The Basics of making a ROBOT

Instantiating AMR Development @ UST-Global

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI

K.K. Birla Goa Campus

October 2022

UST GLOBAL

Trivandrum

Innovation Labs

Prepared for:

Dr. Sindhu S.

Prepared by:

Prateek Nanhorya

In partial fulfillment of the requirements of PS2, Sem1, 2022-23

Acknowledgements

I would like to thank my mentors **Rafi S.M.**, **Rajendra Singh** for their invaluble inputs until midsem, while working on this project and finally in the preparation of this report. Besides my mentors, I am grateful to the BITS faculty Prof./Dr. Sindhu S. as well as the PS Division for providing me opportunity to work on the present project that helped me to widen my knowledge from various perspectives and increased my understanding of how Corporate companies work. I also owe my sincere thanks to Gopika Bindu from HR Team at UST Global for approving my internship.

Abstract

Robotics is a broad, fascinating engineering field aimed to develop devices capable of assisting humans in wide variety of tasks. Robots can be found useful in multiple instances of this dynamic world. An integration of conception, design, mechanical operation makes an robot intelligent. An intelligent robot can be deployed in Army, Space, Hospitals, Warehouses, and pretty much everywhere. A robotic dependency can be a life changer for the disabled. With the maturing of industry 4.0, efficiency and creativity is becoming the key motive in every sector. Also, autonomy becomes the backbone of most of the upcoming developments.

The use of Automotive Mobile Robots (AMR) is a historical idea and is used in multiple industries now-a-days. Amazon warehouses use AMR to move, arrange, load, unload packages and cargo efficiently and effectively using well known navigation algorithms. But still these use guide lanes/points to move through the field. Handling any dynamic obstacle is out of scope off these robots to handle. These limitations make them unfit to use in some other warehouse with dissimilar environments.

The paper focuses on broad overview of development of AMR using ROS2, and a basic idea of how to overcome these limitations.

Table of Contents

Table of Figures

Chapter 1: Introduction

1.1. What is Robotics?

Robotics is a popular engineering branch with the integration of mechanical, electrical and algorithmic concepts. In the modernization and development of today's industrial era, automation is playing a vital role. On one hand, software automation techniques have breached a new level, hardware automation is not much behind. The integration of hardware with new algorithms and software ideas give way to new development which are far refined. Modern designs give efficient, aesthetic looks to a device. Robotics platform is a huge field which requires deep knowledge of multiple domains. Often the implementation an idea in Robotics requires a team of multiple domain experts.

Prerequisites

Design of a robot is a key component of making a robot. CAD designing in freeCAD is suitable for the purpose. For analysis, COMSOL provided the best interface to develop the understanding.

Talking about the code, programming in c++ provides the fastest way to communicate to hardware. Python on the other hand provides us with the multiple libraries best for implementation probably because of the simplicity of the language. Knowledge and proficiency in these two languages can create a strong base in pursuing the field.

1.2. Motivation

Having a mechanical background, kinematics and mechanics had always been my field of absolute interest. The idea of automating the mechanical movements so as to achieve a purpose along with increase in efficiency, preciseness, strength fascinated me the most.

1.3. Goal

Understanding and becoming proficient in implementing c++ algorithms in robot development. Developing a full-fledged robot using ROS2-foxy.

1.4. Introduction to AMR

Automotive Mobile Robots (AMR) can be deployed for multiple applications. One such application is moving packages and cargo within a warehouse which can improve workflow and productivity. AMR are already deployed in some huge Amazon Warehouses and uses guide lanes for guided movement. The difference in the objective of this paper is to develop AMR for dynamic warehouse environments.

Physical conditions like temperature, terrain, moisture, etc. can vary in different warehouses. Keeping these things in mind AMR development at UST proposes the following objectives :

1. Avoiding use of guide lanes to make it compatible with different terrains.
2. Using an efficient path planning algorithm RRT*.

Chapter 2: Software and simulation

2.1. Basic Requirements

Robot Operating System (ROS) is a set of libraries, tools and algorithms for building robot applications. It is an open source tool which provides a systematic and convenient way to design the robot software. It is also termed as the middle-ware of Robotics. ROS also provides with simulation tools and libraries so as to keenly introspect the bot before deploying in physical environments.

In this paper I would be using ROS2-Foxy Fitzroy version of ROS2. Although this is not the latest version, we used this because of the availability of proper documentation and examples to aid us throughout the development.

