BCI & EYEGAZE CONTROLLED WHEELCHAIR

WITH A CARE GIVING MOBILE APPLICATION

ROBOBRAIN

Ahmed Samy Kamal Mostafa

Samy Ayman Abbas Ahmed

Ahmed Esmail Ali Mohamed

Ibrahim Mohamed Ahmed Diab

Ahmed Hamada Soliman

Khallad Mohamed Saad Mohamed

Shehab Ashraf Shawqi

Mohamed Abdelaziz Abdullah

Zahraa Ashraf Ebrahim Elsayed

Heba Ahmed Mohamed Mohamed

Nervana Mosaad Abdullah Abdelmonem

Hind Hasan Foaad

Supervisor : Dr/ Ahmed Amer Shahin

A Thesis Presented for the Degree of
Bachelor of Computer and Systems Engineering

Computer and Systems Department
Faculty of Engineering
Zagazig University
Egypt 2022 – 2023

"If you can sense what somebody's trying to do with their limbs, then you can actually do a second implant that's at the base of the spine. Wherever the spinal injury occurs, you can create a neural shunt. So, I think in the long term it will be possible to restore somebody's full body motion." — Elon Musk, NEURALINK

About Us

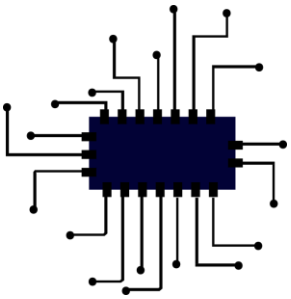
We are a group of technology enthusiasts engineers sharing a passion for challenging real-life issues. We have practical experience in several areas: including machine learning, robotics, embedded systems, and Mobile Application. Our prime focus was to integrate our different skills and experiences to solve a significant problem facing the medical field that requires innovative solutions. We decided to implement a BCI & Eye Gaze -based wheelchair associated with Mobile Application as it allowed us to approach the problem from an embedded and machine learning viewpoint and fully utilize the full potential of our team.

Table of Contents

Hardware

1.1 Introduction

1.1.1 Introduction to Embedded Technology



Embedded systems are specialized computer systems designed to perform specific functions within larger devices or products. They are characterized by their compact size, low power consumption, and ability to interact with the physical world through sensors and actuators.

Embedded systems span various domains, including consumer electronics, medical devices, and industrial automation. Their development requires expertise in hardware and software integration, real-time operating systems, and communication protocols.

This chapter introduces the fundamental principles of embedded systems, highlighting their significance in modern technology and paving the way for further exploration in this field.

1.1.2 The Role of Embedded System

The embedded system plays a vital role in our Project system, which involves the development of a multimode wheelchair with a healthcare monitoring system. It serves as the main controller, seamlessly integrating various subsystems together. The specific tasks performed by the embedded system include:

- Reading the selected mode of control from the Mode Selector system.
- Gathering movement direction inputs from the chosen controller based on the selected mode.
- Controlling the motors to enable wheelchair movement according to the inputs received.
- Monitoring the distance to detect obstacles using the Obstacle Avoidance system.
- Activating the buzzer to provide alerts in the presence of obstacles.
- Reading data from medical sensors (such as temperature, SPO2, and heart rate).
- Transmitting the medical sensor data to the Raspberry Pi for further transmission to the application server as part of the healthcare system.
- Handling the Joystick inputs when the Manual Control system is chosen.

In addition, when the ROS (Robot Operating System) is selected as the mode controller, the embedded system also performs the following tasks:

- Reading motor encoders to obtain precise information about the wheelchair's position.
- Utilizing the IMU sensor to capture orientation data for the ROS system.

Overall, the embedded system acts as the central hub for coordinating and controlling the various components and functionalities of the multimode wheelchair, ensuring smooth operation and integration of the entire system.

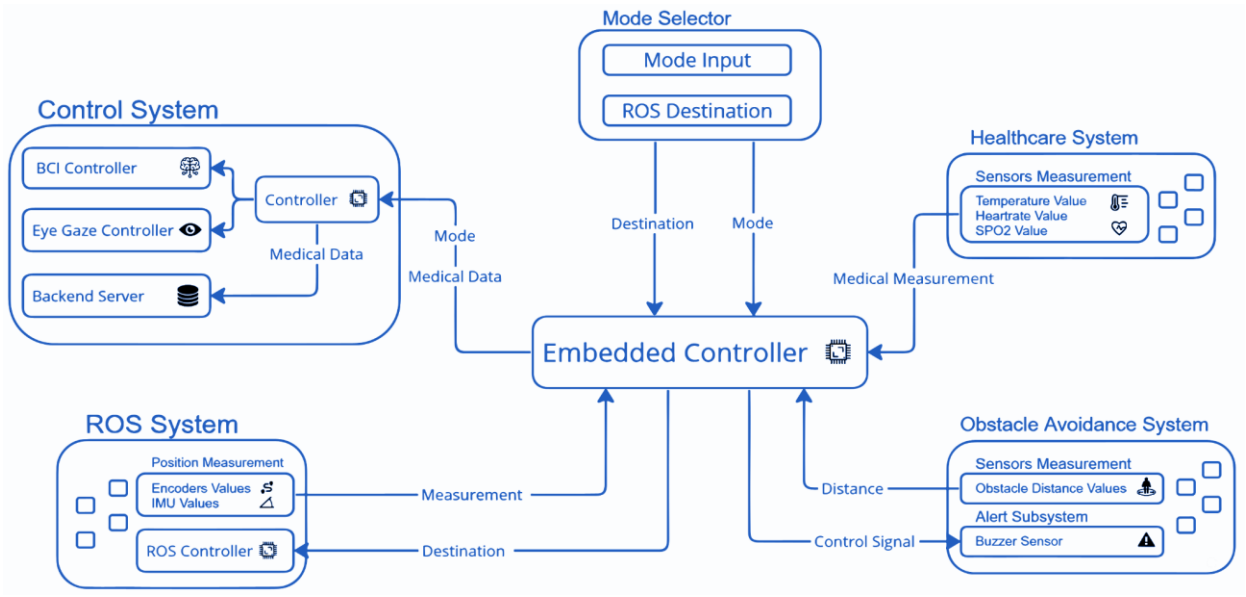


Figure 1 – system design

Figure 2 – system design

1.2 Objectives

The objectives of the embedded system in our graduation project, a multimode wheelchair with a healthcare monitoring system, are as follows:

- Serve as the central controller to seamlessly integrate the various subsystems and components of the wheelchair.
- Read the selected mode of control from the Mode Selector system and interpret the corresponding movement commands from the chosen controller.
- Control the motors to enable precise and responsive wheelchair movement based on the selected control mode.
- Facilitate obstacle detection through the Obstacle Avoidance system, triggering alerts through the Alert subsystem if obstacles are detected.
- Retrieve vital health data from medical sensors, including temperature, SPO2, and heart rate, and transmit it to the Raspberry Pi for onward transmission to the application server for healthcare monitoring.
- Enable manual control through the Joystick in cases where the Manual Control system is selected as the mode.

These objectives highlight the embedded system's role in providing seamless control, integration, obstacle detection, and healthcare monitoring functionalities within the multimode wheelchair.

1.3 Development Process

The implementation of the embedded system will be conducted in two phases, each playing a crucial role in the overall development of the multimode wheelchair project:

a. Prototype Implementation Phase:
During the prototype phase, the initial version of the embedded system will be developed. This stage emphasizes the creation of a functional prototype that demonstrates the core functionalities of the system. An iterative approach may be adopted, allowing for refinement and improvement based on feedback and testing results.

b. STM MCU Implementation Phase:
The subsequent phase focuses on integrating the embedded system with an STM microcontroller unit (MCU). This phase leverages the capabilities of the chosen STM MCU to enhance the performance and functionality of the embedded system, ensuring optimal integration with the multimode wheelchair.

Furthermore, **GIT** will be used as the version control system to organize the code and files. This allows for efficient collaboration, tracking of changes, and maintaining a well-structured development environment throughout the implementation process.

Prototype implementation

2.1 Introduction

The implementation of a prototype serves as a crucial milestone in our graduation project. As an initial step, we constructed a prototype using a small model implemented with Arduino, aiming to transform our conceptual design into a tangible representation. The prototype implementation marks a significant stride towards realizing our innovative solution and assessing its feasibility in a practical context.

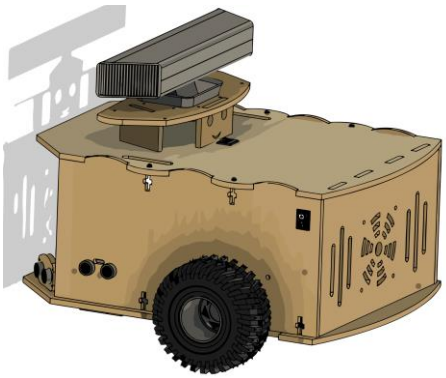


Figure 3 - prototype

Figure 4 - prototype

2.1.1 The importance of Prototype phase

The prototype phase holds immense importance in the development of our project. It serves as a critical stage that allows us to test and validate the core functionalities and design choices wheelchair system. Through the construction of the prototype

Moreover, the prototype phase enables us to gather real-world feedback and user insights, helping us refine and enhance the system's functionality and usability. By engaging potential end-users, we ensure that our design aligns with their needs, preferences, and requirements, ultimately leading to the development of a user-centric solution.

Additionally, the prototype serves as a powerful communication tool, allowing us to showcase our progress, discuss technical aspects, and receive valuable input from experts in the field. The tangible nature of the prototype facilitates effective collaboration, as it enables stakeholders to visualize the envisioned solution and provide constructive feedback.

Furthermore, the prototype phase enables us to identify and address potential design flaws, technical challenges, and limitations early in the development process. This iterative approach promotes efficiency and reduces risks, allowing us to make informed design decisions and optimize the performance of the wheelchair system.

2.1.2 Reasons for Choosing Arduino

We chose Arduino as the development platform for our graduation project due to several key reasons:

- **Open-Source Platform:** Arduino's open-source nature provides access to a diverse community, tutorials, and resources, facilitating implementation.
- **The user-friendly interface:** Arduino is an excellent choice for students and beginners, allowing us to focus on the specific features of our wheelchair project.
- **Extensive Library Support:** Arduino offers a wide range of libraries, saving development time and effort for complex functionalities.

- **Versatility and Scalability:** Arduino supports various sensors, actuators, and communication modules, enabling flexible hardware integration and future scalability.
- **Rapid Prototyping:** Arduino's plug-and-play modules and simplified workflow enable quick prototyping, encouraging experimentation and design iteration.

2.1.3 Arduino Mega Overview

Arduino is an open-source electronics platform that combines microcontroller boards, a user-friendly IDE, and an active community. Its key features include:

- **Configurability:** Arduino provides various board options, each with a specific microcontroller variant, allowing for customization based on project requirements.
- **Form Factor Variety:** Arduino boards come in different sizes and shapes, offering flexibility to accommodate diverse project constraints.
- **Power Flexibility:** Arduino boards can be powered through USB connections, external supplies, or batteries, providing adaptability to different power sources.

2.1.4 Arduino Mega Pinout

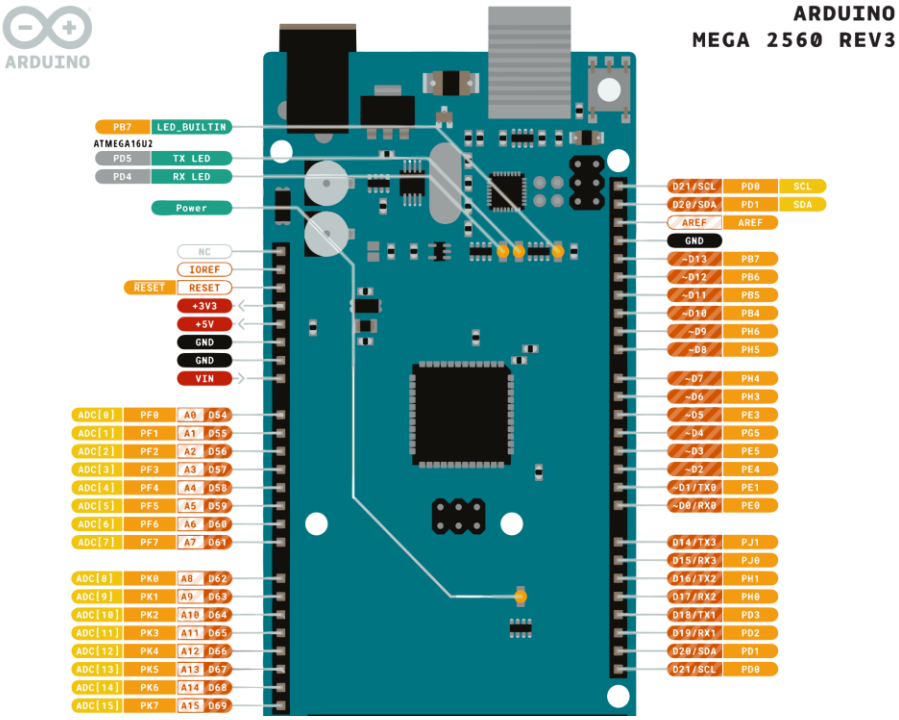


Figure 5 Arduino pin outs

Figure 6 Arduino pin outs

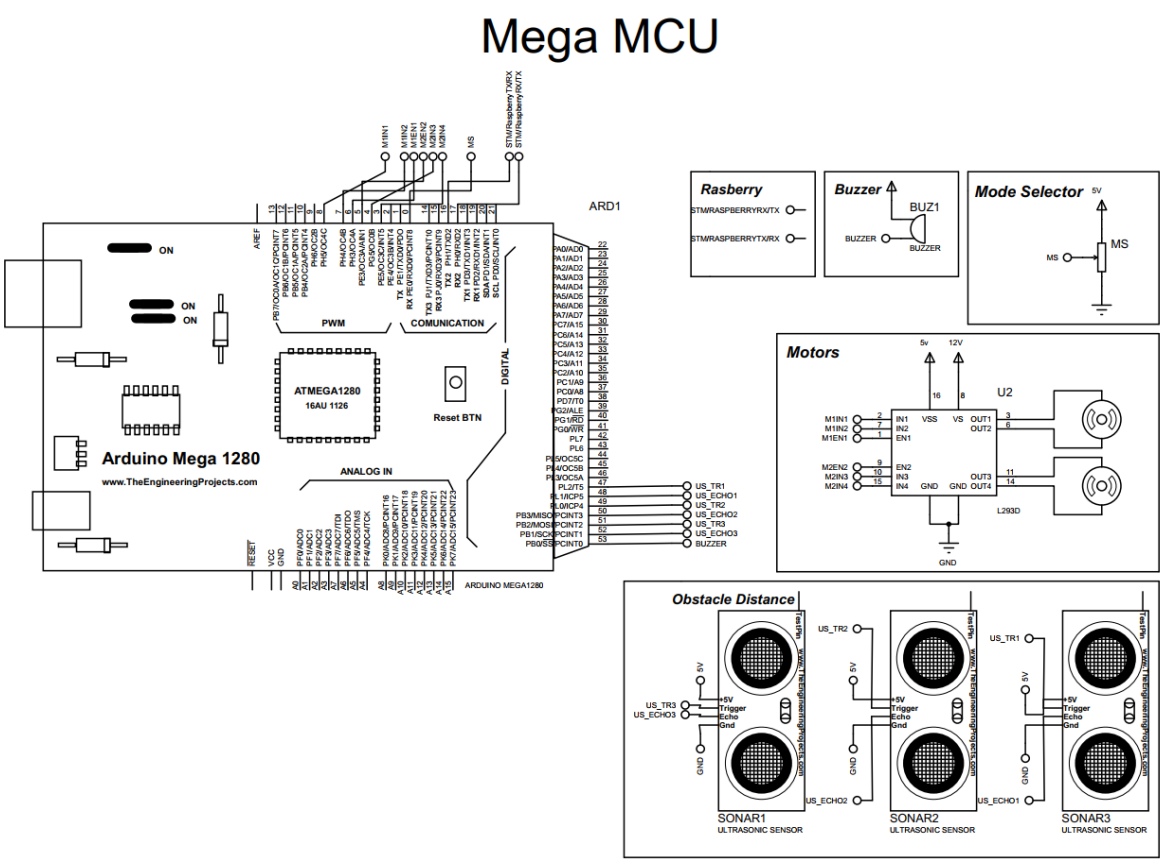
Arduino encompasses a diverse range of functionalities that contribute to its versatility and broad applicability. The following are key functionalities offered by Arduino:

- **Digital and Analog I/O Pins:** Arduino boards provide a substantial number of digital and analog input/output pins, allowing for seamless interfacing with a variety of external devices and components.
- **Communication Interfaces:** Arduino supports multiple communication interfaces, including UART, I2C, SPI, and USB, enabling effective data exchange and integration with other devices.
- **Sensor Integration:** Arduino facilitates the integration of various sensors such as temperature sensors, humidity sensors, accelerometers, and more, enabling data acquisition and environmental monitoring.
- **Actuator Control:** Arduino allows precise control of actuators like motors, servos, LEDs, and displays, empowering physical output and interactive functionalities.
- Wireless Connectivity: Arduino can be equipped with wireless modules or shields, such as Wi-Fi or Bluetooth, enabling wireless communication and remote-control capabilities.
- **Data Processing and Logic:** Arduino's microcontroller executes code written in a simplified variant of C/C++, facilitating data processing, logic implementation, and algorithm execution.
- **Third-Party Libraries:** Arduino boasts an extensive collection of third-party libraries that provide pre-written code for specific functionalities, expediting development and implementation.
- **Expansion Capabilities:** Arduino boards support expansion through shields or add-on modules, allowing for enhanced functionality such as GPS, GSM, NFC, audio processing, and user interface improvements.

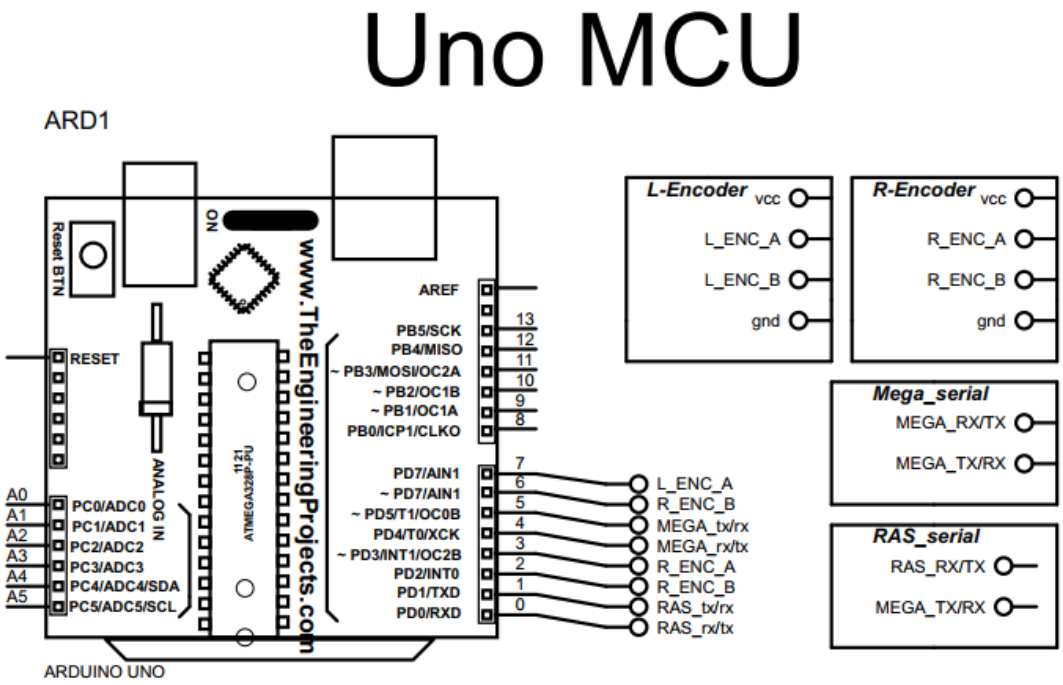
These features collectively contribute to Arduino's flexibility, making it a powerful platform for realizing innovative projects across various domains.

2.2 Schematic Diagram & Pins Connections

2.2.1 Schematic Diagram for Arduino Mega



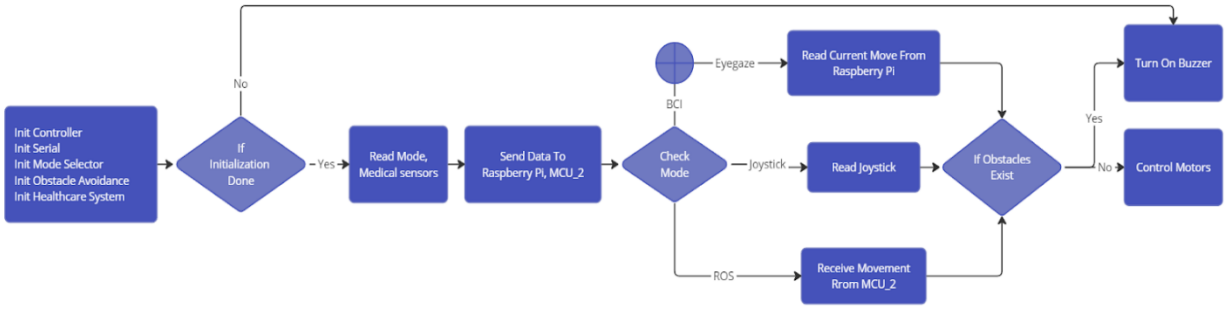
2.2.2 Schematic Diagram for Arduino UNO



2.3.3 Functions Description

2.3.3.1 Flow Chart

Wheelchair system flow chart:



2.3.3.1 Descriptions

These functions are responsible for initializing the system:

```
init_Controller()
init_Serial()
init_Current_Mode_Selector()
init_Obstacle_Avoidance()
init_Healthcare_System()
```

- `read_Current_Mode()`
  Description: The function is used to read and retrieve the current control mode from the current mode selector component in the system. It fetches the selected control mode and provides it as a return value or stores it in a designated variable.

- `send_Data_To_Raspberry()`
  Description: The function is responsible for transmitting data from the system to a Raspberry Pi. It establishes a communication link between the embedded system and the Raspberry Pi and sends the relevant data over the established connection.

- `send_Data_To_Uno()`
  Description: The function is responsible for transmitting data from the system to a Raspberry Pi. It establishes a communication link between the embedded system and the Raspberry Pi and sends the relevant data over the established connection.

- `read_Current_move()`
  Description: The function is responsible for reading and retrieving the current movement or motion information from control inputs in the system.

- `check_Obstacle_Alert()`
  Description: The function is responsible for detecting using ultrasonic sensors and assessing potential obstacles in the environment to ensure safe navigation of the system.

- `controller()`
  Description: The function serves as the central control mechanism. It orchestrates the interaction between various components, sensors, actuators, and algorithms to ensure smooth and reliable operation.

- `read_Oximeter()`
  Description: The function is responsible for retrieving and reading the oxygen saturation level and heart rate data from an oximeter sensor. It accesses the sensor readings and provides the oxygen saturation level and heart rate as output.

- `read_Temperature_Sensor()`
  Description: The function is responsible for retrieving and reading temperature data from a temperature sensor accessing the sensor readings and providing the temperature value as output.

2.3.3.2 libraries dependencies

- **MAX30100_PulseOximeter:** It provides accurate pulse oximetry measurements, flexible configuration options, and supports interrupt-driven operation.

- **SoftwareSerial:** It creates virtual serial ports on digital pins, allowing communication with external devices. It supports adjustable baud rates and multiple software serial ports, making it a valuable tool for expanding serial connectivity on Arduino boards.

2.4.1 Digital Fabrication For Prototype Phase

2.4.1.1 Introduction

The digital fabrication part of this phase focuses on designing and assembling a small prototype for the wheelchair. The prototype integrates electronic components and microcontrollers for healthcare and obstacle avoidance systems to enhance user experience and safety. The project involves developing an advanced electric wheelchair with multiple modes of operation.
Fusion 360 was used for CAD design, creating detailed 3D models of wooden parts, while LaserCAD software generated precise cutting instructions for the laser cutting process.

2.4.1.2 Design Process

- Identify the specific components required for the prototype, including three ultrasonic sensors for obstacle avoidance (left, right, and forward), motors, encoders, and a Kinect camera for the ROS system.
- Consider the stability and mounting requirements of each component when designing the wooden parts. Ensure that the wooden structure can securely hold and stabilize the motors, encoders, and Kinect camera.
- Utilize Fusion 360 to create 3D models of the wooden parts, taking into account the dimensions and placement of each component. Pay attention to proper alignment and mounting points for stability and ease of assembly.
- Iteratively refine the design by testing and evaluating the fit and stability of the components within the wooden structure. Make adjustments as necessary to ensure proper functionality and stability.
- Consider cable management and routing within the design to ensure neat and organized wiring between the components.
- Perform thorough testing and validation to verify that the wooden parts adequately support and stabilize the components during operation. Make any necessary modifications based on the test results.

2.4.1.3 Tools, Software, and Materials

In the development of the prototype, various tools, software, and materials were employed to facilitate the design and fabrication process:

1. **Laser Cutter Machine:**

- A laser cutter machine was utilized to precisely cut the wooden parts for the prototype design.

- This machine enabled clean and accurate cuts, ensuring the components fit together seamlessly.

2. Plywood (5mm):

- 5mm thick plywood was chosen as the material for the design.
- Plywood offers a balance of strength and flexibility, making it suitable for constructing the wooden parts of the prototype.

3. Fusion 360:

- Fusion 360, a comprehensive computer-aided design (CAD) software, was employed for designing the prototype.
- This powerful software facilitated the creation of detailed 3D models, ensuring accurate dimensions and structural integrity.

4. LaserCAD Software:

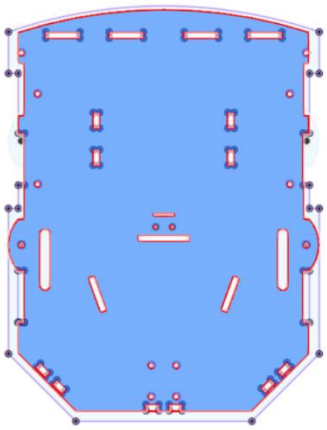
- LaserCAD software played a crucial role in preparing the design files for the laser cutter machine.
- It enabled the generation of precise cutting instructions, ensuring the wooden parts were accurately cut according to the design specifications.

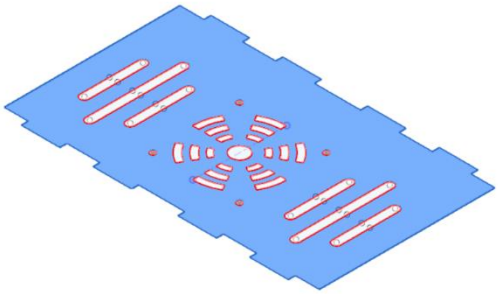
5. Nuts and Nails (M3):

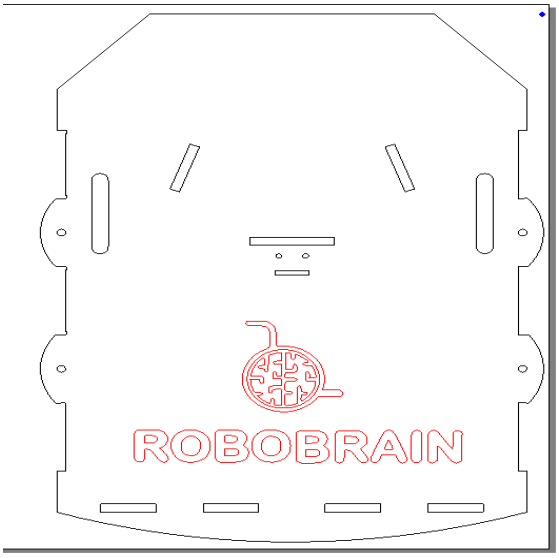
- M3 nuts and nails were utilized for assembly purposes.
- These standard-sized fasteners provided secure connections between the wooden components, ensuring structural stability.

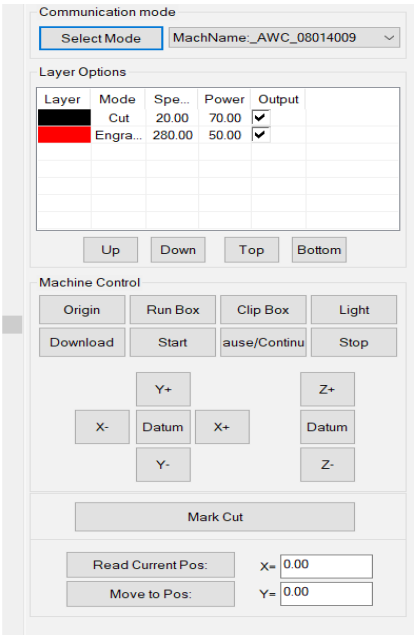
The combination of these tools, software, and materials facilitated the efficient and precise fabrication of the prototype, ensuring accurate cuts, robust construction, and structural stability.

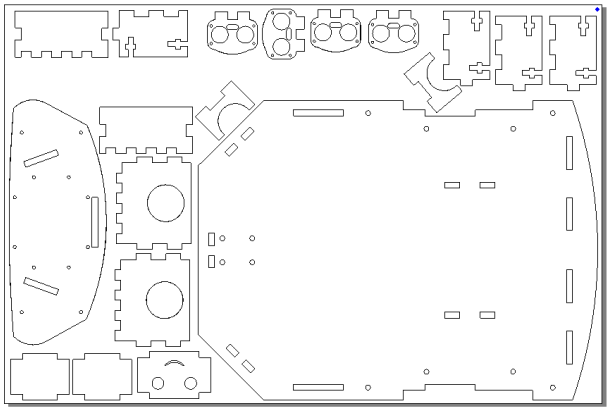
2.4.1.4 Process Visualization

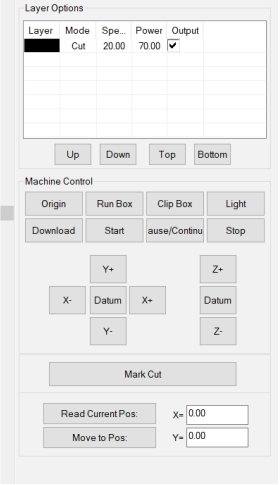




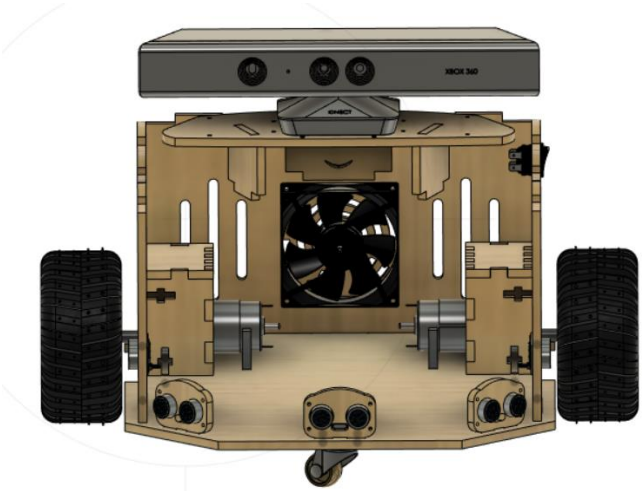


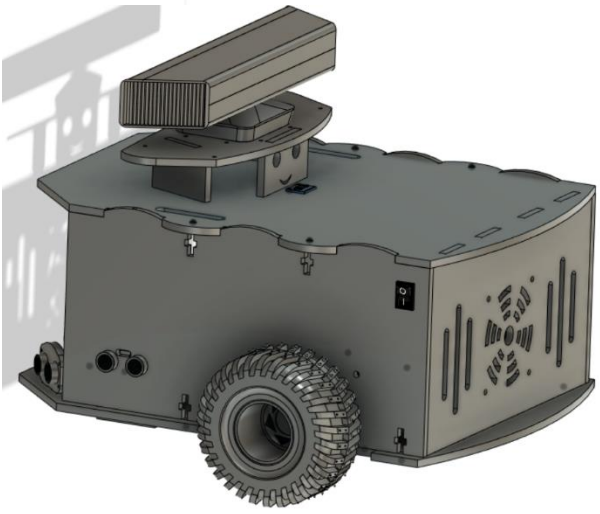






2.4.1.5 Showcase





STM Implementation

3.1 Introduction

3.1.1 Implementation phase

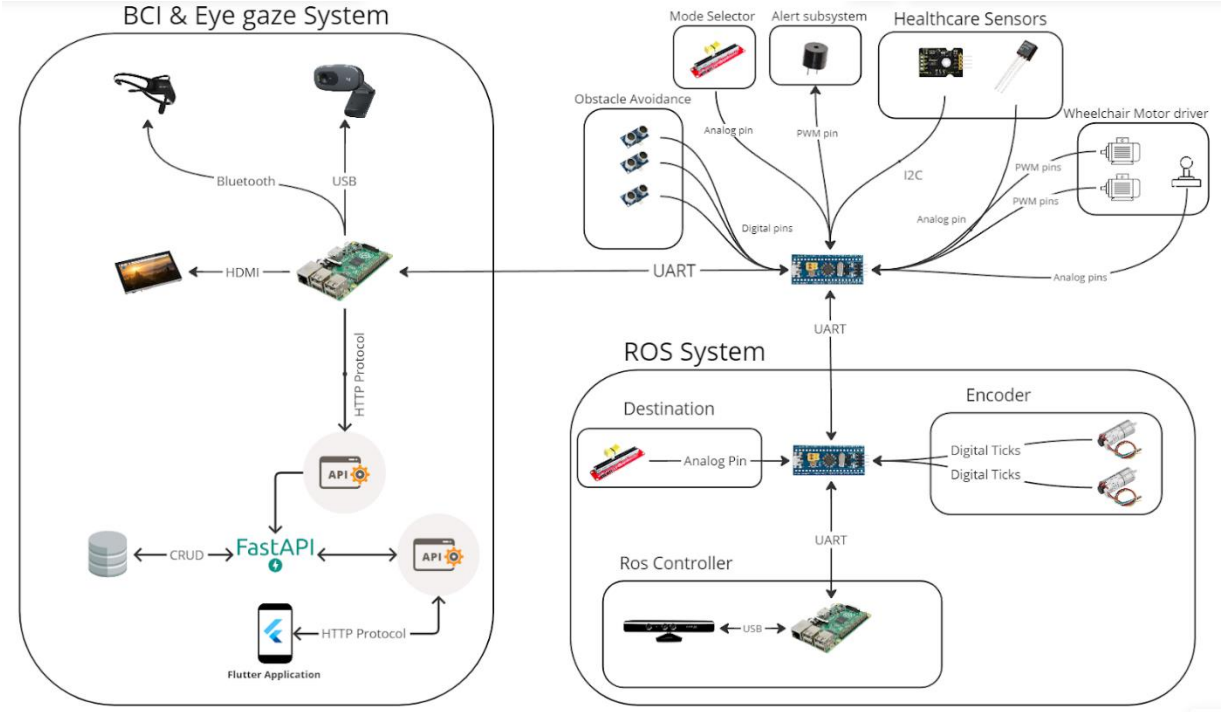


Figure 7 system overview

The implementation phase of our wheelchair project marks a significant milestone in transforming our prototype into a fully functional and reliable system. In this phase, we transition from proof-of-concept to practical realization, deploying the project on the STM32F103C8T6 microcontroller and the physical wheelchair itself.

The implementation phase encompasses the integration of hardware, software, and firmware components, as well as the calibration and fine-tuning of various system parameters. It involves translating the theoretical concepts, algorithms, and designs into tangible and functional features that enable real-world applications.

By migrating our project to the STM32F103C8T6 microcontroller, we harness its advanced processing capabilities, ample I/O options, and robust architecture to handle the intricate computations, sensor data processing, and control algorithms required for BCI and eye gaze control. Additionally, the physical wheelchair serves as the platform for incorporating autonomous driving features, allowing us to demonstrate the practicality and effectiveness of our system in real-life scenarios.

During the implementation phase, we focus on optimizing system performance, ensuring responsiveness, accuracy, and reliability. This involves refining the integration between the microcontroller, the wheelchair's actuators, and the various sensors such as the BCI, eye gaze tracking, encoder, and environmental sensors. Additionally, we address power management,

communication protocols, and safety considerations to guarantee the smooth and safe operation of the wheelchair.

Through the diligent execution of the implementation phase, we aim to bring our BCI and eye gaze-controlled wheelchair with autonomous driving to a tangible solution that holds the potential to enhance the mobility and independence of individuals with physical disabilities.

3.1.2 Reasons for Choosing STM MCU

1. Powerful Processing Capability: The STM microcontroller units (MCUs) offer robust processing capabilities, making them suitable for complex tasks in our BCI and eye gaze-controlled wheelchair project.
2. Extensive I/O Options: STM MCUs provide a wide range of I/O options, facilitating seamless integration with sensors, actuators, and communication modules.
3. Reliable and Robust: STM MCUs are known for their reliability and resilience in demanding environments.
4. Active Developer Community: STM MCUs have a large and active developer community, providing valuable resources and support.

3.1.3 STM Overview

- Model: STM32F103C8T6.
- Core: ARM Cortex-M3.
- Clock Speed: Up to 72 MHz
- Flash Memory: 64 KB.
- SRAM: 20 KB.
- GPIO Pins: Multiple GPIO pins for interfacing with external components.
- Communication Interfaces: UART, I2C, SPI for seamless communication.
- ADC Channels: Precise analog signal acquisition.
- Timers: Accurate timing control and PWM generation.
- Real-Time Clock (RTC): Timekeeping and scheduling functionality.
- Low-Power Modes: Power optimization features.
- Development Tools: STM32Cube software development platform, integrated development environments (IDEs) like STM32CubeIDE.

3.1.4 STM32f103C8T6 Pinouts

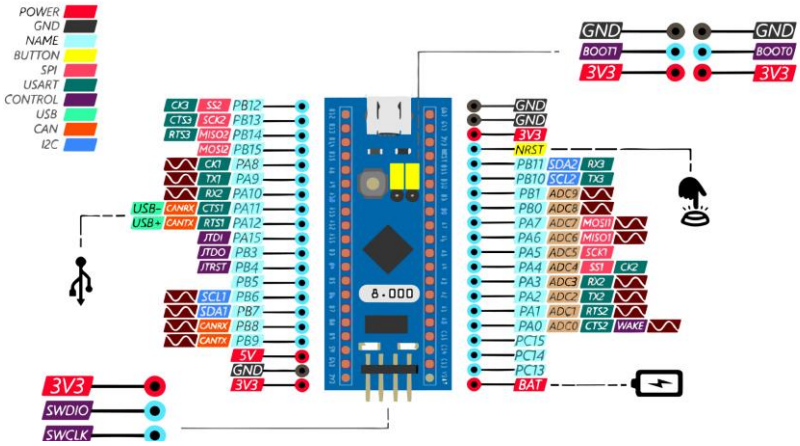


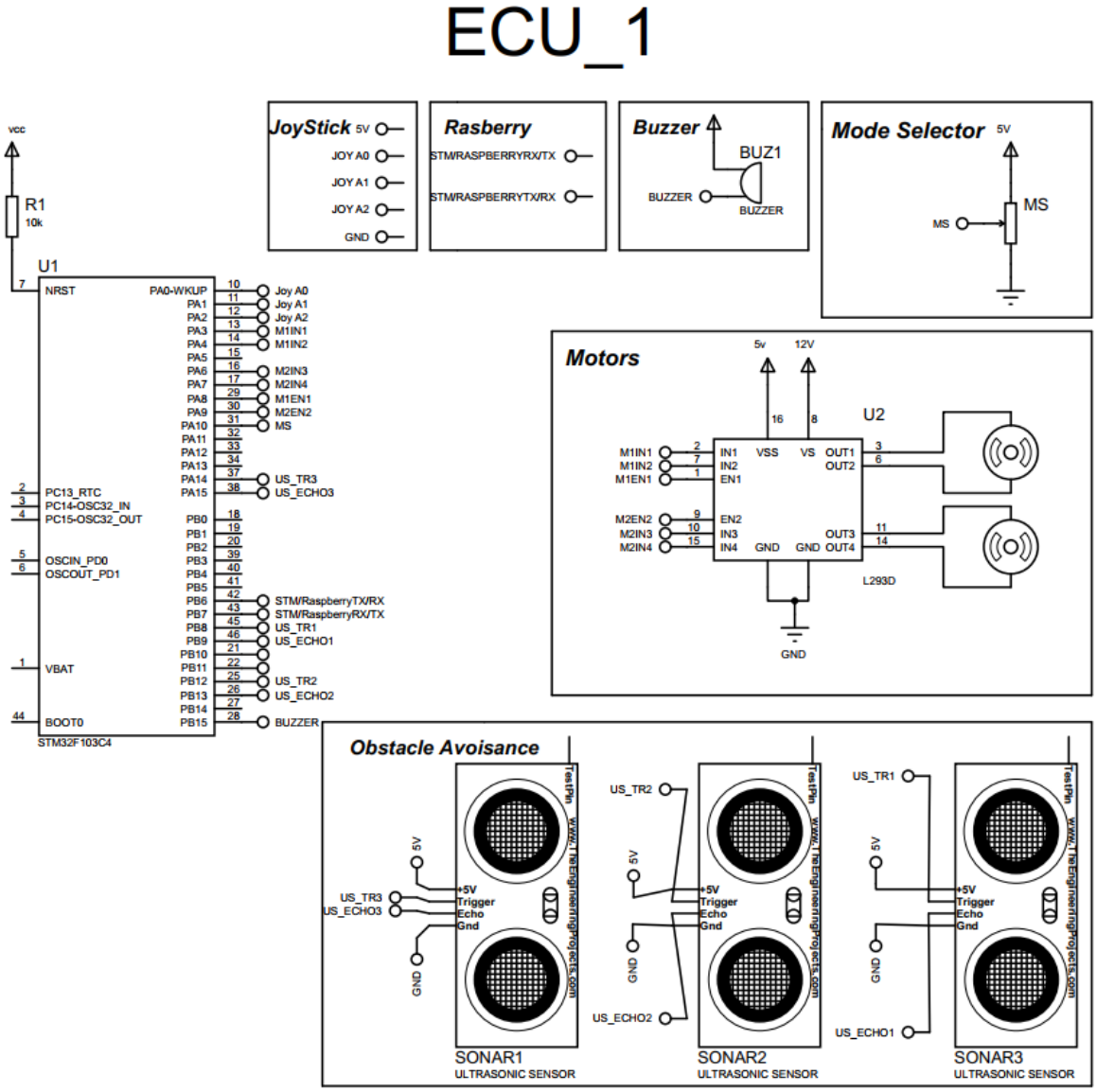
Figure 8 - STM32f103C8T6

Configurations and Functionalities:

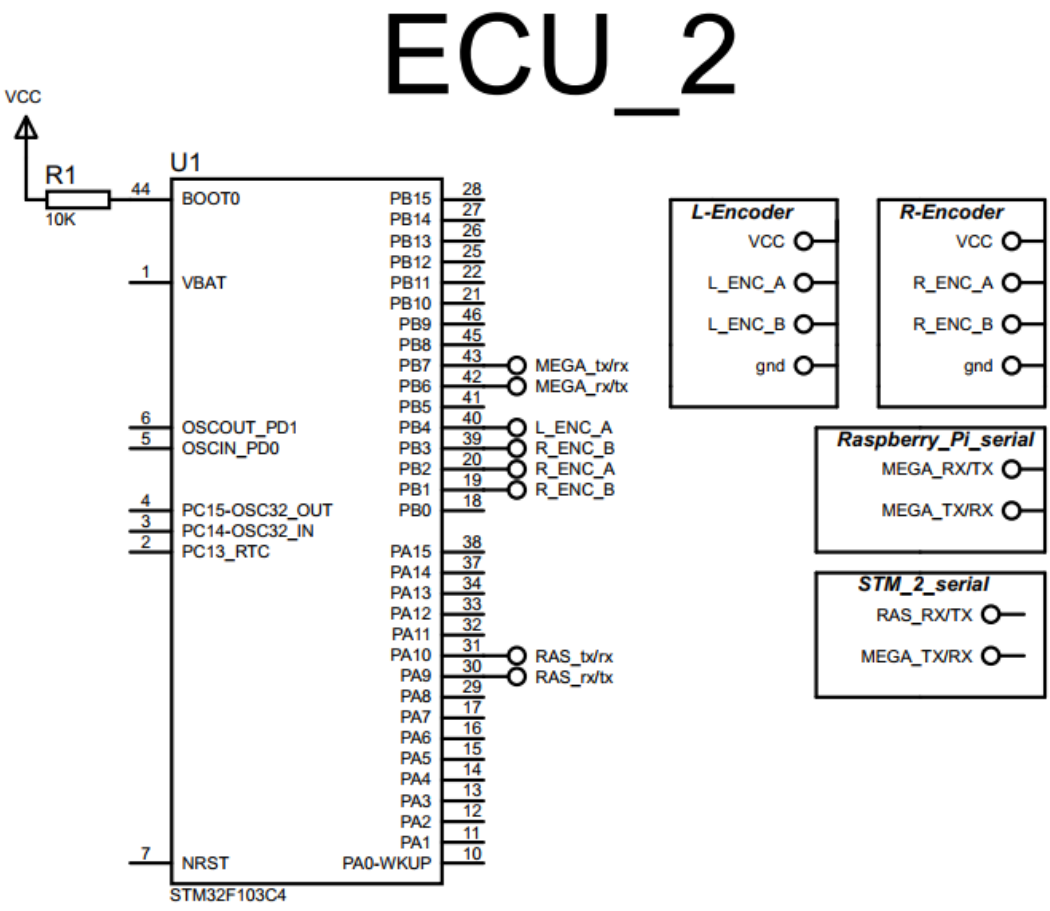
1. GPIO Pin Configuration: The GPIO pins on the STM MCU can be configured as inputs or outputs to interface with various components such as sensors, actuators, and external devices. They allow for digital signal reading or output control based on the application requirements.
2. Communication Interface Configuration: The STM MCU provides multiple communication interfaces such as UART, I2C, and SPI. These interfaces enable seamless communication with other devices, modules, or microcontrollers, facilitating data exchange and control commands.
3. Analog-to-Digital Conversion: The STM MCU incorporates Analog-to-Digital Converters (ADC) that allow for precise measurement and conversion of analog signals into digital values. This functionality is useful for capturing sensor readings or other analog inputs accurately.
4. Timer Configuration: The STM MCU features built-in timers that offer various functionalities, including accurate timing control, generation of PWM (Pulse Width Modulation) signals, or implementing time-based operations such as delays or periodic tasks.
5. Interrupt Configuration: The STM MCU supports interrupt functionality, allowing for efficient handling of external events or triggering specific actions in response to interrupt requests from external devices or components.
6. Low-Power Modes Configuration: The STM MCU provides different low-power modes, such as Sleep or Standby, that can be configured to minimize power consumption when the system is in an idle or low activity state. These modes help optimize battery life and energy efficiency.
7. Peripherals Configuration: The STM MCU offers various peripherals, including SPI, I2C, UART, PWM, and more, which can be configured and utilized to interface with specific devices or modules required for your BCI and eye gaze-controlled wheelchair project.

3.2 Schematic Diagram & Pins Connections

3.2.1 Schematic Diagram for MCU1



3.2.2 Schematic Diagram for MCU2



3.3 Software Architecture

3.3.1 Module Organization

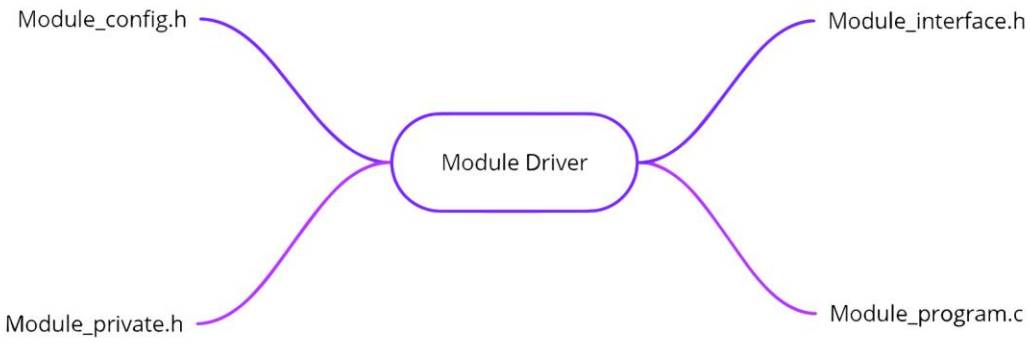


Figure 9 - module driver

The division of the STM driver into four folders, namely config.h, private.h, interface.h, and program.c, serves a specific purpose in organizing and structuring the driver implementation. Here's an explanation of each folder's purpose:

config.h: The config.h file typically contains configuration settings, macros, and constants related to the specific module or peripheral being controlled by the driver. It provides a centralized location to define and modify the parameters that affect the behavior and functionality of the module.

private.h: The private.h file includes private function prototypes, data structures, and other internal declarations that are specific to the driver implementation. It serves as a header file for the internal functions and structures used within the driver module.

interface.h: The interface.h file contains the public interface or API (Application Programming Interface) of the driver. It defines the functions, structures, and constants that are intended to be accessed by other modules or applications that utilize the driver. This file acts as a contract or documentation for using the driver's functionalities.

program.c: The program.c file holds the actual implementation of the driver's functionality. It contains the definitions of the functions declared in the interface.h file, along with any necessary auxiliary functions or helper routines. This file is responsible for the logic and operations required to interact with the specific module or peripheral being controlled by the driver.

By dividing the STM driver into these four folders, it becomes easier to manage and maintain the codebase. Each folder serves a distinct purpose, allowing for a clear separation of concerns and facilitating code organization.

3.3.2 Layer Architecture

In a typical embedded software architecture, the layers are structured hierarchically to provide a modular and organized approach to development. Here's an explanation of the purpose and responsibilities of each layer, along with naming conventions commonly used:

Library Layer (Lib):

- Purpose: The library layer consists of reusable software libraries that provide generic functionalities and utilities. These libraries are designed to be independent of the hardware platform and application-specific requirements.
- Responsibilities: The library layer offers a collection of functions, data structures, and algorithms that can be utilized by higher layers. It abstracts low-level operations and provides higher-level abstractions to simplify development and promote code reuse.
- Naming Convention: The naming convention for library functions and modules typically follows a descriptive and consistent approach, indicating the functionality provided by each module.

Microcontroller Abstraction Layer (MCAL):

- Purpose: The MCAL layer serves as an abstraction between the hardware platform (microcontroller) and the higher-level software layers. It provides a uniform interface to interact with the underlying hardware, regardless of the specific microcontroller being used.
- Responsibilities: The MCAL layer handles low-level hardware operations, such as initializing and configuring peripherals, managing interrupts, and accessing the hardware registers. It provides hardware-specific drivers and APIs for efficient utilization of the microcontroller's features.
- Naming Convention: The naming convention in the MCAL layer often includes prefixes or suffixes that identify the specific microcontroller or peripheral being accessed. This helps differentiate between similar functionalities on different hardware platforms.

Hardware Abstraction Layer (HAL):

- Purpose: The HAL layer abstracts the hardware-specific details of the microcontroller, providing a standardized interface for higher-level software layers. It enables the development of portable code that can be easily migrated across different microcontrollers within the same family or series.

- Responsibilities: The HAL layer provides a set of functions and APIs that encapsulate the microcontroller's hardware features, such as GPIO, UART, SPI, timers, and interrupts. It offers a consistent and uniform interface for application development, ensuring hardware portability.
- Naming Convention: The naming convention in the HAL layer often follows a standard naming convention defined by the microcontroller vendor, ensuring consistency across different peripheral modules.

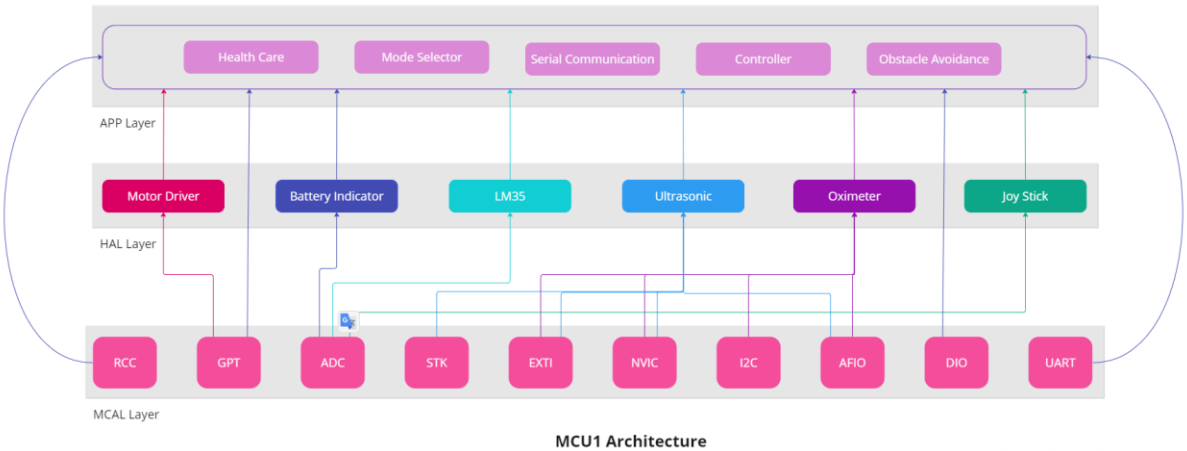
Application Layer (App):

- Purpose: The application layer is the topmost layer in the software architecture, where the actual application-specific logic and functionality reside. It includes the main program, task management, and higher-level algorithms and functionalities specific to the application domain.
- Responsibilities: The application layer handles the overall system behavior, coordinates the interaction between different modules, and implements the specific requirements of the target application. It utilizes the services provided by lower layers to achieve the desired functionality.
- Naming Convention: The naming convention in the application layer depends on the specific requirements and conventions of the development team or project. It is recommended to follow descriptive and meaningful names that reflect the purpose and functionality of each module or component.

Overall, the layering approach in embedded software development promotes modularity, reusability, and maintainability. It enables the separation of concerns, facilitates code organization, and allows for easier troubleshooting and future enhancements.

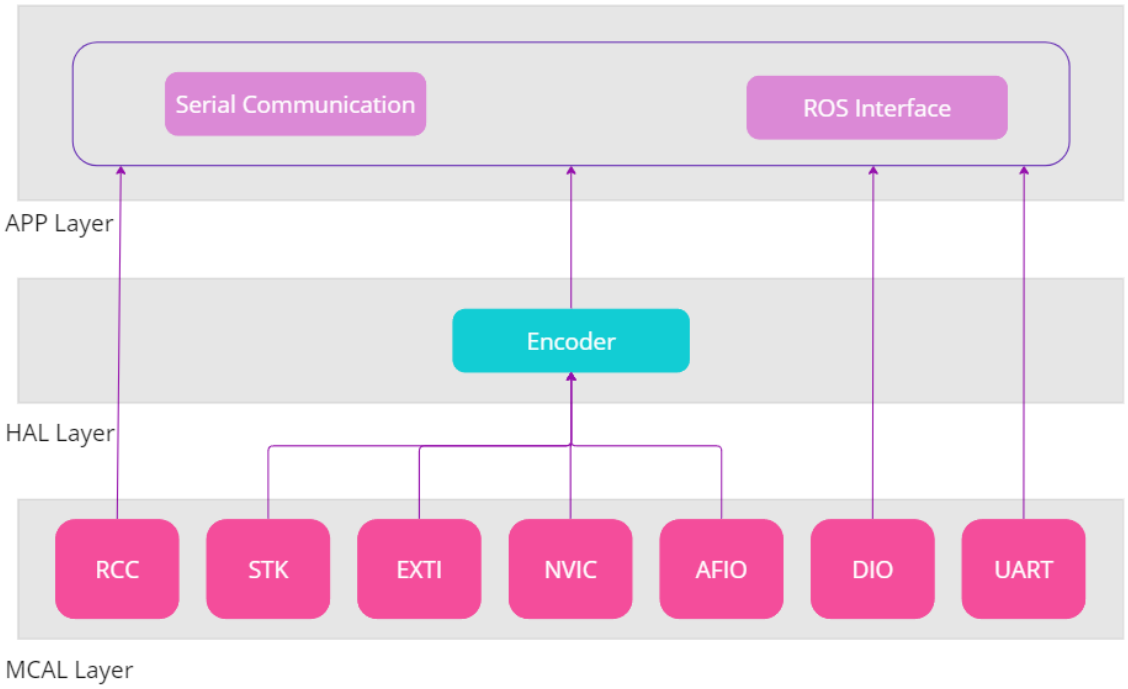
3.3.3 Architecture diagram

MCU1 Architecture:



MCU1 Architecture

MCU2 Architecture:



MCU2 Architecture

3.4 MCAL Layer

3.4.1 RCC Module

3.4.1.1 Overview

Introduction:
The Reset and Clock Control (RCC) module is a crucial component of the STM32F103 microcontroller. It manages system reset and clock functions, ensuring reliable startup and precise control over clock frequencies. As an embedded system software engineer, understanding the RCC module is vital for configuring and controlling the microcontroller effectively.

RCC Module Features:

- Reset Management: Controls system reset process for reliable startup.
- Clock Source Selection: Enables selection of HSE, HSI, and LSI as clock sources.
- Clock Frequency Adjustment: Provides prescalers and dividers for fine-tuning clock frequencies.
- Clock Source Status Flags: Indicates stability and availability of selected clock sources.
- Clock Output: Supports clock synchronization with external devices.
- Peripheral Clock Control: Allows enabling/disabling of clocks for specific peripherals.

3.4.1.2 Contribution

The RCC module contributes to our embedded part in the following ways:

- Clock Generation.
- Power Management.
- System Initialization.
- Clock Source Selection.
- Clock Frequency Configuration.

3.4.1.3 Configuration

To configure RCC for the Stm microcontroller, the following settings need to be considered:

1. Clock Source Selection.
2. System Clock Prescaler.
3. Clock Configuration.

3.4.1.4 API Functions

RCC Module API Functions:

- `void RCC_initSysClk(void)`
  Description: The function initializes and configures the system clock. It selects the clock source, sets dividers and prescalers, and ensures reliable timing and performance..

- `void RCC_enableClk (uint8 busId, uint8 perId)`
  Description: The function enables the clock signal for a specific peripheral (perId) on a designated bus (busId). It ensures the peripheral can operate properly by activating the required clock.

- `void RCC_disableClk(uint8 busId, uint8 perId)`
  Description: The function deactivates the clock signal for a specific peripheral (perId) on a designated bus (busId). It effectively stops the operation of the peripheral by disabling its clock.

3.4.2 DIO Module

3.4.2.1 Overview

Introduction:
Digital Input/Output (DIO) is a key feature of the STM microcontroller, allowing for versatile interfacing with the digital world. DIO pins can be individually configured as inputs or outputs, enabling the microcontroller to receive or send digital signals. With configurable pin modes and interrupt capabilities, the STM32F103 offers efficient control and manipulation of digital signals for various applications.

