Zagazig University
Faculty of Engineering
Computer and Systems Engineering Dept.

Computer & Systems, 4th Year
Graduation Project
"Smart Wheelchair"

Under Supervision of

Dr. Ahmed Amer Shahin

2020/2021

Prepared By

Amr Hani Abdelrahman Gouda Zeada

Ahmed Mohamed Osman Soliman

Ahmed Mohamed Mohamed Attia

Islam Tarek Abdelmoniem Abdelhamid

Abdelaziz Abdelfattah Abdelaziz

Mahmoud Khaled Abdallah Mohamed

Medhat Samir Medhat Mounir

Mostafa Soliman Mohamed

Abdelrahman Madgy Abdelraheem

Mahmoud Tarek Mahmoud

Dedication

We dedicate this to our families, our college, our professors, our department students, our community and all of our country.

.....

Acknowledgements

First, all thanks to Allah, for providing us the blessings and the strength to complete this project.

Second, we would like to express our deepest appreciation to our professor Dr. Ahmed Amer Shahin, our department professors and teaching assistants, our university, our families and to anyone that helped us to reach this part of our life.

.....

Table of Contents

Chapter (1): Overview – Project Description & Its Purpose

Since the begging of mankind, humans were always looking for ways to make their daily tasks easier and decrease the man power needed to do a certain job and as a result, humans were always trying to develop new tools with the help of science.

Now we are in the 21th century and the technological development has been never this fast and almost every day new inventions are introduced.

Some people are born with disabilities and other become disabled due to sickness or unfortunate accidents. These people need help in daily tasks that are easy to do by any non-disabled person and that makes them feel distant from society because they can't interact with other people or take part in activities.

The technological development is now heading towards helping these disabled people more than ever before, as the technology now broke many barriers that were holding us back. It's our job as Engineers and developers to devote our time to improve the lives of these people and make them able to engage in the society.

In our project "Smart Wheelchair" we are targeting people with movement impairment disabilities who always need help from somebody to be able to move from one place to another. The aim of the project is to give them the ability to move freely in their houses without the need of any help from others.

The main idea is that the user of Smart Wheelchair will interact with it through voice commands telling it where he/she wants to go inside the

house, and the voice command will be processed giving a navigation goal to the navigation algorithm which will first check its current location relative to the house and then produce a path plan from the position of the chair to the goal location.

Wheeled Autonomous Robot

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 aren't driven by a motor these extra wheels help keep it balanced.

Components Overview

The hardware is designed to be modular and scalable in the future, but mostly all the platforms are built using:

- L298N motor driver: that can handle up to 2 Amperes.
- Motors with built-in encoders.

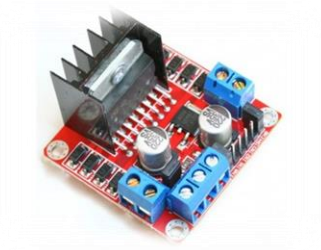
Also, for higher efficiency of motor rating or higher encoder resolution, we have optimized and selected components with better specification.

Components that we have chosen are:

- DC Geared Motors of the type “SG555123000-60K”, which have:
    - Maximum No Load RPM of “50 r/min”.
    - Load Torque Speed (With Load) of “36 r/min”.
    - A Torque of “6.3 Kgf.cm”.
    - A Motor-to-Gear Ratio of “60:1”.
    - An Output Power of “2.3 W”.
    - Load Torque Current (With Load) of “800mA”.
    - A Stall Current of “3000 mA”.
    - Weight of about “350~400 gm”.

  You can find more info about it using this Link.

- DC Motor Controller of the type “RO.RS011MC.4CH”, which has a total number of “4 Channels”, where each has:
    - A Rated Stall Current of “4 A”.
    - An operation Voltage of “4.5 to 12 Volts”.



  You can find more info about it using this Link.

- DC Motor Controller of the type “L298”, that can replace the previous one, and it has:
    - Double H bridge drive.
    - Drive Voltage of “5 to 35 Volts”.
    - Drive Current of “2 A per Channel”.
    - Maximum Power of “25 Watt”.

  You can find more info about it or purchase it using this Link.

- DC Motor Shaft Encoders of the type “MY-37, SG555123 Family”, which have:

  - Supply Voltage of “4.5 to 24 Volts”.
  - Supply Current of “14 to 20 mA”.
  - A Resolution of “3 PPR (Pulses Per Revolution)”.

  You can find more info about it or purchase it using this Link.

- Ultra-Sonic Sensors of type “SR04”, which have:

  - Supply Voltage of “+ 5 Volts DC”.
  - Supply Current of “10 mA”.
  - Measurement Distance Range from “2 cm” to “400 cm”.

  You can find more info about it using this Link.

- IMU of type ”IMU MPU6050” which is a 3-Axis Acceleration Gyroscope 6DOF Module and has:

  - Power Supply of “3 to 5 Volts”.
  - Gyroscopes Range of “+/- 250 500 1000 2000 degree/sec”.
  - Acceleration Range of “+/- 2g, +/- 4g, +/- 8g, +/- 16g”.
  - Chip built-in 16-bit AD converter with 16-bit data output.

  You can find more info about it using this Link.

- Arduino Mega Board of type “Mega2560 Rev3” which acts as our main controller, and has:
    - Microcontroller of type “ATmega2560”.
    - Operating Voltage of “5 Volts”.
    - Input Voltage (recommended) of “7 to 12 Volts”.
    - Input Voltage (limits) of “6 to 20 Volts”.
    - A Digital I/O Pins number of “54 (of which 15 provide PWM output)”.
    - An Analog Input Pins number of “16”.
    - DC Current per I/O Pin of “40 mA”.
    - Flash Memory of “256 KB of which 8 KB used by bootloader”.
    - SRAM of “8 KB”.
    - EEPROM of “4 KB”.
    - Clock Speed of “16 MHz”.

You can find more info about it or purchase it using this Link.

- Connectors of various types, as:
    - Male-Male Connectors.
    - Male-Female Connectors.
    - Female- Female Connectors.

- And finally, a body of our design which is from acrylic to bring all these parts together.

And these are almost all the parts you may need to build the low Level part of this project.

And there are other parts to continue with our navigation and recognition system that will be discussed later in this book, such as:

- Kinect Camera.
- High Definition Microphone.

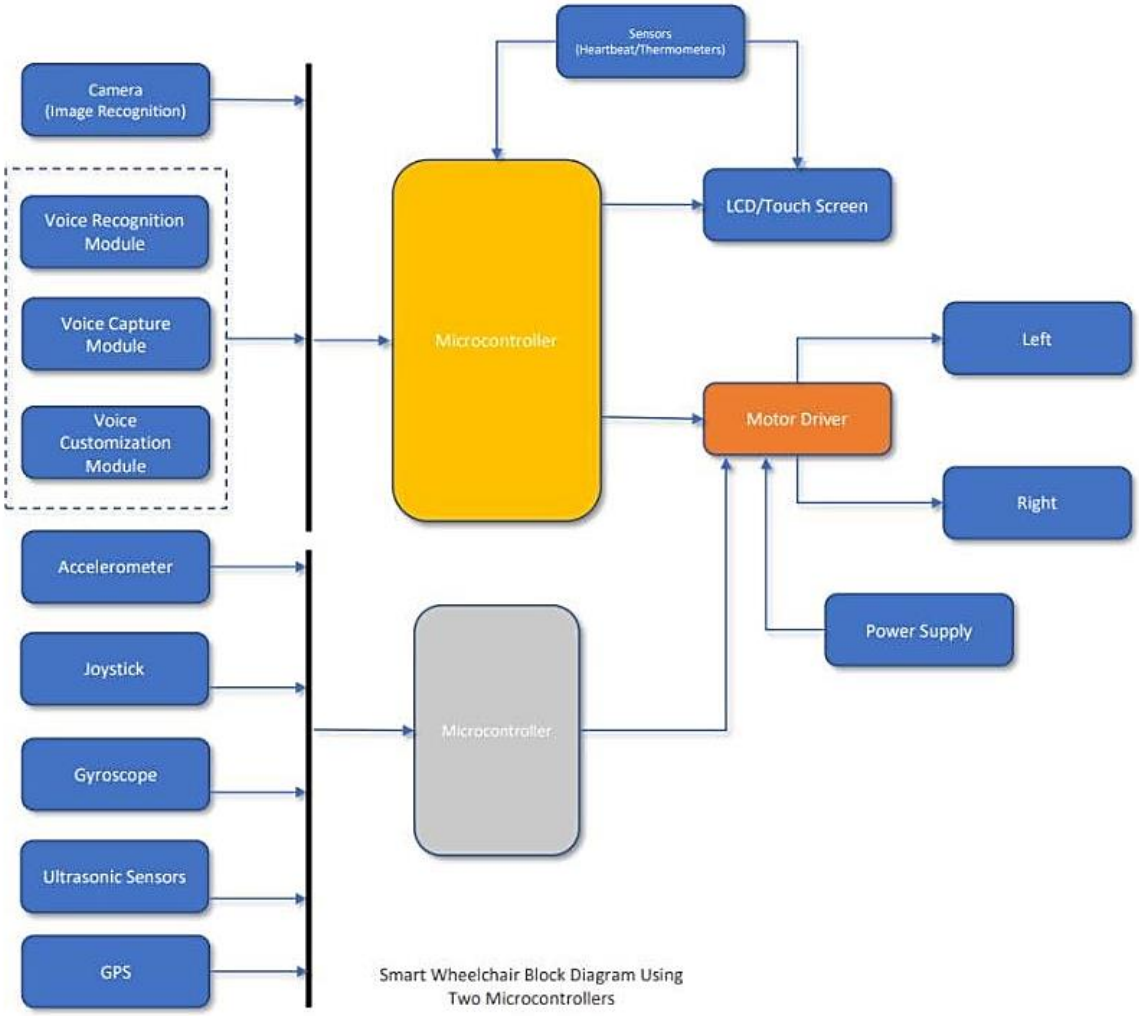
etc…

Software Overview

We will need different requirements in this part such as Robot Operating System (ROS), Raspberry Pi Operating System, Ubuntu OS, Windows OS, Drivers for Kinect Sensor, GPU Drivers and more….

All previous requirements should be **compatible** with each other.

Project Diagram



Smart Wheelchair Block Diagram Using Two Microcontrollers

.....

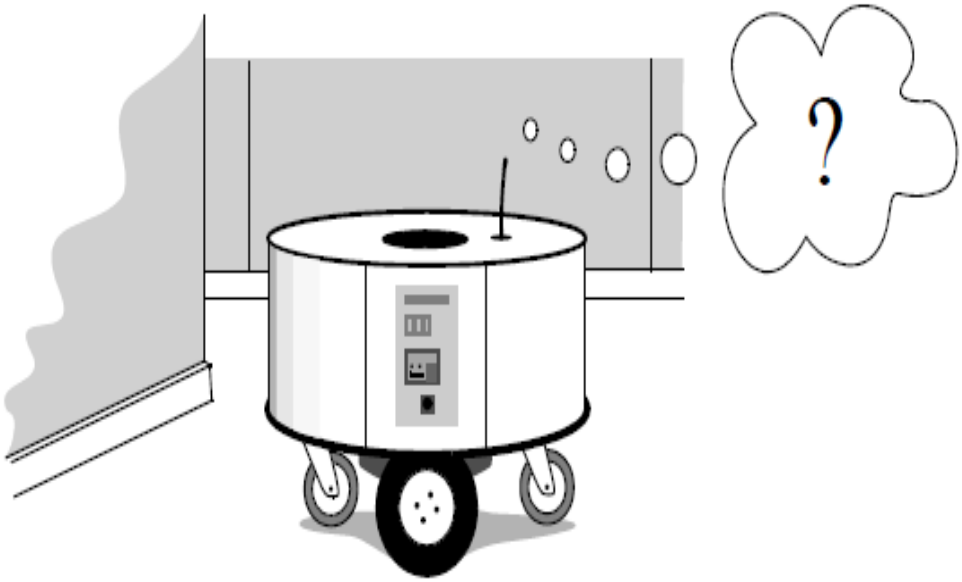
Chapter (2): Navigation

Localization

Introduction

Navigation is one of the most challenging competences required for a mobile robot. Success in navigation requires success in main four blocks which are **Perception** (Extracting meaningful data from sensors) **Localization** (Determination position of robot in the environment) **Cognition** (Actions taken by robot to achieve the goal) **motion control** (Modulation of motors outputs to achieve the desired trajectory).

In this chapter I will talk about localization. First section I will describe how sensors and effectors uncertainty is responsible for difficulties of localization. Second section I will describe how to avoid localization altogether and how to perform a map-based localization then we talk about SLAM in the next section and finally talk about SLAM in ROS at final section.



(1)Where am I?

Challenge of Localization

Localization does not mean that robot only determines its absolute position with respect to Earth's reference frame but, robot must also determine its relative position with respect to people and objects in the environment which it works at so to apply good localization the robot build a map to its environment then use to determine its position with respect to this map.

Why don't we use GPS?

Reasons which prevent us from using GPS are:

- The existing GPS network provides accuracy to within several meters.
- GPS technologies cannot function indoor area.

So, building robot's map will depend on sensors information and effectors actions which will be inaccurate because of these reasons:

- Sensor noise
- Sensor aliasing
- Effector noise

Sensor noise reduces useful information content of sensor reading and this can be solved by taking multiple readings into account also by applying multisensory fusion to increase overall information content of robot's inputs.

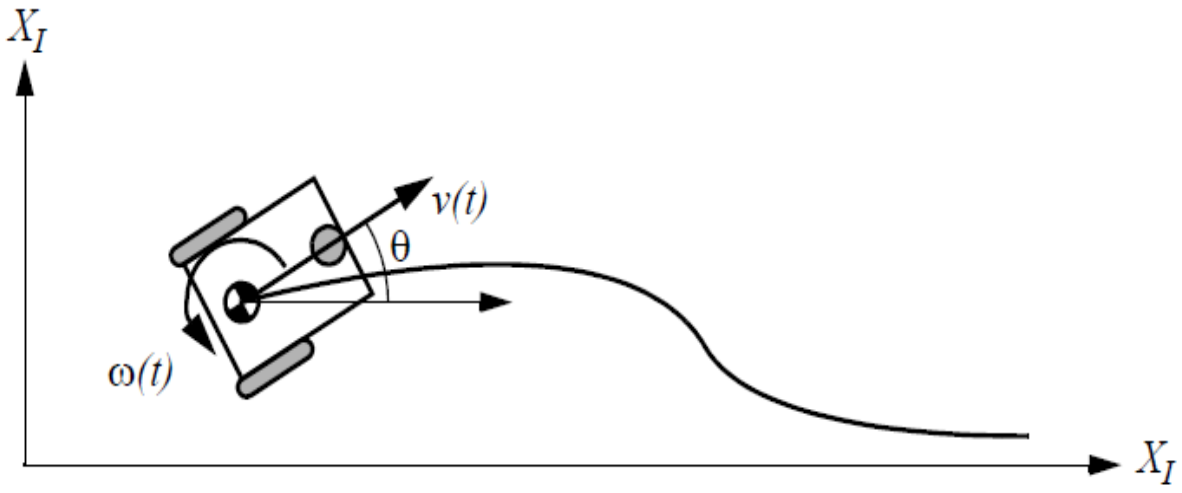
Sensor aliasing appears mostly in visual systems as lack of information to identify or distinguish robot's position from single percept reading this can be solved by applying techniques make robot's localization depend on series

of readings so there is sufficient information to recover robot's position over time.

Effector noise this problem depends mainly on odometry and there are many sources of odometric error:

- Limited resolution during integration
- Misalignment of wheels
- Uncertainty in wheel diameter and unequal wheel diameter
- Variation in the contact point of wheel
- Unequal floor contact

Error model for odometric position estimation



(2)Movement of differential drive motor

The position of the robot is represented by a vector as

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

If we calculate the distance moved by a robot at a time interval Δt

$\Delta x = \Delta s \cos(\theta + \Delta\theta/2)$

$\Delta y = \Delta s \sin(\theta + \Delta\theta/2)$

$\Delta\theta = (\Delta s_r - \Delta s_l) / b$

$\Delta s = (\Delta s_r + \Delta s_l) / 2$

Δx, Δy, Δθ: The path traveled in last time interval.

Δs_r, Δs_l: The distance traveled by right and left wheels of the robot.

b: The distance between two wheels of the robot.

The updated position will be:

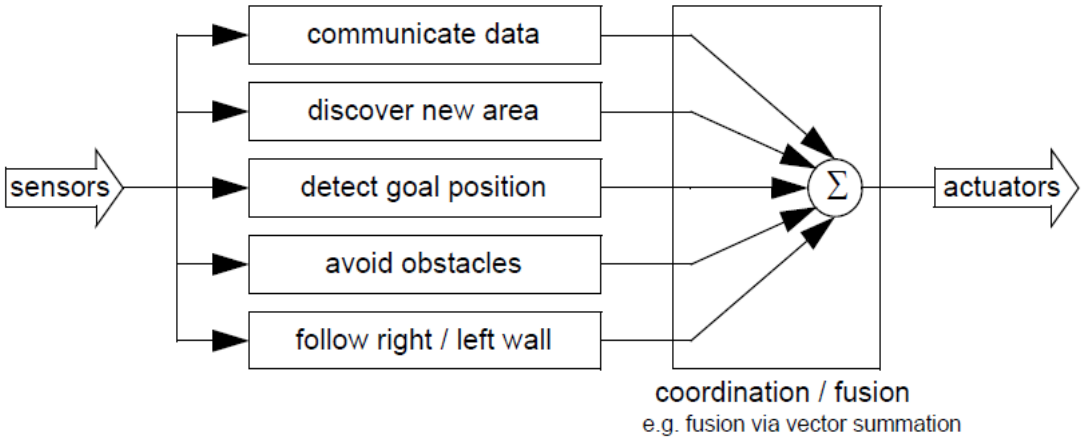
$$p` = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}$$

Should we use localization?

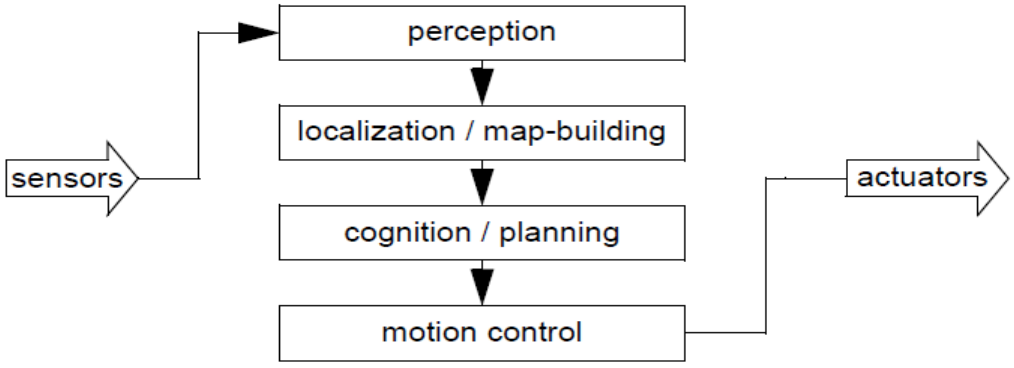
Some people will ask if we should use localization and build map or we can just give the robot a specific orders to navigate from one place to another, the answer will be yes, we should use localization because if u put an algorithm for a robot to navigate from one place to another u have to change this algorithm and these orders every time you want to make your robot navigate at different at different places but when you use localization

although challenges mentioned before but robot knows what it has to do like building map of environment it works at then it can navigate easily at it.

An architecture for behavior-based navigation vs architecture of map-based navigation



(3)Architecture of behavior-based navigation



(4)Architecture of map-based navigation

Advantages of map-based navigation:

- Explicit map-based concept of position makes system's belief about position transparently to human operators.
- Existence of map represents a medium for communication between human and robot, human can give the robot a new map if the robot goes to new environment.
- The map which created by robot can be used by human.

Belief representation

The robot must have a representation of its belief regarding its position on map and does the robot has a single unique position in next movement or it has set of possible position and who can it ranked them.

There are two important beliefs which are single-hypothesis belief and multiple hypothesis-belief. **Single-hypothesis belief** covers the solution in which the robot postulates its unique position and this make it easy for robot to take its decision at cognitive level while **multiple-hypothesis belief** enables the robot to describe the degree to which it's uncertain about its position and this not only takes robot to its goal but also it improve the trajectory of the robot as at each movement the robot can determine its degree of uncertainty which enables it to take the more certain trajectory depending on new readings of sensors and effectors.

There are three fundamental relationships must be considered when represent a map:

- The precision of the map must appropriately match the precision with which the robot needs to achieve its goals.
- The precision of the map and the type of features represented must match the precision and data types returned by the robot's sensors.
- The complexity of the map representation has direct impact on the computational complexity of reasoning about mapping, localization, and navigation.

SLAM

Now we will talk about SLAM, **what is SLAM meaning?**

SLAM refers to simultaneous localization and mapping which is a technique that enables the robot to build and update a map and find a location in an unknown environment. SLAM helps the robot to navigate even with noisy data.

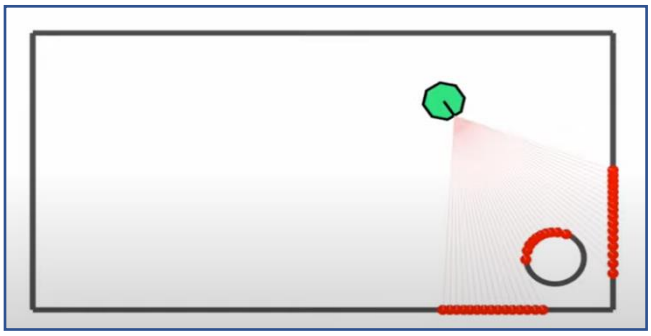
SLAM can be classified into two groups filtering and smoothing**. Filtering** like extended kalman filter (EKF) and particular filter which model the problem as online state estimation where the robot state and may be part of the environment is updated on the go as new measurements become available. **Smoothing techniques** estimate the full robot trajectory form complete set of measurement not just new measurements.

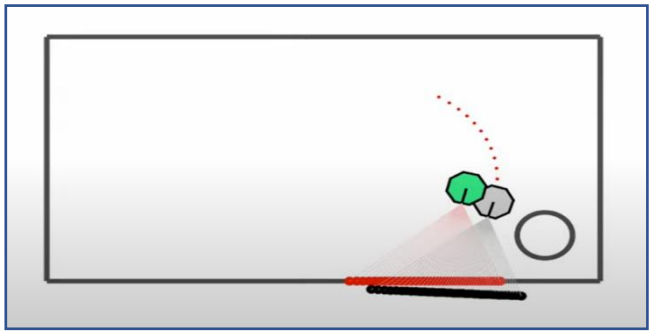
Pose graph optimization:

The robot takes measurements of environment and these measurements are associated with the current estimated position of the robot so we can add both to pose graph after the robot moves its estimated position deviates from real position expected and we take both measurements and position of robot and add them to pose graph although we don't know where are these two positions are in real environment but we have a knowledge about the distance between them which can be calculated from counting the wheel rotations we can rotate these positions relative to each other so we imagine that there is a constrain of rubber bar between these positions as there is uncertainty in the odometry of the robot and the strength of rubber bar depends on confidence of the odometry. In pose graph, we called poses as nodes of and rubber bar is called edge. Notice that this constraint is free to move in three dimensions x ,y and θ trying to get environment measurements back to its estimated distance it's difficult to do more with only two nodes and on edge so the robot keep moving and adds new estimated positions and its environment measurements to pose graph until it finally reaches the last position which usually shows the same environment measurements of the first position so we just need to know how to align these features so to figure out where these two poses have to be relative to each other which usually needs to be in same position so we add a new constraint between last and first poses and closes the loop of pose graph and the last edge (rubber bar) length will be zero so as will be shown in figures blow the blue link wants to pull two nodes, which are connected to it, together while the purple links wants to keep relative distances between

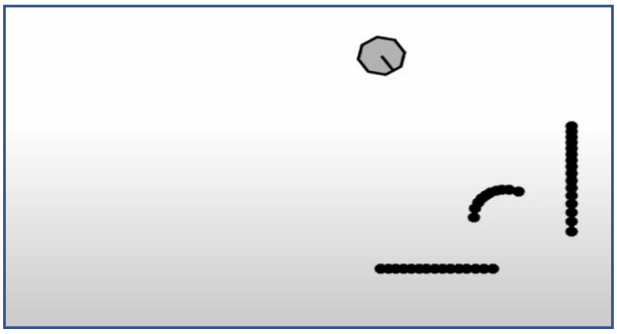
nodes as they are and so allowing the graph to settle to equilibrium and this is the optimization part of pose graph optimization. Although the resulting model isn't the perfect one but it still acceptable and if we want to make it more accurate, we just move robot in loop so as to create closure between nodes and reach to more accurate model at end. After we have the pose graph optimization, we can represent model by map using binary occupancy grid at which the environment is broken into grid of cells and if you believe the cell is occupied you set it to one and if not occupied you set it to zero and there is too probabilistic occupancy grid at which the cell don't have to be fully occupied or not it's just show the probability for cell to be occupied or not so black cell means that we have all confidence of cell to be occupied and if it's white, this means that we have all confidence that the cell is empty and if it is gray it means that cell may be empty or occupied with some probability. All of mentioned above is shown in figures blow.

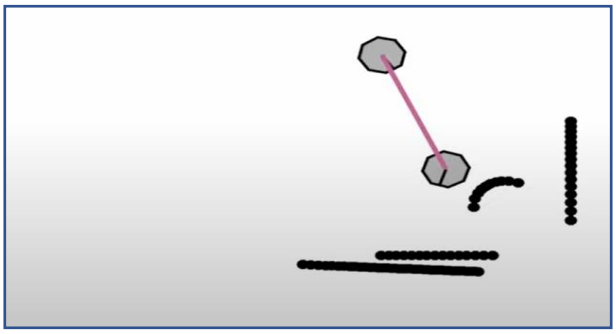
Real and estimated robot position

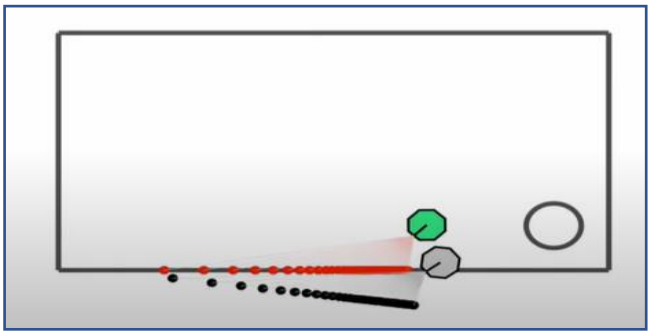
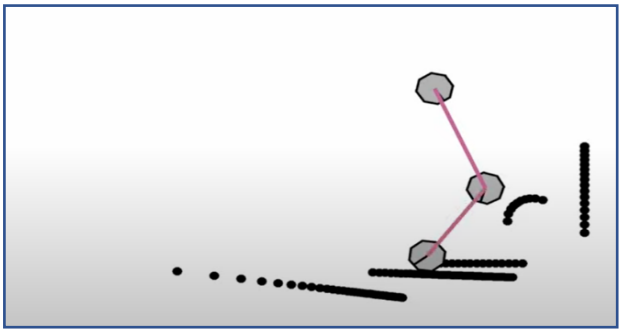
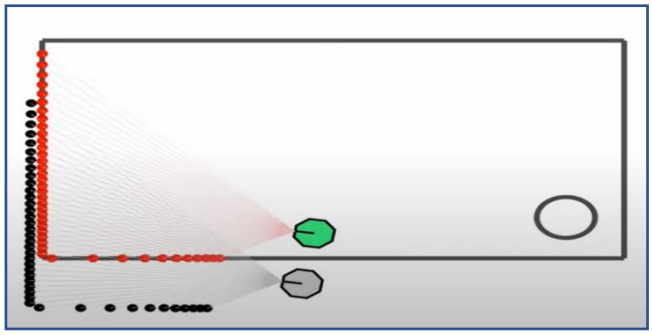
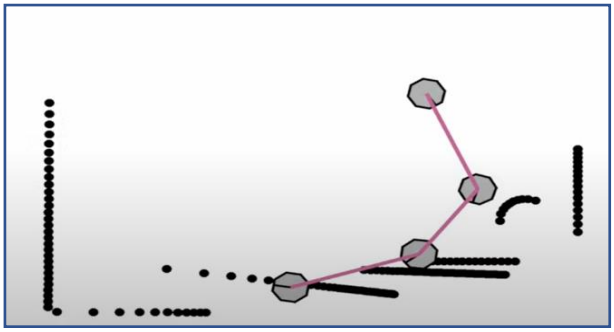
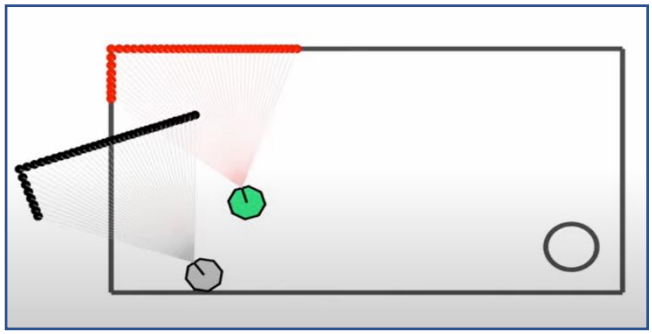
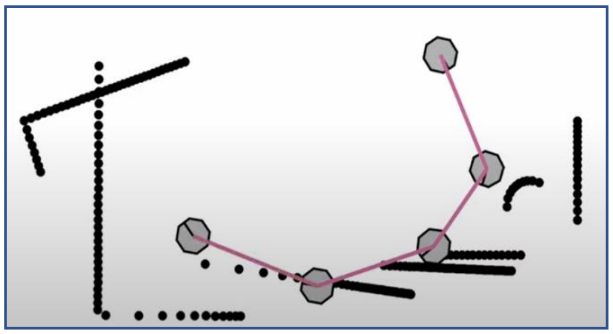
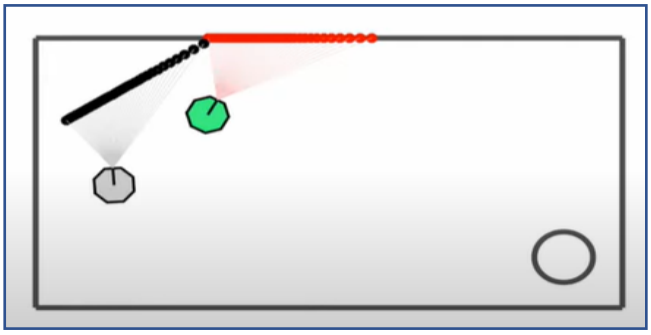
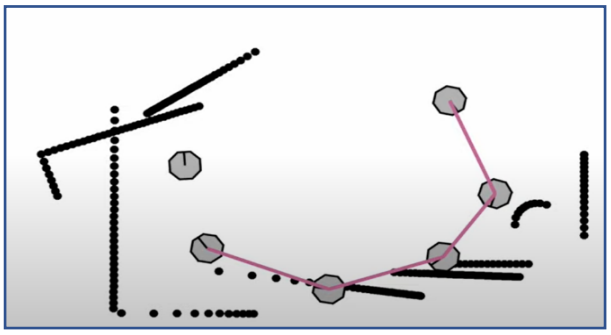
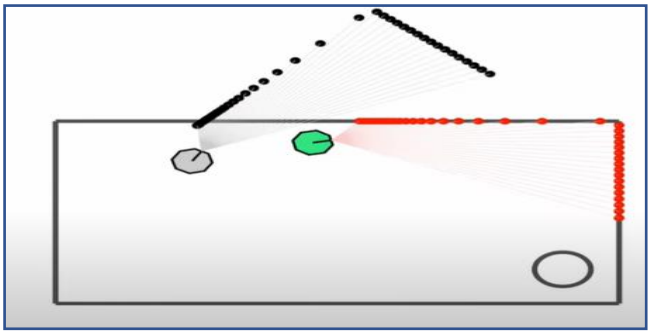
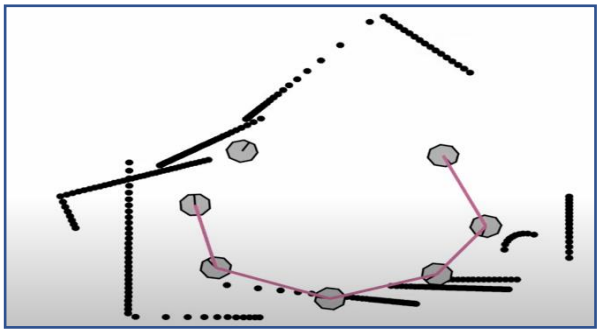


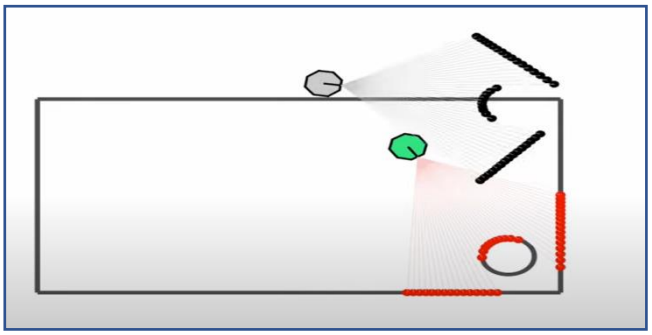


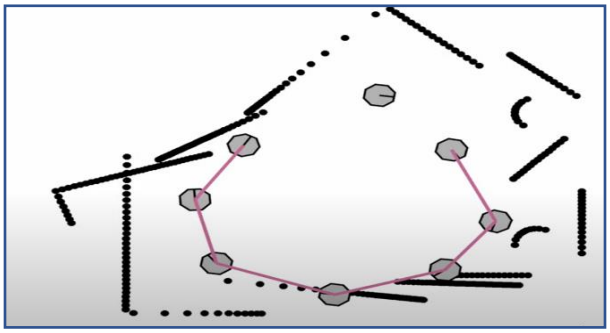
Pose graph optimization



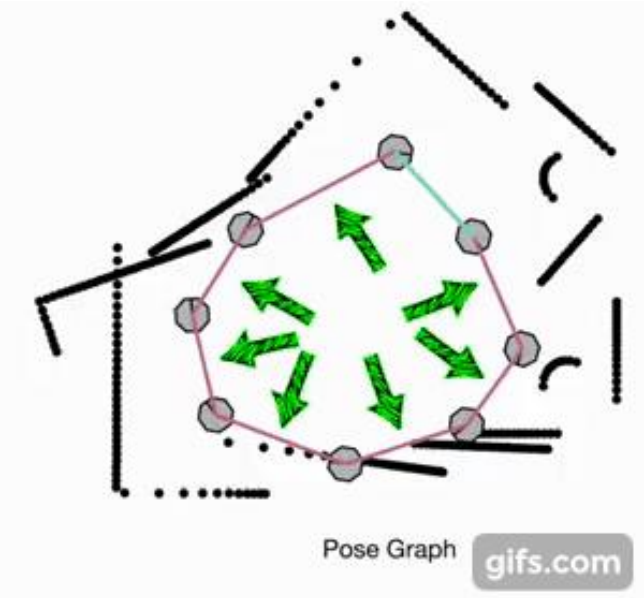


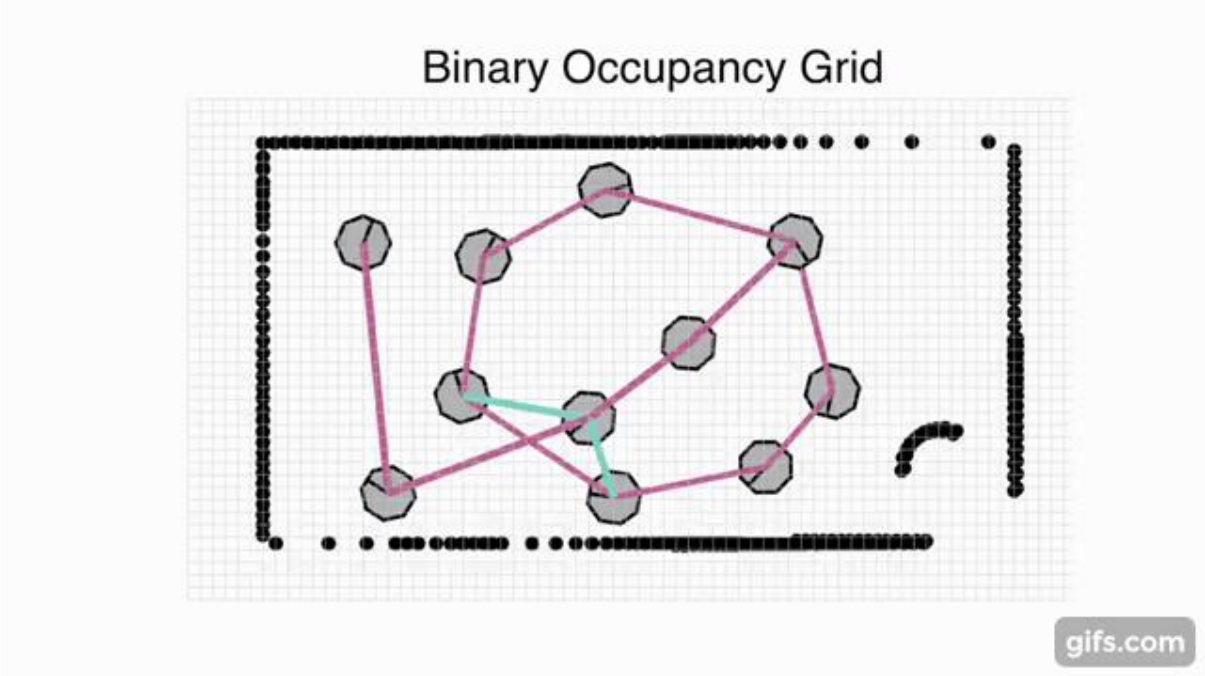




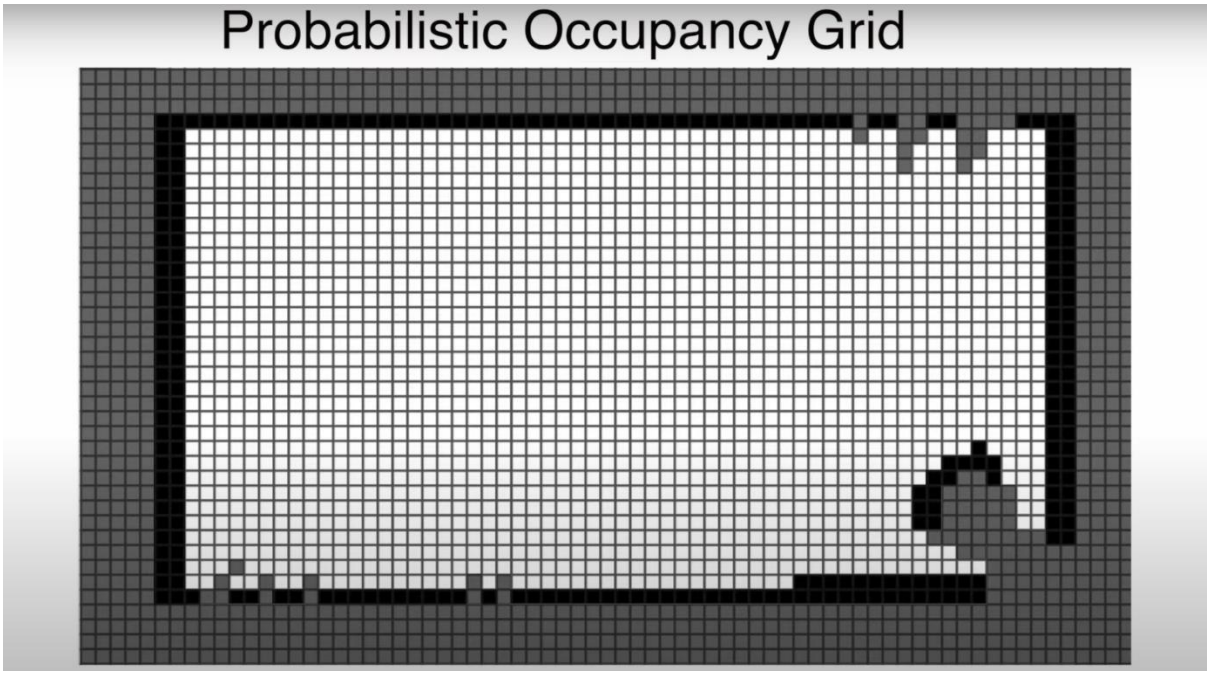
(6)These figures show robot estimated positions and environment measurements related to them for some movements of robot.



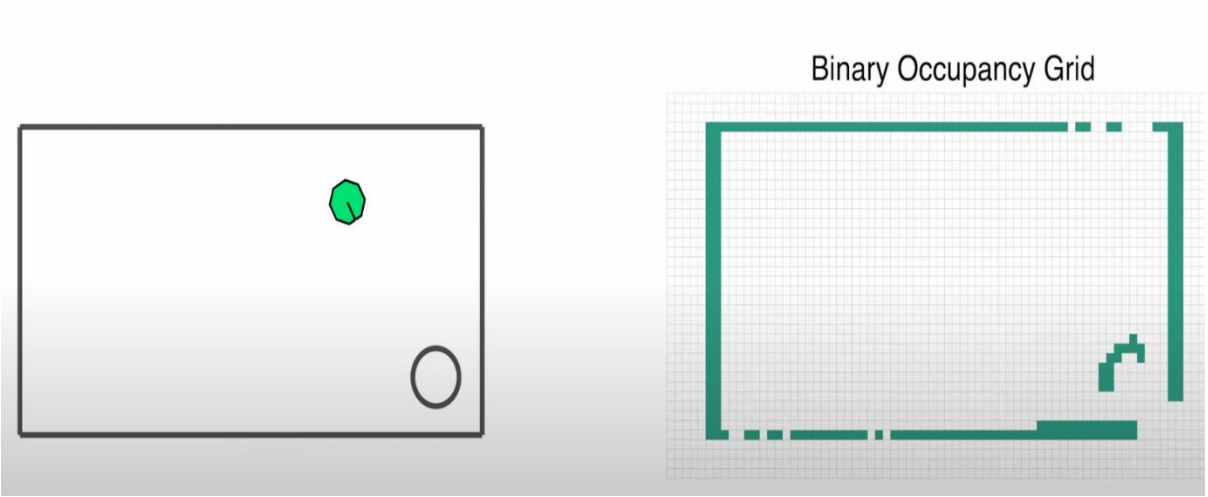
(7)This figure shows how to optimize pos graph to reach acceptable model.



(8)This figure shows how to build binary occupancy grid.



(9)Probabilistic occupancy grid



(10)The real environment and the model built by robot estimated positions and environment measurements.

Path Planning

After we have the map of the environment and we also have select the location of the smart chair (start point) now we want to move the smart chair from the start point to another point (from point A to point B) we must make the smart chair calculate the path .

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 navigation. So we will explain methodology to efficiently plan the trajectory of the smart chair in dynamic and complex environments, which it should traverse autonomously. A planner based in the A* and D* algorithms are used to plan a less costly trajectory to the destination. The methodology enables the chair to reach

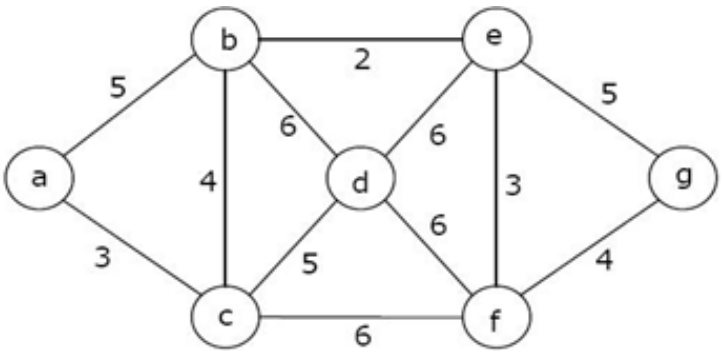
the goal using a local planner, which is applied to determine the speed and steering of the robot.

Path planning overview

Is calculation of the shortest path from the start point to the end point so we use AI algorithms to determine the shortest path that minimize the costs (time , distances)

Graph theory

- Graph theory: is a set of vertices (V) which represent nodes and a set of edges (E)
- Vertex: represents cells of the configuration space
- Edges : represent a line connects two vertices and show the relations between them
- 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.)



(11)Graph theory

Graph based planning algorithms

The set of algorithms that the computer uses it to guide the robot from its start location to its end location , and minimizing the total number of steps that are performed.

And these algorithms based on graph theory that have been explained in the previous part.

Dijkhstra algorithm

Goal: finding the shortest path from any vertex to all other vertices

Constraints : run for weighted graphs only and non-negative edge weights

Work funcution

Given a start node , I go from the start node to another node by choosing the shortest path (optimal) and after that it assigned as a visited node

Any node in the visited set already assigned their optimal distance now so I choose the node in visited set that is close to the unvisited set

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

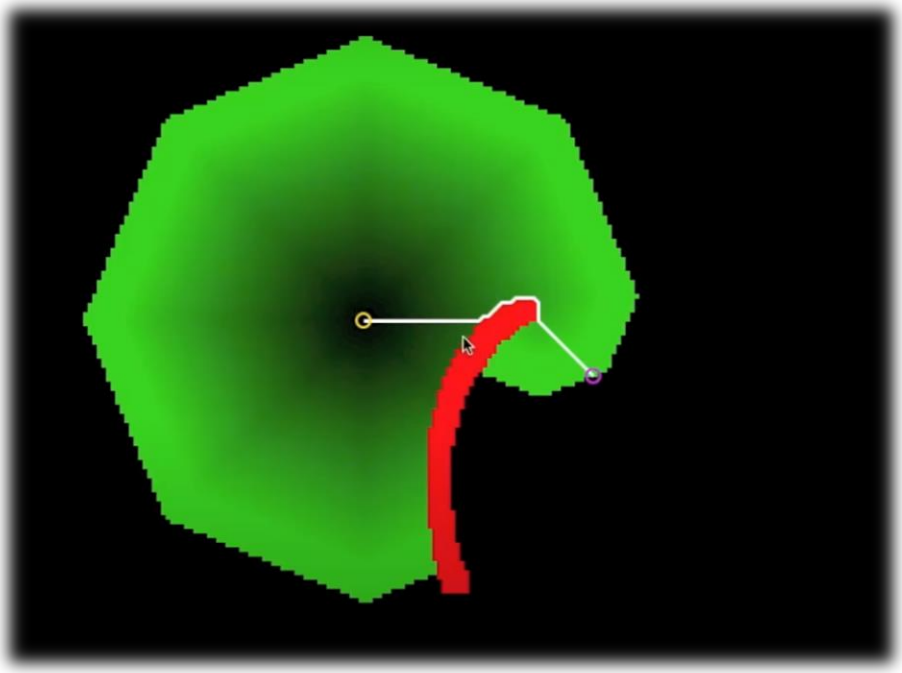
where we can delete a node and replace it with number of edges equal to (NO. of incoming edges * NO. of outgoing edges) then relax the graph (remove the heavy weighted edges).

PSEUDOCODE FOR DIJKSTRA'S ALGORITHM

- 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

THE PROBLEMS WITH USING DIJKESTRA

When we use dijkestra as a path planning algorithm with a robot it require a complex processing as to reach the goal node it explore all nodes in all directions which consume more time all increase the number of nodes.



(12)explored nodes by using dijkestra

A* ALGORITHM

To overcome the problems of the dijkstra algorithm we want more greedy algorithm that can reach the goal point with smallest cost and doesnot need to explore all nodes so we use A* algorithm.

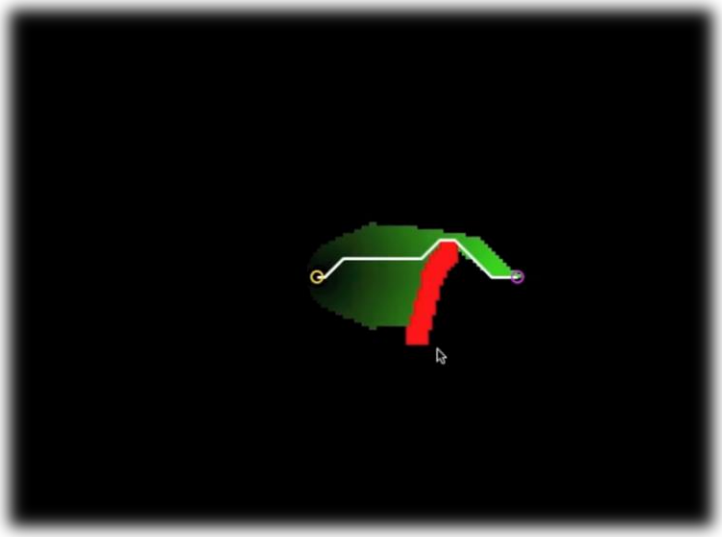
A* Search attempts to improve the performance of Dijkstra by incorporating a heuristic function that guides the path planner For consistent heuristics, the A* algorithm is similar to Dijkstra‘s algorithm. However, the inclusion of a heuristic function, which encodes additional knowledge about the graph, makes this algorithm especially efficient for single node to single node queries. In order to guarantee solution optimality, the heuristic is required to be an underestimating function of the cost to go. In robotics, A* is mainly employed on a grid, and the heuristic is then often chosen as the distance

between any cell and the goal cell in absence of any obstacles. If such knowledge is available, it can be used to guide the search toward the goal node. Generally, this dramatically reduces the number of node expansions required to arrive at a solution compared to Dijkstra‘s algorithm.

We use the A* algorithm with the global planner (static map)

PSEUDOCODE FOR 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

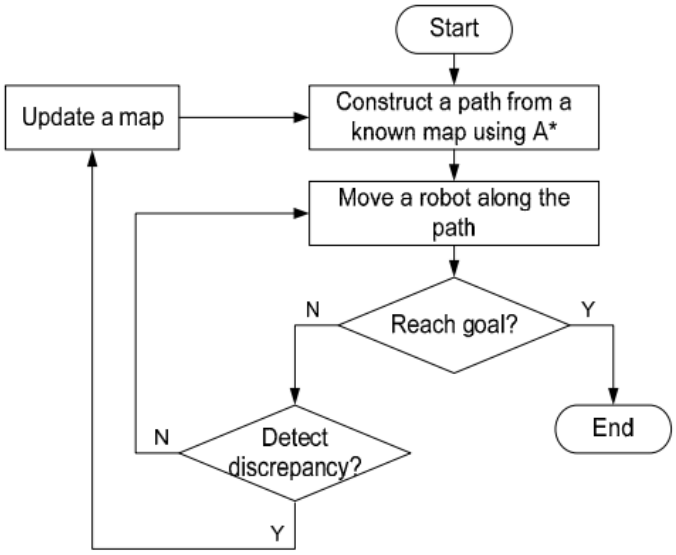


(13)explored nodes by using A*

Now we notice that the A* algorithm is more effective than dijkstra algorithm there is a small number of explored nodes

D* ALGORITHM

D* algorithm Stands for (Dynamic A* Search) as the dynamic refer to Arc cost parameters can change during the problem solving process (replanning online)



(14)replanner

WORK FUNCTION OF D* ALGORITHM

D* algorithm is reuse the algorithm of A* in repeating the incoming search iteration . so at the beginning if we have an environment and we want the chair to move from point A to point B then the chair will use the initial map(static map) it built it at the beginning of configuration the environment and starts to build the shortest path between the two point using the A* algorithm and start to move but continue to execute the path if it observes some changes in the environment by using the sensors it will add new object to the map and this require to create new path, so we use the D* algorithm to modify the old path that has been created with A* algorithm instead of creating new path from scratch and this will consume time.

.....

Chapter (3): ROS

What is ROS?

ROS (Robotics Operating System) is a software framework used for creating robotic applications. It is sometimes called a meta operating system because it offers many of the operating system functions but it needs to be run on an actual operating system.

