applied sciences

MDPI

Article

Modifications to ArduSub That Improve BlueROV SITL Accuracy and Design of Hybrid Autopilot

Patrick Ng [1] and Michael Krieg [2,*]

[1] Department of Mechanical Engineering, University of Hawaii at Manoa, Honolulu, HI 96822, USA; png@hawaii.edu
[2] Department of Ocean and Resources Engineering, University of Hawaii at Manoa, Honolulu, HI 96822, USA
* Correspondence: kriegmw@hawaii.edu

Abstract: Improvements to ArduSub for the BlueROV2 (BROV2) Heavy, necessary for accurate simulation and autonomous controller design, were implemented and validated in this work. The simulation model was made more accurate with new data obtained from real-world testing and values from the literature. The manual control algorithm in the BROV2 firmware was replaced with one compatible with automatic control. In a Robot Operating System (ROS), a proportional–derivative (PD) controller to assist augmented reality (AR) pilots in controlling angular degrees of freedom (DOF) of the vehicle was implemented. Open-loop testing determined the yaw hydrodynamic model of the vehicle. A general mathematical method to determine PD gains as a function of the desired closed-loop performance was outlined. Testing was carried out in the updated simulation environment. Step response testing found that a modified derivative gain was necessary. Comparable real-world results were obtained using settings determined in the simulation environment. Frequency response testing of the modified yaw control law discovered that the bandwidth of the nonlinear system had a one-to-one correspondence with the desired closed-loop natural frequency of a simplified linear approximation. The control law was generalized for angular DOF and linear DOF were operated with open-loop control. A full six-DOF simulated dive demonstrated excellent tracking.

Keywords: ROV; UUV; control; ArduSub; ocean robotics; ROS; AR/VR

Citation: Ng, P.; Krieg, M. Modifications to ArduSub That Improve BlueROV SITL Accuracy and Design of Hybrid Autopilot. *Appl. Sci.* **2024**, *14*, 7453. https://doi.org/10.3390/app14177453

Academic Editor: Seongyeol Yoo

Received: 31 July 2024
Revised: 17 August 2024
Accepted: 21 August 2024
Published: 23 August 2024

1. Introduction

Remotely operated vehicles (ROVs) are unmanned marine submersible vehicles that play a vital role in the ocean industry and ocean research, in that they enable intervention and visual access to underwater environments inhospitable to humans. Large work-class ROVs have long served the oil and gas industry in the surveying and inspection of deep-sea operations. Through efforts like the Scientific and Environmental ROV Partnership using Existing Industrial Technology (SERPENT) Project, industrial ROVs' broad access to the deep recesses of the ocean is leveraged to share scientific data with the research community [1,2]. Other work-class ROVs like the Hercules, Jason, SuBastian, and Lu'ukai [3–6] are dedicated solely to research interests, contracted to collect samples from unique ocean locations and events, and building marine infrastructure (Figure 1).

Thanks to the ubiquity of powerful electronics and advances in manufacturing, the hardware behind ROVs and other marine technologies has become increasingly affordable, paving the way for more users and varied use cases. For example, the mining industry has shown a growing interest in deep-sea exploration because of dwindling land-based resources, massive monetary potential, and the improving costs of underwater exploration [7]. Some research groups have built their own custom vehicles like the Reef Rover for monitoring coral reefs [8], the Hovering Autonomous Underwater Vehicle that explores vision-based simultaneous localization and mapping [9], and the Acrobatic Low-cost Portable Hybrid Autonomous Underwater Vehicle [10]. Other projects have used or

modified consumer-priced ROVs like the Trident OpenROV for conservation planning [11]. OpenROV started out as a crowd-funded project that grew into an open-source platform for affordable ROV hardware and software. They have since been merged into Sofar Ocean Technologies [12] and discontinued their support for Trident, now focusing on passive ocean sensing.

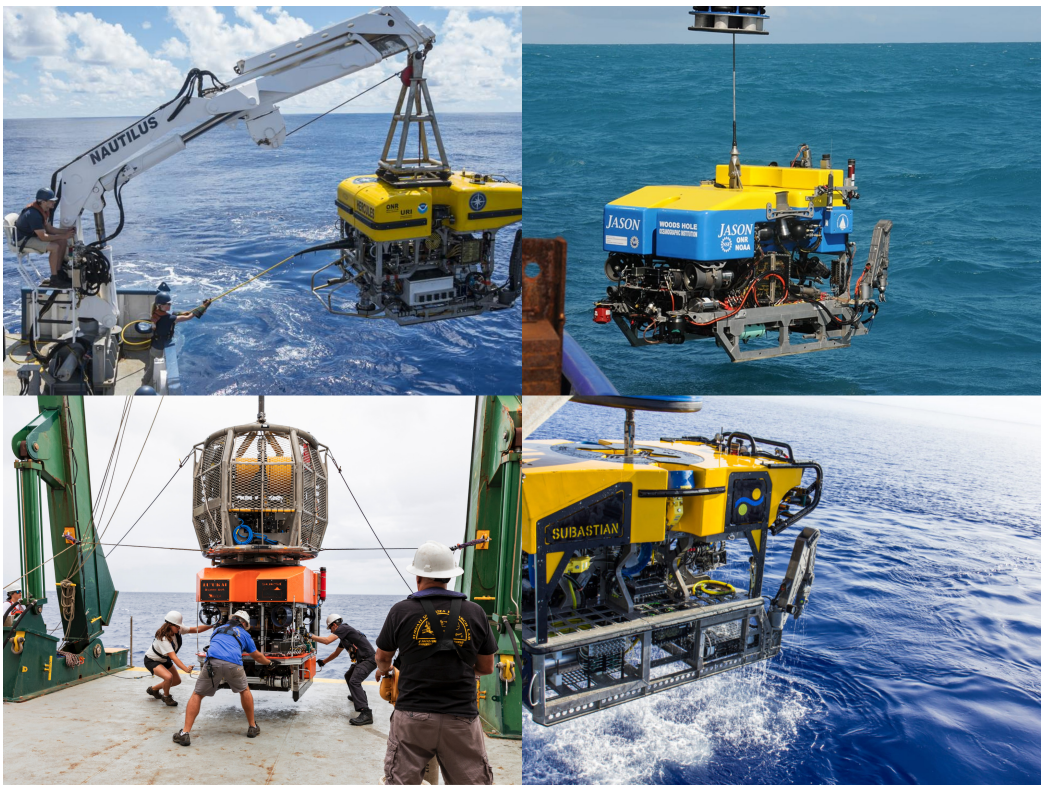

Figure 1. Clockwise from top left: Hercules [3], Jason [4], SuBastian [5], and Lu'ukai [6] are examples of research remotely operated vehicles (ROVs) currently in service.