Linux based open source environments are considered best for ROS based robotics software development as they provide flexibility by giving root access to the system and code. The operating system Ubuntu 20.04-Focal Fossa supports the aforementioned ROS version.

ROS is a vast development platform whose proper documentation as well as tutorials can be found at (2).

RViz

RViz is a software tool used to give a visualization of the robot pose and velocity by setting up transformations (2.3.1). It is also used to publish the information published on topics (used to store sensor data) by different sensors. For e.g.: /map topic is used to store the global costmap of the perceived surroundings.

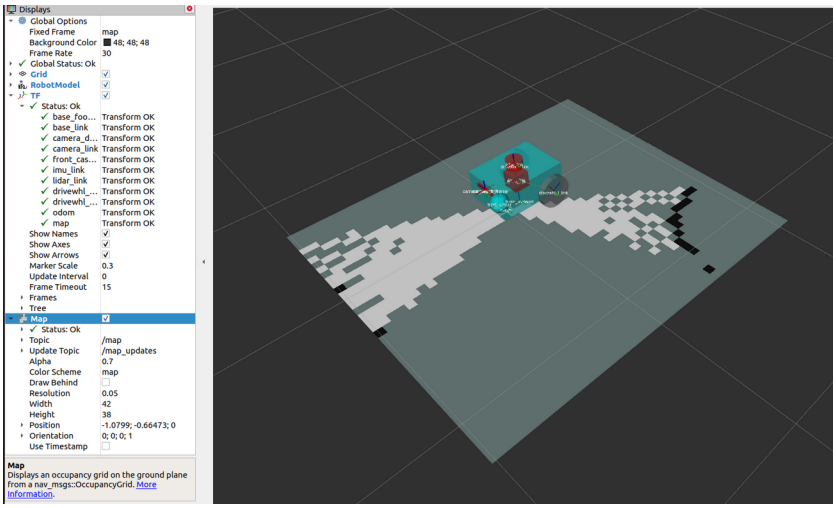


Figure 1: RViz window with the visual of map topic

Gazebo

Gazebo is a stand-alone application which can be used with ROS using *gazebo_ros* packages. The primary objective of using gazebo is to simulate the physical environment in the robot vicinity so as to get desired output from the sensors. It is a powerful tool capable of simulating physical environment with simple 3D shapes as well as complex warehouse environment.

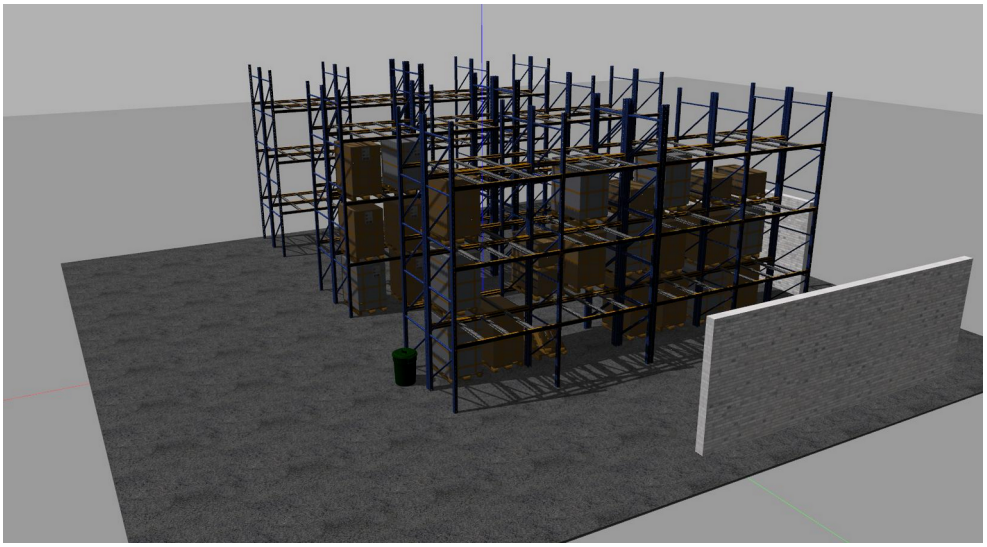


Figure 2: A gazebo world showing a warehouse

2.2. CAD Modeling

Modeling the basic design can useful to begin the development with. For the cad model of AMR, Fusion 360 is used. Parts and joints are made separately and are imported and assembled to achieve the final product. Major parts include Mecanum wheels, screws, L-shaped joints, Truss, Motors, Frame.

