Intelligent Mobile Robot

Intelligent
Mobile Robot

Graduation Project
2023

HIET

Higher Institute of Engineering and Technology Kafr el-sheikh

Department of Computer and Control Engineering

Supervise By:

Prof.Dr. Wael Elwady

Dr. Ghada Hamisa

Prepared By:

Abdulrahmn Elmorshedy	**Mohammed Mostafa**	**Aliaa Shaban**
Essam Fayed	**Hend Elsayed**	**Ahmed Salman**
Abdelmawla Elamrosy	**Ammar Elgendy**	**Nada Essam**
Hassan Abbas	**Mostafa Saleh**	**Dina Mostafa**
Taher Mohamed	**Alaa Elghoul**	

Abstract

This project introduces an autonomous mobile robot with an arm to assist it we called our Robot Intelligent Mobile Robot (IMR) designed to be used for indoor navigation. It can be operated either autonomously or controlled by a man remotely over the network or wired. Odometry information is used to estimate the robot's position relative to its origin. To achieve robust odometry, the robot uses two sources of odometry, the linear velocities from the encoders and the angular velocity from the IMU. A Kinect v2 mounted on the robot creates a SLAM map with the help of edge detection and computer vision. The robot can navigate autonomously through the predefined map using AMCL. A robot arm is mounted to pick and place objects. The project is based on Robots Operating System (ROS) which makes its functionality reusable in other projects. Modularity and readable codes are considered in the design and implementation of software nodes. About future work, there is a wide field of updates like object detection, On-line Mapping of new Environments and Voice commands, gesture control for a variety of tasks.

The main objective of this project is to demonstrate a robust mobile robot to perform autonomous navigation from one point to another. Using SLAM and AMCL to define the map and navigate through it, and avoid the faced obstacles. Also, the robot is designed around the concept of Robot-as-a-service (RASS), as it can help humans in their daily life. The idea was about implementing this robot in a warehouse (industrial) environment to organize and help define the storage map and navigate through it, then an arm can pick and place objects in their defined or specified place. This robot can be implemented in various ways not just in a warehouse as its application is very wide and needed.

Table of Contents

Table of Figures

Chapter One

1 Introduction and Literature Review

1.1 DEFINITION AND HISTORY OF ROBOTS

1.1.1 What is a robot?

What is a robot?! First, we review the meaning of the word robot, as it is the machine designed or (invented) through an engineering system that makes it work as a substitute for human labor or to carry out a special task for which it was designed, as it is the machine that operates automatically without human involving. Why do we resort to making a robot? The answer to this question clarifies more meanings about mobile robots.

Robots are being built to reduce the cost of human labor, save time, do some work that is dangerous to humans, and do a lot of work. And as we know that everything is considered a (double-edged sword), and as there is a robot that has advantages, it certainly has disadvantages, the most famous of which is unemployment, because some places use robots and dispense with the human factor, and this is not true, because no matter how high the robot reaches a high degree of accuracy and intelligence, it is There will come times and situations that it cannot deal with, and just as there are robots that are useful to humanity, there are robots that exploit the destruction of humanity.

Robots came to perform much work on behalf of human beings or operate in an environment to support human beings, thus robots can perform routine work which is just some tasks that iterate over time. Robots can save a lot of time and effort, as it is known that manual work does have some disadvantages like, lack of complete conformity, or inaccuracy of similarity. Saving money by replacing manpower with robots, which means reducing production costs and increasing profit margins. Robots are tasked with doing very difficult tasks. Availability of flexibility in completing tasks in factories. Reducing the rates of physical injury among workers due to the completion of the work by the robot on their behalf.

some of the disadvantages of robots are that the world is buzzing with the use of robots and that they can replace humans in all fields, but each issue has two angles of consideration, the first is positive and the second is negative, and among the most prominent disadvantages of robots are the following: The robot does not have the power of the human mind that enables the human to adapt to the problem and solve it in an innovative ways and new because the robot is programmed in a specific way, it does not know anything else, and it cannot deal with emerging problems or influences. Humans can adapt according to difficult climatic conditions, such as extreme cold or sweltering heat, while robots need a specific environment and certain conditions, so that they can adapt.

The high construction or foundational cost of the robot, compared to humans, will be adequate to employ it. Robots need to be programmed correctly and implemented in a way that maximizes their efficiency.

1.1.2 Robot History

Robots appeared in ancient times, but their spread began in the (industrial revolution), when humans and structural engineering developed the ability to control electricity, and robots began to develop more in the early twentieth century. It was the first industrial machine in 1954. It was a robot that moved big objects to the intended places. After that, robots have been developing and entering various fields, including medicine

(surgical operations, etc.), agriculture, industry, and many other fields. The history of robots dates back to ancient times when civilizations such as the Greek, Roman, and Chinese empires developed machines that could perform simple tasks. However, the modern history of robots began in the 20th century, with the development of electrical and mechanical engineering.

In the 1950s and 1960s, researchers in the United States and Japan began developing industrial robots that could perform repetitive tasks, such as welding and assembly, in factories,

These early robots were large and expensive, and they were controlled by analog computers. In the 1970s and 1980s, advances in microelectronics and computer technology led to the development of more sophisticated robots that were capable of performing more complex tasks. These robots could be programmed to perform a variety of functions, including assembly, inspection, and material handling. In the 1990s and 2000s, robots began to be used in a wider range of applications, including medical surgery, military operations, and space exploration. Advances in artificial intelligence, machine learning, and computer vision have made it possible for robots to perform more complex tasks and interact more effectively with humans. Today, robots are implemented in a wider range of applications, from manufacturing and logistics to healthcare and entertainment. The field of robotics continues to evolve rapidly, with new advances in technology and new applications emerging all the time.

1.2 ROBOTS TYPES

1.2.1 Legged robots

Legged robots are mobile robots that use mechanical limbs for movement, but their movement methods are more complex than wheeled robots, as they can cross many different terrains and enter rugged places, and legged robots are indispensable for most applications, such as one-legged robots, and two-legged robots legged, three-legged robots, four-legged robots, six-legged robots, and multi-legged robots. Legged robots are robots that use legs as their primary means of locomotion, and they are becoming increasingly important in the field of robotics for several reasons:

1. Versatility: Legged robots are highly versatile and can navigate a wide range of terrains, including rough or uneven terrain, stairs, and obstacles. This makes them ideal for a variety of applications, including search and rescue, exploration, and military operations.
2. Efficiency: Legged robots are more energy-efficient than wheeled or tracked robots in certain environments, such as rocky or sandy terrain, where wheels or tracks may get stuck or struggle to move. Legged robots can also more easily navigate through cluttered environments.
3. Adaptability: Legged robots can adapt to changing environments and situations more easily than other types of robots. They can adjust their gait and movement patterns to accommodate changes in terrain or obstacles, and they can even climb over obstacles that are too tall for wheeled or tracked robots.
4. Human-like movement: Legged robots can move in a way that is more similar to human movement, which makes them more capable of performing tasks that require dexterity and mobility. This includes tasks such as opening doors, manipulating objects, and interacting with humans.

Overall, legged robots have the potential to be highly useful in a variety of applications, and their versatility, efficiency, adaptability, and human-like movement make them an important area of research and development in the field of robotics

1.2.2 Wheeled Robots

Wheeled robots are robots that use wheels as their primary means of locomotion. Their shapes vary depending on the purpose for which they are used and the tasks they need to perform. Some wheeled robots are designed to be autonomous, meaning they can operate independently of human control (artificial intelligence). These robots use sensors and algorithms to navigate, avoid obstacles, and perform the tasks they are designed for. Other wheeled robots are operated by humans, either through remote control or by riding on the robot itself. There are many forms of robots with wheels of different shapes and sizes, from small and simple robots that can fit in the palm of your hand, to large and complex robots that are several meters long. They can be designed for specific missions, such as Mars exploration, or for general uses, such as transporting goods in warehouses and much more. In general, wheeled robots are a versatile and useful type of robot that can be used in a wide range of applications. Wheeled robots are robots that use wheels as their primary means of locomotion, and they are important in the field of robotics for several reasons:

- **Efficiency:** Wheeled robots are typically more energy-efficient than legged robots on smooth, flat surfaces, making them ideal for applications such as warehouse automation, where they can transport goods quickly and efficiently.
- **Stability:** Wheeled robots are generally more stable than legged robots, which makes them better suited for tasks that require carrying or transporting heavy objects.
- **Simplicity:** Wheeled robots are often simpler and easier to design and build than legged robots, which can make them more cost-effective and accessible for a wider range of applications.
- **Ease of control:** Wheeled robots are typically easier to control than legged robots, as their movement is more predictable and stable. This makes them well-suited for applications such as teleportation, where a human operator can control the robot remotely.

Wide range of applications: Wheeled robots are used in a wide range of applications, including manufacturing, logistics, exploration, and service robotics. Their versatility and ease of use make them an important tool for a variety of industries.

Overall, wheeled robots are an important type of robot that offer a range of benefits, including efficiency, stability, simplicity, ease of control, and versatility. While they may not be as adaptable as legged robots in certain environments, they are well-suited for a wide range of applications and will continue to be an important area of research and development in the field of robotics.

1.3 INTRODUCTION TO ROBOT AS SERVICE

Robot as a service (RaaS) is fast becoming an essential element in our daily life, particularly in industries that traditionally depend on labor and are sensitive to minimal labor wages. These industries include the cleaning services, safety and security services, delivery services, logistics and warehouse operations, and manufacturing operations.

In a nutshell, what is robot as service?

A complete robot-as-a-service solution goes beyond the simple leasing of industrial or service robotic hardware. It offers users continuous value while charging users based on what they use or need. The continuous value generation comes from the combination of a service app, a service cloud, and a generally available fleet of robots that can be deployed as needed.

Figure 1.1: Rass

Robots as a service often need to interact with complex people and environments, they need to be more flexible. Therefore, an important requirement for a service robot must have a certain degree of autonomy to get rid of the rigid form of instructions and procedures and to be able to respond to incoming information and adapt to changes in the environment. Like the structure of industrial robots: Service robots also consist of driving devices, motorized devices, control systems, sensors, and communications. In addition to the improvement of robot movement and simulation architecture, the development path of service robots is one of the most eye-catching technologies.

Because the crowds and environments that service robots encounter are so complex, they are equipped with different sensations that mimic the five senses of humans. In addition to vision, hearing, and touch, he also focused on systems such as facial/body gesture recognition and voice recognition to enhance communication capabilities. In addition, the combination of artificial intelligence and cloud network systems has gradually become the trend of future development. Intelligent service robots can respond to environmental conditions

instantly, become more autonomous in operation, and can be easily integrated into society and used by humans.

A service robot is defined as "improving the quality and comfort of human life, by combining a movable base and a robotic arm, and is equipped with a control device to provide semi-automatic or fully automatic services. Its forms and applications are very wide, so, if it is used in non-manufacturing in the industry, Industrial robots can sometimes be defined as service robots. Compared with industrial robots, the application of service robots The scope is wider and the technology is more diversified, including the sectors of manufacturing, agriculture, mining, construction, transportation, warehousing, accommodation, catering, public administration, and national defense. Due to the high technical threshold of multi-tasking robots that can seamlessly adapt to society and deal with various living conditions, the robots that currently account for most of the market production value are still mainly robots that serve specific needs.

Robots as a service is a growing trend in the field of robotics, and it offers several important benefits:

- **Cost savings:** RaaS allows companies to avoid the upfront costs of purchasing and maintaining their robotic systems. Instead, they can pay for the use of a robot on a subscription or pay-as-you-go basis, which can be more cost-effective in the long run.
- **Scalability:** RaaS allows companies to scale their robotic systems up or down as needed, depending on changes in demand or production. This flexibility can help companies avoid the costs and risks of investing in their robotic systems that may become obsolete or underutilized over time.
- **Access to the latest technology:** RaaS providers are responsible for maintaining and updating their robotic systems, which means that companies can have access to the latest technology without the need for constant investment in hardware and software upgrades.
- **Expert support:** RaaS providers typically offer expert support services, including training, maintenance, and troubleshooting, which can help companies get the most out of their robotic systems and avoid downtime.
- **Improved efficiency and productivity:** RaaS can help companies improve their efficiency and productivity by automating repetitive or dangerous tasks, reducing the risk of errors, and allowing human workers to focus on more complex or creative tasks.

1.4 INTRODUCTION TO ROS

A robot operating system (ROS) is an open-source robotics software framework that provides tools and libraries for building and running complex robotic systems. ROS was initially developed by Willow Garage, a robotics research laboratory in California, and now is managed by Open Robotics.

ROS provides a modular and distributed architecture that allows developers to build complex robotic systems by incorporating a wide variety of sensors, actuators, and algorithms.

It supports various programming languages, including C++, Python, and Java, which makes it accessible to a wide community of developers. There are many benefits of using ROS (Robot Operating System) in robotics development, including:

- **Open-source:** ROS is an open-source framework, which means that it is free to use and can be modified and customized to meet the needs of individual developers. This makes it accessible to a wide range of developers, from hobbyists to large companies.
- **Modular and distributed architecture:** ROS provides a modular and distributed architecture that makes it easy to build complex robotic systems by integrating multiple sensors, actuators, and algorithms. This modular approach allows developers to reuse code and build on existing work, which can save time and reduce development costs.
- **Large and active community:** ROS has a large and active community of developers who contribute to its development and share their work with others. This community provides a wealth of resources, including tutorials, documentation, and code libraries, which can help developers get started with ROS quickly and easily.
- **Support for multiple programming languages:** ROS supports multiple programming languages, including C++, Python, and Java, which makes it accessible to a broad community of developers with different backgrounds and skill levels.
- **Message-passing system:** ROS provides a message-passing system that allows different components of a robotic system to communicate with each other using a common message format. This makes it easy to integrate multiple sensors and actuators into a single system and can help reduce development time and costs.
- **Simulation and visualization tools:** ROS provides simulation and visualization tools that allow developers to test and debug their robotic systems in a virtual environment before deploying them in the real world. This can help reduce the risk of errors and can save time and money in the long run.

ROS provides a variety of tools for building and testing robotic systems, including a package manager for managing and sharing code, simulation tools for testing and debugging, and imaging tools for monitoring and analyzing robot behavior, ROS has been widely adopted in the robotics community and is used in a variety of applications, including autonomous vehicles, industrial automation, and service robots.

It has a large and active community of developers who contribute to its development and share their work with others.

Overall, ROS provides a robust and flexible framework for building complex robotic systems and has become an essential tool for robotics researchers and developers.

What is Ros's prime purpose?

The primary goal of ROS is to support code reuse in robotics research and development. ROS is a distributed process framework (also known as a contract) that enables executable scripts to be individually designed and loosely coupled at runtime.

These processes can be grouped into packages and stacks, which can be easily shared and distributed. ROS also supports a unified system of code repositories that enable distributed collaboration as well.

This design allows the developers to make independent decisions about development and implementation, but it can all be brought together with the ROS infrastructure tools.

Is Ros an operating system?

Robot Operating System (ROS) is not an operating system in the traditional sense. Instead, it is a middleware framework that runs on top of an operating system. Ros is developed to be platform-independent thus, it can run on many operating systems, including Linux, macOS, and Windows.

However, Ros is commonly used on Linux-based operating systems, particularly Ubuntu and Debian, this is because many of the ROS packages and tools are developed and tested on these platforms, and also because Linux is an open-source operating system that is widely used in the robotics community.

These include a message-passing system for communication between different components of a robot system, a package manager for installing and managing software packages, and a set of visualization and simulation tools for testing and debugging robotic systems.

Ros is designed to be flexible and adaptable, and it can run on a variety of operating systems depending on the needs of the developer.

Chapter Two

2 IMR’s Requirements

2.1 INTRODUCTION

Building a mobile robot needs certain specifications and studying its constraints carefully, so in this section, we will discuss the main requirements that a mobile robot should meet to build a working and interactive mobile robot successfully.

IMR’s requirements consist of three major parts:

Design requirements, hardware requirements, and software requirements each of these requirements will be discussed in detail, as this step is one of the most important procedures that any project should fulfill its objectives.

2.2 DESIGN REQUIREMENTS

The first thing when building a robot is considering the design or choosing a robot model that can be suitable for your implementation later, so determining a robot chassis and shape is significant, hence it can later affect the motion of the robot as we will see later on in this book.

2.2.1 Mobile Robot

Considering a robot chassis and shape is a difficult task because today there is a variety and many combinations of mechanical designs, so we will discuss the most commonly used ones through the robotics community.

A mobile robot body can be built with different Materials like metal, plastic, acrylic, and wood as example the turtlebot2 consists of a metal and plastic body, as shown in figure1.1

Turtlebot2 is a simple design and can fulfill its objectives, so after considering several designs, we choose to model a simple one.

We chose a simple wood design because its weight is lighter than metal and its price is lower compared to acrylic.

Fig 2.1: Turtlebot2

2.2.2 Robot Arm

When considering any arm design, there are some important points to consider, Where the arm can be designed from metal or plastic, according to the required standards, where the metal design can withstand high temperatures and is suitable for the difficult industrial environment, but its weight is heavy and requires weight than metal and is environmentally friendly and does not corrode due to Weather. Motors with strong torque to move it, while the plastic material is characterized by a lighter

We chose to design the arm using 3D printing for its ease of formation, lighter weight, and lower cost, taking into account the basic requirements, as it is corrosion-resistant, environmentally friendly, and can be safely used in the chemical environment.

Fig 2.2: 3D model of our arm

2.3 HARDWARE REQUIREMENTS

2.3.1 Processing units

We need appropriate controllers in terms of memory capacity and processing capacity to execute programming instructions without delay.

Required processing units:

Raspberry pi 4 (4GB)

This unit implements the complex instructions in the ROS system, where it implements the kinematics equations and then sends the final outputs to the Arduino.

This includes a power cable, mini-to-full-size HDMI cable, case, fan, heat sinks, a class 10 micro card at least of 32GB, and of course, the Raspberry Pi itself, 4GB RAM option seems to be pretty good. Note that using less than 4GB RAM on the RPi 4 can result in it freezing during code compilation, like when running the installation script for any of our Raspberry Pi compatible robots.

Fig 2.3: Raspberry 4 pi

Arduino mega 2560 R3

The Arduino unit represents a communication protocol between the ROS system and the rest of the hardware, as it receives code from the computer or the Raspberry Pi and executes it on the motors, it receives the reading of the sensors and sends them to the computer

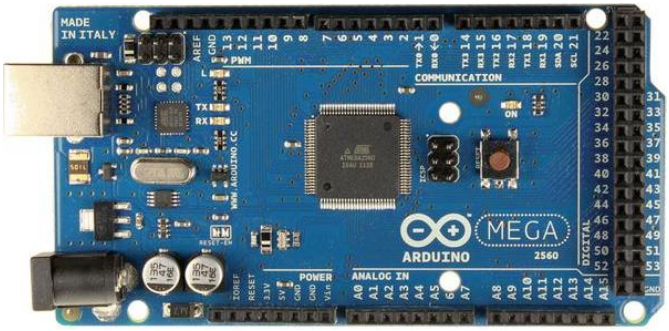


Fig 2.4: Arduino Mega

2.3.2 Sensors

2.3.2.1 Kinect V2

Kinect is a line of motion sensing input devices. Generally contain RGB camera, and infrared projectors and detectors that map depth through either structured light or time of flight calculations, which can in turn be used to perform real-time gesture recognition and body skeletal detection, among other capabilities. They also contain microphones that can be used for speech recognition and voice control.

Fig 2.5: Kinect V2

The general idea of a Time of Light is that the IR emitter turns on and off periodically, and the output is from the depth sensor, being sent to two different ports when the light is on and off, respectively. When the light is on, the output is A, and the output when the light is off is B. A contains the light from the emitted infrared light and ambient light (that is, the light already in the field of view), and B contains only the ambient light. Hence, subtracting B from A (that is, A-B) gives only the modulated light reflected from the infrared emitter, which can be used to accurately calculate depth. Moreover, the size from (A-B) gives a high quality IR image without ambient light. This design makes Kinect v2 produce much better depth images and images than infrared, as shown in Figure 4. Another major change is the design of the Kinect runtime. For Kinect v1, the

Kinect sensor is used exclusively for a single app, and multiple Kinect sensors can be used and controlled by a single app. The app has full control of the Kinect sensors you connect to, including various settings for color accuracy.

		PrimeSense Carmine 1.08	Kinect v2
Infrared/depth camera	**Resolution**	320 × 240 px	512 × 424 px
	Field of view (h×v)	57.5° × 45.0°	70.6° × 60.0°
	Angular resolution	0.18°/px	0.14°/px
	Operating range	0.8–3.5 m	0.5–4.5 m
Color camera	**Resolution**	640 × 480 px	1920 × 1080 px
	Field of view (h×v)	57.5° × 45.0°	84.1° × 53.8°
Frame rate		30 Hz	30 Hz
Minimum latency			20 ms
Shutter type			Global shutter
Dimensions (w×d×h) [mm]		180 × 35 × 25	249 × 66 × 67
Mass (without cables)		160 g	970 g
Connection type		USB 2.0	USB 3.0
Voltage		5 V DC	12 V DC
Power usage		2.25 W	∼15 W

Fig 2.6: Kinect Specifications

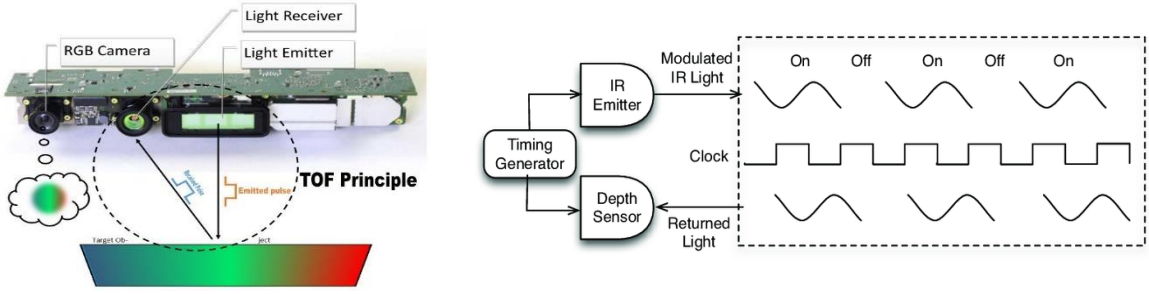


Fig 2.7: Kinect Inside sensor

2.3.2.2 IMU

MPU6050 sensor module is complete 6-axis Motion Tracking Device. It combines 3-axis Gyroscope, 3-axis Accelerometer and Digital Motion Processor all in small package. Also, it has additional feature of on-chip Temperature sensor. It has I2C bus interface to communicate with the microcontrollers.

