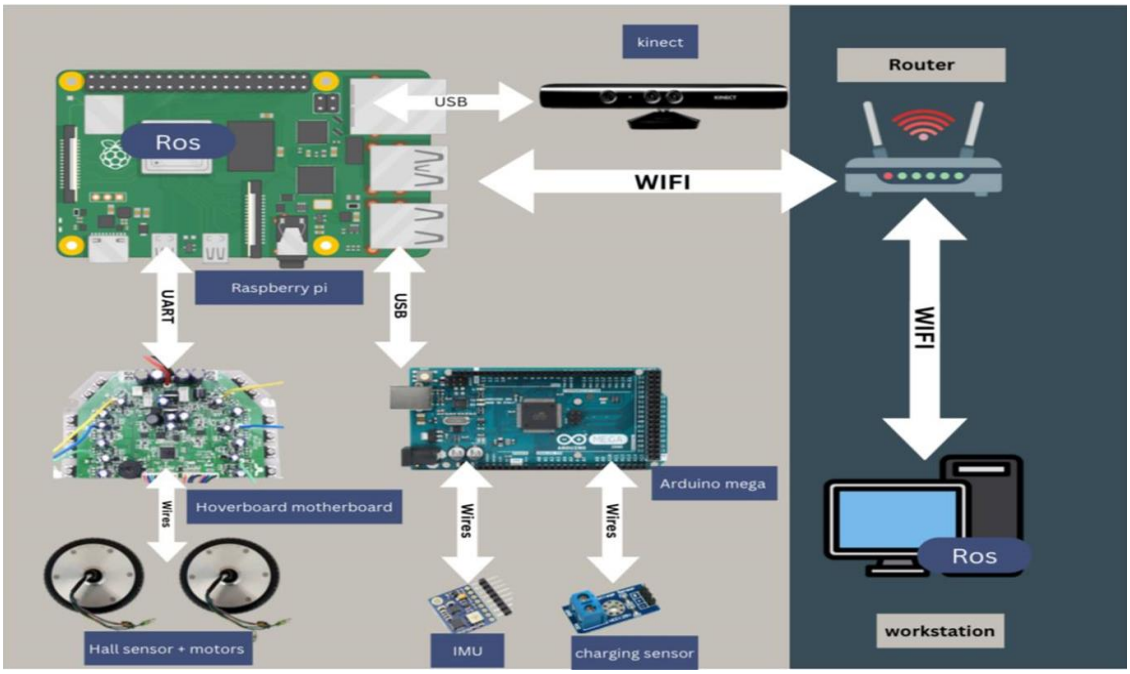

Figure 49 Sensors used in this project and the communication between them

3.6 Docking

Robot docking is a process where a robot autonomously navigates and aligns itself with a docking station. The docking station serves as a designated location for the robot to recharge its batteries or establish a connection for data exchange. Through localization and navigation capabilities, the robot can accurately locate the docking station using sensors like cameras, laser scanners, or proximity sensors.

3.6.1 Batteries

Batteries play a vital role in powering robots, providing the necessary energy for their operation. They serve as portable and rechargeable energy storage devices that enable robots to function independently without being tethered to a power source. The choice of batteries for robots depends on factors such as energy capacity, voltage, weight, size, and the specific requirements of the robot's application.

3.6.2 Types of Batteries

Batteries can be classified into two main types: primary batteries and secondary batteries.

- **Primary Batteries:** Primary batteries, also known as disposable batteries, are designed for single use. Once their stored energy is depleted, they cannot be recharged and must be replaced. Primary batteries are commonly used in devices where long-term power storage is not required, or where it is inconvenient to recharge the battery frequently. Some examples of primary batteries include alkaline batteries, lithium batteries, zinc-carbon batteries, and silver oxide batteries.

- **Secondary Batteries:** Secondary batteries, also known as rechargeable batteries, are designed to be recharged and reused multiple times. They can be recharged by applying an electrical current that reverses the chemical reactions inside the battery, restoring its energy storage capacity. Secondary batteries are widely used in devices that require frequent or continuous use, as they offer the advantage of reusability and cost-effectiveness over time. Common examples of secondary batteries include lithium-ion (Li-ion) batteries, nickel-cadmium (NiCd) batteries, nickel-metal hydride (NiMH) batteries, and **lead-acid batteries**.

Lead-acid batteries have proved superior in the context of the project. This is for reasons explained below.

3.6.3 Why Lead-Acid Batteries?

In our project, we have carefully considered the choice of batteries, and after weighing various factors, we have opted to use lead-acid batteries. One of the primary reasons for this choice is the emphasis on **safety**. While lithium-ion batteries offer higher energy density, they are associated with a higher risk of thermal runaway and potential explosions if mishandled or improperly charged. In contrast, lead-acid batteries have a proven track record of safety and are known for their robustness.

Lead-acid batteries are rechargeable, allowing us to use them multiple times without the need for frequent replacement. This reusability factor adds to their cost-effectiveness, as it eliminates the need for frequent battery purchases.

Additionally, lead-acid batteries come with built-in safety features during charging and discharging. These batteries can be charged using a suitable charging system that incorporates safeguards against overcharging, over-discharging, and thermal runaway. This ensures the safe and efficient operation of our system while minimizing the risk of accidents.

In our project, we have opted to use a 40 ampere 12-volt lead-acid battery as the primary power source to supply all the components required. This battery choice offers several advantages and considerations.

3.6.3.1 Specifications

- Brand: Long
- Model: WPS40-12N
- Voltage: 12V
- Battery Capacity: 40Ah
- Weight: 12.6KG (27.7Lbs.)
- Battery Dimensions: Length (L) 199+3-1 (7.83+0.12-0.04); Width (W) 166+3-1 (6.54+0.12-0.04); Height (H) 171+3-1 (6.73+0.12-0.04)
- Design Life: 3-5 Years.

Figure 50: Lead-acid battery

3.6.4 Docking Circuit Components

Components used in the system are:

- A Buck Converter
- A Relay
- Lead-acid batteries
- Wires
- Switches

3.6.4.1 Buck Converters

A buck converter, also known as a step-down converter, is a type of DC-DC (direct current to direct current) converter that converts a higher voltage input to a lower voltage output. It is widely used in various electronic devices and power supply applications to efficiently regulate voltage levels.

We determined that the **XL4015 5A DC-DC Step Down Adjustable Power Supply Module Buck Converter 4-38V to 1.25-36V** is the best option for the project.

3.6.4.2 Specifications

- **Input Voltage:** 4 to 38V DC; Input Current: 5A MAX, over 3A replace existing heatsink with bigger heatsink to enhance heat dissipation, 2. Related to the input and output voltage difference, the greater voltage difference is, the higher output current will be.
- **Output Voltage:** 1.25-36V continuously adjustable.
- **Output Current:** 5A MAX, over 3A replace existing heatsink with bigger heatsink.
- **Input Power:** 50W, peak 70W, Input voltage * load current (less 5% loss due to efficiency)
- **Working temperature:** -40~+85 degrees; Operating frequency: 180KHz; Efficiency: 96% (max)

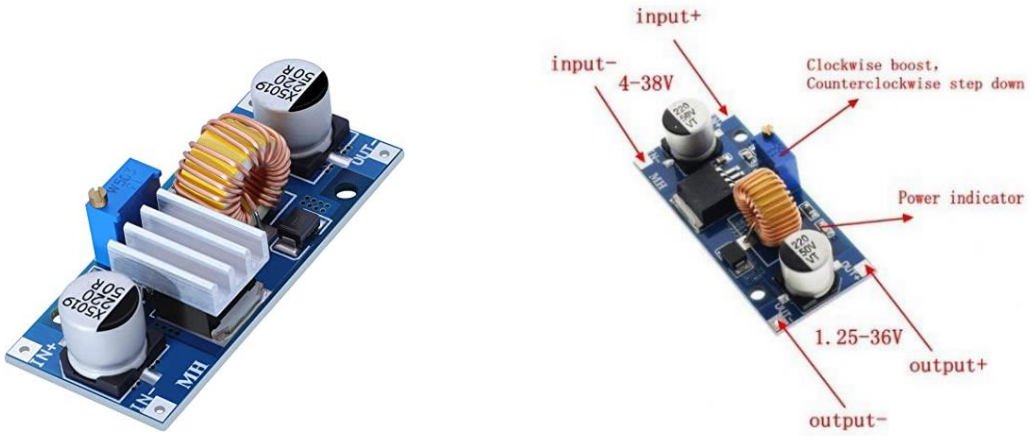

Figure 51: Buck convertor

3.6.4.3 Relay

A relay is an electrically operated switch that is commonly used in various applications to control high-power or high-voltage circuits using a low-power signal. It allows the control of one electrical circuit by another circuit, which may have different voltage levels or characteristics.

To achieve the best results, a **5 Pin 12V Relay Switch (SPDT) (30/40 Amp)** has been integrated.

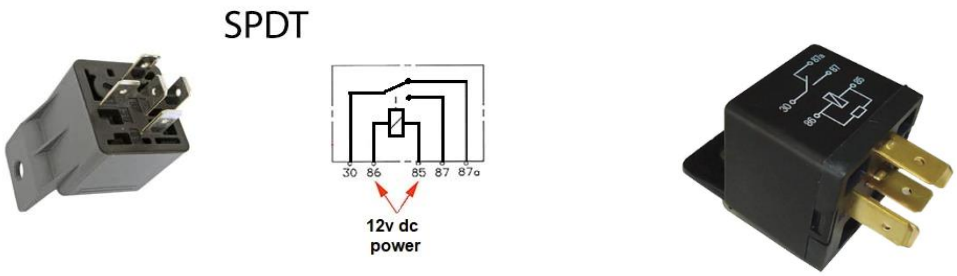

Figure 52: A relay

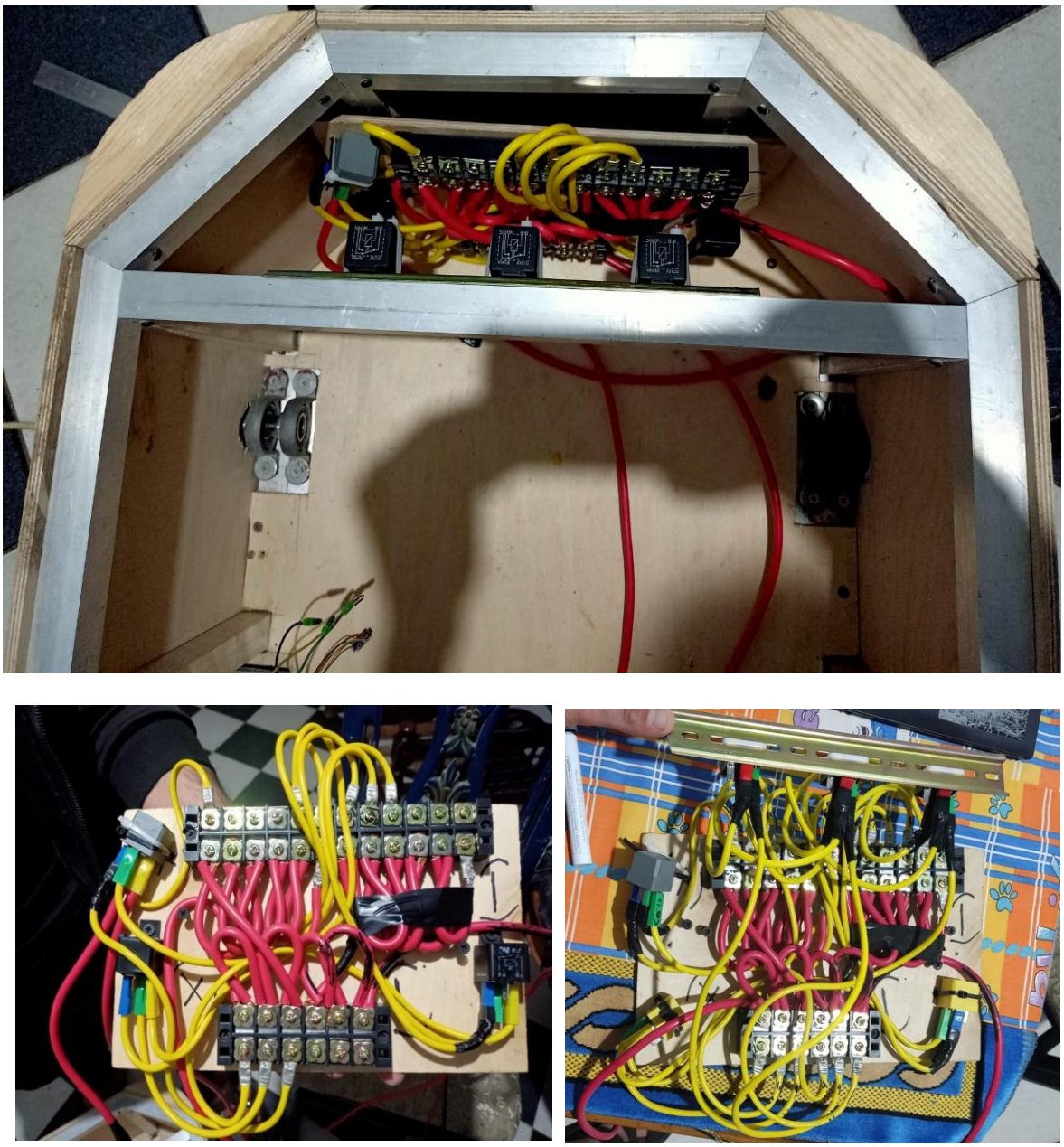

Figure 53: Circuit implementation in the real-world model

Chapter 4: **The Proposed Software**

4.1 **Inceptive Overview to Software Section**

The software component of an autonomous mobile robot is critical in achieving its objectives of navigating through a warehouse environment, identifying and locating objects, and transporting them from one location to another autonomously. In this section of the book, we will focus on the software development process of the graduation project, which involved designing, implementing, and testing a range of software modules that enabled proposed robot to perform its tasks in a warehouse environment.

The proposed software development process began with the selection of a suitable operating system and framework for the proposed robot. We chose ROS Noetic, a widely used open-source robotic middleware, as it provided a range of powerful tools and libraries for developing complex robotic applications. We then designed and implemented several software modules to enable our robot to perform key tasks, such as navigation, object detection, and motion planning.

One of the most important modules we developed was the navigation module, which allowed the robot to move autonomously through a warehouse environment. This module relied on sensor data from the Kinect 360 and the MPU6050 to build a map of the environment, localize the robot's position, and plan its trajectories. We also developed a custom motion planning module, which used a combination of A* and RRT* algorithms to generate optimal paths for the robot to follow.

One of the most important modules we developed was the mapping module, which allowed our robot to build a map of the warehouse environment it was operating in. The mapping module relied on sensor data from the Kinect 360 and the MPU6050 to construct a 2D and 3D occupancy grid map of the environment, which was used to guide the robot's navigation and avoid obstacles.

We utilized the RTAB-Map package in ROS, which is an open-source implementation of the simultaneous localization and mapping (SLAM) algorithm, to generate and update the map. The algorithm used the robot's odometry data and sensor measurements to estimate its position and construct a map of the environment. RTAB-Map also utilized loop closure detection and visual feature mapping to improve the accuracy and consistency of the map.

Throughout this section, we will provide a detailed account of the software development process of our graduation project, including the design and implementation of key software modules, as well as the challenges we encountered and how we overcame them. We will also present the results of our experiments, including performance metrics and evaluations, and discuss future work and potential applications of our software.

Overall, this section of the book represents a comprehensive and detailed account of the software development process of our autonomous mobile robot, providing insights into the design, development, and testing of key software modules that enabled the robot to perform its tasks in a warehouse environment. We hope that it will serve as a useful resource for students, researchers, and practitioners interested in the field of robotics and automation and inspire future innovations and advancements in this field.

.

4.2 Operating System and Framework Selection

The selection of an operating system and framework is a critical decision in the development of an autonomous mobile robot. It determines the tools and libraries available for software development and the ability to interface with hardware components. In this chapter of the book, we will discuss the decision-making process behind the selection of ROS as the operating system and framework for our graduation project.

ROS, or Robot Operating System, is an open-source robotics middleware that provides a range of powerful tools and libraries for developing complex robotic applications. It has become a standard in the robotics industry due to its flexibility, scalability, and robustness.