Blue Robotics has a similar crowdfunding backstory, but has continued operations to the present day. It has established itself as a prominent source of affordable high-quality marine robotics equipment like thrusters, tethers, and pressure vessels. Their flagship product is the BlueROV2 (BROV2, Blue Robotics, CA, USA) [13], which comes in standard six-thruster or eight-thruster Heavy configurations (Figure 2), which have been popular choices among researchers. The BROV2 has been used in aquaculture to track the length of kelp with machine learning image processing [14] and been modified for under-ice sensing [15]. In light of the popularity of the BROV2, other works have focused on control topics, modeling the BROV2 and incorporating automation techniques. A comprehensive simulation study of the Heavy configuration was conducted in [16] but no real-world validation was performed. The study conducted by [17] looked at the six-thruster configuration and also conducted real-world validation; however, they swapped out the original controller components and implemented an entirely new software library. In this paper, both simulation and real-world testing shall be performed using stock components of a BROV2 Heavy and a modified version of the original software library so that the results may be more immediately applicable to other users.

Although ROVs are not suitable for long-distance travel due to their large drag and tether, they are equipped with numerous thrusters granting enhanced mobility over their hydrodynamic torpedo-like (AUV) counterparts. Piloting ROVs can be a daunting control task because of the challenges in underwater station keeping and managing several degrees of freedom, sometimes including one or more manipulator arms simultaneously, often requiring multiple pilots [18]. In order to address these challenges, the authors and

other collaborators have proposed an augmented reality (AR) pilot–ROV interface that aspires to incorporate a human's natural intuitive understanding of control by simplifying interactions in commanding the vehicle and to reduce the hurdles for training the future workforce [19,20]. This proposed interface uses virtual reality (VR) goggles and haptic feedback to immerse AR pilots. Given the novelty of this interface, several details need to be addressed. Throughout this study, we keep in mind the goal of gesture-based pilot commands and how they should be implemented within a flight controller. A long-term goal of this project involves collaboration between the University of Florida (UF) and the University of Hawaii at Manoa (UHM). The AR/VR system is managed by collaborators at the UF; pilots are hooked up to the system at their location. These pilots then send control commands to our group's location at the UHM where the physical BROV2 is located and is controlled by remote pilots over seven thousand kilometers away. Before accomplishing this vision, we must first address the control software that lies between the pilot and the vehicle.

Figure 2. The BlueROV2 (BROV2) Heavy has eight thrusters; four are positioned vertically on the upper layer and four are positioned at symmetric angles on a lower plane.

Ardupilot is a well established open-source autopilot firmware and software repository which provides user interfaces to aerial drones, terrestrial rovers, marine submersibles, and their ground stations via MAVLink messaging protocols [21]. The software is compatible with off-the-shelf controller hardware options from manufacturers like CUAV, CubePilot, mRO, and Holybro [22]. Its ability to provide guidance, navigation, and control features while also allowing for customization thanks to its open-source nature and compatibility with the Robot Operating System (ROS) makes it a popular choice among researchers. The Ardupilot project is divided up into vehicle-specific libraries (e.g., ArduCopter, ArduPlane, and ArduSub), shared common libraries, and packages dedicated to simulation. Software-in-the-loop (SITL) simulation lets users test out programs written for vehicles in a virtual environment and later use the code verbatim to control vehicle hardware in the real word. This interchangeability is valuable to researchers because it can achieve massive savings in resources by allowing a significant portion of the vehicle testing and familiarization phases to be carried out virtually before physical deployment.

Other simulation environments using the ROS exist, like Project Dave [23], Stonefish ROS [24], and ROS-MVP [25] that offer detailed three-dimensional (3D) graphical environments capable of simulating sensors and collision. None of these options integrate directly with off-the-shelf controllers like the SITL capabilities that are included with ArduSub, which is its major advantage. The compilation process that builds ArduSub vehicle

firmwares is tied to SITL so that alterations to software that work in the virtual environment will also work on the physical vehicle. Unfortunately, the existing BROV2 hydrodynamic model used in ArduSub SITL is not accurate and only useful for the most rudimentary test cases. This study fixes this issue and updates the vehicle model with experimental results from real-world testing.

Another limitation of ArduSub is the existing manual control implementation that follows a heuristic scheme that prioritizes the prevention of motor saturation by scaling down outputs with respect to the largest input. While this control method does generate smooth motion under manual control, such as joystick input with multiple degrees of freedom, thruster power levels no longer match those commanded when large inputs cause small commands to be damped indiscriminately. A new thruster distribution algorithm is proposed that operates within the existing software framework, granting precise quantitative control. This ability to command exact thrust levels instead of heuristically scaling down commands is required to accept commands from, and interface correctly with, error feedback autonomous control. This study updates the manual control algorithm to this quantitative description.

Using the revamped algorithm, proportional–derivative (PD) control laws for rotational degrees of freedom are implemented via ROS. The PD control law is intended to assist a human pilot sending commands over an AR interface, in which movements of the pilot's physical body are interpreted and assisted by the autopilot to carry out control. Consider a non-AR pilot controlling a six-degree-of-freedom (6DOF) vehicle with a joystick; the most natural interpretation of a joystick's movement is to relate the movement of any given joystick axis directly to an open-loop control force corresponding to that degree of freedom. This design is ineffective for an AR pilot because while some gestures/motions, such as bending forward, backward, or to the side, can be interpreted as direct thruster commands similar to joystick motions, other gestures such as pose should be interpreted as a desired vehicle orientation state. As such, an autopilot is required to provide thruster commands to achieve this pose based on closed-loop error feedback.

The development of the AR/VR project is an ongoing process; at the time of writing, the authors and collaborators have decided on a "hybrid autopilot" for control of the ROV. "Hybrid" means that the angular degrees of freedom are controlled with closed-loop error feedback, and linear degrees of freedom are controlled with open-loop inputs that do not use error feedback; this is in contrast to an autopilot that would use error feedback for all 6DOF, which would not need to be differentiated with the "hybrid" qualifier. During actual operations, a human pilot asserting control over the vehicle would consider sensor information in determining trajectory, and this would therefore be a form of closed-loop control known as pilot in the loop (PIL). However, from the perspective of the onboard controller the pilot inputs for linear degrees of freedom directly command thruster forces and so are considered open-loop.

The underwater setting is a challenging one; due to viscous effects and thruster limitations, the resulting system has nonlinear dynamics. A standard PD control law for a linear time-invariant (LTI) approximation of the nonlinear dynamics is proposed as a starting template. In this paper, the results of the yaw degree of freedom calibration are covered in depth. Preliminary experimental water tank testing was performed to determine the yaw hydrodynamic coefficients, which were then generalized and combined with existing values available from the literature to update simulation and controller hydrodynamic models. The source code of the vehicle firmware was also updated to accept quantitative inputs from the ROS hybrid autopilot and produce quantitative motion of the vehicle.

After performing step response experiments, it was determined that a modified version of the derivative gains best accommodated nonlinearities and it was found that results in simulation corroborated the water tank testing. Real-world testing of the yaw step response was also conducted to verify the efficacy of the results derived from theory and simulation. A final series of tests uses the complete hybrid autopilot, incorporating control laws for all angular DOF and open-loop linear control, in which the ability to follow a simulated pilot

6DOF input provided by collaborators from the UF is measured in simulation. On one hand, this work improves the ArduSub SITL model and we demonstrate its effectiveness and practicality through experimental results. On the other hand, we also describe alterations to ArduSub that connect to the hybrid autopilot developed in the ROS for quantitative control. In building this hybrid autopilot system, we pave the way for AR/VR pilots to command the BROV2 with gesture-based control.

