REMOTLY OPERATED UNDERWATER VEHICLE

Project Members:	**ID Number:**
Maged Yasser Ibrahim Ali Seada	803084671
Manar Hossam Mohamed Abdrabo	803084702
Omar Tarek Attia Elsayed Moharam	803084642
Mohamed Medhat Mohamed Elbayaa	803116355
Ahmed Mostafa Ali Dawood	803084865
Bemwa Emeil Farouk Azmy	803090352
Mohamed Hesham Abdelfattah Khattab	803090410
Mohamed Hamdi Moltash	803084878
Ali Taher Ali Shoman	803084640
Nourhanne Ahmed Hamed Rabea	805644532

A Project Graduation Document Submitted in Partial Fulfillment of the Requirements for the BSc. Degree in Mechatronics Engineering.

(**Mechatronics Engineering**)

Faculty of Engineering, Mansoura University
Mansoura, Egypt

2024/2025

REMOTLY OPERATED UNDERWATER VEHICLE

Project Members:	**ID Number:**
Maged Yasser Ibrahim Ali Seada	803084671
Manar Hossam Mohamed Abdrabo	803084702
Omar Tarek Attia Elsayed Moharam	803084642
Mohamed Medhat Mohamed Elbayaa	803116355
Ahmed Mostafa Ali Dawood	803084865
Bemwa Emeil Farouk Azmy	803090352
Mohamed Hesham Abdelfattah Khattab	803090410
Mohamed Hamdi Moltash	803084878
Ali Taher Ali Shoman	803084640
Nourhanne Ahmed Hamed Rabea	805644532

A Project Graduation Document Submitted in Partial Fulfillment of the Requirements for the BSc. Degree in Mechatronics Engineering.

(**Mechatronics Engineering**)

Under the Supervision of

Prof. Dr. Islam Mohamed Ismael **Eng. Hossam Abdelmaksode**

Faculty of Engineering, Mansoura University
Mansoura, Egypt

2024/2025

Acknowledgments

We would like to begin by expressing our heartfelt gratitude and sincere appreciation to

Prof. Dr. Islam Mohamed Ismael and **Eng. Hossam Abdelmaksode** for their unwavering support, invaluable efforts, and continuous guidance throughout the course of this project. Their mentorship and expertise have been a beacon of inspiration, guiding us through challenges and helping us turn our vision into reality. Their dedication to fostering academic excellence and their willingness to share their knowledge have not only enriched this project but also enhanced our understanding and skillset. Without their valuable insights and constant encouragement, this work would not have been possible.

We are also deeply thankful to our friends and colleagues who generously shared their prior experiences and offered us continuous support. Their collaborative spirit and constructive feedback have contributed greatly to the refinement of our ideas and the successful execution of this project. The encouragement we received from them during moments of doubt and uncertainty kept us motivated and steadfast in our pursuit of excellence.

Furthermore, we extend our profound gratitude to our families, whose endless sacrifices, unwavering encouragement, and unconditional love have been the cornerstone of our success. Their patience and faith in our abilities have provided us with the strength and determination to overcome obstacles and achieve our goals. They have been a constant source of inspiration and comfort, not only during this project but throughout our entire academic journey.

Finally, we wish to acknowledge everyone who has contributed, directly or indirectly, to the success of this project. Whether through technical support, moral encouragement, or simply being there for us during challenging times, every single effort has played a significant role in this accomplishment. This project is a testament to the collective support and belief of all the amazing individuals around us. To all of you, we are forever grateful.

Table of Content

List of Tables

List of Figures

Abstract

Underwater exploration poses numerous challenges, including limited visibility, restricted access to confined spaces, and the need for effective underwater monitoring. Our graduation project introduces a robust and versatile Remotely Operated Vehicle (ROV) designed to address these challenges effectively.

The ROV is equipped with an advanced camera system that provides clear, real-time visuals of underwater environments. This camera enables researchers to explore marine life and study underwater ecosystems in detail. Additionally, the camera incorporates object detection capabilities using the Object Detection Model, allowing the identification and classification of underwater objects such as fish, humans, and debris (e.g., glass and waste materials).

One of the unique features of the ROV is its ability to navigate through narrow and confined spaces that are difficult or dangerous for human divers to access. This capability makes it suitable for inspecting underwater infrastructure and identifying potential issues in these hard-to-reach areas.

In summary, the proposed ROV provides a cost-effective, efficient, and safe solution for underwater exploration, environmental research, and infrastructure inspection. By integrating advanced technologies and innovative design, our project aims to contribute significantly to the fields of marine research and underwater operations.

CHAPTER 1: INTRODUCTION

1.1 OUR ROV DESIGN

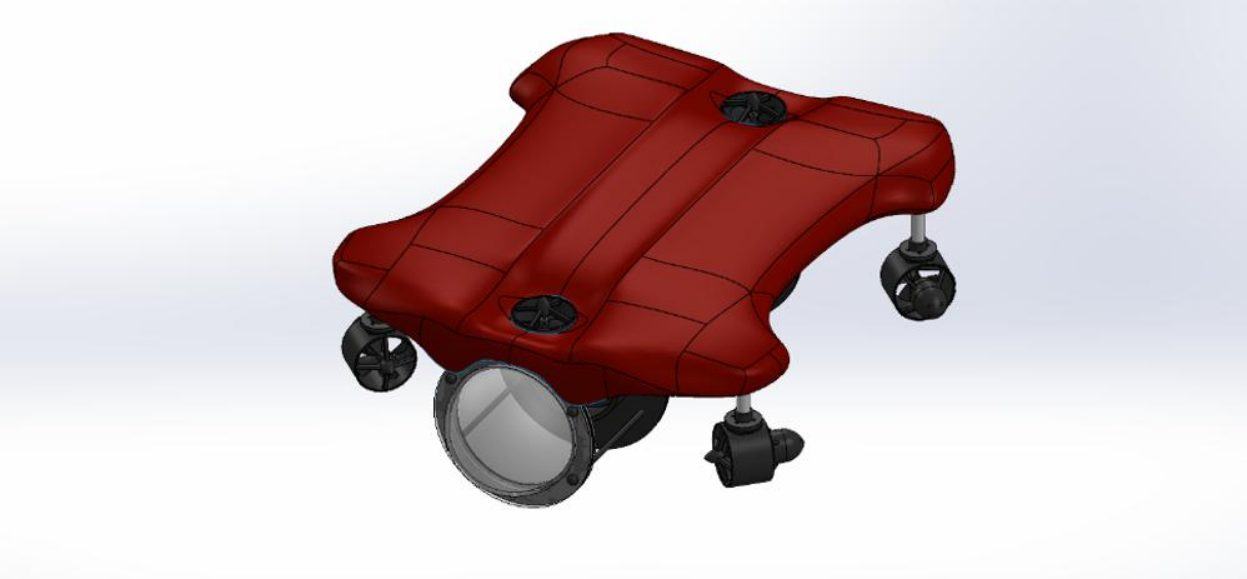


Figure 1.1 ROV Design

1.2 PROJECT OVERVIEW

The increasing environmental pressures on marine ecosystems are alarming. Coral reefs, which are home to 25% of all marine species and generate nearly $36 billion annually through tourism, fisheries, and coastal protection, are under severe threat. Studies show that 50% of the world's coral reefs have already been lost, and if current trends continue, more than 90% could be gone by 2050 due to rising sea temperatures, pollution, and overfishing.

Marine infrastructure is also at risk; corrosion and pollution cost industries billions of dollars each year, with the global cost of marine biological pollution alone estimated at $7.5 billion annually. These challenges not only threaten biodiversity but also disrupt economies that rely on marine resources.

Our project addresses these critical issues by developing a remotely operated vehicle that can:

- Provide a clear, real-time view of underwater environments, enabling the study of aquatic life and underwater ecosystems.
- Detect and classify underwater objects, such as fish, humans, and debris like glass and waste materials.
- Navigate confined and hazardous underwater spaces, offering safer alternatives for inspecting submerged infrastructures, reducing maintenance costs, and enhancing safety in aquaculture and marine industries.

This innovative solution integrates advanced technology to support environmental research, infrastructure inspection, and the preservation of marine ecosystems, addressing some of the most pressing challenges in underwater exploration.

Figure 1.2 Underwater work risks

Figure 1.3 Aqua culture

1.3 EXPANDED SDG CONTRIBUTIONS

In addition to previously highlighted SDGs, our ROV contributes further as follows:

1.3.1 SDG 2 (Zero Hunger):

Supports sustainable aquaculture by monitoring marine health and enhancing fish farming practices.

1.3.2 SDG 11 (Sustainable Cities and Communities):

Assists in maintaining coastal infrastructure, ensuring safety and sustainability for communities reliant on marine resources.

1.3.3 SDG 14 (Life Below Water):

Aids in the detection and removal of invasive species, preserving biodiversity and promoting ecosystem balance.

1.3.4 SDG 13 (Climate Action):

Provides critical data for understanding the impacts of climate change on marine ecosystems, aiding in mitigation strategies.

1.3.5 SDG 9 (Industry, Innovation, and Infrastructure):

Enhances innovation by employing cutting-edge robotics and AI, fostering advancements in underwater technology.

1.3.6 SDG 7 (Affordable and Clean Energy):

Supports the renewable energy sector by inspecting and maintaining offshore wind and tidal energy structures, ensuring their efficient operation.

Figure 1.4 SDGs the project covers

1.4 IMPACT

This project is of great importance for its potential to have a significant impact on the environment and various industries that rely on underwater operations. The versatility and autonomy of the ROV provides several important benefits:

1.4.1 Environmental Impact:

ROVs play a crucial role in preserving the marine environment. By utilizing the camera system to provide us with a clear view and to identify and classify underwater objects, which is essential for preserving biodiversity and supporting marine life.

1.4.2 Community Impact:

Society, especially in coastal and offshore areas, will benefit from improved efficiency and safety of underwater operations. The ability to inspect and photograph underwater infrastructure without the need for divers makes these activities safer and more cost-effective. It also reduces human risks in challenging underwater environments.

1.4.3 Market Impact:

The underwater robotics market is growing, especially in areas such as marine research and infrastructure maintenance. The unique capabilities of this ROV make it a valuable tool in multiple sectors, from environmental protection to industrial applications. It provides a competitive advantage by addressing the industry's need for efficient, safe, and cost-effective solutions.

1.4.4 End-user impact:

For researchers, the ROV's ability to collect data on underwater ecosystems with minimal human intervention improves the speed and accuracy of research.

In short, the project has far-reaching impact across multiple sectors, contributing to environmental sustainability, community safety, market innovation, and operational efficiency for end-users.

CHAPTER 2: HARDWARE

2.1 INTRODUCTION

The term "hardware" typically refers to the physical parts of a computer system, including the interconnecting cables. For our project, we have chosen specific hardware components to support its execution. This chapter provides an overview of these components and their functions.

2.2 THE SYSTEM DIAGRAM

Description: This diagram describes how the vehicle operates and the contents of the hardware system in terms of each component and its connection with the microcontroller.

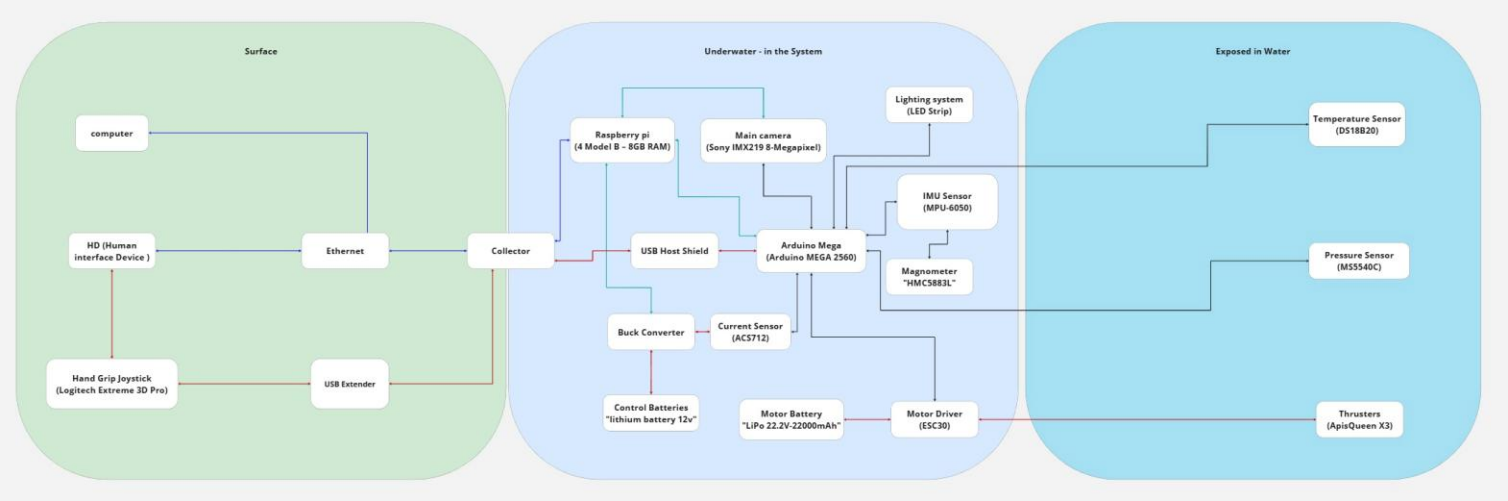


Figure 2.1 System Diagram

- The diagram is divided into three basic units:
    - On the surface
    - Inside the vehicle
    - Exposed in water

2.2.1 On the surface

Here there is a human operator who controls the operation of the vehicle and directs it through the joystick and sees what is happening below on the computer because the readings of the cameras and indicators are displayed on the computer.

2.2.2 Inside the vehicle

This inside system contains the internal system of the vehicle, including the microcontroller (Arduino), sensors, and the ESC that directs the movement as we want by controlling the trusters and the presence of the power system (LIPO battery and lithium batteries), which are responsible for operating the system. There is also the Raspberry Pi, through whose signals we see what is monitored by the camera

LIPO batteries are responsible for providing power to the trusters, and the lithium batteries are responsible for feeding the control system.

2.2.3 Exposed in water

There are some components outside the system, such as motors that work to steer and stabilize the vehicle, as well as a pressure and depth sensor that reads the pressure surrounding the vehicle and the depth to which we reach, and a temperature sensor that reads the temperature surrounding the vehicle.

2.3 HARDWARE COMPONENTS

This chapter delves into the hardware components integral to our graduation project. The chosen components form the backbone of our system, each playing a crucial role in ensuring functionality, efficiency, and reliability. Below is a detailed discussion of each hardware component:

2.3.1 Microcontroller: an overview

A microcontroller, often abbreviated as MCU (microcontroller unit), is a compact computer system integrated onto a single VLSI (Very Large-Scale Integration) chip. It incorporates one or more CPUs (processor cores), along with memory and programmable input/output peripherals. Microcontrollers are distinct from microprocessors used in personal computers or general-purpose applications, as they are specifically designed for embedded systems.

Microcontrollers play a vital role in a wide range of automated products and devices, including automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys, and various other embedded systems. These small, specialized computing units enable automation and control within these devices.

2.3.1.1 Raspberry Pi 4 Model B – 8GB RAM

The Raspberry Pi 4 Model B – 8GB RAM is a powerful mini-computer developed by the Raspberry Pi Foundation in the UK. With 8GB of RAM, it's the highest RAM option available for Raspberry Pi, offering high performance for a variety of applications, including programming, robotics, IoT projects, and desktop computing.

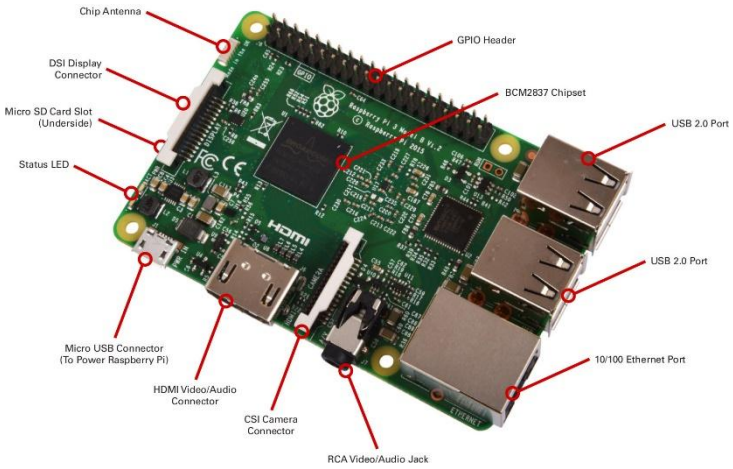


Figure 2.2Raspberry Pi 4 Model B – 8GB RAM

- Key Features:
    - Processor: Quad-core ARM Cortex-A72 (ARM v8) 64-bit SoC at 1.5GHz, providing strong processing power for multitasking.
    - Memory: 8GB LPDDR4 RAM, allowing more complex applications, multitasking, and smoother performance in memory-intensive tasks.
    - Connectivity: Dual-band 802.11ac Wi-Fi, Bluetooth 5.0, and Gigabit Ethernet for high-speed internet and peripheral connectivity.
    - USB Ports: Two USB 3.0 and two USB 2.0 ports for fast data transfer and connection with multiple devices.

- Video and Display: Dual micro-HDMI ports supporting up to 4K resolution, ideal for dual monitor setups.
- Storage: MicroSD card slot for main storage, supporting high-capacity cards for various applications.
- Power Supply: Requires a 5V 3A USB-C power adapter.

2.3.1.2 Arduino MEGA 2560

The Mega 2560 R3 with ATmega16U2 is an Arduino compatible board that uses the same USB device as the original Arduino Mega 2560 R3 for the highest level of compatibility. The Mega 2560 provides a large amount of I/O and memory and is ideal for larger projects.

Figure 2.3 Arduino MEGA 2560

- Key Features:
  - Processor: ATmega2560 processor running at 16MHz
  - Memory: 256KB Flash memory
  - Number of PWM shared with the digital I/O: 15 pins
  - Number OF Analog inputs: 16 Analog inputs that can also be used as digital I/O
  - Number of Digital I/O: 54 pins total of up to 70 digital I/O
  - Hardware serial ports: 4 ports

- Operation voltage: 5V

- Data sheet Table:

Microcontroller	ATmega2560
Serial to USB Converter	ATmega16U2
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Digital I/O Pins	54
PWM I/O Pins (Shared with Digital I/O)	15
Analog Input Pins (can also be used as digital I/O)	16
DC Current per I/O Pin	20mA
DC Current which can be drawn from 3.3V Pin	50mA
Flash Memory	256 Kbytes
SRAM	8 Kbytes
EEPROM	4 Kbytes
Clock Speed	16MHz

Table 2.1 Data sheet of Arduino Mega 2560

- The Pinout of Arduino Mega:

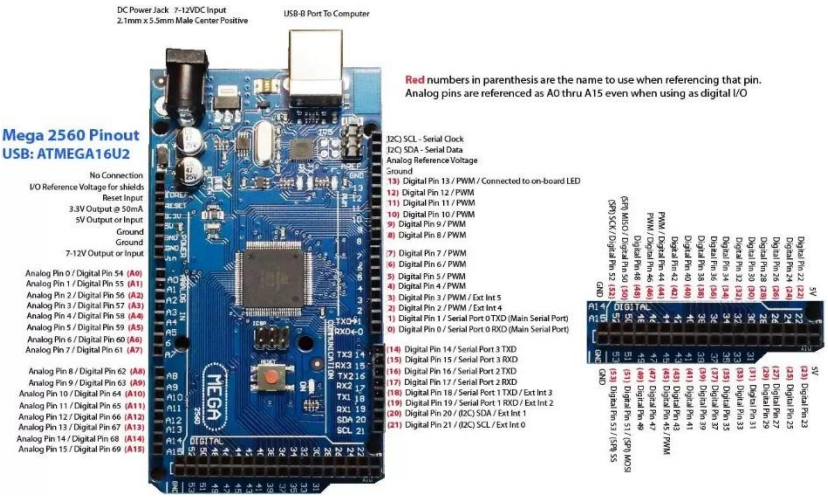


Figure 2.4 Pinout of Arduino Mega

- Role in the Project**:** It is the main controller that receives signals from the sensors and controls the operation of Thrusters.

2.3.1.3 Arduino Shield (USB Host Shield)

The Arduino USB Host Shield allows you to connect a USB device to your Arduino board. The Arduino USB Host Shield is based on the MAX3421E, which is a USB peripheral/host controller containing the digital logic and analog circuitry necessary to implement a full-speed USB peripheral or a full-/low-speed host compliant to USB specification rev 2.0. The shield is Tinker Kit compatible, which means you can quickly create projects by plugging Tinker Kit modules onto the board.

Figure 2.5 Arduino USB Host Shield

- Specifications
    - Works with standard (dual 5/3.3V) and 3.3V-only (for example, Arduino Pro) boards.

- Operates over the extended -40°C to +85°C temperature range
- Complies with USB Specification Revision 2.0 (Full-Speed 12Mbps Peripheral, Full-/Low-Speed 12Mbps/1.5Mbps Host)

- Devices that supported by Board
  - HID Devices –mouse, joystick.
  - Game Pads – PS3, PS4, XBOX360, Wii…
  - USB-Serial Converters – FTDI, PL-2303, GPS…
  - ADK Android Phones and Tablets
  - Digital Cameras – Canon EOS, PowerShot, Nikon DSLR and P&S
  - Storing Devices – USB Memory, SD card reader, hard disk
  - Bluetooth Dongles

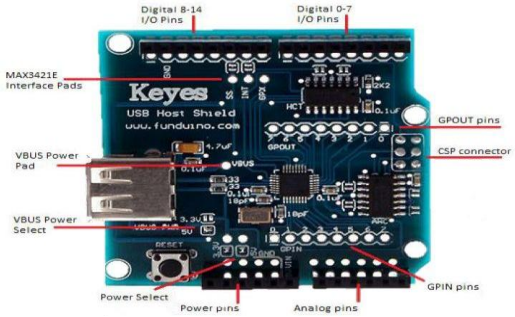


Figure 2.6 Arduino USB Host Shield pins

- Compatible with Arduino following hardware
  - Arduino Uno 328
  - Arduino Diecimila / Duemilanove 328
  - Arduino Mega 2560 (recommended)
  - Arduino Mega 128

i. **Power Select:** 2 solder jumpers marked “5V” and “3.3V”. They are used for different powers The configuration shown, when both jumpers are closed, is suitable for official

Arduinos, such as UNO, Duemilanove, Mega and Mega 2560. See Power Options section for detailed explanation.

ii. **Power pins:** are used to connect to the power pins of Arduino board. RESET, 3.3V, 5V and GROUND signals from this connector are used.

iii. **Analog pins:** are not used by the shield. They are provided to simplify mounting and provide pass-through for shields mounted atop of USB Host Shield in a stack.

iv. **GPIN pins:** Eight 3.3V general-purpose digital input pins of MAX3421E. They are used primarily to interface with buttons, rotary encoders and such. GPIN pins can also be programmed as a source of MAX3421E interruption. An example of GPIN use can be seen in digital camera controller

v. **ICSP connector** is used by the shield to send/receive data using SPI interface. SCK, MOSI, MISO and RESET signals from this connector are used.

vi. **GPOUT pins** are eight 3.3V general-purpose digital output pins of MAX3421E. They can be used for many purposes; I use it to drive HD44780-compatible character LCD, as can be seen in digital camera controller circuit, as well as this keyboard example. Max_LCD library which is part of standard USB Host library software package uses some of GPOUT pins.

vii. **Digital I/O pins 0-7**: like already mentioned analog pins are not used by the shield and provided only for convenience.

viii. **Digital I/O pins 8-13**: In this group, the shield in its default configuration uses pins 9 and 10 for INT and SS interface signals. However, standard-sized Arduino boards, such as Duemilanove and UNO have SPI signals routed to pins 11-13 in addition to ICSP connector, therefore shields using pins 11-13 combined with standard-sized Arduinos will interfere with SPI. INT and SS signals can be reassigned to other pins (see below); SPI signals cannot.

ix. **MAX3421E interface pads** are used to make shield modifications easier. Pads for SS and INT signals are routed to Arduino pins 10 and 9 via solder jumpers. In case pin is

taken by other shield a re-routing is necessary; a trace is cut and corresponding pad is connected with another suitable Arduino I/O ping with a wire. To undo the operation, a wire is removed, and jumper is closed. See interface modifications section for more information. GPX pin is not used and is available on a separate pad to facilitate further expansion. It can be used as a second interrupt pin of MAX3421E.

x. **VBUS power pad:** This path is used in advanced power configurations, described in Power Options section.

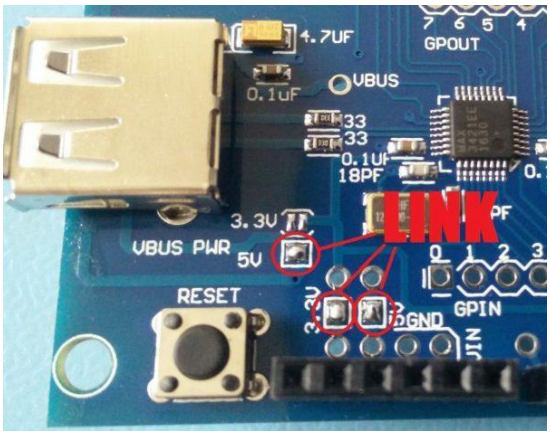


Figure 2.7 Arduino USB Host Shield power

- Preparing the shield for Arduino

Please note: Before this shield can be used on an Arduino UNO/MEGA etc., a few "bridges" need to be soldered (the shield is universal with other microcontroller platforms):

- POWER SELECT 3.3v
- POWER SELECT 5V
- VBUS PWR 5V

Figure 2.8 Arduino with USB Host Shield

- Method of connection: Connected as a shield on Arduino
- Role in the project: Used to connect the joystick to Arduino

2.3.2 Power System

The power system has been divided into two separate systems to provide stable power to feed the vehicle system.

- The first system: LIPO Battery provides the power necessary to operate the thrusters .
- The second system: to feed the rest of the system, including microcontrollers, sensors, and the remaining components.

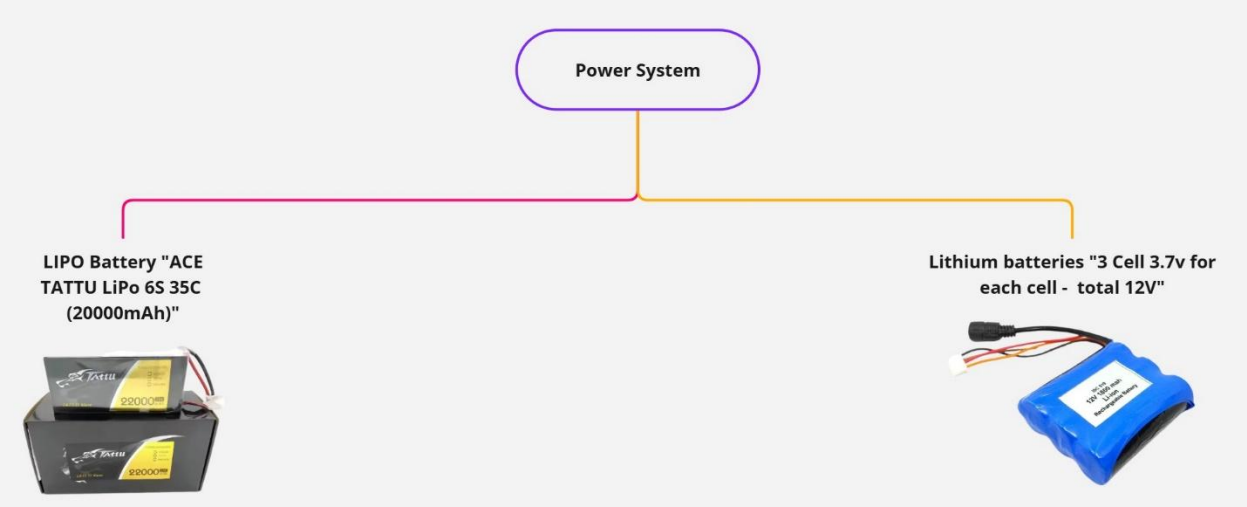


Figure 2.9 Power Sources

2.3.2.1 ACE TATTU LiPo 6S 35C (20000mAh)

The ACE TATTU LiPo 6S 22000mAh battery is a high-capacity, high-performance lithium polymer battery designed for ROVs, with a 22.2V output and a 35C discharge rate, it delivers consistent power and ensures extended operation time. The battery is lightweight yet durable, with a robust design that withstands challenging environments, ensuring safety and reliability. Its high energy density and efficiency.

Figure 2.10 ACE TATTU LiPo 6S 35C (20000mAh)

- Role In the Project:
    - LIPO Battery provides the power necessary to operate the thrusters with consistent power, enabling uninterrupted operation.
- Calculations of battery operation time for the drive system:
    - These calculations describe how long the battery can operate all thrusters when fully operational at maximum speed
- Battery Specification:
    - Voltage: 22.2V (6S)
    - Capacity: 22000mAh (22Ah)
    - Discharge Rate: 35C (22000mAh)
    - Dimensions:200mm Length x 91mm Width x 64mm Height
    - Net Weight(±30g):2544g
- Motor Consumption:
    - Each motor draws 8A at 24V.
    - Total current for 6 motors: I total=6×8=48A
- Battery Voltage and Current Matching:

- Battery is rated at 22.2V, while the motors require 24V.
- The slight difference in voltage means the motors may draw slightly more current to compensate. However, we assume 48A remains consistent for simplicity.

- Working Time Calculation:

The formula for calculating runtime is:

$$Runtime\ (hours) = Battery\ Capacity\ (Ah)/Total\ Current\ (A)$$

- For 22000mAh (22Ah) Battery:

$$Runtime = 22/48 \approx 0.458 hours = 27.5 minutes.$$

- Conclusion:
  - Using a 22000mAh battery: Motors will run for approximately 27.5 minutes.

2.3.2.2 Lithium batteries " 3 Cell each cell 3.7V - total 12V "

Lithium batteries are used to power the entire system. Known for their high energy density, long cycle life, and reliability, lithium batteries are essential for providing a stable and continuous power supply.

Figure 2.11 Lithium batteries 12V

- Key Features:
  - Type: Rechargeable Lithium-ion
  - Voltage: 12 V

- Role In the Project: it feeds the rest of the system, including microcontrollers, sensors, and the remaining components.