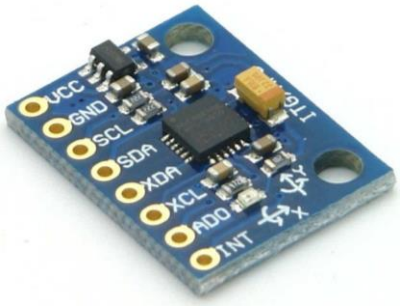


Fig 2.8: IMU Sensor

2.3.2.3 DC Motor Encoders

25GA371 DC Gear Motor with Encoder 170RPM 12Vdc

This type of motor enables us to control its position and speed as well as distance and angular velocity accurately through the values of the pulses sent by the encoder.

Fig 2.9: Dc Motor

Servo Motor

Metal Servo MG996R Continuous Rotation (11k.g)

Weight: 55g

Dimension: 40.7×19.7×42.9mm

Stall torque: 9.4kg/cm (4.8v); 11kg/cm (6.0v)

Operating speed: 0.19sec/60degree (4.8v); 0.15sec/60degree (6.0v)

Operating voltage: 4.8 – 6V

Gear Type: Metal gear

Temperature range: 0- 55deg

- Micro Servo MG90S (Half Metal) 180 Degree

-Micro Servo SG90 0:180 Degree

Fig 2.10: Servo Motor

2.3.3 Power source

We need two batteries, one battery is 10000 mAh and is the power feeder to DC motors and Kinect camera. It can support up to 10A making it lasts for our operations under full load condition.

2.3.4 Components list

#	Components	Quantity
1.	Raspberry pi 4 (4GB)	1
2.	Arduino mega 2560 R3	1
3.	Kinect camera v2	1
4.	Mpu6050	1
5.	25GA371 DC Gear Motor With Encoder	4
6.	robot wheels 6.5 cm	4
7.	Servo motor mg996r metal gear	5
8.	Servo motor mg90s	1
9.	Design 3D printing Arm	1
10.	wooden structure	1
11.	Battery 10000 mAh	2
12.	Step Down Module LM2596	2
13.	Pcb board	1

2.4 SOFTWARE REQUIREMENTS

2.4.1 Ubuntu 18.04

Ubuntu Budgie 18.04 LTS and Ubuntu 18.04 LTS are the latest versions of Ubuntu Linux distribution featuring different desktop environments, keeping the software base the same for both of these flavors.

Budgie desktop environment is lightweight. It takes less RAM than GNOME 3 desktop environment. In my case, Budgie desktop environment of Ubuntu Budgie 18.04 LTS uses about 800 MB of RAM, when GNOME 3 desktop environment of Ubuntu GNOME uses about 1300 MB of RAM. You may take a look at Disk and RAM Usage of Ubuntu, Kubuntu, Lubuntu, Xubuntu, Ubuntu MATE, Ubuntu Budgie 18.04 LTS and see for yourself how much RAM and disk space each of the Ubuntu 18.04 flavors usages.

Budgie desktop environment is very responsive. GNOME 3 desktop environment is not as responsive as the Budgie desktop environment.

Budgie desktop environment is modern and works on both old and new hardware without any lag. GNOME 3 desktop environment does not work well on old hardware.

Budgie desktop environment is ready to use when Ubuntu Budgie 18.04 LTS is installed. No additional extensions are required to make it useable like GNOME 3 desktop environment of Ubuntu 18.04 LTS. That of course saves a lot of time.

2.4.2 Robotic operating system (ROS)

ROS (Robot Operating System) is an open source software development kit for robotics applications. ROS offers a standard software platform to developers across industries that will carry them from research and prototyping all the way through to deployment and production.

ROS is relied upon throughout the robotics industry. It’s the norm for teaching robotics. It’s the basis for most robotics research, from single-student projects to multi-institution collaborations and large-scale competitions. And it’s inside robots that are running in production all around the world today, where ROS provides the tools, libraries, and capabilities that we need to develop our robotics applications, allowing us to spend more time on the work that is important for our business. Because it is open-source, we have the flexibility to decide where and how to use ROS, as well as the freedom to customize it for our needs. Moreover, ROS isn’t exclusive, we don’t need to choose between ROS or some other software stack; ROS easily integrates with our existing software to bring its tools to our problem.

2.5 V MODEL DESIGN PROCEDURE

Before starting work on the robot, we must first had to put an approach to work on, where will help save time and decreases errors in the whole designing system The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively

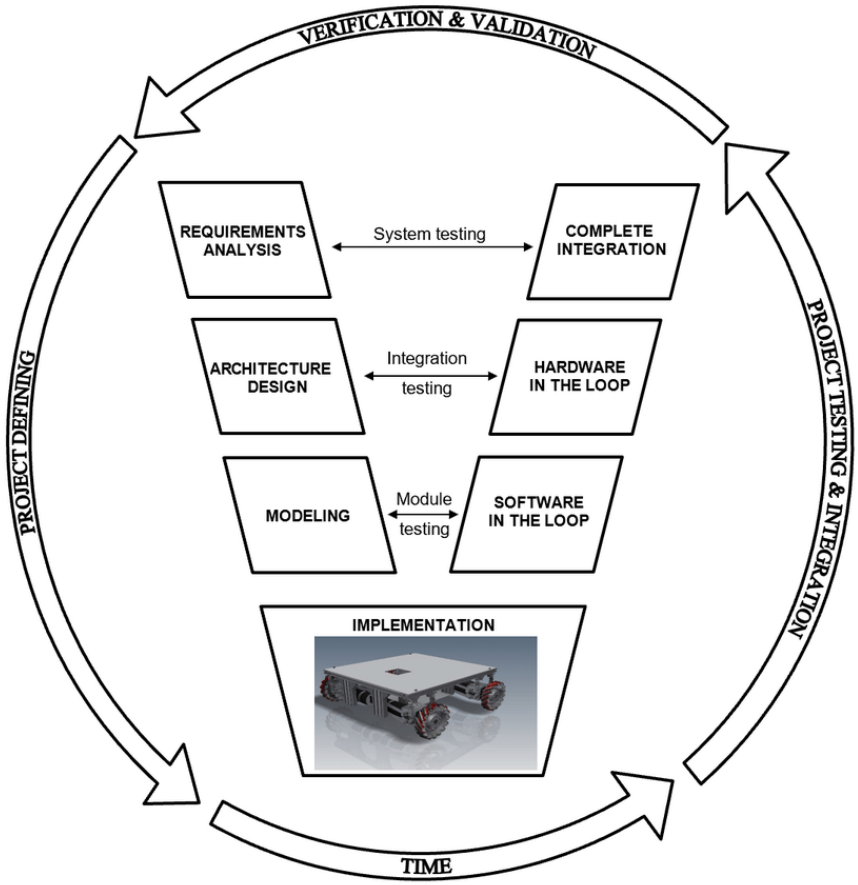
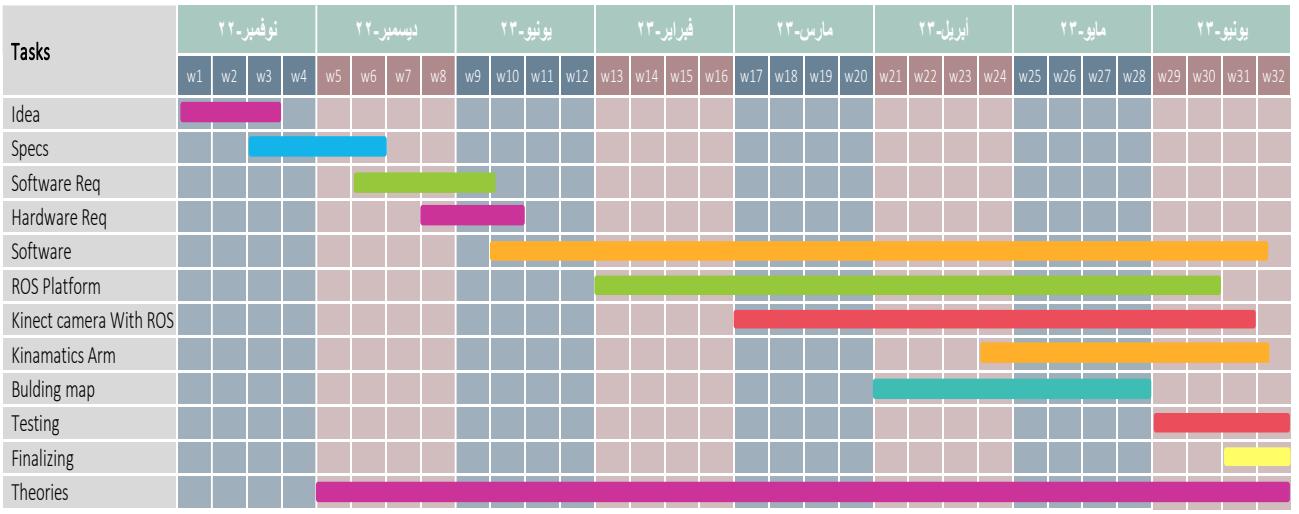


Fig 2.11: V Mode Diagram

2.6 GANTT CHART



Chapter Three

3 IMR’s Theories and Dynamics

3.1 INTRODUCTION

Through this chapter we will discuss the mathematical model of our intelligent mobile robot, this model is consisting of kinematics, dynamics, degree of freedom, and many other things.

The mathematical modeling of a robot's kinematics is motivated by the complexity of robotic systems, which possess highly nonlinear characteristics.

In general, to have a good and accurate modeling of your robot means that we can understand and predict the robot's behavior and motion, and also solve any associated problems we may face in the future.

Modeling our robot is a very complex task, but as it is complicated it is also a very significant step because it enables us to control the whole system and get the desired results from it and also diagnose and solve any failures or errors that also may occur.

In this chapter also we will discuss many other theories that are crucial to our IMR like, path planning, motion planning, navigation, and localization with its algorithms.

Also, we will discuss the different approaches of how we control the robot's parts of our IMR and how we achieve the autonomy needed.

3.2 KINEMATICS OF IMR

Our robot consists of two main parts the mobile robot body and the robotic arm, so our kinematics will be divided into two parts the part of the mobile body that teleports with wheels and the part that have the arm.

Kinematics is the most fundamental study of the behavior of mechanical systems. In the field of mobile robotics, we need to understand the robot's mechanical behavior to design a mobile robot suitable for the task and how to create control software for the mobile robot hardware instance.

Of course, mobile robots are not the first complex mechanical systems to require such analysis. Robotic manipulators have been intensively studied for over three decades. In recent years, the robotics community has achieved a fairly complete understanding of kinematics and even the dynamics relating to the force and mass of robot manipulators.

A mobile robot’s workspace is substantial because it defines the range of possible poses that a mobile robot can achieve in its environment. The robot arm’s controllability defines how motors can be used to move from pose to pose in the workspace.

Also, a mobile robot’s controllability defines possible paths and trajectories in its workspace. Robot dynamics places additional constraints on workspace and trajectory due to mass and torque considerations. The mobile robot is also defined by its dynamics.

The process of understanding the robot's motion begins with the process of describing the contribution each wheel provides for motion. Each wheel has a role in enabling the whole robot to move, as wheels also impose constraints on the robot's motion, for example refusing to skid laterally.

In the following section, we introduce notation that allows the expression of our robot motion in a global reference frame as well as the robot's local reference frame.

Then, using this notation, we demonstrate the construction of simple forward kinematic models of motion, describing how the robot as a whole move as a function of its geometry and individual wheel behavior.

Next, we formally describe the kinematic constraints of individual wheels and then combine these kinematic constraints to express the whole robot's kinematic constraints. With these tools, one can evaluate the paths and trajectories that define the robot's maneuverability.

3.2.1 Mobile robot kinematics

3.2.1.1 Pose representation and transformation

A complete mobile robot mathematical model is defined by three sets of differential equations that govern its dynamics. These describe the forces and moments acting on it and its orientation towards a reference frame.

A 2D-simplified non-linear model is presented along with its linearization and state-space representation. As usual, the model has been decoupled into an independent kinematic and dynamic formulation.

To obtain the kinematic one, the non-homonymic constraint of the vehicle will be considered and derived using the chassis' geometrical properties and a transformation between the coordinate systems.

The dynamical properties, on the other hand, will be derived using the chassis' geometrical properties and a transformation between the coordinate systems.

The dynamical properties, will also be derived using the Euler-Lagrange equations. Some mathematical tools that are necessary to fully understand the equations proposed are introduced first. In essence, these include the reference frames and the Euler angles used.

Reference frame

A reference frame consists of an abstract coordinate system and a set of physical reference points that uniquely represent the position and orientation of a dynamic system; in this case, the mobile robot. Due to the great variety of existing reference frames, an introduction to those used in this work is necessary for the sake of clarity.

Local frame

With Local Frame, we refer to those generic body axes that have origin in the mobile robot Centre of Gravity (COG). They are defined as follows: $X_1$and Z_1 lie in the robot plane of symmetry, with X_1generally parallel to the robot body reference line and directed towards the robot nose, Z_1 is directed from the lower to the upper surface of the shell; the Y_1 axis is selected so that the coordinate frame is right-handed. See fig.2

The local frame is fixed with respect to the mobile robot; therefore, it is an inertial reference frame: the moments of inertia of the rover calculated within this frame do not change during the motion.

Global Frame

With Global Frame, we refer to those generic Cartesian axes fixed in a determined origin. They are defined as follows: $X_g$and Y_g lie on a plane parallel to the ground, and the $Z_g$axis is selected so that the coordinate frame is right-handed. See fig.2.1

The global frame is fixed with respect to the external environment; therefore, it is a non-inertial reference frame: the moments of inertia of the rover calculated within this frame change during the motion.

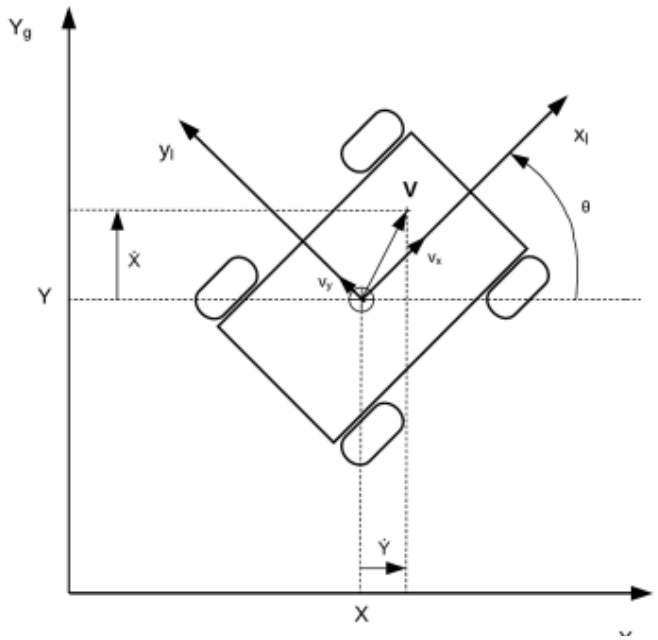


Fig 3.1: Global and local Reference frame

Euler Angles

The Euler angles are a mathematical tool used to define the orientation of a reference frame with respect to another. Indicated as Φ, Θ, Ψ, they represent three independent angular rotations necessary to align two reference frames. For example, considering the two coordinate systems $g_1(X_1, Y_1, Z_1)$ and $g_1(X_1, Y_1, Z_1)$ represented in Fig.2.2, the rotations[1] Φ, Θ, Ψ, aligns $g_2$to g_1.

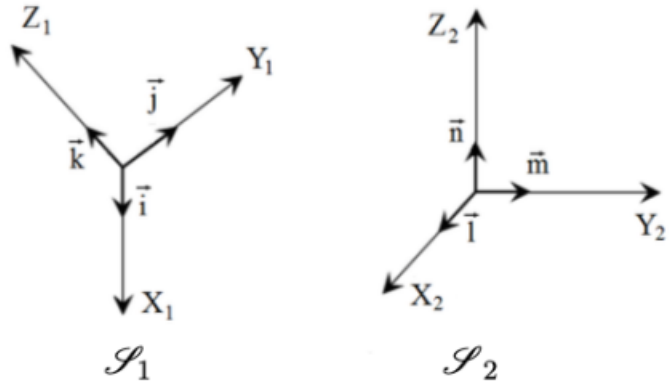


Fig 3.2: Generic reference frames.

According to this description, Euler angles can also be used to define the transformation of the components of a generic vector between two reference frames through the elementary rotation matrix. Assuming that $[Fx1\ , Fy1\ , Fz1]^T$ is a generic vector of the coordinate system g_1, the relationship with the corresponding $[Fx2\ , Fy2\ , Fz2]^T$ vector in the final coordinate system g_2 is:

$$\begin{bmatrix} fx_2 \\ fy_2 \\ fz_2 \end{bmatrix} = [R_{21}] \begin{bmatrix} fx_1 \\ fy_1 \\ fz_1 \end{bmatrix}$$

Where $R21 = [\Phi]\,[\theta]\,[\Psi]$ represent the matrix of complete transformation and $[\Phi], [\theta], [\Psi]$ are the elementary rotation matrices defined as:

$$[\Phi] = \begin{bmatrix} \cos\Phi & -\sin\Phi & 0 \\ \sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix}, [\Theta] = \begin{bmatrix} \cos\Theta & 0 & \sin\Theta \\ 0 & 1 & 0 \\ -\sin\Theta & 0 & \cos\Theta \end{bmatrix}, [\Psi] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Psi & -\sin\Psi \\ 0 & \sin\Psi & \cos\Psi \end{bmatrix}$$

Since the elementary matrices are orthogonal, also $R_{21}$is orthogonal, so $R_{21}^{-1} = R_{21} = R_{21}^T$. This allows to define the inverse transformation as:

$$\begin{bmatrix} fx_1 \\ fy_1 \\ fz_1 \end{bmatrix} = \begin{bmatrix} R_{21}^T \end{bmatrix} \begin{bmatrix} fx_2 \\ fy_2 \\ fz_2 \end{bmatrix}$$

Global-Local Transformation

Euler angle used in the present work representing the rotation necessary to transform the components of a generic vector between the reference frames described in previous section or to align the two reference frames. In order to project the local frame $g_1\ (x, y, z)$ to the global frame $R_{21}\ (X, Y, Z)$, reported in Fig. 2.1, it is necessary one rotation of f magnitude ϑ about $z_1$so the corresponding Euler angle is:

$$\Phi = \vartheta$$

And the local-global rotation matrix R_{lg} is:

$$R_{lg} = \begin{bmatrix} \cos\vartheta & -\sin\vartheta & 0 \\ \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.2.1.2 4-wheel skid-steering model (SSMR)

Skid-steering motion is widely used for wheeled and tracked mobile robots. Steering in this way is based on controlling the relative velocities of the left and right side drives. The robot turning requires slippage of the wheels for wheeled vehicles. Due to their identical steering mechanisms, wheeled and tracked skid-steering vehicles share many properties. Like differential steering, skid steering leads to high maneuverability, and has a simple and robust mechanical structure, leaving more room in the vehicle for the mission equipment. In addition, it has good mobility on a variety of terrains, which makes it suitable for all-terrain missions. However, this locomotion scheme makes it difficult to develop kinematic and dynamic models that can accurately describe the motion. It is very difficult for skid-steering kinematics to predict the exact motion of the vehicle only from its control inputs. As a result, the kinematics models with pure rolling and no-slip assumptions for non-holonomic wheeled vehicles cannot apply in this case. Furthermore, other disadvantages are that the motion tends to be energy inefficient, and difficult to control, and for wheeled vehicles, the tires tend to wear out faster.

Four skid-steered mobile robots, since the lack of a steering system, differential velocities between the two sized are applied to achieve steering requirements which will easily lead to lateral skidding of the wheels for the kinematic control.

On the other side, the inevitable uncertainties in the mobile robot's dynamics will also take important effects on the performance of mobile robots, which becomes another challenging issue for the control design. Recently, a great deal of control methods has been proposed to handle the model uncertainties, such as adaptive control, neural network, and PID control which we used in our IMR.

Another problem should be noted that the skid-steered mobile is four-wheel independently driven, which means the robot is an over-actuated system. In the present work, the wheels on each side are often supposed to rotate at the same speed.

To guarantee this condition, two methods are usually used:

a) The wheels on each side are mechanically connected, and the controller design is not much different from the two-wheel mobile robot:

b) The wheels are regulated with a velocity controller, where the controller design is similar to the kinematic control.

However, it can't be ignored that there are fundamental differences between the four-wheel independently driven mobile robot and the above Thus, to achieve a better performance, the driving torque allocation becomes a possible solution for this problem.