ROS is an open source project so it can be used by anyone for any purpose whether this purpose is commercial or educational.

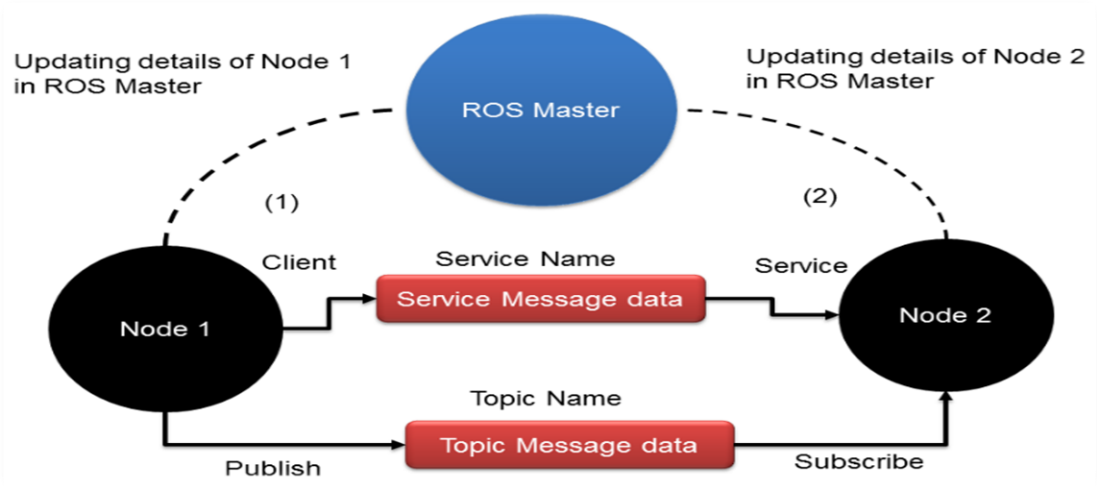
There are several distributions for ROS and each distribution is akin to a Linux distribution specifically Ubuntu distributions where Ubuntu is the official supported operating system for ROS.

ROS uses a P2P network for processing data and the main features are:

- **ROS nodes** which are just python or c++ scripts using ROS functionalities
- **ROS Master** which is the core of ROS and without it the nodes will not be able to find or talk to each other where this Master node provides name, registration, and lookup to the rest of the computational graph
- **ROS Topics** which are the tunnels connecting the nodes together where each node can publish data to this topic and other nodes can subscribe to this data and use it as desired
- **ROS Messages** which are the data that flows in the topics and topics can only contain one type of messages so a topic is defined by its name and its message type. We can create our own type of messages or use

predefined messages by ROS which can be just some primitive data types.

- **ROS Services** which are the request/reply type of interaction between nodes. One thing to be careful with when using services is that they are slow because ROS stops everything until the requesting node gets its response from the response node. Services also has a specific type of data that can be created or just use the predefined service message type.



(1)The previous graph shows the communication between nodes and master and also the communication between nodes using topics and services.

Why ROS?

ROS offers many advantages to make the development much easier and some of these advantages are:

- Hardware abstraction where developers can create a robot-agnostic applications that can be used with any robot

- Package management as nodes are collected in ROS packages and it is easy to create, build or install these packages
- Easy control of low level devices
- It is language independent as it can be programmed using python, c++ or Lisp
- There are many third-party libraries that are integrated with ROS such as Open-CV and PCL
- Some applications need a lot of computing power which can be a problem but with ROS the computing can be distributed among several machines

Installation

Every ROS distribution corresponds to a specific Ubuntu distribution for example ROS melodic is supported only on Ubuntu 18.04 and ROS kinetic is supported only on Ubuntu 16.04.

ROS are either LTS (Long Term Supported) or non-LTS. The distributions that are released on even number years are LTS while on odd number years are non-LTS. So which distribution should we install? Well, it's kind of a trade off. First of all LTS are the most commonly used because of the longed support time and so more stability but we still have to choose between the latest and older versions, this depends on if we want a longer support time or more stability where older versions had more time to be supported and so they are more stable but still much older versions like ROS Box Turtle distribution is a very bad choice because it is relatively very old and there is no support.

We choose kinetic as our distribution and that is because it is more stable even though the support ends this year by the time of writing this book and that is due to some problems we faced while using the melodic distribution and these problems are mentioned in later chapters.

To install ROS kinetic the operating system to be installed on should be Ubuntu 16.04 then follow the installation steps as follow:

1. Our system must be able to accept ROS packages from packages.ros.org and this is done by storing packages.ros.org in Ubuntu's apt repository list using the following commands:

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

2. To make ROS packages trusted, we have to add apt-key:

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

3. Now update the ubuntu package list:

$ sudo apt-get update

4. Now that everything is ready we can install ROS kinectic which can take some time:

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

5. After ROS kinetic is done installing, initialize rosdep so we can install system dependencies for the ROS packages:

$ sudo rosdep init

$ rosdep update

6. add ROS variable to the .bashrc file so that ROS commands and tools can be accessed from the shell:

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

7. The final step of installing ROS is installing a very usefull tool which enables us to install ROS packages with a simple command:

$ sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential

Now that we have installed ROS, we are ready to start using it!

Workspace

To create packages in ROS first we have to make a ROS workspace where the packages will be built inside of it.

To create a ROS workspace we will use the catkin build system which is the offical build system in ROS now and before it the offical build system was rosbuild.

Catkin combines Cmake macros and Python scripts to provide the same workflow that Cmake offers.

The workspace folder contains a src folder which contains all the packages that we build. To make a workspace first a folder (which is commonly named catkin_ws) should be created and inside of it another folder named src:

$ mkdir -p ~/catkin_ws/src

Now switch the directory to the src folder:

$ cd ~/catkin_ws/src

Then initialize the workspace:

$ catkin_init_workspace

After initializing the workspace, go back to the catkin workspace directory and use the command that builds the packages even if we don't have any packages to build yet:

$ cd ~/catkin_ws/

$ catkin_make

Before we only had the src folder inside the catkin workspace but now we should see a build and a devel folder. The build folder contains all the source code of the packages that we build and devel folder contains the shell script file to add the workspace on the ROS environment. If these two folders doesn't show that means a step is missing so please make sure you follow these steps in order.

ROS packages

Finally we have everything ready to start building our packages in ROS but that doesn't mean we can only build packages. There are so many pre-build packages in ROS that can be installed and be used directly and that is one of the main reasons why ROS is so powerfull because of the open-source community which helps in developing many powerful and useful packages.
To install any ROS package all we need to do is know the name of the package and then installing it using this command:

apt-get install ros-<name of distro>-<name of package>

sometimes it needs super used (sudo) to be able to download the packages.
We can now install packages but what if someone wants to create his own package and build it ? That's easy too, all we need to go to the src directory inside the catkin workspace and use the following command:

catkin_create_pkg <name of package> <dep1> <dep2> <dep3>

Where dep1, dep2 and dep3 are the dependencies of the package that you want to build.
After executing this command these should be a directory with the name of the package created inside src and inside this directory there are package.xml and CmakeLists.txt files. The package.xml file defines information about the package such as package name, version number, authors, maintainers, and dependencies on other packages while the CmakeLists.txt is the input to the Cmake build system for building packages where it describes how to build the code and where to install it.

To try out this package that have been created, refer to the following tutorial in ros wiki which explains how to make a simple publisher and subscriber where the publisher publishes a simple message to a topic and the subscriber subscribes or listens to this topic to receive the data that have been published http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29 and there is also a similar tutorial using c++ instead of python.

TF

In almost any robotics project specially projects with many movable joints and have many DOF it is so hard to keep track of all of the frames at all time and provide transforms between these frames and the more joints and links the project has the harder the transforms between them are.

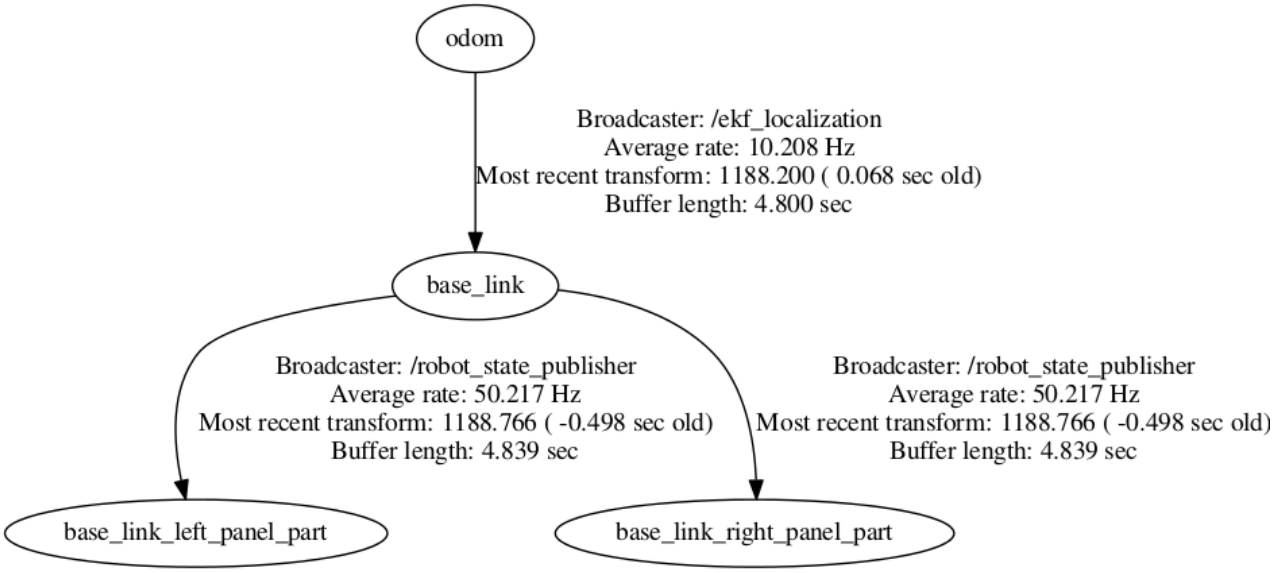
ROS provides many packages that makes the development much easier and one of these packages that is essential in almost any project is tf or the later version tf2.

This package lets the user keep track of multiple frames over time and it maintains relation between all frames in tree structure buffered in time.

In general, this package handles all the hard work of frame transformation, so we don't have to worry about it in our projects.

There are two tasks any user would use tf for which are:

- **Listening for transforms** which is receiving and buffering all the frames that are broadcasted in the system and search for a specific transform between frams.
- **Broadcasting transforms** which send out relative position of frames to all the system and a system can have more than one broadcaster each broadcaster provides information about different parts of the robot



The previous figure is an example of a tf tree and data related to each frame such as the broadcaster of this frame and most recent transform.

The are some helpful tf debugging tools and one of them is the previous tf tree which we can obtain using the following command:

$ rosrun tf view_frames This command produces a PDF file with a graphical representation of the tf tree as figure above.

Another helpful tool is tf_echo which echos a transform between two frames and it can be used using the following command:

$ rosrun tf tf_echo <reference frame> <target frame>

The result from this command looks like this:

```
At time 1253924110.083
- Translation: [-1.877, 0.415, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.162, 0.987]
            in RPY [0.000, -0.000, -0.325]
At time 1253924111.082
- Translation: [-1.989, 0.151, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.046, 0.999]
            in RPY [0.000, -0.000, -0.092]
....
```

Commands

ROS offers some commands related to packages, topics, services and others that help in making the development easier and faster and some of the most common commands are:

- ***roscd <name of package>*** This command changes the current directory into the directory of a package if this package is installed in our workspace.
- ***rospack list*** This command lists all the packages that have been installed in the catking workspace.
- ***roscore*** This command starts the ros master node which is needed for any ros project.

- ***rostopic list*** This command lists all the topics that are currently running in the ros network.
- ***rostopic echo <name of topic>*** This command subscribes or listens to a specific topic to see the data that is being published on this topic.
- ***rostopic info <name of topic>*** This command gets information such as the type of the topic (which is the message type being published on this topic), publishers who are publishing to this topic and subscribers subscribing to this topic.
- ***rosservice call <service name> <service args>*** This command call the service passing it the arguments it need for the services.
- ***rosservice list*** This command lists all the services currently available.

- ***rosservice info <service name>*** This command shows the information about a specific service.
- ***rosnode list*** This command lists all the nodes available.
- ***rosnode info <name of node>*** This command shows information about a node such as the topics it publishes or subscribe to or services it provides.
- ***rosrun <package name> <node name>*** This command runs a node but the name of the package have to be provided.

There are a lot more commands available but these are the most commonly used commands.

RVIZ

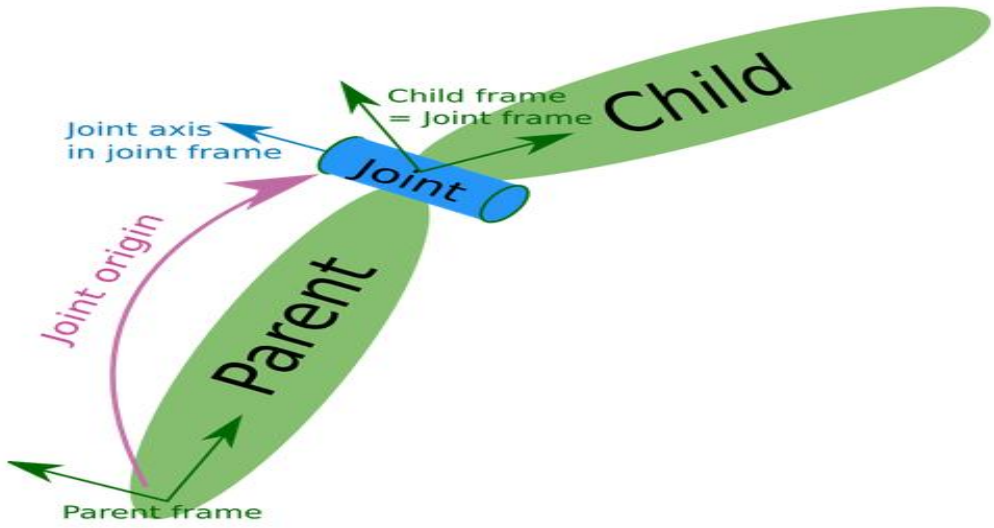
Every robot have some sensors to sense the surrounding environment and use it to calculate for example the odometry but it is hard to keep track of all these sensors with just numbers specially if they are laser sensors or cameras.

ROS provides rviz which is a very useful tool used to visualize the sensor data or cameras image stream and this tool is now essential in almost any ros project for debugging.

Links and joints

Before getting into rviz, first we need to grasp the understanding of links and joints and how we can make and visualize a robot model in ROS.

Any robot consist of many links which are connected together using joints and these joints can be moving or non moving joints.



URDF

In ROS we can build such links and joints to match our robot specs to make it easier to visualize and use it in simulations.

To build a robot model we use URDF (Unified Robotic Description Format) which is a xml format used to describe all elements of the robot.

The following shows how a URDF looks like:

```
<robot name=”name of robot”>
    <link name=”base_link”>
        <visual>
        </visual>
        <collision>
        </collision>
    </link>
</robot>
```

The urdf must start and end with a <robot> tag and inside of it there are mainly two other tags which are the <visual> tag and <collision> tag. In the <visual> tag as it is named is the visualization of the link and how we see it while the <collision> tag is for adding physical and collision properties to the URDF model.

Inside both the <visual> and <collision> tags we specify how this link should look like and there are three main shapes we can choose from which are cylinder, box and sphere. After choosing one shape we specify the dimensions of the shape. There is also the option to add a mesh file with our

own shape or design but it is recommended to add the mesh files only for the visual part not the collision part as it will be hard to process it.

Starting RVIZ

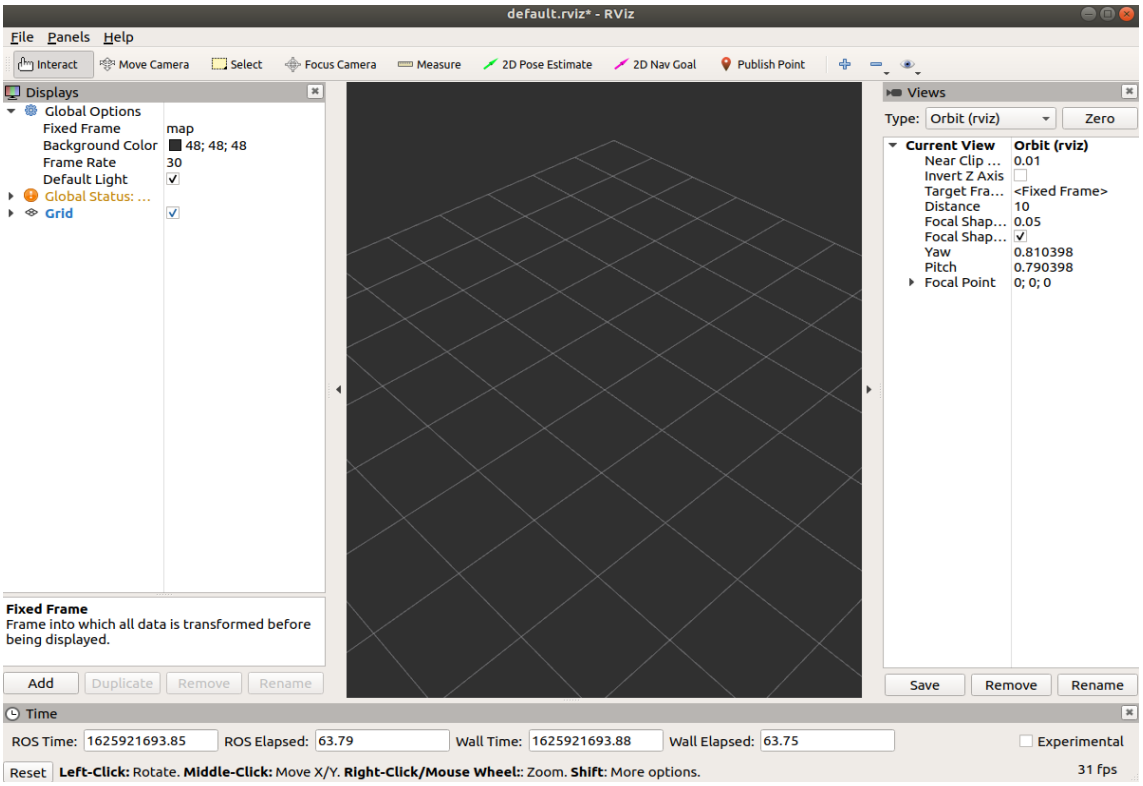
This robot description can be visualized using rviz. To start rviz we first need to run roscore then in the terminal run the command:

$ rosrun rviz rviz

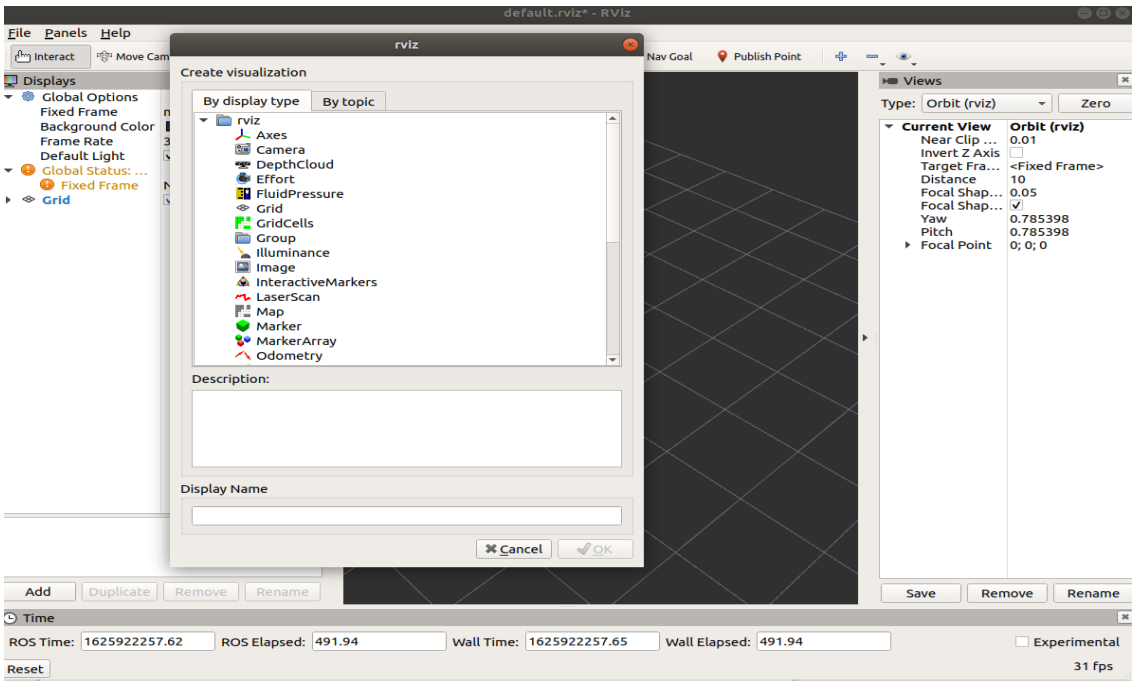
or simply just use

$rviz

The following figure shows the rviz window:



A fixed frame should be specified. For the previous figure the fixed frame is map frame.



We can add things such as senors, odometry or robot model from the add button on the bottom left.

Gazebo

Gazebo is an open-source 3D robotics simulator.

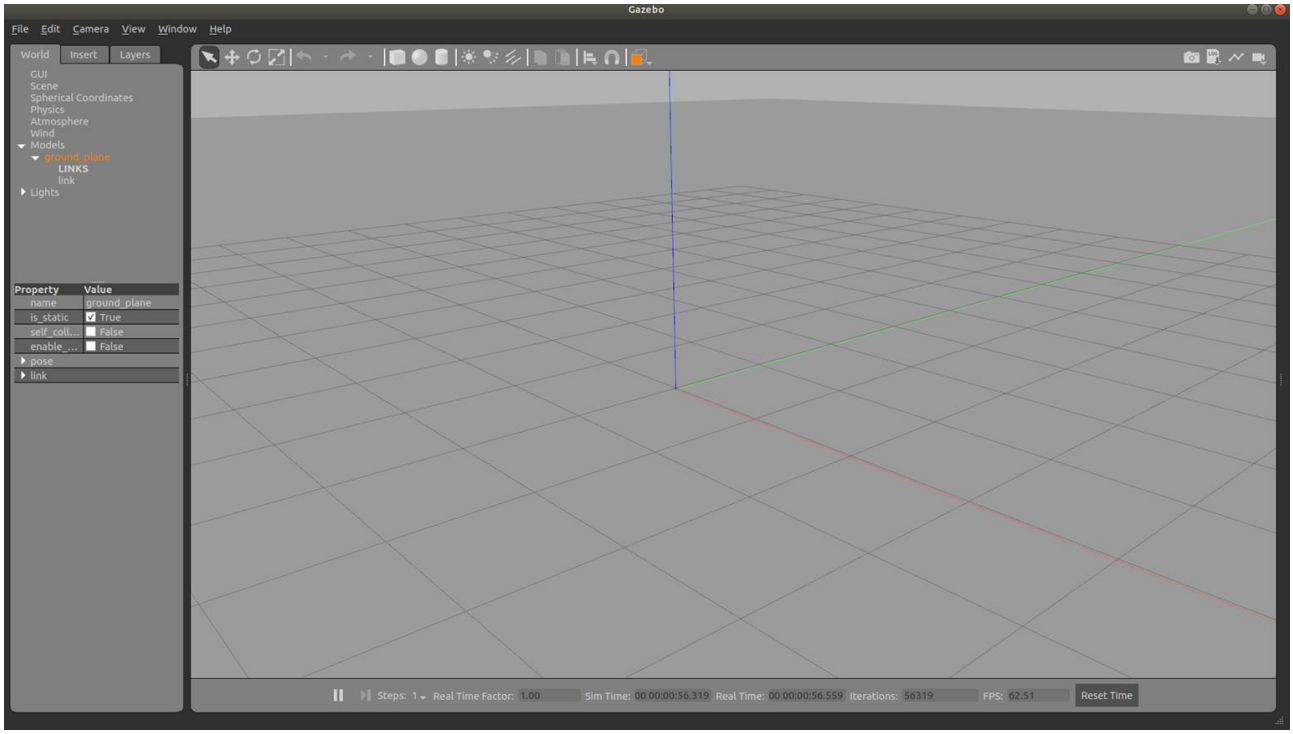
Robot simulation is important in the process of developing robots because it helps to test or make new algorithms or even train AI using realistic scenarios without having to build any actual robot so it saves a lot of time.

Start gazebo

To start gazebo simply just type gazebo in the terminal:

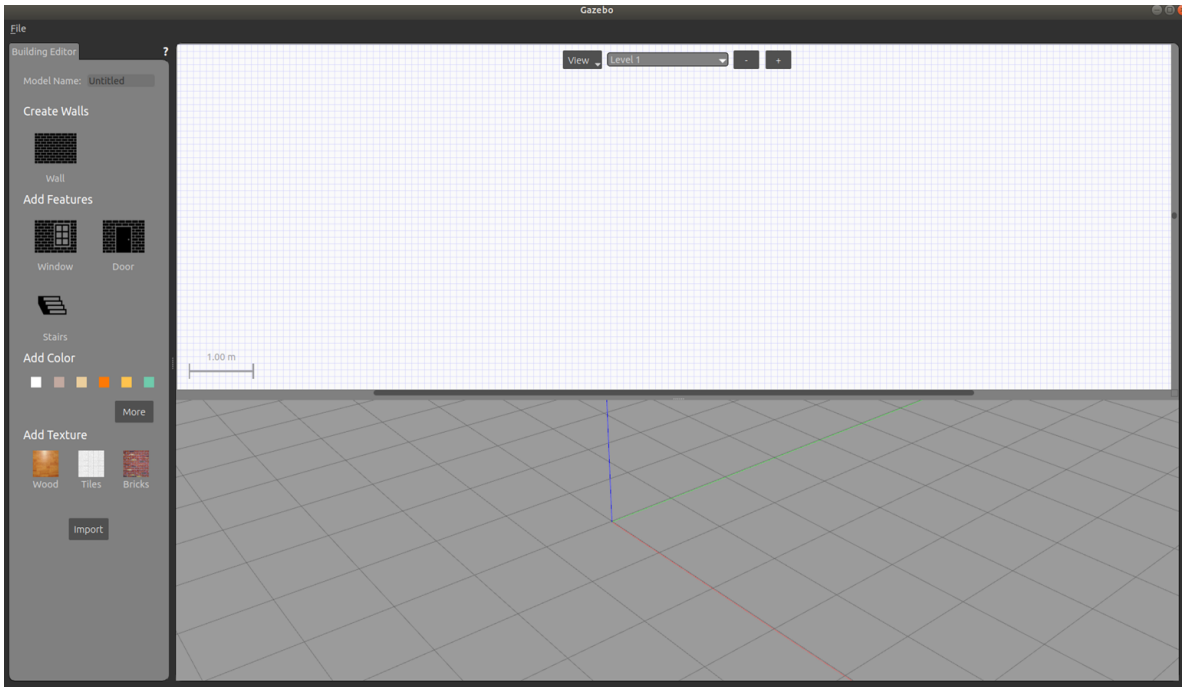
$ ***gazebo***

The following figure shows the gazbeo window:



Gazebo offers a lot of tools that help in building the simulation and one of them is the building tool where we can build our own environment to test our robot in it.

To start the building tool from edit choose building editor and the following should show:



Gazebo plugins

Gazebo is connected to ROS through a plugin called gazebo_ros_api_plugin and previously we ran gazebo only but to run gazebo connected to ROS we use the following command:

$ ***rosrun gazebo_ros gazebo***

And of course, **roscore** needs to be running.

Gazebo offers a lot of plugins and one of them is the plugin that connects ros with gazebo and there are other plugins to know more about gazebo plugins please refer to http://gazebosim.org/tutorials?tut=ros_gzplugins .

Simulation

A simulation is just the whole project or robot but without any hardware or wiring problems so all a person need to worry about or focus on is the software part of the project or robot.

Launch files and configuration files

ROS projects consist of a lot of nodes communicating with each other through topics or services.

It will be really hard to run each node separately in the terminal and then more terminals to monitor these nodes so it is almost impossible in large or projects.

There is a simple solution for this problem which is the ROS launch file.

These launch files also can be used to set parameters to each node if needed or load some parameters into the ROS parameter server which contains the robot description. Launch files can also include other launch files so it makes the management of the project easy.

Launch files are of .launch format and use a special XML format and can be placed within any directory inside the package but it is a common practice to make a launch directory and put all the launch files related to that package inside of it. The launch file start with <launch> tag and everything is included inside of it and at the end close the the tag.

To use launch files we use the following command:

$ roslaunch <package name> <launch file name>

Some nodes require a lot of parameters to be set and it can be messy if we set all these parameters inside the launch file so a more elegant way it to use configuration files to set all the parameters.

Configuration files are of .yaml format which is a light weight markup language that support all parameter types and can be loaded inside the launch file using the load command.

To load parameters into the ROS parameter server we use rosparam in launch files so we can load the config file using the <rosparam> tag inside the <node> tag in the launch file as follow:

```
<launch>

  <node name="name_of_node" pkg="name_of_package_containing_the_node"
  type="type_of_the_node">

        <rosparam command="load" file="path/to/the/config/file" />

  </node>

</launch>
```

This is a simple example of a launch file that launches a node and this node needs some parameters to be set so a config file is used to define these parameter then loaded using the load command inside the <rosparam> tag.

Instead of giving the full path to the file we can find the directory of the package then continue the path from the directory of the package:

```
<rosparam command="load" file="$(find package_name)/path/to/file" />
```

rosparam uses 1 to 1 correspondence between parameter server types and YAML types so this is how a config file looks like:

```
string: 'foo'

integer: 1234

float: 1234.5

boolean: true

list: [1.0, mixed list]

dictionary: {a: b, c: d}
```

Smart-Wheelchair simulation

So far we have explained everything needed to make an autonomous indoor navigation robot from SLAM and path planning to ROS and all tools needed to make a full simulation.

The link to our simulation: https://github.com/Ahmed-Osman-01/Smart-Wheelchair

After downloading this package, move it to the src file in your workspace and then install it using catkin_make. To launch the simulation use the following command:

```
$roslaunch project project.launch
```

project.launch is the main launch file of the project as it includes all the other launch files of the project.

Now lets start explaining the simulation part by part.

Robot description

The first thing we need to have is a full description for our robot to be able to simulate and visualize it.

The URDF of our project: https://github.com/Ahmed-Osman-01/Smart-Wheelchair/blob/master/urdf/robot.urdf.xacro

This URDF is written in the xacro format. Xacro makes the URDF cleaner and easier to modify using three main things:

- Constants
- Simple Math
- Macros

At the top of the URDF, we must specify a namespace in order for the xacro file to parse properly as follow:

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

The next few lines are constants specified using the <xacro:property tag> which makes it easier to modify any values in the file without having to search for where it was.

After that we included robot.gazebo file which contains the gazebo plugins needed for the simulation using the <xacro:include> tag.

The <material> tag is used to make the visualization in rviz look better as we can add a color to each link after defining this color using the <material> tag.

The links and joints of the robot are then defined. There is an additional <inertial> tag added to each link as gazebo requires each link to have this tag

specifying the inertia and mass of the link so the physics of the link can be simulated.

The <origin> tag specifies where the link is relative to the joint connecting this link and the orientation of this link.

<xacro:macro name="wheel" params="prefix_x prefix_y reflect_x reflect_y">

The previous line uses the macro property of xacro. Macros is one of the great advantages of using xacro instead of normal URDF as it can be used to repeat a part of the robot with just calling this macro without having to re-write the whole part again.

Inside this macro is the robot wheels where prefix_x is the position of the wheel relative to the x-axis (front or back) and prefix_y is the position of the wheel relative to the y-axis (right or left). reflect_x and reflect_y is used in calculating the position of the wheel in the following line:

<origin xyz="${reflect_x((box_length/2)-0.0)-(reflect_x*(wheel_diam/2))}*
${reflect_y(((wheel_len))+(box_width/2))} -0.07" rpy="0 0 0"/>*

This line uses the math property of macro which can be used by adding the math between curly brackets and put a $ sign before it.

The following two lines calls the wheels macro:

<xacro:wheel prefix_x="front" prefix_y="right" reflect_x="1" reflect_y="-1"/>
<xacro:wheel prefix_x="front" prefix_y="left" reflect_x="1" reflect_y="1"/>

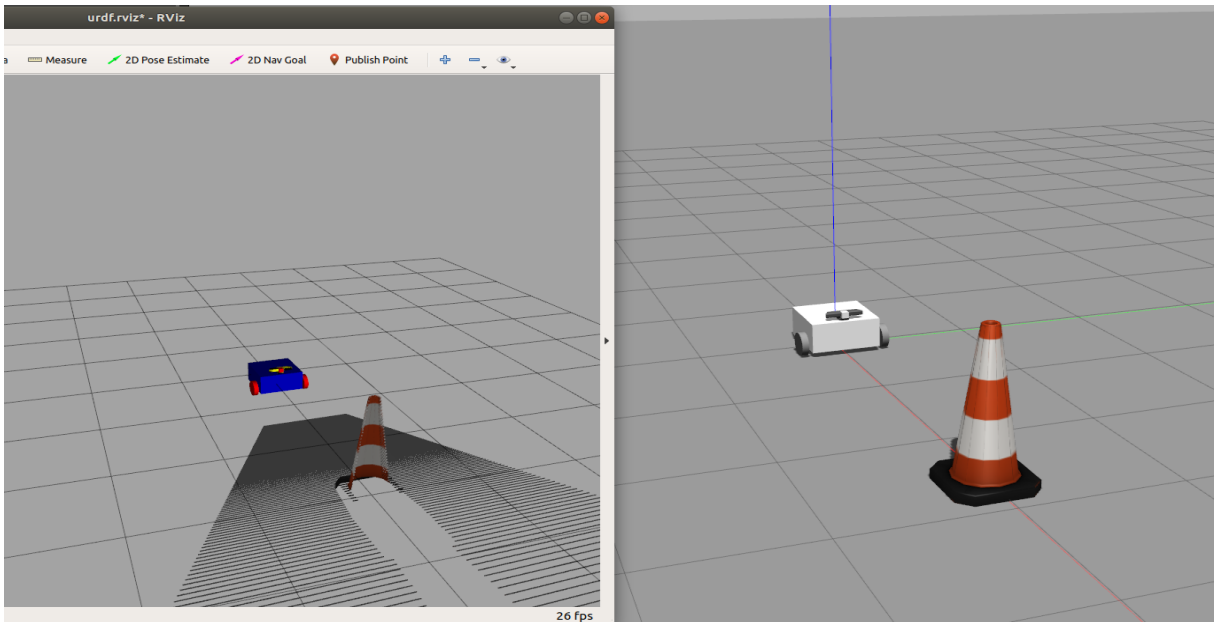
reflect_x takes value 1 for front wheels and -1 for back wheels while reflect_y takes value 1 for left wheel and -1 for right wheel.

For this simulation we will use only 2 wheels as our actuators and replace the back wheels with a caster.
Since we will control the robot through gazebo so it needs to know what to do with the actuators using the <transmission> tag and for more about the transmission please refer to http://wiki.ros.org/urdf/XML/Transmission .

Some gazebo physical parameters needs to be specified using the <gazebo> and we can also add colors to the simulated robot in gazebo using <material> tag inside the <gazebo> tag.
At the end of the file, kinect and IMU are added. There is an extra link called kinect_optical and it is needed because the output of the normal kinect link when used it rotated (frames of kinect output image is different) so we add this link and rotate it relative to the normal kinect frame and then use kinect_optical as our main kinect_frame and pass to gazebo kinect plugin this frame instead of the normal kinect frame.

The IMU and kinect data are produced by gazebo using the gazebo plugins mentioned previously and saved in file called robot.gazebo and this file is included at the top of the xacro file.

This is how the final model looks like in rviz and gazebo and using rviz to visualize the point cloud data published in the camera/depth/points topic buy the kinect plugin in gazebo:



Now to be able to move and control the robot we need a controller for the wheels and this is done by the joint _state_controller and dif_drive_controller in the controllers.yaml in the config directory and this controller is launched in the controllers.launch in the launch directory. For more information about the ros controllers please refer to http://wiki.ros.org/ros_control .

The diff_drive_controller calculates the odometry of the robot using only the feedback from the motors. This odometry, just like any other odometry calculated from the motors feedback alone, is not accurate

and the error in the odometry data increases exponentially specially when the robot turns.

This is why in most navigation applications whether it is outdoor or indoor IMU is used.

Robot localization

IMU gives much more accurate readings when it comes to orientation compared to that calculated by the motors feed back so we can use this fact to make the odometry data more accurate.

IMU data can be fused with the odometry data calculated from motors feedback using ekf (extended kalman filter) and there is a ROS package that handles this job called robot_localization.

When using ROS packages we don't need to worry about the source code and how it is implemented (but reading the source code can be also helpful). Every ROS package has a documentation on ros wiki (ros official site) which contains information such as nodes, parameters, published topics and published transforms.

The parameters of robot_localization is set in the configuration file localization_config.yaml inside the config directory in the localization directory:https://github.com/Ahmed-Osman-01/Smart-Wheelchair/blob/master/localization/config/localization_config.yaml .

The most important parameters are:

- **two_d_mode** this is set to true if we are using a robot that moves in 2D (planer movement) only so it fuses 0 to all the 3D variables of fused sensors.
- **frequency** this is the frequency of publishing data.
- **publish_tf** if this is set to true, it publishes the transform from the frame specified by world_frame to the frame specified by the base_link_frame.
- **odom_frame** the name of the odometry frame.
- **base_link_frame** the name of the base_link frame.
- **map_frame** the name of the map frame (remove it if there's not map frame).
- **world_frame** this is set to map frame if it exist and if it doesn't it is set to the odometry frame.

Now we need to configure the sensors that need to be fused and in our case we want to fuse odometry with IMU:

- **odom0** we can have multiple odometry data to fuse that is why we need to number the odom. This is set to the topic that odom data is being published to.
- **odom0_config** this takes an array containing (X,Y,Z,$roll$,$pitch$,yaw,$\dot{X}$,$\dot{Y}$,$\dot{Z}$,$\dot{roll}$,$\dot{pitch}$,$\dot{yaw}$,$\ddot{X}$,$\ddot{Y}$,$\ddot{Z}$) in the same order and each one can take a value of true or false. So this configures what

data to fuse and in odom case we only need to fuse linear velocity in X and Y and angular velocity in Z.

- **imu0** This is set to the topic when imu data is being published on.
- **imu0_config** This is the same as the odom config, it takes an array containing 15 element, each element can be set true or false. We will set the angular velocity in Z and linear acceleration in X to true and the others to false.

There are also **process_noise_covariance** & **initial_estimate_covariance** but they are very hard to tune and the defualt values works just fine for most of the projects.

Now that we have finished setting the parameters in configuration file, we can launch and start using this package using the localization.launch file inside the launch directory in localization https://github.com/Ahmed-Osman-01/Smart-Wheelchair/blob/master/localization/launch/localization.launch

In the launch file all we need to do is launch one of the localization nodes which are ekf and ukf (unscented kalman filter).
We launch ekf_localization_node from the package and load the configuration file inside the node.

For more detailed information about robot_localization package refere to http://docs.ros.org/en/noetic/api/robot_localization/html/index.html .

SLAM package in ROS

Gmapping package

introduction

Gmapping package contains ROS wrapper for Open's Slam Gmapping. It provides a laser-based SLAM as a ROS node called slam_mapping. By using slam_mapping node we can create 2D occupancy grid map from laser and pose data collected by mobile robot as explained at pose graph optimization part at navigation chapter. It's a highly efficient Rao-Blackwellized particle filter to learn grid maps from laser range data. The particle filter used to reduce number of particles which are taken in the prediction step of the filter and this is done by taking into account not only robot movement but also it takes the most recent observations. To use slam_gmapping, you need a mobile robot that provides odometry data and is equipped with a horizontally-mounted, fixed, laser range-finder. The slam_gmapping node will attempt to transform each incoming scan into the odom (odometry) tf frame.

Package source

Gmapping package can be found at github through this link https://github.com/ros-perception/slam_gmapping

Package inputs

The approach takes raw laser range data and odometry. This version is optimized for long-range laser scanners like SICK LMS or PLS scanner. Short

range lasers like Hokuyo scanner will not work that well with the standard parameter settings.

Node

slam_gmapping node which takes in LaserScan message and builds a map (occupancyGrid message). The map can be retrieved via a ROS topis or service.

Subscribed topics

1. Tf(tf/tfMessage): Transforms necessary to relate frames for laser, base, and odometry.
2. scan (sensor_msgs/LaserScan): laser scans to create the map.

Published topics

1. map_metadata (nav_msgs/MapMetaData): get the map data from this topic which is updated periodically.
2. map (nav_msgs/OccupancyGrid): get the map data from this topic which is updated periodically.
3. ~entropy (std_msgs/Float64): Estimate of the entropy of the distribution over the robot's pose (a higher value indicates greater uncertainty).

Services

dynamic_map (nav_msgs/GetMap): get the map data.

Messages

1. **LaserScan message** shows single scan from planer laser range-finder and it contains:

Header at which timestamp is the acquisition time of the first ray in the scan.

float32 angle_min defines start angle of scan in radians.

float32 angle_max end angle of scan in radians.

float32 angle_increment defines angular distance between measurement in radians.

float32 time_increment defines time between measurements in seconds.

float32 scan_time defines time between scans in seconds.

float32 range_min defines minimum range value in meters.

float32 range_max defines maximum range value in meters.

float32[ ] ranges defines range data in meters.

float32[ ] intensities defines intensity data of device.

2. **OccupancyGrid message** which represents a 2D grid map, in which each cell represents the probability of occupancy and it contains:

   Header

   MapMetaData info defines meta data for the map.

   int8[ ] data defines occupancy probabilities in range [0,100] and unknown is -1.

3. **tfMessage** contains:

   geometry_msgs/TransformStamped[] transforms

4. **Float64 Message** contains:

   Float64 data

5. **MapMetaData Message** holds basic information about occupancy grid and contains:
   time map_load_time defines the time at which map is loaded.
   float32 resolution defines the map resolution meters per cell.
   unit32 width defines map width in cells.
   unit32 height defines map height in cells.
   Geometry_msgs/pose origin defines the origin of map.

Important parameters

1. **maxRange(**float**)** shows the maximum range of the sensor and it should be set to slightly higher than the maximum real range.
2. **maxUrange(**float**)** defines the maximum usable range, the beam is cropped to this value.
3. **minimumScore(**float**)** is set to consider laser reading.
4. **linearUpdate(**float**)** process scan each time the robot translates given distance.
5. **angularUpdate(**float**)** process scan each time the robot rotates given angle.

For more parameters see gmappin (http://wiki.ros.org/gmapping)

Localization package in ROS

After building a map we give it to the robot, but the robot doesn't now it's place within the map, so the robot needs to use its sensors to determine its position and orientation within the building. Readings which robot get from sensors are noisy and it has also a weighted dead reckon in its position using odometry data. We don't use Kalman filter because it deals with gaussian

probability distribution and this isn't the case for localization problem so we used a filter that can handle non gaussian probability distribution and that was particle filter.

Particle filter

This filter used because we don't know the position of the robot so it generates a number of particle (positions) and compare their measurements with the measurement of what robot see then giving the most liker particles high probability while giving other particles low probability then it replaces particles to be in positions of higher probability areas and move the robot and calculate new measurements to determines which particle has higher probability and which isn't and keep repeating this method with reducing number of particles at each new step to decrease computational resources and this is done by adaptive monte carlo localization (AMCL). If we don't remove the low probability particles, they will consume computational resources and this will cause the filter to never converge therefore we resample particles according to new probability distribution and as I said above this will cause more particles at places that are more likely and less particles everywhere else. This random discrete resampling of the probability distribution this why this type of localization is also referred as monte carlo localization.

AMCL Package

Introduction

AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach which uses a particle filter to track the pose of a robot against a known map. It allows robot to use particle filter algorithms to determine its location on map as the robot doesn't know where it is.

Package source

amcl package can be found at github through this link (https://github.com/ros-planning/navigation).

Node

amcl node which takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates. On startup, amcl initializes its particle filter according to the parameters provided. Note that, because of the defaults, if no parameters are set, the initial filter state will be a moderately sized particle cloud centered about (0,0,0).

Subscribed topics

1. Tf(tf/tfMessage): Transforms necessary to relate frames for laser, base, and odometry.
2. initialpose (geometry_msgs/PoseWithCovarianceStamped) : Mean and covariance with which to (re-)initialize the particle filter.

3. scan (sensor_msgs/LaserScan): laser scans to create the map.
4. Map(nav_msgs/OccupancyGrid): when the use_map_topic parameter is set, AMCL subscribes to this topic to retrieve the map used for laser-based lacalizatio.

Published topics

1. amcl_pose (geometry_msgs/PoseWithCovarianceStamped): robot estimated pose in the map, with covariance.
2. particlecloud (geometry_msgs/PoseArray): the set of pose estimates being maintained by the filter.
3. tf (tf/tfMessage): publishes the transform from odom to map.

Services

1. global_localization (std_srvs/Empty): initiate global localization, wherein all particles are dispersed randomly through the free space in map.
2. request_nomotion_update (std_srvs/Empty): Service to manually perform update and publish updated particles.
3. set_map (nav_msgs/SetMap): Service to manually set a new map and pose.
4. static_map (nav_msgs/GetMap)(called service): amcl calls this service to retrieve the map that is used for laser-based localization; startup blocks on getting the map from this service.

Messages

A. LaserScan message shows single scan from planer laser range-finder and it contains:

Header at which timestamp is the acquisition time of the first ray in the scan.

float32 angle_min defines start angle of scan in radians.

float32 angle_max end angle of scan in radians.

float32 angle_increment defines angular distance between measurement in radians.

float32 time_increment defines time between measurements in seconds.

float32 scan_time defines time between scans in seconds.

float32 range_min defines minimum range value in meters.

float32 range_max defines maximum range value in meters.

float32[ ] ranges defines range data in meters.

float32[ ] intensities defines intensity data of device.

B. OccupancyGrid message which represents a 2D grid map, in which each cell represents the probability of occupancy and it contains:

Header

MapMetaData info defines meta data for the map.

int8[ ] data defines occupancy probabilities in range [0,100] and unknown is -1.

C. tfMessage contains:

geometry_msgs/TransformStamped[] transforms

D. **poseWithCovaranceStamped Message** expresses an estimated pose with a reference coordinate frame and timestamp and contains:

Header header

poseWithCovarance pose

E. **poseArray Message** defines an array of poses with a header of global reference and contains:

Header header

Pose [ ] poses

- **important parameters** (http://wiki.ros.org/amcl**)**
  1) **~odom_model_type**(string) Which model to use, either "diff", "omni", "diff-corrected" or "omni-corrected".
  2) **~odom_frame_id**(string): which frame to use for odometry.
  3) **~base_frame_id**(string): which frame to use for robot base.
  4) **~glaobal_frame_id**(string): the name of coordinate frame published by localization system.
  5) **~use_map_topic**(bool): When set to true, AMCL will subscribe to the map topic rather than making a service call to receive its map.
  6) **~min_particles**(int)**:** minimum allowed number of particles.
  7) **~max_particles**(int): maximum allowed number of particles.

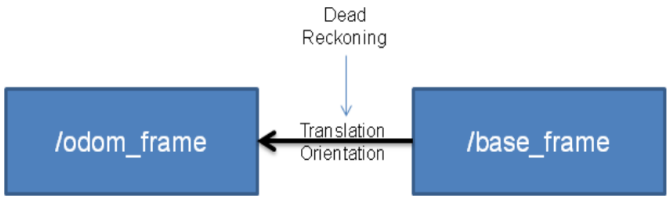
8) **~kid_err**(double): Maximum error between the true distribution and the estimated distribution.
9) **~update_min_d**(double): Translational movement required before performing a filter update.
10) **~update_minimum_a**(double): rotational movement required before performing a filter update.
11) **~resample_interval**(int): Number of filter updates required before resampling.
12) **~transform_tolerance**(double): Time with which to post-date the transform that is published, to indicate that this transform is valid into the future.
13) **~gui_publish_rate**(double): Maximum rate (Hz) at which scans and paths are published for visualization, -1.0 to disable.

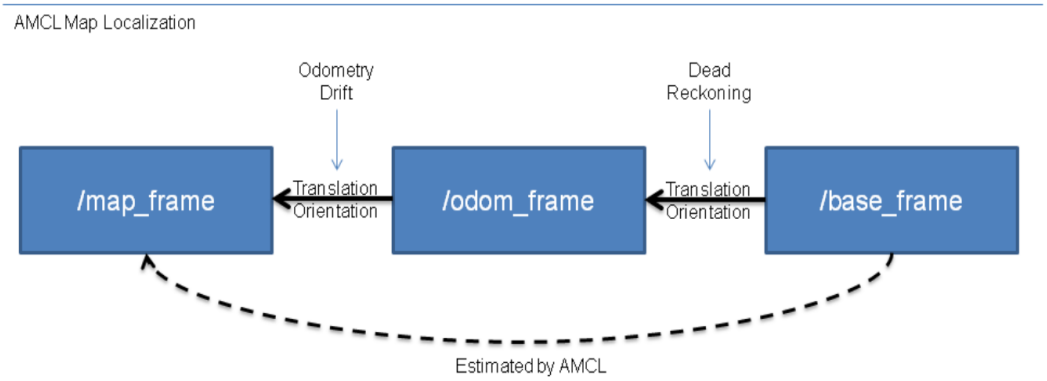
Transforms

amcl transforms incoming laser scans to the odometry frame (~odom_frame_id). So there must exist a path through the tf tree from the frame in which the laser scans are published to the odometry frame.

An implementation detail: on receipt of the first laser scan, amcl looks up the transform between the laser's frame and the base frame (~base_frame_id), and latches it forever. So amcl cannot handle a laser that moves with respect to the base.

The drawing below shows the difference between localization using odometry and amcl. During operation amcl estimates the transformation of the base frame (~base_frame_id) in respect to the global frame (~global_frame_id) but it only publishes the transform between the global frame and the odomtry frame (~odom_frame_id). Essentially, this transform accounts for the drift that occurs using Dead Reckoning. The published transforms are future dated.





Difference between localization using odometry and amcl

Path Planning In ROS

To visualize the path planning in the RVIZ we use three main part

1. global planner
2. map (global cost map)
3. local planner
4. map (local cost map)

THE GLOBAL PLANNER

When a new goal come to the move base node. this goal is immediately sent to the global planner. Then the global planner starts calculating a safe path using A* algorithm in order to arrive to the goal pose this path is calculated before the robot starts moving .so It will not take into account the readings that the robot sensors are doing while moving

THE MOVE_BASE PACKAGE

The move_base package contain the **move_base node** and the main function of the move_base node is to move the robot from its current position to a goal position, this node is an implementation of SimpleActionServer which take a goal pose with message type geometry_msgs/PoseStamped, therefore we can send position goal to this node by using a SimpleActionClient .

This action server provides the topic move_base/goal which is the input of the navigation stack. this topic is then used to provide the goal pose.

Move_base node: The move_base node provides a ROS interface for configuring, running, and interacting with the navigation stack on a robot. A high-level view of the move_base node and its interaction with other components is shown above.

MOST IMPORTANT PARAMETERS

1/ ~base_global_planner (string, default: "navfn/NavfnROS" For 1.1+ series)

- The name of the plugin for the global planner to use with move_base, see pluginlib documentation for more details on plugins. This plugin must adhere to the nav_core::BaseGlobalPlanner interface specified in the nav_core package. (1.0 series default: "NavfnROS")

2/ ~controller_frequency (double, default: 20.0)

- The rate in Hz at which to run the control loop and send velocity commands to the base

3/ ~planner_patience (double, default: 5.0)

- How long the planner will wait in seconds in an attempt to find a valid plan before space-clearing operations are performed.

