Misr University for Science & Technology
Mechatronics Engineering Department
Spring/Fall 2017-2018

# "LINDO"
# The Home Assistant Robot

**Prepared By:**

| | |
|---|---|
| Mohamed Abd Rab | 51329 |
| Islam Mohamed El-Nagdy | 51619 |
| Ahmed El-Sayed Hamed | 51565 |
| Mahmmod Mahmmod | 51569 |

A B.Sc Senior Project Report Submitted to the Faculty of Engineering at
**MUST University**
For the Fulfillment of the Bachelor Degree In
**Mechatronics Engineering Department**

June 20, 2018

# DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Bachelor of Science in Mechatronics Engineering is entirely our own work, that we have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others and to the extent that such work, if any, has been cited and acknowledged within the text of our work.

**Signed**: _____

_____

_____

**Date**: 20, June 2018.

# ACKNOWLEDGMENT

First of all, we would like to thank all of our professors and all the academic staff of the *Mechatronics department, Faculty of Engineering, MUST University,* for all of their help over the past five years. They have been a great source of support and encouragement. They have always made themselves available when it was needed, provided good directions and optimum advices.

Finally, we would like to thank everyone trusted in us and encouraged us in our project.

# Project purpose

The main objective of this project is to demonstrate a robust mobile robot to perform an autonomous navigation from point to another. Using SLAM and AMCL to define the map and navigate through it, and avoiding the faced obstacles. Also the robot is to socialize with humans using AI, help making their life easier, fetch them information from APIs from external sources and to entertain them as well. Also the robot upgrades your home by applying home automation system and security monitoring system using IOT, thus helps you control and monitor your home directly by voice or from anyplace using your phone.

# ABSTRACT

This project introduces a home assistant mobile robot designed to be used for indoor navigation. It can be operated either autonomously or controlled by man remotely over the network. Odometry information is used to estimate the robot's position relative to its origin. In order to achieve a robust odometry, the robot uses two sources of odometry, the linear velocities from the encoders and the angular velocity from the IMU. A Kinect 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.

An assistant system is integrated in the robot allows it to communicate and socialize with humans, via a Chatbot powered by Artificial Intelligence and Machine Learning, connected to external APIs to bring real-time information. Also the robot can Automate and Secure your home via an IOT system.

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 installation of manipulator (i.e. robot arm) for a variety of tasks.

# Table of Contents:

# Table of figures:

# Chapter One

## Introducation and Literature review

# 1.1 History:

Mobile robots have the capability to move around in their environment and are not fixed to one physical location. Mobile robots can be "autonomous" (AMR - autonomous mobile robot) which means they are capable of navigating an uncontrolled environment without the need for physical or electro-mechanical guidance devices. Alternatively, mobile robots can rely on guidance devices that allow them to travel a pre-defined navigation route in relatively controlled space (AGV - autonomous guided vehicle). By contrast, industrial robots are usually more-or-less stationary, consisting of a jointed arm (multi-linked manipulator) and gripper assembly (or end effector), attached to a fixed surface. Mobile robots have become more commonplace in commercial and industrial settings. Hospitals have been using autonomous mobile robots to move materials for many years. Warehouses have installed mobile robotic systems to efficiently move materials from stocking shelves to order fulfillment zones. Mobile robots are also a major focus of current research and almost every major university has one or more labs that focus on mobile robot research. Mobile robots are also found in industrial, military and security settings. Domestic robots are consumer products, including entertainment robots and those that perform certain household tasks such as vacuuming or gardening.

# 1.2 Introduction:

In daily life there are a lot of situations in which robots are needed to perform some tasks humans cannot deal with or make life easier for humans. Robots also can help people of special needs with what they cannot do like carrying heavy things, holding and placing parts or even home cleaning. Robots have many configurations, styles and mechanisms for motion. Some are legged, others are wheeled and the rest can fly, swim or dive. Each configuration has its functionality that others cannot do and also has limitations.

Wheeled robots are robots that navigate around the ground using motorized wheels to propel themselves. This design is simpler than using treads or legs and by using wheels they are easier to design, build, and program for movement in flat, not-so-rugged terrain. They are also better controlled than other types of robots. Disadvantages of wheeled robots are that they can't navigate well over obstacles, such as rocky terrain, sharp declines, or areas with low friction. Wheeled robots are most popular among the consumer market, their differential steering provides low cost and simplicity. Robots can have any number of wheels, but three wheels are sufficient for static and dynamic balance. Additional wheels can add to balance; however, additional mechanisms will be required to keep all the wheels on the ground when the terrain is not flat. In our project we introduce a robot with good navigation and localization technique to solve a lot of problems mentioned above in this section.

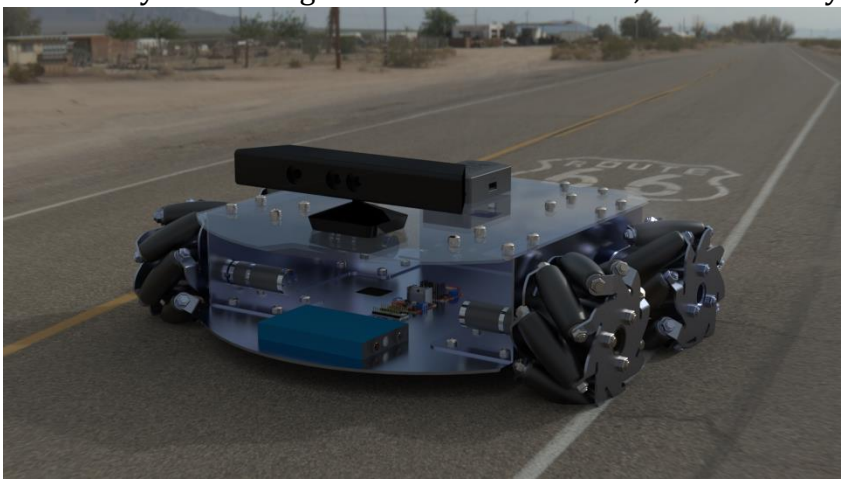


**Figure 1: LINDO**

## ❖ 1.2.1 Introduction To Wheeled robots:

Most wheeled robots use differential steering, which uses separately driven wheels for movement. They can change direction by rotating each wheel at a different speed. There may be additional wheels that are not driven by a motor these extra wheels help keep it balanced.

### ○ 1.2.1.1 Two Wheeled Robots:

Two wheeled robots are harder to balance than other types because they must keeping moving to maintain upright. The center of gravity of the robot body is kept below the axle, usually this is accomplished by mounting the batteries below the body. They can have their wheels parallel to each other, these vehicles are called dicycles, or one wheel in front of the other, tandemly placed wheels. Two wheeled robots must keep moving to remain upright and they can do this by driving in the direction the robot is falling. To balance, the base of the robot must stay with under its center of gravity. For a robot that has the left and right wheels, it needs at least two sensors. A tilt sensor that is used to determine tilt angle and wheel encoders which keep track of the position of the platform of the robot.

#### ▪ Examples:

- **Roomba**: Roombas are two-wheeled vacuum cleaners that automatically moves around cleaning up a room. They utilizes a contact sensor in the front and a infrared sensor on its top.

Figure 2: Roomba Robot

- **Segway**: Segways are self-balancing dicycle electric vehicles.

Figure 3: Segway Robot

## ○ 1.2.1.2 Three Wheeled Robots:

Three wheeled robots may be of two types: differentially steered (2 powered wheels with an additional free rotating wheel to keep the body in balance) or 2 wheels powered by a single source and a powered steering for the third wheel. In the case of differentially steered wheels, the robot direction may be changed by varying the relative rate of rotation of the two separately driven wheels. If both the wheels are driven in the same direction and speed, the robot will go straight. Otherwise, depending on the speed of rotation and its direction, the center of rotation may fall anywhere in the line joining the two wheels..

**Figure 4: Three Wheeled Robot**

## ○ 1.2.1.3 Four Wheeled Robots:

Four wheeled robots may be of two types: 2 powered, 2 free rotating wheels, Same as the Differentially steered ones above but with 2 free rotating wheels for extra balance.

More stable than the three wheel version since the center of gravity has to remain inside the rectangle formed by the four wheels instead of a triangle. This leaves a larger useful space. Still it's advisable to keep the center of gravity to the middle of the rectangle as this is the most stable configuration, especially when taking sharp turns or moving over a non-level surface.

The other type is 2-by-2 powered wheels for tank-like movement This kind of robot uses 2 pairs of powered wheels. Each pair (connected by a line) turn in the same direction. The tricky part of this kind of propulsion is getting all the wheels to turn with the same speed. If the wheels in a pair aren't running with the same speed, the slower one will slip (inefficient). If the pairs don't run at the same speed the robot won't be able to drive straight. A good design will have to incorporate some form of car-like steering.

**Figure 5: Four Wheeled Robot**

## ○ 1.2.1.4 Omni Wheeled Robots:

Another option for wheeled robots that makes it easier for robots with wheels not all mounted on the same axis to have Omni Wheels. An omni wheel is like many smaller wheels making up a large one, the smaller ones have axis perpendicular to the axis of the core wheel. This allows the wheels to move in two directions, and the ability to move holonomically, which means it can instantaneously move in any direction. Unlike a car, which moves non-holnomicallly and has to be in motion to change heading. Omni-wheeled robots can move in at any angle in any direction, without rotating beforehand. There is three types of omni wheeled robots:.

**3 Omni Wheeled Robots**: Omni wheeled robots use a triangular platform, with the three wheels spaced at 60 degree angles. With 90 degree rollers slope.

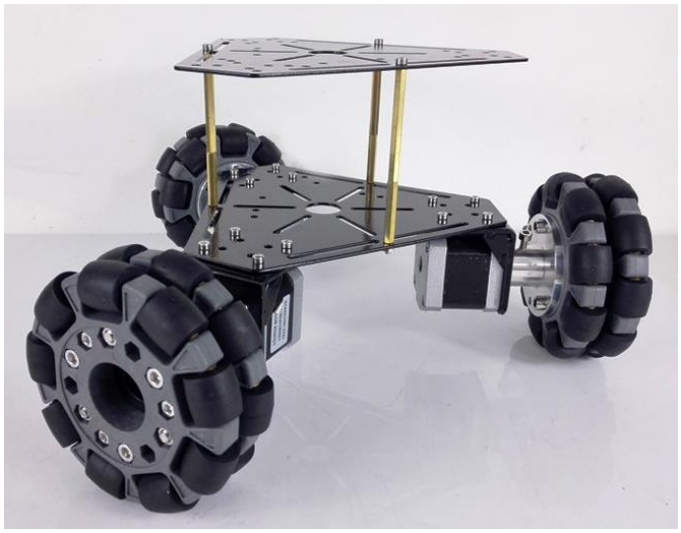


Figure 6: 3 Omni Wheeled Robot

**4 Omni Wheeled Robots**: Omni wheeled robots use a normal 4 wheeled platfor. With 90 degree rollers slope.

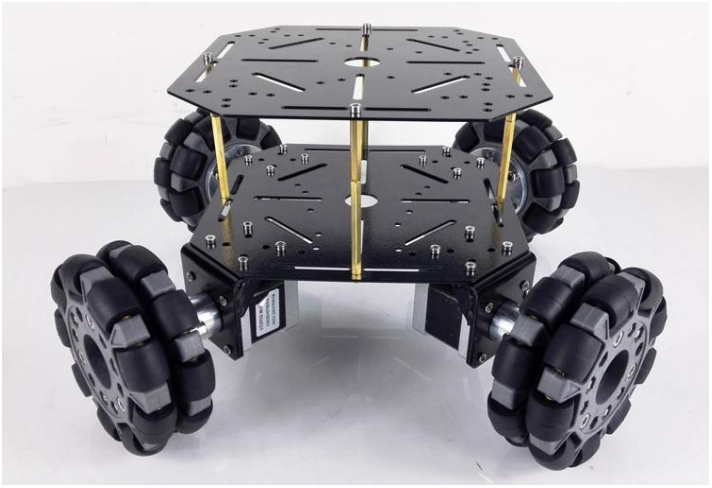


Figure 7: 4 Omni Wheeled Robot

**Mecanum Wheeled Robots :** Mecanum wheeled robots use a normal 4 wheeled platform, with 45 degree rollers slope.

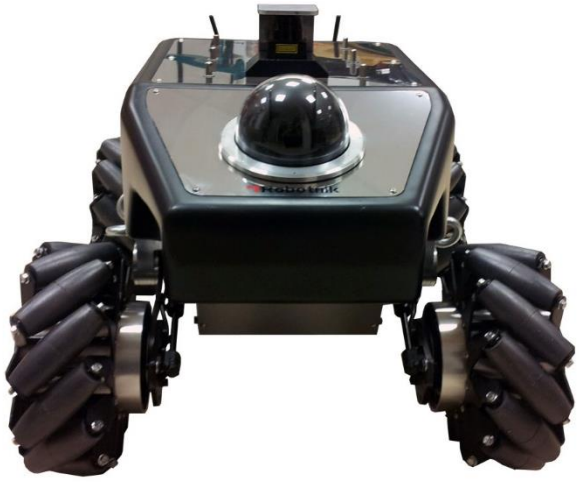


Figure 8: Mecanum Wheeled Robots

The disadvantages of using Omni wheels is that they have poor efficiency due to not all the wheels rotating in the direction of movement, which also causes loss from friction, and are more computationally complex because of the angle calculations of movement. Nonetheless they provide a wide range of ways to move in many directions.

## ❖ 1.2.2 Introduction To Social robot:

Figure 9: Social Robot

A social robot is an autonomous robot that interacts and communicates with humans or other autonomous physical agents by following social behaviors and rules attached to its role. Like other robots, a social robot is physically embodied (avatars or on-screen synthetic social characters are not embodied and thus distinct). Some synthetic social agents are designed with a screen to represent the head or 'face' to dynamically communicate with users. In these cases, the status as a social robot depends on the form of the 'body' of the social agent; if the body has and uses some physical motors and sensor abilities, then the system could be considered a robot.

### ○ 1.2.2.1 Background:

Another While robots have often been described as possessing social qualities (see for example the tortoises developed by William Grey Walter in the 1950s), social robotics is a fairly recent branch of robotics. Since the early 1990s artificial intelligence and robotics researchers have developed robots which explicitly engage on a social level. Notable researchers include Cynthia Breazeal, Tony Belpaeme, Aude Billard, Kerstin Dautenhahn, Yiannis Demiris, Hiroshi Ishiguro, Maja Mataric, Javier Movellan,

Brian Scassellati and Dean Weber. Also related is the Kansai engineering movement in Japanese science and technology --- for social robotics, see especially work by Takayuki Kanda, Hideki Kozima, Hiroshi Ishiguro, Micho Okada, Tomio Watanabe, and P. Ravindra S. De Silva.

Designing an autonomous social robot is particularly challenging, as the robot needs to correctly interpret people's action and respond appropriately, which is currently not yet possible. Moreover, people interacting with a social robot may hold very high expectancies of its capabilities, based on science fiction representations of advanced social robots. As such, many social robots are partially or fully remote controlled to simulate advanced capabilities. This method of (often covertly) controlling a social robot is referred to as a Mechanical Turk or Wizard of Oz, after the character in the L. Frank Baum book. Wizard of Oz studies are useful in social robotics research to evaluate how people respond to social robots.

### ○ 1.2.2.2 Societal Impacts:

The increasingly widespread use of more advanced social robots is one of several phenomena expected to contribute to the technological posthumanization of human societies, through which process "a society comes to include members other than 'natural' biological human beings who, in one way or another, contribute to the structures, dynamics, or meaning of the society.

## ❖ 1.2.3 Introduction To IOT:

The Internet of Things (**IoT**) is the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these things to connect and exchange data, creating opportunities for more direct integration of the physical world into computer-based systems, resulting in efficiency improvements, economic benefits and reduced human intervention.

The number of IoT devices increased 31% year-over-year to 8.4 billion in 2017 and it is estimated that there will be 30 billion devices by 2020. The global market value of IoT is projected to reach $7.1 trillion by 2020.

### ○ 1.2.3.1 History:

The definition of the Internet of things has evolved due to convergence of multiple technologies, including wireless communication, real-time analytics, machine learning, commodity sensors, and embedded systems. Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), and others all contribute to enabling the Internet of things.

The concept of a network of smart devices was discussed as early as 1982, with a modified Coke machine at Carnegie Mellon University becoming the first Internet-connected appliance, able to report its inventory and whether newly loaded drinks were cold. Mark Weiser's 1991 paper on ubiquitous computing, "The Computer of the 21st Century", as well as academic venues such as UbiComp and PerCom produced the contemporary vision of IoT.

## ○ 1.2.3.2 Applications:

The extensive set of applications for IoT devices is often divided into consumer, enterprise (business), and infrastructure spaces.

The concept of a network of smart devices was discussed as early as 1982, with a modified Coke machine at Carnegie Mellon University becoming the first Internet-connected appliance, able to report its inventory and whether newly loaded drinks were cold. Mark Weiser's 1991 paper on ubiquitous computing, "The Computer of the 21st Century", as well as academic venues such as UbiComp and PerCom produced the contemporary vision of IoT.

## ○ 1.2.3.3 Real World Applications of IoT:

- Smart Home
- Wearables
- Connected Cars
- Industrial Internet
- Smart Cities
- IoT in agriculture
- Smart Retail
- Energy Engagement
- IOT in Healthcare
- IoT in Poultry and Farming

## ○ 1.2.3.4 Smart Home:

With IoT creating the buzz, 'Smart Home' is the most searched IoT associated feature on Google. But, what is a Smart Home?

Wouldn't you love if you could switch on air conditioning before reaching home or switch off lights even after you have left home? Or unlock the doors to friends for temporary access even when you are not at home. Don't be surprised with IoT taking shape companies are building products to make your life simpler and convenient.

Smart Home has become the revolutionary ladder of success in the residential spaces and it is predicted Smart homes will become as common as smartphones.

The cost of owning a house is the biggest expense in a homeowner's life. Smart Home products are promised to save time, energy and money. With Smart home companies like Nest, Ecobee, Ring and August, to name a few, will become household brands and are planning to deliver a never seen before experience.

## ❖ 1.2.3 Introduction To ROS:

Robot Operating System (ROS) is robotics middleware (i.e. collection of software frameworks for robot software development). Although ROS is not an operating system, it provides services designed for heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. Despite the importance of reactivity and low latency in robot control, ROS, itself, is not a real-time OS (RTOS), though it is possible to integrate ROS with real-time code.[1] The lack of support for real-time systems is being addressed in the creation of ROS 2.0.[2]

Software in the ROS Ecosystem[3] can be separated into three groups:

- language-and platform-independent tools used for building and distributing ROS-based software;
- ROS client library implementations such as roscpp,[4] rospy,[5] and roslisp;[6]
- packages containing application-related code which uses one or more ROS client libraries.[7]

Both the language-independent tools and the main client libraries (C++, Python, and Lisp) are released under the terms of the BSD license, and as such are open source software and free for both commercial and research use. The majority of other packages are licensed under a variety of open source licenses. These other packages implement commonly used functionality and applications such as hardware drivers, robot models, datatypes, planning, perception, simultaneous localization and mapping, simulation tools, and other algorithms.

The main ROS client libraries (C++, Python, and Lisp) are geared toward a Unix-like system, primarily because of their dependence on large collections of open-source software dependencies. For these client libraries, Ubuntu Linux is listed as "Supported" while other variants such as Fedora Linux, macOS, and Microsoft Windows are designated "Experimental" and are supported by the community.[8] The native Java ROS client library, rosjava, however, does not share these limitations and has enabled ROS-based software to be written for the Android OS.[9] rosjava has also enabled ROS to be integrated into an officially supported MATLAB toolbox which can be used on Linux, macOS, and Microsoft Windows.[10] A JavaScript client library, roslibjs has also been developed which enables integration of software into a ROS system via any standards-compliant web browser..

# Chapter Two

# Design Procedure

# 2.1 Design Procedure:

In the first, we thought of building a humanoid instead of mobile robot, we had many pre-applied concepts in mind, just like :

Figure 10: Poppy

Figure 12: THORMANG 3

Figure 11: RoboSavvy V2

Due to some financial problems and they lack of components available we decided to focus on wheeled robots/mobile robots, and after some researches we had a cool idea regarding the design, AMIGO and Q.bo.

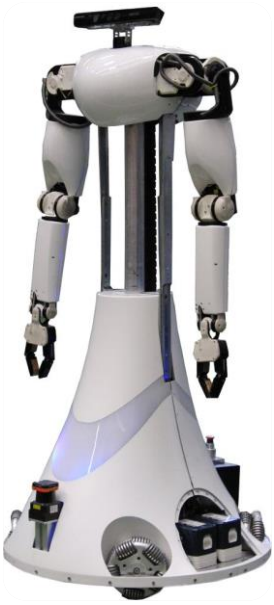


Figure 13: AMIGO

Figure 14: Q.bo

## ❖ 2.1.1 Specifications And Constrains:

In system design, a design constraint refers to some limitation on the conditions under which a system is developed, or on the requirements of the system. The design constraint could be on the systems form, fit or function or could be in the technology to be used, materials to be incorporated, time taken to develop the system, overall budget, and so on. A design constraint is normally imposed externally, either by the organisation or by some external regulation. During system design, it is as important to identify each design constraint as it is to elicit requirements since the design constraints place an overall boundary around the system design process.

### ○ 2.1.1.1 Desired Velocity and Acceleration:

The Desired velocity of our robot is to be 0.5 m/s, while the desired acceleration is to be 1 $m/s^2$.

### ○ 2.1.1.2 Maximum Incline:

A 5° max incline is to be set on our motor sizing calculations as a safety procedure.

### ○ 2.1.1.3 Supply Voltage:

A 12 V battery as a voltage supplier.

### ○ 2.1.1.4 Desired Operating Time:

About 20 minutes of operating time shall be enough for a home assistant robot prototype.

### ○ 2.1.1.5 Total Efficiency:

A 80%  total efficiency is to be set on our motor sizing calculations as a safety procedure.

## ❖ 2.1.2 Motor Sizing:

To obtain the rest of our constrains and design requirements, we need to complete our motor sizing calculations, Proper sizing and selection of a motor for your equipment is key to ensuring performance, reliability and cost of the equipment. The first step is to determine the drive mechanism for your equipment. Some examples are direct rotation, a ball screw, a belt and pulley or a rack and pinion. Along with the type of drive mechanism, you must also determine the dimensions, mass and friction coefficient, etc. that are required for the load calculation:

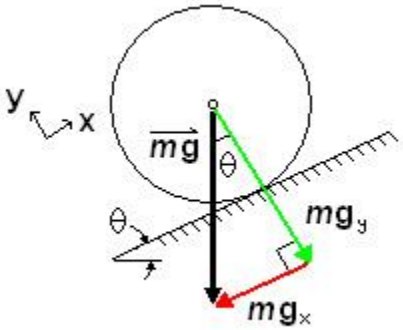
### ○ 2.1.2.1 Calculations:

To calculate the required torque, power, current and battery pack required by a wheeled mobile robot, there are several principles that must be understood: concept of vectors; 2D Force balance; Power; Current and Voltage.

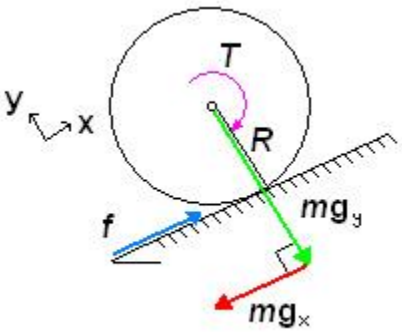
In order to roll on a horizontal surface, a wheeled robot's motors must produce enough torque to overcome any imperfections in the surface or wheels, as well as friction in the motor itself. Therefore theoretically, a robot (small or large) does not require much torque to move purely horizontally. Obviously there will be more friction and resistance in a large robot than in a small robot, though it is still exponentially less than when a robot encounters an incline.

In order for a robot to roll up an incline at a constant velocity (no acceleration or deceleration) it must produce enough torque to "counteract" the effect of gravity, which would otherwise cause it to roll down the incline. On an inclined surface (at an angle theta) however, only one component of its weight (mgx parallel to the surface) causes the robot to move downwards. The other component, mgy is balanced by the normal force the surface exerts on the wheels.

$$mg_x = mg * \sin(\theta)$$
$$mg_y = mg * \cos(\theta)$$




In order for the robot not to slide down the incline, there must be friction between the wheel and the surface. The motor in a heavy truck may be able to produce 250 horsepower and significant torque, but we have all seen (in person or in video) large trucks simply spinning their wheels as they fall backwards on an icy street. It is friction (f) that "produces" the torque.

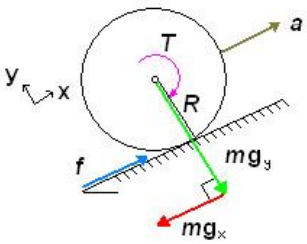



The torque (T) required is:

$$T = f * R$$

To select the proper motor, we must consider the "worst case scenario", where the robot is not only on an incline, but accelerating up it.

Note now that all forces (F) are along the x and y axes. We balance the forces in the x-direction:




$$\sum F_x = M * a = f - m * g_x$$

Inserting the equation for torque above, and the equation for mgx, we obtain:

$$M * a = T/R - M * g * sin(\Theta)$$

Rearrange the equation to isolate T:

$$T = R * M * (a + g * sin(\Theta))$$

This torque value represents the total torque required to accelerate the robot up an incline. However, this value must be divided by the total number (N) of drive wheels to obtain the torque needed for each drive motor. Note that we do not consider the total number of passive wheels as they have no effect on the torque required to move the object aside from adding weight.

$$T = \frac{(a+g*\sin(\theta))*M*R}{N}$$

The final point to consider is the efficiency (e) in the motor, gearing and wheel (slip).

$$T = (100/e)*\frac{(a+g*\sin(\theta))*M*R}{N}$$

This increases the torque required and compensates for inefficiencies.

Total power (P) per motor can be calculated using the following relation:

$$P = T*\omega$$

T is known from above and the angular velocity (w) is specified by the builder. It is best to select the maximum angular velocity to be able to find the corresponding maximum power. Knowing the maximum power and the supply voltage (V) which the builder chooses, we can find an idea of the maximum current (I) requirements:

$$P = I*V$$

The two equations above are used to produce the following relation:

$$I = \frac{T*\omega}{V}$$

Finally, the capacity (c) of battery pack required can be estimated using the equation:

$$c = I*t$$

You may wonder why such a large value is needed. This is because when choosing a battery pack, the rated amp hours are not an accurate indicate of the maximum current the pack can produce for extended periods of time. Also, the total charge is rarely retained over time.

## ○ 2.1.2.2 Motor Sizing Tool:

Using the calculations above we made our own motor sizing tool, our motor sizing tool was build using **C#**. The Drive Motor Sizing Tool is intended to give an idea of the type of drive motor required for your specific robot by taking known values and calculating values required when searching for a motor. DC motors are generally used for continuous rotation drive systems, though can be used for partial (angle to angle) rotation as well. They come in an almost infinite variety of speeds and torques to suite any need. Without a geardown, DC motors turn very fast (thousands of revolutions per minute (rpm)), but have little torque. To get feedback of the angle or the speed of the motor, consider a motor with an encoder option.

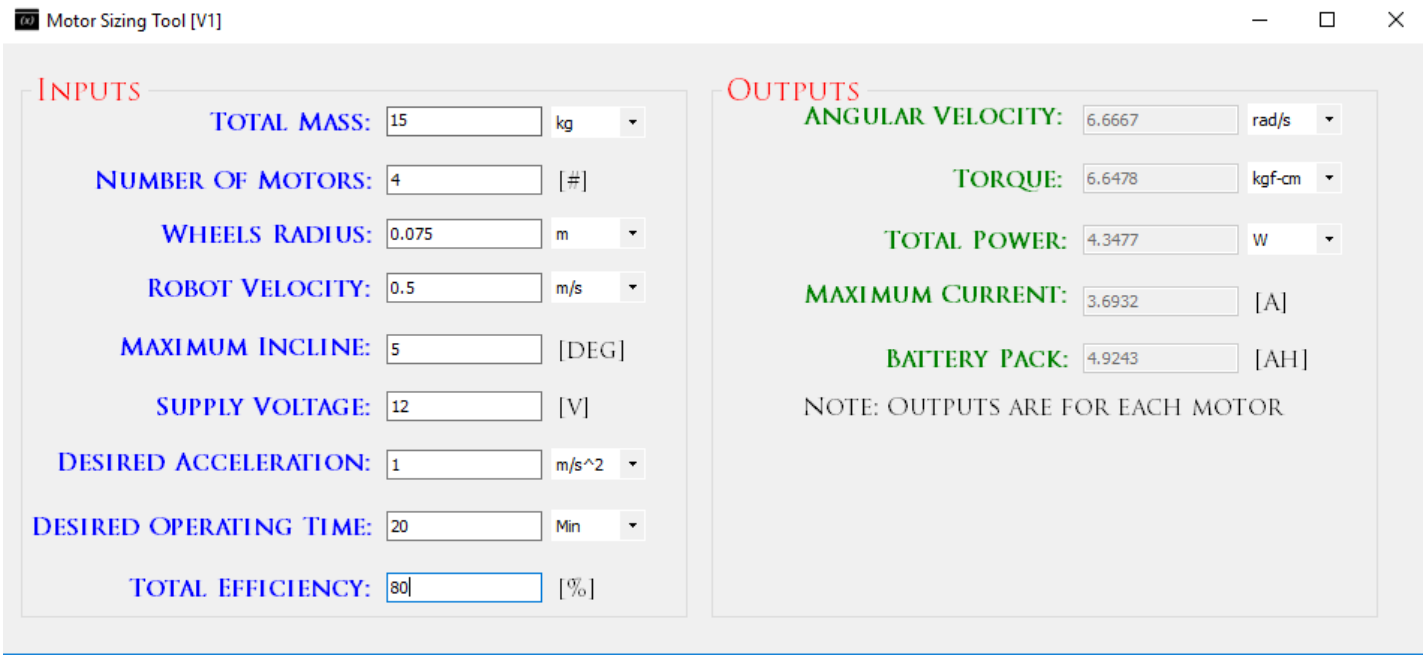



**Figure 15: Motor Sizing Tool**

## ○ 2.1.2.5 Motor Selection:

Using the gained outputs from the motor sizing calcuations, we can pick up the motors knowing these specs.:

- Torque : 8.9 Kgf.cm
- RPM: 133 rpm
- Voltage: 12 V
- Max Current: 3.3 A

## ❖ 2.1.3 Matrial Selection:

We picked Aluminum to be our main design material for the robot base and body, also an Acrylic upper layer is to be mounted on the robot.

- ### Aluminum:

#### 1- Weight:

One of the best known properties of aluminum is that it is light, with a density one third that of steel, 2,700 kg/m3. The low density of aluminum accounts for it being lightweight but this does not affect its strength .