We chose ROS for our project for several reasons. Firstly, ROS provides extensive support for various sensors and hardware components, making it easy to interface with the components we chose for our robot, such as the hoverboard controller, motors, and sensors. Secondly, ROS has an extensive library of algorithms for tasks such as mapping, localization, and motion planning, which made it easier to develop the software components of our robot.

Additionally, ROS has an active community of developers, which provides us with a wealth of resources, tutorials, and examples to draw upon during the development process. This community support also ensured that we had access to up-to-date documentation and software updates, which helped us stay on top of the latest developments in the field of robotics.

Overall, the selection of ROS as our operating system and framework provided us with a solid foundation for the development of the proposed autonomous mobile robot. In the following sections of the book, we will describe how we utilized ROS to develop the software components of the proposed robot and the challenges we faced during the development process.

4.3 **ROS Distribution Selection**

The choice of ROS distribution depends on several factors such as the requirements of the project, the hardware and software configuration of the robot, and the availability of community support. ROS Kinetic, for example, is designed for Ubuntu 16.04 LTS (Xenical) and is best suited for robots with limited computational resources. ROS Melodic, on the other hand, is designed for Ubuntu 18.04 LTS (Bionic) and includes several new features and improvements over its predecessors. ROS Noetic, the latest ROS distribution, is designed for Ubuntu 20.04 LTS (Focal) and includes several new packages and libraries for robotics development.

For our graduation project, we chose ROS Noetic as our operating system and framework for several reasons. In the following sections, we will discuss the decision-making process and the factors that influenced our choice of ROS Noetic. here is a snapshot of the ROS distribution.

Figure 54 ROS Distributions List

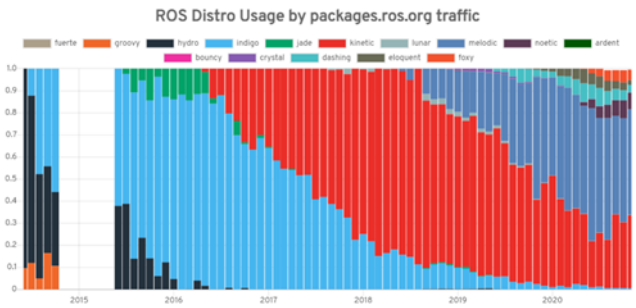

Figure 55 ROS Distro Usage

4.4 ROS Noetic Installation

4.4.1 Overview of ROS Noetic Version

ROS Noetic is the 12th release of the Robot Operating System (ROS), which is a flexible framework for writing robot software. It provides a collection of tools, libraries, and conventions that aim to simplify the development of robotic systems. Installing ROS Noetic allows users to leverage its features and capabilities for building and controlling robots.

4.4.2 Installation Process

Installing ROS Noetic involves several steps, which are detailed in the official ROS wiki documentation. The installation process typically includes the following major steps:

System Requirements: Before installing ROS Noetic, it is important to ensure that your system meets the minimum requirements specified in the ROS documentation. These requirements include the operating system version, processor architecture, and other dependencies.

Set Up Sources and Keys: ROS Noetic requires adding the ROS package repository to your package manager's sources list. This step involves configuring your system to fetch packages from the ROS repository and importing the repository's key for package verification.

Installation: Once the sources and keys are set up, you can proceed with the installation process. This step involves using the package manager to download and install the necessary ROS packages and dependencies.

Environment Setup: After installation, the environment needs to be properly configured to work with ROS. This involves setting up environment variables, adding the necessary sourcing lines to your shell configuration file, and potentially creating a ROS workspace.

4.4.3 Common Problems and Troubleshooting

During the installation process, you may encounter certain issues or errors. Here are some common problems and their potential solutions:

Dependency Issues: ROS Noetic has dependencies that need to be resolved during installation. If you encounter dependency errors, make sure you have the necessary packages and libraries installed. Consult the ROS documentation for specific dependencies and their installation instructions.

Network Connectivity: If you are installing ROS Noetic on a system with restricted internet access, you may face issues with downloading packages. In such cases, consider using a network proxy or downloading the required packages on a different machine and transferring them manually.

Version Compatibility: ROS Noetic is designed to work with specific versions of the operating system and other software packages. Ensure that your system meets the required version compatibility as specified in the ROS documentation.

Build Errors: During the installation, you may encounter build errors related to compiling packages. These errors can often be resolved by carefully examining the error message, checking for missing dependencies, and ensuring that the necessary build tools are installed.

4.4.4 Conclusion

In this chapter, we discussed the process of installing ROS Noetic, which is an essential step to utilize the capabilities of the Robot Operating System for robot development. We covered the major installation steps, including system requirements, setting up sources and keys, the actual installation process, and environment setup. Additionally, we explored some common problems that may arise during installation and provided general troubleshooting tips.

For detailed installation instructions and troubleshooting specific to your system, it is recommended to refer to the official ROS Noetic documentation available on the ROS wiki. Following the provided steps and troubleshooting guidelines will help you successfully install ROS Noetic and get started with developing your own robotic application.

4.5 **Project workspace preparations**

Before setting up our workspace we must know how ROS workspace is organized?

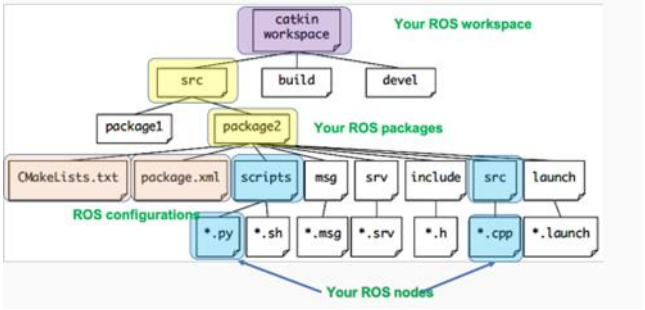

Figure 56 ROS workspace

As you can see in the figure above ROS uses a workspace to organize your projects and code. A workspace is a directory that contains all the packages and code that are related to a specific project. The workspace is organized as follows:

1- src Directory: This is the directory where you put all your packages. Each package should have its own directory in the src directory.

2- build Directory: This is the directory where the build files are stored. When you build your packages, the compiled files are stored here.

3- devel Directory: This is the directory where the compiled files are stored after they have been built. When you run your code, the executable files are executed from this directory.

4- When you create a new workspace, you should create a src directory and put all of your packages in this directory. Each package should have its own directory within the src directory. You should also create a CMakeLists.txt file and a package.xml file for each package.

5- The CMakeLists.txt file is used to build your code. It specifies the source files that should be compiled, the libraries that your code depends on, and the executable that should be created.

6- The package.xml file is used to describe your package. It includes information such as the name of the package, the version number, the author, and the dependencies that your package has.

When you are ready to build your code, you should navigate to your workspace directory and run the following commands:

```
catkin_make
source devel/setup.bash
```

The catkin_make command will build your code and create the build and devel directories. The source devel/setup.bash command will set up your environment so that you can run your code.

In summary, ROS workspaces are organized into three main directories: src, build, and devel. The src directory contains all your packages, the build directory contains the build files, and the devel directory contains the compiled files. Each package should have its own directory within the src directory, and you should create a CMakeLists.txt file and a package.xml file for each package.

4.6 URDF and Gazebo Plugins

In the previous chapter, we discussed ROS installation and the common problems. In this chapter, we will focus on the Unified Robot Description Format (URDF) and Gazebo plugins, which are essential tools for robot simulation and visualization.

4.6.1 URDF

URDF is an XML format used to describe robots and their components. It is used by ROS to represent the physical structure of a robot, including joints, links, sensors, and other components. The URDF file specifies the geometry, visual and collision properties, and kinematic information of the robot, making it an essential part of the robot development process.

4.6.2 Gazebo Plugins

Gazebo is a powerful simulation tool used to test and validate robot designs in a virtual environment. It supports several physics engines, including ODE, Bullet, and Sim body, and provides a range of sensors, controllers, and other plugins to simulate the behavior of the robot. Gazebo plugins are software modules that can be added to the simulation environment to extend its capabilities. Some of the commonly used Gazebo plugins include:

1- **Model Plugins:** These plugins are used to add custom behavior to a robot model. They can be used to add controllers, sensors, and other components to the robot.

2- **Sensor Plugins:** These plugins are used to simulate sensors such as cameras, lidars, and sonars. They provide realistic sensor data that can be used for testing and validation.

3- **World Plugins:** These plugins are used to modify the Gazebo world properties, such as gravity and time. They can be used to simulate different environmental conditions and scenarios.

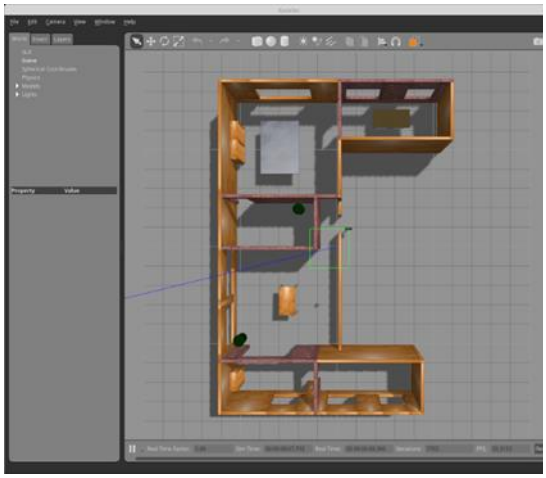

Figure 57 Gazebo world

Figure 58 Gazebo

4.6.3 URDF and Gazebo Plugins in Action

To demonstrate the use of URDF and Gazebo plugins, let's consider an example of a robot with a custom sensor setup. In this example, we will use the URDF to describe the physical structure of the robot, including its joints, links, and sensors. We will then use Gazebo plugins to simulate the behavior of the robot in a virtual environment.

First, we need to create a URDF file that describes the robot. The URDF file should include the joint and link definitions, as well as the sensor models. Once the URDF file is created, we can use it to generate the robot model in Gazebo. We can then add sensor plugins to the robot model to simulate its behavior.

Let's assume that our robot has a custom sensor setup that includes a Kinect sensor and a fake lidar sensor. We can use the Gazebo ROS plugins to simulate the behavior of these sensors. The ROS Gazebo plugin provides a range of sensor models, including a Kinect sensor and a lidar sensor. We can use these sensor models to simulate the behavior of our custom sensor setup.in this project we used Kinect plugin, differential drive plugin and IMU plugin.

- **Kinect Plugin:** The Kinect plugin is used to simulate the output of a Kinect sensor in Gazebo. This allows us to test and develop our robotics algorithms in a simulated environment before deploying them on a physical robot with a real Kinect sensor. The plugin takes in the raw depth and RGB data from the sensor and publishes them as ROS topics, which can then be used by other nodes in our robotics system.

- **Differential Drive Plugin:** The differential drive plugin is used to model the movement of a differential drive robot in Gazebo. This plugin allows us to specify the linear and angular velocities of the robot, which are used to calculate the position and orientation of the robot in the simulated environment. The plugin also considers the physical properties of the robot, such as the wheel radius and separation distance, to accurately model its movement.

By using these plugins, we can simulate the behavior of a robot equipped with a Kinect sensor and differential drive system in a virtual environment. This allows us to test and refine our algorithms before deploying them on a physical robot, which can save time and resources in the development process.

The following launch files show an example of URDF, and gazebo plugins used in the project.

```
  <<?xml version="1.0"?>
  <!--  +++++++++++++++++ Diff drive plugin++++++++++++++++++          /-->
  <gazebo>
<plugin name="differential_drive_controller"
     filename="libgazebo_ros_diff_drive.so">
      <legacyMode>false</legacyMode>
      <alwaysOn>true</alwaysOn>
      <updateRate>20</updateRate>
      <leftJoint>top_left_wheel_hinge</leftJoint>
      <rightJoint>top_right_wheel_hinge</rightJoint>
      <wheelSeparation>0.4</wheelSeparation>
      <wheelDiameter>0.1</wheelDiameter>
      <torque>20</torque>
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <robotBaseFrame>base_link</robotBaseFrame>
    </plugin>
  </gazebo>
```

```
  <?xml version="1.0" encoding="utf-8"?>
  <!-- ================================================================================ -->
  <!-- |    This document was autogenerated by xacro from mybot_description.urdf        | -->
  <!-- |    EDITING THIS FILE BY HAND IS NOT RECOMMENDED                                 | -->
  <!-- ================================================================================ -->
  <!-- This URDF was automatically created by SolidWorks to URDF Exporter!
     Originally created by
  Stephen Brawner (brawner@gmail.com)
      Commit Version: 1.6.0-1-g15f4949  Build Version: 1.6.7594.29634
      For more information, please see http://wiki.ros.org/sw_urdf_exporter -
     ->
  <robot name="myrobot" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:include filename="$(find mybot_description)/urdf/mybot.gazebo" />
  <link name="left_motor">
    <inertial>
     <origin   rpy="0  0   0"   xyz="-4.20679446833105E-09   0.0331497387034928
     4.87590441999863E-06" />
      <mass value="1.43144480602327" />
     <inertia   ixx="0.00331965861699228"   ixy="-1.65643329348428E-10"   ixz="-
     2.49861736040813E-10"
                    iyy="0.00595557706241427"     iyz="-3.90156064035793E-07"
     izz="0.00331969353968103" />

    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0" />
      <geometry>
        <mesh filename="package://mybot_description/meshes/left_motor.STL" />
      </geometry>
      <material name="">
        <color rgba="0.792156862745098 0.819607843137255 0.933333333333333 1"
      />
      </material>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0" />
      <geometry>
        <mesh filename="package://mybot_description/meshes/left_motor.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="left_motor_joint" type="continuous">
```

```
    <origin rpy="0 0 3.1416" xyz="0.21185 0.327 -0.242" />
    <parent link="base_link" />
    <child link="left_motor" />
    <axis xyz="0 -1 0" />
  </joint>
```

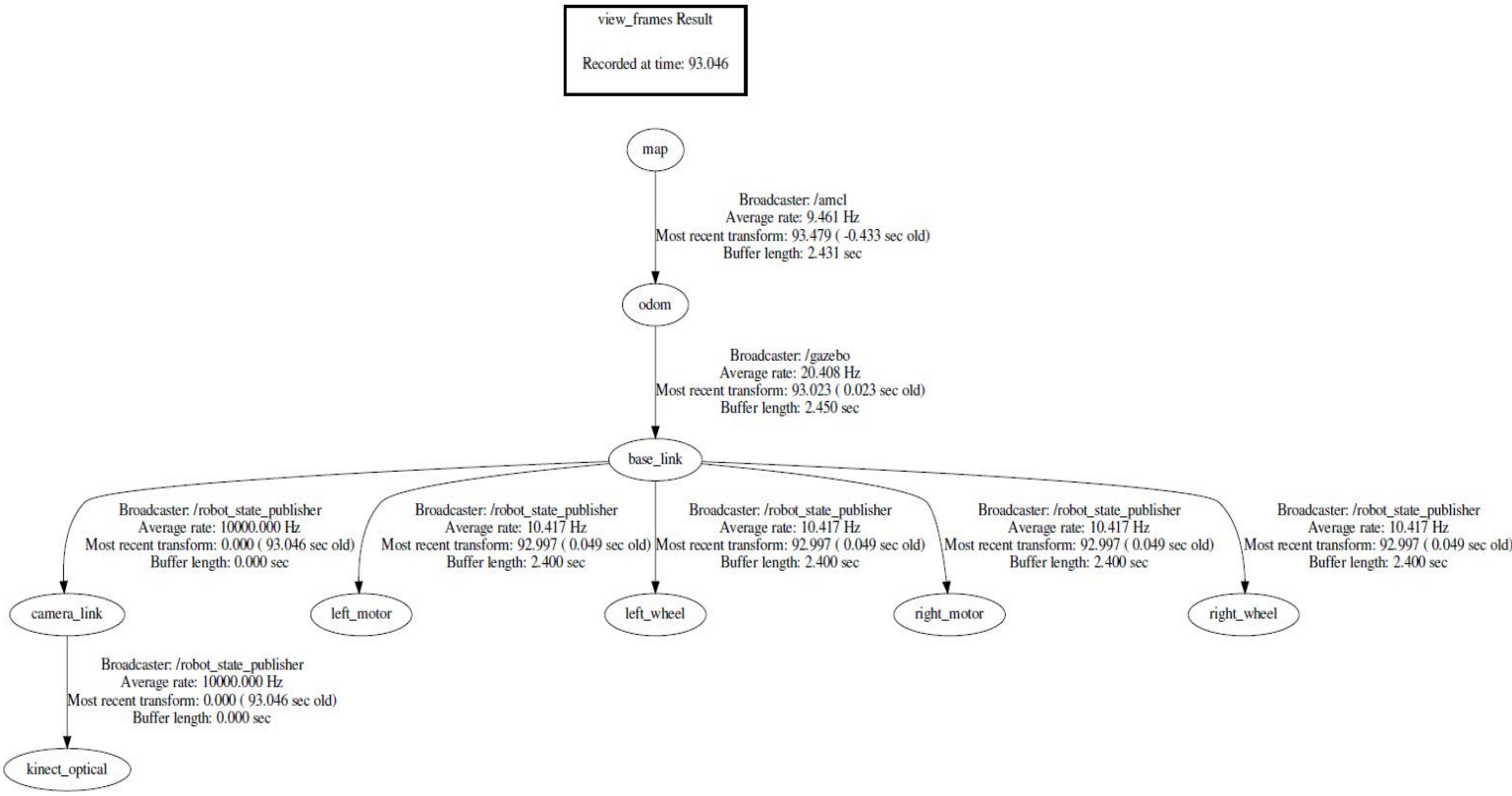

Figure 59 TF tree that describes the relation between frames

4.6.4 Xacro Headers

1- **XML Version Declaration:** This is the first line in any XML file and declares the version of XML being used. It should be included at the beginning of every Xacro file. For example:

<?xml version="1.0"?>

2- **Robot Model Declaration:** This is the root element of the Xacro file and provides a namespace for the robot model. It should be included directly below the XML version declaration. For example:

<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

3- **Name:** This header specifies the name of the robot model. It is a required field and should be unique. For example:

<robot name="my_robot">

4- **Version:** This header specifies the version of the robot model. It should be included to ensure compatibility when making changes to the robot model. For example:

<robot name="my_robot" version="1.0">

5- **Material:** This header specifies the material properties of the robot model, such as its color and texture. It can be used to visualize the robot in simulation or to generate CAD drawings. For example:

```
<material name="red">
    <color rgba="1 0 0 1"/>
</material>
```

6- **Include:** This header is used to include other Xacro files or URDF files within the robot model. It is useful for modularizing large robot models. For example:

<include file="$(find my_package)/urdf/robot.urdf.xacro"/>

7- **URDF:** This header is used to declare the URDF version used in the Xacro file. URDF is an XML format for representing robot models. For example:

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

8- **<robot>:** This is the root element of the Xacro file, and it indicates that the file contains a description of a robot.

9- **<material>:** This element is used to define a material that can be applied to a mesh in the robot description. It can specify the color, texture, and lighting properties of the material.

10- **<link>:** This element defines a rigid body in the robot model. It specifies the geometry, mass, and inertia properties of the link, as well as its name and any visual or collision elements associated with it.

11- **<joint>:** This element defines a joint that connects two links in the robot model. It specifies the type of joint (e.g., revolute, prismatic), the joint axis, and any limits or other properties of the joint.

12- **<gazebo>:** This element is used to include Gazebo-specific information in the robot description. It can specify things like physics properties, sensors, and plugins for controlling the robot in a Gazebo simulation.

13- **<transmission>:** This element is used to define a transmission that connects a joint to a motor or other actuator. It specifies the type of transmission (e.g., gearbox, differential), as well as any gear ratios or other properties of the transmission.

14- **<sensor>:** This element is used to define a sensor that is attached to a link in the robot model. It specifies the type of sensor (e.g., camera, lidar), as well as any properties of the sensor such as its field of view or resolution.

4.7 Proposed Mapping Algorithms'

4.7.1 Mapping Preliminary Discussion

Mapping is a crucial task in robotics that enables a robot to build a representation of its environment. There are several algorithms for mapping in ROS, each with its own strengths and weaknesses. In this section, we will discuss some of the most commonly used mapping algorithms in ROS and their output.

After defining mapping, it's important to understand how to visualize and monitor the mapping process. One way to do this is by using RQT graph.

RQT graph is a graphical tool that displays the ROS computation graph, which shows the relationships between nodes and topics in a ROS system. In the context of mapping, RQT graph can be used to monitor the communication between the mapping algorithm node and the sensors, as well as any other nodes that may be involved in the mapping process.

In our case, since you used Kinect 360 as your sensor, you had the option to use GMapping, Octomapping or RTAB for mapping. GMapping would create a 2D occupancy grid map, while Octomapping would create a 3D occupancy grid map. You could also have used the 2D version of Octomapping to create a 2D occupancy grid map. RTAB-Map is a real-time appearance-based mapping system, which can create a 3D point cloud map of the environment while also performing loop closure detection and graph-based SLAM optimization. It can work with various sensors, such as RGB-D cameras, stereo cameras, and even LIDARs. RTAB-Map can also perform localization, allowing the robot to navigate and move around in a previously mapped environment. In summary, when using a Kinect 360 sensor for mapping, the options available are GMapping, 2D and 3D Octomapping, and RTAB mapping. GMapping creates a 2D occupancy grid map, while Octomapping creates either a 2D or a 3D occupancy grid map. RTAB mapping is a 3D SLAM approach that can be used with different sensors. RQT graph can be used to visualize the ROS nodes and their communication in real-time during the mapping process.

4.7.2 First GMapping using Kinect 360

3D SLAM approach that can be used with different sensors. RQT graph can be used to visualize the ROS nodes and their communication in real-time during the mapping process.

GMapping is a popular algorithm for 2D mapping with laser range finders, but it can also be used with RGB-D sensors like the Kinect. The algorithm uses a particle filter to estimate the robot's pose and map at the same time. It also uses a scan matching algorithm to align each new range measurement with the map.

The RQT graph for GMapping with Kinect would typically involve a node for the Kinect driver, a node for the GMapping algorithm, and a node for visualization, such as RViz. The communication between these nodes can be visualized using RQT graph. Here's an example of the output of GMapping with Kinect 360 and RQT graph of it:

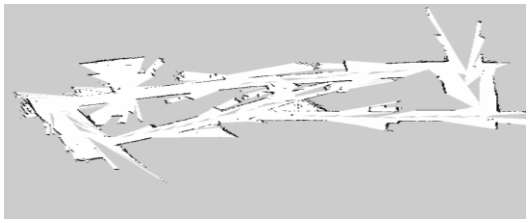

Figure 60 GMapping with Kinect 360

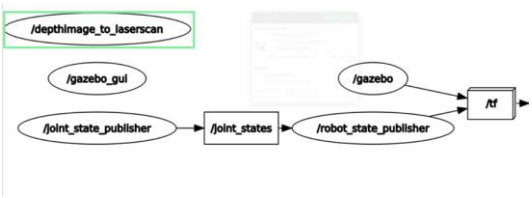

Figure 61 GMapping with Kinect 360 RQT graph

The motion model for the robot is usually modeled as a Gaussian distribution around the expected pose of the robot at the next time step. This can be represented by the equation:

$$P(x_t|\, u_t, x_{t-1}) = \mathrm{N}(x_t; \mathrm{f}(\, u_t, x_{t-1}), sigma_t)$$

Where x_t is the robot pose at time t, u_t is the control input at time t, x_{t-1} is the robot pose at the previous time step, f is the motion model, $sigma_t$ is the covariance matrix representing the uncertainty in the motion model.

The measurement model in GMapping uses the likelihood of a sensor reading z_t given the pose x_t of the robot and the map m, represented by the equation:

$$P(z_t|\, x_t, \mathrm{m}) \;= P(z_t|\, m, x_t) \times\; P(m|\, x_t)$$

where $P(z_t|\, m, x_t)$ is the likelihood of the measurement given the map and the pose, and $P(m|\, x_t)$ is the prior probability of the map given the pose of the robot.

The posterior distribution over the robot pose and the map is represented by the equation:

$$P(x_t\,, m|\, z_{\{1:t\}}, u_{\{1:t\}}) \\ = \alpha \times\; P(z_t|\, x_t, \mathrm{m}) \times\; P(x_t|u_t, x_{t-1}) \;\times\; P(x_{t-1}\,, m|\, z_{\{1:t-1\}}, u_{\{1:t-1\}})$$

where $z_{\{1:t\}}$ and $u_{\{1:t\}}$ are the sensor measurements and control inputs up to time t, α is a normalization constant to ensure that the posterior is a valid probability distribution.

Advantages of GMapping using Kinect 360:

1- Fast and accurate mapping with a high-resolution 2D occupancy grid map.

2- Suitable for mapping large indoor environments with many features.

3- Can be run in real-time, allowing the robot to navigate in real-time.

Disadvantages of GMapping using Kinect 360:

1- Limited to 2D mapping, which can result in loss of information about the height of objects in the environment.

2- Can struggle with mapping environments with low contrast or reflective surfaces.

3- Requires a flat floor for accurate mapping,

4.7.3 3D Octomap

Octomap is a popular algorithm for 3D mapping using sensors like lidar and RGB-D cameras. It uses an octree data structure to represent the map, where each leaf node in the octree represents a small volume in the 3D space. The occupancy probability of each leaf node is updated based on the sensor data, resulting in a 3D occupancy grid map.

The RQT graph for 3D Octomap would typically involve a node for the sensor driver, a node for the Octomap algorithm, and a node for visualization, such as RViz. The communication between these nodes can be visualized using RQT graph.

Here's an example of the output of 3D Octomapping with Kinect 360 and its RQT graph:

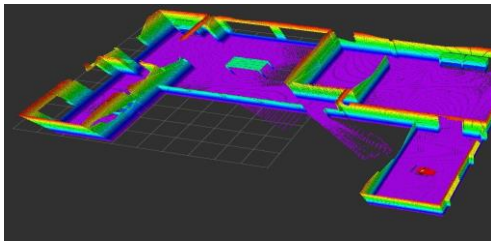

Figure 62 3D Octomapping

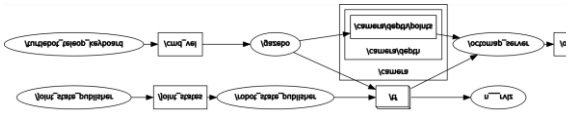

Figure 63 3D Octomapping RQT graph

The probability of occupancy of a voxel in Octomap can be represented using a binary Bayes filter. Assuming a binary occupancy state for a voxel, the Bayes filter update equation can be written as:

$$P_{(\text{Occupied} \mid \text{Measurement})} = P_{(\text{Measurement} \mid \text{Occupied})} \times \left(P_{(\text{Occupied})} / P_{(\text{Measurement})} \right)$$

where $P_{(\text{Occupied} \mid \text{Measurement})}$ is the posterior probability of the voxel being occupied given a measurement, $P_{(\text{Measurement} \mid \text{Occupied})}$ is the probability of observing a measurement given the voxel is occupied, $P_{(\text{Occupied})}$ is the prior probability of the voxel being occupied, and $P_{(\text{Measurement})}$ is the marginal probability of the measurement. Similarly, the posterior probability of a voxel being unoccupied can be computed as:

$$P_{(\text{Unoccupied} \mid \text{Measurement})} = P_{(\text{Measurement} \mid \text{Unoccupied})} \times \left(P_{(\text{Unoccupied})} / P_{(\text{Measurement})} \right)$$

The probability of occupancy can be further refined using the update rule:

$$P_{new} = (P_{old} \times N_{new} + \text{z}) / (N_{old} + 1)$$

Where P_{old} and P_{new} are the old and new probabilities of occupancy, N_{old} is the old number of observations, and z is the new observation.

Advantages of 3D Octomapping using Kinect 360:

1- Produces a detailed 3D map of the environment, which can provide more information about the height and shape of objects in the environment.

2- Can handle more complex environments with uneven terrain and objects of varying heights

3- Can provide more accurate results in environments with low contrast or reflective surfaces.

Disadvantages of 3D Octomapping using Kinect 360:

1- More computationally expensive than 2D mapping, which can slow down real-time mapping and navigation.

2- The 3D map can be harder to visualize and interpret compared to 2D occupancy grids.

3- May require more tuning of parameters to produce accurate results.

4.7.4 2D Octomap

2D Octomapping is like 3D Octomapping, but it generates a 2D occupancy grid map of the environment instead of a 3D map. It is also capable of generating a high-resolution map that represents the environment in detail.

Here's an example of the output of 2D Octomapping:

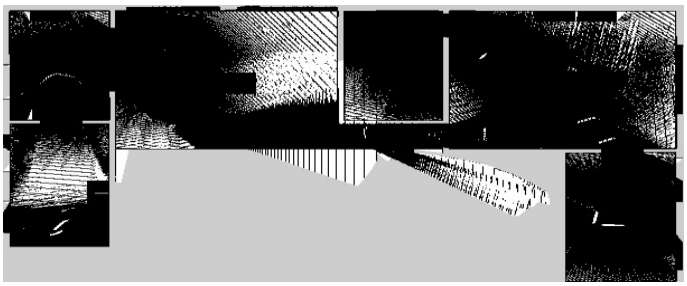

Figure 64 2D Octomapping

The 2D Octomap algorithm uses a similar approach to the 3D version but projects the sensor measurements onto a 2D plane. The occupancy probability of each cell in the grid map is modeled using a log-odds representation, where $$p_i$$ is the probability that a cell is occupied and $$m_i$$ is the measured occupancy probability at that cell:

$$\log \frac{p_i}{1-p_i} = \text{logit}(p_i) = \text{logodds}(m_i)$$

The occupancy probability of each cell is updated based on the sensor measurements and the previously estimated occupancy probabilities using Bayes' rule. The resulting map is a 2D grid map with each cell containing the estimated occupancy probability.

Advantages of 2D Octomap:

1- Provides higher resolution maps compared to other 2D mapping algorithms such as occupancy grid maps.

2- Can efficiently handle dynamic environments, as it allows for incremental map updates without the need to recompute the entire map.

3- Can handle environments with complex geometries and varying height levels.

4- Can be used for obstacle avoidance and path planning.

Disadvantages of 2D Octomap:

1- Requires more computational resources compared to other 2D mapping algorithms due to the additional computational complexity of the 3D representation.

2- The increased computational complexity can lead to longer processing times, making it unsuitable for real-time applications with high frequency sensor data.

4.7.5 RTAB mapping

RTAB-Map (Real-Time Appearance-Based Mapping) is an open-source software framework for simultaneous localization and mapping (SLAM) using RGB-D cameras, stereo cameras, or laser sensors. It was designed to provide a solution for mapping and localizing robots in dynamic and complex environments in real-time. RTAB-Map is known for its robustness in handling dynamic environments, its ability to build large-scale maps, and its compatibility with various sensors and robot platforms. In addition to mapping and localization, RTAB-Map also offers features such as loop closure detection, graph-based SLAM optimization, and 3D point cloud visualization.