In Section 2, the changes to the original ArduSub shall be covered, in which the original implementation is reviewed and the proposed changes that were tested are justified. Section 3 describes the open-loop testing performed and its analysis to determine the yaw hydrodynamic model of the BROV2. Section 4 covers the mathematical and control theory framework behind the PD controller for the autopilot. Section 5 walks through the selection of trim conditions for nonlinear dynamics approximations, an experimental procedure to adjust the control law for the yaw degree of freedom, characterizing the tuned yaw PD control law performance with Bode plots, and an experiment testing the full 6DOF autopilot with a simulated AR pilot flight. Finally, Section 6 shall summarize and draw conclusions about the current state of the project and preview future work.

2. ArduSub Alterations

This section discusses the hardware and software limitations of ArduSub used on the BROV2 platform. Some conjecture and commentary on why such limitations exists are included in the discussion. After the system overview, a brief introduction to the coordinate systems of the following theoretical framework is laid out. Changes to the autopilot firmware are reviewed by walking through the organization of the ArduSub control software, identifying flaws in the original algorithm, and introducing a new algorithm and the mathematical theory supporting it. Other changes to the SITL model as well as the simulation control allocation for virtual vehicle state updates are also documented.

2.1. Hardware and Software Limitations

Out of the box, the R1, R2, and R3 versions of the BROV2 come equipped with a Pixhawk and the R4 version (those sold after 7 June 2023) comes with a Navigator [26], both of which are flight management units (FMUs). Regarding hardware connections, onboard the FMU are input and output serial ports, sensors, and pulse width modulation (PWM) output connections for accessories and motor electronic speed controllers (ESCs). Onboard the R1-R3 version, the Pixhawk FMU handles the pilot inputs, sensor processing, and state estimation on a STM32F427 system on a chip (SoC). The Pixhawk SoC has a 180 MHz CPU, 256 KB of RAM, and a 2 MB flash drive which the ArduSub firmware is stored on [27].

Navigator FMUs on R4 versions act as a hardware-on-top augmentation of the Raspberry Pi 4 companion computer, giving FMU computations direct access to the much more powerful single-board computer.

Perhaps in consideration of the lean technical specifications of the Pixhawk and other lightweight FMUs that may be utilized on marine vehicles, the ArduSub source code was written with efficiency in mind [17]. That is to say, certain software limitations stem from hardware limitations. The autopilot libraries indicated with the '`AP_`' filename prefix are built using custom math functions and primitive data types, performing the bulk of computations with three element vectors, three-by-three matrices, and a limited selection of linear algebra operations. While this implements an Extended Kalman Filter on a Pixhawk, it must do so in roundabout ways to circumvent a more extensive matrix math library that would require larger processing overhead. In the ArduSub source code, this manifests as frequently breaking apart calculations into smaller pieces, like separating linear and angular dimensions into two three-dimension vectors, using program loops that iterate over the elements, and then recombining pertinent values with summation and scaling.

The key challenge in developing ArduSub is operating within the existing software framework because external libraries cannot be used.

2.2. Changes to ArduSub and Motivation

In this study, bolded variables represent vectors or matrices and non-bolded variables represent scalars. The body-fixed frame standard for ROS vehicles follows Forward–Left–Up (FLU) relative to an East–North–Up (ENU) inertial frame, whereas ArduSub's body-fixed frame is Forward–Right–Down (FRD), referencing a North–East–Down (NED) inertial frame. In robotics applications that use the ROS, there are certain guidelines that exists known as the ROS Enhancement Proposals (REPs). Generally, the REPs are guidelines to follow that help developers integrate with the larger ROS ecosystem. They serve as valuable touchstones which provide common threads for the community to latch on to, and standardize many fundamental things that help demystify the open-source landscape. For example, the convention that all ROS vehicles follow FLU-ENU is because of REP 103 [28]. There is a longstanding issue that those using the ROS in maritime applications were urged to use FLU-ENU frames by REP 103, but the rest of the industry followed the FRD-NED, also known as forward–starboard–down, FSD-NED, frame conventions. A new REP, 156, puts this issue to rest and advises that ROS users in maritime applications preserve FLU-ENU as the primary frames, but also maintain secondary frames with the appended suffixes "`_fsd`" or "`_ned`" [29]. Readers are also encouraged to follow the ROS Maritime Working Group updates on the ROS Discourse forums [30]. Moving forward, in this section all discussion and theory will reference an FRD-NED/FSD-NED frame if applicable.

The thruster configuration of the BROV2 Heavy can be seen in Figure 3: horizontal azimuthal thrusters 1–4 create surge, sway, and yaw motions; vertical thrusters 5–8 create heave, roll, and pitch motions; and thrusters are numbered with indices that are referenced from this point onward. The symbols for vehicle states in the inertial frame are $\boldsymbol{\eta} = [\boldsymbol{\eta_1}, \boldsymbol{\eta_2}]^\top$, where linear components are grouped in $\boldsymbol{\eta_1} = [x, y, z]^\top$ and angular components are grouped in $\boldsymbol{\eta_2} = [\phi, \theta, \psi]^\top$. The symbols for velocities in the body-fixed frame are $\boldsymbol{\nu} = [\boldsymbol{\nu_1}, \boldsymbol{\nu_2}]^\top$, where linear components are grouped in $\boldsymbol{\nu_1} = [u, v, w]^\top$ and angular components are grouped in $\boldsymbol{\nu_2} = [p, q, r]^\top$. The symbols for forces and torques are grouped together in a single vector $\boldsymbol{\tau} = [X, Y, Z, K, M, N]^\top$, where $\boldsymbol{\tau}_{1:3} = [X, Y, Z]^\top$ are linear forces and $\boldsymbol{\tau}_{4:6} = [K, M, N]^\top$ are angular torques.

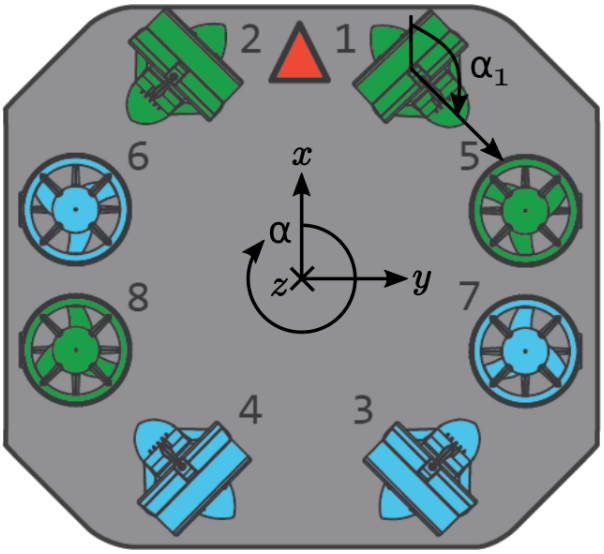

Figure 3. The BROV2 Heavy has four horizontal azimuthal thrusters and four vertical upright thrusters; the red triangle indicates the forward orientation of the vehicle; propellers of the thrusters shown with the color green (1, 2, 5 and 8) rotate counter-clockwise and those in blue (3, 4, 6, and 7) rotate clockwise, i.e., for the vertical thruster, the green ones have negative rotation about the z-axis and the green ones have positive rotation about the z-axis; the force resultant on the vehicle from positive forward thrust is in the direction of the more textured half of the azimuthal thrusters and downwards (on the page) for the vertical thrusters [31].