#### 2- Strength:

Aluminum alloys commonly have tensile strengths of between 70 and 700 MPa. The range for alloys used in extrusion is 150 – 300 MPa. Unlike most steel grades, aluminum does not become brittle at low temperatures. Instead, its strength increases. At high temperatures, aluminum's strength decreases. At temperatures continuously above 100°C, strength is affected to the extent that the weakening must be taken into account.

#### 3- Linear expansion:

Compared with other metals, aluminum has a relatively large coefficient of linear expansion. This has to be taken into account in our designs.

#### 4- Machining:

Aluminum is easily fabricated using most machining methods – milling, drilling, cutting, punching, bending, etc. Furthermore, the energy input during machining is low.

#### 5- Formability:

Aluminum's superior malleability is essential for extrusion. With the metal either hot or cold, this property is also exploited in the rolling of strips and foils, as well as in bending and other forming operations.

#### 6- Joining:

Features facilitating easy jointing are often incorporated into profile design. Fusion welding, Friction Stir Welding, bonding and taping are also used for joining.

### 7- Corrosion resistance:

Aluminum reacts with the oxygen in the air to form an extremely thin layer of oxide. Though it is only some hundredths of a (my)m thick (1 (my)m is one thousandth of a millimeter), this layer is dense and provides excellent corrosion protection. The layer is self-repairing if damaged.

Anodizing increases the thickness of the oxide layer and thus improves the strength of the natural corrosion protection. Where aluminum is used outdoors, thicknesses of between 15 and 25 ¥ìm (depending on wear and risk of corrosion) are common.

Aluminum is extremely durable in neutral and slightly acid environments. In environments characterized by high acidity or high basicity, corrosion is rapid.

## • Acrylic:

### 1- Weight:

ACRYLITE FF sheet is only half the weight of glass and 43% the weight of aluminum.

### 2- Strength:

Although the tensile strength of ACRYLITE FF sheet is 10,000 psi (69 Mpa) at room temperature (ASTM D 638), stress crazing can be caused by continuous loads below this value. For glazing applications, continuously imposed design loads should not exceed 750 psi (5.2 Mpa) at 73°F (23°C). Temporary loads of up to 1,500 psi (10.4 Mpa) may be imposed for short durations of time at 73°F (23°C). Localized, concentrated stresses must be avoided. For this reason, and because of thermal expansion and contraction, large sheets should never be fastened with bolts, but should always be installed in frames. All thermoplastic materials, including ACRYLITE FF sheet, will gradually lose tensile strength as the temperature approaches the maximum recommended for continuous service–160°F (71°C).

### 3- Dimensional Stability:

Although ACRYLITE FF sheet will expand and contract due to changes in temperature and humidity, it will not shrink with age. Some shrinkage occurs when ACRYLITE FF sheet is heated to forming temperature, but post-forming stability is excellent.

### 4- Heat Resistance:

ACRYLITE FF sheet can be used at temperatures from -30°F (-34°C) up to +190°F (+88°C), depending on the application. It is recommended that temperatures not exceed 160°F (71°C) for continuous service, or 190°F (88°C) for short, intermittent use. Components made of ACRYLITE FF sheet should not be exposed to high heat sources such as high wattage incandescent lamps, unless the finished product is ventilated to permit the dissipation of heat.

## 5- Cutting and Machining:

ACRYLITE FF sheet can be sawed with circular saws or band saws. It can be drilled, routed, filed and machined much like wood or brass with a slight modification of tools. Because the sheet softens quickly, it is necessary to keep the cutting tool and machined edge of the sheet as cool as possible. Cooling of the cutting tool is recommended. Tool sharpness and "trueness" are essential to prevent gumming, heat buildup and stresses in the part. Heat buildup at the machined edge could lead to subsequent stress crazing and therefore must be avoided.

## 6- Laser Cutting:

Laser technology is rapidly being accepted by the industry for quick and accurate cutting, welding, drilling, scribing and engraving of plastics. CO2 lasers focus a large amount of light energy on a very small area which is extremely effective for cutting complex shapes in acrylic sheet. The laser beam produces a narrow kerf in the plastic allowing for close nesting of parts and minimal waste. CO2 lasers vaporize the acrylic as they advance resulting in a clean polished edge but with high stress levels; annealing acrylic sheet after laser cutting is recommended to minimize the chance of crazing during the service life of the part.

## 7- Surface Hardness:

The surface of plastic is not as hard as that of glass. Therefore, reasonable care should be exercised in handling and cleaning ACRYLITE FF sheet.

## 8- Electrical Properties:

ACRYLITE FF sheet has many desirable electrical properties. It is a good insulator. Its surface resistivity is higher than that of most plastics. Continuous outdoor exposure has little effect on its electrical properties

## ❖ 2.1.4 Mechanical Design:

### ○ 2.1.4.1 Old Design:

As a start we decided to used something like the AMIGO robot base design, which used 4 Omni wheels placed as a squared shape as follows:

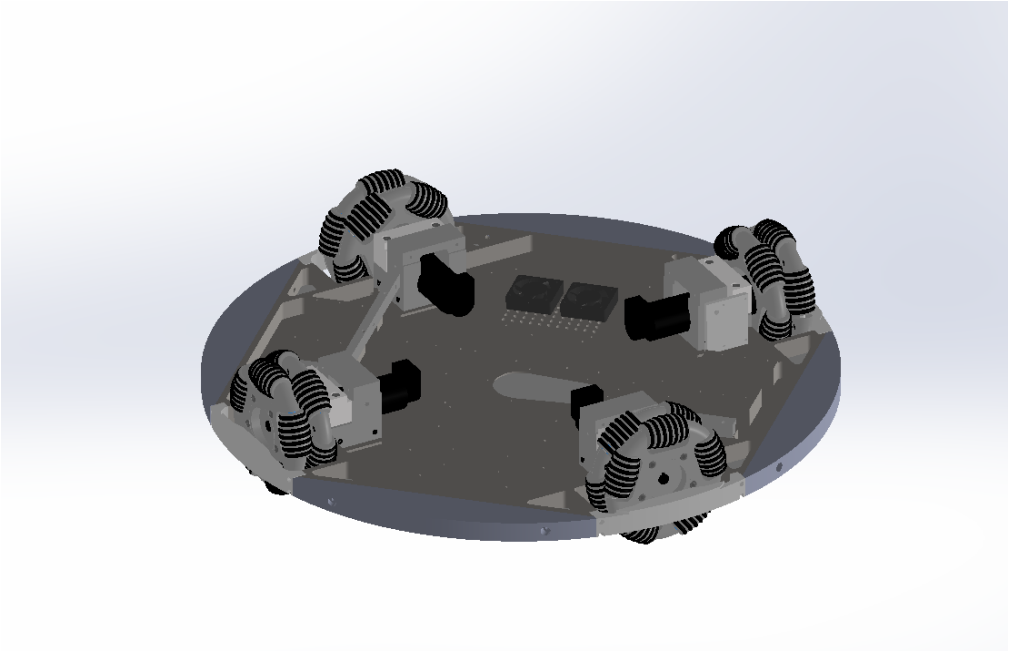


**Figure 16: First Design: 4 Omni Wheeled**

Due to the bigger size of the AMIGO robot, we had to build our own design, which was still based on the previous design. Which wasn't a bit different from the previous one, only in size.

## 2.1.4.2 Final Design:

After many further designs and justification, we decided to change the design due to the un-available wheels, a full new design was built this time using mecanum wheels.

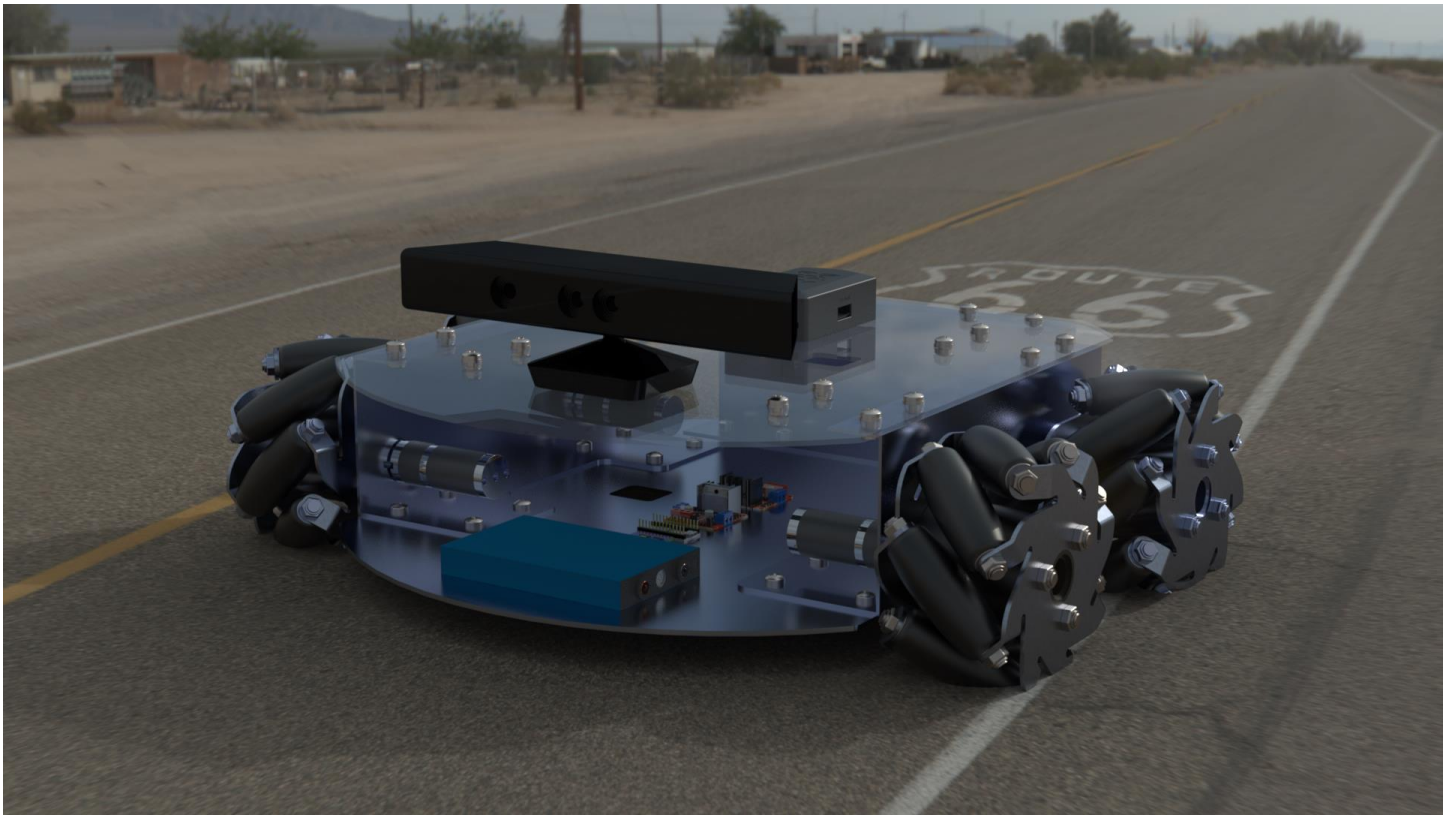


**Figure 17: Final Design: 4 Mecanum Wheeled**

After many further designs and justification, we decided to change the design due to the un-available wheels, a full new design was built this time using mecanum wheels

### 2.1.4.2.1 Design Constrains:

- **Length**: 50 cm
- **Width**: 40 cm
- **Height**: 15 cm

## 2.1.4.3 Outer Body Design:

After finalizing our mechanical design, we modeled an outer body design, which was meant to be built using 3D printing.

Sadly this proved to be extensively expensive, still the design is fully simulated on solidworks and ready to put in action, maybe one day.

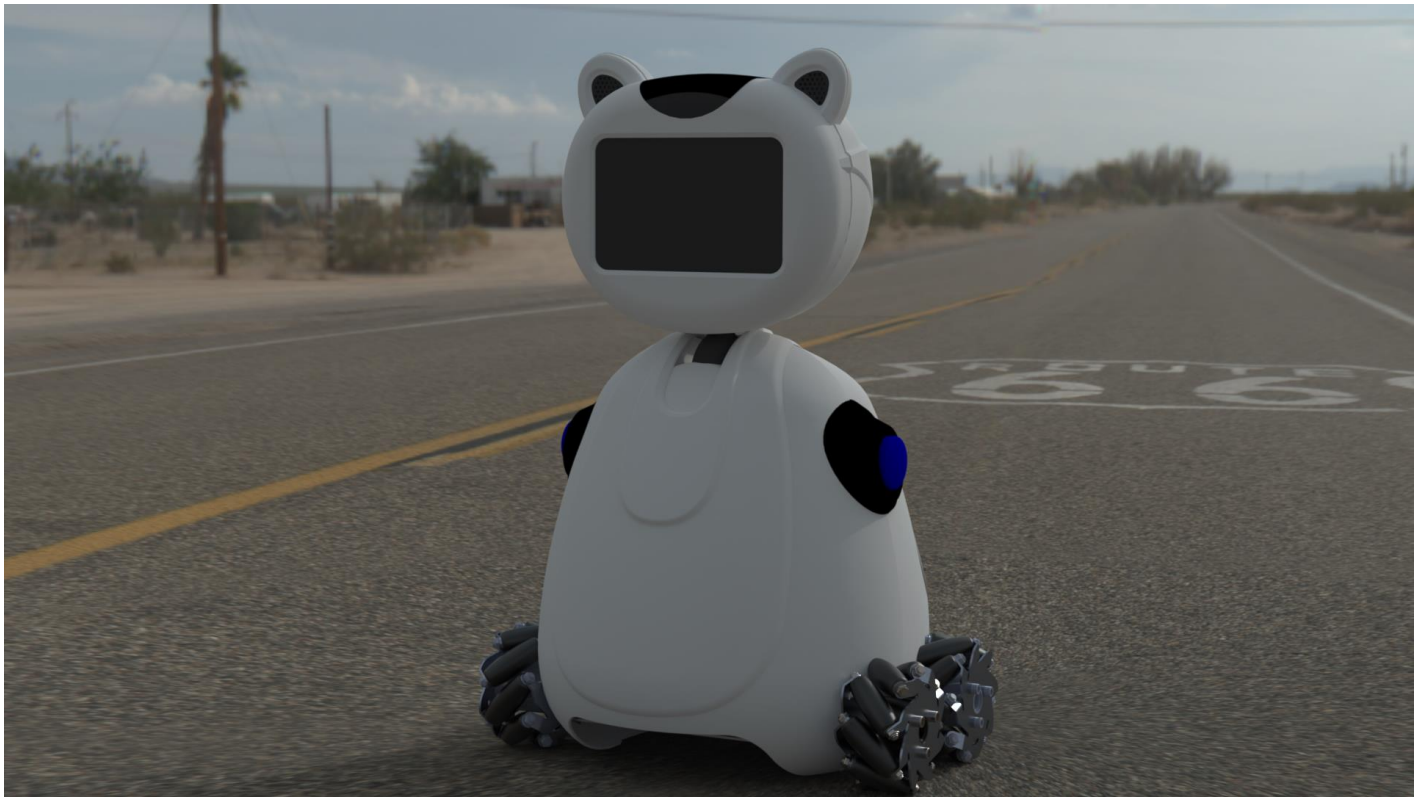


Figure 18 : Outer Body Design

### 2.1.4.3.1 Design Constains:

- **Length**: 50 cm
- **Width**: 40 cm
- **Hight**: 100 cm

## ○ 2.1.4.4 Home Design:

| ITEM NO. | PART NUMBER | QTY. |
|---|---|---|
| 1 | Part1 | 2 |
| 2 | Part5 | 1 |
| 3 | Part2 | 1 |
| 4 | Part3 | 1 |
| 5 | Part4 | 1 |
| 6 | Part6 | 1 |
| 7 | Part7 | 3 |

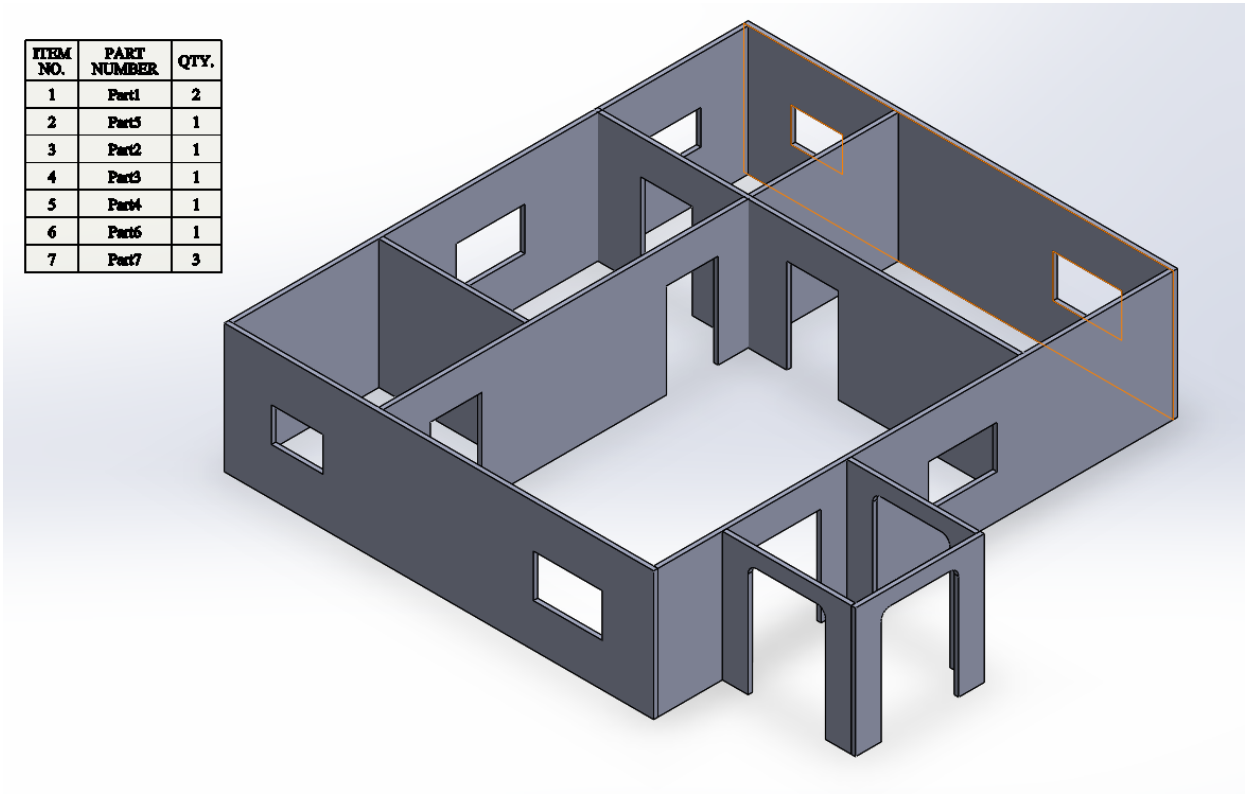


**Figure 19: SolidWorks Home Design**

**Figure 20: Rendered Design**

## ❖ 2.1.5 V-Model Design Procedure Approach:

Upon working on the robot and moving further with the tasks. We had to put an approach to work on, an approach which will help saving time and decreases errors in the whole desiging system.

In development, the V-model represents a development process that may be considered an extension of the waterfall model, and is an example of the more general V-model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. 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 represents time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.

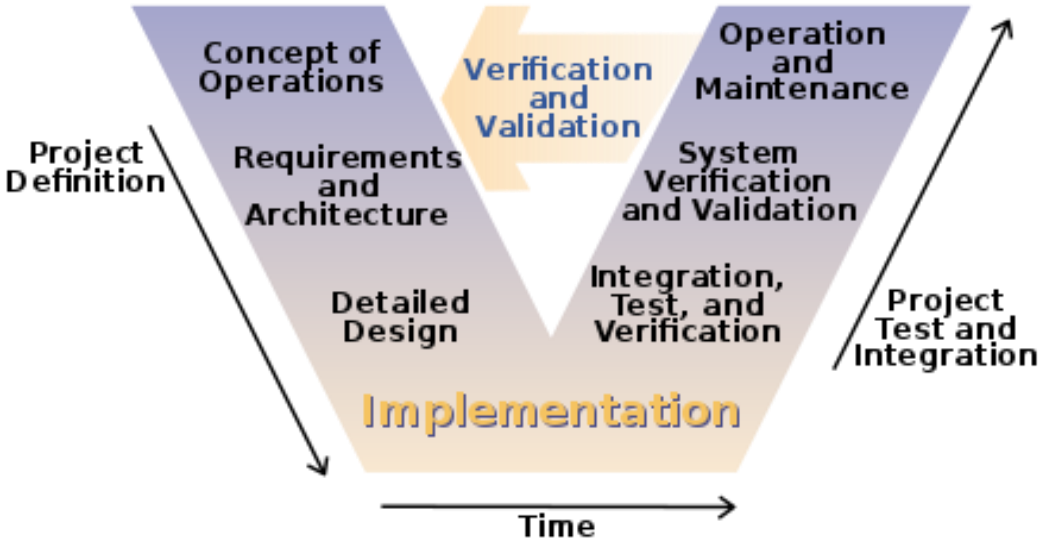


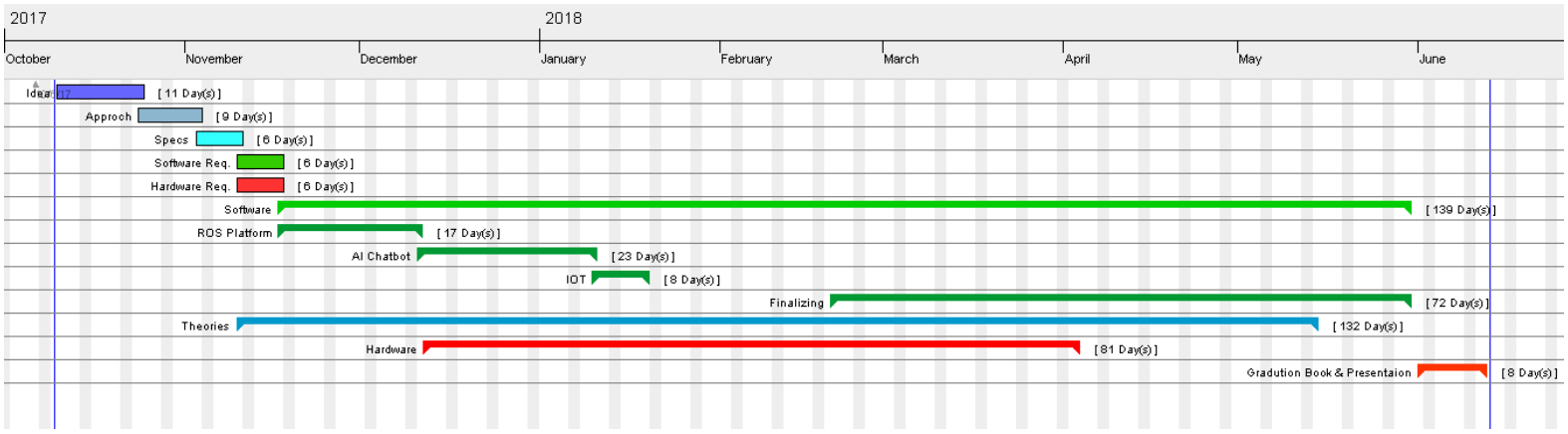

Figure 21: V Model

And that was exactly how we worked in parallel to take the maximuim advantage of this approach.

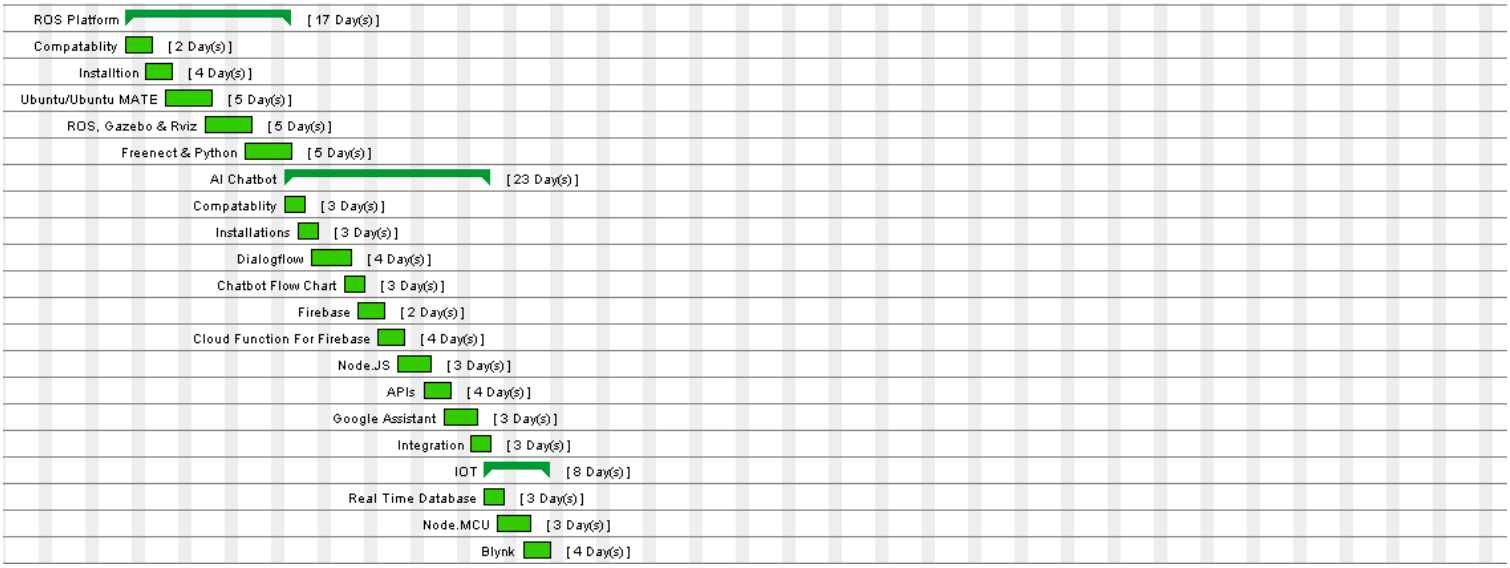
## ❖ 2.1.6 Gantt Chart:

Because our project was large and complex, we had to have our own very detailed timeline and tasks. This helped us reducing time (because we worked in parallel), to reduce errors, following deadlines and improved the output.

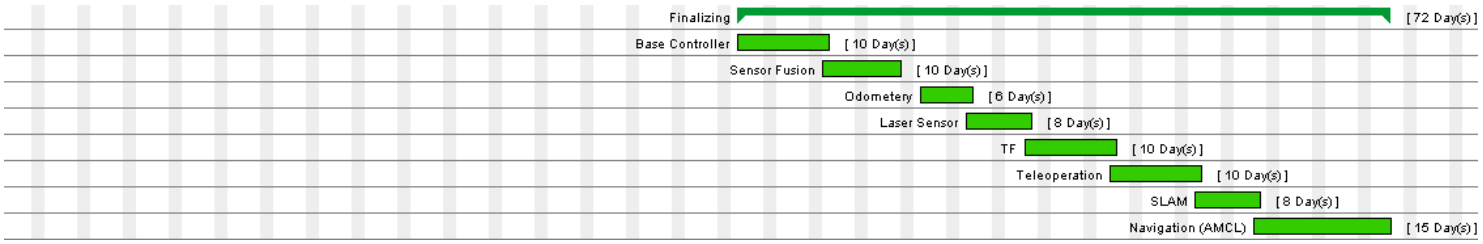
### ○ 2.1.6.1 Overview Of Our Chart:



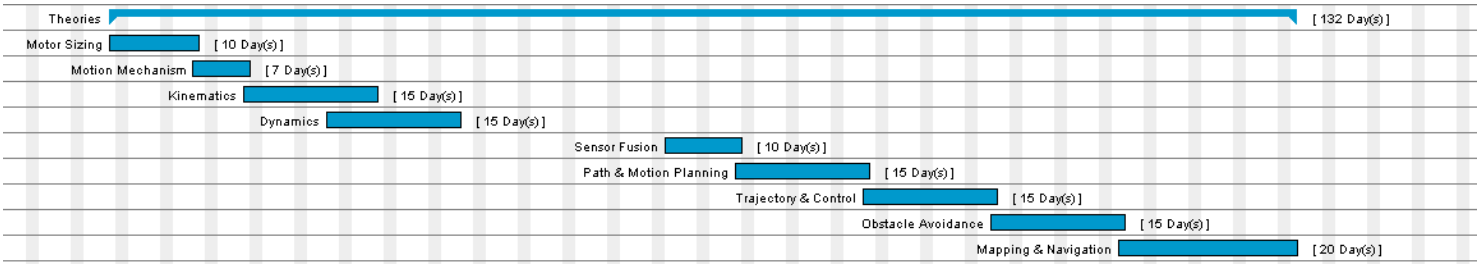

### ○ 2.1.6.2 Ros Platform, AI ChatBot, IOT:



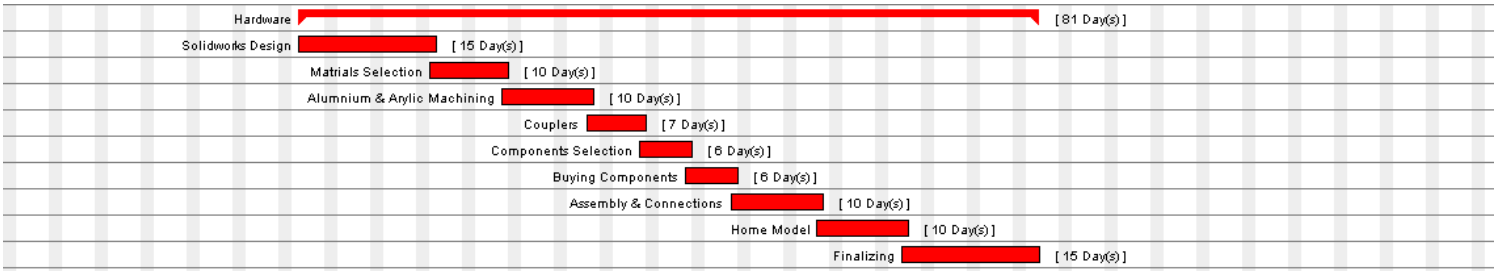
## ○ 2.1.6.3 Software Finalizing:




## ○ 2.1.6.4 Software Finalizing:




## ○ 2.1.6.5 Hardware:



# Chapter Three

# Hardware Node and Modules Tree Structure

# 3.1 Hardware Ros Node:

In the first, we thought of building a humanoid instead of mobile robot, we had many pre-applied concepts in mind, just like :

In prior sections we always look at hardware as a black box. This view is sfficient when we want to describe the overall operation of the robot. But lets now take this part in some details to know how instructions are processed after being received from either mapping or computer vision node. In the design stage of hard ware, One of the most important aspects taken into consideration is modularity of design. Each Part in the hardware structure has a unique role and a method by which this role can be triggered for execution. Another important style of design followed is assigning low level tasks to multiple separate modules rather than having a central unit responsible for the whole operation. This method of design saves a lot of resources in the master unit opening the way for other complicated operations to be performed faster. In the following sections we will talk first about the role of hardware ROS node and then go down at low level.