Next, a proper stress-strain analysis is done on the model with varying load, differing load points and areas to determine the deflection. Factor of safety measure is set to analyze the peak load. The mentioned AMR model supports around 100Kg load as per the software analysis.

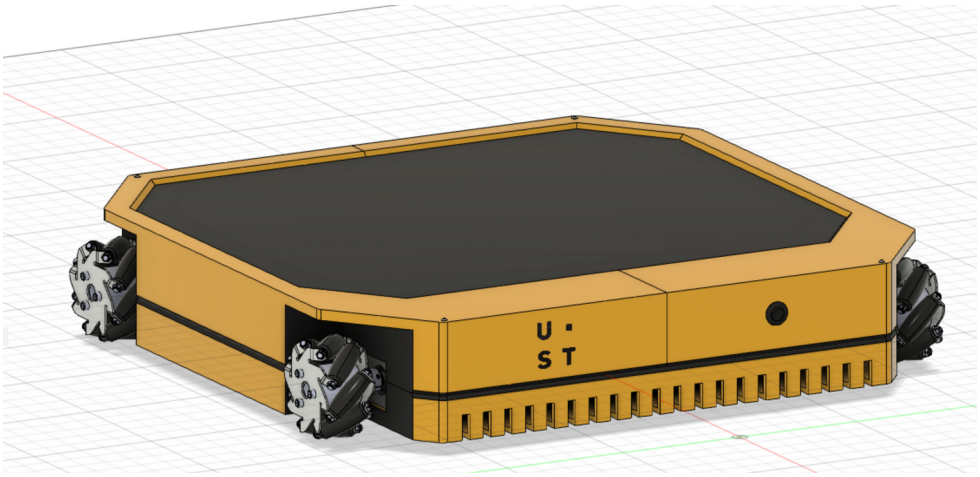


Figure 3: CAD model for AMR

2.3. Creating URDF file

The following documentation steps can be found in (2) Nav2: *First Time Robot Setup Guide*. A simple overview to with the motive of understanding the working is the objective of this report.

URDF(Unified Robot Description Format) is a file format for specifying the geometry and organization of Robot in ROS. It follows XML style format. One can model transformations, set up Odom of a robot and establish sensors.

2.3.1. Applying Transformations

Transforms allow Nav2 to interpret information coming from various sensors and odometry by transforming them to the desired coordinate frame. It refers to separate coordinate axis frame for each movable joint, sensor.

Generally CAD file mentioned before is converted to URDF file format using converer plugins in software. Make a second check on the URDF file to ensure the authenticity of the code.

For Nav2 to function, three primary transformations are necessary:

1. **map** -› **odom**
2. **odom** -› **base_link**
3. **base_link** -› **base_laser**

MAP refers to the physical surroundings and environment present.

ODOM refers to the pose and velocity related inputs from odometry sensors such as wheel encoders.

base_link is a point on robot which corresponds to point representation of robot while navigating.

base_laser is the coordinate frame of the sensing components.

```
<?xml version="1.0"?>
<robot name="sam_bot" xmlns:xacro="http://ros.org/wiki/xacro"

 <!-- Define robot constants -->
  <xacro:property name="base_width" value="0.31"/>
  <xacro:property name="base_length" value="0.42"/>
  <xacro:property name="base_height" value="0.18"/>

  <xacro:property name="wheel_radius" value="0.10"/>
  <xacro:property name="wheel_width" value="0.04"/>
  <xacro:property name="wheel_ygap" value="0.025"/>
  <xacro:property name="wheel_zoff" value="0.05"/>
  <xacro:property name="wheel_xoff" value="0.12"/>

  <xacro:property name="caster_xoff" value="0.14"/>

  <!-- Define intertial property macros  -->
  <xacro:macro name="box_inertia" params="m w h d">
    <inertial>
      <origin xyz="0 0 0" rpy="${pi/2} 0 ${pi/2}"/>
      <mass value="${m}"/>
      <inertia ixx="${(m/12) * (h*h + d*d)}" ixy="0.0" ixz="0
    </inertial>
  </xacro:macro>

  <xacro:macro name="cylinder_inertia" params="m r h">
    <inertial>
      <origin xyz="0 0 0" rpy="${pi/2} 0 0" />
      <mass value="${m}"/>
      <inertia ixx="${(m/12) * (3*r*r + h*h)}" ixy = "0" ixz
    </inertial>
  </xacro:macro>
```

Figure 4: Instance of URDF file

Map odom transformation is provided by a different ROS package dealing with localization and mapping such as AMCL.

