

# 1. Chapter 4: Hardware Cabling and Configuration

## 4.1 Introduction

The success of any autonomous system not only depends on the quality of its hardware and software but also on the precision and reliability of its electrical integration. In the Ocean Cleaner project, electronic cabling forms the backbone of communication and power distribution across all critical components, from sensors and actuators to processing units and power systems.

This chapter focuses on the careful planning and execution of the electronic cabling layout, ensuring that all modules are correctly connected and securely fastened to withstand harsh marine conditions. It also highlights the safety protocols applied to prevent short circuits, interference, and overheating. Additionally, the chapter describes the communication protocols used, such as UART, which ensure reliable data exchange between the Raspberry Pi, Arduino, sensors, and actuators. These protocols were selected based on their suitability for real-time, low-latency communication in embedded marine applications.

In addition, rigorous testing of both the hardware and software was conducted to validate the system's performance and reliability. This includes unit testing for individual components, integration testing to evaluate how different modules interact, and system-level testing in simulated and real-world environments to ensure the Ocean Cleaner performs its mission of autonomous waste collection effectively and safely.

## 4.2. Preparing the Raspberry Pi Environment

This section outlines the essential steps to prepare the Raspberry Pi for robotics development. It covers downloading and installing Ubuntu 20.04, performing initial setup, and installing the ROS 1 Noetic framework.

### 4.2.1. OS Instalation

The Raspberry Pi Imager software was utilized to configure the system's SD card efficiently. This tool streamlines the process of selecting a compatible operating system and ensures a reliable installation on the device. For this project, Ubuntu Server 20.04.5 LTS (64-bit) was chosen due to its stability and robust support for robotics applications. The user-friendly interface of Raspberry Pi Imager allows for quick selection and writing of the OS image, minimizing setup time and reducing the likelihood of errors [1]

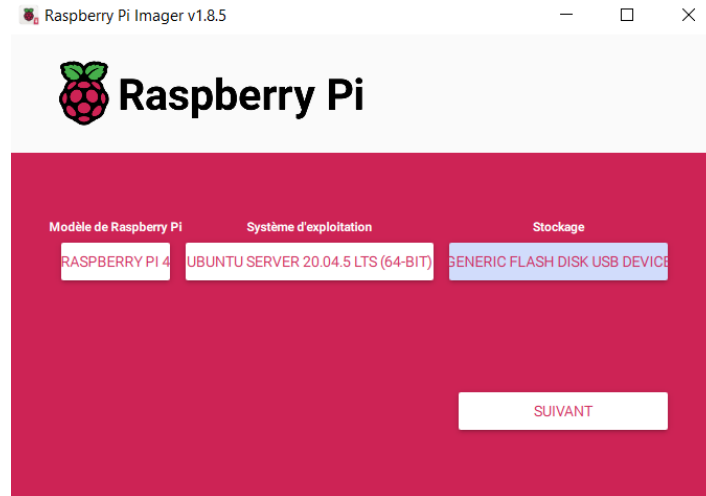


Figure 1 Raspberry Pi Imager - Operating System and Storage Selection

#### 4.2.2. System Initialisation

Following the initial boot sequence of the Raspberry Pi, the system was prepared for development by ensuring that all software packages were current. This was accomplished by executing the following command to update the package index and upgrade all existing packages:

➤ `sudo apt update && sudo apt upgrade -y`

To establish a comprehensive graphical user interface conducive to development and general usage, the complete Ubuntu desktop environment was installed using the command below:

➤ `sudo apt install ubuntu-desktop -y`

Once the installation was complete, I rebooted the Raspberry Pi to apply all changes and activate the newly installed desktop interface. This initialization step ensured a fully functional and up-to-date software environment, providing a stable foundation for subsequent configuration and deployment of the robotic system.