The lithium batteries ensure that all components receive consistent power, enabling uninterrupted operation. Their rechargeable nature also contributes to the sustainability and cost-effectiveness of the project.

2.3.2.3 Buck Converter "LM2596"

A buck converter is a type of DC-DC power converter that reduces a higher input voltage to a lower output voltage with high efficiency. It operates using a switching regulator to control the energy transfer from input to output through a combination of active components (such as transistors) and passive components (like inductors and capacitors).

Figure 2.12 Buck Converter "LM2596"

- Features:
  - Input and Output Voltage: Converts a higher input DC voltage to a stable and lower output DC voltage.
  - Efficiency: Highly efficient, typically achieving 85% to 95%, due to minimal energy loss in the switching process.
  - Switching Operation: Utilizes a high-frequency switch to control the duty cycle, determining the output voltage.
  - Compact Design: Compact and lightweight, making it ideal for modern electronic applications.

- Thermal Management: Operates efficiently with lower heat generation compared to linear regulators.

- Data sheet:
  - Input voltage:3V-40V
  - Output voltage:1.3V-35V
  - Output current: Rated current is 2A, maximum 3A
  - Conversion efficiency:92% (highest)
  - Output ripple:30mV (maximum)
  - Work temperature: (-40 – 85) degrees

- Role In the Project:
  - The power receives from the lithium batteries and feeds the control board with the appropriate voltage and current, as well as the Raspberry Pi system.

2.3.3 Sensors

2.3.3.1 IMU (3 Axis Gyro + 3 Axis Accelerometer) renovate MPU-6050 + HMC5883L Magnetometer

2.3.3.1.1 IMU (3 Axis Gyro + 3 Axis Accelerometer) renovate MPU-6050

The MPU-6050 provides stable readings even under high motion conditions, which is why it's frequently used in robotics and drone navigation for orientation and motion tracking. In a typical setup, it allows users to track real-time position changes and movements, making it integral for stabilizing systems.

Figure 2.13MPU-6050

- MPU6050 Gyroscope Accelerometer:
    - Axis Count: 6 axes total (3-axis gyroscope + 3-axis accelerometer)
    - Gyroscope: Measures rotational speed across three axes (X, Y, Z) to detect angular motion.
    - Accelerometer: Measures linear acceleration in three axes, allowing it to sense tilt, orientation, and movement
    - Digital Motion Processing (DMP): Built-in DMP performs complex calculations on-chip, freeing up the main microcontroller.
    - I2C Communication Protocol: Communicates with a host microcontroller (like Arduino or Raspberry Pi) via the I2C bus.
    - Power Consumption: Low-power consumption, ideal for battery-powered projects.
- Key Features of MPU6050:
    - Gyroscope Specifications:
        - Measures angular velocity
        - Full scale range options: ±250, ±500, ±1000, and ±2000 degrees per second
        - Resolution: 16-bit ADC per channel

- Accelerometer Specifications:
  - Measures linear acceleration
  - Full scale range options: ±2g, ±4g, ±8g, and ±16g
  - Resolution: 16-bit ADC per channel
- Interface:
  - I2C protocol for communication (up to 400kHz Fast Mode)
  - Also compatible with SPI (not always available depending on the breakout board)
- Power Supply:
  - Voltage range: 2.3V to 3.4V for the MPU-6050 IC
  - Typically powered at 3.3V in most applications
  - Current consumption: ~3.9mA (active), 5μA (sleep mode)
- Digital Motion Processor (DMP):
  - On-chip DMP for motion processing, reducing load on the main controller
  - Computes complex motion algorithms (e.g., quaternion calculations)
- Temperature Range:
  - Operating temperature: -40°C to +85°C

2.3.3.1.2 HMC5883L Magnetometer:

The HMC5883L is a high-performance 3-axis magnetometer designed for precision magnetic field sensing and orientation detection. It operates over the I2C communication protocol and supports data output rates (ODR) up to 75 Hz.

The HMC5883L measures the strength and direction of magnetic fields along the X, Y, and Z axes, enabling accurate heading (yaw) calculation when combined with accelerometer and

gyroscope data. Its onboard circuitry provides signal conditioning and precise calibration for reliable magnetic field measurements.

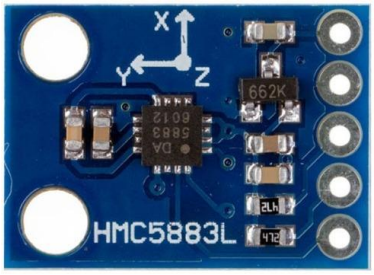


Figure 2.15HMC5883L

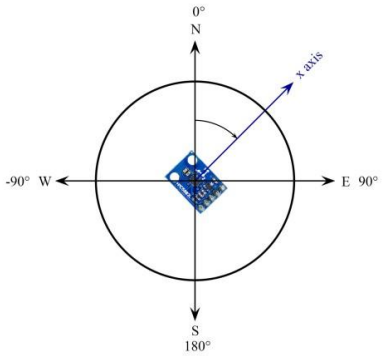


Figure 2.14 Directions of magnetic fields

- Key Features of HMC5883L:
    - Magnetometer: Measures magnetic field strength along the X, Y, and Z axes with a full-scale range of ±1.3 Gauss to ±8.1 Gauss.
    - Resolution: Outputs magnetic field data as 16-bit signed integers with a sensitivity of 0.92 mG/LSB for the default ±1.3 Gauss range.
    - Output Data Rate (ODR): Configurable ODR from 0.75 Hz to 75 Hz, suitable for a wide range of applications.
    - Power Supply: Operates at a voltage range of 2.16V to 3.6V, ensuring low power consumption.
- Role In the Project of (MPU6050 + HMC5883L):
    - The MPU6050 combines a 3-axis accelerometer and 3-axis gyroscope, making it capable of detecting angular velocity and linear acceleration. However, due to the gyroscope's drift and the accelerometer's susceptibility to noise, we will also integrate the HMC5883L, a 3-axis magnetometer, to enhance accuracy by providing reliable yaw measurements.

2.3.3.2 Water Depth and Water Pressure Sensor (MS5540C)

The MS5540C pressure sensor measures Absolute water or air pressure. It also measures temperature at the same time. So, you can use it for measuring water depth in the sea or water

level in a tank. It can also be used with different fluids measurements such as oil. You might use it also with air for measuring altitude (barometer) and temperature.

Figure 2.16 MS5540C

The MS5540C carries a metal protection cap filled with silicone gel for enhanced protection against water and humidity. The properties of this gel ensure function of the sensor even when in direct water contact. The MS5540C is qualified referring to the ISO Standard 2281 and can withstand a pressure of 100 m in salt water. Nevertheless, the user should avoid drying hard materials like for example salt particles on the silicone gel surface. In this case it is better to rinse with clean water afterwards.

The sensor measure water depth with a very good accuracy of 0.1 mbar (1 cm water level). The pressure module has a wide range of applications ranging from liquid level measurements in tanks, water depth and temperature in diving, snorkeling and ROV.

- Key Features of MS5540C:
    - Pressure Linear Range (Air Applications): 10 to 1100 mbar
    - Under Water Depth: can reach to 100m under some calibrations
    - Resolution 0.1 mbar (1 cm water)
    - 16 Bit ADC
    - Low voltage (2.2 to 3.6V)
    - Low power (Standby current:0.1uA)

- Temperature range -40 to 85°C

2.3.3.3 Temperature Sensor (DS18B20)

All submersible temperature probe, specifically designed to be used in the field. This sealed digital temperature probe lets you precisely measure temperatures in wet environments with an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from the output to obtain convenient Centigrade scaling.

Figure 2.17 DS18B20

The DS18B20 temperature sensor is enclosed in a sealed, waterproof housing, allowing it to measure the temperature of places away from the main controller or in wet conditions. The sensor has a temperature range up to 125 °C but the insulation is made of PVC, so it is recommended to keep it below 100 °C. It communicates via a 1-Wire interface and operates with voltages from 3 V to 5.5 V. The sensor is compatible with Arduino.

- Key Features DS18B20:
  - Usable temperature range: -55 to 125°C (-67°F to +257°F)
  - Waterproof
  - Temperature-limit alarm system
  - Multiple sensors can share one pin
  - ±0.5°C Accuracy from -10°C to +85°C

- Supply voltage: 3 V to 5.5 V
- Accuracy: +/- 0.5°C
- Resolution: from 9 to 12 bit
- Interface: 1-Wire
- Unique 64-bit identifier

- The corresponding wire colors are as follows:
  - Red = Vdd (3.0 to 5.5V). Typically, 3.3 or 5V to match the MCU it is being used with
  - Black = Ground. Must be common with the MCU
  - Yellow = DQ (1-Wire Data Bus) Connect to a digital I/O pin on MCU.

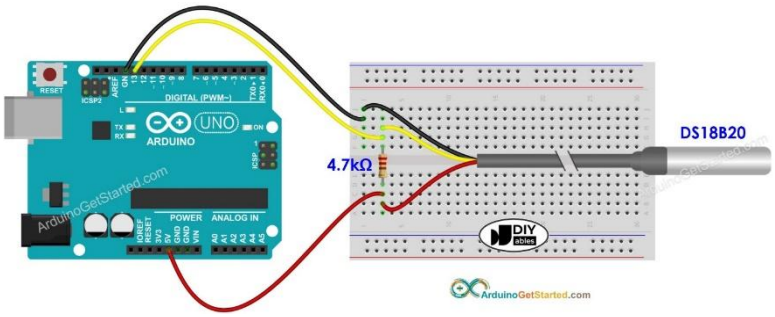


Figure 2.18 Connection DS18B20 with Arduino

- Role In the Project:

The temperature sensor senses the temperature of the water around the vehicle by sending signals to the Arduino. Then we can know the temperature value in the environment surrounding the vehicle and the plants around it.

2.3.3.4 Current Sensor (ACS712 (AC or DC) 30A):

This is a breakout board for the fully integrated Hall Effect based linear ACS712 current sensor. The sensor gives precise current measurement for both AC and DC signals. Thick copper

conductor and signal traces allows for survival of the device up to 5 times overcurrent conditions.

Figure 2.19 ACS712

The ACS712 Low Current Sensor Breakout outputs an analog voltage that varies linearly with sensed current. To calibrate, first set the output offset to the desired level (with zero current on the sense lines, read output with a DVM). Then with a known current input (a 100mA limited supply works well for this), set the output deflection with the gain pot. Sensitivity is then calculated as (Vref - Vdeflect)/(current input).

- Key Features ACS712:
    - x30B (30 Amp) version
    - Low noise analog signal path
    - Device bandwidth is set via the FILTER pin
    - 5us output rise time in response to input current
    - 1.5% output error at 25 degrees C
    - 1.2mOhm internal conductor resistance
    - 2.1 kVRMS minimum isolation voltage from pins 1-4 to pins 5-8
    - 5.0 VDC, single supply operation
    - 185 mV/A output sensitivity

- Role in the Project:
    - The current sensor measures the value of the current that is drawn from the battery, which feeds both controllers such as “Arduino - Raspberry pi” and feeds the sensors.

2.3.4 Actuators

2.3.4.1 Lumen Subsea Light for ROV/AUV

lumen subsea LED light with servo signal control, daisy-chain connection for multiple lights, and smart over-temp protection.

Yellow line, the red line is connected to the power supply at the same time, the light will be illuminated at the highest brightness, forced switch mode, etc. It takes 3-4s for the internal programs to judge the light after the mode is lit. When the lamp temperature exceeds 80 degrees Celsius, it will automatically lower the brightness to protect the LED light. When the temperature is lowered, the brightness will be restored.

Figure 2.20 Lumen Subsea Light

Specification:

- Power:20W
- Voltage: DC 12-28V
- Water depth: 300m at most
- brightness: 2200 lumens

- PWM range: 1100 – 1900
- Working temperature: -10~80 degree

- Role in the Project:
    - It provides lighting for the camera and vehicle

2.3.4.2 Movement System Component (Thruster- ESC-Joystick)

2.3.4.2.1 Thrusters (ApisQueen X3):

The thruster is a critical component of an ROV (Remotely Operated Vehicle), playing a vital role in providing propulsion and dynamic control. It serves as the 'muscle' that enables the vehicle to move and maneuver in underwater environments such as deep-sea exploration, oceanic research, and underwater missions.

Figure 2.21 Thruster

Functions:

- Propulsion
    - Thrusters generate the necessary force to move the vehicle in various directions: forward, backward, upward, downward, and laterally. This allows precise navigation through underwater environments.

- Maneuverability
    - Thrusters enable the ROV to make smooth and controlled movements in all directions, ensuring stability and control during operations such as inspections, surveys, and repairs.
- Positioning and Stability
    - Thrusters are critical for maintaining accurate positioning and stability, especially in environments with strong currents or turbulent waters.
- Redundancy and Reliability
    - ROVs are often equipped with multiple thrusters to ensure redundancy, allowing continued operations even if one thruster fails.
- Control in Harsh Conditions
    - Thrusters are designed to withstand deep-sea pressures and challenging underwater conditions, making them essential for deep-sea exploration and intervention tasks.
- Efficient Task Execution
    - With precise control provided by thrusters, tasks such as object manipulation, surveying, data collection, and repairs can be executed more effectively and efficiently

Key Features:

- Power: 260W
- Working Current: 8A
- Maximum thrust: 2.6Kg
- Propeller size: 60mm
- Weight: 98g

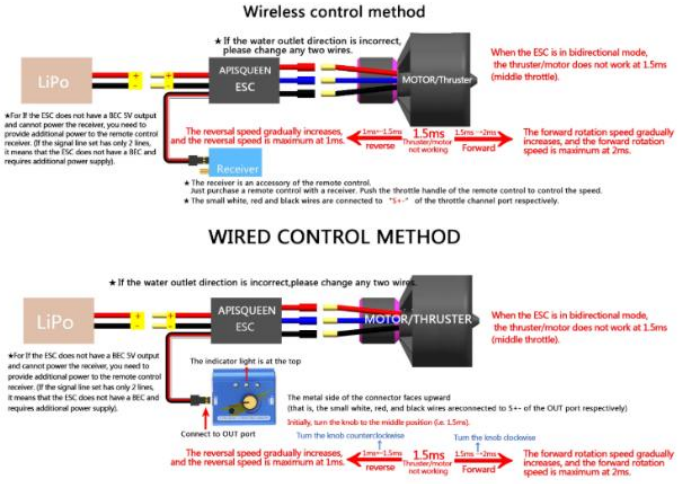


Figure 2.22 Thruster connection with ESC With LIPO

2.3.4.2.2 *Electronic Speed Controller "ESC" (30A – 6S* LIPO):

An Electronic Speed Controller (ESC) is a critical component in motor-driven systems, such as underwater vehicles (ROVs), and drones, robotics, its primary function is to regulate and control the speed and direction of motors or thrusters by adjusting the electrical power supplied to them.

Figure 2.23 ESC 30A

Functions:

- Speed Regulation:
    - The ESC modulates the motor's speed, allowing smooth and precise control over acceleration, deceleration, and dynamic movements. This ensures optimal performance in various applications.

- Direction Control:
    - Alongside speed regulation, the ESC manages the direction of motor rotation, enabling vehicles to move forward, backward, left, right, or rotate in any desired direction.
- Safety and Protection
    - ESC provides protection features such as over-current, over-voltage, and thermal management, ensuring the longevity and reliability of motors in demanding conditions.
- Feedback and Monitoring
    - Advanced ESCs offer real-time data feedback, enabling efficient system monitoring and integration with other control systems for enhanced functionality.

Key Features:

- Continuous current: 30A
- Input Voltage: 6-24V
- Instantaneous current: 40A (10 seconds)
- Size: 52.4*21.5*7mm
- Weight: 14g

2.3.4.2.3 Joystick (Logitech Extreme 3D Pro Joystick)

Fully equipped with precision twist rudder control, 12 programmable buttons, 8-way hat switch, rapid-fire trigger, contoured grip, and stable weighted base.

Figure 2.24 Logitech Extreme 3D Pro Joystick

Features:

- Precision twist rudder control
    - It's all in the wrist. Research has shown that advanced flying comes down to instincts and reaction time.

Figure 2.25 Joystick twist rudder control

- 12 programmable buttons
    - Extreme 3D Pro has every command at your fingertips and exactly where you want it so you can keep your eyes on the horizon. Each programmable button can be configured to execute simple single commands or intricate macros involving multiple keystrokes, mouse events, and more.

Figure 2.26 Joystick programmable buttons

- 8-way hat switch
    - Quickly and easily switch from points of view to weapons and more—the 8-way hat switch is designed to accurately capture specialized input specific to flight sims.

Figure 2.27 Joystick way hat switch

- Comfortable hand grip
    - Enjoy the flight even after hours of action. Sculpted curves support and form to your hand for hours of comfortable flying.

Figure 2.28 Joystick hand grip

- Stable, weighted base
    - Knows its place. An extensive study has shown that having a joystick sliding all over the place in the thick of the action is detrimental to performance. The heavy base will hold fast without tipping for lifting while you do your thing.

Figure 2.29Joystick base

- Method of connection:
    - Connected to Arduino with USB Host Shield Using Serial Communication Protocol
- Role in the project:
    - Used to control the ROV direction and the Camera angle of view

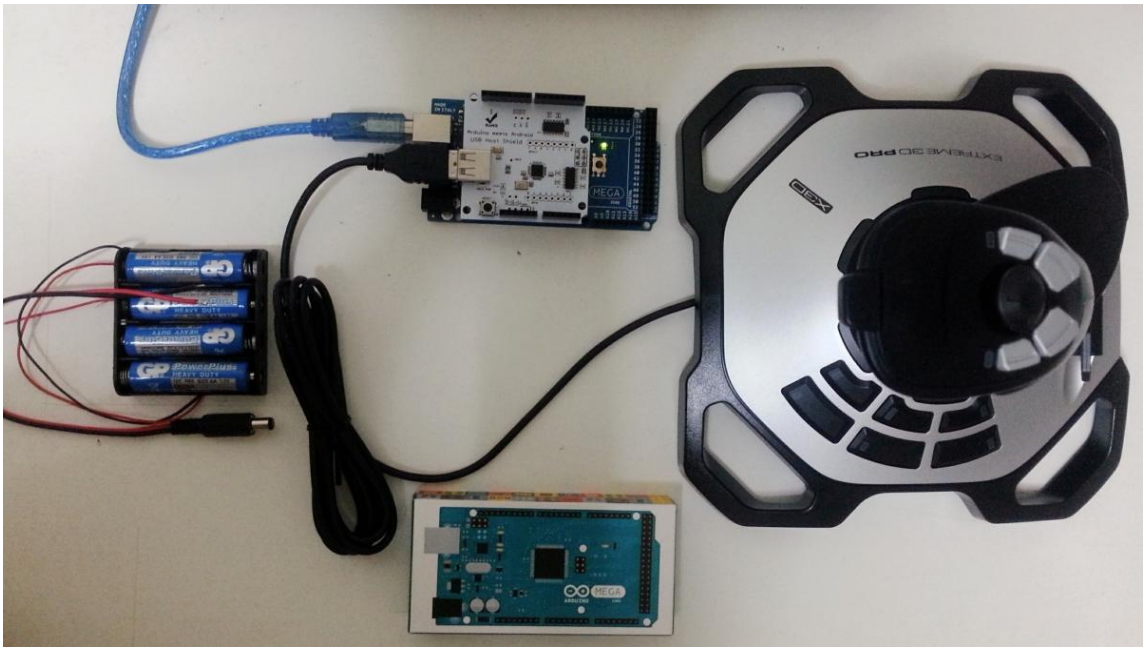


Figure 2.30 Joystick connection with Arduino

2.3.5 Connections

2.3.5.1 Ethernet cable

An Ethernet cable is a type of network cable used to connect devices within a local area network (LAN). It facilitates data transmission between devices like computers, microcontrollers, and routers, providing reliable and high-speed communication. For our project, a Cat6 shielded cable was selected due to its superior data transfer rate, durability, and resistance to electromagnetic interference (EMI), making it ideal for outdoor or underwater applications. Additionally, the cable is equipped with a waterproof jacket to ensure functionality in harsh environmental conditions.

Figure 2.31 Ethernet Cable

cable supports up to 1000Mbps data transfer (10x compared to Cat5e with 100Mbps) and 250MHz bandwidth (2.5x compared to Cat5e with 100MHz). It's a perfect choice for playing games, online video streaming, downloading, uploading, etc.

2.3.5.2 Motors Wires

- Type:
    - Wire 1.5mm^2

Figure 2.32 Wire

- Role in the project:
    - Wires with a cross section of 1.5 mm^2 will be used to connect the battery and the motor driver, as well as between the motor driver and the thruster, because this type of wire can withstand currents higher than 8 amps.

Wire Selection Factors:

- Current Capacity:

  - A wire's current-carrying capacity depends on its material (copper or aluminum), insulation, and cross-sectional area.

- Length of the Wire:
  - Longer wires have higher resistance, leading to voltage drops and heat generation. If the wire is long (e.g., >10m), a larger gauge is needed.

- Environment:
  - If the wire is used in high-temperature environments or enclosed spaces, you may need a thicker wire to avoid overheating.

Wire Recommendations for 8A Current:

- Copper Wire:
  - For 8 amperes, a 1.5mm² copper wire is sufficient for short runs (<10m).
  - Covered by (High quality PVC or Polyurethane (PU), which is suitable for marine environments).

Voltage Drop Calculation in a Wire

Voltage drop occurs when electrical current flows through a wire due to its resistance. The formula to calculate it is:

$$Vd = I \times R$$

Where:

- Vd Voltage drop (in volts).
- I: Current flowing through the wire (in amperes).
- R: Resistance of the wire (in ohms).

Calculate the Wire Resistance (R):

The resistance of a wire depends on its material, length, and cross-sectional area:

$$R = (\rho \times L)/A$$

Where:

- R: Resistance of the wire (in ohms).
- ρ: Resistivity of the wire material (in ohm·meters or ohm·mm²/m):
    - Copper: ρ≈1.68×10−8 Ω·m
- L: Total wire length (round trip: supply and return, in meters).
- A: Cross-sectional area of the wire (in square meters or square millimeters).

2.3.5.2.1 Ethernet switch

- Connects cabled devices (computers, Wi-Fi access points, PoE lighting, IoT devices, and servers) in an Ethernet LAN so they can communicate with each other and to the internet.

Figure 2.33 Ethernet switch

- Uses multiple ports to communicate between devices in the local area network (LAN).
- Operates by receiving data packets from connected devices and directing them to their intended destinations.
- Adds extra Ethernet ports to your router for wired internet connections.

2.3.6 Control board

The Control Board is a PCB that was designed based on Arduino Mega using proteus software to control and monitor the ROV functions such as:

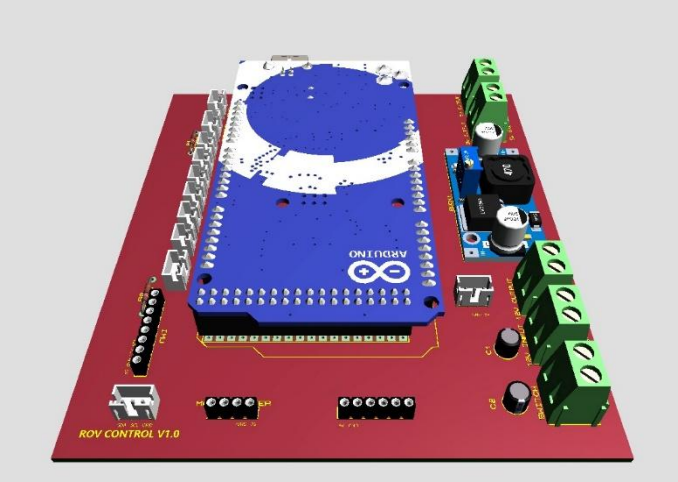


Figure 2.34 The Control board

i. Temperature

ii. Pressure

iii. Current

iv. Orientation

v. Camera Servos

vi. Thrusters (ESC)

vii. ROV Movement (Through I2C)

Key features:

- The Control board size is (120mm width – 111.5mm Height).
- Arduino Mega: Programable Microcontroller.
- Buck Converter: Used to provide required power for the circuit.
- C1, C2: Filter capacitors to smooth voltage and reduce noise.
- The main power tracks have Thickness (1mm).
- The signals tracks have Thickness (.6mm).

- The sensors power tracks have Thickness (.8mm).

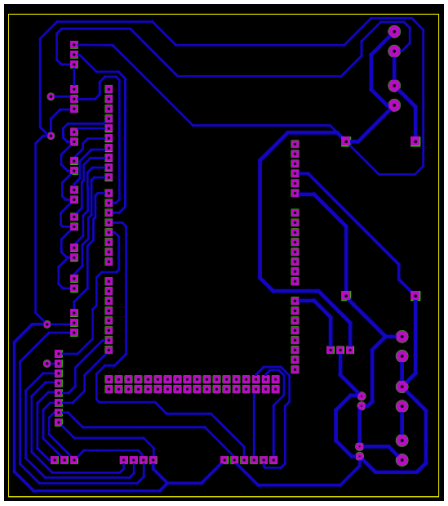


Figure 2.36 The PCB layout

Figure 2.35 Control Board 3D

Schematic

It shows the connection points between the points and each other, such as the connection of point MCLK in the pressure sensor and point 4 in the Arduino.

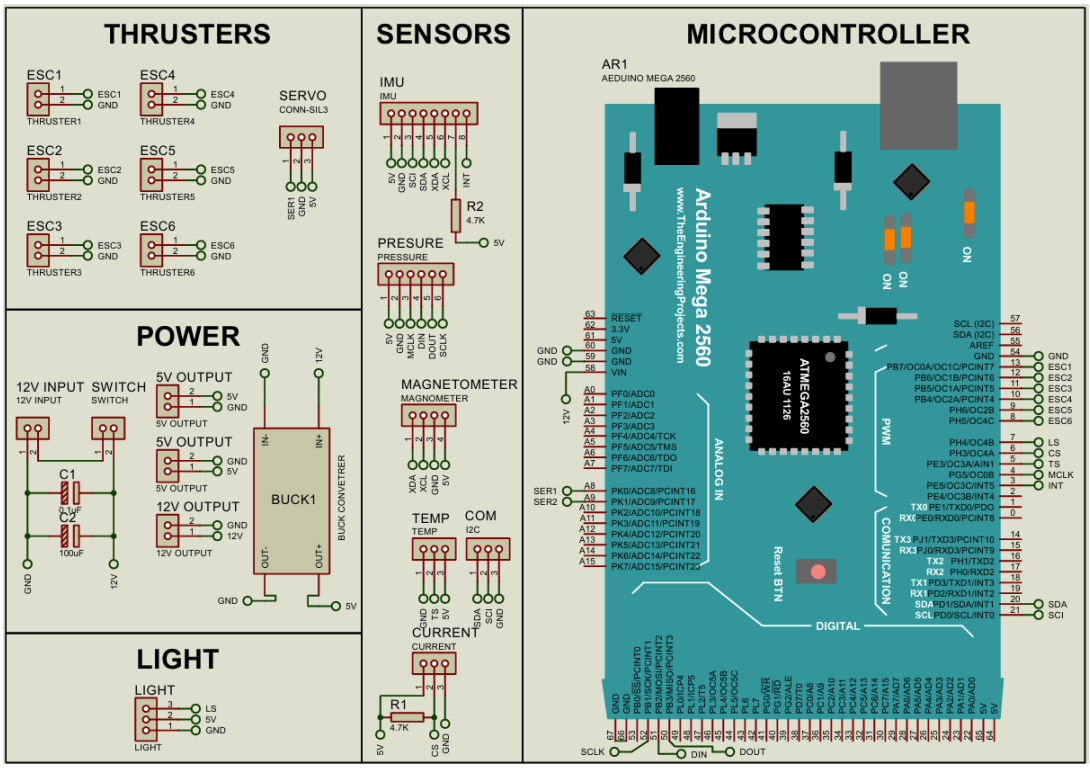


Figure 2.37 Control board schematic

Tracks Ready for printing

These tracks are ready for printing to be placed on the copper plate and then the plate is on reality.

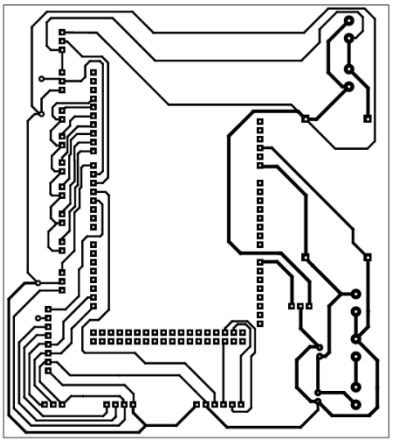


Figure 2.38 Control board Tracks Ready for printing

CHAPTER 3: Software

3.1 INTRODUCTION

This document provides a structured overview of the ROV software system, its core components, and operational logic. The ROV system is divided into two main subsystems: the Ground Station and the ROV, each playing distinct roles in operation and control.

3.1.1 Overview of ROV Software

The ROV software system is designed to provide precise control and data acquisition for underwater exploration and manipulation. It integrates various hardware and software components to ensure stability, responsiveness, and user-friendly operation.

3.1.1.1 Key Features:

- Modular and scalable architecture for future extensions.
- Real-time control and telemetry data processing.
- Integrated safety mechanisms to monitor current, depth, and operational parameters.
- PID-based control for stable navigation and depth control.

3.1.1.2 Core Subsystems:

- Ground Station: Responsible for user input and data visualization.
- ROV Unit: Executes commands, collects sensor data, and ensures precise actuator control.

3.1.2 Main ROV Logic

The ROV system operates by processing user commands at the Ground Station and executing them via the ROV's onboard systems

3.1.2.1 Ground Unit

The Ground Station acts as the control hub for the entire ROV system. It provides an interface for the operator to:

- Send Commands:
    - Joystick inputs for roll, pitch, yaw, and depth adjustments.
    - On/off toggles for lights, sport mode, and other features.
- Receive Data:
    - Sensor telemetry, including depth, orientation, current draw, and temperature.
    - Diagnostic messages for safety and operational feedback.

3.1.2.1.1 Ground Station Logic:

- Input Processing:
    - Read joystick inputs.
    - Normalize and map input values to control parameters (e.g., thruster speeds, light intensity).
- Command Transmission:
    - Send mapped control signals to the ROV via I2C communication.
- Data Visualization:
    - Display sensor readings and system status to the operator.

