RMIT UNIVERSITY

School of Engineering

FINAL REPORT

Advanced Mechatronics System Design – MANU2451

Project Title: **Development of an Autonomous Drone for Aerial and Deep-Sea Operations**

Team members:

Student Number	Student Name	Tasks
s4061315	Sidharth Sreedas Nambiar	Drone path simulation, Reference, Results and Discussions
s4137396	Dharson Raju Mariappan	Drone Dive simulation, Initial and Final Design,
s4145582	Mohan Senthil	Motor Simulation, Literature review, conclusion
s4146093	Raghul Chinnathambi	Drone Path Simulation, Project Timeline, Budget
s4153605	Wanninayaka Mudiyanselage Janith Bandara Wanninayaka	Underground Rover Simulation, Proposed Design, Introduction

Submission Date: 08/06/25

Contents

1 Introduction

The planning of a dual-domain drone capable of operating in air and underwater represents a major advancement in the field of vehicles. Currently, drones operate in a single operating domain (air or underwater), limiting missions requiring cross-domain operations. A drone that would be able to fly or dive or complete a series of missions using a combination of both water and air modes empowers displacing the traditional cross-domain roles of a multitude of devices in one device. As would be the case for a myriad of applications including but not limited to oceanography-related research, inspection/repair of underwater or shoreline infrastructure after disasters, defence operations such a hybrid device represents a substantial improvement in mapping of depth into an operational theatre. Most of the project focuses on three unique challenges: propulsion system adaptation; pressure resistance; and waterproofing of the air and aquatic operating area, while creating an efficient transition from one environment to another while allowing for optimal operational efficiency. To define operational acceptance of remained functionality under a myriad of separate domains including deep sea event driven domains including reliability, an autonomous drone requires endless amounts of unique design elements, materials, designs, AI navigation, and integrated electronic systems within a tight system structure. Possibly the most unique focus on innovating by combining the speed of UAVs with the energy sustainability offered by ROVs, the realized drone will unleash the ability to do real-time data collection in remote locations or locations with potentially dangerous events, limiting risks and operational costs by eliminating humans from the operational picture at the very least until the final data acquisition and visual analysis. The following sections detail the design, system innovation, and prospective utilization of the dual-domain drone and highlight the prospect of documenting and surveying in deep-sea and air.

Literature Review

DEVELOPMENT OF A HYBRID AERIAL-UNDERWATER VEHICLE

Liang et al. (2020) introduce a hybrid UAV/UV able to provide seamless transition between air and water. The design combines a special quadrotor configuration with waterproofing and the ability to adjust buoyancy with such precision that the robot can be used in both water and air. A dynamic model is developed for motion in both domains, despite the large disparity in forces. Air and water propellers are altered for them. A dual-mode PID controller is designed to stabilize both flying and undersea navigation. Simulation and experimental results demonstrate the effectiveness of the proposed model and control schemes. The results show that the medium exchange and motion could be accomplished in both domains. The paper focuses on the issues of cross-medium propulsion and structural integrity. It provides a comprehensive treatment of hybrid vehicle systems, covering modelling, control design, and testing. This study lays a solid foundation for the development of underwater-aerial drones.

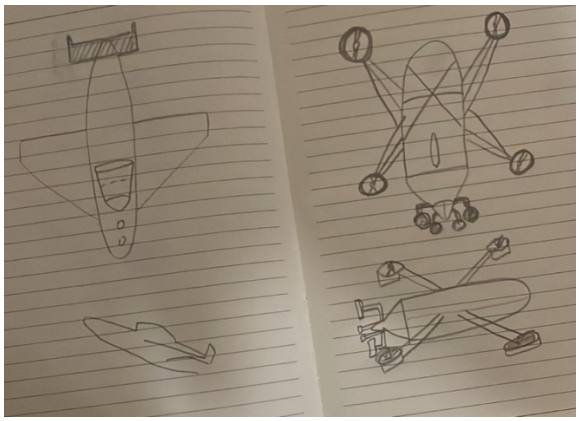
STUDY ON TRANS-MEDIA HYDRODYNAMICS OF A CYLINDRICAL HYBRID VEHICLE

Zhang et al. (2022) study the trans-media hydrodynamics of cylindrical hybrid flying underwater vehicle. The work is looking specifically at forces during liftoff and landing in the transition area between the air and water, an obstacle for hybrid drone operation. Experimental tests were performed in a test facility that measured drag and lift (in a range of velocities and immersion angles) in a controlled environment using a custom test platform. The analyses indicate considerable differences in hydrodynamic coefficients when the vehicle was oriented and moving at entry. The stability and motion are studied including the flow separation and splash effects. Their results reveal that perfect transitions should have both the optimal body geometry and the best entry angle. The paper constitutes important experimental evidence for improving the design of vehicles to reduce energy wastage and to stay in control. It also mentions the compromise of aerodynamic and hydrodynamic performance. In total, this study contributes to the knowledge of medium transition physics and facilitates the future development of hybrid drones.

MODELLIG AND CONTROL OF UNMANNED AERIAL/UNDERWATER VEHICLES

Yang et al. (2019) develop an approach to model and control unmanned vehicles that can move through the air and water. The paper deals with the discontinuities and nonlinearities associated to the interface between two extremely dissimilar media. A hybrid system architecture is presented to coordinate the discrete transition between dynamic models as the vehicle transitions between the air and water. The researchers formulate separate equations of motion in the aerial and underwater regimes that are integrated within a state machine control structure. This enables the vehicle to adjust control strategy in real time in response to operational conditions. The simulation results indicate the superiority of the proposed controller/observers in stability and trajectory tracking for both domains. The paper also discusses sensing and decision-making challenges during medium transition. The hybrid-system framework is found to be effective in facilitating the design of control laws for multimodal tasks. In general, this study provides a useful basis for the autonomous control of aerial-underwater hybrid drones.

2 Initial Design



Initial Design

The design of the autonomous drone is aimed at efficient operation in both aerial and underwater environments. The overall concept is inspired by birds, specifically cormorants that fly and dive to catch fish. The design of the robot drone is being piloted with a dual-fin tail structure designed to manipulate direction and a fuselage that houses all control systems and sensors. The next design will now be an evolved design that will enable vertical take-off and landing (VTOL) as well as underwater mobility; really building off the concept into a new design drone design using an X-configuration giving 4 long extending arms and ducted rotors or propellers at the end. The drone used the rotors for both lift in air and thrust to travel underwater and used the secondary arms - with potential for protective seal and/or retraction when travelling submerged. The tail of the drone will include re-usable modular fittings, possibly propulsion or ballast - to add increase or control stability and mobility in either environment. The essential qualities of the design will still express modularity, resistance to damage, and level of autonomy, not only to improve the drone's ability to travel in the air or underwater but also balance function and control.

3 Final Design

This is a hybrid aerial–underwater drone that autonomously makes the transition from flight to submerging. The drone has a cylindrical submarine that forms part of its design and involves low levels of human interaction.

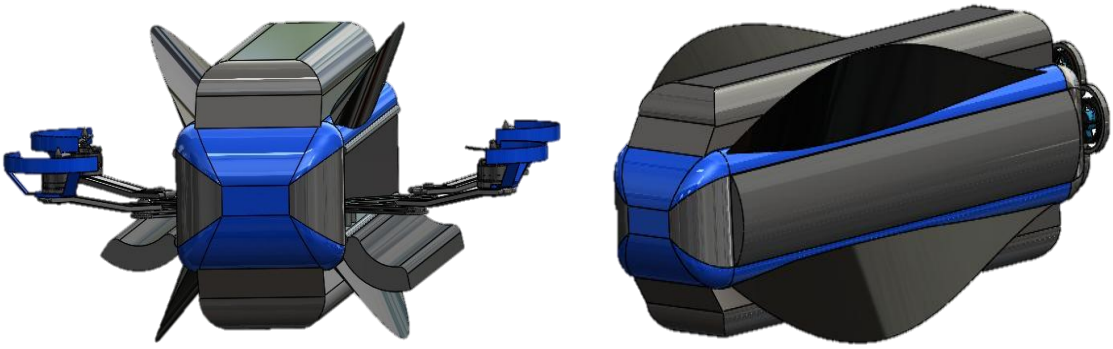


Figure 1. Front And Sideview of the Drone

As seen from Figure 1 and 2, the submarine is cylindrically shaped with the dimension of 1590mm, 570mm and a height of 630mm. With four stabilizing fins mounted at intervals of 45° along the body. The retractable drone arm is kept in a central opening that opens by curling open a hatch through the movement of a hinge mechanism powered by a stepper motor. The wings are foldable and are actuated by servo motors, and there is convenient switching between air and underwater modes.

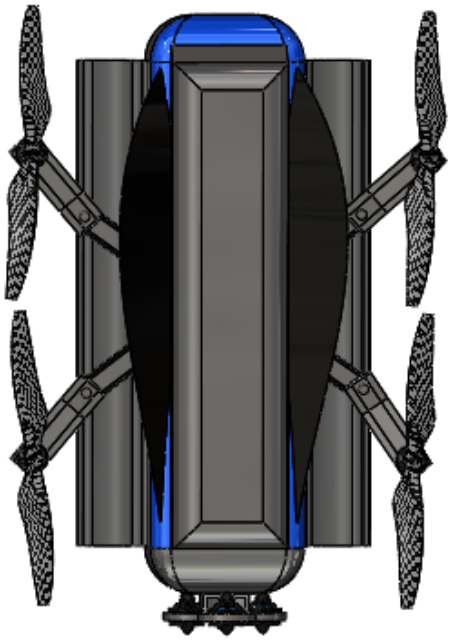


Figure 2. Top view of the Drone

The mechanical framework of the hybrid drone-submarine includes a complex arched hatch door mechanism (Figure 3), which hermetically closes the drone before its underwater launch. The 1258 mm long, 130 mm wide, and 50 mm thick hatch is curved at a 20 mm radius to precisely match the cylindrical drone body. Attached to a hinge powered by a stepper motor, the hatch is smoothly swung into place and forms a watertight seal to protect internal components during underwater operations.