DIO Module Features:

- Digital Input Capability: DIO pins receive signals from external devices.
- Digital Output Capability: DIO pins drive signals to control external devices.
- Pin Mode Configuration: Configurable pin modes (push-pull, open-drain) for flexible signal driving.
- Interrupt Capabilities: Pins generate interrupts for event-driven programming.
- GPIO Peripheral: Simplifies configuration and control of DIO pins.
- Versatility: DIO pins handle various applications such as sensing, actuation, and communication.
- Efficiency: Efficient control and power-conscious programming for optimal resource usage.

3.4.2.2 Contribution

The DIO module contributes to our embedded part in the following ways:

- Input Handling.
- Output Control.
- GPIO (General-Purpose Input/Output).
- Pin Configuration.
- I/O Voltage Levels

3.4.2.3 Configuration

To configure DIO for the Stm microcontroller, the following settings need to be considered:

1. Pin Direction: Set the direction of the GPIO pins as either input or output.
2. Pull-Up/Pull-Down Configuration: Configure the internal pull-up or pull-down resistors for input pins.
3. Alternate Functions: Some GPIO pins of the STM microcontroller can serve alternate functions beyond general-purpose input/output.
4. GPIO Speed: Set the desired speed of the GPIO pins based on the application's timing requirements.
5. I/O Voltage Levels: Ensure that the voltage levels of the GPIO pins are compatible with the requirements of the connected external devices.

3.4.2.4 API Functions

DIO Module API Functions:

- `void DIO_setPinDirection(uint8 Port, uint8 Pin, uint8 Mode)`
  Description: The function sets the direction of a specific pin (Pin) on a designated port (Port) for Digital Input/Output (DIO) operations. The Mode parameter determines whether the pin is configured as an input or output.

- `void DIO_setPinValue(uint8 Port, uint8 Pin, uint8 Value)`
  Description: The function sets the logic value (Value) of a specific pin (Pin) on a designated port (Port) for Digital Input/Output (DIO) operations.

- `uint8 DIO_getPinValue(uint8 Port, uint8 Pin)`
  Description: The function retrieves the logic value of a specific pin (Pin) on a designated port (Port) for Digital Input/Output (DIO) operations. It returns the current logic level of the pin as either HIGH or LOW.

- `void DIO_setMultiPinsDirection(uint8 Port, uint8 startPin, uint8 Numpins, uint8 Mode)`
  Description: The function sets the direction or mode (Mode) of multiple pins starting from a specified pin (startPin) on a designated port (Port) for Digital Input/Output (DIO) operations. It allows for efficient configuration of multiple pins in a contiguous range.

- `void DIO_setMultiPinValue(uint8 Port, uint8 StartPin, uint8 NumPins, uint16 Value)`
  Description: The function sets the logic value (Value) of multiple pins starting from a specified pin (StartPin) on a designated port (Port) for Digital Input/Output (DIO) operations. It allows for efficient simultaneous control of multiple pins within a contiguous range.

- `uint16 DIO_getMultiPinValue(uint8 Port, uint8 StartPin, uint8 NumPins)`
  Description: The function retrieves the logic values of multiple pins starting from a specified pin (StartPin) on a designated port (Port) for Digital Input/Output (DIO) operations. It returns a 16-bit value where each bit represents the logic state of a pin, allowing for efficient retrieval of multiple pin statuses.

3.4.3 AFIO Module

3.4.3.1 Overview

Introduction:
The AFIO (Alternate Function Input/Output) module in the STM32F103 microcontroller provides a means to configure and control the alternate functions of GPIO pins. It allows to assign different functionalities to GPIO pins, such as UART, SPI, or I2C, in addition to their default input/output capabilities. The AFIO module offers flexibility and enables the microcontroller to support various communication protocols and interface with external devices efficiently.

AFIO Features:

- Pin remapping: AFIO allows changing pin assignments for specific hardware requirements.
- Multiple alternate functions: AFIO supports various functions for GPIO pins, including serial communication, timers, and interrupts.
- External interrupt configuration: AFIO enables configuring external interrupts on specific GPIO pins.
- Event output capability: AFIO allows generating events based on specific conditions.
- Enhanced performance and flexibility: AFIO optimizes GPIO pin performance and offers configurable speed and drive strength.
- Pin configuration locking: AFIO provides the option to lock pin configurations for stability and security.

3.4.3.2 Contribution

The AFIO module contributes to our embedded project in the following ways:

- configure an external interrupt line for utilizing ultrasonic devices.
- configure an external interrupt line for allowing the utilization of encoders for accurate position tracking.
- configure an external interrupt line for the Max30102 Oximeter.

3.4.3.3 Configuration

To configure AFIO for the Stm microcontroller, the following settings need to be considered:

1. Enable the necessary RCC peripheral(s) before using this driver.
2. Only Lines from 0 to 4 are available in the STM32f103c6.
3. Only ports A, B, and C are supported by the STM32f103c6.

3.4.3.4 API Functions

AFIO Module API Functions:

- `void AFIO_setEXTIConfig(uint8 Line ,uint8 PortMap)`
  Description: The function is used to configure an external interrupt line on a GPIO port. It takes two parameters: Line, which represents the ID of the interrupt line, and PortMap, which represents the ID of the GPIO port to be mapped. This function allows you to set up and assign specific GPIO ports to handle external interrupts efficiently.

3.4.4 EXTI Module

3.4.4.1 Overview

Introduction:
The EXTI (External Interrupt) module in STM32F103 microcontrollers provides a mechanism for handling external events and interrupts. It allows to configure and control interrupt lines for GPIO pins, enabling the microcontroller to respond to external triggers and perform actions based on specific events. The EXTI module enhances the flexibility and responsiveness of the microcontroller in interacting with the external environment.

EXTI Features:

- Multiple interrupt lines: EXTI supports handling interrupts from various GPIO pins.
- Edge detection: EXTI detects rising and falling edges for interrupt triggering.
- Interrupt request management: EXTI enables enabling/disabling interrupt lines, prioritizing interrupts, and clearing pending flags.
- Wakeup from low-power modes: EXTI allows waking up from low-power modes on external interrupts.
- Shared interrupt lines: EXTI manages shared interrupts and identifies the triggering GPIO pin.
- Interrupt nesting and preemption: EXTI supports interrupt priority and allows higher priority interrupts to interrupt lower priority ones.
- Debouncing: EXTI offers hardware debouncing to eliminate spurious interrupts caused by signal fluctuations or noise.
- Generation of up to 20 software event/interrupt requests.

External interrupt/event controller block diagram:

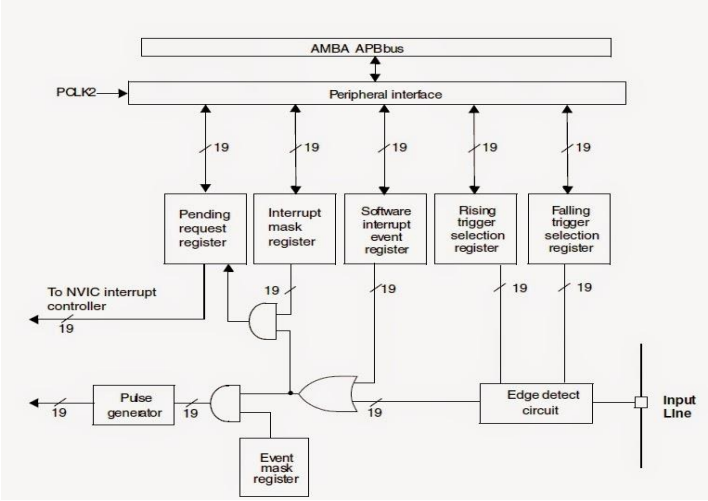


Figure 10 - external event controller BD

3.4.4.2 Contribution

The EXTI module contributes to our embedded project in the following ways:

- Enable an external interrupt line and set the callback function for utilizing ultrasonic devices.
- Enable an external interrupt line and set the callback function for allowing the utilization of encoders for accurate position tracking.
- Enable an external interrupt line and set the callback function for the Max30102 Oximeter.

3.4.4.3 Configuration

To configure EXTI for the Stm microcontroller, the following settings need to be considered:

1. Enable the RCC peripheral clock for AFIO.
2. Configure the needed external interrupt lines.
3. Enable the NVIC for the EXTI IRQ.
4. Set the callback function before enabling the external interrupt.

3.4.4.4 API Functions

EXTI Module API Functions:

- **void EXTI_enableEXTI(uint8 Line, uint8 Mode)**
  Description: The function is used to enable external line interrupts. It takes two parameters: Line, representing the ID of the external interrupt line to be enabled, and Mode, indicating the mode of the interrupt (such as rising edge, falling edge, or both). This function allows you to activate and configure external line interrupts, enabling the microcontroller to respond to specific events on the specified interrupt line.
- **void EXTI_disableEXTI(uint8 Line)**
  Description: The function is used to disable the external line interrupt specified by the Line parameter. This function stops the microcontroller from monitoring and responding to events on the specified external interrupt line, effectively disabling the interrupt functionality for that line.
- **void EXTI_setSWTrigger(uint8 Line)**
  Description: The function is used to manually trigger the external line interrupt specified by the Line parameter. This function allows the microcontroller to simulate an external event or trigger on the specified interrupt line, initiating the interrupt handling routine as if it were triggered by an actual external event.
- **void EXTI_setCallBack(uint8 EXTINumber, void (*ptr)(void))**
  Description: The function is used to set the callback function for an external line interrupt. It takes two parameters: EXTINumber, which represents the number of the external interrupt line, and ptr, which is a pointer to the callback function. This function allows you to define a custom function that will be executed when the specified external interrupt occurs, providing a way to handle and respond to the interrupt in a user-defined manner.

3.4.5 NVIC Module

3.4.5.1 Overview

Introduction:

The Nested Vectored Interrupt Controller (NVIC) is a vital component in STM microcontrollers, responsible for managing interrupts and controlling the interrupt priorities. It provides a flexible and efficient way to handle various interrupt sources in the system. The NVIC module allows for easy configuration of interrupt priorities, enabling efficient handling of time-critical events. By using the NVIC, developers can effectively manage and prioritize interrupt-driven tasks, ensuring the smooth execution of their embedded applications.

NVIC Module Features:

- Efficient Interrupt Handling: The NVIC module enables efficient handling of interrupts, ensuring quick response to time-critical events.
- Flexible Interrupt Prioritization: It allows assigning priorities to different interrupt sources, ensuring higher priority interrupts are serviced first.
- Nested Interrupt Support: The NVIC supports nested interrupts, allowing higher-priority interrupts to interrupt lower-priority ones.
- Interrupt Vector Table: It utilizes an interrupt vector table to map interrupt sources to their corresponding ISRs, simplifying the management of multiple interrupts.
- Group and Subgroup Priority: The NVIC module supports the grouping and subgrouping of interrupts, providing fine-grained control over interrupt priority levels.

3.4.5.2 Contribution

The NVIC module contributes to our embedded part in the following ways:

- Enabling external interrupts is essential for effectively utilizing ultrasonic devices.
- Enabling external interrupts for allowing the utilization of encoders for accurate position tracking.

3.4.5.3 Configuration

To configure NVIC for the Stm microcontroller, the following settings need to be considered:

1. The interrupt vector table is configured and loaded in memory.
2. Any necessary initialization for the peripheral before generating the interrupt is performed.

Our specific NVIC configuration for the project:

- We configure the NVIC to have 4 bits for groups and 0 bits for subgroups, resulting in a total of 16 groups and 0 subgroups.

3.4.5.4 API Functions

NVIC Module API Functions:

- **void NVIC_enableInterrupt(uint8 IntNum)**
  Description: The function is used to enable the specific interrupt identified by "IntNum" in the NVIC module, allowing it to trigger the corresponding interrupt service routine (ISR) when the associated event occurs.

- **void NVIC_disableInterrupt(uint8 IntNum)**
  Description: The function disables the specific interrupt identified by "IntNum" in the NVIC module, preventing it from triggering the corresponding interrupt service routine (ISR) when the associated event occurs, thus effectively deactivating the interrupt functionality.

- **void NVIC_setPendingFlag(uint8 IntNum)**
  Description: The function sets the pending flag for the specific interrupt identified by "IntNum" in the NVIC module, indicating that an interrupt request is pending and needs to be serviced by the corresponding interrupt service routine (ISR) when the system allows.

- **void NVIC_clearPendingFlag(uint8 IntNum)**
  Description: The function clears the pending flag for the specific interrupt identified by "IntNum" in the NVIC module, indicating that the interrupt request has been serviced and no longer requires attention from the corresponding interrupt service routine (ISR).

- **uint8 NVIC_getActiveFlag(uint8 IntNum)**
  Description: The function retrieves the active flag status for the specific interrupt identified by "IntNum" in the NVIC module, indicating whether the interrupt is currently active or not.

- **void NVIC_setPriority(uint8 IntNum, uint8 GroubPriority, uint8 SubPriority)**

  Description: The function sets the priority level for the specific interrupt identified by "IntNum" in the NVIC module, allowing for flexible configuration of interrupt prioritization using the provided group priority and sub-priority values.

3.4.6 STK Module

3.4.6.1 Overview

Introduction:

The STK (SysTick) module in STM32F103 microcontrollers is a system timer used for generating periodic interrupts. It provides a 24-bit countdown timer that can be used for various timing and scheduling purposes. The STK module offers a flexible and accurate timing mechanism, allowing precise control over time-based operations and periodic interrupt generation. It is commonly used for tasks such as system tick generation, timekeeping, and creating software delays or timeouts.

STK Features:

- System tick timer: SysTick provides a 24-bit countdown timer for periodic interrupts.
- Configurable interrupt frequency: SysTick allows adjusting the interrupt frequency for precise timing control.
- Interrupt-driven system scheduling: SysTick enables preemptive multitasking and task scheduling based on priorities.
- Timekeeping: SysTick serves as a reliable source for measuring and tracking time intervals.
- Software delays and timeouts: SysTick creates precise software delays or timeouts.
- RTOS support: SysTick works with an RTOS for timing services and task scheduling.
- Low-power operation: SysTick can operate in low-power modes for efficient timekeeping and interrupts.

SysTick timer - control flow:

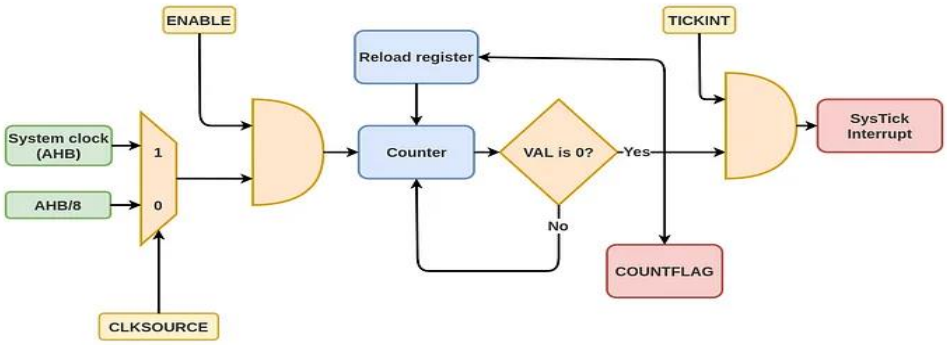


Figure 11 - systick timer

3.4.6.2 Contribution

The STK module contributes to our embedded project in the following ways:

- The SysTick timer provides accurate timing for generating ultrasonic pulses, synchronizing pulse and echo signals, managing timeouts, enabling real-time responsiveness, and optimizing power consumption for the ultrasonic sensors.
- The SysTick timer provides precise timing for measuring the speed and position of a motor. It enables accurate counting of encoder pulses, synchronization with motor control signals, calculation of motor speed and position.

3.4.6.3 Configuration

To configure STK for the Stm microcontroller, the following settings need to be considered:

1. Set number of Ticks for one microsecond.
2. Enable the SysTick.
3. Set theSYSTick availability flag.
4. Configure the SysTick timer with the selected Clock (AHB or AHB/8).

3.4.6.4 API Functions

STK Module API Functions:

- **void STK_init(void)**
  Description: The function is used to initialize the SysTick timer by setting up its parameters and configurations for proper operation.
- **void STK_delayMicroSec(uint32 NoMicroSec)**
  Description: The function introduces a delay in microseconds by utilizing the SysTick timer, allowing for precise timing control in the program.
- **void STK_delayMilliSec(uint32 NoMilliSec)**
  Description: The function introduces a delay in milliseconds by utilizing the SysTick timer, allowing for precise timing control in the program.
- **void STK_setIntervalSingle (uint32 NoMicroSec, void (*ptr)(void));**
  Description: The function configures the SysTick timer to generate a single interrupt after a specified number of microseconds, allowing you to associate a callback function (ptr) with that interrupt.
- **void STK_setIntervalPeriodic (uint32 NoMicroSec, void (*ptr)(void));**
  Description: The function sets up the SysTick timer to generate periodic interrupts at a specified interval in microseconds, allowing to associate a callback function (ptr) with each interrupt.
- **uint8 STK_isSYSTickFree()**
  Description: The function checks if the system Tick is not being used by an interval interrupt and returns 1 if it is available or 0 if it is currently in use.
- **void STK_stopInterval(void)**
  Description: The function stops the SysTick timer from generating interrupts, effectively disabling the periodic interrupt functionality.
- **uint32 STK_getElapsedTime(void)**
  Description: The function returns the amount of time in microseconds that has passed since the last SysTick interrupt, measured in ticks.
- **uint32 STK_getRemainingTime(void)**
  Description: The function returns the remaining time in microseconds until the next SysTick interrupt, measured in ticks.

3.4.7 ADC Module

3.4.7.1 Overview

Introduction:

The ADC (Analog-to-Digital Converter) in STM32F103 microcontroller enables precise conversion of analog signals to digital data. With multiple channels, configurable resolution, and features like sampling time control, it provides flexibility and accuracy for acquiring analog measurements. The ADC module in STM32F103 is a valuable tool for applications requiring analog signal processing and data acquisition.

ADC Module Features:

- Multiple Channels: It supports multiple channels, allowing simultaneous conversion of multiple analog signals.
- Configurable Resolution: It provides configurable resolution options, allowing developers to adjust the precision of the analog-to-digital conversion based on their requirements.
- Sampling Time Control: It allows control over the sampling time, enabling optimization for different analog signal characteristics and improving accuracy.
- Conversion Trigger Options: It supports various conversion trigger options, including software-triggered, hardware-triggered, and continuous conversion modes, offering flexibility in data acquisition.
- Calibration Mechanisms: It includes calibration mechanisms to compensate for inherent errors or variations, ensuring accurate and reliable measurements.

3.4.7.2 Contribution

The ADC module contributes to our embedded part in the following ways:

- Analog Sensor Integration.
- High-Resolution Conversion.
- Multiple Channel Support.
- Calibration and Self-Testing.

3.4.7.3 Configuration

To configure ADC for the Stm microcontroller, the following settings need to be considered:

1. ADC Resolution.
2. Sampling Time.
3. Channel Selection.
4. Reference Voltage.
5. Conversion Mode.

3.4.7.4 API Functions

ADC Module API Functions:

- **void ADC_Init(ADC_Num ADCx, const ADC_InitTypeDef *ADC_Cnfg)**
  Description: The function is used to initialize the ADC module of the STM microcontroller. It takes parameters specifying the ADC instance and a configuration structure. This function sets up the ADC hardware based on the provided configuration, preparing it for analog-to-digital conversions. Once initialized, the ADC is ready to perform conversions according to the specified settings.

- **uint16 ADC_GetValue(ADC_Num ADCx,ADC_channel_t Copy_u8Channel,uint8 Copy_u8SamplingTime)**
  Description: The function retrieves the converted digital value from the specified ADC channel. It initiates a conversion on the given channel with the specified sampling time and returns the resulting 16-bit digital value. This function is useful for real-time analog-to-digital conversion applications.

3.4.8 GPT Module

3.4.8.1 Overview

Introduction:

The General-Purpose Timer (GPT) module in STM is a versatile peripheral that offers precise timing capabilities and event management. It provides various timer modes and synchronization options, enabling accurate time measurement, event handling, and PWM signal generation for embedded system applications.

GPT Module Features:

- Multiple Timer Units: It consists of multiple timer units, allowing for concurrent timing operations and event management.
- Flexible Timer Modes:It supports different timer modes, including input capture, output compare, and PWM generation, catering to various timing requirements.
- Precise Time Measurement: With high-resolution timers,It enables accurate time measurement for tasks such as measuring time intervals or determining time durations.
- Pulse Width Modulation (PWM) Generation: It supports PWM signal generation, allowing for precise control of external devices.
- Wide Timer Range: It offers a wide range of timer resolutions and clock frequencies to accommodate different application requirements..

3.4.8.2 Contribution

The GPT module contributes to our embedded part in the following ways:

- Reading medical sensors and ultrasonic output every interval periodic of time.
- Setting single and periodic intervals .
- Setting period and duty cycle of Pulse Width modulation (PWM).

3.4.8.3 Configuration

To configure GPT for the Stm microcontroller, the following settings need to be considered:

1. The timer mode is selected.
2. Any necessary initialization for the peripheral before generating the interrupt is performed.

Our specific GPT configuration for the project:

- We configure two modes for units by modifying the prescaler value:
    - MilliSeconds for Microseconds and MilliSeconds.
    - Seconds for MilliSeconds and Seconds.

3.4.8.4 API Functions

GPT Module API Functions:

- **void TIM_Init(uint8 TIMx, TIM_ConfigType *TimConfig)**
  Description: The function is used for initializing a Timer/Counter with the provided configuration settings.

- **void TIM_SetBusyWait(uint8 TIMx,uint16 Ticks,uint16 TicksTybe)**
  Description: The function provides a blocking delay using a specified Timer/Counter (TIMx), number of ticks , and tick type.It configures the timer, waits until the desired delay is reached, and then returns.
- **void TIM_SetIntervalSingle (uint8 TIMx, uint16 Ticks, uint16 Copy_u32TicksTybe, void (*Copy_voidpFuncPtr)(void))**
  Description: The function configures a Timer/Counter to generate a single interval or delay. It takes parameters for the number of timer ticks, tick type, and a callback function to execute after the interval elapses.
- **void TIM_SetIntervalPeriodic(uint8 TIMx,uint16 Ticks ,uint16 TicksTybe , void (* Copy_vpFuncPtr) (void))**
  Description: The function configures a Timer/Counter to generate periodic intervals or delays. It takes parameters for the number of timer ticks, tick type, and a callback function to execute at each interval.
- **uint16 TIM_uGetRemainingTime(uint16 TIMx ,direction_m Direction)**
  Description: The Time function calculates and returns the remaining time before the Timer/Counter reaches its next event or timeout. The direction of counting is taken into account for the calculation.
- **void PWM_Init(uint8 TIMx,PWM_ConfigType* PWM_Confnig)**
  Description: The function initializes PWM functionality on a specific Timer/Counter using the provided PWM configuration structure. It configures the timer with the specified PWM settings such as frequency, duty cycle, and polarity.
- **void TIM_SetPeriod (uint8 TIMx , uint32 Period)**
  Description: The function sets the period or duration for a Timer/Counter. It takes the Timer/Counter instance or channel number and the desired period in timer ticks) as input parameters.
- **void PWM_voidSetDutyCycle(uint8 TIMx ,channel_t Channel, uint8 Duty)**
  Description: The function sets the duty cycle of a PWM signal on a specific Timer/Counter and channel. It takes the desired duty cycle as a percentage and adjusts the corresponding register to achieve the desired output.
- **void PWM_DeInit(uint8 TIMx ,channel_t Channel)**
  Description: The function einitializes or reset the configuration of a Pulse Width Modulation (PWM) channel on a specific Timer/Counter.
- **uint16 PWM_GetCounterValue(uint8 TIMx)**
  Description: The function retrieves the current counter value of a Timer/Counter used for Pulse Width Modulation (PWM) operation.

3.4.9 UART Module

3.4.9.1 Overview

Introduction:

UART (Universal Asynchronous Receiver-Transmitter) is a widely used serial communication protocol that enables the transmission and reception of data between devices. The STM microcontroller features a UART interface, which provides a simple and versatile method for asynchronous data communication.

UART Features:

- Enable asynchronous data transmission between devices.
- Provide a widely adopted and straightforward serial communication protocol.
- Facilitate seamless integration with peripheral devices.
- Ensure reliable data transfer in challenging environments.
- Support flexible configuration options for baud rate, data bits, parity, and stop bits.

UART Frame:

- Start Bit: Marks the beginning of a data frame (typically logic level 0).
- Data Bits: Represents the actual data being transmitted (5 to 9 bits).
- Parity Bit (optional): Provides error-checking by allowing error detection.
- Stop Bit(s): Indicates the end of a data frame (typically logic level 1). Usually, one or two-stop bits are used.

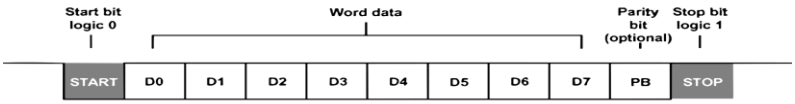


Figure 12 - uart frame

3.4.9.2 Contribution

UART module contributes to our embedded project in the following ways:

- Medical Sensor Data Transmission: Enables the seamless transmission of medical sensor data to the Raspberry Pi for real-time monitoring and analysis.
- Movement Direction Reception: Receives movement direction commands from the Raspberry Pi, allowing dynamic control and adjustment of the embedded system's movement.
- Destination Transmission: Sends destination information from the main microcontroller to the ROS-enabled subsystem, facilitating autonomous navigation and control.

3.4.9.3 Configuration

To configure UART for the Stm microcontroller, the following settings need to be considered:

1. Enable the clock for the UART peripheral using the RCC peripheral.
2. Configure the GPIO pins used for UART communication as alternate function inputs/outputs.
3. Connect the UART pins to the appropriate external device for communication.
4. Set the UART frame configuration.

Our Specific UART Frame Configuration:

- Baud Rate: 9600
- Stop Bits: 1
- Data Bits: 8
- Parity Check: Disabled
- Interrupt/Callbacks: Disabled

3.4.9.4 API Functions

UART Module API Functions:

- **void USART1_init(uint32 baudRate)**
  Description: The function initializes the USART1 module with the specified baud rate. It enables communication between the microcontroller and external devices through USART1. This function is responsible for configuring the necessary settings to establish reliable serial communication at the desired baud rate.

- **void USART1_sendByte(uint8 byteToSend)**
  Description: The function sends a single byte of data through the USART1 module. It allows for the transmission of data from the microcontroller to external devices via the USART1 interface. This function is responsible for sending the specified byte of data reliably and efficiently.

- **uint8 USART1_receiveByte()**
  Description: The function receives a single byte of data through the USART1 module. It allows the microcontroller to retrieve data from external devices via the USART1 interface. This function returns the received byte of data for further processing or analysis.

- **void USART1_sendString(uint8 *stringToSend)**
  Description: The function sends a null-terminated string of characters through the USART1 module. It facilitates the transmission of a sequence of characters from the microcontroller to external devices via the USART1 interface. This function is responsible for sending the specified string reliably and efficiently.

- **void USART1_receiveString(uint8 *strToReceive)**
  Description: The function receives a null-terminated string of characters through the USART1 module. It allows the microcontroller to capture a sequence of characters from external devices via the USART1 interface. This function stores the received string in the specified memory location for further processing or analysis.

3.4.10 I2C Module

3.4.10.1 Overview

Introduction:

I2C (Inter-Integrated Circuit) is a popular serial communication protocol used for connecting integrated circuits in a microcontroller system. In STM microcontrollers, I2C is commonly utilized for communication between different components, such as sensors, memory devices, and other peripherals. It employs a master-slave architecture, where a master device initiates and controls communication with one or more slave devices.

I2C Features:

- I2C (Inter-Integrated Circuit) is a widely used serial communication protocol.
- It allows for the communication between integrated circuits in a microcontroller system.
- I2C utilizes a master-slave architecture, where a master device controls communication with one or more slave devices.
- It supports multi-device connectivity, enabling efficient communication with multiple peripherals.
- I2C is known for its simplicity, efficient use of pins, and ability to transfer data at relatively high speeds over short distances.

I2C Frame:

- Start Condition: Communication begins with a specific start signal (high-to-low transition on the SDA line while the SCL line is high).
- Address Byte: The master device sends the specific address (7-bit) of the target slave device.
- Data Bytes: Multiple specific data bytes (8 bits each) are transmitted or received between the master and slave devices.
- Acknowledgment: The receiver sends a specific acknowledgment bit (ACK) to indicate successful reception.
- Stop Condition: Communication ends with a specific stop signal (low-to-high transition on the SDA line while the SCL line is high).

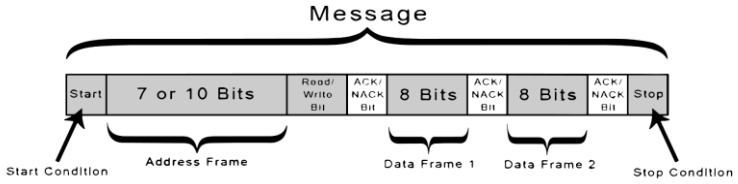


Figure 13 - i2c frame

3.4.10.2 Contribution

The I2C module contributes to our embedded project in the following ways:

- I2C is essential for our project as it enables communication with the oximeter sensor, allowing us to retrieve important medical values from the user.
- I2C plays a crucial role in communicating with the IMU sensor to obtain accurate readings for data analysis and enhance the functionality of our application during the ROS sub-system.

3.4.10.3 Configuration

To configure I2C for the Stm microcontroller, the following settings need to be considered:

1. Enable the I2C peripheral in the RCC driver.
2. Configure the GPIO pins as Alternate output open-drain mode.
3. Initialize the I2C configuration structure before calling the I2C_init function.

Our specific I2C configuration structure for the project:

- The I2C clock speed is set to standard mode.
- I2C mode is set to master mode.
- I2C address mode is set to 7-bit addressing.
- I2C general call is disabled.
- An I2C software reset is enabled.
- I2C wakeup from STOP mode is disabled.
- Clock stretching is enabled.
- I2C DMA is disabled.
- I2C SMBus is disabled.
- I2C peripheral clock: speed is set to 100 kHz.
- I2C's own address is set to 0x30.

3.4.10.4 API Functions

I2C Module API Functions:

- **void I2C_init(uint8 I2CNum, I2C_config_t * I2C_config)**
  Description: The function is responsible for initializing the I2C module in our project. It takes in the I2C number and a pointer to the I2C configuration structure as parameters. By calling this function, we can easily set up and configure the I2C module with the specified parameters, ensuring proper functionality and communication within the project..

- **void I2C_sendData(uint8 I2CNum, uint16 address, uint8 * dataBuffer, uint16 dataLength, uint8 startState, uint8 stopState)**
  Description: The function plays a crucial role in our project's I2C communication. It allows us to send data over the I2C bus by specifying the I2C number, target address, data buffer, data length, start state, and stop state. By utilizing this function, we can reliably transmit data from the microcontroller to the intended recipient, ensuring efficient and accurate communication in our application.

- **void I2C_receiveData(uint8 I2CNum, uint16 address, uint8 * dataBuffer, uint16 dataLength, uint8 startState, uint8 stopState)**
  Description: The function is essential for our I2C communication in the project. It enables us to receive data over the I2C bus by providing the I2C number, target address, data buffer, data length, start state, and stop state as inputs. With this function, we can reliably retrieve data from the desired source, ensuring smooth and efficient data reception in our application.

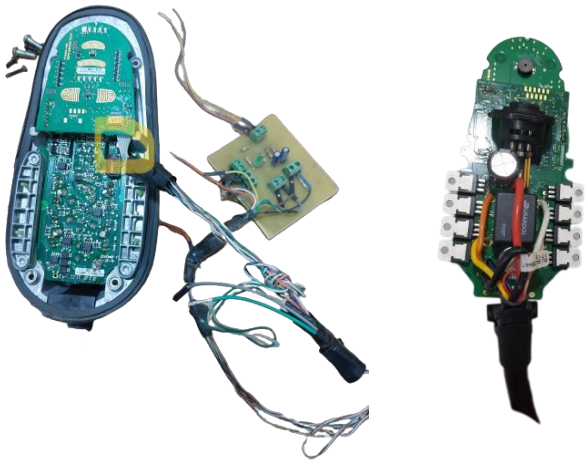
3.5 HAL Layer

3.5.1 Motor Controller Module

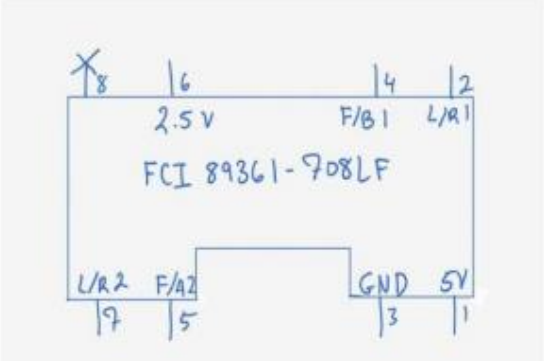
3.5.2 Joystick Module

3.5.1.1 Overview

Introduction:
The VR2 Motor Controller is a specific model of motor controller commonly used in power wheelchairs. It is designed to control the movement and operation of the wheelchair's motors based on user input. The VR2 Motor Controller offers a range of features and functionalities tailored for wheelchair applications.
Working Principle:
Before designing the control unit for operating the wheelchair, the joystick on the wheelchair needs to be removed from the wheelchair to characterize the signal voltages, from the joystick unit to the VR2 controller. A digital multi-meter and an oscilloscope are used to measure various voltage outputs of the joystick while the joystick is tested in every direction. Obtaining the joystick signals sent to the wheelchair allows us to replicate the signals to create our own controller that can control the wheelchair in the same manner.
The wheelchair uses a VR2 controller by PGDT with a D51161.01 JoyStick MAID by PG Drives Technology. The D51161.01 Joystick Maid and VR2 controller are connected via an 8 pin connector.



Characterizing the control signals from the Joystick is comprised of four signal pins, a Forward/Back pin and a Left/Right pin. The other two pins are a second Forward/Back pin and Left/Right pin for redundancy and fault tolerance. The pin out of the connector.



The signals from the eight pin connector control the movement of the left and right motors as well as the brakes. Characterizing the control signals involves powering the joystick with a 5V supply and using a four channel oscilloscope to simultaneously monitor the four signal wires as the joystick is moved in specific directions. The results of reverse engineering the joystick output signals are characterized in Table x. Once we have properly characterized the signals from the joystick, we replicated the control signals and replace the joystick with an Microcontroller.

	Netural(V)	F(V)	B(V)	R(V)	L(V)	F/R(V)	F/L(V)	B/R(V)	B/L(V)
F/B1	2.5	1.1	3.9	2.5	2.5	1.1	1.1	3.9	3.9
F/B2	2.5	1.1	3.9	2.5	2.5	1.1	1.1	3.9	3.9
R/L1	2.5	2.5	2.5	1.1	3.9	1.1	3.9	1.1	3.9
R/L2	2.5	2.5	2.5	1.1	3.9	1.1	3.9	1.1	3.9

Table x: Voltage Characterization for all 8 directions after Testing

The wheelchair control unit involves several steps that include characterizing the control signals sent between the wheelchair joystick and the wheelchair control board, setting up a microcontroller to receive and implement commands from the controller.
After the microcontroller receives the specific command from the controller the specific output voltage can be sent out using the microcontroller and Digital to Analog Converters (DAC). We didn't have a DAC so we used capacitors and resistors with PWM signals to simulate the work of the DAC.

Motor Controller Module Specs:

- Operating Temperature: -25℃ to +60℃
- Applicable Drive Motor Form: 0-50A output current,24v DC Brush Motor

Motor Controller Module Features:

- The VR2 Motor Controller is compatible with power wheelchair joystick input devices, receiving signals from the joystick and translating them into motor commands.
- The VR2 Motor Controller includes safety features such as overcurrent protection, fault detection, and diagnostics for safe operation and motor protection.
- Power Supply: The VR2 Motor Controller typically operates on a DC power supply, compatible with the power requirements of the specific wheelchair system.

3.5.1.2 Contribution

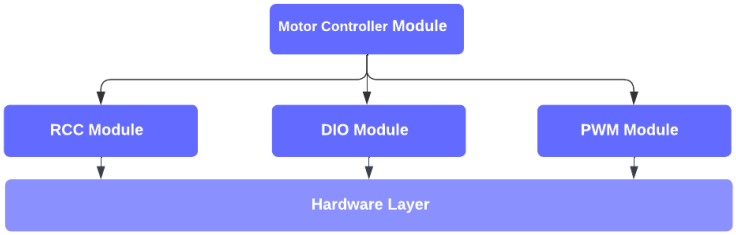
The Motor Controller module contributes to our embedded part in the following ways:

.

- controlling the motors of the wheelchair

3.5.1.3 Dependencies

The Motor Controller Module requires the DIO module, PWM module, and RCC module as essential dependencies for its operation.



3.5.1.4 API Functions

Motor Controller Module API Functions:

- `void MotorDriver_init();`
  Description: This Function is responsible for Initializing RCC and PWM Mode to controlling the motor driver.

- `void MotorDriver_setMotion(int vertValue, int horzValue);`
  Description: The function sets the motion of the wheelchair by sending the F/B AND R/L values..

- `void MotorDriver_stop();`
  Description: The function stops the motors by sending the natural values to the motor driver.

3.5.2.1 Overview

Figure 14

Introduction:

The joystick is a device that translates your hand movement into an electrical signal, and the movements are converted by the computational unit into entirely mathematical, in other words, the joystick translates entirely physical movement. It will be useful for controlling the wheelchair easily. The system has many modes of control. one of them, for ordinary electric wheelchair users, use the joystick that is shown in Figure .

Working Principle:

The joystick gives values along the X and Y axes. It contains two potentiometers, one for each axis. The two potentiometers allow us to measure the movement of the stick in 2-D. The potentiometer is a variable resistor. As we move the joystick, the value of resistance of both the potentiometer changes. This change, in turn, gives us values along the X and Y axes.

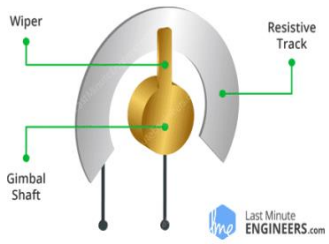


Figure 15

Joystick Module Specs:

- Operating Temperature: -25℃ to +60℃
- Output Analog Range: 0 to 1024
- D51422 JoyStick by PG Drives Technology.

Joystick Module Features:

- Easy wheelchair control: The joystick allows easy and intuitive control of the wheelchair by translating hand movements into electrical signals.
- Multiple control modes: The system offers various control modes, including the joystick control mode, catering to different wheelchair users' needs and preferences.
- 2-Dimensional movement: The joystick provides movement detection along both the X and Y axes, allowing for precise control in two dimensions.
- Potentiometer-based measurement: The joystick utilizes two potentiometers, one for each axis, to measure the movement of the stick.

3.5.2.2 Contribution

The Joystick module contributes to our embedded part in the following ways:

- Input Sensing: The module detects movements along the X and Y axes, providing precise control and feedback.
- Versatility: The Joystick module enables control in multiple directions, allowing for versatile functionality.
- Integration Flexibility: The module easily integrates into the embedded system, facilitating seamless interaction.
- Customization: The module's output analog range and potentiometer-based measurement enable customization and mapping of the joystick's movements to specific functionalities within the embedded system. This flexibility allows tailored control based on the project's requirements.

3.5.2.3 Dependencies

The Joystick Module requires the DIO module, ADC module, and RCC module as essential dependencies for its operation.

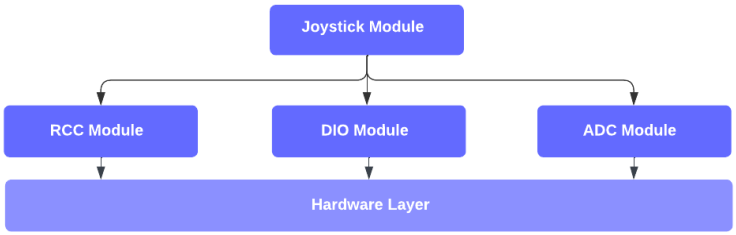


Figure 16

3.5.2.4 API Functions

Joystick Module API Functions:

- **void JoyStick_init();**
  Description: This Function is responsible for Initializing ADC Mode and Channels to determine the analog pins that interface with Joystick pins

- **int JoyStick_isJoystickReady()**
  Description: The function reads the 3 analog values from the joystick and ensure they match the initial values (512) and return 1 if correct values and 0 if not.

- **void JoyStick_getJoystickValues(int* horzValue, int* vertValue)**
  Description: The function reads the values along the X and Y axes from the joystick to send them to the controller.

3.5.3 Encoder Module

3.5.3.1 Overview

Introduction:

Encoders are devices commonly used in various applications, including robotics, industrial automation, and motion control systems. They provide feedback on the position, speed, and direction of rotation of a shaft or object. In our project, encoders play a crucial role in accurately measuring the wheelchair's movement and rotation.

Working Principle:

The working principle of an encoder is based on the detection of position changes or rotational motion. Encoders consist of a rotating disk or wheel with evenly spaced slots or markings, along with a sensor that detects these markings as they pass by. As the object or shaft rotates, the sensor generates electrical signals corresponding to the position changes. These signals can be used to determine the precise position, speed, and direction of the rotation.

Encoder Specs:

- Resolution.
- Operating Voltage.
- Output Type.
- Speed and Frequency Response.
- Environmental Considerations.

Encoder Features:

- Position Feedback: Encoders provide precise position feedback, enabling accurate tracking and control of motion.
- Speed and Direction Sensing: By monitoring the changes in position over time, encoders allow for speed and direction sensing of rotating objects.
- Incremental or Absolute Positioning: Encoders can offer either incremental or absolute positioning.

3.5.3.2 Contribution

The Encoder module contributes to our embedded part in the following ways:

- Motion Tracking: The Encoder module provides precise tracking of the wheelchair's movement, allowing you to accurately measure the distance traveled and the speed of rotation.
- Position Feedback: By utilizing the Encoder module, you can obtain accurate position feedback of the wheelchair.
- Direction Sensing: With the Encoder module, you can determine the direction of rotation of the wheelchair.

3.5.3.3 Dependencies

The Encoder requires the RCC module, DIO module, AFIO module, EXTI module, NVIC module, and STK module as essential dependencies for its operation.

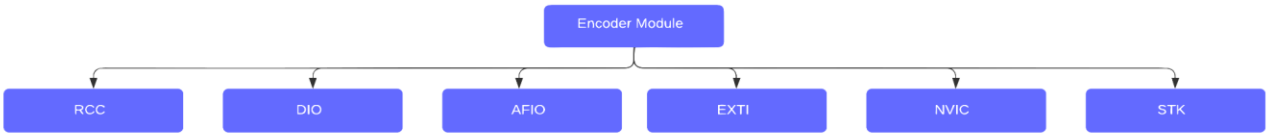


Figure 17

3.5.3.4 Configuration

To configure the encoder for the Stm microcontroller, the following settings need to be considered:

1. Configure the microcontroller's input pins to receive the encoder signals.
2. Conduct testing and calibration procedures to verify the accuracy and performance of the encoder.

3.5.3.5 API Functions

LM35 Module API Functions:

- **General APIs:**

- **void Encoder_init (Encoders_ID_t Encoder_ID)**
  Description: The function initializes the specified encoder, configuring the necessary pins and interfaces for communication with the STM microcontroller.

- **void Encoder_reset(Encoders_ID_t Encoder_ID)**
  Description: The function resets the specified encoder, clearing any accumulated counts or position values and restoring it to its initial state.

- **int32 Encoder_getCounts(Encoders_ID_t Encoder_ID)**
  Description: The function retrieves the current count value from the specified encoder, allowing you to access and utilize the position information.

- **float32 Encoder_getConvertCounts(Encoders_ID_t Encoder_ID, Encoders_Retuen_Format_t Encoders_Retuen_Format)**
  Description: The function retrieves the current count value from the specified encoder and converts it to a floating-point value according to the desired format.

- **float32 Encoder_getSpeed(Encoders_ID_t Encoder_ID, Encoders_Retuen_Format_t Encoders_Retuen_Format)**
  Description: The function calculates and returns the speed of the specified encoder in the desired format.

- **Encoders_Motors_Direction_t Encoder_getDirection(Encoders_ID_t Encoder_ID)**
  Description: The function retrieves the direction of rotation for the specified encoder.

3.5.4 Ultrasonic Module

3.5.4.1 Overview

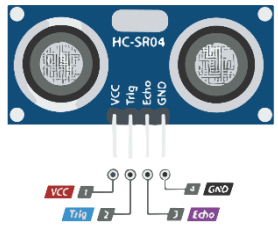


Figure 18

Introduction:
Ultrasonic sensors are widely used in various applications, including robotics, automation, and distance measurement. They utilize sound waves at ultrasonic frequencies to detect the presence or measure the distance of objects in their vicinity. These sensors offer several advantages, such as non-contact operation, reliability in different environments, and versatility in applications.

Working Principle:
The working principle of ultrasonic sensors is based on the propagation of sound waves. The sensor emits high-frequency sound waves and measures the time it takes for the waves to bounce back after hitting an object. By analyzing the time of flight, which is the round-trip travel time, the distance to the object can be calculated based on the speed of sound.
The sensor's transmitter generates sound waves, which propagate through the medium, and when they encounter an object, they reflect back towards the sensor. The sensor's receiver detects these echoes and measures the time it takes for them to return. This time measurement is used to determine the distance to the object.
Ultrasonic HC-SR04 Sensor Specs:

- Working Voltage: DC 5V.
- Working Current: 15mA.
- Trigger Input Signal: 10µS TTL pulse
- Measuring Angle: 15 degrees.
- Operating frequency of 40 kHz
- Echo Output Signal Input TTL level signal and the range in proportion.
- Accurate distance measurements with +/- 0.5 cm precision
- Wide measuring range from 2 cm to 400 cm.
- High resolution of 0.3 cm

Ultrasonic HC-SR04 Sensor Features:

- Non-contact distance measurement
- Easy to use with trigger and echo signals.
- Compact size for easy integration into various projects
- Low power consumption
- Reliable performance in different environmental conditions

3.5.4.2 Contribution

The Ultrasonic HC-SR04 module contributes to our embedded part in the following ways:

- Distance Measurement: The Ultrasonic sensor plays a vital role in our system by checking if there any obstacles near to the wheelchair or not.

3.5.4.3 Dependencies

The Ultrasonic HC-SR04 sensor requires the RCC module, DIO module, AFIO module, EXTI module, NVIC module, and STK module as essential dependencies for its operation.

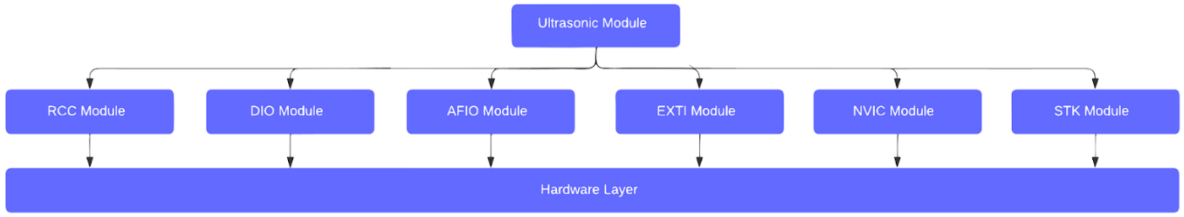


Figure 19

3.5.4.4 Configuration

To configure the Ultrasonic HC-SR04 for the Stm microcontroller, the following settings need to be considered:

1. Configure the Ultrasonic sensor by passing the distance calculations.
2. Initialize the system by calling the Ultrasonic_init() function, which sets up the necessary registers and configurations for the Ultrasonic to function properly.

Our specific Ultrasonic configuration for the project:

- We set the timer period to be 1000.

3.5.4.5 API Functions

Ultrasonic HC-SR04 Module API Functions:

- **General APIs:**

- `void Ultrasonic_init(Ultrasonic_ID_t ID, Ultrasonic_config_t * Ultrasonic_config)`
  Description: The function initializes the ultrasonic sensor module. It sets up the necessary configurations for proper sensor operation.

- `float32 Ultrasonic_readDistance(Ultrasonic_ID_t Ultrasonic_ID)`
  Description: The function reads the distance from the specified ultrasonic sensor.

- `void Ultrasonic_setRange(Ultrasonic_ID_t Ultrasonic_ID, uint16 Range_cm)`
  Description: The function sets the maximum range for the specified ultrasonic sensor.

- `void Ultrasonic_setState(Ultrasonic_ID_t Ultrasonic_ID, Ultrasonic_state_t state)`
  Description: The function sets the state of the specified ultrasonic sensor. It allows the software engineer to enable or disable the sensor based on the desired functionality.

3.5.5 Oximeter Module

3.5.5.1 Overview

Introduction:

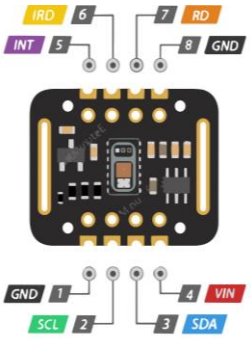


Figure 20

The Oximeter 30102 is a medical device used for measuring blood oxygen saturation levels (SpO2) and heart rate. It is commonly used in healthcare settings, as well as for personal monitoring at home. The device utilizes a non-invasive method that involves placing a sensor on a patient's fingertip or earlobe, which then measures the oxygen saturation levels in the blood. With its compact and portable design, the Oximeter 30102 provides quick and accurate readings, making it an essential tool for assessing respiratory and cardiovascular health.

Working Principle:

The working principle of the Oximeter 30102 is based on the optical absorption characteristics of oxygenated and deoxygenated hemoglobin in the blood. It uses light-emitting diodes (LEDs) to emit red and infrared light into the fingertip or earlobe.

The photodetector in the device measures the intensity of the light that passes through the tissue, which varies based on the amount of oxygenated and deoxygenated hemoglobin present. By analyzing the ratio of the two types of hemoglobin, the device calculates the blood oxygen saturation (SpO2) level and provides a numerical reading. This non-invasive and painless method allows for quick and reliable monitoring of oxygen levels in the blood.

Oximeter30102 Sensor Specs:

- Non-invasive blood oxygen measurement (SpO2) range: 0% to 100%
- Quick and convenient monitoring
- LED light emission for measurement
- Photodetector technology for accurate results
- Compact and portable design for easy use

Oximeter30102 Sensor Features:

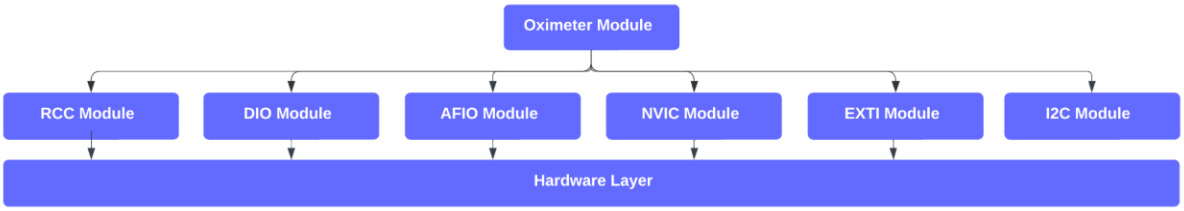
- Accurate SpO2 measurement: Provides precise non-invasive measurement of blood oxygen saturation levels.
- Fast and convenient readings: Enables quick and easy monitoring of SpO2 levels without the need for invasive procedures.
- LED light technology: Utilizes LED light emission to measure SpO2 levels accurately.
- Compact and portable design: Designed to be lightweight and portable, allowing for convenient use in various settings.

3.5.5.2 Contribution

The Oximeter30102 module contributes to our embedded part in the following ways:

- SpO2 Measurement: The Oximeter sensor plays a vital role in our system by accurately measuring blood oxygen saturation (SpO2) levels.
- Heart Rate Monitoring: It also provides the capability to monitor the user's heart rate, allowing us to gather important cardiovascular data.

- Temperature Sensing: In addition to SpO2 and heart rate measurement, we utilized the Oximeter sensor to read temperature values, making it an integral part of our temperature measurement system.



3.5.5.3 Dependencies

The Oximeter30102 sensor requires the RCC module, DIO module, AFIO module, EXTI module, NVIC module, and I2C module as essential dependencies for its operation.

3.5.5.4 Configuration

To configure the Oximeter30102 sensor for the Stm microcontroller, the following settings need to be considered:

1. Configure the Oximeter sensor by passing the required values instead of the default ones specified in the config file.
2. Initialize the system by calling the Oximeter_init() function, which sets up the necessary registers and configurations for the Oximeter to function properly.

Our specific Oximeter configuration for the project:

- Sample Averaging: 32 samples
- FIFO Rollover State: Enabled
- FIFO Almost Full Value Flag: 32 (Number of unread data samples)
- Mode: Multi-mode
- SPO2 ADC Range: 16384
- SPO2 Sample Rate: 3200 Hz
- ADC Resolution: 18-bit

3.5.5.5 API Functions

Oximeter Module API Functions:

- **General APIs:**
- **void Oximeter_init()**
  Description: The function sets up the required registers and configurations for the proper functioning of the Oximeter. It initializes various modules such as RCC, GPIO, EXTI, NVIC, I2C, and AFIO, along with the necessary pins and interrupts, and initializes the Oximeter with the chosen configuration values from the config file.
- **void Oximeter_getSPO2AndHeartRateSenq(int32 * SPO2_value, int32 * HeartRate_value, uint8 Num_samples)**

Description: The function calculates the SPO2 (blood oxygen saturation) and heart rate values using the Maxim algorithm. It returns these values through reference parameters in a synchronous manner, allowing for the retrieval of accurate SPO2 and heart rate measurements.

- **void Oximeter_getSPO2AndHeartRateASenq(uint8 * SPO2_value, uint8 * HeartRate_value, uint8 Num_samples, Measure_Mode mode)**
  Description: This function asynchronously calculates the SPO2, and heart rate values using the Maxim algorithm and updates the SPO2 value and heart rate value by storing them in the provided uint8 reference parameters. The Num_samples parameter determines the number of samples used for the calculation, and the measure mode parameter allows for selecting the specific measurement mode.

- **float32 Oximeter_readTemperatureSynq()**
  Description: The function reads the temperature value from the MAX30102 sensor in synchronous mode and returns it as a floating-point value. It provides an easy and convenient way to obtain accurate temperature measurements from the sensor.

- **void Oximeter_readTemperatureAsenq(float32 * temperature, Measure_Mode mode)**
  Description: This function reads the temperature value from the MAX30102 sensor in asynchronous mode and updates the temperature value by storing it in the provided float reference variable. The measurement mode parameter allows for selecting the specific mode of temperature measurement.

- **void Oximeter_setMode(Mode_t Mode)**
  Description: The function is used to set the operating mode of the oximeter. It allows you to select from available modes such as heart rate, spo2, and multi-led mode, providing flexibility in capturing different types of physiological data.

- **void Oximeter_shutdowm(State_t state)**
  Description: The function is used to control the power state of the oximeter. Bypassing the desired state argument, you can either shut down or enable the oximeter module, providing control over its operation and power consumption.

- **void Oximeter_reset()**
  Description: The function is used to reset the oximeter module. It restores the module to its default settings and clears any previous configurations or data, allowing for a fresh start in the operation of the oximeter.

- **Setting APIs**

These functions are responsible for changing the settings of the oximeter sensor:

```
void Oximeter_setSampleAverging(Sample_Averaging_t sampleAveraging)
void Oximeter_setSPO2ADCRange(SPO2_ADC_Range_t SPO2_ADC_Range)
void Oximeter_setSPO2SampleRate(SPO2_Sample_Rate_t SPO2_Sample_Rate)
void Oximeter_setADCResolution(ADC_Resolution_t ADC_Resolution)
void Oximeter_setRedPulseAmplitude(uint8 value)
void Oximeter_setRollOnFullState(State_t state)
void Oximeter_setSlot(Max30102_slot_t slotNumber, led_State_t ledNumber)
```

3.5.6 LM35 Temperature Sensor Module

3.5.6.1 Overview

Introduction:

The LM35 temperature sensor is a popular and widely used integrated circuit (IC) sensor designed to accurately measure temperature. It offers a simple and straightforward method for obtaining temperature readings in various applications. The sensor provides a linear analog output voltage that is directly proportional to the temperature being measured, allowing for easy conversion and interpretation of temperature data. With its wide temperature range and low power consumption, the LM35 is suitable for both industrial and consumer electronics projects, making it a reliable choice for temperature sensing needs.

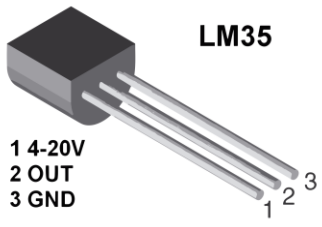


Figure 21

Working Principle:

The LM35 temperature sensor is based on the linear relationship between temperature and output voltage. It generates an output voltage that increases by 10mV for every one-degree Celsius rise in temperature. This voltage can be measured and used to determine the temperature value.

The sensor does not require any external calibration or signal conditioning circuitry, making it a convenient and straightforward solution for temperature sensing.

LM35 Sensor Specs:

- Operating Temperature Range: -55°C to +150°C
- Output Voltage Range: 0V to +1.5V (proportional to temperature)
- Accuracy: ±0.5°C (typical)
- Low Self-Heating: 0.08°C in still air
- Supply Voltage: 4V to 30V DC

LM35 Sensor Features:

- Temperature Measurement: The LM35 temperature sensor is capable of measuring temperature accurately.
- Linear Output: It provides a linear analog output voltage that is directly proportional to the temperature being measured.
- Wide Temperature Range: The LM35 operates over a wide temperature range, typically from -55°C to +150°C, making it suitable for various applications.
- Low Power Consumption: It consumes low power, making it energy-efficient and suitable for battery-powered devices.
- Easy to Use: The LM35 is easy to integrate into electronic circuits due to its straightforward interface and compatibility with microcontrollers and other components.

3.5.6.2 Contribution

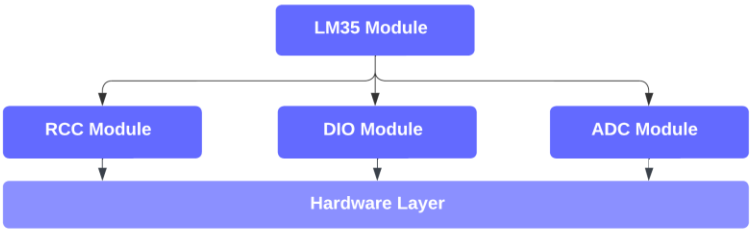
The LM35 module contributes to our embedded part in the following ways:

- Temperature Sensing Solution: The LM35 sensor plays a crucial role in our temperature measurement system, contributing to obtaining accurate temperature values from the user.

- Reliable Temperature Detection: By integrating the LM35 sensor, we ensure reliable and precise temperature sensing as part of our overall approach to capturing the user's temperature.

3.5.6.3 Dependencies

The LM35 temperature sensor requires the DIO module, ADC module, and RCC module as essential dependencies for its operation.



3.5.6.4 Configuration

To configure the LM35 sensor for the Stm microcontroller, the following settings need to be considered:

1. Define the MEASURE_SETUP macro to choose the desired temperature range (LM35_BASIC or LM35_FULL_RANGE).
2. Set the LM35_channel_ID and LM35_convertion_NUM variables to the desired values.