3.1.2.2 ROV Unit

The ROV is the operational component that receives commands from the Ground Station, collects environmental data, and performs necessary actions.

3.1.2.2.1 ROV Logic:

- Sensor Data Acquisition:
    - Use sensors such as IMU (MPU6050), depth sensors, current sensors, and temperature sensors to gather real-time data.

- Filter and process data for actionable insights (e.g., complementary filtering for orientation).

- Actuator Control:
  - Drive thrusters to achieve desired movement.
  - Control additional mechanisms like lighting and sport mode motor.
- PID Control:
  - Ensure stability and precise depth maintenance using PID algorithms for roll, pitch, yaw, and depth.
- Communication:
  - Receive user commands via I2C.
  - Transmit telemetry data and diagnostic messages back to the Ground Station.
- Safety Features:
  - Monitor current levels to avoid overdraw (using a predefined `SAFEST_CURRENT` limit).
  - Restrict depth to a safe operational range (`maxDepth` and `minDepth`).
  - Shut down thrusters or actuators during critical conditions.

3.1.2.3 Code Overview

3.1.2.3.1 Header File

The header file defines all necessary interfaces, constants, and function declarations used for the sensors and actuators.

3.1.2.3.1.1 Key Components:

- Sensors:
  - IMU (MPU6050): For orientation and motion tracking.
  - HMC5883L: Magnetometer for compass direction.
  - Pressure Sensor: Measures depth.

  - Temperature Sensor: Monitors environmental and system temperatures.
  - Current Sensor: Tracks power usage.
- Actuators:
  - Thrusters: Control movement and orientation.
  - Lighting System: For illumination.
  - Sport Mode Stepper Motor: Provides high-performance control in sport mode.
- PID Controllers:
  - Roll, pitch, yaw, and depth controllers for precise movements.
- Function Prototypes:
  - Initialization routines for sensors and actuators.
  - Joystick input polling.
  - Thruster calculation and control logic.
  - Safety monitoring and emergency shutdown.

3.1.2.3.2 Source File

The source file implements the logic defined in the header file, managing sensor data processing, control logic, and actuator operations.

3.1.2.3.2.1 Key Implementations:

- Sensor Initialization:
  - Set up communication and calibration for all sensors.
  - Configure data filtering algorithms for smooth operation.
- Joystick Handling:
  - Map joystick inputs to desired setpoints for the PID controllers.
  - Ensure input values are within operational limits.
- PID Control Loop:
  - Compute PID outputs for thruster adjustments.
  - Continuously monitor error values and update actuator commands in real-time.
- Actuator Logic:
  - Translate PID outputs into motor speed and direction.
  - Control lighting intensity and sport mode motor.
- Safety Mechanisms:
  - Monitor current levels to detect overdraw and trigger shutdown.

- Enforce depth limits to prevent damage to the ROV.
- Stop thrusters when critical conditions are detected.

3.1.2.3.3 Communication Protocols

The ROV and Ground Station communicate using the I2C protocol.

- Data Flow:
  - Ground Station sends control commands.
  - ROV responds with telemetry data.
- Message Format:
  - Commands: [Control_ID | Parameter_1 | Parameter_2 | …]
  - Telemetry: [Sensor_ID | Data_1 | Data_2 | …]

3.1.2.3.4 Safety and Diagnostics

The software includes robust safety features to ensure reliable operation:

- Current Monitoring:
  - Prevents motor overdraw and power system overload.
- Depth Restrictions:
  - Avoids exceeding safe operational depth.
- Emergency Shutdown:
  - Stops thrusters during critical failures.
- Diagnostic Feedback:
  - Continuously logs and transmits system health data.

3.2 MICROCONTROLLERS AND IDES

3.2.1 Arduino

Arduino is an open-source hardware and software platform designed for creating interactive projects. It is built around microcontrollers, which are small computers capable of running embedded software. Arduino boards are widely used in robotics, IoT, and automation due to their ease of use, extensive community support, and versatility.

3.2.1.1 Why using Arduino Mega?

The Arduino Mega 2560 is a powerful microcontroller board based on the ATmega2560 chip. It is well-suited for projects that require multiple inputs and outputs, complex computations, or handling a variety of peripherals simultaneously.

- Ample Input/Output Pins:
    - 54 digital I/O pins and 16 analog input pins accommodate a wide range of sensors and actuators.
    - Multiple PWM-enabled pins are ideal for motor control and thrusters, crucial for ROV movement.
- Memory Capacity:
    - 256 KB of flash memory allows for storing large and complex programs, including sensor processing, motor control algorithms, and communication protocols.
    - 8 KB SRAM supports dynamic data handling, essential for real-time control systems.
- Communication Support:
    - Includes UART, SPI, and I2C interfaces for seamless integration with sensors, motor drivers, and other peripherals.
    - USB port for easy programming and data communication with a host computer.
- Ease of Integration:
    - Extensive library support simplifies coding for interfacing with depth sensors, pressure sensors, and IMUs used in ROVs.
    - Compatible with shields and modules for wireless communication (e.g., Bluetooth, Wi-Fi), which can enhance ROV control.
- Scalability and Expansion:

  - Ideal for scaling the project to include advanced features such as automated navigation or additional sensor arrays.

- Cost-Effectiveness:
  - Affordable compared to other microcontroller platforms with similar capabilities.
  - Open-source nature ensures availability of documentation, tutorials, and community support.
- Reliability:
  - Proven track record in underwater robotics and similar projects due to its robust design and stability.

3.2.2 Arduino IDE

The Arduino Integrated Development Environment (IDE) is a software tool used to write, compile, and upload code to Arduino boards. It is beginner-friendly and supports a wide range of microcontrollers. Features of the Arduino IDE include:

- Simple Interface: Intuitive design that simplifies coding for beginners and experts alike.
- Cross-Platform Support: Available for Windows, macOS, and Linux.
- Extensive Libraries: Pre-built libraries for sensors, actuators, and communication protocols.
- Serial Monitor: A built-in tool for debugging and monitoring data.
- Community Contributions: Plugins and extensions developed by the Arduino community.

3.2.3 Raspberry Pi

The Raspberry Pi plays a critical role in the ROV system, acting as a processing hub for the camera and other connected tools. It is strategically placed within the ROV near the camera to ensure efficient data handling with minimal transmission losses. This proximity enables rapid data reception and processing, making the Raspberry Pi an indispensable component for underwater operations.

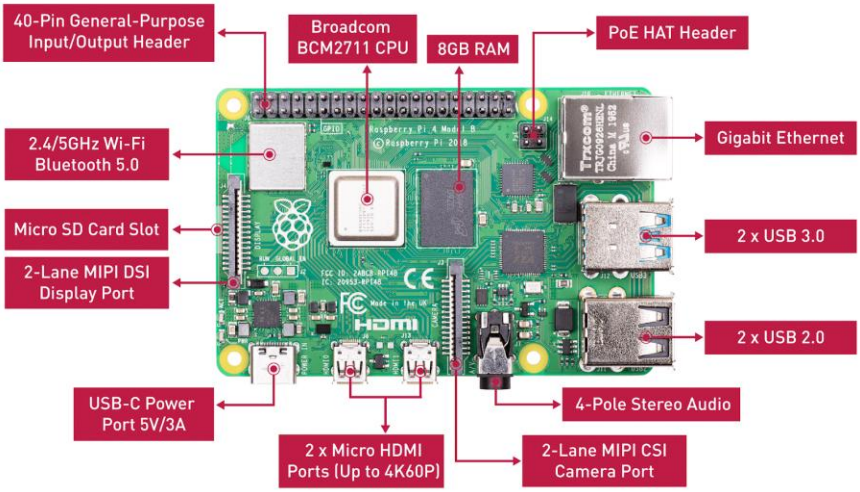


Figure 3.1 Raspberry Pi architecture

3.2.3.1 Raspberry Pi Functionality:

3.2.3.1.1 Data Reception

- The Raspberry Pi is directly connected to the camera via an Ethernet cable, ensuring high-speed and reliable data transfer.
- This setup minimizes data loss during transmission, which is crucial for underwater video streaming and analysis.

3.2.3.1.2 Data Processing and Analysis

- The Raspberry Pi processes the incoming data from the camera in real time.
- It performs essential tasks such as image and video analysis, object detection (integrated with the YOLOv8 model), and data preprocessing for further use by the control station.

3.2.3.1.3 Integration with Onboard Tools

- Positioned alongside other tools within the ROV, the Raspberry Pi coordinates seamlessly with these systems, enhancing the ROV's overall functionality.

3.2.4 Object Detection IDEs

3.2.4.1 Development Environment

For the implementation of the YOLOv8 object detection model, the following tools and technologies were utilized:

- IDE:
    - The development was carried out using Visual Studio Code, an efficient and versatile code editor with extensive support for Python and debugging tools.
- Programming Language:
    - The project uses Python 3.10, a robust and widely adopted language for machine learning and computer vision applications.

3.2.4.2 Libraries and Dependencies

To develop the YOLOv8 object detection system, several Python libraries were utilized, each serving a specific purpose:

- Ultralytics:
    - This library is used for implementing and running the YOLOv8 model. It includes tools for training, validation, and real-time object detection tasks.
- Multiprocessing:
    - This module facilitates the efficient handling of multiple processes, particularly in systems requiring parallel computations or processes, such as object detection tasks.
    - The function freeze_support() is included to ensure compatibility when the script is run on Windows systems

3.3 SENSOR INTEGRATION

3.3.1 Current Sensor

The ACS712 30A is a Hall-effect-based current sensor capable of measuring AC and DC currents up to 30A. It provides an analog output voltage proportional to the current flowing through the sensor.

3.3.1.1 Sensor Purpose

A current sensor is a device that detects electric current in a wire and generates a signal proportional to that current. The generated signal can be:

- Analog voltage or current for direct measurement.
- Digital output for processing in microcontrollers or data acquisition systems.

The output signal can be used for:

- Displaying the measured current on an ammeter.
- Storing data for further analysis.
- Implementing control mechanisms in systems like the ROV.

3.3.1.2 Interfacing the ACS712 Sensor with the Microcontroller

The ACS712 sensor outputs an analog voltage proportional to the current passing through it. This output is connected to the ADC (Analog-to-Digital Converter) pin of the microcontroller for processing.

3.3.1.3 Current Calculation Formula

The current can be calculated using the following formula:

$$Current\ (A) = \frac{(ADC\ Value\ -\ Zero\ Current\ ADC\ Value)}{ADC\ Resolution} \times \frac{VCC}{Sensitivity}$$

Where:

- Zero Current ADC Value: The ADC value when no current is flowing (e.g., 512 for 2.5V at 10-bit resolution).
- ADC Resolution: 1023 for a 10-bit ADC.

- VCC: Supply voltage (5V).
- Sensitivity: 66 mV/A.

3.3.1.4 Pseudocode

- Main File

```
BEGIN
  // Define constants and variables
  CURRENT_PIN = A0
  INPUT_VOLTAGE = 5.0
  ARDUINO_ADC = 1023
  SENSOR_SCALE_FACTOR = 66
  current_u16Interval = 1000
  SAFEST_CURRENT = 8
  MIDPOINT_CYCLES = 100
  READING_CYCLES = 10

  // Initialize sensor and serial communication
  current_sensor = ACS712(CURRENT_PIN, INPUT_VOLTAGE, ARDUINO_ADC,
SENSOR_SCALE_FACTOR)
  Serial.begin(9600)

  // Calibrate sensor midpoint
  current_sensor.autoMidPoint(MIDPOINT_CYCLES)
  PRINT current_sensor.getMidPoint()

  // Main loop
  WHILE TRUE
    current_u32CurrentMillis = millis()

    IF current_u32CurrentMillis - current_u32PreviousMillis >=
current_u16Interval
      current_u32PreviousMillis = current_u32CurrentMillis
      Local_u8DCReading = current_sensor.mA_DC(READING_CYCLES)  // Take
average of 10 cycles
      PRINT Local_u8DCReading
```

```
      IF Local_u8DCReading > SAFEST_CURRENT
        // Stop motor or take action
      END IF
    END IF
  END WHILE
END
```

3.3.2 MPU6050

3.3.2.1 Introduction

The MPU6050 is a 6-axis motion tracking device that integrates a 3-axis gyroscope and a 3-axis accelerometer on a single chip. It is commonly used in applications like robotics, drones, game controllers, and ROVs due to its compact size, low power consumption, and accuracy.

3.3.2.2 Operation

To understand the operation of the MPU-6050, let us break it down step-by-step, beginning with raw data acquisition from the gyroscope and accelerometer, their conversion into usable units (degrees and g), and finally the application of a complementary filter for accurate orientation estimation.

3.3.2.2.1 Raw Data Acquisition

The MPU-6050 measures angular velocity using its gyroscope and linear acceleration using its accelerometer. The raw data is accessed via I2C or SPI communication. The device registers hold the 16-bit signed integers for each axis of the gyroscope and accelerometer.

- Gyroscope Data:
  The gyroscope provides raw angular velocity in counts per second for each axis (X, Y, Z). These values represent the rate of rotation about each axis.

- Accelerometer Data:
  The accelerometer provides raw acceleration in counts for each axis (X, Y, Z). These values represent linear acceleration, including the influence of gravity.

Register Mapping:

- Gyroscope raw data: Registers `GYRO_XOUT_H`, `GYRO_XOUT_L`, etc.

- Accelerometer raw data: Registers `ACCEL_XOUT_H`, `ACCEL_XOUT_L`, etc.

Raw data values are signed integers ranging from -32768 to 32767.

3.3.2.2.2 Conversion of Raw Data

The raw values must be converted into meaningful units like degrees per second (for gyroscope) and g (for accelerometer).

Gyroscope Conversion:

The gyroscope's raw data is converted into degrees per second using the selected full-scale range (±250, ±500, ±1000, or ±2000 dps). The conversion formula is:

$$Angular\ Velocity\ (dps) = \frac{Raw\ Gyro\ Value}{Gyro\ Sensitivity}$$

- Gyro Sensitivity values depend on the full-scale range:
    - ±250 dps → 131 LSB/dps
    - ±500 dps → 65.5 LSB/dps
    - ±1000 dps → 32.8 LSB/dps
    - ±2000 dps → 16.4 LSB/dps

Accelerometer Conversion:

The accelerometer's raw data is converted into g (gravitational force units) using the selected full-scale range (±2g, ±4g, ±8g, or ±16g). The conversion formula is:

$$Acceleration\ (g) = \frac{Raw\ Accel\ Value}{Accel\ Sensitivity}$$

- Accel Sensitivity values depend on the full-scale range:
    - ±2g → 16384 LSB/g
    - ±4g → 8192 LSB/g
    - ±8g → 4096 LSB/g

- ±16g → 2048 LSB/g

3.3.2.2.3 Combining Gyroscope and Accelerometer Data

The gyroscope and accelerometer are complementary sensors:

- Gyroscope measures angular velocity and calculates the angular position by integrating over time.
- Accelerometer measures the direction of gravity to estimate tilt.

Each sensor has limitations:

- The gyroscope drifts over time.
- The accelerometer is noisy and affected by external accelerations.

To overcome these limitations, a Complementary Filter is applied.

3.3.2.2.4 Applying the Complementary Filter

The complementary filter combines the short-term stability of the gyroscope and the long-term stability of the accelerometer to calculate a stable tilt angle.

Calculating Tilt Angle from Accelerometer:

Using trigonometric functions, the tilt angles in the pitch (x-axis) and roll (y-axis) directions are calculated as:

$$Pitch\ (Accel) = arctan\left(\frac{AccelY}{\sqrt{AccelX^2 + AccelZ^2}}\right)$$

$$Roll\ (Accel) = arctan\left(\frac{-AccelX}{\sqrt{AccelY^2 + AccelZ^2}}\right)$$

Calculating Tilt Angle from Gyroscope:

The gyroscope's angular velocity is integrated over time to estimate the angular position:

$$Angle\ (Gyro) = Previous\ Angle + (Angular\ Velocity \times \Delta t)$$

Where Δt is the time elapsed between measurements.

Combining with Complementary Filter:

The complementary filter blends the accelerometer and gyroscope data to get a stable angle estimate:

$$Filtered\ Angle = \alpha(Angle\ (Gyro)) + (1 - \alpha)(Angle\ (Accel))$$

- α is the filter constant, typically between 0.95 and 0.99, to prioritize gyroscope data.

3.3.2.3 Software Interface

3.3.2.3.1 Dependencies

To interact effectively with the MPU6050 and enhance its functionality, several libraries are used in the software architecture. Each library serves a specific purpose:

- Core MPU6050 Libraries
    - `Adafruit_Sensor`: Provides a unified sensor interface that standardizes the way sensors like the MPU6050 are managed in software.
    - `Adafruit_BusIO`: Facilitates low-level communication via I2C or SPI protocols, ensuring efficient and error-free data transfer between the microcontroller and the sensor.
- Auxiliary Libraries for Data Visualization
    - `Adafruit_GFX_Library`: A graphics library for rendering text, shapes, and bitmaps on displays. It is used to visually represent data from the MPU6050, such as accelerometer readings or angular tilt.
    - `Adafruit_SSD1306`: Specifically designed for OLED displays, this library renders MPU6050 output on small OLED screens, providing real-time monitoring of motion data.

You can download it by opening the Arduino libraries in the sidebar and searching for these libraries to download them.

3.3.2.3.2 Interface

- Install Libraries:
    - Open Arduino IDE and install the `MPU6050_tockn` library using the Library Manager.
    - Verify that the `Wire` library is already included with yo.
- Setup Code:
    - Include the `MPU6050_tockn.h` and `Wire.h` headers in your project.
    - Create an instance of the `MPU6050` class, passing the `Wire` object for I2C communication.
- Initialize the Sensor:
    - Start the Serial communication for debugging.
    - Use `Wire.begin()` to start the I2C interface.
    - Initialize the MPU6050 sensor with `mpu6050.begin()`.
- Calibrate Gyroscope Offsets:
    - Use the `calcGyroOffsets(true)` function if needed to measure offsets while the sensor is stationary.
    - Alternatively, set the offsets manually using `setGyroOffsets(x, y, z)`.
- Read and Print Sensor Data:
    - Call `mpu6050.update()` in the `loop` function to fetch the latest sensor data.
    - Use the library's built-in functions to get filtered angles (pitch, roll, yaw) via `getAngleX()`, `getAngleY()`, and `getAngleZ()`.

3.3.2.3.3 Pseudocode

- Main File

```
BEGIN InterfaceWithMPU6050

    // Step 1: Install Required Libraries
    // a. Open Arduino IDE.
    // b. Go to Sketch > Include Library > Manage Libraries.
    // c. Search for "MPU6050_tockn" and install it.
    // d. Ensure the "Wire" library is pre-installed (default with Arduino
IDE).

    // Step 2: Include Libraries
    INCLUDE MPU6050_tockn.h
    INCLUDE Wire.h

    // Step 3: Initialize MPU6050 Object
    CREATE mpu6050 OBJECT USING Wire

    // Step 4: Define Global Timer
    INITIALIZE MPU_longTimer TO 0

    // Step 5: Setup Function
    FUNCTION setup()
        BEGIN
            // Initialize Serial Communication
            CALL Serial.begin WITH PARAMETER (9600)

            // Initialize I2C Communication
            CALL Wire.begin

            // Initialize MPU6050
            CALL mpu6050.begin

            // Optional: Measure Gyroscope Offsets
            // a. Place MPU6050 in a stationary, level position.
            // b. Uncomment the following line to calculate offsets:
            //    CALL mpu6050.calcGyroOffsets WITH PARAMETER (true)
            // c. After obtaining offsets, use the setGyroOffsets function
below.
```

```
            // Set Manual Gyroscope Offsets (Obtained from Measurements)
            CALL mpu6050.setGyroOffsets WITH PARAMETERS (-5.49, 0.64, -0.75)
        END FUNCTION

    // Step 6: Loop Function
    FUNCTION loop()
        BEGIN
            // Update MPU6050 Sensor Data
            CALL mpu6050.update

            // Check Timer for Output Interval
            IF (CURRENT_TIME - MPU_longTimer > 1000) THEN
                BEGIN
                    // Fetch Filtered Angles from MPU6050
                    SET angleX TO CALL mpu6050.getAngleX
                    SET angleY TO CALL mpu6050.getAngleY
                    SET angleZ TO CALL mpu6050.getAngleZ

                    // Print Filtered Angles (Pitch, Roll, Yaw)
                    CALL Serial.print WITH PARAMETER ("angleX: ")
                    CALL Serial.print WITH PARAMETER (angleX)
                    CALL Serial.print WITH PARAMETER ("\tangleY: ")
                    CALL Serial.print WITH PARAMETER (angleY)
                    CALL Serial.print WITH PARAMETER ("\tangleZ: ")
                    CALL Serial.println WITH PARAMETER (angleZ)

                    // Update Timer
                    SET MPU_longTimer TO CURRENT_TIME
                END
            END IF
        END FUNCTION

END InterfaceWithMPU6050
```

3.3.2.4 Problems and Solutions

Yaw angle, representing rotation around the vertical axis, is crucial in applications like navigation and robotics. Calculating yaw using gyroscope data alone involves integrating angular velocity over time. However, due to small errors in gyroscope readings (bias and noise), the

integration accumulates these errors, leading to cumulative drift. Over time, the yaw angle becomes increasingly unreliable, especially for long-duration operations.

- Detailed Analysis
    - Gyroscope Contribution:
        - The gyroscope measures angular velocity (ω) in degrees per second (or radians per second).
        - Yaw is derived by integrating this angular velocity over time:

$$Yaw\ (Gyro) = \int \omega z\, dt$$

    - Any bias or noise in ωz gets integrated, causing the yaw value to drift.
    - Lack of Absolute Reference:
        - Unlike pitch and roll, which can be stabilized using accelerometer data due to gravity, yaw lacks a natural stabilizing reference. This absence makes it more prone to drift.
    - Impacts of Drift:
        - Drift leads to incorrect orientation, causing errors in navigation and control systems.
        - For instance, a robot using yaw for direction will deviate from its intended path.
- Solution

To mitigate cumulative drift, a magnetometer can be combined with the gyroscope data. The magnetometer measures the Earth's magnetic field and provides an absolute reference for the yaw angle.

    - Yaw Calculation Using a Magnetometer:

- The magnetometer outputs the magnetic field components Mx and My in the horizontal plane.
- The yaw angle is calculated as:

$$Yaw\ (Magnetometer) = \arctan 2(My, Mx)$$

- This provides an absolute yaw measurement, free from drift.

- Complementary Filter Integration:
  - To combine the short-term accuracy of the gyroscope with the long-term stability of the magnetometer, a complementary filter is used:

$$Filtered\ Yaw = \alpha(Yaw\ (Gyro)) + (1 - \alpha)(Yaw\ (Magnetometer))$$

- Here:
  - α is a tuning parameter (0.98) that balances responsiveness and stability.
  - The gyroscope contributes rapid changes, while the magnetometer corrects drift over time.

By integrating a magnetometer and applying a complementary filter, the cumulative drift problem in yaw calculations is effectively resolved, enhancing the overall reliability of orientation systems

3.3.3 HMC5883L 3-Axis Magnetometer Sensor

3.3.3.1 Overview

The HMC5883L is a 3-axis digital magnetometer designed to measure magnetic fields in three perpendicular axes (X, Y, and Z). It is widely used in applications such as:

- Compassing: Determining heading or direction relative to the Earth's magnetic field.
- Navigation: Assisting in orientation and positioning systems.
- Robotics: Enabling robots to sense and navigate their environment.

- Consumer Electronics: Used in smartphones, tablets, and wearables for orientation sensing.

The sensor communicates via the I2C protocol, making it easy to interface with microcontrollers like Arduino.

3.3.3.2 Operating Principle

3.3.3.2.1 Magnetic Field Measurement

The HMC5883L measures the Earth's magnetic field using its magneto resistive sensors. Each axis (X, Y, Z) outputs a voltage proportional to the magnetic field strength. The ADC converts these voltages into digital values, which are then processed to calculate the heading or direction.

3.3.3.2.2 Heading Calculation

The heading (direction) is calculated using the arctangent of the Y and X axis data:

$$Heading = arctan\,(\frac{Y}{X})$$

- If $X > 0$, the heading is:

$$Heading = 90° - arctan\,(\frac{Y}{X})$$

- If $X < 0$, the heading is:

$$Heading = 270° - arctan\,(\frac{Y}{X})$$

- If $X = 0$:
    - If $Y < 0$, the heading is $180°$.
    - If $Y > 0$, the heading is $0°$.

3.3.3.2.3 Measurement Modes

- Single-Measurement Mode:
    - The sensor takes a single reading and then enters idle mode.

- Suitable for low-power applications.

- Continuous-Measurement Mode:
  - The sensor continuously takes readings at a specified data rate.
  - Ideal for real-time applications.

3.3.3.3 Calibration

The HMC5883L sensor may have inherent offsets and sensitivity variations due to manufacturing tolerances and external magnetic interference. Calibration ensures accurate and reliable measurements.

3.3.3.3.1 Calibration Procedure

- Collect Raw Data:
  - Rotate the sensor slowly in all directions (360° in the X-Y plane).
  - Record the raw X, Y, and Z values for multiple orientations.

- Calculate Offsets:
  - For each axis, calculate the average of the minimum and maximum values:

$$Offset = \frac{Max\ Value + Min\ Value}{2}$$

- Calculate Scale Factors:
  - Normalize the data to account for sensitivity differences between axes:

$$Scale\ Factor = \frac{Reference\ Value}{Max\ Value - Min\ Value}$$

- Apply Calibration:
  - Subtract the offsets and multiply by the scale factors to correct the raw data:

$$Corrected\ Value = (Raw\ Value - Offset) \times Scale\ Factor$$

3.3.3.4 Software Interface

- Initialization
    - Configure the I2C communication protocol.
    - Set the sensor's configuration registers (e.g., data rate, gain, measurement mode).
- Data Acquisition
    - Read the raw magnetic field data from the X, Y, and Z axis registers.
    - Apply calibration coefficients to compensate for sensor offsets and sensitivity variations.
    - Calculate the heading using the arctangent function.

3.3.3.4.1 Pseudocode

- Header File

```
DECLARE FUNCTION initHMC5883L()
    DESCRIPTION: Initializes the HMC5883L sensor and configures its settings.

DECLARE FUNCTION setMeasurementMode(mode)
    INPUT: mode (Integer)
    DESCRIPTION: Sets the measurement mode (single or continuous).

DECLARE FUNCTION setDataRate(rate)
    INPUT: rate (Integer)
    DESCRIPTION: Sets the data output rate.

DECLARE FUNCTION setGain(gain)
    INPUT: gain (Integer)
    DESCRIPTION: Sets the sensor gain.

DECLARE FUNCTION readRawData(data)
    INPUT: data (Array of 3 integers)
    DESCRIPTION: Reads raw magnetic field data from the X, Y, and Z axes.

DECLARE FUNCTION calibrateSensor(offsets, scaleFactors)
```

```
    INPUT: offsets (Array of 3 floats), scaleFactors (Array of 3 floats)
    DESCRIPTION: Applies calibration offsets and scale factors to raw data.

DECLARE FUNCTION calculateHeading(x, y) RETURNS Float
    INPUT: x (Float), y (Float)
    DESCRIPTION: Calculates the heading from X and Y axis data.

DECLARE FUNCTION getHeading() RETURNS Float
    DESCRIPTION: Returns the current heading in degrees.
```

- Source File

```
BEGIN PROGRAM

DEFINE I2C_ADDRESS = 0x1E
DEFINE offsets AS float array of size 3 initialized to 0
DEFINE scaleFactors AS float array of size 3 initialized to 1

FUNCTION initHMC5883L()
    INITIALIZE I2C communication
    SET configuration registers:
        - Measurement mode: Continuous
        - Data rate: 15 Hz
        - Gain: 1090 LSB/Gauss
END FUNCTION

FUNCTION setMeasurementMode(mode)
    WRITE mode to measurement mode register
END FUNCTION

FUNCTION setDataRate(rate)
    WRITE rate to data rate register
END FUNCTION

FUNCTION setGain(gain)
    WRITE gain to gain register
END FUNCTION

FUNCTION readRawData(data)
```

```
    READ 6 bytes from data output registers
    COMBINE bytes into X, Y, and Z axis data
    STORE data in the provided array
END FUNCTION

FUNCTION calibrateSensor(offsets, scaleFactors)
    FOR each axis (X, Y, Z) DO
        APPLY offset and scale factor to raw data
    END FOR
END FUNCTION

FUNCTION calculateHeading(x, y)
    RETURN atan2(y, x) * 180 / PI
END FUNCTION

FUNCTION getHeading()
    DECLARE rawData AS integer array of size 3
    CALL readRawData(rawData)
    CALL calibrateSensor(offsets, scaleFactors)
    RETURN calculateHeading(rawData[0], rawData[1])
END FUNCTION

END PROGRAM
```

- Main File

```
INCLUDE LIBRARY: HMC5883L.h

DEFINE FUNCTION setup()
    BEGIN
        START serial communication at 9600 baud rate.
        CALL initHMC5883L() to initialize the sensor.
    END

DEFINE FUNCTION loop()
    BEGIN
        PRINT "Heading = " to Serial Monitor.
        PRINT RETURN VALUE of getHeading() to Serial Monitor.
```

```
        WAIT for 1000 milliseconds.
    END
```

3.3.4 Temperature Sensor

3.3.4.1 Sensor Purpose

The DS18B20 temperature sensor is integrated into the ROV to monitor environmental conditions critical to the system's operation. Its primary purpose is to ensure thermal conditions for marine creatures, especially underwater coral reefs. Temperature data is also used to issue alerts if thresholds are exceeded and contribute to long-term performance logging.

3.3.4.2 Software Requirements

To interface with the DS18B20 temperature sensor, the following software tools and configurations are required:

- Libraries:
    - OneWire: Manages the 1-Wire communication protocol for data transfer.
    - Dallas Temperature: Provides high-level functions for working with the DS18B20.
- Microcontroller Resources:
    - One GPIO pin for communication.
    - Digital pin configured for bidirectional communication on the 1-Wire bus.

3.3.4.3 Overview

The DS18B20 is a 1-Wire digital temperature sensor designed to provide precise temperature readings with minimal wiring. It features four primary data components:

- 64-bit Lasered ROM:
    - Uniquely identifies the sensor on a shared 1-Wire bus, enabling multiple sensors to operate simultaneously on a single communication line. This ROM allows addressing specific sensors for commands.
- Temperature Sensor:

- Captures temperature measurements and converts them to digital format with a configurable resolution (9 to 12 bits).