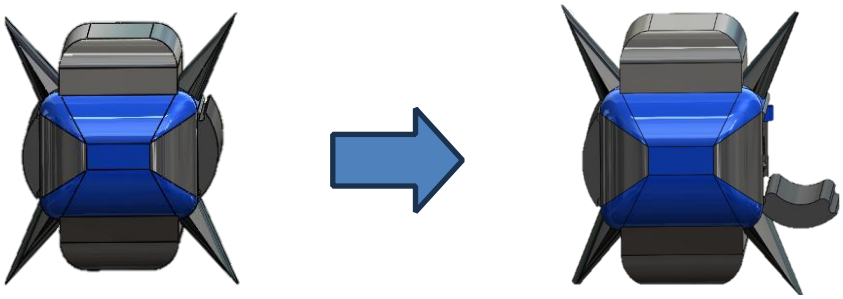


Figure 3. Arched Hatch Door.

One of the most important transition stages is when the drone switches from aerial to underwater mode. Once the drone reaches the destination and the onboard Inertial Measurement Unit (IMU) detects a stable diving attitude, the vehicle pitches forward to initiate the conversion. The retractable wings are closed by servo motors to reduce drag and allow streamlined diving. The wings are sealed by a hatch curved and closed for underwater protection once more and this is opened by a material-driven hinge. The system uses the employment of Nitinol, a shape-memory alloy, to automatically open and close the wing doors through heat, which introduces reliability and autonomy.

In addition, the drone's rotors are attached to extendable arms that fold out from the body through two stepper motors one for each side. The motors work through a gear system that provides synchronized deployment and compact storage, once more maximizing space in the drone design.

To enable aerial lift, the drone has larger propellers that increase take-off stability and efficiency. The propellers feature a split propeller concept where the blades fold into two parts for better storage when in underwater mode, enabling spatial efficiency.

For diving, the system depends on a ballast tank at the bottom of the drone. The intake of water into the tank is regulated by solenoid valves added in front of the drone as it faces the sea for diving the water will be easily flowed inside, which provide additional weight to the drone to enable it to descend. Buoyancy at desired depths is ensured by modulating the valves, allowing hovering or spinning stably beneath the water. This entire transition—from flight to dive—is autonomous, combining sensor feedback, mechanical actuation, and closed-loop control to switch modes without any external intervention.

3.1 Mechanical structure

DRONE MOTION PATH SIMULATION

To represent the real-world behaviour of the drone, a 3D path was created using sinusoidal horizontal movement and a staged descent profile in the vertical direction. The trajectory is divided into three key phases: steady flight, transition at the water surface, and underwater submersion. A time vector of 60 seconds was used, allowing for clear observation of the drone's dynamic profile across all stages.

The simulation begins with the drone maintaining a consistent altitude at approximately 3 meters, demonstrating stable flight. It then gradually descends to the water surface and subsequently enters a downward dive to a depth of 6 meters. This trajectory was visualised using a MATLAB animation script with Euler angle-based body orientation control to reflect realistic pitch, roll, and yaw movements.

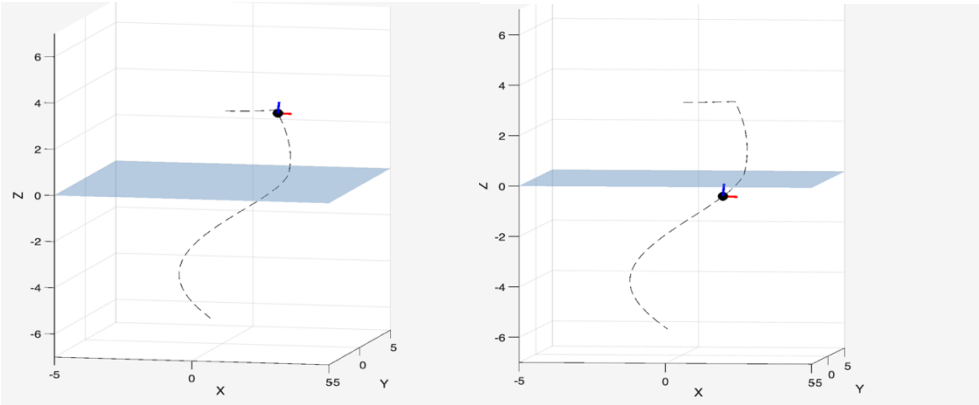


Figure 4: Hybrid drone simulation showing steady aerial flight and transitioning at the air–water interface and drone submerged and navigating underwater.

The animation environment was enhanced with a translucent light blue plane at the water surface to visually separate aerial and underwater phases. This feature added depth and realism to the simulation, helping better illustrate the transition point.

Position vs Time Analysis

To further understand the drone's motion profile, its X, Y, and Z coordinates were plotted against time. These plots give insight into the drone's movement in each dimension during the simulation.

- The **X Position vs Time** plot shows a sinusoidal pattern, indicating horizontal oscillation as the drone progresses forward.
- The **Y Position vs Time** plot captures lateral drift, helping simulate crosswind or underwater current effects.

- The **Z Position vs Time** plot is the most critical for understanding transition, it confirms stable height during aerial flight, a gradual descent, and a continued linear dive underwater.

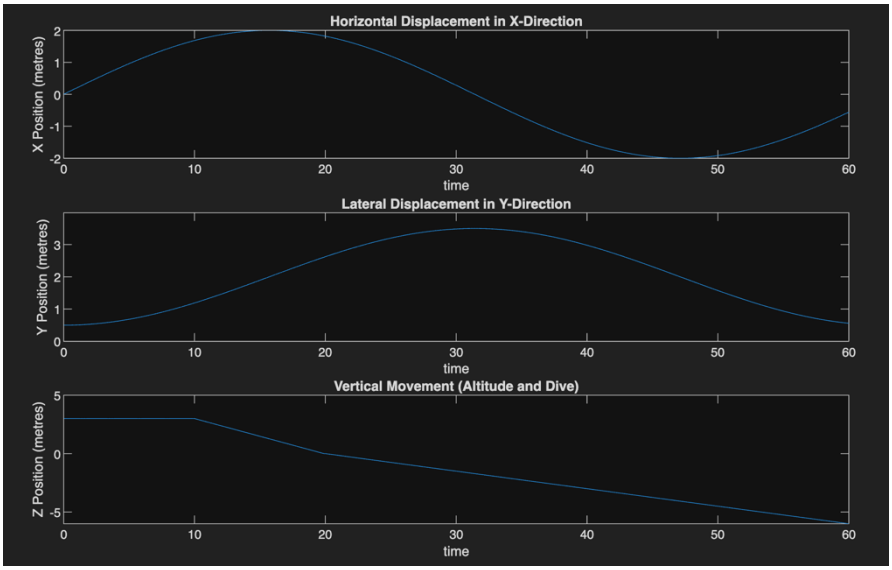


Figure 5: Drone position over time.

These trends validate the smooth motion planning and effective descent control that the simulation aimed to replicate. The changes in Z clearly demonstrate the multi-environment traversal of the drone, supporting the expected operational design.

Simulink-Based Control Model

The simulation was also supported by a simplified Simulink block model, which provided a control logic flow for processing input parameters such as thrust, position, and feedback loops. Although simplified for this stage of development, the control diagram reflects essential systems such as:

- **PID-based attitude control** for maintaining balance during transition phases.
- **Mode switching logic** to handle change in dynamics between air and underwater.
- **Motion integration blocks** for tracking position and orientation over time.

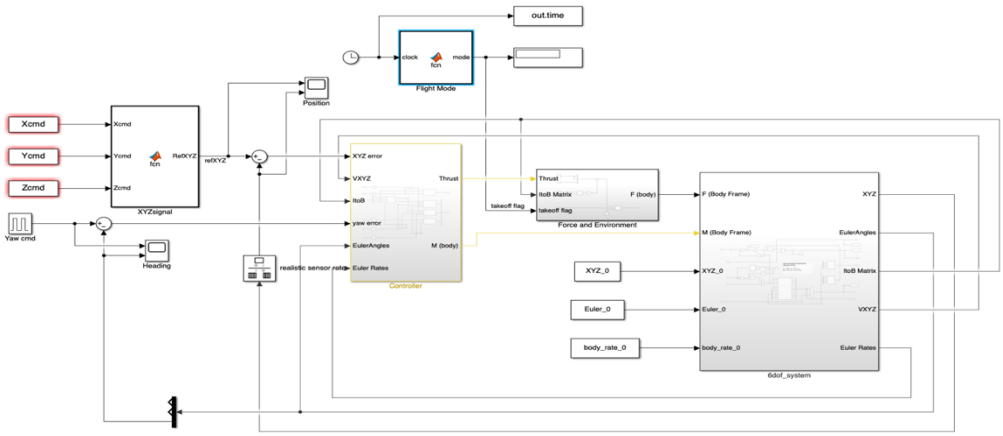


Figure 6: Simulink block diagram of drone control architecture.

DRONE DIVE SIMULATION

This simulation models the behaviour of a drone designed to transition from flight to underwater operation in a controlled and efficient manner. The drone begins its journey flying at a high altitude and follows a downward, curved path toward a specific dive coordinate.

As it approaches about **10 meters above sea level**, a key transition takes place:

- **Dive Preparation Initiates:**
  At this altitude, the drone starts preparing for the dive. It does this by **turning off its motors** and **retracting its wings**. This helps reduce aerodynamic drag and allows the drone to adopt an optimized posture for entering the water smoothly.
- **Target Dive Coordinate:**
  The simulation marks this point (around **8.63 meters altitude**) as the drone's dive location — a point where all preparations are complete and it's ready to enter the water.
- **Sea Entry and Underwater Transition:**
  Once the drone crosses **sea level (0 meters)**:
  - **Ballast tanks activate** to help manage buoyancy and assist the drone in sinking smoothly.
  - **Underwater thrusters engage**, allowing it to begin active navigation beneath the water.
- **Retrieval Path:**

  It also has a return path that follows where it started it dive for easy retrieval and once it comes up to the surface it will upload the data collected below the sea

The graph provides a visual reference:

- The **black dashed line** shows the full path from sky to underwater.
- The **blue shaded region** represents the sea.
- The **black "X" marker** shows the dive point.
- A red label indicates the status: **"Reached Dive Coordinates - Preparing..."**

This simulation is useful for validating how well the drone manages its transition phases, ensuring minimal energy loss, stable dive posture, and readiness for underwater operation.