Here's an example of the output RTAB mapping:

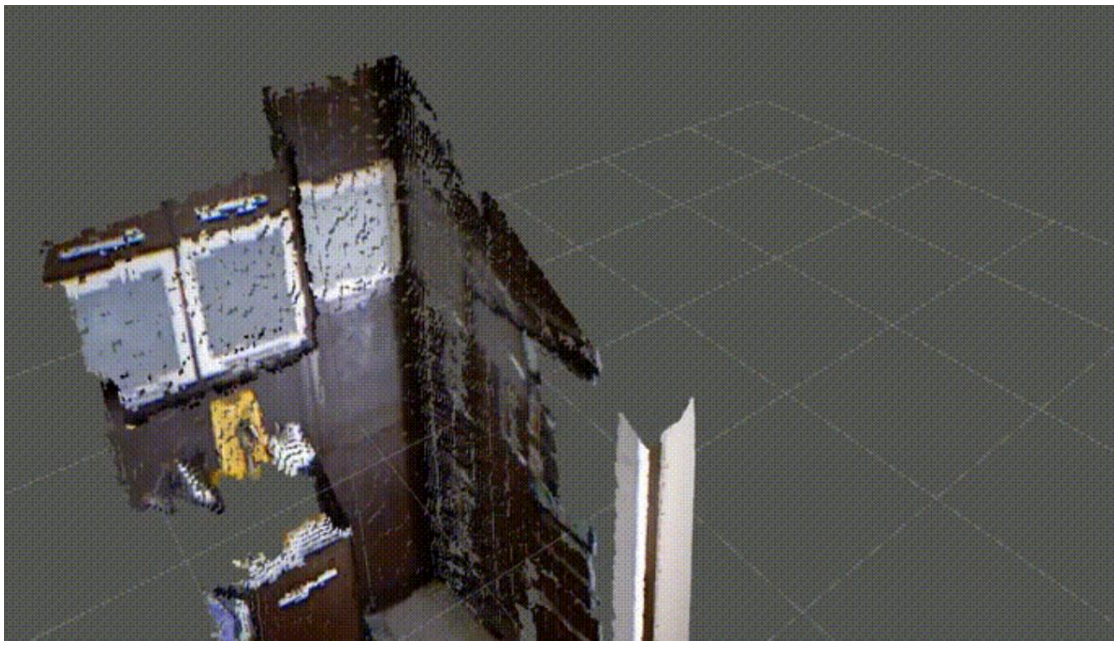

Figure 65 RTAB mapping

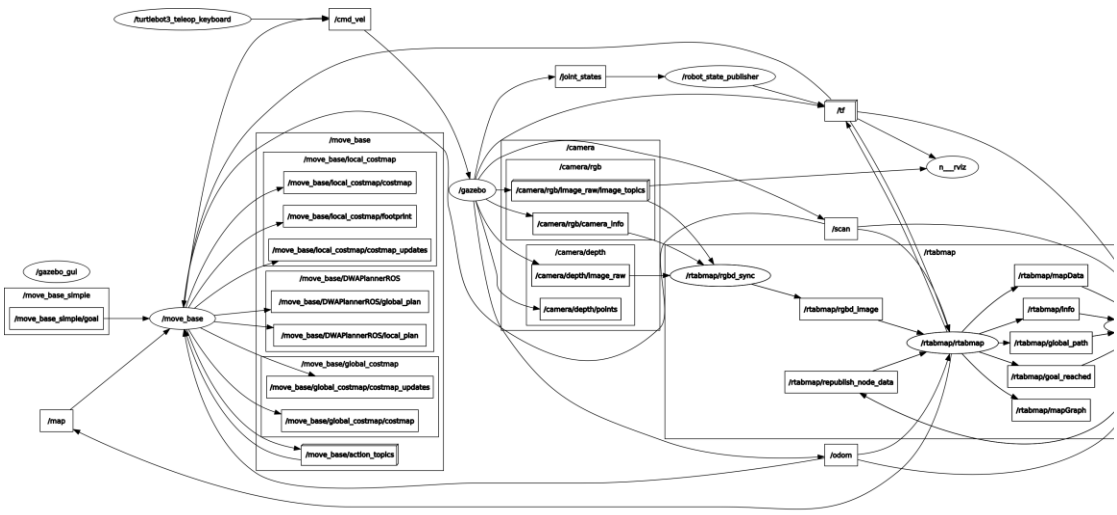

Figure 66 RQT graph for RTAB mapping with Kinect 360:

The Equations for the RTAB-Map Algorithm Are as Follows:

Feature extraction:

Keypoint detection: $$k_1, k_2, ..., k_n$$

Keypoint description: $$d_1, d_2, ..., d_n$$

Association: $$a_{i,j} = \begin{cases} 1, &\text{if}\ k_i\ \text{and}\ k_j\ \text{are matched}\ 0, &\text{otherwise} \end{cases}$$

Loop closure detection:

Bag-of-words (Bow) model: a vocabulary of visual words $$w_1, w_2, ..., w_m$$ is created from the keypoint descriptors

Inverted index: a database of images that contain each visual word is created

Query image: the current frame is matched against the database using the BoW model to retrieve the top-n candidate loop closures

Geometric verification: the top-n candidates are verified using a RANSAC algorithm to estimate the transformation between the current frame and the candidate loop closure

Pose graph optimization:

Nodes: each keyframe is a node in the graph, with its position and orientation represented by a 6D pose vector x_i

Edges: the loop closures between keyframes are represented by edges in the graph, with their relative transformation represented by a 6D pose vector $x_{i,j}$

Cost function: the optimization problem is formulated as a non-linear least squares problem, with the goal of minimizing the sum of the squared errors between the observed and predicted relative poses

Map update:

After the optimization step, the optimized poses are used to update the map by reprojecting the points in each keyframe onto the map and updating their occupancy values.

Advantages:

1- RTAB can create both 2D and 3D maps, depending on the input sensor data and the chosen settings.

2- RTAB can perform real-time SLAM, meaning it can update the map as the robot moves through the environment, rather than requiring pre-mapped areas or a fixed environment.

3- RTAB uses loop closure detection and relocalization, which helps improve the accuracy and consistency of the map over time.

4- RTAB can also provide estimates of the robot's position and orientation (i.e. pose), which can be useful for navigation and control tasks.

Disadvantages:

1- RTAB requires more computational resources than some other mapping algorithms, particularly for larger environments or higher resolutions.

2- The quality of the map created by RTAB can depend heavily on the quality and accuracy of the input sensor data, particularly for 3D mapping.

3- RTAB may not work well in environments with few or no distinguishable features or landmarks, as it relies on visual cues for loop closure detection.

4- RTAB can be more complex to set up and configure compared to some other mapping algorithms.

4.7.6 Quick comparison and conclusion:

Algorithm	2D Octomap	3D Octomap	GMapping	RTAB Map
Map Type	2D	3D	2D	3D
Environment	Indoor	Indoor	Indoor	Indoor
Real-time	Yes	No	Yes	No
Accuracy	High	High	Medium	High
Computational Load	Low	High	Medium	High
Map Quality	High	High	Medium	High
Limitations	No 3D data	Large memory usage	Limited to planar surfaces	Requires well-calibrated sensor
Advantages	Fast, accurate, low memory usage	Detailed 3D map, useful for object detection	Suitable for planar surfaces, fast	Robust, supports loop closures
Used in applications	Mobile robots, UAVs, navigation	Object detection, inspection, 3D modelling	SLAM, navigation	Autonomous driving, robotics research

Table 4 Quick comparison between difference mapping algorithms

- In conclusion, we have explored various mapping algorithms for robotics applications, including GMapping, RTAB-Map, and 2D Octomap. Each algorithm has its own strengths and weaknesses, making them suitable for different scenarios.

- GMapping is a popular choice for 2D mapping in mobile robot navigation due to its simplicity and efficiency. It relies on the data from a laser range finder to build occupancy grid maps

- RTAB-Map stands out with its simultaneous mapping and localization capabilities. It combines data from various sensors, performs loop closure detection, and builds 3D maps. This algorithm is well-suited for real-time mapping applications that require accurate localization.

- 2D Octomap is specifically designed for 2D mapping in indoor environments. It provides high-resolution occupancy grid maps and efficient collision checking for navigation. This algorithm is computationally efficient and suitable for mapping indoor spaces.

- In addition to these algorithms, 3D Octomap is a powerful mapping solution for 3D environments. It constructs volumetric maps by integrating sensor measurements from sources such as lidar or depth cameras. 3D Octomap offers a detailed representation of the environment, enabling collision checking and path planning in 3D.

According to Table 4, our selection is RTAB mapping. This algorithm is good and will be discussed in detail in the next chapters.

4.8 2D Octomapping in details

4.8.1 Understanding 2D Octomapping

2D Octomapping is an advanced algorithm that builds upon the concept of occupancy grids. It represents the environment as a grid of cells, where each cell corresponds to a specific location and is assigned a probability value indicating the likelihood of occupancy. Unlike traditional occupancy grids, 2D Octomapping utilizes an octree data structure to represent each cell, enabling a more detailed representation of space.

The octree structure allows for adaptive resolution, which means that the map can have higher resolution in areas with more detailed information, such as regions close to the robot, while using coarser resolution in less significant regions, such as areas far away from the robot. This adaptive resolution provides several benefits, including efficient memory usage and faster map updates. By dynamically adjusting the resolution based on the level of detail required in different areas, 2D Octomapping ensures an accurate representation of the environment while optimizing resource consumption.

4.8.2 The Mapping Process

The mapping process with 2D Octomapping involves multiple steps:

- **Sensor Data Acquisition:** The first step is to acquire sensor data, typically from a Kinect V1 sensor. The Kinect V1 sensor provides depth measurements, allowing the algorithm to perceive the distance to objects in the environment. The sensor scans the surroundings and captures depth information for each point in its field of view.

- **Data Preprocessing:** The acquired sensor data undergoes preprocessing to remove outliers, noise, and unwanted reflections. This step helps to enhance the quality and reliability of the data. Additionally, the data may undergo transformations to align it with the robot's coordinate system, ensuring accurate mapping.

- **Occupancy Update:** Using the preprocessed sensor data, the algorithm updates the occupancy of each cell in the map. Cells that intersect with the measured depth points are marked as occupied, indicating the presence of obstacles, while cells that are unobstructed or not measured are marked as free. This update process takes into account the probability values associated with each cell and adjusts them accordingly.

- **Octree Update:** The occupancy updates are propagated through the octree structure. The algorithm recursively traverses the octree and updates the occupancy probabilities of cells at different levels of detail. This allows for an efficient representation and storage of the map, where regions of interest have higher resolution while less significant regions have lower resolution.

- **Map Visualization:** The final step involves visualizing the occupancy grid map. The map provides a 2D representation of the environment, with cells color-coded to indicate their occupancy probabilities. This visualization helps in understanding the generated map and aids in subsequent tasks, such as path planning and obstacle avoidance.

4.8.3 Advantages of 2D Octomapping

When using a Kinect V1 sensor for mapping indoor environments, 2D Octomapping offers several advantages over GMapping:

1- **Higher Resolution:** 2D Octomapping provides a higher resolution representation of the environment compared to GMapping. The adaptive octree structure allows for more detailed maps that capture small-scale features and obstacles. This level of detail is crucial for accurate navigation and path planning.

2- **Adaptive Resolution:** The adaptive nature of 2D Octomapping optimizes memory usage and computational efficiency. By adjusting the resolution based on the level of detail required in different areas, the algorithm ensures efficient utilization of available resources. This is particularly advantageous when using sensors with limited range, such as the Kinect V1, as it allows for effective representation of the available data.

3- **Handling Sparse Data:** 2D Octomapping handles sparse data, where there are fewer depth measurements, more effectively than GMapping. The octree structure in 2D Octomapping helps to better handle sparse data by allowing for a more detailed representation of the environment. Even with limited depth measurements from the Kinect V1 sensor, the algorithm can accurately estimate the occupancy probabilities of cells based on the available data. This capability is particularly useful in large indoor environments where the sensor's range may not cover the entire space.

4- **More Accurate Mapping:** Due to its higher resolution and adaptive nature, 2D Octomapping can provide more accurate mapping results compared to GMapping. The octree structure allows for a finer-grained representation of obstacles and their boundaries, resulting in a more precise depiction of the environment. This level of accuracy is crucial for tasks such as localization, path planning, and object detection.

5- **Efficient Resource Utilization:** With its adaptive resolution and efficient memory usage, 2D Octomapping ensures optimal resource utilization. By adjusting the resolution based on the level of detail required, the algorithm minimizes memory consumption while still capturing the essential information needed for navigation. This efficiency is beneficial, especially in resource-constrained robotic systems.

In conclusion, when using a Kinect V1 sensor for mapping indoor environments, 2D Octomapping offers significant advantages over GMapping. Its higher resolution, adaptive resolution capabilities, ability to handle sparse data, more accurate mapping results, and efficient resource utilization make it a suitable choice for applications that require detailed and reliable mapping. By leveraging the strengths of 2D Octomapping, robotic systems can enhance their perception capabilities and perform tasks such as navigation, obstacle avoidance, and environment understanding with improved accuracy and efficiency.

4.9 **RTAB In Details**

RTAB-Map is a graph-based simultaneous localization and mapping (SLAM) algorithm that can create 3D maps of environments while localizing a robot within them. It is an open-source solution that is compatible with many sensors and robots, including the Kinect 360 that you used.

RTAB-Map operates by constructing a graph where each node represents a keyframe, which is an estimate of the robot's position and orientation at a given time. The edges between the keyframes represent the relative pose between them. The algorithm uses loop closure detection to identify and correct errors in the graph, which can arise from drift in the robot's localization.

The key to RTAB-Map's effectiveness is its ability to fuse information from multiple sensors, including visual, RGB-D, and laser data. This allows it to create accurate 3D maps even in environments with limited sensor coverage or challenging lighting conditions. Additionally, RTAB-Map can be configured to run in real-time on a robot, making it suitable for applications where speed is critical.

To use RTAB-Map with the Kinect 360, you would first need to install the necessary dependencies and then set up your ROS workspace to include the RTAB-Map package. Once you have launched the RTAB-Map node and specified the Kinect 360 as the input source, you can begin to move the robot around the environment to create a 3D map. The algorithm will automatically detect loop closures and update the map as needed. RTAB has many configurations that will be mentioned below.

4.9.1 Kinect + Odometry + 2D Laser

In this configuration, we assume that you have a wheeled robot constrained to the plane XY with only rotation on yaw (around z-axis).

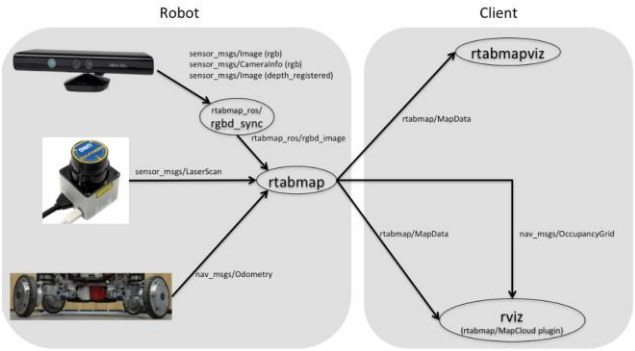

Figure 67 Kinect + Odometry + 2D laser RTAB configuration

4.9.2 Kinect + Odometry + Fake 2D Laser from Kinect

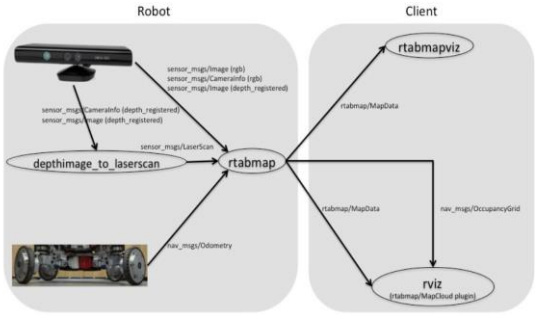

Figure 68 Kinect + Odometry + Fake 2D laser from Kinect RTAB configuration

If you don't have a laser and you want to create a 2D occupancy grid map (for which laser scans are required), you can simulate a 2D laser with the depth image from the Kinect using depthimage_to_laserscan ros-pkg. This was the configuration used for the IROS 2014 Kinect Contest. This could be also used with a robot like the TurtleBot.

4.9.3 Kinect + 2D Laser

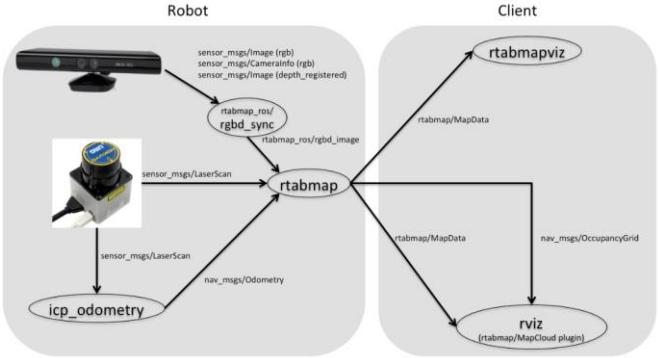

Figure 69 Kinect + 2D laser RTAB configurations

In you don't have odometry, you can create one using the 2D laser scans and ICP. Once you have this new odometry node, you can do the same steps as above (with odometry). To generate odometry from laser scans, look at these packages: laser_scan_matcher or hector_slam. NEW RTAB-Map has now its own icp_odometry node.