2.3.2. Setting up Odometry

The odometry system is responsible to determining the pose and velocity of the robot accurately based on its motion. Odometry information can be retrieved from numerous sensors such as IMU, Lidar, Radar, VIO and wheel encoders.

Some odometry sensors can be used to counter the negative effects produced by other. For instance, IMU sensor readings drift over time whereas encoder values drift over distance. The information published is less accurate over distance but still can be used to navigate the robot to its immediate vicinity.

2.3.3. Setting up sensors

There are a multitude of sensors that can be used in robotics for the purpose of perceiving the environment. The obtained information is used to build and maintain the map of its surroundings and environment. Sensors also aid the robot to see obstacles around it. An accurate pose and velocity of robot can be determined by merging the output from these sensors and odometry outputs. Such information gains become necessary to to safely and efficiently navigate a robot through a dynamic environment.

Some commonly used sensors include lidar, radar, RGB camera, depth camera, IMU, and

GPS. ROS provides certain packages to communicate with these sensors. Some common sensor interfaces are stored in *sensor_msgs, radar_msgs, vision_msgs* package. Including sensor in URDF file with proper transformations make the simulation easier.

After setting up the sensors, one can simulate the environment using gazebo and see the sensors working and publishing to their respective topics.

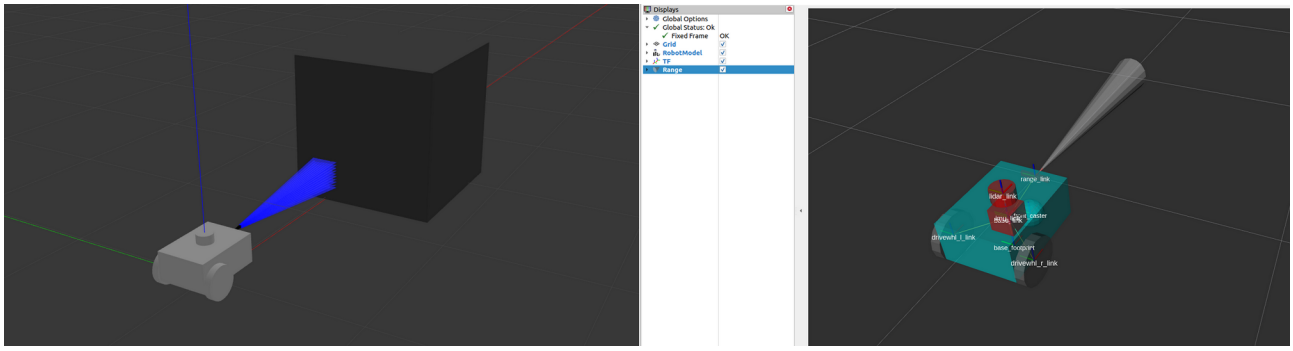


*Figure 5: **Left**: Gazebo world simulating a block, **Right**: RViz producing the visual of sensor_msg/Range topic*

2.4. Setting up footprint

It is often convenient to assume the robot as a point object while implementing navigation algorithms. To accomplish this task, it becomes necessary to set footprint outlines around the robot to avoid collisions during planning. The algorithms involved and costmap further ensures that the obstacles are avoided.

NAV2 provides with a set of custom navigation algorithms and helps to set up layered costmaps. For AMR, we make a Nav2 Parameters file containing two different costmaps namely *global_costmap* which is used primarily for long term planning over the whole

map, and *local_costmap* for short term planning and collision avoidance. The output from desired sensors is used to make layers in costmap. Some of the basic layers of a costmap include:

1. **Static Layer**

   It represents the static portions of the map, like those produced by SLAM.

2. **Obstacle Layer**

   It tracks and updates obstacles as detected by the sensors. *Voxel layer* accomplish the same in 3-D.

3. **Inflation Layer**

   Inflation layer is an optimization to the map which adds cost values around the lethal obstacles. The cost is generally kept in such a way that it reduces exponentially with distance from obstacle. It represents the configuration space of the robot.