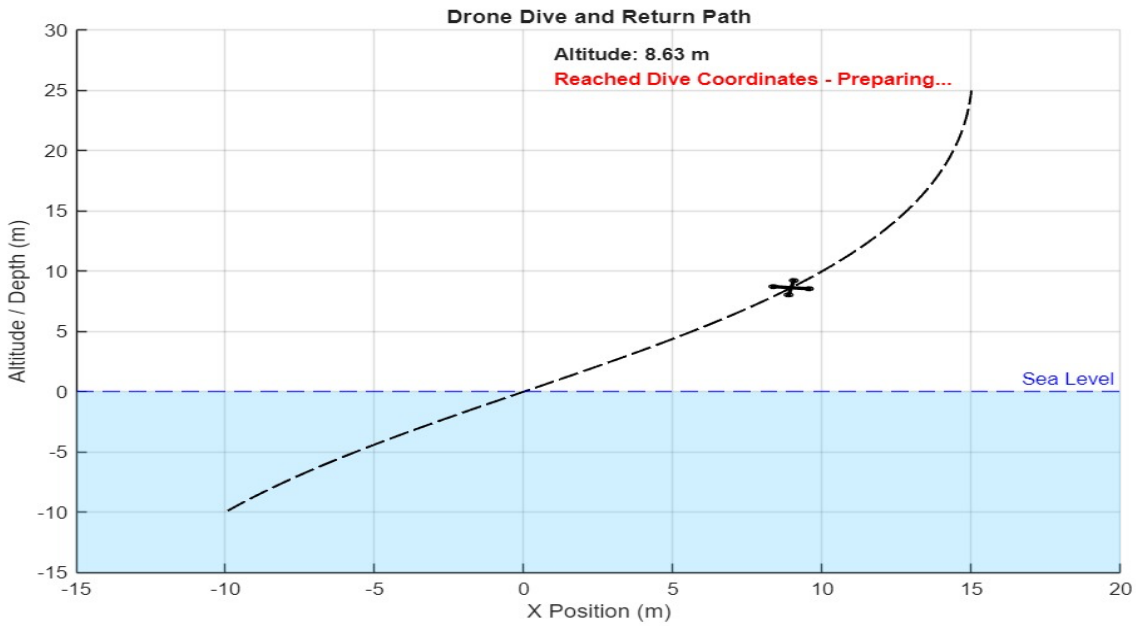
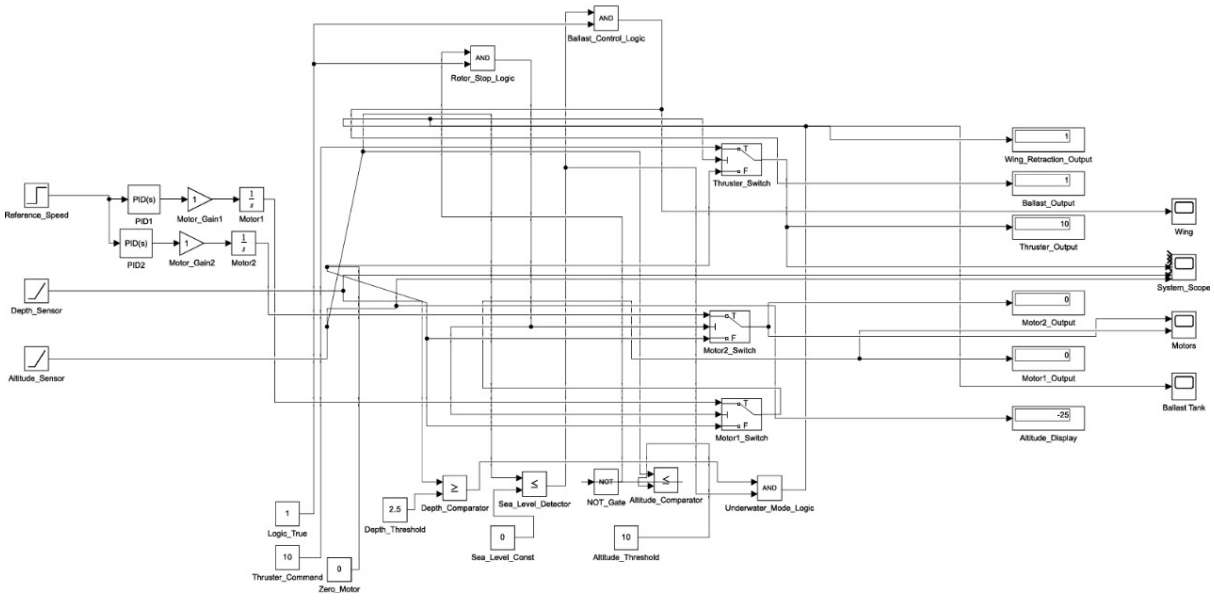


Figure 7: Drone dive and Return Path

System Working and Output Analysis



Process Flow Overview

This hybrid drone system is designed to transition seamlessly from aerial flight to underwater navigation using altitude and depth sensors, PID control, and logical triggers.

Key Stages in System Operation:

- 1. Aerial Phase:
    - Motors (Motor 1 and Motor 2) are active to maintain flight.

- The drone begins at a high altitude (~30–40 meters).

- 2. Altitude-Based Motor Shutdown:
  - When the altitude drops below 10 meters, the motors are deactivated.
  - A 10-second delay starts to ensure safe wing retraction.
- 3. Wing Retraction:
  - After 10 seconds post motor cutoff, the wing retraction logic activates.
  - This prepares the drone aerodynamically for underwater movement.
- 4. Sea Entry and Ballast Activation:
  - Upon reaching 0 meters altitude (sea level), the ballast system turns ON to aid the descent.
  - This ensures controlled diving instead of a free fall.
- 5. Underwater Mode and Thruster Activation:
  - As the depth exceeds 2.5 meters, the underwater thrusters are activated.
  - The system switches to underwater propulsion.

Scope Output Analysis

The simulation outputs were visualized using a multi-input scope block. Below is a breakdown of each signal:

Altitude Sensor(blue): Starts high from 30–40m and decreases linearly. Triggers logic at 10m and 0m.

System Behaviour Timeline

Initial Flight	0–30s	Motors ON, wings deployed, drone descends gradually.
Altitude Threshold	~30s	Altitude < 10m → Motors OFF, delay begins.
Wing Retraction	~40s	Wing retraction logic activates after delay.
Sea Entry	~50s	Altitude = 0 → Ballast ON.
Diving	50–70s	Drone submerges; depth increases gradually.
Underwater Navigation	>70s	Depth > 2.5m → Thrusters activate → underwater propulsion begins and then prerecorded path planning or exploration continues

Logical Control Summary

This section outlines the decision logic based on sensor readings:

- If altitude > 10m → Motors remain ON.
- If altitude ≤ 10m → Motors OFF, 10s timer for wing retraction starts.
- After timer → Wing retraction activated.
- If altitude = 0 → Ballast system activates.
- If depth ≥ 2.5m → Underwater thrusters activated.

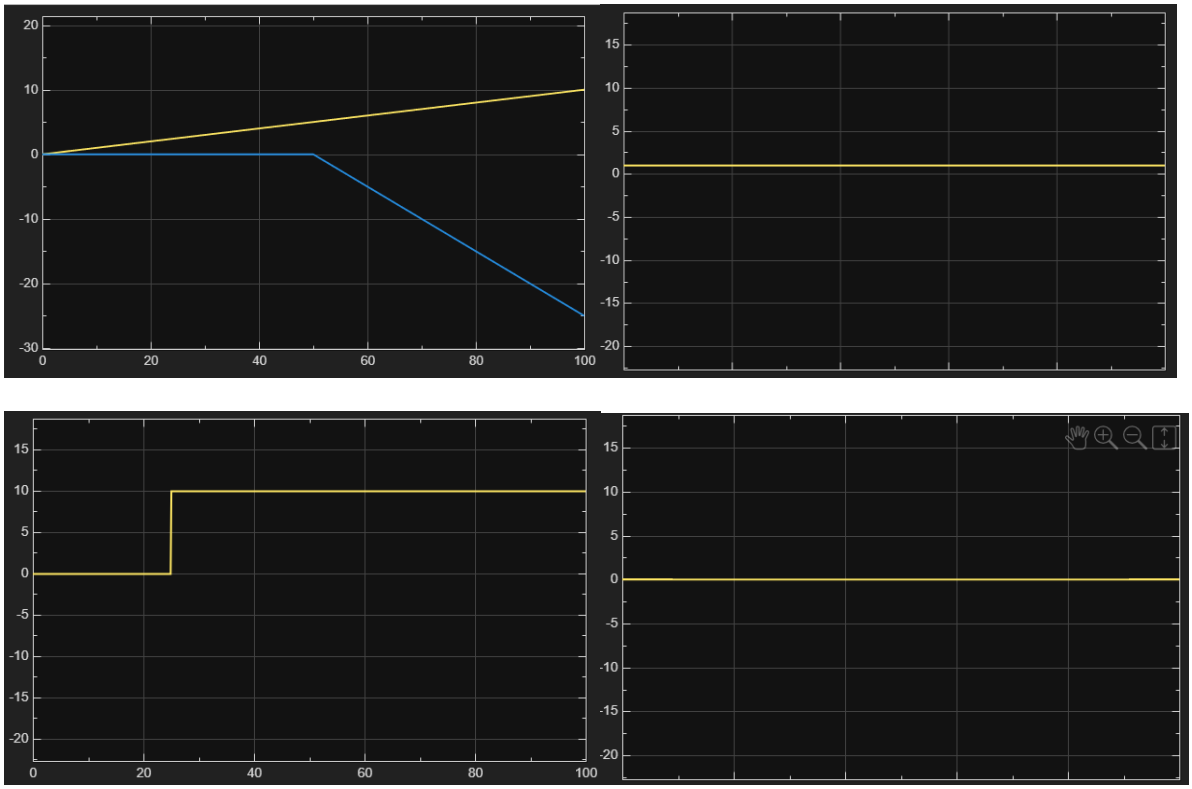


Figure 8: Depth sensor, Aerial Motors, Thruster Output, Ballast Control signal.

Depth Sensor – Gradually increases after sea entry.

Aerial Motors – Active until 10metre altitude, then switches off.

Thrusters Output – Thrusters activates after the drone enters the water & reaches a depth of > 2.5 metres.

Ballast Control signal – changes from 0 to 1 at sea level (0m).

