**HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**BACHELOR THESIS PROJECT**

**Serving Mobile Robot For Restaurant**

**Firmware Design**

**LY DUC TRUNG**

trung.ld181930@sis.hust.edu.vn

**Ngành Kỹ thuật điện**

**Chuyên ngành Hệ thống điện**

|  |  |
| --- | --- |
| **Supervisor:** | Le Minh Thuy, Ph. D  Chữ ký của GVHD |
| **Department:** | Instrumentation and industrial informatics |
| **School:** | School of Electrical & Electronics Engineering |
| **HA NOI, 08/2022** | |

**ĐỀ TÀI TỐT NGHIỆP**

Biểu mẫu của Đề tài/khóa luận tốt nghiệp theo qui định của viện, tuy nhiên cần đảm bảo giáo viên giao đề tài ký và ghi rõ họ và tên.

Trường hợp có 2 giáo viên hướng dẫn thì sẽ cùng ký tên.

Giáo viên hướng dẫn

Ký và ghi rõ họ tên

**Lời cảm ơn**

Đây là mục tùy chọn, nên viết phần cảm ơn ngắn gọn, tránh dùng các từ sáo rỗng, giới hạn trong khoảng 100-150 từ.

**Tóm tắt nội dung đồ án**

Tóm tắt nội dung của đồ án tốt nghiệp trong khoảng tối đa 300 chữ. Phần tóm tắt cần nêu được các ý: vấn đề cần thực hiện; phương pháp thực hiện; công cụ sử dụng (phần mềm, phần cứng…); kết quả của đồ án có phù hợp với các vấn đề đã đặt ra hay không; tính thực tế của đồ án, định hướng phát triển mở rộng của đồ án (nếu có); các kiến thức và kỹ năng mà sinh viên đã đạt được.

Sinh viên thực hiện

Ký và ghi rõ họ tên

**TABLE OF CONTENTS**

[CHAPTER 1. GENERAL INTRODUCTION ABOUT MOBILE ROBOT 1](#_Toc109659275)

[1.1 Overall information about mobile robots 1](#_Toc109659276)

[1.1.1 Mobile Robot Introduction 1](#_Toc109659277)

[1.1.2 Mobile robot in service delivery 2](#_Toc109659278)

[1.2 Serving mobile robot design planning 6](#_Toc109659279)

[1.2.1 Design planning of project 6](#_Toc109659280)

[1.2.2 Scopes of work 7](#_Toc109659281)

[CHAPTER 2. GENERAL HARDWARE DESIGN FOR SERVING MOBILE ROBOT 8](#_Toc109659282)

[2.1 Hardware components 8](#_Toc109659283)

[2.1.1 Raspberry Pi 4 8](#_Toc109659284)

[2.1.2 ESP32 microcontroller 8](#_Toc109659285)

[2.1.3 Motors 8](#_Toc109659286)

[2.1.4 Motor driver 8](#_Toc109659287)

[2.1.5 Encoders 8](#_Toc109659288)

[2.1.6 Ultrasonic sensors 8](#_Toc109659289)

[2.1.7 Lidar 8](#_Toc109659290)

[2.1.8 Inertial measurement unit 8](#_Toc109659291)

[2.1.9 OLED display 8](#_Toc109659292)

[2.1.10 Infrared sensor 9](#_Toc109659293)

[2.1.11 Voltage sensor 9](#_Toc109659294)

[2.2 Hardware structure design 9](#_Toc109659295)

[CHAPTER 3. ROS 2 FRAMEWORK FOR DEVELOPING ROBOTS 9](#_Toc109659296)

[3.1 Robot software platforms 9](#_Toc109659297)

[3.2 The basis for selecting ROS 10](#_Toc109659298)

[3.2.1 The program can be reused 10](#_Toc109659299)

[3.2.2 ROS is a communication-based program 10](#_Toc109659300)

[3.2.3 The support of development tools 10](#_Toc109659301)

[3.2.4 The active community 11](#_Toc109659302)

[3.2.5 The formation of the ecosystem 11](#_Toc109659303)

[3.3 The change of the ROS version 11](#_Toc109659304)

[3.4 ROS 2 introduction 14](#_Toc109659305)

[3.4.1 Communication 15](#_Toc109659306)

[3.4.2 Visualization 15](#_Toc109659307)

[3.4.3 Simulation 15](#_Toc109659308)

[CHAPTER 4. FIRMWARE DESIGN FOR SERVING MOBILE ROBOT 16](#_Toc109659309)

[4.1 Raspberry Pi 4B configuration 16](#_Toc109659310)

[4.1.1 Ubuntu version 16](#_Toc109659311)

[4.1.2 zRAM 17](#_Toc109659312)

[4.1.3 ROS 2 Galactic 18](#_Toc109659313)

[4.1.4 Multiple I2C buses 18](#_Toc109659314)

[4.2 Serving mobile robot firmware architecture 18](#_Toc109659315)

[4.3 Firmware for motor driver block 19](#_Toc109659316)

[4.3.1 Choosing sample time of encoders 19](#_Toc109659317)

[4.3.2 Firmware for ESP32 microcontroller 20](#_Toc109659318)

[4.3.3 Firmware for Raspberry Pi 4 27](#_Toc109659319)

[4.4 Firmware for voltage sensor 31](#_Toc109659320)

[4.4.1 Firmware for ESP32 microcontroller 31](#_Toc109659321)

[4.5 Firmware for ultrasonic sensors 31](#_Toc109659322)

[4.6 Firmware for lidar 35](#_Toc109659323)

[4.7 Firmware for inertial measurement unit 38](#_Toc109659324)

[4.8 Firmware for OLED display 39](#_Toc109659325)

[4.9 Firmware for infrared sensor 42](#_Toc109659326)

[CHAPTER 5. ALGORITHM EXPLANATION FOR FIRMWARE 43](#_Toc109659327)

[5.1 PID controller 43](#_Toc109659328)

[5.2 Fuzzy logic library 43](#_Toc109659329)

[5.3 Kinematic Model for calculating Odometry data 43](#_Toc109659330)

[5.4 Kalman filter 43](#_Toc109659331)

[5.5 Median filter 43](#_Toc109659332)

[CHAPTER 6. EXPERIMENTAL RESULTS 43](#_Toc109659333)

[6.1 Comparison of PID controller and Fuzzy PID controller 43](#_Toc109659334)

[6.2 Obstacle avoidance using an array of ultrasonic sensors test cases 43](#_Toc109659335)

[6.3 Restaurant serving test cases 43](#_Toc109659336)

[6.4 Serving mobile robot design result ? 43](#_Toc109659337)

[REFERENCES 44](#_Toc109659338)

[APPENDIX 45](#_Toc109659339)

**DANH MỤC HÌNH VẼ**

[Figure 1: Scheduling latency in an idle environment. (a) ROS 1.0 (b) ROS 2.0 [3] 12](#_Toc109658196)

[Figure 2: Scheduling latency in a stressed environment (a) ROS 1.0 (b) ROS 2.0 [3] 13](#_Toc109658197)

[Figure 3: Maximum communication latency given the data size in environments with network traffic 14](#_Toc109658198)

[Figure 4: System requirements from official website for ROS 2 Galactic Geochelone 15](#_Toc109658199)

[Figure 5: Ubuntu 20.04.4 LTS on Raspberry Pi 4 16](#_Toc109658200)

[Figure 6: Initial amount of RAM on Raspberry Pi 4 16](#_Toc109658201)

[Figure 7: No-load performance of RAM on Raspberry Pi 4 16](#_Toc109658202)

[Figure 8: Firmware architecture of the system 17](#_Toc109658203)

[Figure 9: Motor driver block 18](#_Toc109658204)

[Figure 10: Flowchart of setup() function for EPS32 firmware 19](#_Toc109658205)

[Figure 11: Flowchart of loop() function for EPS32 firmware 20](#_Toc109658206)

[Figure 12: JSON form of receiving data of ESP32 microcontroller 21](#_Toc109658207)

[Figure 13: Commonly­used Frequencies and Resolutions 21](#_Toc109658208)

[Figure 14: Flowchart of driveMotors() function for EPS32 firmware 22](#_Toc109658209)

[Figure 15: H-SW Control Function of TB6612FNG motor driver 22](#_Toc109658210)

[Figure 16: Solution for instantaneous operating of motors 23](#_Toc109658211)

[Figure 17: Block of code for counting pulses in ESP32Encoder C++ library 23](#_Toc109658212)

[Figure 18: ESP32 Firmware - RPMCalculator class 24](#_Toc109658213)

[Figure 19: JSON form of sending data of ESP32 microcontroller 25](#_Toc109658214)

[Figure 20: Transmission speed and Real transmission speed of serial port 25](#_Toc109658215)

[Figure 21: Flowchart of Raspberry Pi 4 for motor driver block 26](#_Toc109658216)

[Figure 22: Raspberry Pi 4 Firmware – PoseCalculator class 27](#_Toc109658217)

[Figure 23: Definition of Pose according to ROS 2 28](#_Toc109658218)

[Figure 24: Data type of position 29](#_Toc109658219)

[Figure 25: Data type of orientation 29](#_Toc109658220)

[Figure 42: Voltage sensor block 30](#_Toc109658221)

[Figure 26: Raspberry Pi 4 Firmware – Sonar class 31](#_Toc109658222)

[Figure 27: Defining a Sonar instance with arguments 32](#_Toc109658223)

[Figure 28: Flowchart of Raspberry Pi 4 for measuring distance of ultrasonic sensor 33](#_Toc109658224)

[Figure 29: Flowchart of Raspberry Pi 4 for sonar node 34](#_Toc109658225)

[Figure 30: Lidar sensor block 34](#_Toc109658226)

[Figure 31: Flowchart of Raspberry Pi 4 for lidar node 35](#_Toc109658227)

[Figure 32: The RPLIDAR A1 Sample Point Data Information 36](#_Toc109658228)

[Figure 33: The RPLIDAR A1 Sample Point Data Frames 36](#_Toc109658229)

[Figure 34: The Obtained Environment Map from RPLIDAR A1 Scanning 37](#_Toc109658230)

[Figure 35: Inertial measurement unit sensor block 37](#_Toc109658231)

[Figure 36: Flowchart of Raspberry Pi 4 for IMU node 37](#_Toc109658232)

[Figure 37: OLED display block 38](#_Toc109658233)

[Figure 38: Raspberry Pi 4 Firmware – Oled class 39](#_Toc109658234)

[Figure 39: Flowchart of Raspberry Pi 4 for OledNode node 40](#_Toc109658235)

[Figure 40: Infrared sensor block 41](#_Toc109658236)

[Figure 41: Flowchart of Raspberry Pi 4 for infrared sensor node 41](#_Toc109658237)

**DANH MỤC HÌNH VẼ**

[Bảng 1.1 Thống kê các thiết bị và giá thành 12](#_Toc20580109)

# GENERAL INTRODUCTION ABOUT MOBILE ROBOT

## Overall information about mobile robots

### Mobile Robot Introduction

Throughout history, the idea of mechanical automata that can resemble or replace human activities has originated in the mythologies of many cultures around the world. However, this becomes more popular in the 19th and especially 20th century. A machine with the ability to carry out a complex series of actions automatically is called robot. Mobile robot is a robot that is not attached to the environment and is able to move in a certain surrounding space.

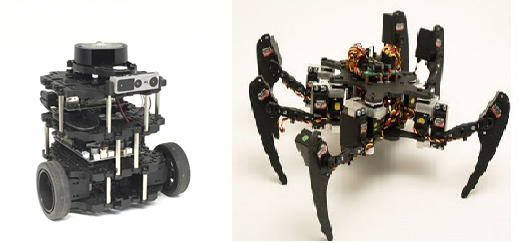
A mobile robot can be designed with an autonomous system, which can self-navigate within their environment or relies on guidance devices to follow a pre-defined route. The autonomy of the mobile robot must be guaranteed with aspects of energy and self-making decision. The level of autonomy can be defined based on the commands of human operator, from operate with desired wheel velocities to achieve desired robot longitudinal and angular velocities, follow path or trajectory to self-operate inside of the known or unknown environment.

The hardware structure of mobile robot is a combination of mechanical and electronic parts, mainly:

* Mechanical parts: rigid and moving parts (body, wheels, etc.)
* Actuators: electrical motors (DC, stepper, servomotor, etc.)
* Sensors: rotation encoders, lidar, ultrasonic sensors, accelerometer & gyroscope sensor, etc.
* Computers:  micro-controllers, embedded computer, …
* Power unit: batteries, solar panels, …

Related to the working environment, mobile robot can be classified into three principal groups:

* Ground mobile systems: They are designed to operate on the ground and included various platform such as mobile vehicles with wheels or caterpillar, legged robots or mimic some other type of animal locomotion.



* Aerial mobile systems: This group consists of systems that can fly in a certain aerial space or orbit the Earth.



* Water and under water mobile system: We can find many kinds of ships, boats, submarines, autonomous underwater vehicles, etc.



Nowadays, with the rapid development of technology, mobile robot is gradually applied to replace human activities in various fields, especially in hazardous or inaccessible environments (planets, minefields, radioactive environments, etc.). Moreover, applying robot in production can also help human to reduce labor costs, and increase productivity. The application field of mobile robot is expanded day by day namely medical service, operation support, cleaning applications, consumer good stores, military and security, etc. It can be concluded that in the near future, mobile robots can become a part of our life with a great support from household chores to discover unknown planets.

### Mobile robot in service delivery

Different from industrial application where the role of robot is to replace humans for carrying out dangerous tasks in hazardous environment, robots for service are more likely to assist humans for working. We can easily find the support of mobile robot in many aspects such as hospitality, agriculture or education. Another application of mobile robot must be mentioned is to deliver goods. The general function of robot in this application is to carry a payload from a source to its destination. With the support of mobile robot, human can focus on higher value activities. The problem of labor shortage for low-value tasks can also be solved and productivity is also increased because robots don’t need breaks or downtime.

One significant example of using robot in delivery is for restaurants where they can deliver food to the table and carry dirty dishes back to the kitchen. Many technology companies are racing to implement their own delivery robots to meet the zooming upward demand day by day. Below are some design models of serving robot in the market.

“Servi”: “Servi” is a production of Bear Robotics, a robotics and artificial intelligence company, and SoftBank Robotics Group.



“Servi” can autonomously travel and carry food when customer select a table to serve on the screen. It can also move while detecting and avoiding obstacles such as people and objects with high accuracy. “Servi” is said to improve the efficiency of the establishment and the quality of customer service. 

|  |  |
| --- | --- |
| ***Main specs*** | ***Value*** |
| Maximum Load Weight | 35kg |
| Navigation Method | SLAM |
| Sensor | LiDAR, 3D Camera |
| Communication | Wi-Fi 2.4/5 GHz |
| Working time | 8 – 12 hours/single charge |
| Maximum speed | 0.6 m/s |
| Rental fee | 738 USD/month |

Beside “Servi” robot, many other robotics company also introduced their invention of serving robot.





These robots’ design shares the same technologies which will be described in detail below:

* Sensors: To localize and navigate, several range sensors, cameras, and radars can be added. One of the most popular sensors that are used in mobile robot is a LIDAR (Light Detection And Ranging), which is normally set up on the top of the robot to achieve maximum coverage. The point cloud produced by the LIDAR sensor contains a huge amount of data about the surroundings. To distinguish, categorize, and identify the objects from the raw point cloud data, advanced data processing is used. Additional sensors like radar or ultrasonic sensors are also added to improve the preciseness about the environment. They are implemented to measure the distance and detect near obstacles around the robot. Besides that, to enable the calculation for odometry, rotation encoders are equipped with robot’s motors. Some other sensors like accelerometers, gyroscopes are also attached on the robot to measure navigation tasks.
* Localization and mapping: Many features can be used to solve these tasks. The recent modern solution becoming more and more popular is using Simultaneous Localization And Mapping method (SLAM) – the method of constructing or updating a map of an uncharted area while keep tracking robot’s position.
* Communication: Wireless technology is used by many mobile robot platforms to connect with other robots, human machine interfaces, and off-line computing resources. Numerous mobile robots are outfitted with wireless technologies including Bluetooth, Wi-Fi, Wireless LAN, and others.
  + Wi-Fi or Wireless LAN: Wireless Local Area Networks, often known as Wi-Fi or WLAN, are based on a set of specifications known as 802.11 from the Institute of Electrical and Electronics Engineers (IEEE). Wi-Fi operates mostly in the unlicensed 2.4GHz spectrum of radio frequency. It makes it possible for someone to use a wireless access point to connect to the Internet using a computer or PDA that supports wireless. A hot spot is the geographic area that one or more access points cover. Wi-Fi is increasingly often utilized for Internet access outside despite being designed for mobile devices and local area networks. ControlNet: ControlNet employs a single media link with (cheap) RG-6 coaxial cables and a bus. It is built on its own physical and data link layer. It has up to 99 nodes, a 5 Mbit/s throughput, data upload/download, and P2P communication.
  + Bluetooth: Most wireless technologies, including Bluetooth and the IrDA standard, provide users the option of bolstering their local wireless network. In the workplace, in labs, or at home, Bluetooth technology, developed by Ericsson in 1994, is utilized in place of wires. a wireless technology that uses a short radio frequency cable. (2012) IACSIT Press, Singapore, IPCSIT vol. 34, International Conference on System Engineering and Modeling (ICSEM 2012) The scientific community is moving distance to 2.4GHz bandwidth to replace unlicensed equipment. Broadband 1 MB per second speech and data exchanges are supported via Bluetooth devices, which typically have a range of around 10 meters. Parts of Bluetooth have been included into a variety of mobile devices, including mobile phones, PDAs, and other wireless equipment, because of Bluetooth's benefits, including inexpensive prices, low power consumption, and nature may be turned in different directions. Around 200 million Bluetooth devices will be sold in 2001, according to research by In-Stat/MDR and Frost & Sullivan, and one billion in 2006. As a result, Bluetooth technology is now being used for mobile robot controllers.