3.5.6.5 API Functions

LM35 Module API Functions:

- **void LM35_init()**
  Description: The function is responsible for enabling the necessary RCC buses and peripherals, initializing the GPIO module, and configuring the ADC pin to enable the LM35 temperature sensor for accurate temperature measurement. It should be called prior to using the LM35_getTemperature function.

- **uint8 LM35_getTemperature()**
  Description: The function reads the temperature from an LM35 temperature sensor connected to an ADC pin and returns the temperature value as a uint8 in degrees Celsius(°C).

3.5.7 Battery Indicator Module

3.5.7.1 Overview

Introduction:

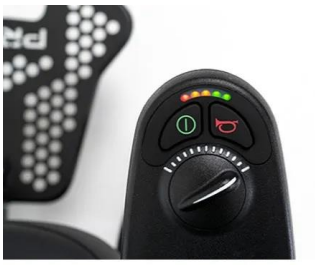


Figure 22

The Battery Indicator Module in the wheelchair is a feature that provides visual indications of the remaining battery level. It allows users to monitor the battery status of the wheelchair, ensuring they have a clear understanding of the available power and can plan accordingly. The module typically includes LED indicators to convey the battery level information in a user-friendly manner. This helps wheelchair users to avoid unexpected power depletion and ensures they can manage their mobility safely and effectively.

Working Principle:

The Battery Indicator is based on the voltage division principle. The battery voltage is sampled using a voltage divider so that it can be applied to the microcontroller pin without damaging it. The sampled voltage is then measured using an ADC to determine the current battery level. The ADC reading is rounded to one of 5 levels available, where the green LED (LED#5) indicates (100%) battery level, and the Red LED (LED#1) indicates (20%) battery level.

The measured voltage is assumed to be directly proportional to the charge level. Figure (x) shows the (Charge-Voltage) curve of a 24V lead-acid battery, just like the one that is used in the electric wheelchair.

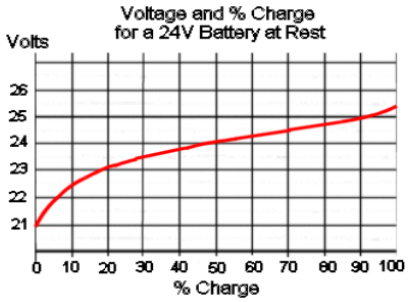


Figure 23

Battery Indicator Specs:

- LED indicators.
- 5 LEDS providing 5 battery levels.
- Every LED express 20% of the charge.

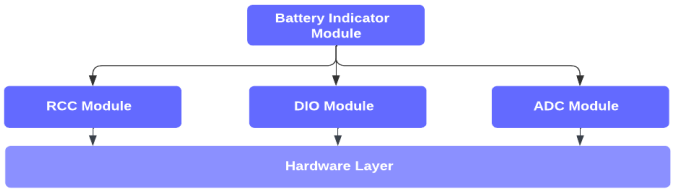
3.5.7.2 Contribution

The Battery Indicator module contributes to our embedded part in the following ways:

- Battery Monitoring: Users can easily monitor the remaining battery level of their wheelchair.
- Prevents Power Depletion: Timely indications of low battery levels prevent unexpected shutdowns.
- Safety and Reliability: Users can plan activities and avoid mobility limitations due to drained batteries.
- Optimal Battery Management: Efficiently recharge batteries before reaching critical levels, extending their lifespan.

3.5.7.3 Dependencies

The Battery Indicator Module requires the DIO module, ADC module, and RCC module as essential dependencies for its operation.



3.5.7.4 Configuration

To configure the Battery Indicator for the Stm microcontroller, the following settings need to be considered:

1. Initialize the driver and the necessary peripheral RCC and DIO.
2. Set the battery discharge curve option using the BATTERY_DISCHARGE_CURVE macro. choose between LINEAR and NON_LINEAR.
3. If using the NON_LINEAR option, this allows defining a lookup table for the battery discharge curve by setting the lookup_Table array.
4. If using the LINEAR option, this allows setting the voltage values for a linear battery discharge curve by defining the VOLTAGE_FULLY_CHARGED and VOLTAGE_FULLY_DISCHARGED macros.

3.5.7.5 API Functions

Battery Indicator Module API Functions:

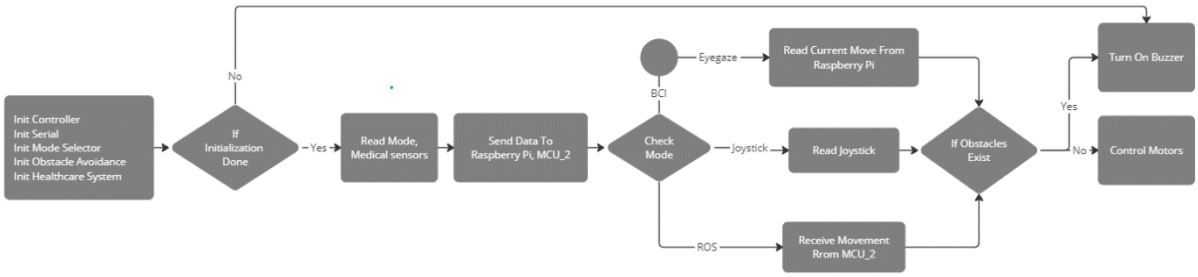
- **void BattInd_init()**
  Description: The function initializes the battery indicator module. It sets up the ADC for measuring the battery voltage, configures GPIO pins and initializes any other necessary variables or settings.
- **#if BATTERY_DISCHARGE_CURVE == NON_LINEAR**
  uint8 BattInd_getBatteryCapacityLevel()

  Description: The function calculates the capacity level of the battery based on the lookup table of voltage values for different capacity levels, obtained from the battery discharge curve. It reads the ADC value of the battery voltage, converts it to voltage value, and compares it to the voltage values in the lookup table to determine the battery capacity level.
- **#elif BATTERY_DISCHARGE_CURVE == LINEAR**
  uint8 BattInd_getBatteryCapacityPercentage()

Description: The function calculates the percentage of remaining capacity of the battery based on the linear discharge curve of the battery, assuming that the battery voltage decreases linearly with capacity level. It reads the ADC value of the battery voltage, converts it to voltage value, and calculates the percentage of remaining capacity based on the difference between the battery voltage and the minimum and maximum voltages of the discharge curve.

3.6 APP Layer

3.6.1 Flowchart



1. Initialization:

- The system starts by initializing the controller, which sets up the necessary configurations and interfaces for the overall system operation.
- Serial communication is initialized to establish communication channels between different modules or devices within the system.
- The mode selector is initialized to enable the selection of different operating modes or control schemes based on user preferences or system requirements.
- Obstacle avoidance is initialized, which includes setting up sensors or algorithms to detect and avoid obstacles in the system's environment.
- The healthcare system is initialized, which includes the setup of medical sensors or devices used for monitoring or measuring vital signs or health-related parameters.

2. Check Initialization:

- The system checks if the initialization process is completed successfully.
- If the initialization is not done, indicating a failure or error, the system activates a buzzer or alarm to indicate the issue.

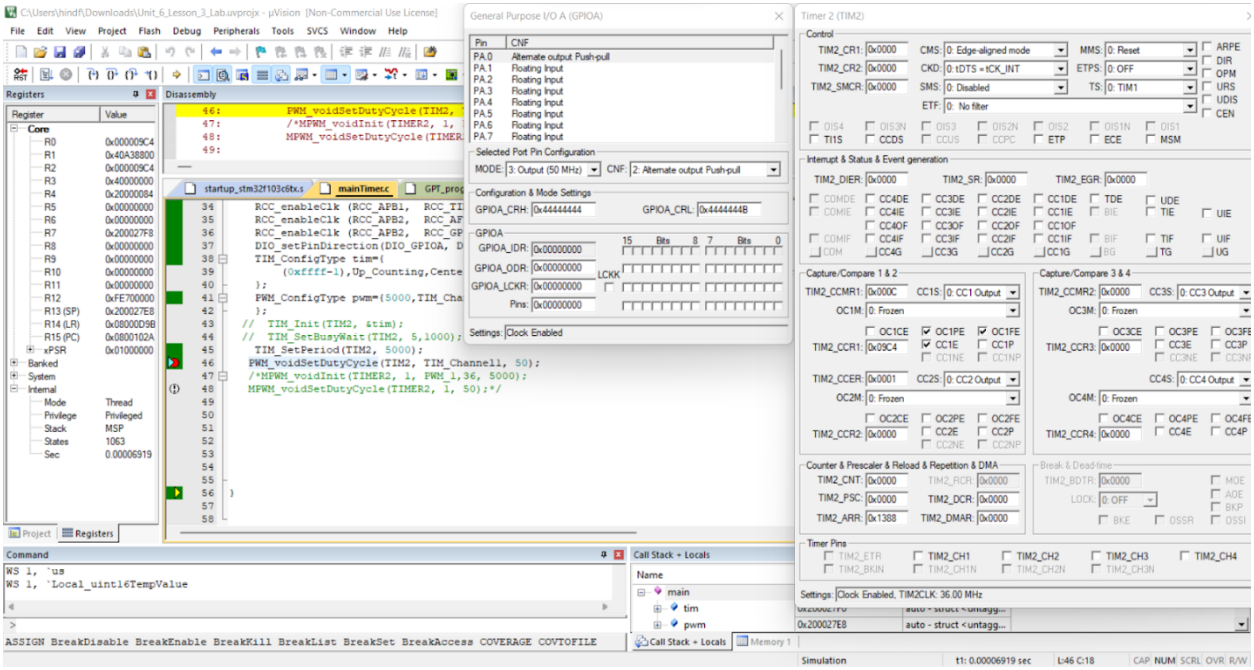
3. If Initialization is Done:

- The system proceeds with the operational phase.
- It reads the current mode selected and collects data from medical sensors to monitor and track the user's health status.

- The collected data is sent to a Raspberry Pi and an MCU_2 (another microcontroller) for processing or further actions.
- Depending on the mode selected (such as Brain-Computer Interface or Eye Gaze), the system reads the user's intended movement or action from the Raspberry Pi or MCU_2.

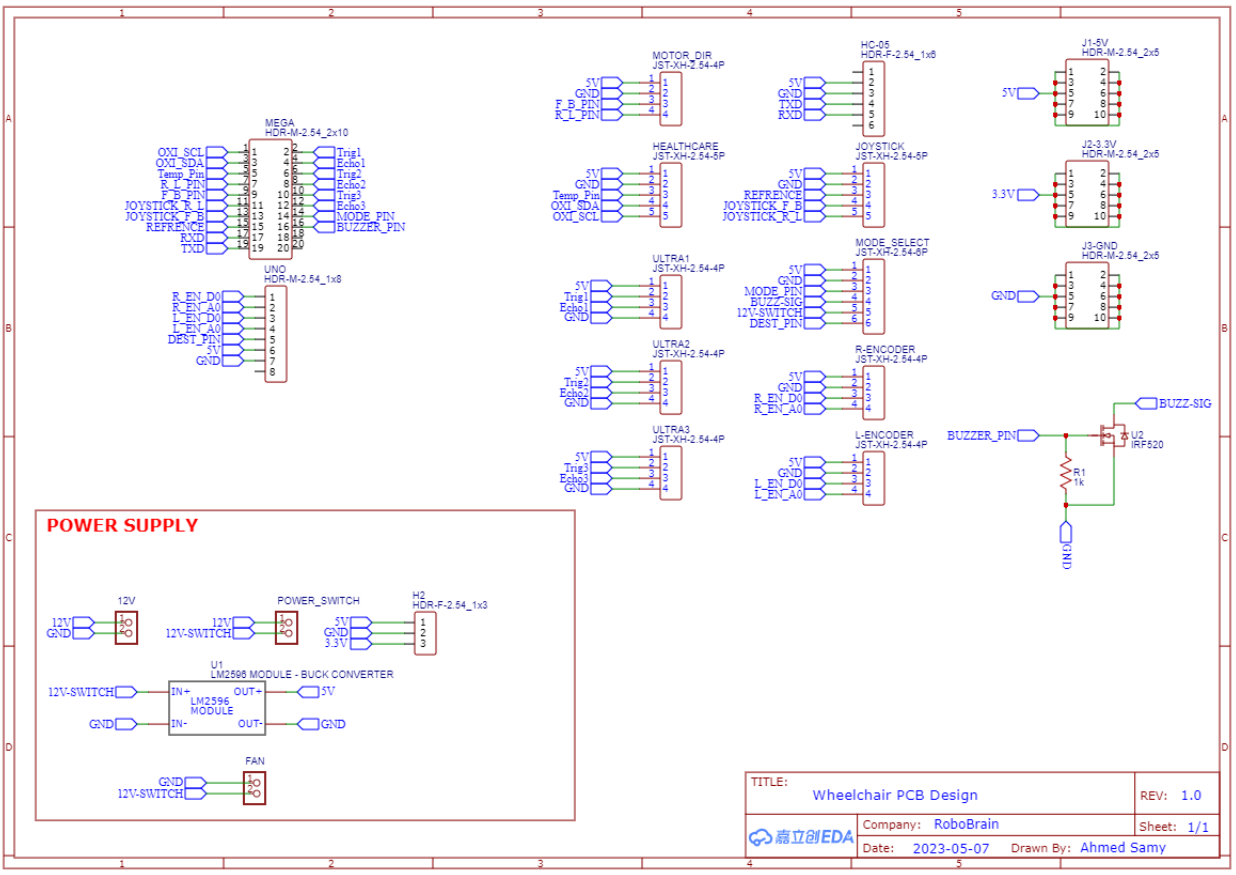
- If the control input is from a joystick, the system reads the input from the joystick to determine the desired movement.
- If the system operates with the Robot Operating System (ROS), it receives movement instructions from MCU_2.
- The system checks if there is an obstacle present in the environment.
- If an obstacle is detected, the system activates the buzzer or alarm to alert the user.
- If no obstacle is detected, the system proceeds to control the motors, enabling the wheelchair or device to move according to the user's intended movement or instructions.

3.6.2 Modules Troubleshooting

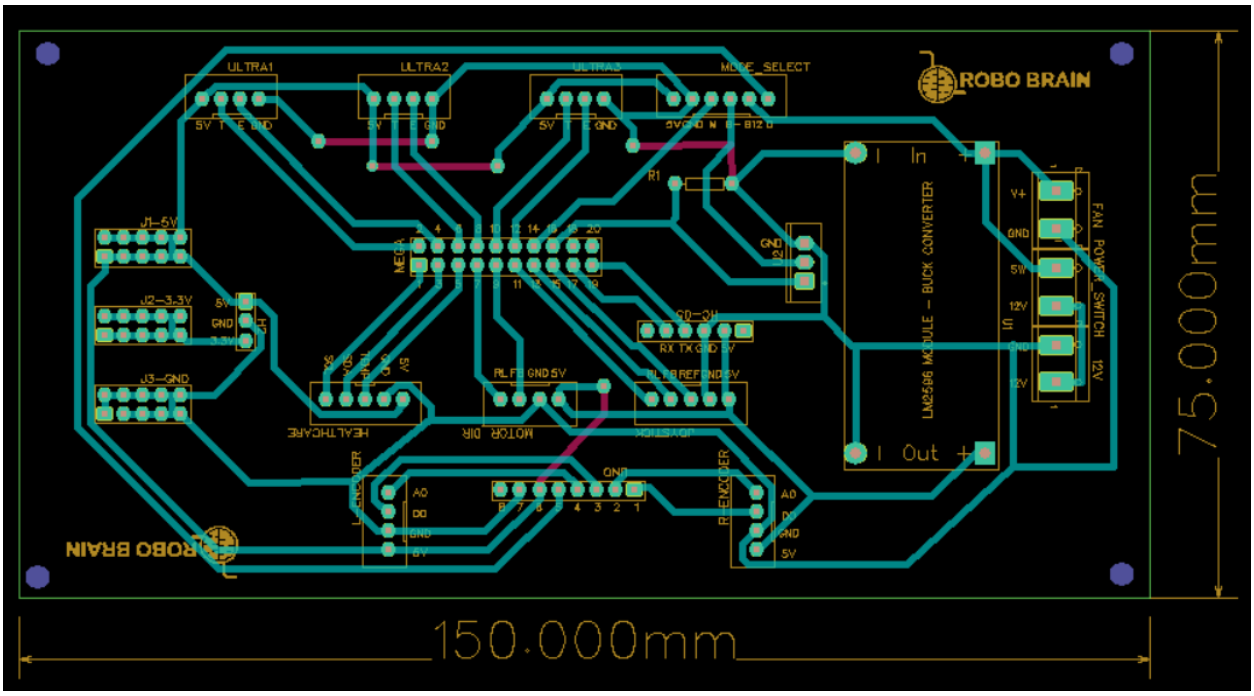


3.7 PCB Design

3.7.1 whole system integration PCB Schematic Board



3.7.2 whole system integration PCB Board



3.8 Digital Fabrication Parts For Wheelchair

3.8.1 Introduction

The digital fabrication part of this phase focuses on manufacturing two crucial components: a Component Housing box and a Healthcare box. The Component Housing box(triangle box) ensures stability by securely housing components such as the Kinect camera and MCUs.
Meanwhile, the Healthcare box(Control box) conveniently integrates the mode selector, destination selector, on/off switch, and healthcare sensors.

3.8.2 Design Process

- Define dimensions and specifications for the triangle box and control box based on component requirements.
- Create 3D models of the triangle box and control box using Fusion 360, ensuring proper placement and stability.
- Design cutouts, joints, and connections in Fusion 360 to accommodate the Kinect camera, MCUs, mode selector, destination selector, on/off switch, and healthcare sensors.
- Export the finalized designs as compatible files as DXF for laser cutting.
- Use laser cutting software (LaserCAD) to configure parameters and generate cutting paths.
- Laser cut the wood parts for the triangle box and control box based on the design files.
- Assemble the wood parts, aligning joints and using adhesives or fasteners for stability.
- Mount components within the boxes, ensuring proper connections, cable management, and secure fitting.

3.8.3 Tools, Software, and Materials

In the development of the parts, various tools, software, and materials were employed to facilitate the design and fabrication process:

1. **Laser Cutter Machine:**

- A laser cutter machine was utilized to precisely cut the wooden parts for the prototype design.
- This machine enabled clean and accurate cuts, ensuring the components fit together seamlessly.

2. Plywood (5mm):

- 5mm thick plywood was chosen as the material for the design.
- Plywood offers a balance of strength and flexibility, making it suitable for constructing the wooden parts of the prototype.

3. Fusion 360:

- Fusion 360, a comprehensive computer-aided design (CAD) software, was employed for designing the prototype.
- This powerful software facilitated the creation of detailed 3D models, ensuring accurate dimensions and structural integrity.

4. LaserCAD Software:

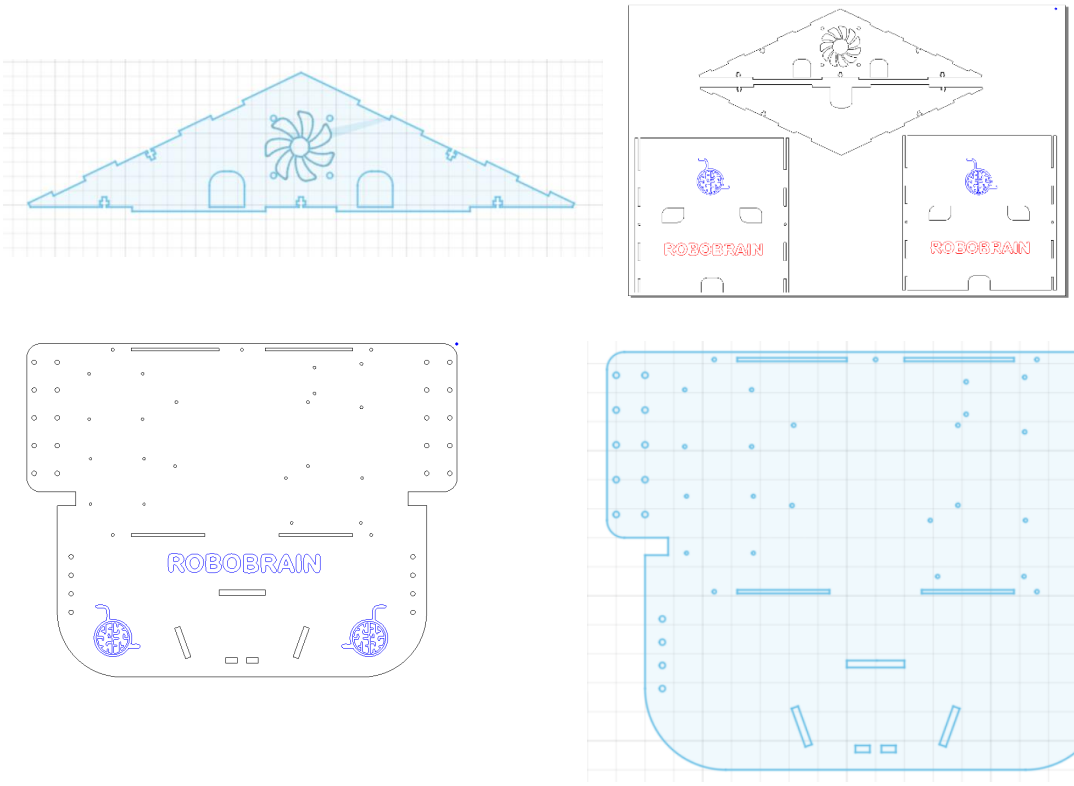
- LaserCAD software played a crucial role in preparing the design files for the laser cutter machine.
- It enabled the generation of precise cutting instructions, ensuring the wooden parts were accurately cut according to the design specifications.

5. Nuts and Nails (M3):

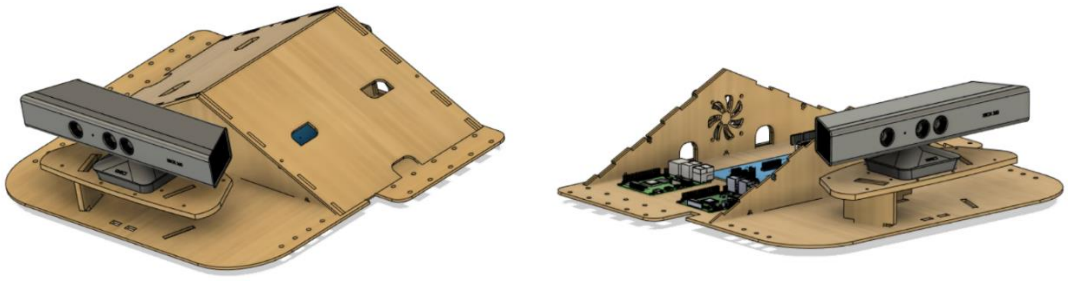
- M3 nuts and nails were utilized for assembly purposes.
- These standard-sized fasteners provided secure connections between the wooden components, ensuring structural stability.

The combination of these tools, software, and materials facilitated the efficient and precise fabrication of the prototype, ensuring accurate cuts, robust construction, and structural stability.

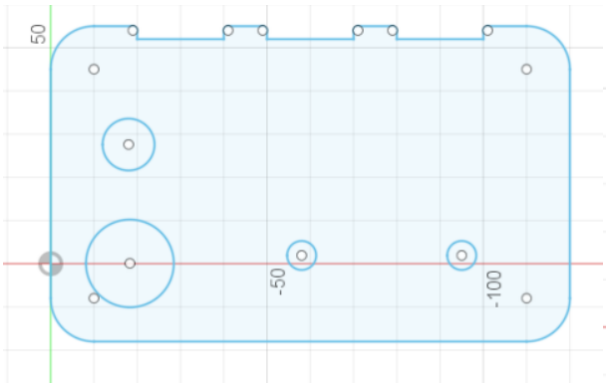
3.8.4 Component Housing Box Process Visualization

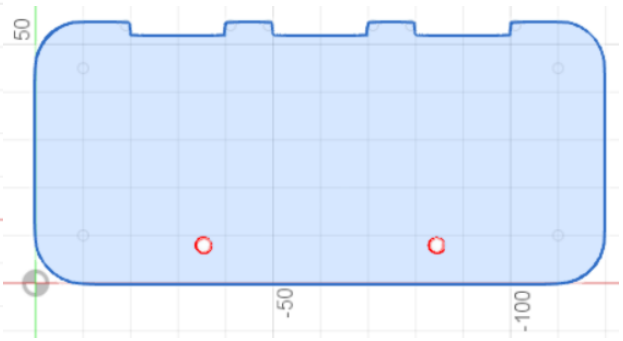


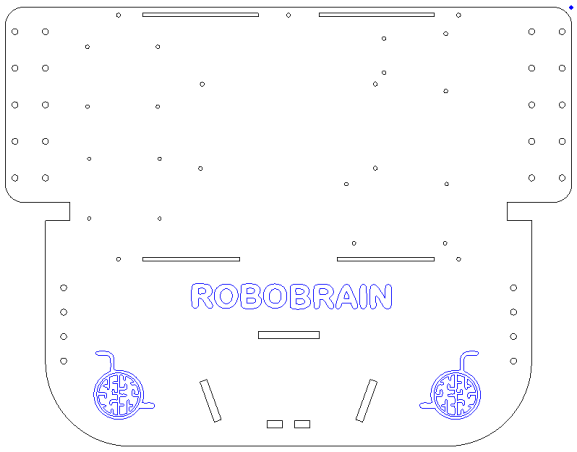
3.8.5 Component Housing Box Showcase

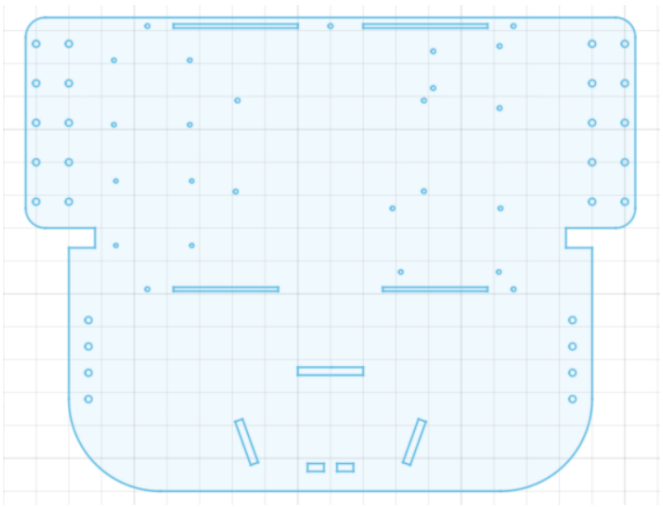


3.8.6 Healthcare Box Process Visualization

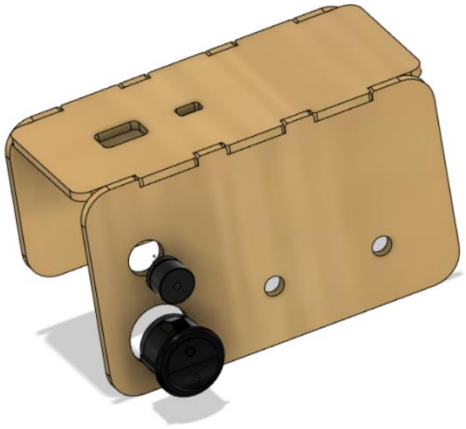


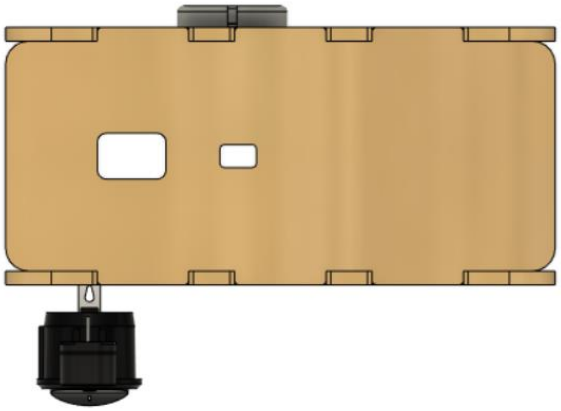






3.8.7 Healthcare Box Showcase





BCI & EYEGAZE

4.1 Introduction:

4.1.1 Eye-Gaze module

In recent years, advancements in smart mobility-aided systems have sparked interest among scientists, particularly in developing innovative solutions for individuals with quadriplegia (paralysis of all four limbs). Compared to other forms of paralysis, quadriplegic patients face greater challenges in terms of physical functionality, well-being, and independence, as revealed by a study investigating the quality of life for individuals with spinal cord injuries. While individuals with monoplegia, hemiplegia, or paraplegia can utilize joystick or gesture-controlled wheelchairs readily available in the market, quadriplegic individuals, who also experience conditions such as Parkinson's disease and Amyotrophic Lateral Sclerosis (ALS), suffer from damage to their nervous system, resulting in the inability to move their voluntary muscles.

However, despite receiving training, a clinical survey has shown that 9-10% of patients find it extremely challenging to operate power wheelchairs. Recognizing this pressing issue, 85% of clinicians have emphasized the urgent need for autonomous wheelchairs that can be controlled through eye and brain mechanisms, thus enhancing the lives of these patients. This has spurred scientists worldwide to dedicate their efforts to developing smart systems that improve the control mechanism of rehabilitative autonomous devices. One significant development in this area is the utilization of bio-signals to detect corneo-retinal potential, which forms the basis of an Electrooculography (EOG) based autonomous wheelchair. A key feature of this research is the use of the eye as a control interface, effectively replacing the traditional mouse in operating various graphical user interfaces. Other studies have also focused on developing EOG signal acquisition systems.

However, EOG has its limitations due to problematic alterations caused by eye blinks, leading to inefficiencies in the dipole potential of the cornea and retina. Thus, further research is required to address these limitations and enhance the effectiveness of EOG-based control mechanisms.

During the course of research, voice command-based wheelchair operation was considered, but the presence of surrounding voices often led to misinterpretation of commands. Systems relying on infrared reflections have been utilized to accurately track the position of the eye pupil. However, a major drawback of this technology is the potential risk of permanent eye damage due to exposure to infrared rays .

To overcome these drawbacks, we developed a real-time camera-based pupil position detection mechanism as an alternative to the infrared reflection eye-controlled system.

This chapter focuses on the development of a gaze-controlled wheelchair system, leveraging the dlib library, Python programming language, and Raspberry Pi platform. By harnessing these tools, we aim to create a robust and efficient control mechanism for wheelchair operation.

A key component of our approach is the utilization of the OpenCV library, which allows for precise tracking of the eye pupil's location in real-time. This tracking is crucial in accurately interpreting the user's gaze direction and translating it into wheelchair movements. By employing Python as the programming language, we ensure a seamless and streamlined implementation of the control mechanism, facilitating smooth and intuitive operation.

Throughout the subsequent sections of this chapter, we will delve into the methodology behind our gaze-controlled wheelchair system, providing insights into the technical aspects of its development. Additionally, we will present the experimental results, demonstrating the advantages and potential applications of this innovative assistive technology within the realm of mobility aids for individuals with disabilities.

4.1.2 System Algorithm And Model Design:

The following section provides a comprehensive description of the system algorithm and Model design, illustrating the step-by-step process of the system's initialization and shutdown.

The algorithm's design ensures a clear and concise understanding of how the system transitions from an inactive state to an active state, as well as how it can be safely shut down when required. This detailed depiction of the system's functionality highlights its usability and demonstrates the successful implementation of a robust and efficient control mechanism.

System Algorithm:

- Real Time Face Detection:
    - Continuously checks if a face is detected.
    - If a face is detected, proceed to the next step.
    - If no face is detected, the system remains in an idle state.
- Eye Detection:
    - Checks if the eyes are successfully detected.
    - If the eyes are detected, proceed to the next step.
    - If the eyes are not detected, the system remains in an idle state.
- Blink Detection:
    - Monitors the subject's blinking pattern.
    - Checks if the subject performs two blinks, that turns on the system and makes it enter the continuous loop of reading gaze direction and taking actions.
    - If two blinks are detected, proceed to the next step.
    - If the subject does not perform two blinks, the system remains in an idle state.
    - If the subject performs two blinks again, the system shuts down and it does not take any actions until the subject performs two blinks again.
- Gaze Direction Detection:
    - Detects the direction of the subject's gaze.
    - Determines the desired direction of movement based on the gaze direction.
    - Sends the corresponding code to the embedded system module.
- Wheelchair Movement:
    - The embedded system module receives the code and controls the DC motors.
    - The DC motors are driven to move the wheels accordingly in the desired direction.

- Idle State:
    - If no face is detected, eyes are not detected, or gaze direction is not detected, the wheelchair remains in an idle state.

Model Design:

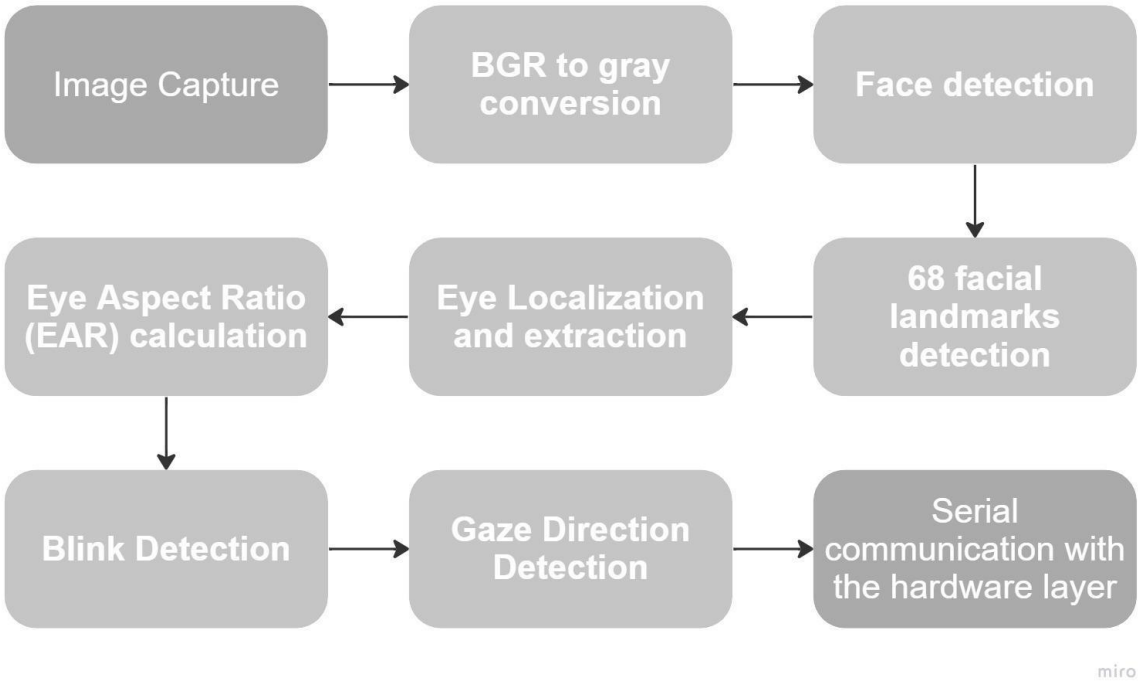


Figure 24 - model design

The model design consists of the following steps, each contributing to the functionality of the system:

- Real-time frame-by-frame pictures from a video stream: Continuous video input is obtained by the webcam and resized to facilitate subsequent image processing.
- BRG to gray conversion: The captured video frames are converted from BRG (blue-red-green) color space to gray scale for efficient analysis.
- Face detection: Face detection is a fundamental process of recognizing human faces within a digital image or video. There are various approaches available for performing face detection. In our project, we utilized the face detection capabilities provided by OpenCV and Dlib libraries to effectively identify and extract facial regions.
- Extraction of 68 facial landmarks: To determine the position of the eyes on the face, we utilized the facial landmark predictor available in the Dlib library. This tool helped us identify important facial structures, enabling us to accurately extract the eye position.
- Eye localization and extraction: Precise isolation of the eye regions is performed based on the extracted facial landmarks.
- Eye aspect ratio calculation: The openness of the eyes is evaluated using the eye aspect ratio, allowing for blink detection.
- Blink detection: Blink events are accurately detected by analyzing changes in the eye aspect ratio.

- Eye gaze direction detection: The position and movement of the eyes are calculated to determine the direction of the subject's gaze.
- Serial communication with the Hardware layer: The model establishes a communication channel to send commands to the wheelchair's embedded system, enabling control over its movement.

5.1.1 Methodology:

Initially, the system used the Logitech webcam to capture real-time frame-by-frame pictures from a video stream. The initial step was to determine the exact region of the face, followed by the precise position of the eyes. After the system takes real time frames from the Logitech webcam. the frames are converted to a gray frame. Because it is lighter than BRG colored image and saves processing power. Then The Python libraries OpenCV, Dlib and Numpy were used to locate facial landmarks. Using face landmarks recognition technology, the system can identify 68 specific face landmarks. Each point has a specific index.

5.1.2 Face and Eye Detection:

Face detection plays a vital role in identifying human faces in digital images or videos. In this system, the Dlib library is employed to locate faces using various methods.

Dlib Library:

Dlib is a popular open-source C++ library for machine learning, computer vision, and image processing applications. However, there is also a Python library for Dlib which provides an interface to use Dlib's algorithms and tools within a Python environment. The Dlib Python library can be used for various tasks such as object detection, face detection, face landmark detection, and more.

One of the most common applications of the Dlib Python library is in face and facial features recognition. The library provides a pre-trained face detector and facial landmark detector models that can be used to identify and locate faces in images and videos, and then detect specific features within those faces such as the eyes, nose, and mouth.

The face detection process in Dlib involves scanning an image for potential face regions using a sliding window approach. Each window is then passed through a machine learning-based face detector that determines whether it contains a face or not. If a face is detected, the facial landmark detection model is applied to identify key points on the face, such as the corners of the eyes, nose, and mouth.

The facial landmark detection model in Dlib is based on a shape prediction algorithm that uses a combination of regression and shape model fitting to estimate the location of key points on a face. The algorithm is trained on a large dataset of annotated facial landmarks and can handle faces with varying poses and expressions. and achieves fast processing times, as fast as one millisecond.

The facial landmarks detected by the Dlib face detector include:

1. The corners of the left and right eye
2. The tip of the nose
3. The corners of the mouth
4. The center of the top and bottom lips
5. The midpoints of the left and right eyebrows

In addition to these key points, the face detector can also detect a number of other points on the face, such as the cheekbones, jawline, and chin.

These points are then plotted on the region of interest in another image, providing a visual representation of the face with 68 unique facial landmarks. A separate index is supplied for each of the 68 points.

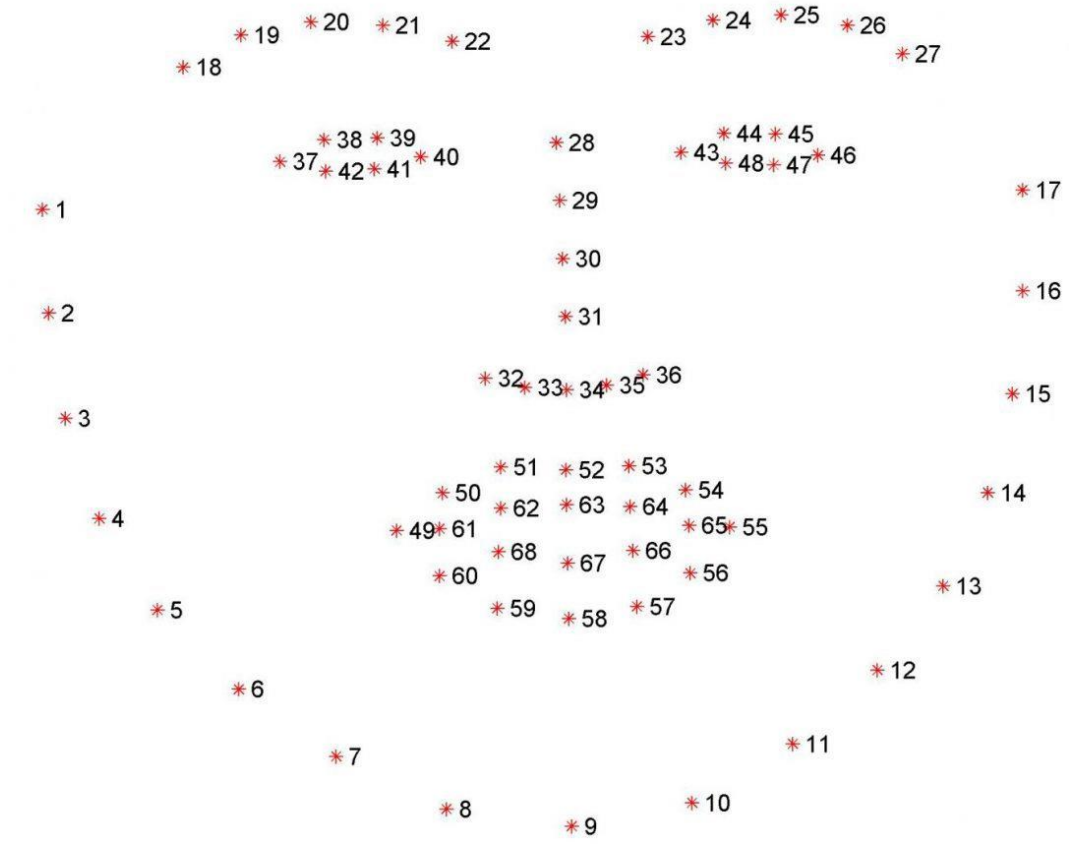


Figure 25 - key points

To accurately detect the eyes, we identify the specific points corresponding to the eyeballs. The Dlib library provides point indices for the left and right eyes. For the left eye, the point indices are 37, 38, 39, 40, 41, and 42. Similarly, for the right eye, the point indices are 43, 44, 45, 46, 47, and 48. (fig) By pinpointing these specific points, we can effectively extract the eye regions and separate them.

5.2.1 Detecting Eye Blinking with Facial Landmarks

The blinking of the eyes can be detected by looking at important facial features. We must examine points 37–48, which represent the eyes, while considering eye blinks. We used an equation to calculate the aspect ratio of the eyes when the eyes blink in real time using facial landmarks. The eye's aspect ratio is a measurement of the condition of the eye opening.

To understand how it works, reference the image above of the 68 facial landmarks, specifically the eyes–notice that each eye is made up of 6 (x, y) coordinates, starting at the left corner of the eye, working clockwise around the remainder of the region. The relationship between these points is referred to as the eye aspect ratio (EAR):

$$EAR = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

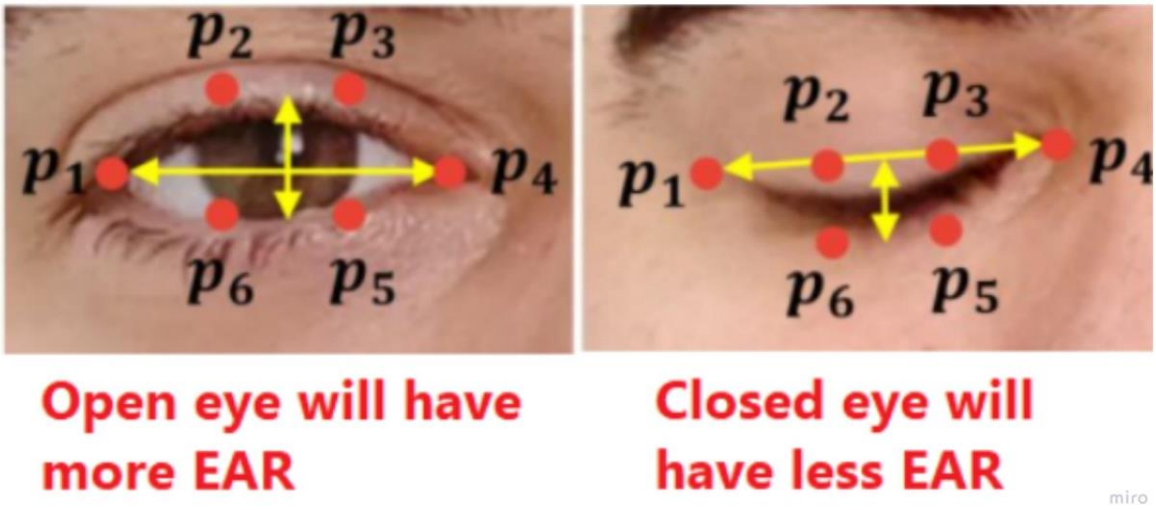
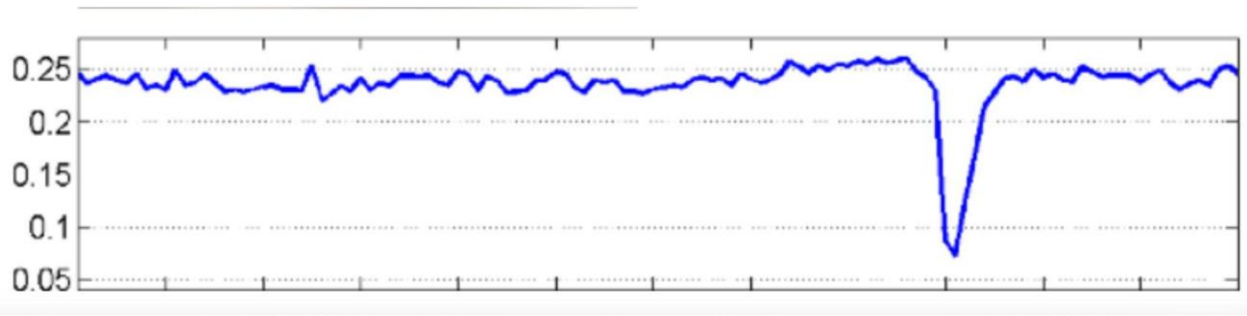


Figure 26

where (for the left eye, for instance) p2 and p6 correspond with points 38 and 42 in the above diagram, p1 and p4 correspond with 37 and 40, and p3 and p5 correspond with 39 and 41. The greater the EAR, the more widely the eye is open. The EAR quickly declines while the eyes are closed as in . When the aspect ratio goes below a particular threshold, the system can detect whether a person's eyes are closed or open.



The eye aspect ratio EAR in Eq. plotted for several frames of a video sequence. A single blink is present.

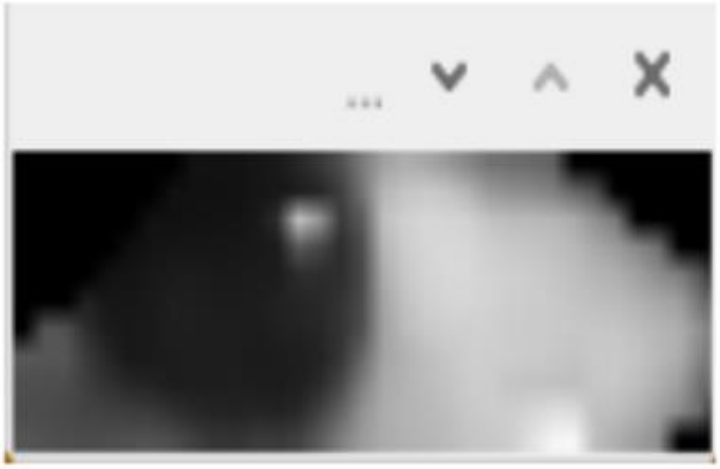
Applying facial landmarks to detect eye movement, the landmarks 37, 38, 39, 40, 41, and 42 relate to the left eye area. The landmarks 43, 44, 45, 46, 47, and 48 belong to the right eye area. After obtaining the left eye's coordinates, we can prepare a mask to accurately extract the region of the left eye while excluding all other inclusions. The right eye is treated in the same way.

Now, the eyes are separated from the face. The picture is sliced into a square form, so the eyes' end points (top left and bottom right) are used to get a rectangle.

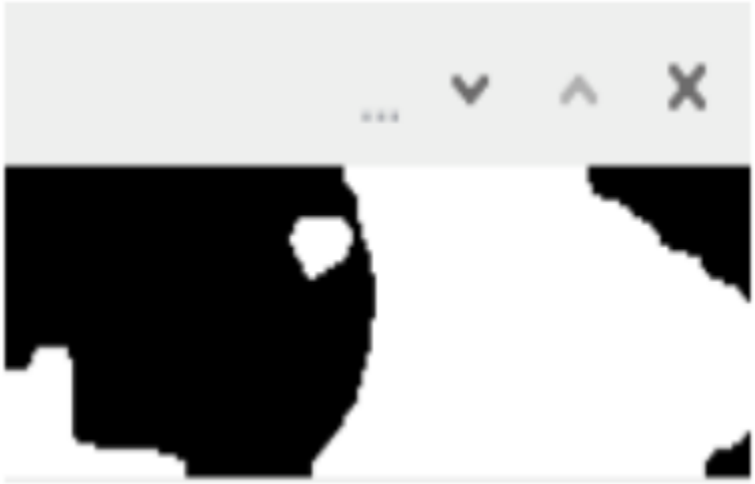
We also determined a threshold value to detect eyes. The eyes are gazing to the right if the sclera is more visible on the left. The eyes were then changed to gray levels, and a threshold was added. Then, we separated the left and right white pixels. The eye ratio was then calculated.

5.3.1 Eye-Gaze Detection and Eye Tracking:

For accurately detecting the gaze direction, the left eye region is identified by landmarks with indices 36, 37, 38, 39, 40, and 41, while the right eye region corresponds to landmarks with indices 42, 43, 44, 45, 46, and 47. The same procedure is followed for both eyes. To extract the inside of the left eye accurately and exclude other facial elements, a mask is created based on the coordinates obtained from the left eye landmarks. The eyes are then separated from the face by cutting a rectangular region using the top-left and bottom-right endpoints of all eyes. A threshold is determined to facilitate eye detection. This process is illustrated below.

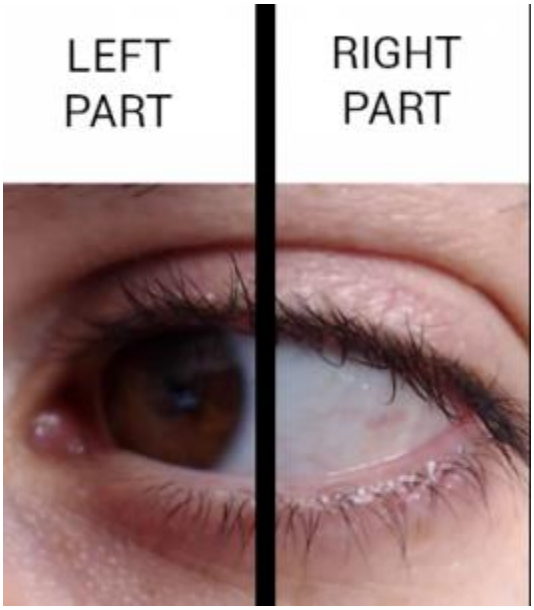


The thresholding process can be easily achieved using the dlib.threshold_image() function, which converts a grayscale image to black and white by applying a specified threshold. This function returns a new image where pixel values are set to either black or white based on the threshold as shown below.



If the sclera (white part of the eye) is more visible on the left side, it indicates that the eyes are looking to the right, To determine the gaze direction, the eyes are converted to gray levels and a threshold is applied to distinguish white pixels. The left and right white pixels are then separated, allowing for the calculation of the eye ratio.

The gaze ratio provides information about the direction of eye gaze. Typically, both eyes tend to look in the same direction. Thus, by examining the gaze ratio of one eye, it is possible to infer the gaze direction of both eyes. To obtain a more accurate estimation, the gaze ratios of both eyes are calculated, and their average value is determined.



When the user looks specifically to the right, the pupils and iris of the eyes point in that direction, while the rest of the eye appears white. Similarly, when the user looks in the opposite direction, the white portion is located on the other side. In the case where the user looks straight ahead, the white portion between the left and right eye regions is balanced. By applying a threshold to separate the iris and pupils from the white areas of the eyes, eye movements can be tracked effectively. Based on these movements, the system provides instructions for wheelchair movement. The system operates by interpreting the position of the eye pupils and controlling the wheelchair, accordingly, allowing movements to the left and right directions.

5.3.2 Brow detection:

In the preceding sections, we have discussed the methodology employed to determine appropriate commands for wheelchair movement based on the subject's gaze direction. However, a crucial challenge remained: how to detect the forward command? To address this issue, we explored several potential solutions, namely asking the subject to look upwards or to raise their eyebrows.

Both alternatives were evaluated through testing, ultimately, we found that raising eyebrows was less visually strenuous compared to looking upwards, while yielding more accurate results.

To effectively detect the raising of eyebrows, it was necessary to calculate the vertical distance between the midpoint of the eyebrow (The landmark with index 25 for the right eyebrow area) and the midpoint of the lower section of the corresponding eye's central region (The landmark with index 47 for the right eye area) .This calculation was replicated for the left eyebrow as well. Importantly, this vertical distance stays almost constant for the resting face, yet exhibited an increase when the eyebrows were raised. Consequently, to detect whether the eyebrows were raised, a threshold was established based on the vertical distance. Once the measured distance passes this threshold, a forward command is initiated.

Through this approach, we successfully determined the activation of the forward command by monitoring the elevation of the subject's eyebrows. Notably, this method proved to be more comfortable for the subject's eyes compared to the alternative of looking upwards, while simultaneously yielding more accurate results.

6.1.1 Implementation:

Our project follows an object-oriented programming (OOP) approach, which involves the utilization of six distinct classes to ensure a robust and highly functional system. These classes serve specific purposes and contribute to the overall functionality of the project. The classes are:

1. Eye module
2. Blink module
3. Calibration module
4. Gaze module
5. Eyebrows module:
6. Movement module

Each class has its own set of functions implemented to its specific responsibilities. By organizing the project into these distinct classes, we achieve a modular and organized codebase, allowing for easier maintenance, scalability, and code reusability.

6.1.2 Eye Module:

The eye class's primary objective is, given webcam input, to create a new frame of the user's eye region by isolating it and extracting relevant information.

The Eye class in this system plays a crucial role in isolating the eye region and extracting relevant information. It provides functionality to calculate various attributes and ratios related to the eye, such as the eye region coordinates, blink ratio, eye width, eye height, and average blinking ratio (EAR).

The main functions of the Eye class are:

- get_eye_region(): This function retrieves the eye region's coordinates using landmarks obtained from the facial landmark's detection. It returns a numpy array of points representing the eye region.
- get_eye_width(): This function calculates the width of the eye by computing the Euclidean distance between the horizontal landmarks of the eye region.
- get_eye_height(): This function calculates the height of the eye by computing the Euclidean distance between the vertical landmarks of the eye region.
- get_blink_ratio(): This function calculates the blink ratio based on the Eye Aspect Ratio (EAR). It involves calculating the Euclidean distances between the vertical and horizontal landmarks of the eye region.
- get_average_blink_ratio(): This function takes the blink ratios of both the left and right eyes and calculates their average. It helps in obtaining an overall measure of blinking for both eyes.

The Eye class provides essential functionality for eye analysis and contributes to the overall functionality of the system, particularly in eye tracking and blink detection.

6.2.1 Blink module:

The Blink class is responsible for detecting eye blinking by calculating the blinking ratio of the eye. It provides functionality to determine whether the eye is blinking or not, set and get the blinking threshold, count the number of blinks, and manage the frames for closed eye detection for determining for how long the eyes are closed.

The main functions and attributes of the Blink class are:

- is_blinking(): This function checks if the eye is blinking based on the Eye Aspect Ratio (EAR). It returns True if the eye is blinking, and False otherwise.
- set_blinking_threshold(): This function sets the blinking threshold value.
- get_blinking_threshold(): This function retrieves the blinking threshold value.
- set_blink_ratio(): This function sets the blink ratio for eye blinking.
- get_blink_ratio(): This function retrieves the blink ratio.
- count_blinks(): This function counts the number of blinks by checking if the eye is closed for a certain number of frames and then open. It returns the total number of blinks.
- set_closed_eye_frame(): This function sets the number of frames that define a closed eye state.

The Blink class main role is to check whether the eye is blinking or not, and how long was the blink? Is it a fast blink or an intended long blink. This is accomplished by counting the number of frames during the closed eye state and calculating the duration of the blink based on this information.

6.3.1 Calibration module:

The Calibration class is responsible for calibrating the eye threshold and blink threshold in the system. Its main functions and attributes include:

- threshold_calibrate(eye_gaze, thresh): This function calibrates the eye threshold by comparing the number of white pixels in the eye region. The threshold is adjusted based on the percentage of white pixels.
- blink_calibrate(): This function calibrates the blink threshold by calculating the mean of the blink ratios that are greater than a certain threshold.
- add_blink_ratio(blink_ratio, blink_threshold): This function adds the blink ratio to the list of blink ratios if it is greater than the blink threshold.
- calibrate(gaze_left, gaze_right, blinking_ratio): This function performs the calibration process for both the eye threshold and blink threshold. It adjusts the eye threshold based on the gaze of the left and right eyes and calculates the blink threshold based on the blink ratio.
- get_cal_blink_threshold(): This function retrieves the calibrated blink threshold value.
- get_cal_left_eye_thresh(): This function retrieves the calibrated left eye threshold.
- get_cal_right_eye_thresh(): This function retrieves the calibrated right eye threshold.
- is_calibrated(): This function checks if the calibration is done.
- is_cal_blink(): This function checks if the calibration is done for the blink threshold.

- is_cal_threshold(): This function checks if the calibration is done for the eye threshold.

The Calibration class ensures that the eye and blink thresholds are calibrated accurately, contributing to the precise functioning of the eye tracking system.

6.4.1 Gaze Module:

The Gaze class is responsible for detecting the direction of the user's gaze (left, right, up) by analyzing the number of white pixels in different parts of the eye region. Its main functions and attributes include:

- get_gaze_ratio(): Calculates the gaze ratio by comparing the number of white pixels in each part of the eye region (left, right, up, center). The gaze ratio value corresponds to a specific gaze direction (0 for left, 2 for right, 1 for up (forward), and 4 for down (stop)).
- get_avg_gaze_ratio(gaze_right, gaze_left): Calculates the average gaze ratio value from the gaze ratios of the right and left eyes.
- get_min_max_eye_region(): Determines the positions of the furthest and nearest points in the eye region.
- get_eye_threshold(eye_region): Generates an eye threshold from the eye region by applying a binary thresholding technique based on a specified threshold value.
- set_threshold(threshold): Sets the threshold value for eye thresholding.
- get_threshold(): Retrieves the current threshold value.
- get_eye_region(): Retrieves the eye region.

6.5.1 Eyebrows Module:

The Brows class is responsible for detecting the eyebrows region, calculating the vertical distance between the midpoint of the eyebrow and the midpoint of the lower section of the corresponding eye's central region, and determining whether the eyebrow is raised or not. Its main functions and attributes include:

- calculate_brow_distance(): Calculates the vertical distance between the midpoint of the eyebrow and the midpoint of the lower section of the corresponding eye's central region.
- get_brow_distance(): Retrieves the vertical distance between the midpoint of the eyebrow and the midpoint of the lower section of the corresponding eye's central region.
- set_brows_threshold(threshold): Sets the threshold value for the brow distance.
- is_up(): Determines whether the brow is raised up or not based on the brow distance and the specified threshold. Returns True if the brow is raised, and False otherwise.