Underwater Rover Simulation and Analysis

Simulation Overview

This simulation models an autonomous underwater rover navigating through a realistic 3D aquatic environment populated with obstacles. The model integrates physics-based vehicle dynamics, multi-loop PID control for navigation and depth, sensor simulation (IMU and sonar), and real-time obstacle avoidance. The purpose is to analyze and demonstrate the rover's capability for robust path-following and obstacle avoidance using integrated sensing and control.

Environment & Obstacles

Domain: 500 m × 500 m × 300 m

Obstacles: 20 randomly placed spheres, radii between 20–60 m, simulate terrain features and debris.
Water properties: density ($\rho = 1000$ kg/m³), gravity ($g = 9.81$ m/s²).

Obstacles are represented as spheres: Obstacle_i = {center: (x_i, y_i, z_i), radius: r_i}

Random placement ensures that each simulation run tests navigation robustness under varied conditions.

Rover Physical and Actuator Model

Dimensions: Length L = 1.59 m, Width W = 0.57 m, Height H = 0.63 m

Mass: Dry mass m_empty = 320 kg; variable ballast (0 to 260 kg) for depth control

Thrusters: 4x, max 294 N each, rear + configuration. Max total thrust = 1176 N

Ballast: Adjusted by a pump/solenoid, rate-limited (max 10 kg/s), used for depth and buoyancy control.

Vehicle volume:

$$V = L \times W \times H$$

Hydrodynamic drag:

$$F_{\text{drag}} = \frac{1}{2}\rho C_D A v^2 \cdot \text{sgn}(-v)$$

Buoyant force:

$$F_{\text{buoyancy}} = \rho V g$$

Weight:

$$F_{\text{weight}} = mg$$

Net vertical force:

$$F_z = F_{\text{buoyancy}} - F_{\text{weight}} + F_{\text{drag},z}$$

Vertical acceleration:

$$a_z = \frac{F_z}{m}$$

Sensor Simulation

IMU (Accelerometer and Gyro): Simulated IMU includes realistic bias and Gaussian noise. Accelerometer: Measures body-frame acceleration, including gravity, bias, and noise. Gyroscope: Measures yaw rate (ψ_dot) with bias and noise.

IMU body frame measurement:

$$\vec{a}_{\text{meas}} = \mathbf{R}_{wb}\vec{a}_{\text{world}} + \text{bias} + \mathcal{N}(0, \sigma^2)$$

Sonar: 360° rotating sonar (Ping360 style), simulated as a 120° sector sweep with 24 beams. For each beam, calculates minimum range to any obstacle in that direction.

Sonar beam minimum range:

$$r_b = \min_i(\text{distance to surface of obstacle } i \text{ along beam})$$

Navigation, Waypoint Tracking, and Control Logic

Waypoint System: Rover is assigned a sequence of 3D waypoints. At each time step, computes the vector to the current waypoint:

$$\vec{d}_{\text{wp}} = \vec{p}_{\text{wp}} - \vec{p}_{\text{rover}}$$

Moves to the next waypoint when within a specified tolerance (<7 m).

Obstacle Avoidance: Threat detection uses sonar and geometry to detect obstacles within a 60° arc and 30 m range ahead. Avoidance maneuver offsets heading by a set avoid angle.

Angle calculation for avoidance:

$$\theta = \arccos\left(\frac{\vec{h} \cdot \vec{v}_{\text{obs}}}{|\vec{h}|\,|\vec{v}_{\text{obs}}|}\right)$$

PID Control Architecture

Yaw (Heading) PID:

$$e_{\text{yaw}} = \psi_{\text{target}} - \psi$$

Yaw control law:

$$u_{\text{yaw}} = K_p e_{\text{yaw}} + K_i \int e_{\text{yaw}} dt + K_d \frac{de_{\text{yaw}}}{dt}$$

Forward Velocity (Surge) PID: Controls surge speed along the heading. Depth (Ballast) PID: Controls vertical speed via ballast, achieving target depth.

Actuator Allocation

Thrust split and allocation:

$$F_{\text{left}} = \frac{F_{\text{fwd}} - \tau_{\text{cmd}}/d}{2}$$

$$F_{\text{right}} = \frac{F_{\text{fwd}} + \tau_{\text{cmd}}/d}{2}$$

Each side's thrust is shared by two thrusters (cross configuration). Thrust outputs are clipped to max/min per thruster. Ballast update: At each time step, adjust ballast to match the commanded value, rate-limited by pump capability.

Simulation Step-by-Step Logic

1. Calculate vector to waypoint and desired velocity.
2. Sonar/geometry check for obstacles ahead. If detected, update target heading to avoid.
3. Update PID controllers for yaw, forward velocity, and depth (ballast), using most recent sensor and state data.
4. Allocate thruster outputs based on current control demands.
5. Apply hydrodynamic forces (drag, buoyancy, gravity).
6. Integrate motion equations for new position, velocity, yaw, and depth.
7. Simulate sensors (IMU, sonar) for the new state.
8. Store all states, actuator, and sensor data for post-simulation analysis.
9. Check if current waypoint is reached; if yes, proceed to the next.

Results and Graphs

Path Tracking and Obstacle Avoidance

The 3D path plot illustrates the rover navigating between waypoints and maneuvering around obstacles.

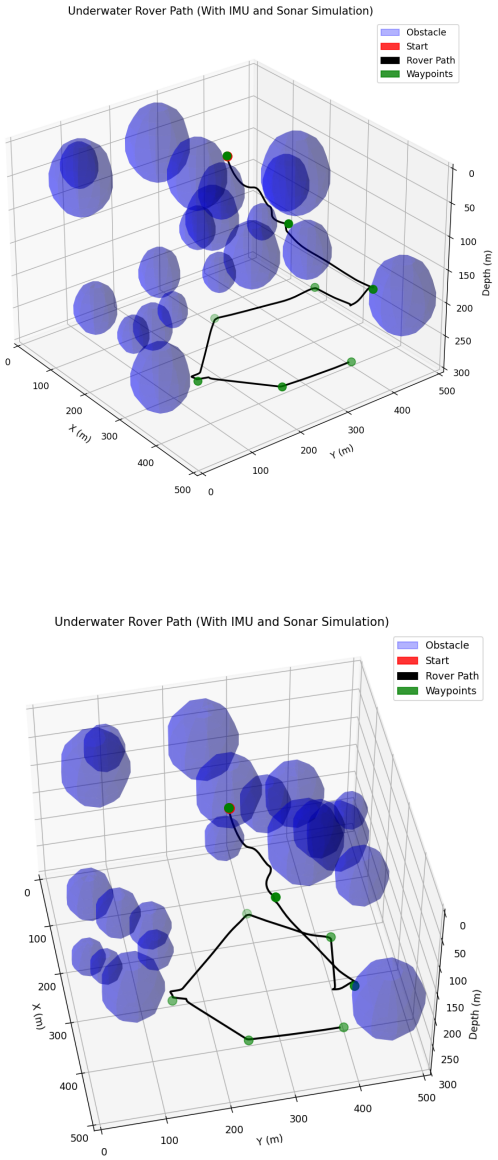


Figure 9: Underwater Rover Path

Thruster and Ballast Activity

- Individual thruster force plots show how each thruster is used for forward movement and turning.
- The ballast plot shows tank filling/emptying during descent and ascent.

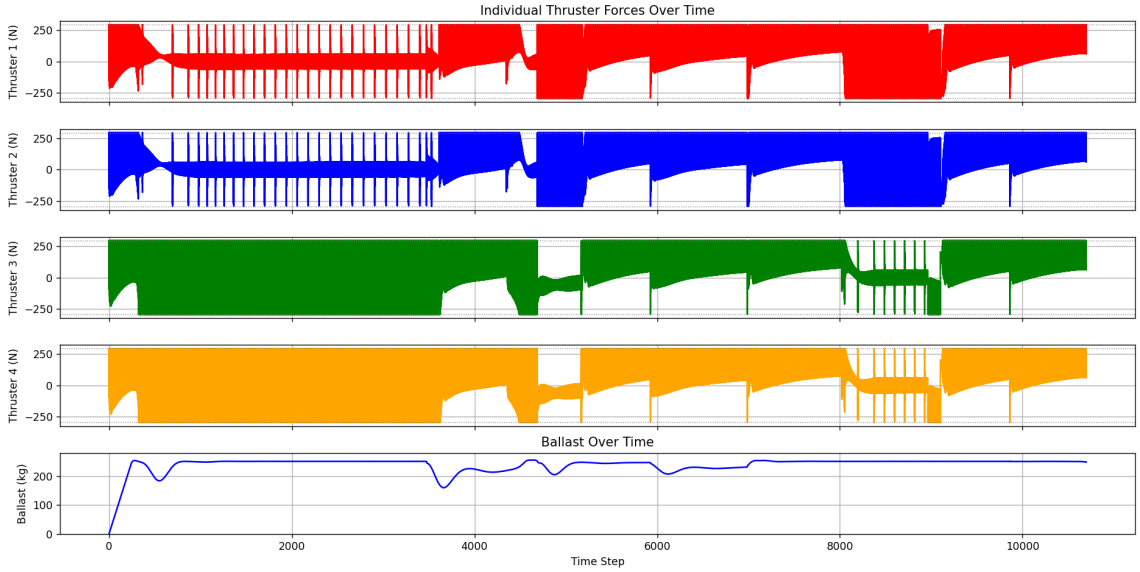


Figure 10: Individual Thruster Forces Over Time

IMU Data

Simulated accelerometer and gyro outputs show the expected sensor noise and bias, useful for testing sensor.