Setting up footprint thus becomes a crucial part in implementing navigation algorithms by setting up desired costmap.

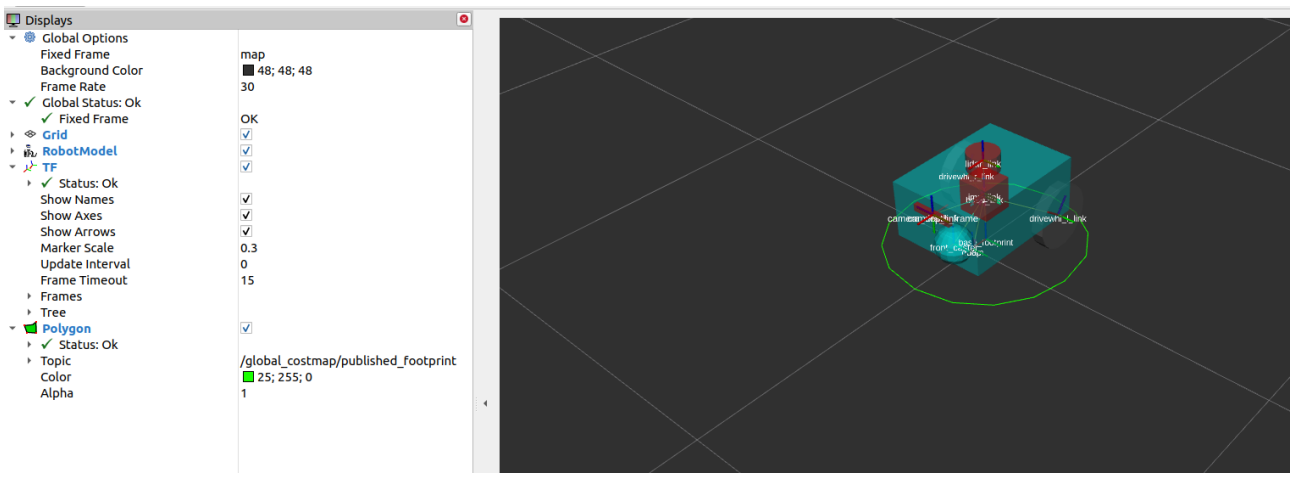


Figure 6: Circular footprint example

2.5. Setting up Navigation plugins

Nav2 uses navigation algorithms which are implemented through the use off plugins running on ROS action servers. Planner and the controller server become the heart of the Navigation stack. Such servers are used to implement one or more algorithm plugins which depends on specific action, state of robot or the environment the robot is deployed in.

Planner Server: It is responsible for generating a robot's path given start and goal. Some already available plugins for planner server in Nav2 are:

Plugin Name	Supported Robot Types
NavFn Planner	Circular Differential, Circular Omnidirectional
Smac Planner 2D	
Theta Star Planner	
Smac Hybrid-A* Planner	Non-circular or Circular Ackermann, Non-circular or Circular Legged
Smac Lattice Planner	Non-circular Differential, Non-circular Omnidirectional

Controller Server: It is responsible for generating appropriate control effects which aid the robot to steer in the local environment. Some already available plugins for controller server in Nav2 are:

Plugin Name	Supported Robot Types	Task
DWB Controller	Differential, Omnidirectional	Dynamic obstacle avoidance
TEB Controller	Differential, Omnidirectional, Ackermann, Legged	
RPP Controller	Differential, Ackermann, Legged	Exact path following

In case of AMR, these servers are not only used for finding path and following it, but also avoiding dynamic obstacles and even charging at docking stations.

Chapter 3: Hardware Interfacing

Hardware interfacing make another half of robotics. It is required to make the communication possible of sensors and surroundings with the software nodes of ROS2.

3.1. Mecanum Wheels

Mecanum wheels are based on a concept which helps achieving traction in one direction while allowing passive motion in another. Due to this the turning radius is greatly reduced which allows flexibility in congested environments. Mecanum wheels consists of a number of rollers (eight in AMR per wheel) around the circumference. These rollers are oriented at a 45° angle.

Figure 7: Mecanum Wheel

3.1.1. Encoder and Hall-effect sensor

Wheel **encoders** are located directly behind each motor. It is used to determine the velocity and distance rotated by the wheel. Encoders consist of a magnetic wheel with magnetic poles distributed alternatively. Each encoder counts the number of times the wheel has rotated by rotating by certain degrees with one wheel revolution.