- Nonvolatile Temperature Alarm Triggers (TH and TL):
  - Stores threshold values for temperature alarms. These registers are nonvolatile and persist even after power loss. They can be repurposed as general-purpose memory.

- Configuration Register:
  - Contains resolution settings (affecting conversion time) and other control parameters.

The DS18B20 supports two power modes:

- Parasite Power Mode:
  - Utilizes energy stored in an internal capacitor charged during high states of the 1-Wire signal.

- External Power Mode:
  - Operates with an external 3V–5.5V supply.

Figure (3.2) explains the sensor's internal architecture and we will discuss later how to use the sensor simply.

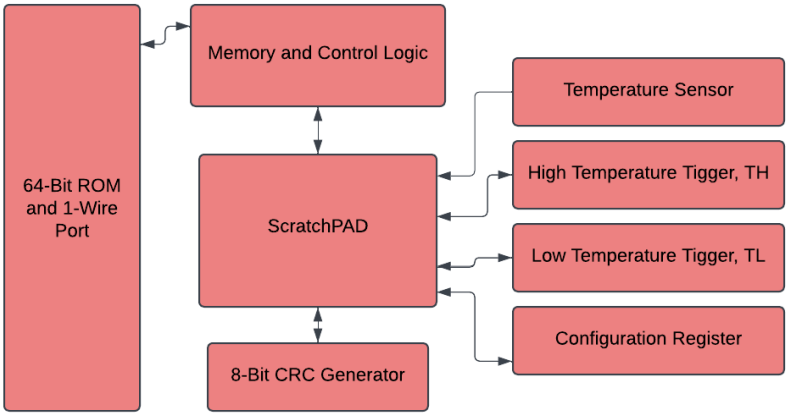


Figure 3.2 Sensor's internal architecture

3.3.4.4 Software Architecture

The temperature sensor's role within the software architecture can be divided into the following layers:

- Data Acquisition Layer:
    - Handles the communication with the DS18B20 sensor using the OneWire protocol.
    - Ensures periodic data requests based on system requirements.
- Processing Layer:
    - Converts raw temperature readings to human-readable formats (Celsius and Fahrenheit).
    - Applies filtering or averaging for noise reduction.
- Integration Layer:
    - Sends processed data to subsystems such as the control system (for decision-making) and UI (for display and logging).
- Error Handling Layer:
    - Detects and addresses disconnection or invalid readings.

3.3.4.5 Sensor Interface

3.3.4.5.1 Initialization and Configuration

The DS18B20 communicates through a 1-Wire protocol, and the DallasTemperature library simplifies this communication by abstracting the low-level operations and not getting involved in the complicated process mentioned before. We first import the OneWire and DallasTemperature libraries so we can use the functions provided by these libraries and then configure the OneWire GPIO pin connected to the sensor (ex, pin 2) and create an object from both libraries.

Then we initialized the sensor and set the resolution to (9 to 12 bits) higher resolution increases accuracy but requires a longer conversion time. (9 bits: ~93 ms and 12 bits: ~750 ms)

3.3.4.5.2 Data Retrieval

Acquiring temperature data involves requesting readings and handling delays for conversion we first start by requesting the temperature data from the sensor as it operates asynchronously, requiring a request before retrieving the data and after that, we fetch the temperature data from the sensor in Celsius or Fahrenheit thanks to DallasTemperature library the provide the functions and attributes to do so

3.3.4.5.3 Data Processing and Units Conversion

Raw data from the DS18B20 is converted into human-readable units directly by the DallasTemperature library. Temperature is available in both Celsius and Fahrenheit formats.

Scaling and Calibration are available due to systematic error by adding or subtracting offsets.

3.3.4.5.4 Integration

We can use the temperature data in logs for performance analysis and maintenance and send warnings or notifications via the UI or telemetry system in case of overheating

3.3.4.5.5 Error Handling

The sensor may be disconnected so we check it and print if it's connected or make It restart if it's disconnected. Another error may happen if the temperature data exceeds plausible temperature ranges, flag it for review.

3.3.4.6 Pseudocode

- Header file

```
DECLARE class TemperatureSensor
    METHOD: begin()
    METHOD: setResolution(resolution)
    METHOD: readTemperature()
ENDCLASS
```

- Source file

```
DEFINE TemperatureSensor
    METHOD: begin()
        START OneWire and DallasTemperature objects

    METHOD: setResolution(resolution)
```

```
        SET desired resolution for the sensor

    METHOD: readTemperature()
        REQUEST temperature measurement
        RETURN temperature value
ENDDEFINE
```

- Main file

```
IMPORT TemperatureSensor library

DECLARE TemperatureSensor object sensor

SETUP:
    CALL sensor.begin()
    CALL sensor.setResolution(12 bits)

LOOP:
    CALL sensor.readTemperature()
    PRINT temperature value
ENDLOOP
```

3.3.5 Pressure and Depth Sensor

3.3.5.1 Overview

The MS5540C is a digital pressure sensor module used for barometric and underwater pressure measurements. It provides pressure and temperature data through a fully calibrated 16-bit ADC converted into depth readings. This sensor is crucial for underwater applications like ROVs. Communication is achieved via a simple SPI-like interface, making it suitable for embedded systems.

3.3.5.2 Functional Description

3.3.5.2.1 Internal Component

The MS5540C consists of a piezo-resistive sensor and a sensor interface IC. The main function of the MS5540C is to convert the uncompensated analogue output voltage from the piezo-resistive pressure sensor to a 16-bit digital value, as well as providing a 16-bit digital value for the temperature of the sensor. The interface IC consists of PROM that stores the sensor's

calibration coefficients used for temperature and pressure compensation. Also have a control logic to manage data acquisition and communication with the microcontroller. Figure (3.3) give a clear visualization for internal components of the IC.

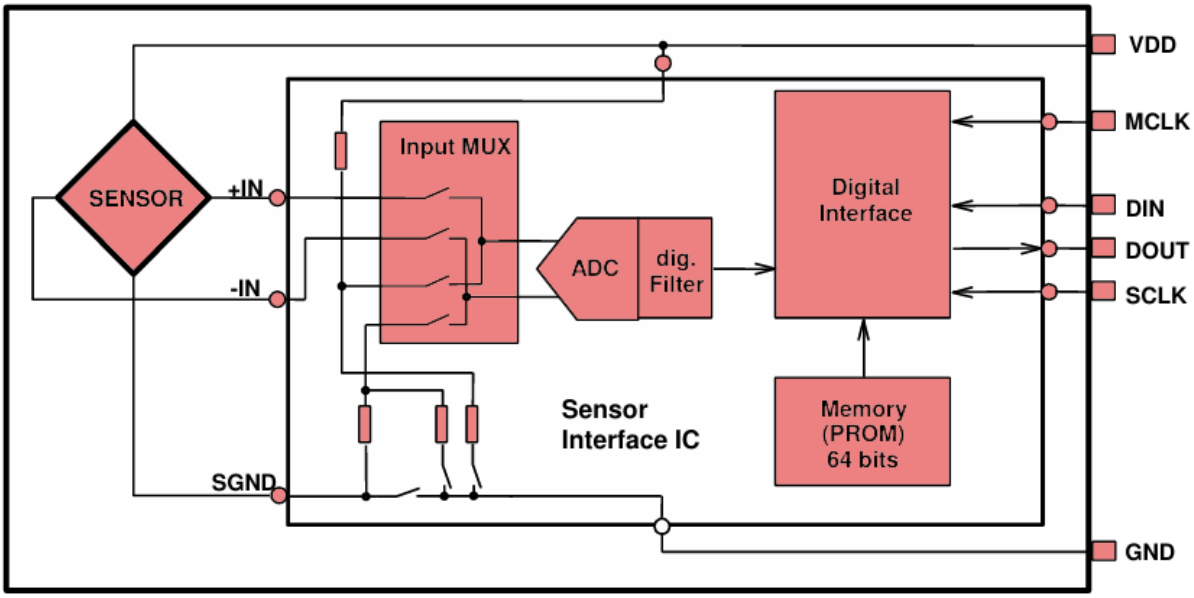


Figure 3.3 internal components of the IC

3.3.5.2.2 Operating Principle

The sensor detects pressure using its piezoresistive element as well as temperature. Since the output voltage of a pressure sensor is heavily influenced by temperature and process tolerances, it is essential to compensate for these effects during pressure measurement. The differential output voltage from the pressure sensor is converted accordingly. For temperature measurement, the resistance of the sensor bridge is monitored and converted using an internal thermistor. Analog signals are digitized by the ADC (sigma-delta converter).

Each module is individually factory-calibrated at two different temperatures and two pressure levels, leading to the calculation and storage of six coefficients necessary for compensating temperature and process variations in the 64-bit PROM of each module as in (Figure 4). This 64-bit data, divided into four 16-bit words, must be accessed by the microcontroller's software and utilized in the program to convert the variables D1 and D2 into compensated pressure and temperature values. The raw data is corrected for both pressure and temperature readings, and data is transferred to the microcontroller through a 3-wire SPI interface.

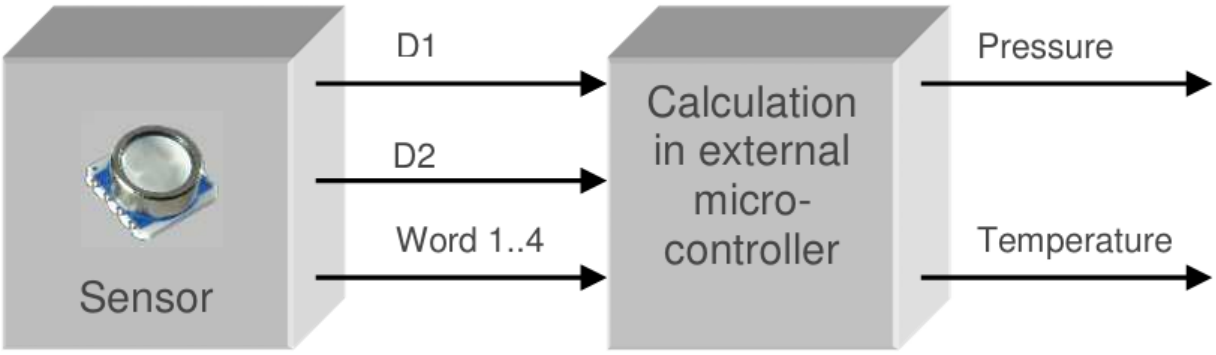


Figure 3.4 MS5540C interface

This behavior is clearly evident in the performance curves below for the raw pressure, the actual pressure (Figure 3.5), and the Absolute Pressure Accuracy after calibration with 2nd order compensation (Figure 3.6).

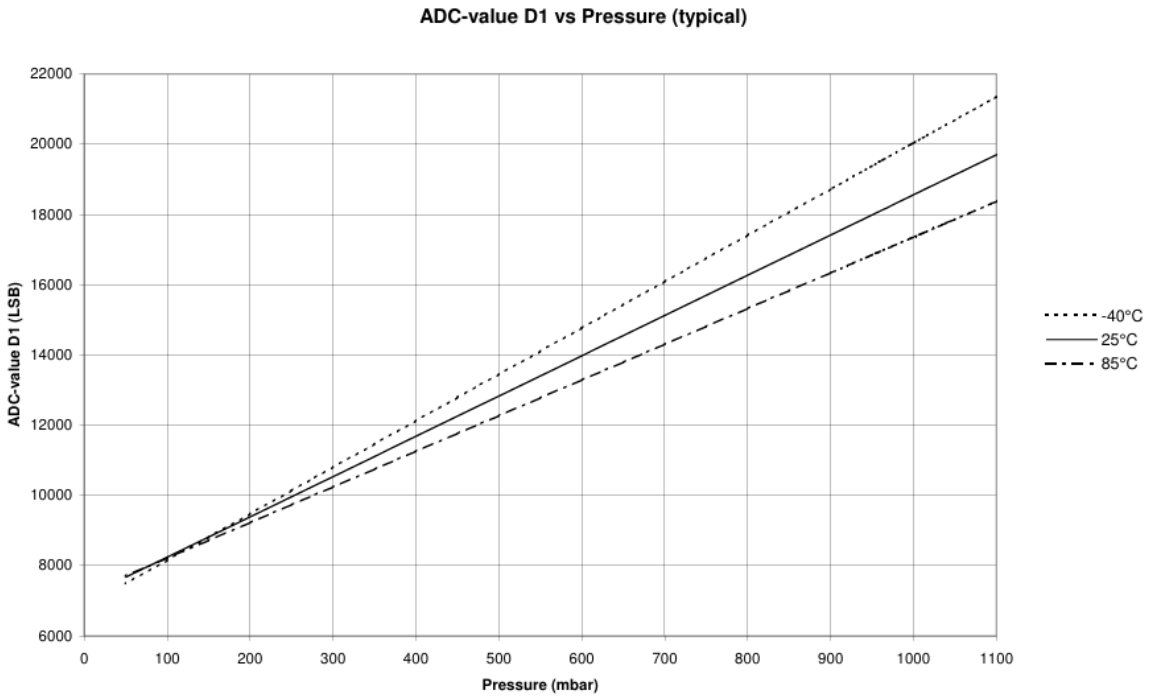


Figure 3.5 performance curves

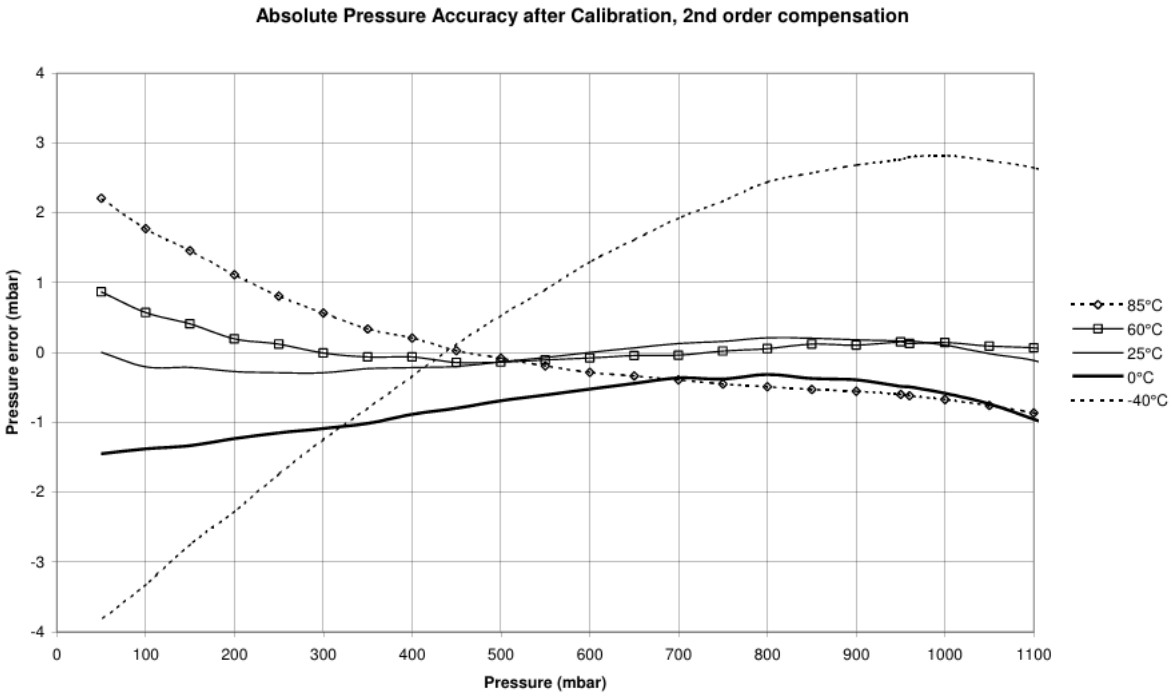


Figure 3.6 Absolute Pressure Accuracy

3.3.5.2.3 Compensation Algorithm

The raw pressure and temperature readings are processed using polynomial equations based on the PROM coefficients:

$$Temperature\ (°C) = A + B \times D2 + C \times D2^2$$

$$Pressure\ (mbar) = P1 + P2 \times D1 + P3 \times D1^2$$

- *D1*: Raw pressure data.
- *D2*: Raw temperature data.
- *A, B, C, P1, P2, P3*: PROM coefficients.

3.3.5.3 Software Interface

1) To begin, we initialize the SPI communication protocol by configuring the settings in the code. We set the bit order to MSB (Most Significant Bit) first and the clock divider to 32.

2) We also created a function to reset the SPI communication. This is crucial to avoid any data corruption that may arise from sensor readings. It is essential to use this function

before retrieving any data, such as before each calibration word retrieval, as well as for the raw pressure and raw temperature readings.

3) Send the reset command to initialize communication and set the default states.
4) Read the PROM (Programmable Read-Only Memory) to fetch the calibration coefficients.
5) Send the Convert D1 command to initiate the pressure data acquisition process.
6) Wait for the specified conversion time, which is approximately 35 ms for high resolution.
7) Send the ADC Read command to retrieve the raw pressure data (D1).
8) Repeat the above steps for obtaining temperature data (D2).
9) Use the calibration coefficients read from PROM.
10) Apply compensation equations to calculate corrected temperature and pressure values.
11) Convert pressure data to depth:

$$Depth\ (meters) = \frac{Pressure\ (mbar) - Atmospheric\ Pressure\ (mbar)}{Fluid\ Density \times g}$$

3.3.5.4 Pseudocode

- Header File

```
DECLARE FUNCTION resetsensor()
    DESCRIPTION: Resets the communication state of the sensor.

DECLARE FUNCTION SPISetup(clockPin)
    INPUT: clockPin (Integer)
    DESCRIPTION: Initializes SPI communication with the given clock pin.

DECLARE FUNCTION clockSetup(clockPin)
    INPUT: clockPin (Integer)
    DESCRIPTION: Starts communication by generating a clock signal.

DECLARE FUNCTION getCallibrationWords(words)
```

```
    INPUT: words (Array of 4 unsigned integers)
    DESCRIPTION: Retrieves calibration words from the sensor.

DECLARE FUNCTION getCoefficients(coefficients)
    INPUT: coefficients (Array of 6 long integers)
    DESCRIPTION: Calculates coefficients from calibration words.

DECLARE FUNCTION calculateRawPressure(D1)
    INPUT: D1 (Array of 1 unsigned integer)
    DESCRIPTION: Reads raw pressure data from the sensor.

DECLARE FUNCTION calculateRawTemperature(D2)
    INPUT: D2 (Array of 1 unsigned integer)
    DESCRIPTION: Reads raw temperature data from the sensor.

DECLARE FUNCTION getCompensatedPressureTemp(values)
    INPUT: values (Array of 2 floating-point numbers)
    DESCRIPTION: Computes temperature and pressure from raw sensor data.

DECLARE FUNCTION getTemperatureinC() RETURNS Float
    DESCRIPTION: Gets temperature in Celsius.

DECLARE FUNCTION getPressureinMBAR() RETURNS Float
    DESCRIPTION: Gets pressure in millibars.

DECLARE FUNCTION getPressureinMMHG() RETURNS Float
    DESCRIPTION: Gets pressure in millimeters of mercury.

DECLARE FUNCTION getDepthinMeter(pressureInMBar) RETURNS Float
    INPUT: pressureInMBar (Float)
    DESCRIPTION: Calculates depth from pressure in meters.
```

- Source Code

```
BEGIN PROGRAM

DEFINE valuesArray AS float array of size 3 initialized to 0

FUNCTION resetsensor()
```

```
    SET SPI data mode to SPI_MODE0
    SEND SPI commands 0x15, 0x55, 0x40
END FUNCTION

FUNCTION SPISetup(clockPin)
    INITIALIZE SPI
    SET SPI bit order to MSBFIRST
    SET SPI clock divider to SPI_CLOCK_DIV32
    CONFIGURE clockPin as OUTPUT
    WAIT for 100 milliseconds
END FUNCTION

FUNCTION clockSetup(clockPin)
    CONFIGURE Timer1 settings to generate MCKL signal
    OUTPUT analog signal 128 on clockPin
END FUNCTION

FUNCTION getCallibrationWords(words)
    DECLARE inByteWords AS unsigned int array of size 4 initialized to 0
    CALL resetsensor()

    FOR each calibration word from 1 to 4 DO
        SEND specific SPI command to request calibration word
        SET SPI data mode to SPI_MODE1
        READ first byte of word and shift it
        READ second byte of word
        COMBINE first and second byte into words array
        PRINT calibration word
        CALL resetsensor()
    END FOR
END FUNCTION

FUNCTION getCoefficients(coefficients)
    DECLARE wordArray AS unsigned int array of size 4
    CALL getCallibrationWords(wordArray)

    EXTRACT coefficients using bit manipulation on wordArray:
        c1 = (wordArray[0] >> 1) & 0x7FFF
```

```
        c2 = ((wordArray[2] & 0x003F) << 6) | (wordArray[3] & 0x003F)
        c3 = (wordArray[3] >> 6) & 0x03FF
        c4 = (wordArray[2] >> 6) & 0x03FF
        c5 = ((wordArray[0] & 0x0001) << 10) | ((wordArray[1] >> 6) & 0x03FF)
        c6 = wordArray[1] & 0x003F

    PRINT all coefficients
END FUNCTION

FUNCTION calculateRawPressure(D1)
    CALL resetsensor()
    SEND SPI command to start pressure conversion
    WAIT for conversion to complete
    SET SPI data mode to SPI_MODE1
    READ and combine two bytes of pressure data into D1
    PRINT raw pressure value
END FUNCTION

FUNCTION calculateRawTemperature(D2)
    CALL resetsensor()
    SEND SPI command to start temperature conversion
    WAIT for conversion to complete
    SET SPI data mode to SPI_MODE1
    READ and combine two bytes of temperature data into D2
    PRINT raw temperature value
END FUNCTION

FUNCTION getCompensatedPressureTemp(values)
    DECLARE coefficientsArray AS long array of size 6
    DECLARE rawPressure, rawTemperature AS unsigned int arrays of size 1

    CALL getCoefficients(coefficientsArray)
    CALL calculateRawPressure(rawPressure)
    CALL calculateRawTemperature(rawTemperature)

    COMPUTE compensation values:
        UT1 = (coefficientsArray[4] << 3) + 20224
        dT = rawTemperature[0] - UT1
```

```
        TEMP = 200 + ((dT * (coefficientsArray[5] + 50)) >> 10)
        OFF = (coefficientsArray[1] * 4) + (((coefficientsArray[3] - 512) *
dT) >> 12)
        SENS = coefficientsArray[0] + ((coefficientsArray[2] * dT) >> 10) +
24576
        X = (SENS * (rawPressure[0] - 7168) >> 14) - OFF
        PCOMP = ((X * 10) >> 5) + 2500

    ASSIGN TEMP and PCOMP to values array
END FUNCTION

FUNCTION getTemperatureinC()
    CALL getCompensatedPressureTemp(valuesArray)
    RETURN valuesArray[0] / 10
END FUNCTION

FUNCTION getPressureinMBAR()
    CALL getCompensatedPressureTemp(valuesArray)
    RETURN valuesArray[1]
END FUNCTION

FUNCTION getPressureinMMHG()
    CALL getCompensatedPressureTemp(valuesArray)
    RETURN valuesArray[1] * 750.06 / 10000
END FUNCTION

FUNCTION getDepthinMeter(pressureInMBar)
    DECLARE rho AS 1025.0
    DECLARE g AS 9.81
    RETURN pressureInMBar / (rho * g)
END FUNCTION

END PROGRAM
```

- Main File

```
INCLUDE LIBRARY: PressureAndDepthSensor.h

DECLARE CONSTANT clock = 9
```

```
    DESCRIPTION: Pin for generating MCKL signal.

DEFINE FUNCTION setup()
    BEGIN
        START serial communication at 9600 baud rate.
        CALL SPISetup(clock) to initialize SPI communication with the clock
pin.
    END

DEFINE FUNCTION loop()
    BEGIN
        CALL clockSetup(clock) to generate MCKL signal.

        PRINT "Pressure in mbar = " to Serial Monitor.
        PRINT RETURN VALUE of getPressureinMBAR() to Serial Monitor.

        PRINT "Pressure in mmHg = " to Serial Monitor.
        PRINT RETURN VALUE of getPressureinMMHG() to Serial Monitor.

        PRINT "Depth in m = " to Serial Monitor.
        CALL getPressureinMBAR() and PASS its RETURN VALUE to
getDepthinMeter().
        PRINT RESULT to Serial Monitor.

        PRINT "Temperature in C = " to Serial Monitor.
        PRINT RETURN VALUE of getTemperatureinC() to Serial Monitor.

        WAIT for 5000 milliseconds.
    END
```

3.3.5.5 Problems and Solutions

The sensor can withstand up to 100 meters under water, however the linear range for factory calibrated coefficients stored on the sensor ROM adjusted for linear range of about 7m, so if we want a more linear range, we may calibrate our own factors and use each group of factors for each range.

3.3.5.5.1 Steps to Calibrate the MS5540C Sensor

A calibrated reference pressure source that can generate pressure within our desired range of 0 to 100 meters is required. Additionally, a temperature-controlled environment is necessary to perform calibrations across the expected operating temperatures. We need a data acquisition system to read and log the sensor's ADC values (D1, D2), along with the reference pressure and temperature.

Software or scripts will be needed to compute the best-fit calibration coefficients. For each pressure step within the desired range (in increments of 5 meters up to 100 meters), we will record the following:

- Raw pressure ADC (D1) and temperature ADC (D2) from the sensor.
- Reference pressure and temperature using a high-accuracy device.

For each pressure step, we will vary the temperature to capture data at multiple points. For example, we will record data at 0°C, 25°C, and 50°C. Each data point must include D1, D2, the reference pressure, and the reference temperature.

The collected data will be used to compute the calibration coefficients (C1–C6) by following these steps:

- Fit the Data: Use curve-fitting or regression analysis to match the recorded D1 and D2 values against the reference pressure and temperature. Software tools such as MATLAB, Python (NumPy, SciPy), or Excel can assist with this process.
- Calculate Coefficients: The coefficients C1 to C6 will represent offsets, scaling factors, and temperature dependencies. We need to fit the following equations:
    - Pressure Offset: This accounts for any zero-pressure offset.
    - Pressure Sensitivity: This relates the raw data to the reference pressure.
    - Temperature Compensation Terms: These correct for the effects of temperature on the pressure readings.

3.4 SENSOR FUSION

3.4.1 Introduction

Sensor fusion is the process of combining data from multiple sensors to produce more accurate, reliable, and meaningful information than could be achieved by using a single sensor alone. In the context of an ROV (Remotely Operated Vehicle), sensor fusion is critical for tasks such as orientation estimation, motion tracking, and stabilization.

Complementary Filter is a simple yet effective sensor fusion algorithm that combines data from accelerometers and gyroscopes to estimate orientation. It is widely used in embedded systems due to its low computational cost and ease of implementation.

3.4.2 Complementary Filter

3.4.2.1 Core Concept

The Complementary Filter operates on the principle that:

- One data source is reliable in the low-frequency range but noisy or unreliable in the high-frequency range.
- Another data source is reliable in the high-frequency range but suffers from drift or inaccuracies in the low-frequency range.

By combining these two sources, the Complementary Filter produces an output that is accurate across the entire frequency spectrum.

3.4.2.2 Mathematical Formulation

The Complementary Filter can be expressed as:

$$y(t) = \alpha \cdot y_{high}(t) + (1 - \alpha) \cdot y_{low}(t)$$

Where:

- $y(t)$: The filtered output at time t.
- $y\ high(t)$: The high-frequency component of the data.
- $y\ low(t)$: The low-frequency component of the data.

- α: The filter coefficient ($0 < \alpha < 1$), which determines the weight given to the high-frequency component.

3.4.2.3 Filter Coefficient (α)

- The value of α determines the balance between the high-frequency and low-frequency components.
    - A higher value of α (closer to 1) gives more weight to the high-frequency component.
    - A lower value of α (closer to 0) gives more weight to the low-frequency component.
- The choice of α depends on the specific application and the characteristics of the data sources.

3.4.2.4 Frequency Domain Interpretation

In the frequency domain, the Complementary Filter acts as a low-pass filter for one data source and a high-pass filter for the other. The cutoff frequency is determined by the value of α.

3.4.2.5 Limitations of the Complementary Filter

- Fixed Weighting:
    - The filter coefficient (α) is constant, which may not be optimal for all scenarios.
- Assumption of Complementary Characteristics:
    - The filter assumes that the data sources have complementary frequency responses, which may not always be the case.
- Limited to Two Data Sources:
    - The basic Complementary Filter is designed for two data sources; additional processing is required for more sources.

3.5 CONTROL SYSTEMS

3.5.1 Joystick

The Logitech Extreme 3D Pro Joystick is an integral part of the ROV control system, providing a precise and intuitive interface for maneuvering the ROV. It combines ergonomic design with advanced features, making it an ideal choice for operating in challenging underwater environments.

Figure 3.7 Joystick

3.5.1.1 Method of Connection

3.5.1.1.1 Joystick to Arduino

- The joystick connects to the above-water Arduino through a USB Host Shield, using the Serial Communication Protocol.
- This setup translates joystick inputs (both analog and digital) into commands that are processed by the Arduino.

3.5.1.1.2 Above-Water Arduino to Underwater Arduino

- Communication between the above-water Arduino and the underwater Arduino is achieved through an Ethernet cable using the I2C communication protocol.
- Three wires from the Ethernet cable are used:
    - Ground (GND): Common reference for electrical signals.
    - Serial Clock Line (SCL): For synchronizing communication.
    - Serial Data Line (SDA): For transmitting data between the Arduinos.

3.5.1.2 Role in the Project

- The joystick's analog inputs (e.g., twist rudder) enable smooth and precise navigation of the ROV in underwater environments.
- The digital buttons are mapped to directional commands, offering quick responsiveness.