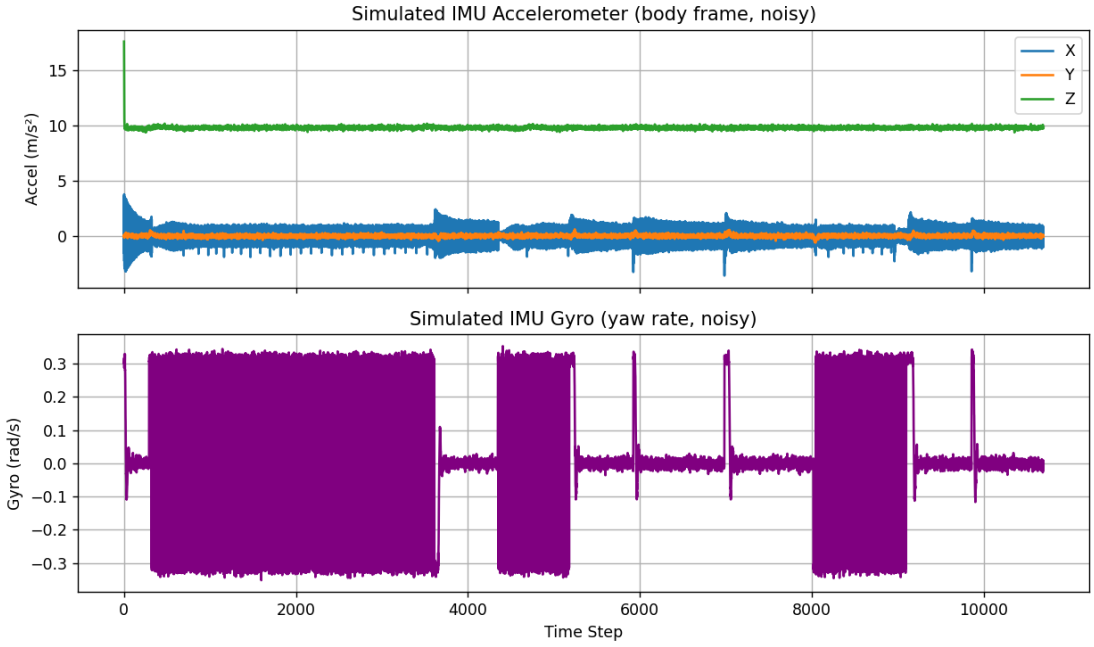


Figure 11: Simulated IMU Accelerometer 7 Gyro

Sonar Mapping

- The final sonar scan and the “sonar waterfall” visualize what the rover “saw” during its journey, indicating proximity to obstacles at every orientation and timestep.

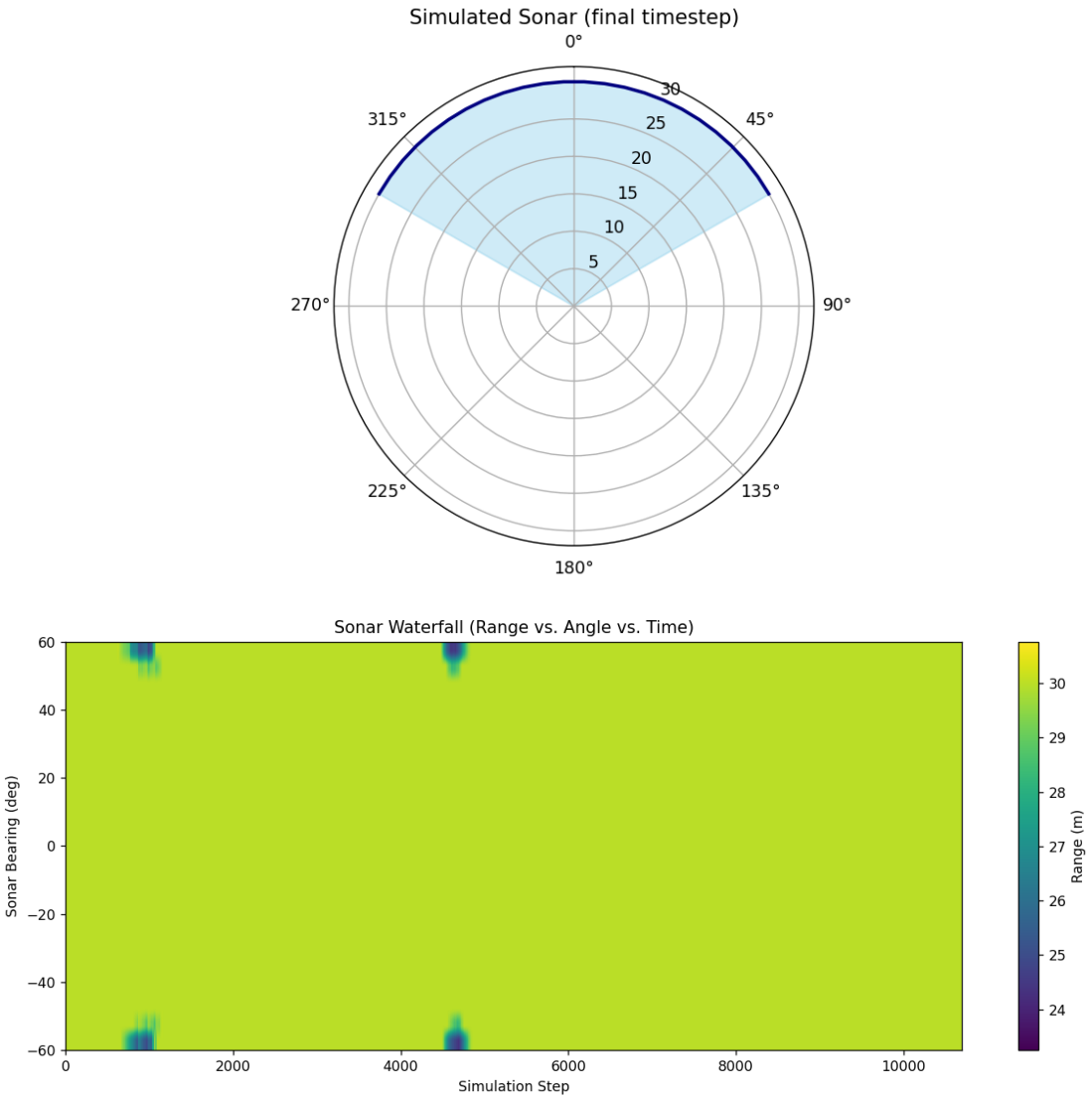


Figure 12: Simulated Sonar Waterfall and timestep.

Analysis

- **Navigation Performance:**
  The simulation demonstrates the rover's ability to reach assigned waypoints, avoid obstacles, and manage buoyancy via ballast control. The separation of thruster usage can be clearly seen in the force graphs.
- **Ballast Function:**
  The ballast logic ensures the rover only fills the tank to descend and empties it to ascend, as in real underwater vehicles. The system prevents the thrusters from being used inefficiently for vertical movement.
- **Sensor Realism:**
  IMU and sonar outputs include realistic noise, allowing for evaluation of navigation/filtering algorithms.

- **Obstacle Avoidance:**
  The rover successfully detects and manoeuvres around obstacles based on sonar simulation.

MOTOR SIMULATION

This simulation illustrates the dynamics of a Brushless DC (BLDC) motor using Hall-effect sensor feedback. The goal is to view the effects of the stator currents to control phase switching and to infer the electromagnetic torque based on the switching logic based on rotor position. The Hall sensors determining the angular position of the rotor are going to control the motor phases using a logic-based commutation approach.

Block Diagram:

The diagram represents a well-organized control system starting from a Hall-effect sensor input as shown in the fig 13. The NOT gate provides the inverse of the Hall signals, while the AND gates logically combine the Hall signals, both original and inverted, to implement the correct phase switching. Each AND gate output drives a "convert" block which simulates the signal conditioning and the application to the motor windings. The outputs of these blocks are then summed and visualized as the stator phase currents and electromagnetic torque, which are finally routed to the scope blocks for analysis.

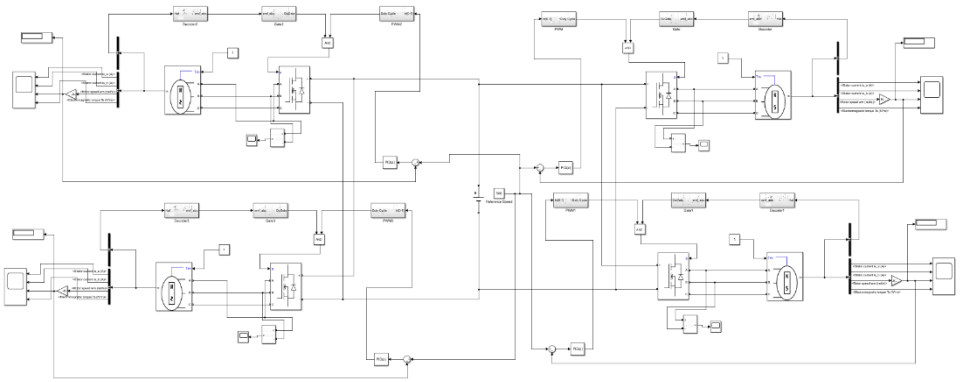


Figure 13 : Block diagram of BLDC motor

A Pulse Width Modulation (PWM) generation system (fig 14) creates a duty cycle that ranges from 0 to 1. The system takes two input signal types: a reference signal, which in this case is a sine wave, and a carrier signal, which is typically a high-frequency sawtooth waveform. The two signals are compared using a relational operator (greater-than block); this produces an output of a logic high (1) if the reference signal is greater than the carrier and logic low (0) otherwise. Therefore, the output control signal will produce a PWM signal with a duty cycle proportional to the value of the amplitudes of the reference signal. To constrain the duty cycle to remain within a valid range, the final output from the PWM monitor (the Duty Cycle signal) is clipped at the upper limit of 1 using a less-than block. The Duty Cycle signal is considered

a form of pulse waveform that can be routed directly to drive power devices like inverters or motor drivers. A PWM control signal permits users to modulate voltages and currents efficiently.

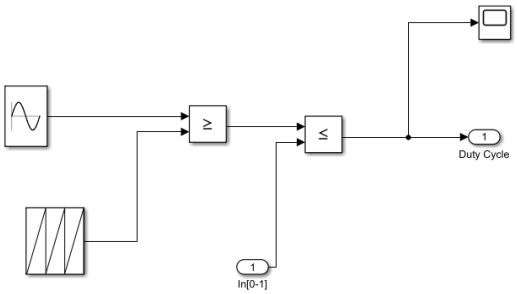


Figure 14 : PWM generator

This diagram shows a gating logic system (fig 15) for controlling the switching states of a power electronic converter or inverter based on back electromotive force (emf) signals. The input emf_abc represents values of three-phase back EMF (likely from a BLDC or PMSM motor). These signals are separated into each component with two devices of ordinal comparison (> 0 and < 0) for each phase which realize whether the each phase EMF is positive or negative at a moment in time. The outputs of these comparisons are logically grouped and transmitted through a bus structure to make the decisions about the appropriate gating signals. The gating signals are combined with a control signal named OuGate, which is some kind of enable or modulation signal (such as PWM or commutation control signal). The final output from this logic defines the switching pattern for the power electronic devices that drive the motor. The gating strategy is structured to turn on switches correlated with the rotor position and back EMF so that the motor is efficiently and properly commutated.