Some control allocation techniques for over-actuated ground vehicles, where the diving torques are directly distributed based on energy consumption.

However, since the skid-steered mobile robot in this book isn't equipped with a steering system and suspension system, the control design for it is much different from the ground vehicle, and the chattering phenomenon is found in practice with the traditional controller. Thus, for the skid-steered mobile robot, the appropriate control strategies are needed to take into account the wheel/ground interactions and the four-wheel driven characteristics.

3.2.1.3 Skid-Steering Mobile Robot Kinematic Model

There are some considerations that we should take in mind before creating the kinematic model, so we should consider the following model assumptions. See fig.3.2:

- (1) The mass center of the robot is located at the geometric center of the body frame;
- (2) The two wheels on each side rotate at the same speed;
- (3) The robot is running on a firm ground surface, and four wheels are always in contact with the ground surface.

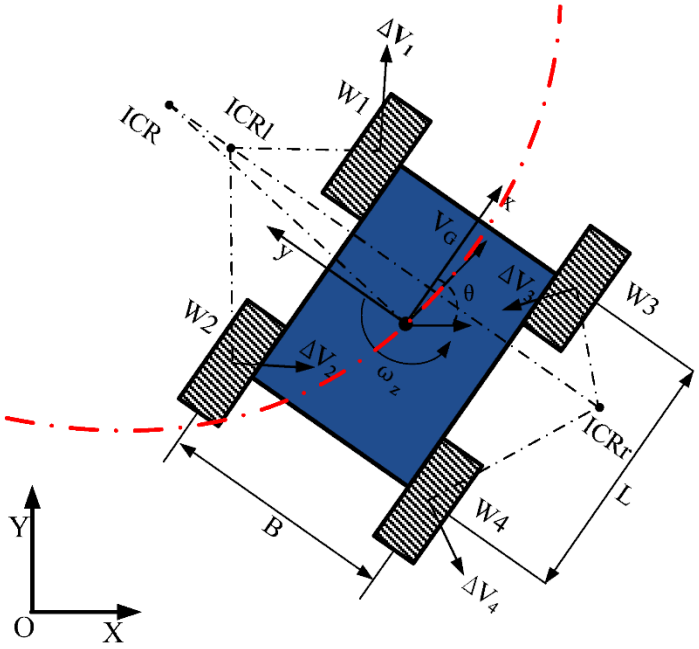


Fig 3.3: The kinematics schematic of skid-steering mobile robot

As we defined previously the inertial frame (X, Y) (global frame) and a local (robot body) frame (x, y), as shown in Figure 2.3. Suppose that the robot moves on a plane with a linear velocity expressed in the local frame as $v = (v_x, v_y, 0)^t$ and rotates with an angular velocity vector $\omega = (0, 0, \omega_z)^t$. If $q = (x, y, \theta)^t$ is the state vector describing generalized coordinate of the robot (i.e., the COM position, X and Y, and the orientation T of the local coordinate frame with respect to the inertial frame), then $\dot{q} = (\dot{x}, \dot{y}, \dot{\theta})^t$ denotes the vector of generalized velocities. It is straightforward to calculate the relationship of the robot velocities in both frames as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \boldsymbol{R_{lg}} = \begin{bmatrix} \cos\vartheta & -\sin\vartheta & 0 \\ \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{v_x} \\ \boldsymbol{v_y} \\ \boldsymbol{\omega_z} \end{bmatrix} \quad (1)$$

Let ω_i , $i = 1,2,3,4$ denote the wheel angular velocities for front-left, rear-left, front-right and rear-right wheels, respectively. From assumption (2), we have:

$$\boldsymbol{\omega_L = \omega_1 = \omega_2}\,, \qquad \boldsymbol{\omega_R = \omega_3 = \omega_4} \quad (2)$$

Then the direct kinematics on the plane can be stated as follows:

$$\begin{bmatrix} \boldsymbol{v_x} \\ \boldsymbol{v_y} \\ \boldsymbol{\omega_z} \end{bmatrix} = \boldsymbol{f} \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix} \quad (3)$$

Where $v = (v_x, v_y)$ is the vehicle's translational velocity with respect to its local frame, and ω_z is its angular velocity, r is the radius of the wheel. When the mobile robot moves, we denote instantaneous centers of rotation (ICR) of the left-side tread, right-side tread, and the robot body as ICR_l, ICR_r, ICR_G respectively. It is known that ICR_l ICR and ICR_G lie on a line parallel to the *x*-axis. We define the *x*-*y* coordinates for ICR_l, ICR_r and ICR_Gas(x_1, y_2),(x_r, y_r) and (x_G, y_G) respectively.

Note that treads have the same angular velocity ω_z as the robot body. We can get the geometrical relation:

$$y_G = \frac{v_x}{\omega_z} \tag{4}$$

$$y_l = \frac{v_x - \omega_l r}{\omega_z} \tag{5}$$

$$y_r = \frac{v_x - \omega_r r}{\omega_z} \tag{6}$$

$$x_G = x_l = x_r = -\frac{v_y}{\omega_z} \tag{7}$$

From Equations (4)-(7), the kinematics relation (3) can be represented as:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = j_\omega \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix} \tag{8}$$

Where the elements of matrix j_ω depend on the tread ICR coordinates:

$$j_\omega = \frac{1}{y_l - y_r} \begin{bmatrix} -y_r & y_l \\ x_G & -x_G \\ -1 & 1 \end{bmatrix} \tag{9}$$

If the mobile robot is symmetrical, we can get a symmetrical kinematics model (i.e., the ICRs lie symmetrically on the x-axis and $x_G = 0$, so matrix j_ω can be written as the following form:

$$j_\omega = \frac{1}{2y_0} \begin{bmatrix} y_0 & y_0 \\ 0 & 0 \\ -1 & 1 \end{bmatrix} \tag{10}$$

Where $y_0 = y_l = -y_r$ is the instantaneous tread ICR value. Noted that for $v_l = \omega_l r$, $v_r = \omega_r r$ the symmetrical model, the following equations can be obtained:

$$\begin{cases} v_x = \frac{\omega_l r + \omega_r r}{2} = \frac{v_l + v_r}{2} \\ v_y = 0 \\ \omega_z = \frac{-\omega_l r + \omega_r r}{2y_0} = \frac{-v_l + v_r}{2y_0} \end{cases} \tag{11}$$

Noted $v_y = 0$, so that $v_G = v_x$ we can get the instantaneous radius of the path curvature:

$$\boldsymbol{R} = \frac{v_G}{\omega_z} = \frac{v_l + v_r}{-v_l + v_r} \boldsymbol{y_0} \tag{12}$$

A non-dimensional path curvature variable λ is introduced as the ratio of sum and difference of left- and right-side's wheel linear velocities [1], namely:

$$\boldsymbol{\lambda} = \frac{v_l + v_r}{-v_l + v_r} \tag{13}$$

And we can rewrite Equation (12) as:

$$\boldsymbol{R} = \frac{v_l + v_r}{-v_l + v_r} \boldsymbol{y_0} = \boldsymbol{\lambda y_0} \tag{14}$$

Then an ICR coefficient χ can be defined as:

$$\boldsymbol{X} = \frac{y_l + y_r}{B} = \frac{2y_0}{B}, \quad \boldsymbol{X \geq 1} \tag{15}$$

Where B denotes the lateral wheel bases, as illustrated in Fig2.3. The ICR coefficient X is equal to 1 when no slippage occurs (ideal differential drive). Note that the locomotion system introduces a non-holonomic restriction in the motion plane because the non-square matrix j_ω has no inverse. It is noted that the above expressions also present the kinematics for ideal wheeled differential drive vehicles, as illustrated in Fig2.4. Therefore, for instantaneous motion, kinematic equivalences can be considered between skid-steering and ideal wheel vehicles. The difference between both traction schemes is that whereas the ICR values for single ideal wheels are constant and coincident with the ground contact points, tread ICR values are dynamics-dependent and always lie outside of the tread centerlines because of slippage, so we can know that less slippage results in that tread ICRs are closer to the vehicle.

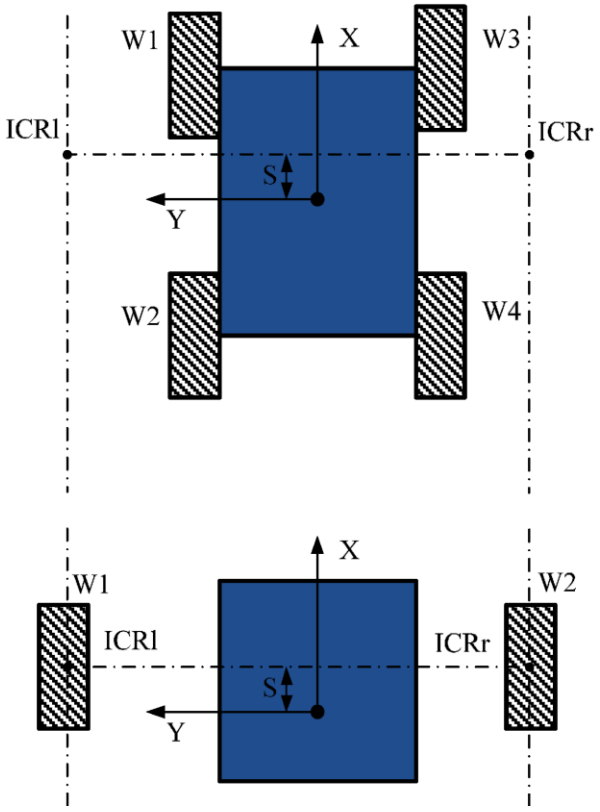


Fig 3.4: Geometric equivalence between the wheeled skid-steering robot and the ideal differential drive robot.

The major consequence of the study above is that the effect of vehicle dynamics is introduced in the kinematics model. Although the model does not consider the direct forces, it provides an accurate model of the underlying dynamics using lump parameters: $ICR_l \; and \; ICR_r$ Furthermore, from assumptions (1) and (3), we get a symmetrical kinematics model, and an ICR coefficient X from Equation (15) is defined to describe the model. The relationship between ICR coefficient and the vehicle motion path and velocity will be studied.

3.2.1.4 Skid-Steering Mobile Robot Dynamic Model

In this section, the effect of vehicle dynamics is introduced in the kinematics model. This section develops dynamic models of a skid-steering wheeled vehicle for the cases of 2D motion. Using the dynamic models, the relationship between the ICR coefficients and the path and velocity of the vehicle motion will be studied in a simulation.

In contrast to dynamic models described in terms of the velocity vector of the vehicle, the dynamic models here are described in terms of the angular velocity vector of the wheels. This is because the wheel velocities

are actually commanded by the control system, so this model form is particularly beneficial for motion simulation.

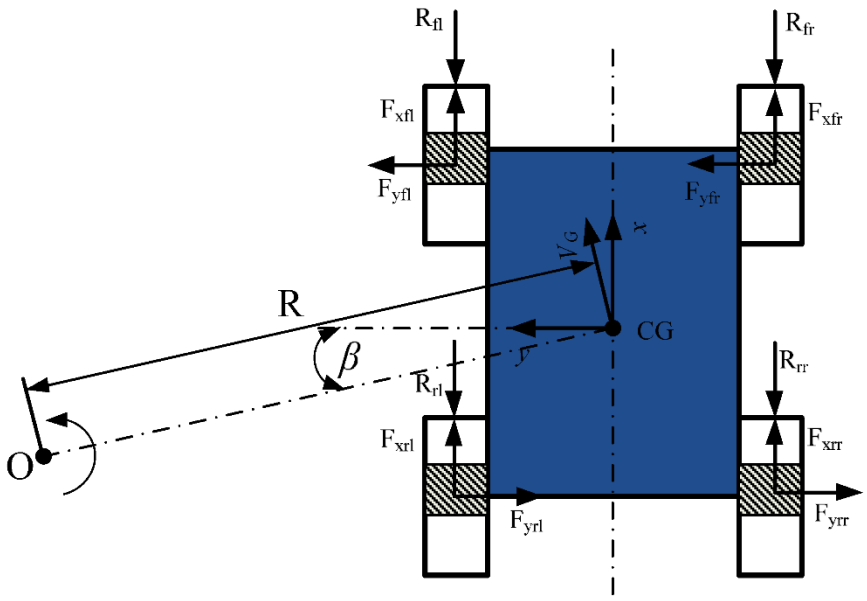


Fig 3.5: Forces and moments acting on a wheeled skid-steering vehicle during a steady state turn

As in Fig3.5, the dynamic model is given by:

$$\begin{cases} F_{xfr} + F_{xrr} + F_{xfl} + F_{xrl} - R_x - m\frac{V_G^2}{R}\sin\beta = 0 \\ F_{yfr} + F_{yrr} + F_{yfl} + F_{yrl} - m\frac{V_G^2}{R}\cos\beta = 0 \\ M_d - M_r = 0 \end{cases} \quad (16)$$

Where $V_G$the vehicle velocity, and β is the angle between the vehicle velocity and x-axis on the local frame. $F_{xfr}, F_{xrr}, F_{xfl}, F_{xrl}$ Are the longitudinal (friction) forces and $F_{yfr}, F_{yrr}, F_{yfl}, F_{yrl}$ are the lateral forces. $R_{fr} + R_{rr} + R_{fl} + R_{rl,}$ are external motion resistances on the four wheels. M_d Is the drive moment and M_r the resistance moment.

Based on the wheel-ground interaction theory, the shear stress τ_{rr} and shear displacement j relationship can be described as:

$$\tau_{rr} = p\mu(1 - e^{-\frac{j}{k}}) \quad (17)$$

Where p is the normal pressure, μ is the coefficient of friction and K is the shear deformation modulus. Fig.3.6 depicts a skid-steering wheeled vehicle moving counterclockwise (CCW) at constant linear velocity v

and angular velocity ϕ in a circle centered at O from position (1) to position (2). The four contact patches of the wheels with the ground are shadows in Fig.3.6. L and C are the patch-related distances. In the inertial X-Y frame, we define that $j_{fr}+j_{rr}+j_{fl}+j_{rl}$ and $y_{fr}+y_{rr}+y_{fl}+y_{rl}$ are respectively the shear displacements and sliding velocity angle (opposite direction of sliding velocity). The readers can refer to Yu's work [???] for a detailed analysis. The longitudinal sliding friction and lateral force of the four wheels can be expressed as follows:

$$\begin{cases} F_{xfr} = \int_{\frac{C}{2}}^{\frac{L}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p_r \mu_r \left(1 - e^{-\frac{j_{fr}}{K_r}}\right) \sin\left(\Pi - y_{fr}\right) dx_r dy_r \\ F_{yfr} = \int_{\frac{C}{2}}^{\frac{L}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p_r \mu_r \left(1 - e^{-\frac{j_{fr}}{K_r}}\right) \sin\left(\Pi - y_{fr}\right) dx_r dx y_r \end{cases} \tag{18}$$

$$\begin{cases} F_{xrr} = \int_{\frac{-l}{2}}^{\frac{-c}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p_r \mu_r \left(1 - e^{-\frac{j_{rr}}{K_r}}\right) \sin(\Pi - y_{rr}) dx_r dy_r \\ F_{yrr} = \int_{\frac{-c}{2}}^{\frac{-l}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p_r \mu_r \left(1 - e^{-\frac{j_{rr}}{K_r}}\right) \sin(\Pi - y_{rr}) dx_r dy_r \end{cases} \tag{19}$$

$$\begin{cases} F_{xfl} = \int_{\frac{C}{2}}^{\frac{L}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p_r \mu_r \left(1 - e^{-\frac{j_{fl}}{K_l}}\right) \sin\left(\Pi - y_{fl}\right) dx_l dy_l \\ F_{yfl} = \int_{\frac{C}{2}}^{\frac{L}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p_r \mu_r \left(1 - e^{-\frac{j_{fl}}{K_l}}\right) \sin\left(\Pi - y_{fl}\right) dx_l dx y_l \end{cases} \tag{20}$$

$$\begin{cases} F_{xrl} = \int_{\frac{-l}{2}}^{\frac{-c}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p_r \mu_r \left(1 - e^{-\frac{j_{rl}}{K_r}}\right) \sin(\Pi - y_{rl}) dx_l dy_l \\ F_{yrl} = \int_{\frac{-c}{2}}^{\frac{-l}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p_r \mu_r \left(1 - e^{-\frac{j_{rl}}{K_r}}\right) \sin(\Pi - y_{rl}) dx_l dy_l \end{cases} \tag{21}$$

Where p_l , μ_l and k_l are respectively the normal pressure, coefficient of friction, and shear deformation modulus of the left wheels, and p_r , μ_r and k_r are the ones of the right wheels, respectively.

With the other parameters directly measured or given, such as mass of vehicle, m, patch-related distances, L and C, width of wheel, b, p_l *and* p_r can be determined by $mg/2(c-l)b$ when a uniform normal pressure distributions assumption used $p_l = p_r$

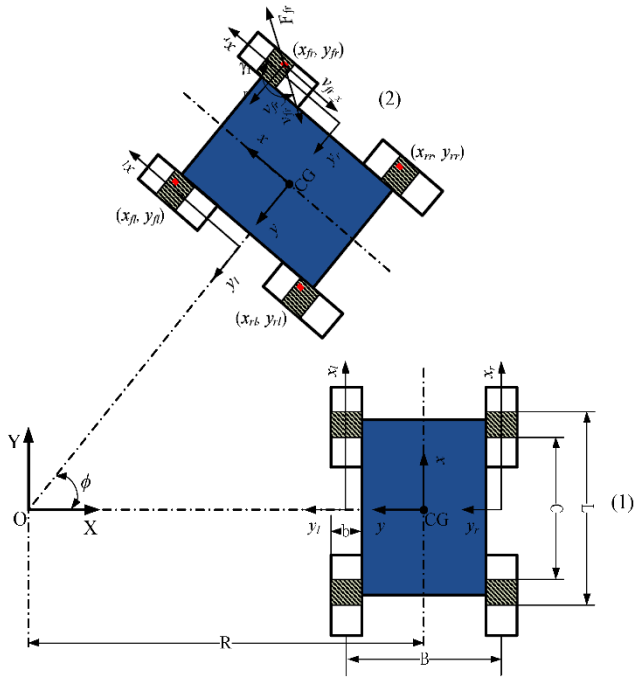


Fig 3.6: Motion of the skid-steering mobile robot wheel element on firm ground from position (1) to (2).

In Equation (16), the rolling resistance is denoted as μ_{roll}. We can obtain the resistance force, such that:

$$\boldsymbol{R_x = mg\mu_{roll}} \qquad (22)$$

3.2.1.5 Nonholononmic and holononmic constraints

Robots can be either mobile or stationary. Mobile robots include rolling robots, crawling robots, swimming robots, and many more. Stationary robots include robot arm, robot face, industrial robots, etc. Although known as stationary, these robots are not actually motionless but are confined to a small boundary. Each of these robots is designed to work on different platforms and the most common ones work either on Land, Air, Water, space, etc. Some of the robots are designed to work on more than one platform and can shift from land to water to air. Based on the way robots move, they can be further classified as "Holonomic" or "Non-Holonomic" drive Robots.

Holonomic Drive

Holonomic refers to the relationship between the controllable and total degrees of freedom of a robot. If the controllable degree of freedom is equal to the total degrees of freedom, then the robot is said to be

Holonomic. A robot built on castor wheels or Omni-wheels is a good example of Holonomic drive as it can freely move in any direction and the controllable degrees of freedom are equal to total degrees of freedom. An omnidirectional wheel is an example of holonomic type.

Non-Holonomic Drive

If the controllable degree of freedom is less than the total degree of freedom, then it is known as a non-Holonomic drive. For example, a normal car has three degrees of freedom, its position in two axes, and its orientation. However, there are only two controllable degrees of freedom which are acceleration (or braking) and the turning angle of the steering wheel. This makes it difficult for the driver to turn the car in any direction unless the car skids or slides.

Our IMR is a non-holonomic type as it depends on the skid steering drive model.

3.2.2 Robot Arm Kinematics

3.2.2.1 Degree of Freedom

In robotics, the term "degree of freedom" (DOF) refers to the number of independent variables or parameters that can be controlled in a robotic system. It represents the number of ways in which a robot or a mechanism can move or change its configuration. Each degree of freedom corresponds to a specific joint or actuator in the robot, allowing it to perform a particular type of motion. The more degrees of freedom a robot has, the greater its range of possible motions and flexibility in performing tasks. In a robot arm, for example, each joint typically contributes one degree of freedom. Common types of joints include revolute joints (allowing rotation about an axis) and prismatic joints (allowing linear motion along an axis). The total number of degrees of freedom in a robot arm is the sum of the degrees of freedom for each joint. The concept of degrees of freedom is crucial for robot design, control, and motion planning. It helps determine the system's complexity, range of motions, workspace, and the number of control inputs required to operate the robot effectively. It also affects the robot's ability to navigate and interact with its environment. Robot systems can have varying degrees of freedom depending on their intended applications. For instance, industrial robot arms can range from 4 to 6 DOF, while humanoid robots often have many DOF to mimic human movements and perform complex tasks. The appropriate number of degrees of freedom is determined based on the specific requirements and tasks the robot needs to accomplish.

IMR’s robot arm

Our robot arm is considered to be 5 degrees of freedom (DOF) refers to a robotic system or manipulator that has five independent parameters or variables that define its configuration or motion. We choose this configuration as it is commonly used and it is not very hard to implement.

1-Robot Arm: A common representation of a robot with 5 DOF is a robotic arm with five articulated joints. Each joint contributes one degree of freedom, allowing the arm to move in different ways.