## Serving mobile robot design planning

### Design planning of project

The market for autonomous delivery robots is still at the beginning. One drawback limiting applications of service mobile robot is the high cost for renting or buying. Therefore, in this project, our purpose is to design a mobile robot for restaurants with the ability to self-navigate and can totally replace the role of servant to serve food or drink to customers. Modern technologies and system are integrated in the robot design, but still ensure the suitable price.

Serving mobile robot specification:

* Sensor: LiDAR, Ultrasonic sensor, IMU.
* Navigation method: SLAM.
* Communication: Wi-Fi.
* Body weight: kg.
* Maximum load weight: kg.
* Maximum speed:  m/s.

### Scopes of work

#### Analysis & decision making

This is the first step of designing a mobile robot. General plans for designing the robot structure are outlined, together with researching hardware components which is available on the market and suitable for robot using’s purpose.

#### Hardware design

After analysis to get the design planning, the hardware of robot is begun with calculation of main mechanical and electronic parts.

Scopes of work include:

* Mechanical design: Consider the kinematic model to design robot’s frame and do the calculation to select suitable actuators (DC motor).
* Electronic system design: Select and combine electronic devices: power supply, sensors, computers, actuator drive, … to complete the robot electronic system. Design the motor controller for controlling speed of the robot.

#### Firmware design

System firmware is designed to combine the data taken from peripherals and preprocess all those data using algorithms.

* Sensors: Configure multiple I2C buses in Raspberry Pi 4. Read data from sensors and create Nodes for communicating between ROS 2 and all the sensors.
* Motors: Configure communication method between ESP32 and Raspberry Pi 4 to control the motors and receive data from encoders.
* Algorithms:
  + Program PID controller in Python for driving motor.
  + Program Fuzzy Logic library in C++ for PID controller.
  + Program Kalman Filter and Median Filter in Python for sensors.
  + Program Kinematic Model for calculating Odometry data of the robot.

#### Software design

Mainly work with ROS 2 framework and focus on improving and customizing Navigation Stack for the robot, which includes tasks:

* Create 3D model of robot and restaurant environment for simulation tests. Build transform frame, which represents the position and orientation of different robot's parts corresponding to each other and to the map frame when robot moves.
* Mapping environment with SLAM.
* Localize the robot on the map based on data from odometry, imu sensor and lidar.
* Plan path around obstacles and control the robot as it follows the planned path.
* Convert sensor data into a 2d costmap representation of the robot’s surrounding world.
* Create robot’s operation flow using behavior trees.

#### Experiment & results

The result of speed controller for robot by Fuzzy PID is analyzed and compared with classic PID controller.

Several specific serving cases for robot are also implemented in real life to evaluate working performance of the robot.

# GENERAL HARDWARE DESIGN FOR SERVING MOBILE ROBOT

## Hardware components

### Raspberry Pi 4

### ESP32 microcontroller

### Motors

### Motor driver

### Encoders

### Ultrasonic sensors

### Lidar

### Inertial measurement unit

### OLED display

### Infrared sensor

### Voltage sensor

## Hardware structure design

# ROS 2 FRAMEWORK FOR DEVELOPING ROBOTS

## Robot software platforms

Platforms have recently attracted more interest in the robotics community. Hardware platforms and software platforms both have their own subcategories. A robot software platform consists of the tools needed to create robot application programs, including hardware abstraction, low-level device control, sensing, recognition, SLAM (simultaneous localization and mapping), navigation, manipulation, and package management, libraries, debugging, and development tools. Software platforms have made it possible for many individuals to contribute to the creation of robots, and robot hardware is now being created using the interface that software platforms give.

The smartphone field and the robot field are advancing along the same path. The robot software platform is in a dynamic stage where anybody may become the market leader, even if there is still more to be done compared to the smartphone operating system.



The most popular robot software systems are those that are listed below:

* ROS: Robot Operating System, Open Robotics - U.S.
* OpenRTM: National Institute of Adv. Industrial Science and Technology (AIST) – Japan.
* ERSP: Evolution Robotics Software Platform, Evolution Robotics – Europe.
* OPROS: ETRI, KIST, KITECH, Kangwon National University - South Korea.

## The basis for selecting ROS

ROS is a robot software platform which helps to reduce the developing and programming time. By entering a few standardized codes, one may quickly convert a non-ROS system to a ROS system without having to completely rewrite the system and programs. Additionally, ROS offers a variety of widely used tools and software and enables users to concentrate on the things they are interested in or would want to contribute to, which eventually cuts down on the time required for development and maintenance.

### The program can be reused

A user may concentrate on the feature they want to build and get the matching package for the other functionalities. They can also distribute the application they created so that others can utilize it. As an illustration, it is said that in order to complete their job in space, NASA employed not only programs created in-house but also ROS, which offers a variety of drivers for multi-platforms, and OROCOS, which facilitates real-time control, message communication restoration, and dependability. Another example of carefully built reusable programs is the Robotbase mentioned above.

### ROS is a communication-based program

Programs like hardware drivers for sensors and actuators and functions like detecting, identification, and operating are frequently written in a single frame in order to deliver a service. However, each program and feature are broken down into smaller bits based on its function in order to accomplish the reusability of robot software. Depending on the platform, this is referred to as componentization or modularization. Platforms have all the information required for transmitting data among nodes, which are processes that execute computing in ROS and are separated into units of minimum functionality. When communication between nodes is based on a network such that nodes are not constrained by hardware, network programming—which is extremely helpful in remote control—becomes available. The Internet of Things (IOT) also uses the idea of network linked minimum functions; hence ROS can take the role of IoT platforms. Because programs can be debugged independently when they are broken down into their smallest functions, this is very helpful for identifying flaws.

### The support of development tools

Debugging tools, 2D visualization tools (rqt, a ROS software framework that implements the different GUI tools in the form of plugins), and 3D visualization tools (RViz) are all provided by ROS and may be utilized without the need to create the essential tools for robot development. For instance, there are several times where visualizing a robot model is necessary for robot development. Users may examine the robot's model directly by matching the established message format, and they can also utilize the included 3D simulator to simulate the robot (Gazebo). The application can also take 3D distance data from the recently included Microsoft Kinect or Intel RealSense cameras and quickly convert it to a point cloud format so that it may be shown on the visualization tool.

### The active community

Due to the aforementioned roles, the traditionally largely insular academic and industrial robot communities are gradually shifting toward encouraging collaboration. Despite the fact that each person's goals may differ, many software platforms do enable collaboration. A community for open-source software platforms is at the heart of this transition. As of 2017, there were more than 5,000 ROS packages that had been produced and shared on a voluntary basis. Individual users had also contributed to more than 18,000 Wiki pages that describe these packages. Additionally, the Q&A section, which is another essential component of the community, has more than 36,000 posts, which has led to a cooperatively expanding community. In addition to reviewing the instructions, the community looks for essential robotics software components and establishes rules for them. Additionally, this is moving toward a stage when users participate to fill in the gaps in the jigsaw by brainstorming ideas for what robot software should include to enhance robotics.

### The formation of the ecosystem

According to popular belief, the ecosystem developed by software platforms like Android or iOS is what led to the aforementioned smartphone platform revolution. The robotics industry is also seeing this kind of development. Every hardware breakthrough was exploding at first, but there was no operating system to incorporate it. There have been several software platforms established, and the most prestigious platform among them, ROS, is now forming its ecosystem. It is developing an environment that everyone can be satisfied with, including consumers, application software developers, and hardware developers from the robotics industry, such as robot and sensor firms. We may expect a vibrant ecosystem in the near future, despite the fact that the beginning may still be in its infancy, given the rise in users, robot-related businesses, and associated tools and libraries, as well as the rising number of users and users.

## The change of the ROS version

While the robot and humans share the same workspace in real time, if there are any system latency-related problems, the user might sustain bodily harm. In order to ensure reliable functioning, real-time limitations must be met. Researchers have suggested a number of methods to enable ROS to operate in real-time because it does not currently carry out these requirements. Data Distributed Services (DDS) was designed as middleware for internal communication in ROS 2.0 to facilitate real-time communication. Data is sent via the Real-Time Publish-Subscribe (RTPS) protocol, multicasting, and connectionless transmission techniques like UDP/IP.

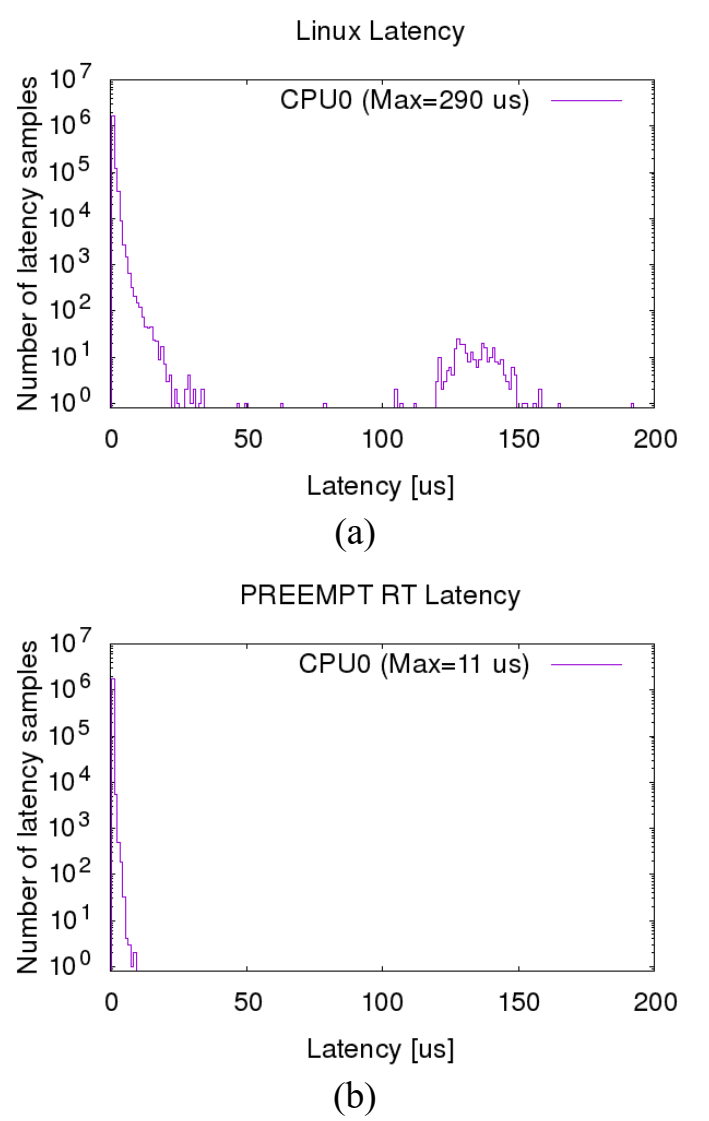


Figure : Scheduling latency in an idle environment. (a) ROS 1.0 (b) ROS 2.0 [3]

As can be observed, the recorded maximum latency, which is 290 us as opposed to ROS 2.0's 11 us, is much greater. The ROS 1.0 findings also display a wider distribution than the ROS 2.0 results, where the majority of the measured data samples are clustered below the highest value. As a result, ROS 2.0 enabled deterministic behavior and more effectively fulfilled the strict temporal requirements than ROS 1.0. Additionally, ROS 1.0 operated inconsistently since it had a wider spread than ROS 2.0. As a result, the ROS 2.0 system worked steadily and offered better real-time performance than ROS 1.0.



Figure : Scheduling latency in a stressed environment (a) ROS 1.0 (b) ROS 2.0 [3]

The greatest delay for ROS 1.0 and ROS 2.0, respectively, was estimated to be 1280 us and 26 us. The maximum latency was higher for both the ROS 1.0 and ROS 2.0 systems when compared to the findings of the performance assessments in idle situations. In this instance, the 1 ms deadline was not met by ROS 1.0. However, ROS 2.0 was still able to achieve the deadline despite with a modest increase in the maximum latency.

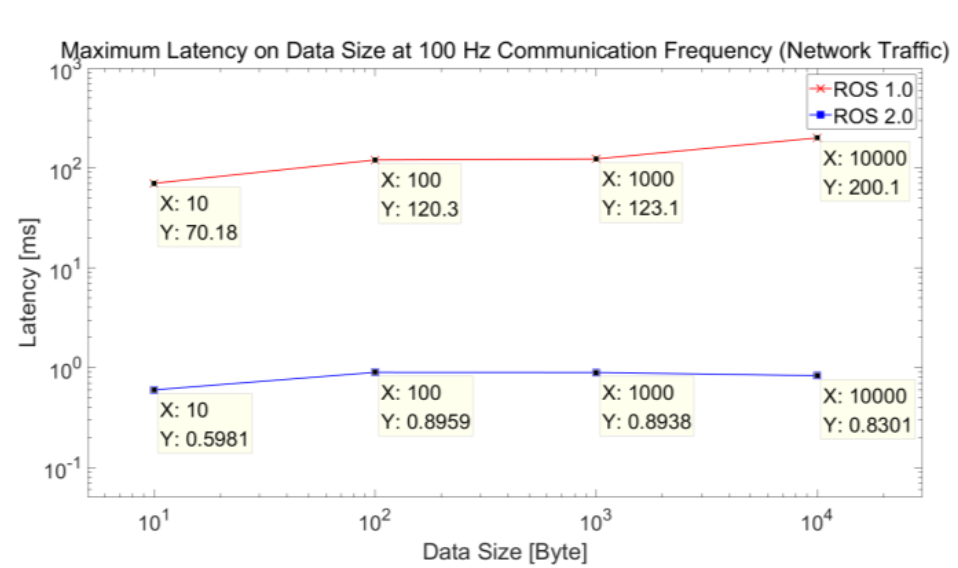


Figure : Maximum communication latency given the data size in environments with network traffic

The outcomes in a dynamic context with network traffic are shown in Fig. 3. The maximum latency was much reduced in ROS 2.0 than in ROS 1.0 in both the steady and idle conditions. While ROS 2.0 demonstrated a little improvement in performance in unreliable situations, latencies in ROS 1.0 made it challenging to transmit and receive messages at the proper time. Additionally, it did not meet the requirements for real-time communication, which makes it challenging to run the system in unsteady conditions.

According to the performance evaluation findings based on data size, ROS 1.0 was unable to meet real-time communication layer requirements and struggled to function properly in unstable environments. The fact that ROS 2.0 has a shorter latency allows it to satisfy these requirements, and because network traffic has a negligible impact on measured performance, steady system operation is achievable even in an unstable environment.

## ROS 2 introduction

The Robot Operating System 2 (ROS 2) is a collection of open-source software libraries, hardware driver framework, state-of-the-art algorithms and robust developer tools to develop a control system for robots.

ROS 2 provides us several components for building and developing a robot or a multi-robot system. Some of them are still being developed and improved, thus they are not yet available. This section will introduce the main elements of ROS 2.

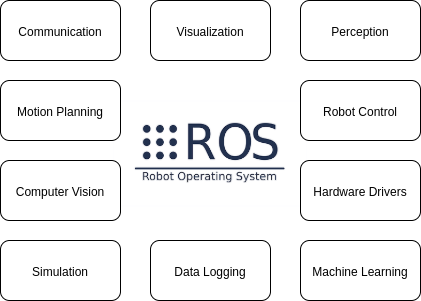


Figure : Components of ROS 2

### Communication

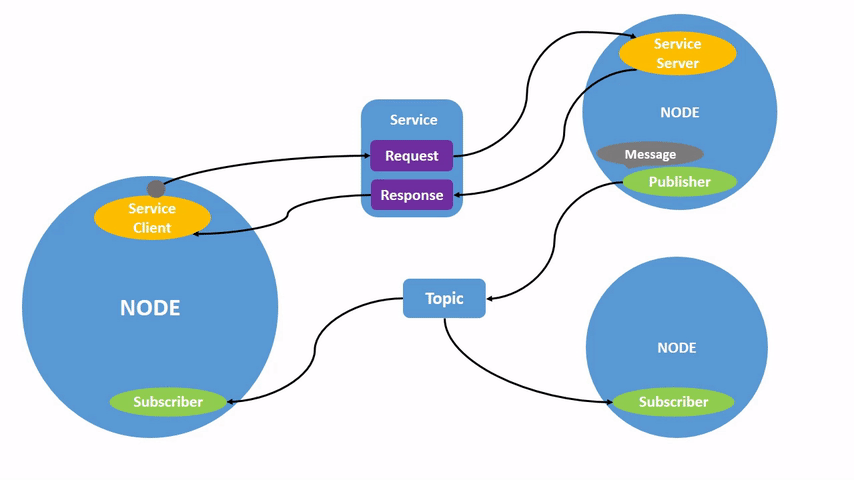


Figure : Ros 2 Graph

The core components of ROS 2's distributed architecture are the node, topic, message structure, and discovery. In the language of ROS 2, this structure is known as a graph.