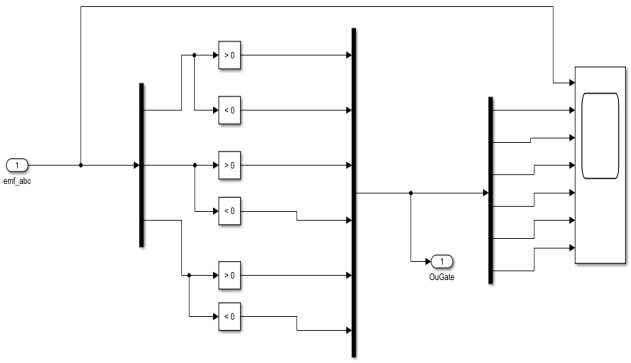


Figure 15 : Gate

The accompanying schematic diagram depicts a decoder system (fig 16) intended for control of a three-phase Brushless DC (BLDC) motor with Hall sensor feedback. The signal labelled Hall transmits rotor position information, which is fed into a NOT gate and processed to

produce both original and inverted logic levels. The level for each Hall signal is derived and demultiplexed into single Hall signals (ha, hb, hc and their inverses /ha, /hb, /hc). Once each combination of Hall inputs is provided as input to each AND gate, the logic circuit become a combinatorial structure that identifies valid commutation states and notifies the appropriate "convert" blocks when those states are true. The "convert" blocks encompass the logic to convert these binary states into motor driver switching outputs. The outputs from the "convert" blocks are combined to indicate which phase winding should be energized by the driver depending on the rotor position. The last part of the system consists of multiplexing and gating logic where the switching outputs are synchronized with the motor back electromotive force (emf_abc) to time inverter gate signals appropriately. This is necessary to ensure proper commutation of the stator windings to enable smooth rotation of the BLDC motor. The decoder is therefore a key component to converting rotor position data into control signals that are usable for the operational control of the motor.

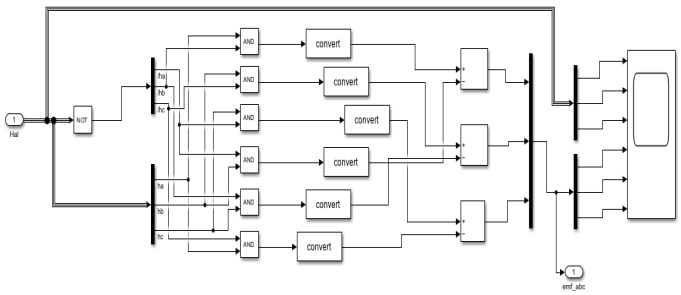


Figure 16 : Decorder

Interpretation of Results:

The graphs are extracted from the component's simulation data and show the phase currents (i_s_a, i_s_b), electromagnetic torque (Te) and rotor speed as they vary over time as shown in *fig* 17. The phase currents show the instantaneous values oscillating around zero over time, confirming the BLDC motor is commutating correctly. The spike shown in one of the graphs at the start of the simulation indicated the transient response of the motor to being energized initially; therefore, the currents and torque enter a steady state behind the kick before stabilizing. The electromagnetic torque rises sharply at first and then converges to a constant value, also indicating the motor almost immediately stabilizes at a solid operating region. This shows the logic-based control scheme is functioning properly along with the simulation, and the overall setup is functional.

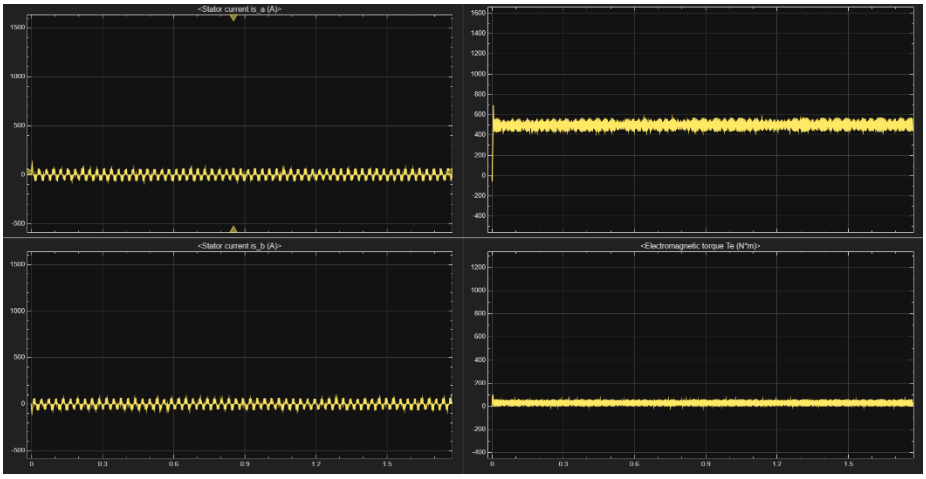


Figure 17: Phase currents, Electromagnetic torque (Te) and Rotor speed

3.2 Actuators

To enable mobility, environmental adaptation, and camera control, the drone uses a dual-propulsion architecture with eight motorized fans - four for flight and four for underwater movement — and supporting actuators for buoyancy and vision management.

Actuator	Model/Part	Purpose	Integration	Price
Aerial Propulsion Motors (x4)	T-Motor F80 Pro V2 2500KV	Provides lift and flight control in aerial mode.	Controlled by ESCs and flight controller. automatically disabled when submerged.	$65 each
Underwater Thrusters (x4)	Blue Robotics T200	Enables under-water mobility and orientation.	Controlled via PWM; coordinated with IMU and sonar data for 3D maneuverability.	$179 each
Ballast Control Pump	Blue Robotics Peristaltic Pump	Manages water intake to increase drone density and initiate descent.	Triggered during water entry; monitored through depth sensor feedback.	$45
Solenoid Valve	Uxcell DC 12V Normally Closed Valve	Releases ballast water to restore buoyancy.	Part of emergency surfacing logic or mode transition sequence.	$12
Camera Tilt Servo	Feetech Waterproof Servo (FT5335M)	Adjusts camera tilt during underwater scans.	PWM-controlled for dynamic or pre-programmed camera orientation.	$60

Table 1. Actuator

This integrated actuation system ensures the drone has full six degrees of freedom underwater, with stable flight control in air, and precise vertical depth regulation via internal ballast adjustment.

3.3 Sensors

The sensor suite supports perception, localization, and environmental interaction both above and below the water surface. All components are integrated with the onboard processor for real-time or logged data fusion.

Component	Model/Part	Purpose	Integration	Price
IMU (Inertial Measurement Unit)	SBG Systems Pulse-40	Estimates orientation and motion using acceleration and angular velocity.	Fused with magnetometer and depth sensor for full pose estimation; essential underwater where GPS fails	$1000
Magnetometer	JW Fishers Proton 5	Provides heading data based on Earth's magnetic field.	Works with IMU to maintain accurate orientation during submerged operation.	$95
GPS Receiver	MakerFocus GTU7	Provides absolute positioning in aerial or surface mode.	Logs surface location pre/post dive; nonfunctional under-water.	$19
Barometric Altimeter	Bosch BMP388	Measures atmospheric pressure to estimate aerial altitude.	Assists with flight and descent-to-water transitions; auto-matically disabled underwater.	$10
Depth/Pressure Sensor	Blue Robotics Bar30	Measures water pressure to determine submersion depth.	Primary underwater depth sensor; used with IMU to maintain vertical positioning accuracy.	$85
RGB/Underwater Camera	Chasing Gladius Mini	Captures visual data underwater in high resolution.	Captures visual data underwater in high resolution.	$1499
Underwater Imaging Sonar	Blue Robotics Ping360	Creates sonar - based imagery in turbid or dark water conditions.	Provides 2D/3D environmental awareness and supports terrainfollowing behaviors.	$2750

Table 2: Sensors

3.4 Processor

At the heart of the system is an edge computing module capable of handling AI workloads, sensor fusion, and mission logic with low power consumption and rugged performance.

Component	Model/Part	Purpose	Integration	Price
AI Processor	NVIDIA Jetson Xavier NX	Processes sensor data, executes mission algorithms, stores logs.	Central controller for sensor fusion, underwater autonomy, image processing, and post-mission offloading.	$700

3.5 Costing

Components	Model/Part	Price
Aerial Propulsion Motors (x4)	T-Motor U13 Pro V2 2500KV	$300 each
Underwater Thrusters (x4)	Blue Robotics T200	$180 each
Ballast Control Pump	Blue Robotics Peristaltic Pump	$45
Solenoid Valve	Uxcell DC 12V Normally Closed Valve	$12
Camera Tilt Servo	Feetech Waterproof Servo (FT5335M)	$60
Sensors (IMP, GPS, etc)	MakerFocus GT-U7, SBG Systems Pulse-40	$90 - 300
Control Board and Software	Cube Orange+ Autopilot (HEX Technology)	$200 - 600
Battery System	DJI Agras T30 Battery (Smart Battery 29,000mAh 51.8V)	$800 - 1300
Waterproofing Material	MG Chemicals 422B Silicone Conformal Coating + IP67 Gaskets	$500 - 750
Miscellaneous	XT60 Connectors, Nylon Standoffs, M3 Hardware Kit	$150 - 300
Frame and Body Materials	AeroDrone Bolt Frame Kit	$400 - 650
Total Estimated Cost	Depending on component choices and specification	**$ 3384 - 5937**

Table 3. Budget Analysis

4 Project Timeline

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
Initiation – Objectives & Feasibility Study											
Planning – Scheduling, Resource & Risk Planning											
Design – CAD Modelling & Component Selection											
Simulation – Control & CFD Simulation											
Final Report											

5 Results and Discussion

From our simulations, we were able to see how our hybrid drone performed across both air and underwater environments. During the aerial phase, the drone flew steadily at around 30 meters, and as it began to descend, it followed the exact control logic we designed. Once it dropped below 10 meters, the aerial motors turned off, the wings retracted after a short delay, and as it reached the water's surface, the ballast system activated for the dive. As it passed the 2.5-meter depth mark, the underwater thrusters engaged, enabling smooth movement below the surface. These transitions were clearly reflected in the motion graphs, particularly in the Z-axis, which showed a smooth descent and underwater path.