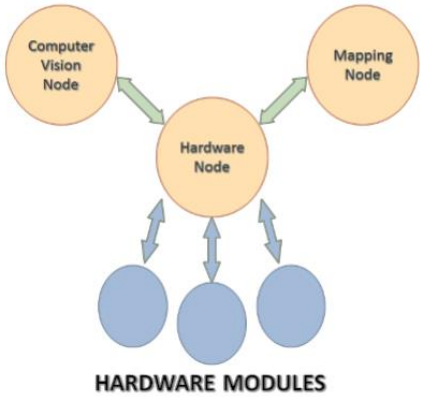



Figure 22: Hardware node standing midway between ROS nodes and hardware modules

The hardware node stands midway between both mapping and computer vision nodes. its main role is receiving instructions from other nodes, refining them and forwarding them to low-level modules to be performed by robot. This leads to two important results. First, the hardware ROS node is free of low-level operations which keeps its main role of regarding and communicating with other nodes. Second, low level modules save their processing capability for optimized motion and performance rather than being concerned with communication of multiple parts.

## ❖ 3.1.1 Hardware Requirements:

### • 3.1.1.1 Motors:

This is 12V DC motor, almost double the power of our 6V DC motor. It is a powerfull motor fitted with a 75:1 metal gearbox. The gears are all steel and the output shaft is 4mm diameter. The motor has extra rear shaft (on the high speed side, before the gearbox) which allow you to mount motor encoder for RPM counting and superior control.

- Specs.:
  - Voltage = 12V
  - No load current = 250mA
  - Sall Current = 3.3 A
  - Sttall Torque = 8.8 Kg/cm
  - Motor Rated RPM = 133 Shaft Diameter = 4 mm
  - Motor Length (without shaft) = 52 mm
  - Motor Diameter = 25 mm


Figure 23: 12V DC Motor

- ## 3.1.1.2 Encoders:

One of the most popular method of tracking the robot position is calculating the number of revolution each motor made. This way is called Odometry feedback. Each encoder has a property of number of pulses generated per revolution. This is an indication of how much distance change the robot can feel and encounter. The type we used in out project is 3 PPR. But our motor is geared with a gear head of ratio 1:30. That makes the exact value 90 pulses generated for each wheel revolution Moreover, This Encoder has 2 channels each generates 3 pulses per revolution with phase shift 90 degrees. The logic circuitry in motor driver kit combines the two channels' pulses by an XOR gate resulting in an interrupt signal of 180 PPR. This gives a resolution of 0.21 cm per encoder interrupt pulse.

Figure 24: Motor shaft encoder 3PPR

- Specs.:
  - Two channel hall effect encoder.
  - Operation Voltage : 4.5~24V
  - Operation Current : 14~20mA
  - 3PPR Motor

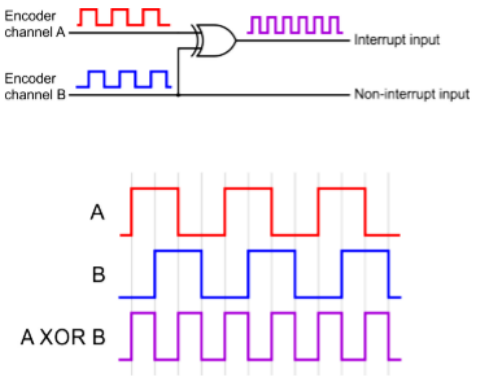



Figure 25: Quadrature encoder pulses explained

- ## 3.1.1.3 DC Motor Driver:

There are a lot of modules having the function of controlling DC motors. Each has it own specs like max current, operating voltage and temperature. We find this kit is most suitable for our robot as it contains 4 channels for 4 dc motors. Moreover it requires just two pins to control both speed and direction of motors. It also have important features; current sensor and impeded logic circuitry for handling Encoder signals come from shaft encoder sensor. Its Electrical specs are 12V operating voltage and max current of 4.5A for each channel. These features and specs make it the best choice for our robot.

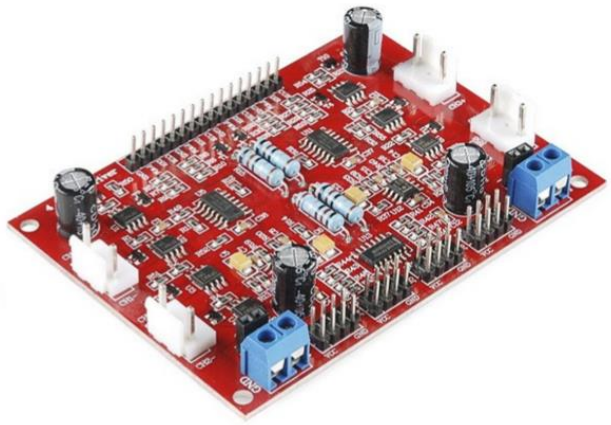


Figure 26: DC Motor Driver for Robot 4 channel

- ## 3.1.1.4 IMU:

This sensor provides measurements of both acceleration and angular velocity. So it can help in the estimation process as we know that displacement is the integral of integral of acceleration. And on the other hand, the heading angle of robot can be calculated from integrating the angular velocity measured by gyroscope. But the integration operation in such case leads to accumulated errors as we will show in next sections.


**Figure 27: MPU6050 Accelerometer and Gyroscope**

- ## 3.1.1.5 KINECT:

The Kinect sensor is a flat, black box that sits on a small platform when placed on a table or shelf near the television you're using with your Xbox 360. This device has the following three sensors that we can use for vision and robotics tasks:

- A color VGA video camera to see the world in color
- A depth sensor, which is an infrared projector and a monochrome CMOS sensor working together, to see objects in 3D
- A multiarray microphone that is used to isolate the voices of the players from the noise in the room


**Figure 28: Kinect V1**

## ❖ 3.1.2 Processing Units:

### • 3.1.2.1 Raspberry PI R3:

This part represents the master unit for the whole process. Its powerful processing capabilities opens the way complex processes that need much resources like CPU cycles or storage. One of these operations is the image processing and computer vision operated in out project. all the other parts are connected to Raspberry Pi by some way in a tree structure fashion. The operating system running on such unit is Ubuntu Mate 16.04 and we installed ROS platform base on it. We communicate and deal with this unit using SSH (Secure Shell) connection operated remotley from any other computer device.

Figure 29:Raspbery Pi R3

### • 3.1.2.2 Teensy 3.2:

Teensy 3.2 adds a more powerful 3.3 volt regulator, with the ability to directly power ESP8266 Wifi, WIZ820io Ethernet and other power-hungry 3.3V add-on boards.
The RAM has quadrupled since 3.0, from 16K to 64K. While 16K is plenty for nearly all Arduino libraries, 64K allows for more advanced applications. Icons and graphics for color displays and audio effects requiring delays, like reverb and chorus, will become possible on Teensy 3.2 & 3.1.

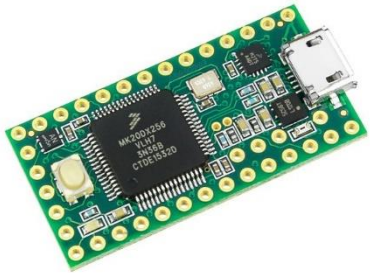


Figure 30:Teensy 3.2

### • 3.1.2.3 NodeMCU:

This unit is a developed with integrated WiFi module. This helped us too much in the design of IOT system. We implemented firebase on it that listened to the values which was sent from the real time database. This design and procedure will be explained in the software development section. We deal with this kit like an ordinary Arduino and program it using same Arduino IDE after installing some updates.

Figure 31:Node MCE

## ❖ 3.1.3 Power Source:

### • 3.1.3.1 Lithium 12V Battery:

This battery is 10000 Ah and is the power feeder to DC motors and flash light. It can support up to 10A making it lasts for one our of operation under full load condition.


Figure 32: 12V Battery

### • 3.1.3.2 NiMH 5V Battery:

This battery powers up the servo motors controlling the camera orientation. Its capacity is 1500 mAh and has su-cient high discharge current


Figure 33: 5V Battery

## ❖ 3.1.4 Components List:

| No. | Component Name | Usage | Quantity |
|---|---|---|---|
| **1** | Raspberry Pi R3 | Master Unit | 1 |
| **2** | Teensy 3.2 | Processing unit | 1 |
| **3** | DC Motor | Manipulators for robot motion | 4 |
| **4** | Motor Driver | Controlling motors speed and derivation | 1 |
| **5** | Encoders | Counting revolutions of motor | 4 |
| **6** | IMU | Obtaining the liner and angular velocities | 1 |
| **7** | Kinect | Obtain depth and real time images | 1 |
| **8** | Node MCU | Micro controller with embedded WiFi module | 1 |
| **9** | 12V Lithium Battery | Power source for the motors | 1 |

# 3.2 Wiring Schematic:

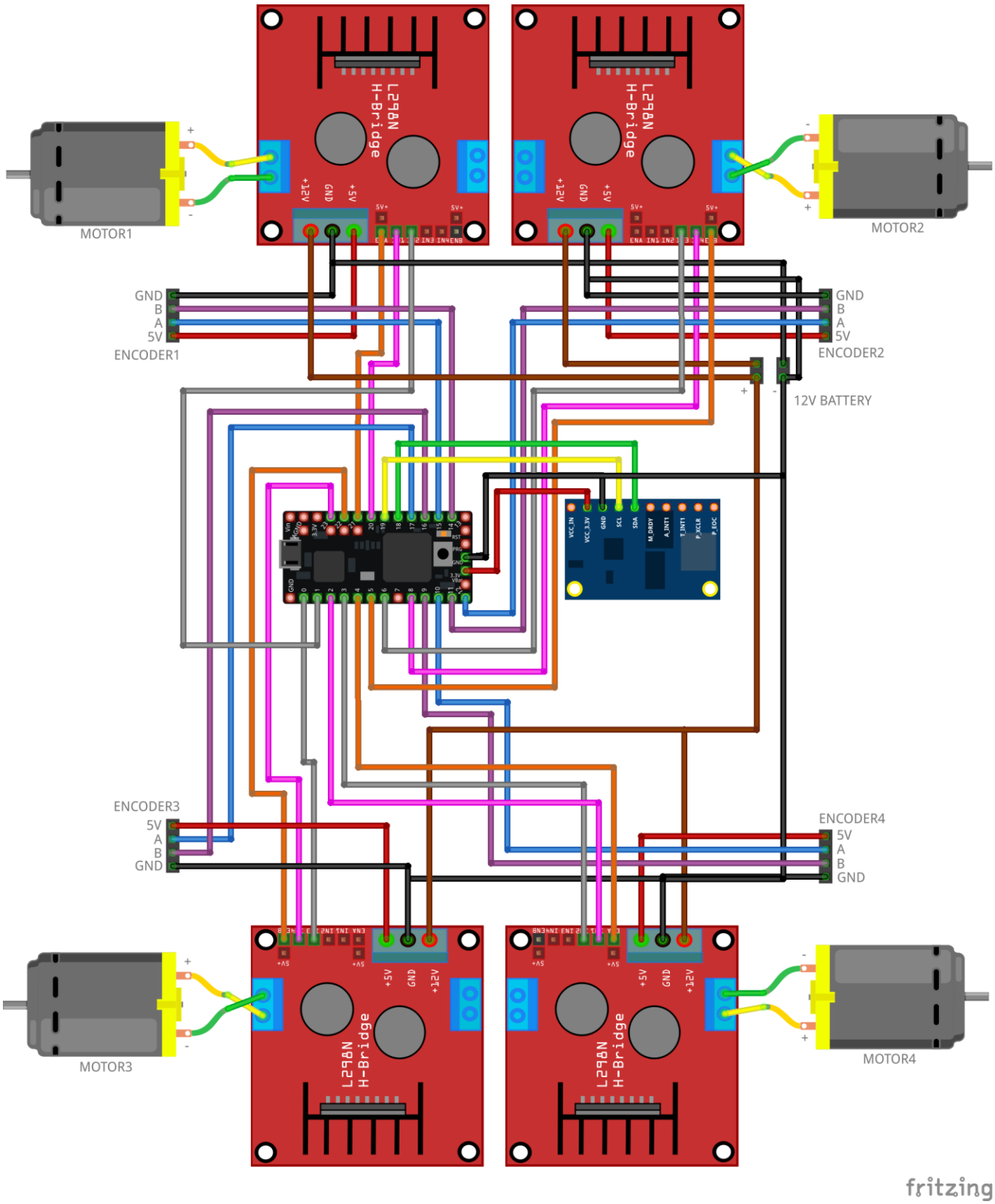



Figure 34: Wiring Schematic

# Chapter Four

# Theories

# 4.1 Kinematics Of The Robot:

## ❖ 4.1.1 What is a mecanum wheel?

In 1973, BengtIlon invented the Mecanum wheel (also called Ilon wheel) when he was an engineer with the Swedish company . The Mecanum wheel is designed with passive rollers mounted around the wheel circumference at an angle of 45 degrees to the wheel plane, thus it allows for in place rotation with small ground friction and low driving torque. Usually the mobile robots using Mecanum wheels, such as an intelligent wheelchair, a forklift, or the URANUS omni-directional robot, are designed with four wheels to provide agile mobility in any direction without changing its orientation. This omni-directional capability provides greater flexibility in congested environments.

Omni-differential locomotion is being using in current mobile robots in order to obtain the additional maneuverability and productivity. These features are expanded at the expense of improved mechanical complication and increased complexity in control mechanism. Omni-differential systems work by applying rotating force of each individual wheel in one direction similar to regular wheels with a different in the fact that Omni-differential systems are able to slide freely in a different direction, in other word, they can slide frequently perpendicular to the torque vector. The main advantage of using Omni-drive systems is that translational and rotational motions are decoupled for simple motion although in making an allowance for the fastest possible motion this is not essentially the case.

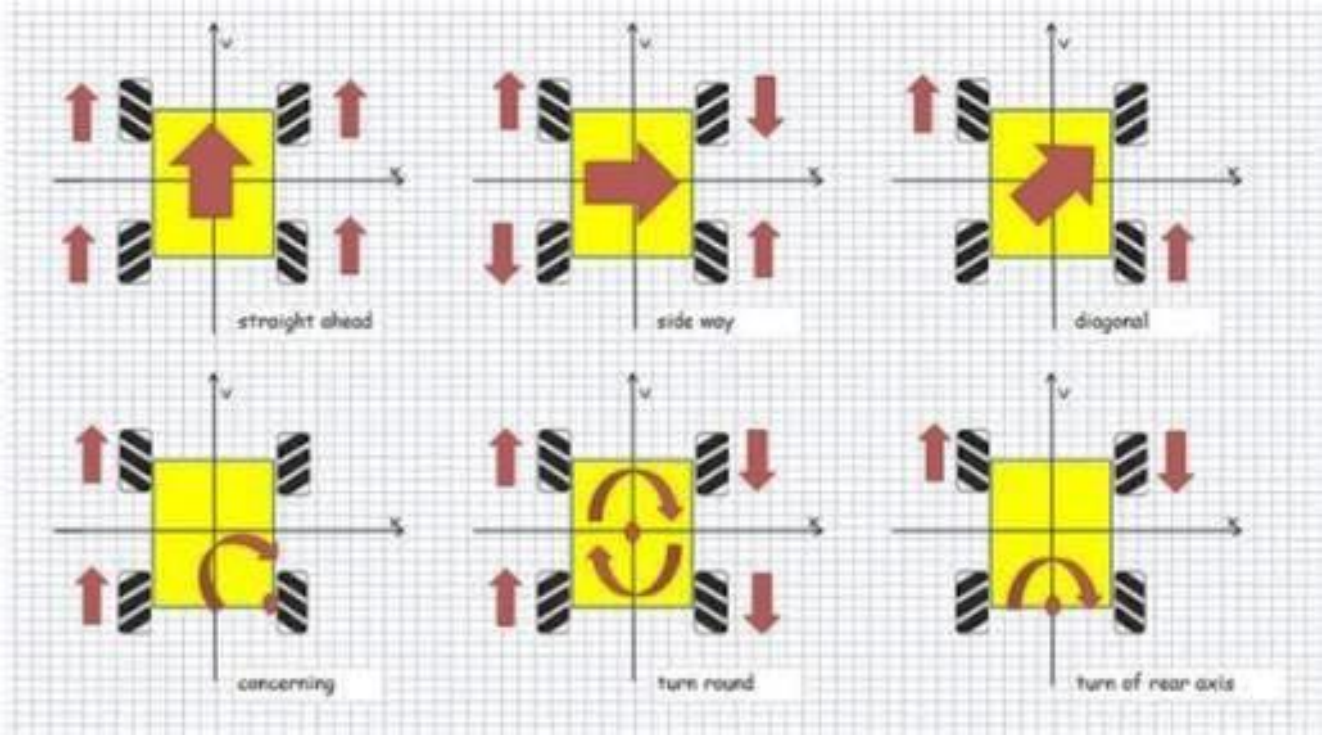



**Figure 35: Mecanum Wheels Combinations**

## ❖ 4.1.2 GEOMETRY:

**The configuration parameters and system velocities are** defined as follows:

•$x, y, \theta$, robot's position (x, y) and its orientation angle θ (The angle between X and $XR$).

•X G Y, inertial frame; x,y are the coordinates of the reference point O in the inertial basis.

•$XR\ O\ YR$, robot's base frame; Cartesian coordinate system associated with the movement of the body center.

•$Si$ , coordinate system of ith wheel in the wheel's center point .

•O, , the inertial basis of the Robot in Robot's frame and $Pi$ = {$XPi$ , $YPi$ } the center of the rotation axis of the wheel $i$ .

• , is a vector that indicates the distance between Robot's center and the center of the wheel $i$th.

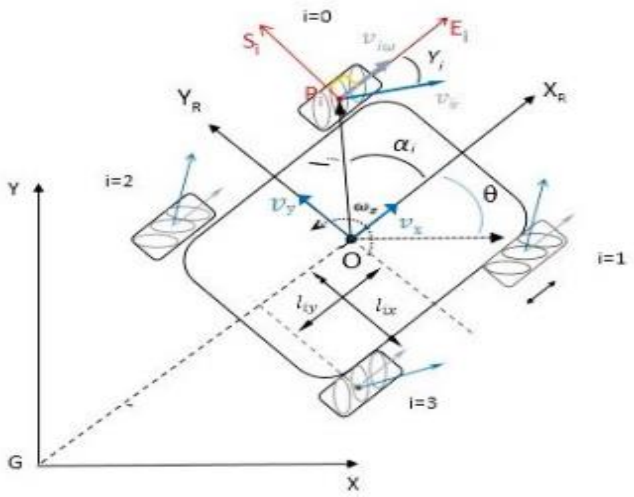



Figure 36: Base and Wheels Constrains

• , : $lix$ is half of the distance between front wheels and , $liy$ is half of the distance between front wheel and the rear wheels.

• , distance between wheels and the base (center of the robot O).

• , denotes the radius of the wheel i (Distance of the wheel's center to the roller center).

• , denotes the radius of the rollers on the wheels.

• , the angle between O$Pi$ and XR.

• , the angle between Si and XR .

• , the angle between $vir$ and .

•$\omega i$ [rad/s], wheels angular velocity.

•$vi\omega$ $[m/s]$, $i$ = 0,1,2,3 ∈ , is the velocity vector corresponding to wheel revolutions

• , the velocity of the passive roller in the wheel i .

•.[$wsi\ wEi$ ωi ] T , Generalized velocity of point $Pi$ in the frame $SiPiEi$ .

•[$vSi\ vEi$ ωi ] T , Generalized velocity of point $Pi$ in the frame $XROYR$.

•$vx$ , $vy$ [m/s] - Robot linear velocity;

•$\omega z$ [rad/s] - Robot angular velocity.

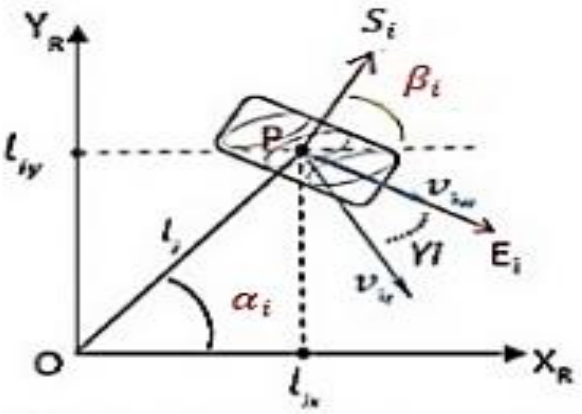



Figure 37: Mecanum Wheels Constains

## ❖ 4.1.3 Calculations:

1- we can calculate the velocity of the wheel i and the tangential velocity of the free roller attached to the wheel touching the floor :
$vir = (1/\cos 45)rr*\omega i$
$wEi = ri*\omega i$

2- the velocity of the wheel $i$ in the frame $Si\ Pi\ Ei$ , can be derived by :
$vSi = vir * \sin \gamma i$ .
$vEi = \omega i*ri + vir *\cos$ .

$$\begin{bmatrix} v_{S_i} \\ v_{E_i} \end{bmatrix} = \begin{bmatrix} 0 & \sin\gamma_i \\ r_i & \cos\gamma_i \end{bmatrix} \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} = {}^{w_i}T_{P_i} \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix}.$$

The transformation matrix from velocities of the ith wheel to its center:

$${}^{w_i}T_{P_i} = \begin{bmatrix} 0 & \sin\gamma_i \\ r_i & \cos\gamma_i \end{bmatrix}.$$

$$\begin{bmatrix} v_{iX_R} \\ v_{iY_R} \end{bmatrix} = \begin{bmatrix} \cos\beta_i & -\sin\beta_i \\ \sin\beta_i & \cos\beta_i \end{bmatrix} \begin{bmatrix} v_{S_i} \\ v_{E_i} \end{bmatrix} = {}^{w_i}T_{P_i}\,{}^{P_i}T_R \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix}.$$

3- Then, the transformation matrix from the ith wheel's center to the robot coordinate's system can be obtained from equation

$${}^{P_i}T_R = \begin{bmatrix} \cos\beta_i & -\sin\beta_i \\ \sin\beta_i & \cos\beta_i \end{bmatrix}.$$

Since the robot's motion is planar, we also have :

$$\begin{bmatrix} v_{iX_R} \\ v_{iY_R} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -l_{iy} \\ 0 & 1 & l_{ix} \end{bmatrix} \begin{bmatrix} v_X \\ v_Y \\ \omega \end{bmatrix} = T' \begin{bmatrix} v_{X_R} \\ v_{Y_R} \\ \omega_R \end{bmatrix}.$$

$$T' = \begin{bmatrix} 1 & 0 & -l_{iy} \\ 0 & 1 & l_{ix} \end{bmatrix}.$$

4- the inverse kinematic model can be obtained :

$${}^{w_i}T_{P_i}\,{}^{P_i}T_R \begin{bmatrix} \omega_i \\ v_{ir} \end{bmatrix} = T' \begin{bmatrix} v_{X_R} \\ v_{Y_R} \\ \omega_R \end{bmatrix}, i = 0,1,2,3.$$

$$\begin{bmatrix} \boldsymbol{v_{X_R}} \\ \boldsymbol{v_{Y_R}} \\ \boldsymbol{\omega_z} \end{bmatrix} = \boldsymbol{T^+} \begin{bmatrix} \boldsymbol{\omega_i} \\ \boldsymbol{v_{ir}} \end{bmatrix}$$

$$\begin{bmatrix} \boldsymbol{\omega_i} \\ \boldsymbol{v_{ir}} \end{bmatrix} = \boldsymbol{T} \begin{bmatrix} \boldsymbol{v_{X_R}} \\ \boldsymbol{v_{Y_R}} \\ \boldsymbol{\omega_R} \end{bmatrix}$$

5-

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix} = \frac{-1}{r}\begin{bmatrix}\frac{\cos(\beta_1-\gamma_1)}{\sin\gamma_1} & \frac{\sin(\beta_1-\gamma_1)}{\sin\gamma_1} & \frac{l_1\sin(\beta_1-\gamma_1-\alpha_1)}{\sin\gamma_1}\\ \frac{\cos(\beta_2-\gamma_2)}{\sin\gamma_2} & \frac{\sin(\beta_2-\gamma_2)}{\sin\gamma_2} & \frac{l_2\sin(\beta_2-\gamma_2-\alpha_2)}{\sin\gamma_2}\\ \frac{\cos(\beta_3-\gamma_3)}{\sin\gamma_3} & \frac{\sin(\beta_3-\gamma_3)}{\sin\gamma_3} & \frac{l_3\sin(\beta_3-\gamma_3-\alpha_3)}{\sin\gamma_3}\\ \frac{\cos(\beta_4-\gamma_4)}{\sin\gamma_4} & \frac{\sin(\beta_4-\gamma_4)}{\sin\gamma_4} & \frac{l_4\sin(\beta_4-\gamma_4-\alpha_4)}{\sin\gamma_4}\end{bmatrix}\begin{bmatrix}v_X\\ v_Y\\ \omega_z\end{bmatrix}.$$

6-

| $i$ | Wheels | $\alpha_i$ | $\beta_i$ | $\gamma_i$ | $l_i$ | $l_{ix}$ | $l_{iy}$ |
|---|---|---|---|---|---|---|---|
| 0 | 1sw | $\pi/4$ | $\pi/2$ | $-\pi/4$ | l | $l_x$ | $l_y$ |
| 1 | 2sw | $-\pi/4$ | $-\pi/2$ | $\pi/4$ | l | $l_x$ | $l_y$ |
| 2 | 3sw | $3\pi/4$ | $\pi/2$ | $\pi/4$ | l | $l_x$ | $l_y$ |
| 3 | 4sw | $-3\pi/4$ | $-\pi/2$ | $-\pi/4$ | l | $l_x$ | $l_y$ |

## ❖ 4.1.4 Inverse kinematics:

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix} = \frac{1}{r}\begin{bmatrix}1 & -1 & -(l_x+l_y)\\ 1 & 1 & (l_x+l_y)\\ 1 & 1 & -(l_x+l_y)\\ 1 & -1 & (l_x+l_y)\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ \omega_z\end{bmatrix}.$$

$$\begin{cases}\omega_1 = \frac{1}{r}(v_x - v_y - (l_x+l_y)\omega),\\ \omega_2 = \frac{1}{r}(v_x + v_y + (l_x+l_y)\omega),\\ \omega_3 = \frac{1}{r}(v_x + v_y - (l_x+l_y)\omega),\\ \omega_4 = \frac{1}{r}(v_x - v_y + (l_x+l_y)\omega).\end{cases}$$

## ❖ 4.1.5 Forward kinematics:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix}$$

Longitudinal Velocity:

$$v_x(t) = (\omega_1 + \omega_2 + \omega_3 + \omega_4) \cdot \frac{r}{4}$$

Transversal Velocity:

$$v_y(t) = (-\omega_1 + \omega_2 + \omega_3 - \omega_4) \cdot \frac{r}{4}$$

Angular velocity:

$$\omega_z(t) = (-\omega_1 + \omega_2 - \omega_3 + \omega_4) \cdot \frac{r}{4(l_x+l_y)}$$

## ❖ 4.1.6 Conceptual Design:

Before going any further, to confirm our kinematics model we made a simple design using matlab and Simmechanics. The design was built on solidworks and then coverted and imported to matlab using Simmechanics.