The autopilot design for our intended pilot interface receives desired angular states of the BROV2 from the virtual reality headset orientation in $\boldsymbol{\eta_2}$, $\boldsymbol{\nu_2}$, and $\boldsymbol{\tau}_{4:6}$ to achieve pilots' desired rotation with closed-loop feedback control. Linear degrees of freedom are currently handled with open-loop control from pilot gestures and handheld controller inputs that do not use error feedback (Figure 4).

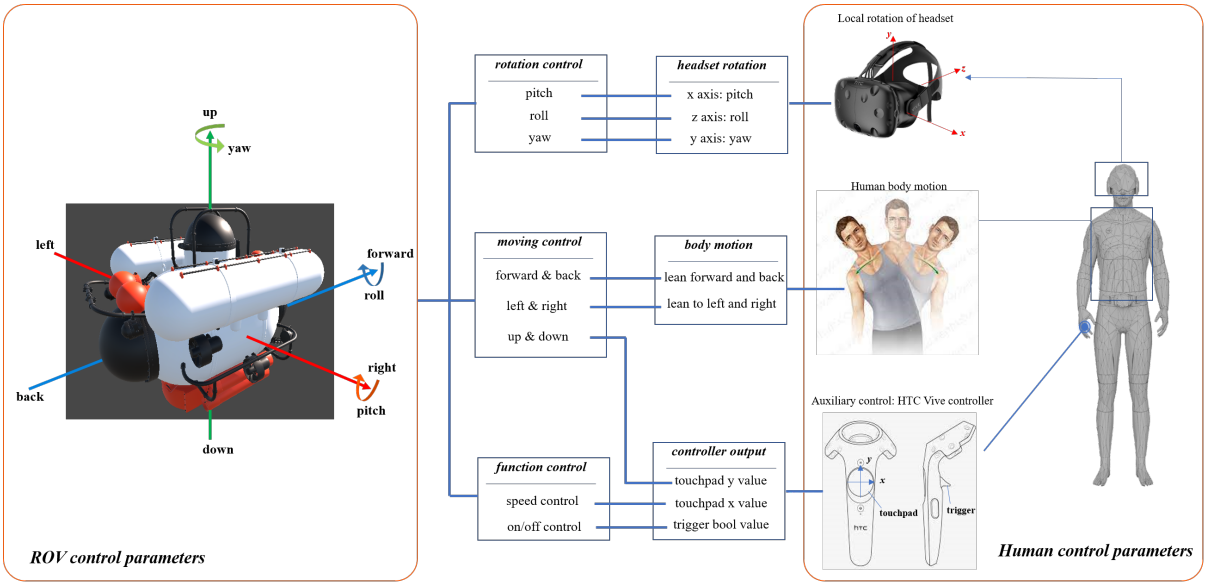

Figure 4. The mapping of gestures to control, as proposed by collaborators from University of Florida (UF); the coordinate systems for the human interface follow the Forward–Right–Down (FRD) body-fixed frame convention [19], as pictured.

2.2.1. Thruster Allocation Matrix Overview

ROVs like the BROV2 owe their high manoeuvrability to the multiple thrusters that their boxy frame carries. Each of these thrusters work in tandem to impart control forces and torques on the body of the vehicle to carry out movements and stabilize the vehicle.

For the BROV2 Heavy equipped with eight thrusters, this can be expressed quantitatively,

$$\boldsymbol{\tau} = \boldsymbol{T}\boldsymbol{m} \tag{1}$$

where $\boldsymbol{\tau}$ is the six-by-one vector of control forces and torques, $\boldsymbol{T}$ is the six-by-eight thruster allocation matrix [32], and $\boldsymbol{m}$ is the eight-by-one vector of the force contributions of the eight thrusters. The constant matrix $\boldsymbol{T}$ is a linear transformation that maps the force and torque inputs of the motors $\boldsymbol{m}$ to the overall resultant forces and torques on the vehicle body $\boldsymbol{\tau}$.

The first three rows of $\boldsymbol{T}$ transform the eight thruster input forces into the resultant body linear forces and are constructed column-wise with

$$\boldsymbol{T}_{1:3,i} = [\cos(\alpha_i) \ \sin(\alpha_i) \ \sin(\beta_i)]^\top, \tag{2}$$

where α_i is the angle of the i-th thruster in the horizontal plane of the vehicle and β_i is the angle of the i-th thruster in the perpendicular upright plane (see Figure 3). The vector described in Equation (2) encodes the geometric orientation of the i-th thruster in space and can be considered a type of unit vector for forces. All the force unit vectors for the eight thrusters' orientations put together form the top three rows of $\boldsymbol{T}$, which when given an input $\boldsymbol{m}$, outputs the linear forces (i.e., the first three entries of $\boldsymbol{\tau}$). Figure 3 displays the FRD body frame coordinate axes, relative locations, and orientations of the thruster configuration. Thruster 1 is annotated as an example of how α_1 is defined: the angle α follows the right-hand convention for clockwise yaw rotation about the downwards-

pointing z-axis; β is not visible in the figure but also follows the right-hand convention for pitch rotation about the y-axis.

The last three rows of $\boldsymbol{T}$ map the thruster force inputs to the resultant body torques and are constructed column-wise with

$$\boldsymbol{T}_{4:6,i} = \boldsymbol{r}_i \times \boldsymbol{T}_{1:3,i}. \tag{3}$$

where $\boldsymbol{r_i}$ is the location of the i-th thruster relative to the vehicle's center of mass, and a cross product is performed with it and the first three rows of the i-th column of $\boldsymbol{T}$ determined earlier in Equation (2). Similar to how Equation (2) is a unit vector for forces, Equation (3) is a unit vector for torques. All the torque unit vectors for the eight thrusters are put together to form the bottom three rows of $\boldsymbol{T}$, which when given an input $\boldsymbol{m}$, output the angular torques (last three entries of $\boldsymbol{\tau}$).

Using the vehicle system geometry presented in [16] and concatenating the eight vectors resulting from Equations (2) and (3), the total thruster allocation matrix for the BROV2 Heavy is given in Equation (4) rounded to two decimal places. Given a vector $\boldsymbol{m}$ containing the variable thrust of all eight thrusters, Equations (1) and (4) can determine the control forces and torques $\boldsymbol{\tau}$ resulting from the thrusters on the BROV2 body.

$$\boldsymbol{T} = \begin{pmatrix} -0.71 & -0.71 & 0.71 & 0.71 & 0 & 0 & 0 & 0 \\ 0.71 & -0.71 & 0.71 & -0.71 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 1.0 & 1.0 & 1.0 \\ -0.06 & 0.06 & -0.06 & 0.06 & 0.22 & -0.22 & 0.22 & -0.22 \\ -0.06 & -0.06 & 0.06 & 0.06 & -0.12 & -0.12 & 0.12 & 0.12 \\ 0.99 & -0.99 & -0.99 & 0.99 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{4}$$