6.6.1 Movement module:

The Movement class is responsible for interpreting the movement command based on the direction of the user's gaze. Its main functions and attributes include:

- run(): Determines whether the movement should be initiated or stopped based on the total number of blinks. If the total blinks are between 2 and 3, the movement is initiated. If the total blinks exceed 3, the movement is stopped.
- driver(): Controls the movement based on the current state of the gaze. If the movement is initiated, it checks the gaze ratio value to determine the direction of movement (forward, left, right). If the gaze ratio is 1, the movement is set to forward. If the gaze ratio is 0, the movement is set to the left. If the gaze ratio is 2, the movement is set to right.
- move_counter_reset(): Resets the counters for different directions (left, right, center, up) to 0. This is done for a certain number of frames when the user is not blinking.
- move_forward(), move_up(), move_left(), move_right(): Sets the movement direction (forward, up, left, right) based on the gaze ratio and the number of frames for which the condition is met. The movement direction is maintained until the specified number of frames has passed.
- stop(): Stops the movement and resets the direction counters. This is triggered when the gaze ratio is not 1, 0, or 2.
- set_gaze_ratio(gaze_ratio): Sets the gaze ratio value.
- set_brow_status(brows_status): Sets the status of the brows (raised or not).
- Methods starting with is_ check the current state of movement (forward, up, left, right, stopped).
- set_total_blinks(total_blinks): Sets the total number of blinks.
- set_eye_direction_frame(eye_direction_frame): Sets the number of frames required to detect a consistent eye direction.

The main.py file is the central script that brings together various modules to complete the gaze project. Here's an explanation of how the modules work together:

1. Importing Modules: The necessary modules, including math, numpy, cv2, dlib, and the custom modules Gaze, Eye, Blink, Calibration, Movement, and Brows are imported.
2. Initialization: Constants and variables are defined, such as the number of closed eye frames, eye direction frames, calibration frames, and the path to the facial landmark model.
3. Object Instantiation: Objects of the Calibration, Movement, and Blink classes are created to handle calibration, movement control, and blink detection, respectively.
4. Set Frames: The number of calibration frames and eye direction frames are set in the Calibration and Movement objects, respectively. The number of closed eye frames is set in the Blink object.
5. Main Loop: The main program runs inside a while True loop.
6. Video Capture: The script captures video frames from the camera using cv2.VideoCapture. The captured frame is then flipped horizontally to account for the mirror effect.
7. Face and Landmark Detection: The **Dlib** library is used to detect faces in the grayscale frame. Facial landmarks are extracted using the **shape_predictor** model.

8. Driver Execution: The program integrates eye and eyebrow detection, blink and gaze detection, calibration, driver execution, and visual feedback to enable gaze-based movement control.
9. Exception Handling: Error handling is implemented to catch and display any errors that occur during the execution of the program.

5. Evaluation and Performance Analysis:

The performance measure criteria is evaluated based on:

1. Calibration time.
2. Accuracy.

According to calibration time: During the testing time, it is very fast in the order of a millisecond.

According to accuracy: Accuracy was calculated by counting the gaze detection for 3 subjects (S1, S2, and S3). For each subject, 40 gazes are tested, 10 for the right gaze, 10 for the left gaze, 10 for the blinking eye, and 10 for the raised eyebrows as shown in Table (xyz).

Where:

$$Accuracy = \frac{No.of\ successful\ instruction}{total\ No.of\ instructions\ given} * 100\%$$

Subject No.	S1	S2	S3
Total Detections No	40	40	40
Total no. of max left Detection	8/10	9/10	9/10
Total no. of max Right Detection	10/10	9/10	9/10
Total no. of max blinking Detection	9/10	10/10	8/10
Total no. of max raised eyebrows Detection	10/10	10/10	10/10
Total no. of max Detection	37/40	38/40	36/40
Accuracy	92.5%	95%	90%
Error	7.5%	5%	10%

The obtained average of accuracy is 92.5%. The results clearly demonstrate the effectiveness of the utilization of the Facial Landmark Detection technique with Dlib. The chosen technique, showcasing its ability to accurately determine the direction of gaze. Moreover, it is noteworthy that the calibration process, which is essential for accurate gaze detection, is accomplished within a negligible timeframe, taking no more than a millisecond.

7.1.1 Brain Control Mode

7.1.2 Introduction

BCI, What, Why, and When?

The first attempt towards direct brain communication was made in 1967 to send Morse code messages based on amplitude changes detected in alpha band of subjects' EEG. The change in amplitude is voluntarily controlled by the subjects.

BCI provides users with an interface to control and perform communication with a system without any physical movement. Initially BCI were considered to be used as a clinical tool to rehabilitate patients with severe motor disabilities. But now it is gradually coming out of the clinical environment and being used in other domains without restricting the focus on disabled subjects.

There are four major building blocks of the BCI system. These include signal acquisition, Signal processing, target application or output device, and operating protocol or paradigm. The following diagram shows a clear view of a closed loop BCI system.

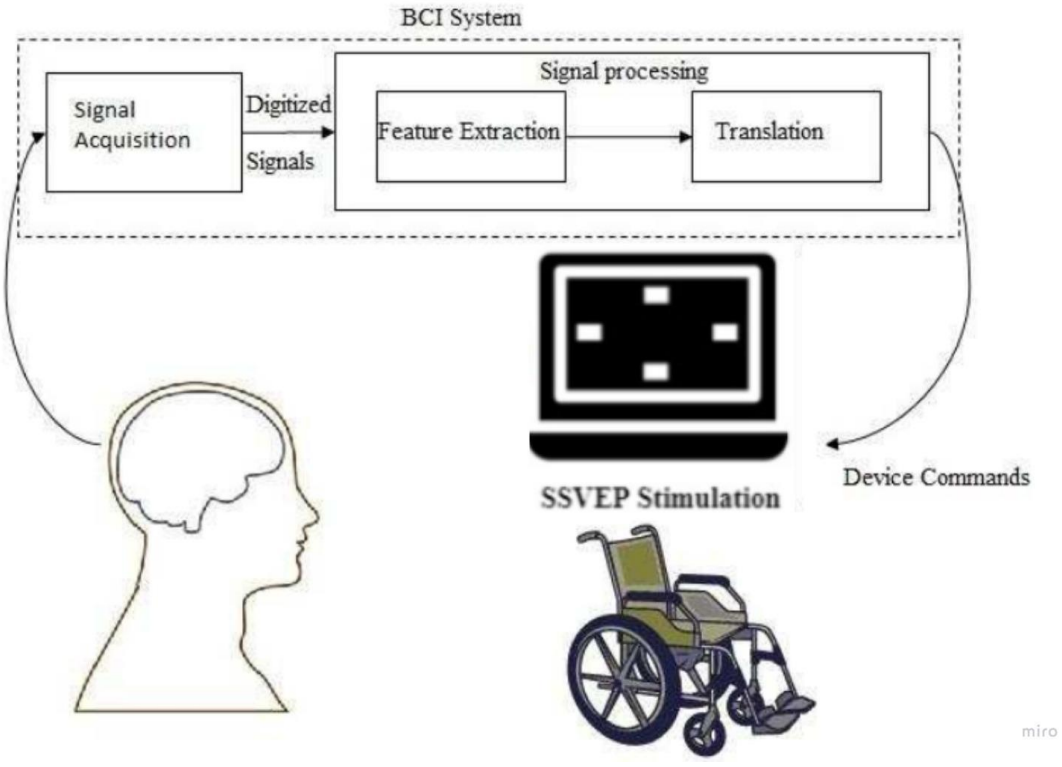


Figure 27 - BCI system

BCI receives brain signals as an input from the human brain by employing some transducers. These signals are processed to extract features which are then translated to a discrete command-set to operate some external device or application e.g. a controller application or a wheelchair etc.

Prime objective of a BCI system is to detect subjects' intentions towards some activity related to problem solving. These intentions are related to specific brain patterns, which are being identified and captured in near real-time. Due to limitations in recording techniques, it is not possible to specify brain patterns for each user intention to provide an unlimited command set. EEG is the most widely used recording technique used in BCI systems due its high temporal resolution and non-invasiveness. EP and ERP are signal processing derivations of EEG, which are quite frequently used in BCI research. EP involves averaging of EEG responses locked in time to aural or visual stimulus, while ERP involves a further complex stimulus processing to averaged EEG activity.

With technological advancements and improvements in signal processing algorithms, BCI is widening its scope and looking for new adobes. BCI at its maturity might change the whole world because of unlimited boundaries and rich content, although for now it seems like a speculation but the glimpses it is showing to the world are quite remarkable. At present BCI is used to develop many practical applications in different domains such as clinical rehabilitation, forensics, virtual reality, gaming, robotics, etc.

7.2.1 BCI Types:

A BCI can be categorized on the basis of electrode positioning and the type of activity the subject's brain is performing.

BCI Based on Electrode placement:

Invasive BCI:

Invasive BCI systems require implantation of an array of micro-electrodes in brain tissue in order to acquire neural signals. Such implantation of electrodes requires a surgery, which can be complicated.

Figure 28

Invasive BCI are generally used to make locked-in paralyzed patients perform some basic tasks, one of the first successful invasive BCI helped the subject to have control over cursor movement and speller application . In few invasive studies, resulting BCI is used to perform complex activities as well, such as controlling a robotic arm and television remote control etc.

Non-invasive BCI:

In non-invasive BCI electrodes are placed on the scalp to acquire signals from the human brain. Noninvasive BCI are more popular and feasible than invasive BCI in devising real life applications targeting both healthy and disabled subjects. Transducers used in non-invasive BCI are quite safe and easy to wear but their signal resolution is relatively poor due to signal dampness caused by scalp, skull and other membranes over brain tissue.

Non-invasive technologies measure brain activity at the scalp hence there is no need for any surgical implantation. These technologies are widely used in many research topics due to their ease of use. During this Project non-invasive technology EEG is employed.

EEG :

Electroencephalography or EEG is a non-invasive brain imaging technology that measures 5–100 μv electrical potentials by electrode placement at the scalp. EEG signals are quite noisy because of artifacts and obstructions created by scalp skin, bone tissue and cortical fluid. EEG signals exhibit excellent temporal resolution with small latencies but their spatial resolution is quite poor and show accuracy in a range of about 2-3cm. Spatial resolution is quite important in classification between different brain states, so EEG setup requires careful placement of electrodes based on the standard discussed in the next section. There are three main derivatives of EEG which makes it possible to use EEG in different experimental scenarios

Rhythmic Responses:

Such kind of brain activity is associated with different known and observed frequency bands like delta (0.1-3.5 Hz), theta (4-7.5 Hz), alpha (8-13 Hz), beta (14-30 Hz) and gamma (>30Hz).

Following are the main treats which make EEG a most widely studied non-invasive brain imaging technology.

- It is relatively cheap than other brain imaging technologies.
- It provides excellent temporal resolution.
- It is directly correlated with functional neural activity.
- It allows implementation of real-time BCI systems.
- Its acquisition setup is quite easy to use and risk free.

Standard Electrode Positioning:

There is an effort made by “International Federation of Societies for Electroencephalography and Clinical Neurophysiology” to standardize electrode placement on the skull to acquire brain signals by adopting a system proposed in. This system is generally referred to as the 10-20 system for electrode placement.

It has been adopted globally since then, in its original form i.e. nomenclature and physical placement of 21 electrodes. Further enhancements are also made by adding more electrode positions.

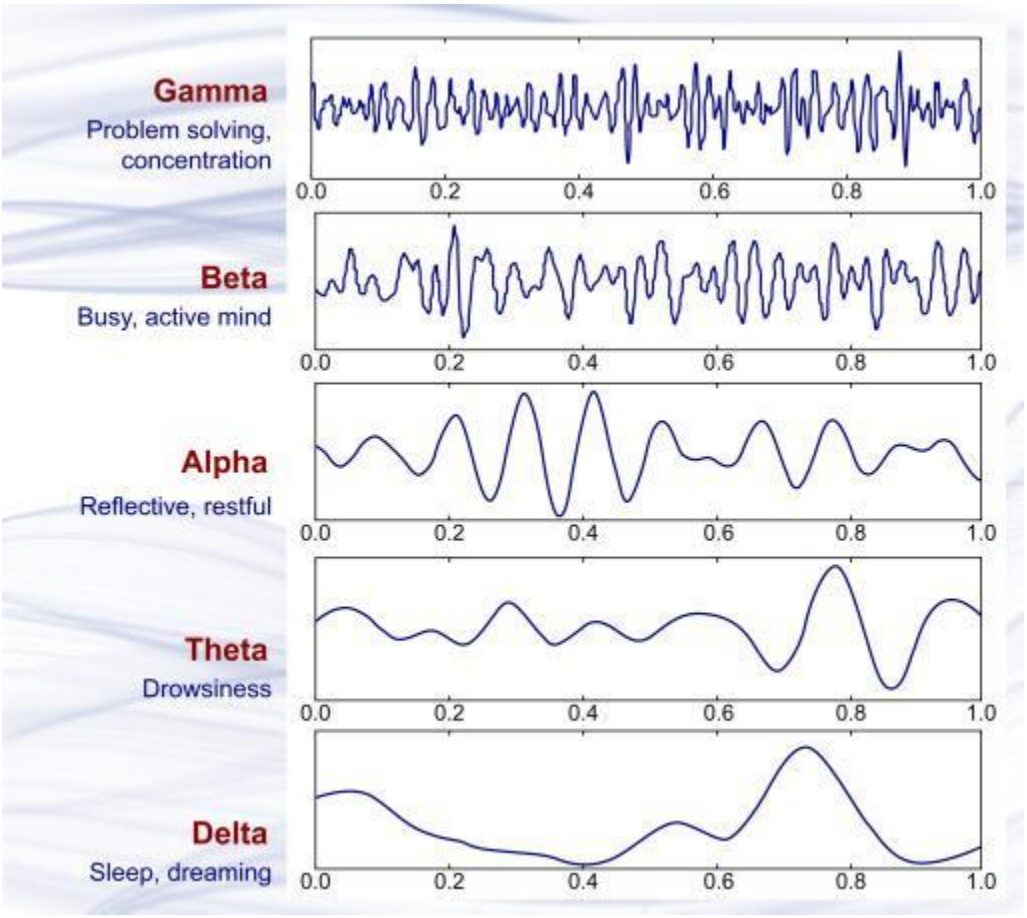


Figure 29 - brain signals

BCI Based on Brain Activity:

Human brain is capable of performing its operations in different states like active, passive and reactive. In order to accommodate these different states of the brain, it is required to set up a BCI system that fits in with the requirements of that particular state.

Active BCI:

In active BCI, the subject is required to actively perform some cognitive task like mental calculation, imagined word association, and motor imagery i.e. imagining physical movement of left or right hand. Mental activity in response to such cognitive tasks is the prime driving force for Active BCI. Active BCI requires quite extensive user training in order to accurately classify between different mental tasks performed by the user. Such training takes even months to achieve acceptable performance. Motor imagery is the most common and widely used paradigm in active BCI where different imagined movements exhibit changes in SMR.

Mental tasks other than motor imagery can be distinguished using different frequency bands like delta (0-3 Hz), alpha (8-13 Hz), beta (14- 20 Hz), and theta (4-7 Hz). These bands are considered as empty slots for different mental tasks and thoughts.

Reactive BCI:

Reactive BCI formulates its output from the brain activity that occurred in response to some external stimulus. User is required to maintain his or her focus on desired visual or aural stimulus in order to make the system work properly.

Main advantage of reactive BCI is that it requires very minimum or no training to adapt to the system. They are rather more robust and easy to set up than active and passive BCI. Downside of such BCI systems is that they are always dependent on external stimuli and sometimes these stimuli are quite annoying especially in the case of flickering visual stimuli.

Major paradigms in reactive BCI are P300 and SSVEP which are elicited by focusing on specific types of stimulus. P300 or simply P3 also referred to as odd-ball paradigm is a positive peak elicited around 300ms after stimulus presentation.

On the other hand SSVEP is elicited in the visual cortex by maintaining focus on a stimulus flickering at constant rate between 3-75 Hz, with similar frequency as of focused stimulus. For example if a subject is focusing on a stimulus that flickers at 10 Hz, the subject's brain response (SSVEP response) exhibits the same frequency along-with its harmonics and subharmonics. SSVEP responses are quite robust and require very minimal training. Probability of SSVEP elicitation is quite higher between 5-20 Hz frequency band.

In this project, we have chosen to employ the non-invasive reactive Steady-State Visually Evoked Potential (SSVEP) paradigm as the underlying methodology for our study. This section provides comprehensive details on the methodology of the SSVEP paradigm, outlining the key components and procedures involved in its implementation.

7.2.2 SSVEP

Steady State Visually Evoked Potentials or SSVEP is an exogenous BCI paradigm that measures the electric potential differences in visual cortex in response to a constantly flickering visual stimulus.

It is an extension to VEP (change in electric potential due to a visual stimulus)), with visual stimuli flickering at constant frequencies in a range 6-75 Hz. SSVEP responses are prominent in occipital and occipito-parietal regions. A typical electrode placement for SSVEP paradigm includes O1, O2, Oz, PO3, PO4, PO7, PO8, and POz standard electrode positions as shown below.

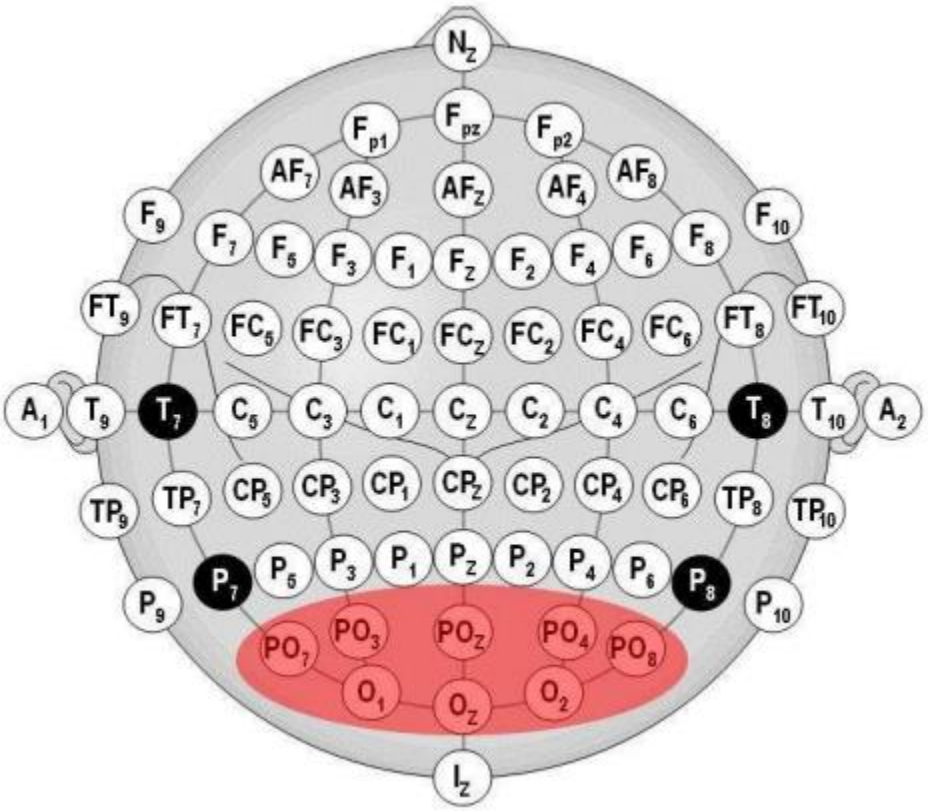


Figure 30 - SSVEP

In the SSVEP paradigm, a stimulus flickering at constant frequency is presented to the user, maintaining focus on the stimulus results in achieving the brain's steady state. SSVEP is elicited due to synchronization of the subject's neural activity and constant flickering visual stimulus. For example if a user is focusing on a stimulus flickering constantly at 10 Hz, brain responses will exhibit the same frequency along with its harmonics. Few important facts about SSVEP are,

- SSVEP is widely used in non-invasive BCI research.
- SSVEP responses are quite robust.
- SSVEP signals show a good signal to noise ratio.
- SSVEP responses are generated normally in frequency range 6-75 Hz.
- Best SSVEP responses can be obtained in a frequency range 6-20 Hz , within this range 10Hz and 16-18Hz are rather more favorable candidates.

Design Considerations for SSVEP Stimuli:

A careful consideration is required for SSVEP stimulus selection and design. There are different important factors that can affect the strength of SSVEP responses i.e. type, frequency, shape, amplitude, contrast, and color etc.

The relationship between visual stimulation and SSVEP-evoked amplitudes, showing that the amplitude of SSVEPs peaks at 15 Hz, forms a lower plateau at 27 Hz, and declines further at higher frequencies (>30 Hz) as shown in

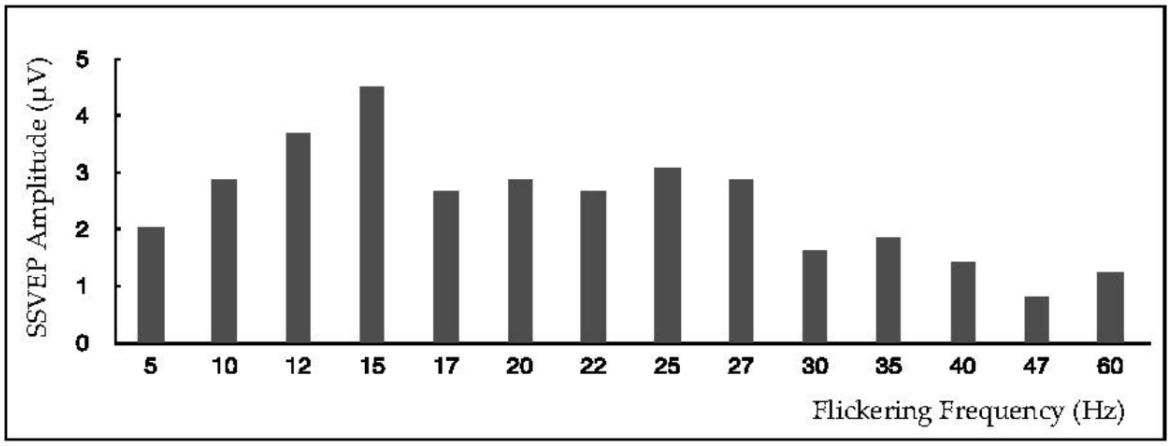


Figure 31

Motivations for Using the SSVEP Paradigm:

The system is ready-to-go which means users do not require any previous training or experience in order to actuate the wheelchair.

Recently, SSVEP BCI systems have gained a special place in the BCI paradigms continuum because of having a variety of different possibilities. SSVEP BCIs are useful in different applications, especially the ones that need some major requirements as follows:

- Large number of BCI commands is necessary (in SSVEP BCI limitations are mostly defined only by the design).
- High reliability of recognition is necessary (in SSVEP BCI, patterns are clearly distinguishable by frequency).
- No training (or just a short time training for classifier training) is allowed.

From a practical point of view, the advantages of SSVEP BCI systems can be summarized as follows:

- User is allowed to have small eye movements.
- User is capable of mild but sustained attention effort.
- User's visual system is not engaged in other activities.
- Visual stimulation can be performed by usual equipment like computer display or LED panel.

Limitations on SSVEP paradigm:

In low-frequency stimulation, SSVEP detection is more accurate. In spite of its favorable detection properties, this band presents two major inconveniences.

1. According to visual perception studies, stimulation frequencies in this band are rather annoying and tiring for the subject
2. The risk for inducing photo epileptic seizures is higher for stimulation frequencies in the 15 – 25 Hz.

A simple solution could be in using higher stimulation frequencies. From empirical and subjective evidence, the threshold could be set to 40 Hz for low stimulation.

BCI System Algorithm And Model Design:

The basic process for controlling a BCI-controlled wheelchair involves several steps:

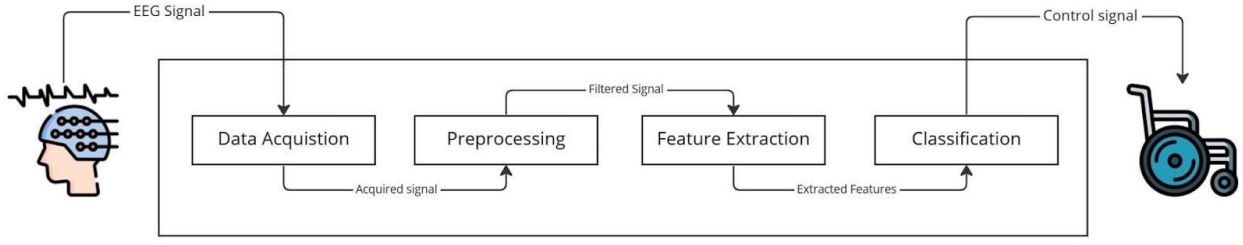
Signal acquisition: The user wears an electroencephalography (EEG) headset, which is a type of BCI device that measures brain activity. The headset collects electrical signals from the user's brain, which are then transmitted to the wheelchair for further processing.

Signal preprocessing and filtration : The signals collected by the EEG headset are typically noisy and need to be preprocessed before they can be used to control the wheelchair. This preprocessing step typically involves filtering the signals to remove noise, as well as extracting relevant features from the signals that can be used for classification.

Classification: The preprocessed signals are then fed into a classification algorithm, which is a type of machine-learning model that can identify specific patterns in the signals. The classification algorithm is trained to recognize specific patterns in the signals that correspond to different commands, such as "move forward" or "turn left."

Interpreted commands: Once the classification algorithm has identified the patterns in the signals, it interprets them as commands and sends them to the wheelchair. The wheelchair then uses these commands to move in the desired direction.

Model Design:



In the subsequent sections, we will provide a comprehensive exploration of the methodology employed in this project, elucidating the implementation details of each step.

Signal Acquisition:

There is a remarkable tendency in using flickering visual stimuli interfaces, assigning a unique frequency to an action, which allows to recognize the user's intentions. The optimal frequency range for the classification of such signals has been a research topic alone.

Acquisition device:

The EEG signal registration Emotiv Epoc headset was placed in each subject, some of the technical descriptions include a sampling frequency of 128Hz, The headset has 14 electrodes and two ground references, distributed in the international system 10 - 20 as shown below

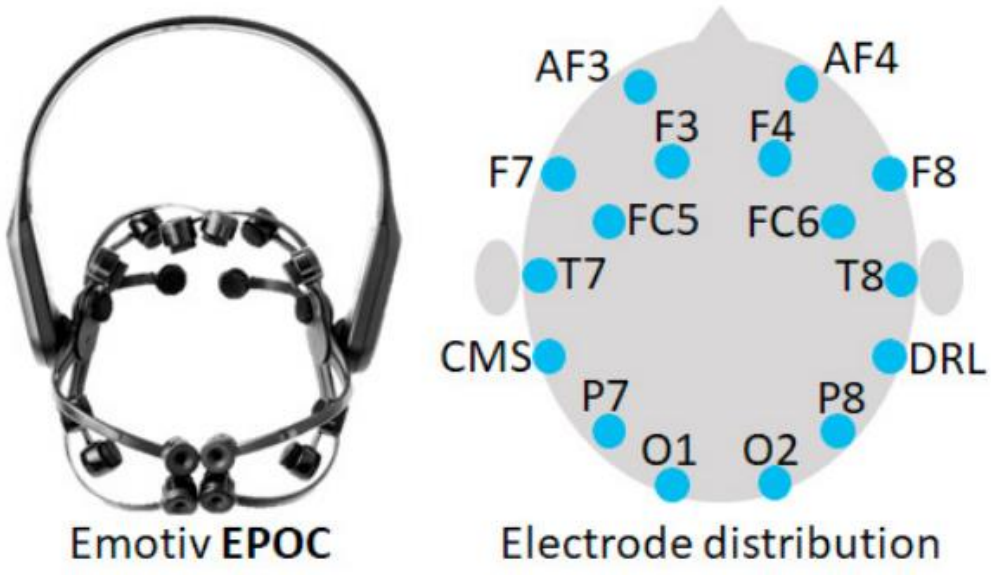


Figure 32 - emotiv epoc

During the EEG signal recording process, a conductive gel was used to reduce the impedance between the electrodes and the scalp which significantly improved the integrity of the recorded signals. The details of the electrodes the Emotiv EPOC device has are as follows:

• EEG.AF3: Electrode located in the frontal area of the brain in the left hemisphere.

• EEG.F7: Electrode located in the temporal frontal area of the brain in the left hemisphere.

• EEG.F3: Electrode located in the frontal area of the brain in the left hemisphere.

• EEG.FC5: Electrode located in the central frontal area of the brain in the left hemisphere.

• EEG.T7: Electrode located in the temporal area of the brain in the left hemisphere.

• EEG.P7: Electrode located in the parietal area of the brain in the left hemisphere.

• EEG.O1: Electrode located in the occipital area of the brain in the left hemisphere.

• EEG.O2: Electrode located in the occipital area of the brain in the right hemisphere.

• EEG.P8: Electrode located in the parietal area of the brain in the right hemisphere.

• EEG.T8: Electrode located in the temporal area of the brain in the left hemisphere.

• EEG.FC6: Electrode located in the central frontal area of the brain in the right hemisphere.

• EEG.F4: Electrode located in the frontal area of the brain in the right hemisphere.

• EEG.F8: Electrode located in the temporal frontal area of the brain in the right hemisphere.

• EEG.AF4: Electrode located in the frontal area of the brain in the right hemisphere.

Frequencies used:

four flickering boxes are used and displayed on a screen infront of the subject to demonstrate the 4 commands for the wheelchairs, the frequencies we chose are as follows,

- 9 Hz for Left command
- 11 Hz for Down command
- 13 Hz for Right command
- 15 Hz for Forward command

Selected Electrodes:

As mentioned before, SSVEP responses are prominent in occipital and occipito-parietal regions. A typical electrode placement for SSVEP paradigm includes O1, O2, Oz, PO3, PO4, PO7, PO8, and POz standard electrode positions as shown below.

To streamline the data acquisition process, we focus on recording signals only from the O1 and O2 channels, which are conveniently accessible through the Emotiv Epoch headset.

Record and store data

Recording data requires designing a session environment and managing users' files carefully. So, in this phase we designed a simple GUI to configure sessions: Username, id, flashing time, frequencies of the boxes, sequence of flashing boxes, … etc. After the configuration, a screen with 4 flickering boxes is displayed, each box is flickering by unique frequency, and display one command (Left, right, down, forward). The flickering of the boxes is captured with the EMOTIV EPOC+ EEG headset

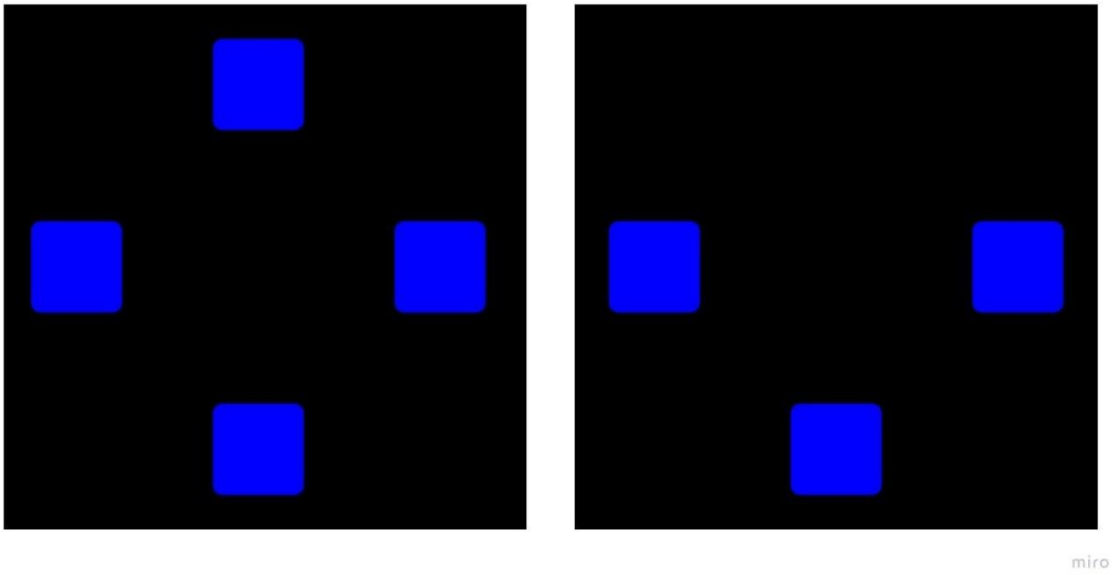


Figure 33 - Gui implementation

In order to perform additional data preprocessing, visualization, and train the AI model, it was necessary to record and save the signals into CSV files.

Signals are recorded into sessions, 5 sessions are recorded succesfully, lasts for ~ 3 minutes, each session consists of 20 command.

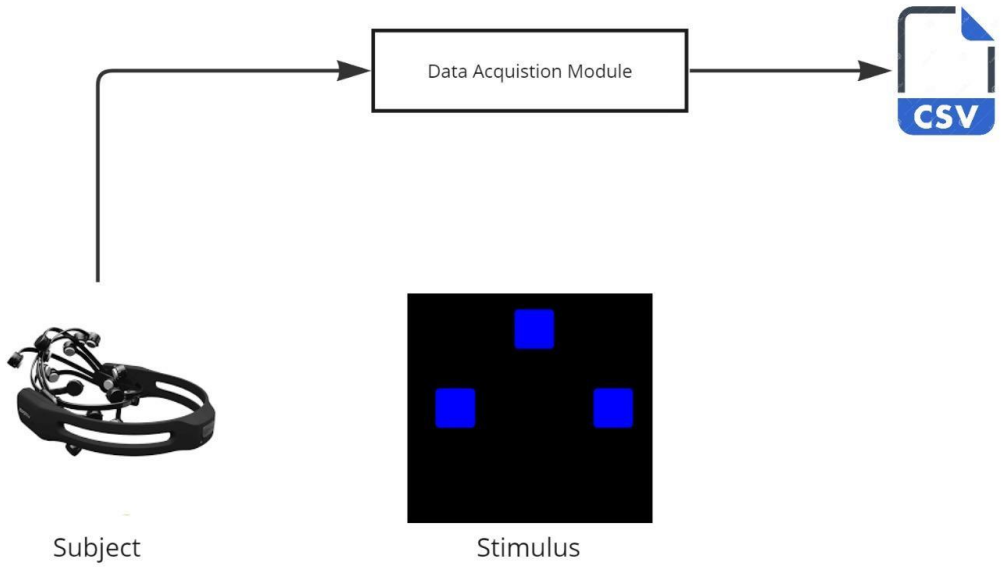
At the beginning of each session, we must make sure all sensors are wet and well connected using the EMOTIV BCI program. We also should make sure the screen has a 60HZ refreshing frequency. The session should be recorded in a quiet place and the volunteer should be relaxed and focused on the flickering boxes. Also, the flickering boxes should be randomly distributed to force the user to keep his mind focused, also the frequency distribution in each session should be equal

The basic characteristics for each session are as following:

- 2 seconds of preparation
- 4 seconds of recording the command signals
- 2 seconds of rest

The process repeats for all commands untill the session is complete.

following diagram illustrates the process of data acquisition and recording:



7.3.1 GUI Implementation:

the GUI is buit using python programming language, The code utilizes the Pygame library, which is a popular library for building games and graphical applications in Python. Pygame provides functionalities for creating graphical windows, handling input events, drawing shapes, and managing animations. It is well-suited for creating interactive visual displays, making it a suitable choice for the SSVEP stimulus GUI.

The GUI utilizes threading to achieve concurrent execution of multiple boxes and ensure the smooth operation of the graphical user interface. Let's dive into how threading is implemented in this GUI.

Threading is a technique in Python that allows multiple threads of execution to run concurrently within a single program. Each thread represents an independent flow of control and can perform tasks concurrently, enabling efficient utilization of system resources and enhanced responsiveness.

In Python, threading is facilitated by the threading module, which provides classes and functions for creating and managing threads.

In the GUI code, each box is associated with its own thread of execution. This approach enables simultaneous operation and independent control over multiple boxes. Here's how threading is implemented in this GUI:

The BlankboardStimulus class creates and manages instances of the Box class, which represent the individual boxes on the screen. Each box runs on its own thread using the threading capabilities provided by Python.

When the GUI is running, the main thread handles the overall control flow, event handling, and GUI updates. Concurrently, each box runs on a separate thread, independently updating its state and visual appearance.

The independent execution of each box thread allows for concurrent animation and responsiveness. While the main thread handles user input and GUI updates, the box threads continuously update and redraw their respective boxes based on their frequencies.

Proper synchronization mechanisms are crucial to ensure thread safety and avoid conflicts when multiple threads access shared resources. In the code you provided, there don't appear to be explicit synchronization mechanisms implemented, which could potentially lead to race conditions if the boxes access shared data. However, the specifics of synchronization mechanisms, if any, may exist in the omitted parts of the code.

It's important to note that since Pygame's display and event handling functions are not thread-safe, the GUI typically uses a single main thread for these operations to avoid potential issues.

Benefits of using Threading in the GUI:

Threading in the SSVEP stimulus GUI offers several advantages:

- Parallelism: Threading enables concurrent execution of multiple boxes, allowing for simultaneous visual stimulation and independent control over each box's behavior.
- Smooth Animation: By updating each box on its own thread, the GUI can achieve smooth animation and responsive visuals.
- Real-time Interaction: Threading helps maintain real-time interaction between the user and the GUI. While the boxes are running their threads, the main thread remains responsive to usr input and updates the GUI accordingly.
- Efficient Resource Utilization: Threading allows for efficient utilization of system resources by leveraging multiple threads for parallel execution, making the GUI more efficient and responsive.

Signal preprocessing:

Recorded data must be analyzed and visualized before passing it through the training phase for many reasons:

1- check the headset works as expected and does not suffer from configuration or hardware problems.

2- check if there is any kind if noise in data to find a solution

3- make sure the data recorded is reasonable due to its Label

4- check the pattern between data of each frequency

5- get the sense of differences in data between different users for prepose of generalization (to build model can work for many users not overfit a specific subject)

Noise elimination:

Noise sources, such as muscle artifacts, eye movement and blinking, power line interference, and interference from other devices, pose significant challenges to obtaining reliable EEG signals.

Sources of noise in SSVEP data can vary, but some common sources include:

1. Environmental Noise: This includes background noise from surrounding electrical devices, room lighting, and electromagnetic interference (EMI) from nearby electronic equipment.
2. Physiological Artifacts: These are noise components introduced by the human body, such as muscle activity (electromyographic noise) and eye movements (ocular artifacts), which can contaminate the SSVEP signals.
3. Electrode and Sensor Noise: Imperfections in the electrodes or sensors used to record the SSVEP signals can introduce noise. This can include electrode impedance fluctuations, thermal noise, and sensor-related noise.
4. Interference from Other Brain Signals: SSVEP signals are often recorded alongside other brain activity, such as electroencephalogram (EEG) signals. The presence of unrelated brain signals can contribute to the overall noise in the recorded SSVEP data.

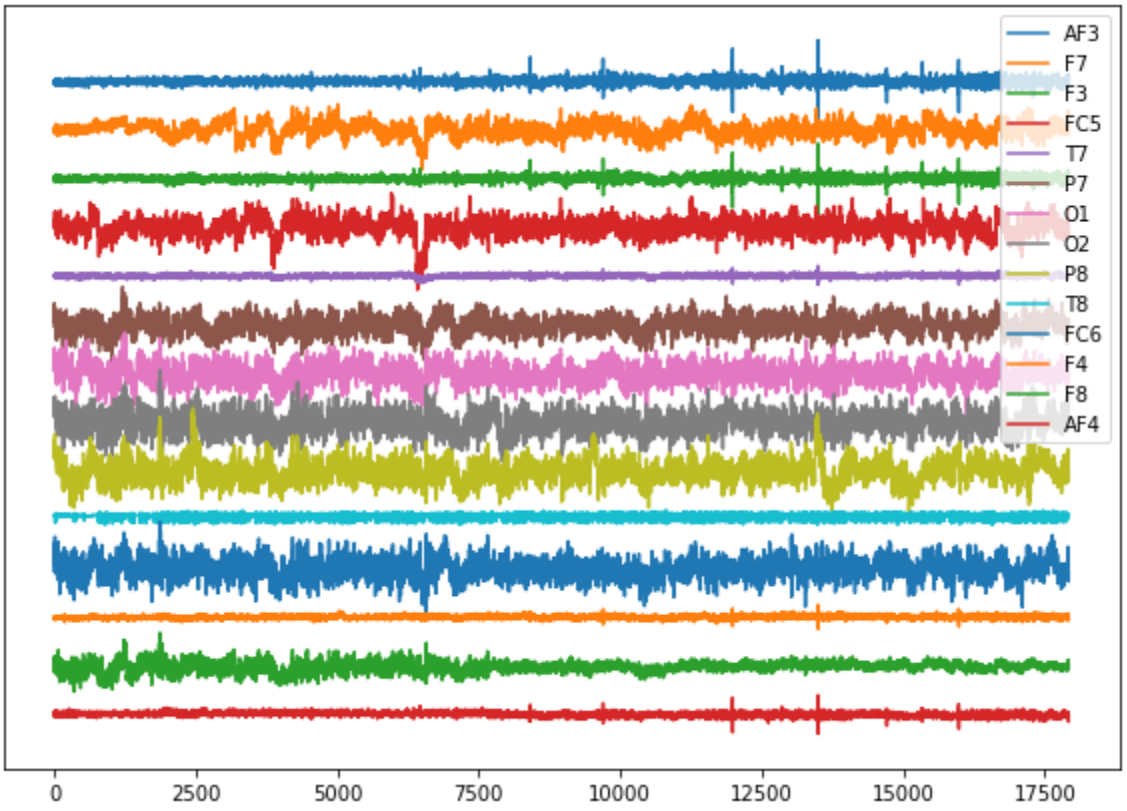
The Common Average Rejection (CAR) method is effective in removing common noise sources shared among sensors.

CAR plays a vital role in mitigating these sources of common noise by creating a reference signal that represents the average activity of all sensors, effectively canceling out shared noise sources. Consequently, and it minimizes the impact of common noise,

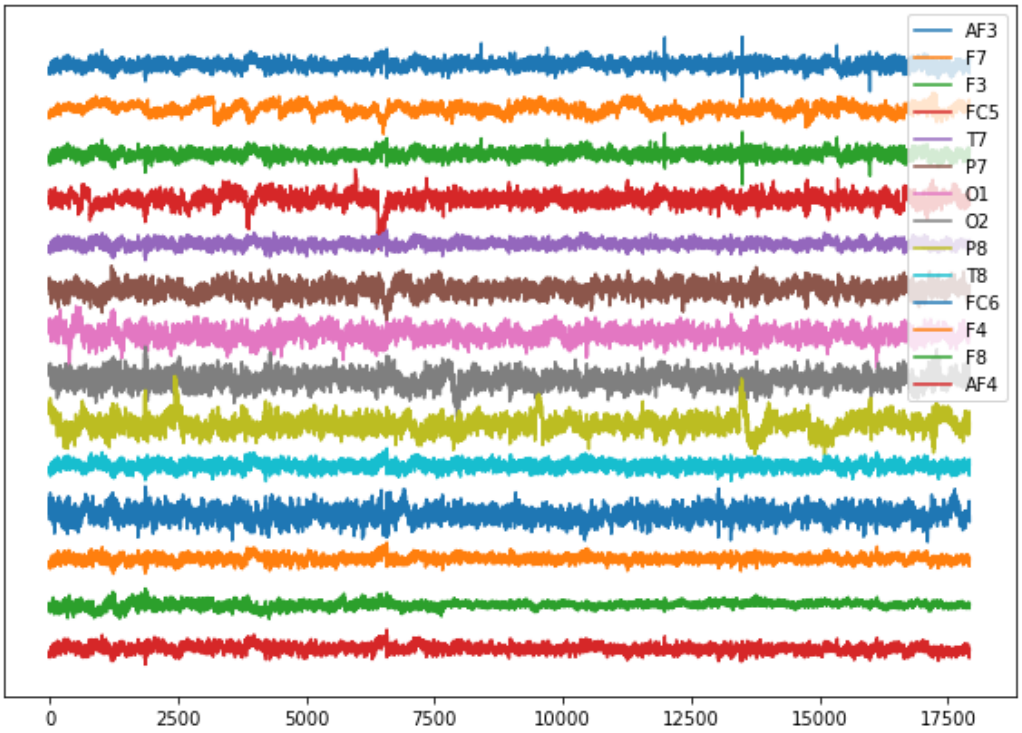
CAR is commonly used in EEG, where it is necessary to identify small signal sources in very noisy recordings. CAR is a computationally simple technique and therefore amenable to both on-chip and real-time applications. As the name implies, an average of all the recordings on every electrode site is taken and used as a reference. Through the averaging process, only signal/noise that is common to all sites (correlated) remains on the CAR. Signal that is isolated on one site (single-unit activity) does not appear on the CAR, unless the signal is so large as to dominate the average. Uncorrelated random noise with a zero mean is minimized through the averaging process. Because the CAR provides an accurate representation of correlated noise at the location of the microelectrode array, but minimizes the contribution of uncorrelated noise sources, we hypothesize that common average referencing will improve neural recording quality with respect to both large and microelectrode references.

The time domain visualizations presented in this section illustrate the changes in signal characteristics before and after the application of the CAR method. The plots clearly demonstrate the notable improvements achieved through CAR processing.

Initially in time domain visualization of the raw SSVEP signals, it is evident that the recorded signals contain considerable noise, including common noise components shared across sensors. This noise can obscure the underlying SSVEP responses of interest and hinder accurate analysis.



After applying the CAR method to the raw SSVEP signals, a distinct change can be observed in the subsequent time domain visualization. The resulting signals exhibit a substantial reduction in noise, particularly in the common noise components. This reduction in noise leads to cleaner and more discernible SSVEP signals, facilitating a more accurate analysis of the desired brain activity.



The visualizations clearly highlight the effectiveness of the CAR method in attenuating common noise components within the recorded SSVEP signals. By subtracting the average activity across all sensors, CAR successfully diminishes the shared noise, enabling the extraction of the true SSVEP responses from the underlying neural activity.

Signal Filtration:

The Recorded SSVEP signals often contain unwanted frequency components and noise, which can interfere with the detection and analysis of the desired SSVEP responses. To address this issue, a filtering process is applied to the raw data, with the purpose of isolating the frequency range relevant to the SSVEP signals of interest.

We designed a fifth order Butterworth bandpass filter to extract the needed range [5Hz-40Hz]. The Butterworth filter is chosen for its favorable attributes, including a flat frequency response within the passband and a steep roll-off beyond the cutoff frequencies.

The butter_band_filter function is implemented as follows:

- The function takes a dataframe representing data from a single trial as input, along with the parameters lowcut and highcut defining the desired frequency range. Additional parameters such as FS (sampling frequency), Order (filter order), and train (training mode) are also provided.
- During the filtering process, the function applies a combination of low-pass and high-pass filters to each channel of the dataframe.

- The low-pass filter (butter_lowpass_filter) is used to attenuate frequencies above the highcut value, while the high-pass filter (butter_highpass_filter) is used to attenuate frequencies below the lowcut value.
- The filtered values are then assigned back to the corresponding channel in the dataframe.

Feature Extraction:

After the signal processing phase, we go through the feature extraction and selection. There are two ways to extract features either using signals in time domain or use it in frequency domain.

For our approach, we have chosen the frequency domain. We employed the Welch method to calculate the power spectral density (PSD) of the signal. This method allows us to obtain a representation of the power at different frequencies within the signal. By applying the Welch method, we can extract the power values associated with the frequencies used to stimulate the subjects during the recorded sessions.

In the following Plot a visualization for the welch result after applying it to a signal containing visual stimuli with a frequency of 11 Hz, representing the "Forward" command in our project. The visualization of the Welch's result reveals a prominent peak at approximately 11 Hz, indicating a strong power component at that frequency.

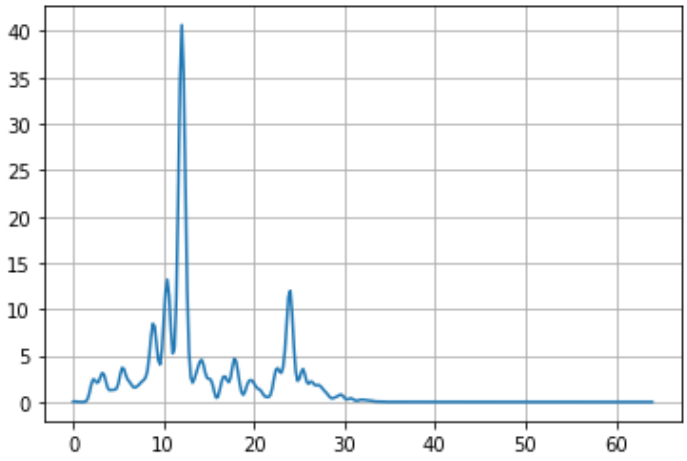
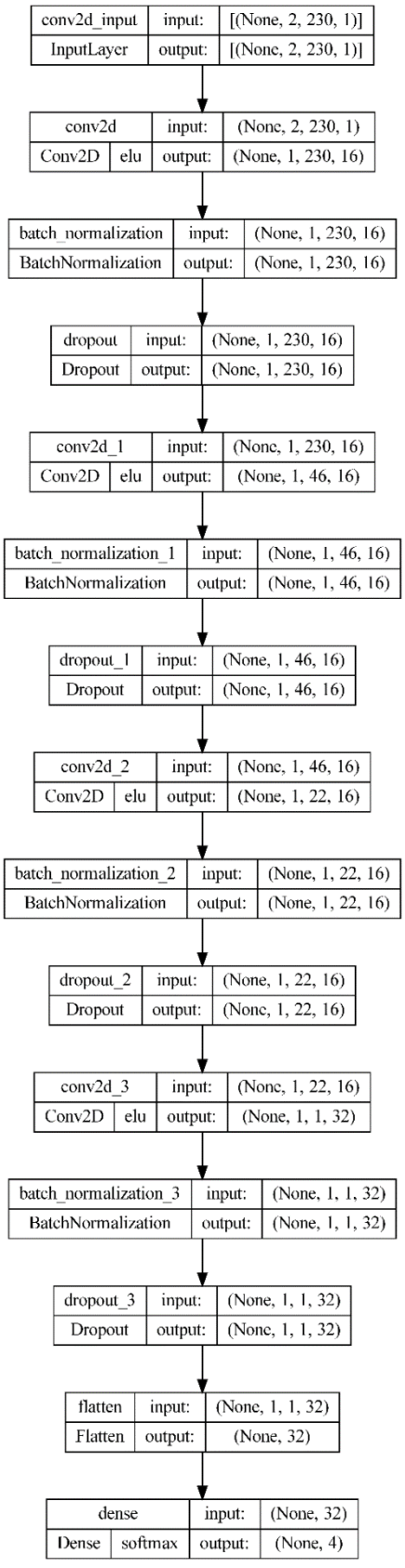


Figure 34 - forward command

After the data has undergone the preprocessing pipeline, involving noise elimination using the Common Average Reference (CAR) method, signal filtration using a Butterworth bandpass filter, and feature extraction using the Welch method, the signals are now ready for the training phase.



8.1.1 TCNN (Time-domain based CNN)

The TCNN architecture is designed to classify SSVEP signals by considering both spatial and temporal information. To accomplish this, the network uses a 1.8 second time-window of 2 electrode channels sampled at 128 Hz. The input data is made up of 2x230 dimensions, where 2 represents the spatial dimension of the input data, and 230 represents the time dimension.

The TCNN architecture as shown in the figure is made up of four two-dimensional convolutional layers, each with different kernel sizes. These layers are followed by a flatten layer and a fully connected layer with softmax activation for classification. The first convolutional layer has a kernel size of 2x1, with a stride of 1 and 16 output channels. The second convolutional layer has a kernel size of 1x230, a stride of 5 with “same” padding, and 16 output channels. The third convolutional layer has a kernel size of 1x25, with a stride of 1 and 16 output channels. Finally, the fourth convolutional layer has a kernel size of 1x22, with a stride of 1 without padding, and 32 output channels. The output of this layer is 32x1x1, which is then passed through a flatten layer to reduce the dimensionality of the feature maps. A fully connected layer with softmax activation is used for classification.

To prevent overfitting, the network employs L2 regularization with a rate of 0.01 and dropout with a rate of 0.4. The network is implemented using TensorFlow and the Keras API.

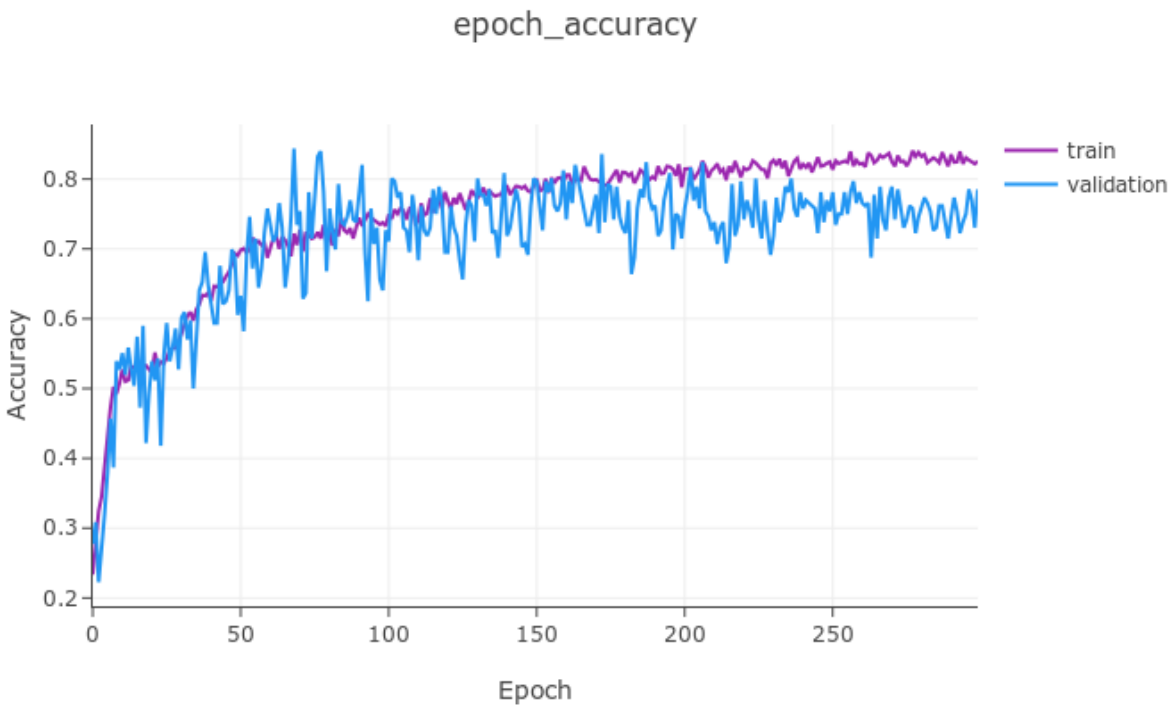
In addition to the architecture and regularization techniques, the TCNN network in the study was trained using the Adam optimizer and categorical cross-entropy loss function. The model was trained for 300 epochs using a batch size of 256.

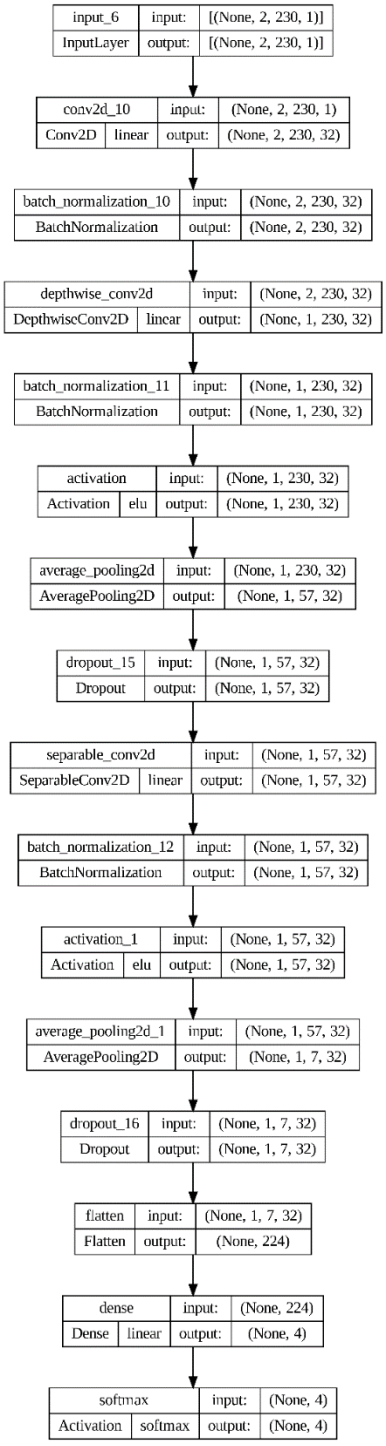
To monitor the training progress and prevent overfitting, several callbacks were used, including ModelCheckpoint, ReduceLROnPlateau, and TensorBoard. ModelCheckpoint saved the best model weights based on the validation loss, while ReduceLROnPlateau reduced the learning rate

when the validation accuracy plateaued. TensorBoard was used to visualize the training and validation accuracy and loss.

The use of callbacks and regularization techniques helped to improve the performance of the TCNN network on SSVEP classification tasks.

After training the TCNN model using the provided architecture and training process, it achieved an average accuracy of 76% on the validation dataset.





8.1.2 EEGNET

The EEGNet, also known as the Compact-CNN, is a compact convolutional neural network specifically designed for SSVEP BCI experiments, where the availability of data is limited. It efficiently represents EEG signals by employing temporal and depthwise spatial convolutions.

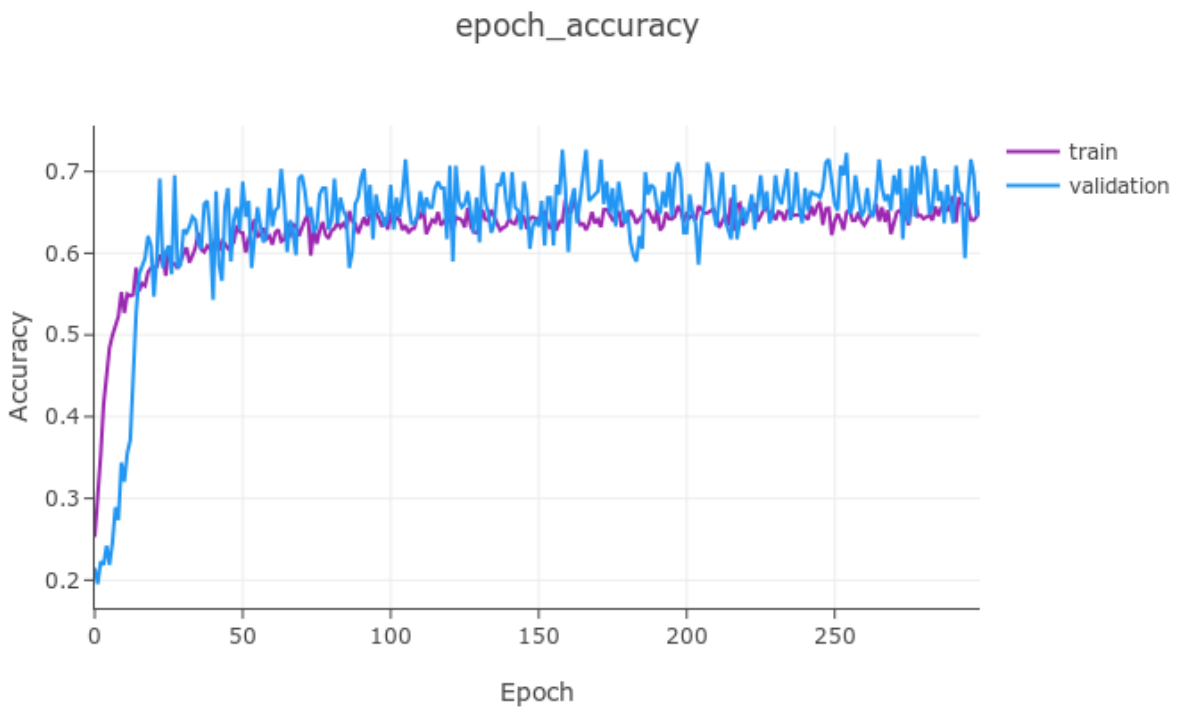
The architecture as shown in the figure starts with a temporal convolution layer that mimics a bandpass frequency filter using convolutional kernel weights derived from the data. This approach is supported by the convolution theorem. Subsequently, depthwise spatial convolutions act as spatial filters, reducing data dimensionality. These convolutions enable the network to learn spatial filters specific to different frequency bands, facilitating efficient extraction of frequency-specific spatial features. The Compact-CNN also utilizes separable convolutions, which combine information across filters while minimizing the number of trainable parameters.