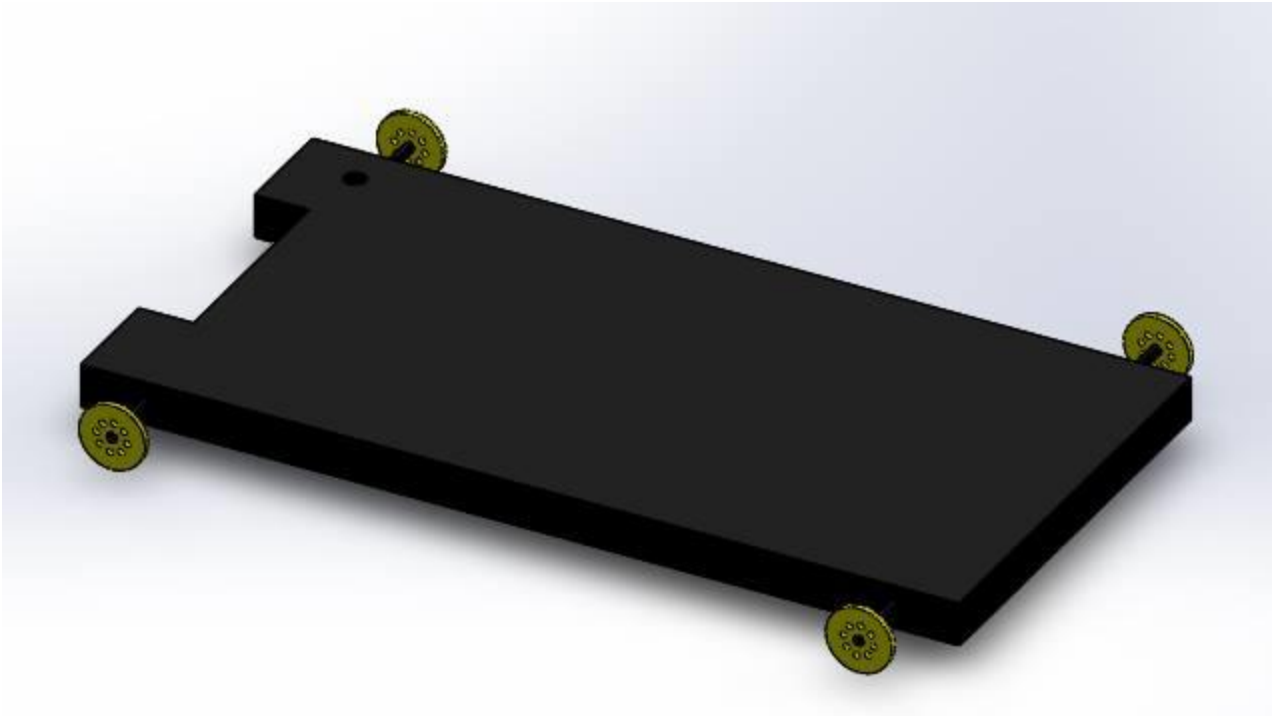


**Figure 38: Conceptual Design For Kinematics Validation**

Using our drived forward kinematics model, we imported that into matlab:

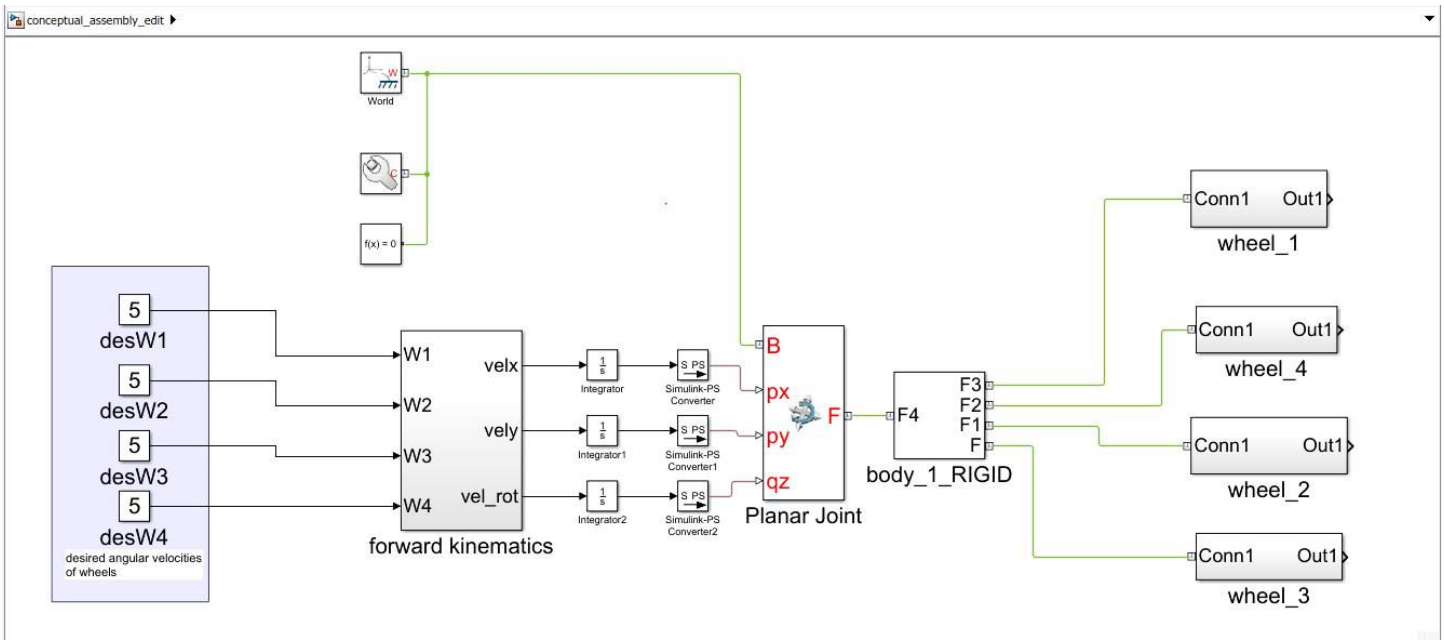
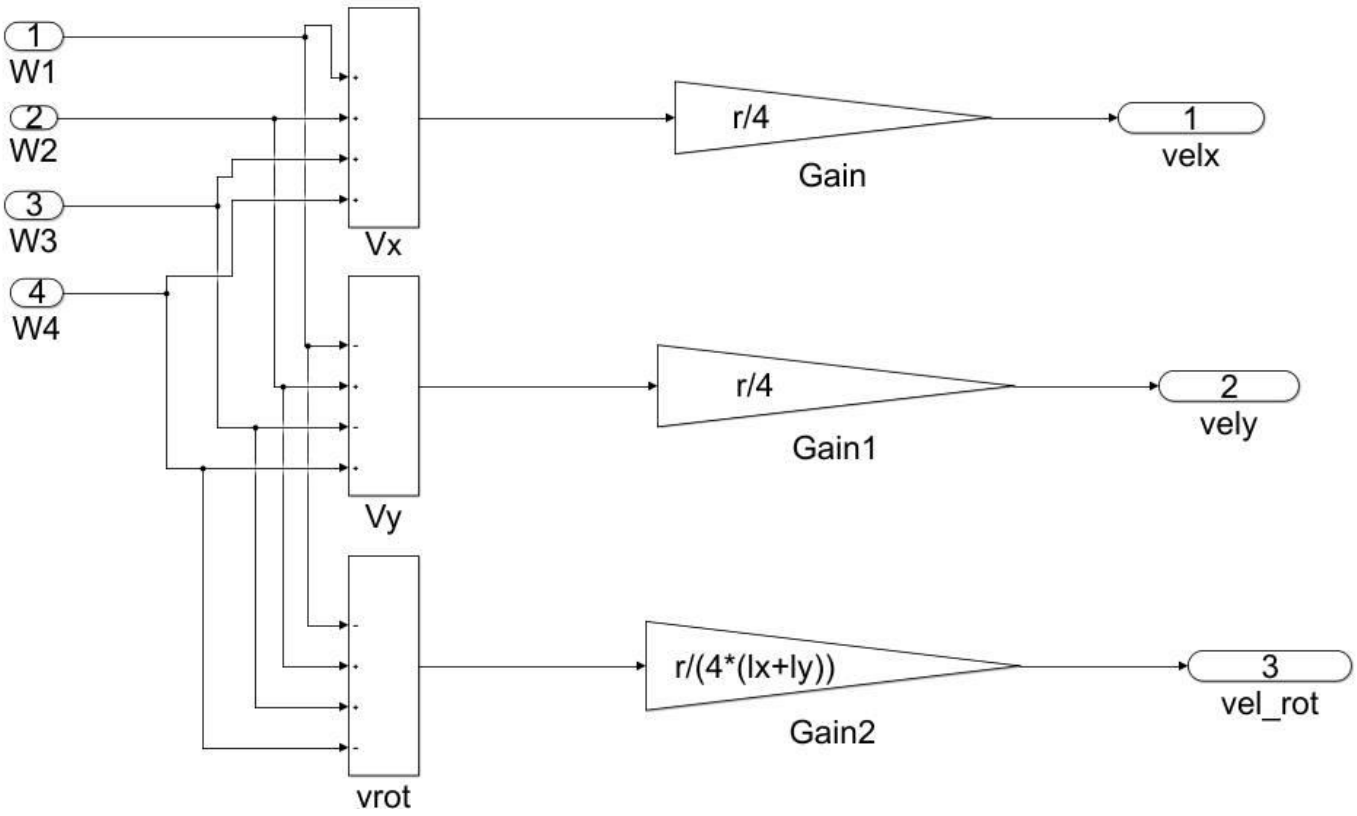



**Figure 39: Kinematic's matlab simmechanics**




As the robot moved as it was meant to be, we proved our kinematics to be perfectly working.

# 4.2 Dynamics Of The Robot:

## ❖ 4.2.1 Linear Motion:

Apply 2nd law of newton

$$Mm\ S''I = FI$$

Using Transformation Matrix:

$$S'I = R(\theta).S'R$$
$$FI = R(\theta).SR$$

After Subustituting and using the matrices forms, and adding the friction and disturbance occurrence:

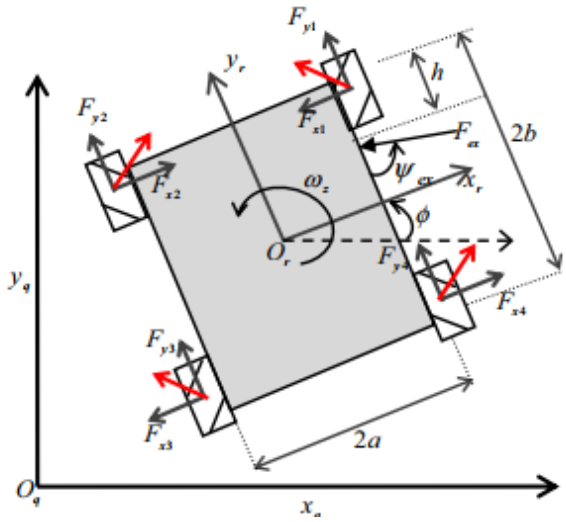



**Figure 40: Base and Wheels Constrains**

$$Friction = [Bx\ X'R \quad By\ Y'R]T$$
$$Disturbance = [-Fex\ \cos(Yaw) \quad Fex\ \sin(Yaw)]T$$

We get:

$$\begin{vmatrix} M \\ 0 \\ 0 \end{vmatrix} \begin{vmatrix} X''_R - \theta'. \\ Y'_R \\ Y''_R + \theta'. \end{vmatrix} = [F_{xR} \quad F_{yR}]^T - [B_x\ X'_R \quad B_y\ Y'_R]^T + [-F_{ex}\cos(Yaw) \quad F_{ex}\sin(Yaw)]^T$$

Model Of Motors to determine FxR And FyR:

$$Fi = (KT/R.Ra).U - (Kb.KT/Ra.R2).W$$
$$FXR = 0.5.(-F1 + F2 - F3 + F4)$$
$$FYR = 0.5.(F1 + F2 + F3 + F4)$$

For Rotational Motion, We Apply Eular Law:

$$I_I\ \theta'' = T + F_{ex}\cos(Yaw).a - F_{ex}\sin(Yaw).(b-h)$$
$$T = 0.5.a(F_1 - F_2 - F_3 + F_4) + 0.5.b(F_1 - F_2 - F_3 + F_4)$$

# 4.3 MOTION PLANNING:

Any robotic system typically consist of 3 components :

1. Mechanisms (mechanical design)
2. Perception (sensors)
3. **Decision making and control system :**

(modulates the behavior of the robot to achieve the desired goal )

a) SLAM
b) motion planning
c) control system

## ❖ 4.3.1 Motion planning problem:

**DEFINITION** :
The process of selecting a motion and associated set of input forces and torques from the set of all possible motions and inputs while ensuring that all constraints are satisfied.

## ❖ 4.3.2 Motion planning classification:

### o Explicit motion planning:

- Decompose MP problem into 3 tasks:
  **Path Planning**, **Trajectory Planning**, & **Robot Control**
- Example: Road Map Method, Cell Decomposition …. , etc.

### o Implicit Motion Planning:

- Trajectory and actuators input are not explicitly compute before the motion occur.
- Combine Path Planning, Trajectory Planning, and Robot Control in a single framework.
- Example : Artificial potential field

## ❖ 4.3.3 Explicit motion planning:

### o 4.3.3.1 Path planning:

This is a geometrical problem where we develop AI algorithms to determine the shortest path that minimize the costs(distance , time , velocity , …etc.)
Here we will interest in :

### o 4.3.3.2 Graph-based planning algorithms :

- **Definition :** Procedure (algorithm) that the computer can use to guide the robot from its start location to its end location , while minimizing the total number of steps that are performed .
- These algorithms developed based on **GRAPH THEORY**

- ## 4.3.3.3 GRAPH THEORY:
  - Definition: is a set of vertices (nodes) (V) and a set of edges (E)
    - G=(V,E)
  - Vertex : represent something ( EX: waypoint in robot configuration space)
    - Edges : represent a line connects two vertices and show the relations between them
  - Some important definitions :
  - Adjacent nodes : nodes that are connected to each other by a single edge
  - Path : vertices in sequence connected by edges and the path length is the summation of edges in a path
  - Directed graph : all edges go both direction
  - Undirected graph : edges go one way
  - Weighted graph: each edge has an associated weight (weight can be cost, distance, etc.)

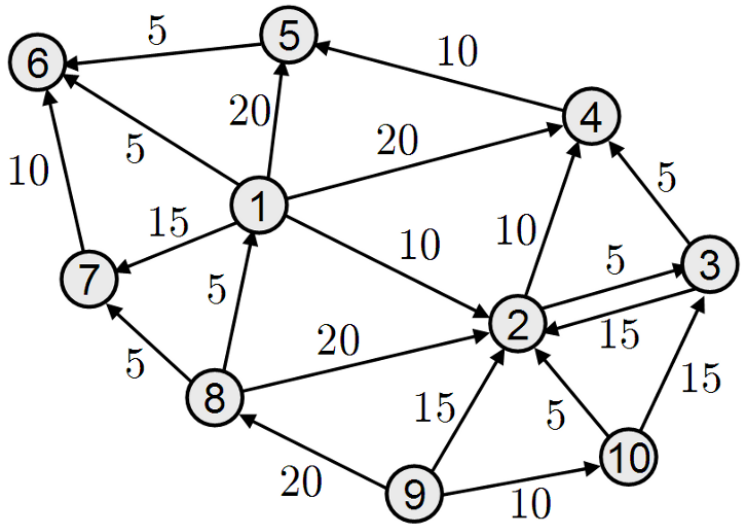



Figure 41: Graph Theory Example

## ❖ 4.3.4 PATH PLANNING SEARCH ALGORITHMS:

### 4.3.4.1 SINGLE SOURCE SHORTEST PATH (SSSP):

- Definition : this family of algorithms analysis **only** weighted graphs to find a path
- Examples : bellman ford , directed acyclic graph , **Dijkstra**.

### 4.3.4.2 DIJKSTRA'S ALGORITHM:

- goal : finding the shortest path from any vertex to all other vertices
- Constraints : run for weighted graphs only and non-negative edge weights

- ## 4.3.4.2.1 How it works ?
  - Given a start node , you can pick smallest edge from it by checking neighbors
  - If we want to remove a node from a weighted graph without miss any information stored in that node ,we can use what is called node relaxation where we can delete a node and replace it with number of edges equal to (NO. of incoming edges * NO. of outgoing edges ) then relax the graph (remove the heavy weighted edges).

- ## 4.3.4.2.1 PSUDOCODE of Dijkstra :

```
For each node n in the graph
n.distance = Infinity
Create an empty list.
start.distance = 0, add start to list.
While list not empty
-Let current = node in the list with the smallest distance, remove current from list
-For each node, n that is adjacent to current
If n.distance > current.distance + length of edge from n to current
n.distance = current.distance + length of edge from n to current
n.parent = current add n to list if it isn't there already
```

  - A naive version of Dijkstra's algorithm can be implemented with a computational complexity that grows quadratically with the number of nodes.
  - By keeping the list of nodes sorted using a clever data structure known as a priority queue the computational complexity can be reduced to something that grows more slowly

## 4.3.4.3 A* Algorithm:

- A* Search attempts to improve upon the performance of grassfire and Dijkstra by incorporating a heuristic function that guides the path planner
- Heuristic function H(n) , takes a node n and returns a non-negative real number that is an estimation of the path cost from node n to the goal

### 4.3.4.3.1 Heuristic Function Criteria:

- H(goal) = 0
- For any 2 adjacent nodes x and y
- H(x) <= H(y) + d(x,y)
- d(x,y) = weight/length of edge from x to y •
- These properties ensure that for all nodes, n
- H(n) <= length of shortest path from n to goal

- For path planning on a grid the following 2 heuristic functions are often used: where (xn, yn) denotes the coordinates of the node n and (xg, yg) denotes the coordinate of the goal.

### 4.3.4.3.2 Euclidean Distance:

$$H(xn, yn) = \text{sqrt} ( (xn\ xg)^2 + (yn\ yg)^2 )$$

### 4.3.4.3.3 Manhattan Distance:

$$H(xn, yn) = |(xn\ xg)| + |(yn\ yg)|$$

### 4.3.4.3.4 PSUDOCODE of A* algorithm:

- For each node n in the graph
- n.f = Infinity, n.g = Infinity
- Create an empty list
- start.g = 0 , start.f = H(start) add start to list
- While list not empty
- Let current = node in the list with the smallest f value, remove current from list
- If (current == goal node) report success
- For each node, n that is adjacent to current
- If (n.g > (current.g + cost of edge from n to current))
- n.g = current.g + cost of edge from n to current
- n.f = n.g + H(n)
- n.parent = current add n to list if it isn't there already

## ❖ 4.3.5 Planning in Configuration space:

• In the motion planning problems we have considered so far we have basically reduced the problem to planning on a graph where the robot can take on various discrete positions which we can enumerate and connect by edges.

• In practice most of the robots that we build can move continuously through space.so we need a method to define space to our robot

• Configuration space is a handy mathematical and conceptual tool which was developed to help us think about these kinds of problems in a unified framework.

definition :

the set of reachable positions robot can attain . so , the positions and orientations of a robot can be identified with the group of spatial rigid transformations (SE(n))

SE(n) = SO(n) * R^n  (n = 1,2,3,..)(n=configuration space dimension)

where (SO(n)) represents the rotation matrices that define orientation of robot frame relative to world frame and( R^n )represents the coordinate of the robot frame relative to world frame .

C.space of mobile robot = (X , Y , theta) [n= 3][SE(3)=SO(3)*R^3]

• Configuration space consist of 3 sub-systems:

-free space [Cfree] >> set of configurations that avoids collision with obstacles

-obstacle space [Cobs]>> the complement of [Cfree]

-goal and start space

• the dimensions and shape of the configuration space obstacle are obtained by considering both the obstacle and the shape of the robot.

More formally the configuration space obstacle in this case is the Minkowski sum of the obstacle .

this means that all geometry of robot and obstacles are captured by Cobs and then we just planning the path for a point .

### 4.3.5.1 PLANNING PROBLEM CAN BE SOLVED IN 3 STEPS:

- Start with planning problem framed in continuous configuration space
- Apply appropriate approaches to reformulate the planning problem in configuration space in terms of graphs . (how to convert the problem into graph?)

  **Some Approaches** :

  1) Grid based algorithms (occupancy grid)
  2) Interval based search
  3) Geometric algorithms (visibility graph , cell decomposition)
  4) Sampling based algorithms
  5) Road maps

- apply graph based search algorithms to find the optimal path

### 4.3.5.2 GEOMETRIC ALGORITHMS:

- robot presented as a point and the obstacles represented by convex polygons
- we associate a node with every configuration space obstacle vertex .
- then we define an edge between any two vertices that can be connected by a straight line that lies entirely in free space , including every pair of neighboring vertices on the same polygon
- check whether an edge between any two nodes intersects an obstacle this established by Collision_check ( ) Let x denote the coordinates of a point in configuration space. [Collision_Check (x) ]should return 0 if x is in frees pace and 1 if x results in a collision with the obstacles
- Finally, you have a graph !

#### 4.3.5.2.1 VISIBILITY GRAPH:

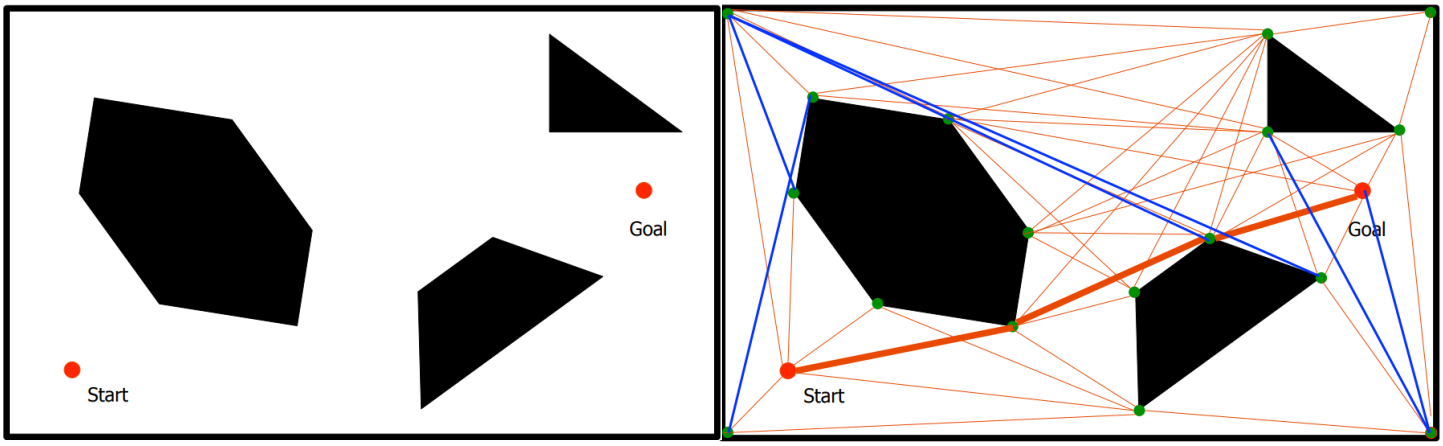



Figure 42: Visibility Graph

- ## 4.3.5.2.2 Cell Decomposition:

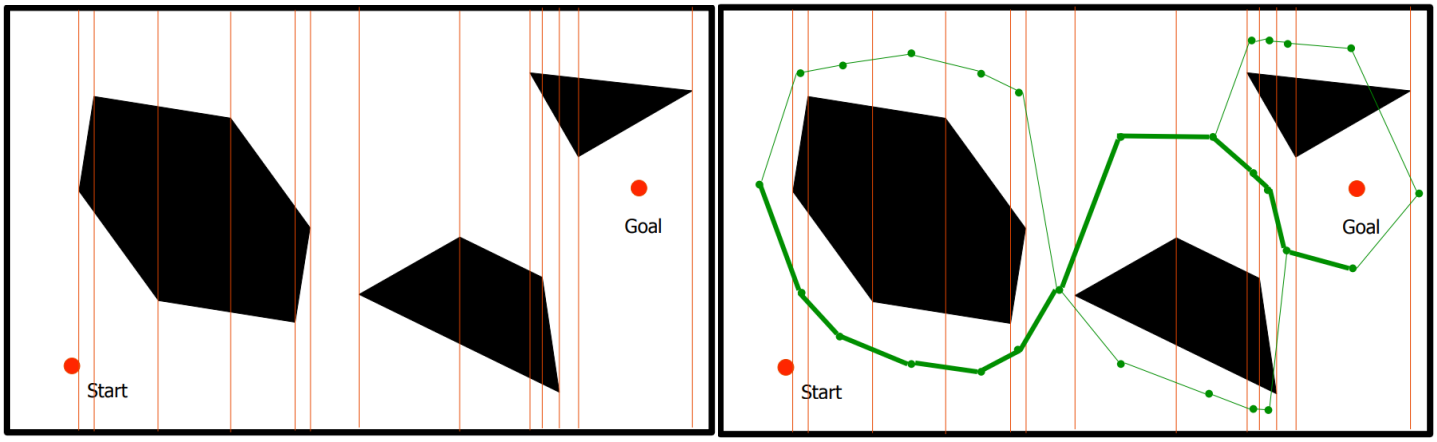



Figure 43: Cell Decomposition

### 4.3.5.3 Probabilistic Road Maps And RRT:

- Polygonal obstacles are convenient to work with because they provide an explicit description of the configuration space obstacles. Oftentimes we do not have this luxury and the obstacles are instead defined implicitly by a collision function. so we need other algorithms to overcome this problem .

### 4.3.5.4 Probablistic Road Map Pseudocode:

```
• Polygonal
• Repeat n times
• Generate a random point in configuration space,
•  If x is in free space
        • Find the k closest points in the roadmap to x according to the Dist function
        • Try to connect the new random sample to each of the k neighbors using the Local Planner
            procedure. Each successful connection forms a new edge in the graph.
```

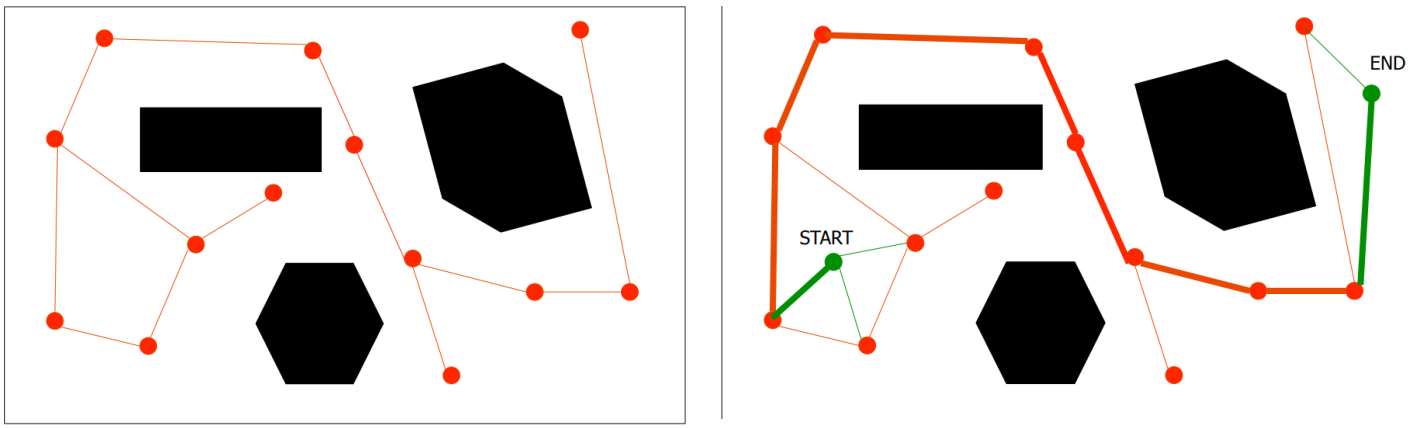



Figure 44: Probablistic Road Map

## ○ 4.3.5.5 Rapidly exploring Random Trees:

```
• Add start node to tree
• Repeat n times
• Generate a random configuration, x
• If x is in free space using the CollisionCheck() function
• Find y, the closest node in the tree to the random configuration
• If (Dist (x, y) > delta) – Check if x is too far from y
• Find a configuration, z, that is along the path from x -to- y such
• that Dist(z,y) <= delta
• x = z;
• If (LocalPlanner (x,y)) – Check if you can get from x to y
• Add x to the tree with y as its parent
```

**Note** : delta is the parameter of the algorithm which means the max step size

- In order to improve the performance we will use something called bidirectional search where , constructing 2 tress one rooted from start and the other one from goal

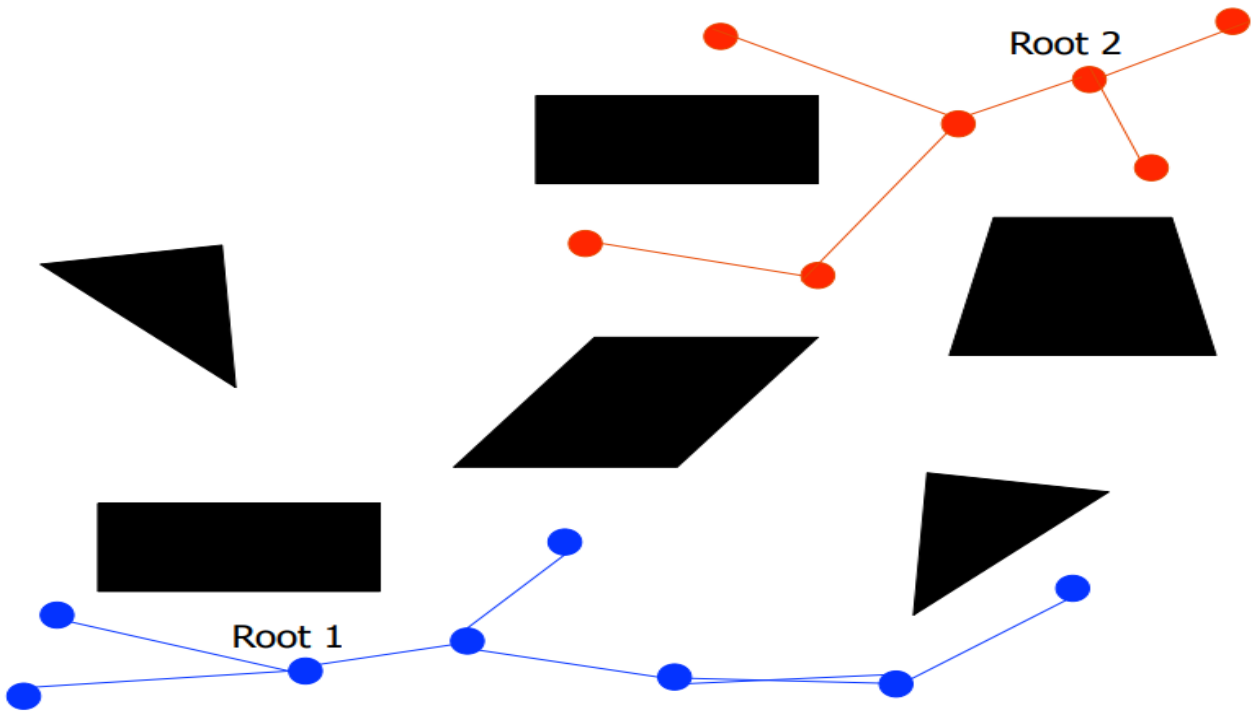



**Figure 45: Bidirectional Search**

# 4.4 Control:

A control loop is the fundamental building block of industrial control systems. It consists of all the physical components and control functions necessary to automatically adjust the value of a measured process variable (PV) to equal the value of a desired set-point (SP). It includes the process sensor, the controller function, and the final control element (FCE) which are all required for automatic control.

## ❖ 4.4.1 PID controller:

A proportional–integral–derivative controller (PID controller or three term controller) is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value e ( t ) as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively) which give the controller its name.

In practical terms it automatically applies accurate and responsive correction to a control function. An everyday example is the cruise control on a road vehicle; where external influences such as gradients would cause speed changes, and the driver has the ability to alter the desired set speed. The PID algorithm restores the actual speed to the desired speed in the optimum way, without delay or overshoot, by controlling the power output of the vehicle's engine.

The first theoretical analysis and practical application was in the field of automatic steering systems for ships, developed from the early 1920s onwards. It was then used for automatic process control in manufacturing industry, where it was widely implemented in pneumatic, and then electronic, controllers. Today there is universal use of the PID concept in applications requiring accurate and optimised automatic control.

### ○ 4.4.1.1 Fundamental operation:

In model beside:

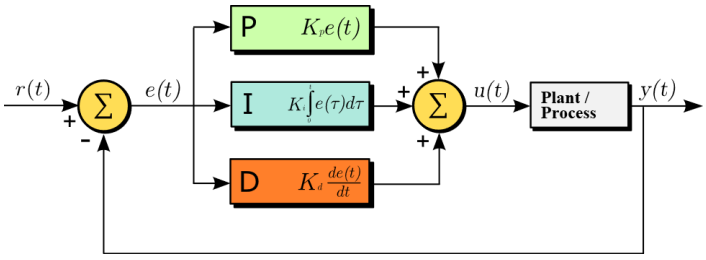



Figure 46: PID Model

- **Term P** is proportional to the current value of the SP – PV error e(t). For example, if the error is large and positive, the control output will be proportionately large and positive, taking into account the gain factor "K". Using proportional control alone in a process with compensation such as temperature control, will result in an error between the setpoint and the actual process value, because it requires an error to generate the proportional response. If there is no error, there is no corrective response.
- **Term I** accounts for past values of the SP – PV error and integrates them over time to produce the I term. For example, if there is a residual SP – PV error after the application of proportional

control, the integral term seeks to eliminate the residual error by adding a control effect due to the historic cumulative value of the error. When the error is eliminated, the integral term will cease to grow. This will result in the proportional effect diminishing as the error decreases, but this is compensated for by the growing integral effect.