Now, consider if some desired forces and torques on the BROV2 were given and the necessary motor activations $\boldsymbol{m}$ needed to be calculated instead, as is the case of an autopilot mechanism; the inverse thruster allocation matrix would be required. This is

$$\boldsymbol{m} = \boldsymbol{T}^{\dagger}\boldsymbol{\tau} \tag{5}$$

where $\boldsymbol{T}^{\dagger}$ is the pseudo-inverse of the rectangular matrix $\boldsymbol{T}$. The actual matrix computed from the pseudo-inverse of Equation (4) has values of

$$\boldsymbol{T}^{\dagger} = \begin{pmatrix} -0.35 & 0.35 & 0 & 0 & 0 & 1.32 \\ -0.35 & -0.35 & 0 & 0 & 0 & -1.32 \\ 0.35 & 0.35 & 0 & 0 & 0 & -1.32 \\ 0.35 & -0.35 & 0 & 0 & 0 & 1.32 \\ 0.18 & 0.10 & 0.25 & 1.15 & -2.08 & 0 \\ 0.18 & -0.10 & 0.25 & -1.15 & -2.08 & 0 \\ -0.18 & 0.10 & 0.25 & 1.15 & 2.08 & 0 \\ -0.18 & -0.10 & 0.25 & -1.15 & 2.08 & 0 \end{pmatrix}. \tag{6}$$

The matrix above can transform an input vector of desired forces and torques on the vehicle body $\boldsymbol{\tau}$ into the required motor activations $\boldsymbol{m}$ to achieve them. These arrays $\boldsymbol{T}$, $\boldsymbol{m}$, $\boldsymbol{\tau}$, and $\boldsymbol{T}^{\dagger}$ have compatible physical units in the Meters–Kilogram–Second (MKS) system of measurements. With these definitions in place, the process of implementing the changes in the software shall be explained next.

2.2.2. Implementation

In implementing custom changes to ArduSub, it is important that modifications are precise and that the rest of the codebase's functionality is preserved. The locations in the code handling individual thruster allocation were identified to be within the files `SIM_Submarine.cpp` and `AP_Motors6DOF.cpp`. The former carries out calculations for state updates of the SITL virtual vehicle due to motor inputs and contains a matrix

$$\widehat{\boldsymbol{T}} = \begin{pmatrix} -1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{7}$$

In this equation, a hat is used to distinguish $\widehat{\boldsymbol{T}}$ from the matrix $\boldsymbol{T}$. The latter contains a matrix for distributing pilot control inputs to the individual thruster activations with the form

$$\widehat{\boldsymbol{T}}^{\top} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 1 \\ -1 & -1 & 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \end{pmatrix}, \tag{8}$$

where $\widehat{\boldsymbol{T}}^{\top}$ is the transpose of Equation (7). Upon closer inspection of the two matrices, both are organized in the same fashion as the thruster allocation matrix and its inverse, and are used for similar purposes: $\widehat{\boldsymbol{T}}$ is to $\boldsymbol{T}$ as $\widehat{\boldsymbol{T}}^{\top}$ is to $\boldsymbol{T}^{\dagger}$. Where the zero entries in $\widehat{\boldsymbol{T}}$ and $\widehat{\boldsymbol{T}}^{\top}$ are the entries close to zero in $\boldsymbol{T}$ and $\boldsymbol{T}^{\dagger}$, the nonzero entries also share the same positive or negative sign. However, the hatted matrices lack the dimensionality that the theoretical ones possess because they do not implicitly encode physical units and were intended for a heuristic control scheme instead of a quantitative one.

Figure 5 provides a simple flow chart visualization of the following description. Every iteration of the software control loop checks the `RC_Input` channel for the six-by-one vector of PWM signals ranging from 1100 µs to 1900 µs, centered on 1500 µs where values $>$ 1500 µs are positive values and $<$ 1500 µs are negative values. The six entries correspond to surge, sway, heave, roll, pitch, and yaw commands and are each normalized to a -1.0 to 1.0 range that becomes the user input vector $\boldsymbol{u}$. This vector $\boldsymbol{u}$ is then read into `AP_Motors6DOF.cpp` where the function `motor_command()` calculates the thruster allocation; Algorithm 1 details this loop and shall be discussed shortly.

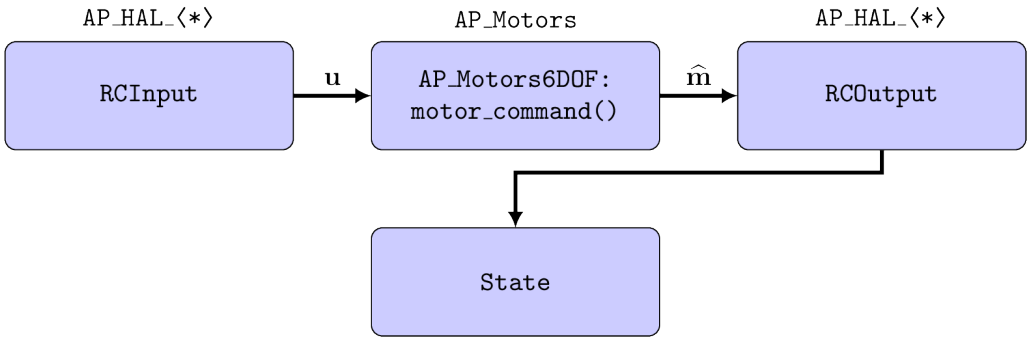

Figure 5. Manual control data travel through programs sequentially in arrays which are marked with arrows below their symbolic variables; above the boxes are the library names and the inside of the boxes are the module names.

Calculations performed in the primary loop of `AP_Motors6DOF.cpp` take the forces and torques requested by $\boldsymbol{u}$ and determine how the eight individual thrusters should activate to serve these commands. The resulting eight-by-one vector of these calculations $\boldsymbol{m}^*$ is analogous to the forces from Equation (5). Each entry represents one of the eight

thrusters and is normalized to a -1.0 to 1.0 range. The entries of $\boldsymbol{m}^*$ are converted once more to PWM signals in the 1100 µs to 1900 µs range and output as $\widehat{\boldsymbol{m}}$ to the `RC_Output` channel. Once in `RC_Output`, the PWMs are distributed throughout the respective ESCs corresponding to the eight individual thrusters. Ultimately, this activates the thrusters that generate forces and torques on the vehicle.

To understand how exactly $\widehat{\boldsymbol{T}}^\top$ is used, Algorithm 1 provides a step-by-step clarification. As of the ArduSub 4.5.0 beta1 release on 22 February 2024, the Ardupilot/ArduSub source code still utilizes Algorithm 1 found in `AP_Motors6DOF.cpp` for the manual control mode [33]. The matrix is used as a constant parameter that helps transform user input $\boldsymbol{u}$ into motor activation $\boldsymbol{m}^*$. The function performing the computation is `motor_command()`. Within the function, there are various forms of ξ which serve as buffer variables used inside the function. The first usage is introduced on lines 2 and 9, with the form ${}^{(j)}\xi_{max}$, where the left superscript j can be either 1 for roll, pitch, and heave control or 2 for yaw, surge, and sway control and the right subscript "max" labels its purpose as a normalizing variable. The second type of usage is introduced on lines 4 and 11, with the form ${}^{(j)}\xi_i$, where the left superscript j again can be either 1 for roll, pitch, and heave control or 2 for yaw, surge, and sway control and the right subscript i is an index indicating the i-th thruster which is being calculated for. The groupings of forces in ${}^{(1)}\xi$ terms correspond to those generated primarily by the vertical thrusters and groupings of forces in ${}^{(2)}\xi$ correspond to those generated primarily by the azimuthal thrusters.