#### 4.2.3. Remote Access via RealVNC Viewer

Efficient remote control and monitoring of the Raspberry Pi during development and testing phases was achieved using **RealVNC Viewer**, selected as the remote desktop solution for this project. This tool offers secure and reliable access to the Pi's graphical interface over a network, enabling full system interaction without the need for a physically connected monitor, keyboard, or mouse.

The **VNC Server** feature was activated using the Raspberry Pi configuration tool (`sudo raspi-config`), under *Interface Options > VNC*. Once enabled, RealVNC Viewer was launched on a host computer, and a connection was established by entering the Raspberry Pi's IP address.

This setup ensured flexibility and convenience throughout the development process, allowing seamless remote system management.

#### 4.2.4 ROS Noetic Installation

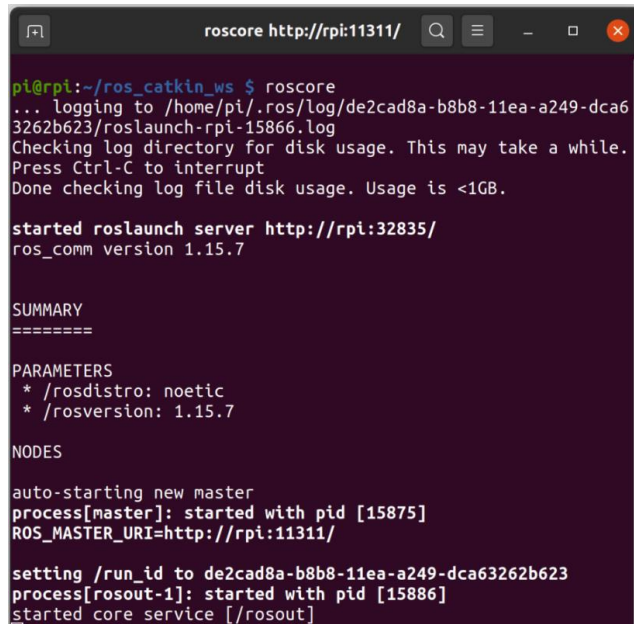
With the system environment fully prepared, the next step involved installing **ROS Noetic Ninjemys**, the final long-term support (LTS) version of ROS 1, compatible with Ubuntu 20.04. ROS Noetic was chosen for its stability, extensive community support, and compatibility with the computational resources available on the Raspberry Pi. The installation process followed the official guidelines provided by the ROS community [2], ensuring a reliable and standardized setup.

The installation process began by configuring the system to use the official ROS software repository, ensuring access to the most recent and stable ROS packages. Next, the full desktop version of ROS Noetic was installed to provide access to a complete development and simulation environment:

➤ `sudo apt install ros-noetic-desktop-full -y`

After installation, the ROS environment was initialized and configured by updating the system's shell profile to source the ROS setup script. Essential development tools and dependencies were then installed and initialized using the 'rosdep' utility to ensure proper package management.

The successful installation of ROS Noetic was verified by running the 'roscore' command. As shown in Figure below, the ROS master and core services started without errors, confirming a functional ROS environment.

A terminal window titled 'roscore http://rpi:11311/' showing the execution of 'roscore' on a Raspberry Pi. The output includes logging information, disk usage checks, and the successful start of the ROS master and core services.

```
pi@rpi:~/ros_catkin_ws $ roscore
... logging to /home/pi/.ros/log/de2cad8a-b8b8-11ea-a249-dca6
3262b623/roslaunch-rpi-15866.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://rpi:32835/
ros_comm version 1.15.7

SUMMARY
=====

PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.15.7

NODES
auto-starting new master
process[master]: started with pid [15875]
ROS_MASTER_URI=http://rpi:11311/

setting /run_id to de2cad8a-b8b8-11ea-a249-dca63262b623
process[rosout-1]: started with pid [15886]
started core service [/rosout]
```

Figure 2 ROS Noetic core successfully running on Raspberry Pi

## 4.3 Hardware Integration and Configuration

With the Raspberry Pi and ROS environment fully prepared, the next phase centered on the physical integration and configuration of essential hardware components. This section provides an overview of the cabling, initial setup, and verification steps performed for each device.