- **Term D** is a best estimate of the future trend of the SP – PV error, based on its current rate of change. It is sometimes called "anticipatory control", as it is effectively seeking to reduce the effect of the SP – PV error by exerting a control influence generated by the rate of error change. The more rapid the change, the greater the controlling or dampening effect.

**Tuning –** The balance of these effects is achieved by "loop tuning" (see later) to produce the optimal control function. The tuning constants are shown below as "K" and must be derived for each control application, as they depend on the response characteristics of the complete loop external to the controller. These are dependent on the behaviour of the measuring sensor, the final control element (such as a control valve), any control signal delays and the process itself. Approximate values of constants can usually be initially entered knowing the type of application, but they are normally refined, or tuned, by "bumping" the process in practice by introducing a setpoint change and observing the system response.

**Control action –** The mathematical model and practical loop above both use a "direct" control action for all the terms, which means an increasing positive error results in an increasing positive control output for the summed terms to apply correction. However, the output is called "reverse" acting if it is necessary to apply negative corrective action. For instance, if the valve in the flow loop was 100–0% valve opening for 0–100% control output – meaning that the controller action has to be reversed. Some process control schemes and final control elements require this reverse action. An example would be a valve for cooling water, where the fail-safe mode, in the case of loss of signal, would be 100% opening of the valve; therefore 0% controller output needs to cause 100% valve opening.

## ○ 4.4.1.2 Mathematical form:

The overall control function can be expressed mathematically as

$$u(t) = K_{\mathrm{p}} e(t) + K_{\mathrm{i}} \int_0^t e(t')\,dt' + K_{\mathrm{d}} \frac{de(t)}{dt},$$

where $K_p$, $K_i$ and $K_d$ all non-negative, denote the coefficients for the proportional, integral, and derivative terms respectively (sometimes denoted P, I, and D).

## ❖ 4.4.2 PID Tunning:

For our PID tunning we used Matlab and Simulink, and here is our Simulink blocks and project:

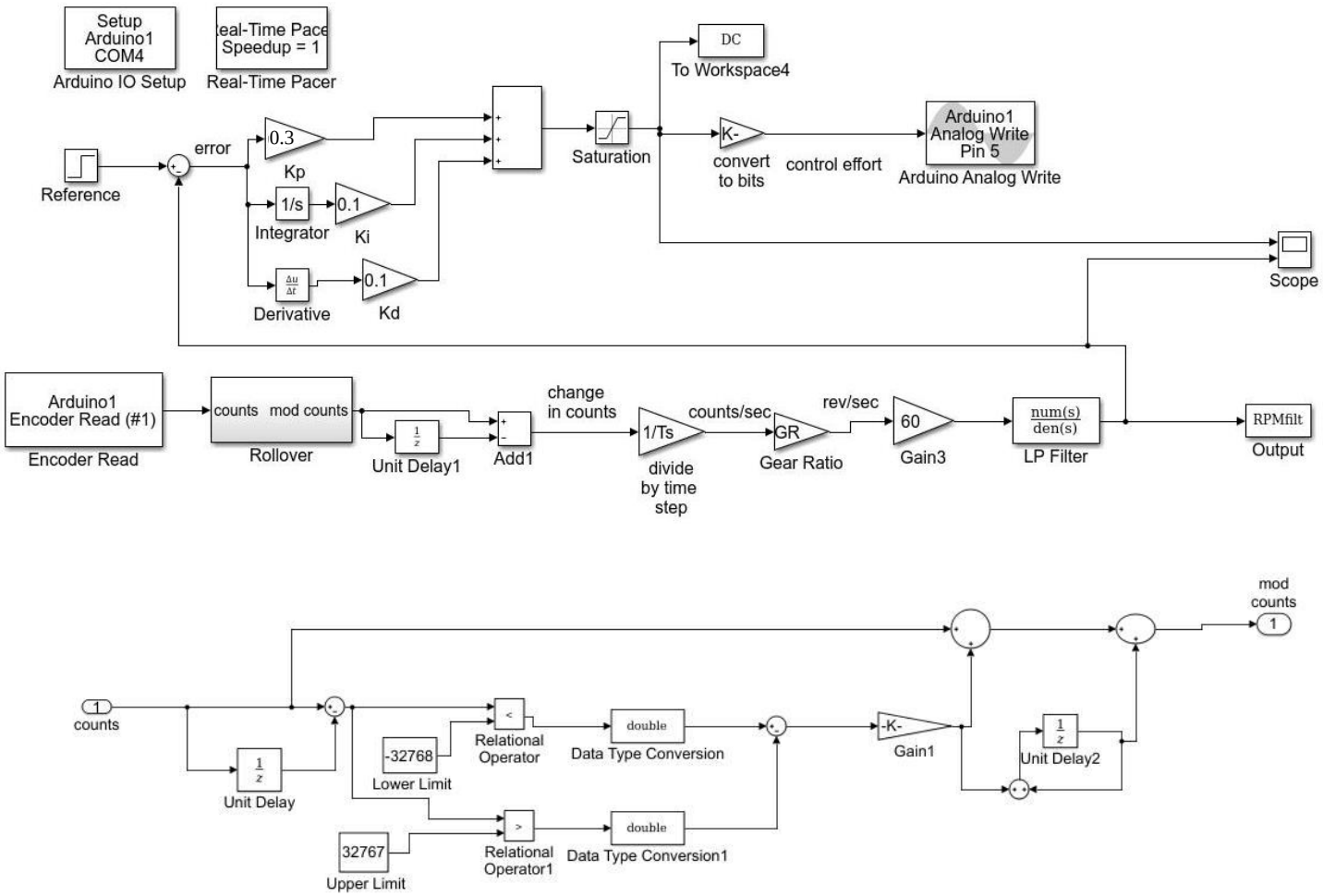



**Figure 47: PID Simulink Blocks**

As we tested we found out, The rollover effect problem existance, which means that the encoder reaches saturation after a number of cycles, this is because the Arduino's interrupt buffer is 16 bit only, which means it can only store integars with values between 32767 and -32767.

We used a low pass filter to remove the noise from our readings and must be filtered, non the less this procedure delays the signal, leading to a lag in the control process, to solve this a small time constant for the filter is used.

Using this procedure above a control values of $\mathbf{K_p = 0.3}$, $\mathbf{K_I = 0.1}$ and $\mathbf{K_d = 0.1}$, were the best performance proving values for our robot.

# 4.5 SLAM:

In robotic mapping and navigation, simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. While this initially appears to be a chicken-and-egg problem there are several algorithms known for solving it, at least approximately, in tractable time for certain environments. Popular approximate solution methods include the particle filter, extended Kalman filter, and GraphSLAM.

SLAM algorithms are tailored to the available resources, hence not aimed at perfection, but at operational compliance. Published approaches are employed in self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers, newer domestic robots and even inside the human body.

## ❖ 4.5.1 Problem definition:

Given a series of sensor observations $O_t$ over discrete time steps $t$, the SLAM problem is to compute an estimate of the agent's location $x_t$ and a map of the environment $m_t$. All quantities are usually probabilistic, so the objective is to compute:

$$P(m_t, x_t | o_{1:t})$$

Applying Bayes' rule gives a framework for sequentially updating the location posteriors, given a map and a transition function $P(x_t|x_{t-1})$.

$$P(x_t|o_{1:t}, m_t) = \sum_{m_{t-1}} P(o_t|x_t, m_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}|m_t, o_{1:t-1})/Z$$

Similarly the map can be updated sequentially by:

$$P(m_t|x_t, o_{1:t}) = \sum_{x_t} \sum_{m_t} P(m_t|x_t, m_{t-1}, o_t) P(m_{t-1}, x_t|o_{1:t-1}, m_{t-1})$$

Like many inference problems, the solutions to inferring the two variables together can be found, to a local optimum solution, by alternating updates of the two beliefs in a form of EM algorithm.

## ❖ 4.5.2 Algorithms:

Statistical techniques used to approximate the above equations include Kalman filters and particle filters (aka. Monte Carlo methods). They provide an estimation of the posterior probability function for the pose of the robot and for the parameters of the map. Set-membership techniques are mainly based on interval constraint propagation. They provide a set which encloses the pose of the robot and a set approximation of the map. Bundle adjustment, and more generally Maximum a posteriori estimation (MAP), is another popular technique for SLAM using image data, which jointly estimates poses and landmark positions, increasing map fidelity, and is used in commercialized SLAM systems such as Google's arCore which replacing their previous augmented reality project 'tango'. MAP estimators compute the most likely explanation of the robot poses and the map given the sensor data, rather than trying to estimate the entire posterior probability.

# 4.6 Autonomous Navigation:

Path planning and autonomous navigation are some of the most important challenges in mobile robotics. These are difficult tasks because the robot has to accurately and safely perform autonomous maneuverings. This paper presents a methodology to efficiently plan the trajectory of a robot in dynamic and complex environments, which it should traverse autonomously. A planner based in the AD* algorithm is used to plan a less costly trajectory to the destination. The methodology enables the robot to reach the goal using a local planner, which is applied to determine the speed and steering of the robot. Experimental results show that the planner is reliable for simulated and real tasks, as the robot has reached the goal while safely avoided obstacles.

Using the above mentioned Path planning algorithms, and after implementing them we were able to constract an autonomous movement robot.

# Chapter Five

## Software Requirements And Implementation

# 5.1 Software Requirements:

| No. | Software/Lib. Name | Usage |
|---|---|---|
| **1** | Solidworks | Designing The Robot |
| **2** | Ubuntu | LINUX opertaing system |
| **3** | ROS | Main platform for controlling the robot and generate maps |
| **4** | Rospy library | Allow using python on our ros packages |
| **5** | Freenect library | Allows working and gaining data from kinect |
| **6** | Turtlebot Package | To start working with robots on ros |
| **7** | Gazebo | Simulate a 3D dynamic environment |
| **8** | Rviz | Visualizing and logging sensors informations |
| **9** | Matlab & Simulink | To obtain pid constrains |
| **10** | DialogFlow | Natural language processor and AI chatting creation |
| **11** | Firebase | Cloud Development Platform |
| **12** | Google Assistant | Importing our AI on it, allowing voice and speech recognition |
| **13** | Google Assistant Package | Importing the google assistant project on ubuntu |
| **14** | Arduino IDE | Uploading our teensy firmware and NodeMCU Codes |
| **15** | Real Time Database | Cloud database to help implementing IOT |

# 5.2 INSTALLATION OF UBUNTU:

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

## 1- BOOT FROM USB FLASH DRIVE:

We used Rufus With a GPT Burning Algorithm to insure the correct installation of the Ubuntu OS.

Most computers will boot from USB automatically. Simply insert the USB flash drive and either power on your computer or restart it. You should see the same welcome window we saw in the previous 'Install from DVD' step, prompting you to choose your language and either install or try the Ubuntu desktop.

If your computer doesn't automatically boot from USB, try holding F12 when your computer first starts. With most machines, this will allow you to select the USB device from a system-specific boot menu.

## 2- PREPARE TO INSTALL UBUNTU:

After choosing to install Ubuntu from the welcome window, you will be asked about updates and third-party software.

We advise enabling both Download updates and Install third-party software.

Stay connected to the internet so you can get the latest updates while you install Ubuntu.

If you are not connected to the internet, you will be asked to select a wireless network, if available. We advise you to connect during the installation so we can ensure your machine is up to date

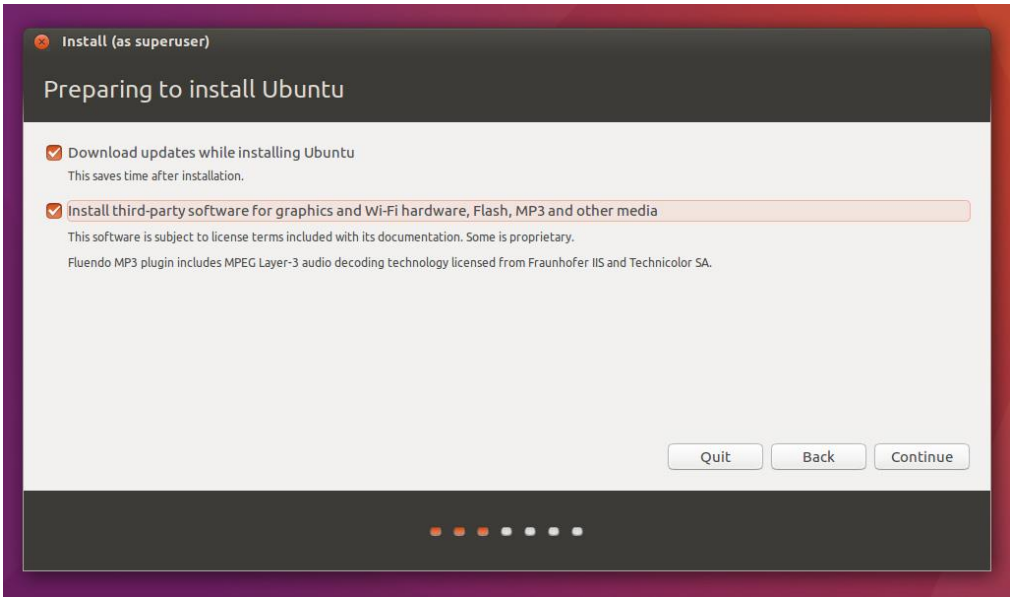



**Figure 48: Installing Ubuntu**

# 3. Allocate drive space:

Use the checkboxes to choose whether you'd like to Install Ubuntu alongside another operating system, delete your existing operating system and replace it with Ubuntu, or — if you're an advanced user — choose the 'Something else' option.

1. Start the installation. Proceed to Step 4 and choose "Something else":
2. You will see your disk as `/dev/sda` or `/dev/mapper/pdc_*` (RAID case, * means that your letters are different from ours)Click "New Partition Table..." You will see that you have free space on your disk now:
3. (**Recommended**) Create partition for swap. Swap is the partition for keeping unneeded memory pages, like Windows swap. Also it can be used for hibernation.

   - Select free space and click +
   - Set parameters like on the picture below:

   Notice that you should set swap size more than you have physical memory in order to use hibernation. Also, you can place it in the end of disk, but thus it will be slow.

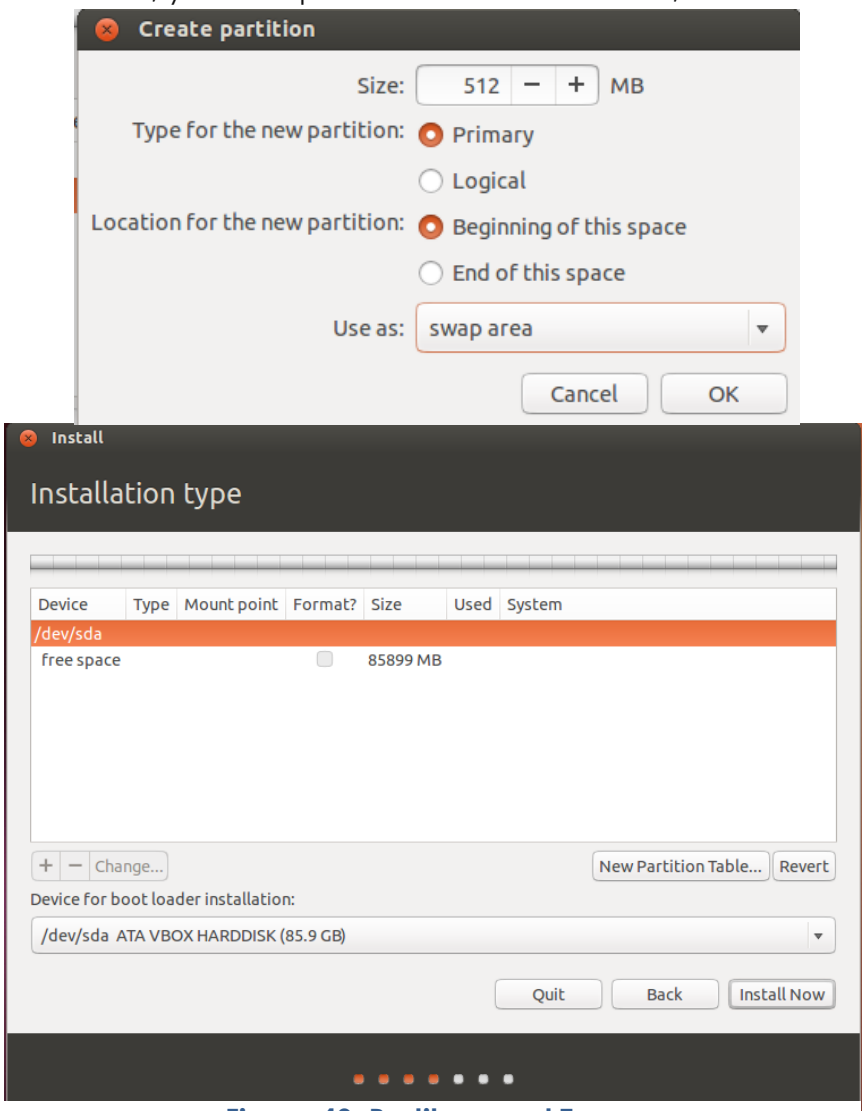



**Figure 49: Partitons and Types**

4. Create partition for `/` (root fs). This is the filesystem that contains your kernel, boot files, system files, command-line utilities, libraries, system-wide configuration files and logs.
   - Select free space and click `+`
   - Set parameters like on the picture below:

   10 – 20 GiB should be enough

5. Create partition for `/home`. This is the filesystem for your user's files: documents, images, music and videos. It's much more like Users folder in Windows.
   You can do this just like in step 5 and even choose other fs type (though I recommend use ext4 instead of reiserfs. Simply, the first is much more flexible and the second is quicker)

6. (**Optional**) Create separate partitions for `/boot`, `/tmp` and `/var`. Set their size according to your needs:
   - `/boot` should be 100 – 500 MiB
   - `/var` and `/tmp` should be > 5 GiB

7. If you doubt about which device for boot loader installation to choose, leave it default. It would be set by installer. But sometimes it does mistakes. Let me guide you how to deal with it:

If you use only one hard disk, select or leave /dev/sda intact. If you use more than one hard disk with no RAID, select the one from which your system does boot. You can also select other disk and set BIOS to boot from it. If you have RAID from which your system starts, it will be /dev/mapper/...

After all, you should see your disk like this:

That's all! You can now click Install Now and proceed to the installation.

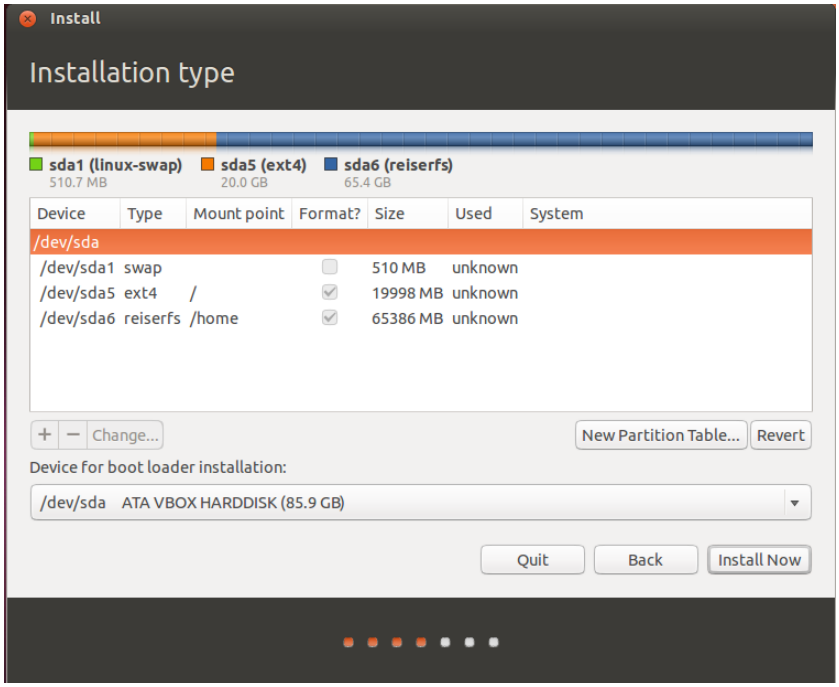



**Figure 50: Final Partitions**

# 5.3 INSTALLATION OF UBUNTU MATE:

Martin Wimpress and Rohith Madhavan have made an Ubuntu MATE image for the Raspberry Pi 2 and Raspberry Pi 3 based on the regular Ubuntu armhf base, not the new Ubuntu "Snappy" Core, which means that the installation procedure for applications uses the traditional tools, ie apt-get.

## 1- MAKING A MICROSDHC WITH WINDOWS:

To make a microSDHC using Windows we used:

- **7-Zip** to extract the image.
- **Win32 Disk Imager** to write the image.

## 2- USEFULL COMMANDS:

### RE-SIZE FILE SYSTEM:

Since Ubuntu MATE 16.04.2 the root parition is automatically resized, to fully utilise the all available space on the microSD card, on first boot.

### SSH:

Since Ubuntu MATE 16.04.2 the OpenSSH server is disabled by default. If you want to enable SSH you can use raspi-config to created a file call ssh in to /boot paritition and reboot.

When you enable SSH via either method explained above sshguard will also be enabled.

### ENABLE AND DISABLE X11:

Since Ubuntu MATE 16.04.2 your can disbale/enable the desktop environment using raspi-config.

### REDIRECTING AUDIO OUTPUT:

The sound will output to HDMI by default if both HDMI and the 3.5mm audio jack are connected. You can, however, force the system to output to a particular device using raspi-config.

For those of you who want to know how to do this without raspi-config: For 3.5mm audio jack:

```
sudo amixer cset numid=3 1
```

# 5.4 Installation of ROS Kinetic:

ROS Kinetic ONLY supports Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) and Jessie (Debian 8) for debian packages.

## 1- Configure Ubuntu repositories:

Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse." You can follow the Ubuntu guide for instructions on doing this.

## 2- Setup your sources.list:

Setup your computer to accept software from packages.ros.org.

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

## 3- Set up the keys:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-ke
y 421C365BD9FF1F717815A3895523BAEEB01FA116
```

## 4- Installation:

First, make sure your Debian package index is up-to-date:

```
sudo apt-get update
```

There are many different libraries and tools in ROS. We took the **Desktop-Full Install Pack** which includes almost everything.

**Desktop-Full Install**: Includes ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception

```
sudo apt-get install ros-kinetic-desktop-full
```

# 5- Environment setup:

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched, so you won't have to add them each time you start the device.

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

source ~/.bashrc.
```

If you just want to change the environment of your current shell, instead of the above you can type:

```
source /opt/ros/kinetic/setup.bash
```

If you use zsh instead of bash you need to run the following commands to set up your shell:

```
echo "source /opt/ros/kinetic/setup.zsh" >> ~/.zshrc

source ~/.zshrc
```

# 5.5 TurtelBot/Python installations:

## ❖ 5.5.1 Installation Process:

we were really having trouble getting this to work, FINALLY found out how to get turtlebot to install cleanly on kinetic!

Here's our route to the solution:

First hint came from doing a wildcard turtlebot install attempt

```
sudo apt-get install ros-kinetic-turtlebot*

The following packages have unmet dependencies:

 ros-kinetic-turtlebot-bringup : Depends: ros-kinetic-realsense-camera but
it is not going to be installed
```

Going further down, we tried

```
sudo apt-get install ros-kinetic-realsense-camera
```

Got this:

```
The following packages have unmet dependencies:

ros-kinetic-realsense-camera : Depends: ros-kinetic-librealsense but it is
not going to be installed
```

Next, you guessed it, we tried:

```
sudo apt-get install ros-kinetic-librealsense
```

Surprise, surprise...

```
The following packages have unmet dependencies:

ros-kinetic-librealsense : Depends: linux-headers-generic but it is not go
ing to be installed
```

Finally, one from the distro that will work!

```
sudo apt-get install linux-headers-generic
```

BUT! It's not THAT easy... You also need to enable the source repositories, since installing ros-kinetic-librealsense incurs a compilation which requires your kernel sources, so from the ROS docs, we can see how to do this, or you can copy me

```
sudo sh -c 'echo "deb-src http://us.archive.ubuntu.com/ubuntu/ xenial main
restricted

deb-src http://us.archive.ubuntu.com/ubuntu/ xenial-updates main restricte
d

deb-src http://us.archive.ubuntu.com/ubuntu/ xenial-backports main restric
ted universe multiverse

deb-src http://security.ubuntu.com/ubuntu xenial-security main restricted"
> \

  /etc/apt/sources.list.d/official-source-repositories.list'

sudo apt-get update
```

Now, magic happens! (assuming you have typical compilers installed)

(Optional)

```
sudo apt-get install build-essential g++
```

Let's try this again

```
sudo apt-get install ros-kinetic-librealsense
```

It should build, you may see some warnings, but as long as it completes OK, you can continue back up out of the rabbit hole

```
sudo apt-get install ros-kinetic-realsense-camera

sudo apt-get install ros-kinetic-turtlebot
```

Here is The Fast Version:

```
>> sudo apt-get install -y linux-headers-generic

>> sudo sh -c 'echo "deb-src http://us.archive.ubuntu.com/ubuntu/ xenial m
ain restricted

deb-src http://us.archive.ubuntu.com/ubuntu/ xenial-updates main restricte
d
```

```
deb-src http://us.archive.ubuntu.com/ubuntu/ xenial-backports main restric
ted universe multiverse

deb-src http://security.ubuntu.com/ubuntu xenial-security main restricted"
> \

  /etc/apt/sources.list.d/official-source-repositories.list'

>> sudo apt-get update

>> sudo apt-get install -y ros-kinetic-librealsense

>> sudo apt-get install -y ros-kinetic-librealsense-camera

>> sudo apt-get install -y ros-kinetic-turtlebot
```

# 5.6 Some other Problems:

## ❖ 5.6.1 Kobuki does not start:

If you receive warnings of the type:

```
[ WARN] [1426225321.364907925]: Kobuki : device does not (yet) available,
is the usb connected?.

[ WARN] [1426225321.615097034]: Kobuki : no data stream, is kobuki turned
on?
```

Then do the obvious - make sure kobuki is turned on (leds should be active on the kobuki) and the cable is plugged in. If you are sure about both of these, check to see that your system has had the udev rule applied for /dev/kobuki. Check for this:

```
> ls -n /dev | grep kobuki
```

## ❖ 5.6.2 Initializing some environment variables:

The create base is not the default configuration for an indigo turtlebot. To ensure that the correct hardware configuration is loaded, you will need to export a few environment variables along with your setup.bash.

```
> export TURTLEBOT_BASE=create

> export TURTLEBOT_STACKS=circles

> export TURTLEBOT_3D_SENSOR=asus_xtion_pro

> export TURTLEBOT_SERIAL_PORT=/dev/ttyUSB0
```

If the turtlebot is your only workspace, adding the above exports to your .bashrc is a useful convenience. Alternatively you can append them to `devel/setup.bash` or even better create your own `setup.bash` in your workspace root so that it persists even when you clean out your build. For example, ~/turtlebot/setup.bash

## Serial Device configiure:

Typically the /dev/ttyUSBx port requires your user to be in the dialout group. If you are not in this group then your turtlebot bring up will fail with permission denied. Add your user to this group with:

```
> sudo adduser <username> dialout
```

# Chapter Six

# Robotics Operating System

# 6.1 Introduction To ROS:

## ❖6.1.1 What is ROS?:

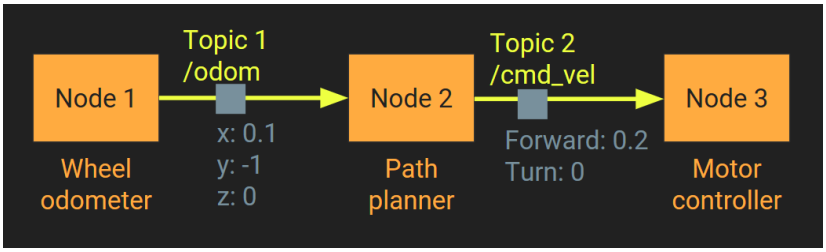
ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

- A software framework for programming robots
- Prototypes originated from Stanford AI research, officially created and developed by Willow Garage starting in 2007
- Currently maintained by Open Source Robotics Foundation
- Consists of infrastructure, tools, capabilities, and ecosystem

| Distro | Release date | Poster | EOL date |
|---|---|---|---|
| Lunar Loggerhead | May 23, 2017 |  | **May, 2019** |
| Kinetic Kame | May 23, 2016 |  | **2021-05-30** |
| Jade Turtle | May 23, 2015 |  | 2017-05-30 |
| Indigo Igloo | July 22, 2014 |  | 2019-04-30 |
| Hydro Medusa | September 4, 2013 |  | 2014-05-31 |
| Groovy Galapagos | December 31, 2012 |  | 2014-07-31 |
| Fuerte Turtle | April 23, 2012 |  | -- |
| Electric Emys | August 30, 2011 |  | -- |
| Diamondback | March 2, 2011 |  | -- |
| C Turtle | August 2, 2010 |  | -- |
| Box Turtle | March 2, 2010 |   | -- |

**Figure 51: ROS Versions**

## ❖ 6.1.2 ROS COMPUTATION GRAPH:




## ❖ 6.1.3 PUBLISHING AND SUBSCRIBING:

- Multiple nodes can publish to the same topic
- Multiple nodes can subscribe to the same topic
- A node can publish to multiple topics
- A node can subscribe to multiple topics

## ❖ 6.1.4 PACKAGES:

- Packages form the atomic level of ROS. A package has the minimum structure and content to create a program within ROS. It may
- Programs in Linux are called packages

## ❖ 6.1.5 NODES:

- Nodes are executables that can communicate with other processes using topics, services, or the Parameter Server. Using nodes in ROS provides us with fault tolerance and separates the code and functionalities making the system simpler.
- A node must have a unique name in the system. This name is used to permit the node to communicate with another node using its name without ambiguity. A node can be written using different libraries such as roscpp and rospy; roscpp is for C++ and rospy is for Python.