Algorithm 1: Original Manual Control Algorithm

Parameters: $\widehat{\boldsymbol{T}}^\top_{\boldsymbol{8\times 6}}$
Input: $\boldsymbol{u_{6\times 1}}$
Output: $\boldsymbol{m^*_{8\times 1}}$

1 **Function** `motor_command(`$\boldsymbol{u}$,$\widehat{\boldsymbol{T}}^\top$`):`
2 $\quad {}^{(1)}\xi_{max} = 1;$
3 $\quad$ **for** $i \leftarrow 1$ **to** 8 **do**
4 $\quad\quad {}^{(1)}\xi_i = u_3 \cdot \widehat{T}^\top_{i,3} + \boldsymbol{u}_{4:5} \cdot \widehat{\boldsymbol{T}}^\top_{i,4:5};$
5 $\quad\quad$ **if** $|{}^{(1)}\xi_i| > {}^{(1)}\xi_{max}$ **then**
6 $\quad\quad\quad {}^{(1)}\xi_{max} = {}^{(1)}\xi_i;$
7 $\quad\quad$ **end**
8 $\quad$ **end**
9 $\quad {}^{(2)}\xi_{max} = 1;$
10 $\quad$ **for** $i \leftarrow 1$ **to** 8 **do**
11 $\quad\quad {}^{(2)}\xi_i = \boldsymbol{u}_{1:2} \cdot \widehat{\boldsymbol{T}}^\top_{i,1:2} + u_6 \cdot \widehat{T}^\top_{i,6};$
12 $\quad\quad$ **if** $|{}^{(2)}\xi_i| > {}^{(2)}\xi_{max}$ **then**
13 $\quad\quad\quad {}^{(2)}\xi_{max} = {}^{(2)}\xi_i;$
14 $\quad\quad$ **end**
15 $\quad$ **end**
16 $\quad$ **for** $i \leftarrow 1$ **to** 8 **do**
17 $\quad\quad m^*_i = \dfrac{{}^{(1)}\xi_i}{{}^{(1)}\xi_{max}} + \dfrac{{}^{(2)}\xi_i}{{}^{(2)}\xi_{max}};$
18 $\quad\quad$ `constrain`$(m_i \in [-1.0, 1.0]);$
19 $\quad\quad \widehat{m}_i = 1500 + 400 m^*_i;$
20 $\quad$ **end**
21 $\quad$ send $\widehat{\boldsymbol{m}}$ to `RC_Output` channels;

The function `motor_command()` consists of three loops, two for calculating preliminary output values based on inputs and the third loop for constructing the output. The first loop and second loop are identical except that the first loop calculates for roll, pitch, and heave control and the second loop calculates for yaw, surge, and sway control. The first two loops

step through the thrusters in order and scale outputs with respect to the corresponding parts of $\widehat{\boldsymbol{T}}^{\top}$ with respect to the corresponding user inputs of $\boldsymbol{u}$ and sum them together in the buffer ${}^{(j)}\xi_i$ (lines 4 and 11). These numbers are then checked against the current value of ${}^{(j)}\xi_{max}$ and replaced with the value of the ${}^{(j)}\xi_i$ buffer if it is larger (lines 5 and 12). After all eight thrusters have been looped over twice, the final third loop normalizes ${}^{(j)}\xi_i$ with respect to the final values of ${}^{(j)}\xi_{max}$ and adds the vertical and azimuthal ratios together (line 17). The trimming function `constrain()` checks that each value of $\boldsymbol{m}^*$ is in the range of -1.0 to 1.0 (line 18); if a value is outside the bounds, the function chooses the limit closest. Then, the values of $\boldsymbol{m}^*$ are scaled up to appropriate PWMs (line 19), and finally sent to `RC_Output` which relays the signals to the respective ESCs.

If a user were to command any combination of movement, the vehicle would respond with all requested movements, but all would be scaled with respect to the largest commanded force or torque. The overall effect of Algorithm 1 is that it generates intuitive vehicle responses that a pilot will recognize as matching joystick inputs. This heuristic method is a good way to prevent cases of motor saturation where the vehicle appears to be unresponsive due to motors being maxed out. However, the exact value of the resulting forces/torques, $\boldsymbol{\tau}$, will not directly correlate to the magnitude of the input commands, which is necessary for automatic control laws. Instead, quantification of forces should be preserved, and saturation avoided through selection of an adequate control bandwidth.

Thus, with this assessment of the original control implementation and the goal of implementing a PD control law, a new simple algorithm is determined which can replace the pre-existing framework without disrupting anything else or introducing unnecessary complexity.

In order for the new manual control algorithm to be integrated seamlessly, without altering processes on either end of the software control loop, it must use the same inputs and produce the same outputs. The main challenge is incorporating physical units that will allow the autopilot to command specific quantities of forces and torques. To achieve this, $\boldsymbol{T}^{\dagger}$ replaces $\widehat{\boldsymbol{T}}^{\top}$ and new parameters and functions are introduced that encode the BROV2 hardware limitations while preserving the control architecture. These new parameters are $\boldsymbol{\tau_{max}}$, $\boldsymbol{PWM}^*$, and $\boldsymbol{t}$, which are a six-by-one vector containing the maximum forces and torques that the BROV2 is capable of, eighty-one normalized PWMs and eighty-one corresponding thrusts produced by a T200 thruster. The contents of $\boldsymbol{PWM}^*$ and $\boldsymbol{t}$ are experimental data collected by Blue Robotics [34] and graphed in Figure 6 and used as a lookup table, utilized by the FMU thrust distribution and the SITL hydrodynamic model. The values of $\boldsymbol{\tau_{max}}$ are determined analytically by solving for $\boldsymbol{\tau}$ using Equation (1) given $\boldsymbol{m}$, constructed of maximum thrusts according to the lookup table in combinations producing solely surge, sway, heave, roll, pitch, or yaw manoeuvrers. This results in $\boldsymbol{\tau_{max,1:3}} = [141.29, 141.29, 199.81]$ N for forces and $\boldsymbol{\tau_{max,4:6}} = [43.56, 23.98, 37.72]$ Nm for torques. To use input signals from `RC_Input` of $\boldsymbol{u} \in [-1.0, 1.0]$, the signals are converted to units of forces and torques with $\boldsymbol{\tau_{max}}$, and transformed with $\boldsymbol{T}^{\dagger}$ to find $\boldsymbol{m}$ directly. Mathematically, these operations are

$$\boldsymbol{m} = \boldsymbol{T}^{\dagger} \boldsymbol{diag}(\boldsymbol{\tau_{max}}) \boldsymbol{u}. \tag{9}$$

Algorithm 2 shows how Equation (9) is accomplished in ArduSub step by step. Using the new parameters $\boldsymbol{\tau_{max}}$, $\boldsymbol{PWM}^*$, and $\boldsymbol{t}$, the same inputs $\boldsymbol{u}$ are used and the same outputs $\widehat{\boldsymbol{m}}$ are produced. However, now $\boldsymbol{u}$ originates from a hybrid autopilot which has calculated physical forces and torques (that have been normalized with respect to $\boldsymbol{\tau_{max}}$), as opposed to interpreting inputs heuristically. Linear DOF control is handled with an open-loop controller that directly passes inputs into $\boldsymbol{u_{1:3}}$. Angular control is handled with a closed-loop PD control law that interprets an AR pilot's gestures with error feedback, before passing inputs into $\boldsymbol{u_{4:6}}$. Section 4 shall discuss the control law in detail.