Following each convolutional layer, batch normalization, 2D average pooling, and dropout layers are applied. The exponential linear unit (ELU) non-linearity, known for superior EEG classification performance, is used in the initial layers. The final layer connects to a dense layer with softmax activation for classification. The model is trained using the Adam optimizer and categorical cross-entropy loss function.

Implementation of the model is carried out using TensorFlow with the Keras API. Training consists of 500 iterations, with a minibatch size of 256. Dropout probability of 0.4 is applied to all layers. In this specific application, the model learns 96 temporal-spatial filter pairs, denoted as F1 = 32, F2 = 32, and D = 1 , indicating the number of filters in each layer.

.

After training the EEGNET model using the provided architecture and training process, it achieved an average accuracy of 64% on the validation dataset as shown in the figure.



8.2.1 CCA

CCA is an effective statistical approach for estimating the frequency of SSVEPs. This section highlights the widespread use of CCA in frequency detection of SSVEPs and its advantages compared to other methods. It references a report demonstrating that CCA-based algorithms outperform FFT-based spectrum estimation methods in terms of target identification accuracy.

The core principle of it is to analyze the correlation between a multichannel EEG signal set and a template signal set. This section explains the step-by-step process: first, calculating representative comprehensive indicators for both sets of signals, then using the correlation coefficients of these indicators to assess the overall correlation between the two signal sets. The largest correlation coefficient corresponds to the identified stimulation frequency.

The canonical correlation coefficient ρ(x; y) of CCA is defined as:

$$\rho(x, y) = max_{w_x, w_y} \frac{E\left[w_x^T X\ Y^T w_y\right]}{\sqrt{E\left[w_x^T X\ X^T w_x\right] E\left[w_y^T Y\ Y^T w_y\right]}}$$

where X and Y are input EEG signals matrices and reference signals matrices respectively, and x, y stand for the linear representation of X, Y respectively. wx and wy indicate the linear coefficient vectors.

sinusoidal signals are used as the reference signals Y to perform the classification in an unsupervised way. The reference signals Y is defined as:

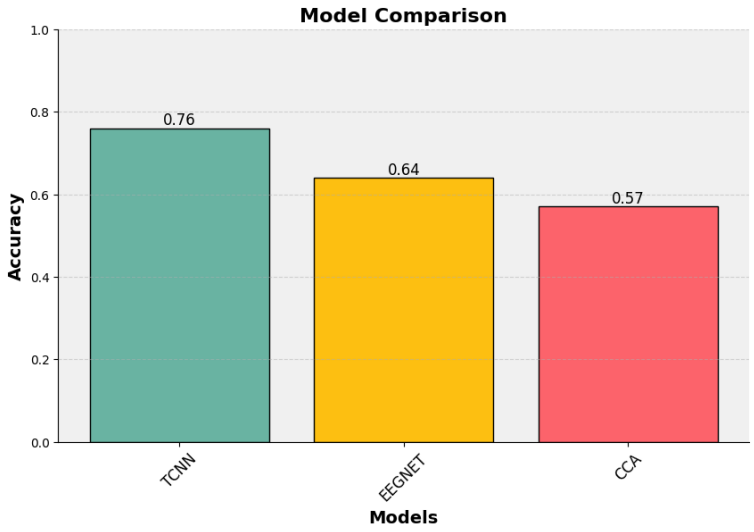
$$Y = \begin{bmatrix} \sin(2\pi ft) \\ \cos(2\pi ft) \\ \vdots \\ \sin(2\pi Nhft) \\ \cos(2\pi Nhft) \end{bmatrix}$$

where f corresponds to the target frequency of the SSVEP,and Nh is the number of harmonics.

By implementing CCA for SSVEP detection, I achieved a target identification accuracy of 57%.

Model Selecting

When it comes to model selection, it is essential to compare the accuracies achieved by different models in order to choose the best one. In our case, we have evaluated multiple models and based on the obtained accuracies, we have determined that the TCNN model performs the best, yielding an accuracy of 76%.



8.2.2 Data augmentation

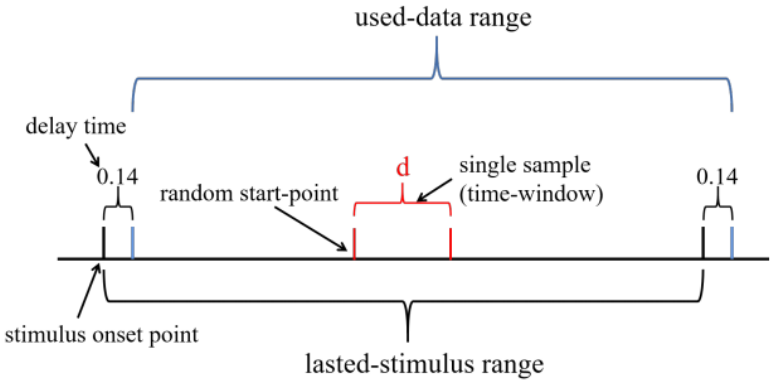
The aim is to enhance the training dataset by generating additional samples using a fixed-length time-window. The information provided suggests the following approach:

Consider a latency (delay) in the visual system. This means that there is a delay between the onset time of the stimulus and the actual response recorded in the SSVEP data.

The data range used in each trial is [0.14, 0.14+L] seconds. Here, 0.14 seconds represents the time after the onset of the stimulus, and L represents the duration of the stimulus.

During the training process, a fixed-length time-window is randomly selected as a single sample for training. This time-window is represented by [0.14+r, 0.14+r+d] seconds.

The starting point of the time-window is 0.14+r seconds after the onset of the stimulus. The value of r is a random number between 0 and L-d, where L represents the duration of the stimulation, and d represents the data length of the time-window.



Indoor Autonomous Navigation Using ROS

9. Introduction

In this Section, we will explain how we implemented the **Indoor Autonomous Navigation**. First, the patient's house is scanned, a complete grid-based map is created and saved, then all final destinations are defined on the map through coordinates. The desired rooms/final goals location are mapped to i.e. letters A, B, C, and D which are sent to the ROS system. Then the wheelchair can autonomously create a path from the start to the destination indoor location and taking into consideration the safety features that are used in this mode to **avoid static or dynamic obstacles**. While ROS System is run in this mode, the user is asked to the next destination.

We did this part Using ROS, first of all, let’s explain and define what is ROS and its using in this section of documentation in the Autonomous Navigation part.

10. What and Why ROS?

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what we need for our robotics project. And it's all open source.

ROS is originally developed in 2007 at the Stanford Artificial Intelligence Laboratory. Since 2013 managed by OSRF. Today used by many robots, universities, and companies. De facto standard for robot programming.

The required effort to develop any robotic application can be daunting, as it must contain a deeply layered structure, starting from driver-level software and continuing up through perception, abstract reasoning, and beyond. Robotic software architectures must also support large-scale software integration efforts.

Therefore, usually, roboticists end up spending excessive time with engineering solutions for their particular hardware setup. In the past, many robotic researchers solved some of those issues by presenting a wide variety of frameworks to manage complexity and facilitate rapid prototyping of software for experiments, thus resulting in the many robotic software systems currently used in academia and industry. Those frameworks were designed in response to perceived weaknesses of available middleware, or to emphasize aspects that were seen as most important in the design process. ROS is the product of trade-offs and prioritizations made during this process.

The major goals of ROS are **hardware abstraction,** low-level device control, implementation of commonly-used functionalities, message-passing between processes and package management. ROS promotes code reuse with different hardware by providing a large number of libraries available for the community. Regular updates and broad community support enable the users to obtain, build, write, test, and run ROS code, even across multiple computers, given its ability to run distributed processors.

Additionally, since it is highly flexible, with a simple and intuitive architecture, ROS allows the **reusing of code** from numerous other open-source projects. As a result, integrating robots and sensors in ROS is highly beneficial.

An obvious example of reusing code is the ATEKS model existing in ROS which we made some changes to it so we can use it in our project easily, we fit it to the measurements of wheels, height, dimensions of the base, and so on…

Figure 35 - Ateks software system written entirely in ROS

11. ROS Definitions

11.1 ROS Master

• Manages the communication between nodes (processes).

• Every node register at startup with the master.

• Start a master with:

- roscore

11.2 ROS Nodes

- Single-purpose, executable program.
- Individually compiled, executed, and managed.
- Organized in packages.
- Run a node with:
  - rosrun package_name node_name
- See active nodes with:
  - rosnode list
- Retrieve information about a node with:
  - rosnode info node_name

11.3 ROS Topics

- Nodes communicate over topics.
- Nodes can publish or subscribe to a topic.
- Typically, 1 publisher and n subscribers.
- Topic is a name for a stream of messages.
- List active topics with > rostopic echo/ topic name

11.4 ROS Messages

- Data structure defining the type of a topic.
- Comprised of a nested structure of integers, floats, booleans, strings, etc., and arrays of objects
- Defined in *.msg files
- See the type of a topic > rostopic type /topic
- Publish a message to a topic> rostopic pub /topic type data

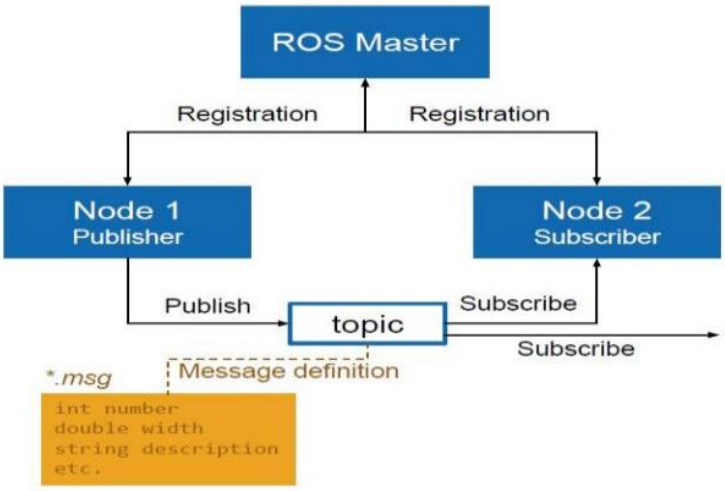


Figure 36 - ROS Definitions

11.5 Exchange Data with ROS Publishers and Subscribers

The primary mechanism for **ROS nodes** to exchange data is sending and receiving messages. **Messages** are transmitted on a **topic**, and each topic has a unique name in the ROS network. If a node wants to share information, it uses a publisher to send data to a topic. A node that wants to receive that information uses a subscriber to that same topic. Besides its unique name, each topic also has a **message type**, which determines the types of messages that are capable of being transmitted under that topic.

This publisher and subscriber communication has the following characteristics:

- Topics are used for many-to-many communication. Many publishers can send messages to the same topic and many subscribers can receive them.
- Publishers and subscribers are decoupled through topics and can be created and destroyed in any order. A message can be published on a topic even if there are no active subscribers.

Figure 37 - Exchange Data with ROS Publishers and Subscribers

12 ROS Workspace Environment

• Workspace is a set of directories (or folders) where we store related pieces of ROS code. The official name for workspaces in ROS is catkin workspaces

• The catkin workspace contains the following spaces:

- src folder: The source space contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.
- build folder: The build space is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.
- devel folder: The development (devel) space is where built targets are placed (before being installed).

• Steps to create work_space:

- Mkdir ~/catkin_ws
- Initialize the work_space using catkin_init
- To build a package we use: catkin build package_name
- Whenever we build a new package, we update the environment using: source devel/setup.bash

• After adding all packages, we will use, we compile and build them using: catkin_make.

12.1 ROS Launch

• roslaunch is a tool for easily launching multiple ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server. It includes options to automatically respawn processes that have already died.

• roslaunch takes in one or more XML configuration files (with the .launch extension) that specify the parameters to set and nodes to launch, as well as the machines that they should be run on.

• If not yet running, launch automatically starts a roscore.

• To launch a file, we use

- roslaunch package_name file_name.launch

12.2 Gazebo (Simulation Tool)

Gazebo provides the interface with ROS. It brings a fresh approach to simulation with a complete toolbox of development libraries and cloud services to make simulation easy. Iterate fast on your new physical designs in realistic environments with high-fidelity sensor streams. Test control strategies in safety, and take advantage of simulation in continuous integration tests.

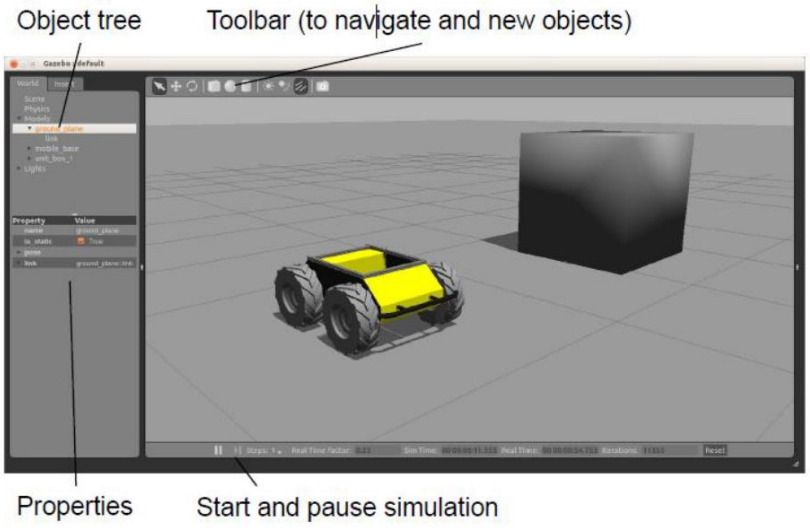


Figure 38 - Gazebo Simulator

- Simulate 3d rigid-body dynamics.
- Simulate a variety of sensors including noise.
- 3d visualization and user interaction.
- Extensible with plugins.
- To run Gazebo, use:
  - rosrun gazebo_ros gazebo
- Includes a database of many robots and environments (Gazebo worlds):

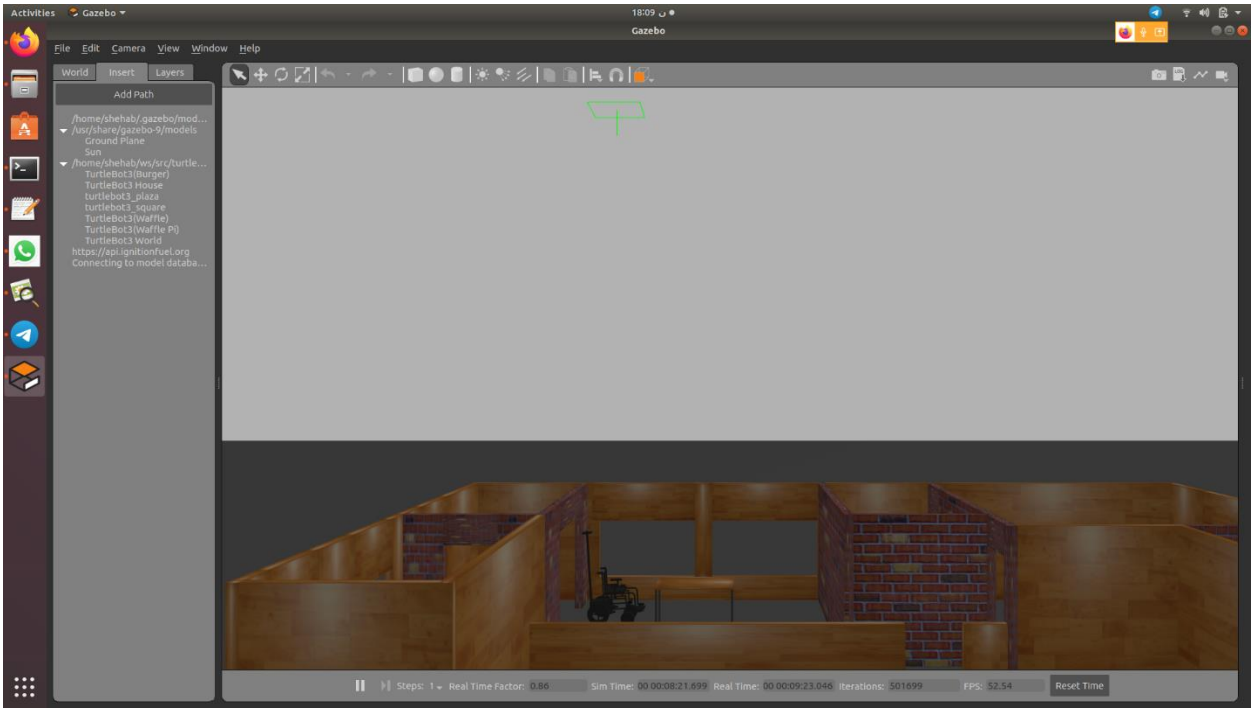


Figure 39 - Example of Gazebo worlds (house for our wheelchair simulation)

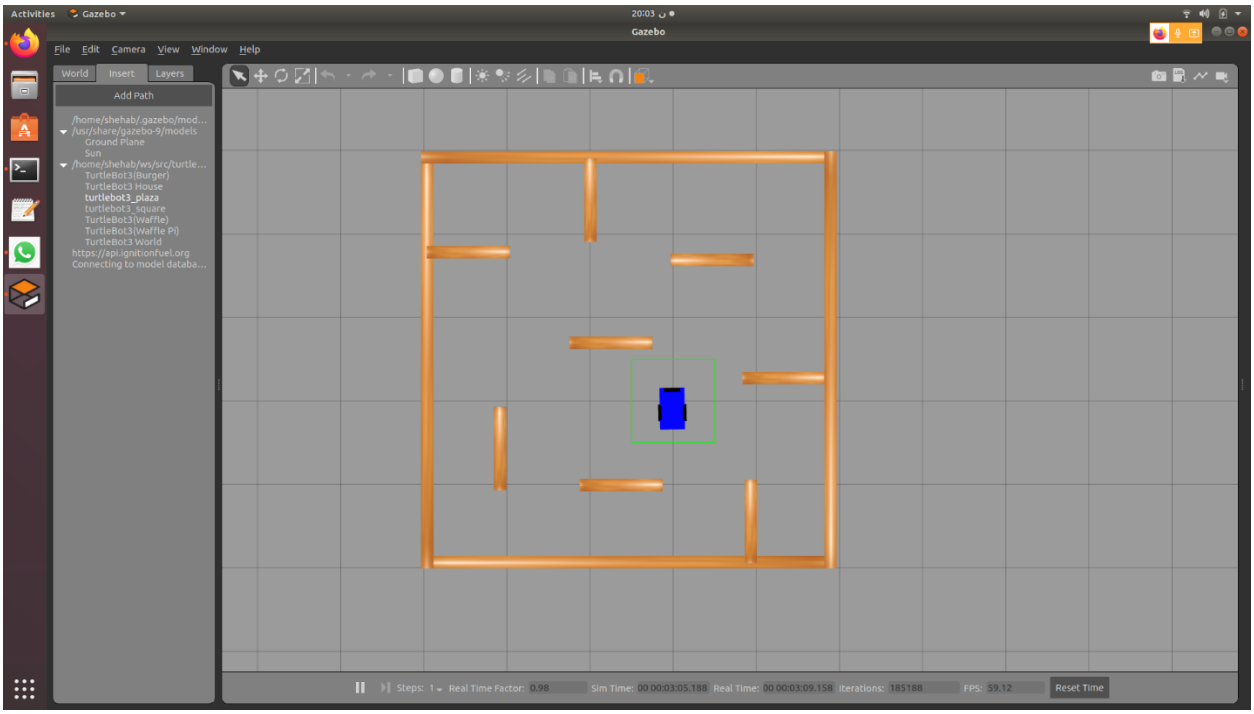


Figure 40- Example of Gazebo worlds (plaza for our prototype simulation)

12.3 Rviz (The Primary Visualizer in ROS)

Rviz is the primary visualizer in ROS and an incredibly useful tool for debugging robotics. It is a 3D visualization tool. It can subscribe to topics and visualize the message contents. It has different camera views (orthographic, top-down, etc.). RViz supplies interactive tools to publish useful information. Using it, you can save and load setup as RViz configuration easily. It is also extensible with plugins.

To run RViz we use:

- rosrun rviz rviz

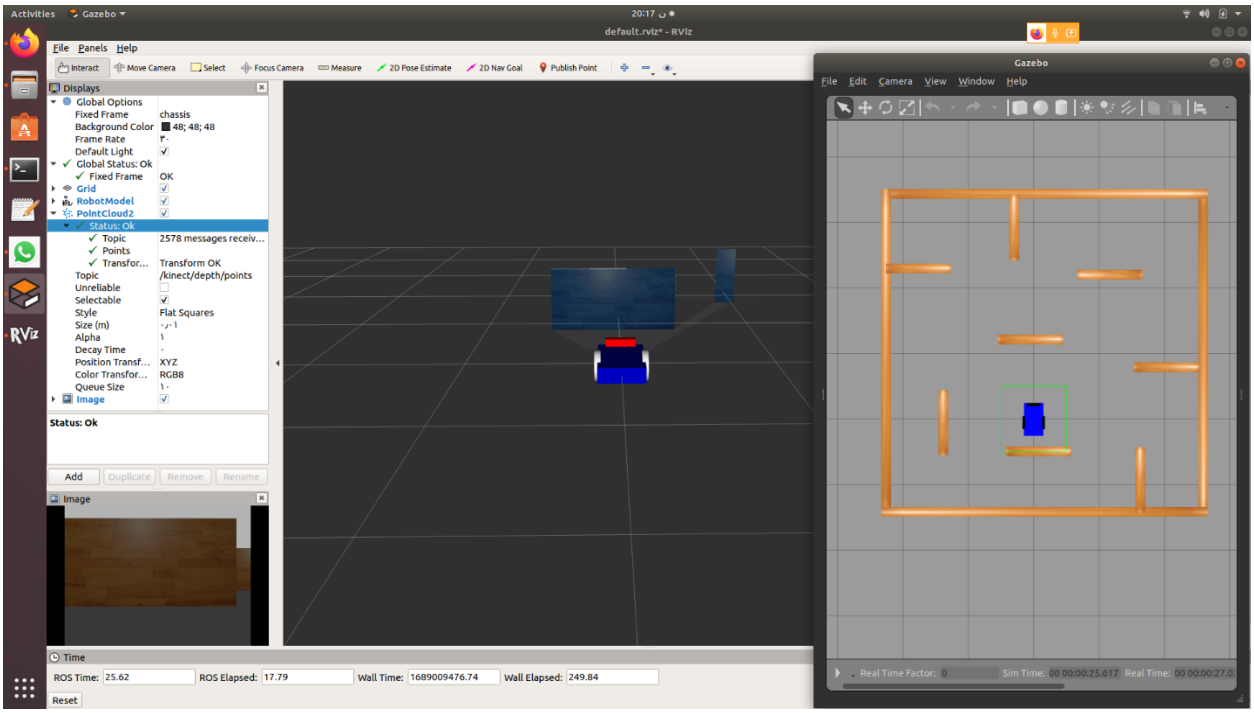


Figure 41 - RViz, the prototype visualizes what it sees in the Gazebo environment

12.4 ROS package structure

- ROS software is organized into packages, which can contain source code, launch files, configuration files, message definitions, data, and documentation.
- A package that builds up on/requires other packages (e.g.message definitions), declares these as dependencies.
- To create a new package, use:
  - catkin_create_pkg package_name {dependencies}
- The package.xml file defines:
  - properties of the package
  - Package name
  - Version number
  - Authors
  - Dependencies on other packages
- The CMakeLists.txt is input to the CMake build system
  - Required CMake Version (cmake_minimum_required)

- Package Name (project())
- Configure C++ standard and compile features
- Find other CMake/Catkin packages needed for build (find_package())
- Message/Service/Action Generators (add_message_files(), add_service_files(), add_action_files())
- Invoke message/service/action generation (generate_messages())
- Specify package build info export (catkin_package()) Libraries /Executables to build (add_library()/add_executable() /target_link_libraries())
- Tests to build (catkin_add_gtest())
- Install rules (install())

Figure 42 - package structure

12.5 Unified Robot Description Format (URDF)

The URDF file is used to write the description of our wheelchair. URDF generation can be scripted with XACRO. It defines an XML format for representing:

- Kinematic and dynamic description
- Visual representation
- Collision model

The description consists of a set of **link elements** and a set of **joint elements** where joints connect the links as shown.

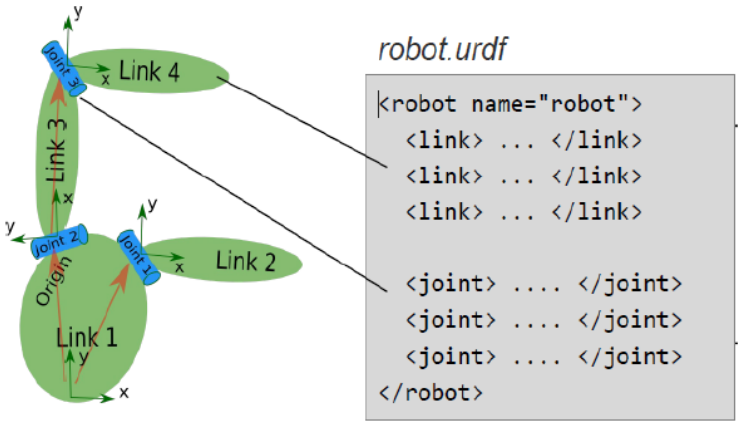


Figure 43 - Robot description in urdf file representing joints & links

For instance, let's have a look at how the Kinect camera is defined in the URDF file of the wheelchair:

```
<!-- Camera Port -->
 <link name="kinect">

<visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
        <mesh filename="package://slam/meshes/kinect.dae"/>
    </geometry>
    <material name="red"/>
</visual>

<collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
        <box size="0.1 0.1 0.1"/>
    </geometry>
</collision>

<inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
    </inertial>
</link>
```

Figure 44 - The Kinect camera defined in the URDF file

- It defines the position and orientation of the laser regarding the base of the wheelchair.
- It defines inertia-related values
- It defines collision-related values. These values will provide the real physics of the laser.
- It defines visual values. These values will provide the visual elements of the laser.

12.6 Simulation Description Format (SDF)

- Defines an XML format to describe:
  - Environments (lighting, gravity, etc.)
  - Objects (static and dynamic)
  - Sensors
  - Robots
- SDF is the standard format for Gazebo
- Gazebo converts a URDF to SDF automatically

Our prototype description:

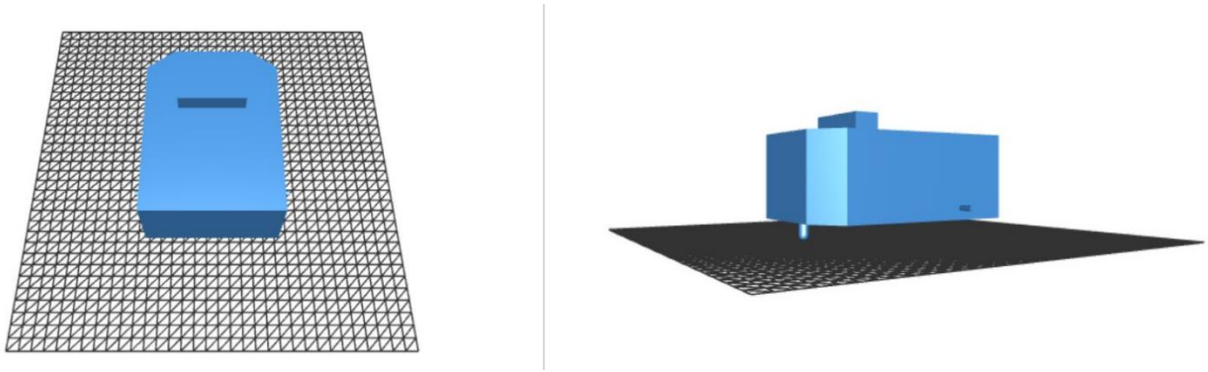
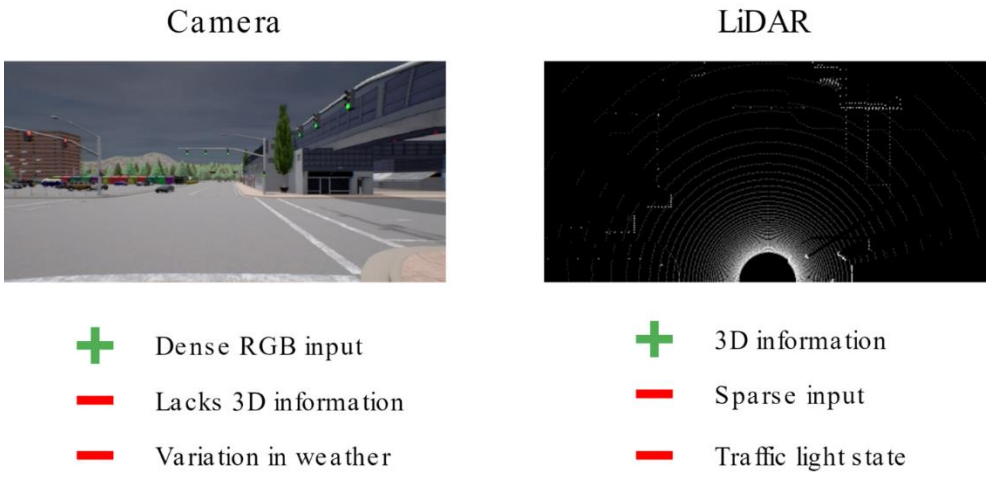


Figure 45 - Our prototype chassis stl file (SolidWorks output) to be included in the URDF file

12.7 Lidar Sensors vs Cameras

Getting depth maps and point clouds from lidar sensors is a lot easier than this as they can be directly computed from point clouds. However, Lidar sensors are extremely expensive compared to cameras. Also, they can be too accurate for a lot of applications as they can get many data points in an extremely small amount of time which requires enormous processing power. That is why in most applications stereo cameras can be more helpful.



12.8 RQt tool

RQt is a graphical user interface framework that implements various tools and interfaces in the form of plugins. One can run all the existing GUI tools as dockable windows within RQt! The tools can still run in a traditional standalone method, but RQt makes it easier to manage all the various windows in a single-screen layout.

We can run any RQt tools/plugins easily by

- rosrun rqt-graph rqt-graph

With the above command, we can get all our nodes and the topics they publish/subscribe to.

For example, our rqt graph for all wheelchair nodes run is shown below_saved as svg file:

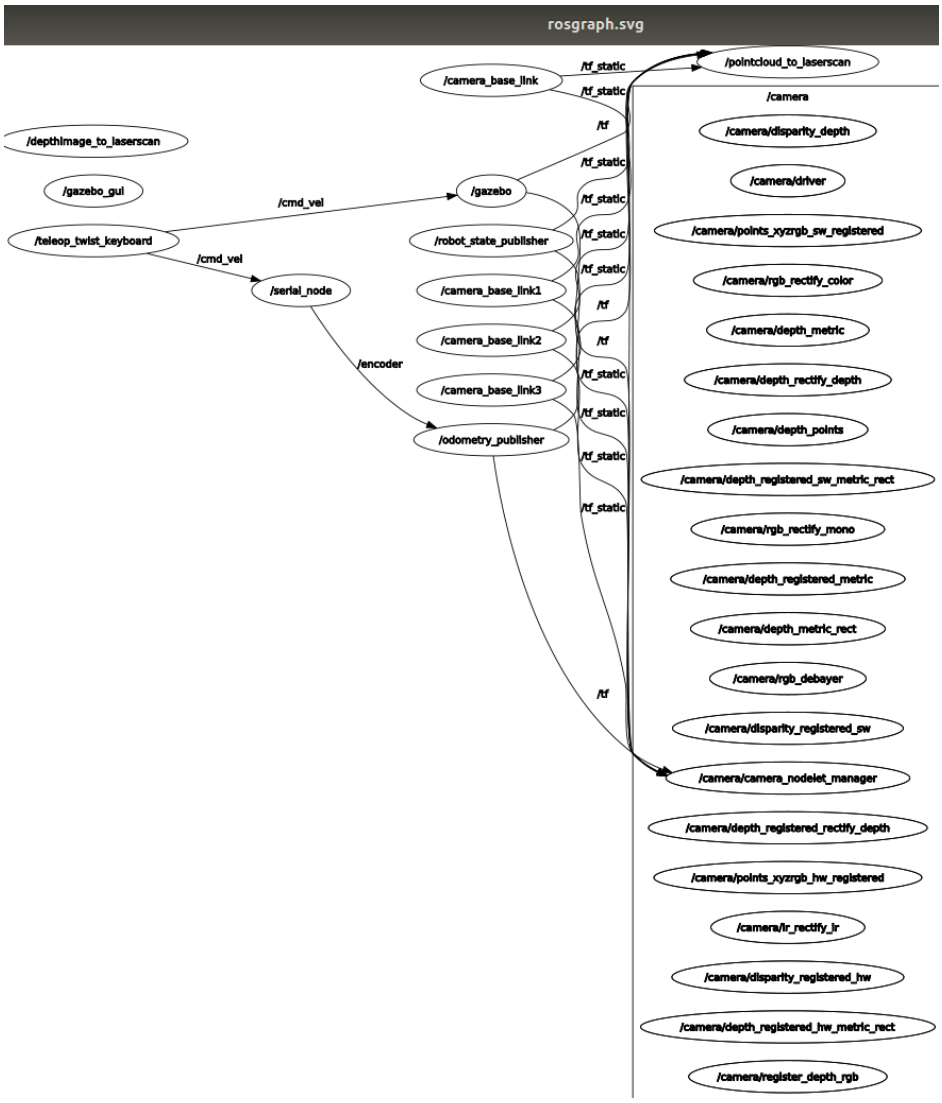


Figure 46 - Our wheelchair rqt graph of all nodes and topics

- rqt is a Qt-based framework for GUI development for ROS. It consists of three parts/meta packages:
    - rqt (you're here).
    - rqt_common_plugins - ROS backend tools suite that can be used on/off of robot runtime.
    - rqt_robot_plugins - Tools for interacting with robots during their runtime.
- There are some graphical tools available to support the launch functionalities of ROS:
    - rqt_launch (very experimental) can run launch files.
    - rqt_launchtree allows you to interactively introspect launch files.
    - roslaunch_to_dot converts a launch file tree to a graph and saves it into a dot file.

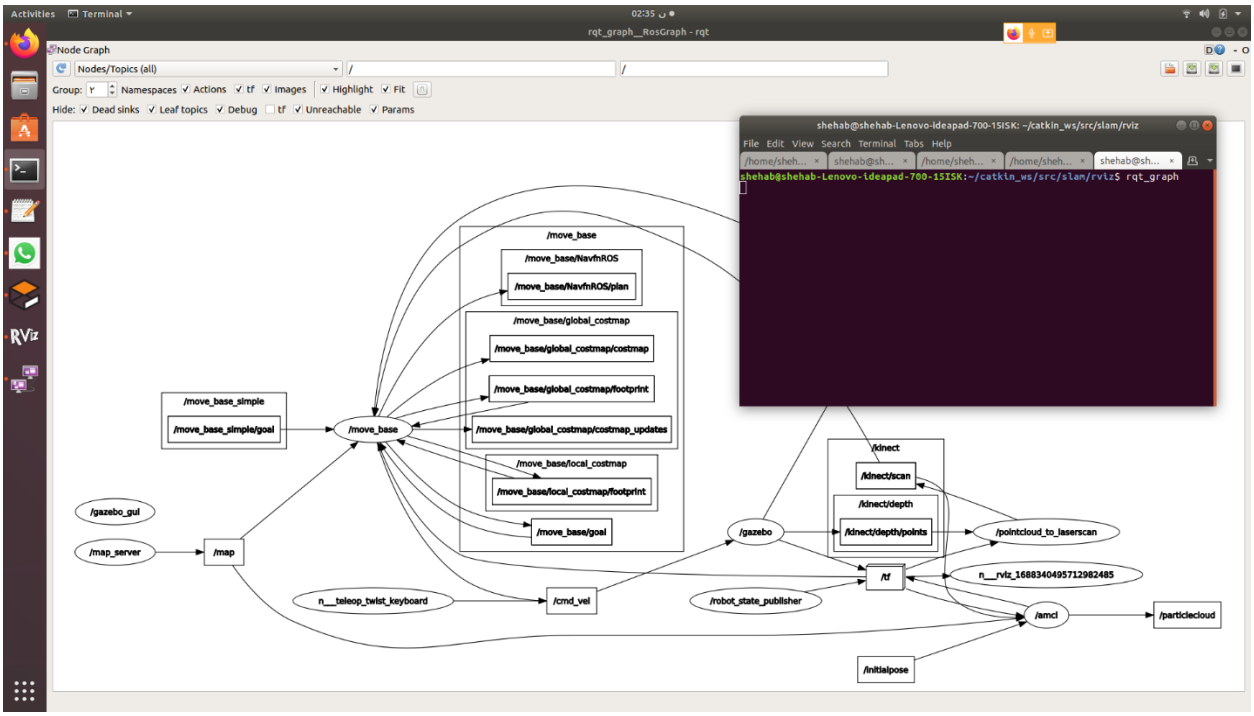


Figure 47 - Our overall rqt graph after running gmapping pkg

13. Our Wheelchair Packages

After explaining ROS as the software we used to implement our project and all its parts we will use, let's go through our project implementation in this part. Our project consists of four packages containing some nodes written in C++ and depends on some libraries. These nodes are subscribers or publishers. The node can subscribe to a topic also can publish on another topic at the same time as explained in the previous section. Some packages consist of some launch files to run multiple nodes at one time and load some tuning parameters.

The packages we implemented are named:

- slam package: wheelchair description URDF, simulation Gazebo, visualization RViz, and mapping (gmapping launch file).
- move_robot package: sending goals to the ROS navigation Stack wheelchair navigation AMCL and Navigation parameters.
- odom_pub package: taking odometry information from encoder ticks on Arduino.
- teleop_twist_keyboard package: keyboard control of the wheelchair
- Arduino node

Now we will talk about each package in detail…

13.1 Robot Description

This step is so essential in our process of implementing the indoor navigation system. The entry point for the system is to model the robot. So, we aim to implement a package - slam/urdf & slam/meshes - used to describe the wheelchair joints, links, and geometry.

Here we list the main functionality of this package:

- This package contains the urdf file which describes the wheelchair to represent a visual representation for it and we will use it in all packages if we want to load its model in simulation tools such as in Gazebo or RViz to visualize motion or in the real world motion step.
- Contains the meshes we used in the wheelchair building and included in the urdf file such as
    - Ateks software system written entirely in ROS.
    - The Kinect camera is written entirely in ROS.

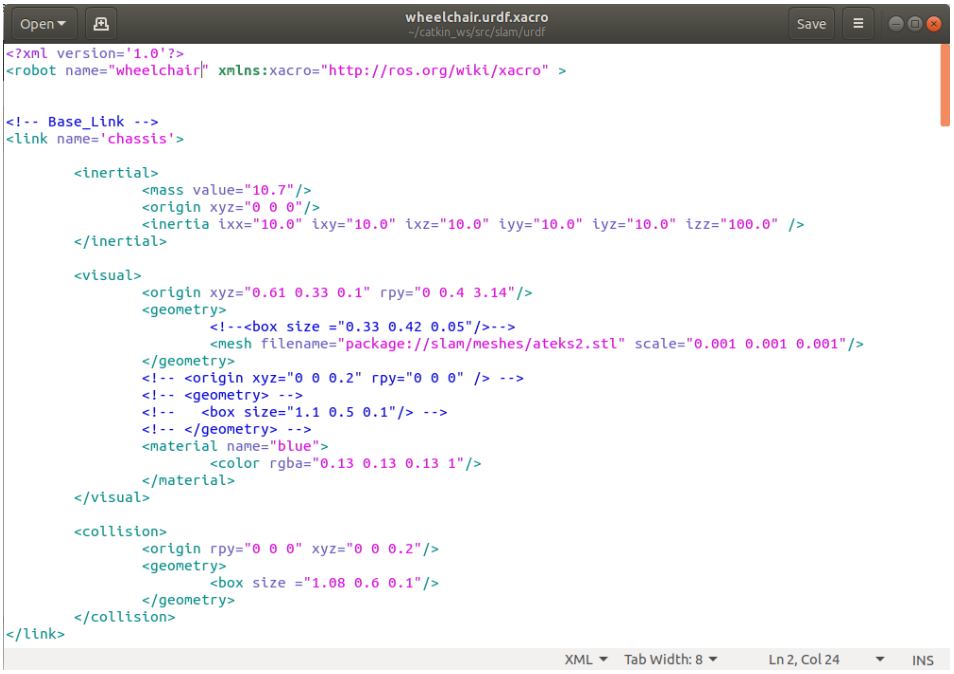


Figure 48 - The 'chassis' part of our wheelchair description

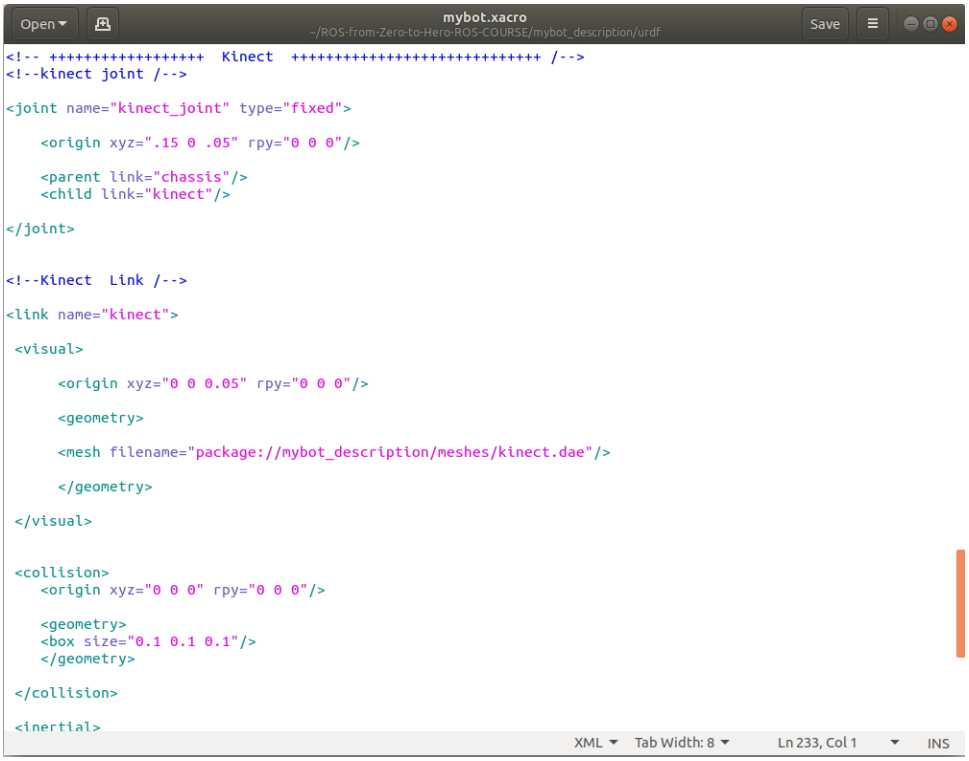


Figure 49 - The 'Kinect' part of our prototype description, included as a mesh file

13.2 Robot Simulation

The second step in our process is to make a simulation for the wheelchair model we implemented before to make sure that all things are ok so we implemented the package -slam/worlds- to do this functionality.

Using this package, we managed to define the best place to put the Kinect. Unfortunately, we only have one Kinect camera so the accuracy of the navigation process won't be the best, but as best as possible, and after many trials, we found the best place. Here are some examples of what we mean…

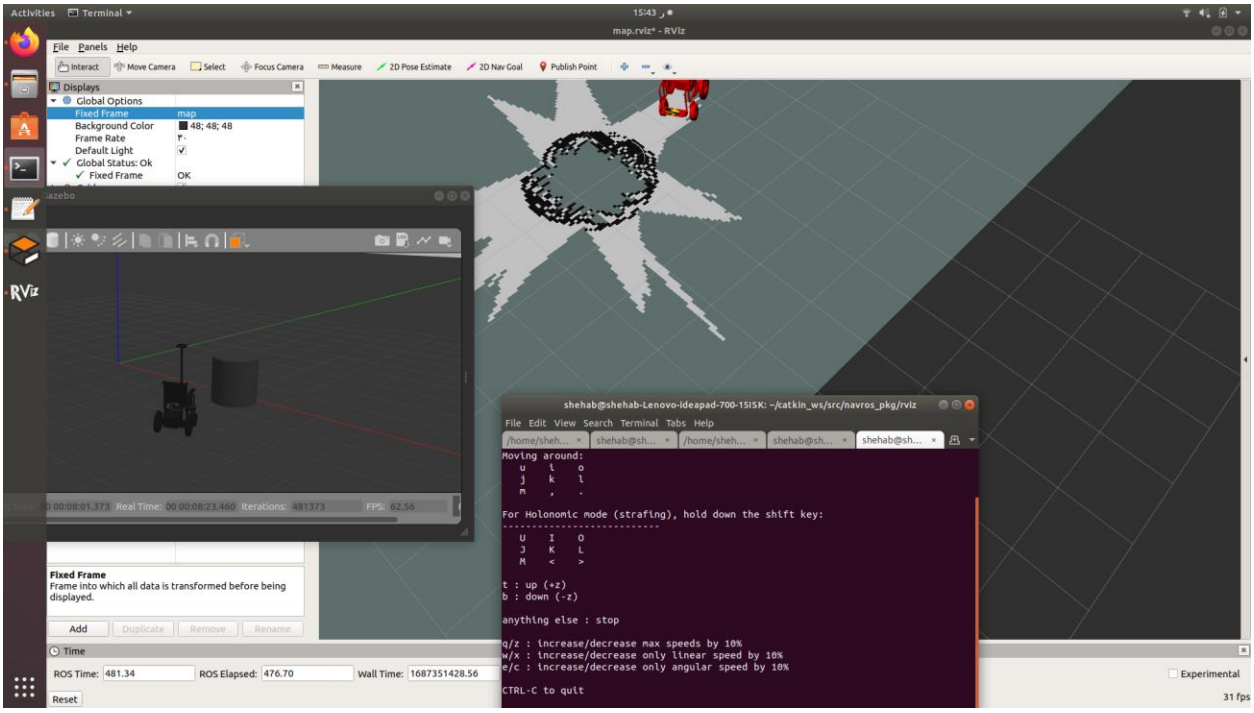


Figure 50 - Kinect centered at the foot base creating a wrong map; it sees the obstacle at a different place

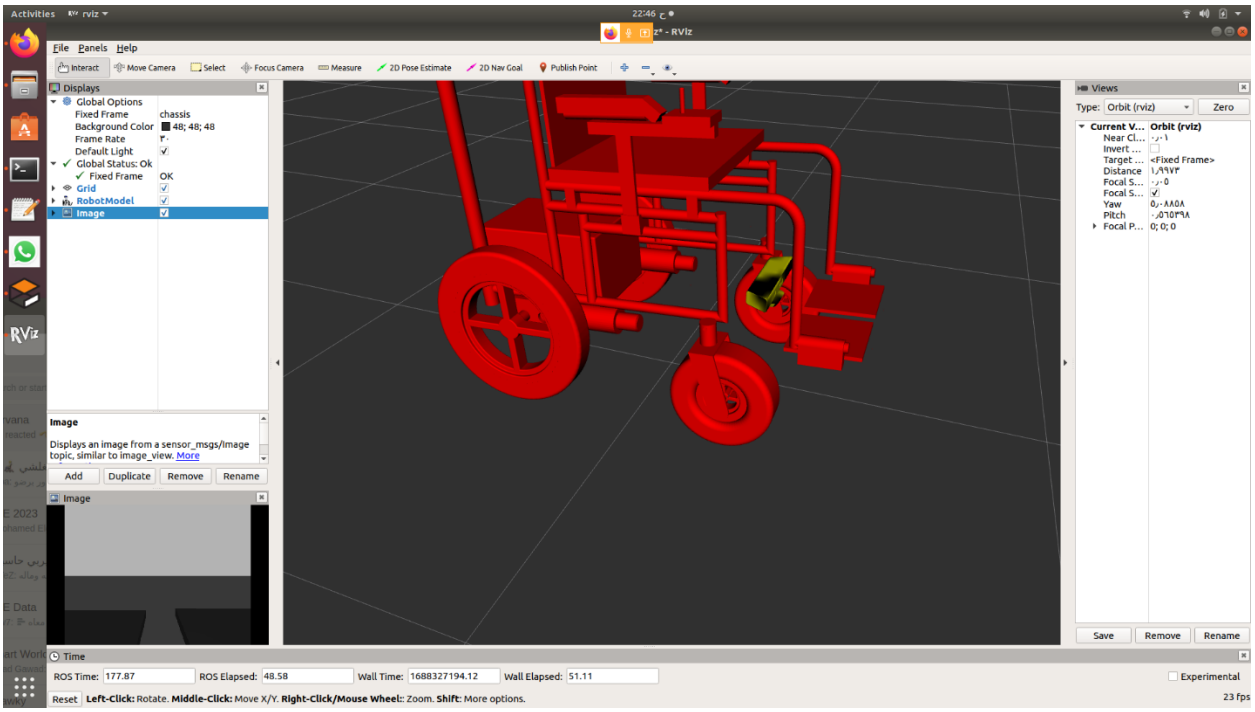
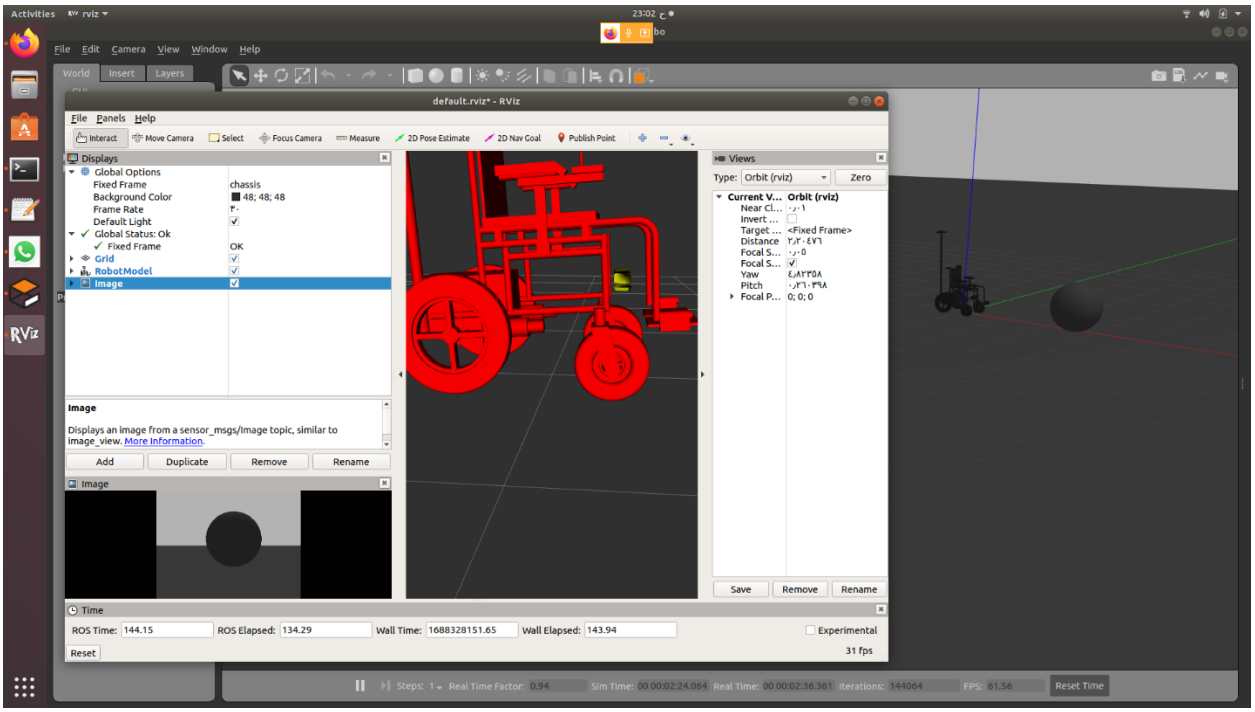


Figure 51 - The Kinect sees a part of the map as an obstacle all the time



Place of the map gives bet results; sees the ball in the correct place, creates a good map, and navigates well

Here is the main functionality of this package:

- This package is used to run the wheelchair in the fake world to make a simulation of the world in which it will work in.
- It contains the wheelchair world file such as rooms, obstacles, lightning, and so on.
- It contains the launch file which loads the wheelchair urdf file and the world file and combines them so we can simulate the real-world case.
- We can control the wheelchair using the (teleop_twist_keyboard.py) node.

Figure 52 - Example of the 'Empty World' implementation of Gazebo

13.3 Robot Odometry Calculations

This is a main important step in our navigation process which is responsible to calculate the robot position referenced to the odom frame to be able to set the wheelchair velocity. We implemented the package - odom_pub - to do this step.

Here is the main functionality of this package:

- This package is used to calculate the transformation between links and set the odometry calculations depending on the encoder's values.
- This package contains a C++ node (odom_pub.cpp) to do the calculations using equations that calculate the distance which the robot do through and calculate the rotation angle as shown later in this chapter.
- From the distance, we calculate the distance in **X and Y directions** and the **Yaw angle** to create the **Quaternion Calculations.**
- We calculate the distance from the encoder's ticks.
- This node is subscribing on **the topic "/encoder"**
- The "/encoder" topic is implemented in the Arduino node and publishes its values for Right wheel ticks and Left wheel ticks.
- Equations to calculate distance and theta:
    - Distance and theta equation:
        - Dc=((2*3.14*R*(RtickNew-RtickOld)/N)+((2*3.14*R*(LtickNew-LtickOld)/N)))/2
        - Th+=((2*3.14*R*(RtickNew-RtickOld)/N)-((2*3.14*R*(LtickNew-LtickOld)/N)))/L
    - X, Y calculations:
        - X += Dc * cos(th)
        - Y += Dc * sin(th)

- Then we do Odom calculations to set the velocity values:
    - Time calculations:
        - dt = (current_time - last_time).toSec();
    - Velocity calculations in X, Y directions and Yaw rotation:
        - vx = Dc * cos(th)/dt;
        - vy = Dc * sin(th)/dt;
        - vth=(((2*3.14*R*(RtickNew-RtickOld)/N)-((2*3.14*R*(LtickNew-LtickOld)/N)))/L)/dt;

- After doing these calculations we set the Odom twist velocity in each direction using these equations:
    - odom.twist.twist.linear.x = vx
    - odom.twist.twist.linear.y = vy
    - odom.twist.twist.angular.z = vth

- The last step in this node is to publish the odometry calculations over a topic named "/odom"

Open ▾ odom_pub.cpp ~/catkin_ws/src/odom_pub/src Save

wheelchair.urdf.xacro × world.world × odom_pub.cpp ×

```
  ros::Time current_time, last_time;
  current_time = ros::Time::now();
  last_time = ros::Time::now();

  ros::Rate r(10.0);
//-----------------------------------While loop--------------------------------
     while(n.ok()){

    ros::spinOnce(); // check for incoming messages

    current_time = ros::Time::now();

    //---odom calculation------
    double dt = (current_time - last_time).toSec();
    vx=Dc*cos(th)/dt;
    vy=Dc*sin(th)/dt;
    vth=(((2*3.14*R*(RtickNew-RtickOld)/N)-((2*3.14*R*(LtickNew-LtickOld)/N)))/L)/dt;

    geometry_msgs::Quaternion odom_quat = tf::createQuaternionMsgFromYaw(th);

    //first,---------- TRANSFORMATION broadcasting over tf
    geometry_msgs::TransformStamped odom_trans;
    odom_trans.header.stamp = current_time;
    odom_trans.header.frame_id = "odom";
    odom_trans.child_frame_id = "chassis";

    odom_trans.transform.translation.x = x;
    odom_trans.transform.translation.y = y;
    odom_trans.transform.translation.z = 0.0;
    odom_trans.transform.rotation = odom_quat;

    //send the transform
    trans_broadcaster.sendTransform(odom_trans);
```

C++ ▾ Tab Width: 8 ▾ Ln 56, Col 86 ▾ INS

Figure 53 - Odometry calculations

13.4 Robot Navigation and Mapping

This step is an integration of the previous steps to combine them to apply our mapping algorithm to get a map of the place where the wheelchair will be. Map building is accomplished using the ROS gmapping package. This package provides an implementation of the SLAM algorithm, which both continuously builds and updates the Gazebo vision sensor (Kinect camera sensor). The resulting scan is transformed into a new coordinate frame at the center of the frames where the camera scan origins lie.

Then we start mapping (scan the environment and create the map) by using the teleop_twist_keyboard node mentioned before.

Here are the main functions of this step:

- This package is implemented to run SLAM Algorithm in two ways:
  - Simulation part
  - Real-world scenario part
- In the Simulation way, we simulate for our wheelchair to be sure that every package is **implemented correctly** and the **transformation between links is correct** and the naming is consistent. we take the **Odom calculations** from the **Gazebo** simulation tool and we create a **fake world** to run the wheelchair in this part works correctly after testing it.

- After the simulation part, we implemented the real-world scenario part and used the odom_pub node to publish the odometry calculations from the real wheelchair using encoder values.
- By connecting the hardware devices and wheelchair with the Laptop to check if working correctly. We tested this part and apply the **SLAM algorithm** “explained in detail in the next chapter” and take a real map to use it after that in autonomous navigation.
- This package contains a slam_real.launch file which runs the gmapping package used in the SLAM algorithm and runs the nodes used for camera sensors and controller nodes such as the teleop node which is used to control the robot's motion.
- Contains a slam_all.launch file which is used to run the SLAM algorithm in Simulation.
- The above launch files load also the urdf of the robot and the RViz configuration file to show the visualization of the robot's motion.