4/ ~planner_patience (double, default: 5.0)

- How long the planner will wait in seconds in an attempt to find a valid plan before space-clearing operations are performed.

THE GLOBAL COST MAP

The global costmap is created from the static map that have been created at first so the costmap is initialized to match the width, height, and obstacle information provided by the static map. This configuration is normally used in conjunction with a localization system

So until now we have seen how ROS plans a trajectory in order to move a robot from a starting position to a goal position in a direct way without any changing in the environment .now we will learn how ROS executes this trajectory, and avoids obstacles while moving.

THE LOCAL PLANNER

After the global planner has calculated the path to follow, this path is sent to the local planner. The local planner will execute each segment of the global plan so we can consider the local plan as a smaller part of the global plan. so the given plan to follow that provided by the global planner and a map, then the local planner will provide velocity commands in order to move the chair.

LOCAL COST MAP

The local planner uses the local costmap in order to calculate the local plan

Unlike the global costmap, the local costmap is created directly from the robot's sensor readings. Given a width and a height for the costmap (whoch are defined by the user), it keeps the robot in the center of the costmap as it moves throughout the environment, dropping obstacle information from the map as the robot moves.

COST MAP PACKAGE

This package provides an implementation of a 2D costmap that takes in sensor data from the world, builds a 2D or 3D occupancy grid of the data (depending on whether a voxel based implementation is used), and inflates costs in a 2D costmap based on the occupancy grid and a user specified inflation radius. This package also provides support for map_server based initialization of a costmap, rolling window based costmaps, and parameter based subscription to and configuration of sensor topics.

MOST IMPORTANT PARAMETERS

1/ ~<name>/global_frame (string, default: "/map")

- The global frame for the costmap to operate in.

2/ ~<name>/robot_base_frame (string, default: "base_link")

- The name of the frame for the base link of the robot.

3/ ~<name>/transform_tolerance (double, default: 0.2)

- Specifies the delay in transform (tf) data that is tolerable in seconds. This parameter serves as a safeguard to losing a link in the tf tree while still allowing an amount of latency the user is comfortable with to exist in the system. For example, a transform being 0.2 seconds out-of-date may be tolerable, but a transform being 8 seconds out of date is not. If the tf transform between the coordinate frames specified by the global_frame and robot_base_frame parameters is transform_tolerance seconds older than ros::Time::now(), then the navigation stack will stop the robot.

4/ ~<name>/update_frequency (double, default: 5.0)

- The frequency in Hz for the map to be updated.

5/ ~<name>/publish_frequency (double, default: 0.0)

- The frequency in Hz for the map to be publish display information.

Global Costmap Package

MOST IMPORTANT PARAMETERS

1/
global_frame: map
robot_base_frame: base_link

- Set the global and robot frames for the costmap

2/ update_frequency: 5.0

- Set the update and publish frequency of the costmap

Local Costmap

MOST IMPORTANT PARAMETERS

1/
global_frame: odom
robot_base_frame: base_link

- Set the global and robot frames for the costmap

2/
update_frequency: 5.0
publish_frequency: 2.0

- Set the update and publish frequency of the costmap

.....

Chapter (4): Object Recognition & Detection

What Is Object Recognition?

Object Recognition is an important computer vision technique for identifying objects in images or videos and it's the key output of deep learning and machine learning algorithms.

When we look at a photograph or watch a video, we can easily spot people, different objects, scenes, and other visual details. So, the goal is to teach our machine in a way that make it do what becomes natural to humans, or in other words to make it understand what an image or scene contains just like humans do.

Also, it's the technique behind autonomous vehicles which make them recognize a stop sign or to distinguish a pedestrian from a lamp post. It is also useful in a variety of applications such as disease identification, security, robotic vision, etc.

So, object recognition has recently become one of the most exciting fields in computer vision and AI. The ability of immediately recognizing all the objects in a scene by a machine seems to be a modern technology and with the continuous development of convolutional neural network architectures, supported by a big training data and advanced computing technology, a computer can now surpass human performance in object recognition tasks under some specific conditions, such as face recognition.

What Is Image Classification?

In Image Classification, it takes an image as an input and then outputs the classification label of that image with some metric (probability, loss).
for example:

an image of a door will be classified as a class label “door”.

What Is Object Localization?

Object Localization is an algorithm that locates the presence of an object in the image and represents it with a bounding box.

It takes an image as input and outputs the location of the bounding box in the form of (position, height, width).

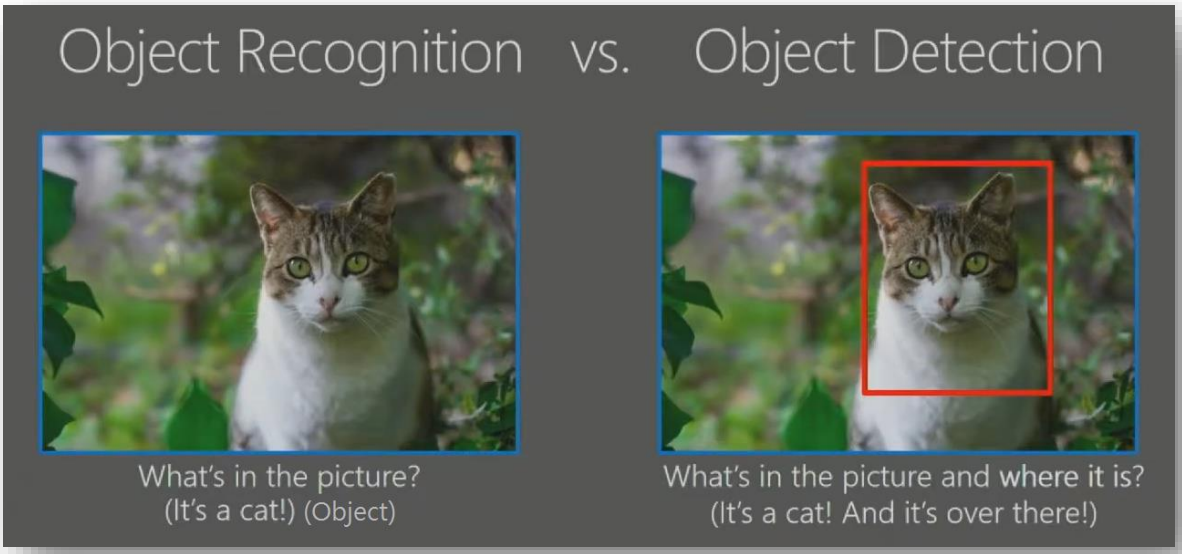
What Is Object Detection?

Object Detection algorithms act as a combination of image classification and object localization as It takes an image as input and produces one or more bounding boxes with the class label attached to each bounding box.

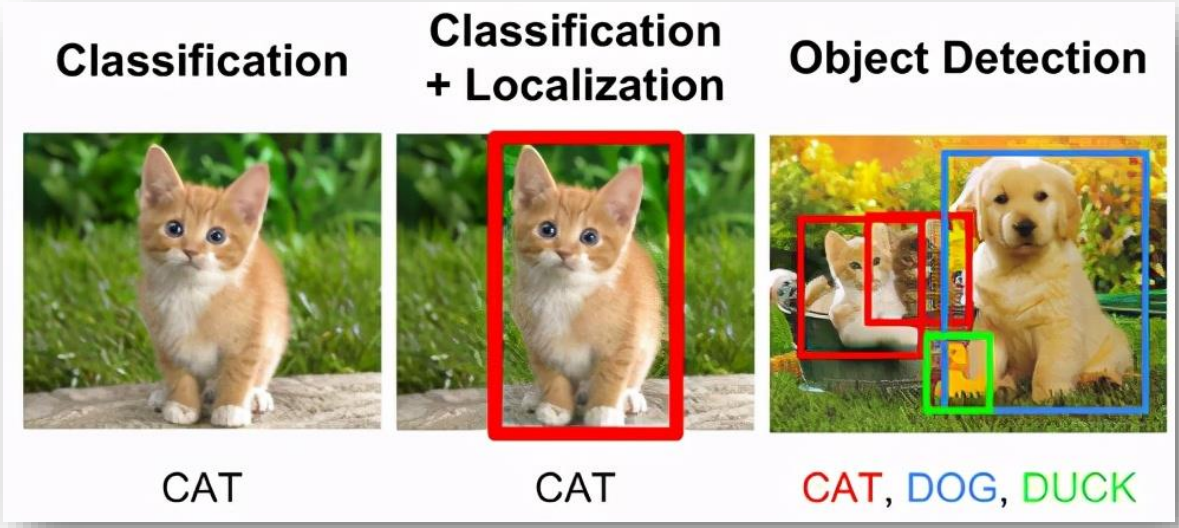
These algorithms are capable enough to deal with multiclass classification and localization as well as dealing with the objects with multiple occurrences.

Object Recognition vs. Object Detection

Object Detection and Object Recognition aren't very different techniques for identifying objects, they vary in their execution. Object detection is the process of finding instances of objects in images and also it's a subset of object recognition, where the object isn't only identified but also located in an image which allows for multiple objects to be identified and located within the same image.



Object recognition (left) and Object detection (right)



Challenges of Object Detection

In object detection, the bounding boxes are always having a rectangular shape. so, it doesn't help with determining the shape of objects if the object contains any curvature parts.

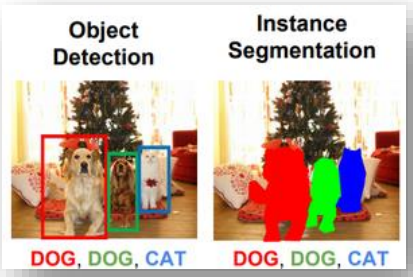
Also, in object detection, we can't accurately estimate some measurements such as the area or the perimeter of an object from image.

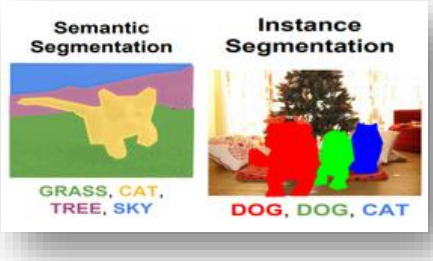
Image Segmentation

Image Segmentation is a further extension of object detection in which we mark the presence of an object through pixel masks generated for each object in the image. This technique is more granular than bounding box generation because this can help us in determining the shape of each object present in the image which help us in various fields such as medical image processing, satellite applications, etc.

There are two types of segmentation:

- **Instance Segmentation**: Which identifying the boundaries of the object and label their pixel with different colors.
- **Semantic Segmentation**: Which labeling each pixel in the image including the background with different colors based on their class label or category class.





How Object Recognition Works?

We can use a variety of approaches for object recognition. Recently, techniques in machine learning and deep learning have become popular approaches for object recognition problems. Both techniques learn to identify objects in images, but they differ in their execution.

Object Recognition Techniques

A) Object Recognition Using Deep Learning

Deep learning models such as convolutional neural networks (CNNs), are used to automatically learn an object's inherent features in order to identify that object. for example:

a CNN can learn to identify differences between cats and dogs by analyzing thousands of training images and learning the features that make cats and dogs different.

There are two approaches for object recognition using deep learning:

- **Training a model from scratch**: By training a deep network from scratch, we gather a very large labeled dataset and design a network architecture that will learn the features and build the model. the results can be impressive, but this approach requires a large amount of training data, and we need to set up the layers and weights in the CNN.

- **Using a pretrained deep learning model**: Most deep learning applications use the transfer learning approach, a process that involves fine-tuning pretrained model.

  we start with an existing network, such as AlexNet, and feed in new data containing previously unknown classes. This method is less time-consuming and can provide a faster outcome because the model has already been trained on thousands or more of images.

What Is Transfer Learning?

Transfer Learning is the reuse of a pre-trained model on a new problem. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another. For example, in training a classifier to predict whether an image contains food, we could use the knowledge it gained during training to recognize drinks.

So, deep learning offers a very good level of accuracy but requires a large amount of data to make accurate predictions.

Why Using Transfer Learning?

Transfer learning has several benefits, but the main advantages are saving training time, better performance of neural networks and not needing a lot of new data.

When to Use Transfer Learning?

It’s hard to form a rule that is generally applicable, but there are some guidelines on when transfer learning could be used:

- There isn't enough labeled training data to train your network from scratch.
- When two tasks have the same input.
- When there are already exists a network that is pre-trained on a similar task, which is usually trained on massive amounts of data and have a very good accuracy.

B) Object Recognition Using Machine Learning

Machine learning techniques are offering different approaches than deep learning as said before.

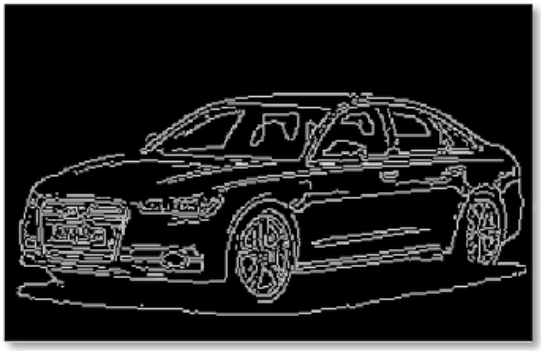
Common examples of machine learning techniques are:

- Histogram of Oriented Gradients (HOG) feature extraction with a Support Vector Machine (SVM) machine learning model.

What Is HOG Feature Descriptor?

Here, we will start with an example, let’s take a look at the two images shown below. can we differentiate between the objects in the image?

We can clearly see that the right image has a dog and the left image has a car. Now, let's make this task slightly more complicated – can we identify the objects shown in the image below?



Still easy, as the first pair of images had a lot of information, such as the shape of the object, its color, the edges and the background.

On the other side, the second pair had much less information, only the shape and the edges but it's still enough to differentiate the two images.

So, we could say that HOG feature descriptor is a simplified representation of the image that contains only the most important information about the image.

HOG Descriptor focuses on the structure or the shape of an object. in it, the complete image is broken down into smaller regions and for each region, the gradients and orientation are calculated.

Then, it will generate a histogram (Figure X) for each of these regions separately.

The histograms are created using the gradients and orientations of the pixel values.

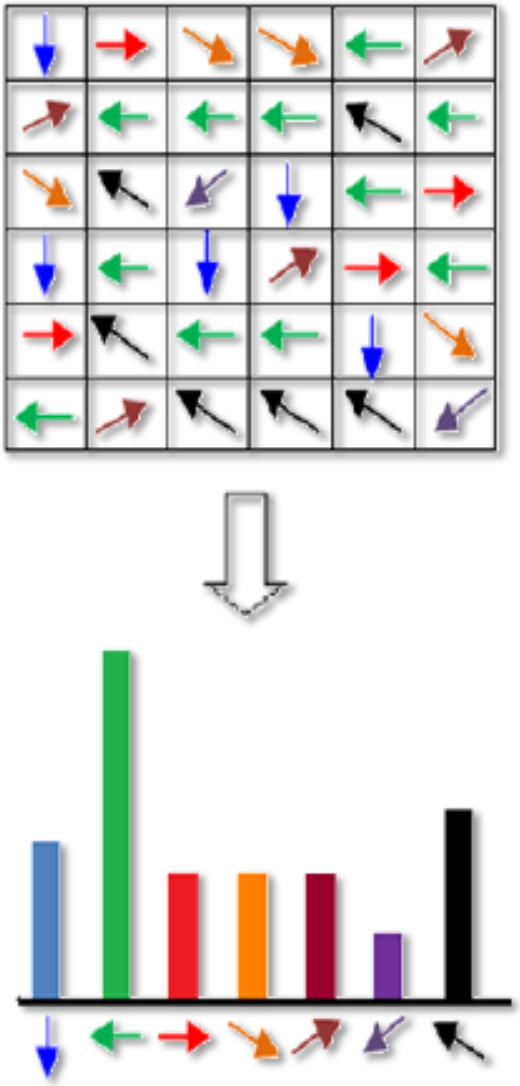


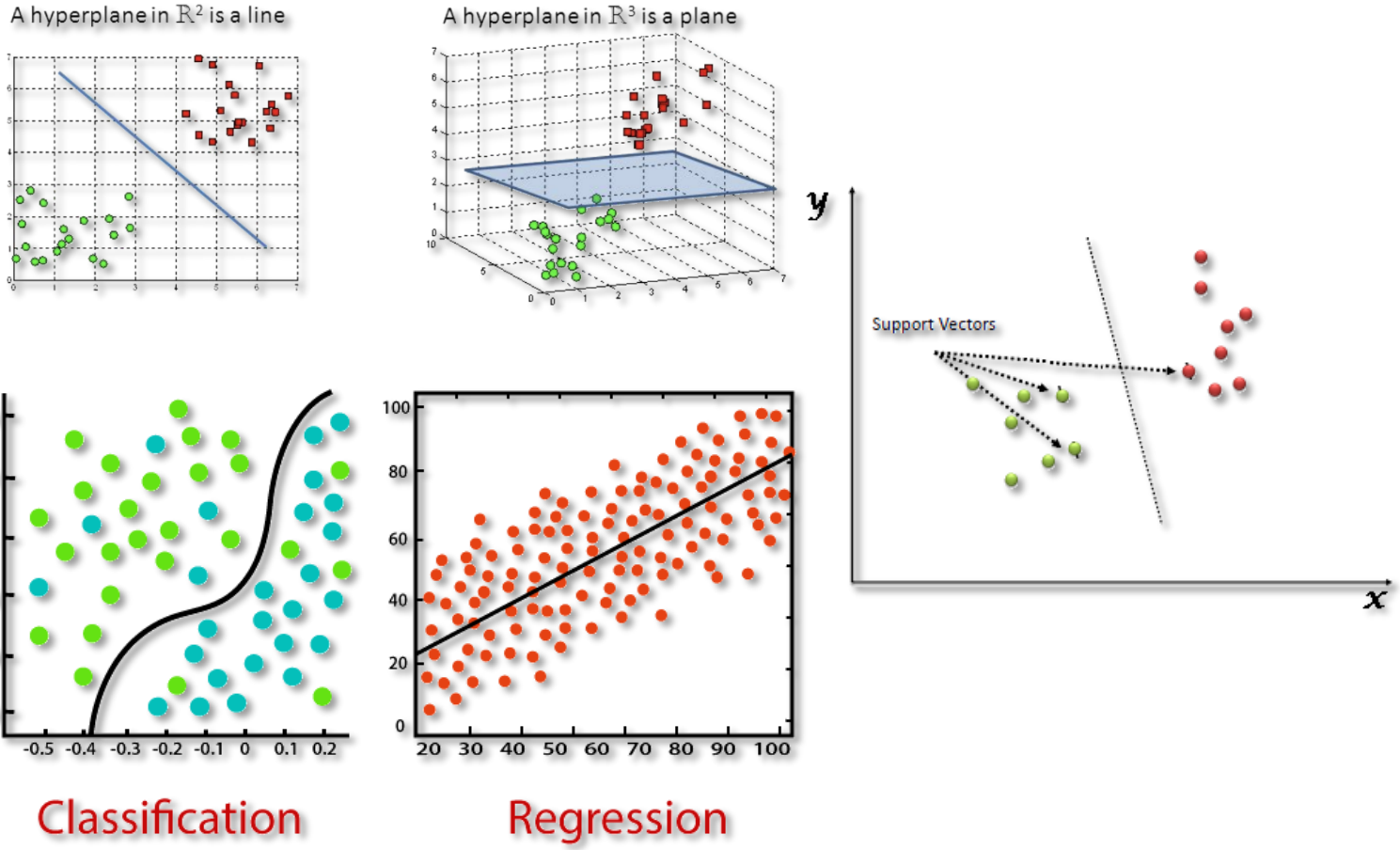
Figure X

What Is Support Vector Machine?

It's a supervised machine learning algorithm which can be used for both classification and regression problems. However, it's mostly used in classification problems.

In the SVM algorithm, we plot each data item as a point in n-dimensional space, where n is number of features you have with the value of each feature being the value of a particular coordinate.

Then, we perform classification by finding the hyperplane or line that differentiates the two classes very well.



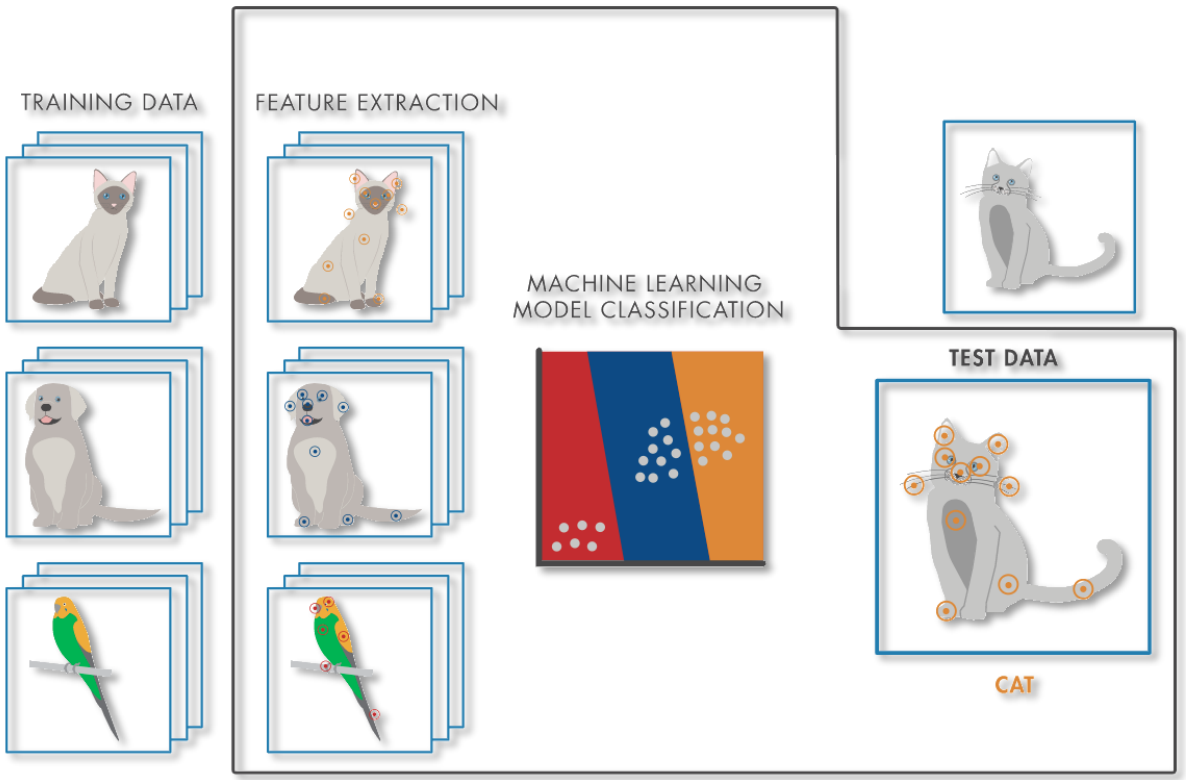
How to Perform Object Recognition Using A Standard Machine Learning Approach?

We could do that by starting with a collection of images and select the relevant features in each image. for example:

a feature extraction algorithm might extract corner or edge features that can be used to differentiate between classes in our data.

These features are added to a machine learning model, which will separate these features into distinct categories and then use this information when analyzing and classifying new objects.

So, using machine learning for object recognition offers a flexibility to choose the best combination of features and classifiers for learning. It can achieve accurate results with small amount of data.

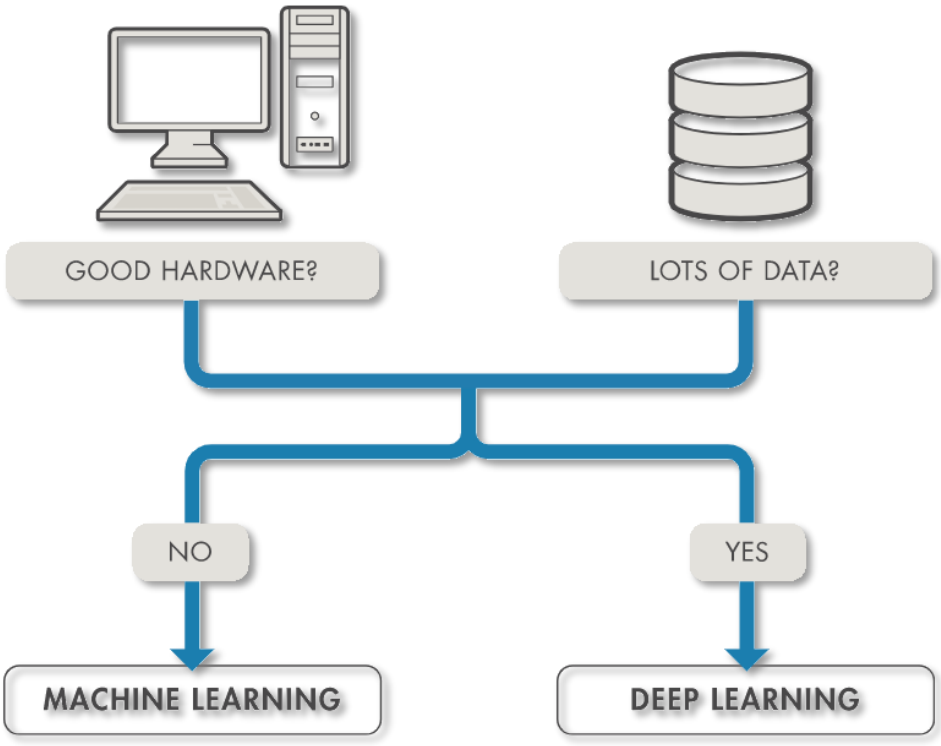


Machine Learning vs. Deep Learning for Object Recognition

Determining the suitable approach for object recognition depends on our application and the problem we're trying to solve. In many cases, machine learning can be an effective technique, especially if we know which characteristics or features of the image are best to be used to differentiate classes of objects.

The main consideration to keep in mind when choosing between machine learning and deep learning is whether we have a powerful GPU and lots of labeled training images.

If the answer to either of these questions is no, a machine learning approach might be the best choice. deep learning techniques tend to work better with more images, and a GPU that helps to decrease the time needed to train the model.

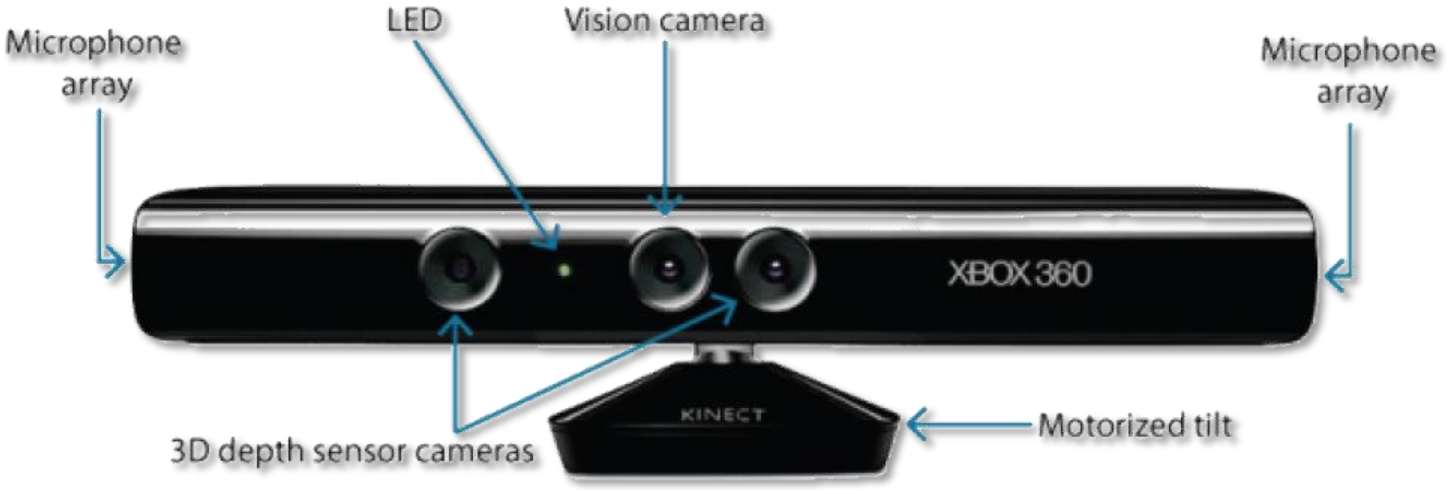


In the following pages, we will talk about the components used, requirements and tasks which have been done in this part for our project.

Different Requirements and Components Which Have Been Used In This Part At The Project

HARDWARE	SOFTWARE
MICROSOFT XBOX 360 KINECT	FREENECT / OPENNI
GPU: NVIDIA GTX 1050 TI 4GB GPU: NVIDIA GTX 950 2GB	Dual-boot OS: WINDOWS UBUNTU 16.04 / 18.04
RASPBERRY PI 4B 8GB	ROS KINETIC / MELODIC
PC WITH +10GB RAM	PYTHON & OPENCV
720 / 1080P WEBCAM	CUDA 10.2
HDMI DISPLAY	RVIZ

Microsoft XBOX 360 Kinect



Kinect v1 Model #: 1414

Kinect sensor is a device which is mainly used in 3D vision applications, we could say that it's the main core and one of the major components of this project.

It is installed on a small fixed platform and it consists of the following three sensors that work together and which we can use for vision and robotics tasks:

- An RGB camera to see pictures 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 users from the noise in the room.

In ROS, we are going to use two of these sensors: the depth sensor and the RGB camera.

As Kinect provides a depth map, this means that the Kinect measures the distance for every pixel seen by the sensor from it. This makes a variety of computer vision tasks like background removal, blob detection, and others easier.

After we get the 3D image of the surrounding, the 3D images are converted to finer points that are gathered to form a “point cloud”.

This point cloud data will have all the 3D parameters that constitute the surrounding environment.

The main use of Kinect sensor in our project is to mock the functionality of a laser scanner. The laser scanner data is essential for this part of the project and to build a map of the environment.

The laser scanner (LIDAR) is a very costly device, so instead of buying an expensive laser scanner, we can convert a Kinect into a virtual laser scanner.

Now, Kinect has officially stopped from production, but it still could be found at some places.

One of the problems which I have faced, that the Kinect is no more compatible with Windows. So, I couldn’t use it on Windows OS and I had to repartition my hard drive and install Ubuntu system then ROS as a dual-boot along with Windows on my pc to deal with Kinect and to perform the project tasks in a proper way.

What Is Point Cloud?

We could think about point clouds as a collection of multiple points. when these multiple points are brought together, they start to show some interesting features.

So, point cloud is a collection of points that represent a 3D shape or feature. Each point has its own set of X, Y, Z coordinates and sometimes could be more attributes such as the color value, which is stored in RGB format and luminance value, which determines how bright the point is.

"It could be seen as the simplest form of 3D models"

How Point Clouds Are Created?

Point clouds are created by performing a scan of an object or structure. scans are completed by using either a laser scanner or through a process called photogrammetry.

- **Laser Scanners**: This method works by sending out pulses of light to the surface of an object and measuring how long it takes each pulse to reflect back and hit the scanner. These measurements are used to determine the exact position of points on the object, and these points then create a point cloud.
- **Photogrammetry**: It's a process of creating measurements from pictures. It uses photos of an object taken at different locations to triangulate points on the object and plot these points in 3D space.

Also, in photogrammetry we can use most cameras for this process, and again the final result is a point cloud.

Installing and Working With Kinect

Note: as said before, we need to work on Ubuntu OS in addition to a compatible ROS version installed on it.

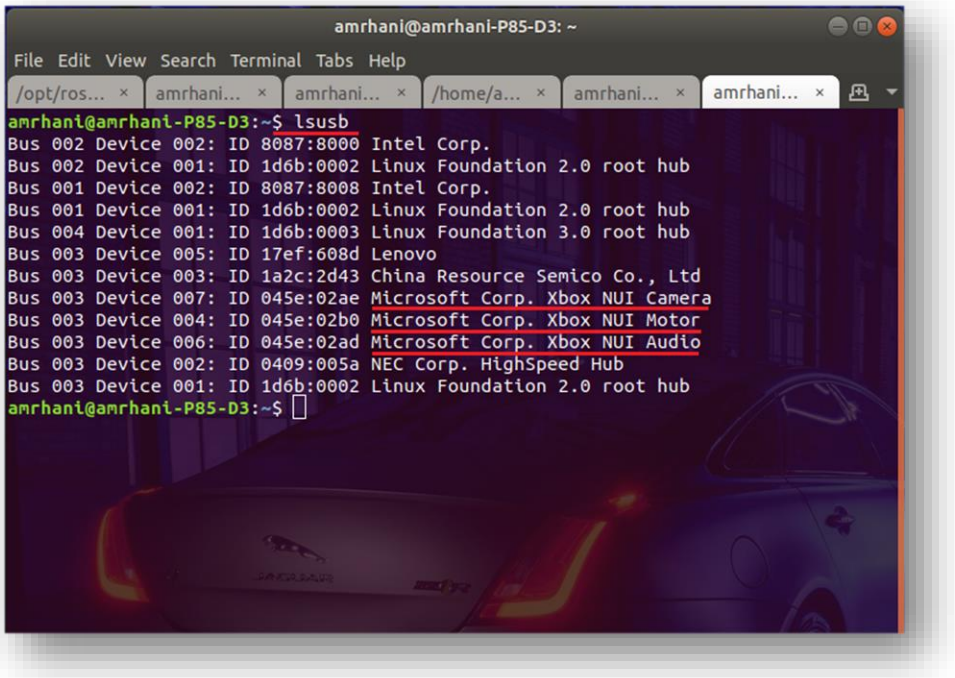
The first step after installing ROS environment is to connect Kinect with Raspberry Pi or with pc then install important drivers and libraries such Freenect or OpenNI libraries which we will need:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install libfreenect-dev
$ sudo apt-get install ros-version-freenect-launch
$ sudo apt-get install ros-version-depthimage-to-laserscan
```

For some ROS versions such ROS melodic, we need to do some different procedures to make Kinect works well:

```
$ git clone https://github.com/OpenKinect/libfreenect
$ cd libfreenect
$ mkdir build
$ cd build
$ cmake -L ..
$ make
$ sudo make install
$ cd
$ cd catkin_ws/src/
$ git clone https://github.com/ros-drivers/freenect_stack.git
$ cd catkin_ws/
$ catkin_make
$ sudo apt-get install ros-version-depthimage-to-laserscan
```

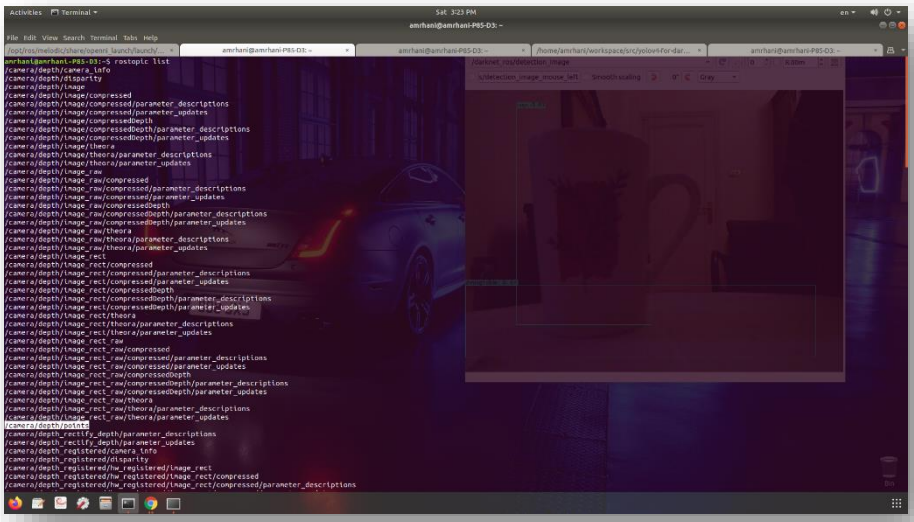
If everything goes well, we should try $ lsusb command and see the following result:



Now, we will launch Kinect sensor with the following command:

$ roslaunch freenect_launch freenect.launch

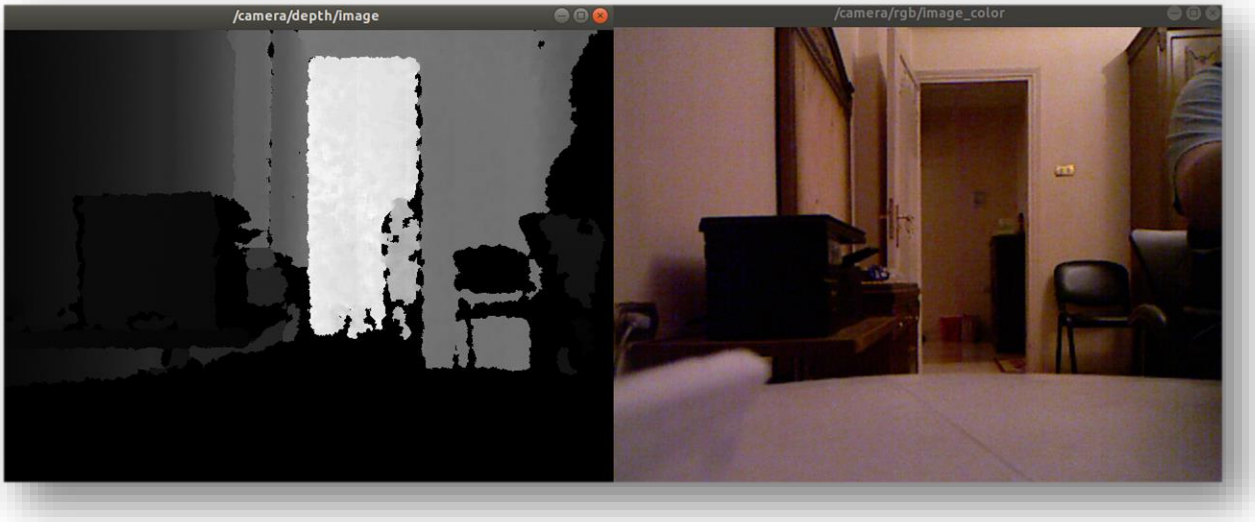
Then, in another terminal if we try $ rostopic list command to list the active topics, we should see the following result, most of it will be related to camera/rgb and camera/depth:



Now, in another terminal window, we should see something similar to the next result if we try the following commands:

$ rosrun image_view image_view image:=/camera/rgb/image_color

$ rosrun image_view image_view image:=/camera/depth/image



RVIZ

Rviz (ROS visualization) is a 3D visualizer for displaying sensor data and other received information from ROS.

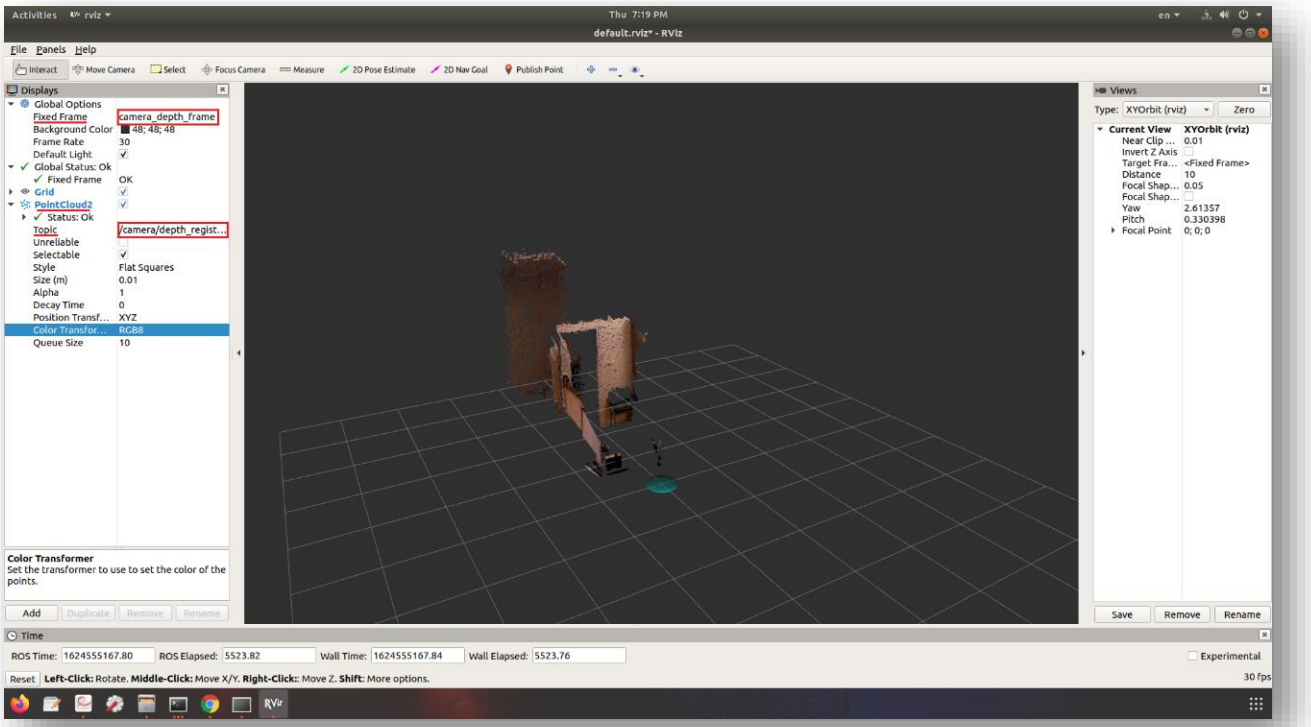
Using this program, we can display live representations of sensor values coming over ROS topics including camera data, infrared distance measurements, etc.

"Rviz lets us see what the robot is seeing, thinking and doing"

The next step is that we will open this GUI program which installed with ROS automatically to see what data received from our Kinect sensor by the following command:

$ rosrun rviz rviz

We will see a GUI window and we should add PointCloud2 topic then change important settings in the left side panel to make the program shows what the Kinect sensors see.



In the global option:

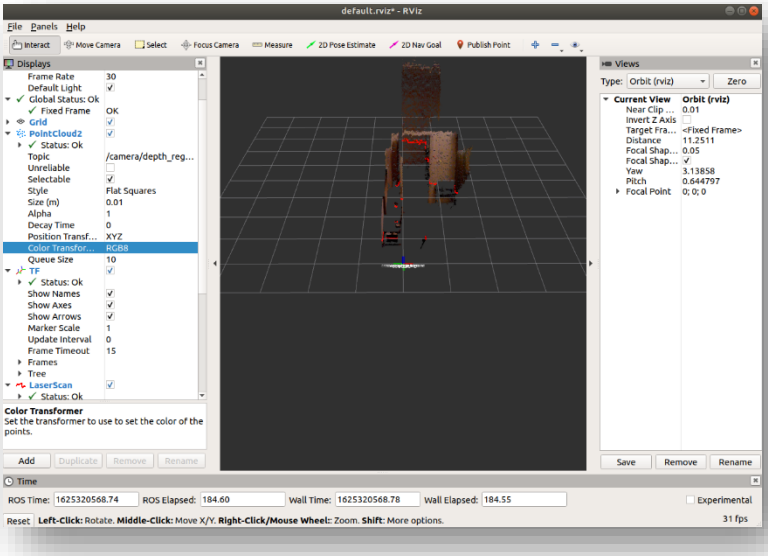
We should set fixed frame at camera_depth_frame, and in pointcloud2 options:

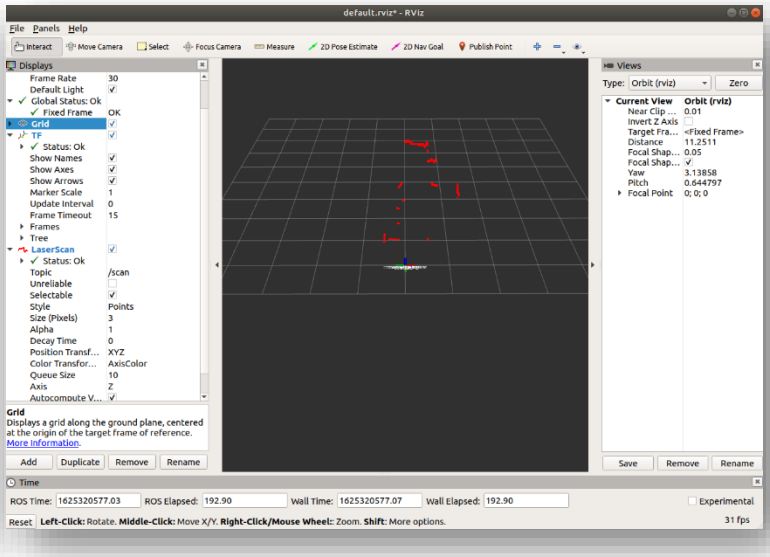
We should set topic as camera/depth_registered/points if we want to show point clouds with RGB colors or camera/depth/points for other forms than RGB.

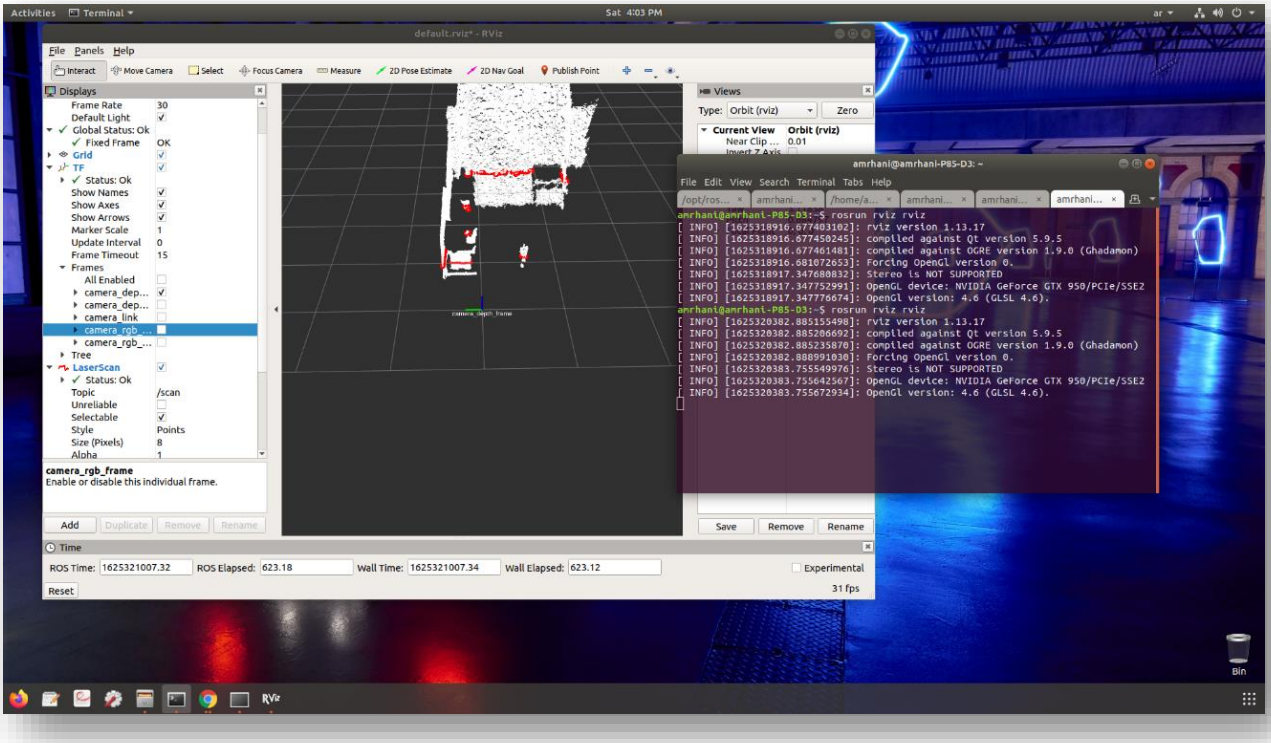
Another problem I have faced in this task for object recognition is that the SSH we used in our project for Raspberry Pi connection has a limited bandwidth, unlike the point cloud data which requires large bandwidth.

So, I had two solutions or approaches for this problem:

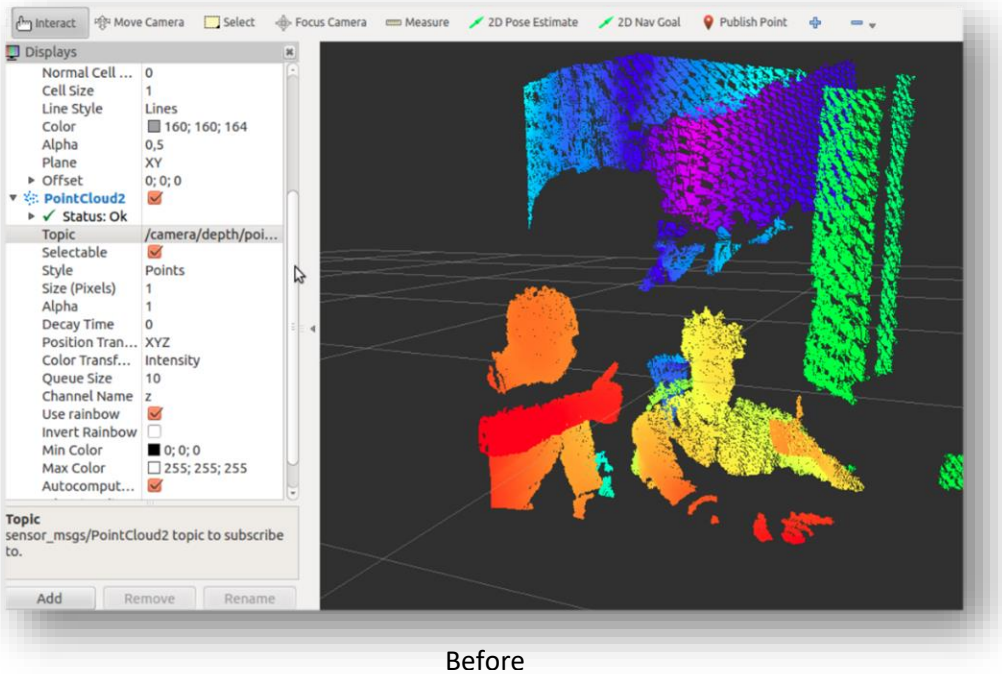
A) The first solution is that I transformed point cloud data to laser data which had an advantage that it doesn't require large bandwidth like point cloud, and this was the recommended solution.



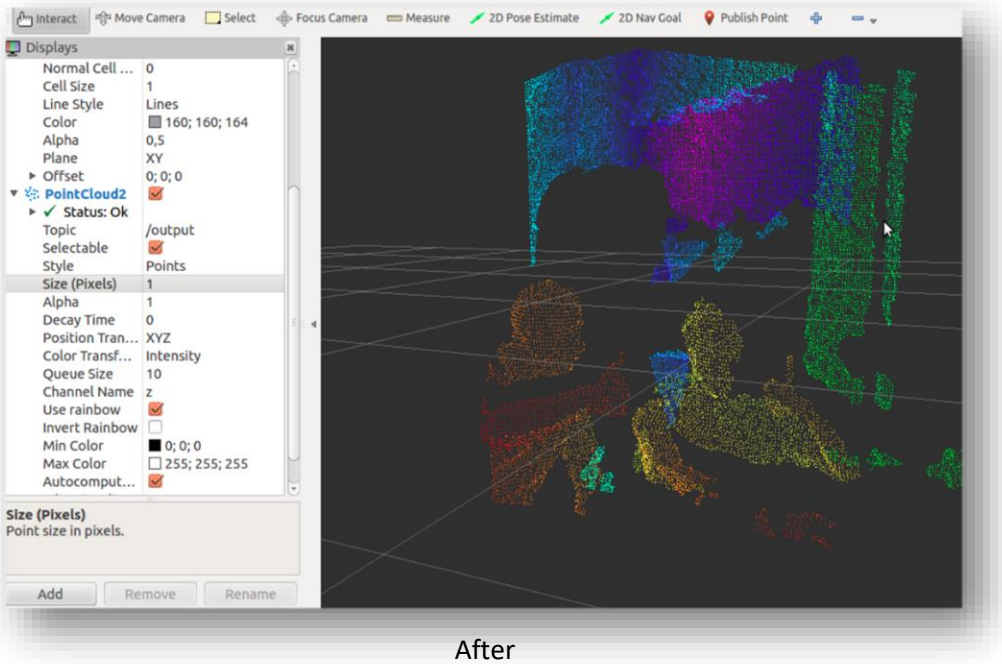




B) Another approach instead of the previous data to laser scan transformation, is that we can implement a program to generate a node that filters the point cloud from the Kinect sensor. This node will apply a filter to reduce the number of points in the original data. It will make a down sampling and huge reduction of data.



Before



After

Now, we have seen how to show point clouds as it's one of the most important parts which many other tasks in this project depend on it.

For further and more detailed explanation about the previous and other coming tasks in this part, it will be found in the video file related to this part.

Object Detection Using Darknet Yolo Algorithm

After previous object recognition task using ROS and Kinect. Now, we will go to another advanced task which is object detection using Yolo algorithm on Ubuntu and also Windows.

What Is Yolo Algorithm?

Yolo is an algorithm which uses neural networks to provide real-time object detection. This algorithm is popular because of its high speed and accuracy.

Yolo is an abbreviation for the term "You Only Look Once" because the network only looks the image once to detect multiple objects.

It's done as a regression problem and provides a class probability of the detected images.

This algorithm employs convolutional neural networks (CNN) to detect objects in real-time. as its name shows, the algorithm requires only a single forward propagation through a neural network to detect objects.

Which means that the prediction for the entire image is done in a single algorithm run.

The Yolo algorithm consists of various and different variants such as tiny Yolo and Yolov4.

What Are Advantages of Yolo Algorithm

- **High accuracy**: It's a predictive technique that provides accurate results with minimal background errors.
- **Speed**: This algorithm provides a real-time object detection.
- **High Learning capabilities**: The algorithm has excellent learning capabilities which make it one of the most speed and high accuracy algorithms.

How Yolo Algorithm Works?

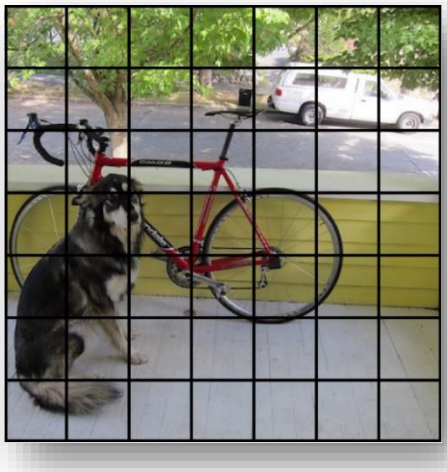
It works using the following three techniques:

- Residual blocks.
- Bounding box regression.
- Intersection Over Union (IOU).

Residual Blocks

In this step, the input image is divided into various grid cells of equal dimension.

Each grid has a dimension of S x S and every cell will detect objects that appear within them.



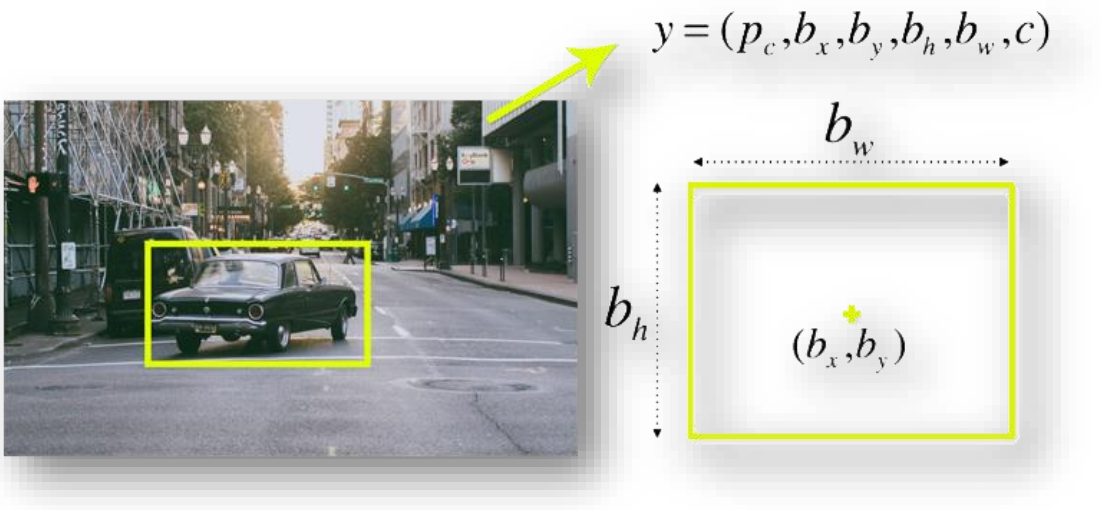
Bounding Box Regression

Bounding box is an outline that highlights an object in an image.

Every bounding box in the image consists of some attributes such as:

Width, Height, Center of The Bounding Box, Class.

Yolo uses a single bounding box regression to predict the width, height, center and class of objects.



Intersection Over Union (IOU)

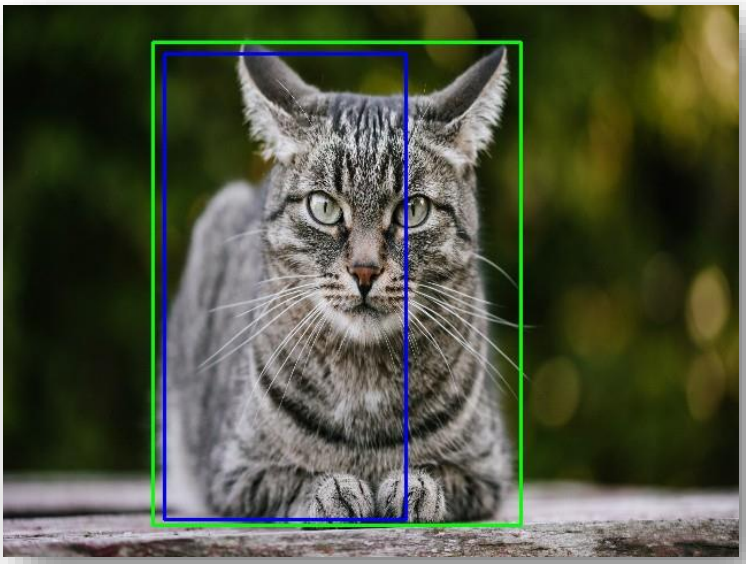
IOU is a phenomenon in object detection that describes how boxes overlap.

Yolo algorithm uses IOU to provide an output box that surrounds the objects perfectly most time.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores.

The IOU is equal to 1 if the predicted bounding box is the same as the real box.

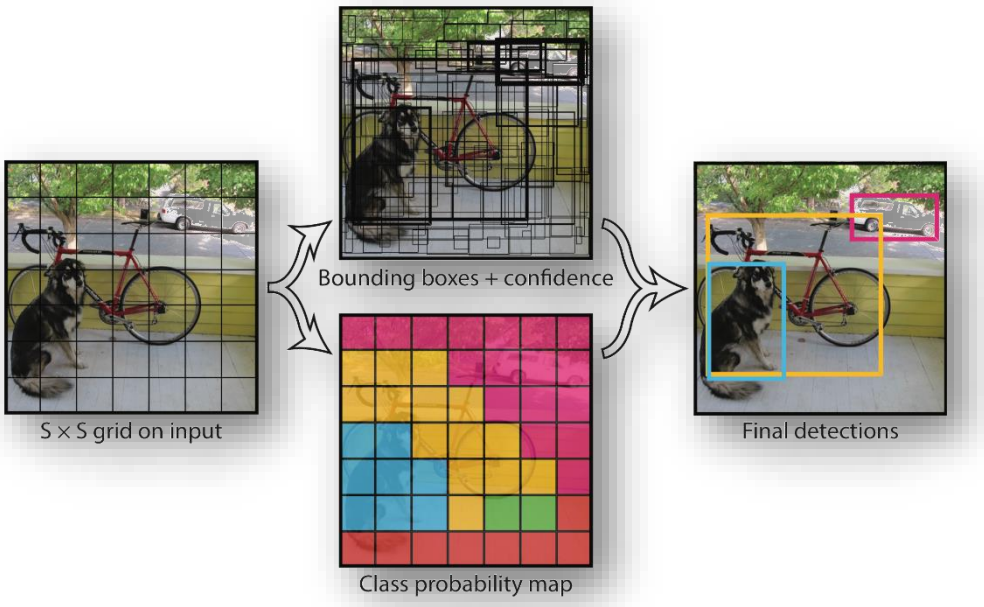
This mechanism eliminates bounding boxes that are not equal to the real box.



In the previous image, there are two bounding boxes, one in blue and the other in green.

The blue box is the predicted box while the green box is the real box, Yolo ensures that the two bounding boxes are equal.

Combination of The Three Techniques

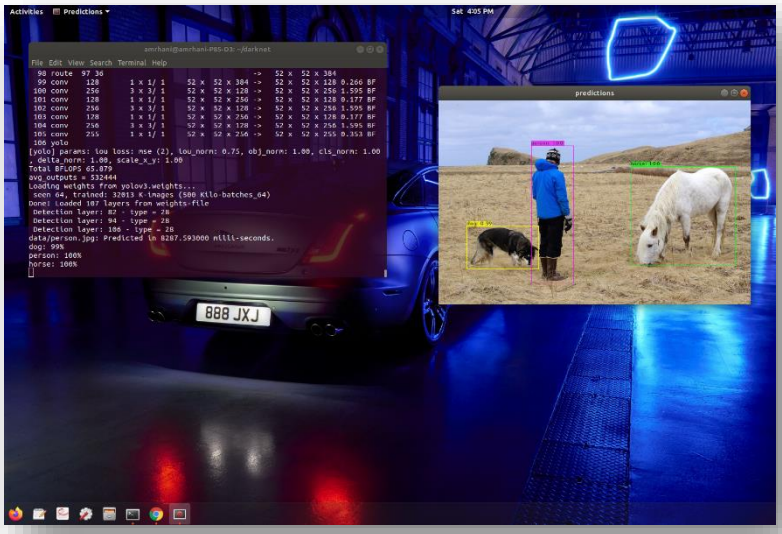


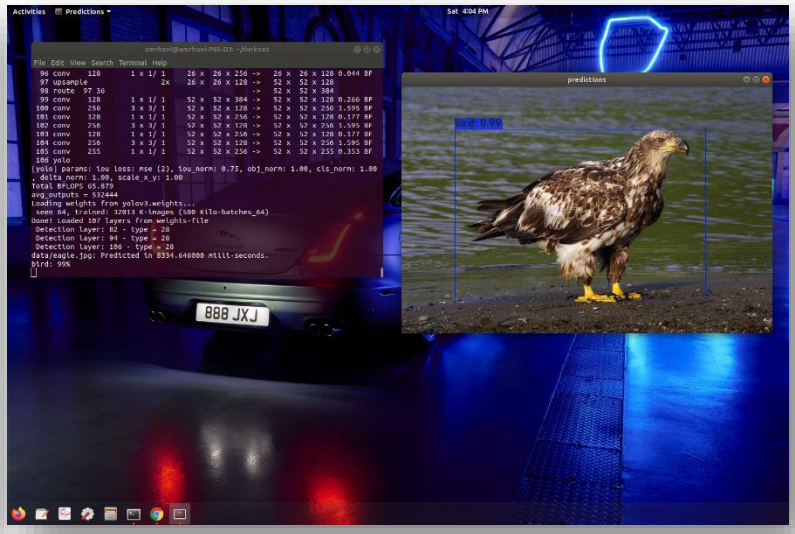
Using Yolo Algorithm With Kinect on Ubuntu And ROS

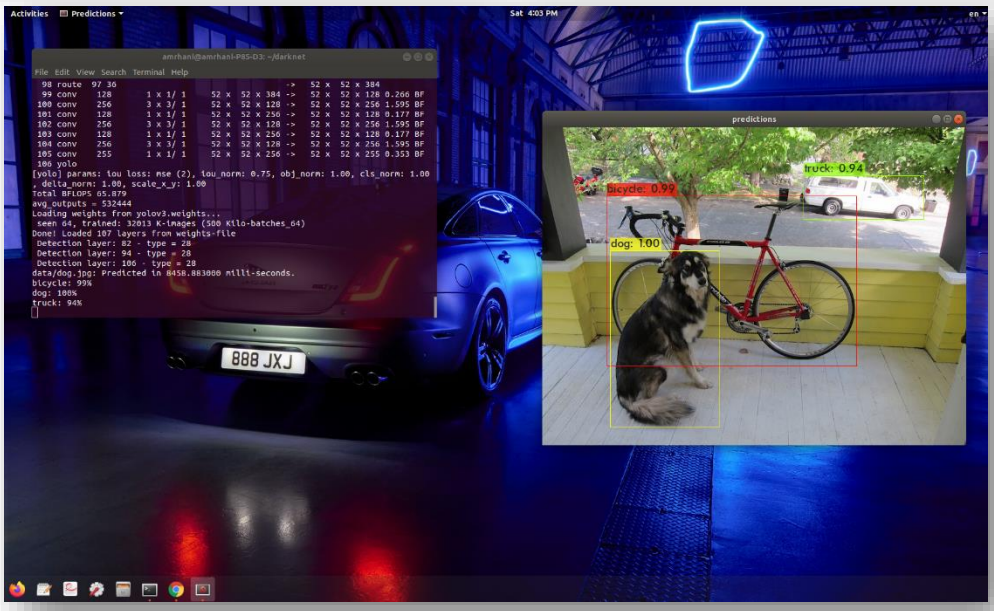
First step is to build a Yolo environment for ROS by installing Darknet for ROS which is an open source neural network framework that runs on CPU and GPU.

After building and installing it, we should be able to try it without a problem.

These are some examples on detecting objects for different images:







Testing Kinect Real-Time Object Detection by Yolo Algorithm For ROS

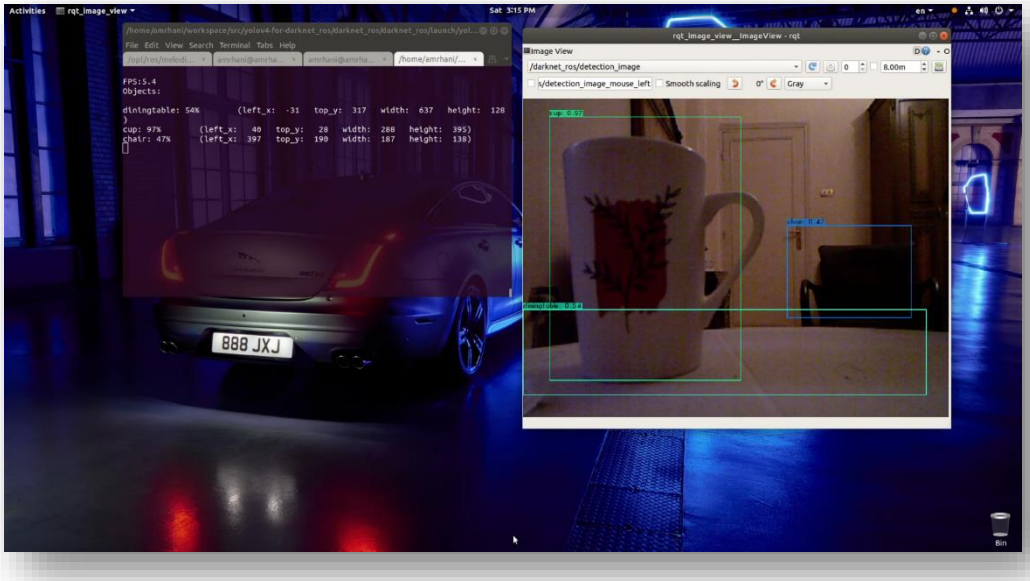
After we prepared our config, launch and weight files for the algorithm first and for example we made its name as yolo_v4 and launch the Kinect sensors and then run the following command:

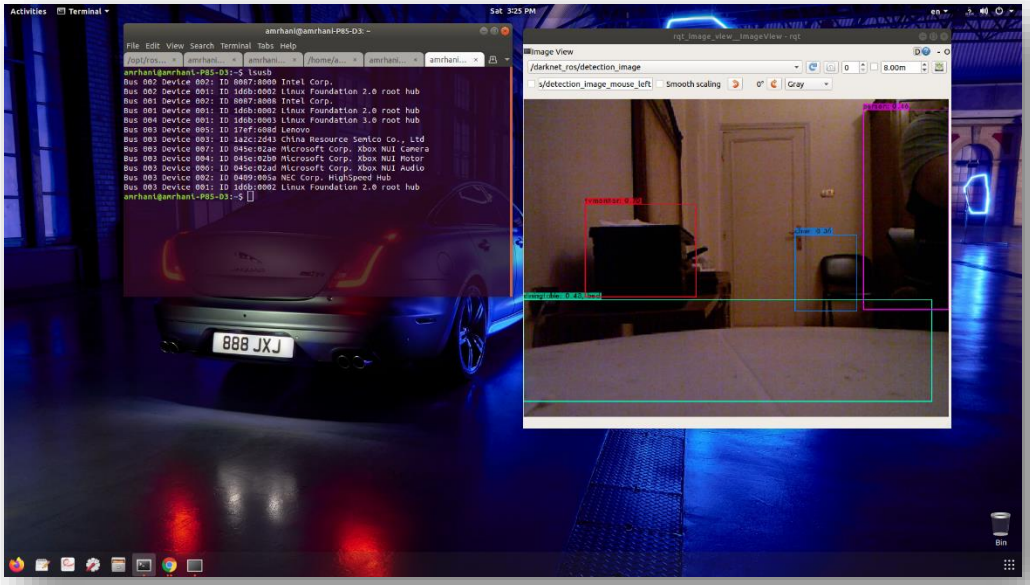
$ roslaunch darknet_ros yolo_v4.launch

And in another terminal window, run the following command:

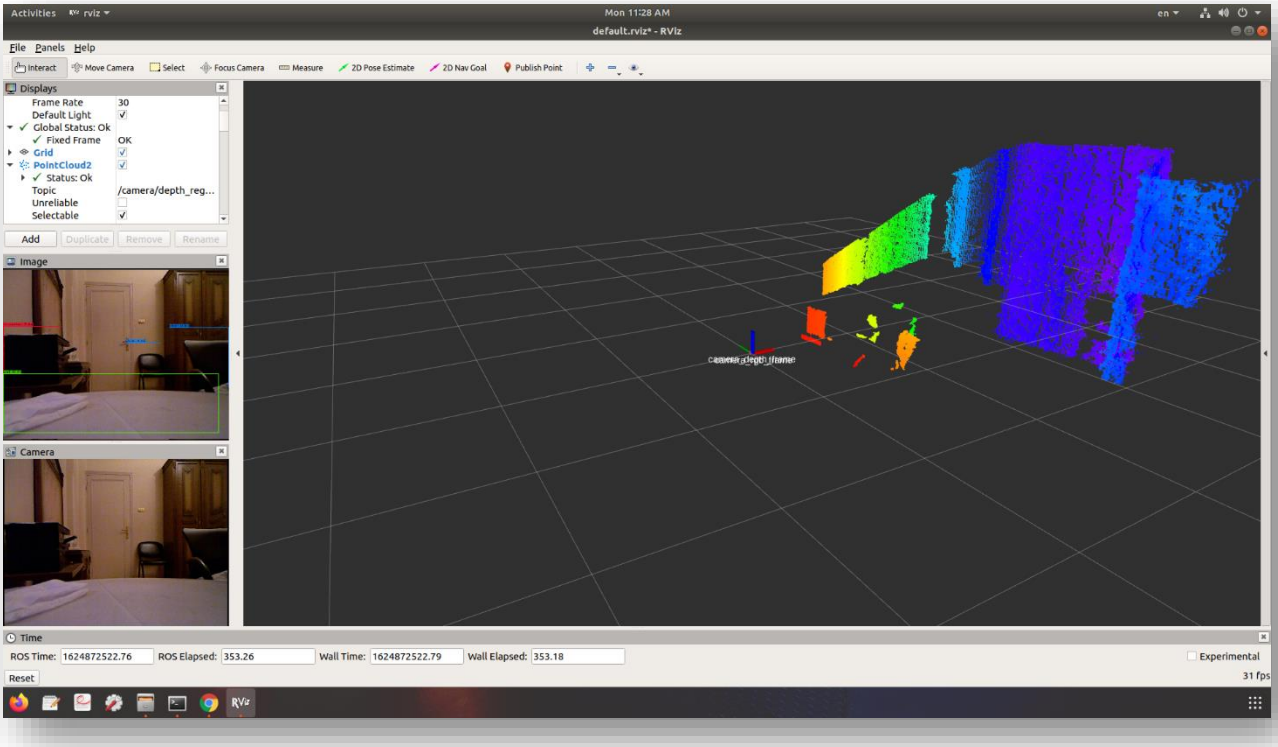
$ rosrun image_view image_view image:=/darknet_ros/detection_image

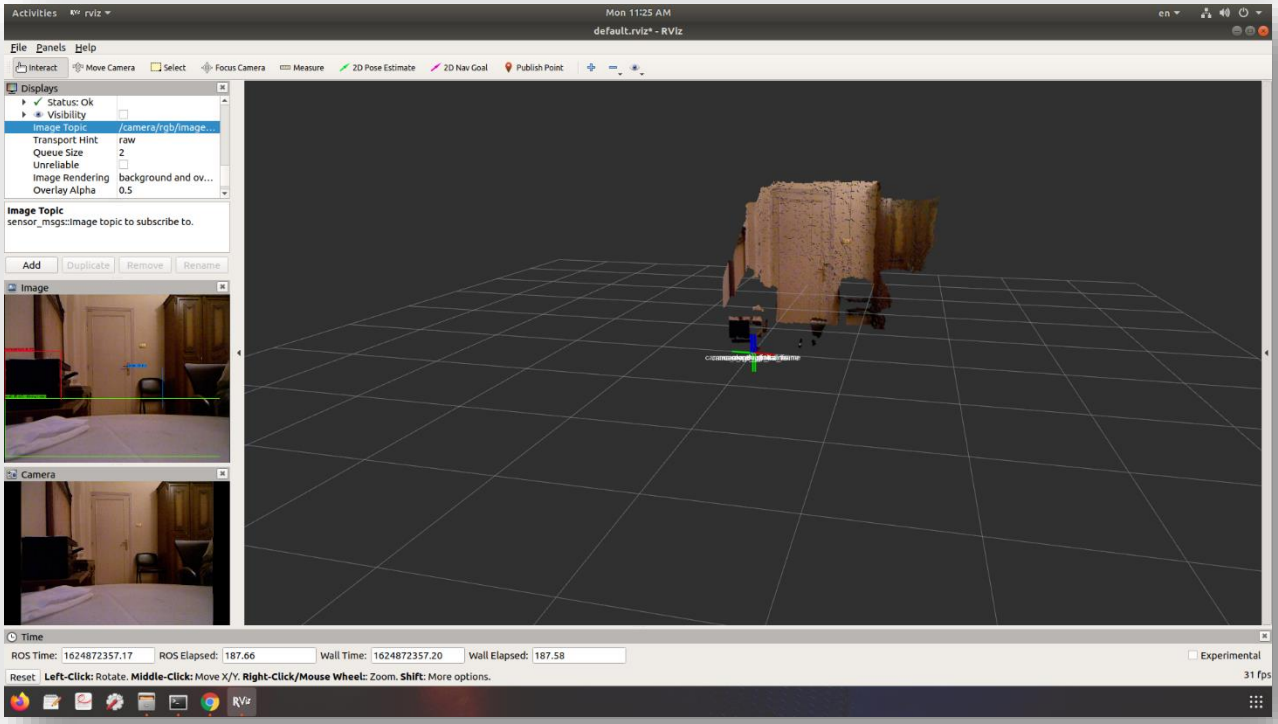
We should be able to see such the following results:





And then if we launch Rviz, we could add real-time detection with point cloud data as the following pictures:





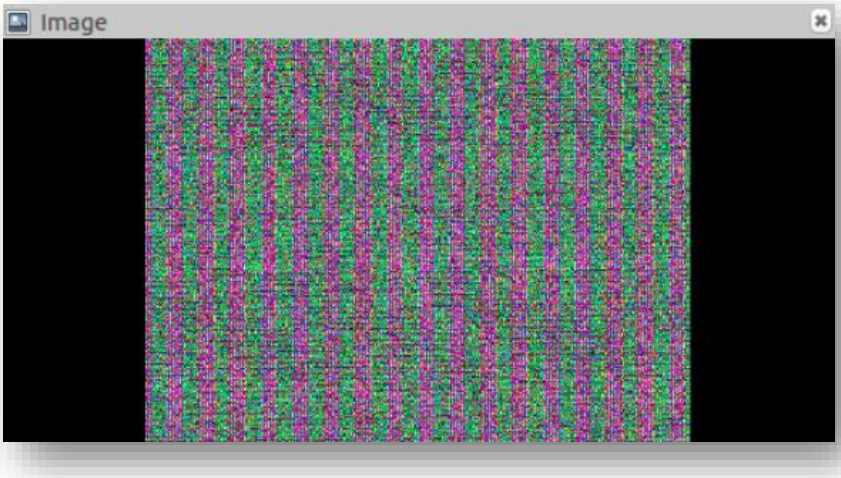
I had some small problems in this part, the first problem is that for local object detection, it needs more powerful hardware and especially more efficient VGA card such as Nvidia GTX 1070 or more if we want to make a good balance between frames and accuracy.

I managed to solve this problem by modifying some configuration in the detection algorithm and I reached to an acceptable result at the end.

The second problem was that Ubuntu not allowing me to run Yolo algorithm on GPU which could make a very big performance boost despite of proper VGA drive is installed and also the compatible CUDA, OpenCV versions were installed and even if I changed some configuration to make it run on GPU, no performance difference found.

But the performance was good and acceptable at the end as said before.

The last problem was a small error appears when try to launch the object detection as the following image:



The solution for the previous error wasn't very hard, only we need to change the enable_opencv: true parameter in the ros.yaml file to false.

Testing Yolo Object Detection For Windows OS

Yolo algorithm used for windows was better than the one used on ROS as it could be working efficiently on the GPU without problems and it has a better optimization.

There are two different approaches or methods to detect objects using Yolo algorithm on Windows OS:

- **Use pretrained weights**: Which have been trained on a dataset that has about 80 different class.
- **Train a custom model**: Which can be used to detect other classes that wasn't found in the 80 class at the pretrained model.

A) Using Pretrained Weights

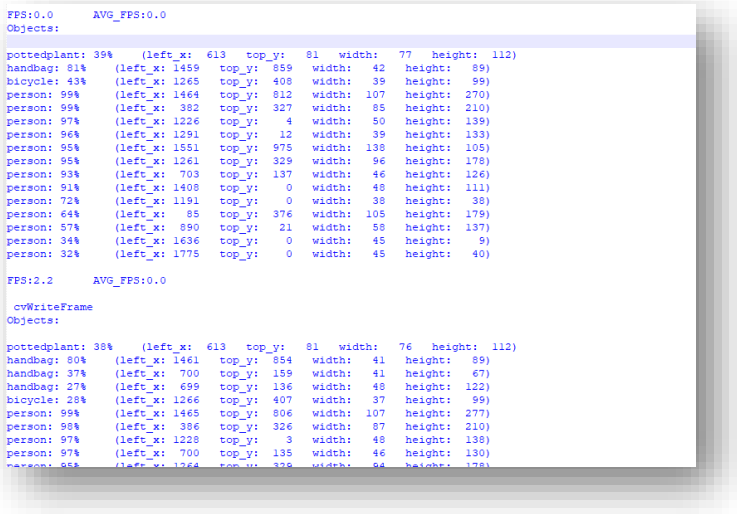
It's the best way from these two ways, as it gives best accuracy, performance, saves the time and resources.

This model had been trained on a very large dataset and big amount of images.

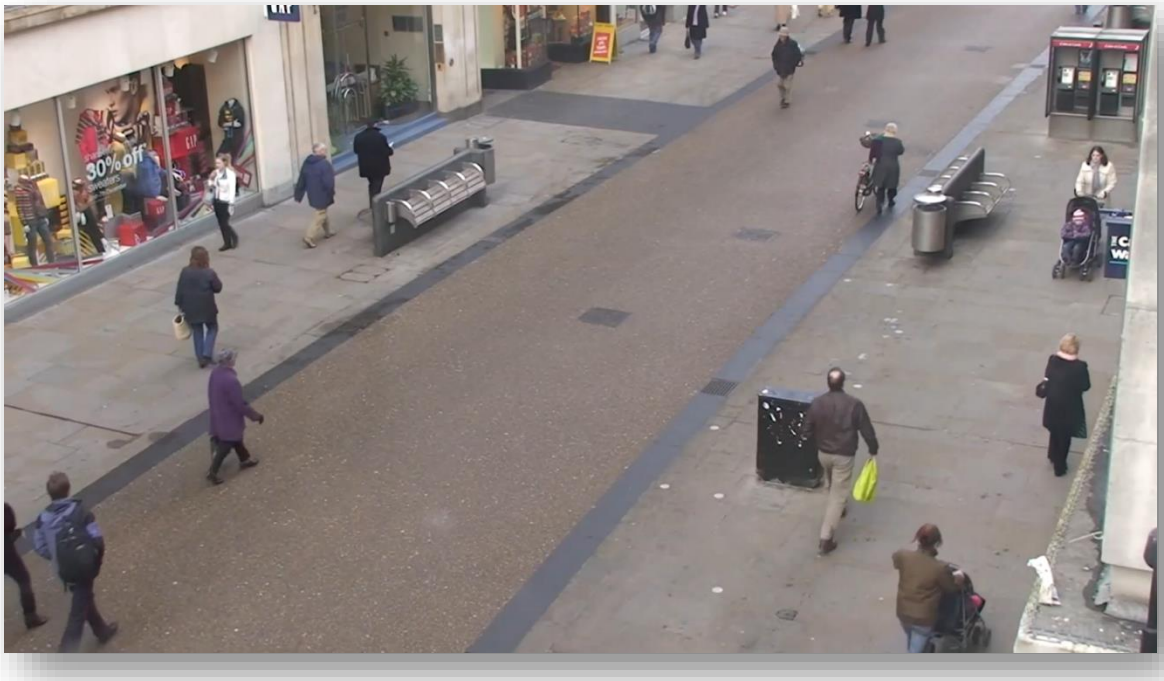
It can be run and make detections on images, recorded and real-time videos.

Also in this model:

we could generate and save bounding box coordinates and detection accuracy percentages on both json or txt files for images or for every frame of videos such the following example.



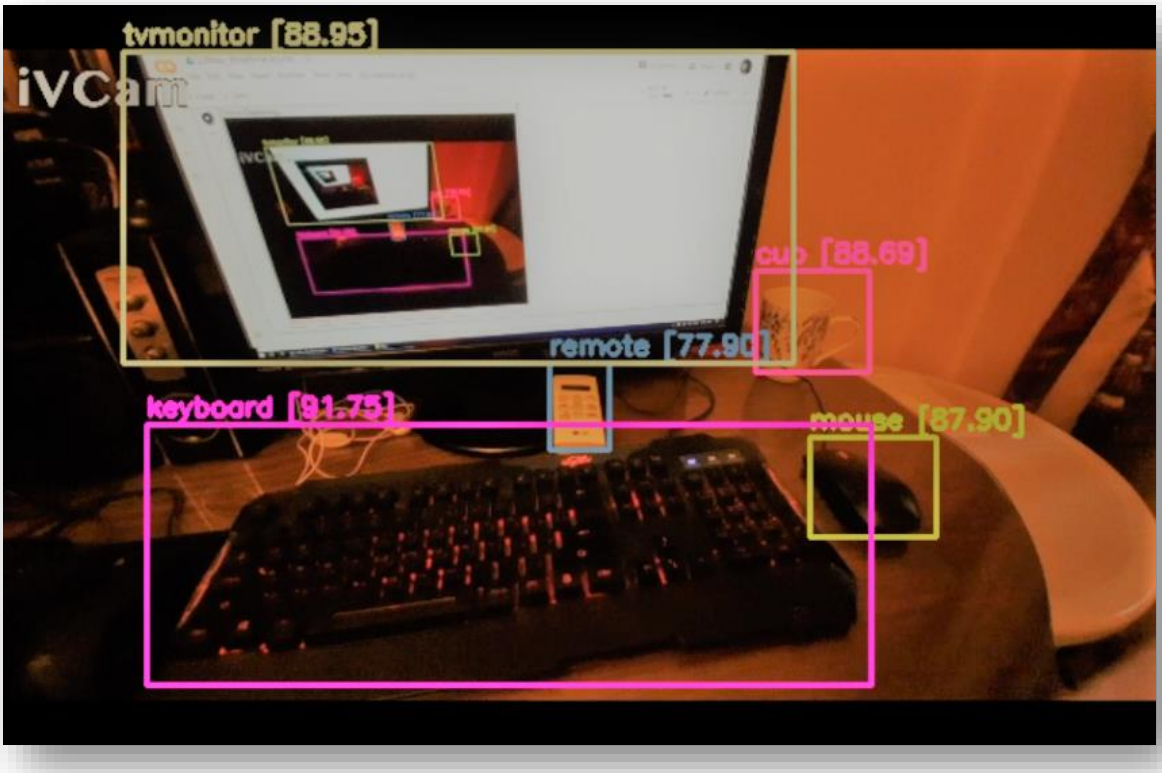
And these are examples before and after of a video that has been processed by Yolo algorithm:

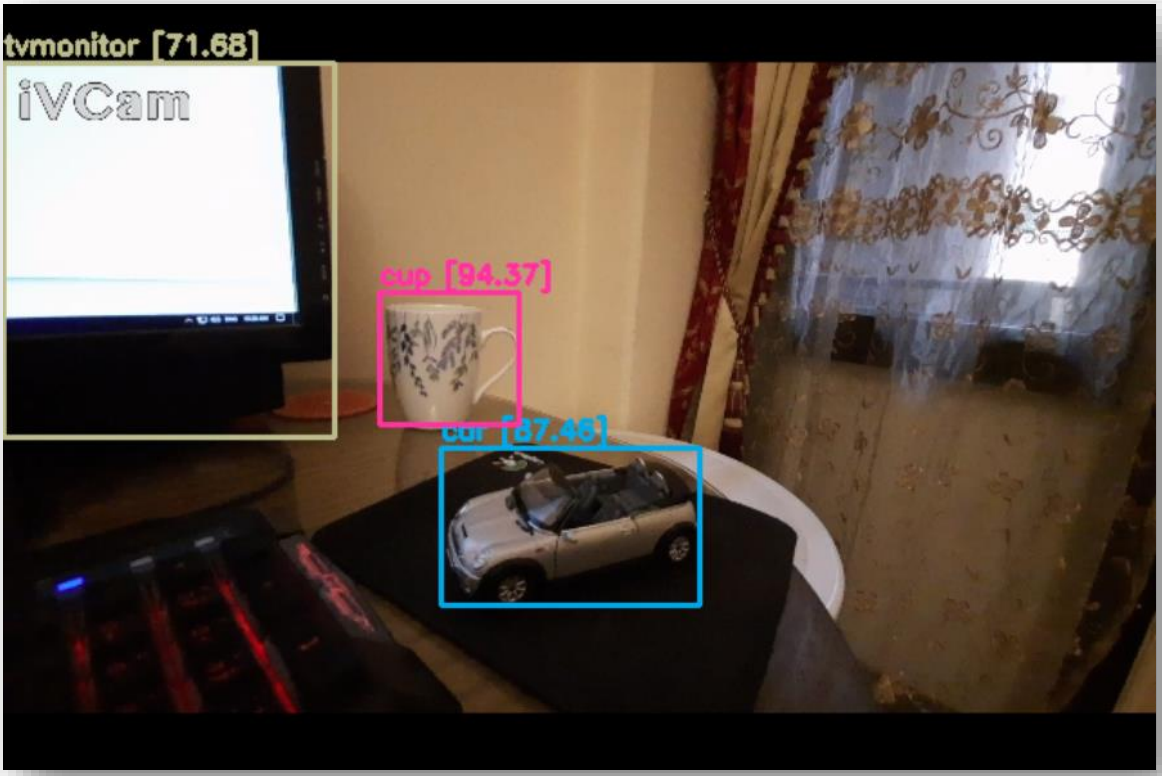


Before

After

And these are some more examples but in real-time by me:





B) Train a custom model

It's another way which used if we want to detect other objects than ones in the pretrained model.

Here, I made 3 models that could benefit our project in the future.

The first model could detect "Stairs" in pictures or videos, the second model could detect "Doors" and the last model could detect about 10 classes but it needed very much time to train as the number of classes is large.

I used about 1500 picture for training and 300 pictures for validation for every class (80% training - 20% validation).

Indeed, the pretrained model is more accurate as it trained on super computers and on very large dataset which have thousands of images and done for a large number of iterations.

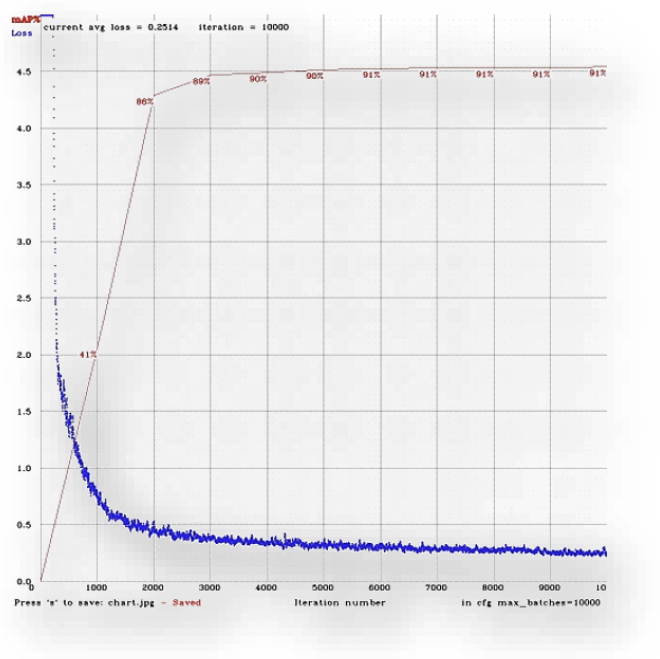
Here, before starting the train, we should modify some configurations such as:

- max_batches = number of classes * 2000, and it should be at least equal to 6000 for 1,2 or 3 classes.
- steps = (80% of max_batches), (90% of max_batches)
- filters = (number of classes + 5) * 3
- width and height are usually equal to 416.
- Batch usually equal to 64.
- Subdivisions usually equal to 8 or 16.

After these modifications, we could be able to train our model without problems.

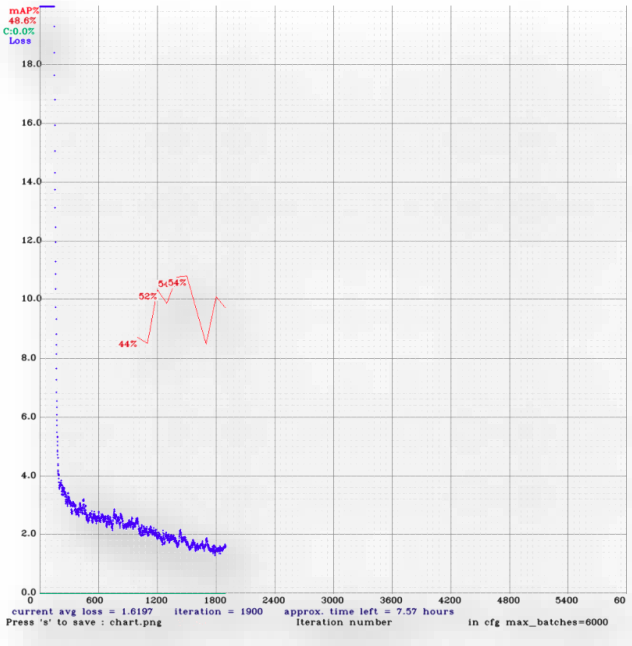
While training our model, we could see a chart of how your model did throughout the training process.

This chart shows our average loss vs number of iterations after finishing the train.

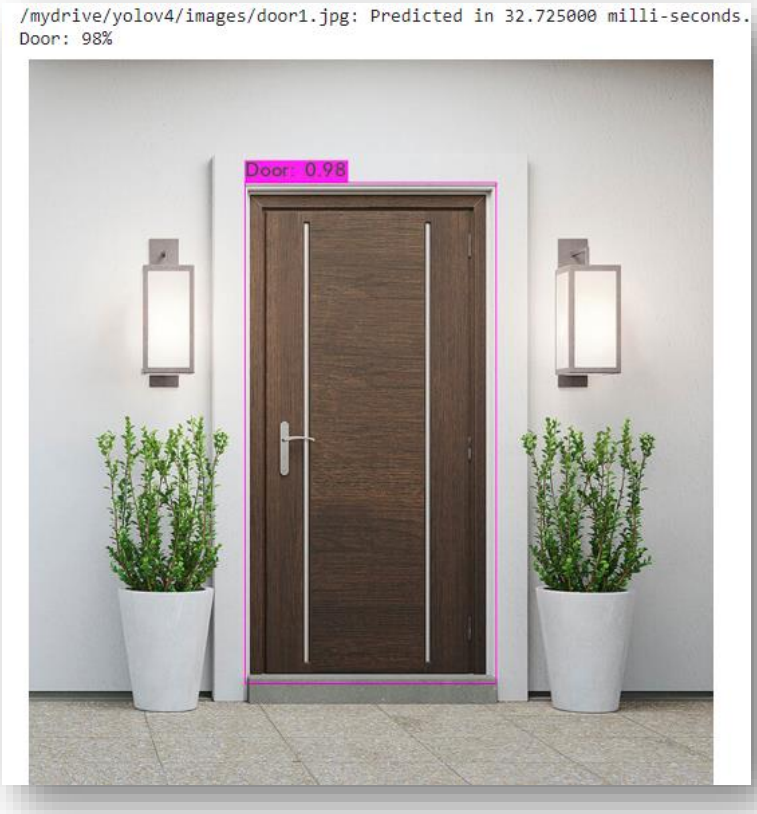


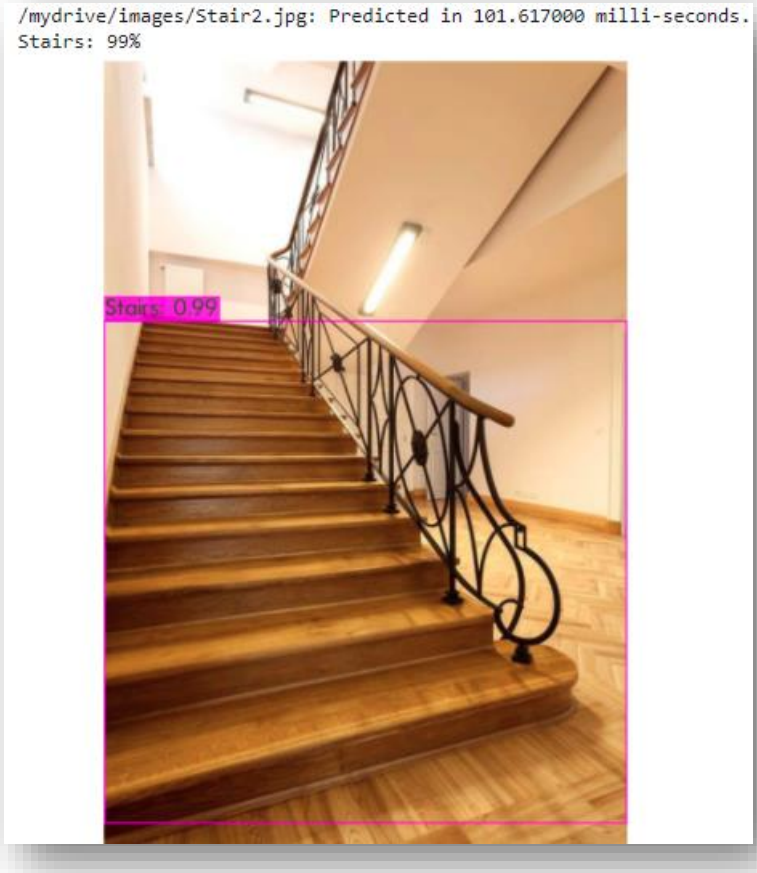
Note that we could stop custom training before totally finishing it if the chart shows that the model overfitted.

And this is another example about how it shows during another custom train:



Finally, we could try it to see how it’s doing:





Conclusion

Now, in this chapter:

We could say that object recognition and detection are very important for computer vision as they have a large number of useful applications as said before.

Also, we have seen what's the difference between object recognition and detection and also another related topics.

Then, we have seen how the Kinect sensor could be considered as one of cores of this Smart Wheelchair project and how it works.

Finally, we have seen how Yolo object detection algorithm works and what's advantages did this algorithm have.

.....

Chapter (5): Low Level

Design Procedure

SPECIFICATIONS AND CONSTRAINS:

A) A design constraint in system design refers to a limitation on the conditions in which a system is created or on the system's needs. The design limitation may be in the system's design, fit, or function, or it may be in the technology to be applied, materials to be incorporated, time required to build the system, total budget, and so on.

B) A design constraint is often applied from outside the company, either by the company or by an external regulation. It is just as vital to establish each design constraint as it is to elicit requirements during system design because the design constraints provide an overall boundary around the system design process.

DESIRED VELOCITY AND ACCELERATION

Our robot's desired velocity is 0.5 m/s, and its desired acceleration is 1 m/s2.

MAXIMUM INCLINE

As a safety, we will use a maximum slope of 5° in our motor sizing calculations.

DESIGN OPERATING TIME

A home assistant robot prototype should be able to run for about 20 minutes.

SUPPLY VOLTAGE

As a voltage source, a 12 V battery is used.

EFFICIENCY

As a safety, we will use an 75% overall efficiency in our motor sizing calculations.

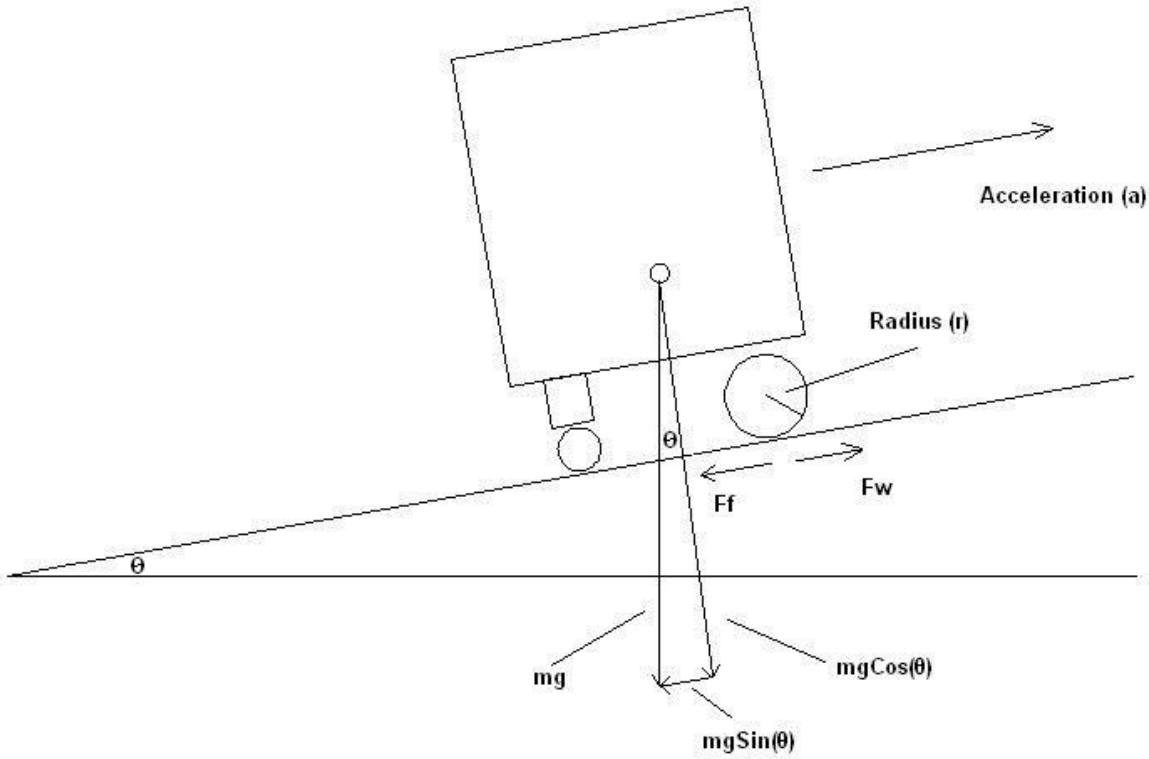
Motor Sizing

Selecting the drive motors is one of the most important decisions you can make while building a mobile robot. It is a perfect example of finding the actual one in an ideal scenario. Some of the basic physics and the practical considerations I use to pick DC drive motors for mobile robots will be covered in this report. You'll need to know what characteristics the robot you want to build would have before you can pick your motors. How big is it going to be? How much is it going to weigh? How fast can it move?

Finally, you can start searching for motors that suit your needs, prepared to practical requirements and an understanding of how they apply to the efficiency and specifications of your robot. You will not find the ideal motors for your robot in most cases, and you will need to make some exchange to make your final choice of motors.

Main Parameters for choosing the right dc-motor

- Torque
- Angular Velocity
- Power
- Maximum Current
- Battery Capacity



Before calculating our desired outputs, we should know:

- Force:

$$Force = ma$$

- Weight:

$$Weight = mass\ x\ acceleration = mass\ x\ gravity = mg$$

  - This is the force pulling the robot down toward the center of the Earth due to gravity.

  - On a free body diagram, the force of weight is broken down into its two components.

$$Force\ Pulling\ Robot\ Down\ Incline = f_g = mgsin\theta$$

- The force is pulling your robot back down the incline and must be overcome by your drive motors. The greater the angle of incline, the greater this force will be. The incline a robot is trying to climb makes a significant difference in the torque required from your drive motors.

$$Force\ Pulling\ Robot\ to\ Incline = f_n = mg\cos\theta$$

- This force is holding your robot onto the incline. This force is required along with friction to allow your robot's drive wheels to push the robot forward up the incline. We will ignore friction, because the force of friction is what the drive wheels need to push to move the robot forward.

- On our free body diagram, torque is the force at the edge of the drive wheels pushing our robot up the incline.

Calculations

- Torque

$$\Sigma Forces = f_{total} = f_w - f_g = Ma$$
$$f_w = Ma + f_g$$
$$T/r = Ma + M\sin\theta$$

Finally:

$$T = M(a + g\sin\theta)r$$

- AngularVelocity

$$W = v \times r$$

v => desired velocity

r => radius

- Power

$$Power = Voltage\ x\ Current$$
$$P_{watts} = V * 1$$

Another key relationship is:

$$Power = Torque\ x\ Angular\ Velocity$$
$$P = T * \omega$$

Maximum Current

Finally, we can find the max current from equaling the last two Equations assuming we have voltage = 12v.

OTHER CONSIDERATIONS

A) We assumed that there was enough friction between the wheel and the surface during the initial motor selection, so that there was no slipping. This is a bad assumption indeed. When the robot gets started at full speed on a slicker surface, most wheels

can spin. This can be a real issue in some situations. Selecting different wheels is one solution to this problem. Or, building up the motor speed instead of starting at full speed is a safer option.

B) Another important factor of getting the best out of your drive motors is battery choice. For the required robot run-time, the correct battery will need to provide current. Two main values are listed for batteries: voltage and amp- hours. An amp-hour is known as the amount of time one amp of current is supplied by the battery, in hours. Only take the battery rating and divide it by the average current drawing of your robot to get an approximation of how long your robot can live on a given battery.

C) My robot has a 10 amp-hr battery, for example. My robot normally consumes less than two amps, so with a fully charged battery, I expect to get four to six hours of runtime. You can use the relationship between power, voltage and current to have a normal range of the power your batteries would need to provide your motors.

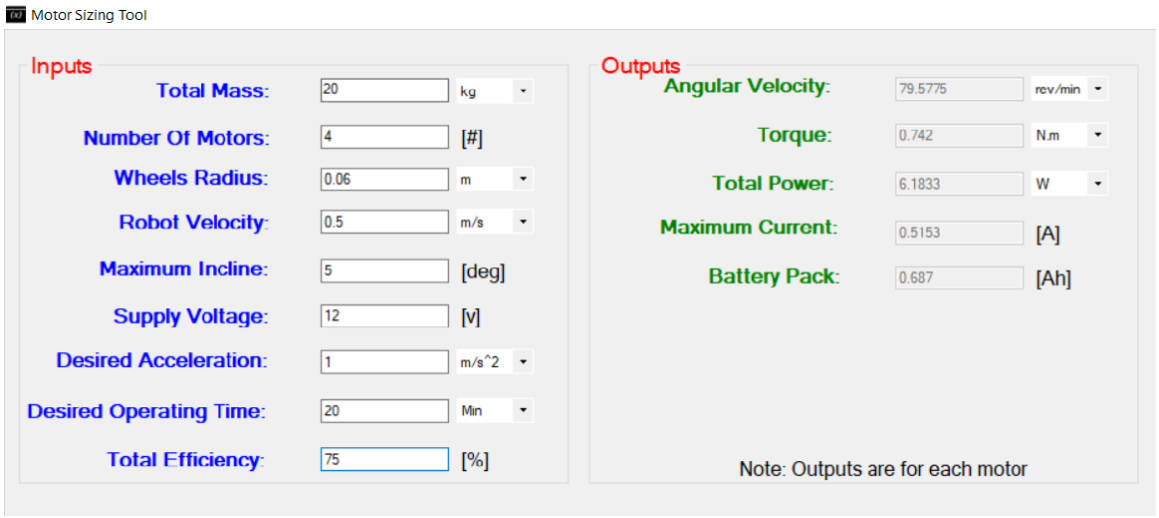
MOTOR SIZING TOOL

A) Using the equations above, we used a motor sizing tool, which was written in C#.

B) The Drive Motor Sizing Tool is designed to give you an idea of the type of motor you'll need for your unique robot by using known data and calculating the values you'll need when looking for a motor.

C) DC motors are typically used in continuous rotation drive systems, although they may also be used for partial (angle to angle) rotation. They are available in an almost limitless range of speeds and torques to fulfill any requirement. DC motors move incredibly quickly (thousands of revolutions per minute (rpm)) without a gear down, but have little torque.

D) Pick a motor with an encoder option if you want feedback on the angle or speed of the motor.



MOTOR SIZING SELECTION

Using the obtained results from the motor size calculations, we can select the motors based on the following specifications:

- Torque: 6.3 Kgf.cm
- Output Power: 2.3 W
- RPM: 50 rpm
- Voltage: 12 V
- Max Current: 800 mA

MATERIAL SELECTION

We picked Acrylic to be our main design material for the robot base and body.

ACRYLIC

WEIGHT

The weight of ACRYLITE FF sheet is only half that of glass and 43% that of Aluminum.

STRENGTH

Although the tensile strength of ACRYLITE FF sheet is 10,000 psi (69 Mpa) at room temperature (ASTM D 638), continuous loads below this value might cause stress crazing. Continuously applied design loads for glazing applications should not exceed 750 psi (5.2 Mpa) at 73°F (23°C). At 73°F (23°C), temporary loads of up to 1,500 psi (10.4 Mpa) can be applied for brief periods of time. Stresses that are focused in one area must be avoided. As a result of this, and due to thermal expansion and contraction, big sheets

should never be bolted together, but should always be placed in frames. As the temperature approaches the maximum suggested for continuous service–160°F (71°C), all thermoplastic materials, including ACRYLITE FF sheet, will gradually lose tensile strength.

DIMENSIONAL STABILITY

Although ACRYLITE FF sheet will expand and contract because of temperature and humidity changes, it will not decrease with age. When ACRYLITE FF sheet is heated to processing temperature, some shrinkage occurs, but post-forming stability is excellent.

LASER CUTTING

Laser technology is quickly gaining acceptance in the plastics industry for quick and accurate cutting, welding, drilling, scribing, and engraving. CO2 lasers concentrate a huge amount of light energy on a tiny region, making them ideal for cutting complicated designs in acrylic sheet. The laser beam creates a narrow kerf in the plastic, allowing for close part nesting and minimal waste. CO2 lasers evaporate the acrylic as they progress, resulting in a smooth polished edge but with high stress levels; heating acrylic sheet after laser cutting is suggested to reduce the possibility of crazing throughout the part's service life.

CUTTING AND MACHINING

Circular saws and band saws can be used to cut ACRYLITE FF sheet. With a few tool modifications, it can be drilled, routed, filed, and machined in the same way as wood or brass can. Because the sheet softens fast, it is critical

to maintain the cutting tool and the sheet's machined edge as cold as possible. It is advised that the cutting tool be cooled. Tool sharpness and "trueness" are critical for preventing gumming, heat accumulation, and stress in the component. Heat accumulation at the machined edge must be avoided since it can lead to stress crazing.

HEAT RESISTANCE

Based on the application, ACRYLITE FF sheet can be applied at temperatures ranging from -30°F (-34°C) to +190°F (+88°C). Temperatures should not exceed 160°F (71°C) for continuous operation or 190°F (88°C) for brief, periodic use. ACRYLITE FF sheet components should not be exposed to extreme heat sources, such as high power light bulbs, unless the finished product is vented to allow heat absorption.

ELECTRICAL PROPERTIES

ACRYLITE FF sheet provides a variety of advantageous electrical characteristics. It is an insulating material. It has a greater surface resistance than other plastics. Its electrical characteristics are unaffected by continuous outside exposure.

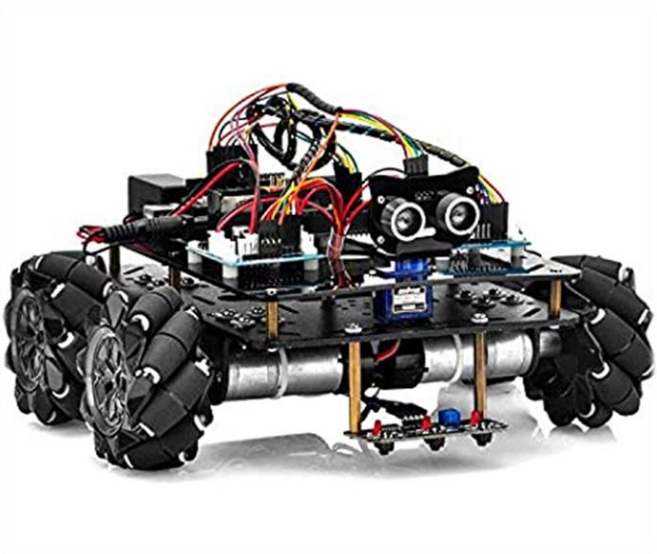
SURFACE HARDNESS

Plastic's surface is not as hard as that of glass. As a result, considerable care should be used when handling and cleaning ACRYLITE FF sheets.

MECHANICAL DESIGN

OLD DESIGN:

- At beginning, we planned to use a similar look to the AMIGO robot base, which used four Omni wheels placed in a squared form as seen below:



- We had to create our own design due to the larger size of the AMIGO robot.

FINAL DESIGN:

- After many more ideas and arguments, we decided to modify the design due to the unavailability of the wheels, and a completely new design was built this time using regular wheels.

DESIGN CONSTRAINS

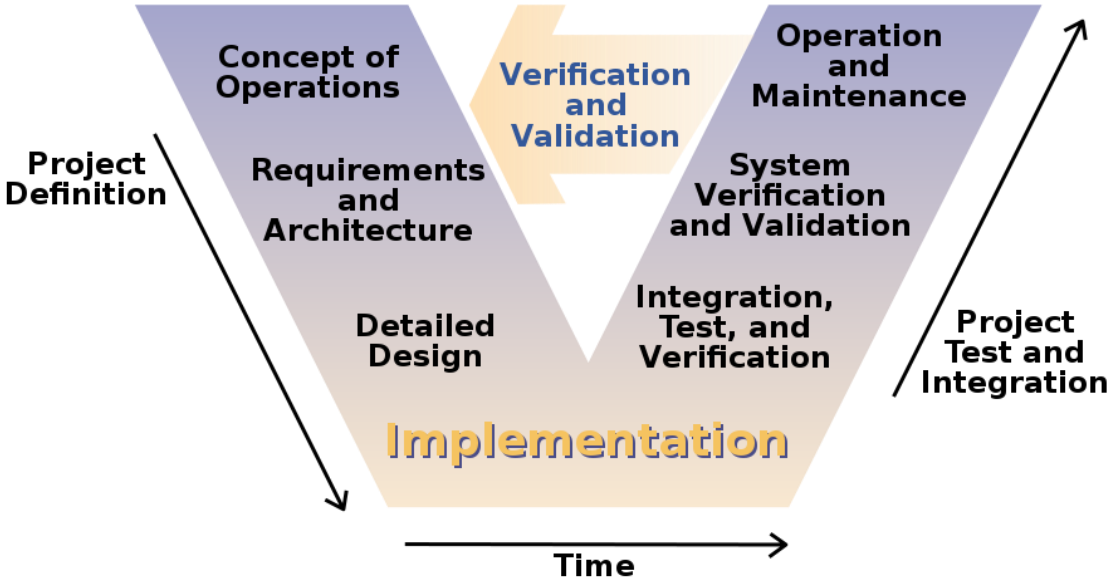
- **Length: 400 mm**
- **Witdth: 300 mm**
- **Height: 90 mm**

V-MODEL DESIGN PROCEDURE APPROACH

- Working on the robot and progressing with the tasks. We needed to design a strategy that would help us save time and reduce mistakes throughout the design process.
- The V-model in development is an example of the more general V-model, and it represents a development process that may be thought of as an extension of the waterfall model. After the coding phase, the

process steps are curved upwards to produce the usual V shape, rather than going down in a linear way. The V-Model shows the connections between each step of the development life cycle and its related testing phase. The horizontal and vertical axes, respectively, show time or project completion (left-to-right) and level of abstraction (smallest abstraction highest).

- The equivalent testing phase of the development phase is scheduled in parallel under the V-Model. So, on one side of the 'V,' there are Verification stages, and on the other, there are Validation stages. The Coding Phase connects the V-two Model's sides.



- And that is exactly how we worked in parallel to optimize the benefits of this strategy.

Wheels

Types

To consider wheels, some parameters should be put into account:

Parameter	Effect
Size	Speed, & Ground clearance
Width	Friction, & Maximum load
Thread	Surface(s) that are traveled upon
Type	Degree(s) of movement

As for the types of wheels:

Standard:

- Has 2 degrees of freedom (Forwards and Backwards)
- Works as drive or idler
- Used indoors and outdoors.
- Easy to control.
- Cheap and common
- Wide selection of treads
- Available in wide range of sizes

Caster/Ball:

- Orientable
- Works as idler or for balance
- Used mostly indoors.
- Not so easy to control due to disorientation.
- Cheap

Mecanum:

- Multidirectional (can work in every direction)
- Works as drive or idler
- Used indoors and outdoors.

- Left wheels are different from right wheels.
- sensitive to nonsmoothed terrains
- low torque for pushing.

Omni:

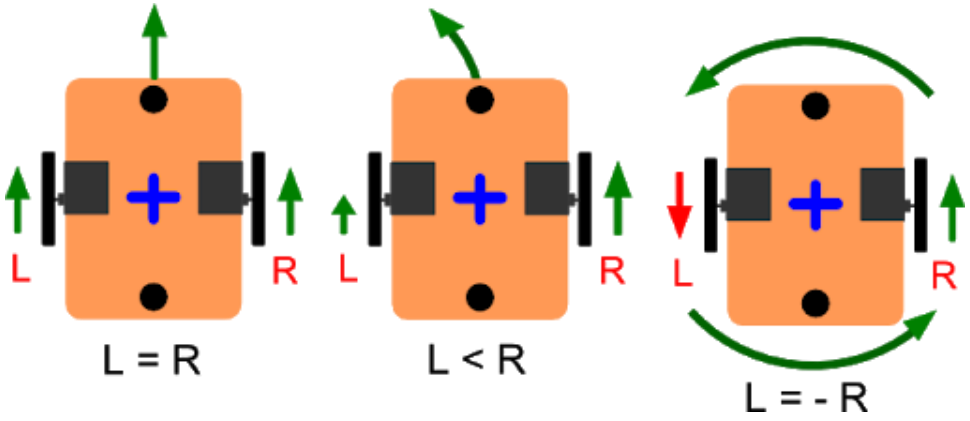
- 4 degrees of freedom (standard + sideways)
- Works as drive or idler
- Best for indoors usage
- sensitive to nonsmoothed terrains
- low torque for pushing.

Arrangements and Steering

Another thing to consider is the arrangement:

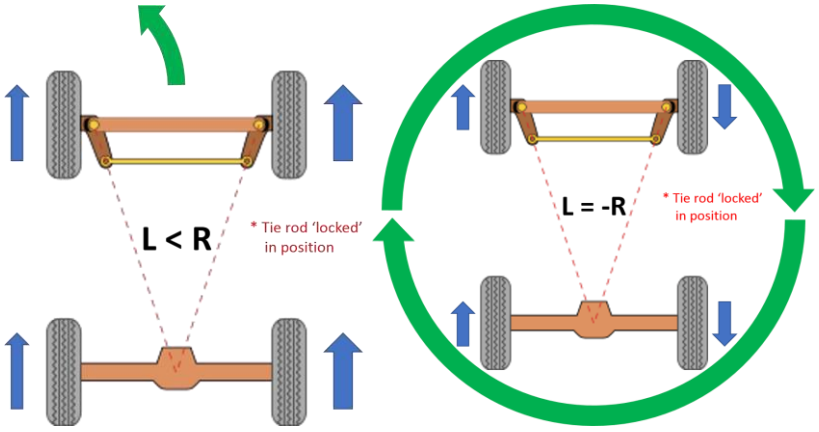
2 Motors for standard wheels:

- optional idler caster can be added for balance.
- Cheap and easy to control.



4 motors for standard wheels:

- Well balanced
- More torque
- Higher power consumptions
- Higher mass due to extra motors but the effect is nullified because of extra torque.



2 Motors for standard wheels and 2 idler omni wheels:

- 2 standards in the back, & 2 omni wheels in the front
- Simple and cheap
- Easy to control.
- Drags on turns due to the rollers, it is not too severe though.
- Some drag going forward.

2 Motors for standard wheels and 2 idler omni wheels:

- Same as the previous one but the standards are on opposite sides and the omni wheels on the other sides.
- Simple and cheap
- Easy to control.
- No drag on turns
- Some drag going forward.

4 Motors, 2 standards, & 2 omni wheels:

- 2 standards in the back, & 2 omni wheels in the front
- More expensive
- High power requirements
- More difficult to control.
- Higher torque
- Some drag on turns.
- Low drag going forward.

4 Motors, & 4 omni wheels:

- Most expensive
- Higher power requirements
- Higher torque
- Less drag on turns
- Low drag going forward.
- Can use only 2 motors to lessen the cost while maintaining the low drag on turns 4 Motors, & 4 omni wheels:
- Same as the previous one but 2 omni wheels are on opposite sides and the other two are on the other sides.
- Complex layout

- Hardest to control
- No drag on turns
- Power consumption is not as high since only 2 motors work at a time.
- The torque does not increase.
- Higher mass due to the extra 2 motors while having no extra torque.

Extra Notes

(forward) velocity = angular velocity (of the wheel) x radius (of the wheel)

This means that both the radius of the wheel and the angular speed at which it is rotating will affect the forward velocity.

force (exerted by the wheel on the surface) x radius (of the wheel) = torque (of the motor)

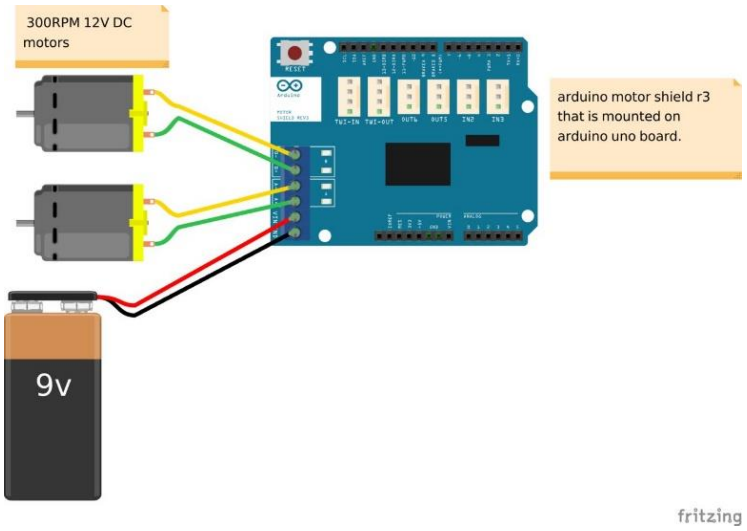
which means, smaller wheels apply larger force so if you need to exert a high force (because the robot is heavy), then you need to increase either the torque or decrease the radius of the wheel.

To achieve the same forward speed for the robot, smaller wheels must rotate faster. Even if we have a faster motor, we cannot indefinitely reduce the size of wheel due to physical size limits of motor or the shaft.

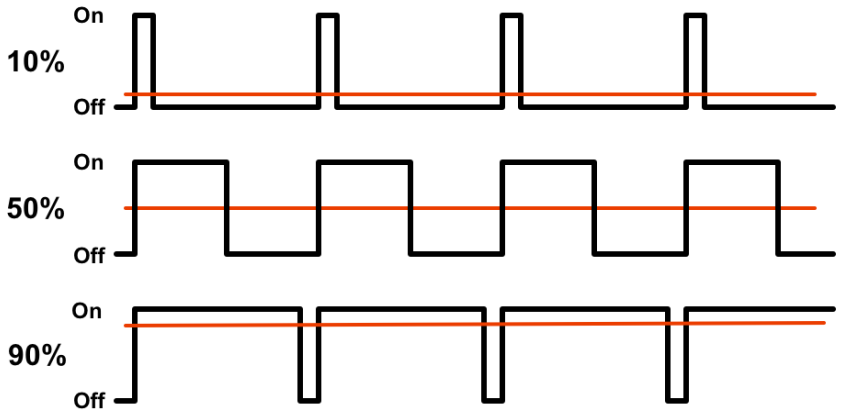
Motors

How to Control A Motor?

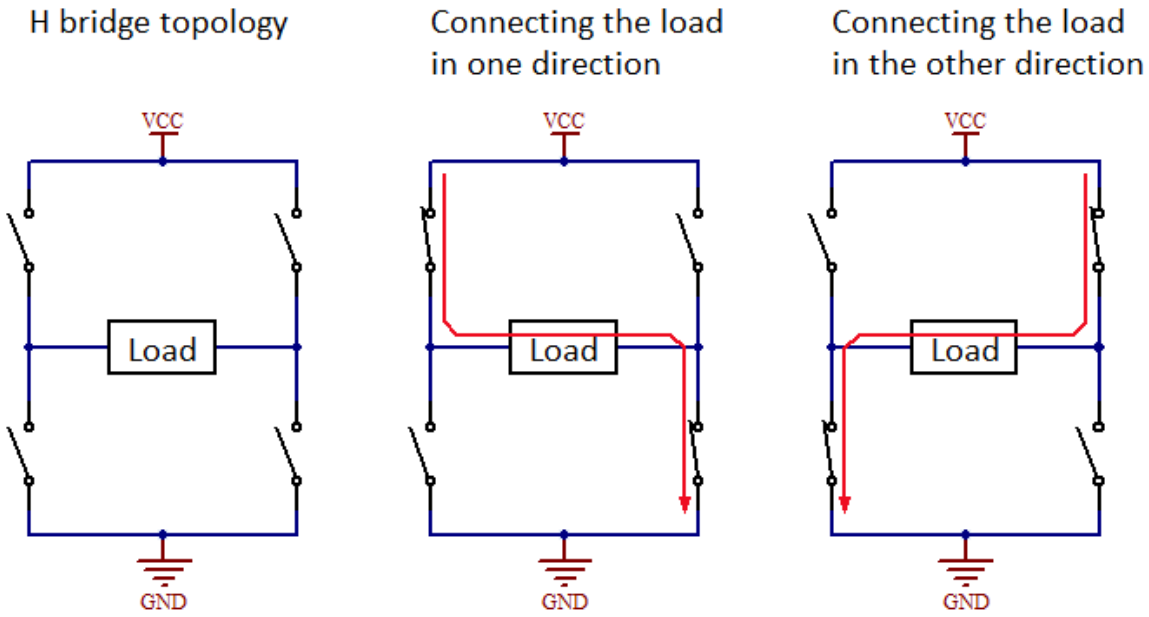
You can control a motor by simply applying enough voltage. That will make the motor rotate in a specific direction at a constant speed.



But if speed is to be controlled, an adjustment must be made. PWM is used to create pulses varied in duty time -which is the ratio between high and low voltage in a cycle- to change the average input voltage which affects the motor speed.



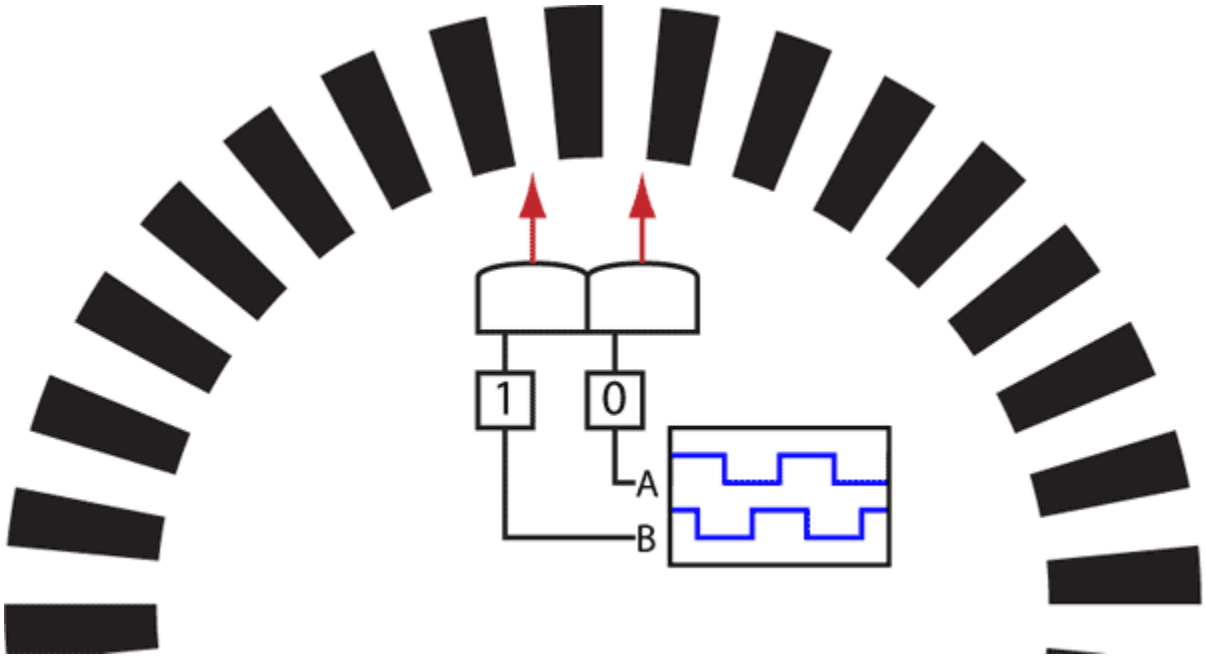
To change the direction, the current direction must be changed, this is achieved by an H-bridge.



With the microcontroller to apply the PWM and an H-bridge, motors can be controlled in speed and direction. That can create an open-loop control for the motor but for feedback an encoder is used.

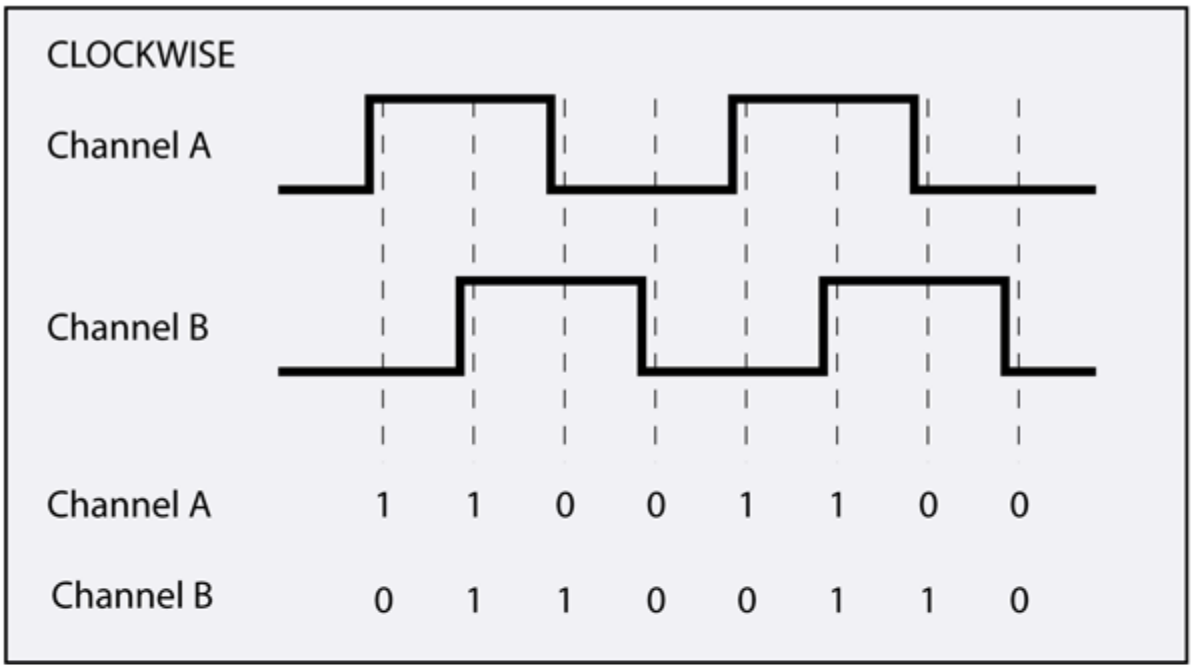
Encoders

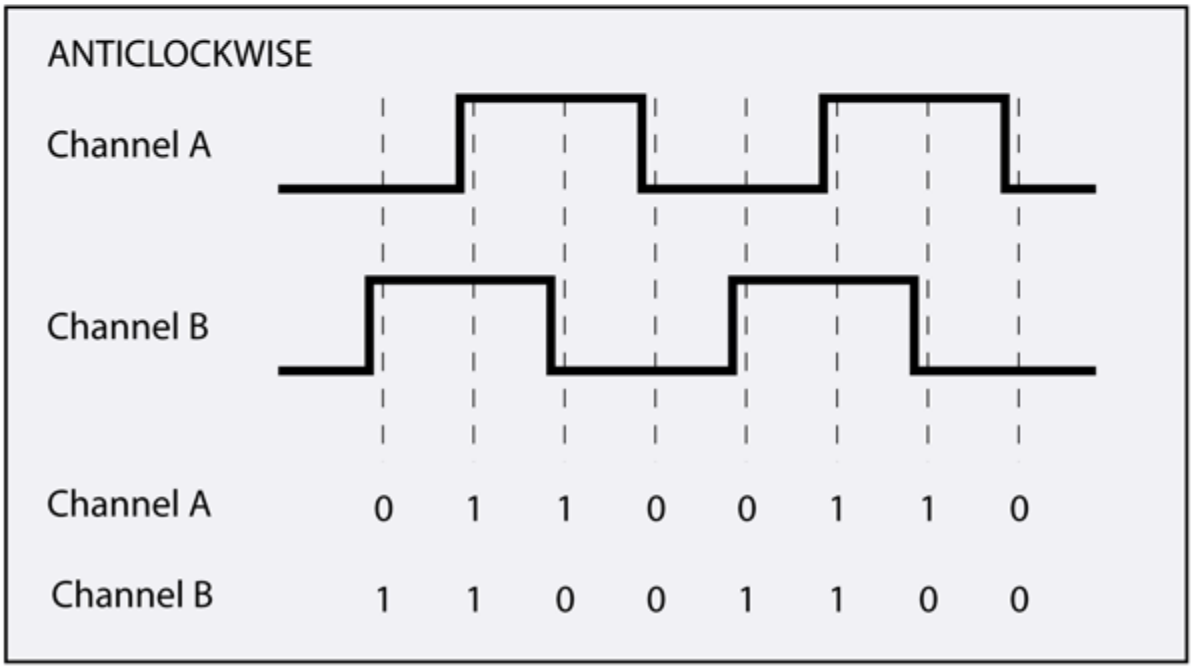
Encoders are used to convert mechanical movement of the motor to electrical variables. They are used to track the speed and angle position of the motor. Shaft encoders are used for motors. They are basically a disk with gaps connected to the shaft and rotate with the motor. Some method of detecting the gaps are used to sense the position and speed.



Gaps detectors may be optical detectors that has a light source in the other side of the desk so when light is detected then a gap is detected, or hall effect encoders that uses a magnet ring instead of disk and a piece of some conductive element say metal, the volt across the element changes according to the gap between it and the magnets, high volt means nearer magnet and lower means a gap. Both output a square wave that can be used to determine speed.

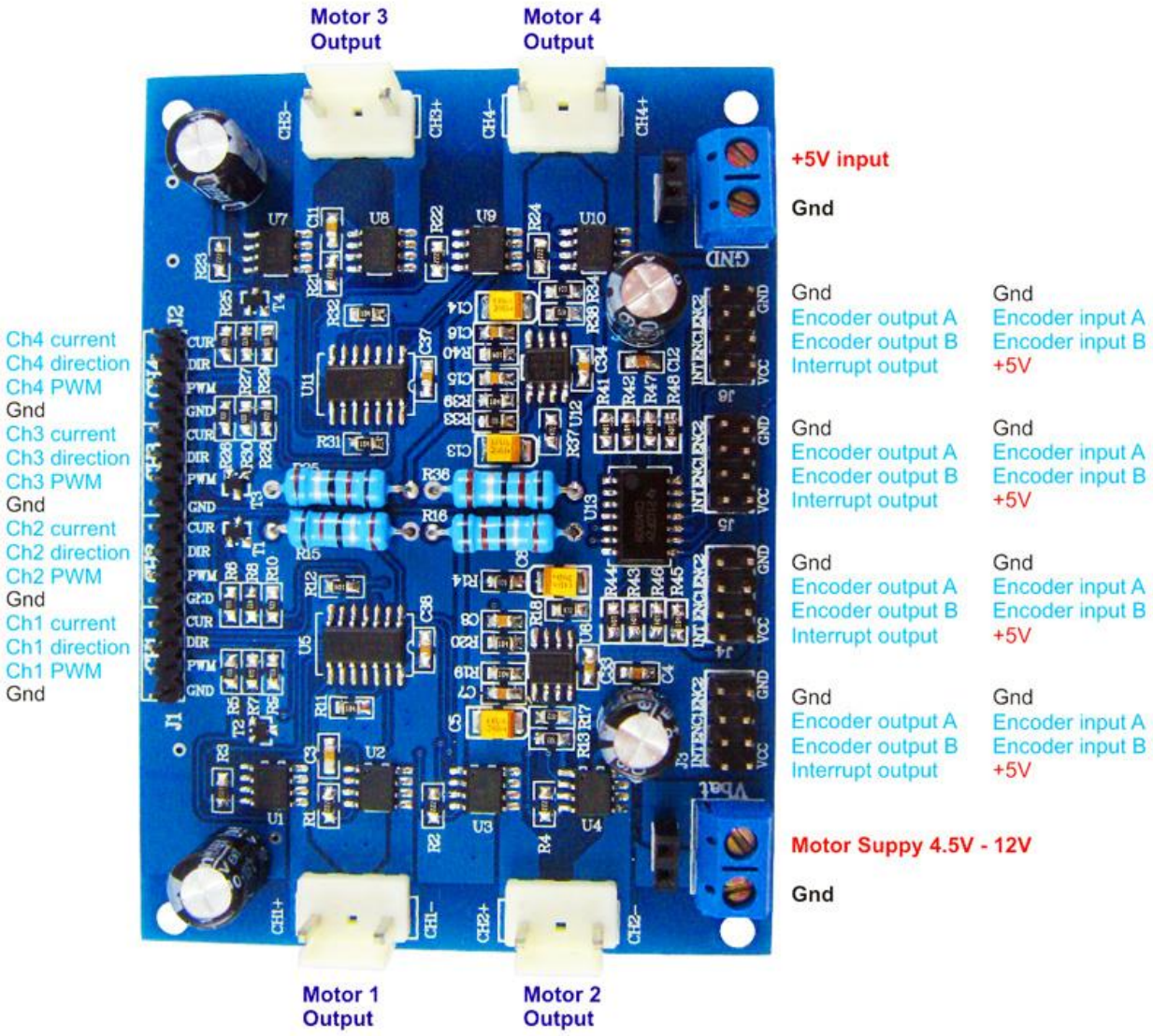
Another sensor is used side by side to detect the position of the rotating as follows.





Parts Which Used and Connection

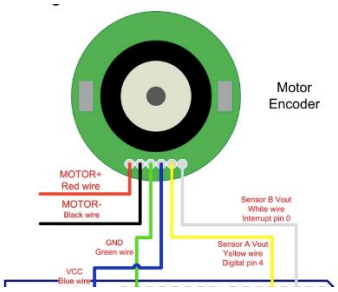
4 channel motor drivers



To fully connect a motor and control it you have to connect the power supply of the motor to pins in the bottom right corner, connect logic power supply (0-5v) to the top right corner pins, connect channel direction and PWM to the microcontroller, and then take the output of the motor and connect it to the encoder.

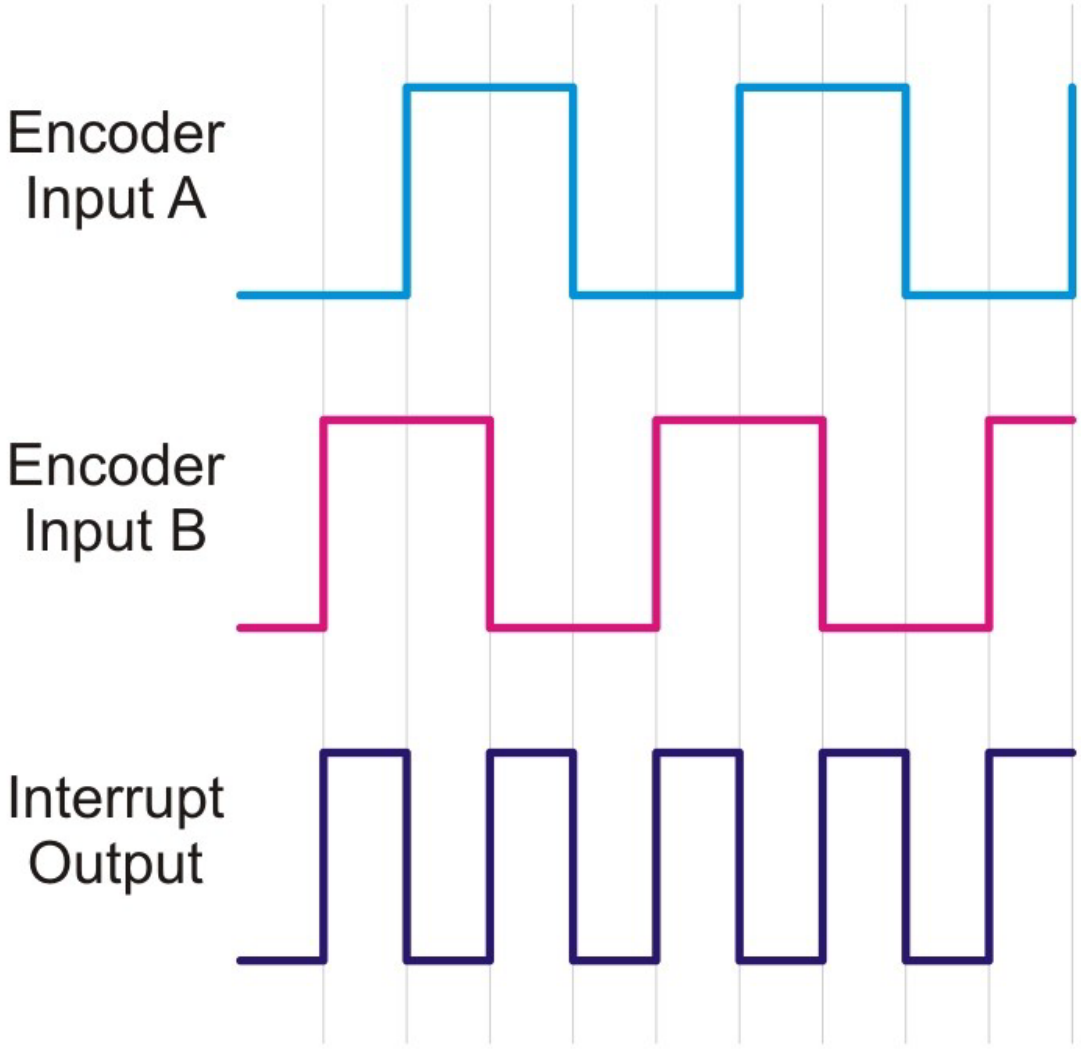
Motor

Encoder



you have to attach the encoder to the shaft connecting the motor+ and motor- to the motor power from the driver, then connect the ground and the VCC of the encoder, and finally the output sensors are connected to the desired pins in the microcontroller.

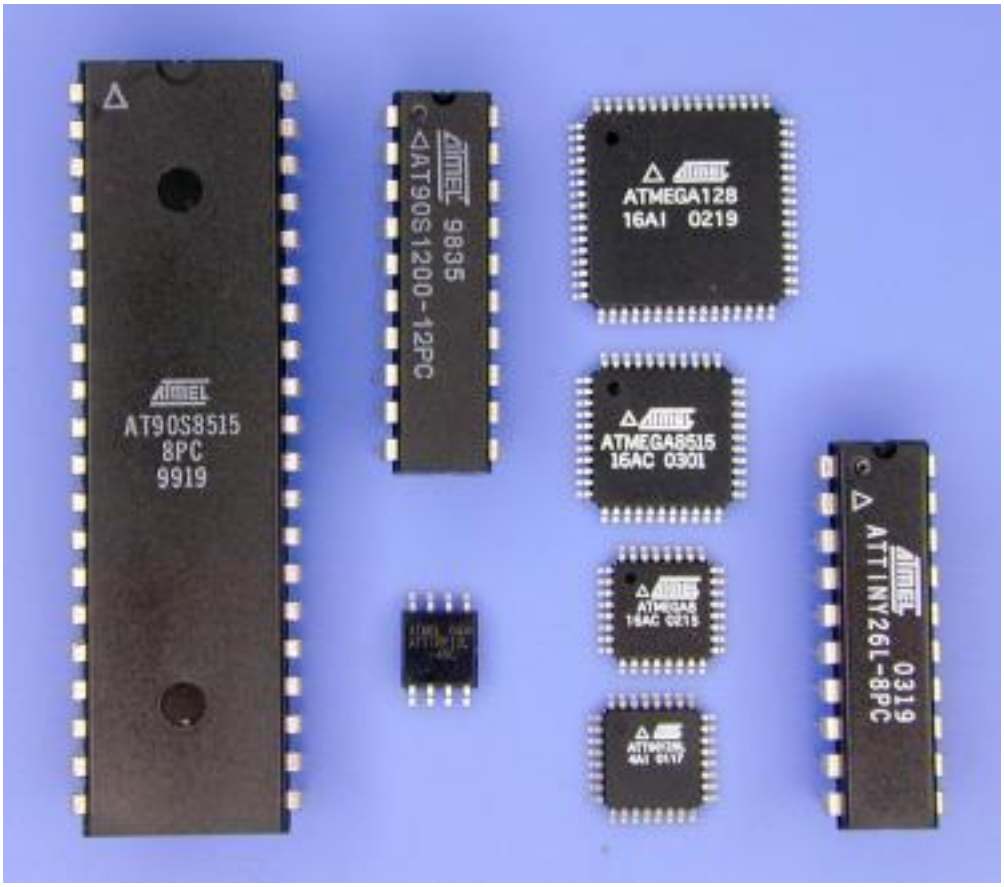
For further optimization, the sensor output can be connected to the right pins in the driver “encoder input” to get an interrupt from “interrupt output” in the beginning of the cycle as follows.



What is the advantage? The start of the wave can be used to determine the speed and direction according to whether A is leading or B, but the rest of the period is useless so instead of wasting processing time in waiting for the other pulse, the microcontroller can only pay attention on interrupt and continue to do other things while the next interrupt occur.

to ensure Raspberry PI is not overloaded with tasks, a microcontroller is added to collect data from sensors and send it to be processed by the ROS then the outcome of processing is sent to another controller that controls the motor to move the chair accordingly.

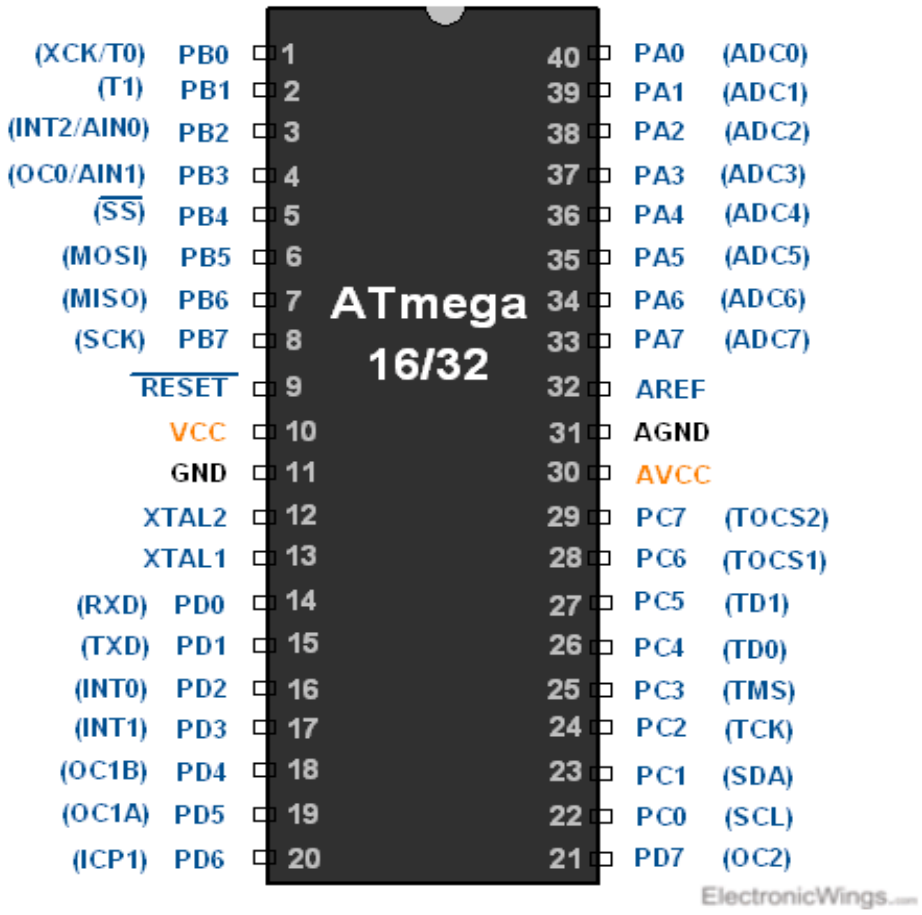
Microcontroller



The microcontroller used is the AVR. AVR is available in the market with a cheap cost and lots of tutorials and references. AVR has many families that comes with different packages and sizes, but we used the atmega32. Atmega32 comes with 44 pins, 4 ports and built-in peripherals such as:

- 2 8-bit timers
- 1 16-bit timer

- 8 channel 10-bit ADC
- Support for UART/USART, SPI, I2C for communication
- ICU
- 4 PWM



Other AVR chips may vary in memory, speed, peripherals, or pin number but atmega32 worked fine and atmega16 works fine as well.

Programming language

AVR uses the C programming language mainly but can be programmed with other languages depending on the compiler. AVR C compiler is available for free from the manufacturer alongside with an IDE called Atmel studio. The official compiler uses C/C++ so that is why C language is used also C has a wide variety of tools that help programming a low level microcontroller chip like the AVR due to the ability to access memory directly and manipulating it with a better speed than other programming languages. Speed is the most important aspect when choosing a language because the ROS needs a continuous stream of sensor readings to perform well so the controller have to collect data and send it in a matter of milliseconds or less.

Simulation

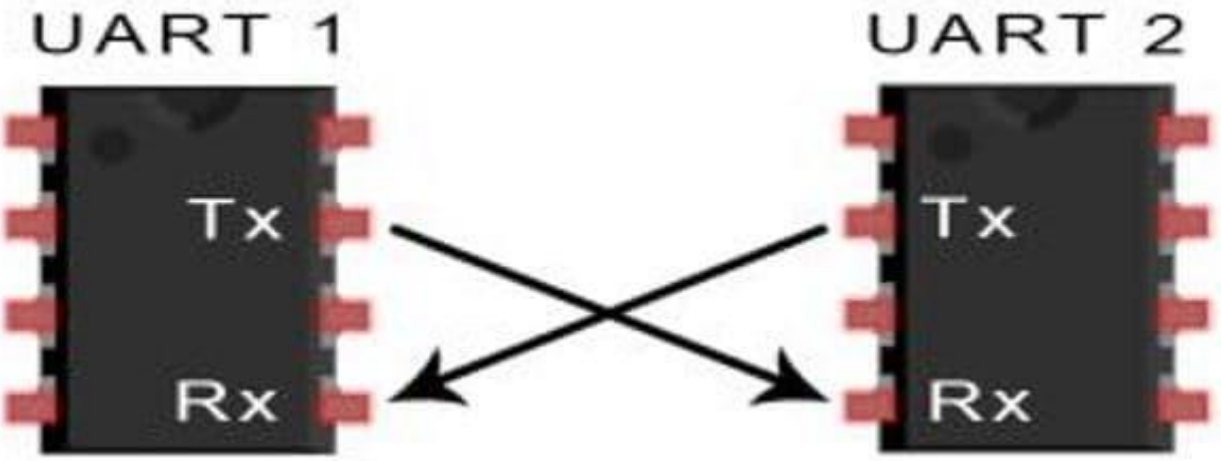
Many simulation programs are available, but Proteus proved to be a reliable software that done its mission with excellency. Proteus provide an extensive library of components and debugging tools with real time simulation and terminal. Version 8.9 is used, and it is stable with all the components needed to simulate the project.

Communication

The atmega32 comes with built in support for different communication protocols such as UART, ISP, and I2C. these protocols in theory provide the same functionality of communicating with other micro controller and even communication with the raspberry pi, but there are some differences about how they achieve that.

UART

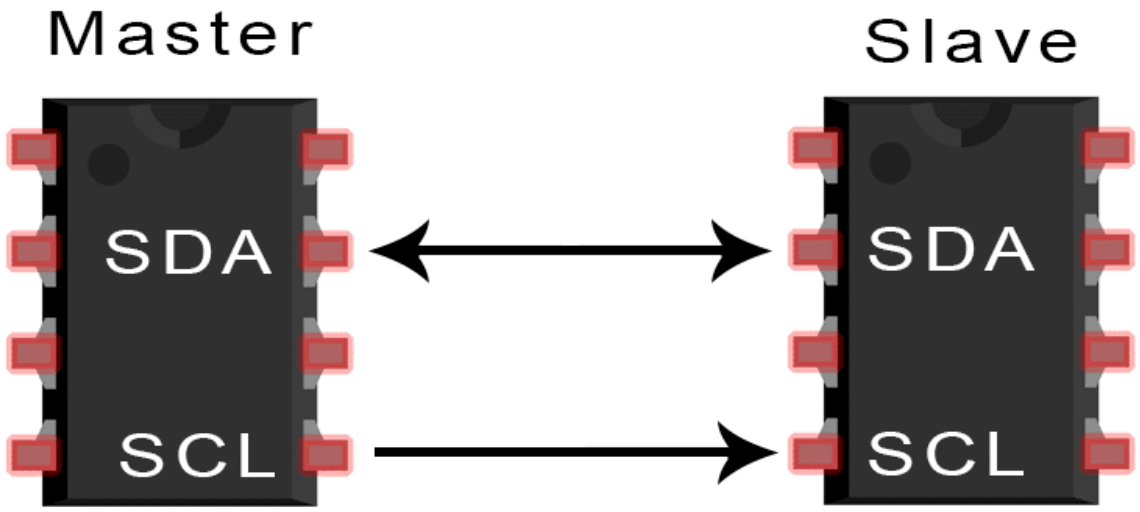
Stands for Universal Asynchronous Reception and Transmission. It is a serial communication protocol that allows the host communicates with the other devices. It supports bi-directional, asynchronous, and serial data transmission and has two data lines, one to transmit (TX) and another to receive (RX). TX and RX are connected between two devices. (e.g. USB and computer). UART can also handle synchronization management issues between computers and external serial devices.



The transmitter UART will first receive data from a data bus that is sent by another component (e.g. CPU) then it will add a start bit, a parity bit, and a stop bit to create the data packet. The data packet is then transmitted at the TX pin where the receiving UART will read the data packet at its RX pin. Data is sent until there is no data left in the transmitting UART.

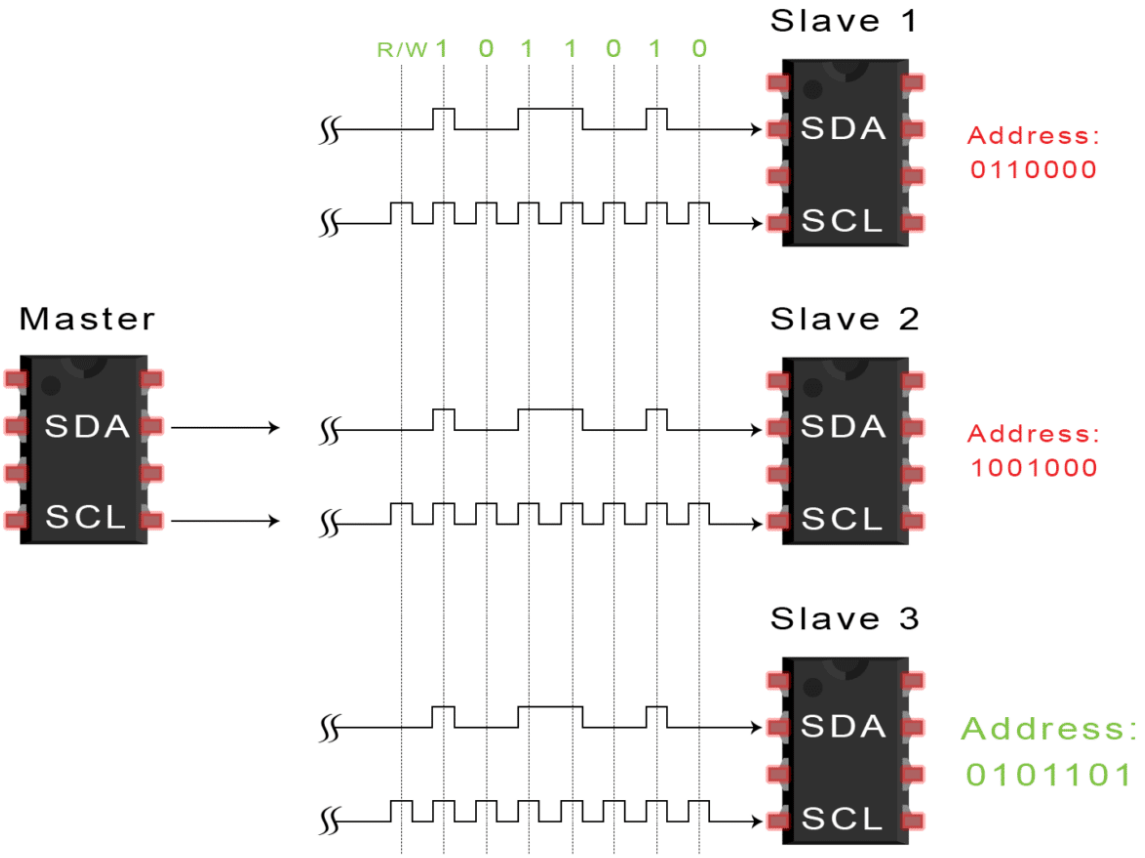
I2C

Stands for Inter-integrated circuit. It is a serial communications protocol similarly to UART. However, it is not used for PC-device communication but instead with modules and sensors. It is a simple, bidirectional two-wire synchronous serial bus and requires only two wires to transmit information between devices connected to the bus. I2C is best when dealing with many different parts (e.g. sensors, pin, expansions, and drivers) working together as they can connect to 128 devices while maintaining a clear communication pathway. The I2C protocol is also used as a two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters,



I/O interfaces, and other similar peripherals in embedded systems. The master sends the transmitting signal to every connected slave by switching the SDA line from a high voltage level to a low voltage level and SCL line from high to low after switching the SDA line. The master sends each slave the 7

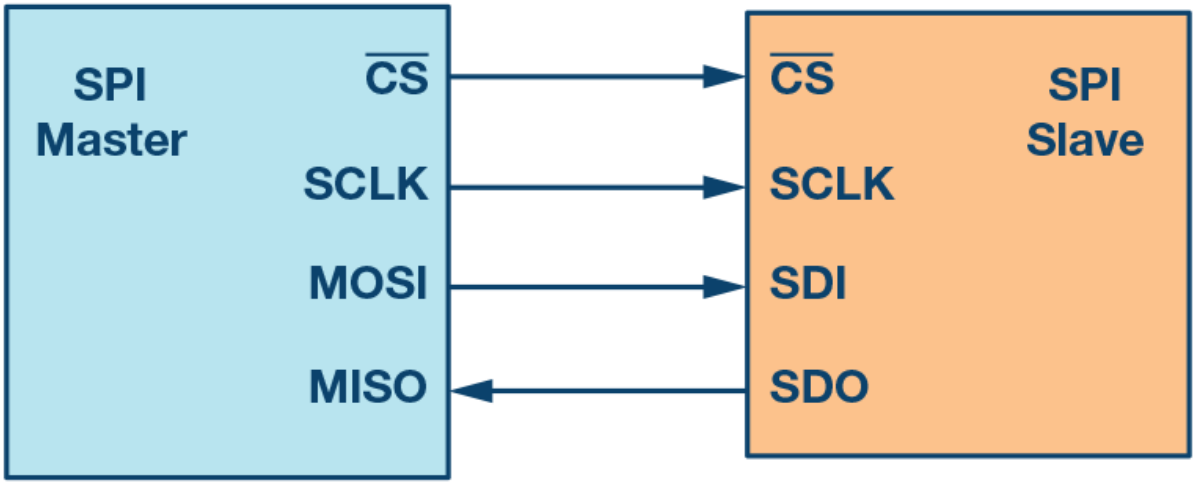
or 10-bit address of the slave and a read/write bit to the slave it wants to communicate with.



The slave will then compare the address with its own. If the address matches, the slave returns an ACK bit which switches the SDA line low for one bit. If the address does not match its address, the slave leaves the SDA line high. The master will then send or receive the data frame. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful transmission. To stop the data transmission, the master sends a stop signal to the slave by switching SCL high before switching SDA high.

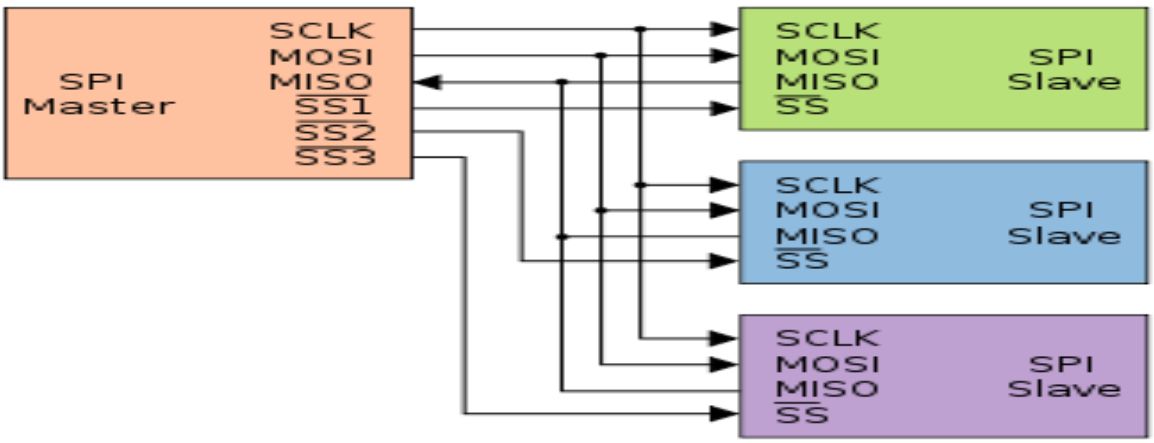
SPI

Stands for Serial Peripheral Interface. It is like I2C and it is a different form of serial-communications protocol specially designed for microcontrollers to connect. Operates at full duplex where data can be sent and received simultaneously. Operate at faster data transmission rates = 8Mbits or more. It is typically faster than I2C due to the simple protocol. Even if data/clock lines are shared between devices, each device will require a unique address wire. Used in places where speed is important (e.g. SD cards, display modules or when info updates and changes quickly like thermometers).



The SPI communicates via 4 ports which are:

- MOSI – Master Data Output, Slave Data Input.
- MISO – master data input, slave data output.
- SCLK – clock signal, generated by the master device.
- CS(NSS/SS) – Slave enabled signal, controlled by the master device.



In a multi-slave system, each slave requires a separate enable signal, which is slightly more complicated on hardware than the I2C system.

The SPI interface is two simple shift registers in the internal hardware. The transmitted data is 8 bits. It is transmitted bit by bit under the slave enable signal and shift pulse generated by the master device. The high bit is in the front and the low bit is in the back.

Used Protocols

The UART protocol is used for the communication between the AVR and the Arduino because it is simple to operate, well documented as it is a widely used method with a lot of resources online, no clock needed since it's an asynchronous protocol, Parity bit is added to allow for error checking.

UART has some disadvantages that doesn't affect its purpose like the size of the data frame that is limited to only 9 bits but since we are sending numerical values with limited range , the effect is not perceivable. UART cannot use multiple master systems and slaves which also does not harm the project since it only has to connect to only one other device.

Baud rate

Baud	% of error
300	0.2
600	0.2
1200	0.2
2400	0.2
4800	0.2
9600	7.5
14400	7.8
19200	7.8
28800	7.8
38400	22.9
57600	7.8
76800	22.9
115200	84.3

While using the UART protocol, the sent and received characters did not match, that is due to high error rate. The error rate is affected by both the CPU speed and the baud rate. Atmega32 comes with a default 1 MHZ clock speed that can be increased using an external clock (Crystal). According to the datasheet, for 1MHZ clock any baud rate higher than 4800 causes the error rate to increase from 0.2% to 7.5% and can increase with higher baud rate up to 84.3% while using 115200 baud rates. In simulation, this error will result in wrong characters being sent, but in real life this will not work at all.

Debugging

Unlike software programming where you can run the code and get the error printed on the screen, in microcontrollers, the code is burnt on the chip and runs independently of the computer. So, in case of any runtime error, there is no way of printing errors and that makes the debugging phase harder.

So, to debug the code, a dedicated connection must be made for that.

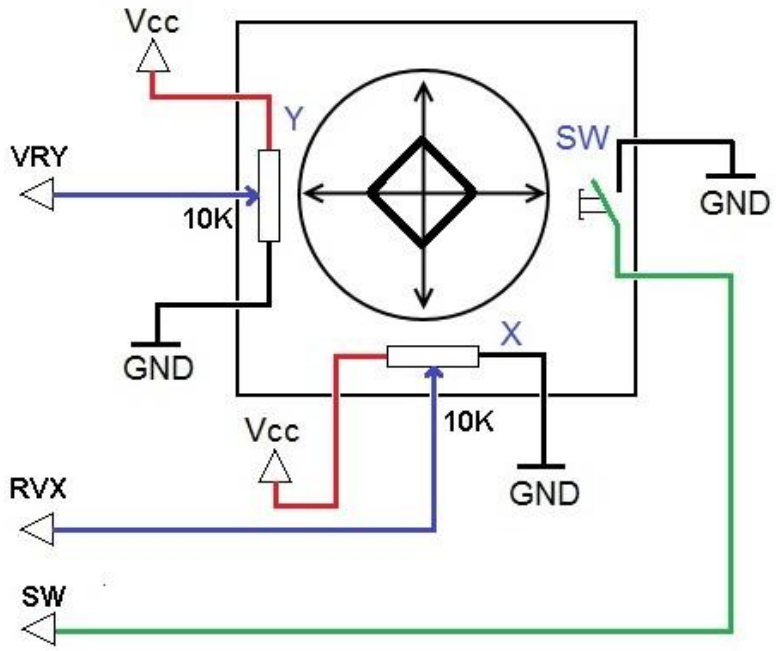
UART or any other communication system but since the UART is widely used and available, we used it to print our own custom-made debugging messages.

One downside is that UART sends only ASCII characters which covers all the capital and small letters in English alphabet, but the numbers are only from 0 to 9. Since the whole AVR job is based around sending sensors reading values to the ROS, sending multi-digits integers and floats is a must. On way to do this is to convert the value into a string using C standard libraries, but also you can split the multi-digit number into many single-digit numbers.

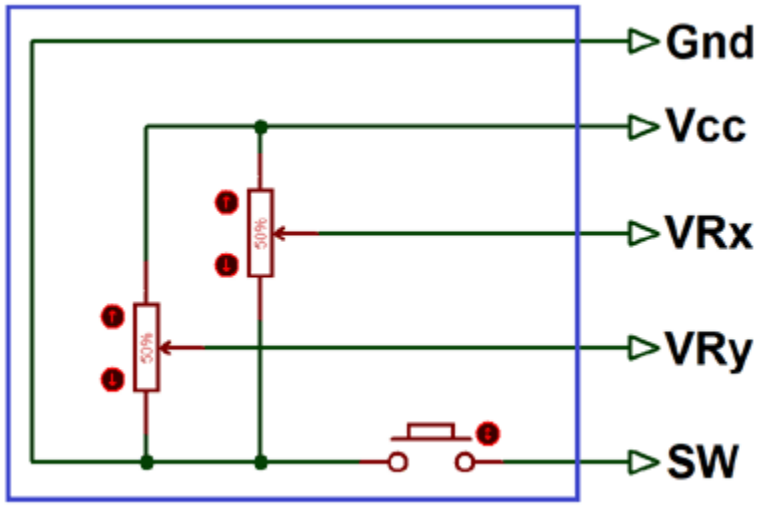
Joysticks

Joysticks are designed to be used to move the chair independently of the ROS which can be useful in many cases like mapping the area which is needed for the path planning to function properly. Also, it is useful if the user wanted to reach unregistered location or in case of a system failure.

Unfortunately, the joystick model was not available in proteus and the library is not available for download but there is a solution to this. The Joystick is made of two potentiometers for the X, and Y axes, and a button.



Using this information, a simulation can be achieved like this.



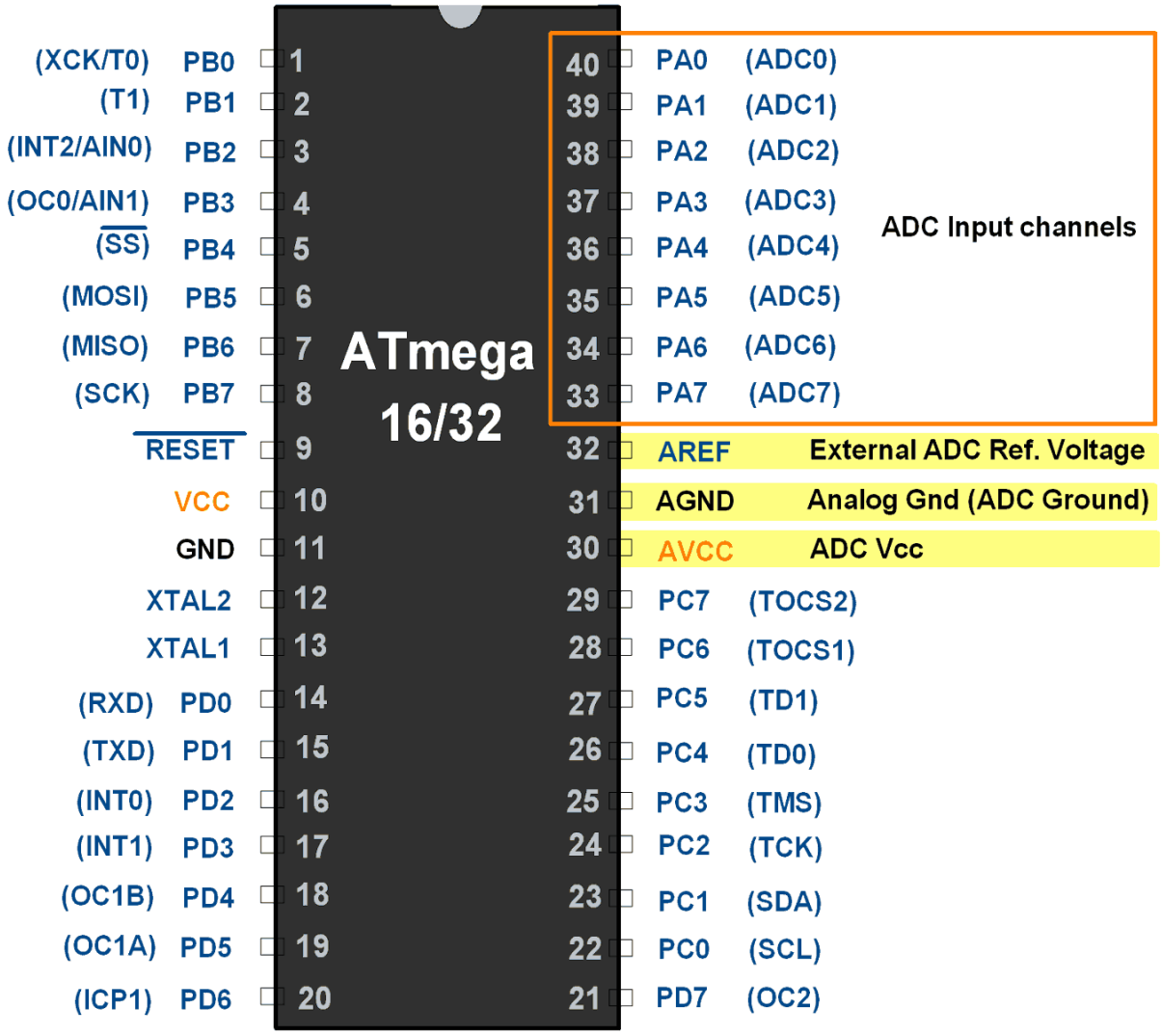
The values the potentiometers readings vary from 0 to 1024 with 512 in the center. Using these values, the direction of the joystick can be found.