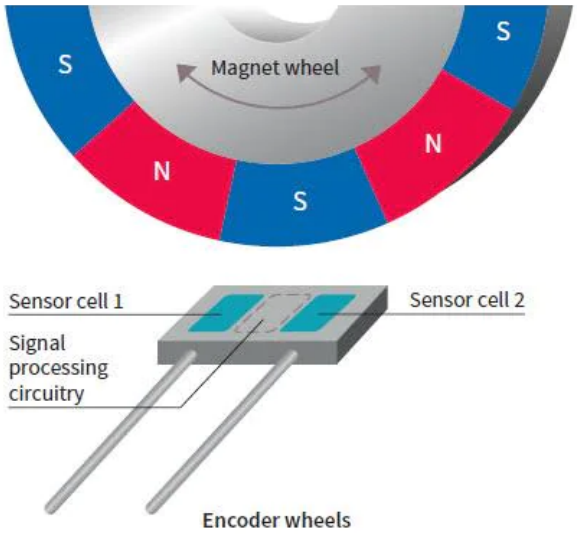
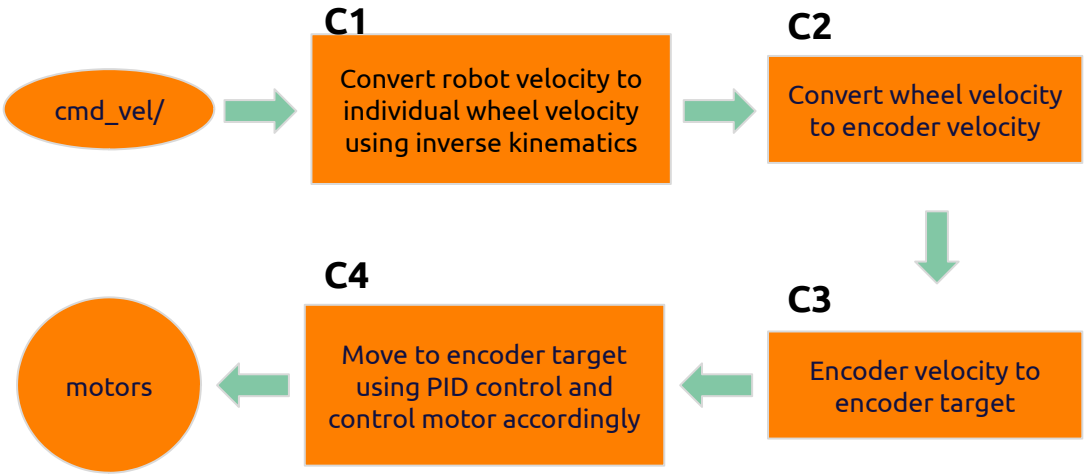


Figure 8: Wheel encoder magnet and Hall-effect Sensor

A **Hall Effect Sensor** can be used to measure the change in magnetic field thus determining the