3.5.1.3 Pseudocode

```
Initialize USB, HID, and Joystick components
Initialize I2C communication
Define joystick input variables and camera control pins

SETUP FUNCTION:
  Start serial communication for debugging
  Initialize USB communication
  IF USB initialization fails:
    Print error message and halt program
  Delay for USB stabilization
  Set up HID report parser for the joystick
  IF parser setup fails:
    Print error message

LOOP FUNCTION:
  Process USB tasks
  Read joystick values (X, Y, Hat, Twist, Slider, Button)

  Map joystick values to range (-100 to 100):
    - Surge: Forward/backward (mapped from Y-axis)
    - Sway: Left/right (mapped from X-axis)
    - Yaw: Rotation (mapped from Twist)
    - Heave: Up/down (mapped from Slider)
    - Pitch and Roll: Derived from Hat switch position

  Send mapped joystick data to ROV via I2C:
    Start I2C transmission
    Write values: Surge, Sway, Yaw, Heave, Pitch, Roll, Button
    End transmission and check for errors
    IF successful:
      Print success message
```

```
    ELSE:
      Print error message

  Optional: Wait for acknowledgment from ROV
    Request acknowledgment byte via I2C
    IF acknowledgment received:
      Check if valid, print corresponding message
    ELSE:
      Print no acknowledgment error

  Delay for a controlled refresh rate
```

3.5.2 PID Control

A Proportional-Integral-Derivative (PID) controller is a feedback-based control loop mechanism widely used in industrial and automation systems to regulate processes requiring continuous control and automatic adjustments. It operates by comparing a desired target value (setpoint or SP) with the actual system output (process variable or PV). The difference between these values, known as the error (e(t)), is used to apply corrective actions through three control terms: Proportional (P), Integral (I), and Derivative (D).

Key Components of a PID Controller:

- Proportional (P): Responds to the current error magnitude, providing immediate correction proportional to the error.
- Integral (I): Addresses residual steady-state errors by integrating past errors over time.
- Derivative (D): Predicts future error trends based on the rate of change, reducing overshoot and improving stability.

PID controllers enhance automation by minimizing human intervention and errors, ensuring precise control.

3.5.2.1 Fundamental Operation

The PID controller continuously calculates the error e(t) = SP – PV and applies corrections using a weighted sum of the P, I, and D terms. The control variable u(t) (e.g., valve position) is adjusted to minimize the error over time.

- Proportional Term (P): Directly proportional to the current error. A high proportional gain (Kp) results in a stronger response but may cause instability if too high.
- Integral Term (I): Eliminates steady-state errors by integrating past errors. However, excessive integral gain (Ki) can cause overshoot.
- Derivative Term (D): Dampens system response by predicting future errors based on the error rate of change. High derivative gain (Kd) improves stability but can amplify noise.

3.5.2.2 Tuning

PID controllers require tuning to balance the P, I, and D terms for optimal performance. Tuning involves adjusting the gains (Kp, Ki, Kd) based on the system's response characteristics.
Common tuning methods include:

- Manual Tuning: Adjust gains iteratively to achieve desired performance.
- Ziegler-Nichols Method: A systematic approach to determine initial gain values.
- Software-Based Tuning: Advanced tools automate tuning by analyzing system responses and suggesting optimal parameters.

3.5.2.3 Mathematical Form

The PID control function is expressed as:

$$u(t) = Kp \cdot e(t) + Ki \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

Where:

- *Kp*: Proportional gain
- *Ki*: Integral gain
- *Kd*: Derivative gain

- $e(t)$: Error (SP – PV)

In the Laplace domain, the transfer function is:

$$L(s) = K_p + \frac{K_i}{s} + K_d s$$

3.5.2.4 Selective Use of Control Terms

Not all applications require all three terms. Common configurations include:

- PI Controller: Used when derivative action is sensitive to noise.
- PD Controller: Applied where integral action is unnecessary.
- P or I Controller: Simplified controllers for specific use cases.

3.5.2.5 Controller Theory

The PID controller output is the sum of the P, I, and D terms. The manipulated variable (MV) is calculated as:

$$MV(t) = Kp \cdot e(t) + Ki \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

3.5.2.6 Detailed Control Terms

3.5.2.6.1 Proportional Term

- Output: $Pout = Kp \cdot e(t)$
- High *Kp* improves responsiveness but risks instability.
- Low *Kp* results in sluggish control.

3.5.2.6.2 Integral Term

- Output: $Iout = Ki \int_0^t e(\tau)\, d\tau$
- Eliminates steady-state errors but can cause overshoot.

3.5.2.6.3 Derivative Term

- Output: $Dout = Kd \frac{de(t)}{dt}$

- Improves stability by damping oscillations but is sensitive to noise.

3.5.2.7 Loop Tuning

Tuning ensures stability and optimal performance. Key considerations:

- Stability: Avoid excessive gains to prevent oscillations.
- Manual Tuning: Start with Kp*Kp*, then adjust Ki*Ki* and Kd*Kd*.
- Software Tools: Automate tuning for complex systems.

3.5.2.8 Common Issues and Solutions

3.5.2.8.1 Integral Windup

- Cause: Integral term accumulates excessive error during large setpoint changes.
- Solution: Disable integration or limit integral term bounds.

3.5.2.8.2 Overshooting

- Cause: Rapid changes in setpoint or disturbances.
- Solution: Use setpoint ramping or derivative of the process variable.

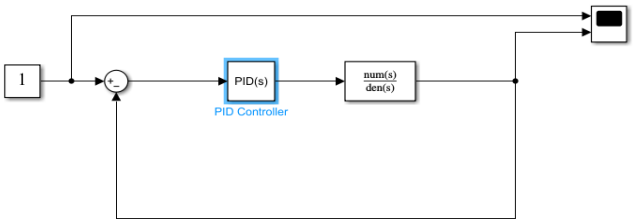
3.5.2.9 PID Controller Simulation Using Simulink and Output Response

This section documents the simulation of a closed-loop control system using a PID controller and a Brushless DC Motor (BLDC) as the plant. The goal is to achieve a desired output response by tuning the PID parameters.

3.5.2.9.1 Simulation Overview

The simulation models a closed-loop system where:

Figure 3.8 closed-loop system



- The setpoint is the desired output value.
- The PID controller adjusts the control input to minimize the error between the setpoint and the process variable (PV).
- The BLDC motor represents the plant, and its dynamics are modeled using a transfer function.
- The output response is observed and tuned using a Scope.

3.5.2.9.2 Components of the Simulation

- Constant Block: Represents the setpoint (desired value).
- Sum Block: Computes the error e(t)=Setpoint−Process Variable (PV)$e(t)$=Setpoint−Process Variable (PV).
- PID Controller: Applies proportional, integral, and derivative actions to correct the error.
- Transfer Function Block: Models the BLDC motor dynamics using the transfer function:

$$G(s) = \frac{1}{s^3 + 3s^2 + 2s}$$

- Scope: Visualizes the system's output response for analysis and tuning.

3.5.2.9.3 Tuning Process and Output Response

The PID controller was tuned manually to achieve the desired output response. The steps and observations are documented below:

Step 1: Initial Setup

- Set *Kp*=0, *Ki*=0, and *Kd*=0.
- Observation: The process variable (PV) is far from the setpoint, indicating no control action.

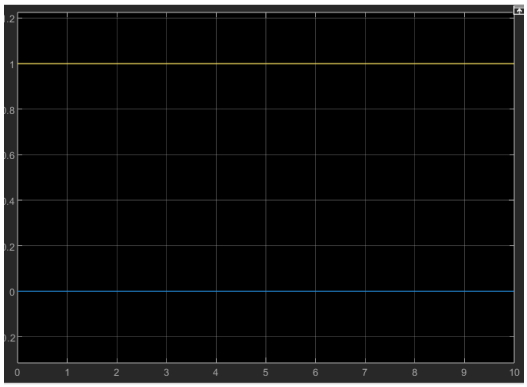


Figure 3.9 Initial Setup

Step 2: Tuning the Proportional Gain (*Kp*)

- Gradually increase *Kp* to reduce the steady-state error.
- Observation: As *Kp* increases, the steady-state error decreases significantly.
- Key Observations:
    - If *Kp* is too low, the system response is slow and sluggish.
    - If *Kp* is too high, the system becomes unstable and oscillates.

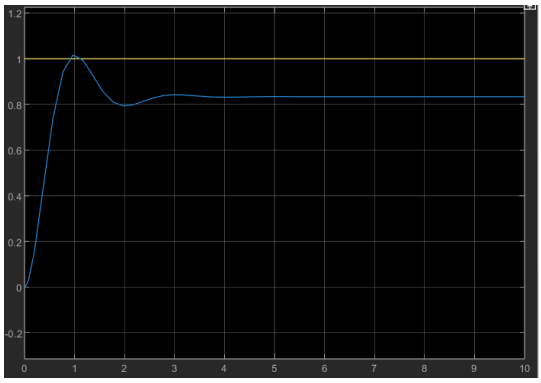


Figure 3.10 Tuning the Proportional Gain

Step 3: Tuning the Integral Gain (*Ki*)

- Increase *Ki* to eliminate residual steady-state error.
- Observation: The steady-state error is eliminated, but overshooting increases.
- Key Observations:
    - If *Ki* is too low, the steady-state error persists.
    - If *Ki* is too high, the system becomes slower and less stable.

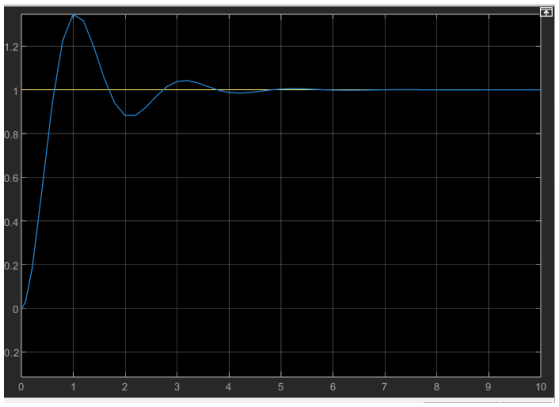


Figure 3.11 Tuning the Integral Gain

Step 4: Tuning the Derivative Gain (*Kd*)

- Increase *Kd* to reduce oscillations and overshooting.

- Observation: The system achieves a stable response with minimal overshooting and no steady-state error.
- Final Tuned Parameters:
    - Kp=10
    - Ki=10
    - Kd=3

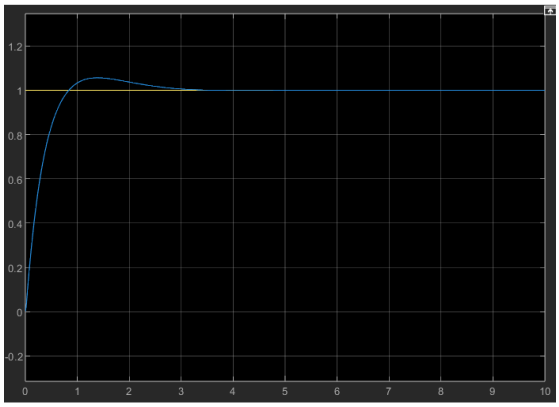


Figure 3.12 Tuning the Derivative Gain

3.5.3 Thrusters and Movement Control

3.5.3.1 Electronic Speed Controllers

3.5.3.1.1 Purpose

The Electronic Speed Controller (ESC) is utilized to regulate the speed and direction of thrusters in the ROV system. This document details the control mechanisms and software integration to ensure reliable and precise motor performance.

3.5.3.1.2 Software Requirements

- Libraries: `Servo.h`

3.5.3.1.3 Software Architecture

- Initialization: Initialize PWM output pins connected to ESC signal inputs. Typical PWM range:
    - 1000 µs: Minimum speed (reverse if supported).

- 1500 µs: Neutral (stationary).
- 2000 µs: Maximum speed (forward).

- Control Loop: Continuously monitor and apply thrust values received via serial communication.
- Safety Mechanisms: Stop motors during signal loss.

3.5.3.1.4 Error Handling

- Detect signal loss using a timeout mechanism.
- Reset all PWM signals to neutral (1500 microseconds) on communication failure.

3.5.3.1.5 ESC and Thruster Integration

- ESC Setup:
  - ESCs require calibration to match PWM ranges. This is usually done once at startup.
  - Each ESC is assigned to a specific thruster.
- PWM Signal Mapping:
  - The code calculates PWM values dynamically based on input commands and control algorithms (e.g., joystick commands, PID output).

3.5.3.2 Thruster Configurations

- Front Thrusters:
  - T1: Front-right thruster (45° from the centerline).
  - T2: Front-left thruster (135° from the centerline).
- Back Thrusters:
  - T3: Back-right thruster (225° from the centerline).
  - T4: Back-left thruster (315° from the centerline).

- Vertical Thrusters:
    - T5: Front vertical thruster.
    - T6: Back vertical thruster.

Here's a diagram of the thruster arrangement in Figure (3.13) and Figure (3.14)

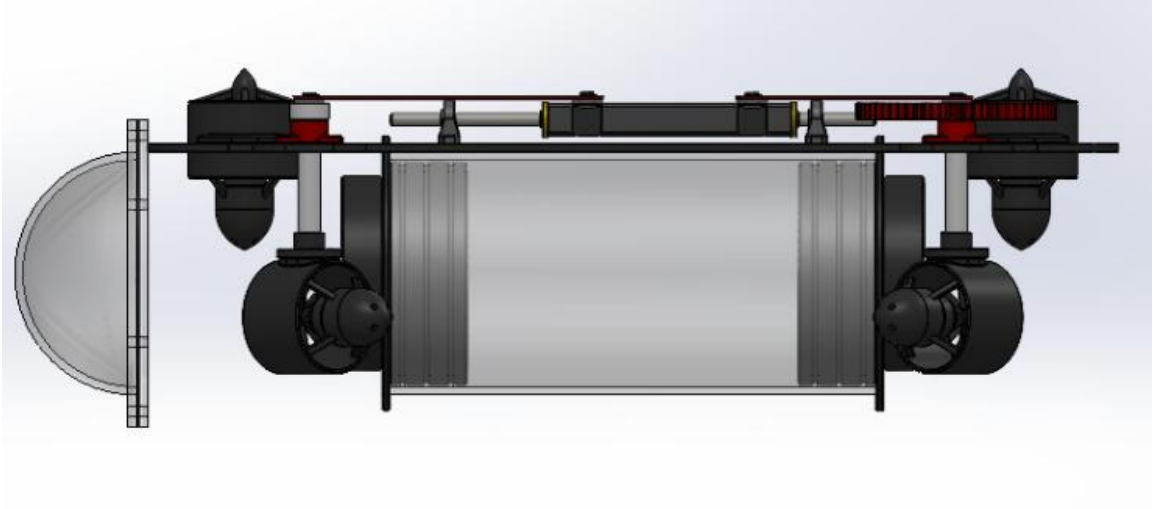


Figure 3.13 diagram of the thruster arrangement

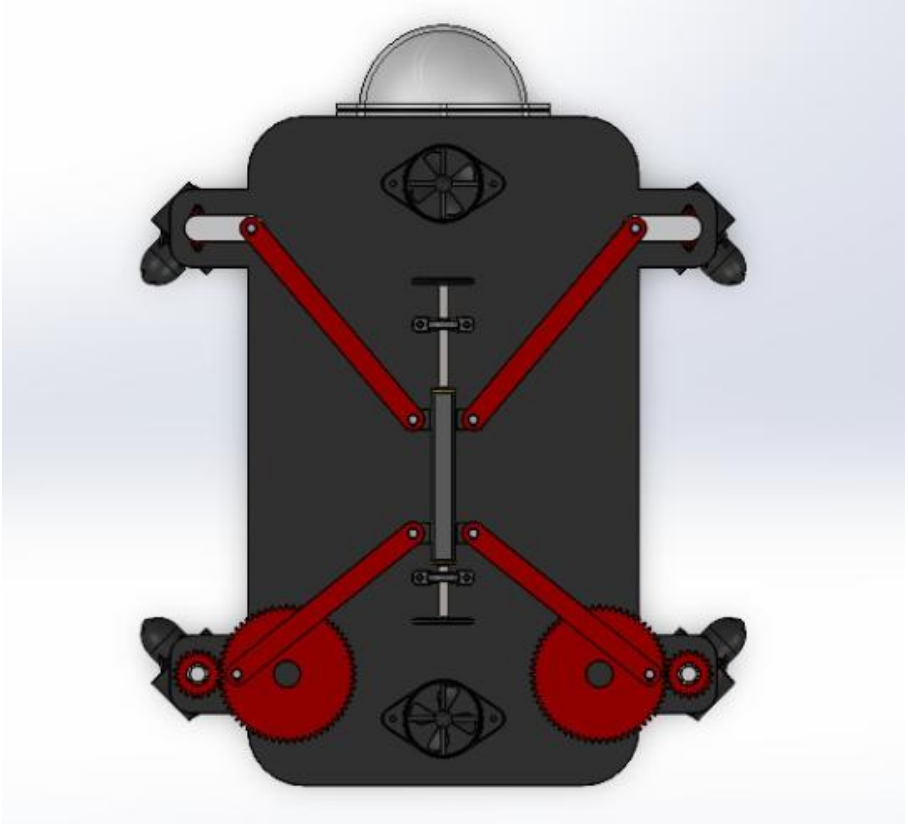


Figure 3.14 diagram of the thruster arrangement

3.5.3.3 Equations for Thrust Calculation

The equations for this configuration will account for the 45-degree angles of the horizontal thrusters. Each thruster contributes to surge, sway, yaw, and roll based on its orientation.

3.5.3.3.1 Horizontal Thrusters (T1, T2, T3, T4)

These thrusters control surge, sway, yaw, and roll:

$$T1 = surge + sway + outputYaw + outputRoll;\ //Front - right\ thruster$$

$$T2 = surge - sway - outputYaw - outputRoll;\ //Front - left\ thruster$$

$$T3 = -surge + sway + outputYaw + outputRoll;\ //Back - right\ thruster$$

$$T4 = -surge - sway - outputYaw - outputRoll;\ //Back - left\ thruster$$

- Explanation:
    - surge: Controls forward/backward motion.

- Positive surge = forward motion.
- Negative surge = backward motion.

- sway: Controls left/right motion.
  - Positive sway = right motion.
  - Negative sway = left motion.
- outputYaw: Controls rotation (yaw).
  - Positive outputYaw = rotate counterclockwise.
  - Negative outputYaw = rotate clockwise.
- outputRoll: Controls tilting (roll).
  - Positive outputRoll = tilt right.
  - Negative outputRoll = tilt left.

Each thruster contributes to these motions based on its orientation:

- T1 (Front-Right, 45°): Combines surge, sway, yaw, and roll.
- T2 (Front-Left, 135°): Combines surge, sway, yaw, and roll.
- T3 (Back-Right, 225°): Combines surge, sway, yaw, and roll.
- T4 (Back-Left, 315°): Combines surge, sway, yaw, and roll.

3.5.3.3.2 Vertical Thrusters (T5, T6)

These thrusters control heave and pitch:

$$T5 = outputHeave + outputPitch; // Front\ vertical\ thruster$$

$$T6 = outputHeave - outputPitch; // Back\ vertical\ thruster$$

- Explanation:

- outputHeave: Controls up/down motion.
  - Positive outputHeave = upward motion.
  - Negative outputHeave = downward motion.
- outputPitch: Controls tilting forward/backward.
  - Positive outputPitch = tilt forward.
  - Negative outputPitch = tilt backward.

Each vertical thruster contributes to heave and pitch:

- T5: Combines heave and pitch.
- T6: Combines heave and pitch.

3.5.3.4 How Thrust Vectors Work

The thrust vectors are calculated by combining the contributions of each motion (surge, sway, heave, roll, pitch, yaw) for each thruster:

3.5.3.4.1 Horizontal Thrusters (T1–T4)

- Surge and Sway:
  - The horizontal thrusters are at 45-degree angles, so they contribute to both surge and sway.
  - For example:
    - T1 (Front-Right, 45°) contributes equally to surge and sway.
    - T2 (Front-Left, 135°) contributes positively to surge and negatively to sway.
- Yaw and Roll:
  - Yaw is achieved by differential thrust between thrusters on opposite sides.
  - Roll is achieved by differential thrust between thrusters on the same side.

3.5.3.4.2 Vertical Thrusters (T5, T6)

- Heave:
    - Heave is achieved by equal thrust from both vertical thrusters (T5, T6).
- Pitch:
    - Pitch is achieved by differential thrust between the vertical thrusters.

3.5.3.5 Example Scenario

- Inputs:
    - surge = 1.0 (forward motion).
    - sway = 0.5 (right motion).
    - outputYaw = 0.2 (rotate counterclockwise).
    - outputRoll = 0.1 (tilt right).
    - outputHeave = 0.8 (upward motion).
    - outputPitch = 0.3 (tilt forward).
- Calculations:
    - Horizontal Thrusters:

$$T1 = 1.0 + 0.5 + 0.2 + 0.1 = 1.8; \; // Front - right\ thruster$$

$$T2 = 1.0 - 0.5 - 0.2 - 0.1 = 0.2; \; // Front - left\ thruster$$

$$T3 = -1.0 + 0.5 + 0.2 + 0.1 = -0.2; // Back - right\ thruster$$

$$T4 = -1.0 - 0.5 - 0.2 - 0.1 = -1.8; // Back - left\ thruster$$

    - Vertical Thrusters:

$$T5 = 0.8 + 0.3 = 1.1; // Front\ vertical\ thruster$$

$$T6 = 0.8 - 0.3 = 0.5; // Back\ vertical\ thruster$$

- Interpretation:
    - T1: Strong forward and right thrust.
    - T2: Slight forward thrust.
    - T3: Slight backward thrust.
    - T4: Strong backward and left thrust.
    - T5: Strong upward thrust with forward tilt.
    - T6: Moderate upward thrust with backward tilt.

3.5.3.6 Sport Mode

3.5.3.6.1 Overview

The Sport Mode feature in the ROV project enhances the speed and maneuverability of the remotely operated vehicle (ROV) by dynamically adjusting the thruster angles. This feature is activated through a joystick input and involves the use of a stepper motor to reconfigure the thruster orientation.

3.5.3.6.2 Key Features

- Objective:
    - Increase the speed and agility of the ROV by adjusting the thruster angles from 45 degrees to 90 degrees.
- Activation Mechanism:
    - The Sport Mode is toggled using Button 11 on the joystick controller.
    - A software flag is used to track the activation state of Sport Mode.
- Thruster Adjustment:
    - When Sport Mode is activated:
        - The stepper motor reorients the thrusters to 90 degrees.

- This configuration optimizes propulsion for maximum forward speed.

- When Sport Mode is deactivated:
  - The stepper motor returns the thrusters to the default 45-degree orientation.
  - This configuration balances speed and maneuverability.

3.5.3.6.3 Pseudocode

```
Initialize sportModeFlag to false
Initialize thrusterAngle to 45

Loop forever:
    Check if Button 11 on the joystick is pressed
    If Button 11 is pressed:
        Toggle sportModeFlag

    If sportModeFlag is true:
        Set thrusterAngle to 90
    Else:
        Set thrusterAngle to 45

    Command stepper motor to adjust to thrusterAngle
    Run stepper motor to execute adjustment
```

3.5.4 Emergency Stop

The Emergency Stop functionality in the ROV code is designed to prevent damage to the system or ensure the safety of the ROV by halting its operations when certain safety thresholds are exceeded. Below are the two conditions that trigger the Emergency Stop

3.5.4.1 Current Limit Exceeded

The first condition monitors the current being drawn by the ROV. If the current exceeds the predefined safe limit `(SAFEST_CURRENT)`, the motors are stopped immediately to prevent overheating or damage to the electrical components.

3.5.4.1.1 Workflow

- The system continuously monitors the current drawn by the thrusters or other electrical components.
- If the `currentReading` exceeds `SAFEST_CURRENT`:
- A warning message is printed to the Serial Monitor.
- The `setThrustersNeutral()` function is called, stopping all thrusters.
- Purpose:
    - To protect the ROV from overcurrent conditions that could lead to component failure or fire hazards.

3.5.4.1.2 Pseudocode

```
if (currentReading > SAFEST_CURRENT) { // Check if current exceeds safe limit
    Serial.println("Current exceeds safe limit! Stopping motors.");
    setThrustersNeutral(); // Stop motors
}
```

3.5.4.2 Maximum Depth Exceeded

The second condition checks the ROV's current depth. If the depth exceeds the predefined maximum safe depth (`maxDepth`), the motors are stopped immediately, and a warning is issued to prevent the ROV from being damaged by excessive water pressure.

3.5.4.2.1 Workflow

- The system calculates the ROV's current depth by:
    - Measuring pressure in millibars using `getPressureinMBAR()`.
    - Converting the pressure into depth using `getDepthinMeter()`.
- If the calculated depth exceeds `maxDepth`:
    - A warning message is displayed in the Serial Monitor.
    - The `setThrustersNeutral()` function is called, stopping all thrusters.
- Purpose:

- To safeguard the ROV against structural damage due to excessive water pressure at depths beyond its design capabilities.

3.5.4.2.2 Pseudocode

```
CALCULATE currentDepth = getDepthinMeter(getPressureinMBAR())

IF currentDepth > maxDepth THEN
    DISPLAY "Warning! YOU REACHED THE MAXIMUM DEPTH, PLEASE HEAVE UP."
    CALL setThrustersNeutral()  // Stop all motors
END IF
```

3.6 COMMUNICATION PROTOCOLS

3.6.1 OneWire

3.6.1.1 Overview

- Type: Asynchronous serial communication.
- Wires: 1 (data line, optionally power).
- Speed: Low to moderate (typically up to 16 kbps).
- Addressing: Unique 64-bit ROM code for each device.
- Use Case: Communication with sensors (e.g., DS18B20 temperature sensor).

3.6.1.2 Key Features

- Single data line for communication and power (parasite mode).
- Supports up to 255 devices on the same bus.
- Simple and cost-effective for low-speed applications.

3.6.1.3 Limitations

- Limited speed and distance.
- Complex timing requirements for communication.
- Requires precise timing for reliable data transfer.

3.6.1.4 Library Installation (Arduino)

- Open Arduino IDE.
- Go to Sketch > Include Library > Manage Libraries.
- Search for OneWire and install the library.

3.6.1.5 Key Functionalities

- Initialization: `OneWire oneWire(pinNumber);`
- Device Detection: `oneWire.search(address);`
- Reset Bus: `oneWire.reset();`
- Write Data: `oneWire.write(data);`
- Read Data: `oneWire.read();`

3.6.2 UART

3.6.2.1 Overview

- Type: Asynchronous serial communication.
- Wires: 2 (Tx for transmitting, Rx for receiving).
- Speed: Configurable baud rates (e.g., 9600, 115200 bps).
- Data Format: Packets with start bit, data frame (5–9 bits), optional parity bit, and stop bits.
- Use Case: Short-distance communication between two devices.

3.6.2.2 Key Features

- No clock signal; synchronization via matching baud rates.
- Simple implementation with minimal hardware.
- Supports full-duplex communication.

3.6.2.3 Limitations

- Limited to one master and one slave.
- Susceptible to noise over long distances.
- Requires precise baud rate matching.

3.6.3 I2C

3.6.3.1 Overview

- Type: Synchronous serial communication.
- Wires: 2 (SDA for data, SCL for clock).
- Speed: Standard (100 kbps), Fast (400 kbps), High-Speed (3.4 Mbps), Ultra-Fast (5 Mbps).
- Addressing: 7-bit or 10-bit unique addresses for slaves.
- Use Case: Communication between multiple devices (e.g., sensors, displays).

3.6.3.2 Key Features

- Supports multiple masters and slaves on the same bus.
- ACK/NACK bits for error detection.
- Low pin count and simple wiring.

3.6.3.3 Limitations

- Slower than SPI.
- Limited distance and susceptible to noise.
- Requires pull-up resistors on SDA and SCL lines.

3.6.4 SPI

3.6.4.1 Overview

- Type: Synchronous serial communication.

- Wires: 4 (MOSI, MISO, SCLK, CS).
- Speed: High (up to tens of Mbps).
- Addressing: Chip Select (CS) lines for each slave.
- Use Case: High-speed communication with peripherals (e.g., SD cards, displays).

3.6.4.2 Key Features

- Full-duplex communication.
- Supports multiple slaves with individual CS lines.
- High-speed data transfer.

3.6.4.3 Limitations

- Requires more pins (CS for each slave).
- Complex wiring for multiple devices.
- Limited distance due to high-speed signaling.

3.6.4.4 Operation

1. Chip Select (CS): Activates the target slave.
2. Clock (SCLK): Synchronizes data transfer.
3. MOSI (Master Out Slave In): Data from master to slave.
4. MISO (Master In Slave Out): Data from slave to master.

3.6.4.5 Clock Modes

- CPOL (Clock Polarity): Idle state of the clock (0 = low, 1 = high).
- CPHA (Clock Phase): Sampling edge (0 = first edge, 1 = second edge).

3.6.4.6 Bus Topologies

1. Multidrop: Each slave has a dedicated CS line.
2. Daisy Chain: Slaves share a single CS line; data shifts through all devices.

3. Expander: Uses additional hardware (e.g., shift registers) to manage CS lines.

3.7 OBJECT DETECTION

3.7.1 Camera Streaming

The ROV is equipped with a single high-performance camera (Surveillance Camera) designed to enhance underwater exploration and operational efficiency.
This camera is strategically positioned at the front of the ROV and is enclosed in a watertight housing with a dome for optimal performance and protection against water pressure and environmental factors.

Figure 3.15 The Surveillance Camera

3.7.1.1 Camera Functions:

3.7.1.1.1 Live Streaming for Exploration

- The Camera Provides a Real-time Video feed, enabling users to explore and monitor aquatic life effectively.
- This feature is instrumental in observing and studying underwater ecosystems, aiding in scientific research and environmental awareness.

3.7.1.1.2 Object Detection

- Integrated with the YOLOv8 model, the camera facilitates advanced object detection.

- This functionality enables the identification and classification of underwater entities such as fish, coral reefs, and marine debris, contributing to targeted operations and data analysis.

3.7.1.2 Camera Specifications:

3.7.1.2.1 Resolution

- 1080p HD: Delivers high-definition video quality for clear and detailed visuals.

3.7.1.2.2 Dark Mode Capability

- The camera supports a dark mode feature, allowing efficient operation in low-light underwater environments.

3.7.1.2.3 Integrated Lighting

- Equipped with built-in LEDs, the camera ensures adequate illumination in dark or murky underwater conditions, enhancing visibility and data accuracy.

3.7.1.2.4 Ethernet Connectivity

- The camera is connected via Ethernet instead of USB, ensuring faster data transmission speeds and reliable performance during live streaming and object detection operations.