#### Node

Units called Nodes are used in the design of distributed systems. In a robot system, nodes might include sensors (such as Lidar and cameras), motion controllers (such as motors that provide mobility), and algorithmic parts (such as a route planner). The operating system level process structure and the node notion are separated in ROS 2. A process can produce numerous nodes that can interact with one another independently if that is what is needed. The system's nodes can run on a single computer or they can be dispersed over a number of machines.

#### Communication Patterns (Topic, Service, Action)

Through Topic, Service Invocation, and Actions, nodes can communicate with one another. The topic-based communication uses a publish-subscribe architecture in which several nodes can subscribe to the data created and published by a single node, and multiple nodes can also produce data for a single topic. The themes listed in ROS 2 correspond to DDS topics perfectly. From the perspective of an application developer, service calls employ a remote method invocation technique. An strategy used for lengthy service calls is actions. The action server begins the lengthy process after receiving a goal-specific request from the client, publishes intermediate outcomes, and delivers the ultimate result after the goal has been reached. The DDS middleware is used to implement these services and actions as well.

Performance and fault tolerance are improved because to the decentralized design of the Topic structure. The ROS 2 architecture was elevated to new heights in part due to the extensive quality of service capabilities offered by the DDS middleware.

#### Messages

Message is the name of the data kinds used in topic communication (Message). The DDS data types are essentially matched by these message data structures (Type). Below is an example camera message:

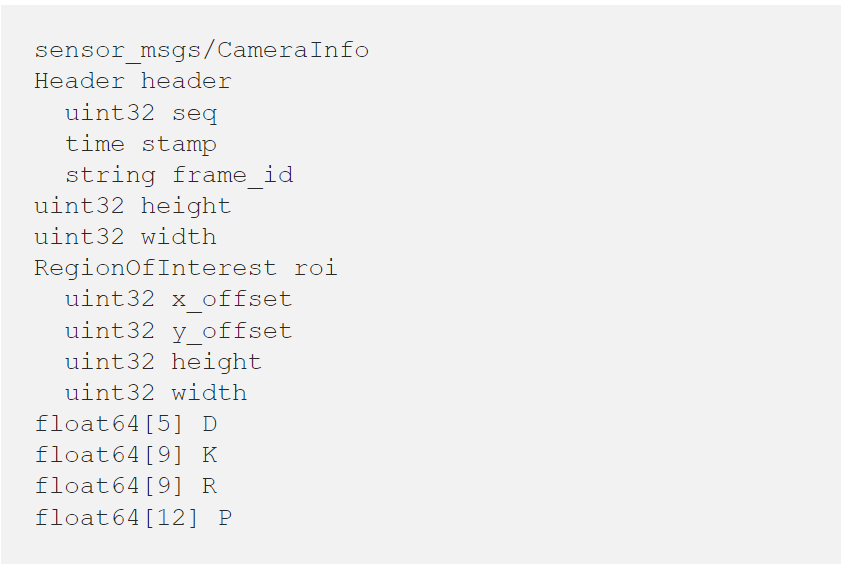


Figure : An example of camera message

In reality, interoperability depends heavily on the messages described in ROS 2. Cameras created by many firms can be rapidly connected into the system and can communicate with standard messages regardless of camera model thanks to these common messages. In this scenario, the Node component created for the pertinent camera will translate the camera's unique communication protocol messages into regular DDS data. The Gebazo simulation's camera simulator can be used for preliminary research.

The quality of services (QOS) outlined in the DDS middleware is also successfully utilized by ROS 2. For instance, an order to the robot to do a task can be safely provided even while continually updated sensor data is sent with best effort. An essential design choice is to match the themes with the extensive quality of service elements of DDS. The service quality connected to freshly specified subjects is decided by selecting one of these profiles from a list of ready-made service quality profiles that may be employed depending on the kind of data.

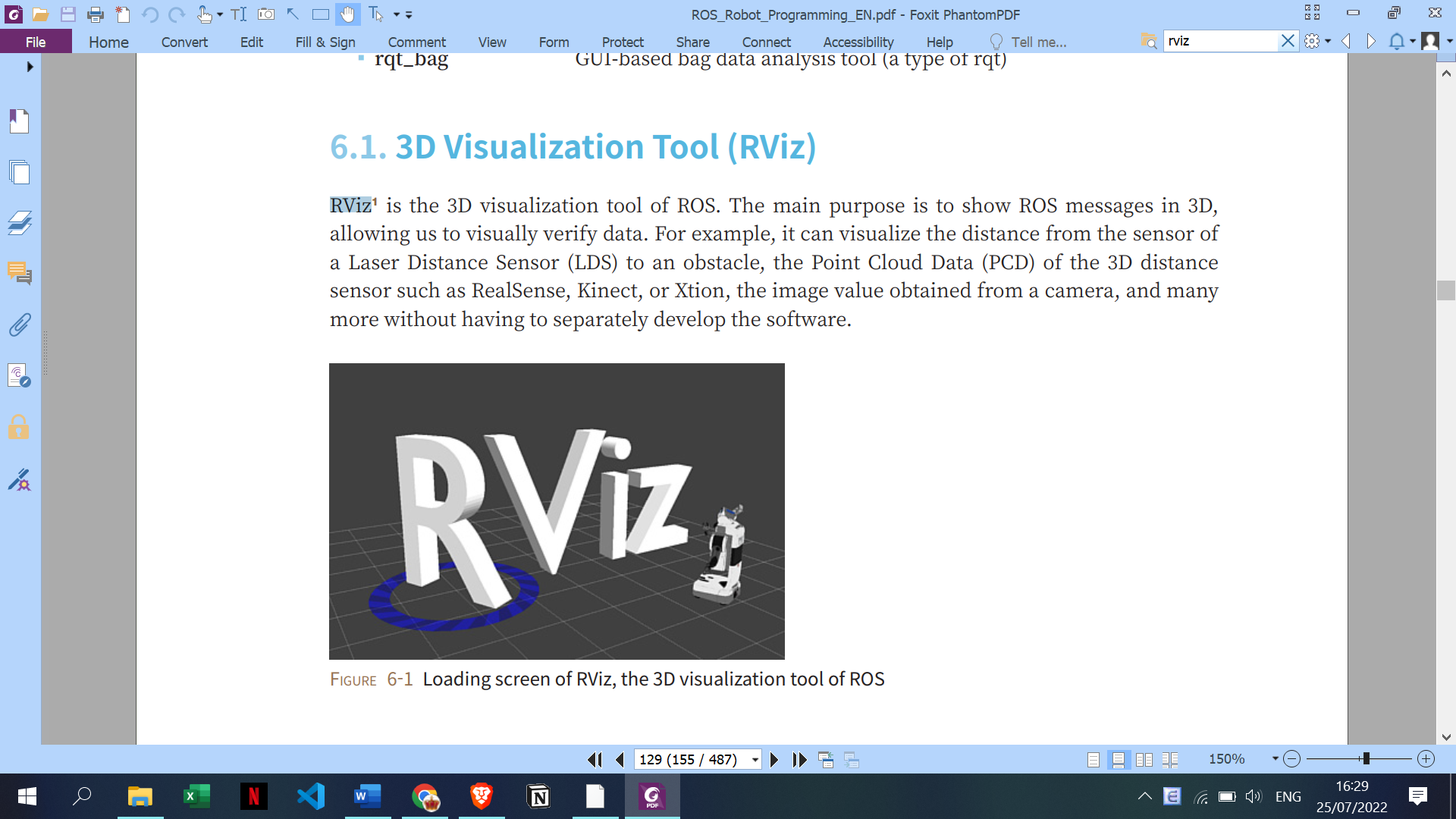
LƯU Ý (đã hoàn thành)

<https://medium.com/software-architecture-foundations/robot-operating-system-2-ros-2-architecture-731ef1867776>

### Visualization

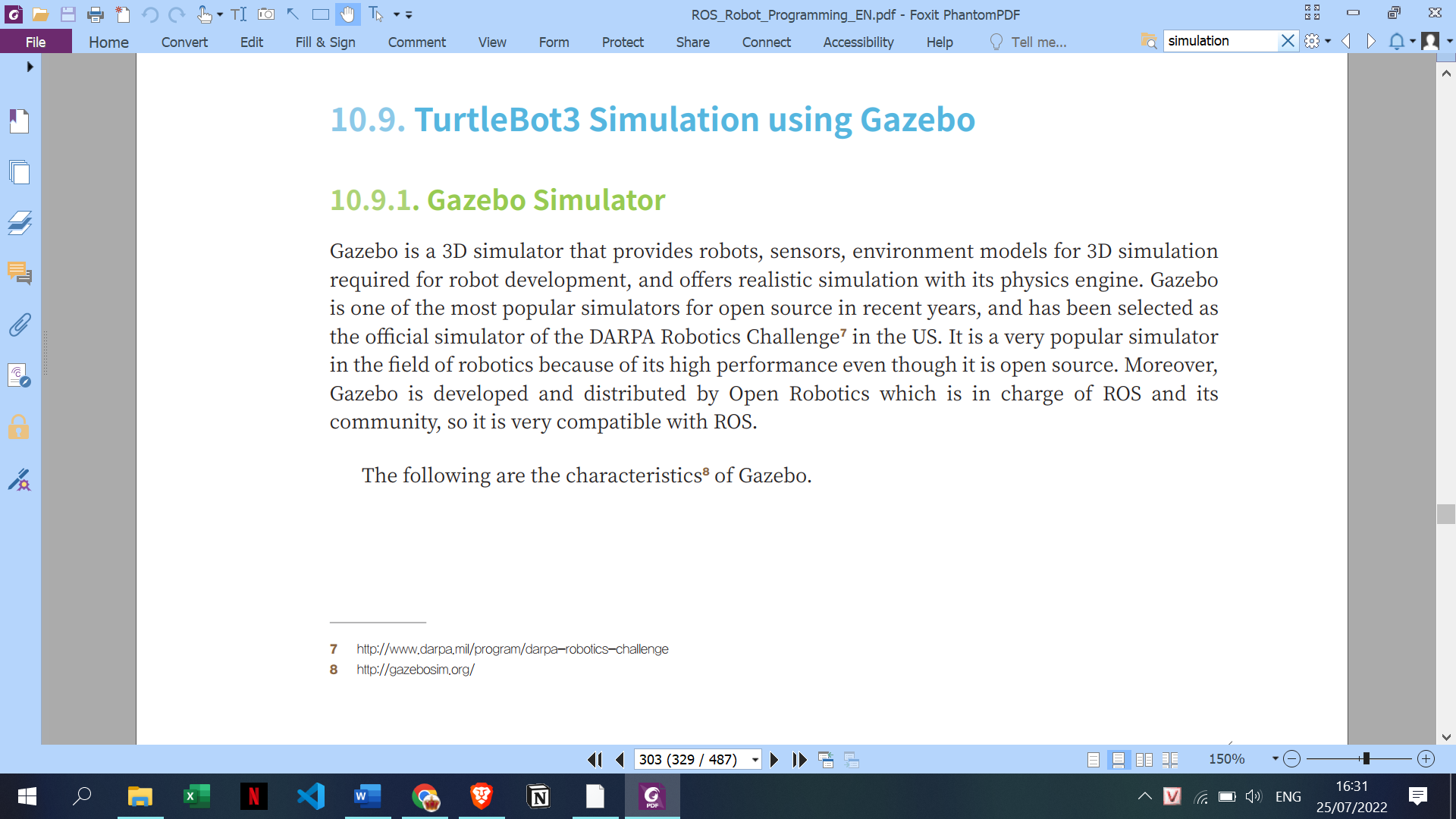
In addition to the tools that ROS users have independently released, there are a lot of other ROS utilities. While not directly processing a ROS function, the tools explored are nevertheless incredibly helpful as add-ons when using ROS to program. For this project, we primarily take advantage of RViz tool.

LƯU Ý



### Simulation

LƯU Ý



# FIRMWARE DESIGN FOR SERVING MOBILE ROBOT

## Raspberry Pi 4B configuration

### Ubuntu version

The project has been implemented at the time that Galactic was the latest release of ROS 2.

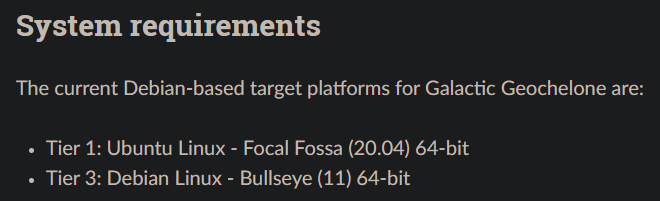


Figure : System requirements from official website for ROS 2 Galactic Geochelone

To have the best support from Operating System, single-board computer Raspberry Pi 4 has Ubuntu Linux – Focal Fossa (20.04) 64-bit installed.

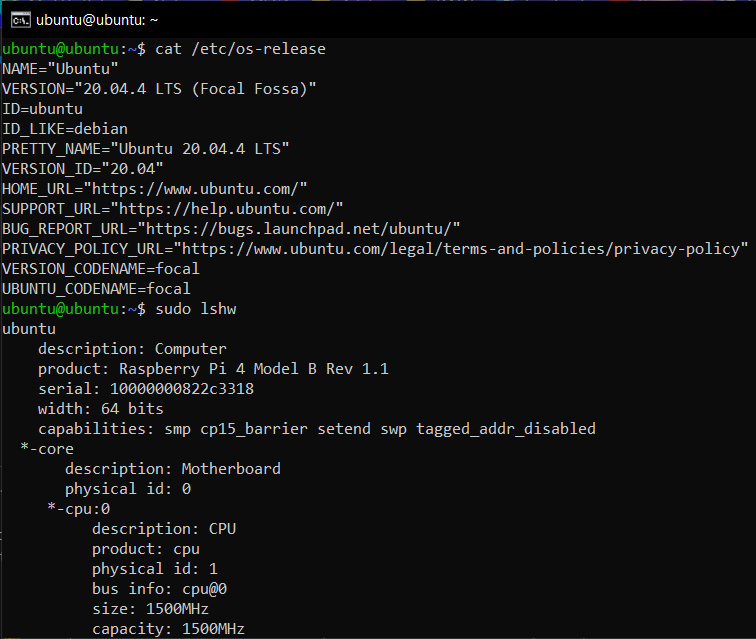


Figure : Ubuntu 20.04.4 LTS on Raspberry Pi 4

### zRAM

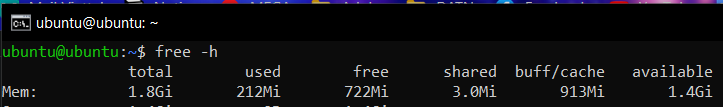


Figure : Initial amount of RAM on Raspberry Pi 4

For this project, Raspberry Pi 4 is the version which has 1.8 Gigabytes of original RAM.