```
directions Joystick_getDirection(){
    int pos[2];
    Joystick_getPosition(pos);
    directions direction = CENTER;
    if(pos[X] == 512) // center x
    {
        if (pos[Y]==512) direction = CENTER;
        else if (pos[Y]>512) direction = NORTH;
        else direction = SOUTH;
    }
    else if (pos[X] > 512)
    {
        if (pos[Y]==512) direction = WEST;
        else if (pos[Y]>512) direction = NORTHWEST;
        else direction = SOUTHWEST;
    }
    else
    {
        if (pos[Y]==512) direction = EAST;
        else if (pos[Y]>512) direction = NORTHEAST;
        else direction = SOUTHEAST;
    }
    return direction;
}
```

To read the analog values of the potentiometers, an Analog to Digital Converter (ADC) is used.

ADC

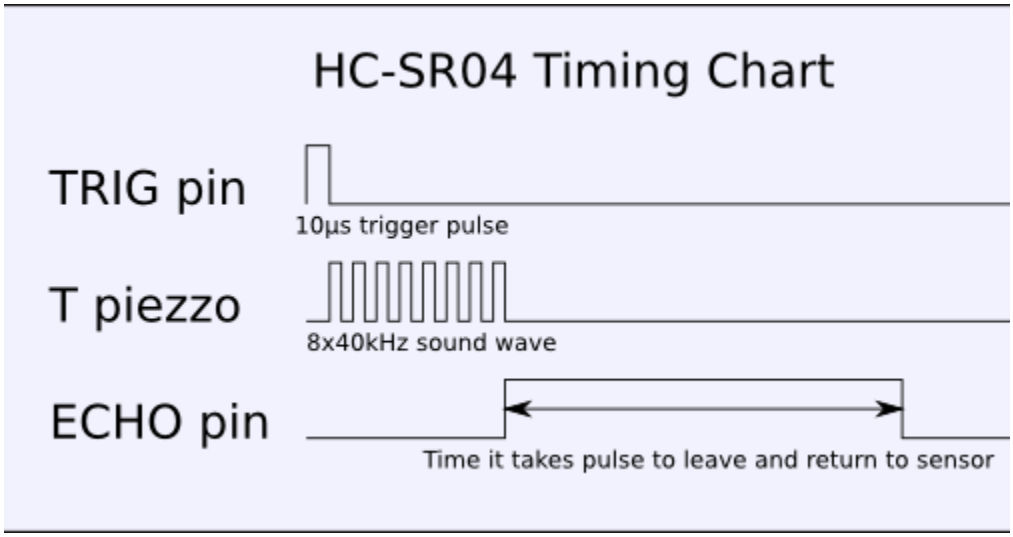
ADC capabilities are built-in the AVR microcontrollers that are connected via the 8 pins of port A. the 8 pins can be used as channels to read multiple analog sensors but cannot be read simultaneously. It also has 10-bit resolution which makes it detect less changes more accurately.



Ultrasonic sensor

The Ultrasonic sensor is used mainly to detect the distance of obstacles that are in the camera's blind spot. The project has 6 ultrasonic sensors located in different places on the platform. The ultrasonic sensor has 4 pins: VCC, GND, TRIG, and ECHO.

When the sensor receives a pulse with width of at least 10 us on the trigger (TRIG) pin it sends eight cycles of ultrasonic waves at 40kHz that is reflected on collision with any object. the sensor then receives the reflected wave and produce a pulse on the echo pin with a width that depends on the time between the sending and receiving of the pulse. Knowing the speed of sound, we can deduce the distance of the object.



We have tried many techniques to optimally read the value of the pulse on the echo pin.

First: stay in a loop waiting for any rising edge from the echo pin then start a timer until detecting a falling edge. The timer value then is the width.

That technique is straightforward and simple to code, but it is far from optimal. The loop waiting for the echo pin is wasting so much of the CPU time, and due to the limited resources available, this solution is not applied.

Second: instead of looping, use the external interrupt pin to detect the rising edge, then in the ISR (Interrupt Service Routine or in other words the code executed when the interrupt happens) a timer is started to count the edge width.

Since the second method uses interrupt it saves the CPU resources that was consumed by the loop in the first method, but two issues appeared. First, AVR supports only up to three external interrupts and we have six ultrasonic. Second is the time taken while the timer counts the width of the ECHO pulse.

Third: use the ICU.

The best solution found, and it helped a lot with the wasting-resources problem, provided an automatic interrupt-based solution to catch the width of the ECHO pin.

ICU

The input capture function is used in many applications such as: Pulse width measurement, Period measurement, Capturing the time of an event.

In AVR, Timer1 can be used as an input capture to detect and measure events happening outside the microcontroller. Upon detection of a defined event

i.e. rising edge or falling edge on ICP pin (PORTD.6), the Timer register value is loaded into the input capture register.

The ICU is based mainly on the timer1 functionality in the AVR.

Timers

The AVR has several timers depending on the chip. Atmega32 has 3 timers, 2 with 8-bit registers (timer0, and timer2), and 1 with 16-bit register (timer1). Timer1 has more functionality than the others and one of those is the ICU.

The timer1 has a 16-bit register, that means it can count to (2^16 – 1), or 65535.

With a clock speed of 1MHz, a tick happens every 1 us, which means the counter can count to 65535 us or 65.535 milliseconds before it overflows and reset to 0.

AVR provides an interrupt for the overflow that can be handled to count how many times the time overflew. By creating a variable to count the times of the overflows by incrementing that variable in the interrupt handler. Then whenever the value of the timer is needed then it will be time ticks = timer register (that counts until overflow) + count of overflows x 65535 (the value of ticks in one overflow)

With the value of the register, the count of the overflows, and the frequency of the clock, we can deduce time.

Ultrasonic readings

After detecting the pulse with the ICU and counting the width with the timer1, we get the value of the distance.

Distance = (sound velocity * timer) / 2 = (34300(cm/s) * timer)/2

Since the timer ticks every us, then we multiply by 10^-6 to get it in seconds.

Then distance = 17150 x (TIMER value) x 10^-6 = TIMER value / 58.309.

One problem for using the ICU functionality is that there is only one pin for the ICU and 6 sensors. The solution is simply using a multiplexer to read the sensors one by one.

Multiplexer

We used 8:1 MUX to multiplex between the 6 sensors. The multiplexer driver is straight forward, using 3 pins to select the sensor and connecting the output to the ICU.