3.7.2 Lighting System

- Purpose: Provide illumination using a controllable LED strip for operational visibility.
- Components:
    - LED Strip: The primary lighting source.
    - Button: Used to control the lighting state (ON/OFF).
    - Relay Module: Acts as a switch to control the power supply to the LED strip.
- Operation:
    - Pressing the button sends a signal to the relay module.
    - The relay toggles the power supply to the LED strip, turning the lights ON or OFF.

- Software Logic:
    - Monitor the button state using a digital input pin.
    - On the button press, toggle the relay control pin's state.

3.7.3 Detection Model

3.7.3.1 YOLOv8 (State of the art Deep Learning Computer Vision Model)

YOLOv8 (You Only Look Once version 8) is the latest iteration in the YOLO series, is a cutting-edge, state-of-the-art (SOTA) model that builds upon the success of previous versions while introducing new features and improvements to enhance performance and flexibility. Designed for speed, accuracy, and ease of use, YOLOv8 excels in a wide range of tasks, including object detection, image segmentation, and image classification. Its capabilities make it ideal for real-time applications across diverse domains such as surveillance, autonomous vehicles, healthcare, and underwater exploration, establishing it as a versatile and powerful tool in modern computer vision.

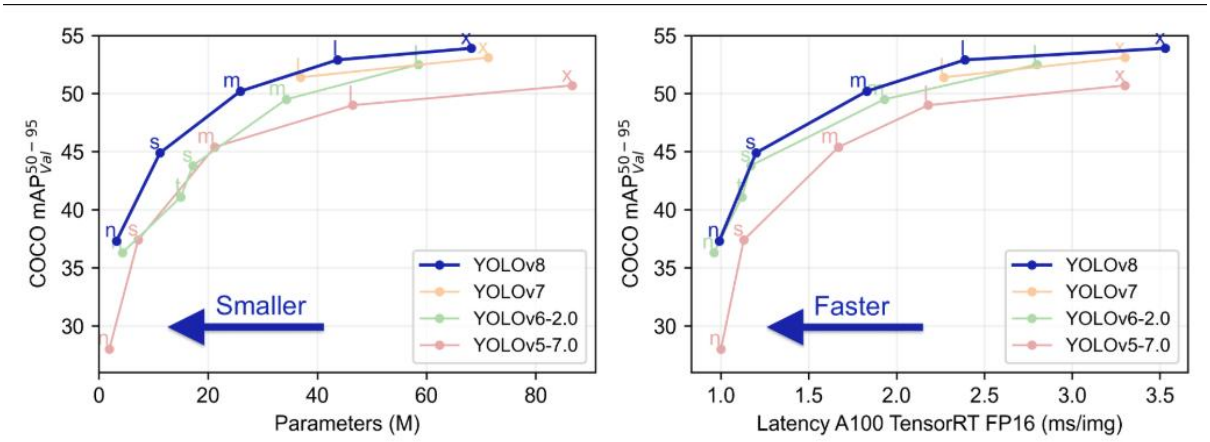


Figure 3.16 The difference between yellow V8 and last versions

3.7.3.1.1 Key Features of YOLOv8:

- High Accuracy and Speed:
    - Achieves exceptional detection accuracy with minimal latency, making it ideal for real-time operations.

- Incorporates cutting-edge algorithms to optimize performance on resource-constrained devices like edge AI and embedded systems.

- Ease of Use:
  - Provides a seamless Python API and command-line interface for effortless integration.
  - Offers built-in support for training, validation, and deployment workflows.
- Scalability and Flexibility:
  - Supports diverse model sizes (e.g., YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x) to cater to various computational resources and application requirements.
  - Easily adaptable to custom datasets and tasks through transfer learning and fine-tuning.
- Advanced Features:
  - Enhanced backbone architecture for better feature extraction and robustness.
  - Incorporates features like object tracking and segmentation, expanding its usability beyond traditional object detection.

3.7.3.1.2 Models Of YOLOv8:

- **Object Detection:** Detecting and classifying objects in images and videos in real time.
- **Instance Segmentation:** Identifying individual objects within complex scenes.
- **Classify:** assigns labels to objects within images, identifying and categorizing them based on predefined classes.
- **Pose:** detects human body landmarks and estimates the position and orientation of the body in 2D or 3D space.

- **Tracking:** Monitoring object movements across video frames for applications like traffic analysis or surveillance.
- **Custom Tasks:** Easily trainable for specialized use cases, such as underwater object detection or industrial defect inspection.

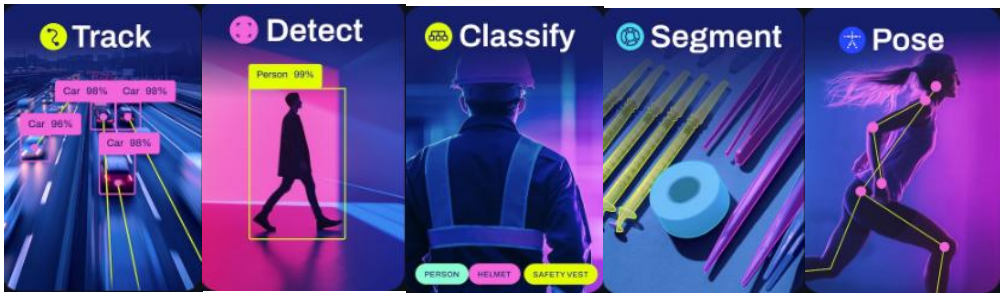


Figure 3.17 Models of YOLOv8

- For Detection Models:

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 3.18 Detection Models

- For Segmentation Models:

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x	640	53.4	43.4	712.1	4.02	71.8	344.1

Figure 3.19 Segmentation Models

- For Classification Models:

Model	size (pixels)	acc top1	acc top5	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B) at 640
YOLOv8n	224	66.6	87.0	12.9	0.31	2.7	4.3
YOLOv8s	224	72.3	91.1	23.4	0.35	6.4	13.5
YOLOv8m	224	76.4	93.2	85.4	0.62	17.0	42.7
YOLOv8l	224	78.0	94.1	163.0	0.87	37.5	99.7
YOLOv8x	224	78.4	94.3	232.0	1.01	57.4	154.8

Figure 3.20 Classification Models

3.7.3.1.3 Key Features of Predict Mode:

YOLOv8's predict mode is designed to be robust and versatile, featuring:

- Multiple Data Source Compatibility: Whether your data is in the form of individual images, a collection of images, video files, or real-time video streams, predict mode has you covered.
- Streaming Mode: Use the streaming feature to generate a memory-efficient generator of **Results** objects. Enable this by setting **stream=True** in the predictor's call method.

- Batch Processing: The ability to process multiple images or video frames in a single batch, further speeding up inference time.
- Integration Friendly: Easily integrate with existing data pipelines and other software components, thanks to its flexible API.

Ultralytics YOLO models return either a Python list of **Results** objects, or a memory-efficient Python generator of Results objects when **stream=True** is passed to the model during inference

3.7.3.2 Creating a Custom Underwater Dataset Using RoboFlow and Google Colab:

3.7.3.2.1 Introduction to Dataset Creation

- Building a custom dataset is essential for ensuring the accuracy and efficiency of machine learning models like YOLOv8.
- The dataset will consist of various underwater objects, including fish, coral reefs, people, and waste items like bottles.
- This process involves two primary platforms: **RoboFlow** and **Google Colab**, which will aid in dataset preparation, training, and analysis.

3.7.3.2.2 Using RoboFlow for Dataset Creation (Steps Just For an Example)

- Step 1:
    - Create an Account on RoboFlow
    - Sign up on RoboFlow to begin the dataset creation process.
    - Once registered, start a new project to manage your dataset.
- Step 2: New Project

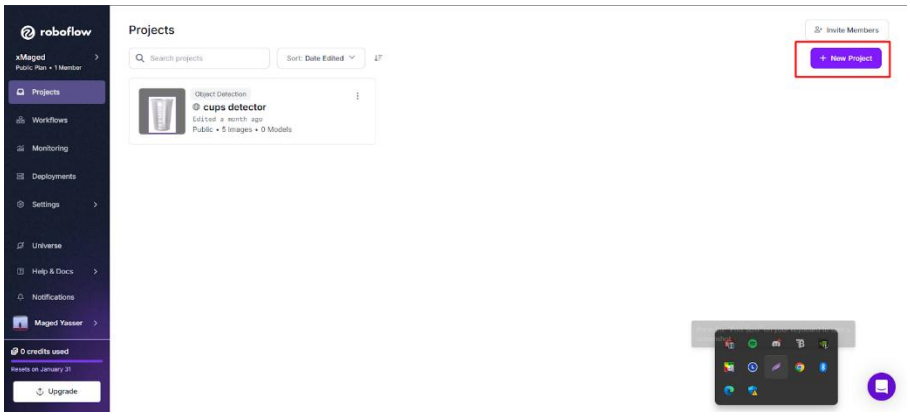


Figure 3.21 New Project

- Step 3: New Project Classification

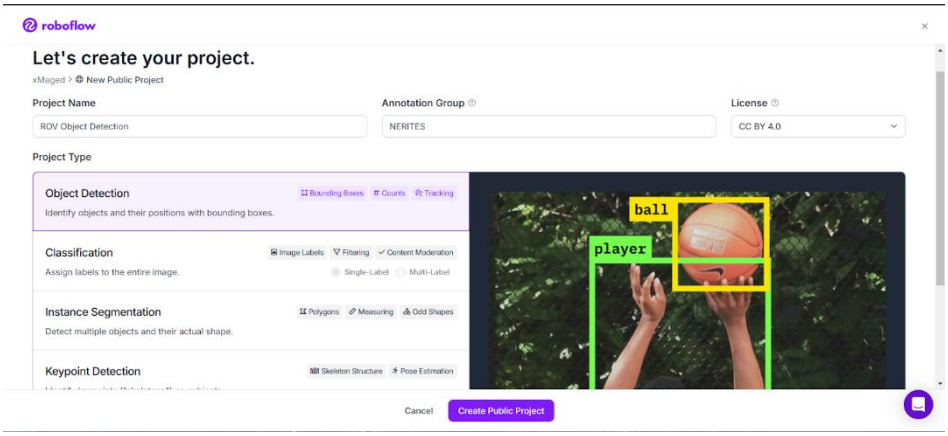


Figure 3.22 New Project Classification

- Step 4: Upload Data

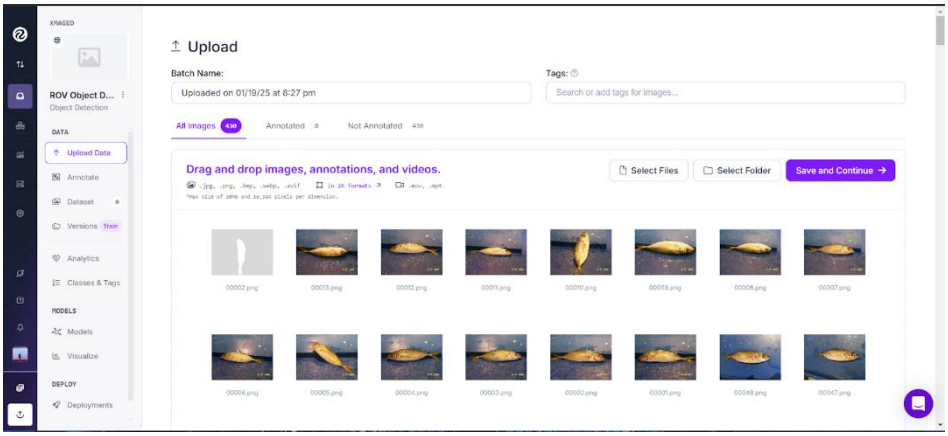


Figure 3.23 Upload Data

- Step 5: Waiting For Uploading

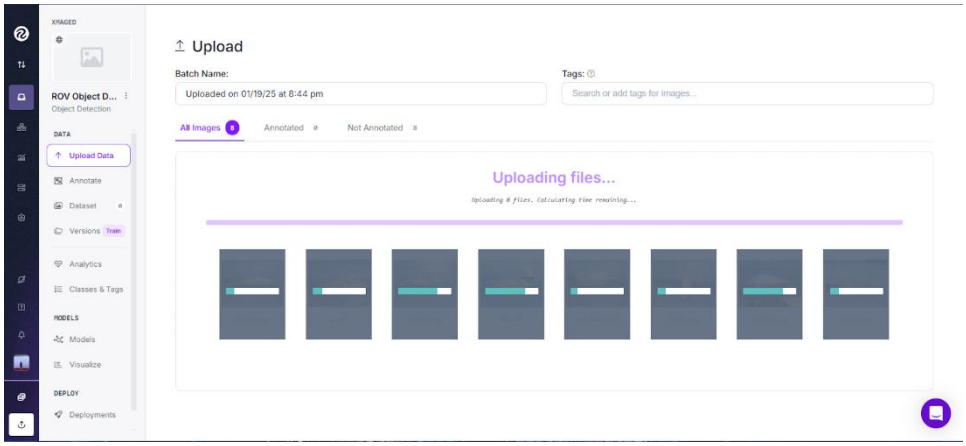


Figure 3.24 Waiting For Uploading

- Step 6: Annotate Data

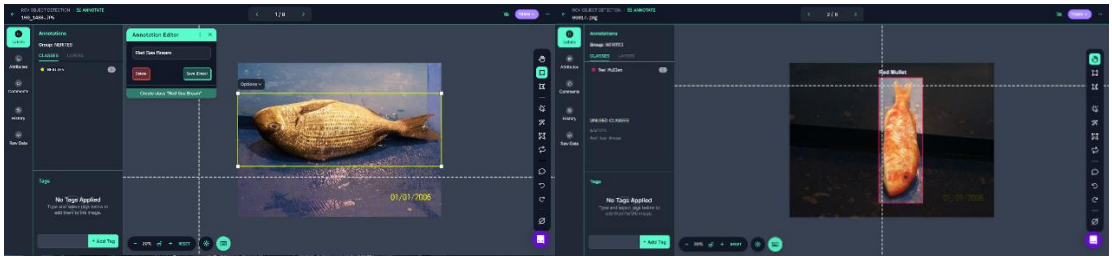


Figure 3.25 Annotate Data

- Step 7: After Annotation

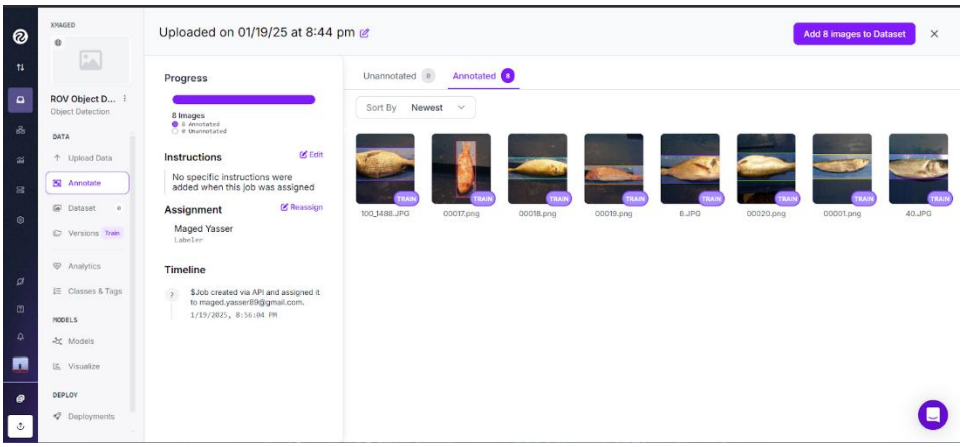


Figure 3.26 After Annotation

- Step 8: Generate New Version

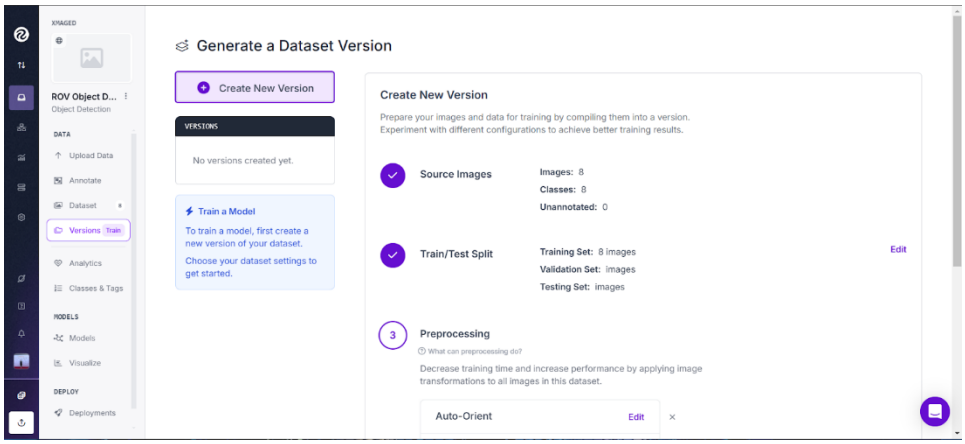


Figure 3.27 Generate New Version

- Step 9: Press "Continue "After Setup Dataset Version, then take Api_Key to put it in **Google Colab**

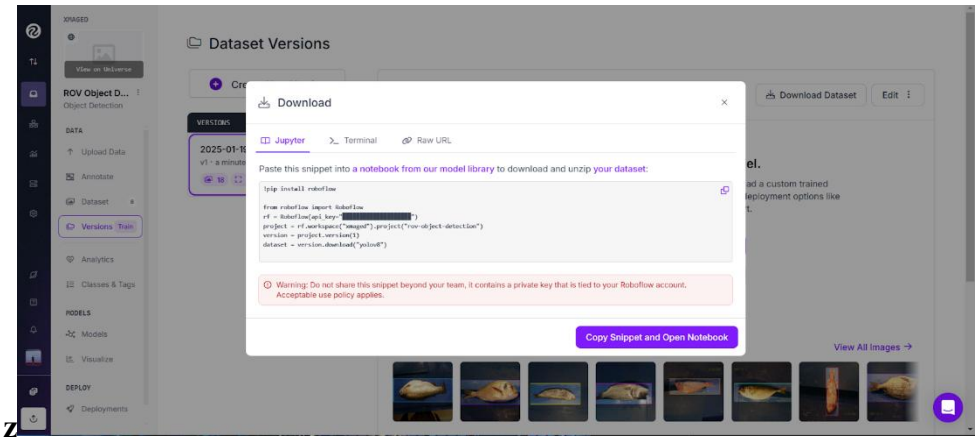


Figure 3.28 Press "Continue "After Setup Dataset Version, then take Api_Key to put it in Google Colab

3.7.3.2.3 Using Google Colab for Organizing and Analyzing Data

- Step 1: Create an Account on Google Colab
    - Sign up on Google Colab to begin the dataset creation process.
- Step 2: Write this Command to installs the ultralytics package
    - which is necessary for working with YOLOv8. The ultralytics library provides a simple and efficient way to train, validate, and deploy YOLO models. By running

this command, you ensure that all the required dependencies for YOLOv8 are installed in your Google Colab environment

```
!pip install ultralytics
```

Figure 3.29 install ultralytics package on Google Colab

- Step 3:
    - Install Roboflow: The first command installs the roboflow package, which is used to access and download datasets from Roboflow.
    - Import Roboflow: The Roboflow class is imported to interact with the Roboflow API.
    - Initialize Roboflow: An instance of Roboflow is created using your API key (####################).
    - Access Project: You access your specific project (rov-detector-xo492) within your workspace (xmaged).
    - Select Version: You specify the version of the dataset you want to use (in this case, version 1).
    - Download Dataset: The dataset is downloaded in the YOLOv8 format, which is compatible with the YOLOv8 model.

```
[ ]  !pip install roboflow

     from roboflow import Roboflow
     rf = Roboflow(api_key="u92QjgQ7t29CGIrBJjcb")
     project = rf.workspace("xmaged").project("rov-detector-xo492")
     version = project.version(1)
     dataset = version.download("yolov8")
```

Figure 3.30 install roboflow package on Google Colab

- Step 4:

- Import YOLO: The YOLO class is imported from the ultralytics package. This class is used to load, train, and evaluate YOLO models.
- Import os: The os module is imported to handle file and directory operations, which might be necessary for managing dataset paths and model files.

```
[ ]  from ultralytics import YOLO
     import os
```

Figure 3.31 Import YOLO, Import OS

- Step 5:
  - Task: The task is set to detect, which is used for object detection.
  - Mode: The mode is set to train, indicating that the model will be trained.
  - Model: The model used is yolov8n.pt, which is a pre-trained YOLOv8 nano model.
  - Data: The path to the dataset configuration file (data.yaml) is specified. This file contains information about the dataset, such as the paths to the training and validation images and labels.
  - Epochs: The model will be trained for 20 epochs.
  - Image Size: The images will be resized to 640x640 pixels during training.

```
# Train YOLOv8n on COCO8 for 20 epochs
!yolo task=detect mode=train model=yolov8n.pt data={dataset.location}/data.yaml epochs=20 imgsz=640
```

Figure 3.32 Train YOLOv8n On COCO8 For 20 epochs

- Step 6:
  - Task: The task is set to detect for object detection.
  - Mode: The mode is set to val, indicating that the model will be validated.

- Model: The path to the best model weights (best.pt) from the training process is specified. These weights are typically saved in the runs/detect/train/weights directory.
- Data: The path to the dataset configuration file (data.yaml) is specified for validation.

```
[ ]  !yolo task=detect mode=val model=/content/runs/detect/train6/weights/best.pt data={dataset.location}/data.yaml
```

Figure 3.33 Command Being Executed

3.7.3.3 Advantages of Using RoboFlow and Google Colab

- RoboFlow:
  - Easy labeling and annotation of objects in images.
  - Advanced data augmentation tools to increase dataset variability.
  - Quick export options for various popular formats compatible with machine learning frameworks.
- Google Colab:
  - Free environment for running Python code with access to GPUs for accelerated training.
  - Flexibility to write and experiment with Python scripts for dataset analysis and model training.

3.7.3.4 Object Detection

It is a task that involves identifying the location and class of objects in an image or video stream.

3.7.3.4.1 Use yolov8n from detection model (Fig3.18)

3.7.3.4.2 The output

The Output of an object detector is a set of bounding boxes that enclose the objects in the image, along with class labels and confidence scores for each box. Object detection is a good choice

when you need to identify objects of interest in a scene, but don't need to know exactly where the object is or its exact shape.

3.7.3.4.3 Main Code for Object Detection by Our Dataset:

```
YOLO_V8 > main.py > ...
1   from ultralytics import YOLO
2   import multiprocessing
3
4   if __name__ == '__main__':
5    multiprocessing.freeze_support()
6
7
8   # Load a model
9   model = YOLO("yolov8n.yaml")  # build a new model from scratch
10  model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)
11
12  # Use the model
13  metrics = model.val()  # evaluate model performance on the validation set
14  model = YOLO (r"C:\Users\maged.XMAGED\OneDrive\Desktop\YOLO_V8\rov-detection.pt") # load a pretra
15  results = model(source=0,show=True,save=True)  # live stream object detection
16  path = model.export(format="onnx")  # export the model to ONNX format
```

Figure 3.34 Main Code for Object Detection

- Results From Model:

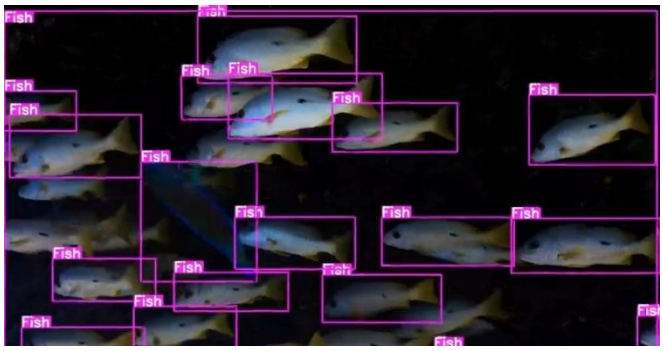


Figure 3.35 Model Result 1

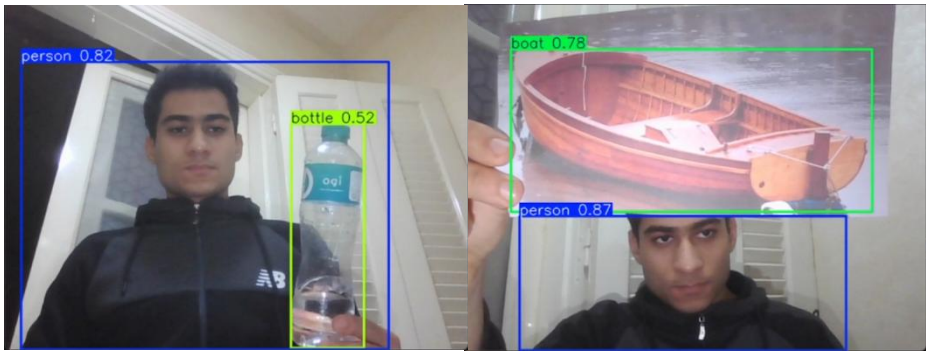


Figure 3.36 Model Result 2

CHAPTER 4: Mechanical System Overview

The mechanical system of the Remotely Operated Vehicle (ROV) serves as the backbone of its underwater capabilities, combining structural durability, hydrodynamic efficiency, and advanced kinematic design to meet the demands of challenging marine environments. This section provides detailed exploration of the mechanical components, from the conceptual design phase to performance evaluations, ensuring the ROV achieves its intended functionality and reliability.

4.1 ROV DESIGN CONCEPT

4.1.1 Design Requirements

A fully submerged vehicle's structure is divided conceptually (and often physically) into three functional subsystems, each of which performs a different role.

- The framework (or frame) serves as the primary skeleton of the vehicle. It is usually made of interconnected beams, plates, or other load-bearing members. The frame defines the vehicle's overall shape, and it provides mechanical support and attachment points for weights, floats, thrusters, cameras, lights, and other components. It also plays a key role as a ribcage to protect other parts of the vehicle from impacts.

- The pressure hull, and its smaller counterparts called pressure canisters or pressure housings, are strong, rigid, watertight containers, usually containing air to normal atmospheric pressure, which are designed to protect their contents from water and extreme pressure. Some ROVs might have several small pressure canisters, each one enclosing a different camera, light, sensor, motor, or other electronic device.

- The fairing is a skin or shell that covers all or part of the frame to give them a smoother, more streamlined shape. A fairing helps the vehicle slide more easily through the water, thereby improving speed and energy efficiency. Fairings are used commonly on vehicles that must travel quickly or far, thus usually used for UAVs but less common in the ROVs.

At a fundamental level, the primary role of a vehicle's structure is to manage all the mechanical forces the vehicle will encounter during its lifetime.

- The frame portion of the structure has the following requirements:
    - It must support the entire vehicle's wight and effectively withstand and distribute the concentrated forces at specific points int the frame.
    - It must withstand the hydrodynamic forces associated with currents, swells, and vehicle propulsion without damage and deformation.
    - It must provide points to position and attach components safely.
- The pressure canisters portions of the structure have these requirements:
    - They must resist water pressure forces, which are enormous at great depths.
    - They must remain watertight sealed against the pressure and, if required, provide a way to access interior components and allow wires to penetrate the canisters while maintaining watertight.
    - They should be easily and securely mounted to the frame.
- The Fairing portion of the structure has the following requirements:
    - It should provide a streamlined shape like fishes and waterdrops to minimize drag force.

4.1.2 Design Choice Criteria

All underwater crafts have a structure, but the materials and designs may vary significantly because of differing mission requirements.

- Strength and Stiffness are arguably the most critical aspects of structure's mechanical performance.
- Weight is another key factor in overall mechanical performance.
- Compatibility with the underwater environment is a crucial factor for safety.

Other than these crucial factors, there are other criteria to consider as they sometimes make a significant difference in choosing the design.

- Ease of construction, maintenance, and repair can affect the choice of using a Commercial Off-The-Shelf part or just design it.
- Cost and availability of parts and raw materials.
- Aesthetics of the design increases performance sometimes as almost all hydrodynamically efficient shapes have a good-looking design.

4.1.3 Material and Design Chosen

For the vehicle frame, an acrylic plate is used for its high stiffness-to-weight ratio and corrosion resistance. The canister is a cylindrical acrylic connected to the main plate. For the fairing, glass-reinforced plastic, also called fiberglass is used for its surface finish to provide a streamlined fair and high strength-to-weight ratio for the vehicle. It also secures the thruster-configurations mechanism used. The mechanism parts' material used are different from each other. The screw, nut, shafts, and connectors are made of Polypropylene (PP), the links and gears are made of acrylic. Finally, the bearings were chosen to be water resistant made of PVDF plastic to be chemically stable and corrosion resistant.

4.2 KINEMATICS

4.2.1 Forces acting on the ROV.

Five basic forces act on all rigid bodies moving in water:

- Drag
- Thrust
- Weight
- Buoyancy
- Lift

The naturally occurring force that resists the relative motion between an object and the fluid surrounding it is called Drag. Dragging is usually undesirable, but also unavoidable. Thus, the design should try to minimize it to increase maximum speed and efficiency. Its opposite force is

the thrust provided by the rotating propellers. To stay in place, the thrust force must equal the drag force to cancel it. While drag acts exactly opposite to the actual direction of vehicle's motion, the thrust acts in the same direction of the motion. Thrust required should be greater than the overall drag which can be calculated using equation 3.1:

$$D = \frac{1}{2}\rho C_d A U^2$$

Drag ($\boldsymbol{D}$ in newtons) discussed above, fluid density ($\boldsymbol{\rho}$ in kilograms per cubic meter) varies with temperature, salinity, and depth, drag coefficient ($\boldsymbol{C_d}$ with no unit), frontal area ($\boldsymbol{A}$ in square meters) perpendicular to the motion axis, and maximum speed ($\boldsymbol{U}$ in meters per second) through the fluid. For complex shapes like ROVs it can be calculated through computational fluid dynamics.

Weight is always a downward force caused by the gravity acting on the vehicle's mass. Its opposing force is the buoyancy force which acts straight upward as shown in figure 6.1. As the weight acts to sink the ROV the buoyancy is the force that helps it float. Buoyancy force can be calculated using Archimedes' principle.

The last force to discuss is Lift that acts perpendicular to the axis of motion. It is naturally generated due to change in pressure; its magnitude depends on the vehicle's speed and shape. Even though its importance in many applications like cars and planes, lift force is insignificant in many ROV applications. These forces do not act on the ROV individually, but to the net force created when all the forces acting on a vehicle are considered simultaneously. Therefore, the design should consider all these combined to know the effect on the vehicle's center of gravity (CG) and Center of buoyancy (CB).