## ❖ 6.1.6 TOPICS:

- Topics are buses used by nodes to transmit data. Topics can be transmitted without a direct connection between nodes, meaning the production and consumption of data are decoupled. A topic can have various subscribers.
- Each topic is strongly typed by the ROS message type used to publish it, and nodes can only receive messages from a matching type. A node can subscribe to a topic only if it has the same message type

## ❖ 6.1.7 MESSAGES:

- A serialization format for structured data
- Allows nodes written in C++ and Python to communicate with each other
- Defined in a .msg file
- Must be compiled into C++ / Python classes before using them

# 6.2 Ubuntu Linux:

## ❖ 6.2.1 Bash Shell:

- Bash shell is a UNIX shell and a command processor
- It's the default command language for Linux consoles
- It's an interface between the user and the OS

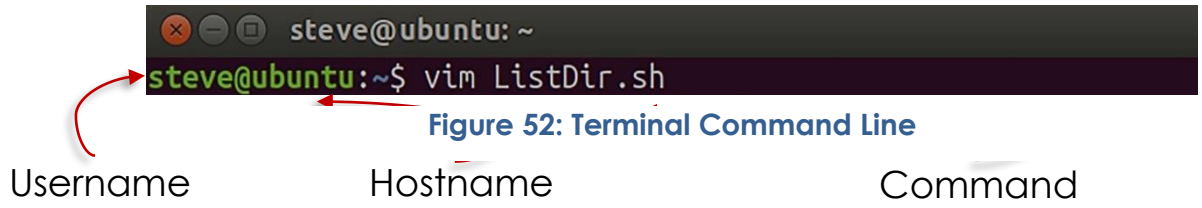



Figure 52: Terminal Command Line

## ❖ 6.2.2 Useful Command-Line Tools:

- `ls` for listing files
- `cd` for changing directories
- `pwd` for current path
- `mkdir` for creating directories
- `man` for displaying manuals of other commands
- `cp` for copying files
- `mv` for moving and renaming files
- `cat` for showing file contents
- `nano` text editor for creating and editing scripts and text files
- `sudo` for administratorpriviliges
- `help` for help menu

## ❖ 6.2.3 Package Management System:

- Programs in Linux are called packages
- Packages are installed using a package management system
- Official package management system in Ubuntu is apt-get
- Example use: `sudo apt-get install google-chromium`
    - `sudo` => because installing packages requires modifying system files and thus requires system admin privileges
    - `apt-get` => command-line tool for managing packages: installing & removing ..etc
    - `install` => option to install a package –alternatives? `remove` & `purge`
    - `google-chromium` => the name of the package we want to install, here it's the browser google chromium

# 6.3 Robotics Operating System:

## ❖ 6.3.1 ROS Filesystem Level:

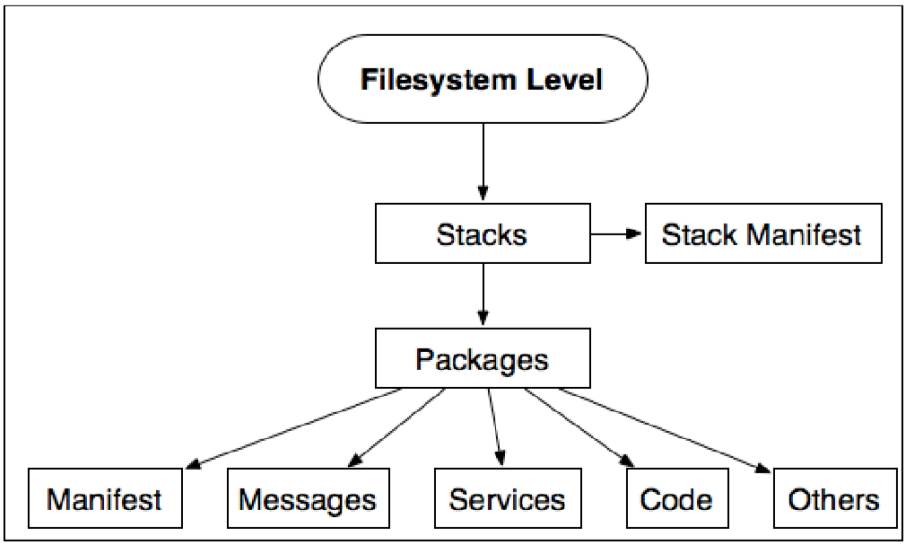



Figure 53: ROS Levels

- **Packages:** Packages form the atomic level of ROS. A package has the minimum structure and content to create a program within ROS. It may have ROS runtime processors(Nodes), configuration files and so on.
- **Manifests**: Manifests provide information about a package, license information, dependencies, compiler flags and so on. Manifests within a file called `manifiest.xml`.
- **Stacks**: When you gather several packages with some functionality, you will obtain a stack. In ROS, there exists a lot of these stacks with different uses, for example, the navigation stack.
- **Stack manifests:** Stack manifests (`stack.xml`) provide data about a stack, including its license information and its dependencies on other stacks
- **Message (msg) types:** A message is the information that a process sends to other processes. ROS has a lot of standard types of messages. Message descriptions are stored in `my_package/msg/MyMessageType.msg`.
- **Service (srv) types:** Service descriptions, stored in `my_package/srv/MyServiceType.srv`, define the request and response data stractures for services in ROS.

## ❖ 6.3.2 Stacks:

Packages in ROS are organized into ROS stacks. While the goal of packages is to create minimal collections of code for easy re-use, the goal of stacks is to simplify the process of code sharing.

A stack needs a basic stracture of files and folders. You can create it manually but ROS provides us with the command tool roscreate-stack for this process.

The following files are necessary for a stack: CMakeList.txt, Makefile, and stack.xml. If you see stack.xml in a folder, you can be sure that this is a stack.

## ❖ 6.3.3 Packages:

Usually, when we talk about packages, we refer to a typical structure of files and folders. This structure looks as follows:

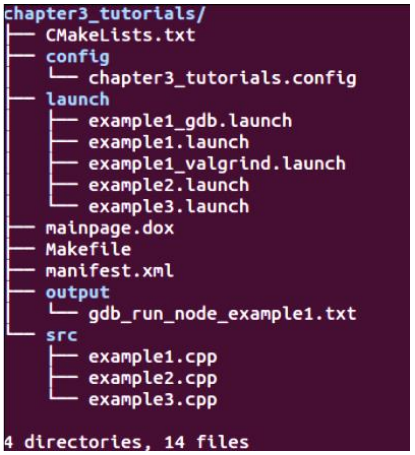
- `bin/`: This is the folder where our compiled and linked programs are stored after building/making them.
- `include/package_name/`: This directory includes the headers of libraries that you would need. Do not forget to export the manifest, since they are meant to be provided for other packages.
- `msg/`: If you develop nonstandard messages, put them here.
- `scripts/`: These are executable scripts that can be Bash, Python, or an other script.
- `src/`: This is where the source of your programs are present. You may create a folder or nodes and nodelests, or organize it as you want.
- `srv/`: This represents the service (srv) types. as you want.
- `CMakeLists.txt`: This is the CMake build files.
- `manifest.xml`: This is the package manifest file.

To create, modify, or work with packages, ROS gives us some tools for assistance:

- `rospack`: This command is used to get information or find packges in system.
- `roscreate-pkg [package_name] [depend1] [depend2] [depend3]`: When you want to create a new package, use this command
- `rosmake`: This command is used to compile a package
- `rosdep`: This command installs system dependencies of a package
- `rxdeps`: This command is used if you want to see the package dependencies as a graph

To move between packages and their folders and files, ROS gives us a very useful package called rosbash, which provides some commands.The following are a few examples:

- `roscd`: helps to change the directory; this is similar to the cd command in Linux
- `rosed`: This command is used to edit a file.
- `roscp`: This command is used to copy a file from package.
- `rosd`: This command lists the directories of a package.
- `rosls`: This command lists the files from a package, this is similar to ls command in Linux.




## ❖ 6.3.4 Messages:

A message must have two principle parts: fields and constants, Fields define the type of data to be transmitted in the message, for example, int32, float32, and string, or new types that you have created before, such as type1 and type2. Constants define the name of the field. An example of an msg file as follows:

```
int32 id
```

```
float32 vel
string name
```

In ROS you can find a lot of standard types to use in messages as shown in the following table:

| Primitive type | Serialization | C++ | Python |
|---|---|---|---|
| bool | Unsigned 8-bit int | uint8_t | bool |
| int8 | Signed 8-bit int | int8_t | int |
| uint8 | Unsigned 8-bit int | uint8_t | int |
| int16 | Signed 16-bit int | int16_t | int |
| uint16 | Unsigned 16-bit int | uint16_t | int |
| int32 | Signed 32-bit int | int32_t | int |
| uint32 | Unsigned 32-bit int | uint32_t | int |
| int64 | Signed 64-bit int | int64_t | long |
| uint64 | Unsigned 64-bit int | uint64_t | long |
| float32 | 32-bit IEEE float | float | float |
| float64 | 64-bit IEEE float | double | float |
| string | ASCII string (4-bit) | std::string | string |
| time | Secs/nsecs signed 32-bit ints | ros::Time | rospy.Time |
| duration | Secs/nsecs signed 32-bit ints | ros::Duration | rospy.Duration |

**Figure 54: Syntax For C++ And Python**

- The accepted parameters are as follows:
- `rosmsg show`: This displays the field of the message.
- `rosmsg list`: This lists all the messages
- `rosmsg package`: This lists all the messages in a package
- `rosmsg packages`: This lists all packages that have the message
- `rosmsg users`: This searchesIRUFRGHÀOHVWKDWXVHWKHPHVVDJHW\SH
- `rosmsg md5`: This displays the MD5 sum of a message

## ❖ 6.3.5 Nodes:

Nodes are executables that can communicate with other processes using topics, services, or the Parameter Server. Using nodes in ROS provides us with fault tolerance and separates the code and functionalities making the system simpler.

- `rosnode info node`: This prints information about the node
- `rosnode kill node`: This kills a running node or sends a given signal

- `rosnode list`: This lists the active nodes
- `rosnode machine hostname`: This lists the nodes running on a particular machine or lists the machines
- `rosnode ping node`: This tests the connectivity to the node
- `rosnode cleanup`: This purges registration information from unreachable nodes.

## ❖ 6.3.6 Topics:

Topics are buses used by nodes to transmit data. Topics can be transmitted without a direct connection between nodes, meaning the production and consumption of data are decoupled. A topic can have various subscribers.

Each topic is strongly typed by the ROS message type used to publish it, and node scan only receive messages from a matching type. A node can subscribe to a topic only if it has the same message type.

The topics in ROS can be transmitted using TCP/IP and UDP. The TCP/IP-based transport is known as TCPROS and uses the persistent TCP/IP connection. This is the default transport used in ROS.

The UDP-based transport is known as UDPROS and is a low-latency, lossy transport. So, it is best suited for tasks such as teleoperation.

ROS has a tool to work with topics called rostopic. It is a command-line tool that gives us information about the topic or publishes data directly on the network. This tool has the following parameters:

- `rostopic bw /topic`: This displays the bandwidth used by the topic.
- `rostopic echo /topic`: This prints messages to the screen.
- `rostopic find message_type` : This finds topics by their type.
- `rostopic hz /topic`: This displays the publishing rate of the topic.
- `rostopic info /topic`: This prints information about the active topic, the topics published, the ones it is subscribed to, and services.
- `rostopic list`: This prints information about active topics.
- `rostopic pub /topic type args`: This publishes data to the topic. It allows us to create and publish data in whatever topic we want, directly from the command line.
- `rostopic type /topic`: This prints the topic type, that is, the type of message it publishes

## ❖ 6.3.7 Building a ROS Package:

Once you have your package created, and you have some code, it is necessary to build the package. When you build the package, what really happens is that the code is compiled.

To build a package, we will use the rosmake tool as follows:

- `rosmake [package-name]`

# ❖ 6.3.8 ROS build-tool catkin:

- **What is a build-system?** A build system is responsible for generating 'targets' from raw source code. These targets may be in the form of libraries, executable programs or anything that is not static code
- **How to create a catkin workspace?** A catkin workspace is the main directory in which ROS packages are created and built. Creating a catkin workspace is done by invoking the command catkin_init_workspace inside the src directory of the main workspace directory
- Creating and building ROS packages using catkin command-line tools A catkin package is created by invoking the command catkin_create_pkg inside the src directory

## • 6.3.8.1 Install & Create Catkin Workspace:

- Install catkin using apt-get

```
sudo apt-get install ros-kinetic-catkin
```

- Let's create and build a catkin workspace:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
$ echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/kinetic/share
```

## • 6.3.8.2 Catkin Create Package:

```
catkin_create_pkg [package-name] [depend1] [depend2]
```

## • 6.3.8.3 Catkin Build Package:

```
catkin_make [package-name]
```

## ❖ 6.3.9 What is the ROS master?

- A server that tracks the network addresses of all other nodes
- Also tracks other information like parameters
- Informs subscribers about nodes publishing on the same topic
- Publisher and subscriber establish a peer-to-peer connection
- Nodes must know network address of master on startup `(ROS_MASTER_URI)`
- Can be started with `roscore` or `roslaunch`

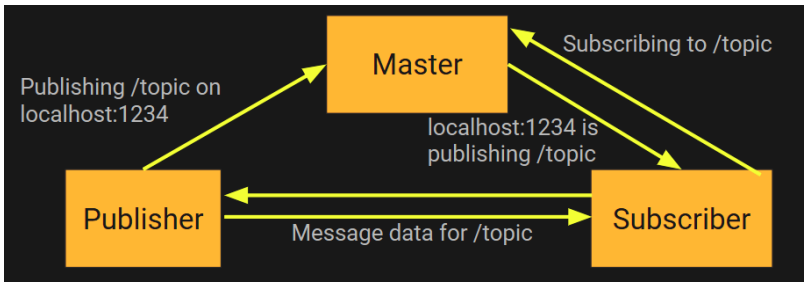



**Figure 55: Ros System**

# 6.4 Gazebo:

## ❖6.4.1 What is Gazebo?

Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. While similar to game engines, Gazebo offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs.

Typical uses of Gazebo include:

- testing robotics algorithms
- designing robots
- performing regression testing with realistic scenarios

A few key features of Gazebo include:

- multiple physics engines,
- a rich library of robot models and environments,
- a wide variety of sensors,
- convenient programmatic and graphical interfaces

## ❖6.4.2 The GUI:

This is what you should see:

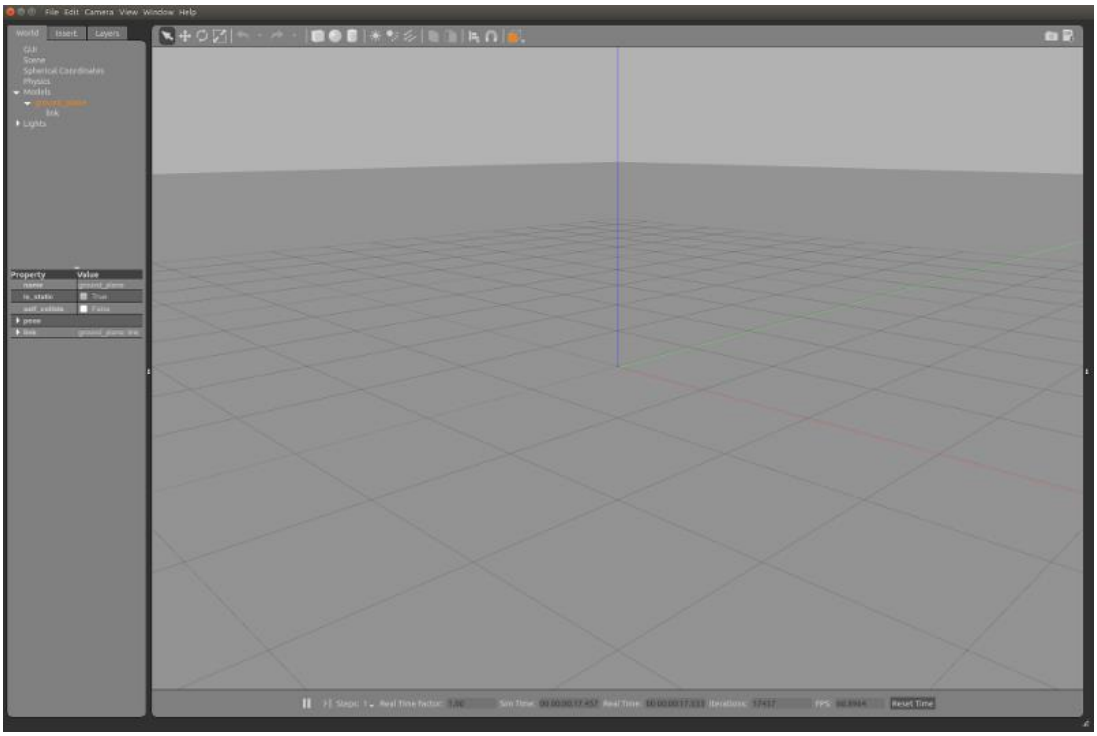
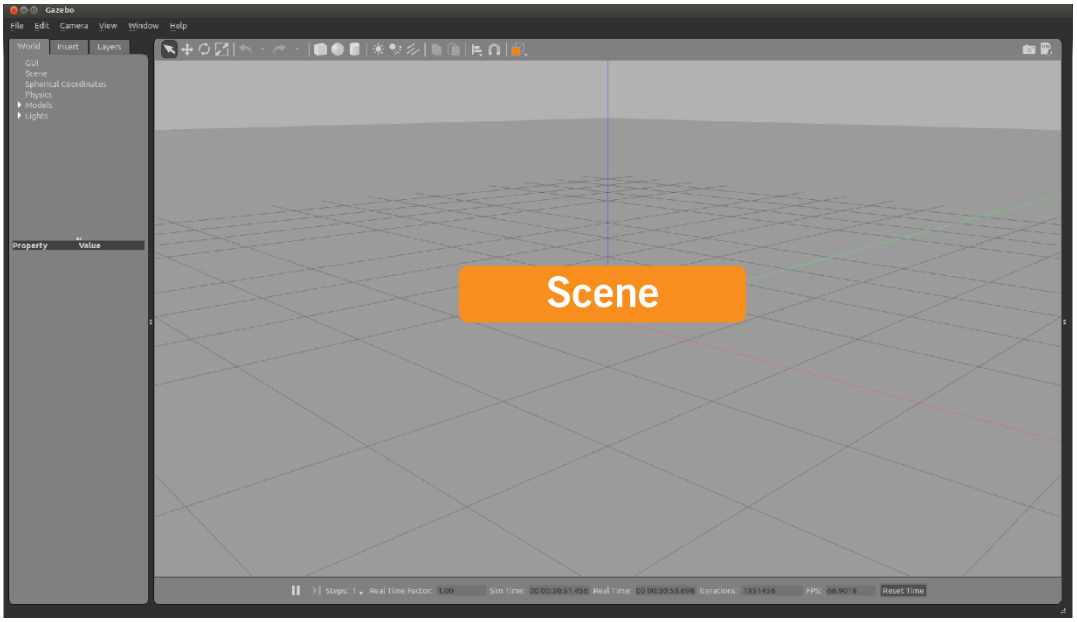


Figure 56: Gazebo Interface

## ❖6.4.3 The Scene:

The Scene is the main part of the simulator. This is where the simulated objects are animated and you interact with the environment.



## ❖6.4.4 The Panels:

Both side panels—right and left—can be displayed, hidden or resized by dragging the bar that separates them from the Scene.

### Left Panel:

The left panel appears by default when you launch Gazebo. There are three tabs in the panel:

- **WORLD**: The World tab displays the models that are currently in the scene, and allows you to view and modify model parameters, like their pose. You can also change the camera view angle by expanding the "GUI" option and tweaking the camera pose.
- **INSERT**: The Insert tab is where you add new objects (models) to the simulation. To see the model list, you may need to click the arrow to expand the folder. Click (and release) on the model you want to insert, and click again in the Scene to add it.
- **LAYERS**: The Layers tab organizes and displays the different visualization groups that are available in the simulation, if any. A layer may contain one or more models. Toggling a layer on or off will display or hide the models in that layer.

  This is an optional feature, so this tab will be empty in most cases.

## ❖6.4.5 The Toolbars

The Gazebo interface has two Toolbars. One is located just above the Scene, and the other is just below.

### Upper Toolbar:

The main Toolbar includes some of the most-used options for interacting with the simulator, such as buttons to: select, move, rotate, and scale objects; create simple shapes (e.g. cube, sphere, cylinder); and copy/paste. Go ahead and play around with each button to see how it behaves.

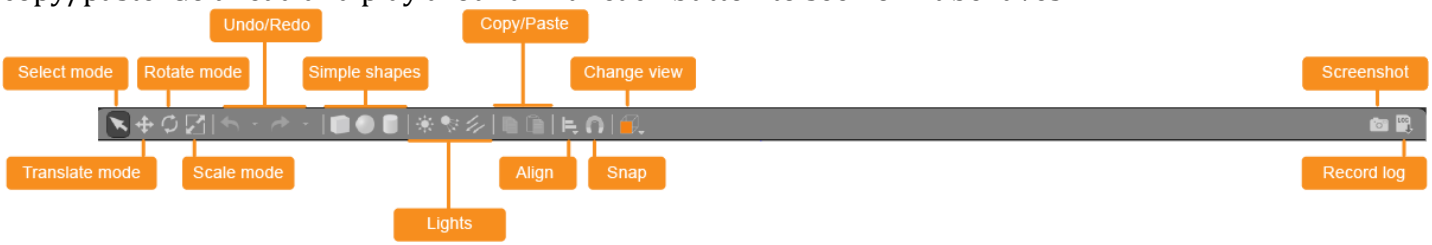



**Figure 57: Upper ToolBar**

### Bottom Toolbar:

The Bottom Toolbar displays data about the simulation, like the simulation time and its relationship to real-life time. "Simulation time" refers to how quickly time is passing in the simulator when a simulation is running. Simulation can be slower or faster than real time, depending on how much computation is required to run the simulation.

"Real time" refers to the actual time that is passing in real life as the simulator runs. The relationship between the simulation time and real time is known as the "real time factor" (RTF). It's the ratio of simulation time to real time. The RTF is a measure of how fast or slow your simulation is running compared to real time.

The state of the world in Gazebo is calculated once per iteration. You can see the number of iterations on the right side of the bottom toolbar. Each iteration advances simulation by a fixed number of seconds, called the step size. By default, the step size is 1 ms. You can press the pause button to pause the simulation and step through a few steps at a time using the step button.

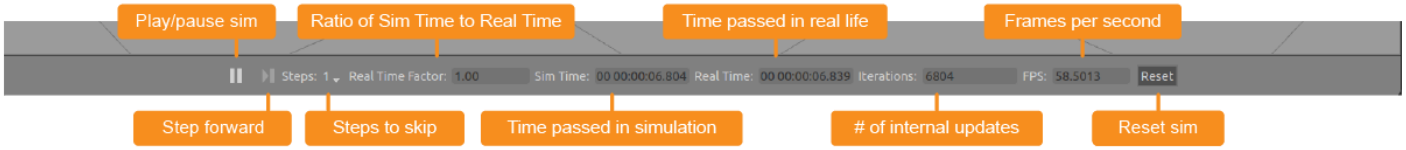
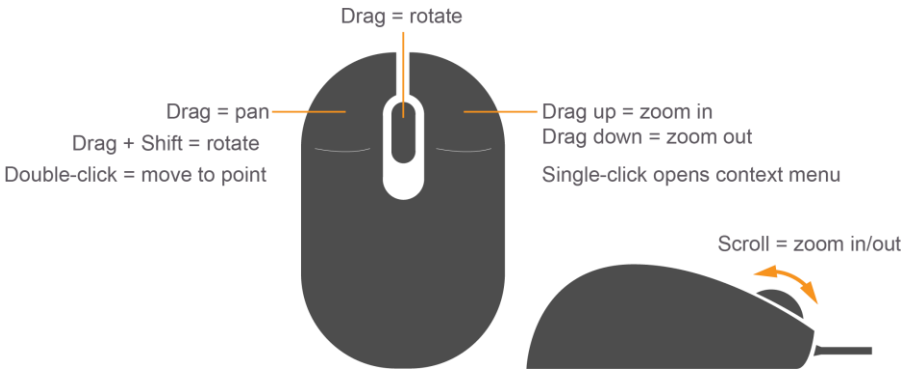



**Figure 58: Bottom ToolBar**

## Mouse Controls:

The mouse is very useful when navigating in the Scene. We highly recommend using a mouse with a scroll wheel. Below are the basic mouse operations for navigating in the Scene and changing the view angle.




Right-clicking on models will open a context menu with various

# 6.5 Rviz:

## ❖6.5.1 What is Rviz?:

Rviz (ROS visualization) is a 3D visualizer for displaying sensor data and state information from ROS. Using rviz, you can visualize Baxter's current configuration on a virtual model of the robot. You can also display live representations of sensor values coming over ROS Topics including camera data, infrared distance measurements, sonar data, and more.

Rviz lets us see what the robot is seeing, thinking and doing. Visualizing and logging sensor information is an important part in developing and debugging.

## ❖6.5.2 Installation:

```
$ sudo apt-get install ros-kinetic-turtlebot-rviz-launchers
```

## ❖6.5.3 Gazebo & Rviz:

- Open a terminal and enter the following command:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

***NOTE:*** *When you launch Gazebo for the first time it may take a few minutes to update its model database.*

- While Gazebo is running, launch Rviz in a new terminal:

```
$ roslaunch turtlebot_rviz_launchers view_robot.launch
```

***NOTE:*** *You can see on the picture how to navigate using a wheeled mouse.*

- Follow the steps below to see what the robot sees:
  - Rotate the camera until you can the TurtleBot from behind.
  - Enable `DepthCloud` option from left-bar and you will see TurtleBot's depth vision.
  - Enable `Image` option from left-bar.
- You need to change `Image->Image Topic` option (choose it from drop-down list) to show different images depending on which topic you use.
  ***NOTE:*** *You can see on the picture how to navigate using a wheeled mouse*

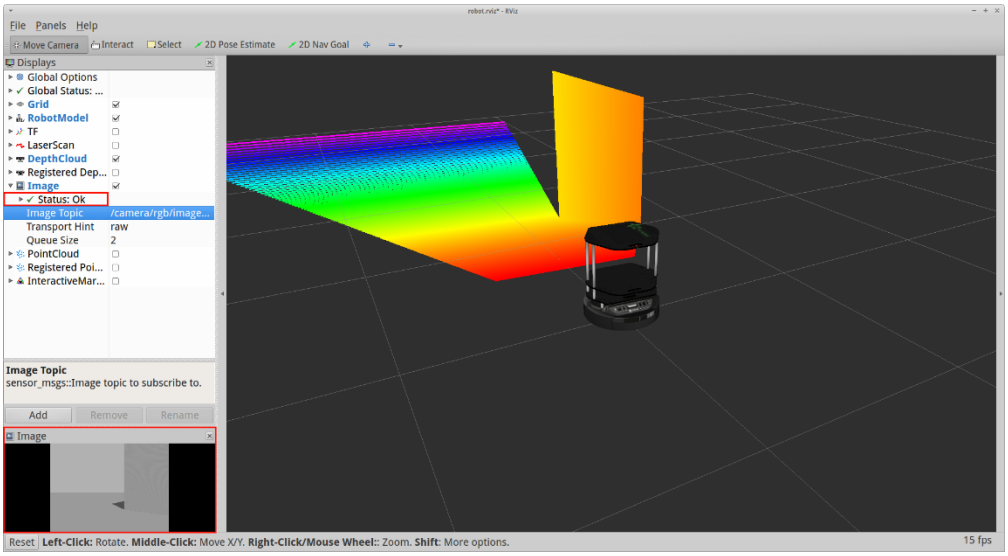



**Figure 59: RVIZ Interface**

## ❖6.5.4 OpenNI & Freenect:

OpenNI and Freenect are industry-led non-profit organization and open source software project focused on certifying and improving interoperability of natural user interfaces and organic user interfaces for Natural Interaction (NI) devices, applications that use those devices and middleware that facilitates access and use of such devices.

Natural Interaction Devices or Natural Interfaces are devices that capture body movements and sounds to allow for a more natural interaction of users with computers in the context of a Natural user interface. The **Kinect** and Wavi X-tion are examples of such devices.

The APIs provide support for:

- Voice and voice command recognition
- Hand gestures
- Body Motion Tracking

### 6.5.4.1 What is Kinect?

Kinect consists of three parts that work together: an **RGB camera**, a **depth sensor**, and a **multi-array microphone**. The TurtleBot uses Kinect to see the world in 3D and for detecting and tracking objects.


### 6.5.4.2 Installation:

- First of all, make sure you have the openni packages installed:

```
$ sudo apt-get install ros-indigo-openni-*
```

- You can check the active 3D sensor of TurtleBot by running:

```
$ echo $TURTLEBOT_3D_SENSOR
```

- You have the right active 3D sensor if you see `kinect`. If you see something else you will need to set another value in .bashrc.