The new function `motor_command_new()` uses one software loop; first, it iteratively performs Equation (9) (line 3) to determine m_i. This is then input into the function `find_bounding_indicies()`, which determines the closest indices of $\boldsymbol{t}$ below ($_{low}$) and above ($_{high}$) the force of m_i (line 4). In the event that m_i is outside the range of $\boldsymbol{t}$, the $_{low}$

index will equal the $_{high}$ index and default to either -1.0 for negative forces or 1.0 for positive forces (lines 5 through 12). For all other instances, the bounding indices will be used to perform linear interpolation that produces a normalized value of m_i^* (lines 13 through 15). This value is trimmed (line 16) and then converted to PWMs (line 18) to finally be sent to `RC_Output`.

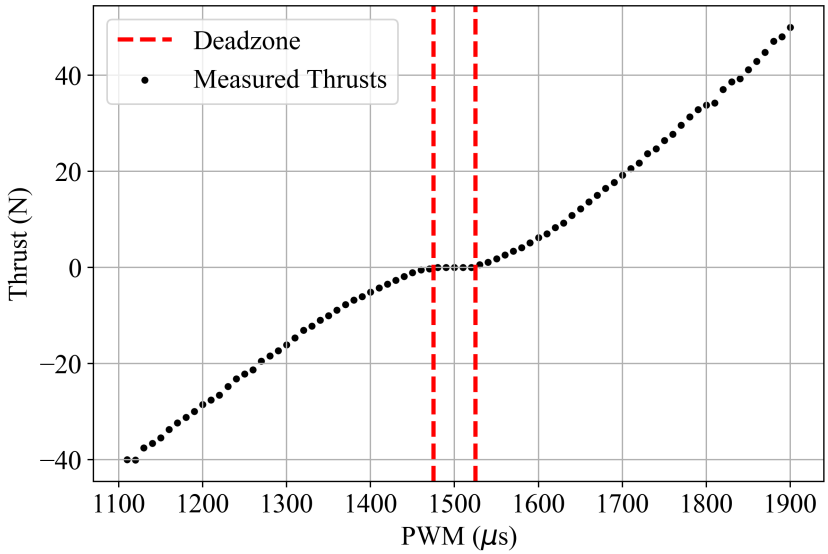

Figure 6. Blue Robotics experimental test results of pulse width modulation (PWM) input and resulting thrust for a single T200 thruster.

It is important to emphasize that while this new algorithm accounts for vehicle limitations, it depends on inputs from the hybrid autopilot that encode vehicle limitations as well.

Algorithm 2: New Manual Control Algorithm

Parameters: $\boldsymbol{T^{\dagger}_{8\times 6}}$, $\boldsymbol{\tau_{max,8\times 1}}$, $\boldsymbol{PWM^{*}_{81\times 1}}$, $\boldsymbol{t_{81\times 2}}$
Input: $\boldsymbol{u_{6\times 1}}$
Output: $\boldsymbol{m^{*}_{8\times 1}}$

1 **Function** `motor_command_new`$(\boldsymbol{u}, \boldsymbol{T^{\dagger}}, \boldsymbol{\tau_{max}})$**:**
2 **for** $i \leftarrow 1$ **to** 8 **do**
3 $m_i = \sum_{j=1}^{6} \left(T^{\dagger}{}_{i,j} \cdot \tau_{max,j} \cdot u_j\right)$;
4 $_{low}$,$_{high}$ = `find_bounding_indices`(m_i);
5 **if** $_{low} =$ $_{high}$ **then**
6 **if** $m_i < 0$ **then**
7 $m_i^* = -1.0$
8 **end**
9 **else if** $m_i > 0$ **then**
10 $m_i^* = 1.0$
11 **end**
12 **end**
13 **else**
14 $m_i^* = PWM_{low}^* + \dfrac{PWM_{high}^* - PWM_{low}^*}{t_{high} - t_{low}}(m_i - t_{low})$;
15 **end**
16 `constrain`$(m_i \in [-1.0, 1.0])$;
17 $\widehat{m}_i = 1500 + 400 m_i^*$;
18 **end**
19 send $\widehat{\boldsymbol{m}}$ to `RC_Output` channels;

2.2.3. Simulation Discussion

The module that enables devices to interface with ArduSub is known as the hardware abstraction layer (HAL). Each supported variety of hardware has a unique library; Pixhawk uses the `AP_HAL_ChibiOS` library, whereas our simulation uses `AP_HAL_SITL`. Once the vehicle's low-level hardware is able to communicate using ArduSub, it is able to access the shared libraries for general-purpose sensors, state estimation, and motor control. For example, the motor libraries that control the thrusters on the BROV2 share a common code with the libraries used to control thrusters on a quadcopter. In the case of a simulated virtual mission, a special SITL HAL is utilized to emulate a physical FMU that does not require a physical device to be in the loop; shared libraries further upstream remain the same regardless of the hardware connected to the HAL.

ArduSub uses the Python script `sim_vehicle.py` to invoke the SITL HAL that on one branch generates the virtual firmware and by default on a separate branch generates the virtual environment for submersible vehicles. At runtime, the latest state of the codebase is built and must pass through a robust verification process that checks that the raw code is written properly and will be able to be flashed to an actual vehicle FMU. This capability that seamlessly bridges virtual firmware testing and real-world vehicles is invaluable.

The other branch called by default when `sim_vehicle.py` is run generates the SITL virtual environment from `SIM_Submarine.cpp` and its header. Contained in these files are the hydrodynamic model of the BROV2 and control allocation algorithms (see Section 2.2.5) for the state updates of the virtual vehicle. Alternatively, users may use Gazebo instead of `SIM_Submarine.cpp` to perform these calculations. Gazebo is a powerful simulation environment that renders 3D dynamic scenes that represent robots and their environments, simulating physics, collisions, and sensors through an ever-expanding library of plugins. Current versions of Gazebo have plugins for thrusters, buoyancy, and hydrodynamics, including added mass effects [35]. Added mass characterizes the inertia of water that needs to be displaced by objects as they accelerate through water, increasing the effective mass. All of this amounts to a high-fidelity simulation that also uses the ArduSub virtual firmware backend.

Despite these appealing characteristics, researchers are likely to encounter challenges in using Gazebo. For projects using ROS 1 (like the subject of this work), there are not any well-documented methods for using Gazebo with ArduSub SITL; the guide available on the ArduSub website [36] is outdated. Versions of Gazebo that include the high-fidelity plugins described run only on ROS 2, which requires operating systems that do not support ROS 1. Using Gazebo also requires a computer with hardware capabilities able to run the 3D simulation at the desired refresh rate, which can be a considerable bottleneck. However, for those who are not limited by hardware or to ROS 1, there is an actively maintained, at the time of this writing, ArduSub SITL with Gazebo for ROS 2 called Orca4 [37].