We also simulated an underwater rover navigating through a 3D space filled with obstacles with MATLAB and Python. The rover managed to reach its waypoints and avoid collisions by using sonar and IMU sensor data. The ballast system handled vertical movement really well, so the thrusters could focus on horizontal propulsion. The PID controllers kept the rover on track and adjusted when needed. These results gave us confidence in the system's ability to adapt to different environments.

Because building the full CAD model and running high-detail simulations was quite complex, we started experimenting with Sim's cape Multibody and Unreal Engine to better visualise and understand the system's physical behaviour. Although we couldn't fully implement it yet, this setup holds a lot of promise for future development. It can also support hardware-in-the-loop (HIL) testing, meaning we could eventually test real control systems in real-time with the virtual model. Overall, our results show that the drone is not just theoretically sound—it's got real potential for practical, dual-environment missions.

6 Conclusion and Further Developments

Design and simulation of an autonomous hybrid aerial-underwater drone were developed using MATLAB and Simulink, including all major operational phases and subsystems. The task involved system development—spans from aerial flight path and control to underwater navigation, sensor integration, and obstacle avoidance.

Critical aspects of the simulation design where the autonomous wing retract feature of the drone during landing at a defined altitude when descending, resulting in a reduced drag profile for sea entry. When entering the sea, the ballast tank system-initiated buoyancy control and commencement of descent phase of the dive. All transitions utilized sensor-based logic and worked successfully in the simulated environment.

The drone was also programmed to follow a pre-programmed underwater path autonomously in event of communication failure and initiate auto upload of data to a remote server when it re-emerges. All these behaviours were designed and developed on MATLAB and Simulink.

This project demonstrates the simulation of a hybrid aerial-underwater vehicle, setting the stage for eventual physical prototyping. The simulation environment provides a solid basis for continued optimization and experimentation in varying mission scenarios.

References

[1] M. Shoaib, M. Faizan, and R. Irfan, "Review of hybrid aerial underwater vehicle: Cross-domain mobility and control challenges," *Ocean Engineering*, vol. 264, 2022, Art. no. 112545.

[2] Y. Zhao, J. Liu, and T. Yang, "Modelling and control of unmanned aerial/underwater vehicles using hybrid systems," *Control Engineering Practice*, vol. 85, pp. 85–95, 2019.

[3] Luo, C., Sun, Y., & Yu, J. (2020). Review on underwater and amphibious aerial-aquatic vehicles. *Ocean Engineering*, 216, 107871. https://doi.org/10.1016/j.oceaneng.2020.107871

[4] Xu, T., Fang, Y., Wang, H., & Ma, G. (2021). A survey of amphibious robots: Solutions and challenges. *Journal of Field Robotics*, 38(5), 675–697. https://doi.org/10.1002/rob.21974

[5] Sattar, J. (2019). Non-destructive underwater inspection robots: State-of-the-art and future directions. *Annual Reviews in Control*, 48, 161–175. https://doi.org/10.1016/j.arcontrol.2019.01.003

[6] Bryson, M., Johnson-Roberson, M., Pizarro, O., & Williams, S. B. (2016). Automated registration for multi-year robotic surveys of coral reefs. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2602–2609. https://doi.org/10.1109/ICRA.2016.7487409

[7] A. Rahul and A. Sanyal, "Design Considerations for Aerial and Underwater Drones," *International Journal of Robotics and Automation (IJRA)*, vol. 7, no. 2, pp. 94–100, 2018.

[8] C. Zhou, Z. Xu, Z. Deng, and Z. Liang, "Development of a Hybrid Aerial-Underwater Vehicle: Design, Modelling, and Experiments," *IEEE Access*, vol. 8, pp. 116450–116461, 2020.

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770– 778. https://doi.org/10.1109/CVPR.2016.90

[10] Qin, Y., He, C., & Song, B. (2019). A survey of solar-powered unmanned aerial vehicles: Developments and challenges. *Renewable and Sustainable Energy Reviews*, 106, 274–287. https://doi.org/10.1016/j.rser.2019.02.039

[11] Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1), 1–41. https://doi.org/10.1007/s11721-012-0075-2

[12] Xu, J., Ma, G., & Wang, H. (2019). A review of autonomous underwater vehicle technology and its applications. *Ocean Engineering*, 175, 33–48. https://doi.org/10.1016/j.oceaneng.2019.02.048

Appendix A: Technical Specifications

The table includes everything from motors and sensors to the control board, battery, and structural materials. We selected each part not just for how well it works in our MATLAB/Simulink simulations, but also for how it would perform in real-world conditions. The choices were made with practical drone operation in mind—ensuring components could handle both air and underwater environments, withstand pressure, stay waterproof, and keep the drone stable and efficient. These technical details form the backbone of a system that's not only functional in theory but also ready for real-life testing and use.

Components	Model/Part
Aerial Propulsion Motors (x4)	T-Motor U13 Pro V2 2500KV
Underwater Thrusters (x4)	Blue Robotics T200
Ballast Control Pump	Blue Robotics Peristaltic Pump
Solenoid Valve	Uxcell DC 12V Normally Closed Valve
Camera Tilt Servo	Feetech Waterproof Servo (FT5335M)
Sensors (IMP, GPS, etc)	MakerFocus GT-U7, SBG Systems Pulse-40
Control Board and Software	Cube Orange+ Autopilot (HEX Technology)
Battery System	DJI Agras T30 Battery (Smart Battery 29,000mAh 51.8V)
Waterproofing Material	MG Chemicals 422B Silicone Conformal Coating + IP67 Gaskets
Miscellaneous	XT60 Connectors, Nylon Standoffs, M3 Hardware Kit
Frame and Body Materials	AeroDrone Bolt Frame Kit

Appendix B: Source Code

The Source code for the above simulation is given below,

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.patches as mpatches

# --- Obstacle avoidance helper ---
def check_obstacle_ahead(state, obstacles, sonar_range=30, avoid_angle_deg=60):
    rover_pos = state['pos']
    rover_yaw = state['yaw']
    heading = np.array([np.cos(rover_yaw), np.sin(rover_yaw), 0])
    min_distance = None
    avoid_sign = 0
    threat_found = False
    for obs in obstacles:
        obs_center = np.array(obs['center'])
        obs_vec = obs_center - rover_pos
        obs_dist = np.linalg.norm(obs_vec)
        obs_dir = obs_vec / (obs_dist + 1e-8)
        angle = np.arccos(np.clip(np.dot(heading, obs_dir), -1, 1)) * 180 / np.pi
        if obs_dist < sonar_range + obs['radius'] and angle < avoid_angle_deg:
            if (min_distance is None) or (obs_dist < min_distance):
                min_distance = obs_dist
                cross = np.cross(heading, obs_dir)
                avoid_sign = np.sign(cross[2])  # +1 = turn left, -1 = right
                threat_found = True
    if threat_found:
        return True, avoid_sign
    else:
        return False, 0

# --- Sonar sweep simulation ---
def sonar_sweep(state, obstacles, sweep_deg=120, n_beams=24, sonar_range=30, noise_std=0.3):
    rover_pos = state['pos']
    rover_yaw = state['yaw']
    bearings = np.linspace(-sweep_deg/2, sweep_deg/2, n_beams) * np.pi / 180
    ranges = np.ones(n_beams) * sonar_range
    for i, b in enumerate(bearings):
        beam_dir = np.array([np.cos(rover_yaw + b), np.sin(rover_yaw + b), 0])
        for obs in obstacles:
            obs_center = np.array(obs['center'])
            rel_vec = obs_center - rover_pos
            proj_length = np.dot(rel_vec, beam_dir)
            if 0 < proj_length < sonar_range + obs['radius']:
                closest_pt = rover_pos + proj_length * beam_dir
                dist_to_surface = np.linalg.norm(obs_center - closest_pt) - obs['radius']
                if dist_to_surface < 0.5:  # Beam intersects obstacle
                    if proj_length < ranges[i]:
                        ranges[i] = max(0.5, proj_length + np.random.normal(0, noise_std))
```

```python
                closest_pt = rover_pos + proj_length * beam_dir
                dist_to_surface = np.linalg.norm(obs_center - closest_pt) - obs['radius']
                if dist_to_surface < 0.5:  # Beam intersects obstacle
                    if proj_length < ranges[i]:
                        ranges[i] = max(0.5, proj_length + np.random.normal(0, noise_std))
    return bearings, ranges

# --- Environment and Obstacles ---
env_size = {'x': 500, 'y': 500, 'z': 300}
num_obstacles = 20
min_radius, max_radius = 20, 60

np.random.seed(42)
obstacles = []
for _ in range(num_obstacles):
    x = np.random.uniform(0, env_size['x'])
    y = np.random.uniform(0, env_size['y'])
    z = np.random.uniform(30, env_size['z'] - 10)
    radius = np.random.uniform(min_radius, max_radius)
    obstacles.append({'center': (x, y, z), 'radius': radius})

# --- Rover parameters ---
rover_length, rover_width, rover_height = 1.59, 0.57, 0.63
volume_rover = rover_length * rover_width * rover_height
mass_empty = 320
ballast_max = 260
Cd = 1.1
A_front = rover_width * rover_height
A_vert = rover_length * rover_width
rho_water = 1000
g = 9.81

# --- Waypoints (deep dive & back to surface) ---
waypoints = np.array([
    [250, 250, 0.5],
    [350, 300, 80],
    [450, 400, 170],
    [300, 400, 220],
    [150, 300, 295],
    [300, 150, 295],
    [400, 250, 295],
    [400, 400, 300],
])
waypoint_tol = 7
sim_time = 1500
dt = 0.1
thrust_max_single = 294
```

```
%% Setup Drone
m = 320;  % Updated mass of the drone in kg
I = [[0.1,0,0];[0,0.1,0];[0,0,0.08]]

% sample time
ts = 0.01

% Initial States (Initial XYZ is generated by XYZsignal script)
Euler_0 = [0;0;0]
XYZ_0 = [0;0;0]
body_rate_0 = [0;0;0]