```
void MUX_init(){
    SELCTOR_DDR |= ( (1<<SELECTOR_0)|(1<<SELECTOR_1)|(1<<SELECTOR_2) );
}
void MUX_select(char i){
    SELCTOR_PORT &= 0b11100011;
    SELCTOR_PORT |= ((i << 2) & 0b00011100);
}
```

Autonomous Mobile Robot

An autonomous mobile robot is one that is capable of performing a wide range of activities with a high degree of autonomy (the capacity to interact with the environment and make uninformed judgments in a variety of scenarios).

Three fundamental problems must be answered in autonomous mobile robots:

1- What am I doing here?

2- What am I going to do?

3- What is the best way to get there?

In order to answer these questions, the robot must possess the following abilities:

1- create an environment model (given or autonomously built — map)

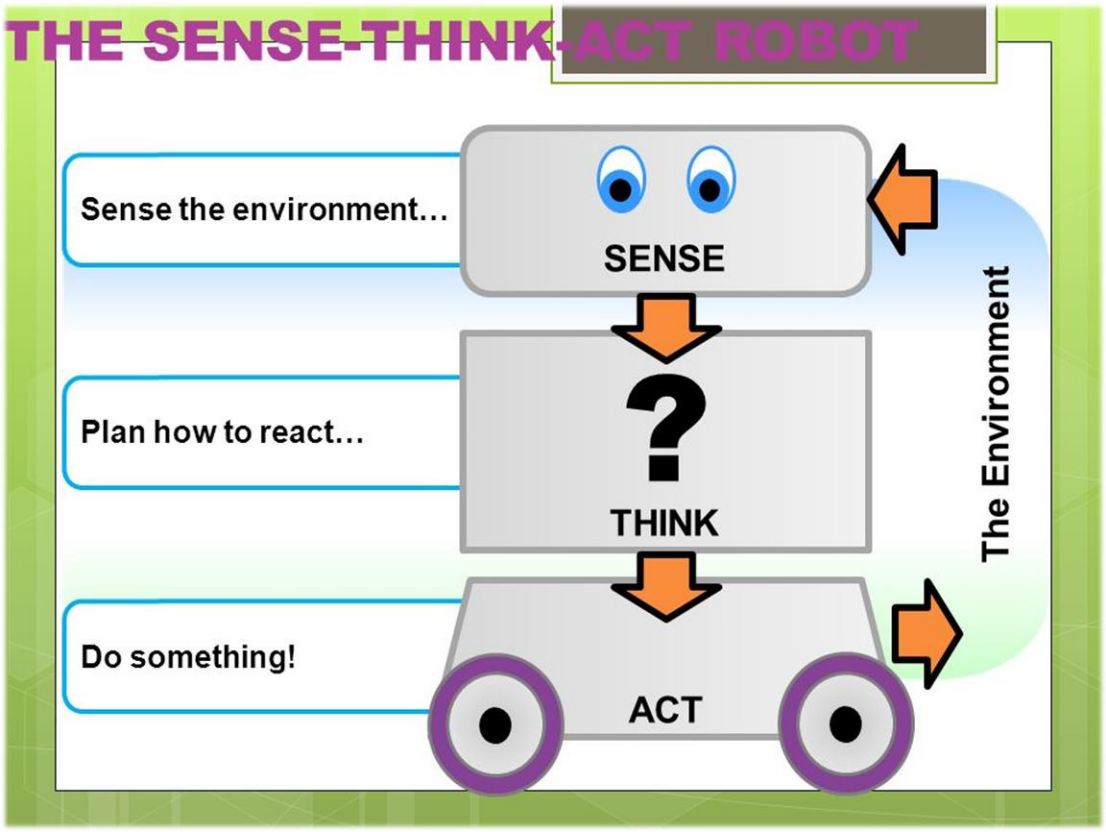
2- observe and assess the surroundings (perception).

3- determine its place/situation in the environment (localization)

4- make a movement strategy and carry it out (planning and motion control)

So, autonomous mobile robots are made up of four fundamental components, each with its own set of sensors, software, and functions.

These four blocks are known as the see-think-act cycle, and they are depicted in the diagram below:



SEE-THINK-ACT CYCLE

In the next sections, we'll go through each of these four blocks' steps in great depth.

Controlling Motion

The initial component of the see-think-act cycle is motion control, and the following three blocks are constructed on top of it.

The work of motion control is separated into two parts:

a- Instructed the robot to move to a certain location.

b- Use constant motion sensor feedback to adjust the robot's route and verify that the orders were successfully performed.

To carry out these two activities, we will require:

i) Create a kinematic motion model for the robot.

ii) Create a control model for the robot motion based on the kinematic model.

Kinematics

The most fundamental study of how mechanical systems act is kinematics. To build acceptable mobile robots for tasks and to understand how to construct control software for an instance of mobile robot hardware, we need to understand the mechanical behaviour of the robot in mobile robotics.

The idea of the robot's workspace, which is the range of possible poses that a mobile robot may attain in its surroundings, is present in mobile robotics. There is no direct means to measure a mobile robot's position in real time in mobile robotics.

Instead, the robot's movements must be integrated throughout time. When you include in the errors of motion estimates caused by slippage, it's apparent that properly determining the location of a mobile robot is a difficult undertaking.

Kinematics Models and Constraints

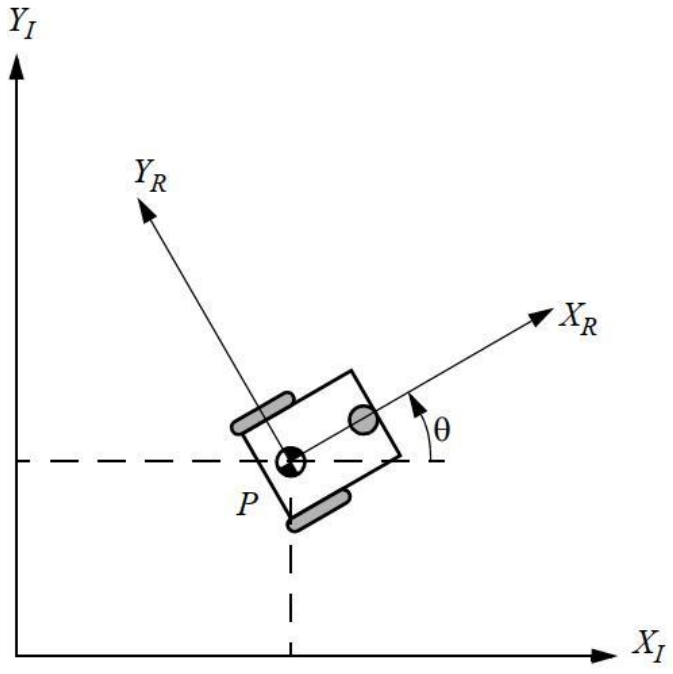
A bottom-up approach is used to create a model for the entire robot's movements. Each wheel contributes to the robot's motion while also imposing limitations on the robot's movement.

Because the geometry of the robot chassis binds the wheels together, their restrictions combine to produce constraints on the robot chassis' total motion. However, each wheel's forces and restrictions must be described in terms of a clear and consistent reference frame. Because of its self-contained and transportable character, this is especially crucial in mobile robots.

As a result, there must be a clear mapping between global and local frames of reference.

Representing the position of the robot

Because we're operating in a horizontal 2D plane, this robot chassis has three dimensions: two for position in the plane and one for orientation along the vertical axis, which is orthogonal to the plane.



The wheel axles, wheel steering joints, and wheel castor joints, of course, provide additional degrees of freedom and flexibility. Robot chassis, on the other hand, refers exclusively to the robot's rigid body, disregarding the robot's internal joints and degrees of freedom, as well as its wheels.

As a result, in order to describe the robot's location on the plane, we construct a link between the plane's global reference frame and the robot's local reference frame, as shown in figure.

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

As the global reference frame, the axes and establish an arbitrary inertial basis on the plane from some origin

O: XI, YI.

Choose a point P on the robot chassis as its position reference point to describe the robot's position. The robot's local reference frame is defined by the basis XR, YR, which specifies two axes relative to P on the robot chassis. The angular difference between the global and local reference frames is supplied by 0 (an angle read as Theta), and P's position in the global reference frame is described by coordinates x and y.

These three components can be used to define the robot's posture as a vector.

The subscript I denotes that we are referring to the global reference frame.

A matrix called the orthogonal rotation matrix can be used to map between multiple reference frames.

This matrix is defined as follows. It is a function of the current angle between the two frames (theta).

For example, using the equation below, this matrix may be used to translate motion in the global reference frame to motion in the local reference frame:

$$\zeta R = R(\theta) * \zeta I$$

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

let theta = pi /2

So, given (x, y, theta) in the global frame, we can use the rotational matrix to transfer these to the robot local frame.

$$R(\frac{\pi}{2}) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Forward Kinematics model:

In forward kinematics, we input a series of actuator data, such as encoder ticks, into a kinematic model equation to retrieve the robot's posture (x, y, theta).

In inverse kinematics, given the desired pose, determine the needed actuator speeds and joint values to achieve this position if at all possible!!

The differential drive concept will be used to decrease the robot's complexity, cost, and size. A differential-drive robot has two primary

wheels that are placed on a shared axis and are driven by independent motors. A nonholonomic system is one that includes limitations on posture change, such as a differential drive system or steering system. Because it cannot alter its position without changing its posture, a vehicle is an example of a nonholonomic system. Let's have a look at how our robot The posture of a mobile works and how we represent it mathematically robot or vehicle has six degrees of freedom (DOF) (x, y, z, roll, pitch, and yaw). It comprises of position (x, y, z) and attitude (a, b, c, d, e, f, g, h (roll, pitch, and yaw). Sidewise rotation is referred to as roll, forward and backward rotation is referred to as pitch, and yaw (also known as heading or orientation) is the direction in which the robot travels in the x-y plane. The differential-drive robot moves in the plane from x to y, thus the 2D posture consists mostly of x, y, and, where is the robot's head.

that points in the forward direction of the robot

The following problem may be solved using the forward kinematics equations for a robot having a differential-drive system:

Determine the posture (x', y',') at t + t using the control parameters V-left and V-right if the robot is standing in the position (x, y,) at time t.

Where V-left denotes the speed of the robot's left wheel, and V-right denotes the speed of the robot's right wheel (is the velocity of the robot right wheel). We'll look at how to calculate them in a later section.

This method may be used to a robot to allow it to follow a certain path.

Deriving forward kinematics equations:

Before deriving our model we will assume some assumptions:

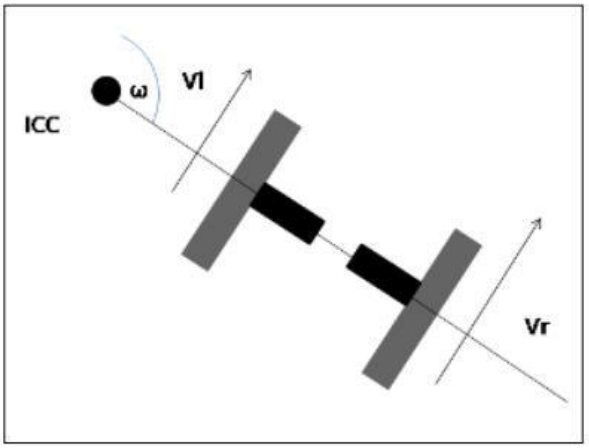
a- Movement on a horizontal plane

b- Point contact of the wheels

c- Wheels not deformable

d- Pure rolling, no slipping, skidding, or sliding

e- No friction for rotation around contact point



f- Wheels connected to a rigid frame (chassis)

When the robot is going to roll, it must spin around a point that is located between its common left and right wheel axis.

as **ICC- Instantaneous** Center of Curvature.

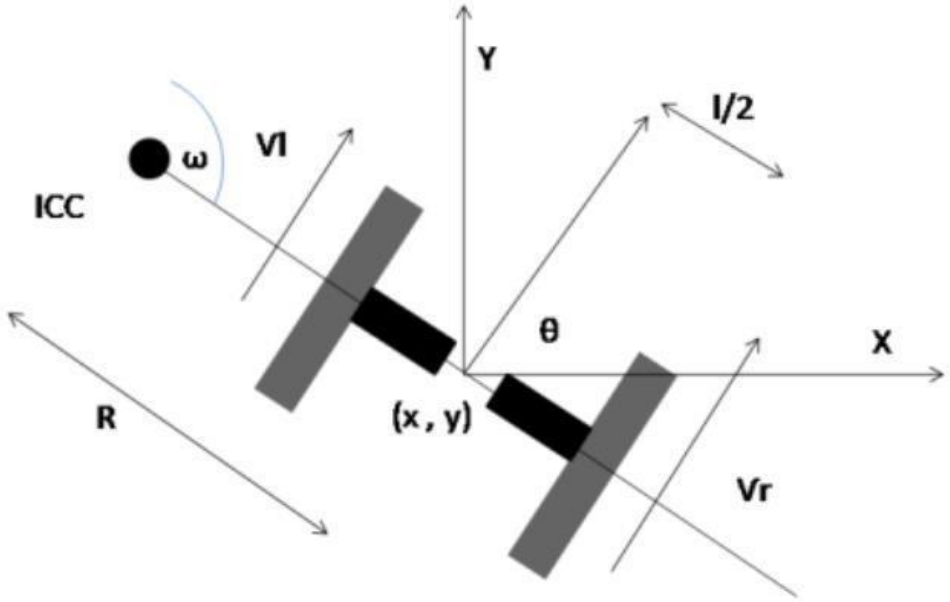
The following diagram shows the wheel

configuration of differential-drive with ICC:

The wheel's speed is v = 2r / T, where T is the amount of time it takes to perform one full round around ICC. The angular velocity is measured in radians (or degrees) per second and is defined as 2 / T. Putting the v and w equations together yields:

$$v = \omega r$$

A detailed model of the differential-drive system is shown in the following figure:



Where:

R: distance from ICC to the mid-point of the robot (center of the robot).

I: Width of the robot (distance between left and right wheels).

Vl, Vr: velocities of both left and right wheels.

W: angular velocity of the robot

Applying eq(1) on both wheels, the result will be :

$$\omega * \left(R + \frac{l}{2}\right) = Vr \quad (2)$$

$$\omega * \left(R - \frac{l}{2}\right) = Vl \quad (3)$$

After solving for we get the following results:

$$R = 1/2 * ((Vl + Vr) / (Vr - Vl)) \quad (4)$$

$$\omega = (Vr - Vl)/l \quad (5)$$

The previous two equations are useful for solving the forward kinematics problem. Suppose the robot moves with an

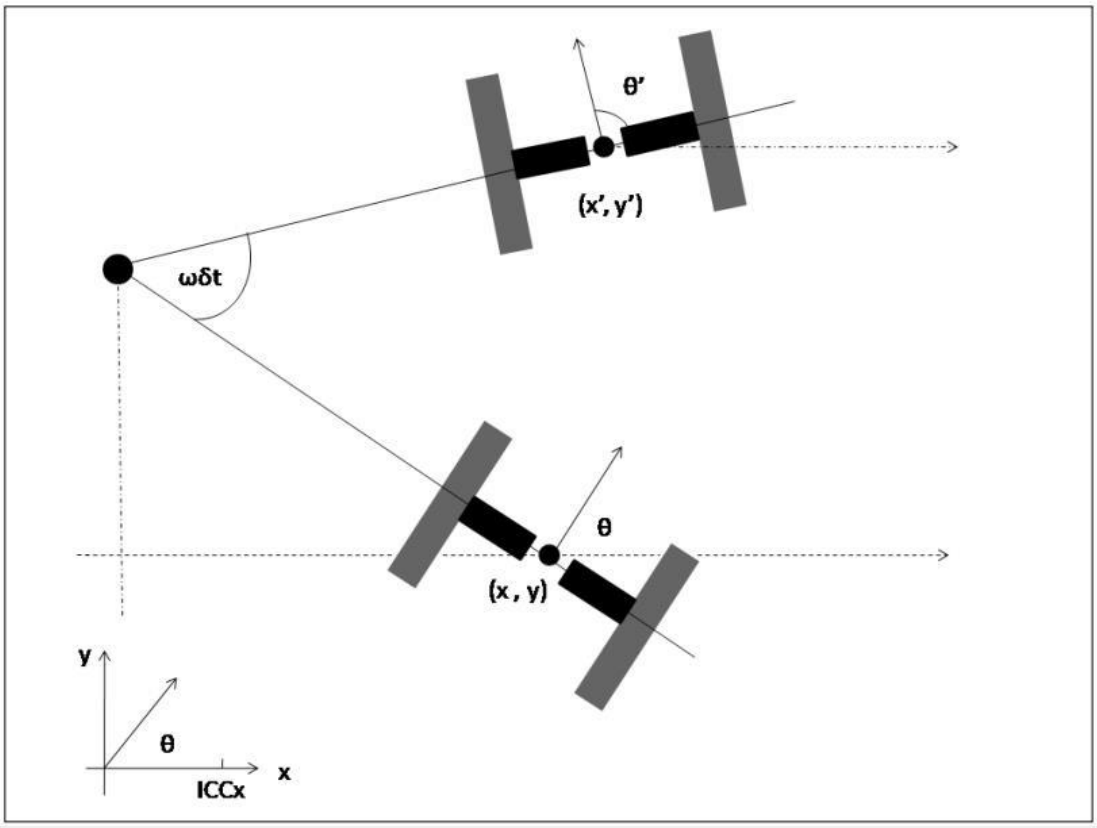
angular velocity of ω for δt seconds, it can change the robot's orientation or where it is

heading to:

$$\theta' = \theta + * \delta t \quad (6)$$

and ICC will be given as, using simple trigonometr

$$ICC = [ICCx , ICCy] = [x - R\sin\theta, y + R\cos\theta] \quad (7)$$



The main result is that using the 2D rotation matrix, the new location (x',y') can be calculated given a starting position (x, y) (explained in the previous pages). The following location at t + t time is obtained by rotating about ICC with angular velocity for t seconds:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cos(\omega\delta t) & -sin(\omega\delta t) \\ sin(\omega\delta t) & cos(\omega\delta t) \end{pmatrix} \begin{pmatrix} x - ICC_x \\ y - ICC_y \end{pmatrix} + \begin{pmatrix} ICC_x \\ ICC_y \end{pmatrix} \qquad (8)$$

The new pose (x‘, y‘, θ') can be computed from eq 6 and 8, given ω, δt, and R.

ω can be computed from equation (5); Vr and Vl are often difficult to measure accurately. Instead of measuring the

velocity, the rotation of each wheel can be measured using a sensor called wheel encoders. The data from the wheel

encoders is the robot's odometry values. These sensors are mounted on the wheel axes and deliver binary signals for

each step the wheel rotates (each step may be in the order of 0.1 mm). These signals are fed to a counter such that v *

δt is the distance travelled from time t to t + δt. We can write:

n * step = v * δt

From this, we can compute *v*:

V = n * step / δt (9)

If we insert equation (9) in equations (3) and (4), we get the following result:

$$R = (1/2 * (V_r + V_l)) / (V_r - V_l) \qquad (10)$$

$$\omega\delta t = (V_r - V_l)\,\delta t / l = (n_r - n_l) * step / l \qquad (11)$$

The encoder counts for the left and right wheels are nl and nr, respectively. The left and right wheels' speeds are Vl and Vr, respectively. Thus, during a time step t, the robot stands in pose (x, y, θ) and moves nl and nr counts; the new posture (x', y', θ ') is provided by:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x\text{-}ICC_x \\ y\text{-}ICC_y \\ \theta \end{pmatrix} + \begin{pmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{pmatrix} \qquad (12)$$

Where R and δt are computed from eq 10,11 respectively and ICC is computed from eq 7

IMPORTANT NOTE: The resulting kinematic equation is heavily influenced by the robot's design and shape. Different designs can result in a variety of equations.

Inverse Kinematics

The forward kinematics equation provides an updated pose at a given wheel speed. We can now think about the inverse problem.

Stand in pose (x, y, θ) at time t and determine the V-left and V-right control parameters such that the pose at time t +

δt is (x', y', θ').

In differential-drive, this problem may not have a solution because this kind of robot can't be moved to any pose by

simply setting the wheel velocity. This is due to nonholonomic constraints.

In nonholonomic robots, there are some ways to increase the constrained mobility if we allow a different sequence (Vleft, V-right). If we insert values from equations 7,10,11,12 , we can identify some special cases of control:

1- If V-right = V-left => nr = nl => R = ∞, ωδT = 0 =>: This means the robot moves in a straight line and θ remains the same.

2- If V-right = -V-left => nr = -nl => R=0, ωδt = 2nl * step / l and

ICC = [ICCx , ICCy] = [x, y] => x' = x, y' = y, θ' = θ + ωδt =>: This means the robot rotates in the position around ICC, that is, any θ is reachable, while (x, y) remains unchanged.

The Result of the above two conclusions:

Combining these operations, the following algorithm can be used to reach any target pose from the starting pose:

1- Rotate until the robot's orientation coincides with the line from the starting position to the target position, V-right = -V-left = V-rot.

2- Drive straight until the robot's position coincides with the target position, V-right = V-left = V-ahead.

3- Rotate until the robot's orientation coincides with the target orientation,

V-right = -V-left = V-rot.. where, V-rot and V-ahead can be chosen arbitrarily

Why do we need control in the first place?!

a- A kinematic controller's goal is to follow a trajectory defined by position and/or velocity profiles as a function of time.

b- Because mobile robots are generally nonholonomic and MIMO systems, motion control is not straightforward.

c- The majority of controllers (like the one shown here) ignore the system's dynamics.re, V-rot and V-ahead can be chosen arbitrarily.

PID (Proportional Integral Derivative) Controller

PID Controller is a programming technique that, when correctly tuned, may be extremely effective and precise. PID stands for Proportional Integral Derivative, which is made up of three independent elements that are sometimes not required. Instead of P control, PI control, or PD control, you may have only P control, PI control, or PD control.

A bit of history...
Governor design in the 1890s inspired PID controllers. PID controllers were later used in ship steering systems. Elmer Sperry created one of the first PID-type controllers in 1911, and Russian American engineer Nicolas Minorsky published the first PID controller theoretical study. Minorsky was designing automatic steering systems for the US Navy, and he based his analysis on observations of a helmsman, who observed that the helmsman controlled the ship not only based on the current error (distance/value remaining), but also on past error and current rate of change; Minorsky then turned this into a mathematical equation.

His objective was stability rather than comprehensive control, which made the problem much easier to solve. While proportional control provided stability in the face of minor disturbances, it was insufficient in the face of a continuous disturbance, such as a severe gale (owing to droop), necessitating the addition of the integral term. Finally, to increase control, a derivative term was introduced.

P – Proportional

Consider a robot that moves at top speed for a value reading of 1000 on the sensor. Now, due to its speed and inertia, it will most likely overshoot and move over 1000 on the sensor. When developing a programme, this may be a big pain because you want to be as precise as possible. In an ideal world, you could instruct the robot to halt and it would stop just where it was.

However, we do not live in a perfect environment, and if we instruct it to halt immediately, it will overshoot. It will overshoot if you instruct it to stop quickly.

Now, if the distance it overshot was always the same, this overshoot wouldn't be a problem.

However, there are lot's of variables that can change the distance it overshoots. Here are a few of the variables:

- The voltage of the battery. The motors will not operate as fast if the battery is low, and the robot will have less inertia. The robot will overshoot less in this situation.
- If the robot collides with something, the overshoot decreases.
- The overshoot will increase if anything pulls the robot in the direction it wants to go.

As you can see, overshoot is a bad thing. As a result, the P controller smoothly regulates the speed, enabling it to slow down as it reaches its objective, reducing overrun. That's why it's called a proportional controller:

the output speed is proportionate to the value that still has to be altered, which we refer to as an error, where error = (target value –current value).

However, P-controller alone is not always enough to do the task as it has some problems:

1- It cannot get rid of steady state error , you will always have some error so the controller can work.
2- The controller is proportional to error value, so if error is high => there will be a bigger overshoot and vice versa
As a result in most applications we need some extra control, this can be achieved using I and D controllers

I – Integral

As a result of the proportionate part of the code, the mistake that remains is rather tiny. The proportionate part is too tiny to make a significant effect. This is when the integral enters the picture. The running total of past mistakes is the integral. So, when your mistake is really little, the integral kicks in, but how exactly does that work?

The operator wants it to go at a rapid enough speed to reduce the mistake, but not too quickly that it overshoots. It will gradually accelerate as it chooses how quickly to travel. This is how you compute the integral:

integral = integral + error*dT;
dt here the time between two continuous sample i.e the sampling rate .
problems with the integral

the first issue

Your integral will most likely still be at a value that keeps the speed high enough to keep the error moving until your error eventually approaches zero. Because the equation will only reach 0 if it passes over a 0 error, the negative error can be subtracted from the existing integral. So, we have an issue if the speed is still high enough to keep the error changing, right? If the error approaches 0, there is a very simple solution to this problem: reset the integral. This may be accomplished in the following manner. The fix is highlighted in bold language in the code.