4.9.4 2D Laser Only

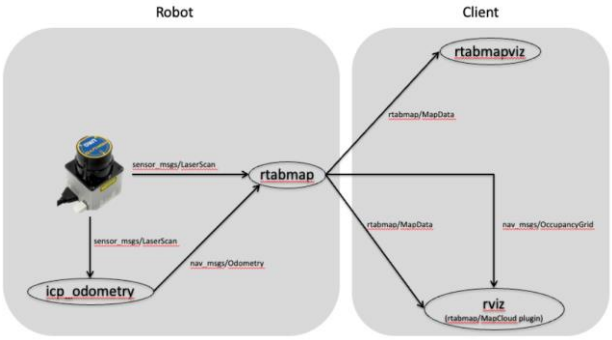

Figure 70 2D laser only RTAB configurations

This is the same example as above with Kinect + Laser but without a camera. If you have only a lidar without wheel odometry, you could try this setup. Obviously without a camera, you lose the ability to detect global loop closures or globally localize using vision. If odometry doesn't drift too much, proximity detections can still be detected to correct odometry over time.

4.9.5 2D Laser + Wheel Odometry as Guess

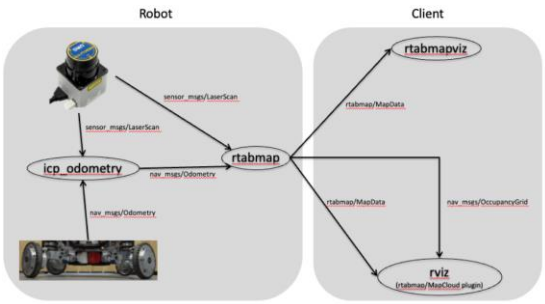

Figure 71 laser + Wheel Odometry RTAB configuration

This is the same example as above with Laser only but here we use wheel odometry as guess for icp_odometry. This will make icp_odometry more robust to corridor-like environments with short-range lidars.

4.9.6 Kinect + Odometry

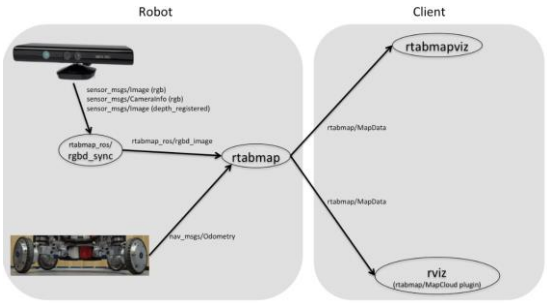

Figure 72 Kinect + Odometry RTAB configuration

In this configuration, we assume that you have a robot not constrained to a single plane (like UAV), which can move in XYZ and roll, pitch, yaw rotations.

4.9.7 Kinect

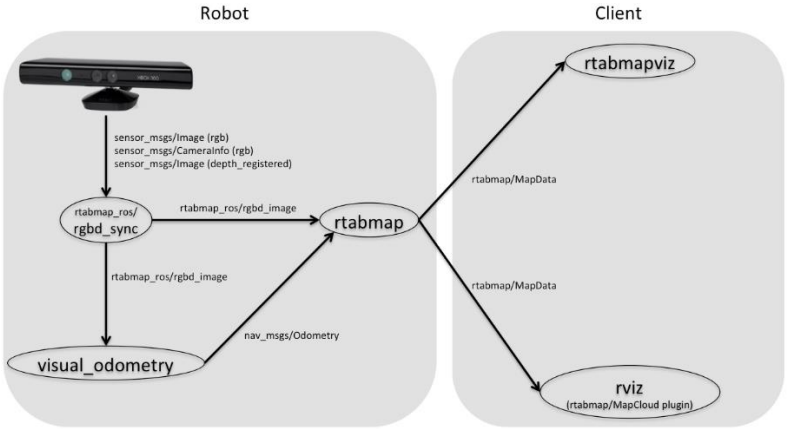

Figure 73 RTAB Kinect configuration

If you can't have a reliable odometry, you can map using only RTAB-Map at the cost of "lost odometry. The robot may detect this "lost" state when a null odometry message is sent by the rgbd_odometry node.

4.10Mapping Implementation

4.10.1 2D Octomapping Implementation

To implement 2D Octomapping using the Kinect V1 sensor, follow the steps below:

1- Install the necessary packages: Begin by installing the required packages for the Kinect sensor and Octomapping in ROS. Open a terminal and enter the following command.

```
sudo apt install ros-melodic-freenect-stack
sudo apt install ros-melodic-octomap-mapping
```

2- Connect the Kinect to your computer: Connect the Kinect V1 sensor to your computer using the appropriate USB cable.
3- Launch the Kinect driver: Launch the Kinect driver using the freenect_launch package to start receiving data from the sensor. Open a new terminal and run the following command.

```
roslaunch freenect_launch freenect.launch depth_registration:=true
```

4- Launch the 2D Octomapping node: Open a new terminal and launch the octomap_mapping node from the octomap_mapping package. This node will perform the 2D Octomapping using the data from the Kinect sensor. Enter the following command:

```
roslaunch octomap_mapping octomap_mapping.launch
```

5- Adjust the resolution (optional): By default, the 2D Octomapping node uses a certain resolution for the generated occupancy grid. If you want to change the resolution, you can modify the octomap_mapping.launch file and adjust the resolution parameter accordingly. Higher resolutions will provide more detailed maps but require more computational resources.
6- Move the Kinect around: Once the mapping node is running, move the Kinect sensor around the environment you want to map. Make sure to cover as much area as possible and capture different perspectives. As you move the sensor, the mapping node will continuously update the occupancy grid based on the depth measurements from the Kinect.
7- Save the map: After capturing the desired environment, you can save the generated occupancy map for later use. Use the following command to save the map:

```
rosrun map_server map_saver -f ~/map map:=projected_map
```

By following these steps, you can implement 2D Octomapping using the Kinect V1 sensor in ROS. The algorithm will generate an occupancy grid map based on the depth data from the sensor, providing a detailed representation of the environment. Remember to adjust the resolution according to your project requirements to balance the level of detail and computational resources.

4.10.2 RTAB Implementation

In a previous section, we talked about RTAB mapping, a powerful algorithm that can be used for simultaneous localization and mapping (SLAM) in robotics applications. Now, it's time to put theory into practice and implement RTAB mapping with the Kinect 360 sensor. In this section, we will go through the steps needed to install RTAB mapping and the necessary drivers for the Kinect 360 sensor. We will also cover how to visualize the output of RTAB mapping in both RViz and rtabmapviz, and how to interpret the generated maps. So, let's dive into the implementation of RTAB mapping and see how it performs in the real world.

So out steps will be:

First things first, let’s get the Kinect working and send data through ROS using the libfreenect library. It is integrated into ROS as the package freenect_stack. This tutorial was made using the original Kinect v1 for Xbox,

1- Install the libfreenect stack: (driver for Kinect sensor which compatible with ROS)

```
sudo apt install ros-melodic-freenect-stack
```

2- Connect the Kinect to your computer and test the Kinect stack by running the convenient launch file (depth registration refers to the pairing of color data with depth points):

```
roslaunch freenect_launch freenect.launch depth_registration:=true
```

3- Install the rtabmap package for ROS

```
sudo apt install ros-melodic-rtabmap-ros
```

4- Due to a bug in rtabmap, you need to specify the location of a particular library by adding a variable to your bash session and restart your terminal:

```
echo export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/ros/kinetic/lib/x86_64-linux-gnu >> ~/.bashrc
```

5- Now start the rtabmap mapping tool. There are 2 optional interfaces, one in rviz and another specific to rtabmap. You can launch the rviz one with this command:

```
roslaunch rtabmap_ros rgbd_mapping.launch rtabmap_args:="--delete_db_on_start" rviz:=true rtabmapviz:=false
```

The non-rviz one is launched with this command:

```
roslaunch rtabmap_ros rgbd_mapping.launch rtabmap_args:="--delete_db_on_start"
```

6- Once the imagery loads, move the Kinect around with your hands, and rtabmap will update the position of the camera. If you move the camera too fast, it may lose track of where it is, and you will have to return it to point where it was so it can regain odometry.

AMR (Autonomous Mobile Robot)

Real-world output:

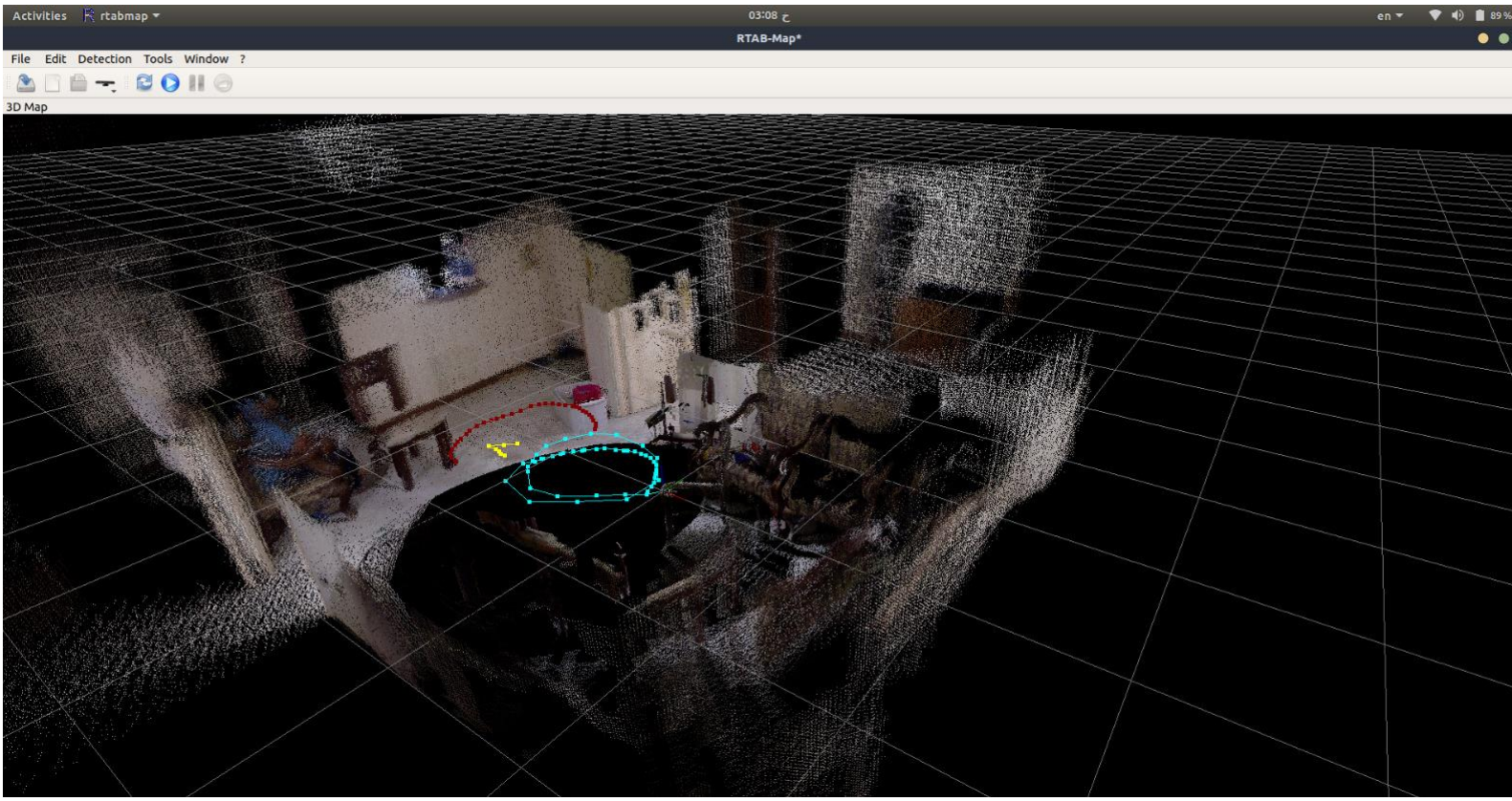

Figure 74 RTAB-mapping of a room in a house

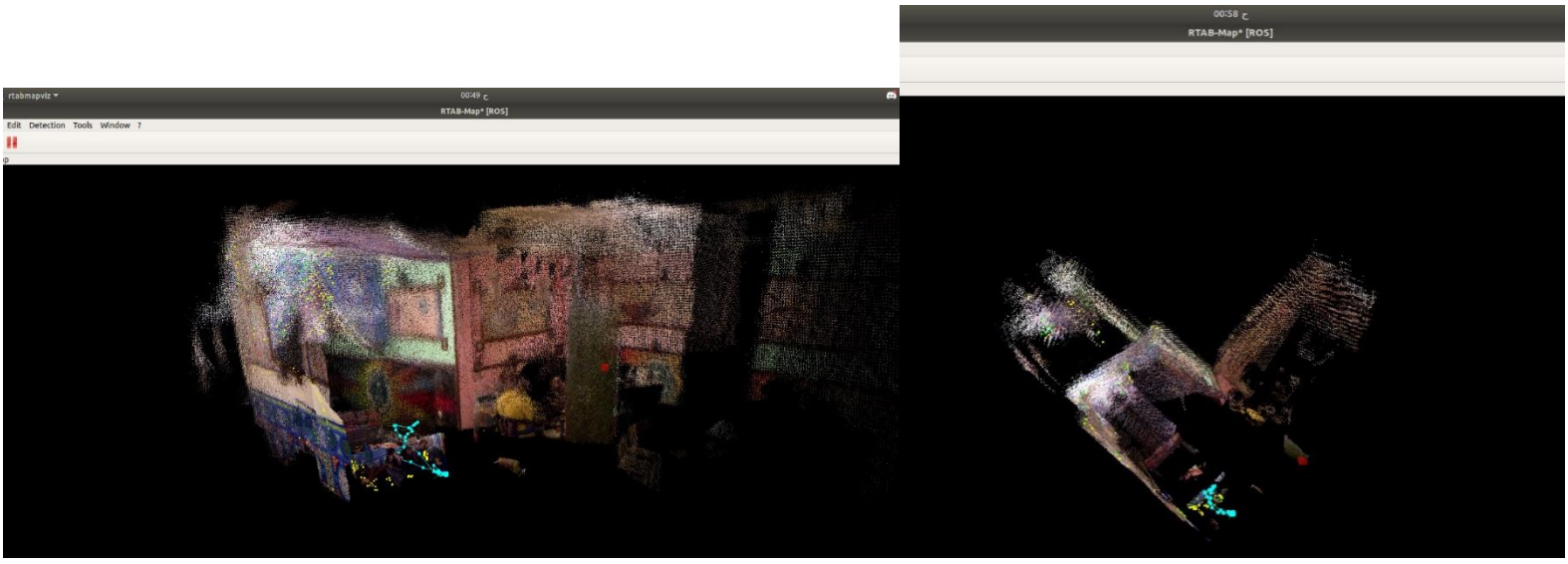

Figure 75 RTAB-mapping of a section of a café

4.11 Localization Analysis and Implementation

In robotics, localization is the process of determining the position and orientation of a robot in its environment. It is a fundamental problem in robotics that is necessary for a robot to perform various tasks such as navigation, mapping, and manipulation. In this chapter, we will explore different localization methods used in robotics, including algorithms and sensors, with a focus on the Adaptive Monte Carlo Localization (AMCL) algorithm.

4.11.1 Odometry-Based Localization

Odometry is a method used by robots to estimate their position by measuring wheel rotation and distance traveled. This method is commonly used in mobile robots, especially those with wheels. The robot uses the data from its encoders to estimate its position and orientation in space. Odometry-based localization is a simple and effective method, but it suffers from cumulative errors that can cause the robot's estimated position to drift over time.

Advantages:

- Simple and cost-effective method using wheel encoders.
- Suitable for indoor environments with relatively flat surfaces.
- Can provide a reasonable estimate of position and orientation.

Disadvantages:

- Prone to cumulative errors and drift over time.
- Wheel odometry may be affected by wheel slippage on certain surfaces.
- Limited accuracy and reliability, especially in environments with uneven or slippery floors.