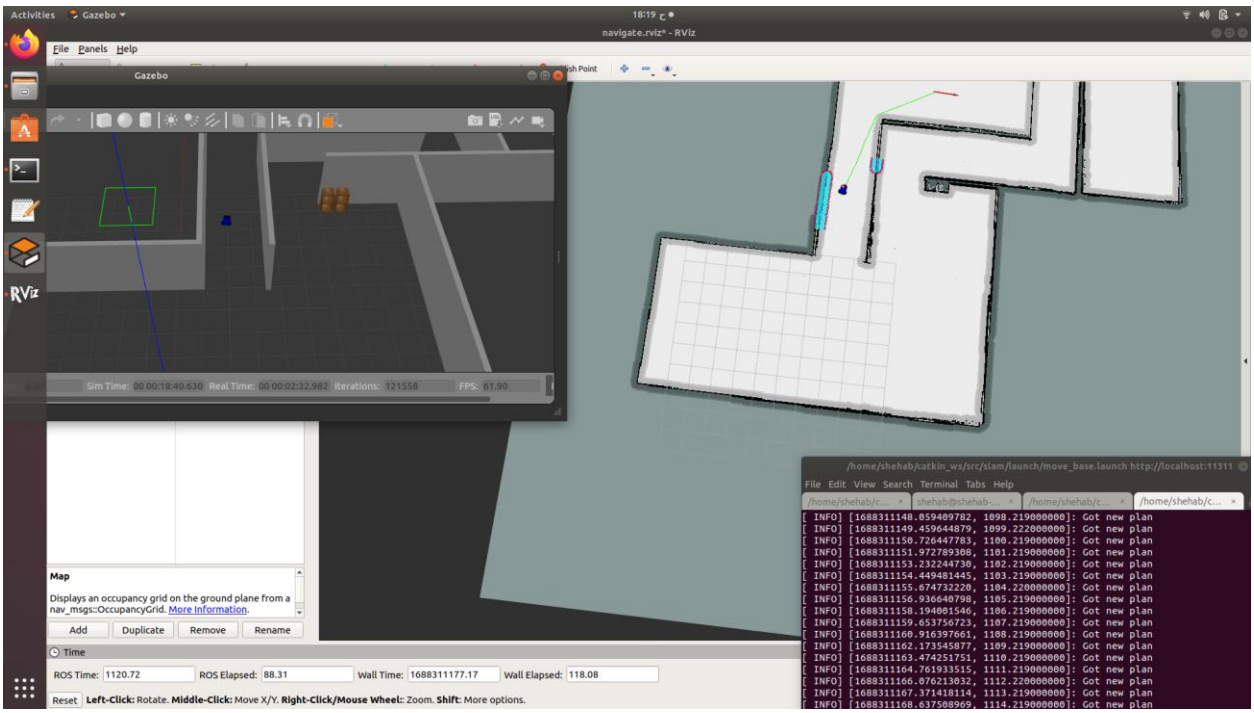


Figure 54 - Our Prototype Navigation and Mapping

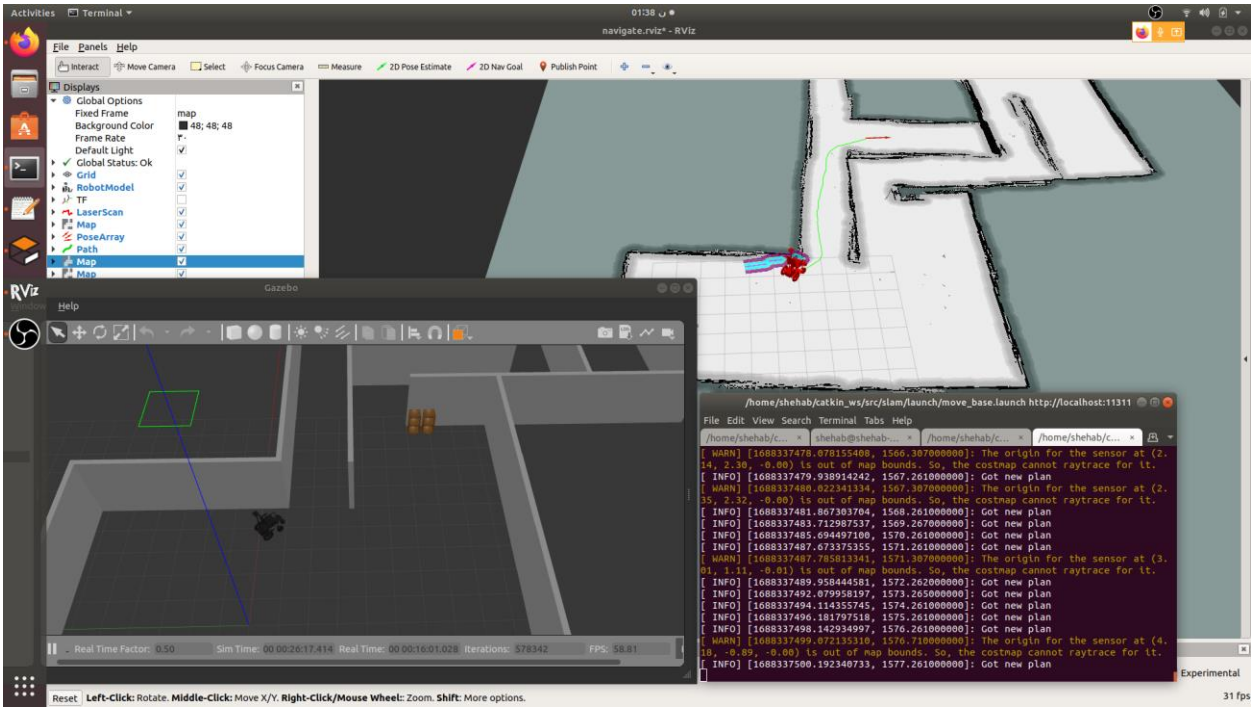


Figure 55 - Our wheelchair Navigation and Mapping

Hint: Robot Navigation and Mapping package is implemented to run SLAM Algorithm. These concepts are essential, so they must be covered well. In this part, we are talking only about our packages implementation so we split the explanation into a separate chapter.

13.5 TF Transformation

The **tf package** lets us keep track of multiple coordinate frames over time. It maintains the relationship between coordinate frames in a tree structure buffered in time and lets us transform points, vectors, etc between any two coordinate frames at any desired point in time.

To use the laser readings, we need to set a transform between the laser and the robot base and add it to the transform tree.

Transform? Transform tree? What is this mean? Ok, let's clarify all this. So we have a wheelchair with a laser mounted on it, right? But, to be able to use the laser data, we need to tell the wheelchair where (position and orientation) this laser is mounted in it. This is what is called a transformation between frames.

A transform specifies how data expressed in a frame can be transformed into a different

frame. For instance, if we detect an obstacle with the laser at 3 cm in the front, this means that it

is 3 cm from the laser, but not from the center of the wheelchair. To know the distance from the center of the wheelchair, we need to transform the 3 cm from the /laser_frame to the /base_link

frame (which is actually what the Path Planning system needs to know, what is the distance from the center to the obstacle).

If we don't provide this information to the wheelchair when the laser detects an object, how can it know where this object is? Is it in front? Is it behind? Is it to the right? There's no way to know this if we don't tell the wheelchair the position and the orientation of the laser.

With the tf package, we can:

- Monitor: Print information about the current coordinate transform tree to the console.
- rosrun tf tf_monitor
- tf_monitor <source_frame> <target_target>
- Tf_echo: Print information about a particular transformation between a source_frame and a target_frame.
- tf_echo <source_frame> <target_frame>
- View Frames: Graphical debugging tool that creates a PDF graph of your current transform tree and views it when we are done, so a typical usage on Ubuntu systems is.
- rosrun tf view_frames
- evince frames.pdf

Before starting navigation we must make sure the transformation tree of the robot is correct:

This is the transformation frames of our wheelchair:

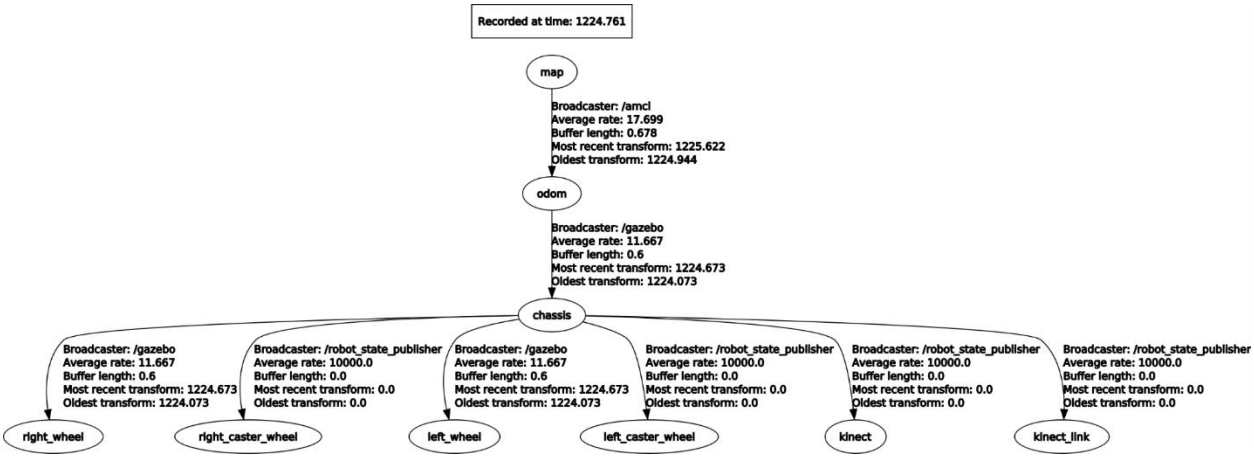


Figure 56 - The wheelchair tf frames

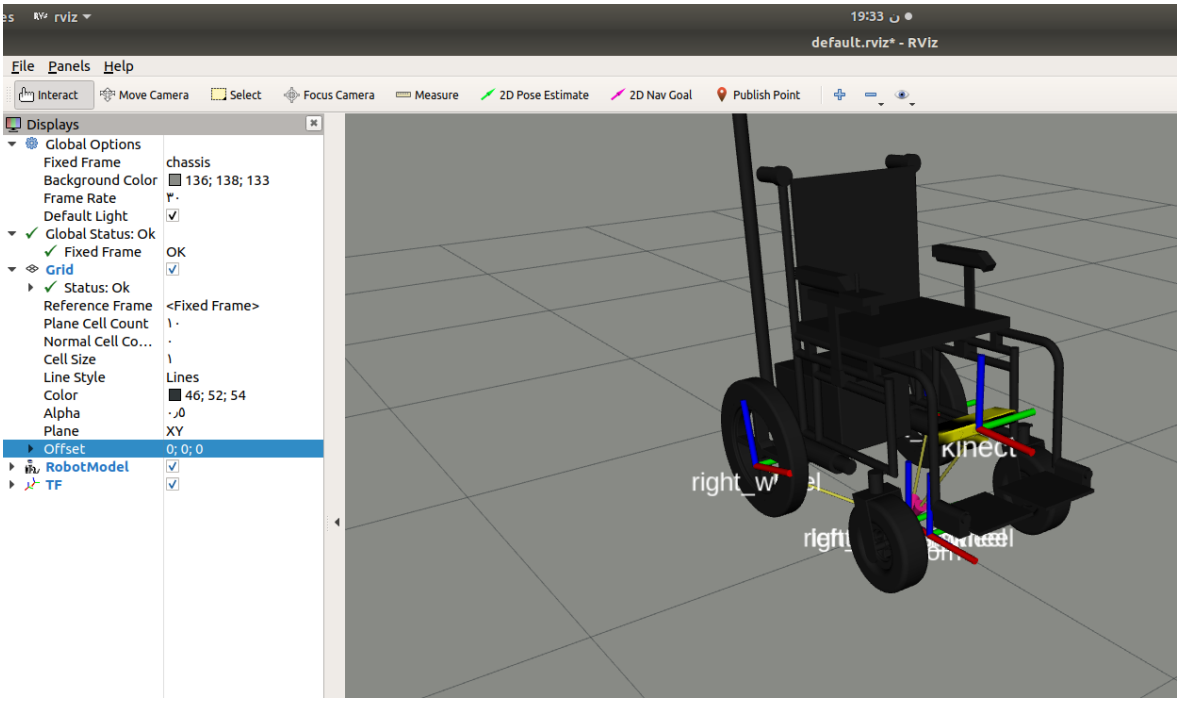


Figure 57 - The wheelchair tf frames in RViz

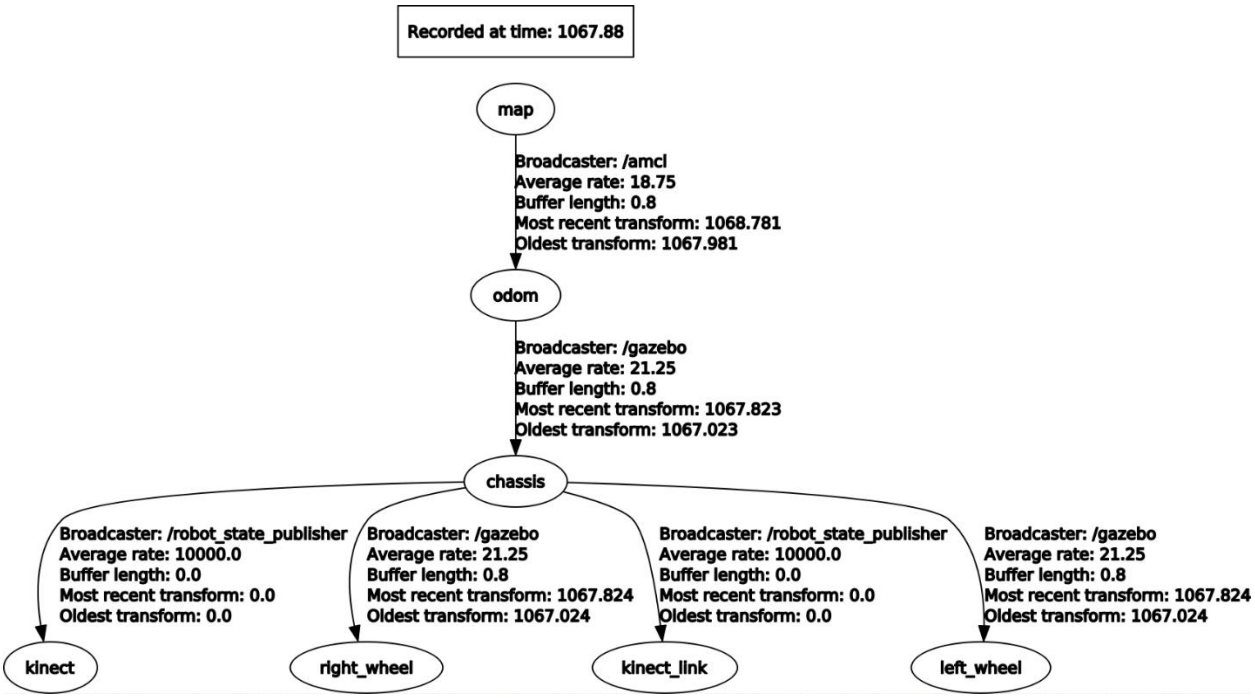


Figure 58 - The prototype tf frames

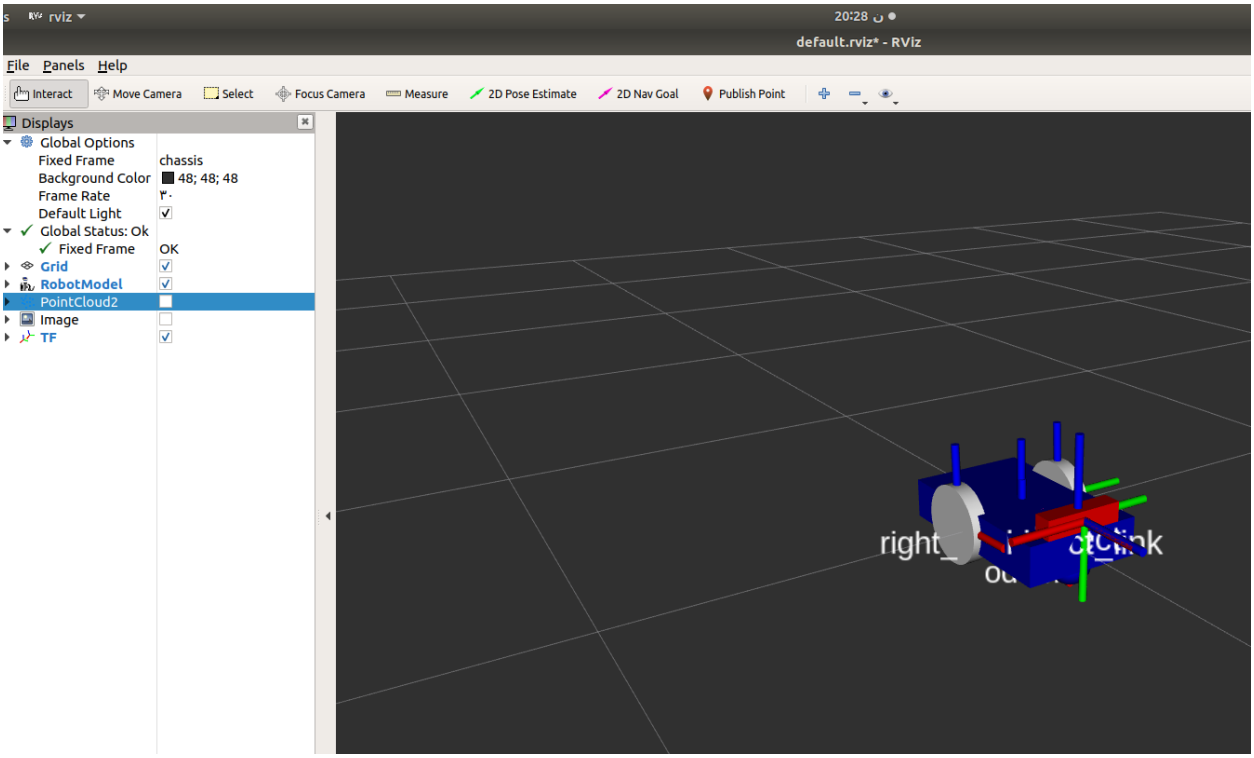


Figure 59 - The prototype tf frames in RViz

We can see our project nodes publish/subscribe to the (/tf topic) using the rqt graph explained earlier:

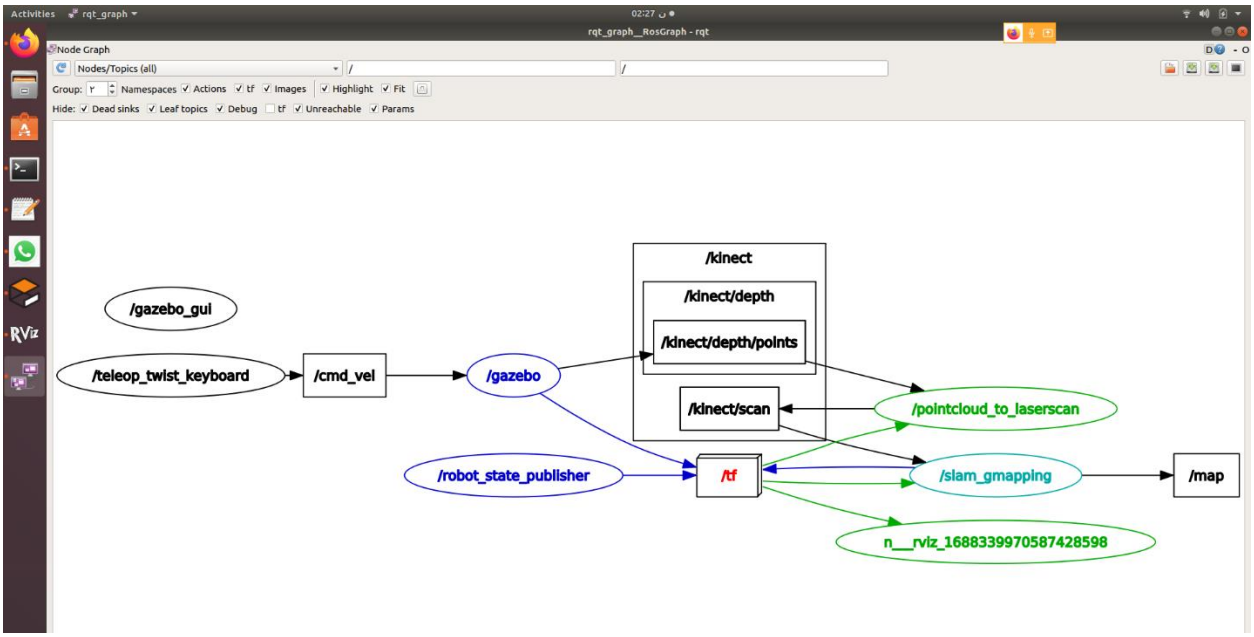


Figure 60 - rqt graph of /tf nodes

13.6 Navigation Stack Setup

- Navigation Stack setup and configuration concepts are essential, so they must be covered well. In this part, we are talking only about our packages implementation so we split the explanation into a separate chapter.
- The Navigation Stack package assumes that all the requirements above for robot setup have been satisfied. Specifically, this means that the robot must be publishing coordinate frame information using tf, receiving sensor_msgs/LaserScan or sensor_msgs/PointCloud messages from all sensors that are to be used with the navigation stack, and publishing odometry information using both tf and the nav_msgs/Odometry message while also taking in velocity commands to send to the base. After creating the package slam package, we created a folder called params which contains some parameters we will use.
- The folder 'params' contains some files, these files are:
    - Costmap Configuration (local_costmap):
      This shows the local costmap that the navigation stack uses for navigation. For the wheelchair to avoid collisions, its footprint should never intersect with a cell that contains an obstacle. In the **local costmap** is everything that can be known from the current position with the **sensors** right know. E.g. walking people and other moving objects, as well as every wall, etc. That can be seen.
    - Costmap Configuration (global_costmap):
      In the global costmap is everything the wheelchair knows from previous visits and stored knowledge (the saved map).
    - Common Configuration (local_costmap) & (global_costmap):
      This file combines the parameters of the local and global cost maps.
    - Base Local Planner Configuration:

The base_local_planner is responsible for computing velocity commands to send to the mobile base of the wheelchair given a high-level plan. We'll need to set some configuration options based on the specs of it to get things up and running.

- Dynamic-Window Approach (Dwa_local_planner):
  The dwa_local_planner package provides a controller that drives a mobile base in the plane. This controller serves to connect **the path planner** to the wheelchair. Using a map, the planner creates a **kinematic trajectory** for the wheelchair to get from a start to a goal location.

  Along the way, the planner creates, at least locally around the wheelchair, **a value function**, represented as a **grid map**. This value function encodes the costs of traversing through the grid cells. The controller's job is to use this value function to determine **dx, dy, and dtheta velocities** to send to the wheelchair.

- Adaptive Monte Carlo Localization (Amcl) configuration:
  AMCL is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.

- Move_base_configuration:
  This configuration file is responsible to bring everything and all configuration files together into a launch file for the navigation stack.

- After implementing all these files, now our wheelchair is ready to apply **autonomous navigation** using **the Navigation Stack package,** and to do this, we **run the move base file** and:
  - Set a goal on a map using the RViz visualization tool.
  - Or Run the (move_robot.cpp) node where we've already defined all the possible pre-asked goals from the user in the form of X, Y coordinates.

We can see all the wheelchair nodes publish/subscribe to all topics after running gmapping pkg using the rqt graph:

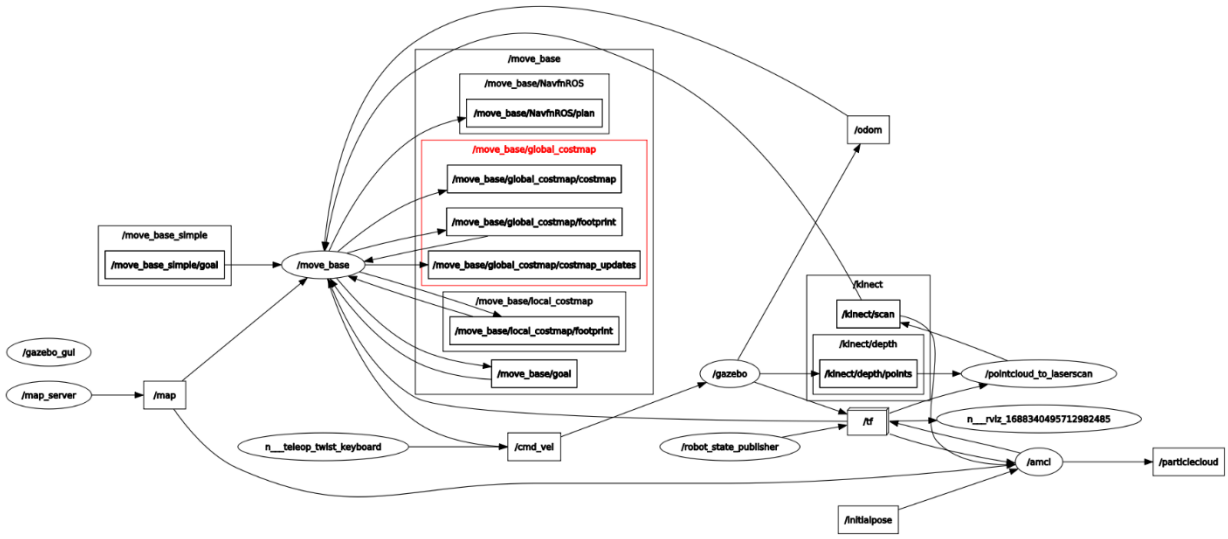


Figure 61 - rqt graph of /tf nodes

14. Autonomous Navigation Algorithms and Concepts

14.1 Basic Concepts

Navigation is to allow the wheelchair to get from one point to another without bumping into obstacles. First, we need a map**,** The very first thing we need to perform Navigation is a Map. We need a map of the environment where we want our wheelchair to navigate. So the first thing we need to do is to get a map of the environment. How do we get this map? Where do I get it from?

We use the wheelchair to create it. A map is just a representation of an environment created from the sensor readings of the robot (for example, from the Kinect camera). So just by moving the wheelchair around the environment, we can create an awesome Map of it. In terms of ROS Navigation, this is known as **Mapping**.

Next, we need to localize the robot on that map**,** we need a Map to Navigate autonomously with the wheelchair, but is it enough? As you may imagine, the answer is NO. we have a Map of the environment, yes, but this is completely useless if our wheelchair doesn't know where it is concerning this map. This means, to perform proper Navigation, it needs to know in which position of the Map it is located and with which orientation (that is, which direction the wheelchair is facing) at every moment. In terms of ROS Navigation, this is known as **Localization.**

Now we can send goal locations to the robot though we're done. Not at all! We still need a couple of things to perform ROS Navigation. For now, we've already covered the first block, which is building our map of the environment and being able to Localize the wheelchair in it. So what's next? We should start Navigating autonomously in it, right?

For this, we'll need some kind of system that tells it where to go, at first, and how to go there, at last. In ROS, we call this system the **Path Planning**. Path Planning takes as input the current location of the robot and the position where the robot wants to go and gives us as an output the best and fastest path to reach that point.

Finally, we need to avoid obstacles anyway, How does ROS manage to avoid them? What happens if someone or something suddenly walks into the path of the wheelchair? Will it know it's there? Will it be able to avoid that obstacle? All of these questions are related to the same topic, which is called **Obstacle Avoidance**.

The Obstacle Avoidance system breaks the big picture (Map) into smaller pieces, which update in real-time using the data it's getting from the sensors. This way, it assures it won't be surprised by any sudden change in the environment or any obstacle that appears in the way.

Robot Configuration is extremely important in all navigation modules. For instance, in the Mapping system, if we don't tell the system where our wheelchair has the laser mounted on, which is the position of the wheels in the robot, etc., it won't be able to create a good and accurate Map. And if we don't have a good Map in ROS Navigation, we have nothing! The following image is an example of how a Map file would look if the laser of the robot is not properly configured.

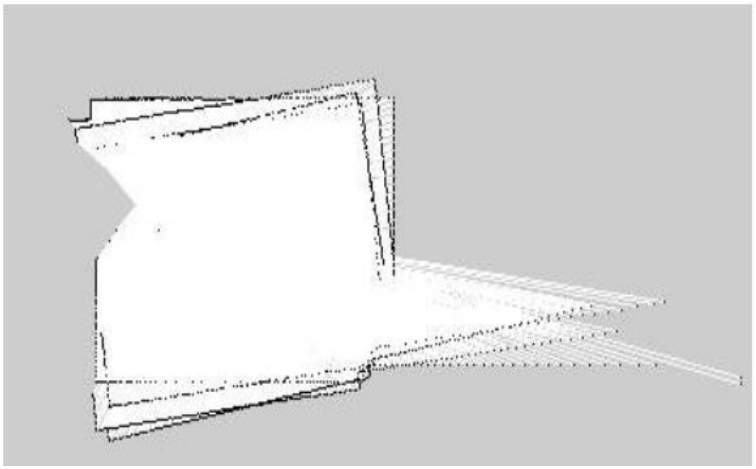


Figure 62 - Map file in case the laser is not properly configured

- We've many different ROS nodes. Each one launched their own ROS programs to execute different tasks. Where did all of these packages come from? The **Navigation Stack** is a set of ROS nodes and algorithms used to autonomously move a wheelchair from one point to another, avoiding all obstacles the wheelchair might find in its way. The ROS Navigation Stack comes with an implementation of several navigation-related algorithms which can help us perform autonomous navigation.

14.2 Navigation Stack

This is the last step to apply autonomous navigation, we implement a package to apply indoor autonomous navigation.

Here are some points describing the main functions of the navigation stack package:

- This package is responsible to integrate packages and apply autonomous navigation by setting a goal on the map and make the wheelchair navigate from the start point to the goal point.
- A 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base.
- Some configurations must be done first before applying the navigation stack:
    - Robot Setup

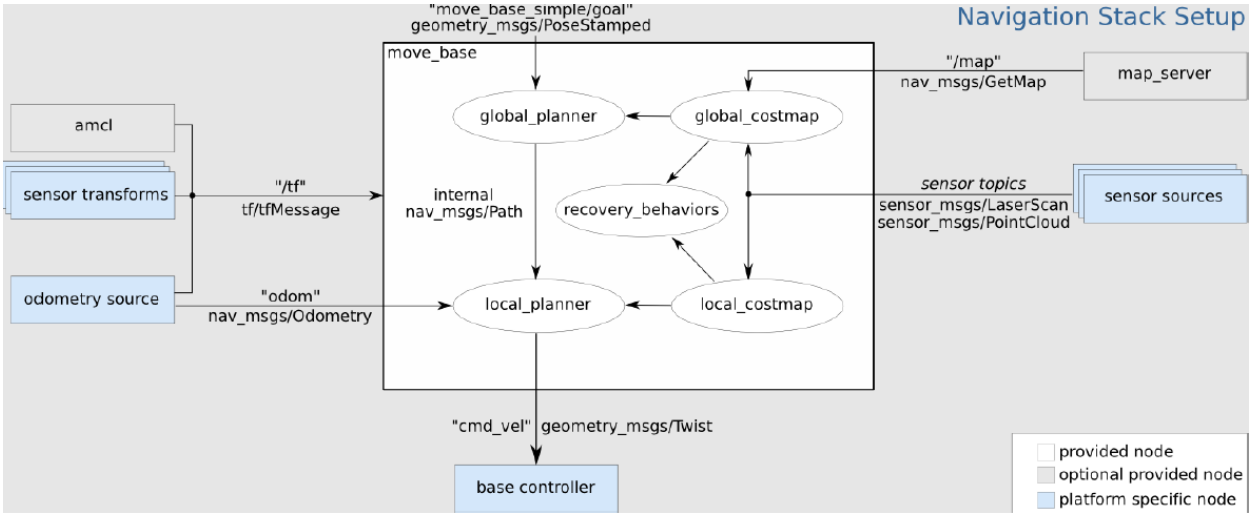


Figure 63 - Navigation Stack Diagram.

- The navigation stack assumes that the robot is configured in a particular manner to run. The previous figure shows an overview of this configuration. The white components are required components that are already implemented, the gray components are optional components that are already implemented, and the blue components must be created for each robot platform and we did all of them.
- Sensor Information (sensor sources):
    - The navigation stack uses information from sensors to avoid obstacles in the world, it assumes that these sensors are publishing either sensor_msgs/LaserScan or

sensor_msgs/PointCloud messages over ROS, in our project we use point cloud to laser scan sensor.
    - Odometry Information (odometry source)
    - The navigation stack requires that odometry information be published using tf and the nav_msgs/Odometry message.
    - Mapping (map_server) which is used to send the map we created to apply navigation on.
    - Base Controller (base controller)

- The navigation stack assumes that it can send velocity commands using a geometry_msgs/Twist message assumed to be in the base coordinate frame of the robot on the "cmd_vel" topic. This means there must be a node subscribing to the "cmd_vel" topic that is capable of taking (vx, vy, vtheta) <==> (cmd_vel.linear.x, cmd_vel.linear.y, cmd_vel.angular.z) velocities and converting them into motor commands to send to a mobile base.

Hardware Requirements, The ROS Navigation Stack is generic. That means it can be used with almost any type of moving robot, but some hardware considerations will help the whole system to perform better, so they must be considered. These are the requirements:

- The Navigation package will work better in differential drive and holonomic robots.
- The robot should mount a planar laser somewhere around the robot. It is used to build the map of the
  environment and perform localization.
- Its performance will be better for square and circular-shaped mobile bases.

To summarize, The Navigation Stack is a set of ROS nodes and algorithms, which work together to move a robot from position A to position B, avoiding the obstacles it may find in its way. To do this, the Navigation Stack requires many data inputs (of different kinds). In exchange, it will give as output the necessary command velocities to safely move the robot to the desired position.

14.3 SLAM Algorithm

- **SLAM** (simultaneous localization and mapping): is a method used for autonomous vehicles that let us build the **map** and **localize** our wheelchair in that map at the same time. SLAM algorithms allow the wheelchair to map out unknown environments. Then we use the map information to carry out tasks such as **path planning** and **obstacle avoidance.**
- Simultaneous Localization And Mapping – it's essentially complex algorithms that map an unknown environment. Using SLAM software, a device can simultaneously localize (locate itself in the map) and map (create a virtual map of the location) using SLAM algorithms.

- SLAM can trace its early development back to the robotics industry in the 1980s and 1990s. Today, SLAM technology is used in many industries. It has opened up opportunities to better map and understand environments whether they are indoor, outdoor, in-air, or underground.
- How SLAM Works? Broadly speaking, there are two types of technology components used to achieve SLAM:
    - The first type is sensor signal processing, including front-end processing, which is largely dependent on the sensors used.
    - The second type is pose-graph optimization, including the back-end processing, which is sensor-agnostic.

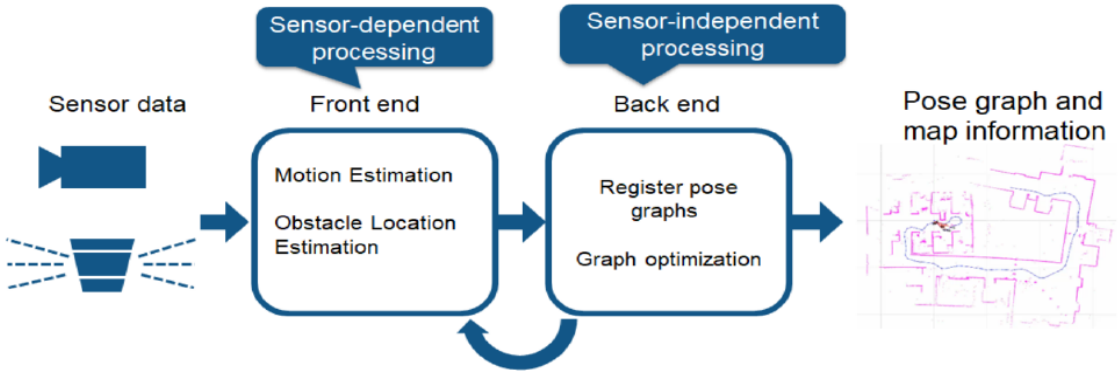


Figure 64 - SLAM Processing Flow

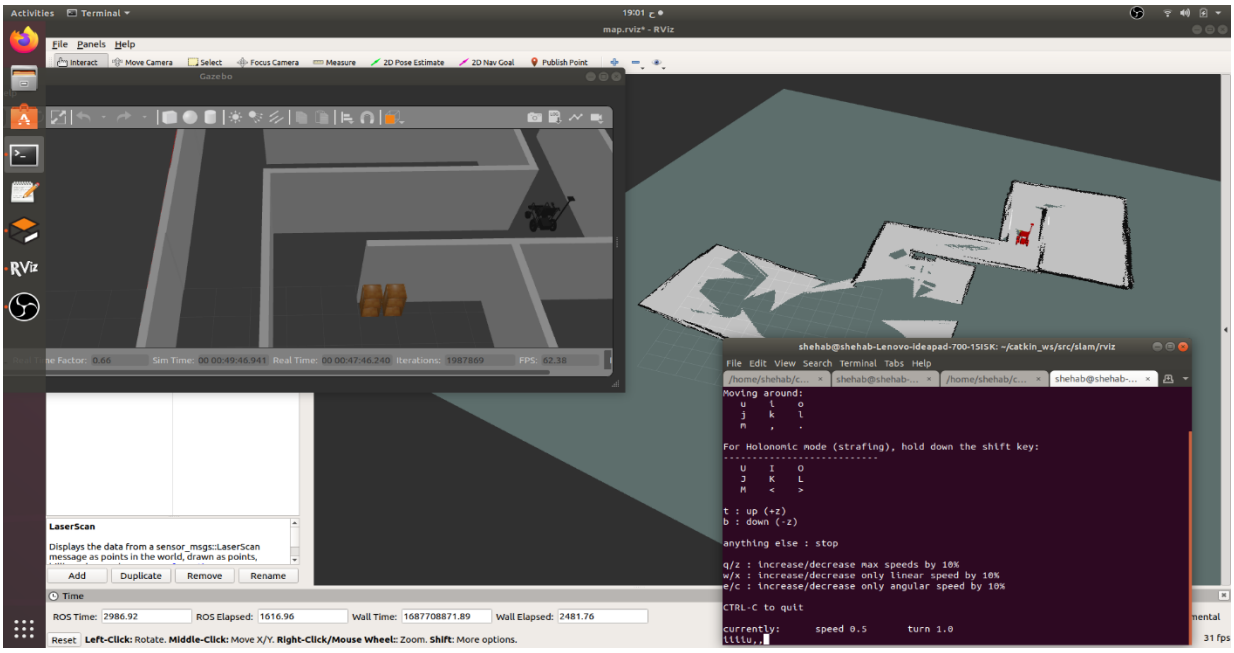


Figure 65 - The map we used to run all packages

As shown in the figure above, we scan the whole environment to create the map and save it using the map-server package as a picture (pgm file) and (yaml file) to be used in the navigation later.

14.4 Mapping

To perform our autonomous Navigation, the wheelchair must have a map of the environment. It will use this map for many things such as planning trajectories, avoiding obstacles, etc.

We can either give the wheelchair a prebuilt map of the environment (in the rare case that we already have one with the proper format), or we can build one by wheelchair. This second option is the most frequent one.

Before starting, though, we need to be introduced properly to a very important tool in ROS Navigation. we've already talked about **RViz**.

We can launch RViz and add displays to watch the Mapping progress. For Mapping, you'll need to use 2 displays of RViz:

- Robot Mode
- LaserScan Display
- Map Display

SLAM, Simultaneous Localization and Mapping (SLAM). This is the name that defines the robotic problem of building a map of an unknown environment while simultaneously keeping track of the robot's location on the map that is being built. This is the problem that Mapping is Solving. So, to summarize, we need to do SLAM to create a Map for the robot.

The gmapping ROS package is an implementation of a specific SLAM algorithm called **gmapping**.

So when we use the ROS Navigation stack, we only need to know (and have to worry about) how to configure gmapping for our specific robot (which in our case wheelchair). The gmapping package contains a ROS Node called slam_gmapping, which allows us to create a 2D map using the laser and pose data that our robot is providing while moving around an environment. This node reads data from the laser and transformations of the wheelchair and turns it into an **Occupancy Grid Map (OGM).**

- roslaunch slam gmapping.launch

1- we created a configuration launch file (gmapping.launch) to launch the gmapping package.

2- That launch file started a slam_gmapping node (from the gmapping package). Then we start moving

the wheelchair around the environment.

3- Then, the slam_gmapping node subscribed to the Laser and the Transform Topics (/tf) to get the data it needs, and it built a map

4- The generated map is published during the whole process into the /map topic, which is the reason we can see the process of building the map with Rviz (because Rviz just visualizes topics).

The /map topic uses a message type of **nav_msgs/OccupancyGrid** since it is an OGM. Occupancy is represented as an integer in the range {0, 100}. With 0 meaning completely free, 100 meaning completely occupied, and the special value of -1 for completely unknown

Let's start by seeing what we can do with the Map we've just created. Another of the packages available in the ROS Navigation Stack is the **map_server** package. This package provides the map_saver node, which allows us to access the map data from a ROS Service, and save it into a file.

Once we completed mapping the whole environment, save the map:

- Rosrun map_server map_saver ~/catkin_w/src/slam/maps/name_of_map

This command will get the map data from the map topic, and write it out into 2 files, name_of_map.pgm, and name_of_map.yaml.

- YAML file: contains the map metadata and the image name.
- Image.pgm: contains the encoded data of the occupancy grid map.

```
image: map1.pgm
resolution: 0.050000
origin: [-12.200000, -23.400000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196

```

Figure 66 - Image of YAML file

The YAML File generated will contain the 6 following fields:

• image: Name of the file containing the image of the generated Map.

• resolution: Resolution of the map (in meters/pixel).

• origin: Coordinates of the lower-left pixel in the map. These coordinates are given in 2D (x,y). the third value indicates the rotation. If there's no rotation, the value will be 0.

• occupied_thresh: Pixels that have a value greater than this value will be considered as a completely occupied zone.

• free_thresh: Pixels that have a value smaller than this value will be considered as a completely free zone.

• negate: Inverts the colors of the Map. By default, white means completely free, and black

means completely occupied.

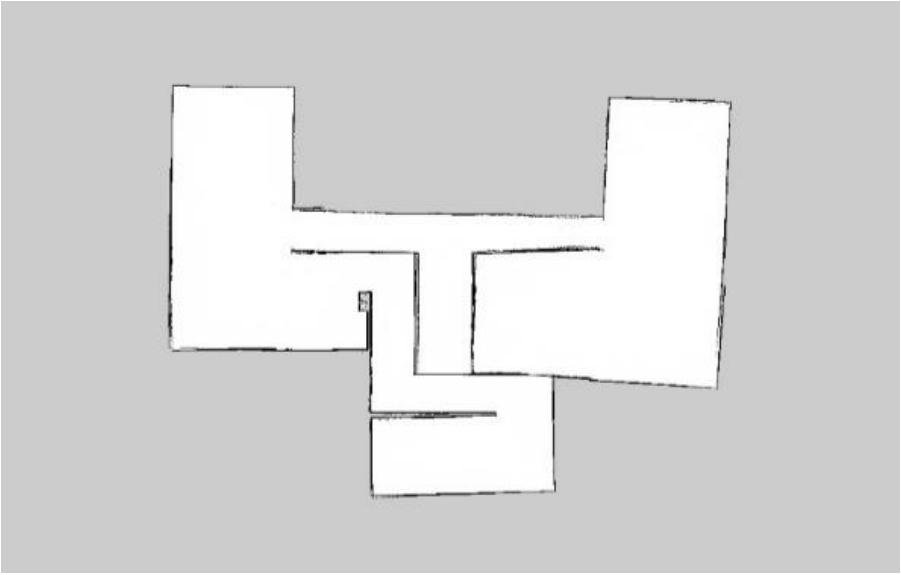


Figure 67 - Our pgm (Image) file of the saved map after finishing the scan

The image describes the occupancy state of each cell of the world in the color of the corresponding-

ing pixel. Whiter pixels are free, blacker pixels are occupied, and pixels in between are Unknown.

Besides the map_saver node, the map_server package also provides the map_server node.

This node reads a map file from the disk and provides the map to any other node that requests it

via a ROS Service. many nodes request to the map_server the current map in which the wheelchair is moving. This request is done, for instance, by the move_base node to get data from a map and use it to perform Path Planning, or by the localization node in order to figure out where in the map the wheelchair is.

The map that you created is a static map. This means that the map will always stay as it was when you created it. So when you create a Map, it will capture the environment as it is at the exact moment that the mapping process is being performed. If for any reason, the environment changes in the future, these changes won't appear on the map, hence it won't be valid anymore (or it won't correspond to the actual environment). The map that you created is a 2D Map. This means the obstacles that appear on the map don't have height. So if, for instance, you try to use this map to navigate with a drone, it won't be valid. There exist packages that allow you to generate 3D mappings.

Hardware Requirements, To build a proper Map, you need to fulfill these 2 requirements:

1. Provide good laser data.

2. Provide good odometry data.

The slam_gmapping node will then try to transform each incoming laser reading to the Odom frame.

Creating a launch file for the slam_gmapping node, The main task to create this launch file is to correctly set the parameters for the slam_gmapping node. This node is highly configurable and has lots of parameters you can change in order to improve the mapping performance. These parameters will

be read from the ROS Parameter Server and can be set either in the launch file itself. If you don't set some parameters, it will just take the default values. You can have a look at the complete list of parameters available for the slam_gmapping node on ROS documentation in the gmapping section (Link in resources below).

Sometimes, the map that you create will contain stuff that you do not want to be there. For example:

1. There could be people detected by the mapping process that has been included in the map. we don't

want them on the map as black dots!

2. There could be zones where you do not want the robot to move (for instance, avoid the wheelchair

going close to the downstairs area to prevent it from falling down). For all these cases, you can take

the map generated and modify it manually.

In summary, The very first thing you need in order to navigate is a Map of the environment. You can't navigate if you don't have a Map. Furthermore, this Map does have to be built properly, so it accurately

represents the environment you want to navigate.

In order to create a Map of the environment, ROS provides the slam_gmapping node (of the gmapping package), which is an implementation of the SLAM (Simultaneous Localization and Mapping) algorithm. Basically, this node takes as input the laser and odometry readings of the robot (in order to get data from the environment), and creates a 2D Map.

This Map is an occupancy representation of the environment, so it provides information about the occupancy of each pixel in the Map. When the Map is completely built, you can save it into a file.

Also bear in mind that you only need the slam_gmapping node (the Mapper), when you are creating a map. Once the map has been created and saved, this node is not necessary

anymore, hence it must be killed. At that point, you should launch the map_server node, which is the node that will provide the map that you just created to other nodes.

10.1.4 Robot Localization

When the wheelchair moves around a map, it needs to know which is its position within the map, and which is its orientation. Determining its location and rotation by using its sensor readings is known as Robot Localization.

To visualize Localization in Rviz, you'll basically need to use 3 elements of RViz:

• LaserScan Display

• Map Display

• PoseArray Display

- roslaunch slam amcl.launch map:='name_of_map'.
- rosrun rviz rviz

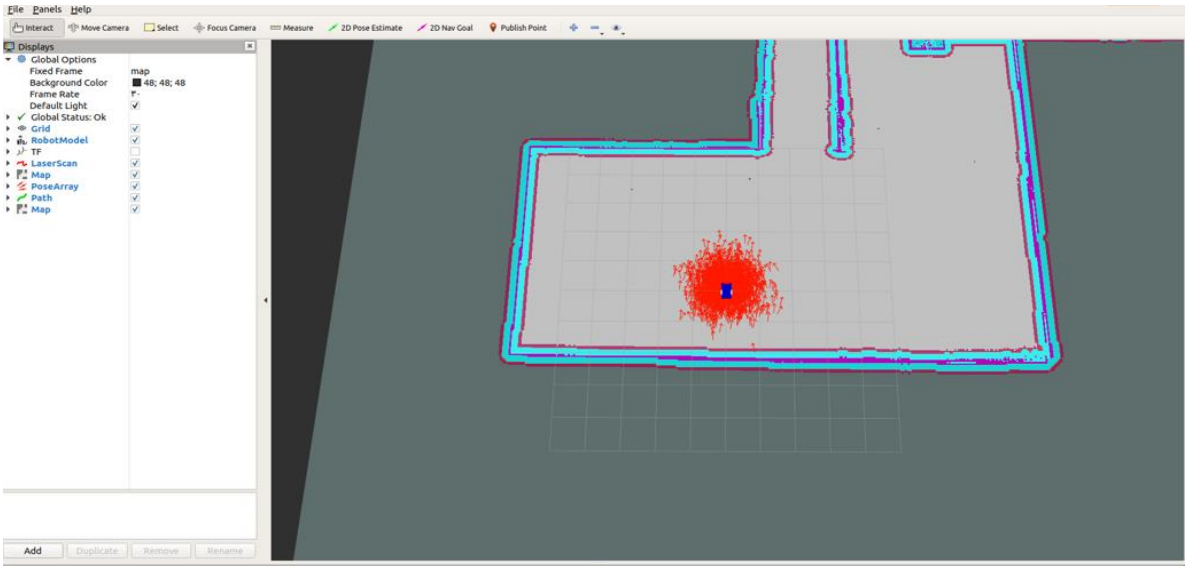


Figure 68 - Particles visualization in Rviz

Ok, but...what just happened? What were those strange arrows we were visualizing in RViz? Let's first introduce some concepts.

Monte Carlo Localization (MCL), Because the robot may not always move as expected, it generates many random guesses as to where it is going to move next. These guesses are known as particles. Each particle contains a full description of a possible future pose. When the robot observes the environment it's in (via sensor readings), it discards particles that don't match with these readings and generates more particles close to those that look more probable. This way, in the end, most of the particles will

converge in the most probable pose that the robot is in. So the more you move, the more data you'll get from your sensors, hence the localization will be more precise. These particles are those arrows that we saw in RViz. This is known as the Monte Carlo Localization (MCL) algorithm, or also particle filter localization.

The AMCL (Adaptive Monte Carlo Localization) package provides the amcl node, which uses the MCL system in order to track the localization of a robot moving in a 2D space. This node subscribes to the data of the laser, the laser-based map, and the transformations of the robot, and publishes its estimated position in the map. On startup, the amcl node initializes its particle filter according to the parameters provided.

Hardware Requirements وIn order to get a proper Robot localization, we need to fulfill 3 basic requirements:

• Provide good laser data.

• Provide good odometry data.

• Provide good laser-based map data.

Transforms, we need to be publishing a correct transform between the laser frame and the base of the wheelchair's frame. the wheelchair uses the laser readings in order to constantly re-calculate its localization.

Creating a launch file for the AMCL node, we need to have a launch file in order to start the amcl node. This node is also highly customizable and we can configure many parameters in order to improve its performance. These parameters can be set in the launch file itself. we can have a look at a complete list of all of the parameters that this node has here: http://wiki.ros.org/amcl Let's have a look at some of the most important ones:

General Parameters

• odom_model_type (default: "diff"): It puts the odometry model to use. It can be "diff," "omni," "diff-corrected," or "omni-corrected."

• odom_frame_id (default: "odom"): Indicates the frame associated with odometry.

• base_frame_id (default: "base_link"): Indicates the frame associated with the robot base.

• global_frame_id (default: "map"): Indicates the name of the coordinate frame published by the localization system.

• use_map_topic (default: false): Indicates if the node gets the map data from the topic or from a service call.

Filter Parameters These parameters will allow you to configure the way that the particle filter performs.

• min_particles (default: 100): Sets the minimum allowed number of particles for the filter.

• max_particles (default: 5000): Sets the maximum allowed number of particles for the filter.

• kld_err (default: 0.01): Sets the maximum error allowed between the true distribution and the estimated distribution.

• update_min_d (default: 0.2): Sets the linear distance (in meters) that the robot has to move in order to perform a filter update.

• update_min_a (default: ⇡/6.0): Sets the angular distance (in radians) that the robot has to move in order to perform a filter update.

• resample_interval (default: 2): Sets the number of filter updates required before resampling.

• transform_tolerance (default: 0.1): Time (in seconds) with which to post-date the transform that is published, to indicate that this transform is valid into the future.

• gui_publish_rate (default: -1.0): Maximum rate (in Hz) at which scans and paths are published for visualization. If this value is -1.0, this function is disabled.

Laser Parameters These parameters will allow you to configure the way the amcl node interacts with the laser.

• laser_min_range (default: -1.0): Minimum scan range to be considered; -1.0 will cause the laser's reported minimum range to be used.

• laser_max_range (default: -1.0): Maximum scan range to be considered; -1.0 will cause the laser's reported maximum range to be used.

• laser_max_beams (default: 30): How many evenly-spaced beams in each scan are to be used when updating the filter?

• laser_z_hit (default: 0.95): Mixture weight for the z_hit part of the model.

• laser_z_short (default: 0.1): Mixture weight for the z_short part of the model.

• laser_z_max (default: 0.05): Mixture weight for the z_max part of the model.

• laser_z_rand (default: 0.05): Mixture weight for the z_rand part of the model.

In summary, In order to navigate around a map autonomously, a robot needs to be able to localize itself to the map. And this is precisely the functionality that the amcl node (of the amcl package) provides us. In order to achieve this, the amcl node uses the MCL (Monte Carlo

Localization) algorithm. You can't navigate without a map, as you learned in the previous chapter. But, surely, you can't navigate if your robot isn't able to localize itself in it either. So, the localization process is another key part of ROS Navigation. Basically, the amcl node takes data from the laser and the odometry of the robot, and also from the map of the environment, and outputs an estimated pose of the robot. The more the robot moves around the environment, the more data the localization system will get, so the more precise the estimated pose it returns will be.

14.5 Motion Planning

As we explained in detail in our packages implementation section, and the algorithms of slam and gmapping that were used in this project in order to make the wheelchair safely deliver to the target point, we saw how exactly our wheelchair goes from the start point to another point

- Implementing the wheelchair description package successfully and visualizing the robot in RViz.
- Show it and add a suitable environment using the Gazebo worlds to simulate and run its simulation using the Gazebo simulation tool.
- Test all transformations and odom frames.
- Implementing the wheelchair odometry package successfully and receiving odom calculations and links transformations correctly
- Implementing mapping and navigation package successfully and building a map using SLAM Algorithm.
- Implementing the Navigation Stack package successfully and applying autonomous navigation by setting a goal for it.
- Finally, ROS can show output to a usable 2D map of the test site with the laser scan data.
- Our wheelchair can navigate to a goal autonomously.

14.6 Path Planning

Now we create a map of an environment and localize the wheelchair in it. So, at this point, we have all that we need in order to perform Navigation. That is, we're now ready to plan trajectories in order to move the wheelchair from pose A to pose B.

Visualize Path Planning in Rviz, we can also launch RViz and add displays in order to watch the Path Planning process of the wheelchair. In order to watch the Path Planning process. we'll basically need to use 3 elements of RViz:

• Map Display (Costmaps).

• Path Displays (Plans).

• 2D Tools.

Visualize Costmaps

• /move_base/global_costmap/costmap in order to visualize the global costmap

• /move_base/local_costmap/costmap in order to visualize the local costmap.

• You can have 2 Map displays, one for each costmap.

Visualize Plans

• /move_base/NavfnROS/plan in order to visualize the global plan.

• /move_base/DWAPlannerROS/local_plan in order to visualize the local plan.

• You can also have 2 Path displays, one for each plan

The move_base package contains the move_base node. The move_base node is one of the major elements in the ROS Navigation Stack, since it links all of the elements that take place in the Navigation process. Without this node, the ROS Navigation Stack wouldn't make any sense!

The main function of the move_base node is to move the wheelchair from its current position to a goal position. Basically, this node is an implementation of a SimpleActionServer, which takes a goal pose with message type geometry_msgs/PoseStamped. Therefore, we can send position goals 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.

So, each time you set a Pose Goal using the 2D Nav Goal tool from RViz, what is really happening is that a new message is published into the move_base/goal topic. But, this is not the only topic that the move_base Action Server provides. As for every action server, it provides the following 5 topics:

• move_base/goal (move_base_msgs/MoveBaseActionGoal).

• move_base/cancel (actionlib_msgs/GoalID).

• move_base/feedback (move_base_msgs/MoveBaseActionFeedback).

• move_base/status (actionlib_msgs/GoalStatusArray).

• move_base/result (move_base_msgs/MoveBaseActionResult).

When this node receives a goal pose, it links to components such as the global planner, local planner, recovery behaviors, and costmaps, and generates an output, which is a velocity

command with the message type geometry_msgs/Twist, and sends it to the /cmd_vel topic in order to move the wheelchair.

The move_base node, just as we saw with the slam_gmapping and the amcl nodes, also has parameters that you can modify. For instance, one of the parameters that you can modify is the frequency at which the move_base node sends these velocity commands to the base controller.

sending a pose goal to the move_base node activates some kind of process, which involves other nodes, and that results in the wheelchair moving to that goal pose. That's very interesting, but....what is this process that is going on? How does it work? What are these other nodes that take place?

Let's start by introducing one of the main parts that take place in this process: **the global planner**.

When a new goal is received by the move_base node, this goal is immediately sent to the global

planner. Then, the global planner is calculating a safe path in order to arrive at that goal pose. This path is calculated before the wheelchair starts moving, so it will not take into account the readings that the wheelchair's sensors are doing while moving.

Each time a new path is planned by the global planner, this path is published into the **/plan** topic.

So, we now know that the first step of this navigation process is to calculate a safety plan so that our wheelchair can arrive to the user-specified goal pose. But...how is this path calculated? There exist different global planners. Depending on our setup.

The **Navfn** planner is probably the most commonly used global planner for ROS Navigation. It uses Dijkstra's algorithm in order to calculate the shortest path between the initial pose and the goal pose. Below, you can see an animation of how this algorithm works. The global planner used by the move_base node it's usually specified in the move_base parameters file. In order to do this, you will add one of the following lines to the parameters file:

base_global_planner: "navfn/NavfnROS"

The global planner also has its own parameters in order to customize its behavior. The parameters for the global planner are also located in a YAML file.

Navfn Parameters:

• /allow_unknown (default: true): Specifies whether or not to allow navfn to create plans that traverse

unknown space.

• /planner_window_x (default: 0.0): Specifies the x size of an optional window to restrict the planner to.

• /planner_window_y (default: 0.0): Specifies the y size of an optional window to restrict the planner to.

we've seen that a global planner exists that is in charge of calculating a safe path in order to move the wheelchair from an initial position to a goal position.

When we plan a trajectory, this trajectory has to be planned according to a map, right? A path without a map makes no sense. A costmap is a map that represents places that are safe for the wheelchair to be in a grid of cells. Usually, the values in the costmap are binary, representing either free space or places where the wheelchair would be in a collision. Each cell in a costmap has an integer value in the range {0,255}. There are some special values frequently used in this range, which work as follows:

• 255 (NO_INFORMATION): Reserved for cells where not enough information is known.

• 254 (LETHAL_OBSTACLE: Indicates that a collision-causing obstacle was sensed in this cell.