### 4.3.1 LIDAR Sensor Connection and Configuration

The RPLIDAR A1 was integrated into the system to provide real-time laser scanning capabilities. Emphasis was placed on the hardware connection and proper system-level permissions to ensure reliable communication before proceeding with software integration.

#### 4.3.1.1. Hardware Connection and USB Permissions

The RPLIDAR sensor was physically connected to the Raspberry Pi using a 'USB-to-microUSB' cable.

To verify that the LIDAR device was recognized by the operating system and to set the correct permissions, the following commands were executed:

- `ls -la /dev | grep ttyUSB`
- `sudo chmod 666 /dev/ttyUSB0`

This process ensured that the ROS nodes would have the necessary access to the serial port for real-time data acquisition.



Figure 3 RPLIDAR A1 connected to Raspberry Pi via USB

#### 4.3.1.2. ROS Driver Installation and Visualization

Once the hardware connection was confirmed, the `rplidar\_ros` package was installed on ROS Noetic. The serial port was configured according to the detected device (typically `/dev/ttyUSB0`), and the LIDAR node was launched. Real-time laser scan data was successfully visualized in RViz, confirming both hardware and software integration (Figure 4.3).

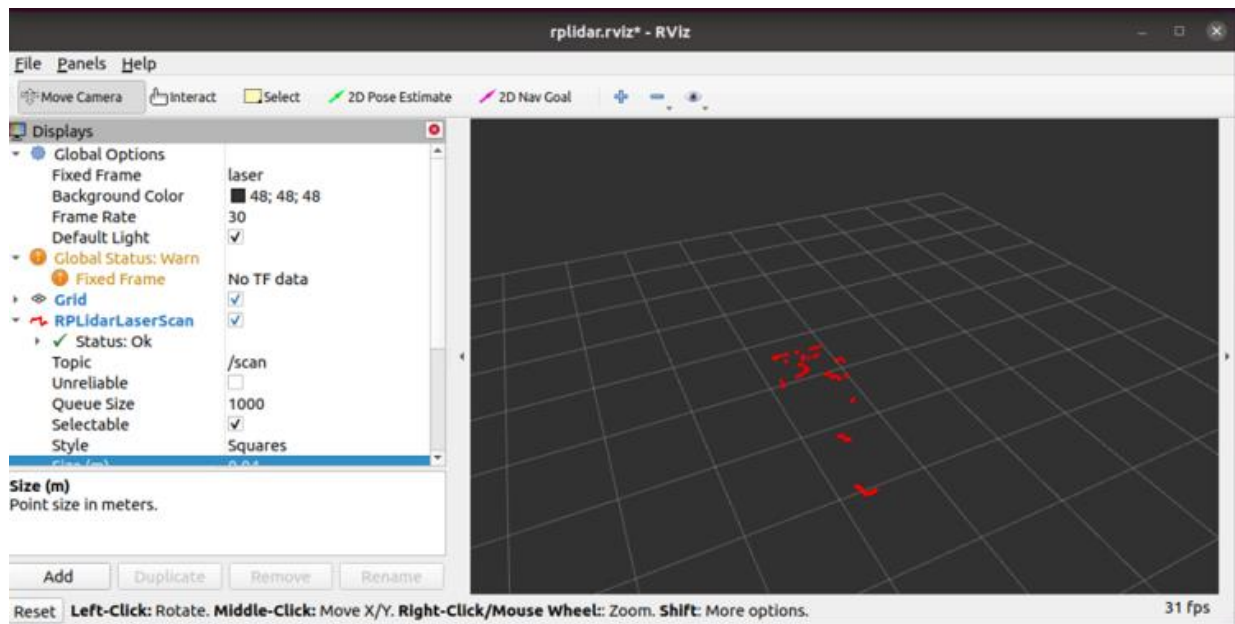


Figure 4 RViz LIDAR scan visualization

This approach ensured robust hardware integration as a foundation for further development and mapping tasks.