2-Joint Configuration: The specific configuration of the 5 joints can vary depending on the robot design, but here's our configuration:

- Base Rotation: The base of the robot arm can rotate horizontally, providing one degree of freedom.
- Shoulder Joint: The arm can raise or lower at the shoulder, contributing another degree of freedom.
- Elbow Joint: The arm can bend or extend at the elbow, introducing the third degree of freedom.
- Wrist Pitch: The wrist can pitch up or down, providing the fourth degree of freedom.
- Gripper: The gripper can be considered also one degree as it grasps objects, contributing to the fifth degree of freedom.

3-Task Capabilities: A 5-DOF robot arm can perform various tasks such as reaching specific points in space, manipulating objects, performing pick-and-place operations, or following predefined trajectories within its workspace.

3.2.2.2 Forward Kinematics

Kinematic analysis

Kinematics analysis of robots has defined the relationship between the links and joints with the position and orientation of the robot. The kinematics analysis is divided into forward kinematics and inverse kinematics.

Forward kinematics is used to determine the position and orientation of the end-effector of the robotic arm from the specified joint angles. The DH method is one of the methods that are commonly used in forward kinematics which represents the relationship of the joint coordinate between two Links as shown in Fig.3.6.

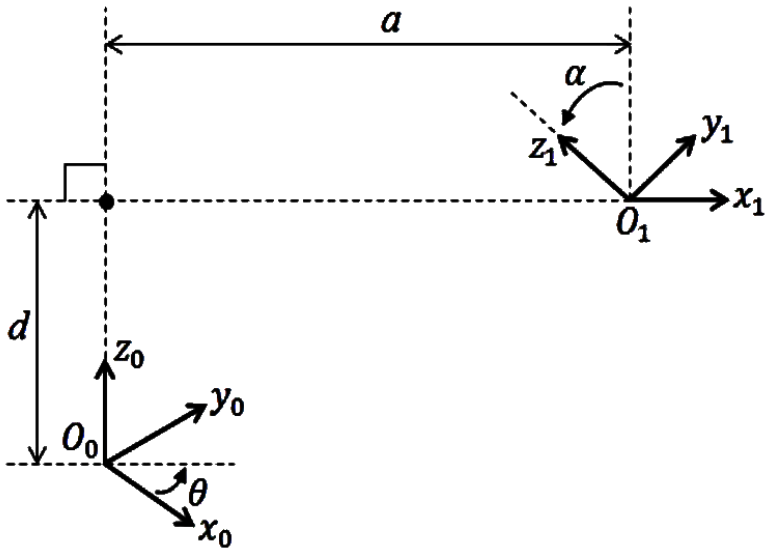


Fig 3.7: Representation of joint coordinates between two links

Where:

a_i : Link length. α_i : Link twist. d_i : Link offset. θ_i : Joint angle

Our robotic arm has 5 DOF and the DH parameters of this arm are listed in Table (1).

Link	$a_i\ (mm)$	$\alpha_i\ (mm)$	$d_i\ (mm)$	$\theta_i\ (degree)$
1	0	90	0	θ_1
2	105	0	0	θ_2
3	100	0	0	θ_3
4	0	90	0	θ_4
5	0	0	150	θ_5

Fig 3.8: DH parameters for the robotic arm

Also, the transformation matrix between two successive links can be obtained using the DH frame as follow:

$$T_i = R_{z,\theta_i} T_{z,d_i} T_{x,a_i} R_{x,\alpha_i}$$

$$T_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_i = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where:

$$c\theta_i = \cos(\theta_i)\ s\theta_i = \sin(\theta_i)$$

$$c\alpha_i = \cos(\alpha_i)$$

$\sin \alpha_i = \sin(\alpha_i)$ The individual matrices and the global matrix of the 5 DOF robotic arm can be obtained by substituting the DH parameters in Table (1) in the equation (1), as showed in fig.3.7

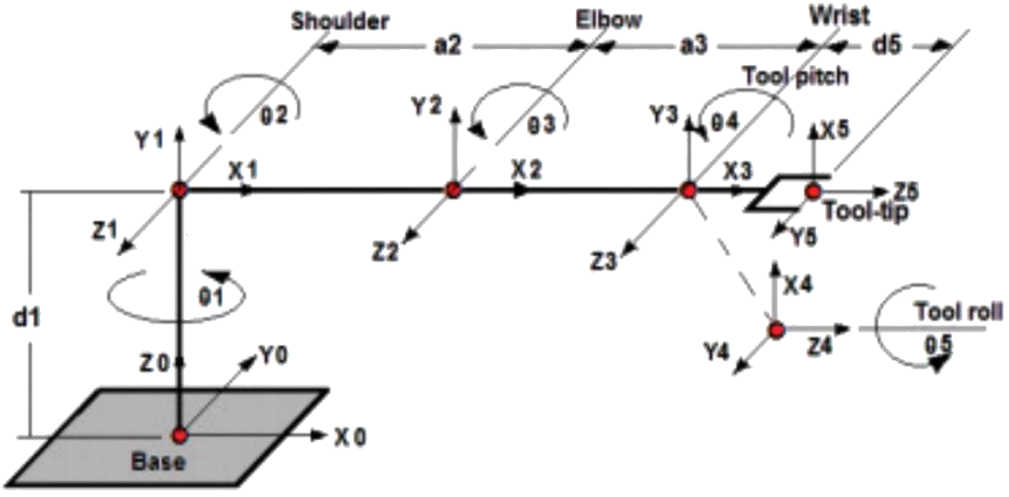


Fig 3.9: The coordinate frame of the arm

$$A_1^0 = R_{z,\theta_1} T_{z,d_1} T_{x,a_1} R_{x,\alpha_1}$$

$$A_1^0 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_2^1 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2 = \begin{bmatrix} c_3 & -s_3 & 0 & a_3c_3 \\ s_3 & c_3 & 0 & a_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_4^3 = \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5^4 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And the global matrix A_5^0 :

$$A_5^0 = A_1^0 A_2^1 A_3^2 A_4^3 A_5^4$$

$$A_5^0 = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} m_{11} &= c_{12345} + s_{15} \\ m_{12} &= -s_5 c_{1234} + s_1 c_5 \\ m_{13} &= c_1 s_{234} \\ m_{14} &= c_1(d_5 s_{234} + a_3 c_{23} + a_2 c_2) \\ m_{21} &= s_1 c_{2345} - c_1 s_5 \\ m_{22} &= -s_{15} c_{234} - c_{15} \\ m_{23} &= s_1 s_{234} \\ m_{24} &= s_1(d_5 s_{234} + a_3 c_{23} + a_2 c_2) \\ m_{31} &= c_5 s_{234} \\ m_{32} &= -s_5 s_{234} \\ m_{33} &= -c_{234} \\ m_{34} &= -d_5 c_{234} + a_3 s_{23} + a_2 s_2 + d_1 \end{aligned}$$

Where:

$$c_{12345} = \cos(\theta_1 + \theta_2 + \theta_3 + \theta_4 + \theta_5)$$

$$s_{15} = \sin(\theta_1 + \theta_5)$$

3.2.2.3 Inverse kinematics

Inverse Kinematics (IK) is used to determine the required joints angles of the robotic arm to achieve the specified position and orientation of the end-effector of the robotic arm. In this work, the geometric approach was used to solve the inverse kinematics of the 5DOF robotic arm:

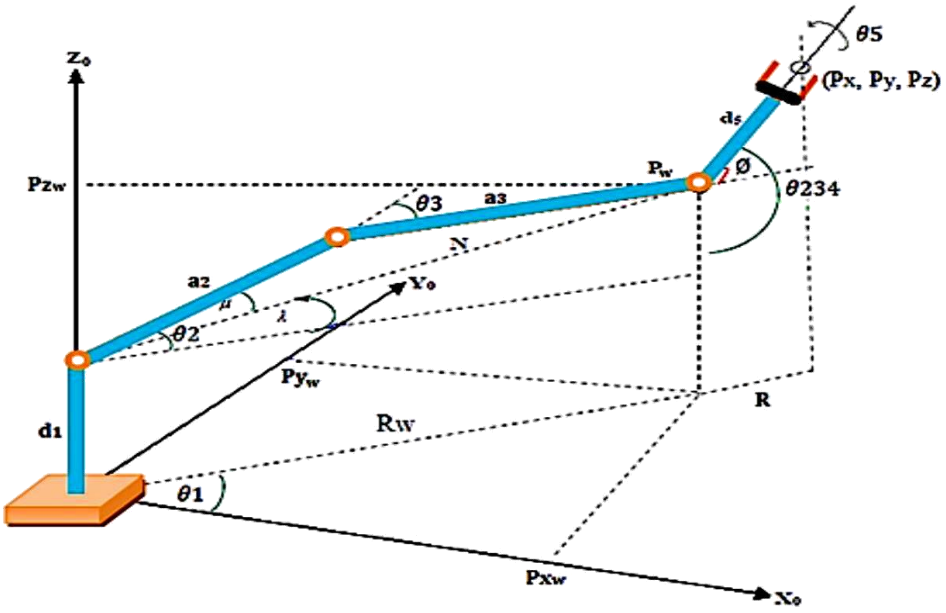


Fig 3.10: Geometric frame of the arm

From fig.3.7, the wrist angle relative to the reference coordinate (θ_{234}) represents the relation between(θ_2,θ_3, and θ_4) :

$$\boldsymbol{\theta_{234} = \theta_2 + \theta_3 + \theta_4}$$

Where θ_{234} can be calculated based on pitch wrist orientation angle ∅

$$\boldsymbol{90 - \theta_{234} = \pm \emptyset}$$

$$\boldsymbol{R = d_5 * \cos \emptyset}$$

$$\boldsymbol{P_{x_w} = P_x - R\cos \theta_1 = X_w}$$

$$\boldsymbol{P_{y_w} = P_y - R\sin \theta_1 = Y_w}$$

$$\boldsymbol{P_{z_w} = P_z + d_5 \sin \emptyset = Z_w}$$

$$R_w = \sqrt{P_x^2 + P_y^2} - R = \sqrt{P_{x_w}^2 + P_{y_w}^2}$$

$$N = \sqrt{\left(P_{z_w} - d_1\right)^2 + R_w^2}$$

Solution for θ_1 : base angle can be calculated:

$$\theta_1 = \tan^{-1}\left(\frac{P_y}{P_x}\right)$$

The other solution will be:

$$\grave{\theta_1} = \theta_1 + 180$$

Solution for θ_2 : by using the law of cosines:

$$a_3^2 = N^2 + a_2^2 - 2a_2 N\cos(\mu) \Rightarrow$$

$$\mu = \cos^{-1}\left(\frac{N^2 + a_2^2 - a_3^2}{2a_2 N}\right)$$

$$\lambda = \tan^{-1}\left(\frac{P_{z_w} - d_1}{R_w}\right)$$

$$\theta_2 = \lambda \mp \mu$$

Solution for θ_3 :

$$N = \sqrt{\left(a_2^2 + a_3^2 - 2a_2 a_3\cos(\pi - \theta_3)\right)}$$

$$\theta_3 = \pm\cos^{-1}\left(\frac{N^2 - a_2^2 - a_3^2}{2a_2 a_3}\right)$$

Solution for θ_4 :

$$\theta_4 = \theta_{234} - \theta_2 - \theta_3$$

3.3 NAVIGATION AND LOCALIZATION

3.3.1 Kalman Filter

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.

The purpose of this section is to provide a practical introduction to the discrete Kalman filter. This introduction includes a description and some discussion of the basic discrete Kalman filter, a derivation, a description, and some discussion of the extended Kalman filter, and a relatively simple (tangible) example with real numbers & results.

Since Kalman published his famous paper, due in large part to advances in digital computing, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation and autonomous robots.

Kalman Filtering can be understood as a way of making sense of a noisy world. When we want to determine where a robot is located, we can rely on two things: We know how the robot moves from time to time since we command it to move in a certain way. This is called state transitioning (i.e. how the robot moves from one state to the other). And we can measure the robot's environment using its various sensors such as cameras, lidar, or any other laser sensor. The problem is that both sets of information are subject to random noise. We do not know exactly how exactly the robot transitions from state to state since actuators are not perfect and we cannot measure the distance to objects with infinite precision. This is where Kalman Filtering comes in.

Kalman Filtering allows us to combine the uncertainties regarding the current state of the robot (i.e. where it is located and in which direction it is looking) and the uncertainties regarding its sensor measurements and to ideally decrease the overall uncertainty of the robot. Both uncertainties are usually described by a Gaussian probability distribution or Normal distribution. A Gaussian distribution has two parameters: mean and variance. The mean expresses, what value of the distribution has the highest probability to be true, and the variance expresses how uncertain we are regarding this mean value.

The algorithm works in a two-step process. The prediction step and the update step, in the prediction cycle the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to

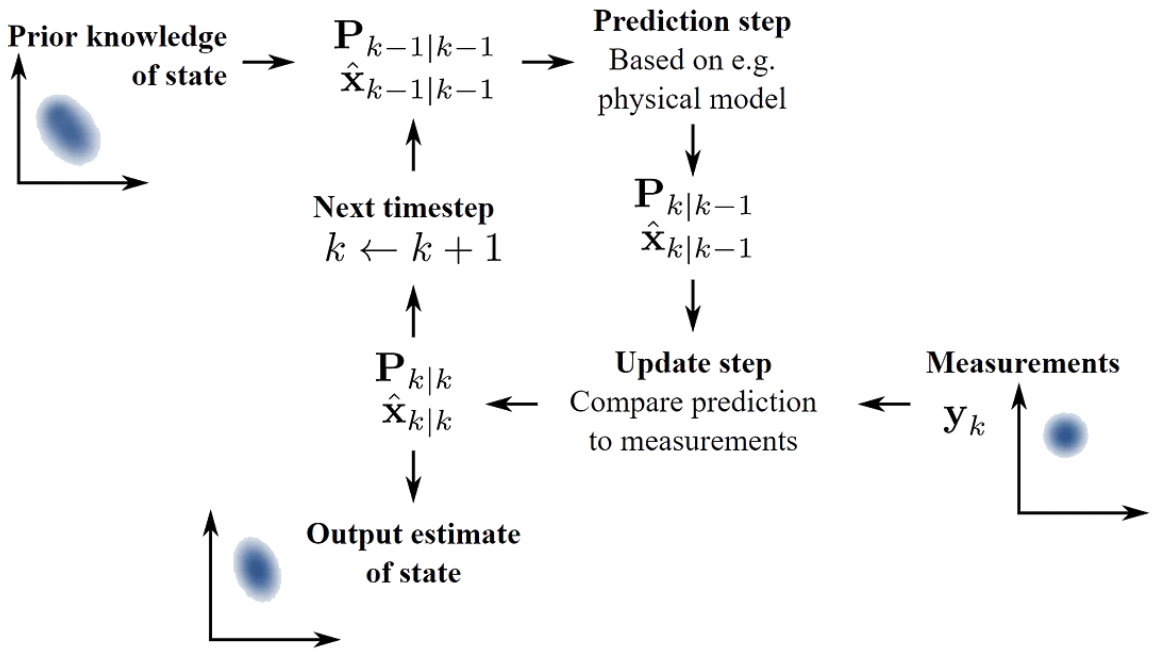


Fig 3.11: Kalman Filtering

estimates with higher certainty. The algorithm is recursive. It can run in real-time using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

3.3.2 Extended Kalman Filter

Extended Kalman Filtering is an extension of "Normal" Kalman Filtering. What I did not tell you in the last section is one additional assumption that was made implicitly when using Kalman Filters: The state transition model and the measurement model must be linear. From a mathematical standpoint, this means that we can use Linear Algebra to update the robot's state and the robot's measurements. In practice, this means that the state variables and measured values are assumed to change linearly over time. For instance, if we measure the robot's position in the x-direction. We assume that if the robot was at position x1 at time t1, it must be at position x1+v*(t2–t1) at time t2. The variable v denotes the robot's velocity in the x-direction. If the robot is accelerating or doing any other kind of nonlinear motion (e.g. driving around in a circle), the state transition model is slightly wrong. Under most circumstances, it is not wrong by much, but in certain edge cases, the assumption of linearity is simply too wrong.

Also assuming a linear measurement model comes with problems. Assume you are driving along a straight road and there is a lighthouse right next to the road in front of you. While you are quite some distance

away, your measurement of your distance to the lighthouse and the angle in which it lies from your perspective changes pretty much linearly (the distance decreases by roughly the speed your car has and the angle stays more or less the same). But the closer you get and especially while you drive past it, the angle, changes dramatically, and the distance, on the other hand, does not change very much. This is why we cannot use Linear Kalman Filtering for the Robot when he is navigating his 2-D world with landmarks scattered across his 2-D plane.

Extended Kalman Filter removes the restriction of linear state transition and measurement models. Instead, it allows you to use any kind of nonlinear function to model the state transition and the measurements you are making with your robot. In order to still be able to use the efficient and simple Linear Algebra magic in our filter, the extended filter linearizes the models around the current robot state. This means that we assume the measurement model and the state transition model to be approximately linear around the state at which we are right now. But after every time step, we update this linearization around the new state estimate. While this approach forces us to make a linearization of this nonlinear function after every time step, it turns out to be not computationally expensive.

Extended Kalman Filtering is basically "Normal" Kalman Filtering just with additional linearization of the now nonlinear state transition model and measurement model.

In our case where the robot is lost and wants to localize in this (arguably) hostile environment, the Extended Kalman Filtering enables the robot to sense the landmarks and update its perception of its state accordingly. If the variance of the state estimate and the measurement estimate is low enough, then the robot is very quickly very sure where he is located with respect to the landmarks and since he knows exactly where the landmarks are, he knows where he is.

3.3.3 Adaptive Monte Carlo Localization (Amcl)

Robot Localization is a critical component of robotics technology, enabling a robot to determine its position and orientation within an environment. One of the most popular localization techniques is Adaptive Monte Carlo Localization (AMCL), which is a probabilistic algorithm that uses a particle filter to estimate a robot's position and orientation in real time.

As we know from earlier robot localization is the process of determining the pose of a robot within a known or unknown environment. The pose of a robot includes its position (x, y, z) and orientation (roll, pitch, yaw) relative to a reference coordinate frame.

There are several methods for robot localization, including odometry, GPS, Imu, and landmark-based methods. Odometry relies on measuring the rotation and displacement of a robot's wheels or other locomotion mechanisms, Landmark-based methods rely on identifying and tracking distinct features in an environment, such as walls or corners.

AMCL is a probabilistic algorithm that uses a particle filter to estimate a robot's 2D pose based on sensor data. The algorithm works by representing the robot's pose as a distribution of particles, where each particle represents a possible pose of the robot.

At each time step, the robot takes sensor measurements and compares them to a map of the environment to determine the likelihood of each particle being the true pose of the robot. The particles with low likelihoods are discarded, while the particles with high likelihoods are resampled to generate a new set of particles for the next time step.

The process of resampling ensures that the particles remain distributed around the true pose of the robot, even if the robot's movement is uncertain or the environment is partially observable.

Advantages and Limitations of AMCL

One of the main advantages of AMCL is its ability to handle non-linear and non-Gaussian sensor models, making it suitable for a wide range of robot applications. The algorithm is also computationally efficient, allowing it to run in real-time on low-power hardware.

However, AMCL has several limitations. One of the main challenges is tuning the algorithm parameters, such as the number of particles and the sensor model, to ensure accurate localization. In addition, AMCL requires a map of the environment, which may not always be available or accurate.

3.3.4 Simultaneous Localization and Mapping (SLAM)

The simultaneous localization and mapping (SLAM) problem asks if a mobile robot can be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. A solution to the SLAM problem has been seen as one of the most remarkable achievements for the mobile robotics community as it would provide the means to make a robot truly autonomous.

The "solution" to the SLAM problem has been one of the notable successes of the robotics community over the past decade. SLAM has been formulated and solved as a theoretical problem in many different forms. SLAM has also been implemented in many different domains from indoor robots to outdoor, underwater, and airborne systems. At a theoretical and conceptual level, SLAM can now be considered a solved problem. However, substantial issues remain in practically realizing more general SLAM solutions and notably in building and using perceptually rich maps as part of a SLAM algorithm.

The first thing to take into account for SLAM is the sensors used. There are a wide variety of intrusive (in the environment, such as QR codes), and non-intrusive (in the robot, GPS, IMU…). Each type of sensor can be used for one purpose, for example, IMUs will allow us to get the robot body's acceleration and velocities while the GPS will get the position of the robot in an outdoor environment. There is also the LIDAR sensor that creates a point cloud that can be analyzed. The ones we will be interested in are those related to visual perception, thus monocular, stereo, and RGB-D cameras. Quick features of each of these are:

- **Monocular camera**: Have you ever tried trying to catch a ball in the air by covering one eye? That's the main problem with monocular cameras, it is hard to measure the real distance from a single image.

- **Stereo cameras**: Having two cameras makes measuring the distance much easier if we know the camera parameters. Yes, you guessed it right, each of our eyes acts as one camera separated from the other one. This distance is called the baseline, and in order to measure the distance to an object, this object needs to be matched between pictures.
- **RGB-D cameras**: This is a newer type of camera, and apart from the RGB image, it provides a Depth image containing the depth values of each point. It offers a dense point cloud and is very useful for indoor environments, as it is affected by sunlight.
- We got the basics, now let's dive deeper into how the Visual SLAM algorithm works. It is divided into five main steps.
- **Sensor data acquisition**: Data is read from our cameras so that it can be processed by the system.
- **Visual Odometry**: Estimate the motion between consecutive camera frames. This is not enough to solve the SLAM problem as it will cause accumulative drift.
- **Backend Optimization**: Adjust the visual odometry results for the noise caused by the sensors.
- **Loop Closing**: Detect when a point has been identified twice, and correct the entire map for the accumulated drift error.
- **Reconstruction**: Build the map based on the measurements and correction step. Maps can be metric (emphasize metrical location for objects) and topological (emphasize relationships among map objects).