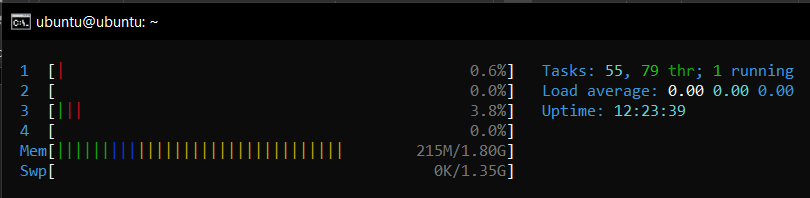


Figure : No-load performance of RAM on Raspberry Pi 4

As observed, when the system is running with no load, the remaining amount of RAM for ROS 2 tasks is 1.4 Gigabytes. During the process of building and running ROS 2 packages, the lag time is quite large and there is a phenomenon where the Raspberry Pi 4 crashes leading to a reboot. To overcome this problem, zRAM was configured for the system of Raspberry Pi 4.

zRam is

LƯU Ý: https://haydenjames.io/linux-performance-almost-always-add-swap-part2-zram/

### ROS 2 Galactic

### Multiple I2C buses

## Serving mobile robot firmware architecture

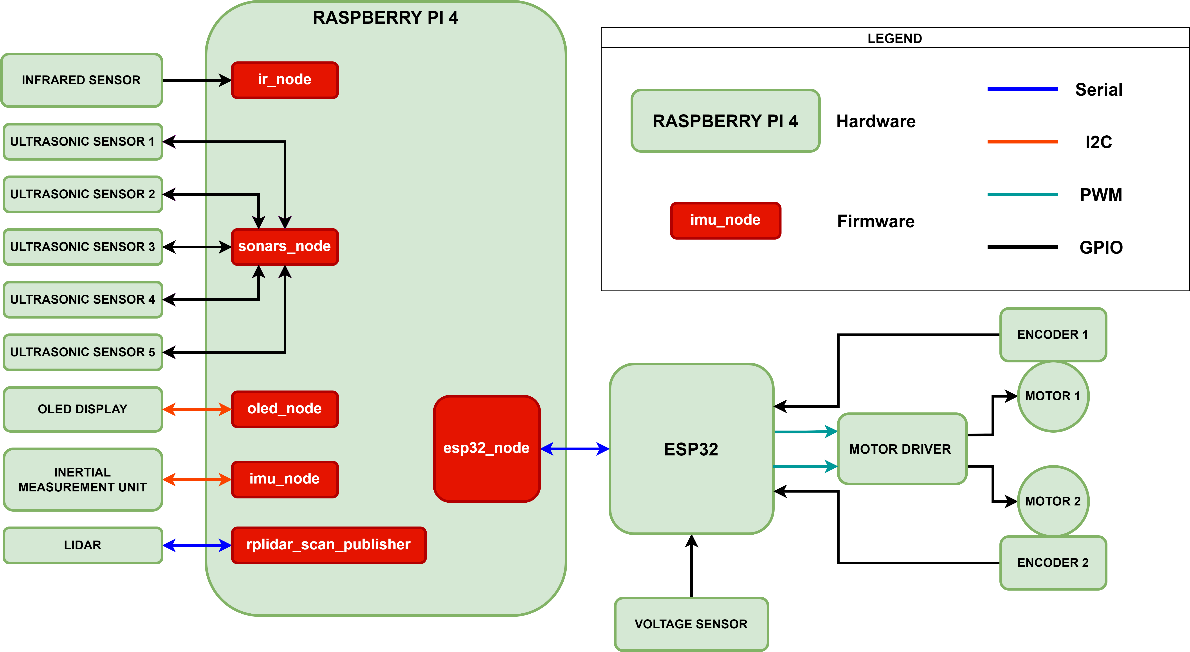


Figure : Firmware architecture of the system

Raspberry Pi 4 and personal computers can both run the ROS 2 system. All of the nodes on the Raspberry Pi 4 are limited to managing peripherals. The other nodes, however, including the SLAM node and the navigation stack, are executed on a personal computer. Each peripheral has an associated node that reads data from the sensor and publishes it to the ROS 2 system.

It takes the right hardware to control motors and receive pulses from encoders, both of which are time-sensitive processes. On the other side, the Raspberry Pi 4 is an inappropriate piece of hardware for applications that need precise timing. The Raspberry Pi 4's RPi.GPIO module is the best option for managing peripherals in Python. It is not, however, appropriate for real-time or timing-sensitive applications. This is due to the fact that Python's garbage collection schedule is unpredictable for developers. Additionally, because Linux is a multitasking operating system and another process could be given priority over the CPU, generating jitter in the program, Linux is not suited for real-time applications. The approach is to use an appropriate microcontroller unit, such as the ESP32 microcontroller, to manage this timing-critical activity. A hardware analog-to-digital converter is also absent from all Raspberry Pi computers. This serves as another evidence that the ESP32, with its 16 ADC pins, is an appropriate MCU for supplying the motor driver with PWM pulses.

## Firmware for motor driver block

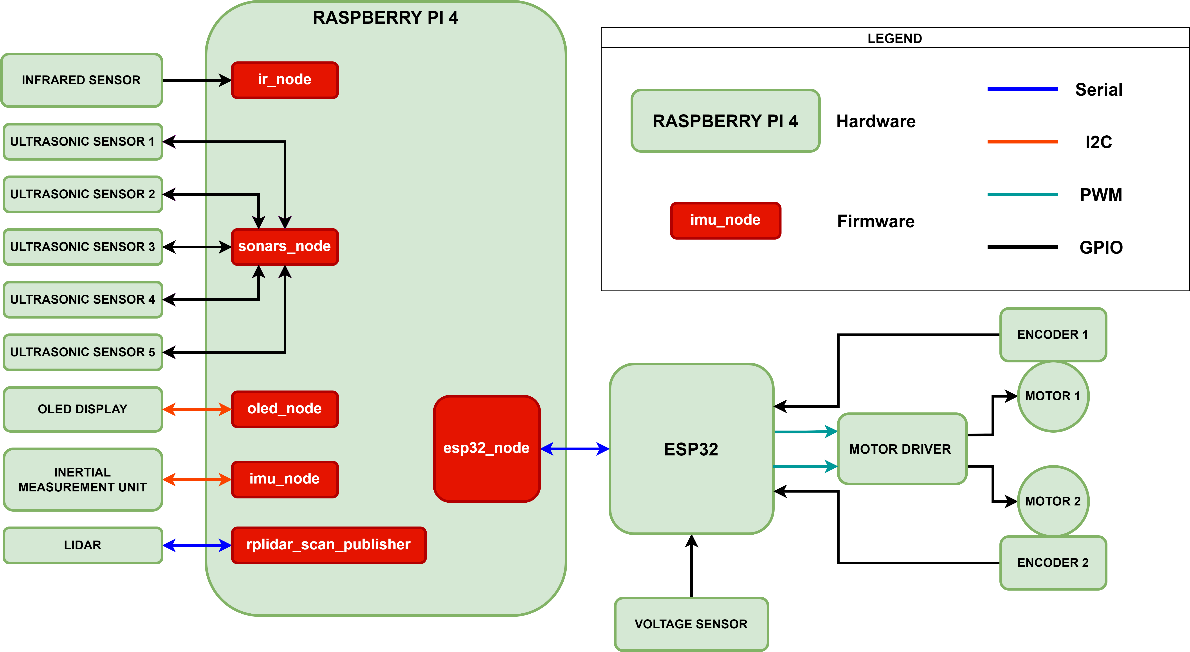


Figure : Motor driver block

Motor driver block includes ESP32, motor driver TB6612FNG, 2 motors and 2 encoders. Firmware for this block contains firmware for ESP32 microcontroller and firmware for Raspberry Pi 4.

### Choosing sample time of encoders

According to the encoder's specifications, each rotation of the motor axis corresponds to a change in 480 pulses. The motor may rotate at a maximum speed of 200 revolutions per minute (RPM). Thus, the maximum alteration of pulses each second could be calculated by the following equation:

|  |  |  |
| --- | --- | --- |
|  |  | Eq. |

The minimum duration of a pulse will be:

|  |  |  |
| --- | --- | --- |
|  |  | Eq. |

We decide to choose the value for sample time as 5 milliseconds. This period is equivalent to eight pulses which is an appropriate time for both firmware and hardware. When the number of pulses per sample period is too high, the inaccuracy will increase since the encoder is a sensor with a lot of noise. The computation for each sample time must finish before the start of the subsequent sample time, therefore if the sample time is too short for firmware, the system will be overwhelmed and unresponsive.

### Firmware for ESP32 microcontroller

The setup() and loop() procedures are the two primary operations in an ESP32 firmware. When a sketch launches, the setup() function is called. Variables, modes, libraries, and other things are initialized using it. Every time the board is powered on or reset, the setup() code will only be called once. Following the creation of a setup() method, which initializes and sets the starting settings, the loop() function does exactly what it says and performs successive loops, enabling your application to adapt and alter.

#### Setup() function

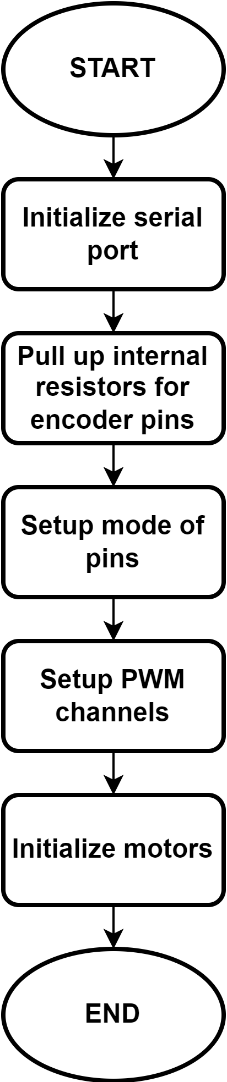


Figure : Flowchart of setup() function for EPS32 firmware

The setup() function includes five successive processes as the Fig. 9. The crucial part in this series of procedures is initializing baud rate for serial port which will be explained later. Pins on the ESP32 microcontroller are configured as input mode or output mode. ESP32's internal resistors pull the input pins, namely the encoder pins, up to a high logic level. By doing this, the pins are prevented from entering an invalid floating condition. The next step is to set up two PWM channels to regulate the speed of the system's two motors.

#### Loop() function

The loop() function includes four vital sections which are receiving data; driving motor; counting pulses, calculating RPM and sending data back to Raspberry Pi 4.

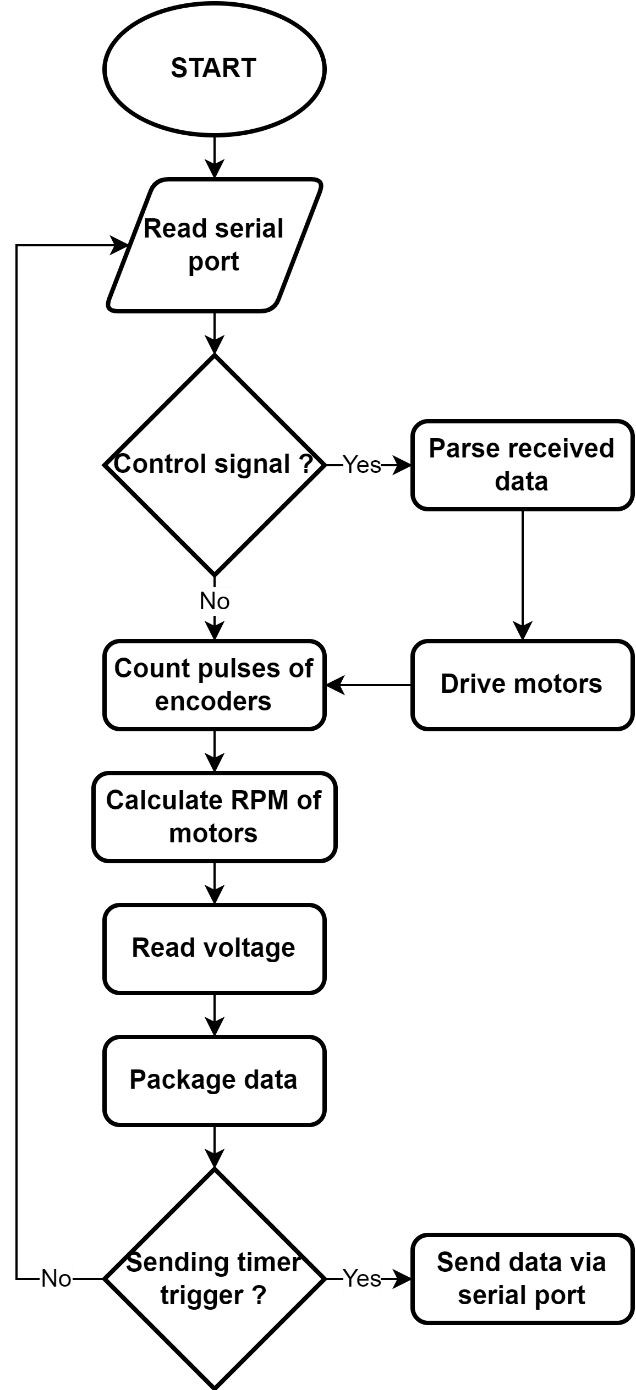


Figure : Flowchart of loop() function for EPS32 firmware

##### Receiving data from Raspberry Pi 4

The data ESP32 acquired from serial port have the form of JavaScript Object Notation (JSON). The form of data for instance can be observed in Fig. 10:

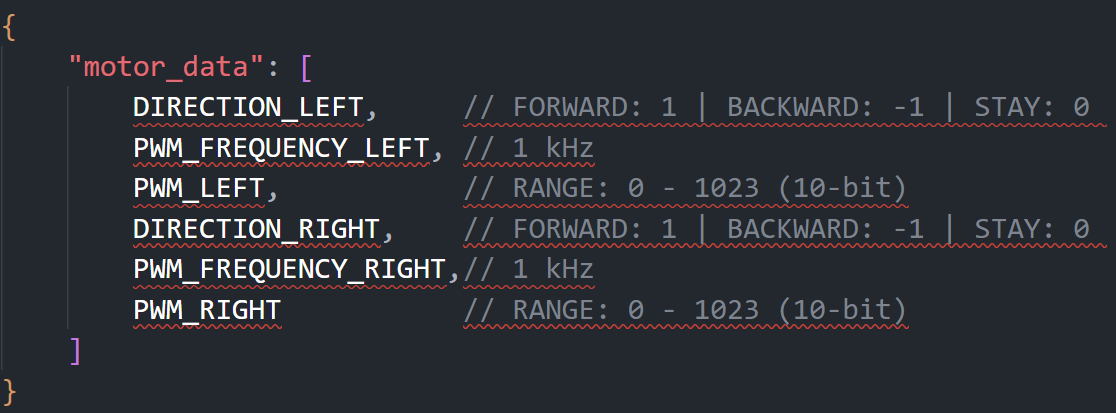


Figure : JSON form of receiving data of ESP32 microcontroller

The direction of the motors is indicated by the direction variable, which has a value of 1, -1, or 0. PWM frequency value selected is 1000 Hz so that the mechanical response of the motor is smooth enough; therefore, there is no sense of tripping due to voltage changes and the motor driver is not overload.

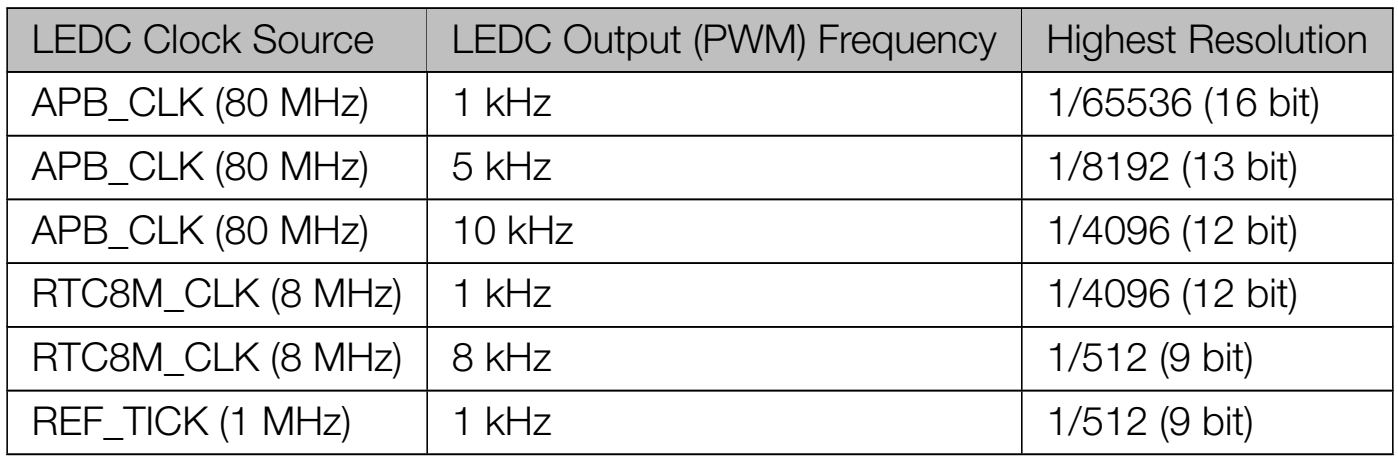


Figure : Commonly­used Frequencies and Resolutions

With a resolution of 10-bit and a PWM frequency of 1 kHz, the motors will operate more smoothly when the resolution is changed between two steps. Moreover, this resolution is within the range of values that make ESP32 works properly as in Fig. 12.

##### Driving motors



Figure : Flowchart of driveMotors() function for EPS32 firmware

Parsed JSON data will be saved to user-define struct DataReceive before extracted for using in driveMotors() function. With a program, only one line of code can be run at a time. This causes the motor voltage to happen at two different times causing a delay on one side of the motor (Fig. 14). we fixed this by using the STANDBY pin of the TB6612FNG motor driver.