```
if (error is 0)
{
integral = 0;
}
```

Problem 2: It's referred to as integrated winding-up. If you have a big error to traverse, the integral will begin to build up as soon as the loop is started. As a result, by the time the integral is needed, it has already reached a value much beyond what may be utilised.

This problem has easy solutions, two of which I shall discuss.

1- Limit the value of the integral

Limit the value that the integral can reach. If it is reaching too high, why not just put a limit on it? A limit could be written as follows:

if (integral is greater than or equal to the maximum value)

{

integral = maximum value;

}

But, if the integral is too big but in it's negative form (I. E. making the speed reverse too fast), you would need to

rewrite the same as above but for the negative version of the integral.

2- Remove pid windup and make controller output maximum

/// integral wind up removal

if(control_signal > max_val){

control_signal = max_val;

integral = integral - (error * ki);

}

if(control_signal < min_val){

control_signal = min_val ;

integral = integral - (error * ki);

}

D - derivative.

The derivative is the last piece of the PID code. The derivative's job is to forecast the error's future value and then have the speed react accordingly. If it believes it may overshoot, for example, it will slow down.

derivative = d error / dT

= ((current error) – (previous error)) / dT

So the overall control signal is calculated using the following eq:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt}e(t)$$

For full PID implementation code see our code in this file:

adjusting the constants

This is the one that takes the most time and effort. There are several ways for adjusting the Kp, Ki, and Kd; I will do my best to explain the one we used in our project. A computer programme, arithmetic computations, or hand tuning can all be used to tune the PID constants. I strongly advise you to keep an eye on all of the error, speed, and other numbers at all times so you can see how near you are to the target point/how much work remains. Check the results using a debugger or another monitoring tool.

Some concepts you need to understand them very well for better tuning:

Rise time – the time it takes to get from the beginning point to the target point for the first time.

Overshoot – the amount that is changed too much; the value further than the error.

Settling time – the time it takes to settle back down when encountering a change i.e the time the output takes to be in

the range of 95 – 98 of the final value.

Steady-state error – the error at the equilibrium

Stability – the —smoothness‖ of the speed

What happens when each of the constants is increased:

Parameter Increase	Rise time	Overshoot	Settling Time	Steady-state error
Kp	↓	↑	Small Change	↓
Ki	↓	↑	↑	Great reduce
Kd	Small Change	↓	↓	Small Change

Table (2) PID controller parameter characteristics on a fan's response

PID Tuning (manual tuning):

The way I tune my constants is as follows:

1. Set Kp, Ki, and Kd to 0. This will disable them for now.

2. Increase Kp until the error is fairly small, but it still gets from the beginning to nearly the end quickly enough.

3. Increase Kd until any overshoot you may have is fairly minimal. But be careful with Kd – too much will make it overshoot.

4. Increase Ki until any error that is still existing is eliminated. Start with a really small number for Ki, don't be surprised if it is as small as 0.0001 or even smaller.

5. Using the rules of tuning the constants (in the table on the previous page), you can change around the constants a little bit to get it working to the best performance.

Knowing how big/small each constant term should be gets easier with practice, you will get a rough idea for it after practicing a few times.

There is a second and good method for tuning which is Ziegler-Nichols method we do not use it in our project but I will explain it if you want to use it in any project.

In this method, the values for the constant terms are mainly found out by math, but you will almost definitely need to tweak them a bit afterwards, to get it all working perfectly. The first bit of this method is done manually, but after you have done a few tests it is on with the math.

1. Set Kp, Ki and Kd to 0. This will disable them for now.

2. Start increasing Kp until there is some oscillation in the error. Not too much oscillation, just a noticeable amount.
Save this value for Kp as —Ku‖ - the value for the ultimate or critical gain.
3. You now need to measure the period of the oscillation. Save the value for the period as —Pu‖ - the period at the ultimate or critical gain.
4. Now we have done the most of the manual part of this method, so we will use the math of finding Kp, Ki and Kd!

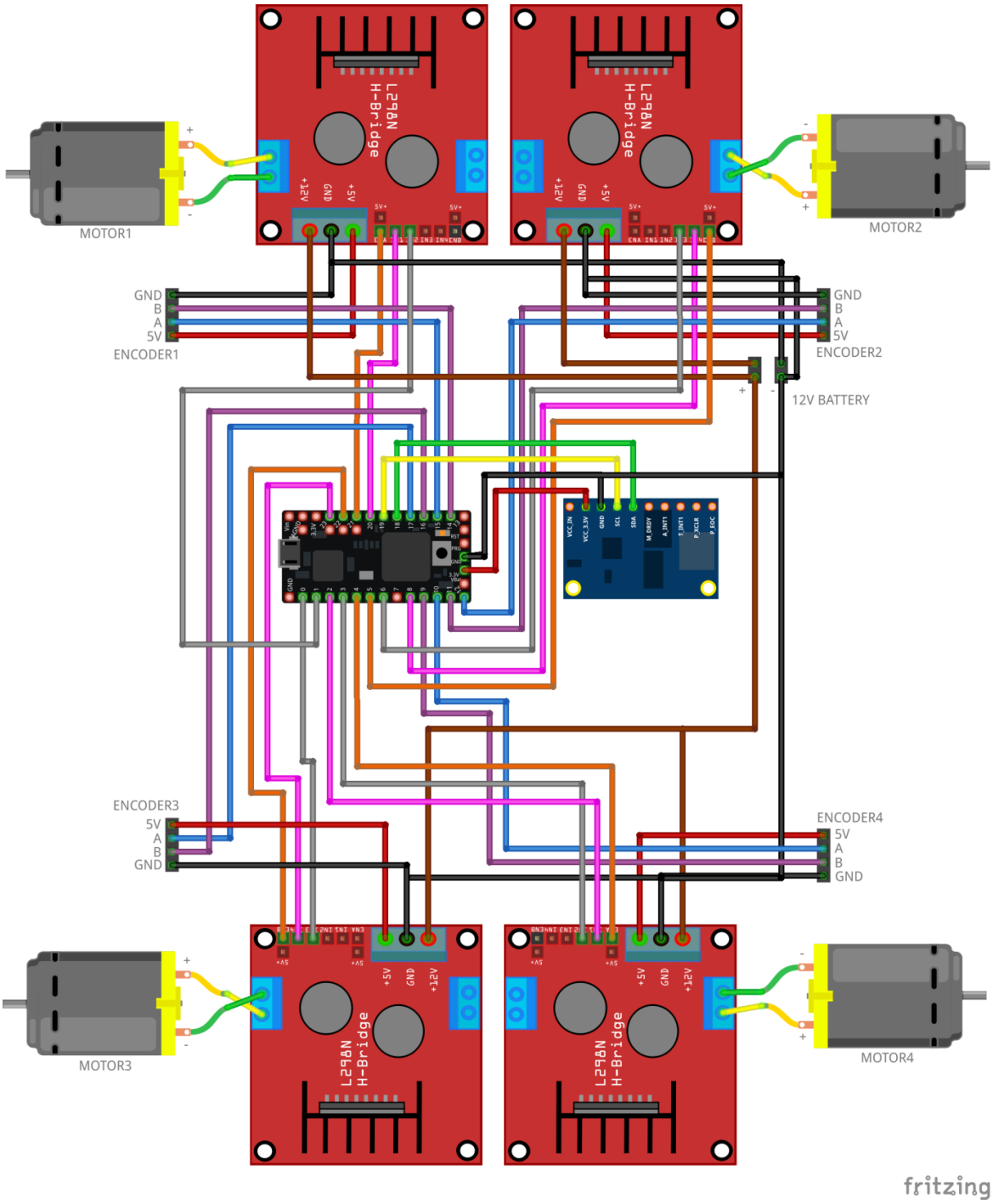
There are 3 —constroller types‖. P is a simple proportional controller, PI is proportional with integral, PID is the full
proportional integral derivative code. There's no PD (proportional with derivative) in the table, but that doesn't say it
doesn't exist. Here is the table for finding Kp, Ki and Kd:
Ziegler-Nichols Method for finding Kp, Ki and Kd:

Controller type:	Kp:	Ki:	Kd:
P	0.50Ku	N/A	N/A
PI	0.45Ku	1.2Kp/Pu	N/A
PID	0.60Ku	2Kp/Pu	KpPu/8

As I previously stated, you will most likely want to change and tweak the numbers for Kp, Ki, and Kd after the computations because they may not be ideal for your situation. To assist you, use the rules for locating constant phrases.

Wiring and Coding

In our robot the wiring will be the same as in 4WD or Mecanum Drive wiring, as in this figure:



Except that we will be taking in consideration only “Two” Encoders reading, one from each side (Left and right) instead of “Four” as we will be driving it as a “2WD” robot, and to be able to do that we have to calibrate the Front and Rear Motors at each side to move at the same speed, which we will discuss later on.

The Encoder

Not all motors have built-in encoders especially high torque motors. Shall you need to use an external encoder, you can use a bi-shaft motor to leave the spare shaft for the encoder or use a pulley to drive both the wheels and the encoder.

How it will be used in our case?

The Encoder we use in our case has 6 Pins:

- Two pins for passing the voltage to the motor “Motor-, Motor+” connected to the motor driver output.
- Two pins for powering the encoder “Vcc & Gnd”.
- Two pins for the Encoder Output “Encoder Output A, Encoder Output B”, that will be connected to our controller giving it the Encoder Readings.

Once you connect these pins as mentioned you can have your Encoder readings, but there is only one problem and it is that both Encoder Outputs only gives output of “0 & 1” alternately when the motor is moving (by external force or by powering it), so how will you be able to use it, and this is what we are going to discuss here.

Our Encoders works by counting in a binary 2-bit number, where:

- Encoder Output A –> Most Significant Bit (MSB).
- Encoder Output B –> Least Significant Bit (LSB).

As follows:

En O/P (A)	En O/P (B)	Then	En O/P (A)	En O/P (B)	Then	En O/P (A)	En O/P (B)	Then	En O/P (A)	En O/P (B)
0	0		0	1		1	0		1	1

And this is the pattern “Counting Up in Binary” it follows when the motor is moving **Forward** and once it reach the maximum value it will start from the beginning and so on.

Also, if our Motor is moving **Backwards** this will be the pattern “Counting Down in Binary” that it will follow and repeat once reaching its last value:

En O/P (A)	En O/P (B)	Then	En O/P (A)	En O/P (B)	Then	En O/P (A)	En O/P (B)	Then	En O/P (A)	En O/P (B)
1	1		1	0		0	1		0	0

Both of these patterns repetition frequency will depend on some factors:

- The Encoder Pulses Per Revolution (PPR), and This is actually a number that indicates how many times the encoder counts per revolution of one motor revolution and this number can be practically indicated through the encoder datasheet, and it may be also called the “Encoder Resolution”.
- The Motor Speed, as it increases the frequency of counting also increases.

- The Motor-to-Gear Ratio, since that the Encoder has a specific PPR the and it is installed on the side of the motor with no gear this means that for every revolution of the geared side the shaft connected To the Encoder side will revolute more than once according to this ratio resulting in an increasing frequency.

From all what is mentioned above we can calculate how many times the encoder counts per Revolution “Counts Per Revolution (CPR)” as follows:

CPR = PPR * Motor-to-Gear Ratio

But until now we didn’t perform the encoder main function which is counting, thus we will have to build a Code Function that maps the changes in EnA “Encoder Output A” and EnB “Encoder Output B” into a Variable that counts Up on going Forward and Down on going Backward, and this can be done using the simple Code that was built by me in the following Repository (Link) under the name of “SimpleEncoderCode”, which:

- Declares Arduino Pins that will take the Encoder output.
- Set them as Read (INPUT) pins, and activates the Interrupts.
- Once the Encoder Output value changes it will start an Interrupt Calling a Function that will change the value of the Variable that we use to count “EncoderPosition”.
- Printing the EncoderPosition on the Serial after some time delay.

Note that in this example you will spin the motor manually as we will discuss how to drive it with the controller later on.

From what is written above we can indicate the how many Revolutions that the Motor has done and by multiplying it by the Wheel Circumference, we can indicate the Distance cut by the Robot:

Number of Revolutions = Encoder Position / CPR

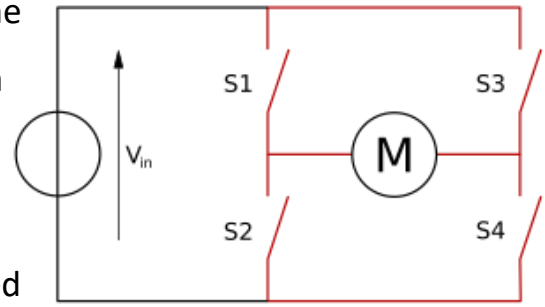
Distance = Number of Revolutions * Wheel Circumference

= Number of Revolutions * 2 * PI * Wheel Radius “R”

The Motor Driver

In our case we can work on one of the two Motor drivers that were mentioned previously the L298, which is known as “H-Bridge” or the Four Channel One.

The H-Bridge



- The term H-bridge is derived from the typical graphical representation of such a circuit, which is shown here.
- An H-bridge is built with four switches.
- When the switches S1 and S4 are closed (and S2 and S3 are open) a positive voltage is applied across the motor.
- By opening S1 and S4 switches and closing S2 and S3 switches, this voltage is reversed, allowing reverse operation of the motor.

And this is mainly how the motor driver works and in our case these switches can be either opened or closed using our controller “Arduino Mega2560”.

S1	S2	S3	S4	Result
1	0	0	1	Motor moves right
0	1	1	0	Motor moves left
0	0	0	0	Motor coasts
1	0	0	0	
0	1	0	0	
0	0	1	0	
0	0	0	1	
0	1	0	1	Motor brakes
1	0	1	0	
x	x	1	1	Short circuit
1	1	x	x	

In case of using the **L298 Controller**, we will have Three main parameters, which are:

- **PWM Signal**: this is a signal sent by the controller to the driver to adjust the Voltage value going to the Motor, is called “Enable Signal” within L298.

- **Input 1**: which is mainly the signal controlling the switches **S1** and **S4** for the **Forward** current path.
- **Input 2**: which is mainly the signal controlling the switches **S3** and **S2** for the **Reverse** current path.

Thus the controller will have to deal with 3 Variables to Control Motor Direction and Speed, as follows:

ENA	IN1	IN2	Description
0	x	x	Motor A is off.
1	0	0	Motor breaks and stops
1	0	1	Motor turns forward
1	1	0	Motor turns backward
1	1	1	Motor breaks and stops

And now since we know how it works we can create our piece of code to drive it after connecting as previously mentioned in this chapter, you can use a simple Code that was built by me in the following Repository (Link) under the name of "MotorDriving1", which:

- Declares Arduino Pins that will take the control the Motor, note that one of them should be a pin that can output PWM signal, and you can find more information about PWM through this Link.
- Set them as Write (OUTPUT) pins.
- Set the Motor on forward and reverse its direction after some time delay (10ms) using within the Void Loop part.

And in case that you are using the “RO.RS011MC” 4 Channel Driver, it will have the same operating principal but instead of controlling the Motor with 3 Signals it only uses Two, which are:

- PWM Signal: that controls the motor speed.
- Direction Signal: which controls the movement direction, either Forward (for LOW Logic Signal) or Backward (for HIGH Logic Signal).

PWM	DIR	M
0	Any	Stop
0 < PWM ≤ 255	LOW	Moves Backward
	HIGH	Moves Forward

And here is a simple code works just as the one with L298 but using this driver instead in the Repository (Link) under the name of “MotorDriving2”

Issues and Testing

Note that: Any Codes within this section will depend on that you are using L298 Motor Driver not The 4 Channel One.

The 1st Main Issue that you may encounter is being unable to calculate the Encoder CPR, and this can be done in two main steps:

1st Step: Determining the **Initial CPR** value **Manually**, and this can be done as follows:

In this step you can use a code similar to the "SimpleEncoderCode", which was mentioned before. Since I am using 4 motors it will be harder to go one by one, and since that "Arduino Mega2560" has 4 interrupt pins, I will test two (since each Encoder requires 2 interrupt bins) at a time, as follows:

- Connect the encoder to the motor and the controller.
- Spin the two motors wheels manually a specific number of Turns "N".
- Record the Encoder Reading appearing on the Serial Monitor for Both motors.
- Repeat the previous two steps for a specific number of times (10 is good) and alternating between spinning in the both directions (Forward and Backward).
- Do these steps also for the other two motors.

After doing these steps you can calculate the **CPR Initial Value** as follows:

- Take the average of the resulting Number of Counts.
- Divide it by the Number of Revolutions you have done.

Initial CPR = Number of Counts / Number of Revolutions

You can also calculate the **Number of Counts Per Meter**, as follows:

Travelled Distance = Number of Revolutions * Wheel Circumference

= Number of Revolutions * 2 * PI * R

Number of Counts Per Meter = Number of Counts / Travelled Distance

And these were **my calculations**:

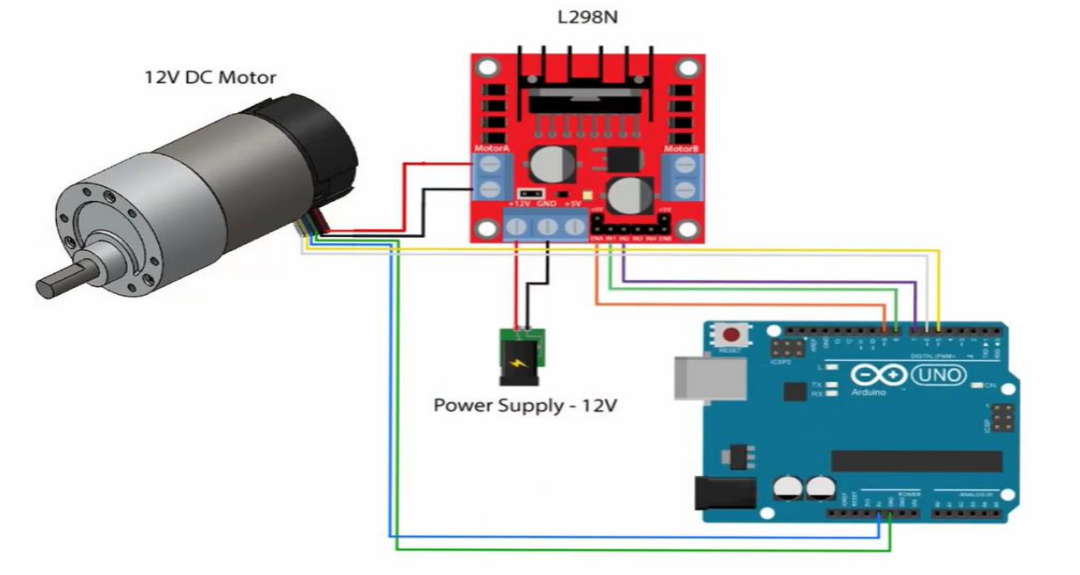
Spinning Direction	Ml1	Ml2	Mr1	Mr2
Forward	2509	2532	2513	2516
Backward	2515	2534	2510	2501
Forward	2519	2509	2513	2511
Backward	2518	2518	2517	2503
Forward	2516	2526	2521	2517
Backward	2511	2526	2509	2505
Forward	2513	2509	2515	2514
Backward	2523	2518	2519	2503
Forward	2512	2519	2515	2500
Backward	2510	2510	2514	2500
Average	2514.6	2520.1	2514.6	2507
Calculations	Average			2514
	Number Of Revolutions			3.5
	Wheel Circumference			0.377
	Counts Per Revolution			718
	Trevelled Disttancd			1.32 m
	Counts Per Meter			1904

You can find the code used in this step in the Repository (Link) under the name of "CPR_Initial".

2nd Step: Determining the **Actual CPR** value:

On operating the motor the encoder may be affected by its magnetic field altering the CPR value, or the rate of change may be faster than what can be detected by our controller, or there may be other sources of errors thus we should operate the motor and detect the how it counts, as follows:

- At the beginning you should connect the circuit as shown:



- Then adjust the previous code to drive the motor by making the "SetMotor Function" which maps between Arduino control signals and the Driver for all possible cases you may use.
- Adjust your loop to run the code for a specific number of Counts using the number that we have got from the previous step, you can make it go for 10 Revolutions and see the result.
- Increase or decrease the Number of Counts until the Motor do the specific Number of Revolutions that you adjusted it to do.

- The Actual CPR can be evaluated as follows:

  Actual CPR = Number of Counts / Number of Revolutions

- Repeat these steps for the other three motors.

Since we will be only taking in consideration the Readings of only 2 Encoders One on the left and the other on the Right we will choose the 2 Encoders with the nearest CPR as possible on the both sides.

You can find the code used in this step in the Repository (Link) under the name of "Actual_Initial".

The 2nd Issue that we encountered is that we are only getting Reading from 2 of the Encoders while the other Two Motors have none, this means we can only control the speed of only 2 of the Motors and if the Motors' parameters differed slightly, this may lead to that the Front wheels and the Rear ones will have different speeds and this can be Solved by alternating the PWM written to both of the motors at the same side, instead of being the same, one may differs slightly than the other, and this can be done as follows:

- Connect two motors at the same side as before but only use the One Selected Encoder.
- Adjust your code to drive both of the motors using a given PWM value for each motor.
- Drive both of the motors for some specific amount of time that you determine.
- Mark the wheels and observe how many revolution each one does.
- If there is no difference then both of them can be driven with the same PWM Signal values and operate at the same speed.

- If their speed differs try to increase or decrease both of their PWM values until their speed match.
- Repeat these steps for the other side.

You can find the code used in this step in the Repository (Link) under the name of "MotorSpeedEqualizing".

After Solving these issues we can write or Main Code and insert in it some functions that will be used to communicate with ROS which will be discussed later on this book.

The Main Code:

You can combine all the parts from before to get into the final code But you will need to customize it a lot:

- You will need to add ROS communication into the code and import its Libraries.
- You will need to Perform Forward Kinematics on the sent velocities from the ROS.
- You will need to add PID Control Code to automatically adjust the Motors speed.
- You will need to Integrate the functions that:
    - Calculate the Encoder Counts.
    - Set the motor Speed.

And this code can be found in this Repository (Link) under the name of "ROS_Robot", and

under the name of "Serial_Robot" you will find my Code that is customized to drive the robot using the Serial Panel.

Chapter (6): Communication between ROS and a micro controller

Some recent questions have shown an interest in how ROS can be used to run a motor or read a sensor on a custom robot. Many hobby robots have one or more microcontrollers that interface with the robot hardware. These microcontrollers should be able to communicate using a serial connection to a robot mounted laptop running ROS.

The avr_bridge is a sophisticated approach, but it requires a larger AVR processor and C++ code development on the microcontroller which may not be available in some circumstances. Is there a simpler alternative?

A simple approach is to write a ROS node that can read/write text messages exchanged with the microcontroller over a serial port and convert these serial messages to ROS style std_msgs/String messages. A separate ROS node can then be written to convert the simple text messages to specific ROS message formats (cmd_vel, odom, etc) based on whatever message protocol was defined for the exchange with the microcontroller.

Communication between ros and micro controller graph:

Using AVR microcontroller like (ATmega32/16, etc) is very difficult because of the lack of packages and it difficult to write a very big code and the complicated hardware connection.

Instead of AVR we use the Arduino to make it easy and there are more available packages.

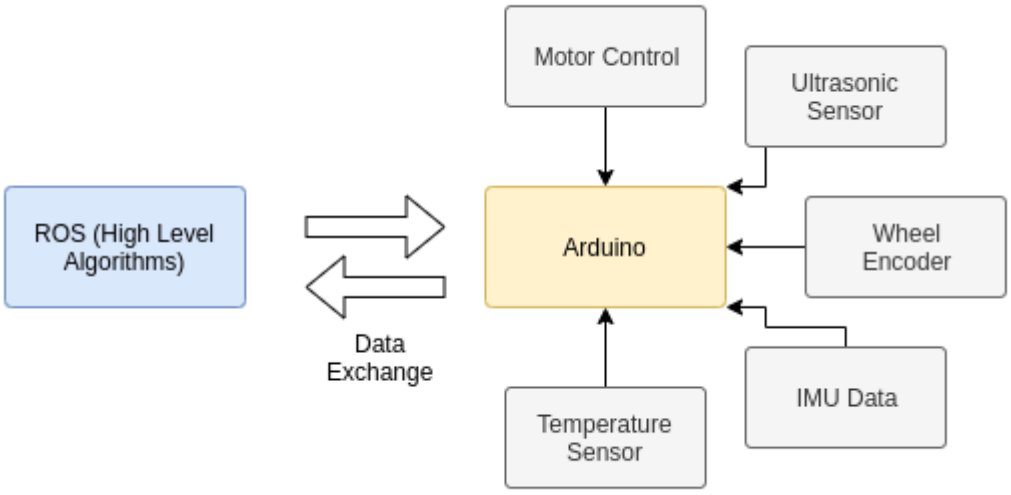
This is an example to show how large the code is to just publish hello world in ros using AVR:

http://wiki.ros.org/rosserial_client/Tutorials/Using%20rosserial%20with%20AVR%20and%20UART

Using Arduino with ROS

Probably the most used open-source microcontroller development board, Arduino offers an easy-to-use software-hardware interface and an ever-expanding user community. Arduino is mostly compatible with all digital and analog circuitry as well as external devices supporting serial communication interfaces like SPI, I2C, and UART.

ROS-based software frameworks run the high-level algorithms and processes like robot models' inference, filtering, motion strategy generation, controller algorithms, and more. An Arduino controller mounted on a robot is used to perform the low-level/embedded control and sensing. It includes any or all of motor control, temperature and ultrasonic sensors, IMUs and wheel encoders, and other devices.



To communicate between ROS and Arduino we are using publisher/subscriber tutorial.

The **rosserial** ROS package uses Arduino's universal asynchronous receiver/transmitter (UART) communication and converts the board to a ROS node that can publish ROS messages and subscribe to messages as well. The Arduino ROS node publisher can send data (from sensors or robot state) from the board to the machine running ROS while an Arduino ROS node subscriber can get instructions from the machine. The ros_lib Arduino library enables the Arduino board to communicate with ROS.

Setting Up the Infrastructure

Software Setup

1. Install ROS on the machine (laptop/PC), Elaborate steps available at:

   ROS-Wiki

   Note that we are using Ubuntu 16.04.3 and ROS Kinetic distribution.

2. Install the Arduino IDE on the machine from instructions mentioned here.

   Note that this method to install Arduino IDE has some issues that every time you open your ubuntu must go to the Arduino ide directory and open terminal and write **./arduino**.

   There is another method to install Arduino IDE using snap:

```
sudo apt install snapd
sudo snap install arduino
```

   In order to upload code an arduino board over USB, you need to add your user to the dailout group and connect the snap to the raw-usb socket.

   Open a terminal window, run the following commands an reboot your computer.

```
sudo usermod -a -G dialout $USER
sudo snap connect arduino:raw_usb
```

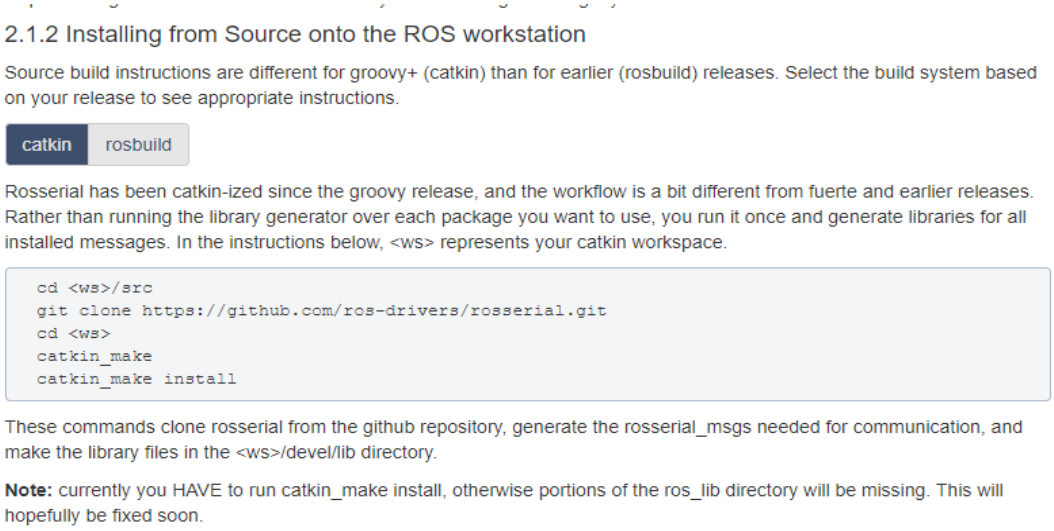
Now restart your computer and you're good to go!

That's it!

3. Install rosserial on the machine (distro used is kinetic)

http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup

Note that in this step



2.1.2 Installing from Source onto the ROS workstation

Source build instructions are different for groovy+ (catkin) than for earlier (rosbuild) releases. Select the build system based on your release to see appropriate instructions.

catkin rosbuild

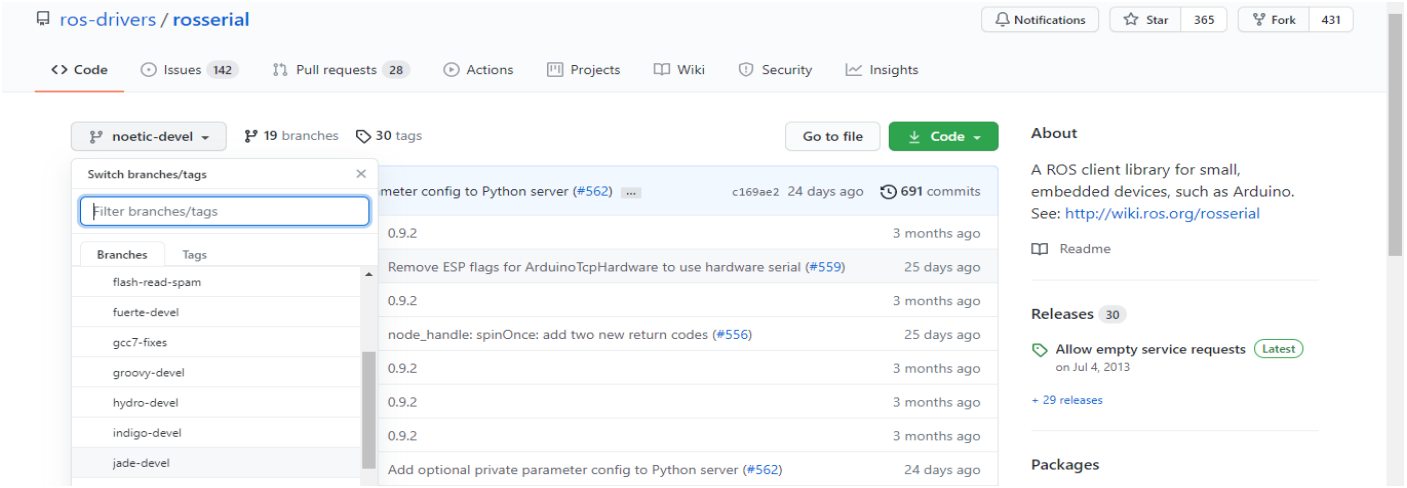
Rosserial has been catkin-ized since the groovy release, and the workflow is a bit different from fuerte and earlier releases. Rather than running the library generator over each package you want to use, you run it once and generate libraries for all installed messages. In the instructions below, <ws> represents your catkin workspace.

```
cd <ws>/src
git clone https://github.com/ros-drivers/rosserial.git
cd <ws>
catkin_make
catkin_make install
```

These commands clone rosserial from the github repository, generate the rosserial_msgs needed for communication, and make the library files in the <ws>/devel/lib directory.

Note: currently you HAVE to run catkin_make install, otherwise portions of the ros_lib directory will be missing. This will hopefully be fixed soon.

Don't use git clone, just open the git web site https://github.com/ros-drivers/rosserial/tree/jade-devel and change the branch from noetic-devel to jade-devel because we are working on kinetic distro not noetic.



To ensure that everything is installed successfully try this simple example in the ros tutorial.

http://wiki.ros.org/rosserial_arduino/Tutorials/Hello%20World

Note:

1- when uploading code to Arduino make sure that the baud rate in the code = 57600, because the ros work on this baud rate

The baud rate is the rate at which information is transferred in a communication channel. Baud rate is commonly used when discussing electronics that use serial communication. In the serial port context, "9600 baud" means that the serial port is capable of transferring a maximum of 9600 bits per second.

2- This cmd will show you the port you are using

```
ls -| /dev/ttyACM*
```

3- Note when using Arduino mega you must mod the port using this cmd

```
Sudo chmod a+rw /dev/ttyACM0
```

Replace 0 in ACM0 if you use another port (ACM1, etc)

chmod is a utility that modifies privileges. What this command is doing is changing privileges for /dev/ttyACM0, the port we're trying to write to the arduino with. We're running chmod a+rw. The "a" signifies that we're in a-

mode, which modifies all users, and the "+rw" means we "+" (add) "rw" (read and write permissions).

After we have installed the ros and rosserial and Arduino and try some examples:

Lets try to write a code to calculate the odometry from encoder (motor ticks) data and MPU sensor this is a code in c++ :

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/Vector3.h>
#include <stdio.h>
#include <cmath>
```

This part just include headers to be used

1- #include **<ros/ros.h>** ros library

ros/ros.h is a convenience include that includes all the headers necessary to use the most common public pieces of the ROS system.

2- #include **<tf/transform_broadcaster.h>**

This class provides an easy way to publish coordinate frame transform information. It will handle all the messaging and stuffing of messages. And the function prototypes lay out all the necessary data needed for each message.

3- #include **<nav_msgs/Odometry.h>**

This will represent an estimate of a position in free space.

The pose in this message should be specified in the coordinate frame given by header.fra

me_id.

The twist in this message should be specified in the coordinate frame given by the child_f rame_id

Header header

string child_frame_id

geometry_msgs/PoseWithCovariance pose

geometry_msgs/TwistWithCovariance twist

4- #include <geometry_msgs/Vector3.h>

This represents a vector in free space.

It is only meant to represent a direction. Therefore, it does not make sense to apply a translation to it (e.g. , when applying generic rigid transformation to Vector3, tf2 will only apply the rotation).

If you want your data to be translatable too, use the geometry_msgs/Point message i nstead.

float64 x

float64 y

float64 z

```
double radius = 0.06;                    //Wheel radius, in m
double wheelbase = 0.38;                  //Wheelbase, in m
double two_pi = 6.28319;
double speed_act_left = 0.0;
double speed_act_right = 0.0;
double speed_req1 = 0.0;
double speed_req2 = 0.0;
double speed_dt = 0.0;
double x_pos = 0.0;
double y_pos = 0.0;
```

Defining variables like wheel radius and height

```
ros::Time current_time;
ros::Time speed_time(0.0);
```

I take an object from ros::time named current_time and speed_time

There are two variable to have the same current time because later we will change the “current_time” variable to be the odom_msg.header.stamp. and the “speed_time ” will be the temp to save real current time or the last time.

```
void handle_speed( const geometry_msgs::Vector3Stamped& speed) {
  speed_act_left = trunc(speed.vector.x*100)/100;
  ROS_INFO("speed left : %f", speed_act_left);
  speed_act_right = trunc(speed.vector.y*100)/100;
  ROS_INFO("speed right : %f", speed_act_right);
  speed_dt = speed.vector.z; // angular vilocity
  speed_time = speed.header.stamp;
```

Handle_speed is a callback function it returns void and must take const and msg type I will use geometry msg vector (to pass the speed of motor) and msg name is speed.

Trunc is function to rounds the argument towards zero and returns the nearest integral value (trunc(10.25) = 10).

Speed.vector.x accessing the data coming from signal (speed of the left motor).

ROS_INFO to display information.

Same for speed of right motor (speed.vector.y).

speed_dt = speed.vector.z; is the angular velocity variable needed ro calculate the theta

```
int main(int argc, char** argv){
  ros::init(argc, argv, "Smart_Wheel_Chair_controller");


  ros::NodeHandle n;
  ros::NodeHandle nh_private_("~");
  ros::Subscriber sub = n.subscribe("speed", 50, handle_speed);
  ros::Publisher odom_pub = n.advertise<nav_msgs::Odometry>("odom", 50);
  tf::TransformBroadcaster broadcaster;
```

To init the node called “Smart_Wheel_Chair_controller”

On creation, if the internal node has not been started already, ros::NodeHandle will start the node. Once all ros::NodeHandle instances have been destroyed, the node will be automatically shutdown.

NodeHandle does not allow passing private names directly to its methods, or to constructors that take a NodeHandle as an argument.

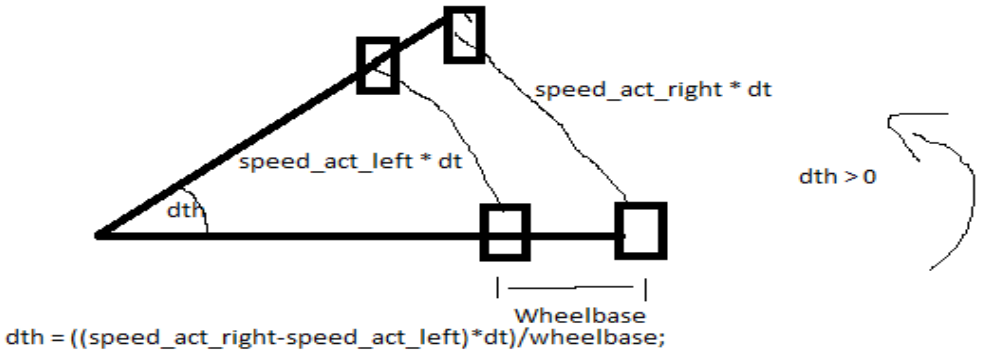
“~” means my_node_name for examples see here.

To subscribe to a topic called “speed” and with rate 50 (Define rate for repeatable operations) and call the function (handle_speed) every time I subscribe to this topic I call the function.

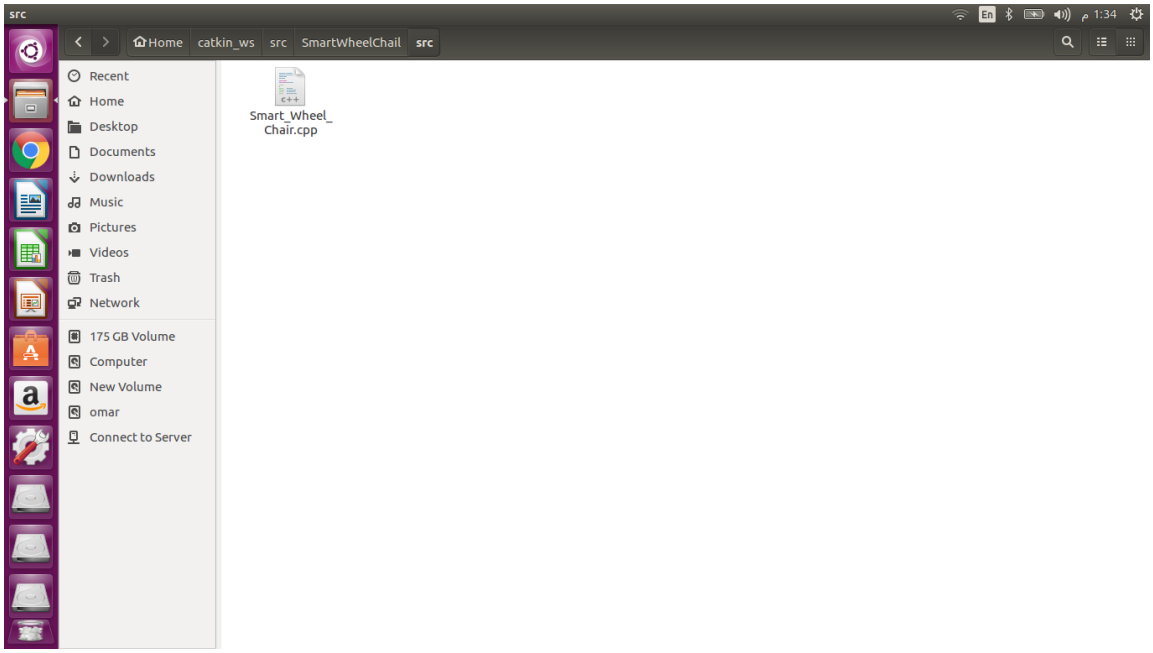
```
double rate = 10.0;
double linear_scale_positive = 1.0;
double linear_scale_negative = 1.0;
double angular_scale_positive = 1.0;
double angular_scale_negative = 1.0;
bool publish_tf = true;
double dt = 0.0;
double dx = 0.0;
double dy = 0.0;
double dth = 0.0;
double dxy = 0.0;
double vx = 0.0;
double vy = 0.0;
double vth = 0.0;
char base_link[] = "/base_link";
char odom[] = "/odom";
char kinect[] = "/kinect";
char camera_link[] = "/camera_link";
ros::Duration d(1.0);
nh_private_.getParam("publish_rate", rate);
nh_private_.getParam("publish_tf", publish_tf);
nh_private_.getParam("linear_scale_positive", linear_scale_positive);
nh_private_.getParam("linear_scale_negative", linear_scale_negative);
nh_private_.getParam("angular_scale_positive", angular_scale_positive);
```

```
 ros::Rate r(rate);

 while(n.ok()){  //Check if ROS is working. E.g. if ROS master is stopped or there was sent signal to stop the system, ros::ok() will return false.

  ros::spinOnce(); // Process all incoming messages. But once because of we are In awhile loop

  current_time = speed_time; // saving the current time

 // time

  dt = speed_dt;                                          //Time in s

  ROS_INFO("dt : %f", dt);

  // average distance to handle it if there is a little  difference between speed of left and right motors

  dxy = (speed_act_left+speed_act_right)*dt/2;

  ROS_INFO("dxy : %f", dxy);

  dth = ((speed_act_right-speed_act_left)*dt)/wheelbase;


  if (dth > 0) dth *= angular_scale_positive; // + ve angle with clockwise

  if (dth < 0) dth *= angular_scale_negative; // -ve angle anti clockwise

  if (dxy > 0) dxy *= linear_scale_positive; // + move forward (encoder data +ve)

  if (dxy < 0) dxy *= linear_scale_negative; // - move backward(encoder data -ve)


  dx = cos(dth) * dxy; // position of robot according to x-axis

  dy = sin(dth) * dxy; // position of robot according to y-axis

  x_pos += (cos(theta) * dx - sin(theta) * dy); //actual position in x-axis using homogeneos

  y_pos += (sin(theta) * dx + cos(theta) * dy);// actual position in y-axis using homogeneos

  theta += dth;

  if(theta >= two_pi) theta -= two_pi;

  if(theta <= -two_pi) theta += two_pi;
```

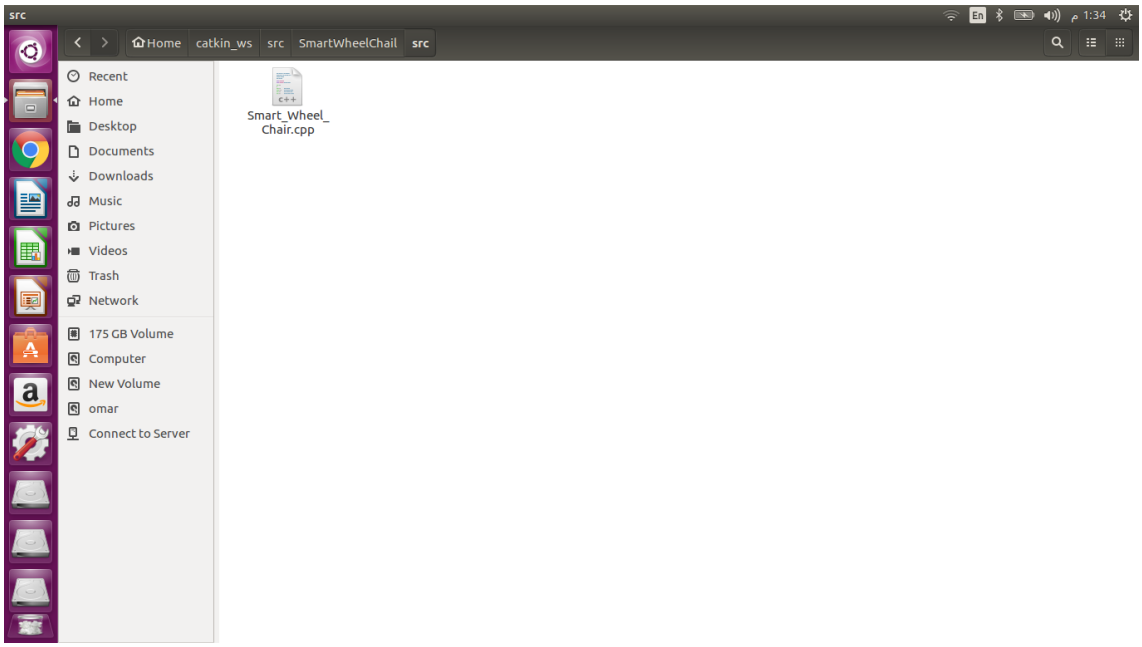
This is the algoritm used to calculate the current position of the robot.



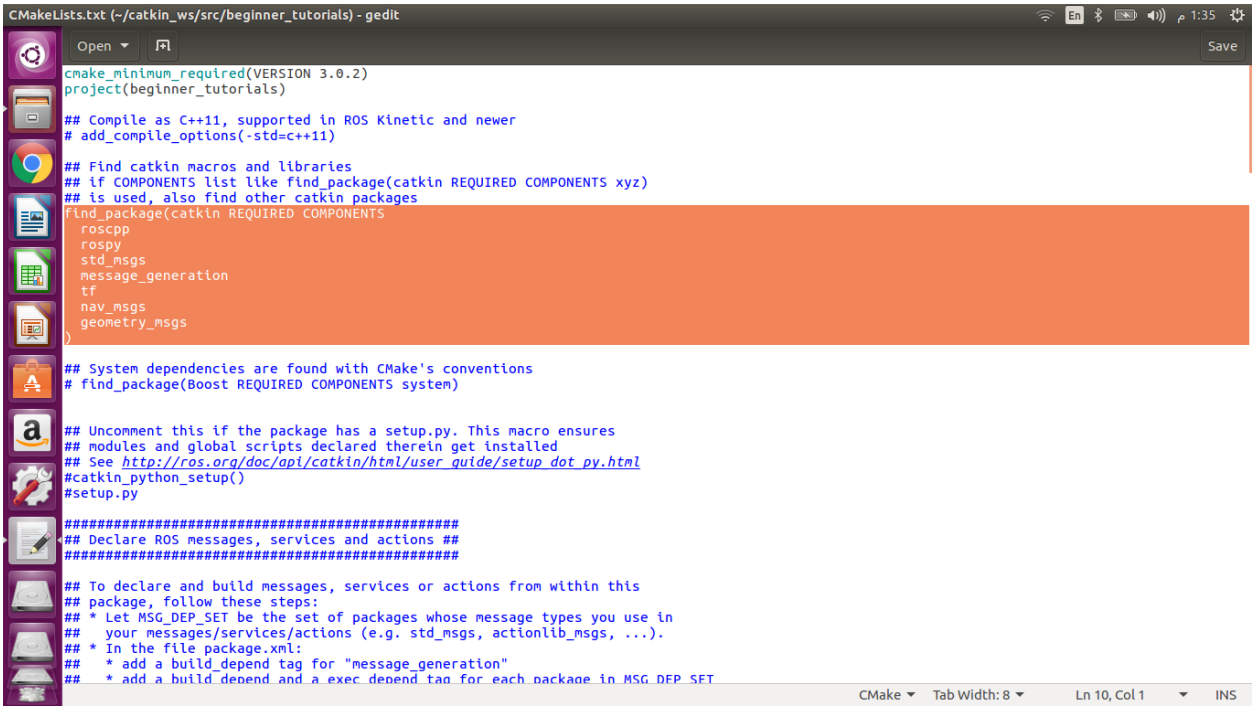
To make the code executable we must make a package that depends on the packages that we have included in the code.

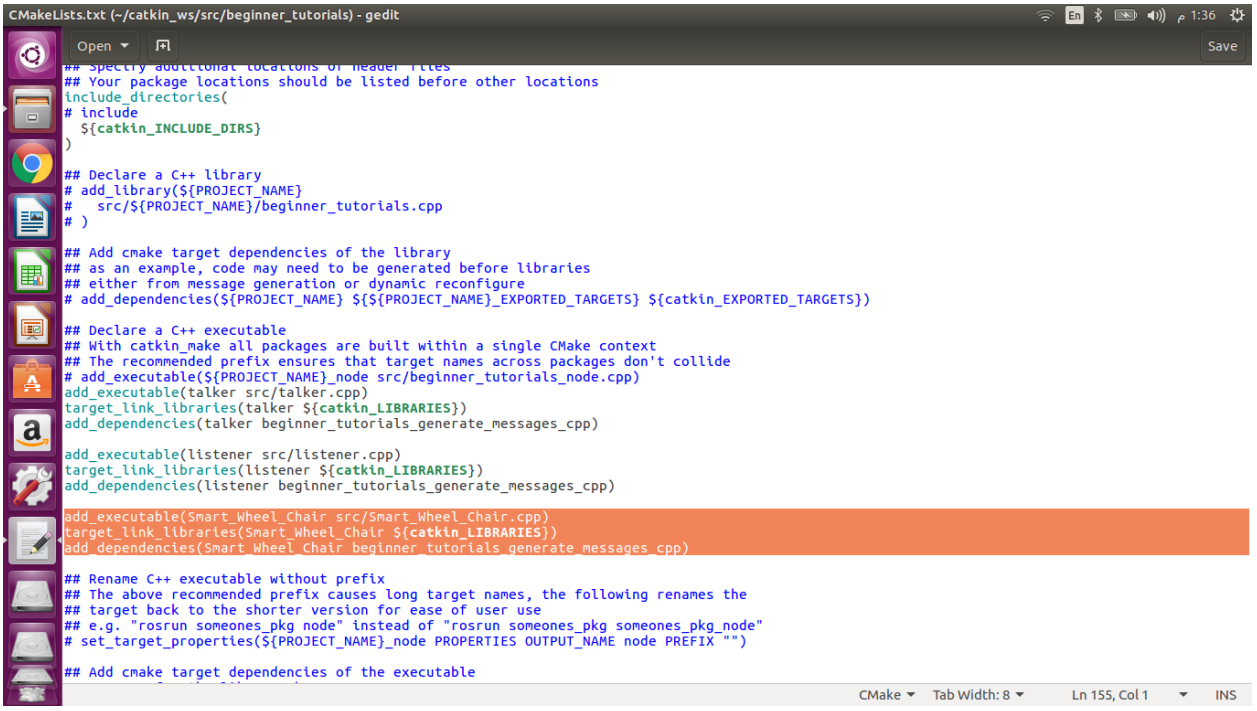


Then we put the c++ code in the src folder

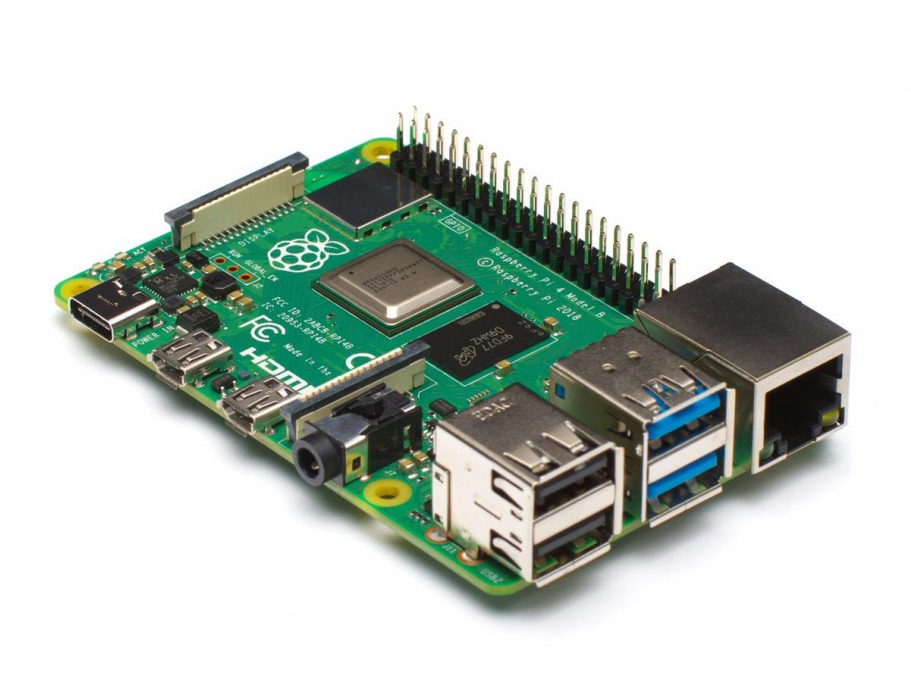


Note that we must modify the CMakeLists file to make our code executable


CMakeLists.txt (~/catkin_ws/src/beginner_tutorials) - gedit
Open
Save
cmake_minimum_required(VERSION 3.0.2)
project(beginner_tutorials)
Compile as C++11, supported in ROS Kinetic and newer
add_compile_options(-std=c++11)
Find catkin macros and libraries
if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
roscpp
rospy
std_msgs
message_generation
tf
nav_msgs
geometry_msgs
)
System dependencies are found with CMake's conventions
find_package(Boost REQUIRED COMPONENTS system)
Uncomment this if the package has a setup.py. This macro ensures
modules and global scripts declared therein get installed
See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
#catkin_python_setup()
#setup.py
##
Declare ROS messages, services and actions
##
To declare and build messages, services or actions from within this
package, follow these steps:
* Let MSG_DEP_SET be the set of packages whose message types you use in
your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
* In the file package.xml:
* add a build_depend tag for "message_generation"
* add a build_depend and a exec_depend tag for each package in MSG_DEP_SET
CMake
Tab Width: 8
Ln 10, Col 1
INS


CMakeLists.txt (~/catkin_ws/src/beginner_tutorials) - gedit
Open
Save
Your package locations should be listed before other locations
include_directories(
include
${catkin_INCLUDE_DIRS}
)
Declare a C++ library
add_library(${PROJECT_NAME}
src/${PROJECT_NAME}/beginner_tutorials.cpp
)
Add cmake target dependencies of the library
as an example, code may need to be generated before libraries
either from message generation or dynamic reconfigure
add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
Declare a C++ executable
With catkin_make all packages are built within a single CMake context
The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/beginner_tutorials_node.cpp)
add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker beginner_tutorials_generate_messages_cpp)
add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener beginner_tutorials_generate_messages_cpp)
add_executable(Smart_Wheel_Chair src/Smart_Wheel_Chair.cpp)
target_link_libraries(Smart_Wheel_Chair ${catkin_LIBRARIES})
add_dependencies(Smart_Wheel_Chair beginner_tutorials_generate_messages_cpp)
Rename C++ executable without prefix
The above recommended prefix causes long target names, the following renames the
target back to the shorter version for ease of user use
e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")
Add cmake target dependencies of the executable
CMake
Tab Width: 8
Ln 155, Col 1
INS

Chapter (7): Raspberry Pi



Robotic projects needs a computer with a good processing power so it can handle all the sensor data it reads and process it with efficient speed.

Computers or laptops offer this processing power but we can't use them on the top of the robot as our processing unit. Well, it is possible to do that but then the robot will have a laptop on the top of it all the time so this robot won't be used in practice.

Fortunately, there is an alternative which is the Raspberry pi.

What is Raspberry Pi?

Raspberry pi is a small computer, as small as a credit card, that can do almost anything a normal computer can do from connecting to a monitor, mouse or keyboard, browsing the internet, playing high-definition video or even playing games.

Generations

There are different generations of Raspberry pi. Each generation has different processing power and memory.

The latest generation that offers the most processing power is the Raspberry pi 4B where it uses a 1.5GHz 64-bit quad-core ARM Cortex-A72 processor with a 1MB cache memory. We will be using the Raspberry pi 4B 8GB RAM in our project.

Operating system

There are some choices for operating systems to be installed on the Raspberry 4B But we have to keep in mind that we need to install ROS kinetic on whatever operating system we choose to install on it.

Previously in the ROS chapter we mentioned that we installed ROS kinetic instead of melodic and the reason is that there was some problems trying to install ROS melodic on the Raspberry pi 4B.

The best Operating system in general for the Raspberry pi 4B is the Ubuntu mate as it offers almost everything Ubuntu offers and it is light weighted so it suits the Raspberry pi so well.

Right now, by the time of writing this documentation, only the 20.04 version can be downloaded from the official site. Note that we need an img not and iso so we can burn it on the SD card and install it on the Raspberry pi.

The problem we faced is that ROS melodic can be installed only on the 18.04 version of Ubuntu and can’t be installed on the 20.04 and there is not any

18.04 img version on the official site or anywhere on the internet (we couldn't find any).

There is another linux distro called raspberian and it is the official distro for Raspberry pi but after installing it we faced some problems and couldn't install ROS on it as well.

After some searching we found a distro that has a pre-installed ROS which is (Ubiquity Robotics Raspberry Pi image) on it but the the version of the pre-installed ROS is kinetic as it is based on ubuntu 16.04 and that was the reason why we changed from ROS melodic to ROS kinetic.

Using Raspberry Pi

Now that we have installed the distro, we are ready to use the Raspberry Pi.

The first way to control the Raspberry pi is by using it as a normal computer connecting it to a monitor and connect a mouse and a keyboard to it and we will have a full access to the GUI so it is easy to use.