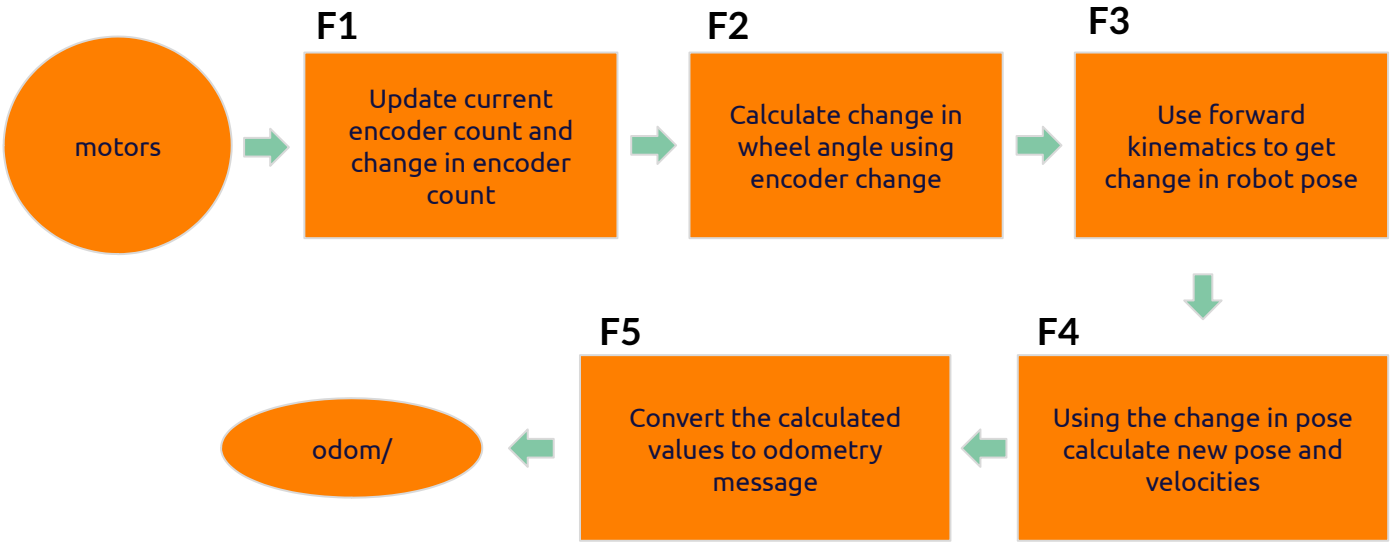
3.2. Command Flow

The establishment of proper communication between the ROS Nodes and Motors/Sensors follows a procedure shown below.



3.3. Feedback Flow

The odometry feedback helps us to determine the position and velocity of the robot. The controlled flow of feedback is shown below.



3.4. Raspberry Pi

To use ROS or any other software on the Hardware, we use Raspberry Pi. Raspberry Pi is a series of single-board computers with specifications enough to run ROS on AMR.

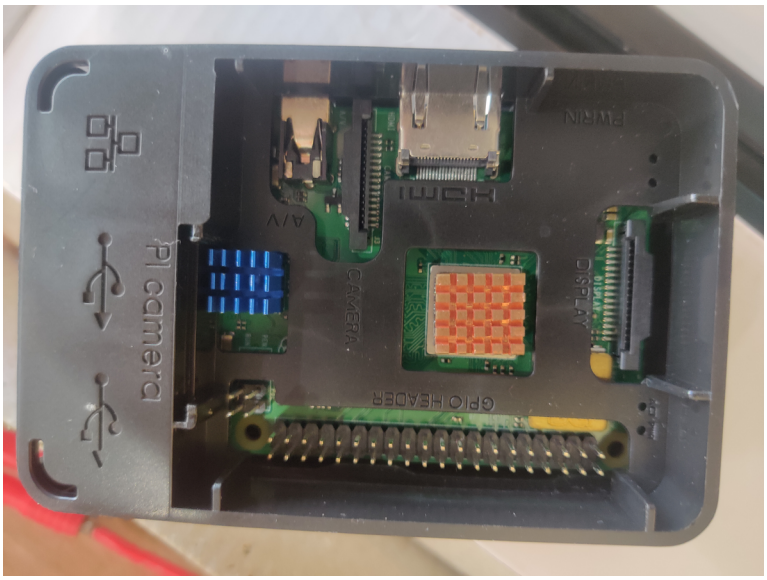


Figure 9: Raspberry Pi 3B+ being used in AMR development

Raspberry Pi 3B+ is used for GPIO testing and hardware's working condition. The testing by running ROS on ubuntu 20.04 is in progress. Some specifications of Raspberry Pi 3B+ is listed as follows:

1. Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
2. 1GB LPDDR2 SDRAM

3. 2.4GHz and 5GHz wireless LAN
4. Bluetooth 4.2, BLE
5. Gigabit Ethernet over USB 2.0
6. Extended 40-pin GPIO header
7. Full-size HDMI
8. 4 USB 2.0 ports
9. Micro SD port for loading your operating system and storing data
10. CSI camera port
11. DSI display port

3.5. Full connection

The below figure shows how a single motor of mecanum wheel is connected to Raspberry Pi.