#### 4.3.2 IMU Module Connection and Configuration

Configuration of the 9-axis IMU communication on the Raspberry Pi 4 is achieved through activation of the I2C interface. The I2C option is enabled via the Raspberry Pi configuration tool, which authorizes the system to recognize and interact with I2C peripherals (see Figure 4.4).

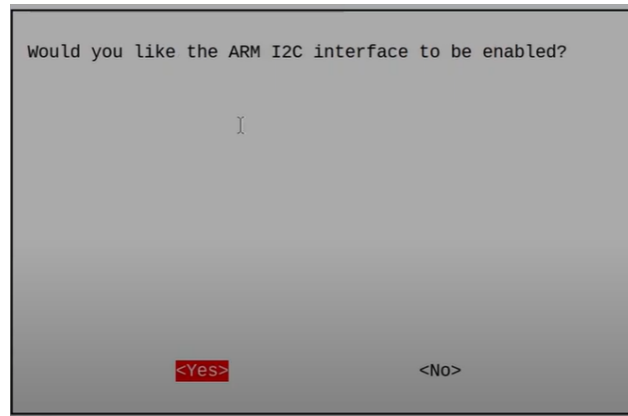


Figure 5 Enabling I2C on Raspberry Pi

The 9-axis IMU's SDA and SC

L lines were connected to pins 3 and 5 on the Raspberry Pi 4 GPIO header, with power and ground properly linked. Wiring followed the I2C protocol, ensuring stable communication for sensor data acquisition.

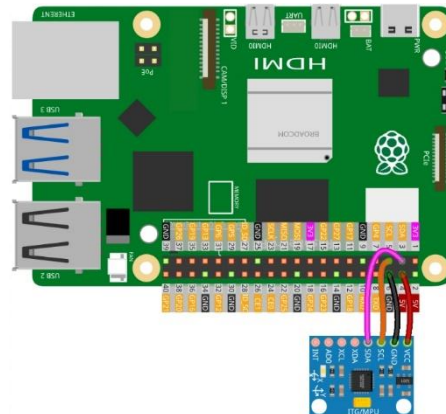


Figure 6 IMU–Raspberry Pi Wiring

A custom ROS node was developed to interface with the 9-axis IMU via the I2C port on the Raspberry Pi 4. This node continuously reads raw sensor data from the IMU and publishes it as standardized ROS messages, making the orientation and acceleration information accessible to the entire ROS ecosystem. IMU data was visualized in RViz for real-time monitoring, as shown in the figure below.

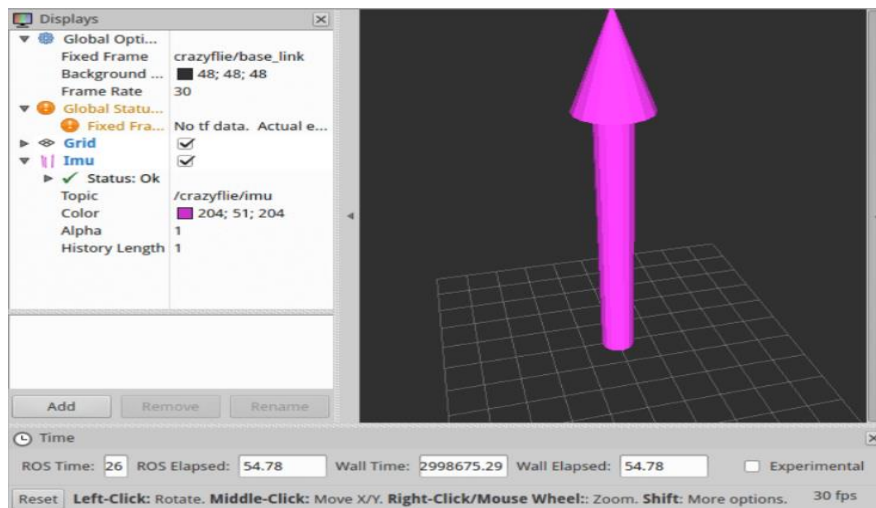


Figure 7 IMU data visualization in RViz

### 4.3.3 GPS Sensor Connection and Configuration

Integration of the SIM808 GPS module was essential for real-time geolocation and remote monitoring of the Ocean Cleaner. This section details the configuration of UART communication, the hardware wiring, and data validation.