4.11.2 Inertial-Based Localization

Inertial-based localization uses sensors that measure the acceleration and rotation of the robot to estimate its position and orientation. These sensors include accelerometers and gyroscopes. By integrating the data from these sensors over time, the robot can estimate its position and orientation relative to a starting position. Inertial-based localization is often used in conjunction with other localization methods to improve accuracy.

Advantages:

- Provides accurate and real-time estimation of position and orientation.
- Can compensate for wheel slippage and uneven surfaces.
- Works well in environments where odometry alone may not be reliable.

Disadvantages:

- Requires sophisticated sensor fusion algorithms to integrate data from accelerometers and gyroscopes.
- Can be affected by sensor biases and noise.
- Initial calibration and setup may be required for optimal performance.

4.11.3 Visual-Based Localization

Visual-based localization uses cameras to estimate the position and orientation of the robot. The camera captures images of the robot's environment, and the robot uses these images to estimate its position and orientation. This method is often used in combination with other localization methods to provide a more accurate estimate of the robot's position. Popular visual-based localization algorithms include feature extraction, image matching, and simultaneous localization and mapping (SLAM).

Advantages:

- Kinect v1 provides depth information along with RGB imagery, enabling 3D perception.
- Can capture rich environmental information in warehouses.
- Works well for detecting objects, obstacles, and features.

Disadvantages:

- Kinect v1 has limited range and field of view compared to other depth sensors.
- Performance can be affected by lighting conditions and occlusions.
- Limited accuracy for long-range measurements.

4.11.4 Sensor Fusion and the Kalman Filter

Sensor fusion plays a crucial role in localization. The Kalman filter is a widely used algorithm for sensor fusion that combines noisy sensor measurements with a dynamic system model to estimate the robot's state (position and orientation). The Kalman filter uses a prediction step to propagate the robot's state forward in time based on motion models, and an update step to correct the predicted state using sensor measurements.

4.11.5 Laser-Based Localization

Laser-based localization uses a laser range finder or LIDAR (Light Detection and Ranging) to estimate the position and orientation of the robot. The laser sends out a beam of light that bounces off objects in the environment and returns to the robot's sensor. By analyzing the time it takes for the beam to travel and return, the robot can estimate its position and orientation in the environment. Laser-based localization is particularly useful for applications that require accurate mapping of the environment. Popular algorithms used in laser-based localization include Monte Carlo Localization (MCL) and Iterative Closest Point (ICP).

Advantages:

- Laser sensors, such as LIDAR, can provide accurate and precise measurements of the environment.
- Suitable for mapping and localization in complex and cluttered indoor environments.
- Can handle a wide range of surfaces and objects.

Disadvantages:

- Laser sensors, including LIDAR, can be relatively expensive compared to other sensors.
- Limited range and field of view depending on the specific sensor.
- Performance may be affected by dust, fog, and reflective surfaces in warehouses.

4.11.6 Adaptive Monte Carlo Localization (AMCL)

AMCL is a particle filter-based algorithm that leverages the power of sensor fusion to achieve robust and accurate localization. It maintains a set of particles, where each particle represents a hypothesis of the robot's pose. The particles are sampled from a prior distribution, and their weights are updated based on the likelihood of the sensor measurements given the particles' poses.

Advantages:

- AMCL is a probabilistic localization algorithm that can handle uncertainties effectively.
- Can work well with a combination of sensor inputs, including wheel encoders, Kinect data, and possibly LIDAR.
- Able to recover from localization failures and handle multi-modal distributions.

Disadvantages:

- Requires accurate sensor models and parameters for optimal performance.
- Performance can be affected by sensor noise and environmental changes.
- Computationally intensive and may require a higher-performance computing system for real-time operation.

According to advantages and disadvantages we decided to use AMCL localization.

4.12 Adaptive Monte Carlo Localization (AMCL) in details

4.12.1 The Algorithm Consists of The Following Steps

1- **Initialization:** AMCL starts with an initial set of particles distributed uniformly across the environment. Each particle has an associated weight.

2- **Prediction:** In the prediction step, the particles are propagated forward using motion models that capture the robot's dynamics. This is like the odometry-based localization, where each particle's pose is updated based on the estimated robot motion.

3- **Measurement Update:** In the measurement update step, the weights of the particles are adjusted based on the likelihood of the sensor measurements given the particles' poses. This is where the sensor fusion takes place. The sensor model defines how likely a given measurement is given a particular particle's pose. The weights of the particles are updated accordingly, with higher weights assigned to particles that have poses consistent with the sensor measurements.

4- **Resampling:** After the measurement update, resampling is performed to select a new set of particles for the next iteration. The resampling step aims to give higher probabilities to particles with higher weights, effectively discarding particles with low weights. This way, the particles representing more accurate hypotheses are retained, while the less probable particles are replaced.

5- **Adaptation:** AMCL dynamically adjusts the number of particles based on the environment and robot motion. When the robot is in a region of high uncertainty, such as areas with limited sensor information or ambiguous features, AMCL increases the number of particles to explore the space more thoroughly. In regions of low uncertainty, where the robot has reliable sensor data and clear landmarks, AMCL reduces the number of particles to focus computational resources on the most relevant areas.

6- **Handling the "Kidnapped Robot" Problem:** AMCL addresses the "kidnapped robot" problem, where the robot is suddenly and forcefully moved to a different location. To recover from such situations, AMCL incorporates random global localization and resampling techniques. When a significant discrepancy occurs between the predicted and observed measurements, AMCL performs global localization by uniformly distributing particles across the entire environment. This allows the algorithm to recover the robot's position and orientation even after such a disturbance.

4.12.2 Equations and Implementation Details

The AMCL algorithm involves several mathematical equations and implementation details. Let's take a closer look at some of the key components:

1. **Particle Representation:**
   - Each particle represents a hypothesis of the robot's pose, consisting of position (x, y) and orientation (θ).
   - The particle set is denoted as P = $\{(x_i, y_i, \theta_i, w_i)\}$, where $(x_i, y_i, \theta_i, w_i)$represents the pose of particle i , and w_i is its weight.
2. **Step:**
   - Given the robot's motion commands $(\Delta x, \Delta y, \Delta\theta)$ from the odometry sensors, the new pose of particle i is updated as follows:

$x_i' = x_i + \Delta x + \eta_X \times y_i' = y_i + \Delta y + \eta_y \times \theta_i' = \theta_i + \Delta\theta + \eta_\theta$

   - η_x, η_y, and η_θ are noise terms representing uncertainty in the motion model.
3. **Measurement Update Step:**
   - The weight of each particle is updated based on the likelihood of the sensor measurements given its pose.
   - The weight w_i is computed as the product of the individual measurement likelihoods for each sensor reading.
   - For example, if we have range measurements z_1, z_2, ..., z_n the weight update equation for particle i is:

$w_i = w_i \times P(z_1|x_i', y_i', \theta_i') \times P(z_2|x_i', y_i', \theta_i') \times \ldots \times P(z_n|x_i', y_i', \theta_i')$

   The sensor model $P(z|x_i, y_i, \theta_i)$ defines the likelihood of obtaining measurement z given the pose (x, y, θ).
4. **Resampling Step:**
   - After the measurement update, particles with higher weights have a higher chance of being selected.
   - The resampling step is performed using a resampling algorithm such as the low variance resampling (LVR) method.
   - Resampling involves selecting particles with replacement based on their weights, resulting in a new set of particles.
5. **Adapting Particle Set Size:**
   - The number of particles in the set can be adjusted dynamically based on the environment and robot motion.
   - The adaptive mechanism allows AMCL to allocate more particles in regions of high uncertainty and fewer particles in regions of low uncertainty.
   - The specific adaptive strategy can vary depending on the implementation and requirements of the robot.

The following image shows the previous steps in simulated environment with gazebo.

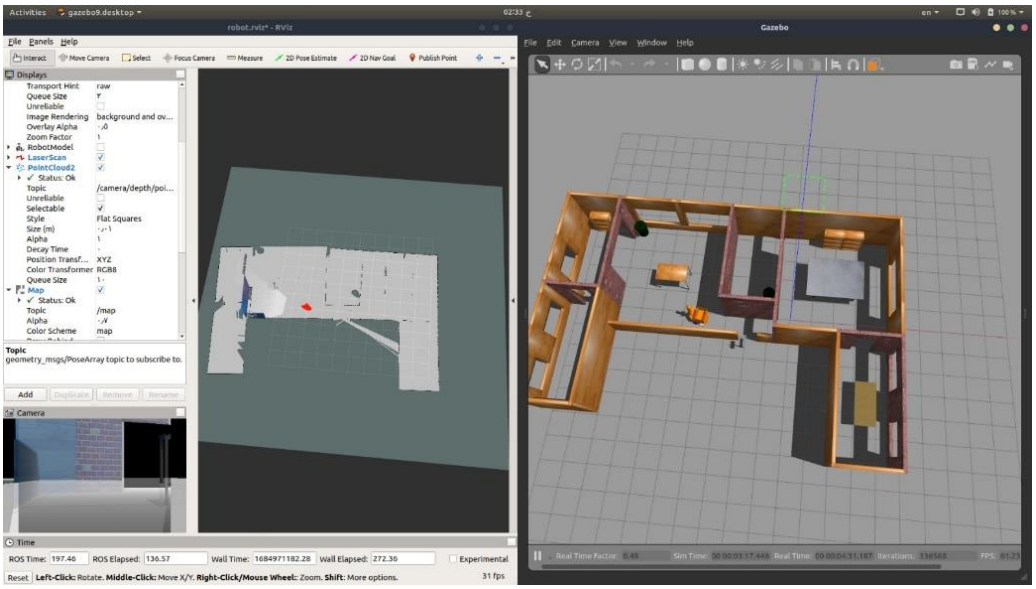

Figure 76 simulated environment with gazebo

And the follow launch file describe AMCl implementation in ROS

```
<launch>
  <!-- Map Server Arguments -->
    <arg name="map_file" default="$(find mybot_nav)/map/map.yaml"/>
    <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)"
    />
  <!-- Arguments -->
  <arg name="scan_topic"     default="scan"/>
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
  <arg name="initial_pose_a" default="0.0"/>
  <!-- AMCL -->
  <node pkg="amcl" type="amcl" name="amcl">
    <param name="min_particles"             value="500"/>
    <param name="max_particles"             value="3000"/>
    <param name="kld_err"                   value="0.02"/>
    <param name="update_min_d"              value="0.20"/>
    <param name="update_min_a"              value="0.20"/>
    <param name="resample_interval"         value="1"/>
    <param name="transform_tolerance"       value="0.5"/>
    <param name="recovery_alpha_slow"       value="0.00"/>
    <param name="recovery_alpha_fast"       value="0.00"/>
    <param name="initial_pose_x"            value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"            value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"            value="$(arg initial_pose_a)"/>
    <param name="gui_publish_rate"          value="50.0"/>
    <remap from="scan"                      to="$(arg scan_topic)"/>
    <param name="laser_max_range"           value="3.5"/>
    <param name="laser_max_beams"           value="180"/>
    <param name="laser_z_hit"               value="0.5"/>
    <param name="laser_z_short"             value="0.05"/>
    <param name="laser_z_max"               value="0.05"/>
    <param name="laser_z_rand"              value="0.5"/>
    <param name="laser_sigma_hit"           value="0.2"/>
    <param name="laser_lambda_short"        value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="laser_model_type"          value="likelihood_field"/>
    <param name="odom_model_type"           value="diff"/>
    <param name="odom_alpha1"               value="0.1"/>
    <param name="odom_alpha2"               value="0.1"/>
    <param name="odom_alpha3"               value="0.1"/>
```

```
    <param name="odom_alpha4"              value="0.1"/>
    <param name="odom_frame_id"            value="odom"/>
    <param name="base_frame_id"            value="base_link"/>
  </node>
</launch>
```

4.12.3 AMCL and Kalman Filter Connection

AMCL incorporates the principles of the Kalman filter to achieve robust and accurate localization. The prediction step in AMCL corresponds to the state prediction in the Kalman filter, where the robot's motion model is used to estimate the new state. Similarly, the measurement update step in AMCL is analogous to the state correction step in the Kalman filter, where the sensor measurements are used to refine the state estimate.

By fusing odometry data, sensor measurements, and adaptive particle sampling, AMCL can achieve localization even in complex and dynamic environments. It is widely used in robotics applications that require accurate and robust position estimation.

In conclusion, Adaptive Monte Carlo Localization (AMCL) is a powerful algorithm for robot localization. By combining odometry data, sensor measurements, and adaptive sampling, AMCL can estimate the robot's position and orientation accurately. The algorithm leverages the principles of the Kalman filter to perform prediction and measurement update steps. With its ability to handle uncertainties, adapt to changing environments, and recover from sudden disturbances,

4.13 Navigation Stack: Autonomous Navigation in Indoor Warehouses

In the previous chapters, we discussed the process of mapping your indoor warehouse environment and implementing the Adaptive Monte Carlo Localization (AMCL) algorithm for robot localization. Now, we will explore the Navigation Stack, a powerful framework that enables autonomous navigation in complex environments.

The Navigation Stack is a collection of ROS packages that work together to provide a robust and flexible navigation system for mobile robots. It incorporates localization, path planning, and control components to enable a robot to autonomously navigate from one location to another while avoiding obstacles.

4.13.1 Overview of the Navigation Stack

The Navigation Stack consists of several key components:

- **Map Server:** The Map Server provides access to the generated map of the environment. It offers the occupancy grid map, which represents the known obstacles and free space in the warehouse.

- **AMCL:** The AMCL node, which we implemented in` the previous chapter, provides localization estimates to the Navigation Stack. It uses sensor measurements and the generated map to estimate the robot's pose accurately.

- **Path Planning:** The Path Planning component is responsible for generating a collision-free path from the robot's current location to the goal location. It takes into account the map, obstacles, and any additional constraints to compute an optimal path.

- **Global Planner:** The Global Planner, often based on Dijkstra's algorithm or A* search, plans the initial global path from the starting location to the goal. It considers the entire map and generates a coarse path that avoids obstacles.

- **Local Planner:** The Local Planner is responsible for local obstacle avoidance and smooth navigation along the planned path. It uses sensor data and the robot's velocity commands to reactively navigate around dynamic obstacles in real-time.

- **Costmap:** The Costmap is a grid representation of the environment that combines information from the generated map, sensor measurements, and obstacles. It provides a dynamic representation of the robot's surroundings and is used by the planners for path computation and obstacle avoidance. The following image describe navigation stack structure.

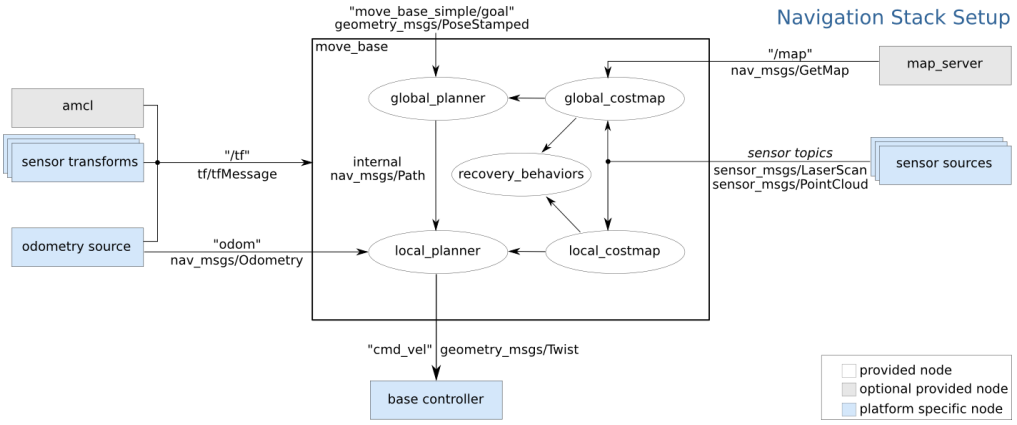

Figure 77 Navigation stack setup

4.13.2 Setting Up the Navigation Stack

To use the Navigation Stack in your indoor warehouse robot system, follow these steps:

- **Create a Launch File:** Create a new launch file, let's name it navigation.Launch, to launch the necessary nodes and configure the Navigation Stack components. In this launch file, include the Map Server, AMCL, Path Planning, Global Planner, Local Planner, and any additional nodes or parameters specific to your setup.