There is another method to use the Raspberry pi which is through the SSH protocol (Source Shell protocol) by connecting raspberry pi to our device through the access point created on ubiquity robotic raspberry pi image or by connecting both raspberry pi and our device on same network through wi-fi or Ethernet.

It is more practical to use the SSH protocol and getting used to it as in real applications the Raspberry pi is not connected to a screen most of the time so by using the SSH protocol we can connect to it wireless and start using the Raspberry pi through the shell commands. For more information about SSH

protocol refer to https://searchsecurity.techtarget.com/definition/Secure-Shell .

Need for workstation

As mentioned above raspberry oi is headless so although it runs sensors and make processing, we aren't able to see what's happening, so we need a workstation to display what is going on. The workstation is setup so that the robot is the ROS master and this allows ROS running on the workstation to monitor ROS topics which are running on robot (Raspberry pi). Both robot and workstation must have the same distro which means they will have same distro of ubuntu.

Steps required to run robot as master and our device as a workstation

Firstly, make sure that both robot and device are on same network then, we need to ssh robot form our device so open a terminal on your device and follow commands below:

ssh username@ ip-address of your robot on network

ssh ubuntu@10.42.0.1

then it will ask for password so u will enter it. Now u are on the raspberry pi terminal so you will enter these two commands at ~./barshc

export ROS_MASTER_URI=http://<robot-ip>:11311

export ROS_HOSTNAME=<robot-ip

export ROS_MASTER_URI=http://10.42.0.3:11311

export ROS_HOSTNAME=10.42.0.1

Now open a new terminal at your device and follow commands:

export ROS_MASTER_URI=http://<robot-up>:11311

export ROS_HOSTNAME=<devcom-ip>

export ROS_MASTER_URI=http://10.42.0.3:11311

export ROS_HOSTNAME=<device ip>

you can get your device ip by the command **ifconfig**

.....

Chapter (8): Voice Recognition

When you bang a drum its skin vibrates. The harder you bang, the bigger the vibrations. The vibrating drum skin causes nearby air particles to vibrate, which in turn causes other nearby air particles to vibrate. These vibrating particles make up a sound wave.

Sound waves move at a speed of 343 m/s in the air and even faster in liquids and solids. The waves transport energy from the sound source, such as a drum, to the environment. When vibrating air particles force your ear drum to vibrate, your ear detects sound waves. The sound gets louder as the vibrations get bigger.

Sound Wave

Artificial Intelligence

Artificial intelligence (AI) is the simulation of human intelligence in robots that have been trained to think and act like humans. The phrase can also refer to any machine that demonstrates human-like characteristics like learning and problem-solving.

The ability of artificial intelligence to rationalize and execute actions that have the best likelihood of reaching a certain goal is its ideal feature. Machine learning is a subset of artificial intelligence that refers to the idea that computer systems can learn from and adapt to new data without the need for human intervention.

Deep learning techniques allow for this autonomous learning by absorbing large volumes of unstructured data including text, photos, and video.

The majority of people immediately think of robots when they hear the words artificial intelligence. That's because big-budget movies and literature have human-like computers wreaking devastation on the planet. Nothing, however, could be further from the truth.

Artificial intelligence is founded on the idea that human intelligence may be characterized in such a way that a machine can simply duplicate it and carry out activities ranging from the most basic to the most complicated. Artificial intelligence's goals include simulating human cognitive processes. To the extent that they can be concretely characterized, researchers and developers in the field are making unexpectedly rapid progress in simulating tasks such as learning, reasoning, and perception. Some predict that in the

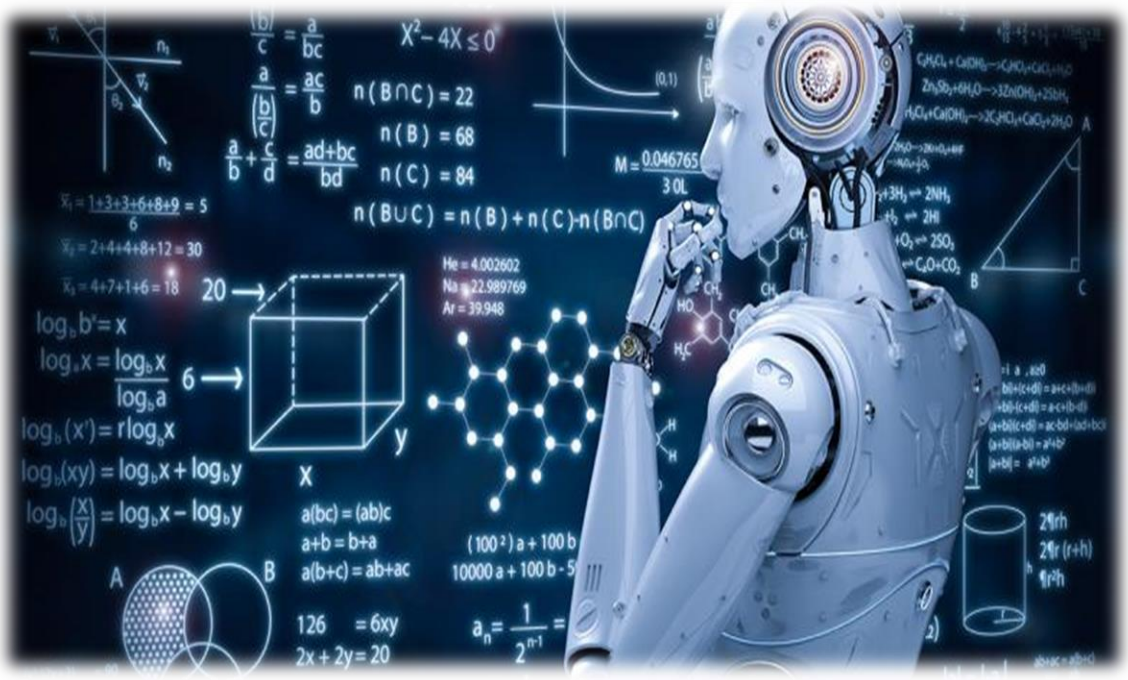
near future, innovators will be able to create systems that can learn and reason about any subject faster than humans can. Others, on the other hand, remain suspicious, claiming that all cognitive activity is loaded with value judgments based on human experience.

As technology progresses, earlier artificial intelligence criteria become obsolete. Machines that calculate basic calculations or recognise text using optical character recognition, for example, are no longer called artificial intelligence because these functions are now regarded standard computer functions.

AI is constantly improving to benefit a wide range of sectors. A multidisciplinary approach based on mathematics, computer science, linguistics, psychology, and other disciplines is used to wire machines.

Machine Learning

The premise of machine learning is that a computer program can learn and adapt to new data without the need for human involvement. Machine learning is a branch of artificial intelligence (AI) that maintains the accuracy of a computer's built-in algorithms.



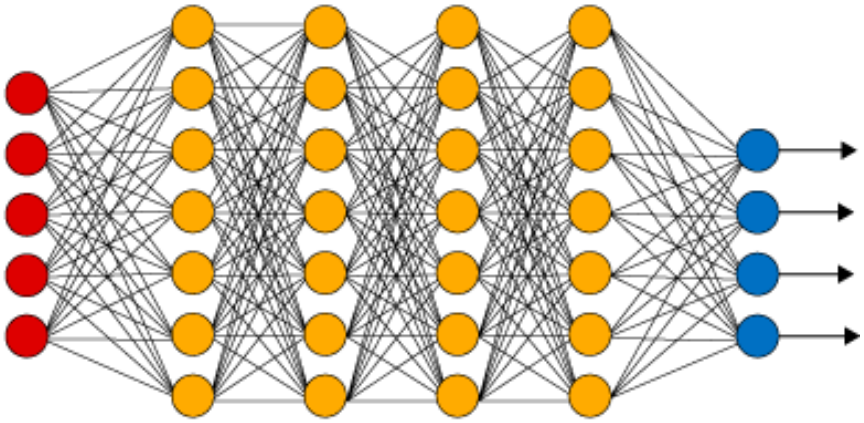
Various areas of the business are dealing with massive amounts of data in various formats gathered from various sources. Because of the advancement of technology, particularly increased processing capabilities and cloud storage, vast amounts of data, known as big data, are becoming more readily available and accessible. Companies and governments recognise the enormous insights that can be obtained from analysing big data, but often lack the resources and time to go through its vast amounts of data.

As a result, various businesses are employing artificial intelligence technologies to gather, process, communicate, and exchange important information from data sets. Machine learning is one type of AI that is increasingly being used for massive data processing.

Machine learning data applications are created using a complicated algorithm or source code embedded in the machine or computer. This programming code constructs a model that recognises data and makes predictions based on that data. For its decision-making process, the model leverages parameters integrated into the algorithm to build patterns. When new or extra data becomes available, the algorithm modifies the parameters automatically to see whether there is a pattern shift. The model, on the other hand, should not be altered.

Deep Learning

Deep learning is an artificial intelligence (AI) function that mimics the human brain's processing of data and pattern creation in order to make decisions. Deep learning is an artificial intelligence subset of machine learning that uses neural networks to learn unsupervised from unstructured or unlabeled data. Deep neural learning or deep neural network are other terms for the same thing.



Deep learning has progressed in lockstep with the digital era, which has resulted in an avalanche of data in all formats and from all corners of the globe. Big data is gathered from a variety of sources, including social media, internet search engines, e-commerce platforms, and online theatres. This massive volume of data is easily accessible and can be shared via fintech tools such as cloud computing.

However, because the data is typically unstructured, it could take decades for humans to analyse and extract useful information. Companies are increasingly using AI systems for automated support as they see the enormous potential that can be realised by unlocking this wealth of data.

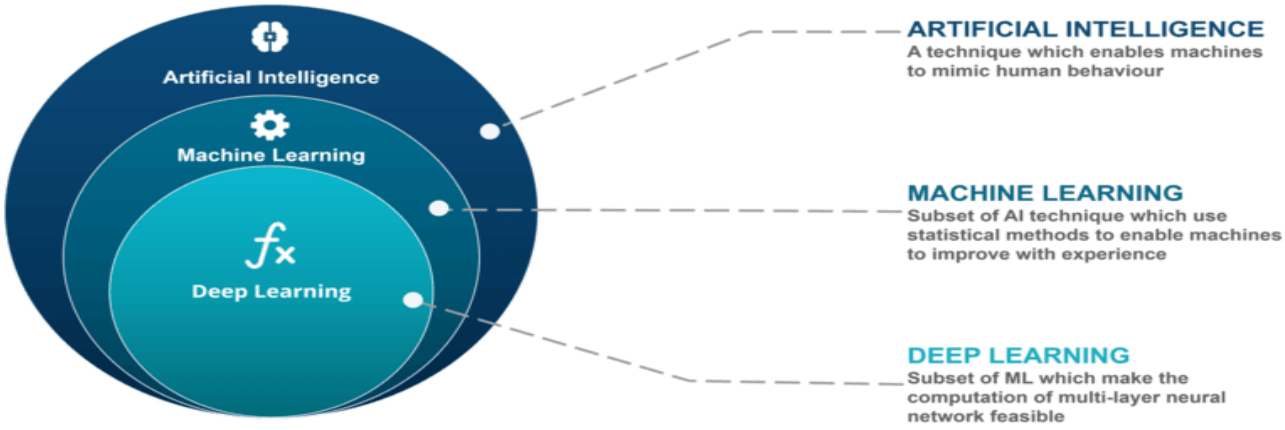
Machine Learning vs. Deep Learning

Machine learning, a self-adaptive algorithm that gains increasingly better analysis of patterns with experience or freshly acquired data, is one of the most common AI techniques used for processing massive data.

Machine learning tools could be used by a digital payments corporation to detect the presence or potential for fraud in its system. A computational algorithm implemented into a computer model will process all transactions that take place on the digital platform, look for patterns in the data set, and flag any anomalies found by the pattern.

Machines may now handle data in a nonlinear manner thanks to the hierarchical function of deep learning systems.

Deep learning, a subset of machine learning, employs artificial neural networks at a hierarchical level to carry out machine learning. The artificial neural networks are constructed in the same way as the human brain, with neuron nodes connected in a web-like fashion. The hierarchical function of deep learning systems allows machines to process data in a nonlinear manner, whereas standard programmes develop analyses using data in a linear manner.



Convolutional Neural Networks (CNN)

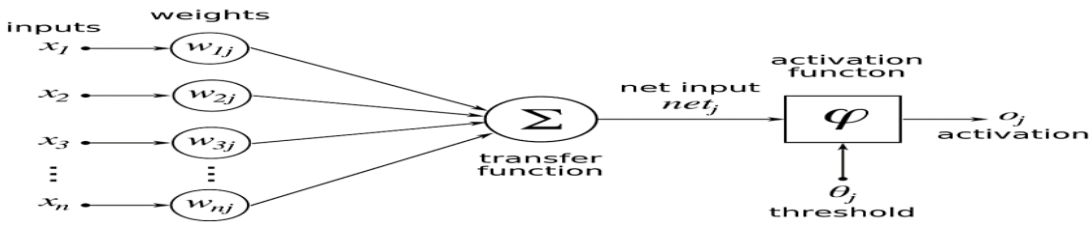
Computer scientists have been trying to construct computers that can understand visual input since the 1950s, when artificial intelligence was in its infancy. Over the next few decades, the field of computer vision progressed at a slow but steady pace. When a group of academics from the University of Toronto built an AI model that outperformed the top image recognition algorithms by a considerable margin in 2012, computer vision took a quantum leap.

The AI system, which became known as AlexNet (named after its main creator, Alex Krizhevsky), won the 2012 ImageNet computer vision contest with an amazing 85 percent accuracy. The runner-up scored a modest 74 percent on the test.

A convolutional neural network (CNN), a specific sort of artificial neural network that essentially resembles the human vision system, was at the heart of the AlexNet. CNNs have grown increasingly important in computer

vision applications in recent years. Here's everything you need to know about CNN's history and operation.

Multiple layers of artificial neurons make up convolutional neural networks. Artificial neurons are mathematical functions that calculate the weighted sum of various inputs and output an activation value, similar to their biological counterparts.



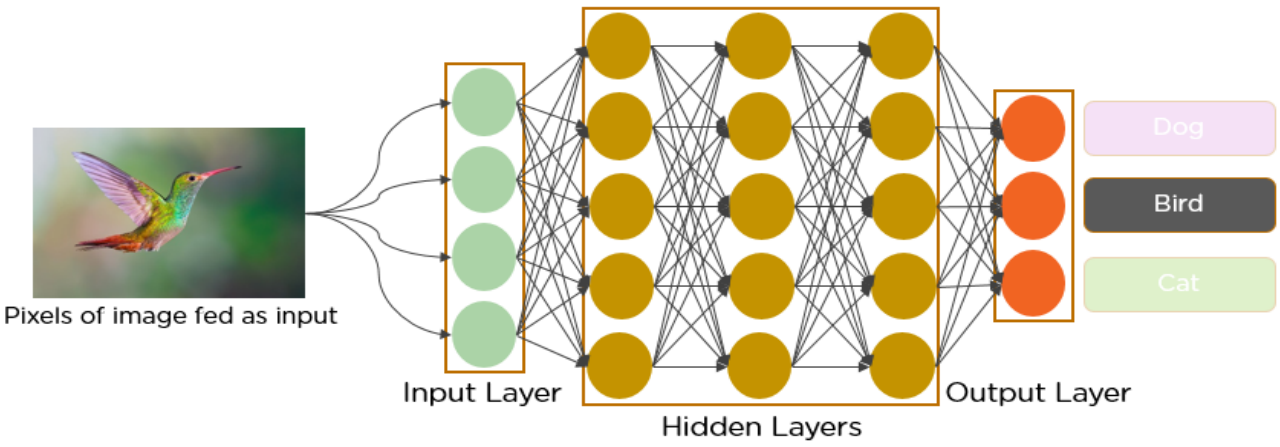
Each neuron's action is determined by its weights. The artificial neurons in a CNN pick out numerous visual properties when fed with pixel values.

Each of the layers in a ConvNet generates various activation maps when you feed it an image. Activation maps highlight the image's most important characteristics. Each neuron takes a pixel patch as input, multiplies its colour values by its weights, adds them together, and runs them through the activation function.

Basic features such as horizontal, vertical, and diagonal edges are usually detected by the CNN's first (or bottom) layer. The first layer's output is sent into the next layer, which extracts more complicated features like corners and edge combinations. The layers recognise higher-level features such as objects, faces, and more as you progress further into the convolutional neural network.

Convolution is the process of multiplying pixel values by weights and summing them (hence the name convolutional neural network). A CNN is typically made up of numerous convolution layers, but it can also include other elements. A classification layer is the final layer of a CNN, and it takes the output of the final convolution layer as input (remember, the higher convolution layers detect complex objects).

The classification layer generates a series of confidence ratings (numbers between 0 and 1) based on the activation map of the final convolution layer, which indicate how likely the image is to belong to a “class.” For example, if a ConvNet detects cats, dogs, and horses, the final layer's output is the likelihood that the input image contains any of those animals.



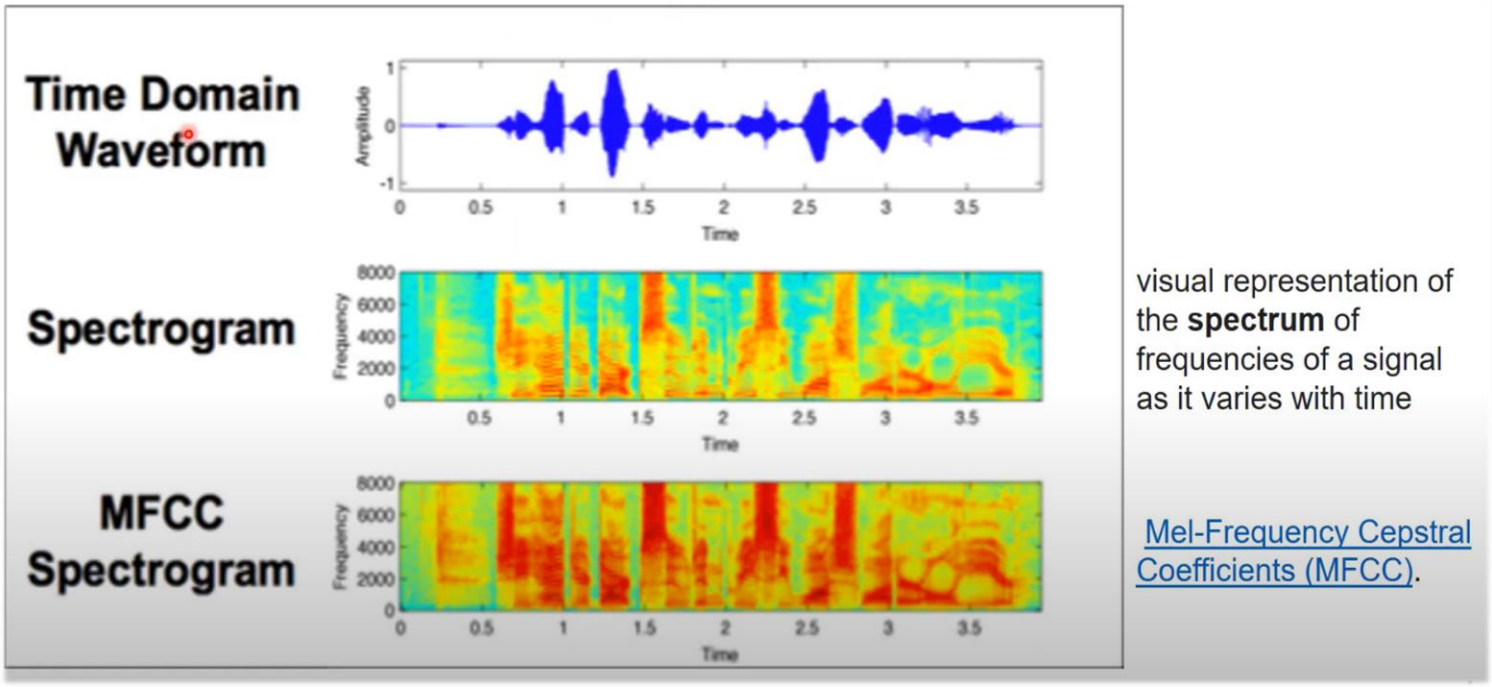
Principle of Speech Recognition

Ai can’t deal with the sound waves so if we need to recognize the voice, we will convert it to the wave form and draw it.

But in this case we will have a problem all waves in time domain are very similar to each other.

So we have a frequency domain we can use it but we have another problem in this case if we have some noise the word which you say will be another word in the model and the number of frequency are very huge.

What if we draw every number of this frequency by a specific color so we can use Mel frequency cepstral coefficient (Mfcc) .

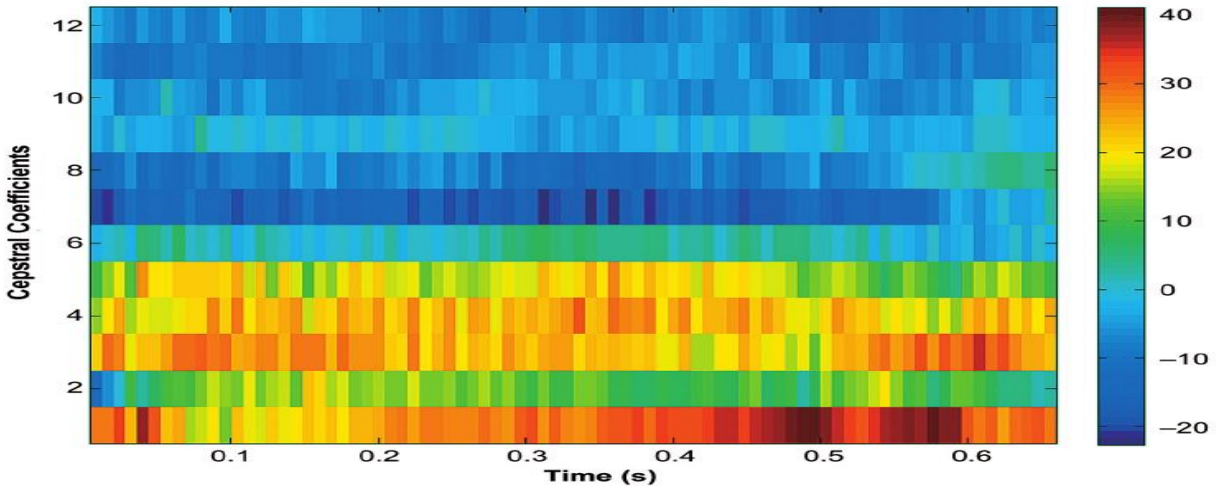


Mel Frequency Cepstral Coefficient (MFCC)

This is the most important thing to make us recognize the speech.

because this tool makes us to convert the sound wave to image contain all the frequency and every specific frequency have a specific color.

Before we make a model to recognize the speech, we need a lot of data to give it to the model to train by it.



Preparing The Data

This the first step to build the speech recognition model.

In this step we need some requirement:

1- Python 3.7

2- Json library

3- Librosa library

4- Os library

Import libraries and define the dictionary:

```
import librosa
import os
import json

DATASET_PATH = "dataset"
JSON_PATH = "data.json"
SAMPLES_TO_CONSIDER = 22050 # 1 sec. of audio


def preprocess_dataset(dataset_path, json_path, num_mfcc=13, n_fft=2048, hop_length=512):
    # dictionary where we'll store mapping, labels, MFCCs and filenames
    data = {
        "mapping": [],
        "labels": [],
        "MFCCs": [],
        "files": []
    }
```

To make a good accuracy you want to have a lot of data so in this model we will predict the English numbers from 0 to 9.

So, we collect 2850 audio file for each number to train the model by it.

To use the library which help you in this code we will import it first.

We should read the whole audio file for all number, so we use os library for that.

We need to convert every audio file to MFCC form to make it easy for CNN take some features from every audio file to train the model by it so librosa will help us for that.

Read all data audio file and extract the MFCCs from it and store it:

```
# loop through all sub-dirs
for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):

    # ensure we're at sub-folder level
    if dirpath is not dataset_path:
        # save label (i.e., sub-folder name) in the mapping
        label = dirpath.split("/")[-1]
        data["mapping"].append(label)
        print("\nProcessing: '{}'".format(label))
        # process all audio files in sub-dir and store MFCCs
        for f in filenames:
            file_path = os.path.join(dirpath, f)
            # load audio file and slice it to ensure length consistency among different files
            signal, sample_rate = librosa.load(file_path)
            # drop audio files with less than pre-decided number of samples
            if len(signal) >= SAMPLES_TO_CONSIDER:
                # ensure consistency of the length of the signal
                signal = signal[:SAMPLES_TO_CONSIDER]
                # extract MFCCs
                MFCCs = librosa.feature.mfcc(signal, sample_rate, n_mfcc=num_mfcc, n_fft=n_fft,
                                             hop_length=hop_length)
                # store data for analysed track
                data["MFCCs"].append(MFCCs.T.tolist())
                data["labels"].append(i-1)
                data["files"].append(file_path)
                print("{}: {}".format(file_path, i-1))
```

Now we will read all audio file and convert it to MFCC form.

Now we ready for preparing the data.

We will read all audio file and convert it to MFCC form and we will make a dictionary that contain (mapping, labels, MFCCs, files)

Mapping contain the name of the directory which have the 2800 audio file so in this case mapping have 10 word from zero to nine.

Labels contain number that pointing to every number. in this case the first number pointing to zero and so on.

MFCCS pointing to the number of MFCC for each audio file.

File pointing to the name file of every number.

We will load the whole audio file by librosa and convert it to MFCC and after we do that, we will save this dictionary in json file.

When we load the audio file and convert it we give it some arguments

Num_Mfcc → Number of coefficients to extract.

N_fft → Interval we consider applying FFT. Measured in # of samples.

hop_length → Sliding window for FFT. Measured in # of samples.

in this data we will load the audio file that contain 1 second of voice only so SAMPLES_TO_CONSIDER = 22050 this number mean 1 sec of speech

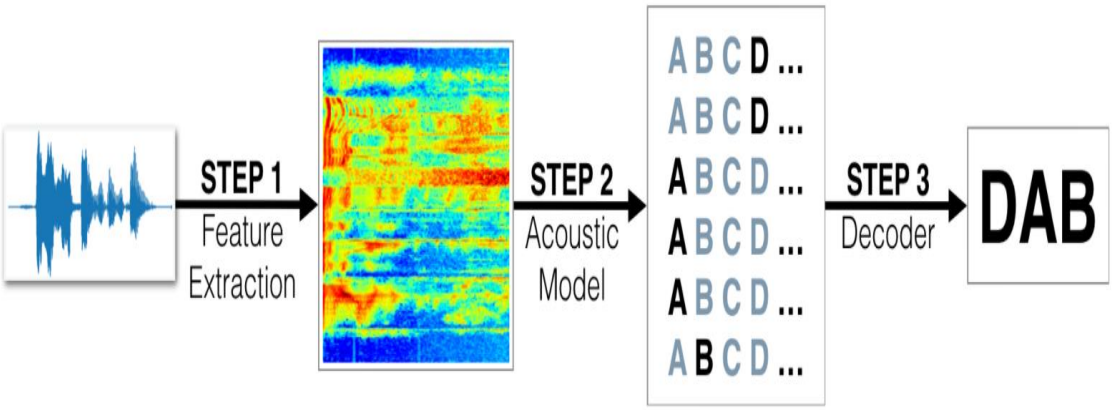
Store the data in a json file:

```
    # save data in json file
    with open(json_path, "w") as fp:
        json.dump(data, fp, indent=4)
```

we need to save all this data in a json file because this type is making the read data is very easy to deal with it, so we use the third library json library.

now we are ready to build a model by our own data which we preparing above.

Building CNN Model



The requirements in the 2nd step are

1- json library
2- Tensorflow library (try to install it by anaconda because this library will make some problem if you try to install it by the terminal by pip so I recommend to download anaconda and use this command ~ conda install tensorflow , and you can do it with the other library by this command ~ conda install lib_name)
3- Numpy library
4- matplotlib.pyplot library
5- from sklearn.model_selection import train_test_split

Import libraries and define variables:

```
import json
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

DATA_PATH = "data.json"
SAVED_MODEL_PATH = "model.h5"
EPOCHS = 40
BATCH_SIZE = 32
PATIENCE = 5
LEARNING_RATE = 0.0001
```

I tried to use vosk library to build a model, but I found it very difficult to install and caused different problems, so I used these libraries.

Read data from json file and store it:

```
with open(data_path, "r") as fp:
    data = json.load(fp)

X = np.array(data["MFCCs"])
y = np.array(data["labels"])
print("Training sets loaded!")
return X, y
```

We start by reading the json file that we made in the 1st step.

And store the MFCC and Labels in x and y variables:

```
test_size=0.2,
```

20% of data we will use it to test the model and the other 80% we will use it to train the model.

Add new axis:

```
# add an axis to nd array
X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]
X_validation = X_validation[..., np.newaxis]
```

X_train is 2d array and we should convert it to 3d array so we use numpy to do this step and we put the third layer by 1 to make the image gray and we do this step to the x_test and x_validation .

```
# build network architecture using convolutional layers
model = tf.keras.models.Sequential()
```

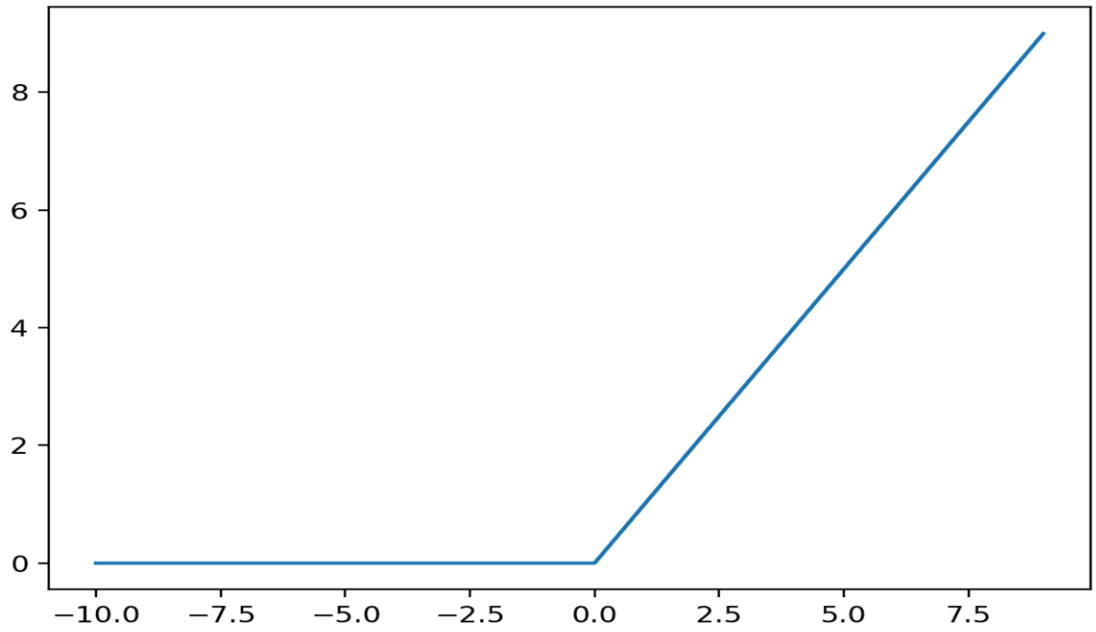
this is line build a model that we try to make.

Make a 3 convolution layers:

```
# 1st conv layer
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=input_shape,
                                 kernel_regularizer=tf.keras.regularizers.l2(0.001)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D((3, 3), strides=(2,2), padding='same'))

# 2nd conv layer
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
                                 kernel_regularizer=tf.keras.regularizers.l2(0.001)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D((3, 3), strides=(2,2), padding='same'))

# 3rd conv layer
model.add(tf.keras.layers.Conv2D(32, (2, 2), activation='relu',
                                 kernel_regularizer=tf.keras.regularizers.l2(0.001)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D((2, 2), strides=(2,2), padding='same'))
```

after that we make a number of convolution layers in this model, we make 3 layers, but we can make more or less number if we want. it's depend on the accuracy so we can say number of layers is a hyperparameter.

for each layer we out the 1st number which pointing to number of filters we use so in the first layer we use 64 filter and we use activation function " relu " and this is relu function f(x)=max(0,x)



Activation Function

activation functions give out the final value given out from a neuron. So, an activation function is basically just a simple function that transforms its inputs into outputs that have a certain range. There are various types of activation functions that perform this task in a different manner, For example, the sigmoid activation function takes input and maps the resulting values in between 0 to 1.

One of the reasons that this function is added into an artificial neural network in order to help the network learn complex patterns in the data. These functions introduce nonlinear real-world properties to artificial neural networks. Basically, in a simple neural network, x is defined as inputs, w weights, and we pass f (x) that is the value passed to the output of the network. This will then be the final output or the input of another layer.

If the activation function is not applied, the output signal becomes a simple linear function. A neural network without activation function will act as a linear regression with limited learning power. But we also want our neural network to learn non-linear states as we give it complex real-world information such as image, video, text, and sound.

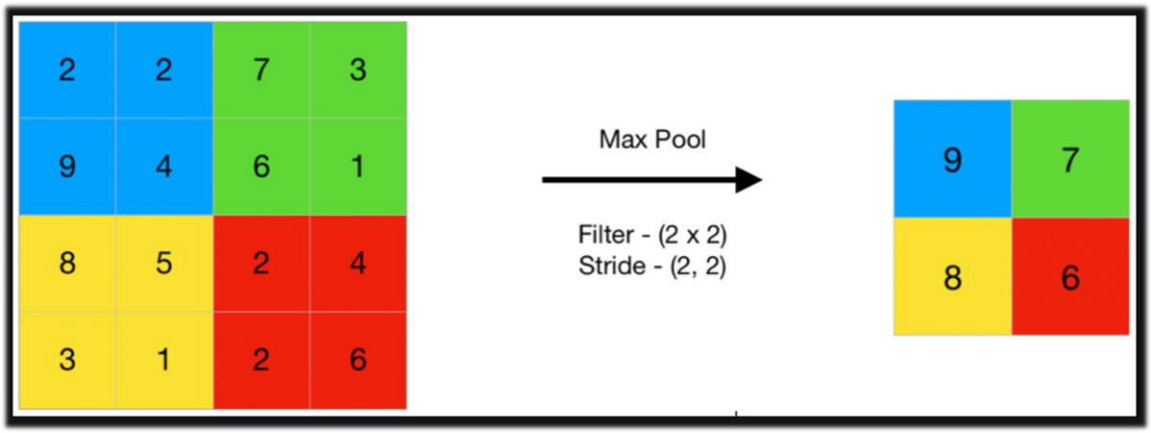
Max Pooling

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

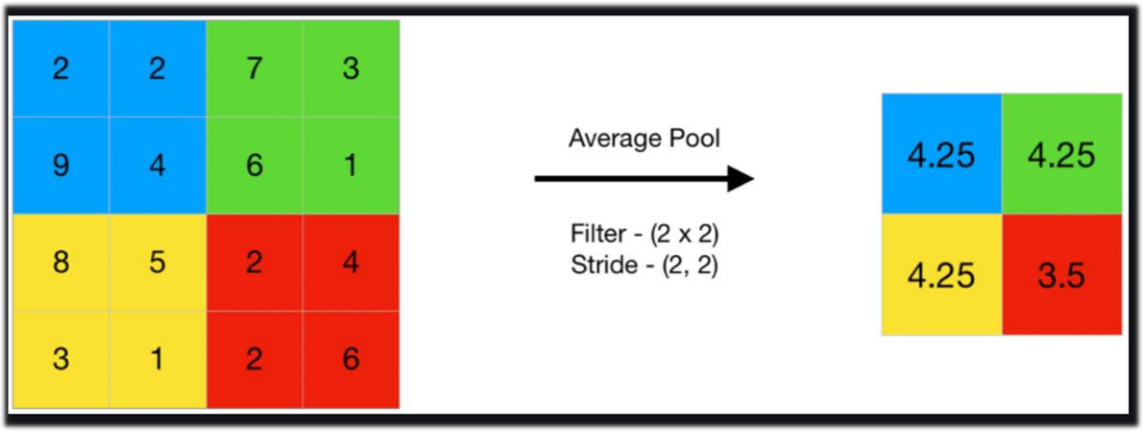
Types of Pooling Layers

1- Max pooling (which recommended in many models) is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



2- Average Pooling

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

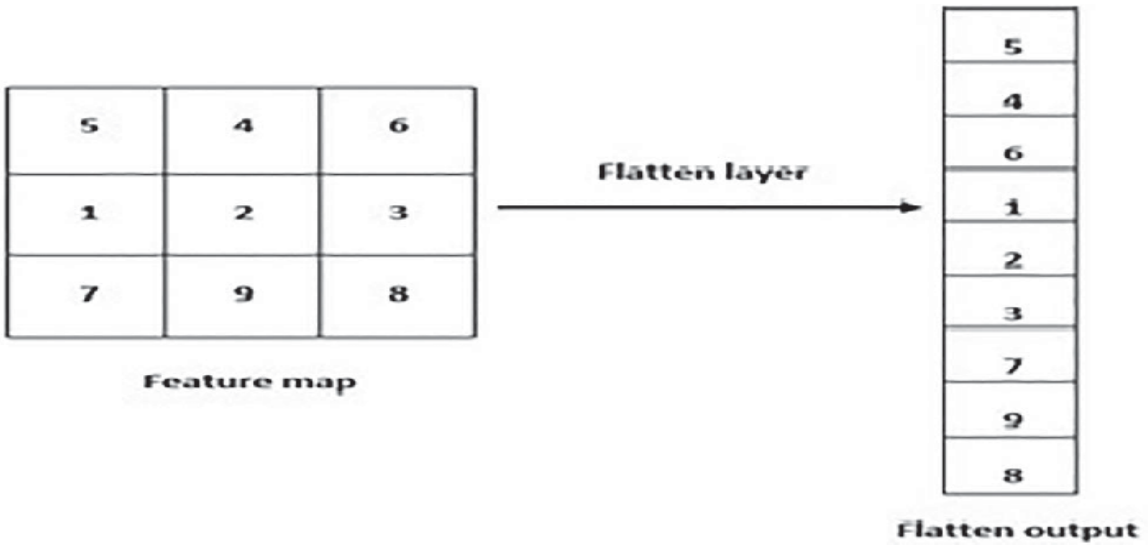


In the 2nd convolution layer, we do the same steps, but we reduce the number of filters to 32.

In the 3rd convolution layer, we do the same steps but we make the kernel of maxpooling (2, 2).

Flatten the output:

```
# flatten output and feed into dense layer
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64, activation='relu'))
tf.keras.layers.Dropout(0.3)
```



Now we will make a flatting step to make the output reshape in 1D array to make it a input to the last neurons to give us the nearby output.

Dropout used to reduce the number of the neurons which we don't need because there are not important so that make the model make processing faster.

Softmax the output layer:

```
# softmax output layer
model.add(tf.keras.layers.Dense(10, activation='softmax'))

optimiser = tf.optimizers.Adam(learning_rate=learning_rate)
```

To reduce the processing of model again we need to make a softmax activation to make the model faster and faster.

Optimizers help model to detect the final value of input we user the common optimizer " adam ".

```
# compile model
model.compile(optimizer=optimiser,
              loss=loss,
              metrics=["accuracy"])
```

after that we should compile the model:

```
def train(model, epochs, batch_size, patience, X_train, y_train, X_validation, y_validation):
```

we use a variable that called epochs that mean the number of the model will see the whole data and that variable is a hyperparameter and in this model we put it 40.

batch_size → Samples per batch

patience → Num epochs to wait before early stop if there isn't an improvement on accuracy.

```
earlystop_callback = tf.keras.callbacks.EarlyStopping(monitor="accuracy", min_delta=0.001, patience=patience)
```

to avoid the processing doesn't give us a improvement in the accuracy we use Earlystopping when the change in delta loss is 0.001 and in this model we use "sparse_categorical_crossentropy" as a loss function.

```
loss="sparse_categorical_crossentropy"
```

Plot Model Summary

```
def plot_history(history):
    """Plots accuracy/loss for training/validation set as a function of the epochs
    :param history: Training history of model
    :return:
    """

    fig, axs = plt.subplots(2)

    # create accuracy subplot
    axs[0].plot(history.history["accuracy"], label="accuracy")
    axs[0].plot(history.history['val_accuracy'], label="val_accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy evaluation")

    # create loss subplot
    axs[1].plot(history.history["loss"], label="loss")
    axs[1].plot(history.history['val_loss'], label="val_loss")
    axs[1].set_xlabel("Epoch")
    axs[1].set_ylabel("Loss")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Loss evaluation")

    plt.show()
```

This part of code helps us to see the summary of the model and the accuracy and the loss in this model.

Now we finish the train of model so we save it by the extension of .h5 file and when we predict we will call the h5 model.

Summary of the model

```
Training sets loaded!
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 42, 11, 64)        640
_________________________________________________________________
batch_normalization (BatchNo (None, 42, 11, 64)        256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 21, 6, 64)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 19, 4, 32)         18464
_________________________________________________________________
batch_normalization_1 (Batch (None, 19, 4, 32)         128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 2, 32)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 1, 32)          4128
_________________________________________________________________
batch_normalization_2 (Batch (None, 9, 1, 32)          128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 1, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 160)               0
_________________________________________________________________
dense (Dense)                (None, 64)                10304
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 34,698
Trainable params: 34,442
Non-trainable params: 256
```

First 5 epochs

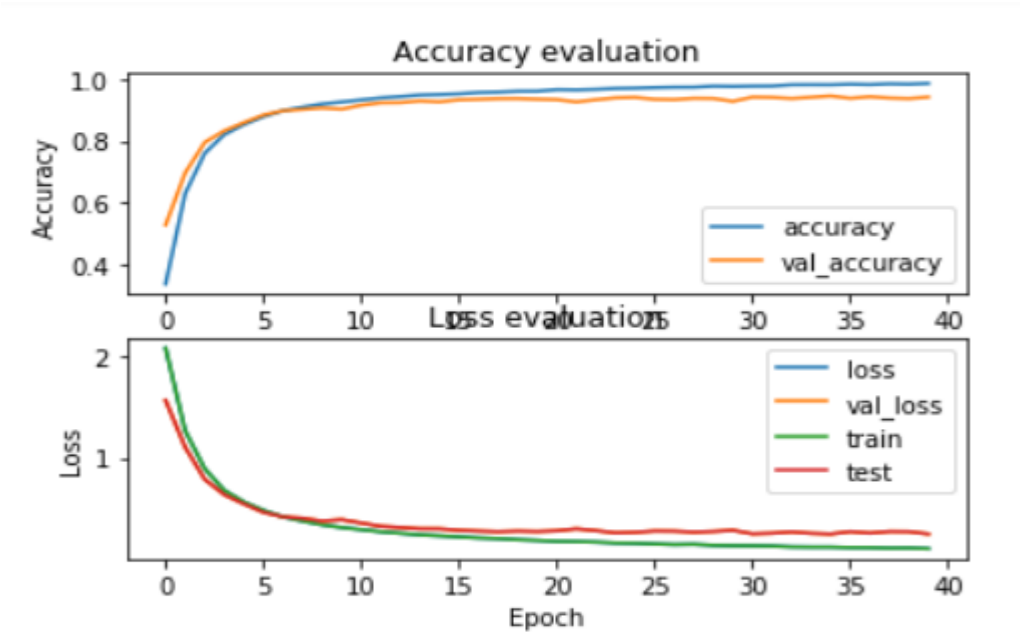
```
Epoch 1/40
432/432 [==============================] - 16s 33ms/step - loss: 2.3940 - accuracy: 0.2359 - val_loss: 1.5287 - val_accuracy:
0.5388
Epoch 2/40
432/432 [==============================] - 16s 38ms/step - loss: 1.3611 - accuracy: 0.5985 - val_loss: 1.1366 - val_accuracy:
0.6670
Epoch 3/40
432/432 [==============================] - 15s 34ms/step - loss: 0.9487 - accuracy: 0.7389 - val_loss: 0.8187 - val_accuracy:
0.7823
Epoch 4/40
432/432 [==============================] - 14s 33ms/step - loss: 0.7338 - accuracy: 0.8050 - val_loss: 0.6647 - val_accuracy:
0.8290
Epoch 5/40
432/432 [==============================] - 14s 33ms/step - loss: 0.6013 - accuracy: 0.8460 - val_loss: 0.5647 - val_accuracy:
0.8545
```

Last 5 epochs

```
Epoch 35/40
432/432 [==============================] - 19s 45ms/step - loss: 0.1232 - accuracy: 0.9829 - val_loss: 0.2863 - val_accurac
y: 0.9339
Epoch 36/40
432/432 [==============================] - 18s 43ms/step - loss: 0.1251 - accuracy: 0.9814 - val_loss: 0.2844 - val_accurac
y: 0.9316
Epoch 37/40
432/432 [==============================] - 20s 46ms/step - loss: 0.1222 - accuracy: 0.9819 - val_loss: 0.2733 - val_accurac
y: 0.9388
Epoch 38/40
432/432 [==============================] - 19s 44ms/step - loss: 0.1161 - accuracy: 0.9833 - val_loss: 0.2761 - val_accurac
y: 0.9374
Epoch 39/40
432/432 [==============================] - 20s 45ms/step - loss: 0.1116 - accuracy: 0.9846 - val_loss: 0.2736 - val_accurac
y: 0.9371
Epoch 40/40
432/432 [==============================] - 19s 44ms/step - loss: 0.1122 - accuracy: 0.9845 - val_loss: 0.2933 - val_accurac
y: 0.9322
```

Accuracy of this model is 93% but if we increase the data the accuracy will increase also.

Accuracy and loss evaluation



Recording

We need to record our voice and make a prediction for this audio file .

The requirement for this step

1- Pyaudio library
2- Wave library

Pyaudio library make the recording is very easy.

```
import pyaudio
import wave

chunk = 1024  # Record in chunks of 1024 samples
sample_format = pyaudio.paInt16  # 16 bits per sample
channels = 2
fs = 44100  # Record at 44100 samples per second
seconds = 2
filename = "output.wav"

p = pyaudio.PyAudio()  # Create an interface to PortAudio

print('Recording')

stream = p.open(format=sample_format,
                channels=channels,
                rate=fs,
                frames_per_buffer=chunk,
                input=True)

frames = []  # Initialize array to store frames

# Store data in chunks for 3 seconds
for i in range(0, int(fs / chunk * seconds)):
    data = stream.read(chunk)
    frames.append(data)

# Stop and close the stream
stream.stop_stream()
stream.close()
# Terminate the PortAudio interface
p.terminate()

print('Finished recording')

# Save the recorded data as a WAV file
wf = wave.open(filename, 'wb')
wf.setnchannels(channels)
wf.setsampwidth(p.get_sample_size(sample_format))
wf.setframerate(fs)
wf.writeframes(b''.join(frames))
wf.close()
```

We record the voice for only 2 second because in our model we use a data contain a 1 second voice for each audio file so in this we record for 2 second and take the voice in middle of this audio file to avoid the noise and the silence.

After we record the voice, we save it by output.wave file.

Prediction

Finally, we reach to the final step in this model now we can record a voice and we have a model that contain data cam make us to detect the word that we say

The requirement of this step

1- Librosa library
2- Tensorflow library
3- numpy library

Import libraries and define variables

```
import librosa
import tensorflow as tf
import numpy as np


SAVED_MODEL_PATH = "model.h5"
SAMPLES_TO_CONSIDER = 22050
```

First step is to load the mapping that we are made in the first step of this mode.

```
model = None
_mapping = [
    "eight",
    "five",
    "four",
    "nine",
    "one",
    "seven",
    "six",
    "three",
    "two",
    "zero"
]
_instance = None
```

```
# load audio file
signal, sample_rate = librosa.load(file_path)

if len(signal) >= SAMPLES_TO_CONSIDER:
    # ensure consistency of the length of the signal
    signal = signal[:SAMPLES_TO_CONSIDER]

    # extract MFCCs
    MFCCs = librosa.feature.mfcc(signal, sample_rate, n_mfcc=num_mfcc, n_fft=n_fft,
                                 hop_length=hop_length)
return MFCCs.T
```

and then we take the path of the audio file that we save in the 3rd step

and load it by librosa library and get the MFCC for it.

and make it a 3d array by putting new axis that make it a gray image by using numpy.

before that we should ensure that audio file is 1 sec and if it large than 1 second, we take the middle 1 second of this audio file.

```
# extract MFCC
MFCCs = self.preprocess(file_path)

# we need a 4-dim array to feed to the model for prediction: (# samples, # time steps, # coefficients, 1)
MFCCs = MFCCs[np.newaxis, ..., np.newaxis]

# get the predicted label
predictions = self.model.predict(MFCCs)
predicted_index = np.argmax(predictions)
predicted_keyword = self._mapping[predicted_index]
return predicted_keyword
```

then we send this MFCC data to the model so the model will compare the data which sent with the data which stored in the model.

after that the model will get the mapping word that compatible with the data of MFCC and put it in keyword variable.

```
# ensure an instance is created only the first time the factory function is called
if _Keyword_Spotting_Service._instance is None:
    _Keyword_Spotting_Service._instance = _Keyword_Spotting_Service()
    _Keyword_Spotting_Service.model = tf.keras.models.load_model(SAVED_MODEL_PATH)
return _Keyword_Spotting_Service._instance
```

in the end we write the word which the model predicted in a file because if anyone need to use this word as a speech command service he will read it from the file and do what he want.

```
# make a prediction
keyword = kss.predict("output.wav")
print(keyword)
f = open("data.txt", "w")
f.write(keyword)
f.close()
```

This is the offline model if we haven't a network.

Online Model

if you can provide a network in your application you can use a google assistant model by using this code

```
import speech_recognition as sr #3.8..1
r = sr.Recognizer()
with sr.Microphone() as M :
    print("speak anything")
    audio = r.listen(M)

    try :
         print("You said " + r.recognize_google(audio))
    except:
        print("sorry")
```

You need to install the speech_recognition library. then this code is recording your voice and send it to google assistant model and this model will know the whole words that you talk and send it back in a string form.

This is the online solution for speech recognition problem.

Version of Libraries

Offline Model:

tensorflow	2.4.1
pyaudio	0.2.11
numpy	1.15.1
matplotlib	2.2.3
keras	2.4.3

Online Model:

speech_recognition	3.8..1

Conclusion

The voice waves are very difficult to recognise so we need to know the behaviour of the sound wave and draw it but if we use the time domain we will have a problem because all wave are very similar so we convert this waves to the MFCC form that mean we are draw a image for this waves by the frequency which this waves contain and every frequency have a specific colour.

we need a python 3.7 or above to make a offline model if we haven't a network and we need some version of libraries which help us to make this model.

CNN is the most powerful of AI. which mean we can take some features of the image of the sound wave and make it a input of this CNN model and model will store the results and when we record a new audio file the model will convert it to the MFCC form and compare this result by the results which stored in the model and the nearest result will be the output . from this we should know the results which stored in the model did stored in neurons and the output of detection will be one of these neurons.

If we need to improve the accuracy of our model, we should collect more data to make a model train well than previous.

If we have a network, we can use a google assistant model (Online model) by record our voice and send the file to google assistant and it will return a sentence of what we said.

.....

Chapter (9): Future Work – References

- In object detection part, we could modify a model of our smart wheelchair that benefits from some trained classes such as “Doors and Stairs” to perform an advanced task such as open doors with an robotic arm attached to this chair.
- Also, we could add belt to wheels such tanks had to make this wheelchair climbing up and downstairs.
- We could implement a smart assistant or an advanced chatbot on a screen attached to our chair which gives recommendations and advices to its user.

References

- Intelligent Robotics and Autonomous Agents series, Roland Siegwart, Illah Reza Nourbakhsh, Davide Scaramuzza - Introduction to Autonomous Mobile Robots.
- ROS Programming: Building Powerful Robots.
- Coursera - Deep Learning Specialization.
- Coursera - Python for Everybody Specialization.
- Effective Robotics Programming with ROS.
- Websites such as: ROS Wiki, OpenCV, Youtube Playlists, etc.