```
$ echo "export TURTLEBOT_3D_SENSOR=kinect" >> .bashrc
```

or run this command in every new terminal:

```
$ TURTLEBOT_3D_SENSOR=kinect
```

### 6.5.4.3 Testing Simulated Kinect:

- To test Kinect sensor in simulated environment, fire up a standard Turtlebot simulation by running the Turtlebot example on both **Gazebo** and Rviz.
- Then click on `DepthCloud` option on the left side and, move your camera with mouse. You should see Kinect's depth cloud in various colors.

# 6.6 GMAPPING:

This lesson shows how to build a map which lets the robot remembers the environment. TurtleBot can autonomously navigate around using the map.

## ❖ 6.6.1 CREATING MAP:

- Create a folder for maps:

```
$ mkdir ~/turtlebot_custom_maps
```

- Launch Gazebo world:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

- Start map building:

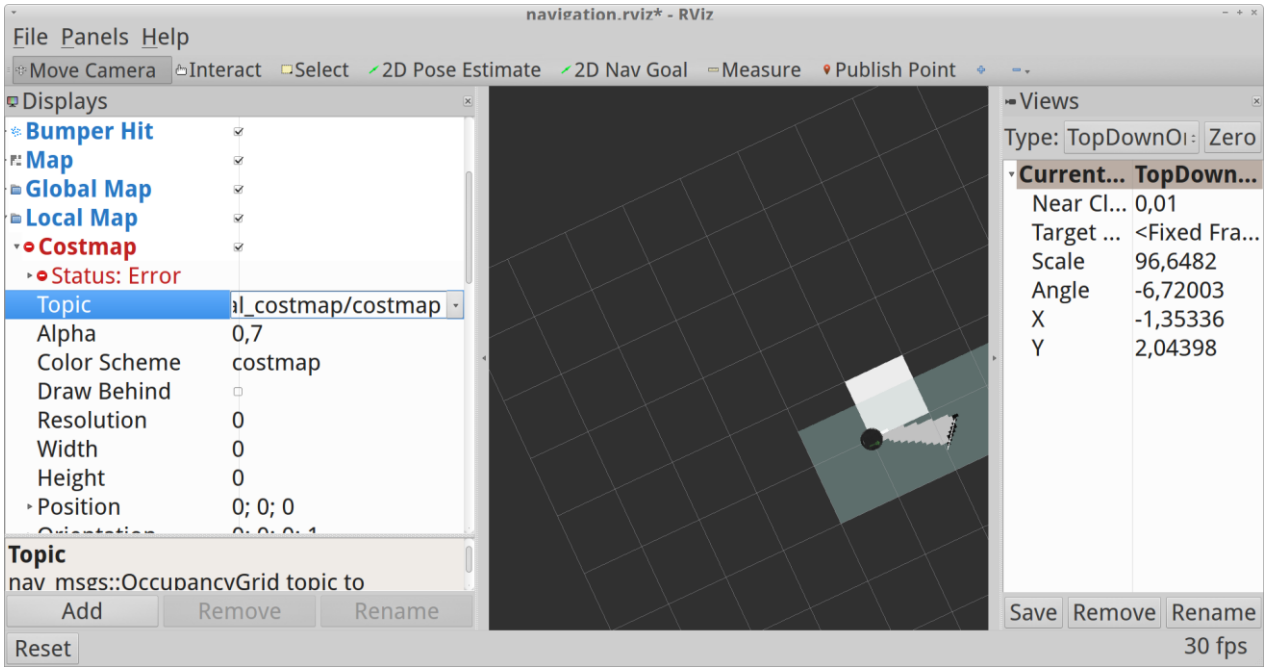
```
$ roslaunch turtlebot_gazebo gmapping_demo.launch
```

- Use Rviz to visualize the map building process:

```
$ roslaunch turtlebot_rviz_launchers view_navigation.launch
```

- Change the options:

`Local map` -> `Costmap` -> `Topic` (choose `/map` from drop-down list). See on the picture:



- Change the options:

`Global map` -> `Costmap` -> `Topic` (choose `/map` from drop-down list).

- Launch teleop.:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

- Drive the TurtleBot around, This is a picture of 360-degrees turn:

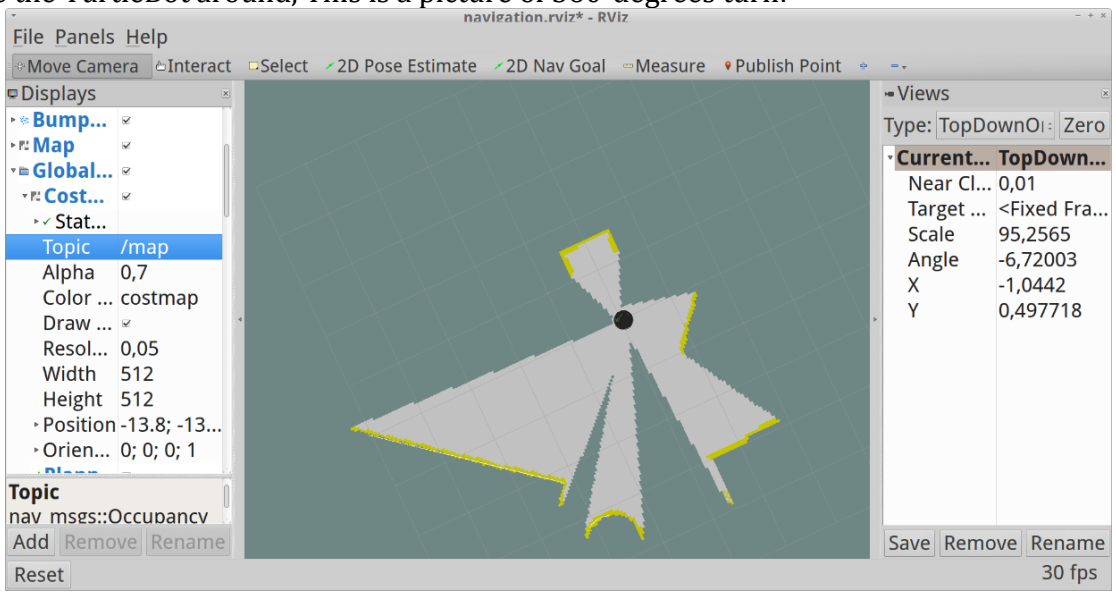
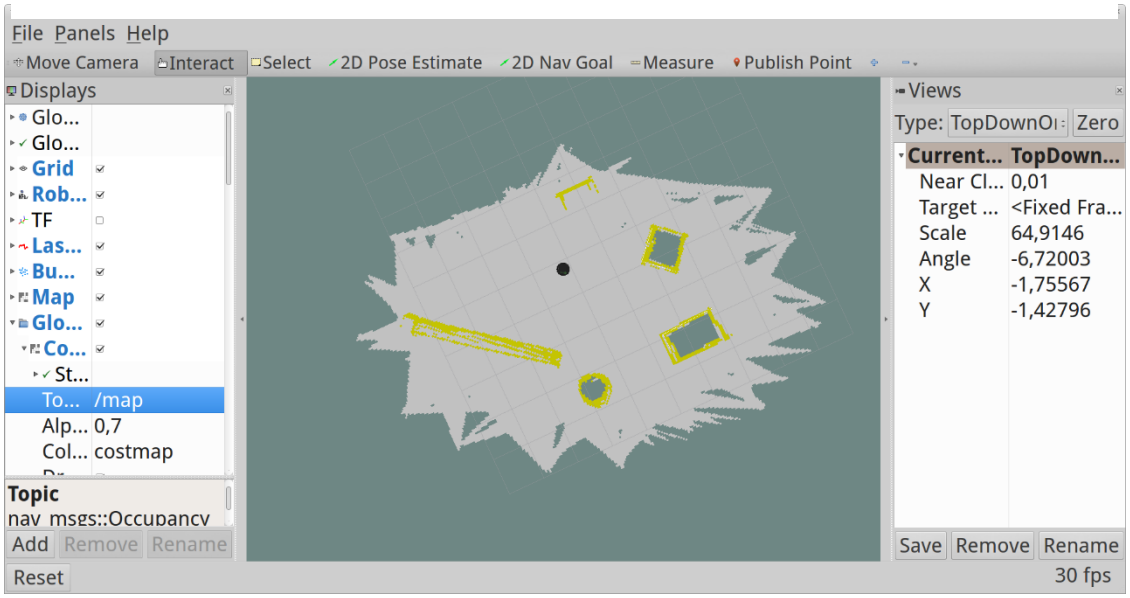



**Figure 60: SLAM With RVIZ**




- Save a map when your picture is good enough (like this):
- Create a folder and save the map in:

```
$ mkdir ~/turtlebot_custom_maps
$ rosrun map_server map_saver -f /home/<user_name>/custom_maps/map1
```

## ❖ 6.6.2 Autonomous Navigation:

- Launch Gazebo world:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

- Launch navigation demo:

```
$ roslaunch turtlebot_gazebo amcl_demo.launch map_file:=/home/<u-n>/custom_maps/map1.yaml
```

- You can launch the default map for playground world if you have not your own map. Run this command:

```
$ roslaunch turtlebot_gazebo amcl_demo.launch
```

- Launch Rviz:

```
$ roslaunch turtlebot_rviz_launchers view_navigation.launch
```

- Send a navigation goal. Click the `2D Nav Goal` button Click on the map where you want the TurtleBot to drive and drag in the direction the Turtlebot should be pointing at the end.

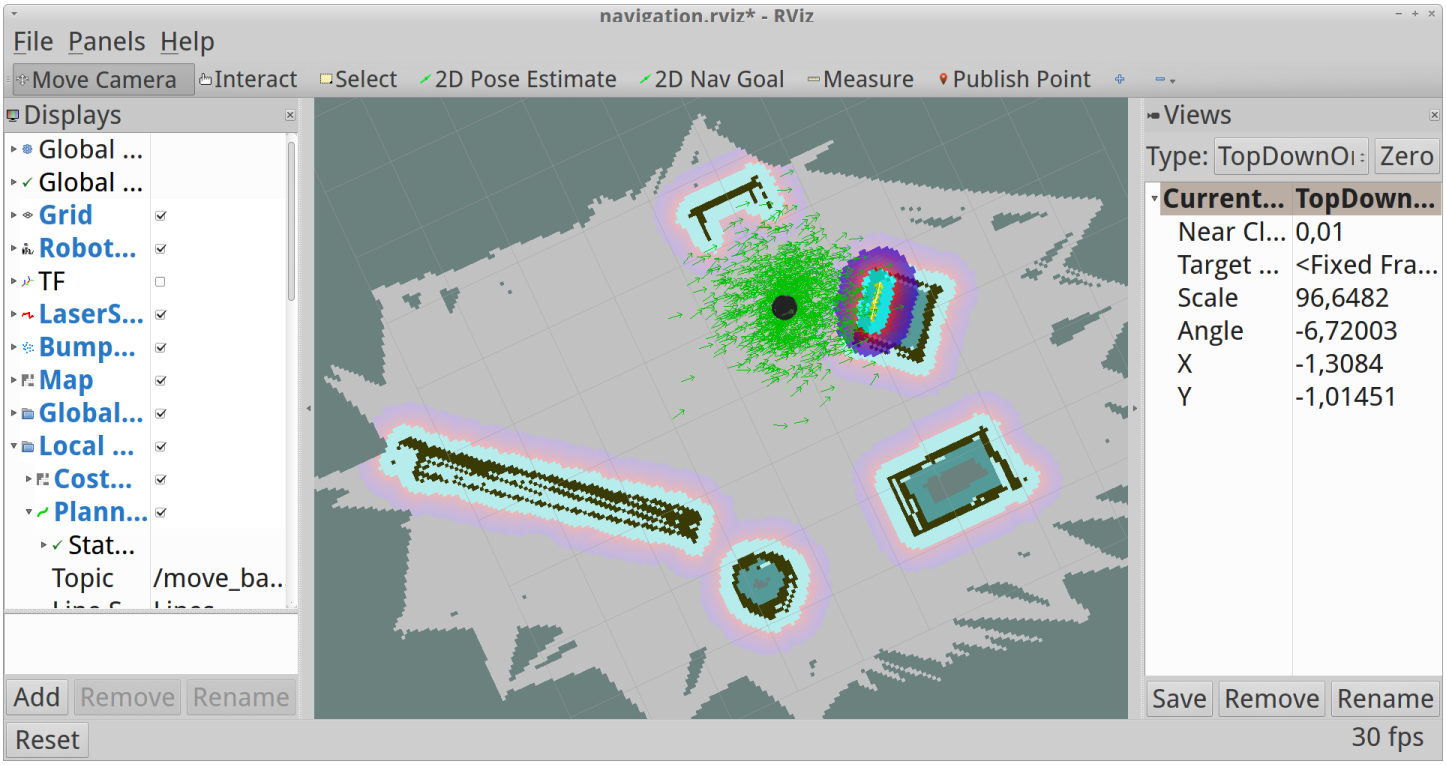



**Figure 61: Obstical Detection**

# Chapter Seven

## The Assisstant System

# 7.1 DialogFlow Agents:

Agents are best described as NLU (Natural Language Understanding) modules. These can be included in your app, product, or service and transform natural user requests into actionable data.

This transformation occurs when a user input matches one of the intents inside your agent. Intents are the predefined or developer-defined components of agents that process a user's request.

Agents can also be designed to manage a conversation flow in a specific way. This can be done with the help of contexts, intent priorities, slot filling, responsibilities, and fulfillment via webhook.

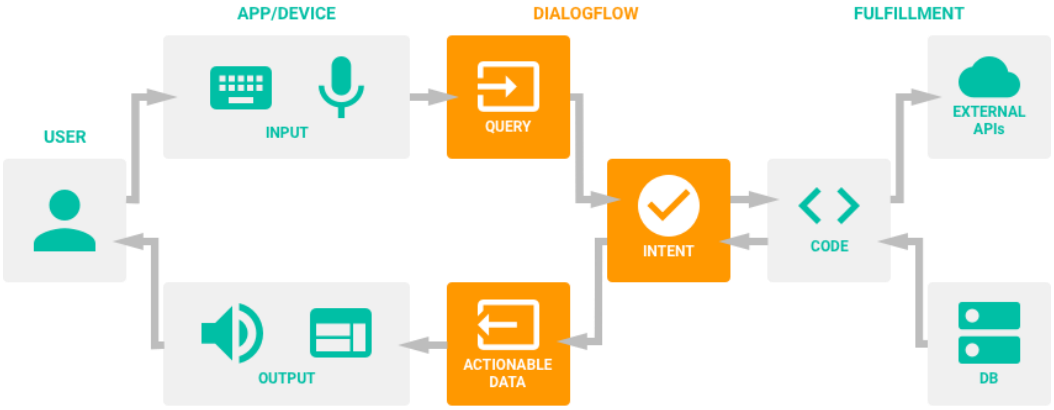
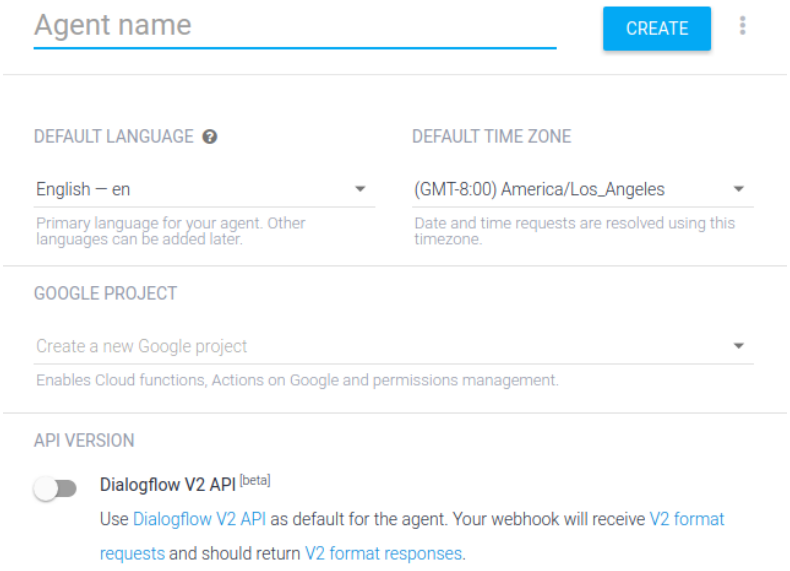



Figure 62: Dialogue Flow Chart

## ❖ 7.1.1 Creating:

❖ To create an agent, click on **Create Agent** in the left menu.



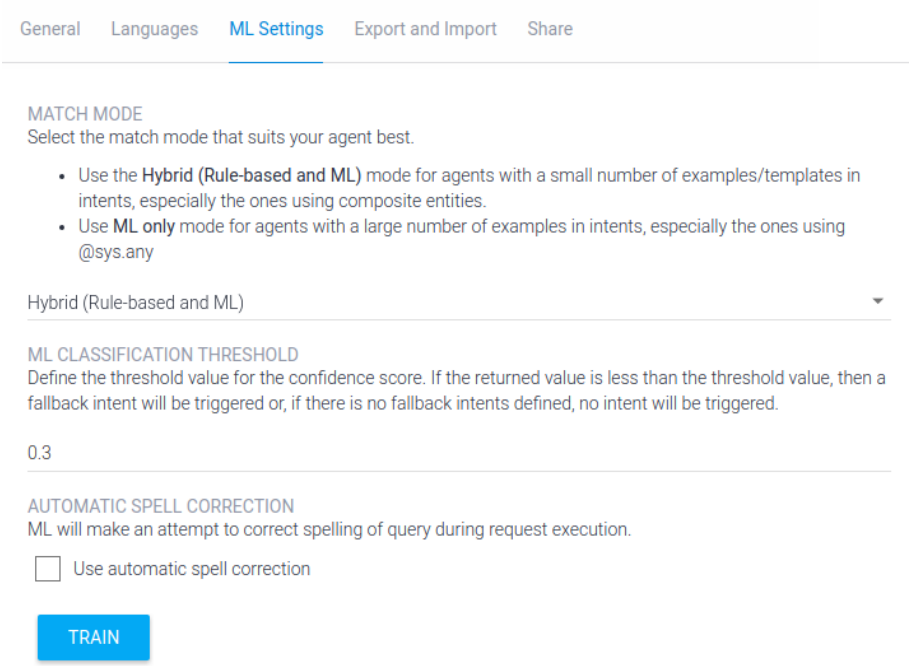
# ❖ 7.1.2 ML Settings (Machine Learning):

- **Match Mode** - This setting defines what algorithm should be used for machine learning. The setting applies to all the intents in which Machine Learning is enabled.
- **Hybrid** - This mode fits best for agents with a small number of examples in intents and/or wide use of composite entities.
- **ML Only** - This mode can be used for agents with a large number of examples in intents, especially the ones using the system entity @sys.any.
- **ML Classification Threshold** - Defines a threshold value between 0 and 1 for triggering fallback intents and returns the "score" in the JSON response to the query. If no fallback intents are provided, no intent will be triggered.

  **Note:** Fallback intents return `"score": 1` and no fallback intent returns a `"score": 0`.
- **Automatic Spell Correction** - When enabled, this allows a misspelled input from a user to be accepted for matching.

  **Caution:** While this allows an incorrect term to be matched, the misspelled input will still be returned in the logs, training and JSON.

  **Warning:** Misspellings that are close to another viable term may be matched, potentially triggering the wrong intent. If you run into matching issues, try disabling this option.



# 7.2 Intents:

An intent represents a mapping between what a user says and what action should be taken by your software.

Intent interfaces have the following sections:

- Training Phrases
- Action
- Response
- Contexts

## ❖ 7.2.1 Example (“) and Template (@) Modes:

Each **User says** expression can be in one of two modes: Example Mode (indicated by the " icon) or Template Mode (indicated by the@ icon).

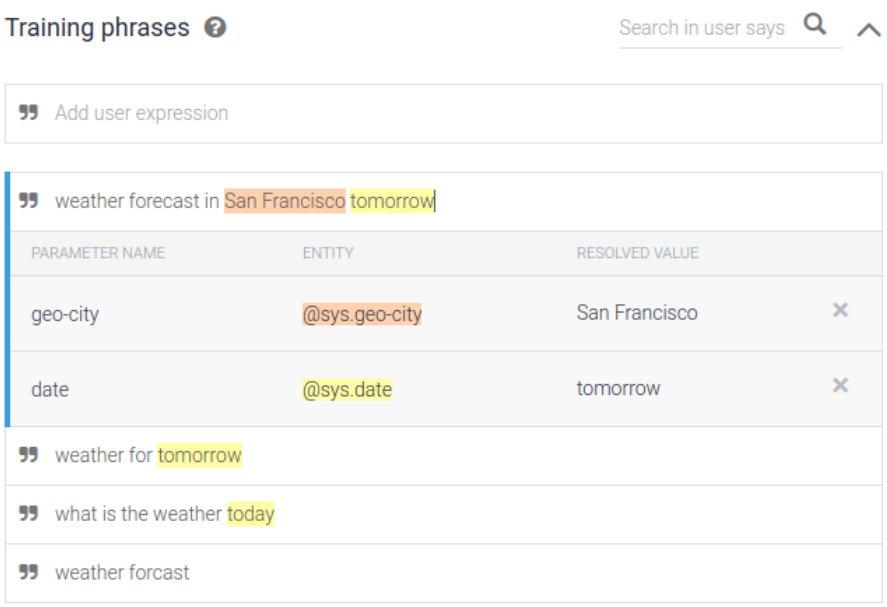
Examples are written in natural language and annotated so that parameter values can be extracted. You can read more on annotation below.

Templates contain direct references to entities instead of annotations, i.e., entity names are prefixed with the @ sign.

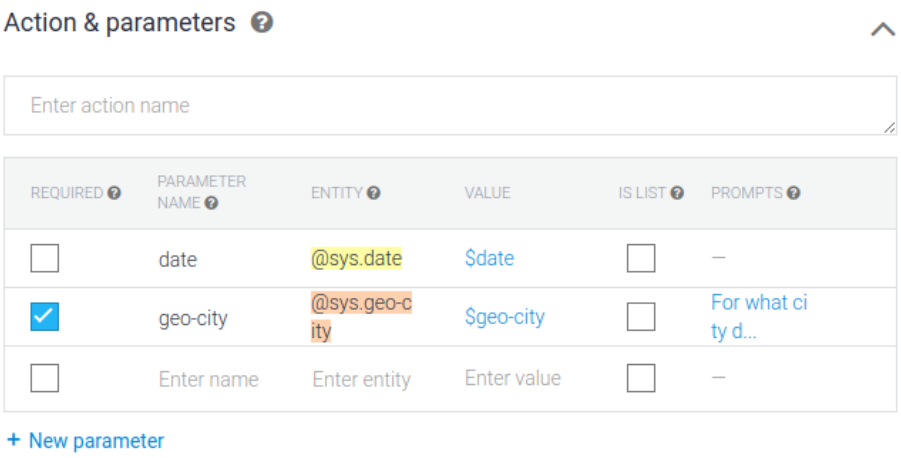
To toggle between modes, click on the " or @ icon.

We recommend using examples rather than templates, because it’s easier and Machine Learning learns faster this way. And remember: the more examples you add, the smarter your agent becomes.

## ❖ 7.2.2 Example Annotation:



The results of the automatic annotation in the review window and parameter table are synchronized. If you change something in the review window, the respective changes will automatically occur in the parameter table, and vice versa.




## ❖ 7.2.3 Response

In this section, you can define **your agent's responses** which will be provided by your application when the intent is triggered.

### Text Response:

You can improve your agent eloquence by adding several variations of the text response per intent. When the same intent has been triggered more than once, different text response variations will be unrepeatable until all options have been used. It'll help make your agent speech more human-like.

### Special Characters:

If you need the **dollar sign** $ or the **number sign** # to appear in your agent's text response, use braces around the value that follows $ or #. For example: ${100} – where 100 is a constant value ${$number} – where $number is a reference to the parameter value #{channel} – where channel is a string #{#channel.name} – where #channel.name is the reference to the parameter value "name" from the context "channel"

If you need to use braces in text responses, use double braces: {{ will be displayed as { and }} will be displayed as }.

### Handling Empty Parameter Values:

If an intent is designed in such a way that some parameters can return empty values after the intent has been triggered, the variants of Text response that contain references to the parameters with empty values won't be given as text responses. Make sure to define different variations of Text response for such intents.

For example, if an intent has 2 parameters and may return both or any of them with empty value, and you want to reference parameter values in Text response, make sure to define at least 4 variants of Text response:

- referencing both parameter values,
- referencing the 1st parameter value,
- referencing the 2nd parameter value,
- without any reference to the parameter values.

### Rich Messages:

If you use one of the following one-click integrations – Facebook Messenger, Kik, Slack, or Telegram – you can define your bot responses as rich messages (images, cards, quick replies etc.) directly in intents.

## ❖ 7.2.4 Contexts:

**Contexts** are designed for passing on information from previous conversations or external sources (e.g., user profile, device information, etc). Also, they can be used to manage conversation flow.

To define the contexts, click on 'Define contexts' right below the intent name.

Input contexts serve as a prerequisite for the intent to be matched; i.e., the intent will participate in matching only when **all** the contexts in the input context field are active.

## ❖ 7.2.5 Fallback Intents:

Fallback intents are triggered if a user's input is not matched by any of the regular intents or enabled built-in Small Talk.

When you create a new agent, a default fallback intent is created automatically. You can modify or delete it if you wish.

If you'd like to create contingency actions or responses for various input contexts, you can add your own fallback intents. Make sure to set a unique set of input contexts for each fallback intent.

To add a new fallback intent:

- Go to **Intents** from the left side menu
- Click on the three dot "more options" menu next to 'Create Intent'

# 7.3 Entities:

Entities are powerful tools used for extracting parameter values from natural language inputs. Any important data you want to get from a user's request, will have a corresponding entity.

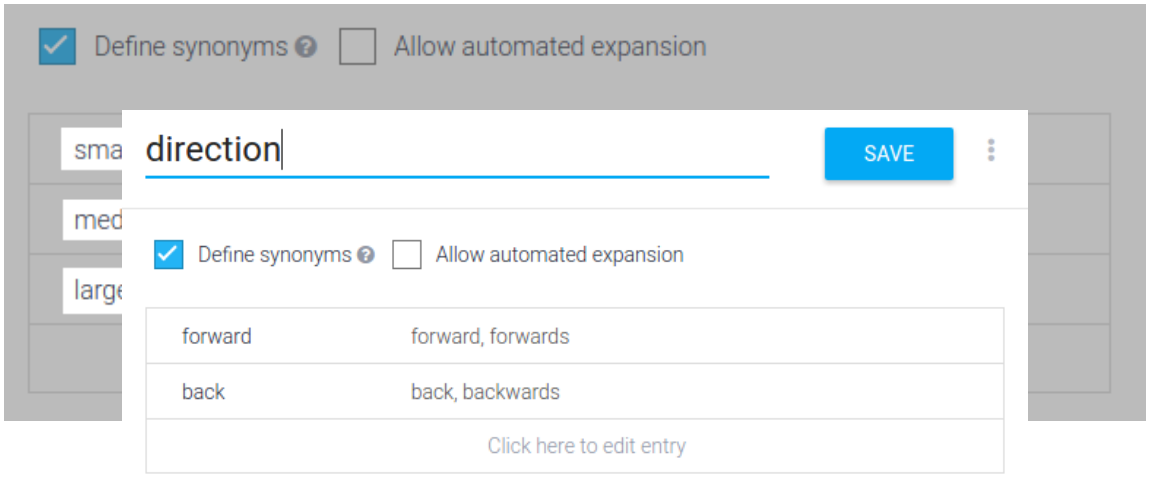
The entities used in a particular agent will depend on the parameter values that are expected to be returned as a result of the agent functioning. In other words, a developer does not need to create entities for every possible concept mentioned in the agent – only for those needed for actionable data.

There are 3 types of entities: system (defined by Dialogflow), developer (defined by a developer), and user (built for each individual end-user in every request) entities. Each of these can be classified as mapping (having reference values), enum (having no reference values), or composite (containing other entities with aliases and returning object type values) entities.

## ❖ 7.3.1 Dev Mapping:

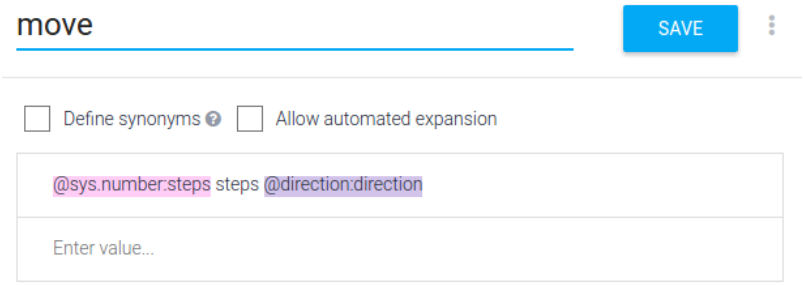
These entities allow for the mapping of synonyms to a reference value. For example, a food type entity could have an entry with a reference value of "vegetarian" with synonyms of "veg" and "veggie".

To create a mapping entity, leave the checkbox **Define Synonyms** checked and add a reference value and synonyms.

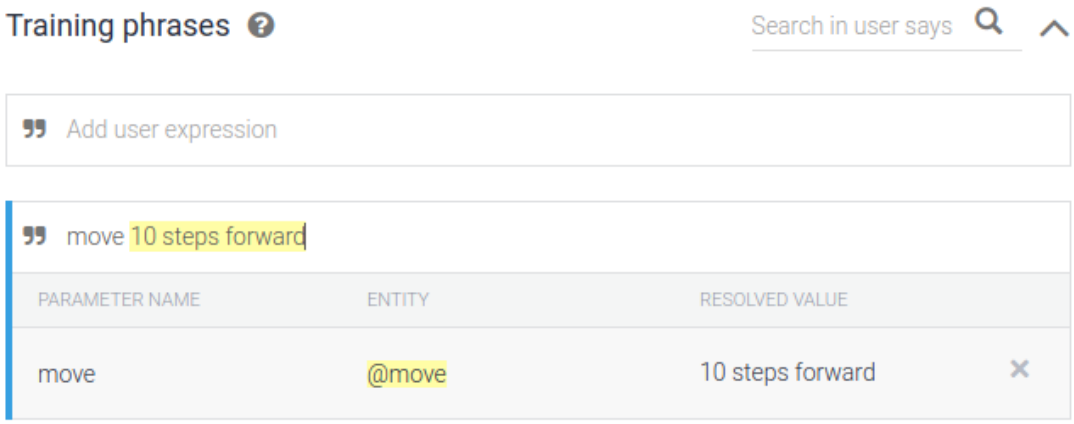



- ❖ Example: a robot's basic movement can have two characteristics: direction and number of steps. In order to describe all possible combinations, you first need an entity that captures the direction.

Then you create a second entity that refers to the direction and the number of steps. Make sure to uncheck Define Synonyms.




By providing the example "Move 10 steps forward", the @move entity is annotated automatically.



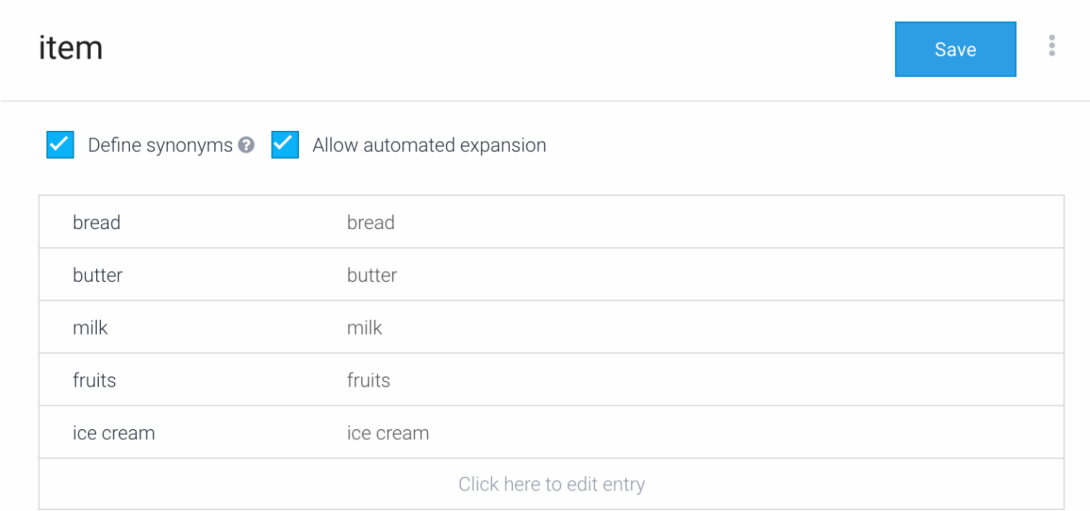

By providing the example "Move 10 steps forward", the @move entity is annotated automatically.

## ❖ 7.3.2 Automated Expansion:

This feature of developer mapping entities allows an agent to recognize values that have not been explicitly listed in the entity.

If a user's request includes an item that isn't listed in the entity, Automatic Expansion recognizes the undefined item as a parameter in the entity. The agent sees the user's request is similar to the examples provided, so it can derive what the item is in the request.

For example, consider a shopping list with items to buy:




If a user says "I need to buy some vegetables", "vegetables" will be picked up as a value, even though it's not included in the @item entity. With Automated Expansion enabled, the NLU sees the user's query is similar to the Training Phrases provided in the intent and can pick out what should be extracted as a new value.

# 7.4 GOOGLE ASSISTANT INTEGRATION:

## ❖ 7.4.1 Setup Google Cloud Project[Part 1]:

1. Sign in to your Google account. If you don't already have one, sign up for a new account.
2. Select or create a Cloud Platform project.


3. Enable billing for your project.


4. Enable the Cloud Functions API.


5. Install and initialize the Cloud SDK.


6. Update and install gcloud components:

```
gcloud components update &&
gcloud components install beta
```

## ❖ 7.4.2 Create a Starter JS File:

Open CMD(As Administrator) To start, create a directory on your local system for the code:

- Linux or Mac OS X:

```
mkdir ~/PROJECT_NAME

cd ~/PROJECT_NAME
```

- Windows:

```
mkdir PATH/PROJECT_NAME

cd PATH/PROJECT_NAME
```

Then create an index.js file in the project directory you just created, with the code, Example:

```
/*
* HTTP Cloud Function.
*
* @param {Object} req Cloud Function request context.
* @param {Object} res Cloud Function response context.
*/
exports.PROJECT_NAME = function PROJECT_NAME (req, res) {
  response = "This is a sample response from your webhook!" //Default response from
the webhook to show it's working

  res.setHeader('Content-Type', 'application/json'); //Requires application/json MIME
type
  res.send(JSON.stringify({ "speech": response, "displayText": response
  //"speech" is the spoken version of the response, "displayText" is the visual
version
  }));
};
```

## ❖ 7.4.3 Setup Google Cloud Project[Part 2]:

- Deploy the function

```
gcloud beta functions deploy PROJECT_NAME --stage-bucket BUCKET_NAME
--trigger-http
```

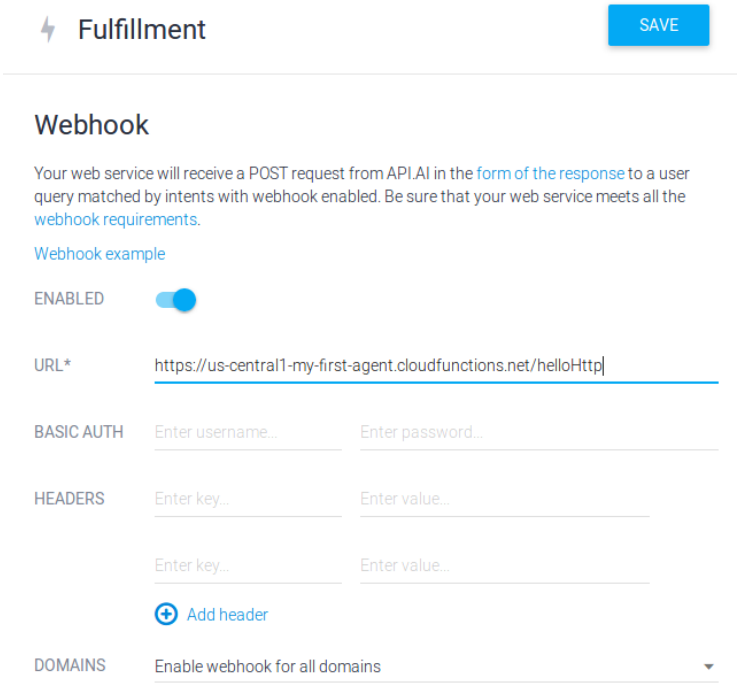
  - `PROJECT_NAME` is the name of our project.
  - `BUCKET_NAME` can be found by going to your related Google Cloud project and click on Cloud Storage under the Resources section.
  - `--trigger-http` *More information*

- Once completed, the status and information related to the function will be displayed. Make note of the httpsTrigger url. It should look something like this:
  `https://[REGION]-[PROJECT_ID].cloudfunctions.net/PROJECT_NAME`

## ❖ 7.4.4 Enable Webhook in Dialogflow:

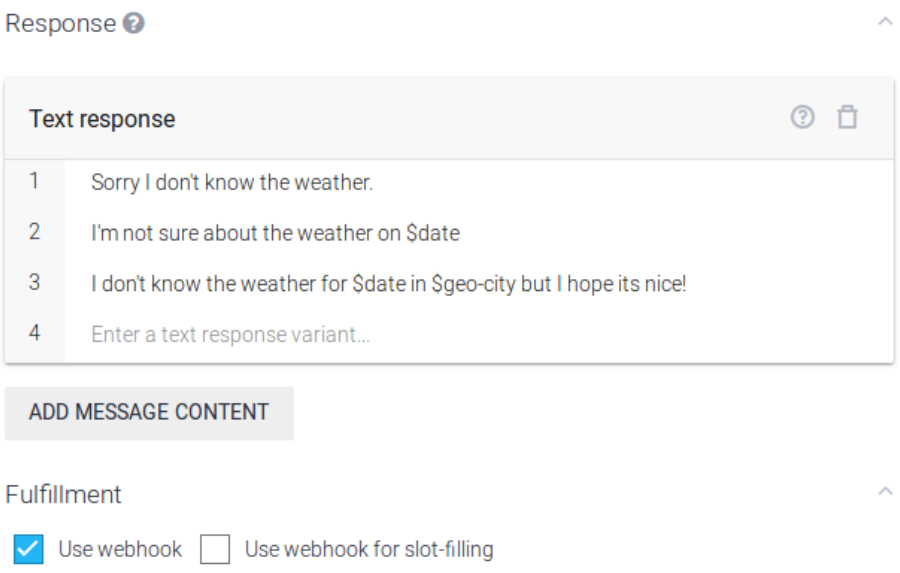
In Dialogflow, make sure you're in the correct agent and click on Fulfillment in the left hand menu

1. Toggle the switch to enable the webhook for the agent
2. In the URL text field, enter the httpTrigger url you got when you deployed your function
3. Click Save




## ❖ 7.4.5 Enable Fulfillment in Intent:

1. Navigate to the desired intent
2. Expand the Fulfillment section at the bottom of the page
3. Check the Use Webhook option
4. Click Save



# 7.5 WebHook:

Setting up a webhook allows you to pass information from a matched intent into a web service and get a result from it.

## ❖ 7.5.1 Requirements

- ***Authentication:***

  Authentication can be done in 2 ways:

  - Basic authentication with login and password.
  - Authentication with additional authentication headers[URL].

If the integrated service doesn't require any authentication, leave the authentication fields blank.

## ❖ 7.5.2 Cloud Functions for Firebase:

For simple webhook testing and implementation, you can use the Cloud Functions for Firebase area of the Fulfillment page. In most cases the free "Spark" tier of Firebase is all you'll need.

To enable:

1. Click on **Fulfillment** in the left menu.
2. Click the switch for **Inline Editor**.
3. Some basic, functional code provided to help you get started.
4. To deploy your fulfillment, click the **Deploy** button under the code editor.

# ❖ 7.5.3 Moving fulfillment outside of the editor:

## Prerequisites:

- Download your code by clicking the download button in the upper right hand corner of the code editor.


- Get your Google Project ID from the **General** tab of your agent's settings.

## Setup Firebase CLI:

Before you can install the Firebase CLI, you will need to install Node.js on your machine. Once you've installed Node.js, you can install the Firebase CLI using npm (the Node Package Manager) by running the following command:

```
npm install -g firebase-tools
```

You should now have a globally available firebase command available from any terminal window on your machine. Once you've installed the Firebase CLI, sign in using your Google account:

```
firebase login
```

This command connects your local machine to your Firebase account and grants access to your projects. To test that authentication worked, you can run firebase list to see a list of all of your Firebase projects. The list should be the same as the projects listed at Firebase console.

### Getting the latest version

You can make sure your Firebase CLI is up to date by re-running the installation command:

```
npm install -g firebase-tools
```

***Deploy your function with the Firebase CLI***

1. Unzip your downloaded code found in "DialogflowFulfillment.zip".
2. Open a terminal or shell and navigate to the "functions" directory inside the unzipped achieve. For example, cd `~/Downloads/DialogflowFulfillment/firebase/functions`.
3. Run `npm install` to install all of your functions' dependencies
4. Run the command `firebase login` and login to the Google account associated with your agent and allow the Firebase CLI to manage your Cloud and Firebase Projects.
5. Run the command `firebase init`. When prompted for feature selection choose Functions. When prompted for a Firebase Project, select the Cloud Project ID found in your Dialogflow agent's settings.
6. Deploy your Cloud Function for Firebase with firebase deploy
7. Copy the Function URL after deployment (may take a minute or two) and paste it in your Dialogflow agent's fulfillment!

## ❖ 7.5.4 Sample Webhook Example: For Addition:

```
's.calculateadd': () => {
      // Use the Actions on Google lib to respond to Google requests; for other requests use
JSON
    var addition = JSON.parse(parameters['clc-number']) +  JSON.parse(parameters['clc-
number1']);
     if (requestSource === googleAssistantRequest) {
        sendGoogleResponse('i can say that '+parameters['clc-number']+' plus
'+parameters['clc-number1']+' equals '+addition+''); // Send Google response to user
     } else {
        sendResponse('i can say that '+parameters['clc-number']+' plus '+parameters['clc-
number1']+' equals '+addition+''); // Send simple response to user
     }
    },
```

## ❖ 7.5.5 Custom Payload For WEBSEARCH:

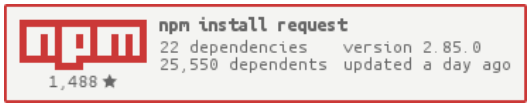
Sending a Custom Payload written in JSON format in a JS file, i faced this when i was using I'm using the Firebase inline editor/Firebase CTL. After this you will be able to send RichMessages and more right from the Index.js file.

#1- In your Index.js search for 'default': () => { after this down bellow you will find "responseToUser, just a little bellow uncomment this line "data: richResponsesV1,"... now you are able to start sending richmessages for both Slack and FB Messenger

#2- Now back to your action handler and add this example for websearch i made, i will explain things in the code:

```
'web.search': () => {
      var a = parameters['searchfor'];
      var r = a.replace(/ /g,"-");
     if (requestSource === googleAssistantRequest) {
        if (app.hasSurfaceCapability(app.SurfaceCapabilities.SCREEN_OUTPUT)) {
   app.ask(app.buildRichResponse()
             .addSimpleResponse('Here is what i found about '+parameters['searchfor']+'')
             .addBasicCard(app.buildBasicCard('Here is what i found about
'+parameters['searchfor']+'')
             .setTitle(''+parameters['searchfor']+'')
             .addButton('Press Here...',
'https://www.google.com.eg/search?source=hp&ei=FGOYWo-
NFIvxUvH7sfgE&q='+parameters['searchfor']+'')
             )
         );
 } // Send Google response to user
     } else {
     const websearch = {
          'facebook': {
            'attachment': {
              'type': 'template',
              'payload': {
                'template_type': 'button',
                'text':'Here is what i found about '+parameters['searchfor']+' ',
                    'buttons': [
                      {
                        'type': 'web_url',
                        'url': 'https://www.google.com.eg/search?source=hp&ei=FGOYWo-
NFIvxUvH7sfgE&q='+r+'',
                        'title': 'Open Link...'
                      }]}}}}
    let responseToUser = {
          data: websearch,
        }; sendResponse(responseToUser);
        } },
```

# ❖ 7.5.6 Connecting To APIs For NAVIGATION:

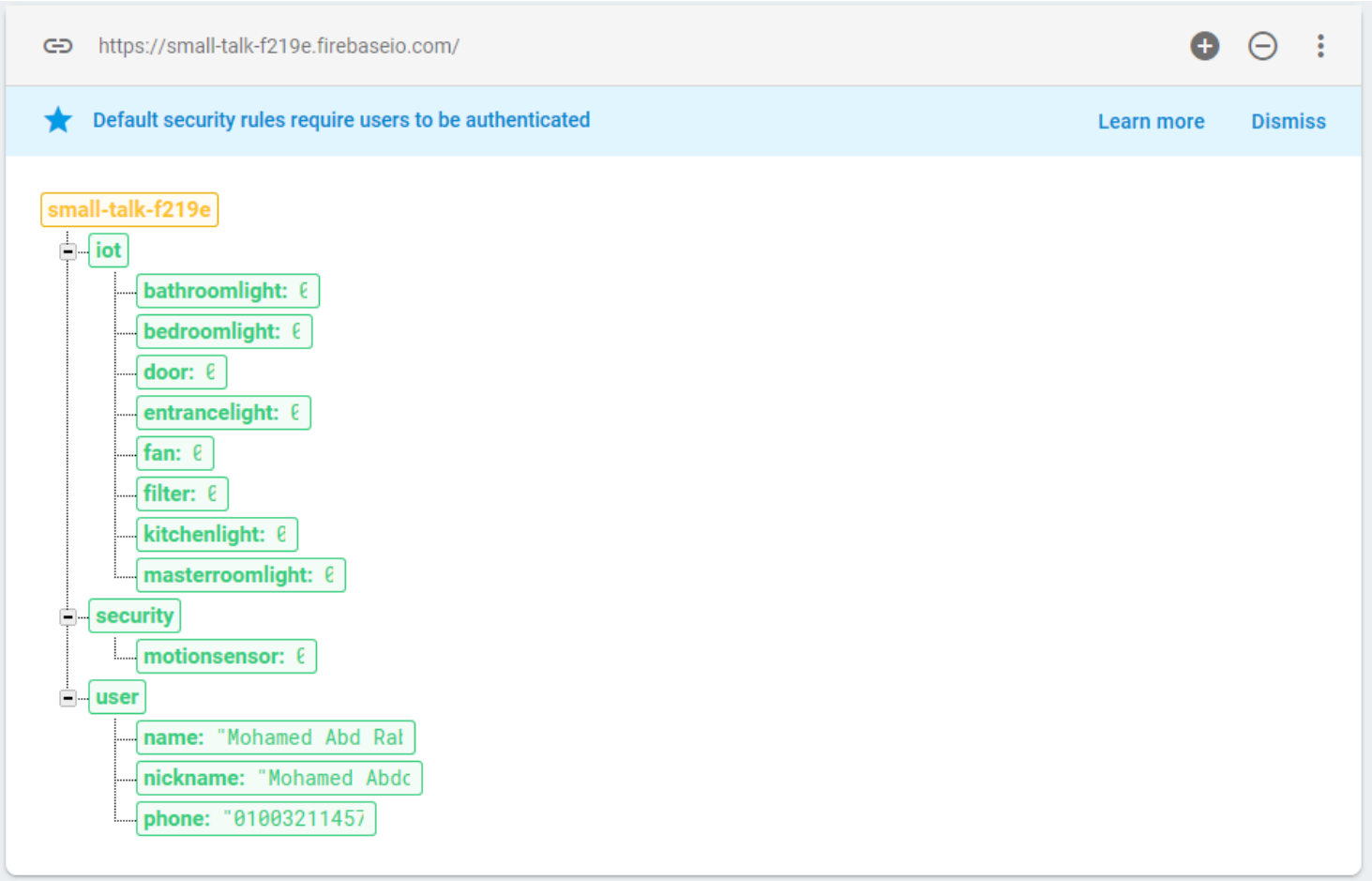



- Using **Google Maps API**:
- Install **request Package** using **npm**
- Getting API Tokens and Credintals From Google Maps

```
'map.navi': () => {
    if (requestSource === googleAssistantRequest) {
       sendGoogleResponse('Okay, i have updated your phone number');
    } else {
       const request = require("request");
       var as = parameters['address'];
        var rs = as.replace(/ /g,"-");
        var rz = as.replace(/ /g,"+");
       const urls = 'http://maps.googleapis.com/maps/api/geocode/json?address='+rs;
       request.get(urls, (error, response, body) => {
           body = JSON.parse(body);
           const mapsearch = {
             'facebook': {
              'attachment': {
               'type': 'template',
               'payload': {
                'template_type': 'button',
                'text':`Location Is ${body.results[0].formatted_address}`,
                    'buttons': [
                      {
                        'type': 'web_url',
                        'url':
`https://www.google.com.eg/maps/place/`+rz+`/@${body.results[0].geometry.location.lat},${body
.results[0].geometry.location.lng},14z`,
                        'title': 'Show location on map...'
                      }
                    ]
                  }}}}
       let responseToUsers = {
             data: mapsearch,
           };
           sendResponse(responseToUsers);
         });
    }},
```

## ❖ 7.5.7 Building And Connecting To Database:

- Using **Firebase Realtime Database** (Cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client.)




- Intializing the database in Node,js:

```
var admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);
```

- Connecting to Database and write values, This is an example of **turning on one device only**:

```
'smarthome.device.switch.on': () => {
      // Use the Actions on Google lib to respond to Google requests; for other requests use
JSON
      var device = parameters['device']
      var textArray = [
        'Okay, i have switched on the '+device+'',
        'Sir, yes sir... Turning '+device+' On.',
        'Consider it done, Turning '+device+' On.',
        'Your '+device+' just got turned On.'
    ];
    var randomNumber = Math.floor(Math.random()*textArray.length);

if (device == 'light'){
       if (room == 'kitchen'){
        return admin.database().ref().child('iot').once('value').then(function(snapshot) {
          var kitchenlight = (snapshot.val().kitchenlight);
        if (kitchenlight == 1){
          itsOn();
        }else{
          itsOff();
        }
        });
      }
function itsOn (){
      if (requestSource === googleAssistantRequest) {
          sendGoogleResponse(textArray[randomNumber]);
        }
      } else {
          sendResponse(textArray[randomNumber]);
        };
      }
```

## ❖ 7.5.8 Installing FirebaseArduino API:

- Install Arduino
- Install Arduino ESP8266 core
- Download FirebaseArduino library
- Start Arduino
- Click Sketch > Include Library > Add .ZIP Library...
- Choose firebase-arduino-master.zip downloaded in step 3.

- **Using FirebaseArduino API:**
    - Start Arduino
    - Open File > Examples > FirebaseArduino > FirebaseRoom_ESP8266
    - In FirebaseRoom_ESP8266: Replace WIFI_SSID and WIFI_PASSWORD with WiFi credentials
    - Go to Firebase Realtime Database
    - Copy the Database hostname for out project it was (small-talk-f219e.firebaseio.com)
    - In FirebaseRoom_ESP8266: replace FIREBASE_HOST with the Database Hostname
    - Go to ⚙ > Project Settings > Database > Database secrets
    - Click Firebase Secrets > Show
    - Copy the Database Secret
    - In FirebaseRoom_ESP8266: Replace FIREBASE_AUTH with Database Secret
    - Select the board Board > ESP8266 Modules > NodeMCU 1.0
    - Select the serial port Port
    - Select the upload speed Upload Speed > 9600
    - Click Sketch > Upload
- **Prototype Hardware:**
    - Couple Of LEDs
    - Mini Servo Motor
    - NodeMCU (ESP8266 WiFi Programming & Development Kit)

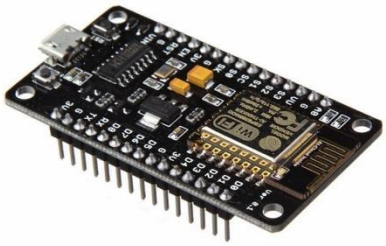


Figure 63: Node MCU

- **Sample Ardunio code for *one device only*:**

```
#include <Servo.h>
#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>
#define FIREBASE_HOST "small-talk-f219e.firebaseio.com"
#define FIREBASE_AUTH "iRNU3w8Hb7NbgHCMxt4PYOgankzJyzFvS6uLoKm"
#define WIFI_SSID "Mohamed Abdo"
#define WIFI_PASSWORD "*********"
Servo myservo;
  // Intilaization
const int kitchenlight = D7;
void setup() {
  Serial.begin(115200);
  pinMode(kitchenlight, OUTPUT);
  // Connect to Wifi.
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("connecting");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
```

```
  }
  Serial.println();
  Serial.print("connected: ");
  Serial.println(WiFi.localIP());
  // Connect to Database.
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}
void loop() {
  digitalWrite(officelight, Firebase.getInt("office-light"));
}
```

# 7.6 Final View To The Assistant System:

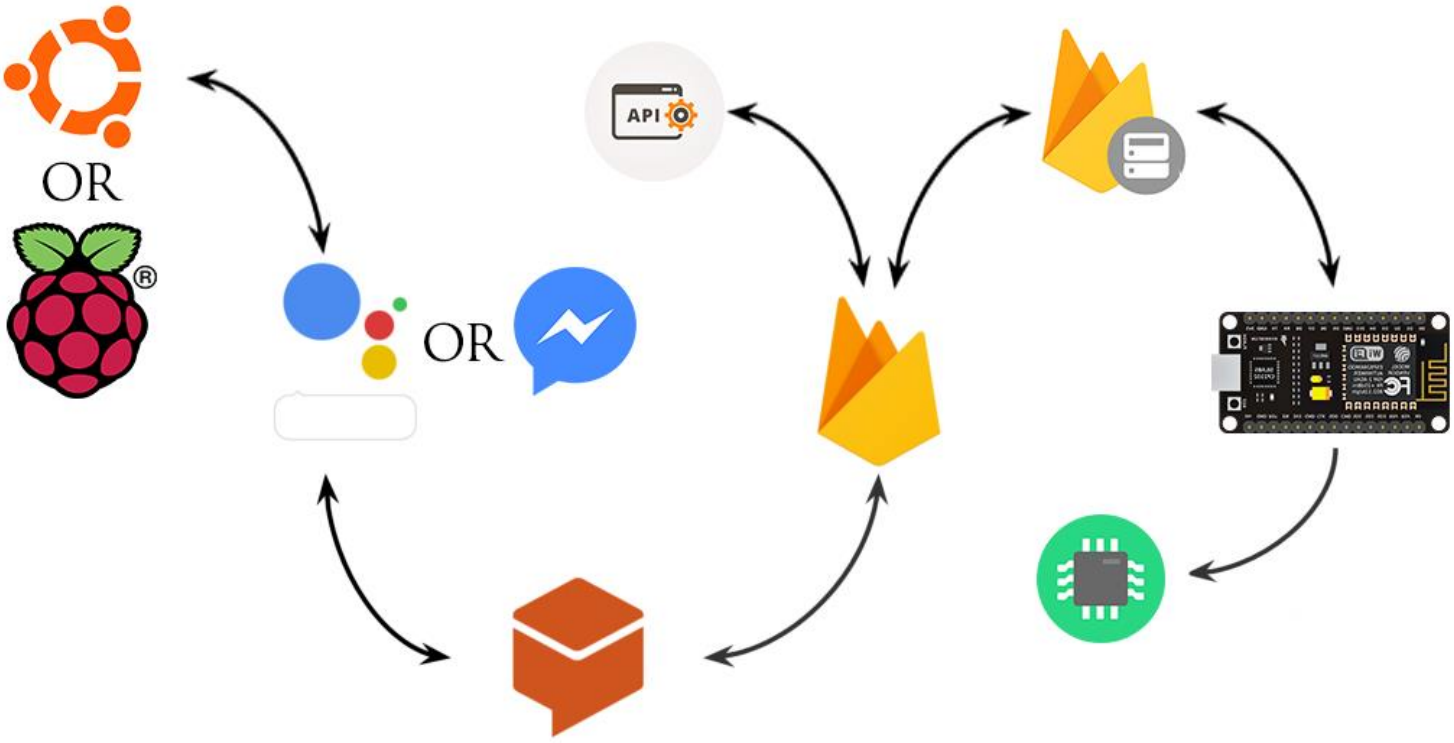



**Figure 64: Final View Of The Assistant System**

What happened next to develop our assistant system?

- Connected Our LINDO Chatbot to RasberryPI 3(Ubuntu Mate) and Ubuntu Desktop
- Added More APIs Connection to Retrive more data right from the internet,
- Implemented Home Security Systems beside the IOT Systems.

# Chapter Eight

# Conclusion

# 8.1 Conclusion :

Coordination of autonomus mobile robots 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/knowm environments, chatting with or helping humans and home automation.

However, there were different challenges in autonomus mobile robots 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 in using ros platform and reaching an output using it.

# 8.2 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. Some few characteristics will be mentioned below to help evaluating our robot.

## ❖ 8.2.1 Cost:

The total cost of the robot is about 12000 L.E which is low cost compairing with other robots .

## ❖ 8.2.2 Manufacturability:

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

## ❖ 8.2.3 Market need:

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

## ❖ 8.2.5 Environmental impact:

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

# 8.3 Future Work :

## ❖ 8.3.1 Further Steps:

- Equip the robot with manipulators to hold objects.
- Improve the AI chatbot by adding more training data .
- 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.

# References:


1. ROS Wiki: http://wiki.ros.org/
2. "Proposal for Implementation of Real-time Systems in ROS 2". Retrieved 16 August 2016.
3. "Browsing packages for indigo". *ROS.org. ROS*. Retrieved 21 February 2016.
4. "Package Summary". *ROS.org. ROS*. Retrieved 21 February 2016.
5. "Package SUmmary". *ROS.org. ROS*. Retrieved 21 February 2016.
6. "Package Summary". *ROS.org. ROS*. Retrieved 21 February 2016.
7. "client libraries". *ROS.org*. Retrieved 12 December 2017.
8. "ROS/Installation - ROS Wiki". *Wiki.ros.org. 2013-09-29*. Retrieved 2014-07-12.
9. "android - ROS Wiki". *Wiki.ros.org. 2014-04-12*. Retrieved 2014-07-12.
10. "Robot Operating System (ROS) Support from MATLAB - Hardware Support". *Mathworks.com*. Retrieved 2014-07-12.
11. Firebase doc . https://firebase.google.com/docs/
12. Dialogue flow doc . https://dialogflow.com/docs/dialogs
13. Turtlebot Summary . http://wiki.ros.org/turtlebot
14. Blynk doc . http://docs.blynk.cc/
15. “Learning ROS for robotics programming” Aaron Martines
16. “Effective robotics programming with ROS” Aaron Martines
17. “Matering ROS for robotics programming” Lentin Joseph

# Appendices:

## ❖ Project cost sheet:

| S.N. | Item Name | Quan. | Unit | Price/Unit L.E. | Subtotal | Supplier Name | Supplier Address | Remarks |
|---|---|---|---|---|---|---|---|---|
| **1** | Aluminum Plate | 1 | Each | 800 L.E. | 800 L.E. | --------- | El Sabtiya | Plate Size (1.5m*1m*3mm) |
| **2** | Aluminum Laser Cutting | 1 | Each | 250 L.E. | 250 L.E. | Radwan Laser | El Sabtiya | Metal Works Included |
| **3** | Acrylic Plate | 1 | Each | 200 L.E. | 200 L.E. | Spiro Plastics | Dst. 10, 6th Of October | Plate Size (1.5m*0.5m*2mm) |
| **4** | Acrylic Laser Cutting | 1 | Each | 25 L.E. | 25 L.E. | Xerox Laser | Hosary, 6th Of October | |
| **5** | Home Martial and Laser Cutting | 2 | Each | 50 L.E | 100 L.E | Xerox Laser | Hosary, 6th Of October | Plate Size (1m*0.7m*3mm) |
| **6** | Couplers | 4 | Each | 75 L.E. | 300 L.E. | --------- | El Warraq | |
| **7** | Motor | 4 | Each | 255 L.E. | 1020 L.E. | Future Electronics | Ain Shames, Ah. Fo. St. | |
| **8** | Motor Driver | 1 | Each | 395 L.E. | 395 L.E. | Ram Electronics | Bab Elouq, Abdeen | 4 Channels Motor Driver |
| **9** | Encoders | 4 | Each | 145 L.E. | 580 L.E. | Ram Electronics | Bab Elouq, Abdeen | 3 PPR |
| **10** | Raspberry PI 3 | 1 | Each | 1550 L.E. | 1550 L.E. | Ram Electronics | Bab Elouq, Abdeen | Full Kit [Borrowed] |
| **11** | Teensy 3.2 | 1 | Each | 750 L.E. | 750 L.E. | Megatronics | Dst. 6, 6th Of October | |
| **12** | NodeMCU Esp8266 | 1 | Each | 60 L.E. | 60 L.E. | Ram Electronics | Bab Elouq, Abdeen | |
| **13** | IMU MPU6050 | 1 | Each | 70 L.E. | 70 L.E. | Ram Electronics | Bab Elouq, Abdeen | |
| **14** | 16 GB Memory Card | 1 | Each | 150 L.E. | 150 L.E. | B2B | Bab Elouq, Abdeen | |
| **15** | Microphone | 1 | Each | 150 L.E. | 150 L.E. | B2B | Bab Elouq, Abdeen | |
| **16** | 12 V Battery | 1 | Each | 1500 L.E. | 1500 L.E. | Ram Electronics | Bab Elouq, Abdeen | [Borrowed] |
| **17** | Kinect V1 | 1 | Each | 3300 L.E. | 3300 L.E. | Online Shop | Online Shop | [Borrowed] |
| **18** | Other Essentials | 1 | Each | 500 L.E. | 500 L.E. | --------- | --------- | Essentials:<br>• Screws & Nuts<br>• Breadboards<br>• Wires & Adaptors<br>• Others |
| **TOTAL** | 11,700 | | | | | | | |

# ❖ User Manual:

## ○ Creating a Map:

On robot's computer, Launch base driver:

```
roslaunch lindorobot bringup.launch
```

On robot's computer, Launch mapping packages:

```
roslaunch lindorobot slam.launch
```

On your development computer, run teleop_twist_keyboard:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

On your development computer, Run rviz:

```
roscd lindo_visualize/rviz
rviz -d slam.rviz
```

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

```
rosrun map_server map_saver -f
~/lindorobot_ws/src/linodorobot/maps/map
```

## ○ Autonomous Navigation:

Launch base driver, on robot's computer:

```
roslaunch lindorobot bringup.launch
```

Launch navigation packages, on robot's computer:

```
roslaunch lindorobot navigate.launch
```

On your development computer, run rviz:

```
roscd lindo_visualize/rviz
rviz -d navigate.rviz
```

## ❖ DATASHEETS:

- DC Motor Controller 4 Channel 4.5A, 4.5V to 12V (http://www.ram-e-shop.com/ds/general/RS011MC_4Channel.pdf)
- DC Motor Shaft Encoder (http://www.ram-e-shop.com/ds/motor/Encoder_MY_37.jpg)
- IMU "9 DOF Module (http://www.ram-e-shop.com/ds/general/IMU_BMP085.rar)
- Raspberry Pi 3 (http://www.raspberrypi.org/help/quick-start-guide/)
- NodeMCU (https://store.fut-electronics.com/products/nodemcu-esp8266-programming-and-development-kit )

## ❖ ABBREVIATIONS:

| | |
|---|---|
| ROS | **Robots Operating System** |
| RPI | Raspberry Pi |
| SLAM | Simultaneous Localization And Mapping |
| PPR | Pulse Per Revolution |
| SSH | Secure Shell |
| IOT | Internet Of Things |
| API | Application Program Interface |
| IDE | Integrated Development Environment |
| PID | Proportion Integral Derivative |
| PWM | Pulse Width Modulation |
| I2C | Inter-Integrated Circuit |
| IMU | Inertial Measurement Unit |
| RPM | Revolution Per Minute |
| GND | Ground |