- **Configure the Costmap:** Set up the costmap parameters to define the inflation radius, resolution, and other properties based on your robot's size and navigation requirements. Adjust the parameters to ensure that the robot avoids obstacles and stays within acceptable boundaries.

- **Define Navigation Goals:** Define the navigation goals for your robot. These can be specified manually, received from an external system, or obtained through human interaction. The goals should be in the form of pose information, including the position and orientation in the map frame.

4.13.3 File Structure

- my_robot_package/
- launch/
- navigation.launch (your Navigation Stack launch file)
- maps/
- map.yaml (generated map file)
- map.pgm (occupancy grid map file)
- src/
- amcl_params.yaml (AMCL parameter configuration file)
- costmap_params.yaml (Costmap parameter configuration file)
- global_planner_params.yaml (Global Planner parameter configuration file)
- local_planner_params.yaml (Local Planner parameter configuration file)

4.13.4 Implement Autonomous Navigation

- Load Map: Load the generated map using the Map Server or a similar method. This step is crucial for the navigation stack to have access to the environment's map.

- Set Up AMCL: Set up the AMCL node to provide accurate localization estimates to the navigation stack. Configure the AMCL parameters based on your robot's sensors and the environment. Ensure that the AMCL parameters align with the ones specified in the amcl_params.yaml file.

- Configure Path Planning: Set up the Global Planner and Local Planner components for path planning and obstacle avoidance. Load the planner parameters from the corresponding parameter files (global_planner_params.yaml and local_planner_params.yaml) and adjust them as per your requirements.

- Create Navigation Goals: Implement a way for the robot to receive navigation goals. This can be done through a user interface, predefined goal poses, or any other method suitable for your application. Convert the received goal poses to the appropriate format for the Navigation Stack.

- Path Planning: When a goal pose is received, the Global Planner will generate a global path from the robot's current location to the goal location, considering the map and any constraints. The planner will output a coarse path, represented by a sequence of waypoints.

- Obstacle Avoidance: The Local Planner component will take the global path and refine it for local obstacle avoidance. It will consider the robot's current position, the sensor data, and the dynamic obstacles in real-time to navigate smoothly along the planned path.

- Publish Velocity Commands: Once the local planner generates the desired trajectory, convert it into velocity commands and publish them to the robot's control system. Ensure that the control system properly interprets the velocity commands and drives the robot accordingly.

With these steps, you can implement the Navigation Stack for autonomous navigation in your indoor warehouse environment using the Kinect v1 sensor.

The launch file below describes how we use Navigation Stack in the proposed model.

```
<launch>
    <include file="$(find mybot_nav)/launch/amcl.launch">
 </include>
      <node pkg="move_base" type="move_base" name="move_base" output="screen">
 <rosparam file="$(find mybot_slam)/config/move_base_params.yaml"/>

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

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

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

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

<rosparam file="$(find mybot_slam)/config/dwa_local_planner_params.yaml" command="load" />
</node>
  </launch>
```

Chapter 5: **Recommendations for Future Work**

5.1 **Project Conclusion:**

In this graduation project, we successfully developed an autonomous mobile robot designed to operate in warehouses and other similar environments. The aim of our project was to address the increasing need for efficient and reliable automation solutions in various industries, including logistics, e-commerce, storage, healthcare, and biotechnology.

Throughout the project, we tackled various aspects, including mechanical design, electrical system and control, and software development. Our team worked collaboratively to design and build a robot that could navigate autonomously, map its surroundings, avoid obstacles, and perform tasks such as package delivery. We carefully selected and integrated a range of sensors and controllers, including the Kinect 360 camera, Arduino Mega, and hoverboard motors and controllers, to enable robust perception and precise control.

One of the key motivations behind our project was the ongoing COVID-19 pandemic, which highlighted the importance of contactless and automated solutions. By developing an autonomous mobile robot, we aimed to contribute to the fourth generation of industrial revolution by providing an efficient and safe solution for warehouse operations, package delivery, and more. Furthermore, the flexibility and modularity of our robot design allow it to be repurposed for different applications, making it a versatile tool for various industries.

The V-model methodology, which we adopted throughout the project, ensured a systematic approach to development, from requirements analysis to design, implementation, and testing. This approach helped us maintain project quality and ensure that our robot met the specified requirements.

Looking ahead, there are several avenues for future work. One potential direction is to explore the application of multi-agent techniques and reinforcement learning algorithms to enhance the robot's navigation capabilities in complex 3D environments. By leveraging these advanced techniques, the robot can learn optimal navigation policies and improve its ability to handle dynamic environments and unexpected situations.

Additionally, the integration of machine learning and data analysis models can enable the evaluation of workers' performance and efficiency within the warehouse setting. By analyzing data collected by the robot during its operations, valuable insights can be gained to optimize workflows, improve resource allocation, and enhance overall productivity.

In conclusion, our graduation project successfully developed an autonomous mobile robot that demonstrates the potential of automation in industrial and logistical operations. The combination of mechanical design, electrical system and control, and software development allowed us to create a functional and adaptable robot that can contribute to the ongoing advancements in the industry. We hope that our work inspires further innovation in the field of robotics and automation, ultimately leading to increased efficiency, safety, and productivity in various sectors. Additionally, throughout the project, we encountered and overcame various challenges related to mechanical design, electrical system integration, and software development. The process of selecting suitable materials, considering budget constraints, and ensuring the robot's ability to withstand the demanding warehouse environment required careful consideration and problem-solving. We successfully addressed these challenges by leveraging our team's expertise and conducting thorough research.

In conclusion, our graduation project has successfully developed an autonomous mobile robot designed for warehouse operations. The integration of mechanical, electrical, and software components has resulted in a functional and adaptable robot that can navigate autonomously, map its surroundings, and perform tasks with precision. The project has not only showcased the potential of automation in industrial and logistical settings but also addressed the challenges and considerations involved in developing such a system.

We believe that our project contributes to the advancement of the fourth generation of industry by providing an efficient and versatile solution for warehouses and other related environments. The successful implementation of our project serves as a stepping stone for future research and development in the field of autonomous mobile robotics.

With the completion of this graduation project, we are confident that our work will inspire further innovation and drive advancements in autonomous mobile robotics, ultimately leading to safer, more efficient, and more productive industrial operations.

5.2 Mechanical design future work

In the future, warehouse robots will continue to evolve and improve in terms of their mechanical systems. Here are some potential areas of focus for future work:

1. **Mobility:**
   Warehouse robots will need to continue to improve their mobility to navigate increasingly complex environments. This could involve developing more advanced sensors and algorithms to help robots avoid obstacles and move more efficiently.

2. **Durability:**
   As warehouse robots are used more frequently and in more demanding environments, they will need to be designed to withstand wear and tear. This could involve using more durable materials, such as carbon fiber or metal alloys, and developing better cooling systems to prevent overheating.

3. **Payload capacity:**
   As warehouses continue to grow in size and demand, robots will need to be able to carry heavier loads.

4. **Energy efficiency:**
   To reduce operating costs and increase the lifespan of the robot, future work will focus on developing more energy-efficient mechanical systems.

5. **Flexibility:**
   As warehouses continue to evolve, robots will need to be more flexible in their design to adapt to changing needs. Future work might involve developing modular robot systems that can be reconfigured to perform different tasks or integrating robots with other automated systems in the warehouse.

6. **Strategic placement:**
   Future work may involve the strategic placement of charging and parking stations within the warehouse to minimize the time robots spend traveling between tasks and charging stations.

7. **Parking optimization:**
   In addition to charging optimization, future work may also focus on optimizing the placement of robot parking stations within the warehouse. This could involve developing algorithms that consider the expected workload of the warehouse and the proximity of the parking stations to the areas where robots are most likely to be needed.

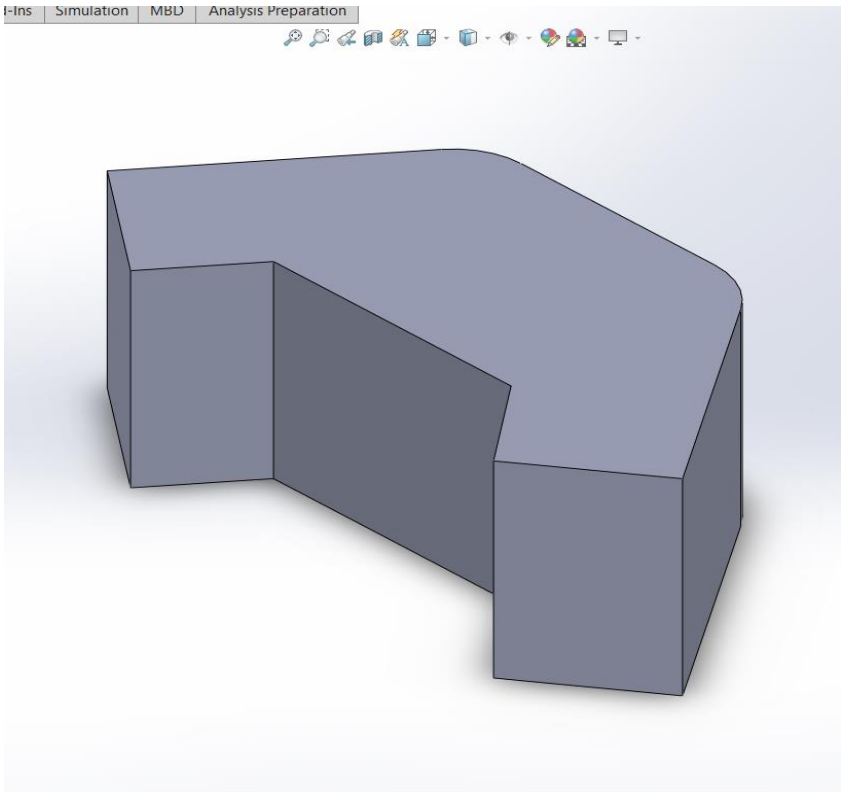

Figure 78 parking and charging station assembly

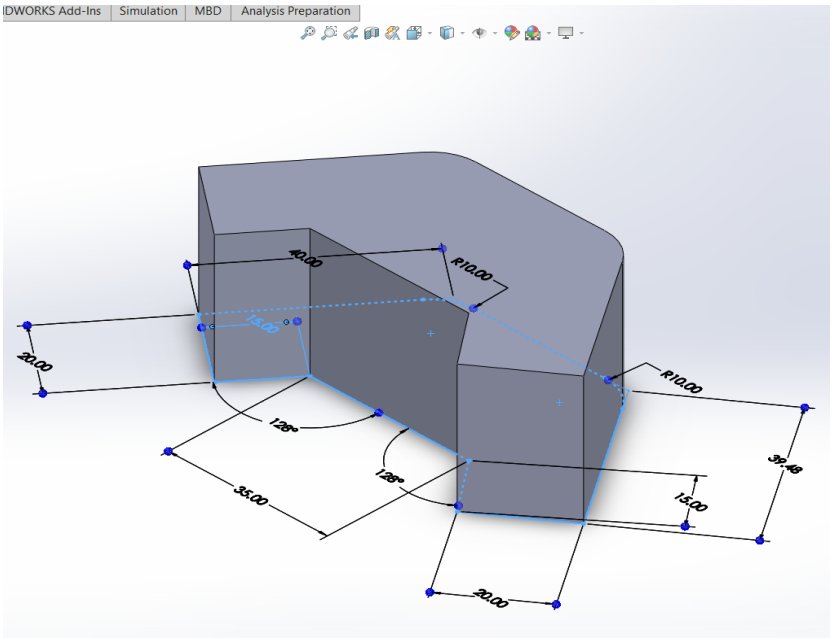

Figure 79 parking and charging station with dimension

5.3 Electric system and control future work

The electric system of the autonomous mobile robot plays a vital role in its overall functionality and performance. In this section, we discuss some key areas for future work that aim to enhance the electric system and its capabilities.

1- Charging System Enhancement:

To ensure a safe and efficient charging process, we propose the enhancement of the charging system. This involves redesigning the charging infrastructure to accommodate faster charging rates, implementing smart charging algorithms to optimize battery life, and incorporating advanced safety features to prevent overcharging and overheating. The improved charging system will result in reduced charging times and increased operational efficiency.

2- Sensor Integration:

As part of future work, we plan to integrate additional sensors into the electrical system. This includes the incorporation of a QR-code sensor, which will enable the robot to detect and interpret QR codes for improved object recognition and localization. Furthermore, the integration of a lidar sensor will provide more accurate and reliable environmental mapping, facilitating precise navigation and obstacle avoidance.

3- Monitoring and Diagnostic System:

In order to ensure effective monitoring and maintenance of the electrical system, we propose the implementation of a comprehensive monitoring and diagnostic system. This system will include temperature sensors to monitor heat dissipation, current sensors to track power consumption, and voltage sensors to ensure stable electrical supply. By continuously monitoring these parameters, any potential issues or anomalies can be detected early on, allowing for timely intervention and preventing critical failures.

4- Data Logging and Analysis:

As part of future work, we envision implementing a data logging and analysis system to capture and analyze various electrical parameters. This system will enable us to collect valuable data on power consumption, battery performance, and charging patterns. By analyzing this data, we can gain insights into the system's efficiency, identify potential optimization opportunities, and make informed decisions regarding system upgrades and improvements.

Through these future work areas, we aim to enhance the electrical system of the autonomous mobile robot, ensuring its reliability, efficiency, and adaptability to changing requirements.

5.4 **Software Future Work**

In the programming aspect of the project, several avenues for future work can be explored to enhance the capabilities and performance of the autonomous mobile robot. These include:

1- Multi-Agent Techniques:

Investigate the potential of implementing multi-agent techniques to enable collaboration and coordination between multiple robots within the same environment. This could involve developing communication protocols and task allocation algorithms to optimize the distribution of tasks among the robot fleet.

2- Reinforcement Learning for 3D Navigation:

Explore the application of reinforcement learning algorithms to improve the robot's navigation capabilities in three-dimensional (3D) environments. By leveraging reinforcement learning, the robot can learn optimal policies for navigating complex terrains, avoiding obstacles, and reaching target locations efficiently.

3- Machine Learning and Data Analysis for Worker Evaluation:

Integrate machine learning techniques and data analysis models to evaluate workers' performance and productivity within the operational setting. By leveraging sensor data and other relevant metrics, machine learning algorithms can provide valuable insights into worker efficiency, identify potential bottlenecks or areas for improvement, and enable data-driven decision-making for process optimization.

These future work areas have the potential to enhance the functionality and performance of the autonomous mobile robot, enabling it to operate more efficiently in dynamic environments and contribute to the optimization of industrial operations.