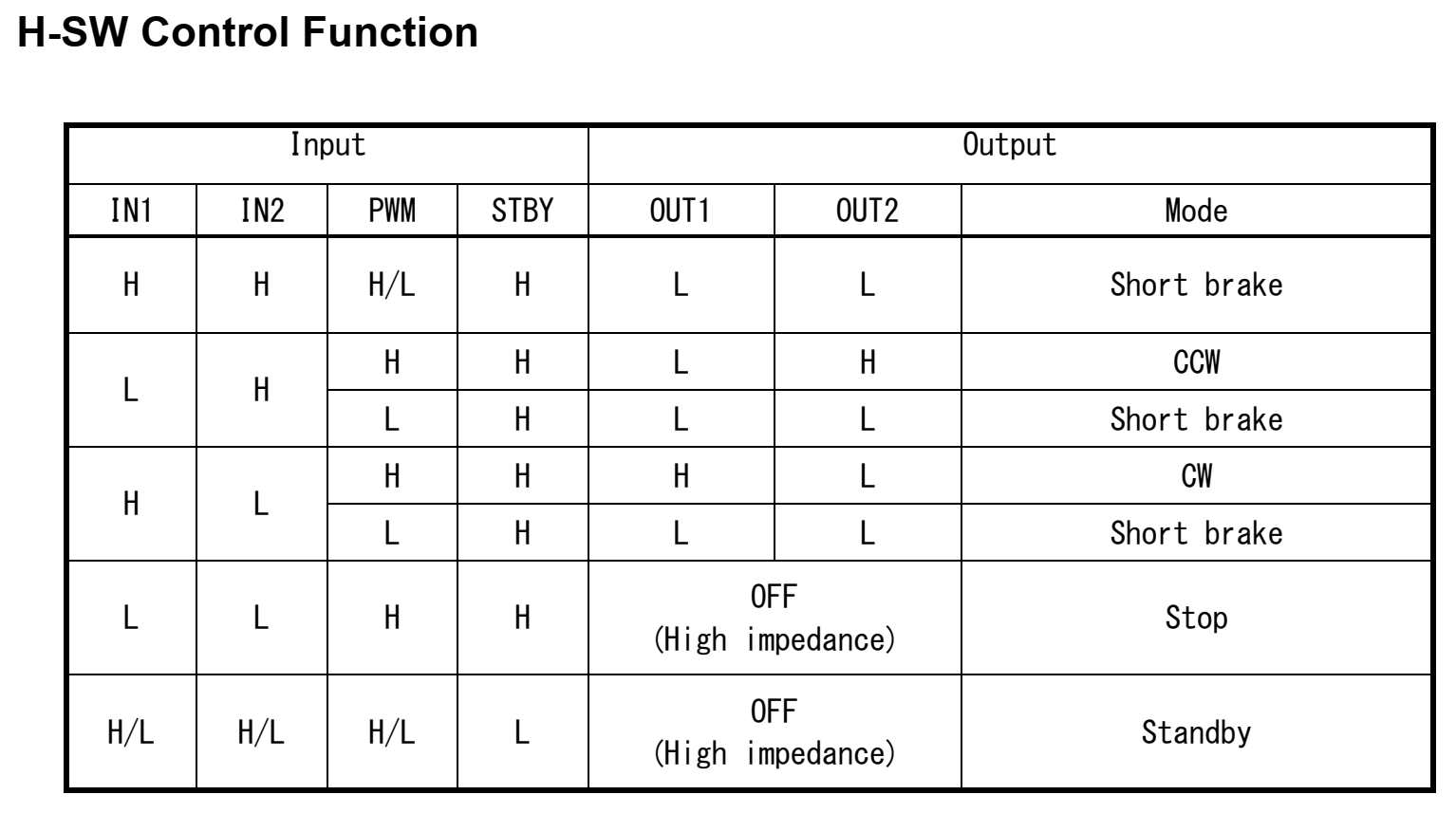


Figure : H-SW Control Function of TB6612FNG motor driver

Both of the TB6612FNG's output PWM pins are disabled while the STANDBY pin is at the low logic level (high impedance mode). The STANDBY pin is switched to a high logic level before the PWM channel value is set using this motor driver function as Fig. 15.

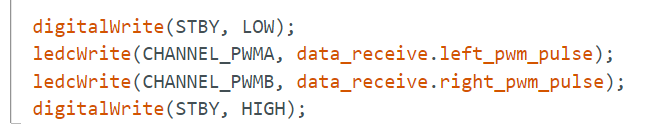


Figure : Solution for instantaneous operating of motors

##### Counting pulses of encoders and calculating RPM of motors

Input signal rising and/or falling edges are counted by the PCNT (Pulse Counter) module. In the module, the ESP32 has several pulse counters. Each unit effectively functions as a separate counter with a number of channels, each of which has the ability to increase or decrease the counter on an edge. We use a C++ library on GitHub by Kevin Harrington to perform pulse reading of the encoders.

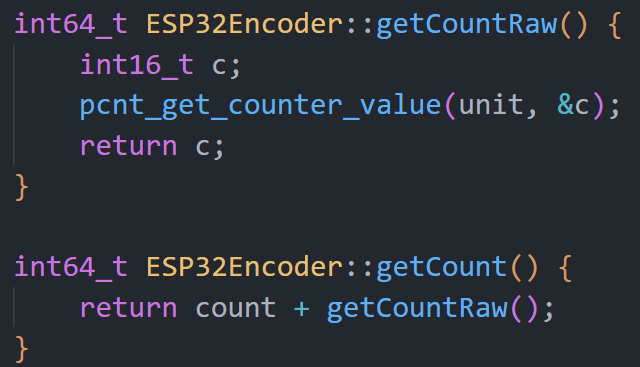


Figure : Block of code for counting pulses in ESP32Encoder C++ library

The function "pcnt\_get\_counter\_value" returns the count value and the "getCount" method of the "ESP32Encoder" class adds the old count value and the new count value to get the current cumulative count value with data type “int64\_t” – 64-bit integer.

A user-define class named “RPMCalculator” was written to handle the calculating revolution per minute (RPM) of motors.

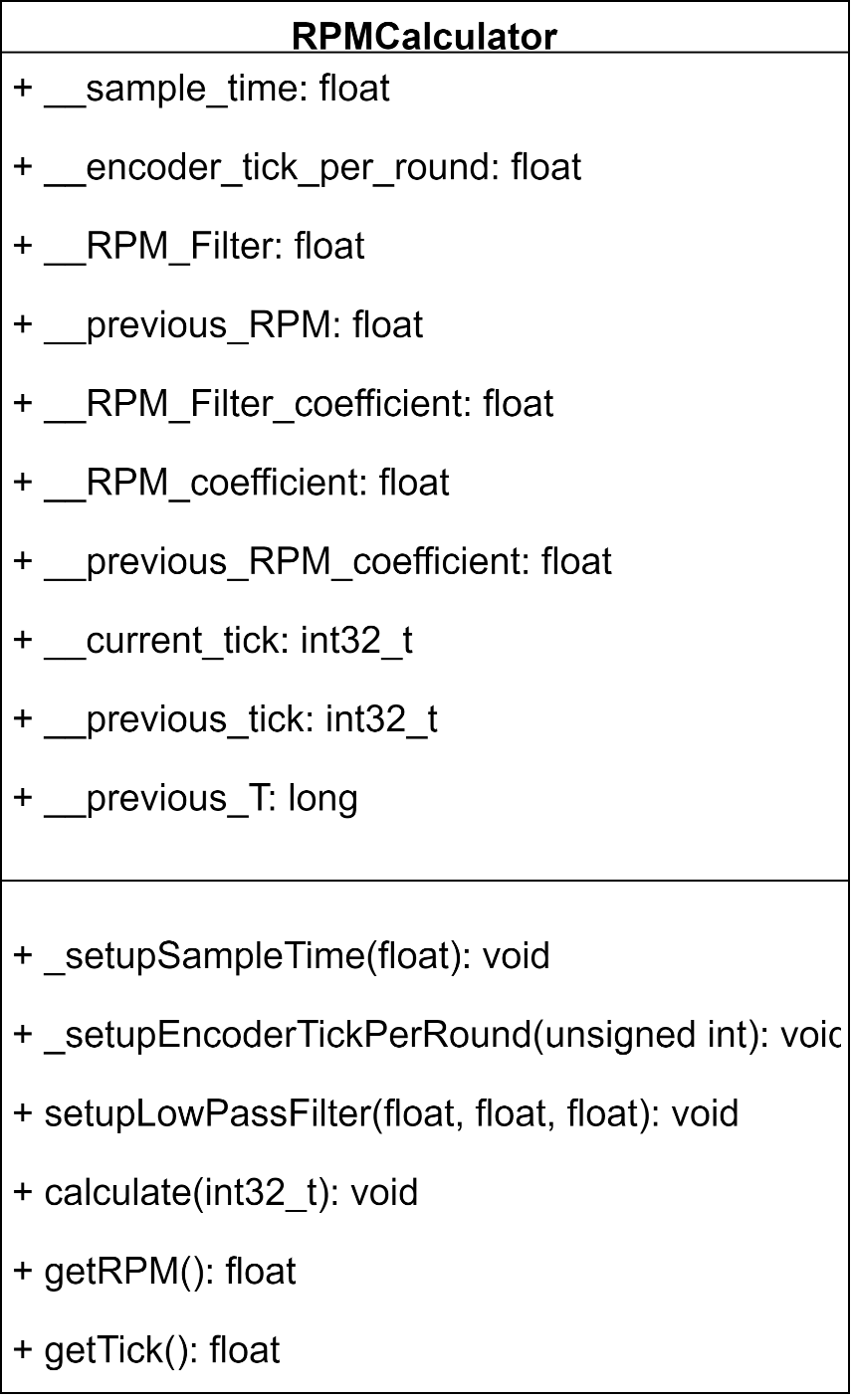


Figure : ESP32 Firmware - RPMCalculator class

“RPMCalculator” class has the primary method which are “calculate”. This method takes one input which is the current pulse of a encoder and update the value of “\_RPM\_Filter” property. This value then is considered as the current RPM of the motor.

##### Packaging data for sending to Raspberry Pi 4

Data which includes pulses of encoders, RPM of motors are serialized for checksum handling. The system uses MD5 checksum algorithm for data authentication between two serial ports of Raspberry Pi 4 and ESP32 microcontroller. A message of any length can be entered into the cryptographic hash function algorithm MD5, which converts it into a fixed-length message of 16 bytes. This string of checksum value is also added to the JSON string which is sent to Raspberry Pi 4 with the frequency of 1000 Hz. The frequency of data flow prior to sampling must be higher than the sampling rate, as the sample duration is 5 milliseconds, or 200 Hz.

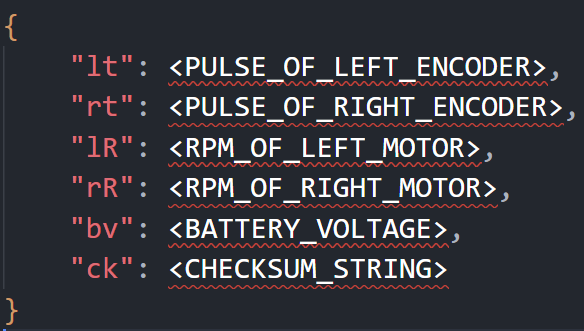


Figure : JSON form of sending data of ESP32 microcontroller

##### Choosing baud rate for serial port

LƯU Ý: số byte của receiving data, ảnh chụp monitor + tính toán

LƯU Ý: số byte của sending data, ảnh chụp monitor + tính toán

* Bao nhiêu byte
* Đảm bảo nhỏ hơn sampling rate
* Chọn baud rate

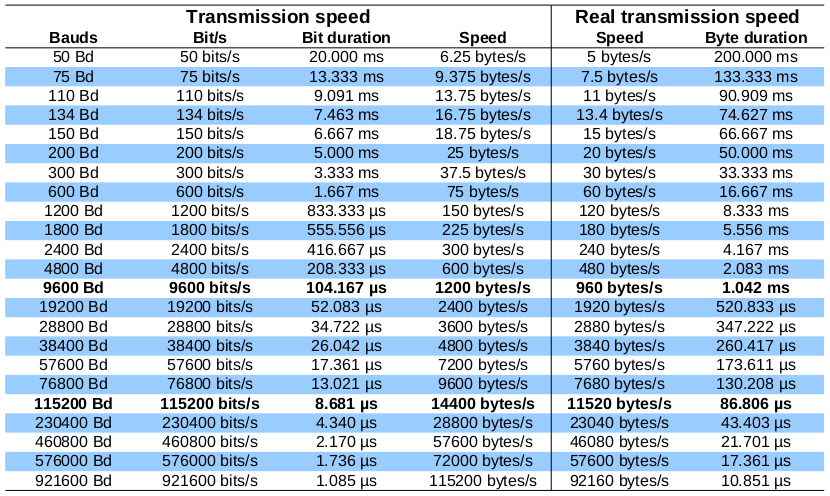
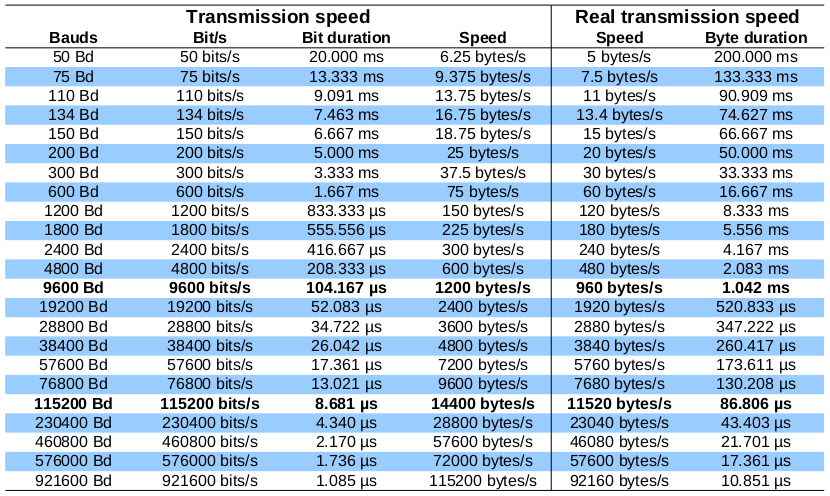


Figure : Transmission speed and Real transmission speed of serial port

### Firmware for Raspberry Pi 4

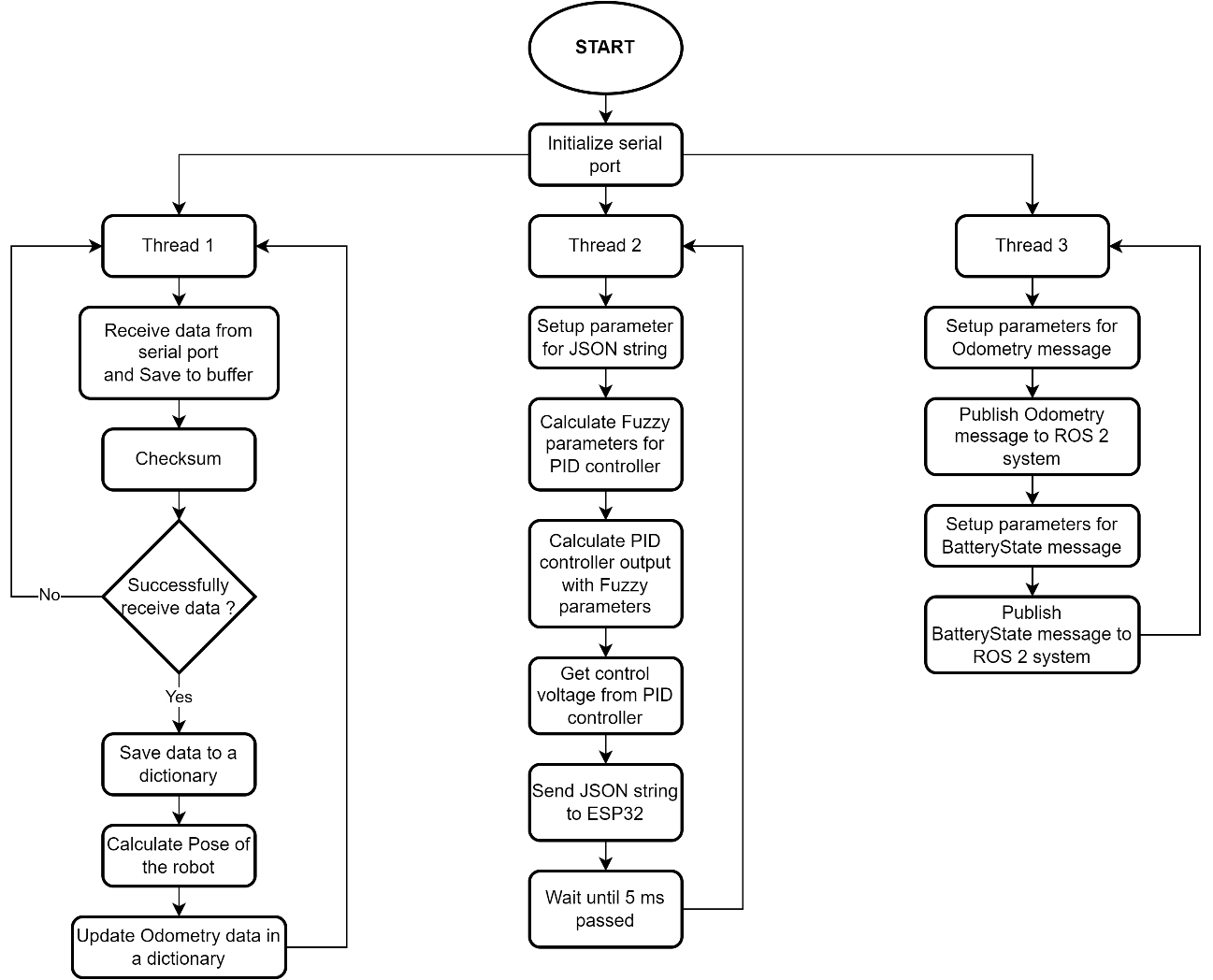


Figure : Flowchart of Raspberry Pi 4 for motor driver block

Initializing the serial port on the Raspberry Pi 4 is the first step towards connecting to the ESP32. As seen in Fig. 8, the Raspberry Pi 4's serial ports are utilized to share data with an ESP32 microcontroller and a Lidar. The names of two 2.0 USB ports are configured at random each time the Raspberry Pi 4 reboots. We build some code for the ESP32 microcontroller's serial port name detection in order to solve this problem.