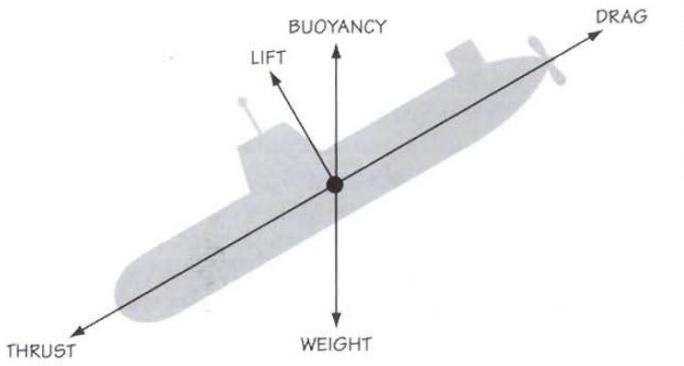


Figure 4.1 Forces acting on the ROV

4.2.2 ROV Movement

ROVs can have at most six degrees of freedom for movement, providing comprehensive control over their orientation and position underwater. The degrees of freedom include:

- **Surge** (forward/backward)
- **Sway** (left/right)
- **Heave** (up/down)
- **Roll**, **Pitch**, and **Yaw** (rotational movements about X, Y, and Z axes respectively)

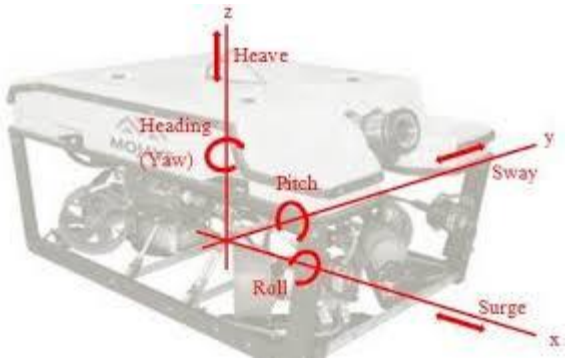


Figure 4.2 ROV Movement

This capability allows the ROV to perform precise maneuvers required for tasks such as pipeline inspections, underwater surveying, and object retrieval. Advanced control algorithms implemented in the software complement the mechanical design to achieve seamless operation.

4.2.3 Thrusters Configuration

Achieving this level of control is made possible by the strategic placement, configuration, and number of the thrusters. The propulsion system comprises six thrusters, each contributing to specific movements:

- **Vertical Thrusters**: Two thrusters are mounted along the Z-axis, enabling precise heave movements and pitch rotations. This design ensures stable ascent, descent, and angular adjustments for tasks like inspecting vertical structures or adjusting to varying depths.
- **Horizontal Thrusters**: Four thrusters are symmetrically positioned to control surge, sway, and yaw movements. These thrusters are connected to a dynamic mechanism that allows them to switch between two configurations:
    - **45-degree alignment**: Optimized for diagonal and complex directional movements, this configuration provides exceptional maneuverability in tight or intricate operational areas.
    - **Forward-facing alignment**: In this mode, all horizontal thrusters point forward, maximizing thrust for high-speed movement. This configuration is particularly useful for long-distance travel or missions requiring rapid deployment.

The ability to seamlessly transition between different thruster configurations provides unparalleled adaptability. For instance, during a pipeline inspection, the 45-degree configuration enables fine adjustments and close-range maneuvering, while the forward-facing alignment allows rapid transit between inspection sites. This versatility ensures the ROV is well-equipped to handle a broad range of operational scenarios, from delicate exploratory missions to high-speed reconnaissance tasks.

By combining robust mechanical design, strategic thruster placement, and advanced kinematic control, the ROV movement system sets a high standard for underwater performance. This comprehensive approach ensures the ROV can effectively meet the diverse challenges of its intended applications.

4.3 FINAL DESIGN

4.3.1 Foundations of Robot Design with SolidWorks

This section explains the design process of the ROV, detailing the steps taken to transform an initial concept into a fully functional product. It highlights the use of advanced computer-aided design (CAD) and engineering software, chosen for their powerful capabilities in modelling as shown in Fig. 4.1, assembly, rendering, and mechanical simulations. Employing reliable tools to create and visualize complex 3D models is crucial for a successful design. To this end, SOLIDWORKS, a widely recognized CAD and CAE software, was utilized throughout the design process. By leveraging SOLIDWORKS' comprehensive features and functionality, the design team translated their ideas into tangible representations of the robot's mechanical components as we have here the final CAD design for the Multi-Functional ROV in Fig. 4.2.

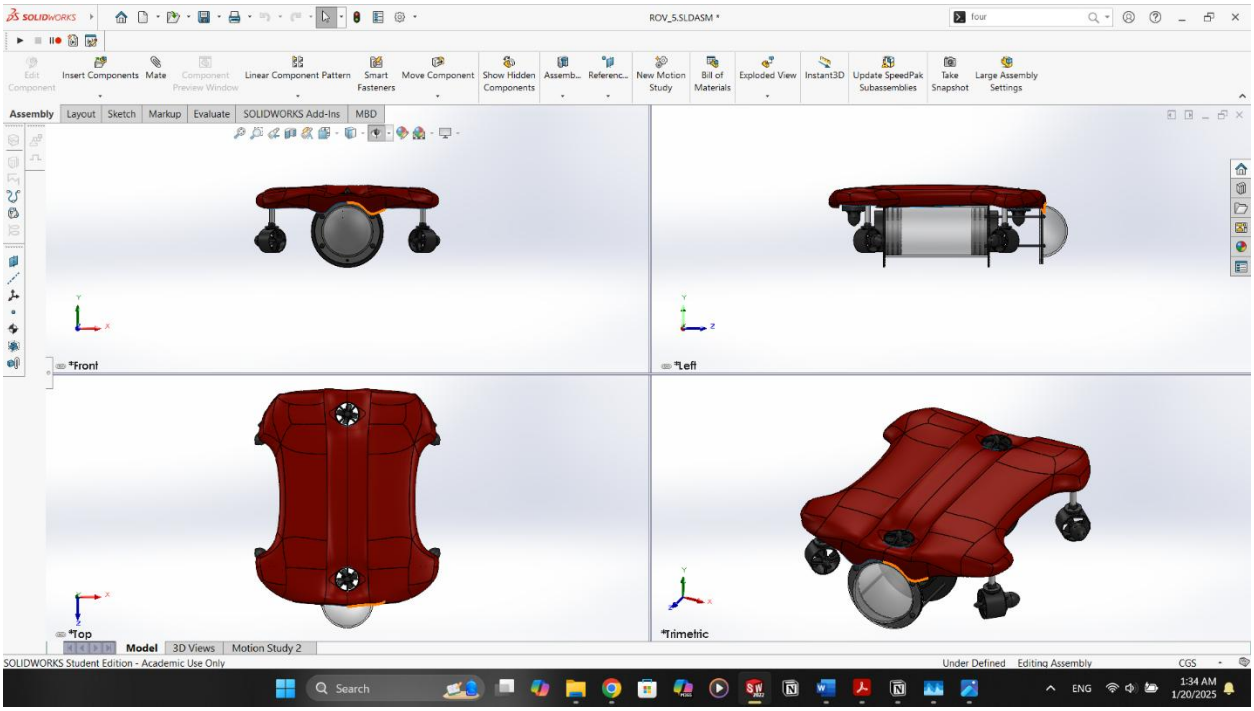


Figure 4.3 Robot Design

SOLIDWORKS' robust solid modelling capabilities enabled the creation of highly detailed 3D models, capturing the intricate geometry and functional aspects of the robot. The assembly features provided by SOLIDWORKS allowed the individual components to seamlessly integrate, forming a coherent and cohesive structure. Additionally,

SOLIDWORKS' rendering capabilities were crucial in generating realistic visual representations of the robot's design. By applying textures, materials, and lighting effects, the design team visualized the robot's appearance and evaluated its aesthetics. Designing with SolidWorks has many advantages:

- **Parametric Modelling**: SolidWorks' parametric modelling allows engineers to create intelligent, adaptable designs, crucial for the iterative process inherent in robot development.
- **Assembly Modelling**: Assembling complex robotic systems is made intuitively with SolidWorks, allowing engineers to visualize the integration of components and mechanisms.
- **Simulation and Analysis**: The software's simulation capabilities enable engineers to test and refine the performance of robotic systems, ensuring they meet design specifications.

4.3.2 Main Assembly

The final design of the remotely operated vehicle (ROV) displays a sleek, streamlined, and robust structure that prioritizes both functionality and aesthetics. The body is covered with a red, contoured fairing, which has been carefully designed to enhance hydrodynamic efficiency by reducing drag as the ROV navigates through underwater environments. This fairing not only improves performance but also provides a protective shield for the internal components, ensuring durability and reliability during operation. Strategically placed cutouts in the fairing allow for ventilation and accessibility to essential components, striking a balance between protection and practicality.

The design incorporates four thrusters, mounted symmetrically at the corners, providing precise control and maneuverability across all six degrees of freedom, including forward, backward, lateral, vertical, and rotational movements. This configuration allows the ROV to perform complex tasks with stability and agility, even in challenging underwater conditions. The modular layout of the design ensures easy assembly and maintenance, with each component accessible for repairs or upgrades, enhancing the system’s adaptability and long-term usability.

Compact and symmetrical structure optimizes space utilization while maintaining structural integrity, making the ROV suitable for a variety of applications, such as inspection, exploration, or environmental monitoring. The overall design represents a balance between advanced engineering and practical considerations, resulting in a highly efficient, versatile, and visually striking underwater vehicle capable of addressing diverse operational challenges. Our ROV was designed using Top-Down assembly starting with top plate then the sport-mode mechanism was designed based on the plate. After thar the canister was added and finally the fairing was attached to the top plate.

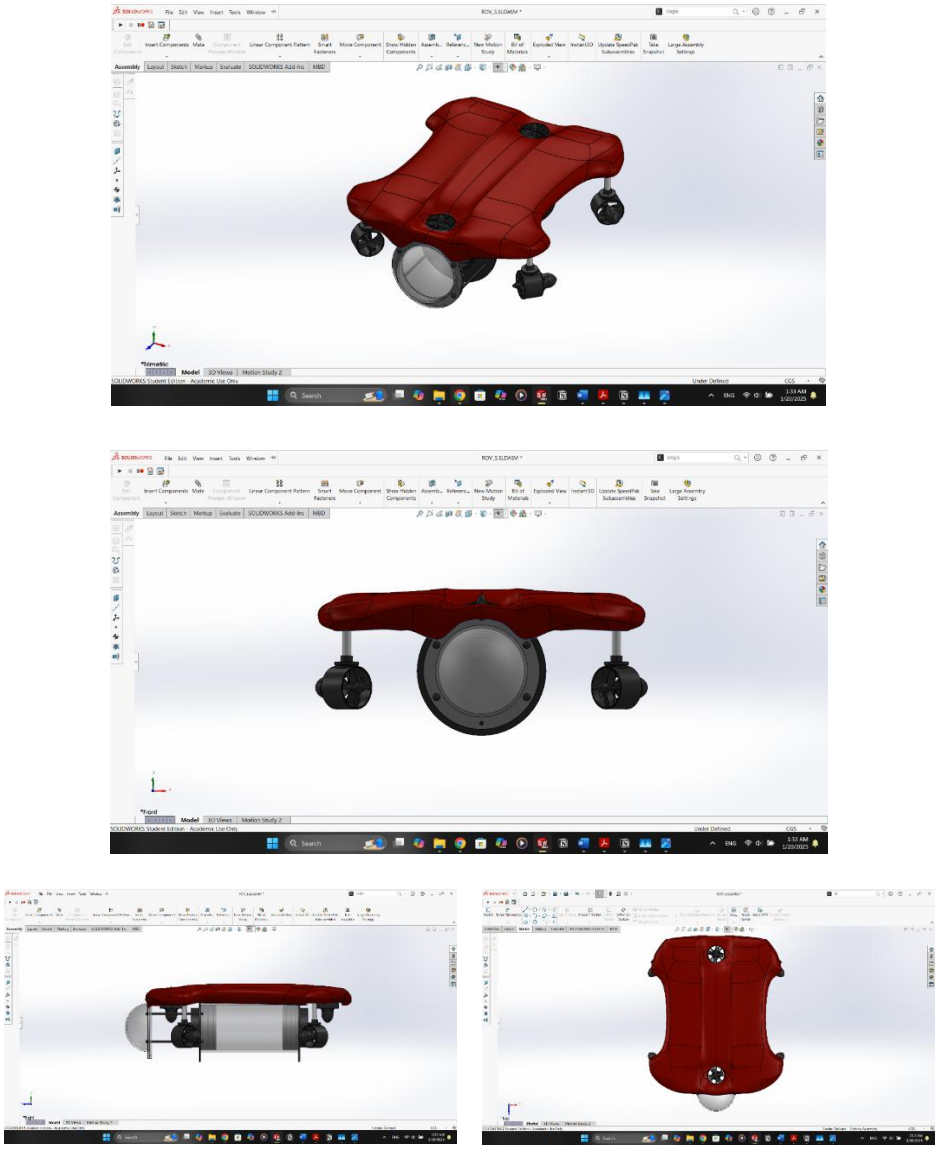


Figure 4.4 Main Assembly

4.3.3 Pressure Canisters

The pressure canister design is an essential component of the ROV, combining functionality with a modular structure to accommodate specific operational requirements. The canister is primarily constructed from acrylic, which provides a balance of transparency for visual monitoring and sufficient strength to withstand underwater pressure at the intended operating depth.

The canister is divided into two main sections to optimize the internal arrangement of components and facilitate efficient performance. The frontal section is dedicated to housing the camera, ensuring a clear field of vision for capturing underwater images and videos. This section is isolated from the rear portion to allow space for the vertical thruster to be positioned in between, providing unobstructed movement and efficient thrust generation. The camera's placement in a transparent and sealed section ensures it remains functional and protected under high-pressure underwater conditions.

The rear section of the canister is designed to house all the critical electrical components of the ROV. This layout ensures a clear separation between the optical and electronic systems, reducing the risk of interference and facilitating easier maintenance and troubleshooting. The electrical components are securely placed within this section, protected from environmental factors while maintaining easy access through modular design.

The two sections of the pressure canister are connected using four robust bolts, ensuring a secure and watertight seal while allowing for disassembly when needed. The structural integrity of this connection is critical to maintaining the ROV's ability to operate at varying depths without compromising the internal components. This design not only enhances the functionality of the ROV but also demonstrates a thoughtful approach to modularity, maintenance, and operational efficiency.

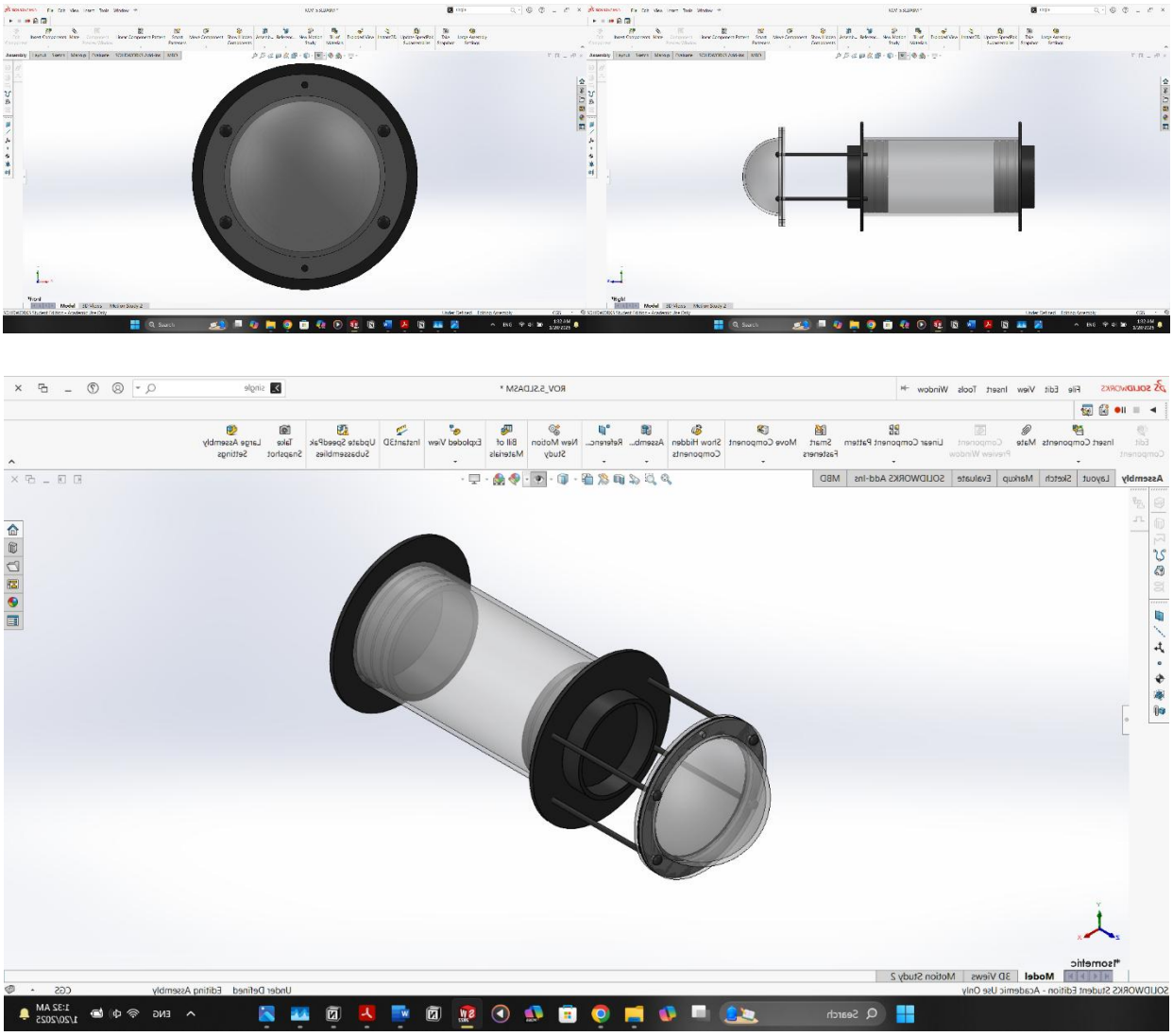


Figure 4.5 Pressure Canisters

4.3.4 Fairing

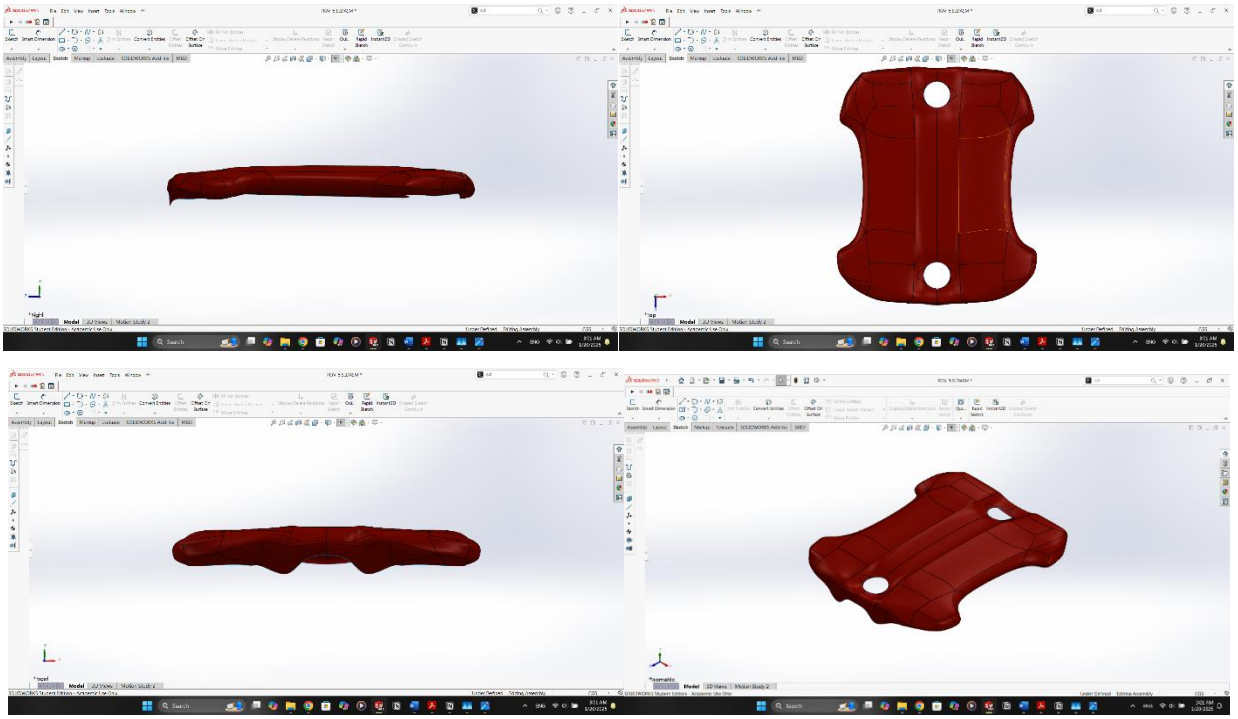
The fairing of the ROV has been thoughtfully designed to optimize underwater performance. It features a streamlined, smooth shape that reduces drag and turbulence, allowing the ROV to move efficiently through water. Its symmetrical structure ensures balanced hydrodynamic performance, which is essential for stability and maneuverability. The fairing incorporates two circular openings, likely for components such as propellers, sensors, or water intake systems. With seamless surface transitions and no sharp edges, the design promotes efficient water flow and minimizes resistance. Subtle ridges and contours add structural reinforcement while

maintaining an elegant appearance. The fairing is constructed from **fiberglass**, a material renowned for its lightweight nature, durability, and corrosion resistance—qualities that make it ideal for underwater applications.

The manufacturing process of fiberglass fairing involves several critical steps. Fiberglass is created by layering woven fiberglass fabric over a mold that defines the shape of the fairing. The mold is typically prepared with a release agent to ensure easy removal after curing. Once the fabric is laid out, it is impregnated with resin, which binds the fibers together and provides structural rigidity. For this fairing, the resin used is likely **epoxy resin**, a common choice for marine applications due to its excellent adhesion, water resistance, and mechanical strength. Epoxy resin also provides a smooth finish, which is important for the hydrodynamic performance of the ROV. After applying the resin, the fairing is cured either at room temperature or with additional heat, depending on the specific resin system. Once cured, the fairing is removed from the mold, trimmed to its final dimensions, and sanded or polished for a clean finish.

The **Power Surface Tool** in SolidWorks was instrumental in designing this fairing. This tool enables the creation of complex, freeform surfaces by manipulating a control mesh made up of vertices, edges, and faces. Adjusting these control points allowed us to achieve the desired curvature and smoothness, ensuring that the final design met both functional and aesthetic requirements. The tool also includes features for symmetry and blending, ensuring that the fairing's structure is both precise and visually appealing.

By combining advanced design techniques with a well-considered manufacturing process and the use of fiberglass reinforced with epoxy resin, the fairing achieves a balance between structural strength, lightweight construction, and hydrodynamic efficiency. This approach ensures that the ROV fairing performs effectively in demanding underwater environments while maintaining durability and resistance to water and pressure.



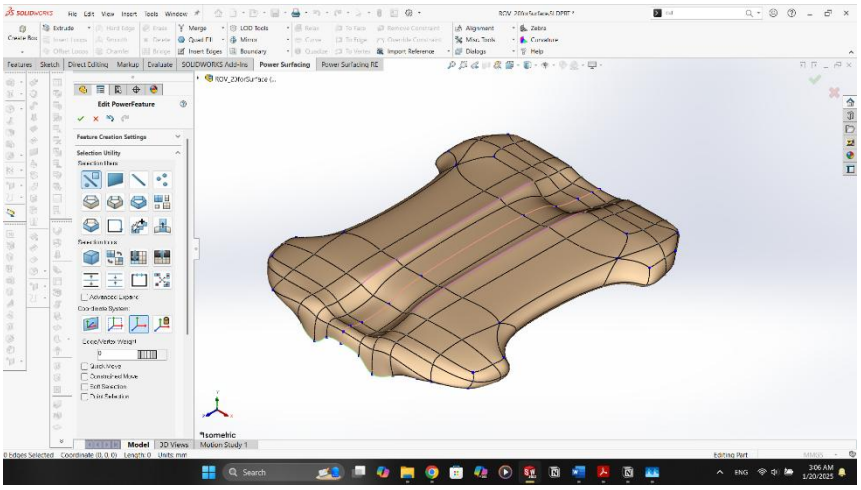


Figure 4.6 Fairing Of ROV

4.3.5 Sport Mode Mechanism “Steering Thrusters Mechanism”

4.3.5.1 Abstract

This section presents the design and implementation of a mechanical system that employs a single stepper motor to synchronize the rotation angles of both front and rear thrusters. The system is engineered to achieve precise rotational control, with the front thrusters rotating at

45 degrees and the rear thrusters at 135 degrees, ensuring consistent directional alignment. The innovative design incorporates a lead screw, an offset crank-slider mechanism, and a gear ratio system for the rear thrusters.

4.3.5.2 Introduction

4.3.5.3 Thrusters are essential components in controlling the directional movement of various vehicles, especially in marine and aerial applications. This project focuses on streamlining the actuation system for steering thrusters by utilizing a single stepper motor to govern the motion of both front and rear thrusters. By harnessing the principles of mechanical linkages and gear systems, the design achieves synchronized and precise control, all while maintaining high mechanical efficiency.

4.3.5.4 Mechanism Design

4.3.5.4.1 Overview

The proposed mechanism utilizes an offset crank-slider system, powered by a single stepper motor connected to a lead screw. The rotational motion of the stepper motor is transformed into linear motion via a slider, which then drives the crank to produce the precise angular rotation required for the thrusters. This innovative approach ensures efficient and accurate control of the system.

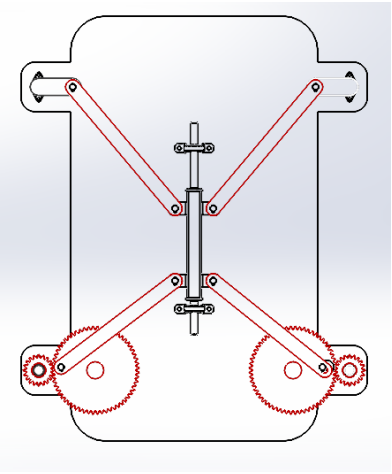


Figure 4.7 Sport Mode Mechanism Design

4.3.5.4.2 Front Thrusters

The front thrusters are directly linked to the crank of the offset crank-slider mechanism. As the slider moves linearly along the lead screw, it drives the crank to rotate. The geometry of the

crank-slider mechanism is meticulously designed to generate a precise rotational output of 45 degrees for the front thrusters, ensuring accurate and reliable performance.

4.3.5.4.3 Rear Thrusters with Gear Ratio System

To achieve the required 135-degree rotation, a gear train is implemented. This system serves two purposes:

- **Direction Reversal**: Ensures the rear thrusters rotate in the same direction as the front thrusters.
- **Angle Multiplication**: A 3:1 gear ratio (60-tooth gear meshed with a 20-tooth gear) triples the crank's rotation, resulting in a 135-degree output for the rear thrusters.

4.3.5.5 Actuation Mechanism

- Stepper Motor and Lead Screw
    - The stepper motor converts electrical signals into precise rotational motion. This motion drives the lead screw, which translates into linear motion of the nut (slider). The lead screw's pitch and the stepper motor's step angle are selected to provide the required resolution and range of movement for the slider.
- Slider-Crank Motion Conversion
    - The linear motion of the slider is transmitted to the crank, causing it to rotate. The offset distance of the crank determines the amplitude of the angular rotation, allowing precise control of the front thrusters' rotation angle.
- Gear System for Rear Thrusters
    - The crank in the rear thruster assembly is coupled to a 60-tooth gear, which meshes with a 20-tooth gear. This arrangement ensures that the angular rotation of the rear thrusters is triple that of the crank, achieving the desired 135-degree rotation.

4.3.5.6 Advantages of the Design

- Single Motor Control:

- Utilizing a single stepper motor streamlines the control system, thereby reducing the overall complexity of the design.

- Synchronization:
  - Both the front and rear thrusters are synchronized to rotate in the same direction, ensuring a consistent thrust direction.

- Compact and Efficient:
  - The offset crank-slider mechanism, in conjunction with the gear train, offers a mechanically efficient and compact solution for achieving the desired motion.

- Scalability:
  - The design can be easily scaled to accommodate different sizes or adapted for a variety of applications.