Bibliography

1- ROS concept and tutorial from: https://wiki.ros.org/ [Accessed 15 June]
2- ROS tutorials from: https://www.youtube.com/watch?v=Qk4vLFhvfbI&list=PLLSegLrePWgIbIrA4iehUQ-impvIXdd9Q [Accessed 15 June]
3- ROS tutorial from: https://www.theconstructsim.com/category/ros-tutorials/ [Accessed 15 June]
4- Author ,year ,title of chapter place of publish:publisher
5- TurtleBot3 Developers ,2017, ROS Robot Programming Korea: ROBOTICS CO ,Ltd
6- Luis Sanchez ,2016, Effective Robotics Programming with ROS United Kingdom: Packet Publishing Ltd
7- Adam Ligocki ,2019, Fusing the RGBD SLAM with Wheel Odometry United Kingdom: Elsevier Ltd
8- Li sun ,2018, Recurrent-Octomap: Learning State-based Map Refinement for Long-Term Semantic Mapping with Kinect v1 United States: Stanford
9- Sagarnil Das ,2018, Simultaneous Localization And Mapping (SLAM) using RTAB-Map Canada: arXiv
10- Nor Afigah Zinuddin ,2015, Autonomous Navigation of Mobile Robot Using Kinect Sensor Malaysia: Tenhnologi
11- Wilbrt G.Aguilar ,2016, 3D Environment Mapping Using the Kinect V1 and Path Planning Based on RRT Algorithms Ecuador: Sangolqui
12- Luigi Villani ,2010, Robotics Modelling ,Planning and Control Italy: Glasgow
13- ROS Navigation From: https://www.youtube.com/watch?v=DBFYZRMLr70&list=PLK0b4e05LnzZWg_7QrIQWyvSPX2WN2ncc [Accessed 15 June]
14- ROS Navigation From: https://www.youtube.com/watch?v=5nZc5iSr5is&list=PLiiw0aSVHcAkF26qR6Q7x6RlLAL6-vuF3 [Accessed 15 June]
15- ROS Navigation Stack From: https://www.youtube.com/watch?v=V20R7EjAz6s&list=PLUnfeehiY56uImWc1iqEyvTOfCGaH3C71 [Accessed 15 June]
16- Octo-Map from: https://www.youtube.com/watch?v=RRP29VnY8go&list=PLrxHDNmQK8nAoDSJRby_ljlwygKO2BJY7 [Accessed 15 June]
17- Octo-Map Implementation from: https://www.youtube.com/watch?v=dF2mlKJqkUg&t=230s&pp=ygULb2N0b21hcHBpbmc%3D [Accessed 15 June]
18- Octo-Map Concept from: https://www.youtube.com/watch?v=Xzu8b6u8ONM&t=206s&pp=ygULb2N0b21hcHBpbmc%3D [Accessed 15 June]
19- Octo-Map vs Gampaaing From: https://www.youtube.com/watch?v=mLMfofIn_PM&pp=ygULb2N0b21hcHBpbmc%3D [Accessed 15 June]
20- RTAB-Map from: https://www.youtube.com/watch?v=gJz-MWn7jhE&t=1716s&pp=ygUMcnRhYiBtYXBwaW5n [Accessed 15 June]
21- RTAB-Map Navigation from: https://www.youtube.com/watch?v=2XkjXmguqk4&t=105s&pp=ygUMcnRhYiBtYXBwaW5n [Accessed 15 June]

22- Navigate With RTAB from: https://www.youtube.com/watch?v=JgP2X-ARj-A&pp=ygUMcnRhYiBtYXBwaW5n [Accessed 15 June]
23- ROS-Master Depth from: https://www.youtube.com/watch?v=YaLUuprRJ_E&t=6s&pp=ygUMcnRhYiBtYXBwaW5n [Accessed 15 June]
24- SolidWorks Tutorials from: https://www.youtube.com/watch?v=hgleruRC7YM&list=PLrOFa8sDv6jfU5Qh83WM5eWLvcU5GjZWG [Accessed 15 June]
25- SolidWorks for Mechanical Engineering from: https://www.youtube.com/watch?v=Ulttc_2p4DY&list=PLrOFa8sDv6jcp8E3ayUFZ4iNI8uuPjXHe [Accessed 15 June]
26- Fusion 360 Tutorial from: https://www.youtube.com/watch?v=ONuJzX-jo88&list=PL0fZjEQc8oaMEkOGcNvueT8lAZvcoKuie [Accessed 16 June]
27- URDF and Gazebo tutorial from: https://www.youtube.com/watch?v=w3bZd7rQG_s&list=PL_t44SOegIa50Po6Ei4mnvsbwEdo8Gx0y [Accessed 16 June]
28- Gazebo Plugins from: https://classic.gazebosim.org/tutorials?tut=ros_gzplugins [Accessed 16 June]
29- Octo-Map Documentation from: https://Octomap.github.io/ [Accessed 16 June]
30- RTAB tutorials from: https://www.youtube.com/watch?v=c5punaP01kU [Accessed 16 June]
31- TurtleBot3 Simulation from: https://automaticaddison.com/how-to-launch-the-turtlebot3-simulation-with-ros/ [Accessed 17 June]
32- Ubuntu Installation on Raspberry pi from: https://www.instructables.com/Install-Ubuntu-18044-LTS-on-Your-Raspberry-Pi-Boar/ [Accessed 18 June]
33- FOC control from: http://grauonline.de/alexwww/motorsim/motorsim.html [Accessed 18 June]
34- FOC Overview from: https://www.ti.com/lit/an/slvaes1a/slvaes1a.pdf?ts=1681894872756&ref_url=https%253A%252F%252Fwww.google.com%252F [Accessed 18 June]
35- Hoverboard Robotics from: https://youtube.com/playlist?list=PLXJDPdNEjPS4lyRpuGIr0aqRVPMaLvePO [Accessed 18 June]
36- SLAM Details from: https://youtu.be/U6vr3iNrwRA [Accessed 18 June]
37- PID control from: https://youtu.be/t7ImNDOQIzM [Accessed 18 June]
38- Introduction to Battery Management System from: https://www.coursera.org/learn/battery-management-systems#syllabus [Accessed 19 June]
39- Self Docking from: https://youtu.be/Zyxcn-MsfCM [Accessed 20 June]
40- ROS control 4 wheels hoverboard robot from: https://www.youtube.com/watch?v=pVyJj-FI5WA [Accessed 20 June]

Appendix

The code for MPU6050

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif
MPU6050 mpu;
#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13       // (Arduino is 13, Teensy is 11, Teensy++ is
   6)
bool blinkState = false;
bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];
Quaternion q;          // [w, x, y, z]
VectorInt16 aa;        // [x, y, z]
VectorInt16 aaReal;    // [x, y, z]
VectorInt16 aaWorld;   // [x, y, z]
VectorFloat gravity;   // [x, y, z]
float euler[3];        // [psi, theta, phi]
float ypr[3];          // [yaw, pitch, roll]
uint8_t teapotPacket[28] = { '$', 0x03, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00, '\r', '\n' };
volatile bool mpuInterrupt = false;  // indicates whether MPU interrupt
   pin has gone high
void dmpDataReady() {
  mpuInterrupt = true;
}
void setup() {
// join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
   Wire.setClock(400000);  // 400kHz I2C clock. Comment this line if
    having compilation difficulties
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif

  // initialize serial communication
  // (115200 chosen because it is required for Teapot Demo output, but
   it's
  // really up to you depending on your project)
  Serial.begin(115200);
  while (!Serial)
    ;  // wait for Leonardo enumeration, others continue immediately
  // initialize device
  Serial.println(F("Initializing I2C devices..."));
  mpu.initialize();
  pinMode(INTERRUPT_PIN, INPUT);

  // verify connection
  Serial.println(F("Testing device connections..."));
```

```
    Serial.println(mpu.testConnection() ? F("MPU6050 connection
   successful") : F("MPU6050 connection failed"));

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

  // supply your own gyro offsets here, scaled for min sensitivity
  mpu.setXAccelOffset(-1169);
  mpu.setYAccelOffset(744);
  mpu.setZAccelOffset(1620);
  mpu.setXGyroOffset(48);
  mpu.setYGyroOffset(47);
  mpu.setZGyroOffset(-8);

  // make sure it worked (returns 0 if so)
  if (devStatus == 0) {
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
     Serial.println(F("Enabling interrupt detection (Arduino external
   interrupt 0)..."));
                  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN),
   dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

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

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

  // configure LED for output
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // if programming failed, don't try to do anything
  if (!dmpReady) return;

  // wait for MPU interrupt or extra packet(s) available
  while (!mpuInterrupt && fifoCount < packetSize) {
  }

  // reset interrupt flag and get INT_STATUS byte
  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();

  // get current FIFO count
  fifoCount = mpu.getFIFOCount();
```

```
  // check for overflow (this should never happen unless our code is
   too inefficient)
  if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    Serial.println(F("FIFO overflow!"));

   // otherwise, check for DMP data ready interrupt (this should happen
   frequently)
  } else if (mpuIntStatus & 0x02) {
    // wait for correct available data length, should be a VERY short
   wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);

    // track FIFO count here in case there is > 1 packet available
     // (this lets us immediately read more without waiting for an
   interrupt)
    fifoCount -= packetSize;

    // display quaternion values in InvenSense Teapot demo format:
    teapotPacket[2] = fifoBuffer[0];
    teapotPacket[3] = fifoBuffer[1];
    teapotPacket[4] = fifoBuffer[4];
    teapotPacket[5] = fifoBuffer[5];
    teapotPacket[6] = fifoBuffer[8];
    teapotPacket[7] = fifoBuffer[9];
    teapotPacket[8] = fifoBuffer[12];
    teapotPacket[9] = fifoBuffer[13];
    // gyro values
    teapotPacket[10] = fifoBuffer[16];
    teapotPacket[11] = fifoBuffer[17];
    teapotPacket[12] = fifoBuffer[20];
    teapotPacket[13] = fifoBuffer[21];
    teapotPacket[14] = fifoBuffer[24];
    teapotPacket[15] = fifoBuffer[25];
    // accelerometer values
    teapotPacket[16] = fifoBuffer[28];
    teapotPacket[17] = fifoBuffer[29];
    teapotPacket[18] = fifoBuffer[32];
    teapotPacket[19] = fifoBuffer[33];
    teapotPacket[20] = fifoBuffer[36];
    teapotPacket[21] = fifoBuffer[37];
    //temperature
    int16_t temperature = mpu.getTemperature();
    teapotPacket[22] = temperature >> 8;
    teapotPacket[23] = temperature & 0xFF;
    Serial.write(teapotPacket, 28);
    teapotPacket[25]++;  // packetCount, loops at 0xFF on purpose

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
  }
}
```

mybot.xacro :

```
<?xml version="1.0" encoding="utf-8"?>
<!--
=================================================================================
======= -->
<!-- |    This document was autogenerated by xacro from
mybot_description.urdf         | -->
<!-- |    EDITING THIS FILE BY HAND IS NOT
RECOMMENDED                                  | -->
<!--
=================================================================================
======= -->
<!-- This URDF was automatically created by SolidWorks to URDF Exporter!
Originally created by Stephen Brawner (brawner@gmail.com)
     Commit Version: 1.6.0-1-g15f4949  Build Version: 1.6.7594.29634
     For more information, please see http://wiki.ros.org/sw_urdf_exporter -
->
<robot name="myrobot" xmlns:xacro="http://www.ros.org/wiki/xacro">
<xacro:include filename="$(find mybot_description)/urdf/mybot.gazebo" />
  <link name="base_link">
    <inertial>
      <origin rpy="0 0 0" xyz="0.128386820110742 0.000686929395924185
0.158512775795314"/>
      <mass value="34.6383267278588"/>
      <inertia ixx="1.73796264850436" ixy="-0.000148699799662171" ixz="-
0.260070022561171" iyy="2.13507888591353" iyz="-1.80220424604975E-06"
izz="2.01858956817808"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://mybot_description/meshes/base_link.STL"/>
      </geometry>
      <material name="">
        <color rgba="0.792156862745098 0.819607843137255 0.933333333333333
1"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://mybot_description/meshes/base_link.STL"/>
      </geometry>
    </collision>
  </link>
  <link name="right_motor">
    <inertial>
```

```
      <origin rpy="0 0 0" xyz="-4.20679424628645E-09 0.0331497387034929
4.87590441966557E-06"/>
      <mass value="1.43144480602327"/>
      <inertia ixx="0.00331965861699229" ixy="-1.65643316648922E-10" ixz="-
2.49861743707994E-10" iyy="0.00595557706241426" iyz="-3.90156064036137E-07"
izz="0.00331969353968102"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://mybot_description/meshes/right_motor.STL"/>
      </geometry>
      <material name="">
        <color rgba="0.792156862745098 0.819607843137255 0.933333333333333
1"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://mybot_description/meshes/right_motor.STL"/>
      </geometry>
    </collision>
  </link>
  <joint name="right_motor_joint" type="continuous">
    <origin rpy="0 0 0" xyz="0.21185 -0.327 -0.242"/>
    <parent link="base_link"/>
    <child link="right_motor"/>
    <axis xyz="0 1 0"/>
  </joint>
  <link name="left_motor">
    <inertial>
      <origin rpy="0 0 0" xyz="-4.20679446833105E-09 0.0331497387034928
4.87590441999863E-06"/>
      <mass value="1.43144480602327"/>
      <inertia ixx="0.00331965861699228" ixy="-1.65643329348428E-10" ixz="-
2.49861736040813E-10" iyy="0.00595557706241427" iyz="-3.90156064035793E-07"
izz="0.00331969353968103"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://mybot_description/meshes/left_motor.STL"/>
      </geometry>
      <material name="">
        <color rgba="0.792156862745098 0.819607843137255 0.933333333333333
1"/>
      </material>
```

```
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://mybot_description/meshes/left_motor.STL"/>
      </geometry>
    </collision>
  </link>
  <joint name="left_motor_joint" type="continuous">
    <origin rpy="0 0 3.1416" xyz="0.21185 0.327 -0.242"/>
    <parent link="base_link"/>
    <child link="left_motor"/>
    <axis xyz="0 -1 0"/>
  </joint>
  <link name="right_wheel">
    <inertial>
      <origin rpy="0 0 0" xyz="4.2067941907753E-09 0.0331497387034929 -
4.87590441988761E-06"/>
      <mass value="1.43144480602327"/>
      <inertia ixx="0.00331965861699229" ixy="1.65643316113872E-10" ixz="-
2.49861736302478E-10" iyy="0.00595557706241427" iyz="3.9015606403935E-07"
izz="0.00331969353968102"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://mybot_description/meshes/right_wheel.STL"/>
      </geometry>
      <material name="">
        <color rgba="0.792156862745098 0.819607843137255 0.933333333333333
1"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://mybot_description/meshes/right_wheel.STL"/>
      </geometry>
    </collision>
  </link>
  <joint name="right_wheel_joint" type="continuous">
    <origin rpy="0 0 0" xyz="-0.063147 -0.327 -0.242"/>
    <parent link="base_link"/>
    <child link="right_wheel"/>
    <axis xyz="0 1 0"/>
  </joint>
  <link name="left_wheel">
    <inertial>
```

```
      <origin rpy="0 0 0" xyz="4.20679427404202E-09 0.0331497387034928 -
4.87590441961006E-06"/>
      <mass value="1.43144480602327"/>
      <inertia ixx="0.0033196586169923" ixy="1.65643308665796E-10" ixz="-
2.49861736506603E-10" iyy="0.00595557706241428" iyz="3.90156064036769E-07"
izz="0.00331969353968103"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://mybot_description/meshes/left_wheel.STL"/>
      </geometry>
      <material name="">
        <color rgba="0.792156862745098 0.819607843137255 0.933333333333333
1"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://mybot_description/meshes/left_wheel.STL"/>
      </geometry>
    </collision>
  </link>
  <joint name="left_wheel_joint" type="continuous">
    <origin rpy="0 0 3.1416" xyz="-0.063147 0.327 -0.242"/>
    <parent link="base_link"/>
    <child link="left_wheel"/>
    <axis xyz="0 -1 0"/>
  </joint>
  <link name="camera_link">
    <inertial>
      <origin rpy="0 0 0" xyz="-0.0277735063558892 0.0143658806531268
5.94217075544634E-09"/>
      <mass value="0.554061895865926"/>
      <inertia ixx="0.00292760923931499" ixy="-9.3576574546003E-07" ixz="-
9.73024378416661E-11" iyy="0.00022646595658487" iyz="-4.83501853093787E-10"
izz="0.00302698572685721"/>
    </inertial>
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://mybot_description/meshes/camera_link.STL"/>
      </geometry>
      <material name="">
        <color rgba="0.494117647058824 0.494117647058824 0.494117647058824
1"/>
      </material>
    </visual>
```

```
    <collision>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh
filename="package://mybot_description/meshes/camera_link.STL"/>
      </geometry>
    </collision>
  </link>
  <joint name="kinect_joint" type="fixed">
    <origin rpy="0 0 0" xyz="0.49448 -0.016314 0.10959"/>
    <parent link="base_link"/>
    <child link="camera_link"/>
    <axis xyz="0 0 0"/>
  </joint>



  <!--  Kinect Optical Link and joints   /-->
<joint name="kinect_optical_joint" type="fixed">

<origin xyz="0 0 0" rpy="-1.5707 0 -1.5707"/> <!-- To invert the coordinates
from image convention (Z is forward) to normal xyz /-->
<parent link="camera_link"/>
<child link="kinect_optical"/>
</joint>

<!--Virtual link to project the images to, and then flip it to us as its
joint is flipped  /-->
<link name="kinect_optical">
</link>
</robot>
```