Gmapping

Gmapping is a ROS package that provides laser-based SLAM, as a ROS node called slam_gmapping. Using slam_gmapping, we can create a 2-D occupancy grid map (like a building floor plan) from laser and pose data collected by a mobile robot. The gmapping ROS package implements the grid-based FastSLAM algorithm. It uses a particle filter approach to estimate the robot trajectory and a low dimensional Extended Kalman Filter to estimate features of the map.

3.4 ROBOT MOTION

3.4.1 Introduction

Robot motion refers to the movement or locomotion of a robot. It involves controlling the robot's actuators, such as motors or pneumatic systems, to achieve desired movements and perform specific tasks. Robot motion can range from simple movements like linear motion (forward, backward) and rotational motion (turning) to more complex motions involving multiple degrees of freedom, such as picking and placing objects, navigating through obstacles, or performing intricate maneuvers.

There are several ways to control robot motion, depending on the type of robot and its application:

1. **Manual Control:** In this method, a human operator directly controls the robot's motion using a joystick, control panel, or other input devices. The operator commands the robot's movement in real-time, providing instructions for speed, direction, and other parameters.

2**. Pre-programmed Motion:** Robots can be programmed to follow a predetermined set of motions. These pre-defined motions are typically created using a programming language or a graphical interface, specifying waypoints, trajectories, and other parameters. Once the program is executed, the robot follows the predefined motion sequence.

3. Path Planning: Path planning involves determining a collision-free path for the robot to move from its current position to a desired goal position. Algorithms, such as A* (A-star) or Dijkstra's algorithm, can be used to calculate an optimal path based on the robot's kinematic constraints and the environment's obstacles.

4. Sensor-based Motion: Robots equipped with sensors, such as cameras, LIDAR (Light Detection and Ranging), or proximity sensors, can perceive the surrounding environment and adjust their motion accordingly. Sensor feedback can be used to detect obstacles, navigate around them, or maintain a specific distance from objects.

5. Reactive Control: Reactive control relies on real-time sensor feedback and immediate reaction to the environment. Instead of planning a complete motion sequence, the robot continuously adjusts its motion based on the sensed data. Reactive control is often used in dynamic and unpredictable environments.

6. Closed-Loop Control: Closed-loop control involves continuously monitoring the robot's position or other relevant parameters and comparing them to the desired values. Any deviations are corrected by adjusting the control signals sent to the actuators. This feedback loop ensures accurate and precise robot motion.

The specific methods and techniques used for robot motion control depend on the robot's design, its intended application, and the complexity of the tasks it needs to perform. Advanced robotic systems may combine multiple motion control approaches to achieve sophisticated movements and behaviors.

3.4.2 Path Planning

Path planning is a technique used in robotics and autonomous systems to determine a feasible and optimal path for a robot to move from its current location to a desired goal location while avoiding obstacles and adhering to various constraints. It is an essential component in robotic motion planning and navigation.

The path planning process typically involves the following steps:

Environment Modeling: The robot's operating environment is modeled, representing obstacles, boundaries, and other relevant features. This can be done using geometric representations such as grids, maps, or point clouds.

Start and Goal Configuration: The robot's initial position (start configuration) and the desired destination (goal configuration) are defined. These configurations are often represented as coordinates in the environment model or as a set of parameters describing the robot's pose.

Path Search: Algorithms are employed to search for a valid path from the start configuration to the goal configuration while avoiding collisions with obstacles. Various algorithms can be used for this purpose, such as A* (A-star), Dijkstra's algorithm, Rapidly-exploring Random Trees (RRT), or Probabilistic Roadmaps (PRM).

Path Optimization: Once a path is found, it can be further optimized to improve its quality. Optimization techniques aim to minimize path length, smooth out the trajectory, reduce energy consumption, or optimize for other criteria. Common optimization methods include spline interpolation, polynomial fitting, or graph-based algorithms like the A* algorithm applied to a finer discretization of the path.

Collision Checking: The generated path is checked for collisions with obstacles in the environment. Collision detection algorithms compare the robot's geometry or bounding volume with the obstacles' representations to ensure that the planned path remains collision-free.

Execution and Feedback: The planned path is followed by the robot using appropriate control commands and motion execution techniques. During execution, the robot's sensors may provide feedback on the actual environment, and the planned path may need to be adjusted in real-time to account for dynamic obstacles or changes in the environment.

Path planning algorithms and techniques can vary depending on the complexity of the environment, the type of robot, and the specific requirements of the application. The goal is to find a safe and efficient path that allows the robot to navigate autonomously while accomplishing its tasks.

Dijkstra's algorithm

Dijkstra's algorithm is a graph search algorithm used to find the shortest path between a start node and all other nodes in a weighted graph. It was named after Dutch computer scientist Edger W. Dijkstra and is commonly used in path planning and routing applications.

Here's a step-by-step explanation of Dijkstra's algorithm:

Initialization: Assign a tentative distance value to every node in the graph. Set the distance of the start node to 0 and all other nodes to infinity or a very large value. Mark all nodes as unvisited.

Select the Node: Choose the node with the smallest tentative distance as the "current node" and mark it as visited. Initially, the start node will be the current node.

Update Distances: If the calculated tentative distance for a neighboring node is smaller than its current assigned distance, update the neighbor's distance value with the newly calculated value.

Repeat Steps 2-4: Once all the neighbors of the current node have been evaluated, mark the current node as visited. If the destination node has been visited, or if the smallest tentative distance among the unvisited nodes is infinity, stop the algorithm. Otherwise, select the unvisited node with the smallest tentative distance as the next "current node" and go back to step 3.

Path Reconstruction: After the algorithm finishes, the shortest path from the start node to any other node can be obtained by backtracking from the destination node to the start node, following the nodes with the lowest cumulative distance.

Dijkstra's algorithm guarantees to find the shortest path from the start node to all other nodes in a graph as long as the graph doesn't contain negative edge weights. If negative weights are present, a modified version of Dijkstra's algorithm called the Bellman-Ford algorithm is typically used.

Dijkstra's algorithm is widely used in various applications, such as route planning in transportation networks, network routing protocols, and finding the shortest paths in maps or graphs.

Here's a more detailed explanation of Dijkstra's algorithm:

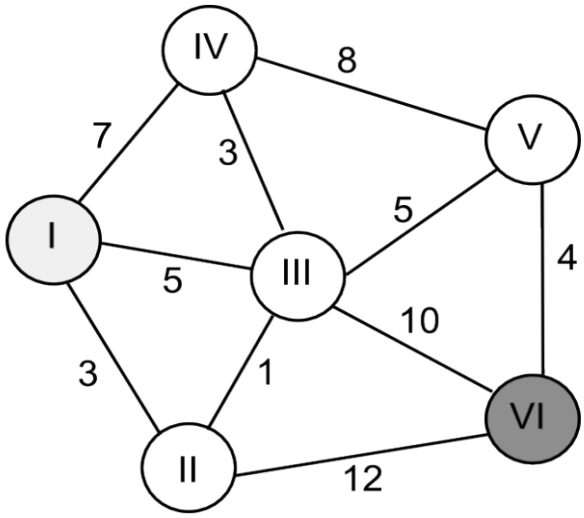


Fig 3.12: Dijkstra's algorithm graph

Input:

- A weighted graph, represented as a collection of nodes and edges.
- Each edge has an associated non-negative weight or cost.
- A start node from which to find the shortest paths.

Output:

- Shortest paths from the start node to all other nodes in the graph.

A* algorithm

The A* (A-star) algorithm is a popular graph search algorithm used to find the shortest path between a start node and a goal node in a weighted graph. It is an extension of Dijkstra's algorithm with the added benefit of using heuristics to guide the search process, making it more efficient.

Here's an explanation of the A* algorithm:

Steps of the A* Algorithm:

1. Initialization:
   - Assign a tentative distance value to every node in the graph. Set the distance of the start node to 0 and all other nodes to infinity or a very large value.
   - Calculate the heuristic value (h-value) for each node, which represents the estimated cost from that node to the goal node. This heuristic function should be admissible, meaning it never overestimates the actual cost.
2. Open and Closed Lists:
   - Create an open list to store nodes that are being considered for expansion.
   - Create a closed list to store nodes that have already been evaluated.
3. Select the Node:
   - Add the start node to the open list.
   - While the open list is not empty, do the following:
   - Select the node from the open list with the lowest sum of the tentative distance (g-value) and the heuristic value (h-value). This node will be the "current node."
   - Move the current node from the open list to the closed list to mark it as visited.
4. Evaluate Neighbors:
   - For each neighboring node of the current node:
   - Calculate the tentative distance from the start node to the neighboring node by adding the weight of the edge connecting them to the current node's tentative distance.
   - If the neighboring node is already in the closed list and the new tentative distance is greater than its current distance, skip it.
   - If the neighboring node is not in the open list or the new tentative distance is less than its current distance, update the neighbor's distance value and add it to the open list.
   - Calculate the f-value (f = g + h) for the neighboring node, representing the total estimated cost from the start node to the goal node passing through that node.
5. Repeat Steps 3-4:
   - Continue selecting nodes from the open list and evaluating their neighbors until either the goal node is reached or the open list is empty.
6. Path Reconstruction:
   - If the goal node is reached, the shortest path from the start node to the goal node can be reconstructed by following the nodes with the lowest f-value, starting from the goal node and tracing back to the start node.

The A* algorithm combines the advantages of both Dijkstra's algorithm (which guarantees finding the shortest path) and a heuristic function (which guides the search towards the goal). By using the heuristic, A* focuses the search on promising paths, reducing the number of nodes to be evaluated compared to uninformed algorithms like Dijkstra's algorithm.

The key component of the A* algorithm is the selection of the node with the lowest f-value, which takes into account both the cost to reach the current node (g-value) and the estimated cost to reach the goal node (h-value). The heuristic function should be admissible, meaning it never overestimates the actual cost, to ensure the optimality of the algorithm.

A* is widely used in various applications, including path planning in robotics, route finding in navigation systems, and artificial intelligence algorithms. It provides an efficient and effective way to find the shortest path in a graph while considering both the actual cost and the estimated cost to the goal.

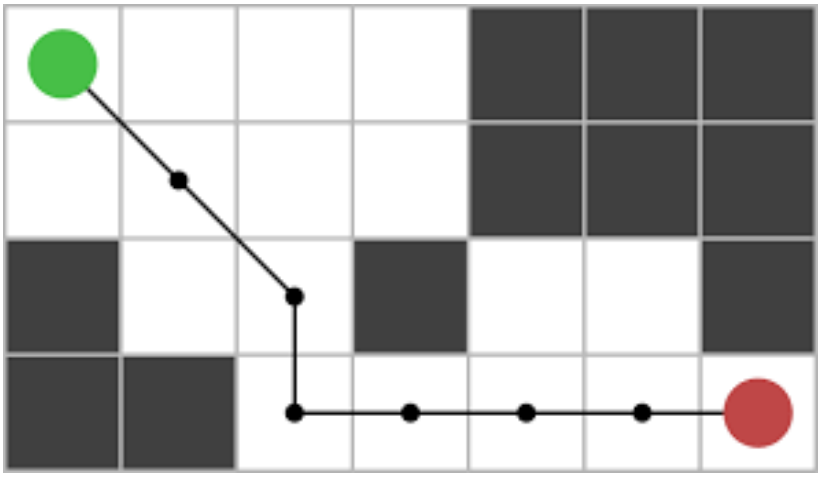


Fig 3.13: A* algorithm graph

Input:

- A weighted graph is represented as a collection of nodes and edges.
- Each edge has an associated non-negative weight or cost.
- A start node from which to find the shortest path.
- A goal node represents the destination.

Output:

- The shortest path from the start node to the goal node.

3.4.3 Path planning in configuration space

3.4.3.1 Introduction

Path planning in configuration space, also known as C-space, is a technique used in robotics to plan the motion of a robot in a high-dimensional space. The configuration space represents all possible configurations or poses of a robot, taking into account the constraints and geometry of the robot and its environment. Path planning in C-space is particularly useful for robots with multiple degrees of freedom (DOF) such as robotic arms, humanoid robots, or mobile manipulators.

Here's an overview of path planning in configuration space:

1. Configuration Space:
- The configuration space, often denoted as C-space, is a mathematical representation of all possible configurations or poses that a robot can attain. Each point in the C-space corresponds to a specific set of joint angles or parameters describing the robot's pose.

2. Environment Modeling:
- The robot's operating environment is modeled in the configuration space, taking into account the obstacles, boundaries, and constraints. The obstacles are represented as regions or volumes in the C-space that the robot cannot occupy.

3. Start and Goal Configurations:
- The start configuration represents the initial pose of the robot, while the goal configuration represents the desired final pose. These configurations are points in the C-space.

4. Path Search:
- Path planning algorithms are employed to search for a feasible and collision-free path from the start configuration to the goal configuration in the C-space. The goal is to find a continuous path that satisfies the robot's constraints and avoids collisions with obstacles.

5. Configuration Space Sampling:
- Path planning algorithms often use configuration space sampling techniques to explore the C-space and find a feasible path. Sampling-based algorithms, such as Rapidly-exploring Random Trees (RRT) or Probabilistic Roadmaps (PRM), randomly sample configurations in the C-space and connect them to form a graph representation.

6. Collision Checking:
- During the path planning process, collision checking is performed to ensure that the planned path does not intersect with obstacles in the C-space. Collision detection algorithms compare the robot's geometry or bounding volume with the obstacles' representations to determine if there is a collision.

7. Path Optimization:
- Once a path is found, it can be further optimized to improve its quality. Optimization techniques aim to minimize path length, smooth out the trajectory, reduce energy consumption, or optimize for other criteria. Common optimization methods include spline interpolation, polynomial fitting, or graph-based algorithms.

8. Execution and Feedback:
- The planned path is followed by the robot using appropriate control commands and motion execution techniques. During execution, the robot's sensors may provide feedback on the actual environment, and the planned path may need to be adjusted in real-time to account for dynamic obstacles or changes in the environment.

Path planning in configuration space is a challenging task due to the high dimensionality and complexity of the C-space. Various algorithms and techniques, including sampling-based methods, optimization algorithms, and collision-checking techniques, are employed to efficiently search for feasible and optimal paths for robots with multiple degrees of freedom. The goal is to find a safe and efficient path that allows the robot to navigate autonomously while accomplishing its tasks.

3.4.3.2 Grid based algorithms

Grid-based algorithms are a class of path-planning algorithms commonly used in robotics and computer graphics to find paths in a grid-like environment. These algorithms discretize the continuous space into a grid, where each grid cell represents a possible location or state. Grid-based algorithms are particularly suited for environments where the robot's motion is restricted to a set of discrete positions or when working with maps represented as grids.

Here are a few grid-based algorithms commonly used for path planning:

- Breadth-First Search (BFS): BFS is an uninformed search algorithm that explores all nodes at the same depth level before moving to the next level. In a grid-based setting, BFS can be used to find the shortest path between two points. It guarantees to find the shortest path but may not be the most efficient algorithm for large grids or complex environments.
- Depth-First Search (DFS): DFS is another uninformed search algorithm that explores paths as deeply as possible before backtracking. While DFS can find paths quickly, it does not guarantee optimality. In a grid-based environment, DFS can be used to explore possible paths and find a feasible solution, but it may not always find the shortest path.
- Dijkstra's Algorithm: Dijkstra's algorithm, discussed earlier, can also be used in a grid-based environment. In this case, each grid cell represents a node, and the weights on the edges correspond to the costs of moving between adjacent cells. Dijkstra's algorithm guarantees to find the shortest path from a start cell to all other cells in the grid.
- A* Algorithm: The A* algorithm, discussed earlier in more detail, can be applied to grid-based environments. In grid-based path planning, A* uses heuristics to guide the search process efficiently. The heuristic function can be based on Euclidean distance, Manhattan distance, or any other appropriate measure to estimate the cost from a grid cell to the goal cell. A* is widely used in grid-based path planning due to its efficiency and ability to find optimal paths.
- D* Lite:D* Lite is an incremental search algorithm designed for dynamic grid-based environments where the costs or the environment itself can change during the planning process. It updates the path incrementally based on new information and adapts to changes efficiently. D* Lite is particularly useful for real-time path planning in dynamic environments.

Grid-based algorithms provide a straightforward approach to path planning in environments represented as grids. They can handle discrete movements and can be implemented efficiently. However, these algorithms may face challenges with high-dimensional spaces or continuous motion. In such cases, other algorithms like sampling-based methods (e.g., RRT) or optimization-based approaches may be more suitable.

3.4.3.3 Occupancy grid

An occupancy grid is a representation of a physical environment used in robotics and autonomous systems for mapping and localization. It divides the environment into a grid of cells and assigns a binary value (occupied or free) to each cell based on the presence or absence of obstacles.

Here's an overview of occupancy grids:

- Grid Representation: An occupancy grid divides the physical environment into a regular grid of cells. The grid can be 2D for planar environments or 3D for volumetric representations. The size of the cells can vary depending on the desired resolution and the characteristics of the environment.

- Binary Occupancy Values: Each cell in the grid is assigned a binary value representing its occupancy state. Typically, the value can be either occupied (1) or free (0). This binary representation simplifies the representation of obstacles in the environment.

- Sensing and Updating: Occupancy grids are updated based on sensor measurements, such as range sensors, lidar, or cameras. When a sensor detects an object or obstacle, the corresponding cells in the grid are marked as occupied. Free cells can be inferred by assuming that the areas surrounding the occupied cells are free from obstacles.

- Map Fusion: In many scenarios, occupancy grids are built incrementally as the robot explores the environment or receives new sensor measurements. These partial grids can be fused together to form a consistent and updated representation of the entire environment.

- Localization and Path Planning: Occupancy grids are used for various robotic tasks, including localization and path planning. By comparing sensor measurements with the occupancy grid, robots can estimate their position and orientation accurately. Path planning algorithms can utilize occupancy grids to find collision-free paths through the environment.

- Probabilistic Representation: Occupancy grids can also be extended to incorporate probabilistic representations. Instead of using binary values, each cell can store a probability value that represents the likelihood of occupancy. This allows for more nuanced and uncertain representations of the environment.

Occupancy grids are commonly used in robotics for mapping, localization, and navigation tasks. They provide a convenient and intuitive representation of the environment, allowing robots to reason about obstacles and plan their movements efficiently. However, occupancy grids have limitations in handling dynamic or partially observable environments, as they assume the environment is static and can be fully observed by the sensors.

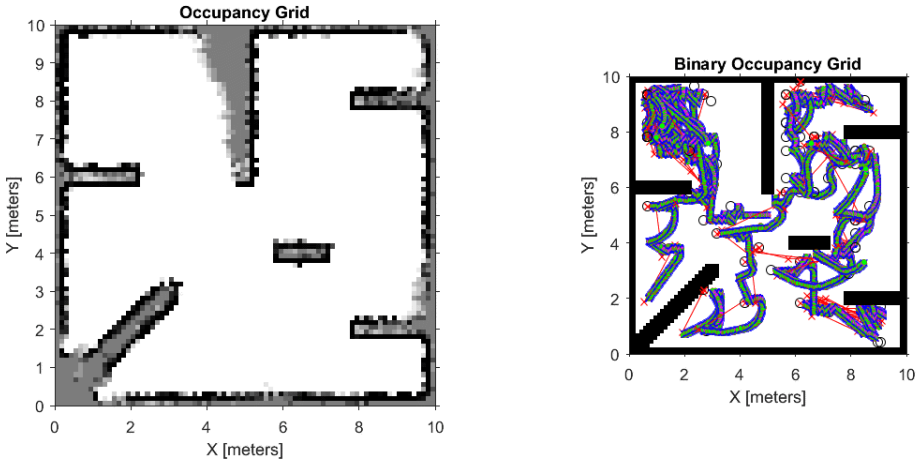


Fig 3.14: Occupancy Grid Map (normal and binary)

3.4.3.4 Geometric algorithms

Geometric algorithms are algorithms that deal with computational problems involving geometric objects, such as points, lines, curves, polygons, and higher-dimensional shapes. These algorithms are used in various fields, including computer graphics, computer vision, robotics, computational geometry, and simulation.

Here are some common types of geometric algorithms:

- Convex Hull: Convex hull algorithms determine the smallest convex polygon that encloses a set of points in the plane or higher-dimensional spaces. Examples of convex hull algorithms include Graham's scan, Jarvis March, and Quickhull. Convex hulls have applications in collision detection, pattern recognition, and shape analysis.

- Line Intersection: Line intersection algorithms determine whether or where two lines intersect in 2D or higher-dimensional spaces. Examples include the Bentley-Ottmann algorithm for line segment intersection and the sweep line algorithm. Line intersection algorithms are used in computer graphics, computational geometry, and CAD systems.

- Triangulation: Triangulation algorithms partition a set of points into triangles, forming a triangulated mesh. Delaunay triangulation and Ear Clipping are popular algorithms used for triangulation. Triangulation has applications in mesh generation, surface reconstruction, and terrain modeling.

- Voronoi Diagram: Voronoi diagrams partition a space into regions based on proximity to a set of input points. Each region consists of all points that are closer to a particular input point than any other. Fortune's algorithm and the incremental algorithm are commonly used for constructing Voronoi diagrams. Voronoi diagrams have applications in computer graphics, spatial analysis, and nearest-neighbor search.