#### 4.3.3.1 UART Configuration on Raspberry Pi

UART communication was enabled by activating the serial interface and disabling the serial login shell using raspi-config. This configuration step was necessary to avoid interface conflicts and ensure seamless data exchange with the Arduino Uno intermediary.

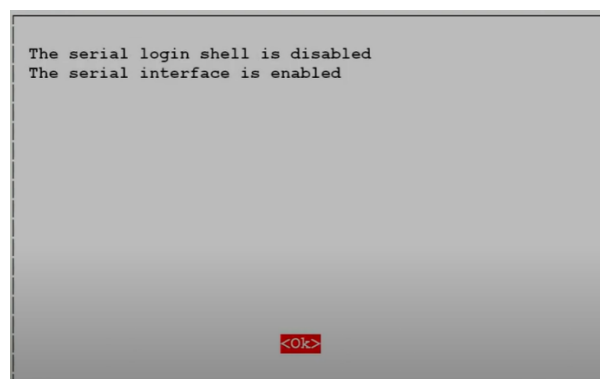


Figure 8 UART configuration in raspi-config

#### 4.3.2.2 Hardware Wiring and Power Management

Due to the lack of a reliable SIM808 GPS library in the Raspberry Pi community, the Arduino Uno was used to interface with the SIM808 module via UART (Arduino pins 7 and 8). The Arduino was then connected to the Raspberry Pi 4 through UART (Arduino pins 0 and 1 to Pi 4 pins 8 and 10), with all devices sharing a common ground. Power was distributed from a single power bank, which supplied both the SIM808 module and the Raspberry Pi 4; the Raspberry Pi, in turn, provided 5V power to the Arduino Uno.

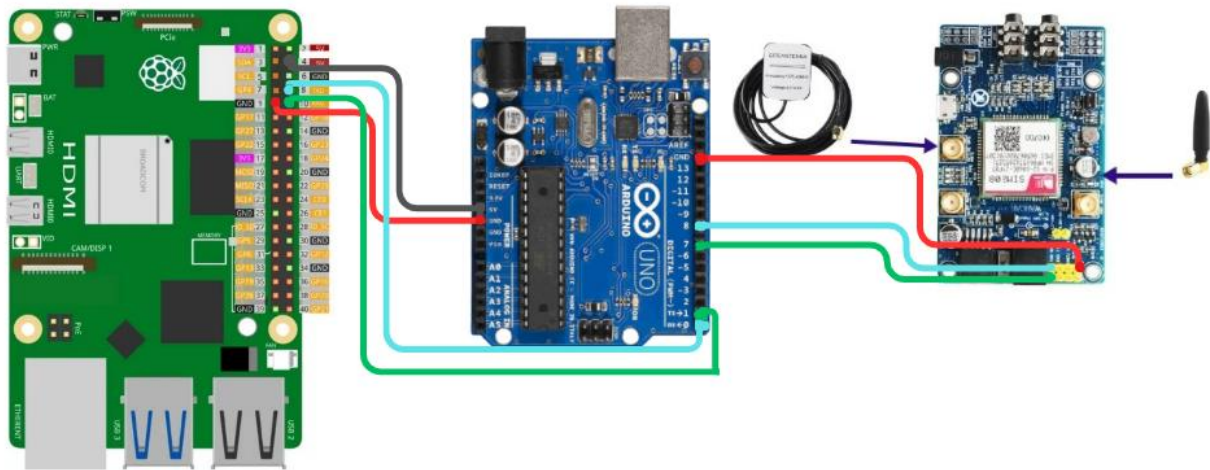


Figure 9 Wiring diagram for UART communication between Raspberry Pi 4, Arduino Uno, and SIM808 module