• 253 (INSCRIBED_INFLATED_OBSTACLE): Indicates no obstacle, but moving the center of the

robot to this location will result in a collision.

• 0 (FREE_SPACE): Cells where there are no obstacles and, therefore, moving the center of the robot to

this position will not result in a collision.

There exist 2 types of costmaps: global costmap and local costmap. The main difference between them is, basically, the way they are built:

• The global costmap is created from a static map.

• The local costmap is created from the robot's sensor readings.

global costmap is the one used by the global planner. So, the global planner uses the global costmap in order to calculate the path to follow.

14.7 Obstacle Avoidance

Once the global planner has calculated the path to follow, this path is sent to the local planner.

The local planner, then, will execute each segment of the global plan (let's imagine the local plan as a smaller part of the global plan). So, given a plan to follow (provided by the global planner) and a map, the local planner will provide velocity commands in order to move the robot.

Unlike the global planner, the local planner monitors the odometry and the laser data and chooses a collision-free local plan (let's imagine the local plan as a smaller part of the global plan) for the wheelchair. So, the local planner can recompute the wheelchair's path on the fly in order to keep the wheelchair from striking objects, while still allowing it to reach its destination.

The local planner uses the local costmap in order to calculate local plans. Unlike the global costmap, the local costmap is created directly from the wheelchair's sensor readings. The costmap automatically subscribes to the sensor topics and updates itself according to the data it receives from them. Each sensor is used to either mark (insert obstacle information into the costmap) or clear (remove obstacle information from the costmap).

It could happen that while trying to perform a trajectory, the robot gets stuck for some reason.

Fortunately, if this happens, the ROS Navigation Stack provides a method that can help your robot

to get unstuck and continue navigating.

Basically, the rotate recovery behavior is a simple recovery behavior that attempts to clear out

space by rotating the robot 360 degrees. This way, the wheelchair may be able to find an obstacle-free

path to continue navigating.

To summarize, After getting the current position of the robot, we can send a goal position to the move_base node. This node will then send this goal position to a global planner which will plan a path from the current wheelchair position to the goal position. This plan is with respect to the global costmap, which is feeding from the map server. The global planner will then send this path to the local planner, which executes each segment of the global plan. The local planner gets the odometry and the laser data values and finds a collision-free local plan for the robot. The local planner is associated with the local costmap, which can monitor the obstacle(s) around the wheelchair. The local planner generates the velocity commands and sends them to the base controller. The wheelchair base controller will then convert these commands into real movement. If the wheelchair is stuck somewhere, the recovery behavior nodes, such as the clear costmap recovery or rotate recovery, will be called.

14.8 Using Raspberry Pi instead of Laptop

In order to get our wheelchair ready to be a good product, We must remove the laptop and move all out project packages to RPI to be used automatically when the user gives an order.

First of all, Scanning the environment and creating a custom map is done using the laptop. Then we save the map in its path in the “slam pkg” to be used later in the navigation.

ROS systems and algorithms are complex so we need to distribute the computational power to be implemented successfully. In our project, we split our nodes into two Raspberry Pi and set a WiFi Communication between them. The workflow is:

1. A letter is sent by a wireless joystick (A, B, C, or D) indicating the required goal location.
2. According to the command and the pre-saved map, a specific command is sent from the RPI to the Arduino node through the topic “/cmd_vel”.
3. The Arduino sends it to the motors, and the wheelchair starts to navigate to the goal by the path planner as we explained previously.
4. The encoder readings are sent back through the topic ”/encoder”.
5. The RPI will process the encoder signal (ticks) and calculate the (X, Y, theta == distances and orientation) as shown in the Odometry Calculations section.
6. These Calculations are passed to the slam algorithm to complete the navigation.

The Raspberry Pi then starts to take images from the camera, does processing on it (converts the depth image into laser scan) to reduce the wifi lag, and sends this to the master computer which takes laser scan, odometry data from Arduino, and velocity commands from the master and send it to Arduino to start move wheelchair to goal.

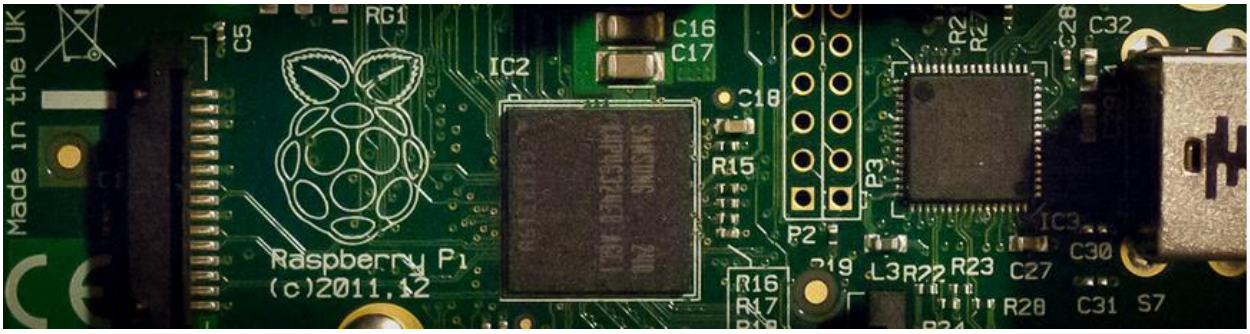


Figure 69 - Raspberry Pi 4, the main brain for the wheelchair

Why do all this work? In order to operate our packages, some work has to be done locally, which will be a nuisance for the user to work back and forth on two computers. So remotely accessing the move_robot node from the remote Raspberry Pi using SSH is recommended. This allows us to execute commands on the node from a remote RPi. Here’s an example of how to remotely connect to the node from the remote PC.

- $ ssh move_pub@192.168.7.200

Notes on SSH:

SSH refers to the application or the protocol that allows us to log into another computer in the network and run commands on a remote system and copy files to another system.

It is often used when connecting to a remote computer and sends a command from a terminal window in Linux. To do this, the ssh application should be installed as follows:

- $ sudo apt-get install ssh

To connect to a remote computer (in this case nodes we loaded to the RPI), we use the following command to connect in the terminal window. Once the connection is established, commands can be entered just like:

- $ ssh username @ ip address of the remote PC

In the case of Raspberry Pi, since the SSH server of Ubuntu MATE 16.04.x and Raspbian are disabled by default. Raspberry Pi OS has the SSH server disabled by default. It can be enabled manually from the terminal or the desktop:

1. Launch Raspberry Pi Configuration from the Preferences menu
2. Navigate to the Interfaces tab
3. Select Enabled next to SSH
4. Click OK

Raspberry Pi setup:

1. Install Ubuntu, ROS, and the Kinect driver on RPI
2. Running ROS over multiple computers
3. Install packages (freenect_camera, rosserial) on RPI.

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.

15 ROS Challenges and Limitations

15.1 ROS System on our wheelchair (Hardware Implementation)

We faced a challenge to find an electric wheelchair and due to its high cost, we couldn't buy it and asked to get it from the university. We decided that until we got the wheelchair, we would be working on a prototype model to which we can apply the concept and the ROS system, then we can scale the project to be on a wheelchair. Unfortunately, the wheelchair came just before the final exams and it needed too much work on its wheels, motors, encoders, etc. so that we can use and implement Indoor Autonomous Navigation.

However, we managed successfully to run all the packages of the wheelchair - all explained in detail above tested all tasks in Gazebo and RViz, run slam algorithms, and navigated the wheelchair autonomously.

Our project **packages and demos of mapping and navigation** are published within our organization on GitHub:

- The Prototype: ROS_Prototype
- The Wheelchair: ROS_Wheelchair

15.2 The most complicated ROS errors and their solutions

Robot Operating System is famous for its complicated errors, and the little community using this technique makes it harder to solve them. This can waste a lot of time searching - it is not excessive if we say that you may spend two days trying to just know where the error is! and the same period for solving it!

We tried in this section to mention some of them so that it may be useful and a time-saver for a person who may read and use ROS in such a project later.

1. Go to Goal:

Try creating a node sending goal from the terminal but it sometimes not works and worked after some tries but it was very slow, and no one knows what error we face but follow this video step by step and all things will be fine.

Path Planning ROS | Move Turtlebot to goal | Actionlib -CPP | Move Base

2. Robot rotating very slowly:

   Try moving the robot. we send cmd_vel to the robot and it was moving forward and backward very well but it was very slow in rotating.

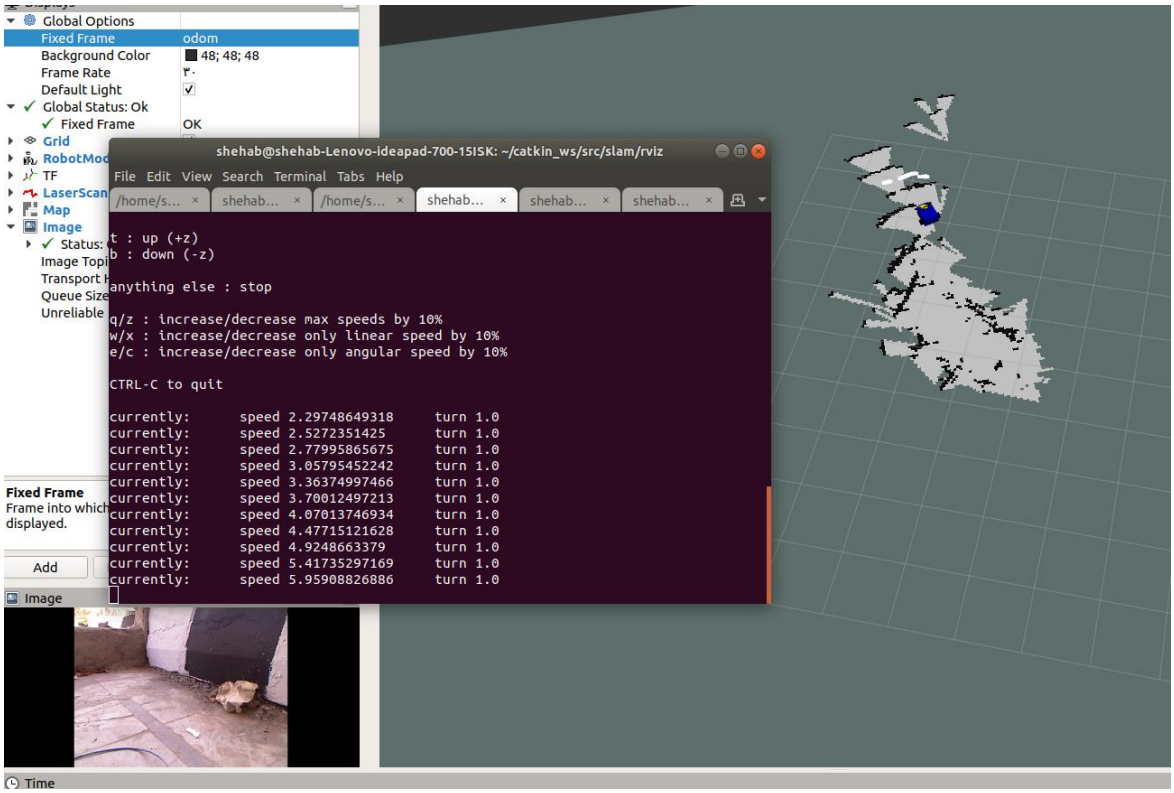


Figure 70- Rotation speed error

Omni-directional robot rotation is very slow with libgazebo_ros_planar_move.so

3. Error while connecting between the Arduino and the laptop running ROS. (Permission denied)

To establish the connection between Arduino and the laptop, we need to run this command:

- rosrun rosserial_python serial_node.py /dev/ttyACM0

As ttyACM0 is the name of the port number, but sometimes this is not correct as what happened with us.

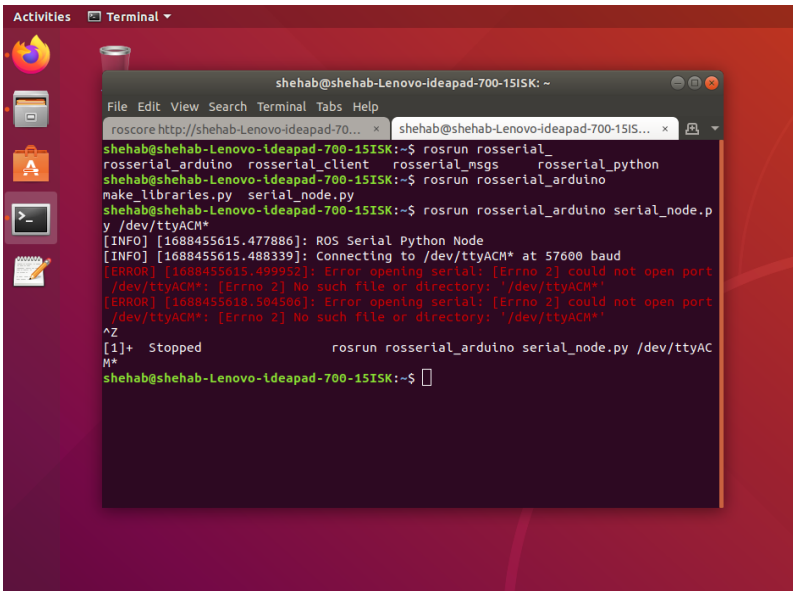


Figure 71 - Permission denied error

Solution is to use:

- rosrun rosserial_python serial_node.py /dev/ttyACM*

4. AMCL randomly loses localization:
   The good thing is when the laser points match exactly the map then the localization is perfect, however when the laser points do not match exactly the map (even slight difference) the AMCL tries correcting the robot's position on the map, but it sets the robot in a completely wrong position.

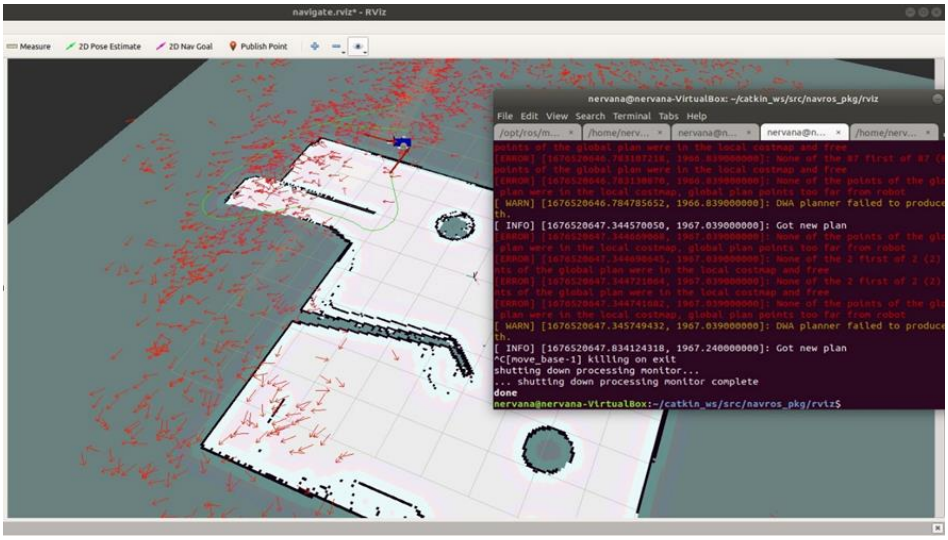


Figure 72 - AMCL randomly loses localization problem

Solution: AMCL randomly loses localization.

5. Autonomous navigation with obstacle avoidance to the goal:

All things were fine and the robot is going to its goal but, when it face an obstacle it did not correct its path.

Solution: No costmap in local map [closed].

6. Kinect camera sees what is behind it not what is forward:

Reason: in computer vision, the z-axis is to forward but the coordinates it's to above.

Solution: Set configuration in origin and rpy of the camera in the URDF file.

7. Issues with Arduino: Wrong checksum, Mismatched protocol version

- Using Arduino uno and listening to motor speed but the rosserial sends wrong messages when connecting with the raspberry pi

Reason: The laptop is a much faster computer than the RPI, which means it would be able to process the data that has arrived faster than the raspberry pi could.

Solution: when replacing the laptop with the raspberry pi, we need to increase the loop delay to decrease the messages sent per second.

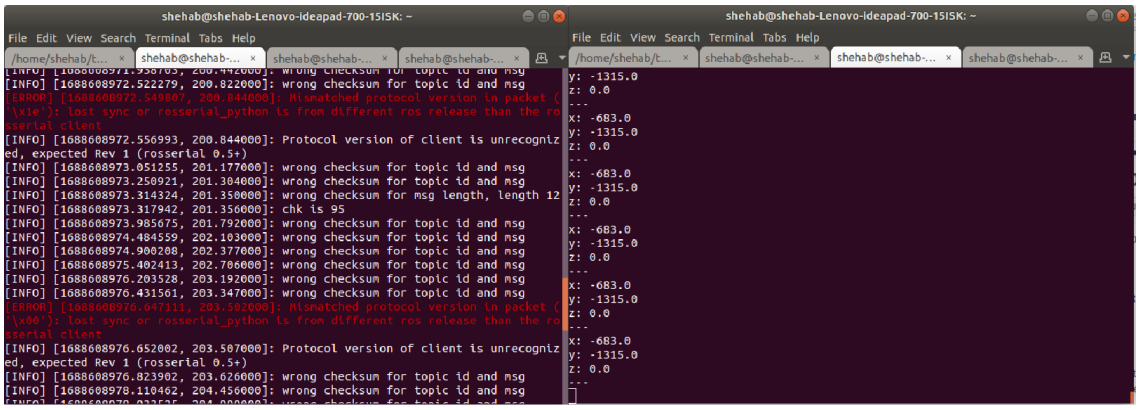


Figure 73 - Wrong checksum

Database

16.1 Caregiving System Database

16.1.1 Introduction

In this chapter, we describe the caregiving system, a mobile app that can visualize the patient's medical state with the use of hardware sensors, to make sure that the patient doesn't suffer from any medical issues. This app gets the medical state from the medical sensors on the chair using API provided by the backend server and used by both the Raspberry PI component attached to the chair and the mobile app.

16.1.2 System Overview

To design our system we divide it into 4 separate subsystems:

- Database
- Backend
- Cloud
- Mobile app

each one has his functionality and job to do to build our health tracking system that includes: measuring body temperature, Heart rate, and blood oxygen level, and a notification alert if there is any danger in these measurements. We will discuss each subsystem separately later.

16.2.1 Database

This sub-system aims to collect, store, and manage our system's data that comes from sensors that are related to patient health like body temperature, oximeter, …etc data comes that related to the patient and caregiver information also database will be our storage and backup for all of our data.

Types of database

There are two types of Database we could use:

- SQL Database (Relational Database): the Items in a relational database are organized as a set of tables with columns and rows.
- NoSQL Database: NoSQL databases come in various types based on their data model and data stored in a kind of document, key-value, wide-column, or graph.

Because SQL database technology provides the most efficient and flexible way to access structured information and also the data are stored in the form of tables that we export later as CSV files and use in creating a dashboard so we decide to stick with it.

16.2.2 Database Management System (DBMS)

And the way we use for creating and maintenance of the SQL database is by using software called Database Management System (DBMS).

DBMS is used also for processing queries from the application program and has a layer used for accessing the database and returning the needed data from it.

There are multiple DBMS software we could use for example:

- MySQL
- PostgreSQL
- Oracle Database
- Microsoft SQL Server

But we choose to use PostgreSQL since we use FastAPI as our main backend framework and because of its advanced features, performance, and support for complex queries, also has excellent support for JSON and provides features like full-text search and geographic data support, which can be beneficial for certain applications.

16.2.3 Object Relational Mapping (ORM)

The main way we can create the database is by using SQL (Structured Query Language) which is a programming language made to use and interact with databases by doing the CRUD (Create, Read, Update, Delete) operation on the database.

But when we work on the database we use ORM instead of SQL queries. ORM is a programming technique and software pattern that allows us to make the CRUD operations by using object-oriented programming with the programming language we use to work with in the backend. ORM also handles the conversion from our programming language (Python) to SQL queries that interact directly with the database.

Many popular ORM frameworks use Python like Django ORM, SQLAlchemy, and Peewee. We use SQLAlchemy in our project and will discuss it later.

16.3 Entity Relationship Diagram (ERD)

The ERD is a way we use to design our database and describe the entities we have and the relation between them.

The ERD can be designed by one of the following schemas:

- Conceptual Schema: A high-level representation of the overall structure and organization of data within a database system.
- Logical Schema: A detailed representation of the structure and organization of data within a database system and describe how data is organized, stored, and related to each other.

We design our database by both schemas and will describe our entities and relations now. The database we design consists of six entities and seven relations between them as shown in the following figure by simplifying it we found we have four main entities which are Chair, Sensor's Data, Patient, and Caregiver the other entities are features not essential.

16.3.1 Chair Entity

This entity is used to store the needed details for the chair such as the following Barcode, Password, and Available each one has his job to achieve. The Barcode attribute is a unique number for the chair that identifies the chair currently used or available to use. For the Password attribute, our chair has a password for security purposes to make it inaccessible for anyone to see the data of the medical sensors. The Available attribute, this attribute shows us if

the chair is available to use from another patient and connect to him to get the medical data from him. Also, this entity has a relationship between two entities which are Sensor's Data, Location, and Patient.

- First the relation between the Chair and Sensor's Data, this is one of the most important entities in our database, this is the entity that works on storing the health status of the patient that comes from the medical sensors on the chair from the backend. And because the data that relates to a specific chair can be many so we needed to store it in a separate entity so we can manage the one-to-many relationship between the chair and the sensor. Also, the chair is available so the chair can be actively used by the user or not active so the relation between the Chair and the Sensor's Data is 'may' because of the possibility to have a chair not having any data.

- Second the relation between the Chair and the Location, one of the features we added in our system is the possibility to track the location of the chair that the patient used to the caregiver can have full knowledge of where the patient is right now and what is his current medical health. And the relation between the Location and Chair is one-to-many because the data is sent to the database to the same chair with multiple data. And the relation also is 'may' because the chair cannot be active and not used so there is no data store for it.

- Third the relation between the Chair and the Patient, this relation is also important because the Patient is the entity that works with all the data of the patient who is the main target of our graduation project and the one who the caregiving system work for. As the chair is only related to only one patient so the relation between them is a one-to-one relationship which means the patient can only use one chair for a time and with the ability that the chair can be active or not, the relation is 'may' relationship.

16.3.2 Sensor's Data Entity

This entity, as we previously explained, works with the data coming from the sensors on the chair that is currently activated and used by the patient. It stored the data that comes from the sensors as body temperature, oximeter (the level of oxygen in the blood), pulse rate, and date (the time the data send for statistics purposes).

The only relation it has is the relation between it and the chair and as we explained it has a one-to-many relationship where the single data in the sensor's data only related to just one chair and the chair can have many sensors' data. Also, the sensor's data 'must' connect to a chair to be valid data.

16.3.3 Location Entity

This entity as we explained stores the location of the chair and has a relationship with the chair one-to-many as for a single location data it is connected to one chair and the chair 'must' be existed to store these data. And store two columns which are the main coordinates of the location's latitude and longitude.

16.3.4 Caregiver Entity

This entity uses to store the information and the data of the Caregiver who is the actual user of the mobile application and the one who is tracking the health status of the patient and usually

one of the parents, relevant, or friend of the patient and want to track his health for any emergency or just want to look after him while he is inside or outside the home. The data we stored for the caregiver are first name, last name, username, email, gender, age, and password. All the first names, last names, gender, and age are general information for all the caregivers who register for the application. For usernames and passwords, we use them for authentication and authorization in the application and backend and the username and the email are unique so no two caregivers can have the same username and email. For the relation between the entities, this entity has two relationships between the Patient and the Emergency Notifications.

- For the first relation between the Caregiver and the Patient, one of the main features inside the application is the caregiver can add a new patient and connect him to the chair and track him or track an already existing patient to the application to track. And because one caregiver can add multiple patients to track, and the single patient can be tracked by multiple caregivers so the relation between them is a many-to-many relationship. And because the caregiver can have just an account on the application without the need to track any patient then the relation is ‘may’.

- The second relation is the relation between the Caregiver and Emergency Notification, normally we don’t store the notification on the database but these act like a history for the caregiver to know when and on what sensor the drop in the health status happens because the data in Sensor’s Data are filling too fast we need a reference to where the drop happens to make it faster to get this drop. As the notification can be sent many times to the same use so the relation is one-to-many. Also, the caregiver can’t receive any notifications because there is no emergency happened or any drop so the relation is ‘may’.

16.3.5 Emergency Notification Entity

This entity works on storing the emergency notification that happens when a health drop happens. As we explained previously the notification can have only one caregiver that connects to him that why the relationship is one-to-many.

16.3.6 Patient Entity

- This entity is storing the information of the patient who is the user of the chair and his attributes are first name, last name, gender, and age which are the basic information of him. The one responsible for entering this information is the caregiver who is the relative of the patient and the organization that owns the chair and links it to the patient. The Patient entity has two entities the Chair entity and the caregiver entity as explained in the previous section.
- The relation is a one-to-one relationship between the Chair and the patient because the patient can only use one chair and vice versa the chair is only used by one patient. And because the chair can be active or not the relation is ‘may’ for the patient and ‘must’ for the chair as the patient must have a chair to connect to.
- The relation between the Chair and the Patient is many-to-many because the patient can be tracked by multiple caregivers who are his parents and relatives and vice versa the caregiver can track multiple patients. And because the patient can use a chair but no one tracks him so the relation is ‘may’ for the caregiver and also ‘may’ for the patient as the caregiver is not necessary to track the patient.

Backend

17.1.1 Introduction to backend

Backend refers to a web application's server side, consisting of the server, an application, and a database. It is responsible for processing and storing data and serving it to the front end of the application, either mobile or web, which is what the user sees and interacts with.

The backend is typically built using languages such as Java, Python, Ruby, and PHP, and frameworks such as Fast API, Node.js, and Django. These languages and frameworks provide developers with the tools they need to build robust and scalable web applications.

One of the backends' key functions is handling requests from the front end and responding with the appropriate data. This involves interacting with the database, performing calculations, and executing necessary logic.

The backend is also responsible for ensuring the security of the application. This includes implementing authentication and authorization mechanisms and protecting against common security threats such as SQL injection and cross-site scripting.

17.1.2 What is Authentication and authorization?

Authentication and authorization are two important security mechanisms used in the backend to control access to resources and protect against unauthorized access.

Authentication refers to the procedure of confirming the identity of a person or user attempting to access a system or resource. This is typically done by requiring the user to provide a username and password, which are compared against a database of authorized users. Other authentication methods include two-factor authentication, biometric authentication, and single sign-on.

Authorization, conversely, is the process of allowing or refusing access to a resource based on the authenticated user's permissions. Once a user has been authenticated, the application checks their permissions to determine whether they can access the requested resource. Permissions are typically defined by assigning roles or privileges to users based on their level of access or responsibility within the application.

Together, authentication and authorization provide a powerful security mechanism that helps to ensure that only authorized users can access sensitive data or perform certain actions within the application. Implementing these mechanisms properly is critical to protecting the overall data of the application.

17.1.3 Fast API Framework and why this technology?

Fast API is a modern, fast, and lightweight web framework for building APIs with Python. It is designed to be easy to use and provide high performance while requiring minimal code.

One of the critical features of Fast API is its automatic data validation and documentation. Fast API uses Python-type annotations to generate documentation and validate incoming requests automatically. It can save developers significant time and reduce the probability of errors.

17.1.4 Fast API design pattern

Fast API follows the Model-View-Controller (MVC) design pattern, a widely used architectural pattern for building web applications. The MVC pattern separates the server side into three main components: the model, the view, and the controller.

In Fast API, the model represents the data and business logic of the application, while the view is responsible for presenting the data to the user. The controller acts as an intermediator between the model and the view, handling user input and updating the model accordingly.

By following the MVC pattern, Fast API allows developers to separate concerns and structure their code in a modular and scalable way. It can make it easier to maintain and update the application over time, as changes to one component will not impact the others.

17.1.5 Django Rest Framework

On the other hand, we also use Django on our server side; Django is a widely used web framework that allows developers to build full-stack web applications with Python. It comes with a powerful administrative interface known as the Django Admin Panel, which provides developers with a convenient way to manage their application's data. Some features of the Django admin panel:

- Built-in support for user authentication and permissions allows secure access control to sensitive data.
- Customizable interface with support for multiple languages and time zones, making it easy to match the look and feel of your application.
- Advanced search and filtering capabilities, making it easy to manage data.
- Built-in support for common data types, such as dates, times, and file uploads, streamlining the process of managing application data.
- Custom views and templates can be created to extend the functionality of the admin panel, providing developers with greater flexibility and control over their application's data management.

17.2 Chair and Sensor's data models

These models work with the way the data from the medical sensors store and also store the information about the chairs that are used or not used right. It could receive the data from the endpoint of the chair and store it in the database or use another endpoint to get specific data and retrieve it.

Chair Model:

The Chair model stores the information about the wheelchair that is used by the patient. It also has a relation with the SensorData model that stores the data coming from medical sensors to make it easy to access these data. Also, it represents a chair object in the system. It has the following attributes and relationships:

- **id (integer, primary key)**: unique identifier for the chair.
- **barcode (integer, unique)**: unique identifier for the chair that is indexed for quick lookups.
- **password (string, not nullable)**: password associated with the chair.
- **available (boolean, default=True)**: indicates whether the chair is currently available or not.

Chair	
PK	chair_id
	barcode
	password
	available

Figure 74

17.2.1 SensorData Model:

The SensorData model represents the data that comes from the medical sensors that send the current health status of the patient and store it frequently. Also, it represents the sensor data associated with a Chair object. It has the following attributes and relationships:

- **id (integer, primary key)**: unique identifier for the sensor data.
- **temperature (float, not nullable)**: temperature reading from the sensor associated with the chair.
- **pulse_rate (float, not nullable)**: pulse rate reading from the sensor associated with the chair.
- **oximeter (float, not nullable)**: oximeter reading from the sensor associated with the chair.
- **date (date, default=func.current_date())**: the date on which the sensor data was created.
- **chair_id (integer, foreign key to Chair table)**: specifies which chair the sensor data is associated with.

Sensor_data	
PK	sensor_id
FK1	chair_id
	temperature
	pulse_rate
	oximeter
	date

Figure 75

17.2.2 Relationship between Chair and SensorData models:

The Chair model and the SensorData model have a one-to-many relationship. This means that each chair can have multiple sensor data entries associated with it, while each sensor data entry belongs to a single chair.

In the Chair model, the relationship is defined using the sensor_data attribute, which is a list-like object of SensorData objects. It is specified with the relationship function and the "SensorData" model name is provided as the argument. The back_populates parameter is set to "chair", indicating that the inverse relationship is defined in the SensorData model using the chair attribute.

```
sensor_data = relationship("SensorData", back_populates="chair")
```

In the SensorData model, the relationship is defined using the chair attribute, which represents the chair associated with the sensor data entry. It is specified with the relationship function and the "Chair" model name is provided as the argument. The back_populates parameter is set to "sensor_data", indicating the inverse relationship is defined in the Chair model using the sensor_data attribute.

```
chair = relationship("Chair", back_populates="sensor_data")
```

This bidirectional relationship allows you to access the sensor data entries associated with a chair using chair.sensor_data, and the chair associated with a sensor data entry using sensor_data.chair.

Overall, the models represent a system where chairs capture sensor data, and each chair can have multiple sensor data entries associated with it. The relationships between the models facilitate the retrieval and navigation of data in the system.

17.3 Schemas

We create schemas also that specify the data coming from the database and use to specify the request-response pattern

17.3.1 ChairRegistration Schema:

The ChairRegistration schema is used when registering a chair to access its data for display in the application. It has the following fields:

- chair_id: An integer field representing the ID of the chair being registered.
- password: A string field representing the password associated with the chair.

```
class ChairRegistration(BaseModel):
    chair_id: int
    password: str

    class Config:
        orm_mode = True
        schema_extra = {
            "example": {
                "chair_id": 55,
                "password": "mypassword",
            }
        }
```

Figure 76

The ChairRegistration schema is a subclass of BaseModel from the pydantic library. It serves as a blueprint for validating and serializing the request body when registering a chair. The Config inner class is used to configure the schema. Here, orm_mode is set to True to allow the schema to be used with SQLAlchemy models. The

schema_extra attribute provides an example JSON payload for the schema, which can be useful for generating documentation.

17.3.2 GetChairData Schema:

The GetChairData schema is used to represent the sensor data retrieved from the database. It has the following fields:

- temperature: A float field representing the temperature value captured by the chair's sensor.
- oximeter: A float field representing the oximeter value captured by the chair's sensor.
- pulse_rate: A float field representing the pulse rate value captured by the chair's sensor.

```
class GetChairData(BaseModel):
    temperature: float
    oximeter: float
    pulse_rate: float
```

Figure 77

Similar to the ChairRegistration schema, the GetChairData schema is a subclass of BaseModel and provides the Config inner class for configuration. It enables validation and serialization of the request/response bodies.

17.3.3 ReadChairData Schema:

The ReadChairData schema extends the GetChairData schema and includes an additional field, chair_id. It represents the sensor data and the associated chair ID received from the chair.

```
class ReadChairData(GetChairData):
    chair_id: int
```

Figure 78

This schema is used when creating a route to handle the upcoming sensor data. It allows for validating the incoming request body and extracting the sensor data and chair ID.

17.3.4 GetChairLocation Schema:

The GetChairLocation schema represents the chair's location data retrieved from the database. It has the following fields:

- latitude: A float field representing the latitude coordinate of the chair's location.
- longitude: A float field representing the longitude coordinate of the chair's location.

```
class GetChairLocation(BaseModel):
    latitude: float
    longitude: float
```

Figure 79

Similar to the other schemas, GetChairLocation is a subclass of BaseModel and includes the Config inner class for configuration.

17.3.5 StoreChairLocation Schema:

The StoreChairLocation schema extends the GetChairLocation schema and includes an additional field, chair_id. It represents the chair's location data along with the associated chair ID.

```python
class StoreChairLocation(GetChairLocation):
    chair_id: int
```

Figure 80

This schema is used when creating a route to store/update the chair's location data. It allows for validating the incoming request body and extracting the location data and chair ID.

17.4 Chair and Sensor's data view

The view here is the endpoint we use to connect with the chair to get the sensors read and the endpoint that use for getting these data

17.4.1 chair_registration Route:

This route is used to register a new chair in the database. It accepts a POST request to the `/chair/signup` endpoint. The request body should include a `ChairRegistration` object.

```python
@router.post("/signup", status_code=status.HTTP_201_CREATED)
def chair_registration(
    chair: schemas.ChairRegistration, db: Session = Depends(database.get_db)
):
    return crud.chair_signup(chair=chair, db=db)


# * Caregiver login to the chair to access sensor's data
@router.post("/login", response_model=Token, status_code=status.HTTP_200_OK)
def chair_login(
    chair: schemas.ChairRegistration,
    db: Session = Depends(database.get_db),
    authorize: AuthJWT = Depends(),
):
    return crud.chair_login(chair=chair, db=db, authorize=authorize)
```

Figure 81

`chair`: The `ChairRegistration` object contains the `chair_id` and `password` fields.

The route calls the `chair_signup` function from the `crud` module, passing the `chair` object and the database session (`db`) as parameters. The function handles the logic to create and save the chair registration data in the database.

To make it suitable for documentation, you can include the following details:

- Endpoint: `/chair/signup`
- Method: POST

- Request body example: Include an example JSON payload that matches the structure of the `ChairRegistration` schema.

17.4.2 chair_login Route:

This route is used for a caregiver to log in to a chair and access sensor data. It accepts a POST request to the `/chair/login` endpoint. The request body should include a `ChairRegistration` object.

```python
@router.post("/login", response_model=Token, status_code=status.HTTP_200_OK)
def chair_login(
    chair: schemas.ChairRegistration,
    db: Session = Depends(database.get_db),
    authorize: AuthJWT = Depends(),
):
    return crud.chair_login(chair=chair, db=db, authorize=authorize)
```

Figure 82

`chair`: The `ChairRegistration` object contains the `chair_id` and `password` fields.

The route calls the `chair_login` function from the `crud` module, passing the `chair`, database session (`db`), and the `authorize` object as parameters. The function handles the login logic, including authentication and generating a JWT token.

To make it suitable for documentation, you can include the following details:

- Endpoint: `/chair/login`
- Method: POST
- Request body example: Include an example JSON payload that matches the structure of the `ChairRegistration` schema.
- Response: The response is expected to include a JWT token, which can be specified in the `Token` schema.

17.4.3 read_new_chair_data Route:

This route is used to store new chair data in the database. It accepts a POST request to the `/chair/data` endpoint. The request body should include a `GetChairData` object.

```python
@router.post("/data", status_code=status.HTTP_201_CREATED)
def read_new_chair_data(
    data: schemas.GetChairData,
    db: Session = Depends(database.get_db),
    authorize: AuthJWT = Depends(),
):
    try:
        authorize.jwt_required()
    except:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid Token"
        )
    current_chair = authorize.get_jwt_subject()
    return crud.store_chair_data(data=data, db=db, chair_id=current_chair)
```

`data`: The `GetChairData` object contains the `temperature`, `oximeter`, and `pulse_rate` fields.

The route calls the `store_chair_data` function from the `crud` module, passing the `data`, database session (`db`), and the `authorize` object as parameters. The function handles storing the chair data in the database, while also checking for authorization using the JWT token.

To make it suitable for documentation, you can include the following details:

- Endpoint: `/chair/data`
- Method: POST
- Request body example: Include an example JSON payload that matches the structure of the `GetChairData` schema.
- Authorization: Mention that the request requires a valid JWT token in the `Authorization` header.

17.4.4 get_chair_data Route:

This route is used to retrieve the last chair data for a specific patient. It accepts a GET request to the `/chair/data/{chair_id}` endpoint, where `{chair_id}` is the ID of the chair.

```
@router.get(
    "/data/{chair_id}",
    response_model=schemas.GetChairData,
    status_code=status.HTTP_200_OK,
)
def get_chair_data(
    chair_id: int,
    db: Session = Depends(database.get_db),
):
    return crud.get_chair_data(chair_id=chair_id, db=db)
```

Figure 83

`chair_id`: The ID of the chair for which the data is requested.

The route calls the `get_chair_data` function from the `crud` module, passing the `chair_id` and database session (`db`) as parameters. The function retrieves the last recorded chair data for the specified chair ID.

To make it suitable for documentation, you can include the following details:

- Endpoint: `/chair/data/{chair_id}`
- Method: GET
- Path parameter: `{chair_id}` represents the ID of the chair for which data is requested.
- Response: The response will include the retrieved chair data, which matches the structure of the `GetChairData` schema.

17.4.5 recieve_location Route:

This route is used to receive and store the location data of a chair. It accepts a POST request to the `/chair/location` endpoint. The request body should include a `StoreChairLocation` object.

```
@router.post("/location", status_code=status.HTTP_201_CREATED)
async def recieve_location(
    location: schemas.StoreChairLocation, db: Session = Depends(database.get_db)
):
    return crud.post_chair_location(db=db, location=location)
```

`location`: The `StoreChairLocation` object contains the `latitude`, `longitude`, and `chair_id` fields.

The route calls the `post_chair_location` function from the `crud` module, passing the `location` object and the database session (`db`) as parameters. The function handles storing the chair location data in the database.

To make it suitable for documentation, you can include the following details:

- Endpoint: `/chair/location`
- Method: POST
- Request body example: Include an example JSON payload that matches the structure of the `StoreChairLocation` schema.

17.5. Chair and Sensor's Data Control

There are six crud functions that handle the functionality of the routes to store or get the data from the database correctly:

1. chair_signup
2. get_chair
3. chair_login
4. store_chair_data
5. get_chair_data
6. post_chair_location

17.5.1 chair_signup function

This function is used to register a new chair in the database. It takes a database session (db) and a ChairRegistration object (chair) as input. The ChairRegistration object contains the chair_id and password of the chair being registered.

```
def chair_signup(db: Session, chair: schemas.ChairRegistration):
```

The function first checks if the chair_id already exists in the database by querying the Chair model. If a chair with the same ID is found, it raises an HTTPException with a status code of 400 (Bad Request).

If the chair ID is unique, a new Chair object is created with the provided chair_id, hashed password (using the create_hashed_password function), and available=True. The new chair is added to the session, committed to the database, and refreshed to obtain the updated database-generated attributes.
Finally, the function returns a dictionary with a detail indicating successful chair registration.

17.5.2 get_chair function

This function is used to validate the login data for a specific chair. It takes a database session (db) and a ChairRegistration object (chair) as input. The ChairRegistration object contains the chair_id and password for the chair login.

```
def get_chair(db: Session, chair: schemas.ChairRegistration):
```

The function queries the Chair model with the provided chair_id to retrieve the corresponding chair object from the database. If a chair is found and the provided password matches the hashed password stored in the database (using the verify_password function), it returns a dictionary with a detail indicating successful chair login.

17.5.3 chair_login function

This function is used to log in to a specific chair. It takes a database session (db), a ChairRegistration object (chair), and an AuthJWT object (authorize) as input. The ChairRegistration object contains the chair_id and password for the chair login.

```
def chair_login(
    db: Session, chair: schemas.ChairRegistration, authorize: AuthJWT | None = None
):
```

The function calls the get_chair function to validate the login data. If the login is successful (i.e., a matching chair is found), it returns a dictionary with a detail indicating successful chair login.

If the authorize object is provided, it means the login is being performed for generating a JWT token. In this case, the function calls the generate_tokens function to generate and return the JWT token.

If no matching chair is found or the provided password is incorrect, the function raises an HTTPException with a status code of 400 (Bad Request).

17.5.4 store_chair_data function

This function is used to store new sensor data from a chair in the database. It takes a database session (db), a GetChairData object (data), and a chair_id as input. The GetChairData object contains the sensor data fields: temperature, oximeter, and pulse rate.

```
def store_chair_data(db: Session, data: schemas.GetChairData, chair_id: int):
```

The function checks if a chair with the given chair_id exists in the database by querying the Chair model. If no matching chair is found, it raises an HTTPException with a status code of 404 (Not Found).

If the chair exists, a new SensorData object is created with the provided sensor data fields, and the chair_id is set to the corresponding chair ID. The new data object is added to the session, committed to the database, and refreshed to obtain the updated database-generated attributes.

Finally, the function returns a dictionary with a detail indicating successful data storage.

17.5.5 get_chair_data function

This function is used to retrieve the latest sensor data for a specific chair. It takes a chair_id and a database session (db) as input.

```
def get_chair_data(chair_id: int, db: Session):
```

The function queries the Chair model with the provided chair_id to check if the chair exists in the database. If no matching chair is found, it raises an HTTPException with a status code of 404 (Not Found).

If the chair exists, the function queries the SensorData model to retrieve the latest recorded data for that chair. The data is ordered in descending order by id (assuming higher id values represent more recent data) and only the first result is fetched.

If no data is found for the chair, it raises an HTTPException with a status code of 404 (Not Found).
Finally, the function returns a dictionary containing the retrieved sensor data.

17.5.6 post_chair_location function

This function is used to store the location data of a chair in the database. It takes a database session (db) and a GetChairLocation object (location) as input. The GetChairLocation object contains the latitude, longitude, and chair_id fields representing the chair's location.

```
def post_chair_location(db: Session, location: schemas.GetChairLocation):
```

The function queries the Chair model to check if a chair with the provided chair_id exists in the database. If no matching chair is found, it raises an HTTPException with a status code of 404 (Not Found).

If the chair exists, a new Location object is created using the provided location data. The chair_id is set to the corresponding chair's ID. The new location object is added to the session, committed to the database, and refreshed to obtain the updated database-generated attributes.

Finally, the function returns a dictionary with a detail indicating successful location storage.

17.6. Patient model

The Patient model represents a patient in the system. It stores information about the patient's identity, such as their first name, last name, gender, and age. Additionally, it establishes relationships with other entities in the system.

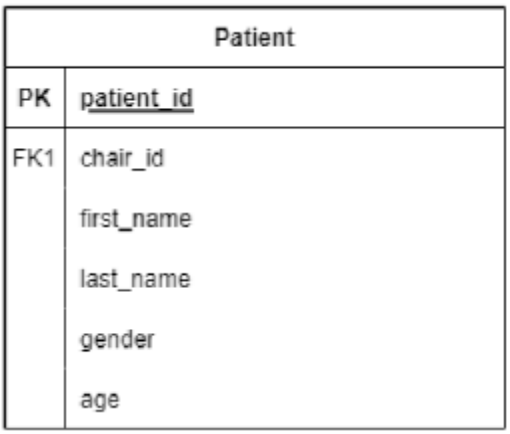


Figure 84

- **id**: An integer column that serves as the primary key for the Patient table.
- **first_name**: A string column that stores the first name of the patient with a maximum length of 50 characters.
- **last_name**: A string column that stores the last name of the patient with a maximum length of 50 characters.
- **gender**: A string column that stores the gender of the patient with a maximum length of 15 characters.
- **age**: An integer column that stores the age of the patient.
- **chair_id**: A foreign key column that establishes a one-to-one relationship
- with the Chair model. It references the id column of the Chair table.

The Patient model represents a fundamental entity in the system, capturing patient information and relationships with chairs and caregivers. It enables tracking patients, associating them with specific chairs, and establishing connections with the responsible caregivers.

17.6.1 Schemas

There are two main schemas we work with when comes to the patient:

17.6.1.1 Schema: PatientData

The PatientData schema represents the data required to create or update patient information. It inherits from BaseModel and includes the following fields:

- **first_name**: A string field that represents the first name of the patient.
- **last_name**: A string field that represents the last name of the patient.
- **gender**: A string field that represents the gender of the patient.
- **age**: An integer field that represents the age of the patient.
- **chair_id**: An integer field that represents the ID of the associated chair.

```python
class PatientData(BaseModel):
    first_name: str
    last_name: str
    gender: str
    age: int
    chair_id: int

    class Config:
        orm_mode = True
        schema_extra = {
            "example": {
                "first_name": "Mohamed",
                "last_name": "Badr",
                "gender": "male",
                "age": 23,
                "chair_id": 7,
            }
        }
```

Figure 85

The Config inner class is used to provide additional configuration for the schema. In this case, orm_mode = True is set to enable SQLAlchemy ORM mode serialization. The schema_extra attribute provides an example of the schema's structure and sample data.

17.6.1.2 Schema: PatientDataRegister

The PatientDataRegister schema extends the PatientData schema and includes an additional field for the patient's password. It inherits all the fields from PatientData and adds the following field:

- password: A string field that represents the password of the patient.

Similarly, the Config inner class provides configuration options for the schema. It includes an example that demonstrates the structure of the schema along with sample data.

These schemas define the structure and validation rules for the patient data used in requests and responses. The PatientData schema is used for creating or updating patient information, while the PatientDataRegister schema is used specifically for patient registration, including the password field. The provided examples illustrate the expected structure of the data.

17.6.2 Patient View

Route: register_patient

This route is used to register a new patient in the system. It requires authentication using a JWT token and expects the following patient information to be provided: first name, last name, gender, age, chair ID, and password.

```python
@router.post("/info", status_code=status.HTTP_201_CREATED)
def register_patient(
    patient: PatientDataRegister,
    db: Session = Depends(database.get_db),
    authorize: AuthJWT = Depends(),
):
    try:
        authorize.jwt_required()
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid Token"
        )
    caregiver_id = authorize.get_jwt_subject()
    return crud.add_new_patient(db=db, patient=patient, caregiver_id=caregiver_id)
```

Figure 86

Method: POST
Endpoint: /patient/info

Parameters:

- **patient (PatientDataRegister)**: The patient data to be registered, including first name, last name, gender, age, chair ID, and password.
- **db (Session)**: The database session.
- **authorize (AuthJWT)**: The AuthJWT object for token authorization.

Response:

- **Status code**: 201 (Created)
- **Body**: Returns the result of the add_new_patient function from the crud module, which adds the new patient to the database.

Route: track_patient

This route is used to track a patient by connecting them to a chair. It requires authentication using a JWT token and expects the chair ID and password to be provided.

```python
@router.post("/track", status_code=status.HTTP_200_OK)
async def track_patient(
    chair: ChairRegistration,
    db: Session = Depends(database.get_db),
    authorize: AuthJWT = Depends(),
):
    try:
        authorize.jwt_required()
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid Token"
        )
    caregiver_id = authorize.get_jwt_subject()
    return crud.connect_patient(db=db, chair=chair, caregiver_id=caregiver_id)
```

Figure 87

Method: POST
Endpoint: /patient/track

Parameters:

- **chair (ChairRegistration)**: The chair data, including chair ID and password.
- **db (Session)**: The database session.
- **authorize (AuthJWT)**: The AuthJWT object for token authorization.

Response:

- **Status code:** 200 (OK)
- **Body:** Returns the result of the connect_patient function from the crud module, which establishes the connection between the patient and the chair.

Route: get_patient_data

This route is used to retrieve patient information based on the chair ID. It expects the chair ID to be provided as a path parameter.

```python
@router.get(
    "/info/{chair_id}",
    status_code=status.HTTP_200_OK,
    response_model=PatientData,
)
def get_patient_data(chair_id: int, db: Session = Depends(database.get_db)):
    return crud.patient_info(chair_id=chair_id, db=db)
```

Figure 88

Method: GET
Endpoint: /patient/info/{chair_id}

Parameters:

- **chair_id (int):** The ID of the chair associated with the patient.
- **db (Session)**: The database session.

Response:

- **Status code:** 200 (OK)
- **Body:** Returns the result of the patient_info function from the crud module, which retrieves the patient information based on the chair ID.

Route: update_chair

This route is used to update the chair associated with a patient. It expects the current chair ID and the new chair data, including chair ID and password, to be provided.

```python
@router.put("/chair-update/{chair_id}", status_code=status.HTTP_200_OK)
def update_chair(
    chair_id: int,
    new_chair: ChairRegistration,
    db: Session = Depends(database.get_db),
):
    return crud.update_patient_chair(
        current_chair_id=chair_id, new_chair=new_chair, db=db
    )
```

Figure 89

Method: PUT
Endpoint: /patient/chair-update/{chair_id}

Parameters:

- **chair_id (int):** The ID of the current chair associated with the patient.
- **new_chair (ChairRegistration):** The new chair data, including chair ID and password.
- **db (Session):** The database session.

Response:

- **Status code:** 200 (OK)
- **Body:** Returns the result of the update_patient_chair function from the crud module, which updates the chair associated with the patient.

These routes provide functionality related to patient registration, tracking, information retrieval, and chair updates. They use the functions from the crud module to interact with the database and perform the required operations. Authentication with JWT tokens is enforced using the authorize object.

17.6.3 Patient Control

Function: patient_info

Description: Retrieves patient information based on the chair ID.

```
def patient_info(chair_id: int, db: Session, update: bool | None = None):
```

Parameters:
- **chair_id (int):** The ID of the chair associated with the patient.
- **db (Session)**: The database session.
- **update (bool | None)**: Optional parameter to specify if the patient's information needs to be updated.

Raises:
- **HTTPException:** If no chair with the specified ID is found, it raises an exception with status code 404.

Returns:
- **Dict:** The patient information associated with the chair.

Function: associate_caregiver_patient

Description: Associates a caregiver with a patient.

```
def associate_caregiver_patient(db: Session, caregiver_id: int, chair_id: int):
```

Parameters:
- **db (Session):** The database session.
- **caregiver_id (int)**: The ID of the caregiver.
- **chair_id (int)**: The ID of the chair associated with the patient.

Raises:
- **HTTPException**: If either the caregiver or the patient is not found, it raises an exception with status code 404.

Returns: **None**

Function: add_new_patient

Description: Adds a new patient to the database with the associated chair.

```
def add_new_patient(db: Session, patient: PatientDataRegister, caregiver_id: int):
```

Parameters:
- **db (Session):** The database session.
- **patient (PatientDataRegister):** The patient data to be stored in the database.
- **caregiver_id (int)**: The ID of the caregiver.

Raises:
- **HTTPException:** If the chair is already in use, it raises an exception with status code 400.

Returns:
- **Dict**: A message confirming the successful registration of the patient.

Function: connect_patient

Description: Connects a patient to a chair and associates them with a caregiver.

```
def connect_patient(db: Session, chair: ChairRegistration, caregiver_id: int):
```

Parameters:

- **db (Session):** The database session.
- **chair (ChairRegistration):** The chair data, including chair ID and password.
- **caregiver_id (int):** The ID of the caregiver.

Raises:

- **HTTPException:** If the chair is not currently in use or the patient or caregiver is not found, it raises an exception with status code 404.
- **HTTPException:** If the patient is already being tracked by the caregiver, it raises an exception with status code 400.

Returns:

- **Dict:** A message confirming that the patient has been added for tracking.

Function: update_patient_chair

Description: Updates the chair associated with a patient.

```
def update_patient_chair(
    current_chair_id: int, new_chair: ChairRegistration, db: Session
):
```

Parameters:

- **current_chair_id (int):** The ID of the current chair associated with the patient.
- **new_chair (ChairRegistration):** The new chair data, including chair ID and password.
- **db (Session):** The database session.

Raises:

- **HTTPException:** If the new chair is already connected to another patient, it raises an exception with status code 400.

Returns:

- **Dict:** A message confirming the successful update of the patient's chair.

These functions provide the necessary functionality for patient registration, chair tracking, patient information retrieval, and chair updates. They interact with the database through the SQLAlchemy ORM and utilize functions from the “api.chair_api.db.crud” module for chair-related operations. The functions handle various scenarios and raise appropriate exceptions when necessary.

17.7.Care Giver and Emergency Notification Models

In this part, we defined two SQL Alchemy models: CareGiver, and Emergency Notification. These models are Python classes that map to database tables in a relational database.
The CareGiver model represents a caregiver caring for one or more patients. It has several attributes:

- Id: This is an auto-incrementing integer that serves as the primary key for the caregiver table.
- first_name: This is a string that stores the caregiver's first name.
- last_name: This is a string that stores the last name of the caregiver.
- Username: This is a string that stores the username of the caregiver.
- Email: This is a string that stores the caregiver's email address.
- Password: This is a string that stores the password of the caregiver.
- Age: This is an integer that stores the age of the caregiver.

The patient's attribute is a many-to-many relationship with a secondary table association_table. This means that a caregiver can be associated with multiple patients, and a patient can be associated with multiple caregivers. The secondary parameter specifies the name of the secondary table that holds the relationship between the CareGiver and Patient models. The back_populates parameter specifies the name of the relationship attribute on the Patient model that points back to the CareGiver model. The Emergency notification attribute is a one-to-one relationship with the Notification model. This means that each caregiver can have one notification associated with them, and each notification is associated with one caregiver. The use list parameter is set to False to indicate that this is a one-to-one relationship.
The Emergency Notification model represents a notification sent to a caregiver when a sensor detects an error drop. It has the following attributes:

- Id: This is an auto-incrementing integer that serves as the primary key for the notification table.
- Sensor: This string describes the type of sensor that triggered the notification.
- Value: This is a float that stores the value of the sensor reading.
- Date: This is a DateTime object that stores the date and time when the notification was created.
- chair_id: This is a foreign key that references the id attribute of a Chair object.
- caregiver_id: This foreign key references the id attribute of a CareGiver object.

The chair attribute is defined as a many-to-one relationship with the Chair model. Each notification is associated with one chair, which can have multiple notifications. The back_populates parameter specifies the name of the relationship attribute on the Chair model that points back to the Notification model.

The caregiver attribute is defined as a one-to-one relationship with the CareGiver model. This means that each notification is associated with one caregiver, and each caregiver can have one notification. The back_populates parameter specifies the name of the relationship attribute on the CareGiver model that points back to the Notification model.

17.8.Caregiver Schemas

In FastAPI, schemas are crucial in defining the structure and validation rules for the data sent and received by the application's endpoints. A schema is a Python class representing an object's expected structure and data types.

FastAPI uses the Pydantic library to provide schema validation. Pydantic allows you to define a schema as a Pydantic model, which provides a lightweight and intuitive way to describe the structure of an object or data structure and specify validation rules and default values.

This part defines a set of Pydantic models that specify the structure of the request and response bodies for the various endpoints in a web application. Pydantic is a library that provides data validation and serialization for Python data structures and is commonly used in web development frameworks like FastAPI.

The models defined in this code include the following:

- **Login:** A model that specifies the structure of the request body for a login endpoint. It includes fields for the user's email and password.
- **SignUpCareGiver:** A model that specifies the structure of the request body for a caregiver sign-up endpoint. It includes fields for the caregiver's first name, last name, username, email, password, and age.
- **CareGiverInfo:** A model that specifies the structure of the response body for a caregiver information endpoint. It includes fields for the caregiver's ID, first name, last name, username, email, password, and age.
- **EditProfileCareGiver:** A model that specifies the structure of the request body for a caregiver profile editing endpoint. It includes fields for the caregiver's first name, last name, username, email, password, and age.
- **CareGiverAssignment:** A model that specifies the structure of the request body for a caregiver assignment endpoint. It includes fields for the caregiver's ID and patient ID list.
- **StoreNotification:** A model that specifies the structure of the request body for a notification storage endpoint. It includes fields for the sensor name, sensor value, and chair ID associated with the notification.
- **GetNotification:** A model that specifies the structure of the response body for a notification retrieval endpoint. It includes fields for the notification ID, sensor name, sensor value, chair ID, and timestamp.

Each model includes a Config class that specifies additional configuration options for the model. In particular, the orm_mode option is set to True for all models, indicating that the models can be used to serialize and deserialize data to and from database ORM models. The schema_extra option is also set for some models, providing an example JSON payload for the model to aid in documentation and testing.

17.9.Caregiver Views

The second part of the caregiver module views, so what are the views represent?
This is where we define the HTTP endpoints of our application, which handle incoming requests and return responses. In FastAPI, we would typically define our endpoints using the @app.route decorator or the FastAPI class's routing methods with @router.route; in our project, we are using @router.route that allows us to define routes for a specific router instance, which can then be mounted onto the main application using the app.include_router() method. This approach is particularly useful as we have a large API with many endpoints, allowing us to separate our routes into smaller, more manageable modules.
First, we import the required libraries and packages from Fastapi and Python for setting up the necessary modules and classes to handle requests related to caregivers, including connecting to the database, defining the structure of the data, and managing authentication with JWT tokens. The router instance is then created to handle all requests related to caregivers with the appropriate tag and prefix.

```
router = APIRouter tags= "caregiver"    prefix="/caregiver"
```

This sets the tags parameter to a list containing the string "caregiver," indicating that the routes defined within this router are related to caregivers, and this tag will include all of the routes related to the caregiver in the documentation of our API. It also sets the prefix parameter to "/caregiver," meaning that all routes defined within this router will have "/caregiver" added to the beginning of their paths. For example, if you represent a route with the path "/login," it will be accessible at "/caregiver/login" when mounted on the main application.

Figure 90

As shown in the figure, we have defined several endpoints, each for a particular purpose on our server side; let us explain.

17.10.Caregiver Authentication Endpoints

1 - Sign Up Caregiver Endpoint
This endpoint is a POST request at the path /signup and is responsible for registering a new caregiver. It takes a JSON payload containing the caregiver's information and returns a JWT token in the response. The response_model parameter is set to Token, indicating that the response body should be a JSON object that matches the Token schema. The status_code parameter is set to status.HTTP_201_CREATED, indicating that the HTTP status code of the response should be 201 Created. This endpoint calls the signup_caregiver function from the crud module to handle the registration logic.