% Environment
g = [0;0;-9.8]
```

```
    'ballast': 0.0,
}
active_wp = 1

# --- PID memories ---
yaw_error_prev = 0
yaw_error_int = 0
v_error_prev = 0
v_error_int = 0
z_error_prev = 0
z_error_int = 0

max_speed = 2.0
thruster_offset = rover_width / 2
inertia_yaw = 25.0

# --- Obstacle avoidance parameters ---
sonar_range = 30
avoid_angle_deg = 60
avoid_turn_strength = np.deg2rad(20)

for step in range(int(sim_time/dt)):
    to_wp = waypoints[active_wp] - state['pos']
    dist_to_wp = np.linalg.norm(to_wp)
    if dist_to_wp > 1e-6:
        dir_to_wp = to_wp / dist_to_wp
    else:
        dir_to_wp = np.array([1, 0, 0])

    v_target_vec = dir_to_wp * min(max_speed, dist_to_wp / dt)
    heading = np.array([np.cos(state['yaw']), np.sin(state['yaw'])])

    # --- OBSTACLE AVOIDANCE LOGIC ---
    threat, avoid_sign = check_obstacle_ahead(state, obstacles, sonar_range, avoid_angle_deg)
    if threat:
        target_yaw = state['yaw'] + avoid_turn_strength * avoid_sign
    else:
        if np.linalg.norm(v_target_vec[:2]) > 1e-5:
            target_yaw = np.arctan2(v_target_vec[1], v_target_vec[0])
        else:
            target_yaw = state['yaw']

    yaw_error = (target_yaw - state['yaw'] + np.pi) % (2*np.pi) - np.pi
    yaw_error_int += yaw_error * dt
    yaw_deriv = (yaw_error - yaw_error_prev) / dt
    torque_cmd = (Kp_yaw * yaw_error + Ki_yaw * yaw_error_int + Kd_yaw * yaw_deriv) * inertia_yaw
    max_torque = thrust_max_single * thruster_offset * 2
    torque_cmd = np.clip(torque_cmd, -max_torque, max_torque)
```

```
figure;
xlimit = [-5 5]; ylimit = [-5 5]; zlimit = [-7 5];
NewFigure(xlimit, ylimit, zlimit, -30, 18, 600, 600);
pause(1);

% Run animation
AniEulerTar(out.time, out.XYZ, out.EulerAngles, []);

%% Plot XYZ vs Time
figure;
subplot(3,1,1); plot(out.time, out.XYZ(1,:)'); xlabel('time'); ylabel('X Pos');
subplot(3,1,2); plot(out.time, out.XYZ(2,:)'); xlabel('time'); ylabel('Y Pos');
subplot(3,1,3); plot(out.time, out.XYZ(3,:)'); xlabel ('time'); ylabel('Z Pos');

%% Support Function
function NewFigure(xlim, ylim, zlim, viewx, viewy, w, h)
    set(gca, 'XLim', xlim, 'YLim', ylim, 'ZLim', zlim);
    view(viewx, viewy);
    set(gcf, 'Position', [10, 10, w, h]);
    hold on; grid on;
end
```

```
figure;
axis equal;
xlim([-5 5]);
ylim([-5 5]);
zlim([-7 7]);
xlabel('X'); ylabel('Y'); zlabel('Z');
grid on;
view(3);
title('Drone Path Simulation');
hold on;

% === Add realistic water surface ===
[xWater, yWater] = meshgrid(-5:0.1:5, -5:0.1:5);
zWater = 0;
surf(xWater, yWater, zWater * ones(size(xWater)), ...
    'FaceColor', [0.4 0.7 1], 'EdgeColor', 'none', ...
    'FaceAlpha', 0.4, 'DiffuseStrength', 0.6, 'SpecularStrength', 0.1);
lightangle(-45, 30);
lighting gouraud;
material dull;

% Plot full path
plot3(XYZ(1,:), XYZ(2,:), XYZ(3,:), 'k--');

% Initialize drone visuals
droneBody = plot3(0, 0, 0, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k');
xAxis = plot3([0 0], [0 0], [0 0], 'r', 'LineWidth', 2);
yAxis = plot3([0 0], [0 0], [0 0], 'g', 'LineWidth', 2);
zAxis = plot3([0 0], [0 0], [0 0], 'b', 'LineWidth', 2);

% === Animation Loop ===
step = 5;
for k = 1:step:length(time)
    pos = XYZ(:,k);

    % Euler Angles (roll, pitch, yaw)
    phi   = deg2rad(EulerAngles(1,k));
    theta = deg2rad(EulerAngles(2,k));
    psi   = deg2rad(EulerAngles(3,k));

    % Rotation Matrix (Z-Y-X)
    Rz = [cos(psi), -sin(psi), 0;
          sin(psi),  cos(psi), 0;
             0 ,        0 , 1];
    Ry = [cos(theta), 0, sin(theta);
             0 , 1,      0;
         -sin(theta), 0, cos(theta)];
    Rx = [1,     0     ,      0;
          0, cos(phi), -sin(phi);
          0, sin(phi),  cos(phi)];
    R = Rz * Ry * Rx;

    % Axis vectors
    len = 0.5;
    Xb = R * [len; 0; 0];
    Yb = R * [0; len; 0];
    Zb = R * [0; 0; len];

    % Update visuals
    set(droneBody, 'XData', pos(1), 'YData', pos(2), 'ZData', pos(3));
    set(xAxis, 'XData', [pos(1), pos(1)+Xb(1)], 'YData', [pos(2), pos(2)+Xb(2)], 'ZData', [pos(3), pos(3)+Xb(3)]);
    set(yAxis, 'XData', [pos(1), pos(1)+Yb(1)], 'YData', [pos(2), pos(2)+Yb(2)], 'ZData', [pos(3), pos(3)+Yb(3)]);
    set(zAxis, 'XData', [pos(1), pos(1)+Zb(1)], 'YData', [pos(2), pos(2)+Zb(2)], 'ZData', [pos(3), pos(3)+Zb(3)]);

    pause(0.3);
    drawnow;
end

end
```

```
    yaw_error = (target_yaw - state['yaw'] + np.pi) % (2*np.pi) - np.pi
    yaw_error_int += yaw_error * dt
    yaw_deriv = (yaw_error - yaw_error_prev) / dt
    torque_cmd = (Kp_yaw * yaw_error + Ki_yaw * yaw_error_int + Kd_yaw * yaw_deriv) * inertia_yaw
    max_torque = thrust_max_single * thruster_offset * 2
    torque_cmd = np.clip(torque_cmd, -max_torque, max_torque)
    yaw_error_prev = yaw_error

    v_target_forward = np.dot(v_target_vec[:2], heading)
    v_forward = np.dot(state['vel'][:2], heading)
    v_error = v_target_forward - v_forward
    v_error_int += v_error * dt
    v_deriv = (v_error - v_error_prev) / dt
    F_forward_cmd = Kp_v*v_error + Ki_v*v_error_int + Kd_v*v_deriv
    F_forward_cmd = np.clip(F_forward_cmd, -thrust_max_total, thrust_max_total)
    v_error_prev = v_error

    F_left = (F_forward_cmd - torque_cmd/thruster_offset) / 2
    F_right = (F_forward_cmd + torque_cmd/thruster_offset) / 2
    thrusters = np.array([F_left, F_left, F_right, F_right])
    thrusters = np.clip(thrusters, -thrust_max_single, thrust_max_single)

    torque = (F_right - F_left) * thruster_offset
    state['yaw_rate'] += (torque / inertia_yaw) * dt
    state['yaw_rate'] = np.clip(state['yaw_rate'], -yaw_rate_max, yaw_rate_max)
    state['yaw'] += state['yaw_rate'] * dt
    state['yaw'] = (state['yaw'] + np.pi) % (2*np.pi) - np.pi

    v_body_xy = state['vel'][:2]
    v_body = np.linalg.norm(v_body_xy)
    F_drag_xy = 0.5 * rho_water * Cd * A_front * v_body**2 * (-np.sign(v_body))
    F_drag_xy_vec = F_drag_xy * (v_body_xy/v_body) if v_body > 1e-5 else np.zeros(2)

    net_F_body = np.sum(thrusters)
    acc_xy = (net_F_body / (mass_empty + state['ballast'])) * heading + F_drag_xy_vec/(mass_empty + state['ballast'])

    v_target_z = v_target_vec[2]
    v_z = state['vel'][2]
    z_err = v_target_z - v_z
    z_error_int += z_err * dt
    z_deriv = (z_err - z_error_prev) / dt
    Fz_cmd = (Kp_z * z_err + Ki_z * z_error_int + Kd_z * z_deriv) * (mass_empty + state['ballast'])
    F_buoyancy = rho_water * volume_rover * g
    F_drag_z = 0.5 * rho_water * Cd * A_vert * v_z**2 * (-np.sign(v_z))
    m_ballast_cmd = (F_buoyancy + F_drag_z - Fz_cmd)/g - mass_empty
    m_ballast_cmd = np.clip(m_ballast_cmd, 0, ballast_max)
    delta_ballast = m_ballast_cmd - state['ballast']
    max_ballast_change = ballast_rate_limit * dt
    delta_ballast = np.clip(delta_ballast, -max_ballast_change, max_ballast_change)
```

Peer Review

Instructions:

Please complete this confidential form honestly.

Mark yourself and your team members based on their involvement in the team project in the percentages of their contribution.

Total percentage should be 100%.

Examples: (25%, 10%, 25%, 20%, 20%)

Group Number / Name:

	Team member name	Description of the work done in no more than 20 words for each team member	Contribution [%]
1	Sidharth Sreedas Nambiar	Worked mainly on drone simulation. Also helped a lot with references and writing the results section.	20%
2	Mohan Senthil	Focused on the electronics side. Also did a big part of the literature review and helped with the conclusion.	20%
3	Dharson Raju Mariappan	Handled the switching part between air and water. Also worked with sensors and did some design work.	20%
4	Raghul Chinnathambi	Helped with drone simulation and budgeting. Also worked on the timeline and added details to the appendix.	20%
5	Wanninayaka Mudiyanselage Janith Bandara Wanninayaka	Did the underwater navigation and sensor logic. Also worked on the introduction and obstacle avoidance.	20%

After completing the form, please save it as PDF file, then submit through Canvas.

Student Number: s4061315

Student Name: Sidharth Sreedas Nambiar

Signed:  Date: 12/06/25