#### 4.3.3.3 Data Acquisition and Validation

A Python script utilizing the pyserial library was developed to verify UART communication on the Raspberry Pi, targeting the `/dev/ttyAMA0` port. The script successfully received GPS coordinates from the Arduino Uno, confirming reliable data transmission and integration.

```
pi@raspberrypi:~$ python3 testuart.py
Listening to Arduino UART on /dev/ttyAMA0...
Received: GPS: 36.849136, 10.274944
Received: GPS: 36.849132, 10.274959
Received: GPS: 36.849121, 10.274966
Received: GPS: 36.849105, 10.274969
Received: GPS: 36.849098, 10.274972
Received: GPS: 36.849098, 10.274969
Received: GPS: 36.849105, 10.274966
Received: GPS: 36.849113, 10.274964
Received: GPS: 36.849109, 10.274962
Received: GPS: 36.849113, 10.274960
Received: GPS: 36.849113, 10.274959
Received: GPS: 36.849113, 10.274957
Received: GPS: 36.849113, 10.274953
Received: GPS: 36.849109, 10.274950
Received: GPS: 36.849109, 10.274952
```

Figure 10 GPS data on the Raspberry Pi terminal.



This multi-stage approach enabled robust GPS-based navigation and monitoring for the Ocean Cleaner.

#### **4.3.4. Camera Connection and Configuration**

The integration of a camera module was essential for providing vision-based perception to the Ocean Cleaner system. This section outlines the hardware connection, configuration, and testing of the camera on Raspberry Pi 4.

##### **4.3.4.1 Hardware Connection**

The camera module was physically connected to the Raspberry Pi 4 using the dedicated CSI (Camera Serial Interface) port. Care was taken to ensure correct orientation of the ribbon cable and a secure connection to prevent signal loss. Figure 11 illustrates the proper connection of the camera module to Raspberry Pi 4.



*Figure 11 Camera module connected to the Raspberry Pi 4 via CSI port.*

##### **4.3.4.2 Enabling the Camera Interface**

During setup on Ubuntu 20, the default `libcamera` suite was found to be incompatible with the operating system. To resolve this, alternative packages `ffmpeg` and `v4l-utils` were installed:

- `sudo apt install ffmpeg v4l-utils`

After installation, camera functionality was successfully verified by streaming live video from the camera module with the following command:

- `ffplay /dev/video0`

This command opened a new window displaying the real-time camera feed, confirming successful integration.

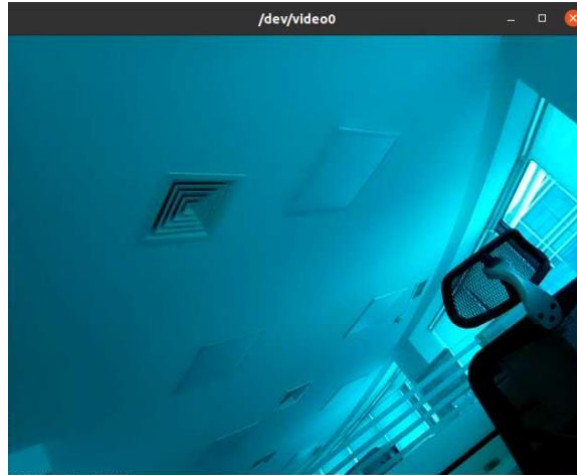


Figure 12 Raspberry Pi Camera Streaming Interface

#### **4.3.5 Motor Control System Integration**

The motor control system for the Ocean Cleaner is based on a modular approach that separates high-level commands and low-level actuation.

##### **4.3.5.1 Motor Control System**

The Raspberry Pi 4 is responsible for overall system control and transmits motor control data to the Arduino Uno via UART. The Arduino Uno then interprets these signals and generates PWM to the Electronic Speed Controllers (ESCs) of each brushless three-phase motor. This architecture enables the Raspberry Pi to focus on navigation and decision-making, while the Arduino handles real-time actuation, ensuring responsive and robust differential steering.