2 – Login Caregiver Endpoint
This endpoint is a POST request at the path /login and authenticates a caregiver. It takes a JSON payload containing the caregiver's login information and returns a JWT token in the response. This endpoint calls the login_caregiver function from the crud module to handle the authentication logic.

3 - Caregiver Profile Endpoint

This endpoint is a GET request at the path /info and is responsible for returning information about the currently authenticated caregiver. This endpoint calls the caregiver_info function from the crud module to retrieve the caregiver's information.

4 - Caregiver Updating Profile Endpoint

This endpoint is a PUT request at the path /update/{caregiver_id} and is responsible for updating the information of a caregiver. This endpoint calls the update_caregiver function from the crud module to handle the update logic.

17.11.Caregiver Assigning Endpoints

5 – Assigning, remove, update Patients
This endpoint is a PUT request at the path /assign-patients and is responsible for assigning one or more patients to a caregiver. It takes a JSON payload containing the caregiver ID and a list of patient IDs and updates the database to assign those patients to the specified caregiver. This endpoint calls the db.query function to retrieve the caregiver and patients from the database and then updates the caregiver.patients attribute to assign the patients to the caregiver.

17.12.Assigning Patients Endpoint

6- This endpoint is a GET request at the path /assigned-patients and retrieves a list of all patients assigned to a caregiver. It retrieves all patients from the database and iterates over each patient to check which caregivers they are set to. For each assigned patient, it creates a dictionary containing the caregiver ID, patient name, patient ID, and chair barcode ID and appends it to a list of assigned patients. The final response is a JSON payload containing the list of assigned patients.

17.13.Caregiver Emergency Notification Endpoints

7- Posting Notification Endpoint
This endpoint is a POST request at the path /notification and is responsible for creating a new notification for a caregiver. It takes in a JSON payload containing the notification details and creates a new notification in the database. This endpoint calls the create_notification function from the crud module to handle the creation logic.

8- Getting All Notifications Endpoint
This endpoint is a GET request at the path /notification and is responsible for retrieving a list of notifications for a caregiver. It retrieves all notifications for the authenticated caregiver from the database and returns them in a JSON payload. This endpoint calls the get_notification function from the crud module to handle the retrieval logic.

9- Deleting Specific Notification Endpoint
This endpoint is a DELETE request at the path /notification/{notification_id} and is responsible for deleting a notification for a caregiver. It takes in a notification ID in the URL and deletes the corresponding notification from the database. This endpoint calls the delete_notification function from the crud module to handle the deletion logic.

17.14.Care Giver and Emergency Notification Controllers

Each function in this part can be thought of as a "controller" in the context of a web application. In web development, controllers are responsible for handling incoming HTTP requests, processing them, and returning the appropriate response. In this case, the functions defined in this code handle different types of requests related to caregivers and their notifications. For example, the signup_caregiver function handles requests to create a new caregiver account, while the get_notification function handles requests to retrieve a caregiver's notifications.
These functions are implemented using the FastAPI framework, which is a modern web framework for building APIs with Python. FastAPI provides a simple way to define API endpoints using Python functions decorated with the appropriate HTTP method (e.g. @app.post, @app.get, etc.). This part also uses FastAPI JWT Auth to handle authentication and authorization for caregivers. This package provides decorators that can be used to protect certain API endpoints and ensure that only authenticated users can access them.
We define several functions that handle the CRUD (Create, Read, Update, and Delete) operations for a caregiver API.

The signup_caregiver function takes three arguments: a database session (db), an instance of the AuthJWT class (authorize), and a SignUpCareGiver object from the schemas module. This function checks if the caregiver's email and username already exist in the database using SQLAlchemy. If either email or username already exists, it raises an HTTPException with a status code of 400 and a message indicating that the email or username is already taken.If the email and username are both unique, the function hashes the caregiver's password using the create_hashed_password function from the db.crud module, creates a new CareGiver model instance using the data from the SignUpCareGiver object, saves it to the database using the SQLAlchemy session, and commits the transaction. The function then generates a JWT token for the caregiver using the generate_tokens function from the db.crud module and returns it.

The get_caregiver function takes three arguments: a database session (db), an instance of the AuthJWT class (authorize), and a Login object from the schemas module.This function

retrieves the caregiver from the database based on their email address using the SQLAlchemy query, and then checks if the hashed password from the Login object matches the hashed password stored in the database for the caregiver using the verify_password function from the db.crud module. If the passwords match, the function generates a JWT token for the caregiver using the generate_tokens function from the db.crud module and returns it. If the passwords do not match, the function returns None.

The login_caregiver function takes three arguments: a database session (db), an instance of the AuthJWT class (authorize), and a Login object from the schemas module.This function calls the get_caregiver function to retrieve the caregiver's JWT token based on their email and password. If the JWT token is found, the function returns it. If the JWT token is not found, the function raises an HTTPException with a status code of 400 and a message indicating that the email or password is invalid.

The caregiver_info function takes two arguments: a database session (db) and a caregiver ID.This function retrieves the caregiver's information from the database based on their ID, using the SQLAlchemy query, and returns a CareGiverInfo object from the schemas module.

The update_caregiver function takes four arguments: a database session (db), an instance of the AuthJWT class (authorize), a caregiver ID, and an EditProfileCareGiver object from the schemas module.This function checks if the caregiver with the given ID exists in the database using the SQLAlchemy query. If the caregiver does not exist, the function raises an HTTPException with a status code of 404 and a message indicating that the caregiver was not found. If the caregiver exists, the function updates their attributes with the new values from the EditProfileCareGiver object. If the password is provided, the function hashes it using the create_hashed_password function from the db.crud module. The function then commits the changes to the database and returns the updated CareGiver model instance.

The get_notification function takes two arguments: a database session (db) and a caregiver ID.This function retrieves all notifications for the given caregiver from the database using the SQLAlchemy query, and then updates each notification's chair_id attribute with the parcode of the corresponding chair using another SQLAlchemy query.

The create_notification function takes four arguments: a database session (db), a StoreNotification object from the schemas module, a caregiver ID, and a chair ID.This function checks if the given chair ID exists in the database using the SQLAlchemy query. If the chair does not exist, the function raises an HTTPException with a status code of 404 and a message indicating that the chair was not found.If the chair exists, the function creates a new Notification model instance using the data from the StoreNotification object, the caregiver ID, and the chair ID. The function then saves the new notification to the database using the SQLAlchemy session, commits the transaction, and returns a message indicating that the notification was stored successfully.

The delete_notification function takes two arguments: a database session (db) and a notification ID.This function retrieves the notification from the database based on its ID using the SQLAlchemy query. If the notification does not exist, the function raises an HTTPException with a status code of 404 and a messageindicating that the notification was not found. If the notification exists, the function deletes it from the database using the SQLAlchemy session, commits the transaction, and returns a message indicating that the notification was deleted successfully

Cloud

18.Cloud Computing

18.1.What is cloud, and what is it different from cloud computing?

The Cloud refers to a Network or the Internet. In other words, a Cloud is something present at a remote location. The Cloud can provide services over the network, i.e., on public or private networks, i.e., WAN, LAN, or VPN. Applications such as e-mail, web conferencing, and customer relationship management (CRM) all run in Cloud.

Cloud Computing is a process of manipulating, configuring, and accessing applications online. It offers online data storage, infrastructure, and application.

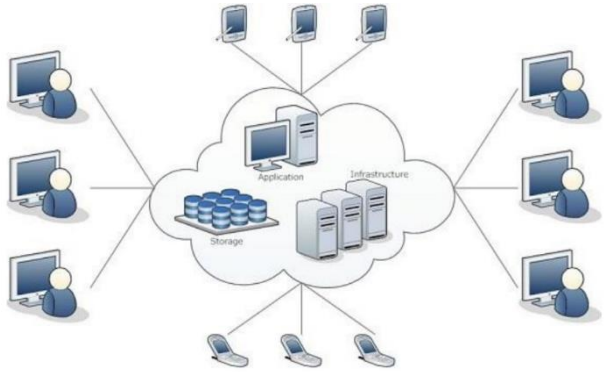


Figure 91

18.2.Service Models

Service Models are the reference models on which Cloud Computing is based. These can be categorized into three primary service models as listed below:

1. Infrastructure as a Service (IaaS) Infrastructure such as Virtual Machines, servers.
2. Platform as a Service (PaaS) Platform such as Database, Web Server.
3. Software as a Service (SaaS) Applications such as Email, CRM, Games.

18.3.Benefits:

Cloud computing allows users to quickly scale their resources up or down as needed without significant upfront investments in hardware and infrastructure. Users can quickly adjust their computing resources to meet changing business needs.

Cost savings: Cloud computing can help users save significant costs by eliminating the need for on-premises hardware, reducing maintenance and upgrade costs, and providing pay-as-you-go pricing models that allow users to only pay for the resources they use.

Accessibility: Cloud computing enables users to access resources and applications from anywhere, if they are connected to the internet. This allows for more flexible work arrangements and can help improve productivity.

Reliability: Cloud providers typically offer high reliability and uptime, often backed by service level agreements (SLAs) that guarantee a certain level of availability for their services.

Security: Cloud providers typically offer robust security measures and compliance certifications, which can help users secure their data and applications more effectively than they might be able to do on their own. Cloud providers also typically provide regular security updates and patches to ensure their services remain secure.

18.4.AWS

AWS means Amazon Web Services, which is used by millions, and to get the answer to this question, we must know that AWS is a cloud provider. It is a safe cloud services platform that offers almost all a business requires to develop sophisticated applications with reliability, scalability, and flexibility. It is a billing model generally called “pay-as-you-go,” with no upfront or capital cost. Amazon offers almost 100 services based on demand, and the list has increased daily. Operation is almost immediate, and it is accessible with reduced setup.

18.5.AWS Services

Some of the most commonly used Amazon Web Services (AWS) services:

Amazon EC2 (Elastic Compute Cloud): Provides scalable computing capacity in the cloud, allowing users to launch and manage virtual servers (instances) as needed. **Amazon S3 (Simple Storage Service):** Provides scalable object storage in the cloud, allowing users to store and retrieve data from anywhere with low latency and high durability.
Amazon RDS (Relational Database Service): Provides a managed database service in the cloud, allowing users to set up, operate, and scale relational databases such as MySQL, PostgreSQL, Oracle, and SQL Server.

Figure 92

Amazon DynamoDB: Provides a managed NoSQL database service in the cloud, allowing users to store and retrieve any amount of data with low latency and high scalability.
Amazon Lambda: Provides a serverless computing service that allows users to execute code without needing to provision or manage servers. They only need to pay for the compute time used by their applications.

18.6.AWS EC2 Instance

An Amazon EC2 (Elastic Compute Cloud) instance is a virtual server that can be launched in the cloud. It provides compute capacity in the form of virtual machines (VMs) that can be configured with different operating system for our purpose we use the last LTS version of ubuntu, applications, and software stacks.

EC2 instances can be launched, stopped, and terminated as needed, providing users with scalability and flexibility in managing their computing resources. Think of an EC2 instance as a remote computer that you can access and use to run your applications and services in the cloud.

For this project, we utilized Amazon EC2 instances to host and run our application in the cloud, allowing us to manage our computing resources while providing reliable and scalable infrastructure easily and flexibly.

18.7.Deployment With EC2 Instance

As mentioned above, we use an ec2 instance to deploy our project into it; in this case, we will set up our virtual machine and do some steps:

1 - Launch an EC2 instance:

- First, go to the EC2 console and launch an instance with the desired name (we will launch two instances, one with FastAPI and the second For Django).

- Choosing the Ubuntu AMI, we want to use (e.g., Ubuntu Server 22.04 LTS) with SSD volume that fits our requirements. In our case, we use t3.micro as it's free for the first 12 months.

- Follow the prompts to configure the instance details, storage, and security groups.

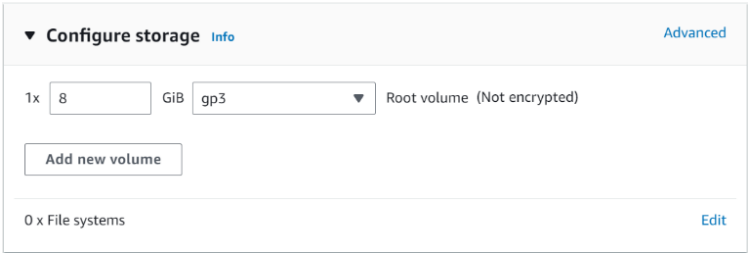
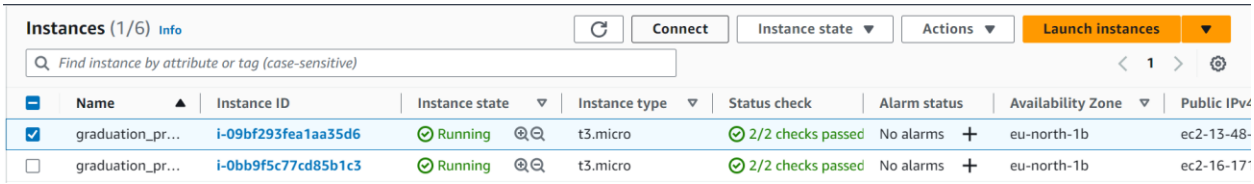


Figure 93

2 - Connect to the instance:

Once launched, we can use SSH, which helps to connect the instance. If we are on a Windows machine, we can use a tool like PuTTY to connect to the instance. If we are on a Mac or Linux machine, After the instance state is running with the status check is 2/2 checks passed, connect to the instance using the Terminal and the SSH command.



3 - Update the instance:
Once we are connected to the instance, we will run the following command to update the package list and install any available updates:
sudo apt-get update && sudo apt-get upgrade

in the following screen we are now connected to the instance, and we have two IPs: Public IP for which allows the instance to be accessed from the internet.

The public IP address can be used to connect to the instance from

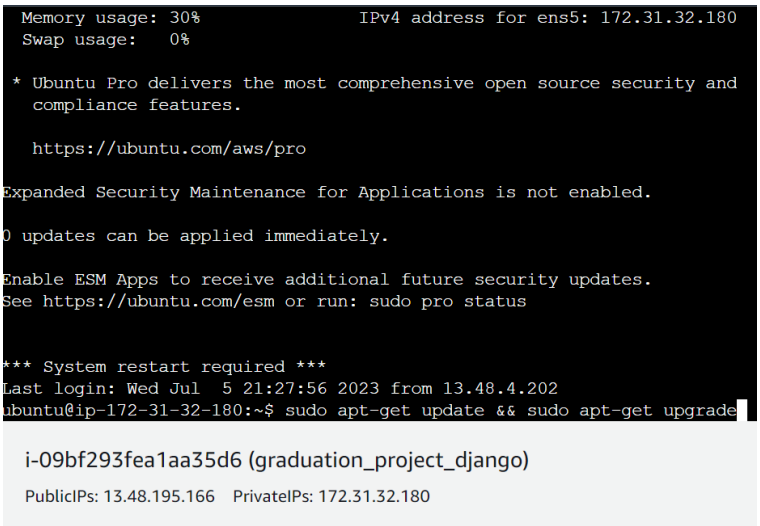


Figure 94

anywhere on the internet. Private IP address: Every EC2 instance is also assigned a private IP address, which is used for communication within the VPC (Virtual Private Cloud) network. Private IP addresses are not accessible from the internet and are used for internal communication between instances within the same VPC. Our public IP is set to “13.48.195.166” and the Private IP is set to “172.31.32.180” .

4 - Install additional packages:
Depending on our requirements, we need to install additional packages on the instance. For example, to run a web server, we must need to install Apache or Nginx. With the following command to install a package:
sudo apt-get install <package-name>

5 - Configure the instance: We may need to configure the instance further depending on our use case. For example, we may need to set up a firewall or configure the network settings. Follow the appropriate guides and tutorials to configure the instance as needed.

In our case, we will configure and edit the input roles of the instance. Nevertheless, first, what are the inbound rules and outbound rules? The inbound rule of an EC2 instance involves receiving incoming network traffic from various sources, while the outbound rule involves sending outgoing network traffic to various destinations.

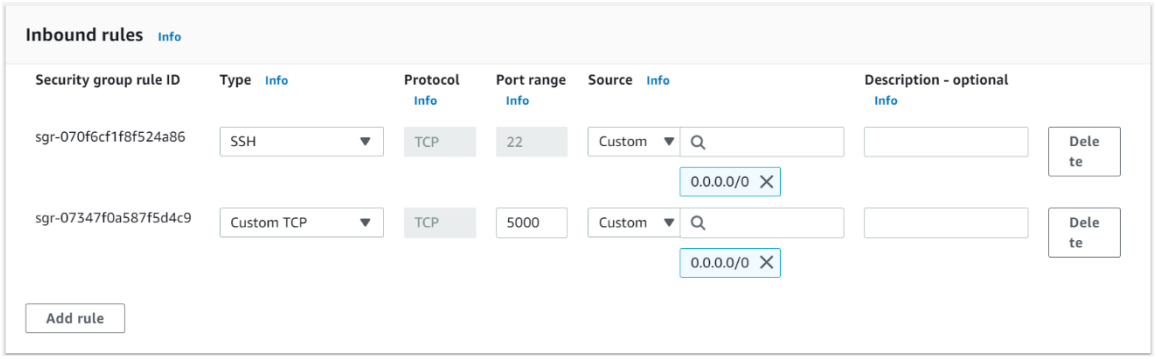


Figure 95

In the above example, the first rule is the initial rule created while creating the instance, the port range is 22, and the protocol is set to TCP (uses the TCP protocol to access the instance.). All this is set to allow incoming traffic from any IP address on the internet.

The second rule, the port range is 5000 (In Django, the default development server runs on port 8000, but we specify a different port number because we use the port 8000 with the second web server of the FastAPI), and the protocol is set to TCP. All this is set to allow incoming traffic from any IP address on the internet.

After setting all of this configuration and installing our dependencies, we can now deploy our web server to the cloud, as shown in the figure.

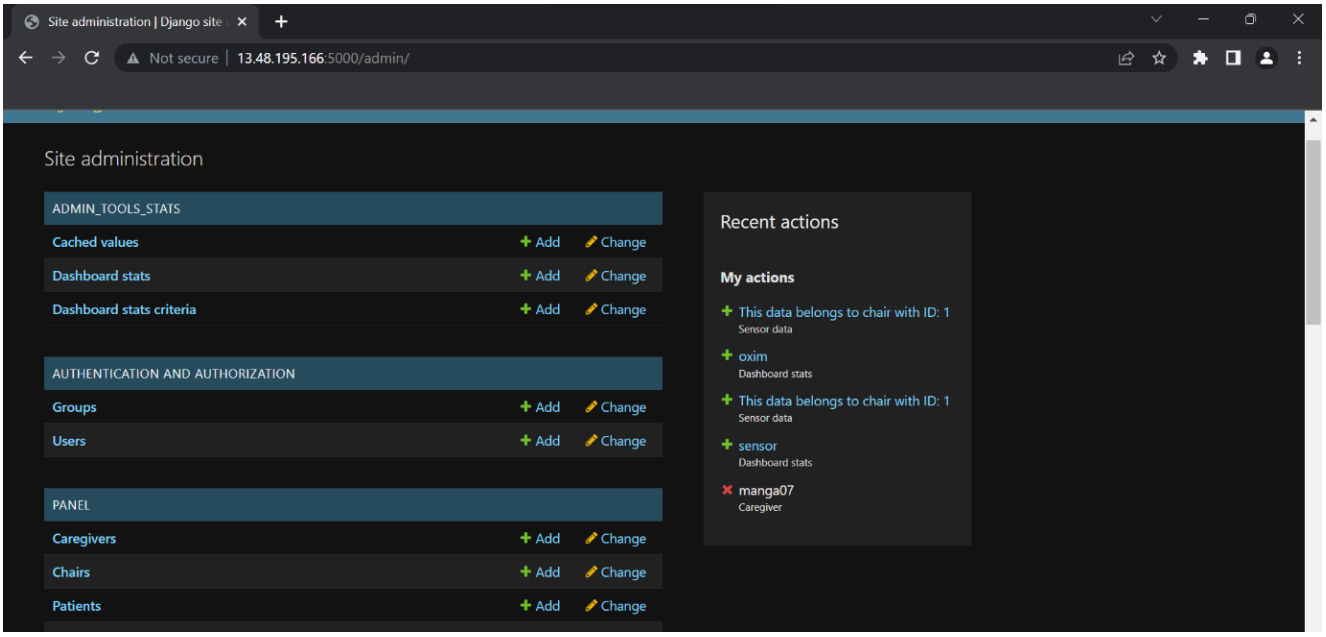


Figure 96

18.8.Amazon Relational Database Service (RDS)

After deploying our web server into the cloud, it is not a logical solution to use a local database to access, manage and control our data; the solution is provided by Amazon web services also, by making a new database not for testing development but for our production purposes so let us know what the RDS.

Amazon Relational Database Service (RDS) This service is a managed database. It is provided by Amazon Web Services (AWS). It allows users to easily create, operate, and scale a relational database in the cloud. With RDS, users can choose from several popular database engines, including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle, and Microsoft SQL Server.

RDS handles database administration tasks such as backups, software patching, and database scaling, allowing users to focus on their applications and data.

RDS provides features such as automated backups, automated software patching, point-in-time recovery, and replication for high availability and disaster recovery. Users can also scale their database instances up or down and choose from several instance types to optimize performance and cost.

18.9.Creating Database Engine (Postgres RDS)

As mentioned, we will use PostgreSQL 14.7-R1 and configure our storage with 20 GiB Allocated storage.

We will enable storage auto-scaling; the purpose of it Enabling this feature will allow the storage to increase after the specified threshold is exceeded with maximum value of 1000 GiB.

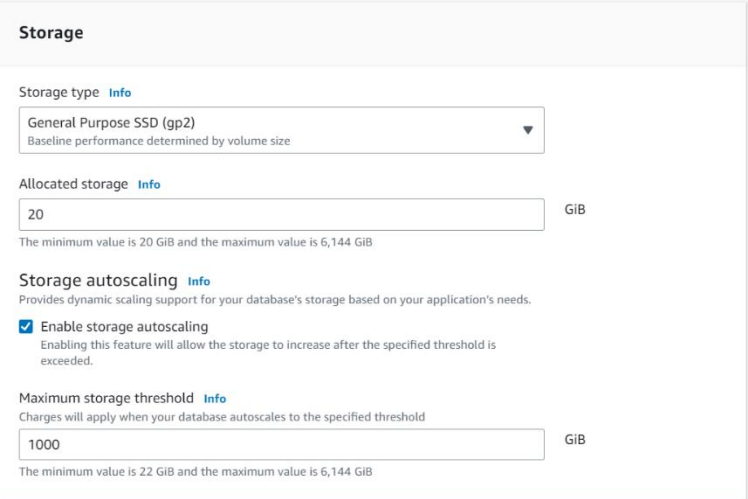


Figure 97

We will also set the DB port to 5432, as the TCP/IP protocol uses Port numbers to identify specific services running on a server. In the case of PostgreSQL, port 5432 is the default port number that PostgreSQL listens on for incoming connections.
After creating our RDS, it will give us an endpoint “gp.cdotyt702xan.eu-north-1.rds.amazonaws.com”.
The endpoint consists of several parts, including the following: gp: This is the name of the RDS instance. It is typically a unique identifier that you choose when you create the instance. cdotyt702xan: This is a randomly generated string of characters added to the instance name to ensure it is unique. eu-north-1: This is the name of the AWS region where the RDS instance is located. The EU-north-1 region is located in Stockholm, Sweden. rds.amazonaws.com: This is the domain name for the RDS service provided by Amazon Web Services. This endpoint is all we need for implementing our new remote database inside APIs.

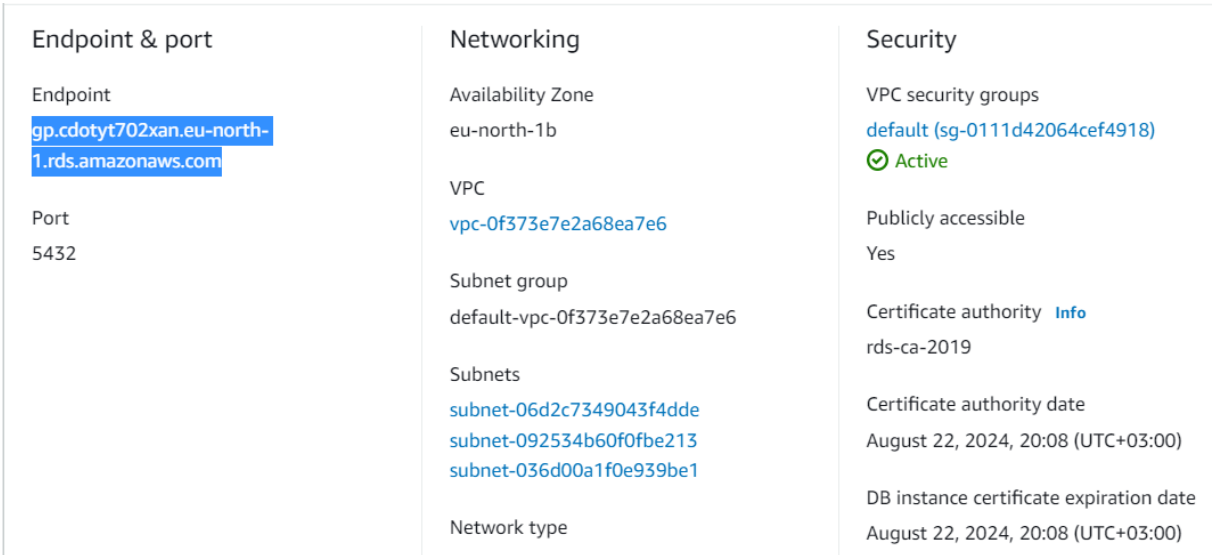


Figure 98

18.10.Connecting RDS to APIs

This is the final step of the deployment process; we will implement the endpoint of the database into Django and FastAPI.
For Django Application:

```
DATABASES  =
    'default'
        'ENGINE'   'django.db.backends.postgresql'
        'NAME'   'graduation_project'
        'USER'   'robobrain'
        'PASSWORD'   'graduation'
        'HOST'   'gp.cdotyt702xan.eu-north-1.rds.amazonaws.com'
        'PORT'   '5432'
```

For FastAPI Application:

```
SQLALCHEMY_DATABASE_URL = "postgresql://robobrain:graduation@\
    gp.cdotyt702xan.eu-north-1.rds.amazonaws.com/graduation_project"
```

18.11.Summary

This part explains the concept of cloud computing and its benefits and provides an overview of Amazon Web Services (AWS). It describes the three primary service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), and lists some commonly used AWS services. The part then explains how to deploy a project using an Amazon EC2 instance, including launching and connecting to the instance, updating packages, installing additional packages, and configuring the instance. It also introduces Amazon Relational Database Service (RDS) and explains how to create a database engine (PostgreSQL RDS) and connect it to APIs. Finally, the text provides a sample code for connecting the RDS to Django and FastAPI applications.

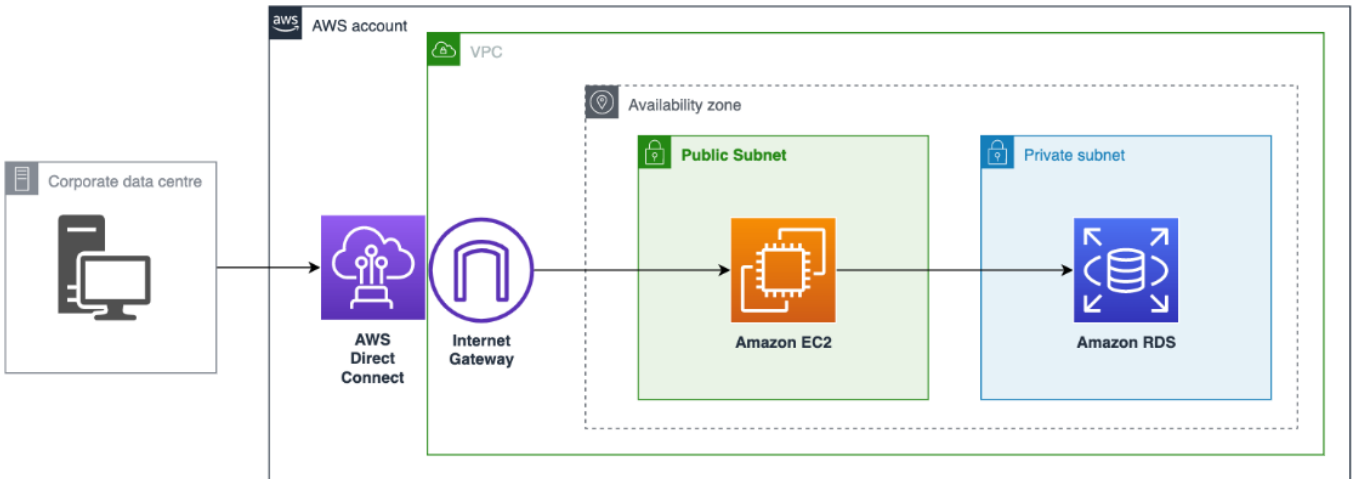


Figure 99

Mobile application

19.1 Introduction

This chapter presents an overview of Remote healthcare monitoring system, which is based on non-invasive and wearable sensors, actuators, and modern communication and information technologies, offers efficient solutions that allow people to live comfortably in any environment, being somehow protected. Furthermore, the expensive healthcare facilities are getting free to be used for intensive care patients as the preventive measures are getting at home. The remote systems can monitor very important physiological parameters of the patients in real-time, observe health conditions, assess them, and most importantly, provide feedback to their doctors or even a family member. Sensors are used in electronic medical and non-medical equipment and convert various forms of vital signs into electrical signals. Sensors can be used for life supporting implants, preventive measures, long-term monitoring of disabled or ill patients. Healthcare organizations like insurance companies need real-time, reliable, and accurate diagnostic results provided by sensor systems that can be monitored remotely, whether the patient is in a hospital, clinic, or at home. Why are these systems so comfortable and necessary to use? The first reason is that they are portable, easy to use with small sizes, and lightweight. A typical example is a Health-care Monitoring System mobile application that tracks and processes health data and location of wheelchair in addition to notify caregiver on emergencies. The main advantage of this system is that a person who is a caregiver or a nurse could track the disabled patient remotely through out their phone Another advantage of these systems is that they can monitor health conditions in real-time and all the time. People can use this application in hospitals, for home care, and any medical institution that provides health care to disabled people All this data can be processed by various sensors integrated into the systems and monitored on a flutter mobile application. This chapter attempts to provide an overview of the roadmap to create an easy , simple and professional medical mobile app.

Figure 100-health monitor mobile application

19.1.2 System overview

We thought in our wheelchair we could add more advantages to make sure that we fully covered the patient's needs as far as possible. So, we created a mobile app that can visualize the medical state of the patient with the use of hardware sensors, to make sure that the patient doesn't suffer from any medical issue. Also, we added features to work on emergency cases so that we make sure that our caregiving system introduced more security and safety to the patient. To design the whole system, we divided it into many small subs which will be gathered later, and they include: Backend, Database, Mobile app, Cloud. And the main features this system aims to provide in the app is a health tracking system that includes measuring body temperature, Heart rate, blood oxygen level, and a notification alert if there is any danger in these measurements. Also, a feature that locates the wheelchair through GPS. How this works? Users of this system will be mainly (the caregiver) and the advantage of this is to make their job in easier, when they use the application they get to check the medical state of the patient they are taking care of to make sure that there is no need for a doctor at this moment, so the hardware medical sensors controlled by raspberry pi sends the current state continuously to the backend which take the main data needed and update the previous one in the database of the system and that will update the data in the mobile application and with some logic functions we enable the features that will make sure that everything is safe. We will have to use the APIs to communicate between user and server, so we have the API and we will have an external application which has the capability to work with API which is our mobile app, and inside the system there is a two way communication of response and request, and in most of the case this data flow will be either in JSON or XML format, inside this application we will be dealing with JSON data so before sending data from API to external app we have to convert the python representation of data into JSON then we will receive JSON from the external app into API back and forth. This diagram shows that API decides which operation to select then operations are made through the database system such as POSTGRES, etc. In our system we will mainly use the two operations GET and POST to request the sensor's readings info and then system response with it. Thus every request should pass from the backend and interact with database every time an operation happens.

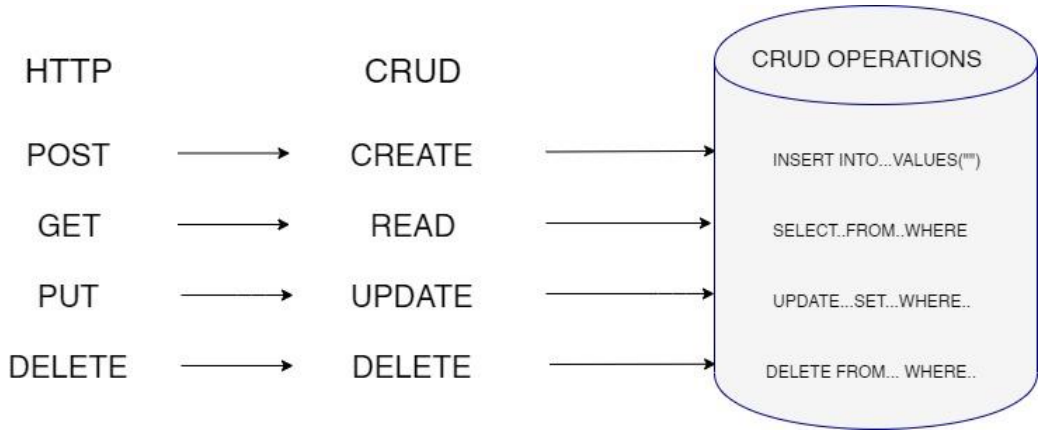


Figure 101 - crud operations

19.1.3 System Architecture Design

We will discuss each phase of this architecture more deeply but first we are going to introduce a short abstract about each subsystem we have in the caregiving module.

1- Back-end:

The main idea in our system is to read from sensors, then monitor these readings onto a mobile app, how this is possible? We connect our app to the back end of a web server that offers a web API could be (REST API). To explain this further, what we need is a web server that receives sensors' readings and sends back information. A web server in this sense is just a program which listens to incoming HTTP calls and follow these steps:

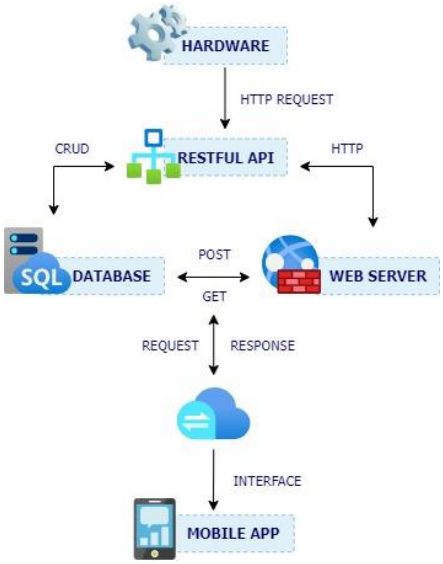


Figure 102 mobile system architecture

- The server is programmed to listen to raspberry pi with the HTTP method POST on the route /request.
- It expects the message data in a certain format, for example a JSON format.
- The data is handled by API and the server checks its database for a crud operation.
- server returns a result in a specified format (JSON).

The app then needs to send a POST call to "https://www.[URL]/request" with the specified data and wait for a response. As for the technology to use, python represents many of them that can match our requirements such as (Django, FASTAPI, …), Also keeping in minds if our app can publicly access the backend, then everybody else can so we will make sure to add some kind of authentication as well. So, the main components we are going to use are first: Raspberry pi to listen to the changes made on database and trigger the functions needed. The database will be the bridge between the application and microcontroller also using API method. The goal of our system is to combine all the topics and techniques mentioned above into one application to deliver to the caregiver. First, we connect the sensor to some type of controller and receive collection of data from it to the backend and database phase, then with integration between backend system and mobile app we can view sensor reading in the mobile app and interact with the app to benefit from other features provided.

Features of the application	
1- Show the medical condition for the patient (heart rate, temperature….)	2- Notify if there is a danger due to the current medical state of the patient.
3- View the current location of the wheelchair.	4- Track multiple wheelchairs at a time

19.2 Cross Platforms

Nowadays, companies tend to consider cross-platform solutions in their development stack mainly for three reasons:

1. Faster Development: working on a single codebase.
2. Lower Costs: maintaining a single project instead of many (N projects for N
1. platforms).
2. Consistency: the same UI and functionalities on any platform.

All those advantages are valid regardless of the framework being used. However, for a complete overview, there is the need to also consider the other side of the coin because a cross-platform approach also has some drawbacks:

1. Lower Performances: a native app can be slightly faster thanks to direct contact with the device. A cross-platform framework might produce a slower application due to a necessary bridge required to communicate with the underlying OS.
1. Slower Releases: when Google or Apple announces a major update for their OS, the maintainers of the cross-platform solution could need to release an update to enable the latest features. The developers must wait for an update of the framework, which might slow down the work.

Every framework adopts different strategies to maximize the benefits and minimize or get rid of the drawbacks. The perfect product does not exist, and very likely we will never have one, but there are some high-quality frameworks you have probably already heard of:

• Flutter: Created by Google, it uses Dart.
• React Native: Created by Facebook, it is based on JavaScript.
• Xamarin: Created by Microsoft, it uses the C#.
• Fire monkey: Created by Embarcadero, it uses Delphi.

Figure 103- different frameworks

For programming such features and developing it more easily we chose Flutter framework to develop our mobile application. And we will discuss the reason for making this choice.

19.2.1 Why Flutter?

Google tries to make the cross-platform development production-ready using the Dart programming language and the Flutter UI framework. Flutter renders everything by itself in a very good way and it does not use any intermediate bridge to communicate with the OS. It compiles directly to ARM, for mobile, or optimized JavaScript, for the web. The app you create in Flutter will look the same on any platform. That is because of the Skia graphics engine. Moreover, it is easy to both create and quickly update the UI, which with other tools is usually more cumbersome. Another thing to notice is plugins and platform channels that help utilize OS-level features without extra hassle at the top of their potential. Finally, there is Flutter for Web, allowing you to deploy your app to the web as well, and Fuchsia OS, which is still under development but will allow using Flutter for IoT purposes as well.

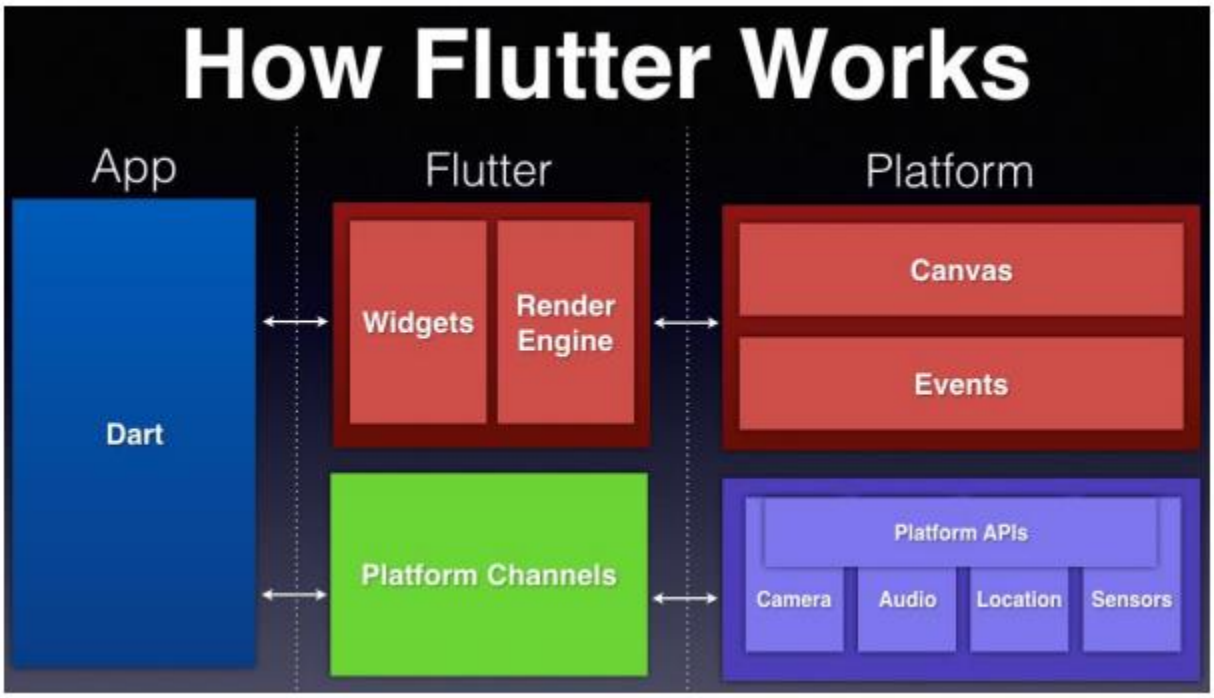


Figure 104 - how flutter works

The above figure shows how Flutter works. Widgets are rendered onto a Skia canvas and sent to the platform. The platform displays the canvas and sends events back to the app. In Flutter, UI is represented as widgets. Widgets describe how the view should look, given its current configuration and state. When the state changes, the widget rebuilds its description, and the framework compares it with the previous description to determine the minimal changes needed to update the UI. Flutter framework Advantages that it is Modern and reactive framework. It Uses Dart programming language and it is very easy to learn. It provides Fast development, Beautiful and fluid user interfaces, Huge widget catalog, Runs same UI for multiple platforms with a High performance application. Flutter's simplicity makes it a good candidate for fast development. Its customization capability and extendibility makes it even more powerful.

19.2.2 Api with Flutter

Flutter provides http package to use http resources. The http package uses await and async features and provides many high-level methods such as read, get, post, put, head, and delete methods for sending and receiving data from remote locations. These methods simplify the development of REST-based mobile applications. the core methods of the http package are as follows:

- **Read**: This method is used to read or retrieve the representation of resources. It requests the specified URL by using the get method and returns the response as `Future<String>`
- **Get:** This method requests the specified URL from the get method and returns a response as `Future<response>`. Here, the response is a class, which holds the response information.
- **Post:** This method is used to submit the data to the specified resources. It requests the specified URL by posting the given data and return a response as `Future<response>`
- **Put:** This method is utilized for update capabilities. It updates all the current representation of the target resource with the request payloads. This method requests the specified URL and returns a response as `Future<response>`
- **Head:** It is like the Get method, but without the response body.
- **Delete**: This method is used to remove all the specified resources.

The http package also provides a standard http client class that supports the persistent connection. This class is useful when a lot of requests to be made on a particular server. It should be closed properly using the `close ()` method. Otherwise, it works as a http class.

19.2.3 Authorization throughout token

When using FastAPI as a backend server and Flutter as the user interface, you can authenticate and authorize users by passing a token from FastAPI to the Flutter user interface. This token can then be included in the headers of HTTP requests made by Flutter to the backend server to access protected resources. This documentation outlines the process of passing the token from FastAPI to Flutter and demonstrates how to use the token in the header of an HTTP request. After implementing the backend server with token-based authentication and authorization implemented and install the important dependencies in flutter (http) to import the library http and dart.convert for extracting the JSON data from the database. Then we follow the steps mentioned in the next page.

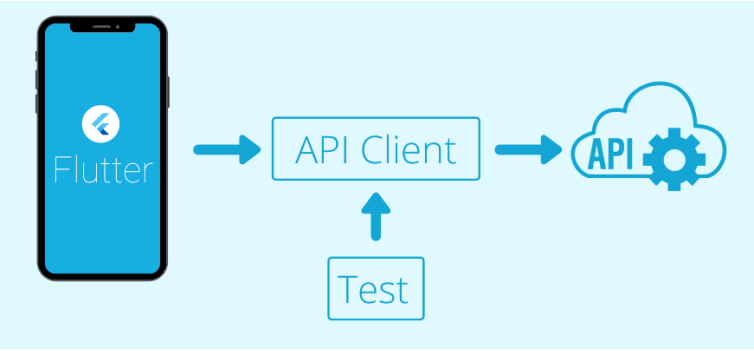
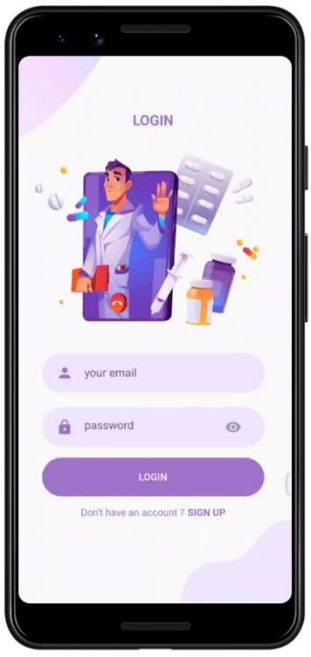
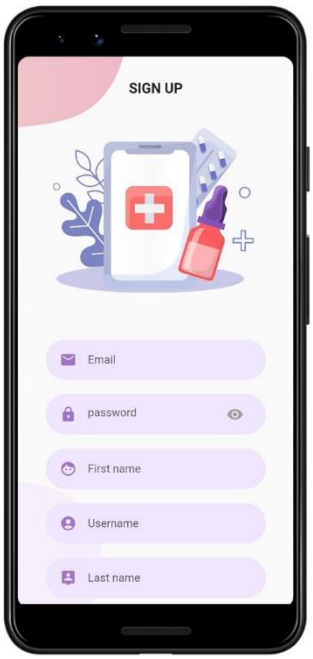


Figure 105- API Client

Passing the Token to Flutter	Using the Token in Flutter
1. Authenticate and generate a token in FastAPI using a suitable authentication mechanism like JWT (JSON Web Tokens) or OAuth. 2. return the generated token in the response body of the authentication endpoint. 3. In your FastAPI backend, create an endpoint that handles user authentication and returns the token. This endpoint could be `/auth/login` or a similar route. 4. In the response of the authentication endpoint, include the token as a JSON object or any other appropriate format.	1. Make an HTTP request to the authentication endpoint in FastAPI from your Flutter application. You can use packages like http or dio to handle the HTTP requests. 2. Extract the token from the response received in Flutter. Parse the response body to obtain the token value. 3. Store the token securely in Flutter, such as using the Flutter Secure Storage plugin or by encrypting and saving it in SharedPreferences. 4. When making subsequent API requests to your FastAPI backend, retrieve the token from storage. 5. Include the token in the headers of the HTTP request using the Authorization field with the appropriate authentication scheme.

Using post and get and token we can achieve the first thing in our application which is account registration and login.





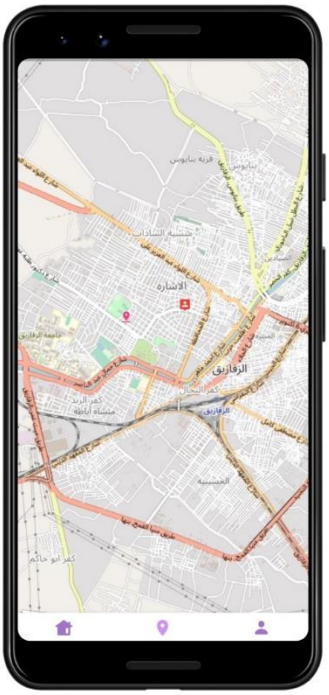
19.3.1 OpenStreetMap (OSM)

We will provide guidance on how to use OpenStreetMap (OSM) to locate the wheelchair in our mobile app. By leveraging OSM's extensive mapping data and APIs, we can enable users to find wheelchair-accessible locations and navigate to them within our healthcare app. And we will discuss the necessary steps to integrate OSM into the app and implement the wheelchair location functionality.

We first add the latest OSM package to the dependencies and then import the library. Next we follow these steps:

1. Set up the Map View:
   - Create a Flutter widget that represents the map view in the app.
   - Configure the map view using the flutter_osm package, specifying the initial position, zoom level, and other properties.
   - Initialize the map view with the OSM tile layer.

2. Fetch Wheelchair Accessible Locations:
   - Utilize the OSM Overpass API or any other suitable OSM API to fetch data about wheelchair-accessible locations. we can specify specific tags or query parameters to retrieve relevant data.
   - Make an HTTP request to the OSM API endpoint using packages like http or dio to retrieve the required data.
   - Parse the response and extract the relevant information such as coordinates, names, and wheelchair accessibility attributes.



3. Display Wheelchair Locations on the Map:
   - Create markers on the map for each wheelchair-accessible location obtained from the OSM API response.
   - Customize the markers to represent wheelchair-accessible locations appropriately, such as using specific icons or visual indicators.
   -
4. Enable User Interaction:
   - Implement user interaction features such as tapping on a marker to display additional information or navigating to the selected wheelchair-accessible location.

By using the GPS mounted on the wheelchair and sending the longitude and latitude to the backend, then in the application we use the get request to monitor the location on the map so the caregiver can track the wheelchair location remotely and even do this with multiple wheelchairs providing more safety and insurance to the patient and facilitation to the caregiver.

19.3.2 Google Maps

Google offers a good data source and mapping services including satellite imagery, Street View, and its extensive database. On the other hand, OpenStreetMap is a collaborative project that relies on contributions from volunteers worldwide who collect and edit mapping data. Google Maps data is proprietary and owned by Google and that restricts the use of its data and imposes certain terms and conditions on developers. In contrast, OpenStreetMap data is open and free. It is licensed under the Open Database License (ODbL), allowing anyone to access, use, and modify the data. But Google Maps is known for its high level of accuracy, comprehensive coverage, and detailed information. It includes features such as business listings, user reviews, real-time traffic information, and transit schedules. That's why after trying the two methods we chose the google maps one, as it provides comprehensive APIs and tools for developers to integrate its mapping services into their application, OpenStreetMap also provides APIs and tools, but they might require additional configuration and setup compared to Google's well-documented and widely-used offerings.
To set up Google Maps in a Flutter application, we need to follow these steps:

1. Obtain API Key:
   - Go to the Google Cloud Console (https://console.cloud.google.com/).
   - Create a new project or select an existing project.
   - Enable the "Maps SDK for Android" and "Maps SDK for iOS" APIs.
   - Create an API key and make sure it has access to the required APIs.
   - Take note of the generated API key, as you'll need it in the next steps.
   - Configure Android Manifest:

   - Open the android/app/src/main/AndroidManifest.xml file.
   - Add the following permissions inside the <manifest> tag:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

   - Then right before the tag application is closed we add the API

```
<meta-data  android:name="com.google.android.geo.API_KEY"
  android:value="_API_KEY" />
```

2. Configure the app/buildGradle into flutter by changing the min & compile SDK versions based on the flutter version and add enable the multiDex also add the dependencies

```
  implementation 'com.google.android.gms:play-services-maps:18.1.0'
  implementation 'com.google.android.gms:play-services-location:19.0.1'
```

3. Make sure that these repositories are added into the buildGradle of the project

```
 google()
 mavenCentral()
```

4. Lastly pub get the google maps library into the flutter project and start using it.

19.3.3 Firebase Cloud Messaging (FCM)

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution provided by Google as part of the Firebase suite of tools. It enables us to send real-time notifications and messages to users' devices efficiently. FCM supports both Android and iOS platforms, making it a reliable choice for building notification-based features in mobile and web applications.
The main idea of our caregiving system is to track the patient's medical state and notify on emergency cases instantly, we chose the FCM method for it's key features:

- **Push Notifications:** FCM allows sending push notifications to devices, providing a way to engage users even when the application is not actively being used.
- **Message Targeting:** Developers can target specific devices, user segments, or topics to ensure notifications reach the desired recipients.
- **Reliability and Scalability:** FCM is built to handle large-scale message delivery with high reliability and low latency, ensuring messages are delivered promptly.
- **Delivery Analytics**: FCM provides analytics and delivery reports to track the success of notifications and monitor engagement.
- **Integration with Other Firebase Services:** FCM seamlessly integrates with other Firebase services, such as Authentication, Cloud Functions, and Cloud Firestore, enabling comprehensive app development and user engagement capabilities.

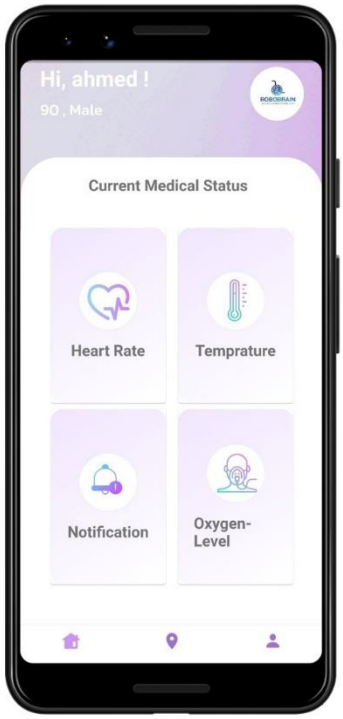
Steps to integrate the FCM into the project:

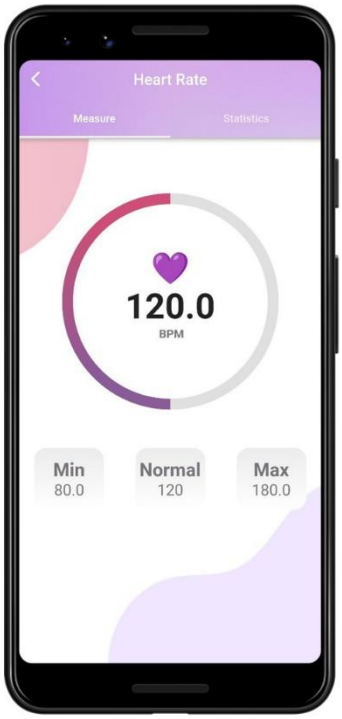
- Create a Firebase project in the Firebase Console.
- Configure the project to use FCM by adding the necessary dependencies and configurations in app's code.
- Implement client-side code to handle the receipt and display of push notifications on user devices.
- Use the Firebase Console or server-side code to send notifications to users.

The following steps in addition to local push notification feature will provide us pushing notifications when biomedical sensors reach a certain limit and therefore the caregiver could track the patient remotely, also by implementing a background task using Work Manager library in flutter we could check the whole sensors every 15 minutes (minimum available time), to track all the wheelchairs periodically. But for tracking patients even when application is terminated we will use the cloud messaging to push notification through out http post using backend server that keeps track of the sensors of all wheelchairs and triggers on emergency. For the FCM token it will be sent during the registration of the account of the caregiver. And it's stored in the database or firebase for other purposes.

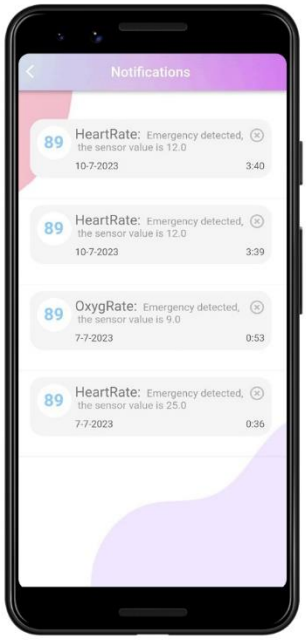
19.4 RoboBrain Application

In the home page the information about the patient (name, age and gender) are shown on top of the screen,The caregiver can view the sensors when tapping into any of the widgets shown, also can scroll through the bottom navigation bar to view the map, and profile.



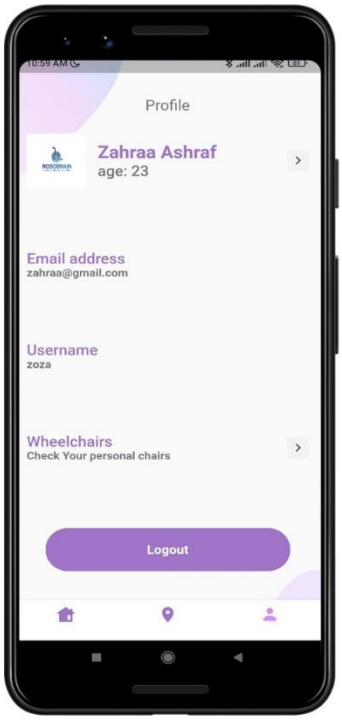


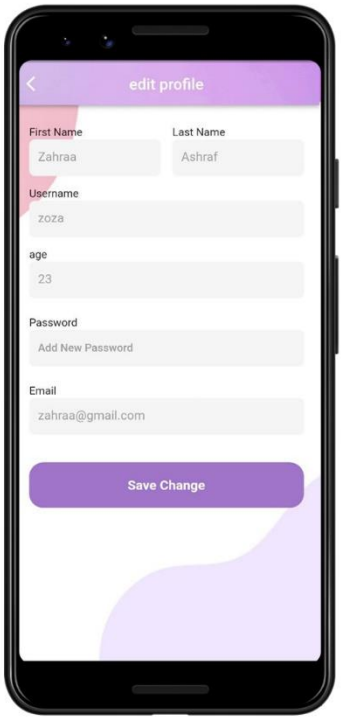
In the map user can track the wheelchair and how far of their own location. And Notifications are shown with the id of the wheelchair, and can be deleted.

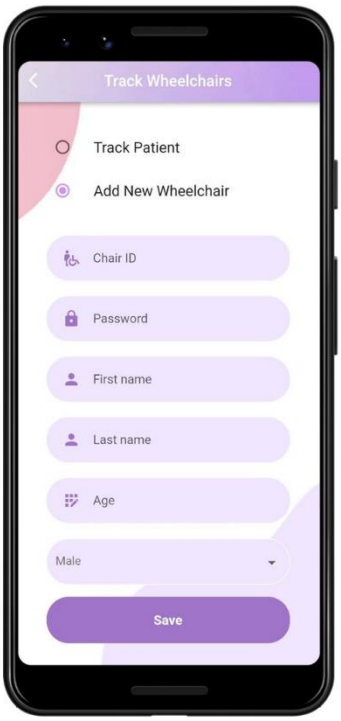


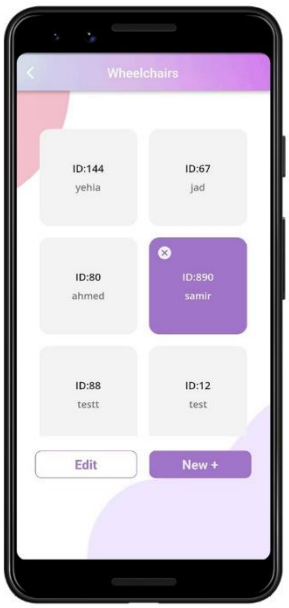
19.4 RoboBrain Application

In the profile of the caregiver they are able to edit their info, add new wheelchairs either by signing up a new patient with their info or signing in a registered patient through their chair id and password to start tracking them, that leads the relation between patient and caregiver to be many to many. Also the edit wheelchair is for cases in which the wheelchair is replaced and we need to move the patient with another one so we transform their registered data into another id instead of signing them up again.









- The Wheelchairs of the caregiver are shown in this screen and

- The caregiver can switch between them by double tap they can view the home page for each and one tap is to select to edit a certain wheelchair.

- Username of the caregiver can't be changed.

- ID and password of the wheelchair must be registered in the main organization and the person who gets the wheelchair or even works in some sort of medical institution gets these information.