- Range Searching: Range searching algorithms determine which geometric objects fall within a specified region of interest, such as points inside a polygon or line segments intersecting a rectangle. Range trees, quad trees, and kd-trees are data structures commonly used for range searching. Range searching is used in spatial databases, GIS systems, and collision detection.

- Polygon Clipping: Polygon clipping algorithms compute the intersection or union of two or more polygons. The Sutherland-Hodgman algorithm and the Weiler-Atherton algorithm are well-known polygon clipping algorithms. Polygon clipping is used in computer graphics, computational geometry, and image processing.

These are just a few examples of geometric algorithms, and there are many other algorithms addressing different geometric problems. Geometric algorithms play a crucial role in solving complex computational problems involving geometric data, enabling efficient processing and analysis of geometric objects in various applications.

3.4.3.5 Visibility graph

The visibility graph is a fundamental geometric algorithm used in computational geometry and robotics for path planning and visibility analysis. It helps determine the visibility relationships between a set of points or objects in a given environment.

Here's an explanation of the visibility graph algorithm:

Input:

- A set of points or objects in a 2D or 3D space.
- Optionally, obstacles or walls block visibility between points.

Creating the Visibility Graph: Connect every pair of points with a line segment if no obstacles or walls are obstructing the direct line of sight between them. These line segments form the edges of the visibility graph.

Visibility Analysis: The visibility graph provides information about the visible connections between points. It represents all possible unobstructed paths between the points.

Path Planning: Once the visibility graph is constructed, it can be used for path planning. The shortest path between two points can be determined by applying graph search algorithms such as Dijkstra's algorithm or A* algorithm on the visibility graph.

Applications:

- The visibility graph algorithm has numerous applications, including:
- Path planning for robots or vehicles in environments with obstacles.
- Visibility analysis for surveillance and sensor placement.
- Line-of-sight analysis for wireless communication and network optimization.
- Motion planning and collision avoidance in virtual environments and video games.

The visibility graph algorithm simplifies the path planning problem by reducing the search space to a graph representation. By considering only the direct line-of-sight connections between points, it enables efficient and effective path planning while avoiding obstacles or areas that are not directly visible.

However, constructing a visibility graph can be computationally expensive, especially in complex environments with many obstacles or a large number of points. As a result, alternative algorithms, such as sampling-based algorithms (e.g., Rapidly-exploring Random Trees), are often used in practical scenarios where the visibility graph construction may be prohibitive.

Overall, the visibility graph algorithm provides a powerful tool for path planning and visibility analysis, enabling efficient and effective navigation and exploration in geometric environments.

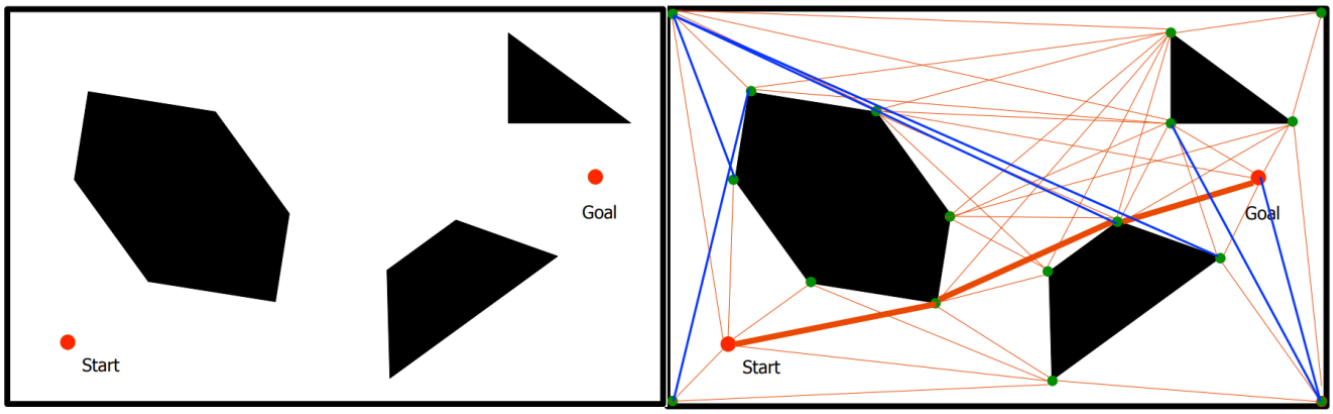


Fig 3.15: : Visibility graph map

3.4.3.6 Cell decomposition

Cell decomposition is a technique used in computational geometry and robotics for decomposing a complex space into simpler cells to facilitate path planning and coverage analysis. It involves partitioning the space into discrete regions, called cells, based on certain criteria or constraints. Each cell represents a sub region of the original space, and together, they cover the entire space.

Here's an explanation of cell decomposition:

Input:

- A 2D or 3D space that needs to be decomposed.
- Optionally, obstacles or walls need to be taken into account during decomposition.

Decomposition Criteria:

- Select a criteria or constraint to guide the decomposition process. These criteria can be based on connectivity, visibility, or other geometric properties. Common types of cell decomposition include:
- -Voronoi Diagram: Partition the space based on proximity to a set of input points, resulting in cells that contain all points closer to a particular point than any other.
- Quad tree or Octree: Divide the space recursively into equal-sized cells, where each cell is further divided into four or eight sub cells.
- Grid-based Decomposition: Divide the space into a regular grid of cells, where each cell represents a fixed-sized region.
- Skeletonization: Identify the medial axis or skeleton of the space, representing the centerlines or critical points, and divide the space based on this structure.

Cell Generation:

Apply the chosen decomposition algorithm to generate the cells that cover the space. The specific method will depend on the chosen criteria or constraint.

Path Planning or Coverage Analysis:

Once the space is decomposed into cells, path planning or coverage analysis can be performed more efficiently within each cell. Algorithms like A* or Dijkstra's algorithm can be applied within the cells to find paths or explore the sub regions individually.

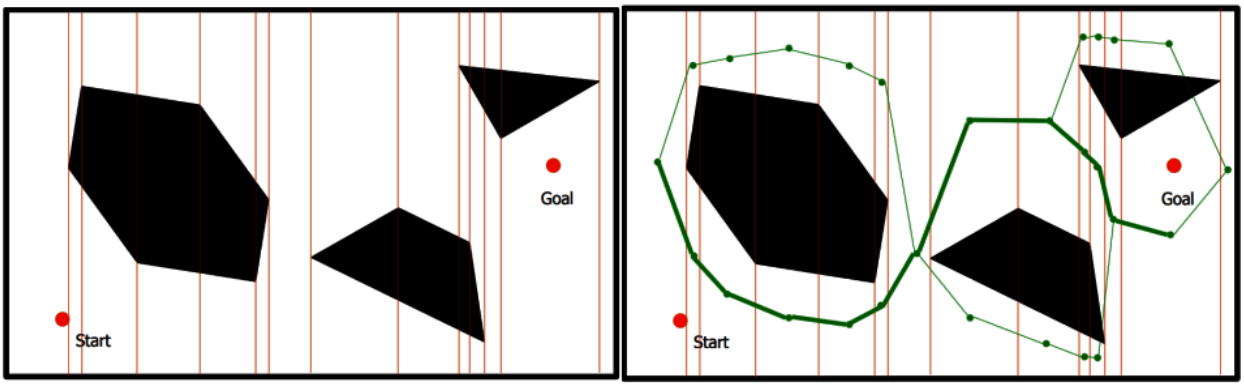


Fig 3.16: Cell decomposition graph map

Applications:

- Cell decomposition has various applications, including:
- Path planning for robots or vehicles in complex environments.
- Coverage analysis and deployment of sensors or resources in an area.
- Decomposition of virtual environments for rendering and collision detection in computer graphics.
- Spatial analysis and partitioning of geographic regions in GIS applications.

Cell decomposition is a versatile technique that allows for the efficient analysis and navigation of complex spaces. It simplifies the problem by breaking down the space into smaller, manageable cells, enabling more effective path planning and analysis within each cell. The choice of the decomposition criteria and algorithm depends on the specific problem and the properties of the space being decomposed.

3.4.3.7 Probabilistic road map (PRM)

Probabilistic Road Maps (PRM) are a popular algorithmic technique used in robotics for motion planning in complex environments. It is a sampling-based approach that builds a roadmap of the environment by sampling valid configurations and connecting them to form a graph representation. PRM is particularly effective in high-dimensional and continuous state spaces.

Here's an explanation of the Probabilistic Road Maps (PRM) algorithm:

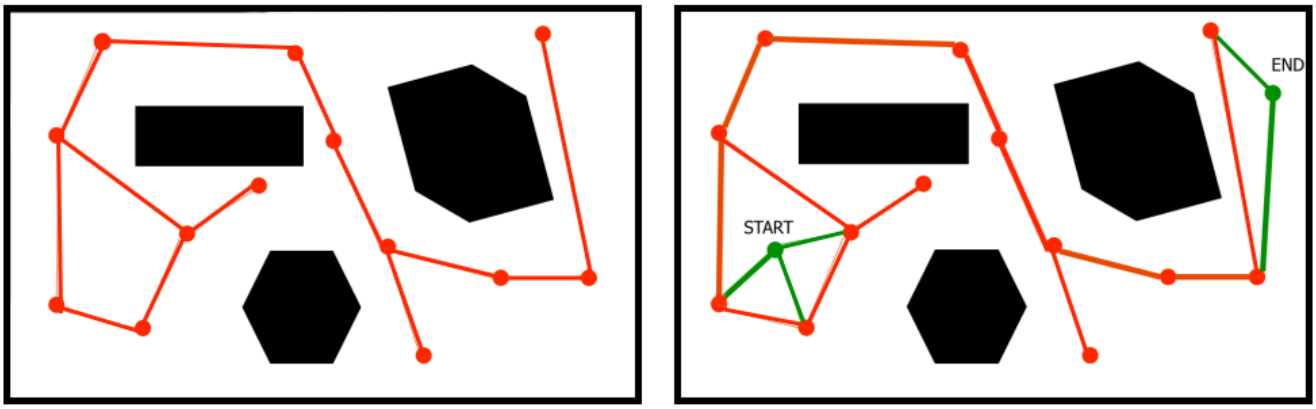


Fig 3.17: Probabilistic road map

Input:

- A representation of the environment, including obstacles or walls.
- A start configuration and a goal configuration.

Sampling:

- Randomly sample configurations from the state space, ensuring that the sampled configurations are valid and collision-free. The sampling can be uniform or biased towards the areas of interest in the environment.

Connection:

- For each sampled configuration, determine the neighboring configurations that are within a certain distance or satisfy certain connectivity criteria. Connect these configurations by adding edges to the roadmap graph. The connectivity can be determined based on proximity or other geometric constraints.

Collision Checking:

- Perform collision checks between the configurations and the obstacles in the environment. Discard any connections that result in collisions. This step ensures that the roadmap only contains valid and collision-free paths.

Roadmap Construction:

- Build the roadmap graph by repeating the sampling and connection steps until a sufficient number of valid configurations and connections are added. The resulting roadmap represents a connected graph that captures the connectivity of the environment.

Path Planning:

- Use a path planning algorithm, such as Dijkstra's algorithm or A* algorithm, on the constructed roadmap to find a path from the start configuration to the goal configuration. The roadmap provides a connected and discrete representation of the state space, making the path planning process more efficient.

Refinement:

- Once an initial path is found, a refinement step can be applied to improve the path's quality. This step can involve local optimization techniques or fine-tuning the path to account for specific constraints or objectives.

Execution:

- Execute the planned path by controlling the robot or system to follow the path generated by the PRM algorithm.
- Probabilistic Road Maps (PRM) provide a flexible and efficient approach to motion planning in complex environments. By sampling valid configurations and connecting them to form a roadmap, PRM captures the connectivity of the state space and enables the exploration of feasible paths. PRM is particularly useful in high-dimensional spaces where exhaustive exploration is not feasible.

3.4.3.8 Rapidly exploring Random Trees (RRT)

Rapidly exploring Random Trees (RRT) is a popular algorithmic technique used for motion planning in robotics and autonomous systems. RRT efficiently explores the configuration space by incrementally building a tree-like graph from randomly sampled configurations. It is particularly well-suited for high-dimensional and complex environments.

Here's an explanation of the rapidly exploring Random Trees (RRT) algorithm:

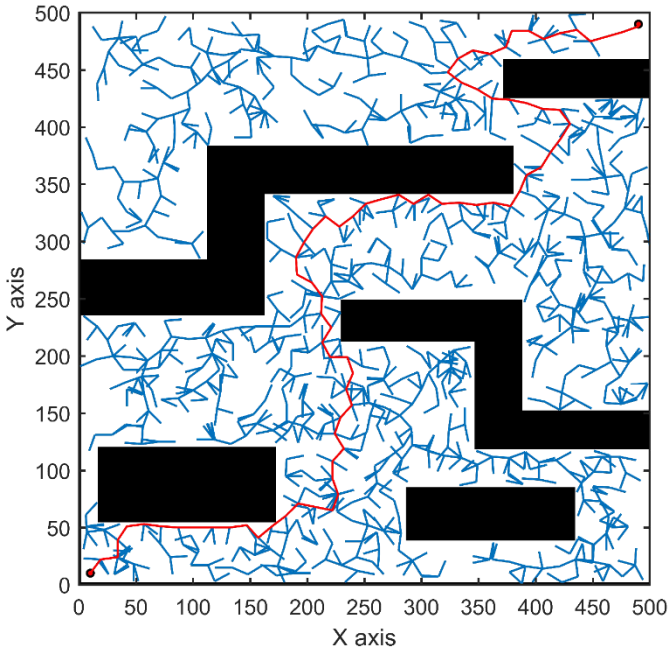


Fig 3.18: Random tree map

Input:

- A representation of the environment, including obstacles or walls.
- A start configuration and a goal configuration.

Tree Initialization:

- Create an empty tree with the start configuration as the root node.

Sampling:

- Randomly sample a configuration from the state space. This sampled configuration can be within the entire state space or biased towards the unexplored areas of the environment.

Nearest Neighbor:

- Find the nearest configuration in the tree to the sampled configuration. This step helps in growing the tree towards unexplored areas and expanding the search space.

Expansion:

- Extend the tree by adding a new node that connects the nearest neighbor to the sampled configuration. Ensure that the connection is collision-free by performing collision checks with the obstacles in the environment.

Repeat:

- Repeat the sampling, nearest neighbor, and expansion steps until a specified termination condition is met. This condition can be a maximum number of iterations, reaching the goal configuration, or other stopping criteria.

Path Construction:

- Once the tree has been built, construct a path from the start configuration to the goal configuration. This can be done by finding the path connecting the goal configuration to the nearest neighbor in the tree.

Refinement:

- Optionally, refine the path by applying optimization techniques or smoothing algorithms to improve the path's quality. This step helps in generating a more feasible and efficient path.

Execution:

- Execute the planned path by controlling the robot or system to follow the path generated by the RRT algorithm.

Rapidly exploring Random Trees (RRT) provides an efficient and probabilistically complete approach to motion planning. By randomly sampling configurations and incrementally expanding the tree towards unexplored areas, RRT explores the state space rapidly. It is particularly effective in high-dimensional spaces where exhaustive exploration is not feasible.

RRT variants, such as RRT-Connect and RRT*, introduce additional techniques to enhance the algorithm's performance and optimality. These variants improve the connection between the start and goal configurations and provide more optimal paths in complex environments

Overall, RRT is a widely used algorithm for motion planning in robotics due to its efficiency, scalability, and ability to handle complex environments. It has applications in autonomous navigation, robot manipulation, and other areas requiring efficient path planning in high-dimensional spaces.

3.4.4 Motion planning

Motion planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement.

Motion planning is a fundamental problem in robotics and autonomous systems that involves establishing the exact actions a robot must execute to follow a predetermined path and reach its goal.

Here's an overview of the motion planning process:

Input:

- Environment representation: A representation of the environment, including the geometry of obstacles or constraints.

- Start and goal configurations: The initial and desired final states or positions of the robot or agent.
- Constraints and objectives: Any additional constraints or objectives, such as avoiding certain areas or minimizing travel time.

State Space:

- Define the state space, which represents the possible configurations or positions of the robot. The state space can be continuous or discrete, depending on the specific problem. It is often defined by the robot's degrees of freedom and the constraints imposed by the environment.

Motion Validity:

- Determine the validity of a motion or path by performing collision checks between the robot and the obstacles in the environment. This involves evaluating whether a proposed path violates any constraints or collides with obstacles.

Search or Sampling:

- Apply a search-based or sampling-based algorithm to explore the state space and find a feasible path from the start configuration to the goal configuration. Search-based algorithms (e.g., A*, Dijkstra's algorithm) systematically explore the state space using graph search techniques, while sampling-based algorithms (e.g., RRT, PRM) randomly sample configurations and build a roadmap or tree structure to connect them.

Path Optimization:

- Once a path is found, it can be further optimized or refined to improve its quality or satisfy additional objectives. Optimization techniques may include smoothing the path, minimizing path length or travel time, or considering dynamic constraints.

Execution:

- Execute the planned path by controlling the robot or agent to follow the planned trajectory. This typically involves generating control commands or executing motion primitives to navigate the robot through the planned path.

Motion planning is a critical component in various robotic applications, such as autonomous navigation, robot manipulation, unmanned aerial vehicles (UAVs), self-driving cars, and more. It involves a balance between finding feasible paths, avoiding obstacles, optimizing for objectives, and accounting for the robot's constraints and capabilities. Various algorithms and techniques have been developed to address the motion planning problem, and the choice of algorithm depends on the specific requirements of the application and the characteristics of the environment.

3.5 ROBOT MOTION CONTROL

3.5.1 PID Controller

What is the PID controller?

The term PID stands for proportional integral derivative and it is one kind of device used to control different process variables like pressure, flow, temperature, and speed in industrial applications. In this controller, a control loop feedback device is used to regulate all the process variables.

This type of control is used to drive a system in the direction of an objective location otherwise level. It is almost everywhere for temperature control and used in scientific processes, automation & myriad chemical. In this controller, closed-loop feedback is used to maintain the real output from a method like close to the objective otherwise output at the fixed point if possible. In this article, the PID controller design with control modes used in them like P, I & D is discussed.

PID systems automatically apply accurate and responsive correction to the control function. An everyday example is a car's cruise control, where going up a hill will reduce speed if constant engine power is applied. The controller's PID algorithm restores the measured speed to the required speed with minimal delay and overshoot by increasing the motor's power output in a controlled manner.

The first theoretical analysis and practical application of PID was in the field of automatic guidance systems for ships, which were developed from the early 1920s onwards. Then it was used for automatic process control in the manufacturing industry, where it was widely implemented in pneumatic and then electronic control devices. Today the PID concept is used globally in applications that require precise and optimized automation control.

PID controller block diagram:

A closed-loop system like a PID controller includes a feedback control system. This system evaluates the feedback variable using a fixed point to generate an error signal. Based on that, it alters the system output. This procedure will continue till the error reaches Zero otherwise the value of the feedback variable becomes equivalent to a fixed point.

This controller provides good results as compared with the ON/OFF type controller. In the ON/OFF type controller, simply two conditions are obtainable to manage the system. Once the process value is lower than the fixed point, then it will turn ON. Similarly, it will turn OFF once the value is higher than a fixed value. The output is not stable in this kind of controller and it will swing frequently in the region of the fixed point. However, this controller is more steady & accurate as compared to the ON/OFF type controller.

Theory of PID controller:

With the use of a low-cost simple ON-OFF controller, only two control states are possible, like fully ON or fully OFF. It is used for a limited control application where these two control states are enough for the control objective. However, the oscillating nature of this control limits its usage and hence it is being replaced by PID controllers.

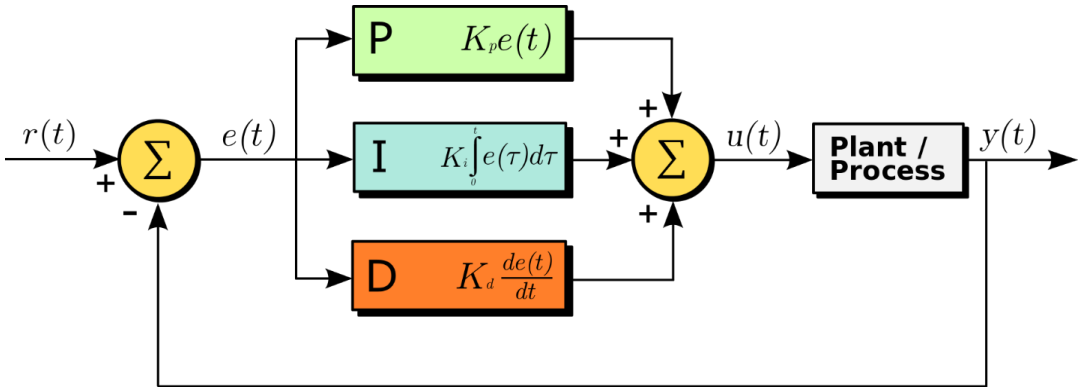


Fig 3.19: PID Block diagram

PID controller maintains the output such that there is zero error between the process variable and set point (desired output) by closed-loop operations.

PID uses three basic control behaviors that are explained below:

P- Controller

The obvious method is proportional control: the armature current is set in proportion to the fault current. However, this method fails if, say, the arm has to lift different weights: a larger weight needs more force applied to the same fault on the downside, but a smaller force if the fault is lower in the upward direction. This is where the integral and derivative terms come into play.

I-Controller

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

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

D-Controller

The I-controller can’t predict the future behavior of error. So it reacts normally once the set point is changed. D-controller overcomes this problem by anticipating the future behavior of the error. Its output depends on the rate of change of error with respect to time, multiplied by the derivative constant. It gives the kick start for the output thereby increasing system response. It improves the stability of the system by compensating for phase lag caused by the I-controller. Increasing the derivative gain increases the speed of response.