##### **4.3.5.2. Hardware Wiring and Power Distribution**

The ESCs are directly powered by separate LiPo batteries (Zeee 4S 14.8V 5200mAh) to meet the high current requirements of the APISQUEEN U2 MINI thrusters. The Arduino Uno receives its power from the 5V output of the Raspberry Pi 4. All modules share a common ground to maintain electrical stability. The UART interface connects Raspberry Pi pins 8 (TX) and 10 (RX) to Arduino pins 0 (RX) and 1 (TX), respectively, facilitating reliable serial communication. PWM signals for motor control are output from Arduino pins 9 and 10 to the signal wires of each ESC.

The complete wiring configuration, integrating the Raspberry Pi 4, Arduino Uno, two ESCs, two brushless motors, and dual LiPo batteries, is illustrated in Figure 4.8.

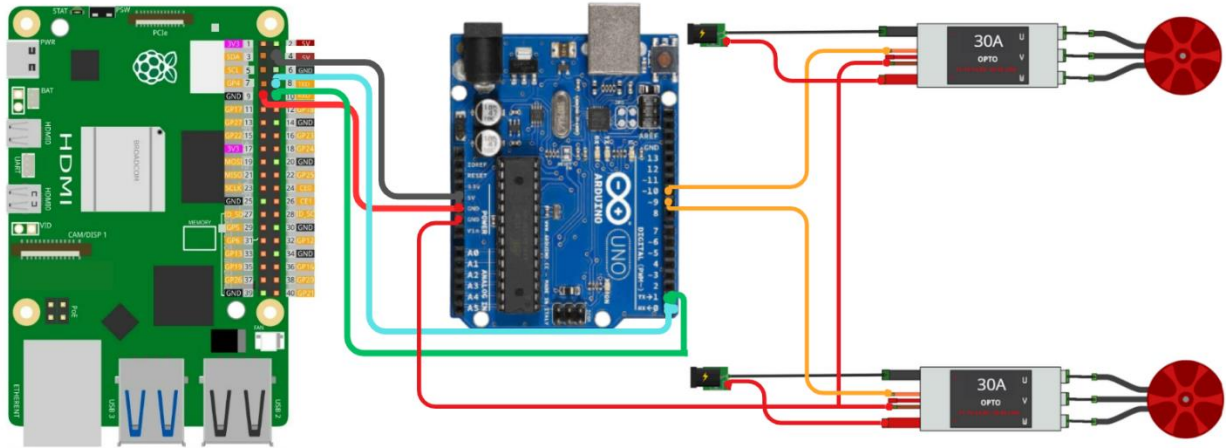


Figure 13 Motor Control System Wiring Diagram

#### 4.3.5.2. Power System Integration

The power system for the Ocean Cleaner was wired to ensure reliable operation, with all components except the motor cables housed in a protective box to prevent water exposure. The Raspberry Pi 4 and SIM808 module were powered via a TECTIN 20000mAh 66W power bank, connected through a USB to Type-C cable for the Pi 4 and a USB to DC cable for the SIM808, with cables secured inside the box. The two APISQUEEN U2 MINI thrusters, each paired with an ESC, were powered by separate Zeee 4S 14.8V 5200mAh LiPo batteries, with positive and negative terminals wired to the ESCs, while only the motor cables extended outside, designed to withstand water contact. This setup ensures stable power distribution, minimizes overheating risks, and supports extended mission durations for autonomous waste collection.

## References

- [1] Raspberry Pi Foundation, "Raspberry Pi Imager documentation," 2024. Available: <https://www.raspberrypi.com/software/>
- [2] ROS Wiki, "ROS Noetic Installation on Ubuntu," [Online]. Available: <https://wiki.ros.org/noetic/Installation/Ubuntu>. [Accessed: June 16, 2025].