LƯU Ý: thêm ảnh chạy “dmesg | grep ttyUSB” để lấy tên của 2 cổng serial và tên của USB driver cho ESP32 và Lidar

The procedure of Raspberry Pi 4 then is divided into three threads which are independent unit of a process, also known as a program being executed. These threads will execute parts of the program in parallel which helps to keep the running time of the program parts does not affect each other.

#### Thread 1

For the first thread, the software continually reads data in the form of JSON strings from the ESP32 microcontroller and saves them to buffer variables. The "calculatePose" function of the "PoseCalculator" class's input parameters are the pulse current count values that are stored in buffer variables.

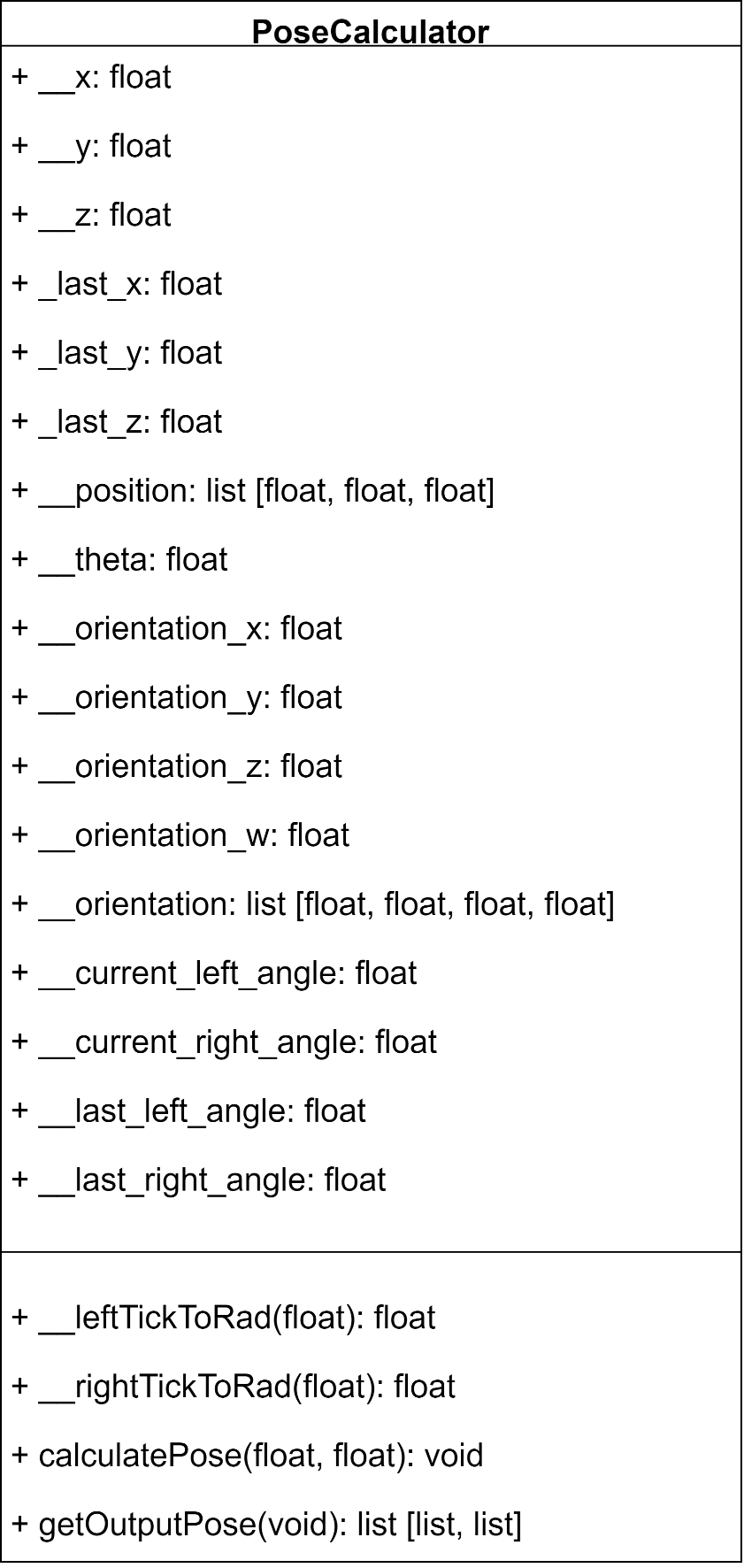


Figure : Raspberry Pi 4 Firmware – PoseCalculator class

According to the definition of ROS, representation of pose in free space is a combination of position and orientation as Fig. 23.

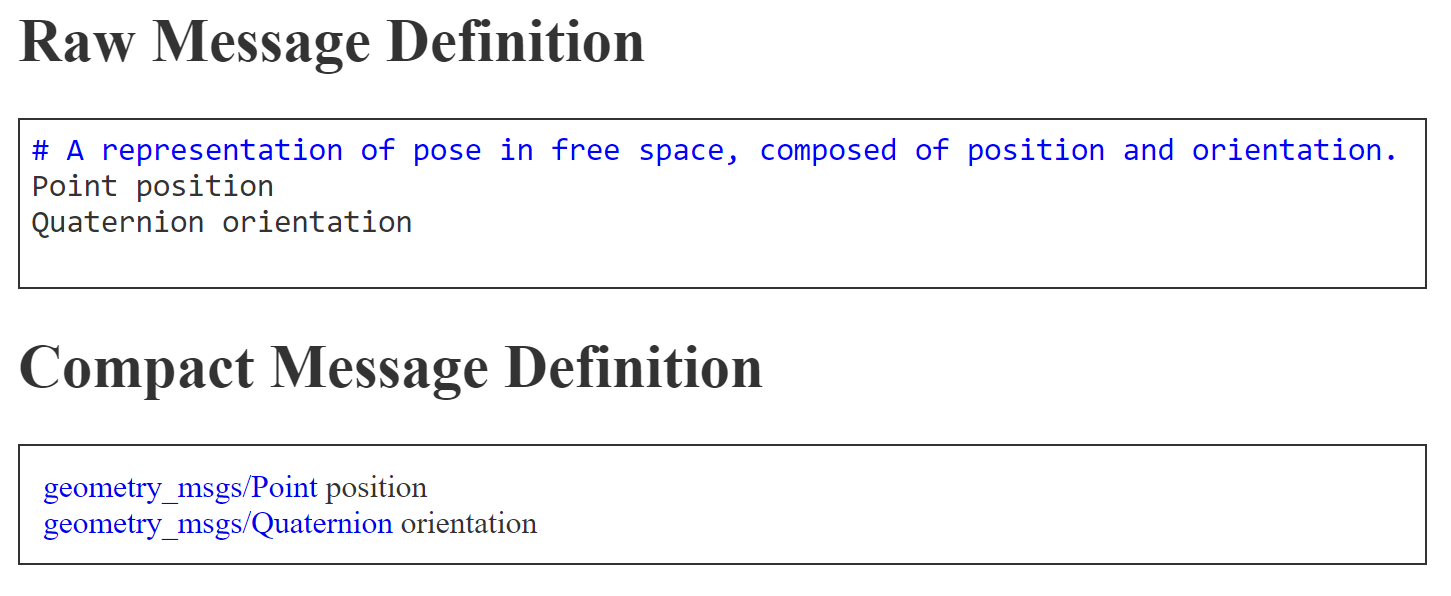


Figure : Definition of Pose according to ROS 2

For velocity along the axes and angular velocity of the robot, we have the equation:

|  |  |  |
| --- | --- | --- |
|  |  | Eq. |

On the other hand, we have the relationship between the linear velocity and the angular velocity of motors:

|  |  |  |
| --- | --- | --- |
|  |  | Eq. |

From Eq. 3 and Eq. 4, we have the equation which shows the relation between the velocity along the axes of the robot and the angular velocity of the motors (the detailed calculation will be presented in the appendix):

|  |  |  |
| --- | --- | --- |
|  |  | Eq. |

As the robot move, Eq. 5 becomes a time-varying equation which describes the motion of the robot. Thus, we have the following equation:

|  |  |  |
| --- | --- | --- |
|  |  | Eq. |

To determine the robot's position, the above equation is calculated and updated continuously through each program cycle which is a brief period of time. This process is carried out by the primary method “calculatePose” of the “PoseCalculator” class. As a result, we acquire Odometry data which contains position and orientation of the robot.

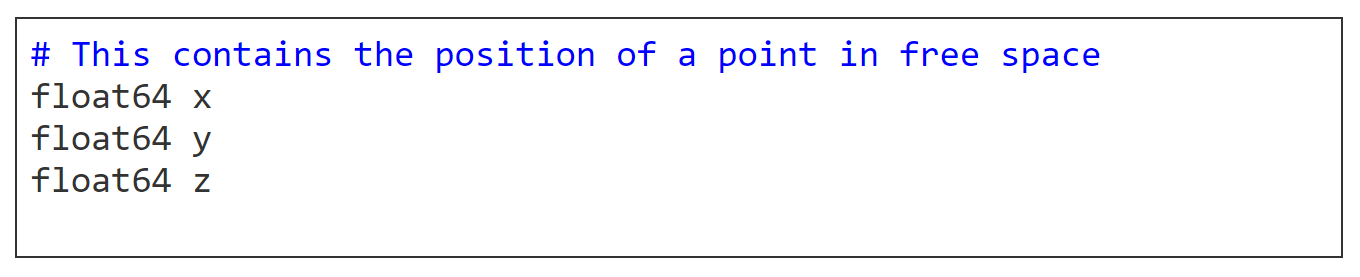


Figure : Data type of position

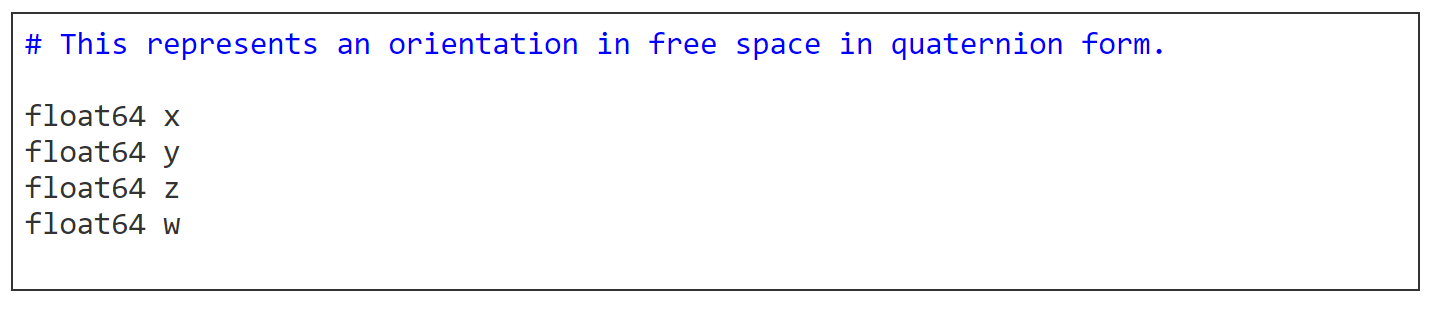


Figure : Data type of orientation

A dictionary type global variable is then continually updated with the Odometry data.

#### Thread 2

This thread plays an crucial role in driving the robot. Along with the flow of the program, the JSON data string is initialized as the first task of the program. The coefficients of Fuzzy logic are calculated before being included as parameters to the PID controller. Each motor has its own PID controller with the Kp, Ki and Kd coefficients which are computed and simulated in MATLAB software. With a setpoint and current angular velocity of motor, the PID controller reckon the appropriate voltage which is saturated before being the output. Considering that the sample time is 5 milliseconds, the program contains a waiting function at the end to guarantee the sample time.

#### Thread 3

This thread handles the communication between motors, encoders and ROS 2 system. All of the program's threads can utilize the Odometry since it is kept as a global variable. The Odometry message then is sent to topic “/wheel/odometry” with the frequency of 30 Hz. Each coefficient of Odometry data is extracted to the message for the ROS 2 system. Another procedure handles publishing the BatteryState message will be presented later.

## Firmware for voltage sensor

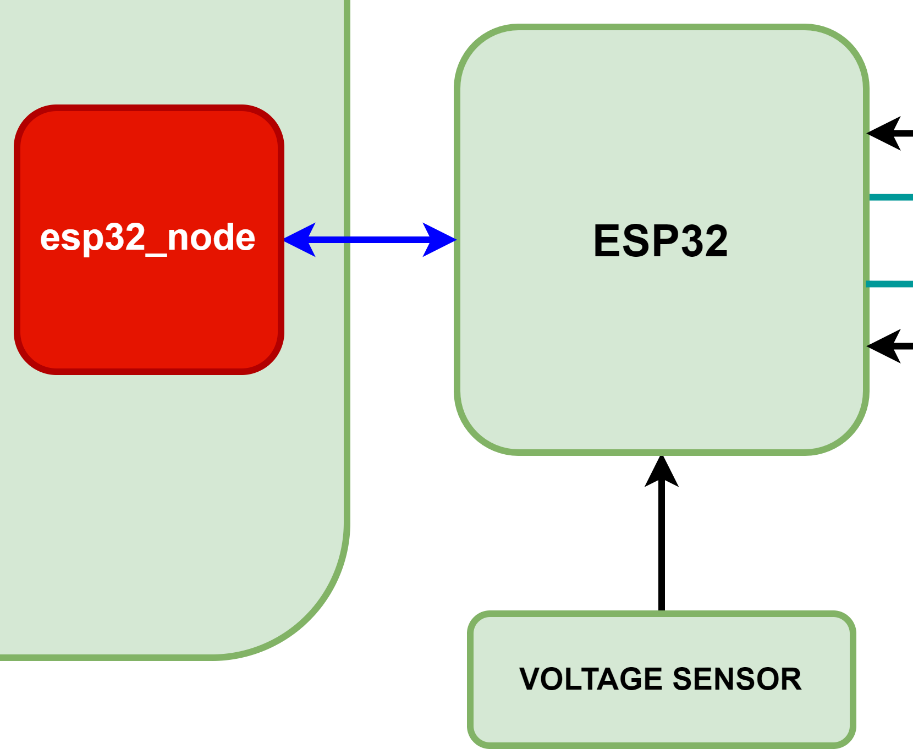
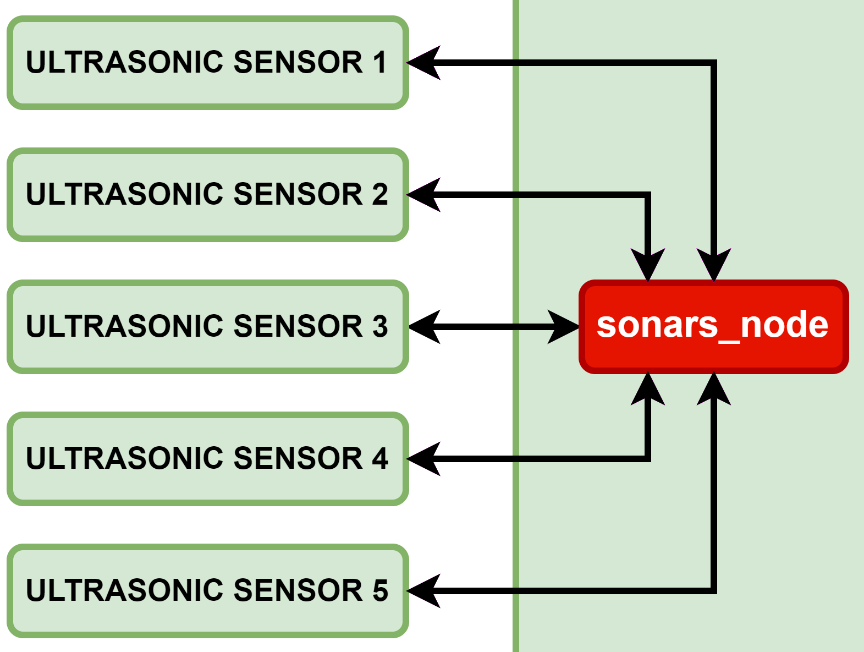


Figure 29: Voltage sensor block

Since Raspberry Pi 4 has no analog-to-digital converter (ADC) module, we use ADC module of ESP32 microcontroller for reading voltage of the Lipo battery. The program comprises two firmware parts which are “esp32\_node” on Raspberry Pi 4 and the firmware on ESP32 microcontroller. The program of the motor driver block in the previous section also handles reading voltage of the battery.