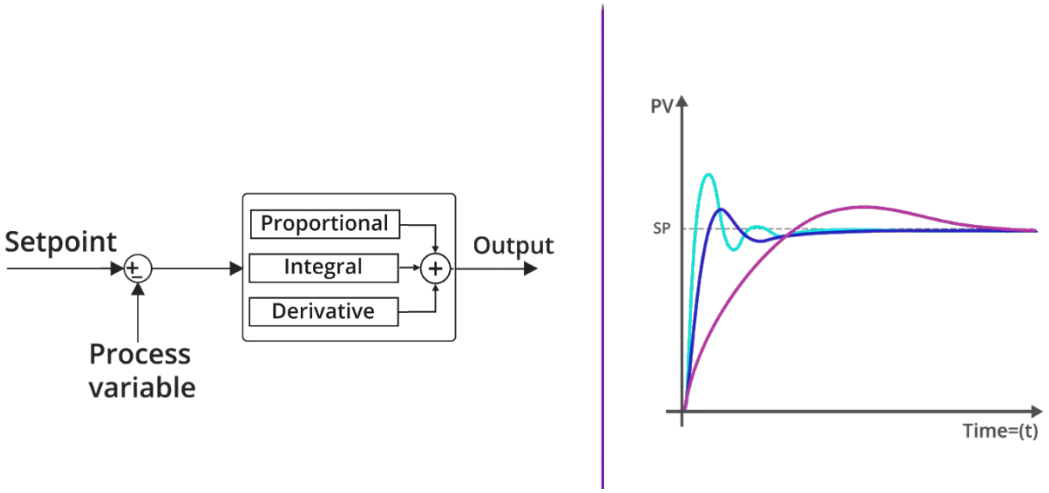


Fig 3.20: PID Block diagram without feedback

Feedback control loop

Let’s look at how a PID Controller fits into a feedback control loop. The Controller is responsible for ensuring that the Process remains as close to the desired value as possible regardless of various disruptions.

The controller compares the Transmitter Process Variable (PV) signal and the Set point.

Based on that comparison, the controller produces an output signal to operate the Final Control Element. This PID Controller output is capable of operating the Final Control Element over its entire 100% range.

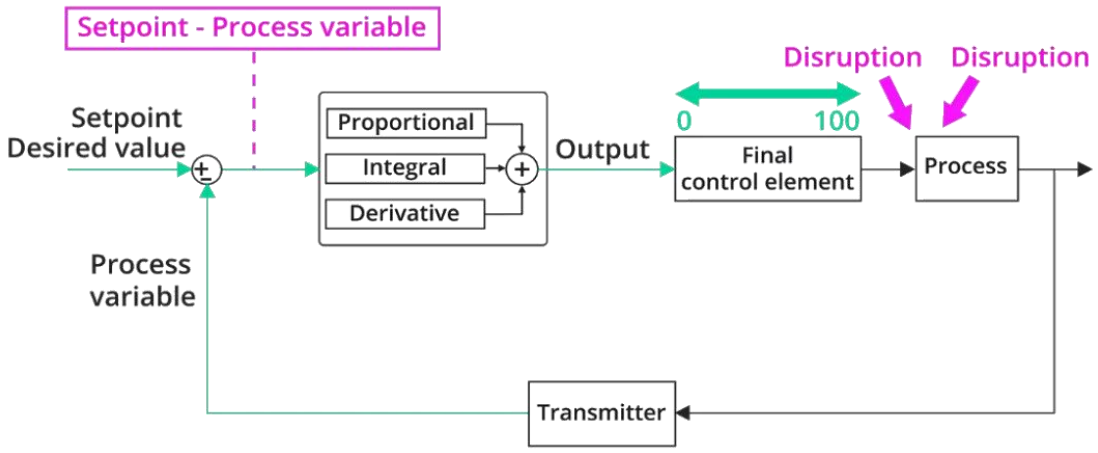


Fig 3.21: PID Block diagram with feedback

3.5.2 PID Tuning

After we have explained the PID and how it works, we will explain how we implemented PID as a controller in the robot and what software were used in it, for example, but not limited to, such as ROS.

ROS Control is a set of packages that includes a controller interface, controller manager, transmissions, hardware interfaces, and control toolbox. All these packages together will allow you to interact and control the joint actuators of the robot.

Ros control will take the joint state data and an input set point (goal) as input from the user and sends the appropriate commands to the actuators as an output. In order to achieve the provided set point (goal), it uses a generic control loop feedback mechanism, typically a PID controller, to control the output. This output is passed to the robot through the hardware interface.

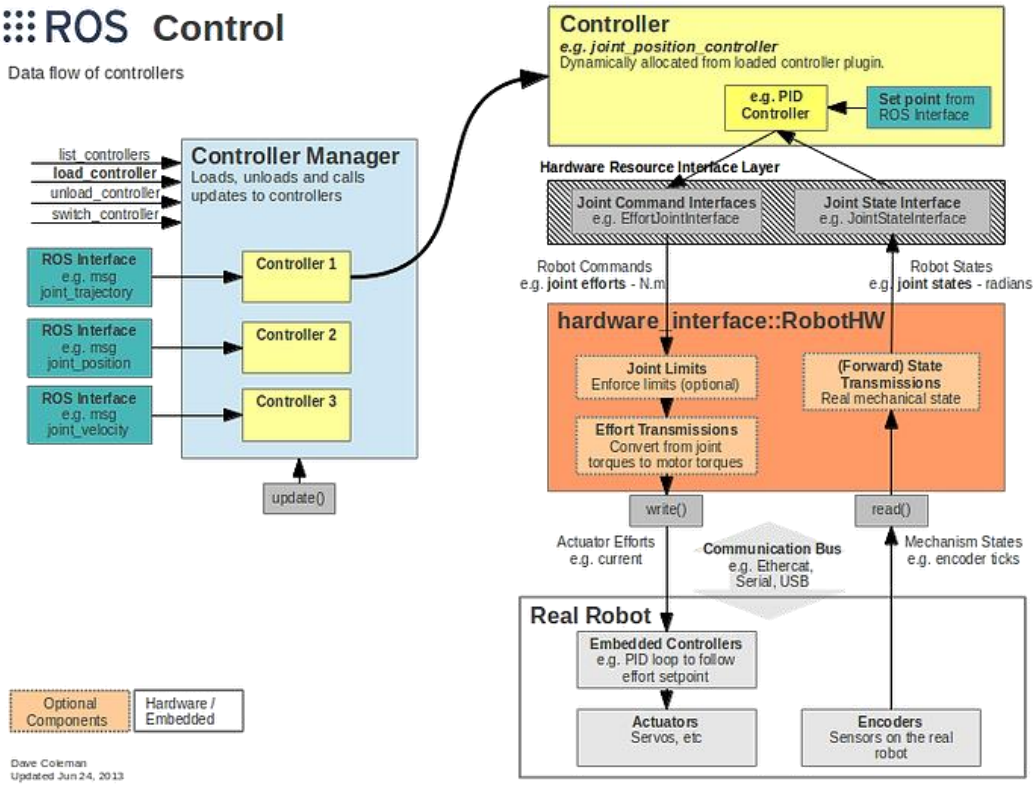


Fig 3.22: Ros Control Package

3.6 AUTONOMOUS NAVIGATION

3.6.1 Introduction

Autonomous navigation is defined as the ability of a robot to plan its path and execute its plan without human intervention. Mobile robots with autonomous capabilities allow the robots to perform various tasks in structured or unstructured environments without continuous human intervention. They have the ability to:

1. Gain information about the environment.

2. Work for an extended period without human intervention.

3. Move either all or part of itself throughout its operating environment without human assistance.

4. Avoid dangerous situations such as collision, obstruction, and any harmful acts to humans, itself, or the environment.

To navigate autonomously, robots have to deal with three working principles: mapping, localization, and planning. Mapping is a process of collecting and connecting information from sensors into a map. Robots execute mapping algorithms to translate the surrounding environment into data that can be read, usually a 2D geometric representation.

Localization is where the robots use the sensors to estimate their position relative to the map. Robots try to locate their location within the map using the sensors by utilizing their sensors and laser scanners to receive geometric data and odometry data to move around the environment, determining their position relative to the map at the same time. Once localization is established, the robot can start planning its path to achieve the goal location. The map generates functions as an eye for the robot. It represents entities such as known space, obstacles, robot position, and unexplored space. This process is known as global planner. Then, a local planner translates this path into the movement of the robot.

3.6.2 Local and Global planner

Global Planner. As said before, the global planner requires a map of the environment to calculate the best route. Depending on the analysis of the map, some methods are based on Roadmaps like Silhouette proposed by Canny in 1987 or Voronoi [6] in 2007. Some of them solve the problem by assigning a value to each region of the roadmap in order to find the path with minimum cost. Some examples are the Dijkstra algorithm, Best First, and another approach by dividing the map into small regions (cells) called cell decomposition as mentioned earlier. A similar approach by using potential fields is described in, the most extended algorithm used a few years ago rapidly exploring random trees or the new approach based on neural networks. Some solutions combine the aforementioned algorithms improving the outcome at the cost of high computational power.

Local Planner. In order to transform the global path into suitable waypoints, the local planner creates new waypoints taking into consideration the dynamic obstacles and the vehicle constraints. So, to recalculate the path at a specific rate, the map is reduced to the surroundings of the vehicle and is updated as the vehicle is

moving around. It is not possible to use the whole map because the sensors are unable to update the map in all regions and a large number of cells would raise the computational cost. Therefore, with the updated local map and the global waypoints, the local planning generates avoidance strategies for dynamic obstacles and trajectory. Tries to match the trajectory as much as possible to the provided waypoints from the global planner. Different approaches are based on trajectory generation. All of them create intermediate waypoints following the generated

Chapter Four

4 IMR’s Implementation

4.1 INTRODUCTION

In this section, we will discuss the implementation of our IMR requirements that we defined earlier in this book.

Our project implementation is divided into two main parts:

The first part is the hardware side: the hardware side consists of the design of the mobile robot and the robot arm, the other part of that hardware is the motors and circuitry of the project that was designed and implemented.

The second part is the software side: software side consists of implementing and installing ROS on the Linux Ubuntu system and using Ros to configure and build our robot system from controlling the motors to navigating autonomously.

4.2 HARDWARE IMPLEMENTATION

4.2.1 Mechanical design

4.2.1.1 Mobile robot

Our robot chassis is made of wood and then refinished well to be painted and give it a good look, we choose wood because it's easy to implement and not very costly compared to other materials.

Our design of wood is very simple as we choose it to be a classic design of a car and used standard wheels to make the robot simple in its navigation control and movement.

The figures below illustrate the phases of our design that took place throughout the year:

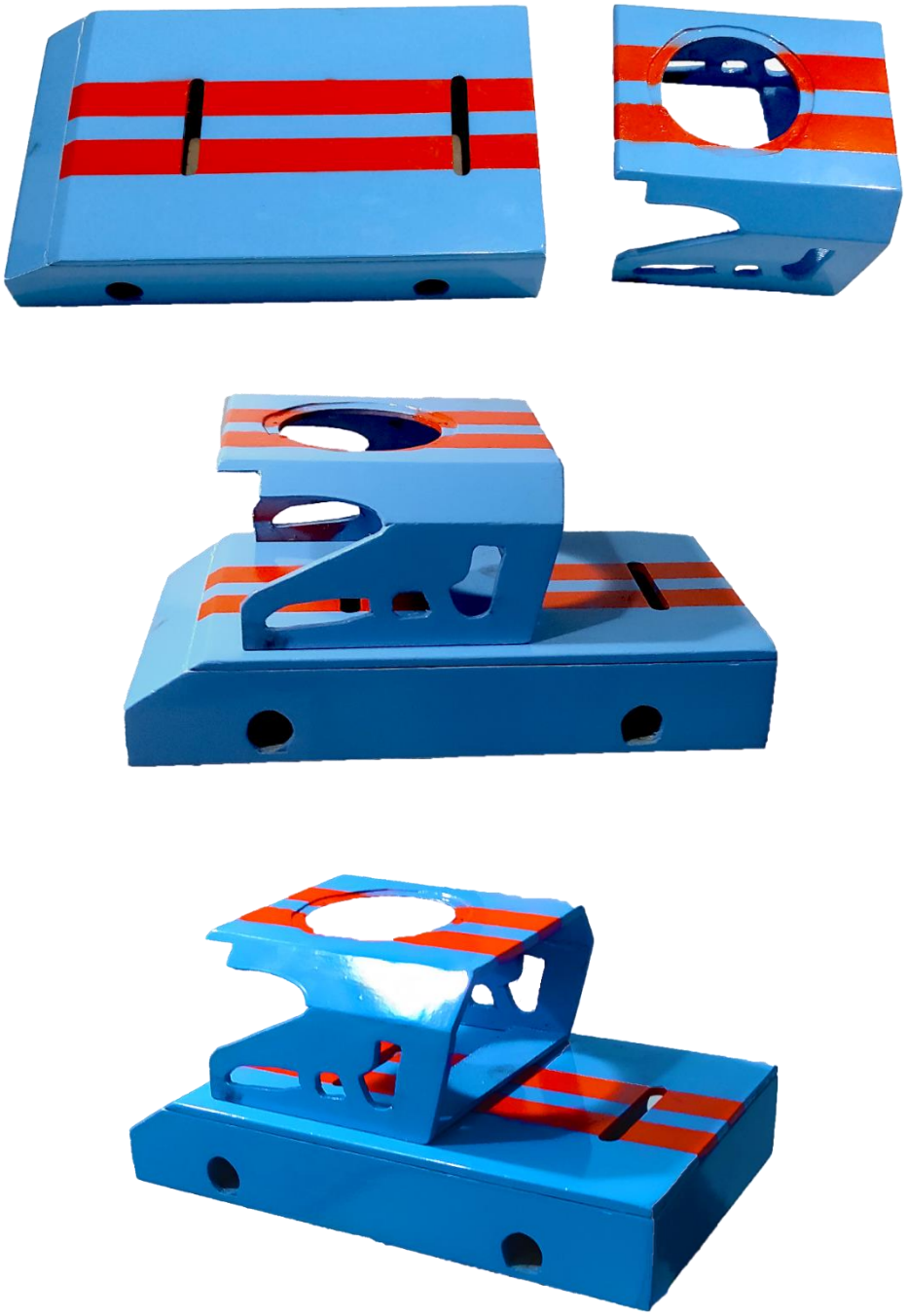


Fig 4.1: Mobile robot chassis

4.2.1.2 Robot Arm

For the robot arm, we decided to 3D print it using PLA+ material as it is very reasonably priced and its quality is very good and stable, the arm was printed in different parts and then assembled. See Fig.4.2:

Fig 4.2: Robot arm parts before assembling

Fig 4.3: Robot arm parts after assembling

4.2.2 Project Assembled

Here is our robot after assembling all parts of the mobile robot and arm:

Fig 4.4: IMR after assembling

4.2.3 PCB Design

This our PCB design for IMR robot its very basic circuit to make the project compact as possible .See Fig.4.4.

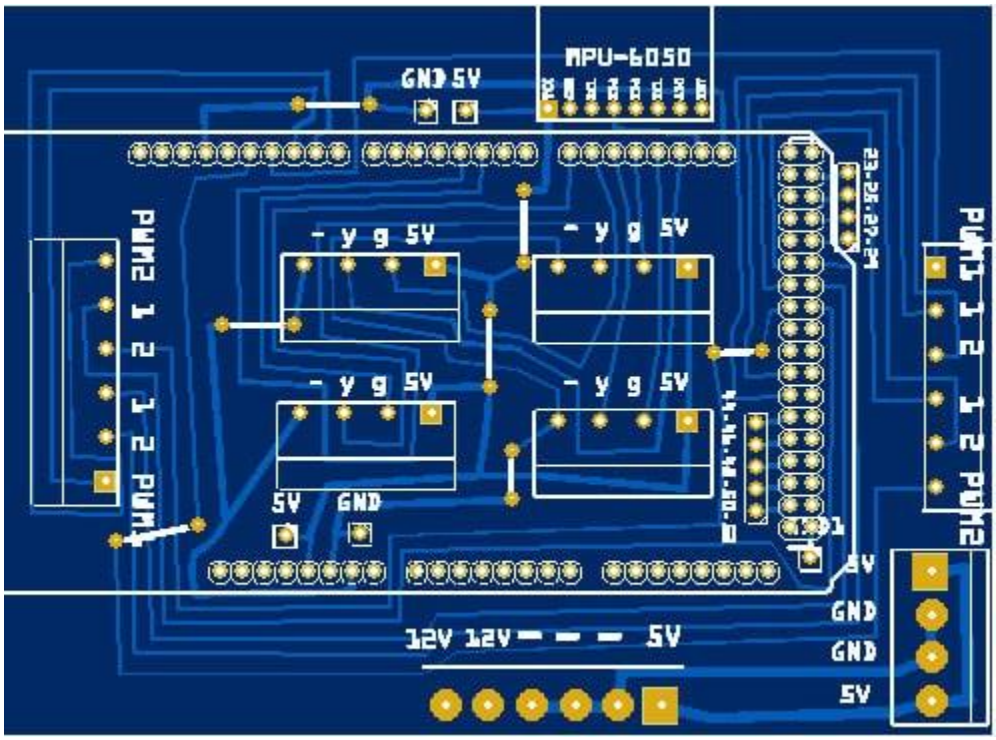


Fig 4.5: IMR PCB

4.3 SOFTWARE IMPLEMENTATION

4.3.1 Installing Ubuntu 18.04

The Ubuntu desktop is easy to use, easy to install and includes everything you need to run your organization, school, home or enterprise. It's also open source, secure, accessible and free to download.

Here the steps to install Ubuntu:

- After download Ubuntu from official website [http://releases.ubuntu.com/18.04/ubuntu-18.04.6-desktop-amd64.iso] and make it a bootable file on USB flash memory
  Open a bootable file from USB.