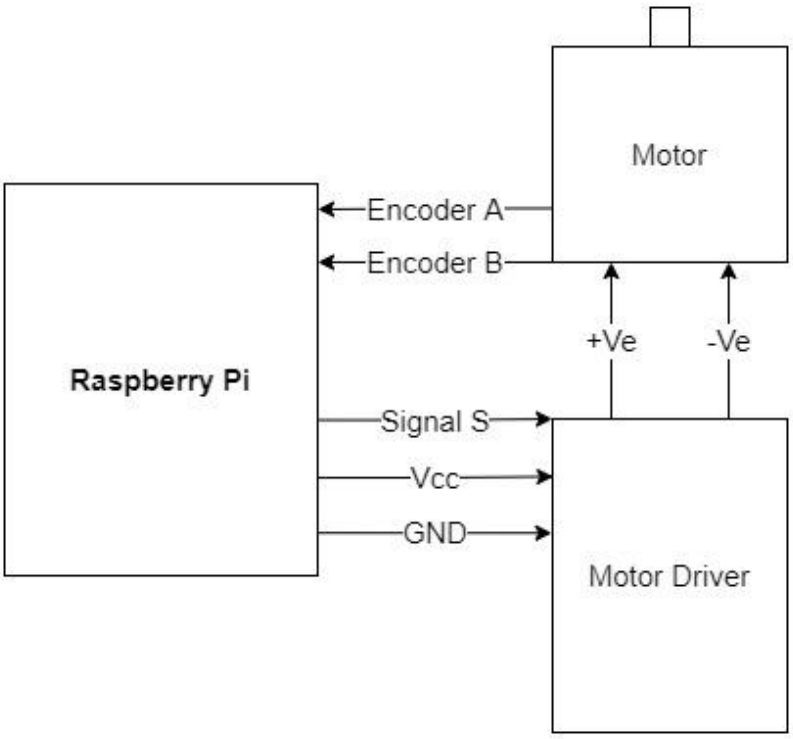


Figure 10: Connection of a single motor to RPi

Motor Drivers are responsible for providing the required voltage to the motors according to the instructions given by the processing unit through GPIO pin. Motor Drivers get direct power from battery source. The feedback from the motors is fed back directly to the raspberry pi for further processing and corrections.

The full connection of four wheels of AMR is shown below. Raspberry pi is connected to two motor drivers and each motor driver powers two wheels.

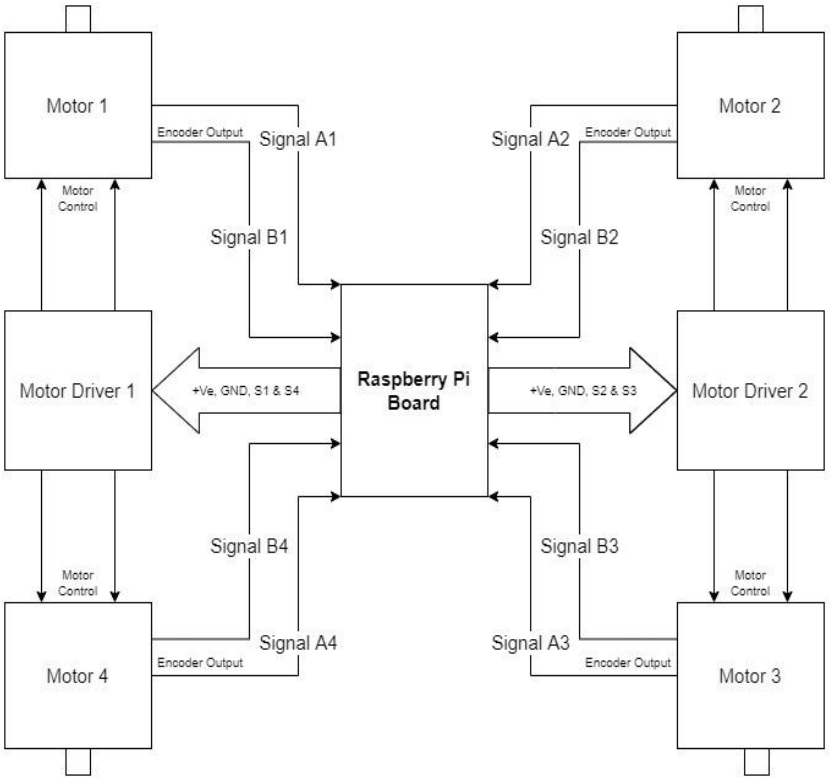


Figure 11: Full connection on board

Chapter 5: References

1. ROS2 Documentation: https://docs.ros.org/en/foxy/index.html
2. Nav2-First Time Robot Setup Guide: https://navigation.ros.org/setup_guides/index.html
3. Layered Costmap ROS: http://wiki.ros.org/costmap_2d/layered
4. Raspberry Pi specifications: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/