### Firmware for ESP32 microcontroller

## Firmware for ultrasonic sensors



The "Sonar" class and "SonarNode" class are two objects that are part of the software for ultrasonic sensors. The "Sonar" class is in charge of measuring the separation between the robot and low-height objects.

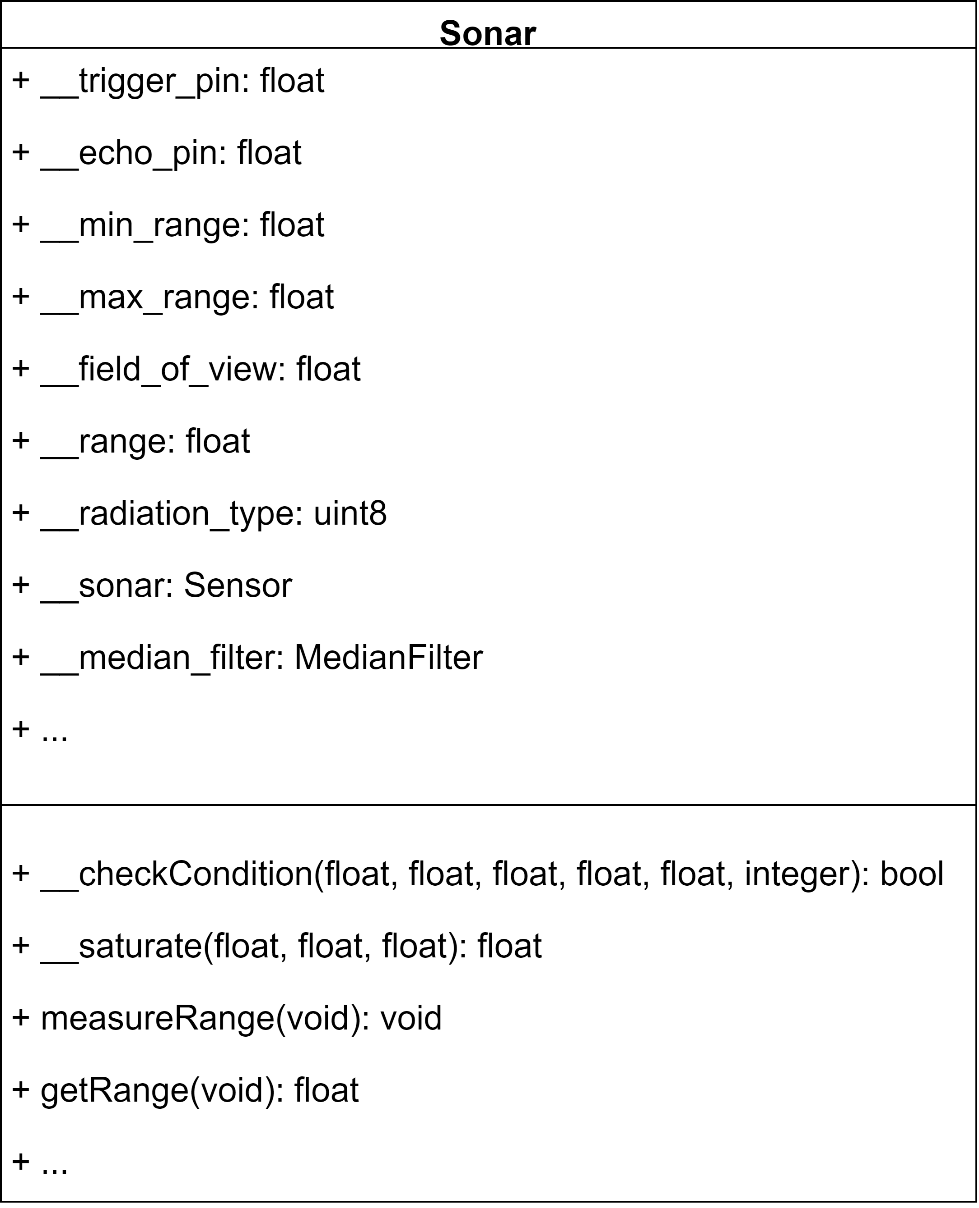


Figure : Raspberry Pi 4 Firmware – Sonar class



Figure : Defining a Sonar instance with arguments

#### “Sonar” class

As observed Fig. 27, a Sonar instance has some main parameters as trigger pin, echo pin and maximum range of measuring. The primary method of “Sonar” class is “measureRange” which apply pulse to trigger pin of ultrasonic sensor to gauge the distance from obstacles.



Figure : Flowchart of Raspberry Pi 4 for measuring distance of ultrasonic sensor

Initially, the trigger pin and echo pin are set up as input and output pins, respectively. The software then starts a pulse that lasts for 10 microseconds, causing a succession of audible bursts to occur in the surroundings. To wait for an echo pulse, a "while" loop is set up. The system next calculates the separation between the obstruction and the ultrasonic sensor if the echo pulse was received; otherwise, the system will attempt a new measurement. The distance is computed using the following equation:

|  |  |  |
| --- | --- | --- |
|  |  | Eq. |

#### “SonarNode” class

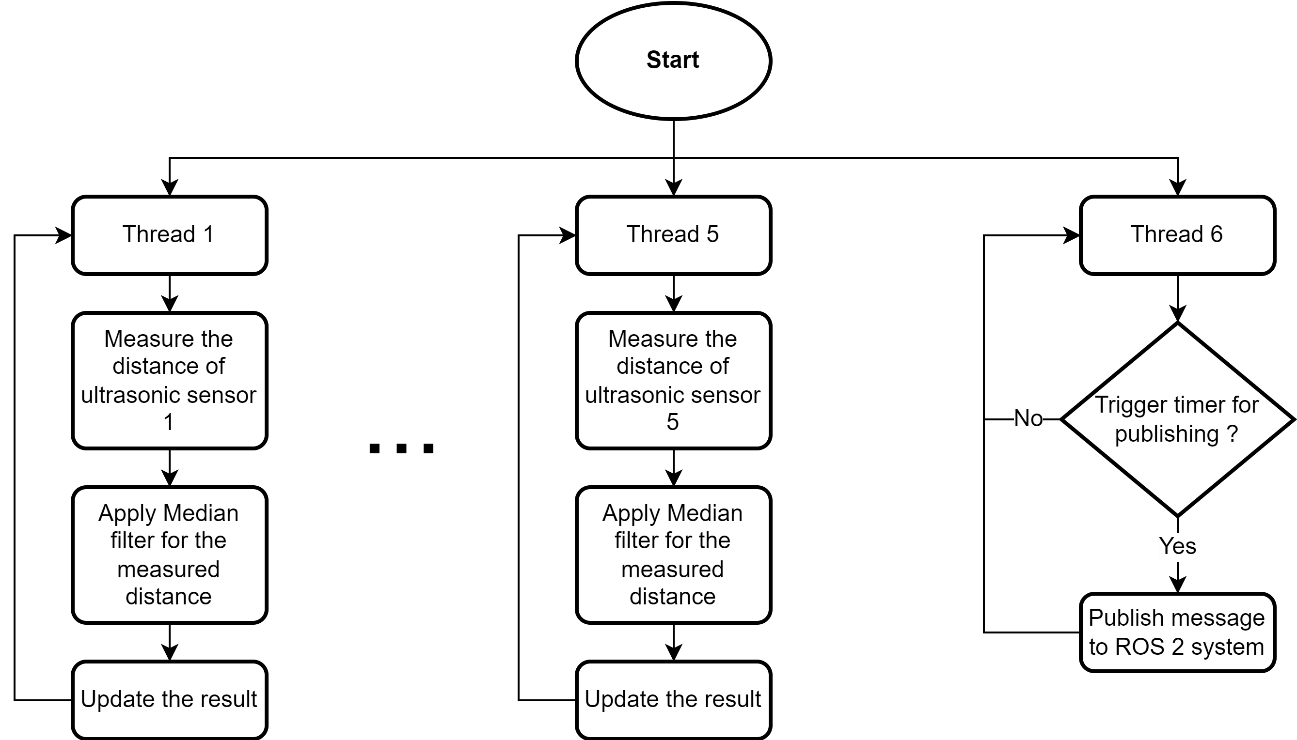


Figure : Flowchart of Raspberry Pi 4 for sonar node

On the other side, communication with the ROS 2 system is done using the "SonarNode" class. The software is broken up into six distinct threads as Fig. 29. The handling of each measuring process by a thread ensures that the distance measurement of one sensor has no impact on the other sensor. An instance of "SonarNode" running in thread 6 then accesses the result, which is saved in a global variable. A message with the type of Range that is delivered to the subject "/ultrasonic sensor x" contains all of the ultrasonic sensor's data compressed. With a frequency of 10 Hz, (x is the order of the ultrasonic sensor, from 1 to 5). However, the middle ultrasonic, which is crucial for identifying obstructions in front of the robot, updates at a frequency of 20 Hz.

## Firmware for lidar



Figure : Lidar sensor block

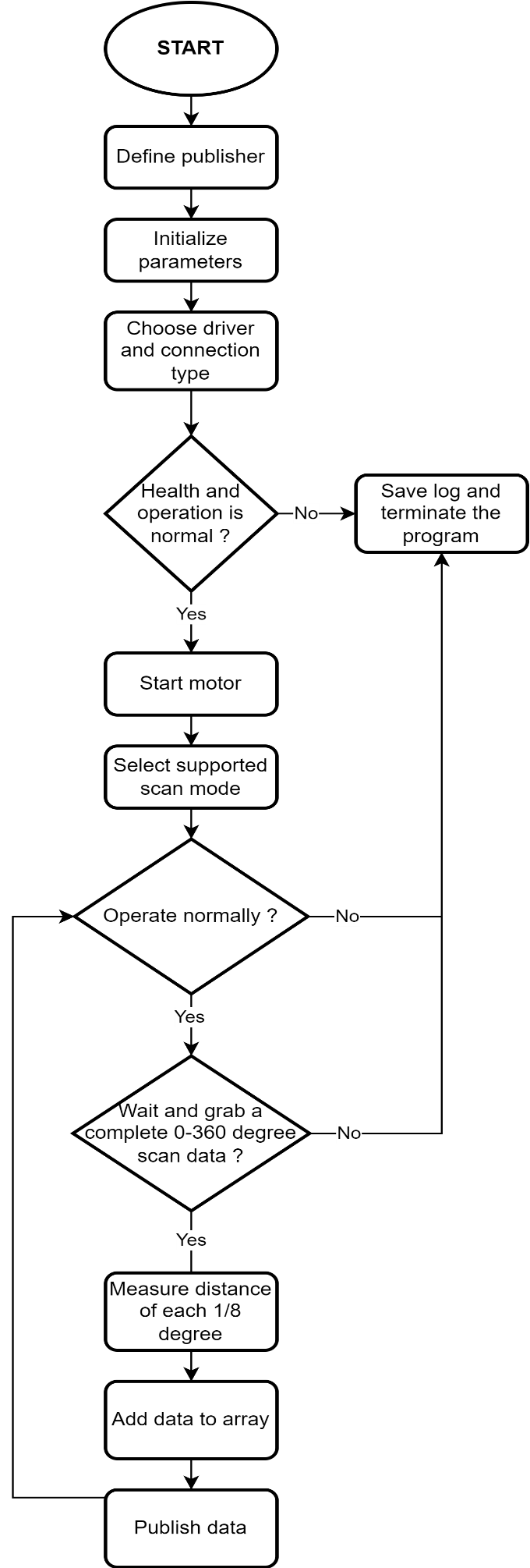


Figure : Flowchart of Raspberry Pi 4 for lidar node

Lidar node is a program which is used for operating and extracting data of lidar sensor. For this project, the program used to operate RPLIDAR A1 is written by RoboPeak Team and Shanghai Slamtec Co., Ltd. Firstly, the parameters are initialized by defining variables and reading information from the device. A driver is selected between TCP (transmission control protocol) type and Serial Port type so that it is appropriate with the programmed firmware of the microprocessor of the lidar sensor.

Status of the lidar is verified before the motor is started. During the acceleration of the motor, the program will choose the scan mode which is supported by the device. RPLIDAR A1 has two scanning mode which are “Standard” mode with 2000 samples per second and “Express” mode with 4000 samples per second.

In the next step, the program runs into a “while” loop which continuously checks the operation status. An array of nodes is initialized to gather data of each Sample Point as Fig. 31. The size of array is 2880 nodes which is equivalent 8 measurement each degree.

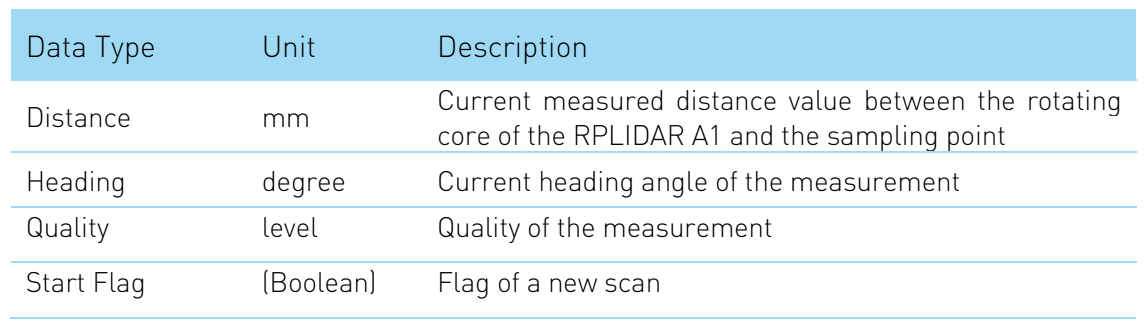


Figure : The RPLIDAR A1 Sample Point Data Information

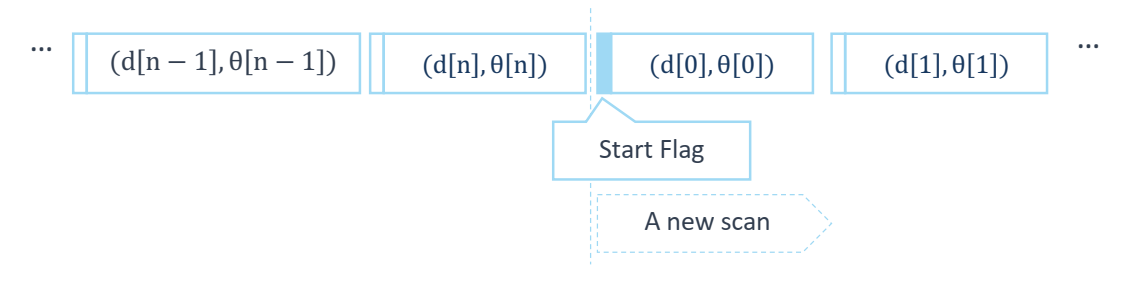


Figure : The RPLIDAR A1 Sample Point Data Frames

Each Sample Point Data has an Boolean variables named “Start Flag” to mark a new scan.

The last procedure of each loop is publishing message with the type LaserScan to ROS 2. All the messages are sent to topic “/scan” with the frequency of 10 Hz.

The interface of lidar sensor is visualized as the Fig. 33.