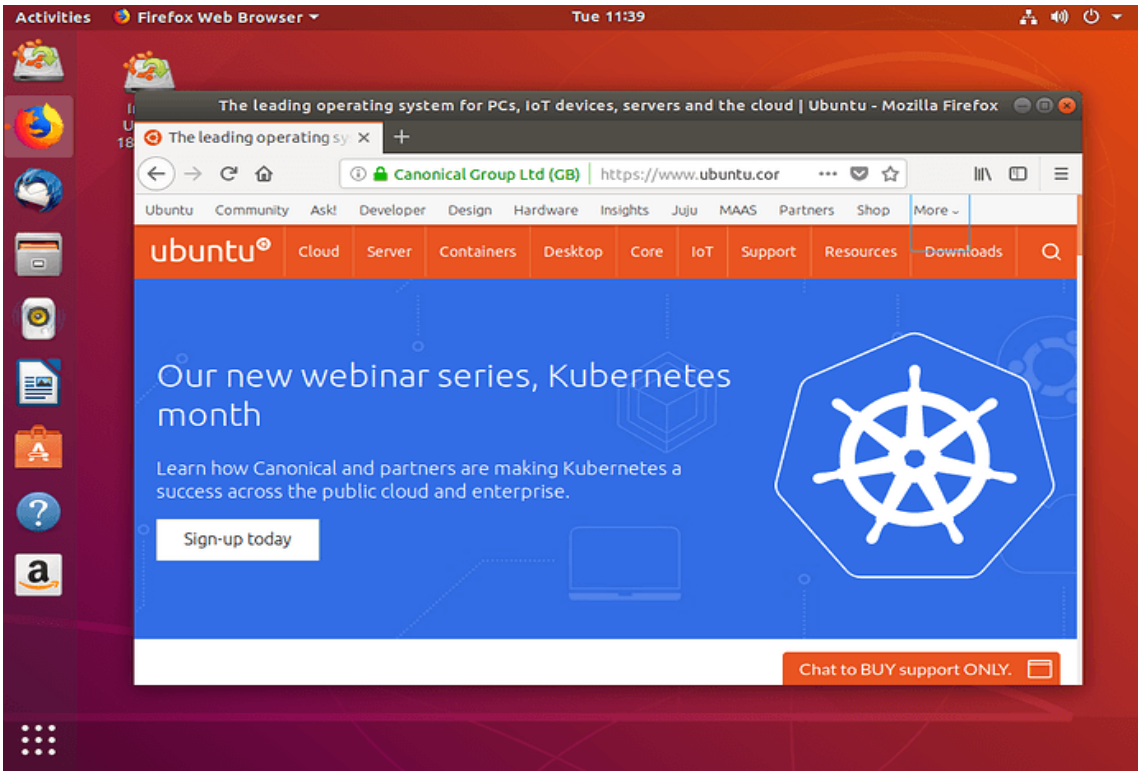
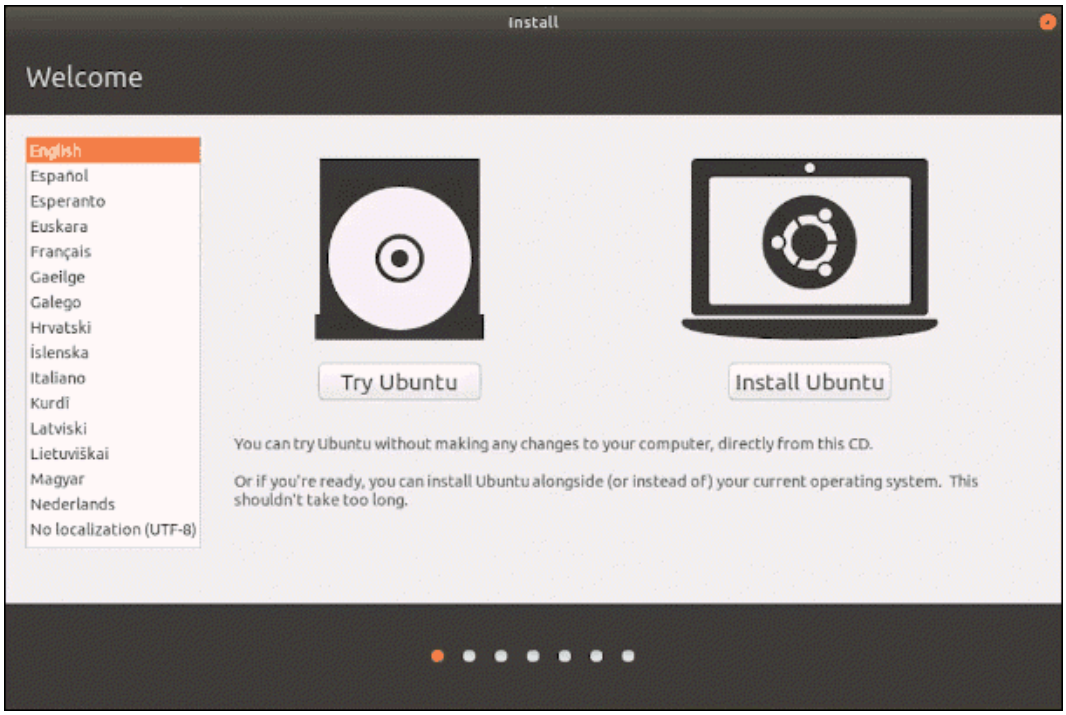
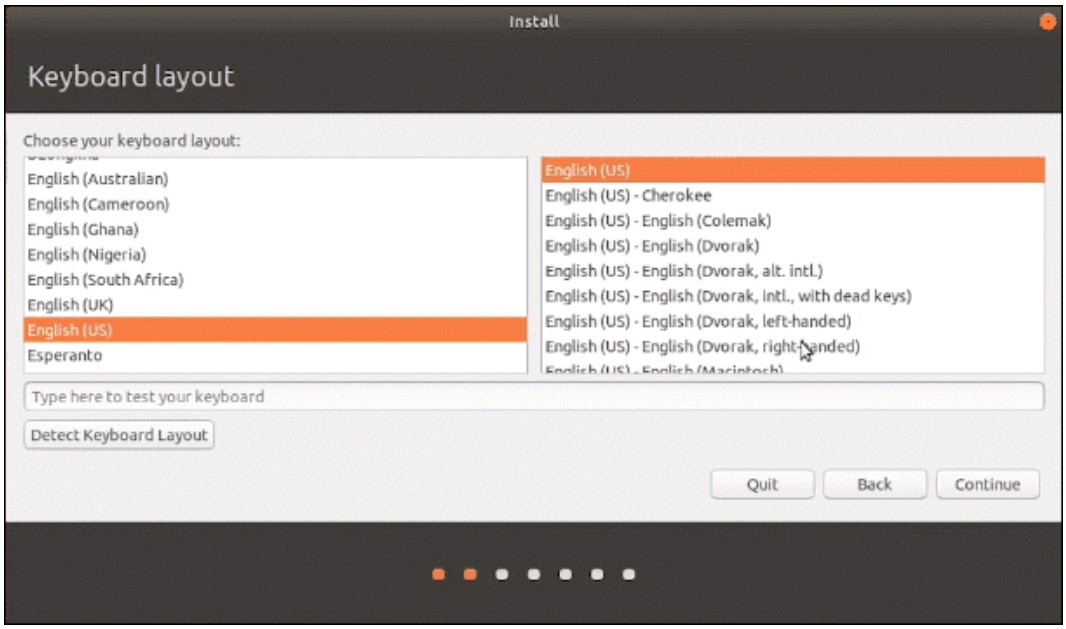


Fig 4.6: Installing Ubuntu

- Click on install Ubuntu

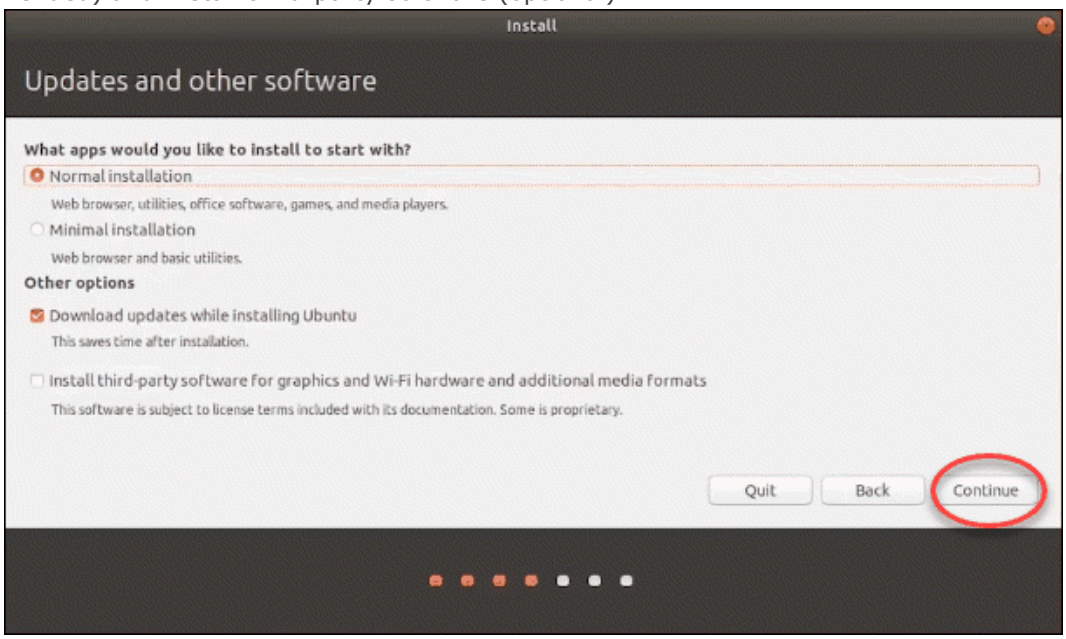


- Choose language and keyboard layout and click continue



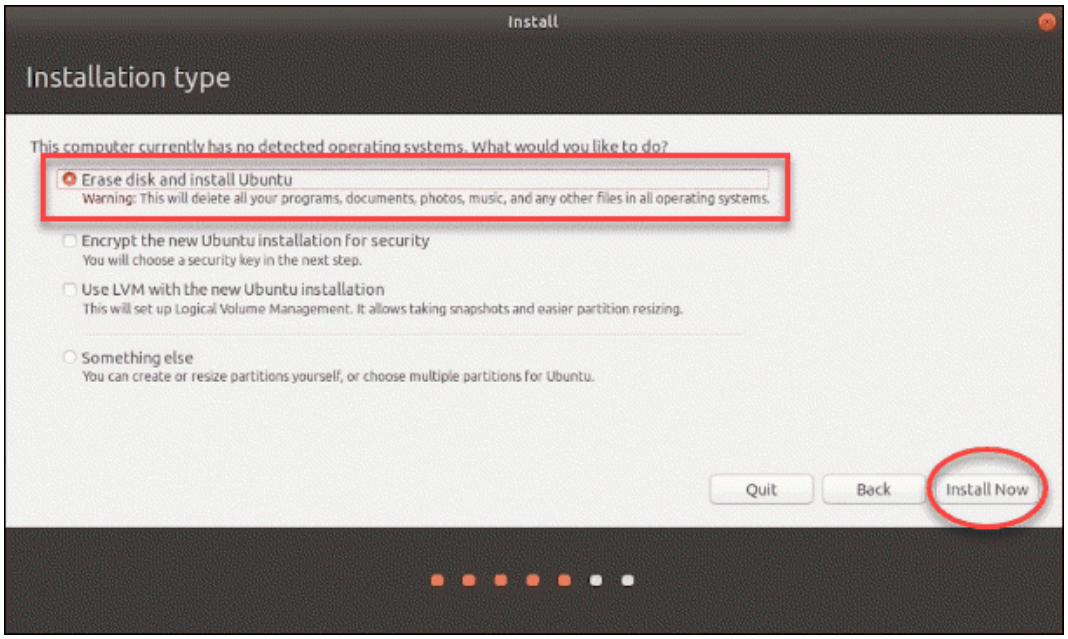
Next, select whether you want the Normal or Minimal installation. The difference is the number of apps it includes.

- Also, you can choose if you want the system to download updates while installing Ubuntu (which is recommended) and Install third-party software (optional).

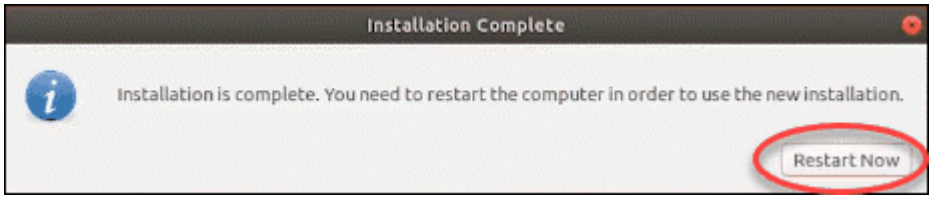


- Once you have decided, click on Continue.

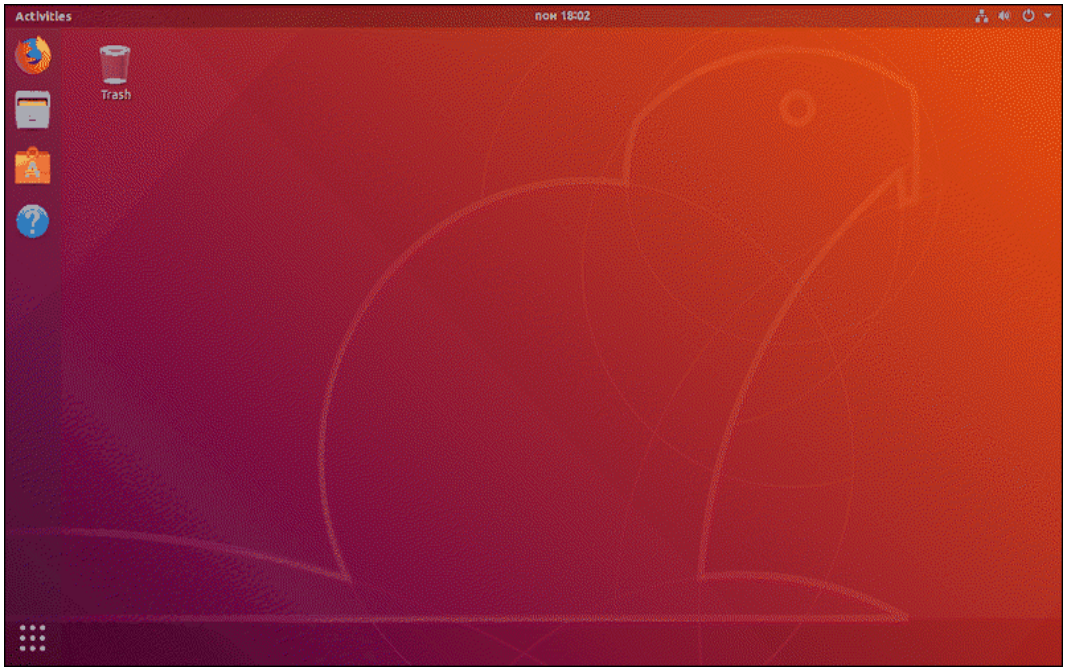
- Click on any method you want to write Ubuntu file on desk
  we choose Erase disk and install Ubuntu and click install now



- Once the installation is complete, the following message will appear. Take out the USB and Restart the system, as suggested



- Congratulations you have successfully installed Ubuntu 18.04



4.3.2 Installing Ros melodic

Open terminal and type

```
$ Sudo apt install update && sudo apt install upgrade -y
```

Setup your sources. List

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

Set up your keys

```
$ Sudo apt install curl && curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

Installation

```
$ sudo apt update
```

Desktop-Full Install

```
$ sudo apt install ros-melodic-desktop-full
```

Environment setup

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

$ source ~/.bashrc

$ source /opt/ros/melodic/setup.bash
```

Dependencies for building packages

```
$ Sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

Initialize rosdep

```
$ sudo apt install python-rosdep

$ sudo rosdep init

$ rosdep update
```

To make sure the ros is installed correctly we run ros core to check

```
$ roscore
```

With this, we have completed the process of installing the ROS robot operating system on the Ubuntu and to.

4.4 MAPPING

4.4.1 Creating map

Run bringup.launch:

```
$ roslaunch linorobot bringup.launch
```

Run slam.launch:

```
$ roslaunch linorobot slam.launch
```

Drive around and map

Run teleop_twist_keyboard:

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

Run rviz:

```
$ roscd lino_visualize/rviz
```

```
$ rviz -d slam.rviz
```

Using teleop_twist_keyboard, drive the robot around the area you want to map.

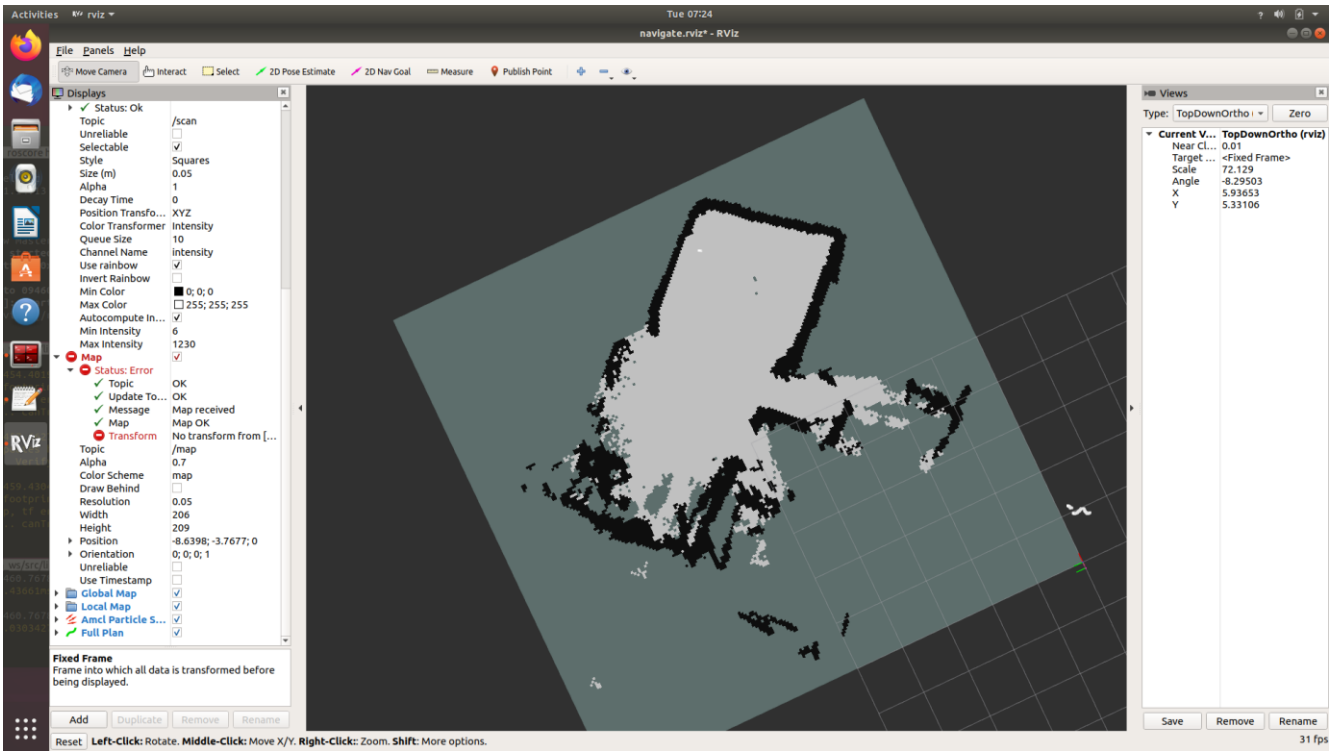


Fig 4.7: Map of our apartment using IMR

Saving the map

Once you are done mapping save the map by running map_server on the robot's computer:

```
$ rosrun map_server map_saver -f ~/linorobot_ws/src/linorobot/maps/map map:=/map/grid_proj
```

Check if map.pgm and map.yaml has been saved:

roscd linorobot/maps

https://github.com/0xDigimon/linorobot-my-workSpace/blob/master/maps/map.pgm
congratulations you finished map and next step create navigation.

4.4.2 Autonomous navigation

Define robot size

Navigation stack needs to be aware of the robot size when it plans the path. This is to make sure that the robot has enough clearance and not collide to any obstacle as it follows the planned path.

- Edit costmap_common_params.yaml:
- roscd linorobot/param/navigation
- nano costmap_common_params.yaml
- and define the robot's footprint:
- footprint: [[-x, -y], [-x, y], [x, y], [x, -y]]
- where x = ROBOT_LENGTH / 2 and y = ROBOT_WIDTH / 2

Launch AMCL app

On the robot’s computer, open 2 new terminals. Run bringup.launch:

roslaunch linorobot bringup.launch

Run navigate.launch:

```
$ roslaunch linorobot navigate.launch
```

On your development computer, run rviz:

```
$ roscd lino_visualize/rviz

$ rviz -d navigate.rviz
```

Set initial pose

- To localize the robot, click "2D Post Estimate".
- Pinpoint on the map the approximate location of the robot and drag towards the robot's heading.

Sending goals

After localizing, click "2D Nav Goal".

Pinpoint your desired location on the map and drag towards the direction you want your robot to be heading at once it has reached its goal.

Chapter Five

5 Conclusion and Results

5.1 CONCLUSION

Coordination of autonomous mobile robot systems and social robots has been regarded as a significant topic in research due to its importance in real-world applications such as exploration, reconnaissance of a wide area, map building of unknown/known environments, chatting with or helping humans, and home automation. However, there were different challenges in autonomous mobile robot systems which made it a bit difficult to apply results proposed in research to get reliable products that can be used in industry, gladly we managed to find our way through these challenges. The first challenge was cost, and gladly we managed to move beyond that with our cheap robot. The second challenge was the complexity of using ros platform and reaching an output using it.

5.2 EVALUATION

The ongoing evolution toward off-line programming of industrial robots together with ever-increasing absolute positioning accuracy requirements call for methods to calibrate the robots and their environment. A few characteristics will be mentioned below to help evaluate our robot.

5.2.1 Cost

The total cost of the robot is about 18000 L.E which is a low cost compared with other robots.

5.2.2 Manufacturability

The used materials are available and relatively cheap and the manufacturing processes applied are easy and not complex

5.2.3 Market Need

Social mobile robots are a raising field nowadays, there are a lot of companies and startups working in this field and have already started selling their products.

5.2.4 Environmental impact

As mobile robots are electrical powered so they don’t have any harmful effects on the environment.

5.3 FUTURE WORK

This section is about futuristic features we can add later to our IMR to make it even better:

- Developing an AI model to recognize objects, gesture Recognition, and Follow Lines to clear Obstacles, so the arm can hold them autonomously by adding more training data.
- Add a microphone array for Voice Control Navigation, Sound Source Localization, and Voice Control Robotic Arm
- Designing a mass production model for mechanical design and exporting more durable and reliable components.
- Adding more features like facial detection, VR mode, and fuzzy logic controller.

6 References

(1) Tzafestas, S. G. (2013). *Introduction to mobile robot control*. Elsevier.

(2) Alassar, A. Z., Abuhadrous, I. M., & Elaydi, H. A. (2010, February). Modeling and control of 5 DOF robot arm using supervisory control. In *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)* (Vol. 3, pp. 351-355). IEEE.

(3) Deshpande, V., & George, P. M. (2014). Kinematic modelling and analysis of 5 DOF robotic arm. *International Journal of Robotics Research and Development (IJRRD)*, *4*(2), 17-24.

(4) Abaas, T. F., Khleif, A. A., & Abbood, M. Q. (2020). Inverse kinematics analysis and simulation of a 5 DOF robotic arm using MATLAB. *Al-Khwarizmi Engineering Journal*, *16*(1), 1-10.

(5) Guglieri, G. (2020). Modelling and Control of a Skid-Steering Mobile Robot for Indoor Trajectory Tracking Applications (Doctoral dissertation, Shanghai Jiao Tong University).

(6) Wang, T., Wu, Y., Liang, J., Han, C., Chen, J., & Zhao, Q. (2015). Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor. *Sensors*, *15*(5), 9681-9702.

(7) Nasir, N. Z. M., Zakaria, M. A., Razali, S., & bin Abu, M. Y. (2017). Autonomous mobile robot localization using Kalman filter. In *MATEC Web of conferences* (Vol. 90, p. 01069). EDP Sciences.

(8) Zhao, J., Liu, S., & Li, J. (2022). Research and Implementation of Autonomous Navigation for Mobile Robots Based on SLAM Algorithm under ROS. *Sensors*, *22*(11), 4172.

(9) Marin-Plaza, P., Hussein, A., Martin, D., & Escalera, A. D. L. (2018). Global and local path planning study in a ROS-based research platform for autonomous vehicles. *Journal of Advanced Transportation*, *2018*, 1-10.

(10) Moshayedi, A. J., Roy, A. S., Sambo, S. K., Zhong, Y., & Liao, L. (2022). Review on: The service robot mathematical model. *EAI Endorsed Transactions on AI and Robotics*, *1*(1).