4.3.5.7 Mathematical Analysis

- Nut Displacement (y)
  - The linear displacement of the slider y, is directly proportional to the number of revolutions of the stepper motor and the lead of the screw rod, given by:

$$y = nLead$$

- Front Crank Rotation Angle (front)
  - The angle of rotation of the front crank arm is determined by the value required in the design, which is 45 degrees. Therefore, the slider must be moved a specific distance d to achieve that angle This value will be found with a graphical solutionز

- Rear Crank Rotation Angle (Rear)
  - The crank rotation angle is derived from the geometry of the crank-slider mechanism. It is given by:

$$\theta_{Rear} = Gr\ \theta_{front}$$

where Gr is gear ratio of rear gear system

4.3.5.8 Calculations

The required displacement d of slider to rotate the front crank arm front by 45 degrees was determined graphically. Using the dimensions of the mechanism, where:

- Crank arm length R = 4.8 cm
- Connecting rod length L = 22.3 cm
- Offset between the sliding axis and the crank center e = 19.3 cm

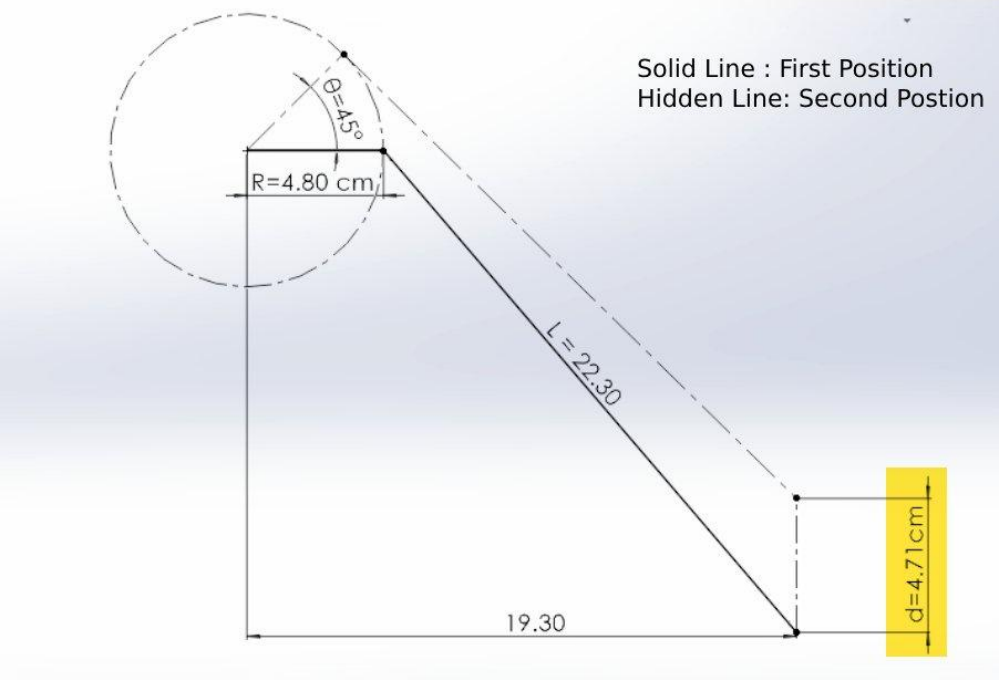


Figure 4.8 Calculations

The graphical solution showed that the necessary displacement $d = 4.71\ cm$

To achieve the rear crank arm rotation by Rear 135 degrees while the front crank rotates by front by 45 degrees a gear ratio of 3:1 was selected. This ensures the rear crank rotates three times the angle of the front crank.

The number of rotations of the lead screw (nnn) required to achieve the slider displacement was calculated by dividing the displacement ddd by the pitch of the lead screw:

$$n = \frac{d}{Lead}$$

With a lead screw pitch of 0.8 cm (8 mm), the number of rotations required is:

$$n = \frac{4.71}{0.8} = 5.8875 \ revolution$$

Degrees of rotation of stepper motor can be calculated by:

$$stepper = 5.8875 \times 360 = 2119.5 \, degree$$

Thus, the stepper motor must complete approximately **2119.5 degrees or 5.8875 revolutions** to achieve the desired movement. This result ensures precise synchronization of the front and rear crank arms' motion.

4.3.5.9 Comparison Between 45˚ Horizontal Thrusters Configuration and Straight Forward Horizontal Thrusters Configuration in Thrust and Speed of ROV

i. 45˚ Horizontal Thrusters Configuration

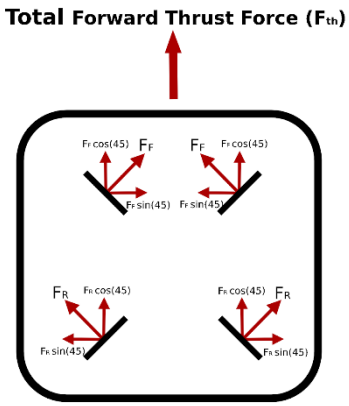


Figure 4.9 Comparison Between 45˚ Horizontal Thrusters Configuration and Straight Forward Horizontal Thrusters Configuration in Thrust and Speed of ROV

- Total Forward Thrust Force (Fth1)

$$F_{th1} = 2F_F cos(45) + 2F_R cos(45)$$
$$where\ F_R = R_r F_F$$
$$F_{th1} = 2F_F cos(45) + 2R_r F_F cos(45)$$
$$F_{th1} = 2F_F cos(45)(1 + R_r)$$

- Maximum Velocity of Vehicle (v1)

$$F_{th1} = 0.5\rho C_D A v_1^2$$

$$v_1 = \sqrt{\frac{2F_F cos(45)(1 + R_r) \times 9.81}{0.5\rho C_D A}}$$

ii. Straight Forward Horizontal Thrusters Configuration

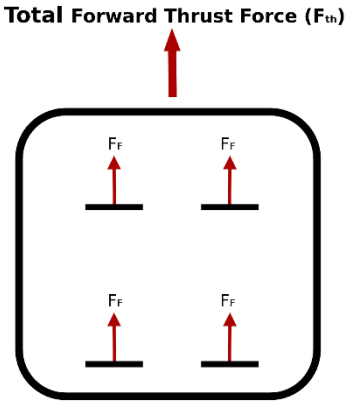


Figure 4.10 Straight Forward Horizontal Thrusters Configuration

- Total Forward Thrust Force (Fth2)

$$F_{th2} = 4F_F$$

- Maximum Speed of Vehicle (v2)

$$F_{th2} = 0.5\rho CDA v_2^2$$
$$v2 = \sqrt{\frac{4F_F \times 9.81}{0.5\rho CDA}}$$

4.3.5.10 Percentage of Increasing in Thrust Force When Use Sport Mode

$$= \frac{F_{th2} - F_{th1}}{F_{th1}} \times 100$$

$$= \frac{4F_F - 2F_F cos(45)(1 + R_r)}{2F_F cos(45)(1 + R_r)} \times 100$$

$$= \frac{2 - cos(45)(1 + R_r)}{cos(45)(1 + R_r)} \times 100$$

for ApisQueen X3 Thruster, $Rr = \frac{15}{26}$

$$= \frac{2 - cos(45)\ (1 + {}^{15}/_{26})}{cos(45)\ (1 + {}^{15}/_{26})} \times 100$$

$$= 79.36\%$$

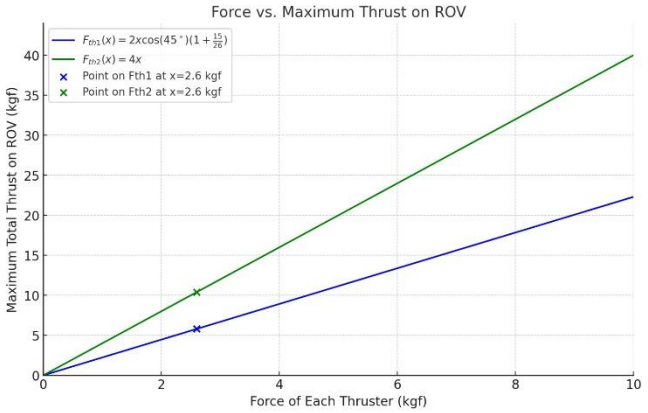


Figure 4.11 Percentage of Increasing in Thrust Force When Use Sport Mode

4.3.5.11 Percentage of Increasing in Speed When Use Sport Mode

$$\frac{v_2 - v_1}{v_1} \times 100$$

$$\frac{\sqrt{\frac{4F_F}{0.5\rho CDA}} - \sqrt{\frac{2F_F cos(45)(1 + R_r)}{0.5\rho CDA}}}{\sqrt{\frac{2F_F cos(45)(1 + R_r)}{0.5\rho CDA}}} \times 100$$

$$\frac{\sqrt{4} - \sqrt{2cos(45)(1 + R_r)}}{\sqrt{2cos(45)(1 + R_r)}} \times 100$$

for ApisQueen X3 Thruster, $Rr = \frac{15}{26}$

Figure 4.12 Percentage of Increasing in Speed When Use Sport Mode

$$\frac{\sqrt{4} - \sqrt{2cos(45)(1 + {}^{15}/_{26})}}{\sqrt{2cos(45)(1 + {}^{15}/_{26})}} \times 100$$

$$= 33.93\%$$

4.3.5.12 Percentage of reduction in power consumption when thrust and speed are constant after using sports mode

APISQUEEN X3 UNDERWATER THRUSTER									
Forward					Reversal				
Voltage (V)	Throttle	Current (A)	RPM	Thrust (Kg)	Voltage (V)	Throttle	Current (A)	RPM	Thrust (Kg)
24	1.600ms	0.3	1131	0.20	24	1.400ms	0.4	1165	0.20
24	1.650ms	0.6	1542	0.40	24	1.350ms	0.6	1645	0.30
24	1.700ms	1.2	1920	0.70	24	1.300ms	1.2	2057	0.50
24	1.750ms	1.8	2262	1.00	24	1.250ms	1.9	2400	0.70
24	1.800ms	2.8	2500	1.20	24	1.200ms	2.7	2700	0.90
24	1.850ms	4	2777	1.50	24	1.150ms	4	3051	1.00
24	1.900ms	5	3017	1.80	24	1.100ms	5	3291	1.20
24	1.950ms	6	3222	2.00	24	1.050ms	6	3497	1.40
24	2.000ms	8	3390	2.60	24	1.000ms	8	3702	1.50

Figure 4.13 Underwater Thruster Datasheet

Maximum Total Forward Thrust $F_{th1} = 2F_F cos(45)(1 + R_r)$ When $F_F = 2.6\ kgf$ and $Rr = \frac{15}{26}$ $F_{th1} = 5.8\ kgf$ From Datasheet Table at $F_F = 2.6\ kgf$ Current = 8 A Power = 192 w	Required Total Forward Thrust $F_{th2} = 5.8\ kgf$ Required Force of Each Motor $5.8 = 4F_F$ $F_F = 1.45\ kgf$ From Datasheet Table at $F_F = 1.45\ kgf$ Current = 4 A Power = 96 w

$$= \frac{P1 - P2}{P1} \times 100$$

$$= \frac{192 - 96}{192} \times 100$$

$$= 50\%$$

Table 4.1 Percentage of reduction in power consumption when thrust and speed are constant after using sports mode

4.3.5.13 Mechanism components

The mechanism assembly is fixed on the main plate which will be laser cut to hold all components as the structure of the ROV.

the main part of the mechanism, the power screw, is an assembly of 2 nuts connected by square shaped rod, nut connector, with inner thread to move through the lead screw.

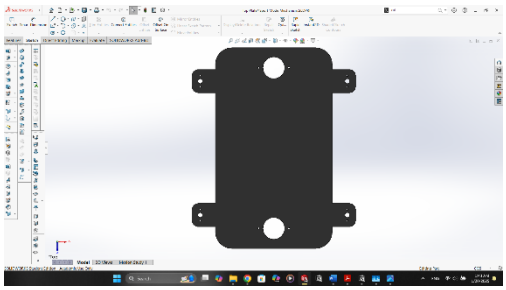


Figure 4.15 main plate

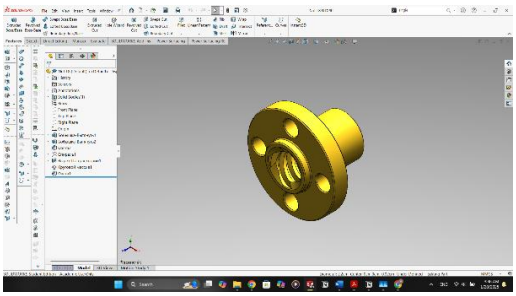


Figure 4.14 The nut

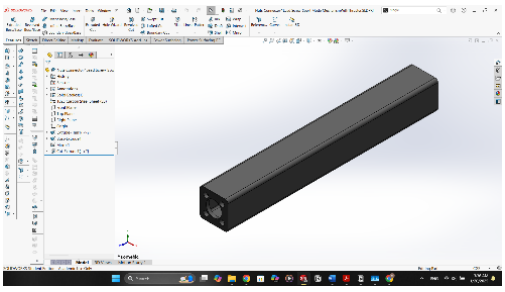


Figure 4.17 square shaped rod "nut connector"

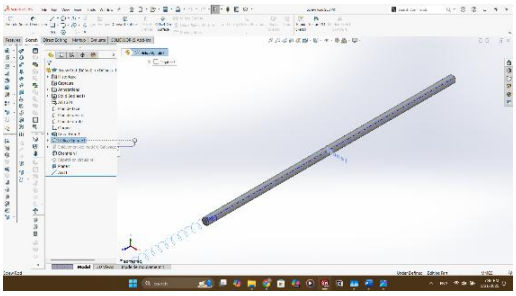


Figure 4.16 Lead screw

The lead screw itself is fixed to the plate itself through 2 PVDF bearings which are made for underwater usage.

A 4fixation parts are connected welded to the connector to translate its movement to the 4 links.

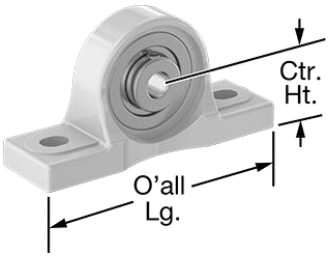


Figure 4.18 PVDF bearing

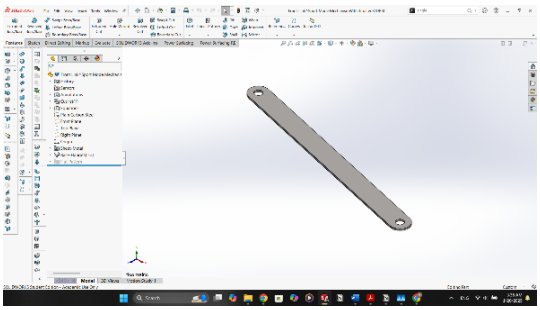


Figure 4.20 Link

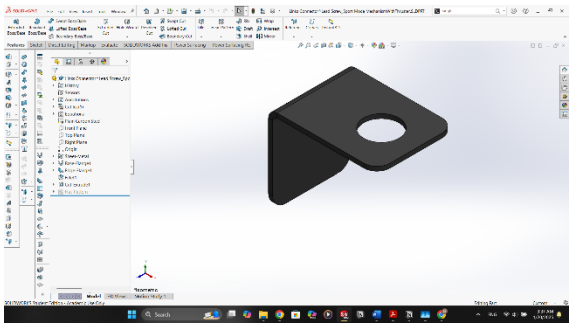


Figure 4.19 Fixation

Each of the 2 front links is connected to a shaft that translates the rotational movement to the thruster via a coupler, while the rear links are connected to a driving gear that drives the driven gear connected to the rear shaft that is also rotated the thruster via a coupler.

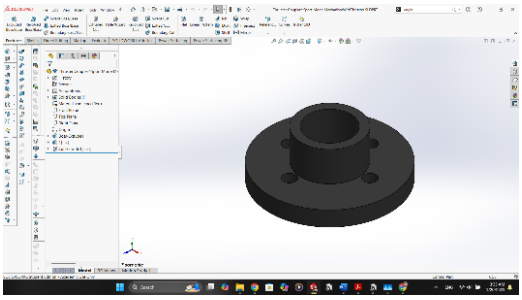


Figure 4.22 Coupler

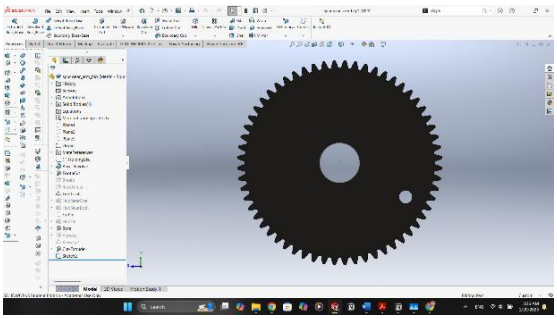


Figure 4.21 Driving gear

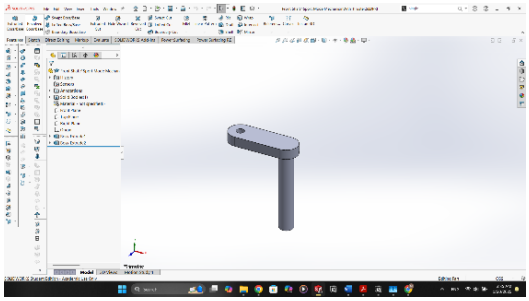


Figure 4.23 The front shaft

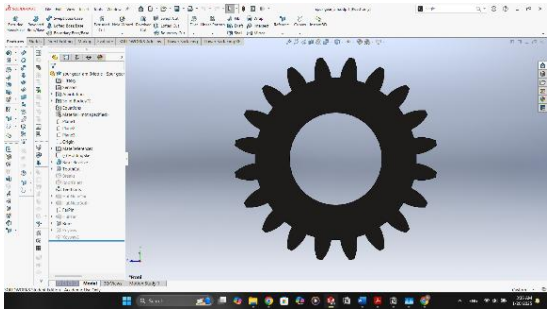


Figure 4.24 Driven gear

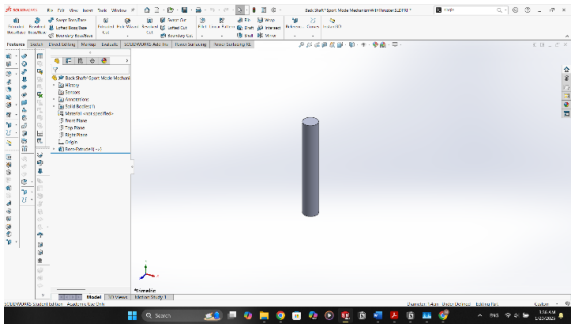


Figure 4.25 Rear shaft

Finally, the thrusters' housing used in the mechanism differs from the vertical thrusters to match its required orientation.

- Horizontal thrusters

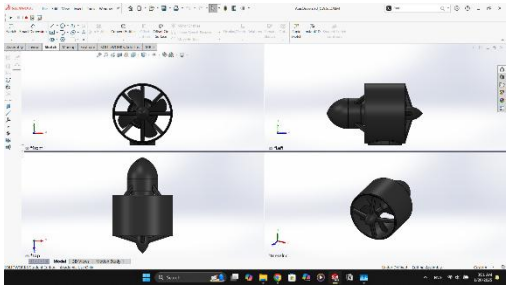


Figure 4.26 Horizontal thruster housing

- Vertical thrusters

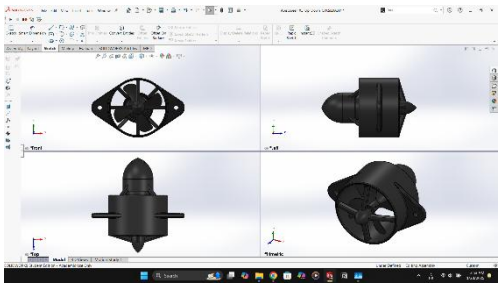


Figure 4.27 Vertical thruster housing

CHAPTER 5: Cost Estimation

item	Type (Hardware/ Software/ Other	Specifications (brief description)	No. of Items	Unit Cost	Total Cost of Items	photo
1	Thruster and its Motor Driver	T200+ESC	6	2,417	16000	 
2	Microcontroller	Arduino Mega	1	1450	1450	
3	Arduino Shield	USB Host Shield	1	500	500	

4	Microcomputer	Raspberry Pi 4 model B 8GB	1	6400	6400	
5	Camera	Surveillance Camera	1	1000	1000	
6	IMU Sensor	MPU6050	1	150	150	
7	Magnetometer	HMC5883L	1	350	350	
8	pressure Sensor	MS5540	1	700	700	
9	Current and Voltage Sensor	MAX471	1	165	165	

10	Joystick	Logitech Extreme 3D Pro Hand Grip Joystick	1	1800	1800	
11	Temperature Sensor	DS18B20	1	80	80	
12	Battery	ACE TATTU LiPo 6S 35C (20000mAh)	1	7000	7000	
13	Batteries	Lithium batteries	3	95	285	
14	Buck Converter	LM2596	1	55	55	

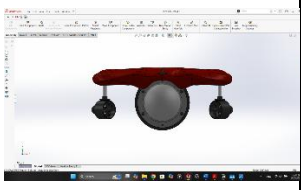
15	Light	Lumen Subsea Light for ROV/AUV	1	4000	4000	
16	Mechanical Components	Outer body, Core, Components Fixations, Mechanisms, joints, bearings and Resin	1	15000	15000	
Total Project Cost			**54,935**			

Table 5.1 Cost Estimation Table

CHAPTER 6: Future work

As part of our commitment to advancing underwater exploration and addressing critical marine challenges, our future work focuses on expanding the capabilities of our ROV through the following initiatives:

6.1 CORAL DETECTION AND DISEASE IDENTIFICATION:

We aim to enhance our dataset significantly to improve the ROV's ability to detect coral reefs and diagnose coral diseases. This involves collecting extensive, high-quality underwater imagery and annotations to train advanced machine learning models. By doing so, we will enable accurate monitoring of coral health, contributing to global efforts to preserve these vital ecosystems

6.2 ENHANCED INSULATION METHODS:

Improving the insulation of the ROV is essential for ensuring its functionality in extreme underwater conditions. Future work will involve exploring and implementing advanced insulation techniques to protect sensitive components from pressure, temperature variations, and water ingress, thereby increasing the ROV's reliability and operational depth.

6.3 ROS SYSTEM INTEGRATION (NAVIGATION AND 3D MODELING):

We plan to further integrate the Robot Operating System (ROS) to enhance the ROV's autonomy. This includes enabling the ROV to navigate and explore previously unknown underwater environments. By leveraging ROS for 3D mapping and modeling, the ROV will be capable of creating detailed representations of its surroundings, facilitating tasks such as infrastructure inspection and ecological studies.

6.4 MANIPULATOR INTEGRATION (GRIPPERS):

To expand the ROV's functionality, we will integrate a manipulator system designed for object handling. This addition will allow the ROV to grasp and transport objects between underwater environments and the surface. The manipulator will be designed to handle a variety of tasks, such as retrieving debris, collecting samples, or delivering tools to underwater locations.

6.5 IMPROVED PID CONTROL:

Implement Cascade PID Control for better system stability and performance by nesting multiple PID loops for interconnected control variables. For example:

- Inner loop for motor control and outer loop for positional control.
- Fine-tune tune the control system to handle varying water conditions and external disturbances dynamically.

By addressing these areas, our future work aims to enhance the versatility and effectiveness of our ROV, ensuring its capability to tackle diverse challenges in marine exploration, research, and industry. These advancements will contribute significantly to the development of innovative solutions for underwater operations and environmental conservation.

CHAPTER 7: Conclusion

The project on the design and implementation of a **Remotely Operated Underwater Vehicle (ROV)** exemplifies a multidisciplinary approach, combining advanced mechanical, electrical, and software engineering techniques to achieve a reliable and efficient underwater system. The project integrates innovative technologies and components, each meticulously selected and implemented to enhance the ROV's functionality, adaptability, and operational performance.

7.1 PROJECT DESCRIPTION AND SCOPE

The project aims to develop a versatile ROV capable of performing a variety of underwater tasks, including environmental monitoring, marine research, and object detection like fish, person, waste and etc..

The ROV is designed to operate effectively in challenging underwater environments, equipped with advanced hardware and software systems that ensure reliable performance and adaptability to various operational scenarios. This comprehensive approach not only addresses the technical challenges associated with underwater navigation but also emphasizes the importance of safety and efficiency in design.

7.2 HARDWARE DESIGN:

The hardware design phase incorporates essential components that work in conjunction with the software:

7.2.1 Microcontrollers (Arduino Mega and Raspberry Pi 4):

The Arduino Mega serves as the primary control unit, managing sensor data and actuator commands, while the Raspberry Pi 4 is utilized for advanced processing tasks, such as image analysis and video streaming. This dual-microcontroller approach allows for efficient data handling and real-time processing, enabling the ROV to perform complex operations underwater.

7.2.2 Sensors (IMU, Pressure, Temperature, Current):

A suite of sensors is integrated to provide critical environmental data. The IMU tracks orientation and motion, the pressure sensor measures depth, the temperature sensor monitors environmental conditions, and the current sensor ensures safe power consumption levels. The software interfaces with these sensors to continuously collect and process data, allowing for real-time adjustments to the ROV's operations.

7.2.3 Actuators (Thrusters and ESCs):

The ROV employs multiple thrusters controlled by electronic speed controllers (ESCs) to achieve precise movement in all six degrees of freedom. This configuration allows for agile navigation and stability in underwater environments, ensuring that the ROV can maneuver effectively during various tasks.

7.3 SOFTWARE ARCHITECTURE

The software architecture is a cornerstone of the ROV's functionality, designed to facilitate real-time control and data processing:

7.3.1 Control Algorithms (PID Control):

The implementation of Proportional-Integral-Derivative (PID) control algorithms is crucial for maintaining stability and precision in navigation. The software continuously calculates the error between the desired and actual states, adjusting the thruster outputs accordingly to ensure accurate depth and orientation control. This technique enhances the ROV's responsiveness to environmental changes and user commands.

7.3.2 Data Acquisition and Processing:

The software is responsible for acquiring data from various sensors, filtering and processing this information to derive actionable insights. For instance, the integration of complementary filtering techniques allows the ROV to combine data from the IMU and magnetometer, providing a more accurate estimation of orientation and heading.

7.3.3 User Interface and Control:

The software includes a user-friendly interface that allows operators to control the ROV via joystick inputs. The joystick data is processed to translate user commands into specific movements, enabling intuitive navigation. The software also implements safety mechanisms to monitor current and depth, automatically shutting down thrusters if predefined limits are exceeded, thereby enhancing operational safety.

7.3.4 Model Detection:

The software also incorporates advanced object detection capabilities using the YOLOv8 model, enabling the ROV to identify and classify underwater entities in real-time. This functionality enhances the ROV's operational efficiency, allowing for targeted data collection and analysis during underwater missions.

7.4 MECHANICAL DESIGN

The mechanical design focuses on creating a robust and efficient structure:

7.4.1 Frame and Pressure Canisters:

The frame is constructed from acrylic for its strength and lightweight properties, while pressure canisters protect sensitive electronics from underwater pressure. This design technique ensures durability and functionality in harsh marine environments.

7.4.2 Fairing Design:

A streamlined fairing is designed to reduce drag and improve hydrodynamic efficiency. This technique is crucial for enhancing the ROV's speed and maneuverability, allowing it to navigate effectively through water.

7.4.3 Sport Mode Mechanism:

This project showcases the successful design of a synchronized steering mechanism for front and rear thrusters, powered by a single stepper motor. By combining an offset crank-slider mechanism with a gear system, the design achieves precise and efficient motion control.

An innovative sport mode mechanism is developed to enhance the ROV's agility.

7.4.3.1 Crank-Slider Mechanism with Gear Ratio System:

This mechanical system synchronizes the rotation of front and rear thrusters, allowing for precise control of thrust direction. The use of a single stepper motor to drive both thrusters simplifies the control system while ensuring consistent performance. This technique is vital for achieving high-speed travel and complex maneuvers.

References

[1] Christ, R. D., & Wernli, R. L., Sr. (2014). *The ROV manual: A user guide for remotely operated vehicles* (2nd ed.). Butterworth-Heinemann.

[2] Moore, S. W., Bohm, H., & Jensen, V. (2010). *Underwater robotics: Science, design & fabrication*. Marine Advanced Technology Education Center.

[3] Pagà, X., & Rodríguez Conde, C. (2019). Design and implementation of a remotely operated vehicle for marine exploration and archaeological research. *Journal of Marine Science and Engineering, 7*(5), 145.

[4] Rady, M., Aly, H., & Elshimy, H. (2020). Design and implementation of a six-degrees-of-freedom underwater remotely operated vehicle. *International Journal of Advanced Robotic Systems, 17*(3), 1–12.

[5] Luo, Z., Xiang, X., & Zhang, Q. (2018). Autopilot system of remotely operated vehicle based on Ardupilot. *IEEE Access, 6*, 61205–61213.

[6] Thone, S. (2009). *The basis ROV control system wiring manual*. CreateSpace Independent Publishing Platform.

[7] Rasheed, A. F., Hagem, R. M., & Khidhir, A. S. M. (2021). Design and implementation of an interactive embedded system as a low-cost remotely operated vehicle for underwater applications. *Journal of Robotics and Control, 2*(4), 234–241.

[8] Gul, K. M., Kaya, C., Bektas, A., & Bingul, Z. (2020). Design and control of an unmanned underwater vehicle. *Turkish Journal of Electrical Engineering and Computer Sciences, 28*(5), 2745–2759.

[9] de Assis, F. H., Takase, F. K., Maruyama, N., & Miyagi, P. E. (2017). Developing an ROV software control architecture: A formal specification approach. *IFAC-PapersOnLine, 50*(1), 13607–13612.

[10] Zhang, Y., Yin, B., Jiao, H., Zhang, J., & Yan, F. (2019). Model establishment and parameter identification of remotely operated vehicle (ROV). *Journal of Marine Science and Technology, 27*(4), 1–10.

[11] Novara, C., & Pettinelli, A. (2018). Modeling and control of remotely operated underwater vehicles. *Control Engineering Practice, 74*, 1–12.

[12] Chin, C. S., Lau, V. W. S., Low, E., & Seet, G. (2017). Design of thruster's configuration and thrust allocation control for a remotely operated vehicle. *Ocean Engineering, 145*, 234–246.

[13] Ghilezan, A., & Hnatiuc, M. (2020). The ROV communication and control. *Journal of Marine Technology and Environment, 2*(1), 45–52.

[14] Homebuilt ROVs. (n.d.). ROV forum: Discussion on ROV design and construction. Retrieved October 10, 2023, from. https://www.homebuiltrovs.com/rovforum/viewtopic.php?f=18&t=836&start=0

[15] Tech Monkey Business. (n.d.). Open source ROV: Building your own remotely operated vehicle. Retrieved October 10, 2023, from. https://www.techmonkeybusiness.com/articles/OpenSource_ROV.html

[16] Obtain a data sheet and information for the used component.

https://www.alldatasheet.com/

https://octopart.com/

https://www.digikey.com/

[17] Obtain a data sheet and pins of Arduino.

https://www.arduino.cc/

[18] Obtain a data sheet and operation information about MPU-6050.

http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Components/General%20IC/PS-MPU-6000A.pdf

[19] Obtain a data sheet and operation information of water depth and pressure sensor MS5540C.

https://web.archive.org/web/20170110161401/http://www.fut-electronics.com/wp-content/uploads/2015/10/water_depth_sensor_MS5540C_Arduino_tutorial.pdf

[20] Obtain an operation information of Current sensor ACS712.

https://cdn.shopify.com/s/files/1/0672/9409/files/acs712-current-sensor-datasheet.pdf?816

[21] Obtain a data sheet and operation information of Raspberry Pi 4 Model B.

https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

[23] Model Establishment and Parameter Identification of Remotely Operated Vehicle (ROV).

[24] Underwater Robotics Science, Design & Fabrication by Steven W. Moore, Harry Bohm, and Vickie Jensen.

[25] Structural design evaluation for Underwater Remotely Operated Vehicle (ROV), case study: Madura Straits.