LƯU Ý: ảnh map lidar

Figure : The Obtained Environment Map from RPLIDAR A1 Scanning

## Firmware for inertial measurement unit

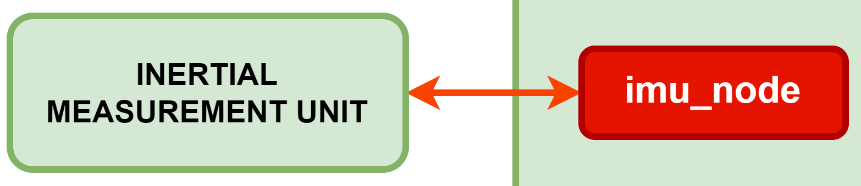


Figure : Inertial measurement unit sensor block

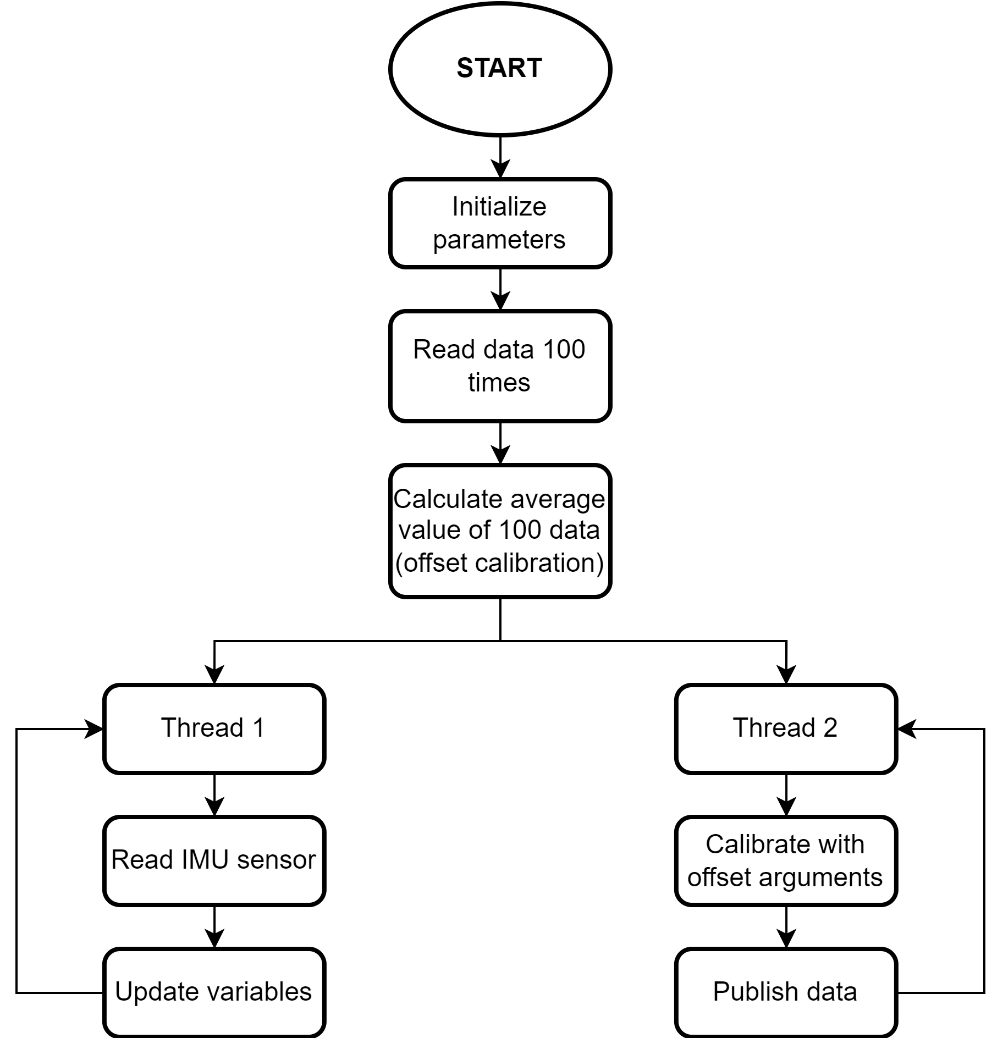


Figure : Flowchart of Raspberry Pi 4 for IMU node

Firmware for IMU carries out two tasks which are reading data from the sensor and publishing those data to ROS 2 system. The program starts with the initializing parameters process. Afterward, a series of 100 data will be measure to take to average values for offset arguments. From now, the program is divided into 2 threads which run independently.

For the first thread, the system continuously measures the data of IMU sensor which contains acceleration and gyro of 3 axes X, Y and Z. The data are kept in two variable with data type of List.

The other thread is in charge of calibration and publishing messages to ROS 2 system. All the measured data will be recomputed with the offset values before being composed as a message. Formed message is published into “/imu/data\_raw” with the frequency of 50 Hz.

## Firmware for OLED display

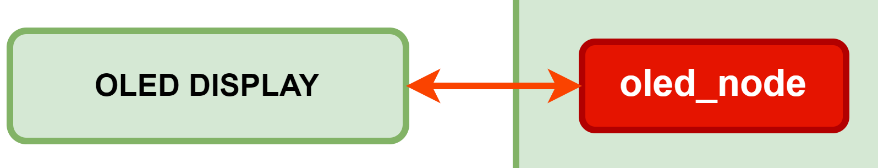


Figure : OLED display block

Firmware for OLED display incorporates “Oled” class and “OledNode” class. “Oled” class manages to compose text and display information on the display hardware. “OledNode” carries out communication with ROS 2 system.

#### “Oled” class

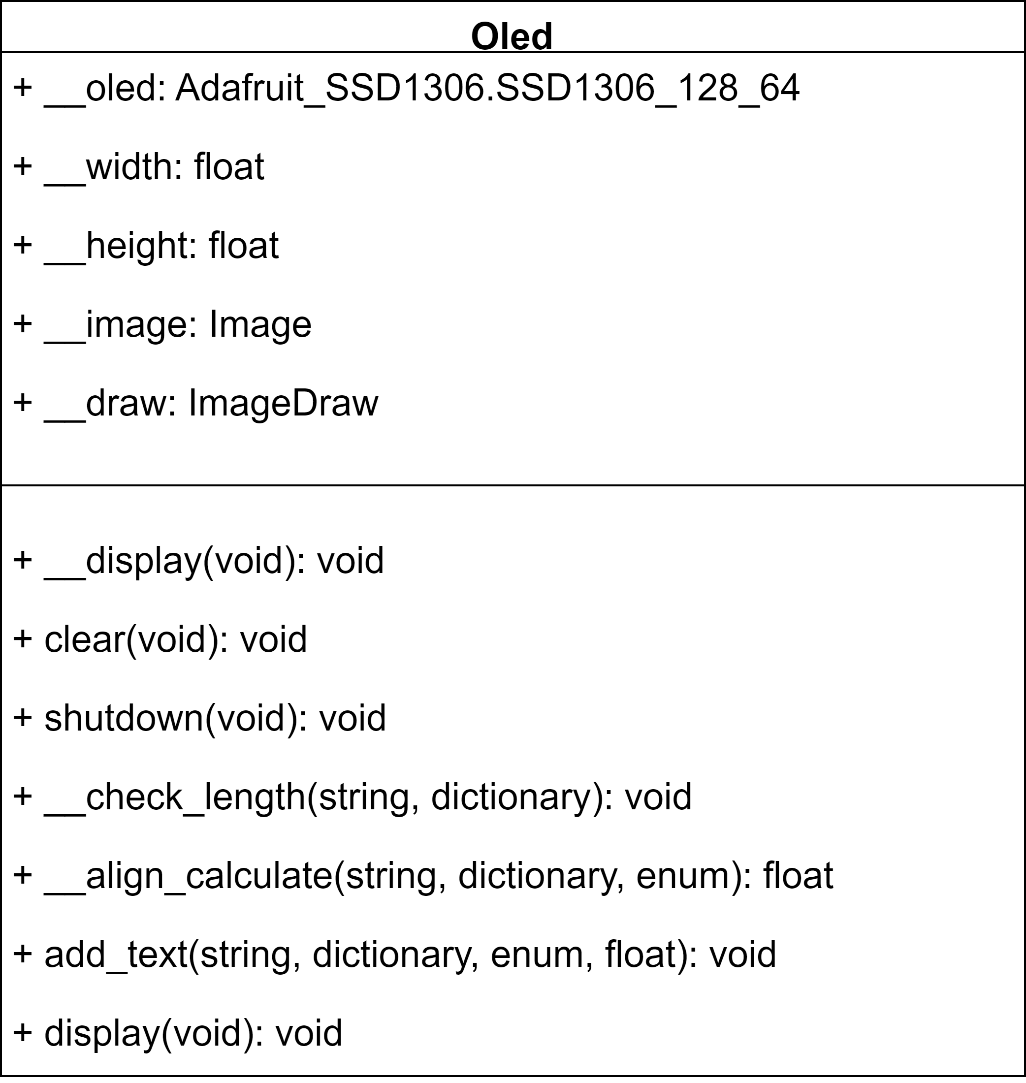


Figure : Raspberry Pi 4 Firmware – Oled class

This program uses Adafruit\_SSD1306 Python library of Tony DiCola & James DeVito from Adafruit Industries. However, it has been marked as deprecated library since 2018. Thus, we create a wrapping class to handle Adafruit\_SSD1306 library more easily.

Our “Oled” class has an crucial variable which is an instance of “Adafruit\_SSD1306” class. “\_\_width” and “\_\_height” variables are the dimension of the OLED display in pixel (128 x 64 pixels). Two variables with the type of “Image” and “ImageDraw” are modules of Python Image Library (PIL) which handles images.

“Oled” class has some methods for displaying texts on the OLED display. "\_\_align calculate", "add text" and “display” are the three major methods of this class. Text strings indicated on the display might have different size and length. Therefore, “\_\_align calculate" method handles aligning content by calculating the pixels value to indent the text string. As its name, “\_\_add\_text” method manages to compose text strings with various font and size. Lastly, the “display” method is used to indicate all the text strings on the OLED display.

#### “OledNode” class

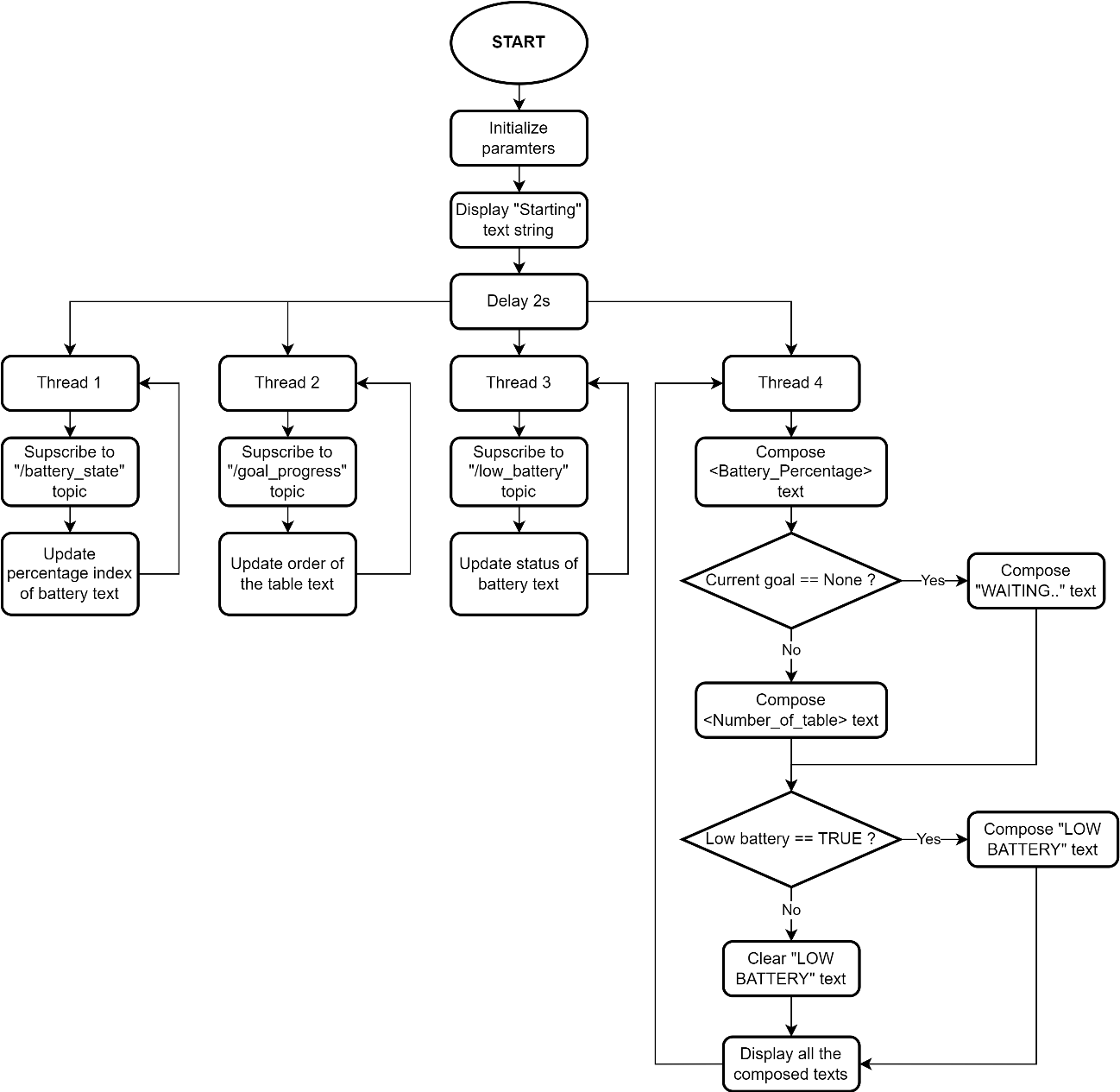


Figure : Flowchart of Raspberry Pi 4 for OledNode node

“OledNode” class is written for communication between OLED display hardware and ROS 2 system. Parameters are initialized in the first procedure. OLED display will indicate “STARTING” text string during the time it starts. After start-up phase, the program is divided into 4 parallel branches which are able to run independently.

For the first thread, the node subscribes to the “/battery\_state” topic to receive message of battery percentage index. It then continuously updates the variable which kept the value of this percentage index.

For the next thread, the node receives the message from “/goal\_progress” topic which is the index of the table that is currently set as the goal point of the robot.

For the third thread, the program checks the low battery signal in a endless loop. It obtains the message from the topic “/low\_battery” in the ROS 2 system.

For the last thread, it is considered as the primary flow of the program. It always displays the current percentage of the battery before checking other conditions. The program then checks whether the current goal is set. If the goal is transmitted, the OLED display will show the index of the table instead of the “WAITING..” text string. The “LOW BATTERY” text string just shows up as the low battery warning signal is sent, otherwise this text string is not shown up on the OLED screen.

LƯU Ý: hình ảnh màn hình OLED trong các trường hợp

## Firmware for infrared sensor

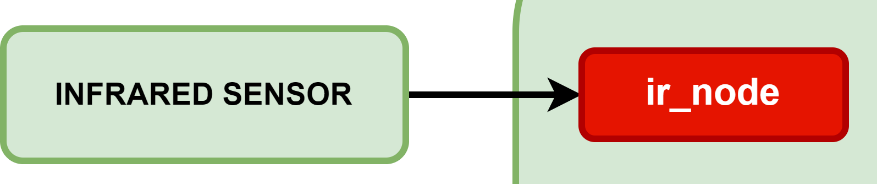


Figure : Infrared sensor block

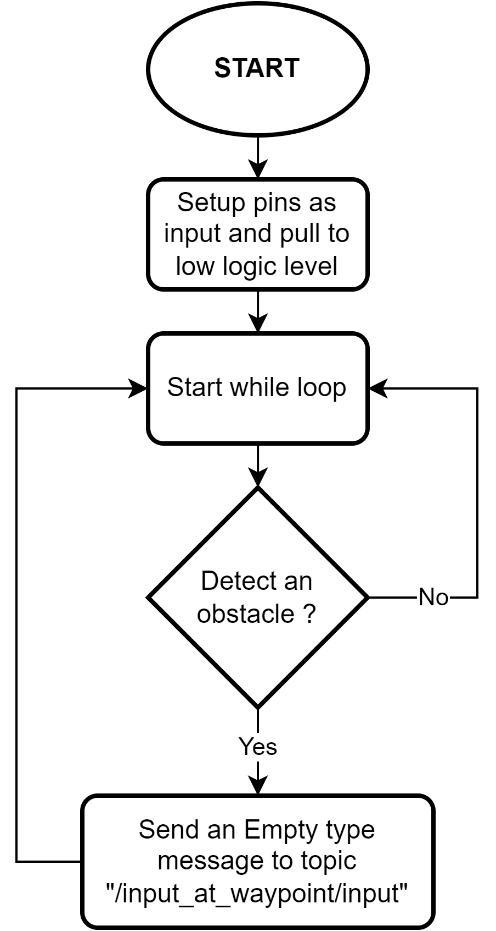


Figure : Flowchart of Raspberry Pi 4 for infrared sensor node

The program starts by setting up the GPIO pin number 4 to receive signal from infrared sensor. This pin is pulled up to avoid the floating state. Whenever an obstacle appears in front of the sensor within its range which is 2.5 cm, a message with the type of Empty is transferred to topic “/input\_at\_waypoint/input”.

# ALGORITHM EXPLANATION FOR FIRMWARE

## PID controller

## Fuzzy logic library

## Kinematic Model for calculating Odometry data

## Kalman filter

## Median filter

# EXPERIMENTAL RESULTS

## Comparison of PID controller and Fuzzy PID controller

## Obstacle avoidance using an array of ultrasonic sensors test cases

## Restaurant serving test cases

## Serving mobile robot design result ?

# REFERENCES

|  |  |
| --- | --- |
| [1] | J. Park, R. Delgado and B. W. Choi, "Real-Time Characteristics of ROS 2.0 in Multiagent Robot Systems: An Empirical Study," vol. 8, pp. 154637-154651, 2020. |
| [2] | Espressif-Systems, *ESP32 Series Datasheet,* 2022. |
| [3] | Toshiba-Cooperation, *TB6612FNG Datasheet,* 2007. |
| [4] | S. Kumar, "geeksforgeeks," 21 6 2022. [Online]. Available: https://www.geeksforgeeks.org/what-is-the-md5-algorithm/. [Accessed 21 5 2022]. |
| [5] | L. Shanghai Slamtec.Co., *RPLIDAR A1 - Low Cost 360 Degree Laser Range Scanner - Introduction and Datasheet,* Shanghai, 2016. |

# APPENDIX

1. **Details of experimental data**
2. **Detailed calculation steps**
3. **Detail simulation diagram**