In this study, we will not be using Gazebo and will be exploring and updating the default `sim_vehicle.py` simulation model in `SIM_Submarine.cpp`. While the default ArduSub SITL environment is a lower fidelity simulation without a 3D visualization component, we update it to use the same hydrodynamic coefficients that Gazebo would, and it can be expected to produce similar results for simple underwater movement not involving collision. The physical properties of the generic vehicle defined as the "`Submarine`" object, meant to represent the BROV2, in the SITL source code was inaccurate because it used several crude approximations, underestimated damping effects, and was missing added mass. For a vehicle like the BROV, there is an expected abundance of acceleration and deceleration because it is likely to be performing many stop-and-go manoeuvrers; therefore, a missing added mass factor amounts to significant errors. The specific updates are catalogued in Section 2.2.4. The virtual frame representing the BROV2 uses the heuristic thruster allocation matrix described in Equation (7). Nonzero elements are appropriately distributed and the signs are correct, but, as was mentioned, values do not correspond to physical units. The fix for this is to replace the thruster allocation matrix with Equation (4), which was calculated earlier.

The simulation code calculated vehicle accelerations similar to Algorithm 1, where the thruster allocation matrix was broken into three element vector pieces of the linear and angular components to be looped over for each motor. Linear and angular accelerations were originally calculated by dividing the thrust and torque by vehicle mass and moment of inertia; this operation was treated as a flat scaling operation where the reciprocal of the mass or inertia was multiplied by each element. In our modified code, an element-wise division function was implemented to allow dividing by the different values of the updated combined inertia and added mass vector in a single operation. The angular acceleration calculations originally approximated the vehicle as a sphere with all thrusters equidistant from the center of mass; this was corrected to model the actual geometric locations of the BROV2 thrusters as described in the thruster allocation matrix of Equation (4). Drag was calculated using the standard drag equation, where the coefficient of drag was set to that of a sphere. This might be an acceptable approximation, but was also implemented erroneously; the fix for this is described in Appendix A.

2.2.4. Vehicle Simulation Model Update

The original ArduSub vehicle model is inaccurate and produces unrealistic behavior. Appendix B introduces the full hydrodynamic model that the updates are based on as well as a method for determining control gains for a critically damped linear system approximation which is relevant to Section 4. As discussed in Section 2.2.1, the thruster allocation matrix used by SITL to calculate state updates was updated from Equation (7) to Equation (4). See the later Section 3 for a description of the experimental testing that was performed to determine the yaw hydrodynamic model of the BROV2. All 6DOF drag models were updated to follow the generalized form of Equation 11. In order to generate simulation results that have practical usability, that yaw hydrodynamic hydrodynamic model updated yaw and pitch degrees of freedom identically, whereas coefficients in roll were scaled to around 86% of the former to match the proportionality suggested by [17]. Missing added mass parameters were introduced. Linear degree of freedom parameters or other untested parameters were updated to the values used in [16]. Vehicle mass was changed from 10.5 kg to 11.5 kg and the center of buoyancy with respect to the center of gravity was changed from $[0, 0, 0.15]$ m to $[0, 0, 0.02]$ m. First-order linear drag coefficients were introduced $\boldsymbol{B_{L,1:3}} = [4.03, 6.22, 5.18]$ Ns/m, as well as first-order angular drag coefficients $\boldsymbol{B_{L,4:6}} = [0.0, 0.0, 0.0]$ Ns/rad. Second-order drag was incorporated with $\boldsymbol{B_{NL,1:3}} = [18.18|u|, 21.66|v|, 36.99|w|]$ Ns^2/m and second-order angular drag coefficients with $\boldsymbol{B_{NL,4:6}} = [2.08|p|, 2.42|q|, 2.42|r|]$ Ns^2/rad^2. The linear and angular added mass vectors were also introduced, with values of $\boldsymbol{M_{A,1:3}} = [5.5, 12.7, 14.57]$ kg m^2 and $\boldsymbol{M_{A,4:6}} = [0.10, 0.12, 0.12]$ kg m^2/rad, respectively.

Other minor changes included a correction of the application of the transformation matrix of the inertial-to-body frame $\boldsymbol{J_1^{-1}}$ to $\boldsymbol{F_B}$; previously, this was applied to $\boldsymbol{r_B}$. Also, thruster force output calculations were updated to reference a lookup table, see Section 2 and Figure 6. In the next sections, the updates to the underlying hydrodynamic models of the autopilot and SITL produce accurate results useful for designing controllers that operate in real water settings.

2.2.5. Simulation Control Allocation

In Section 2.2.1 and Algorithm 2, the process of how desired forces and torques are converted to thruster activations was detailed. In this section, how the updated SITL code calculates state updates given these thruster inputs is explained. The simulation code receives the vector of forces from the eight thrusters and determines the body forces and torques by applying the thruster allocation matrix introduced in Section 2.2.1 and Equation (1). Unlike Equation (1), however, the matrix operation $\boldsymbol{T} : \boldsymbol{m} \mapsto \boldsymbol{\tau}$ is performed one thruster at a time using software array looping, and the inputs representing the motors are in units of PWMs not force since the simulation is reading the `RC_Output` channel. There is also a lookup table that complements the one used in Algorithm 2; where the one used in the algorithm used linear interpolation to find PWMs from forces, the

simulation code determines forces from PWMs. The 6DOF force and torque effect of each thruster is determined with column-wise multiplication with the thruster allocation matrix Equation (4). The force and torque values are then divided element-wise with the inertia coefficients $(\boldsymbol{M} + \boldsymbol{M_A})$ from Equations (A7) and (A8) of the hydrodynamic model described in Appendix B, to solve for intermediate values of linear and angular accelerations that are stored in a buffer. The simulation code then uses the velocity states of the given time step to perform drag force and torque calculations for each degree of freedom using the generalized model of Equation (11), where element-wise multiplication of the corresponding $\boldsymbol{B_L}$ from Equation (A10) scales first-order terms and $\boldsymbol{B_{NL}}$) from Equation (A11) scales second-order terms, to determine 6DOF forces and torques resulting from drag. These drag forces and torques are then element-wise divided with $(\boldsymbol{M} + \boldsymbol{M_A})$ as well to determine linear and angular accelerations which are added to the buffer. Buoyancy forces and torques and resulting accelerations are also calculated based on the virtual vehicle's depth and orientation, which are then element-wise divided with $(\boldsymbol{M} + \boldsymbol{M_A})$ and added to the buffer.

3. Open-Loop Experimental Testing and Hydrodynamic Characterization

In this section, a description of the open-loop yaw torque experimental testing that was performed to determine the yaw hydrodynamic coefficients for hybrid autopilot model and ArduSub SITL shall be covered. This round of testing was necessitated because initial trials of the autopilot described in the following sections produced inadequate performance due to discrepancies in the models available from other works and the actual BROV2 used in this work. The yaw degree of freedom isolated from the full hydrodynamic model in Appendix B is

$$N = (I_z + N_{\dot{r}})\dot{r} + N_r r + N_{r|r|} r^2 \tag{10}$$

where N is the external yaw torque acting on the BROV2, I_z is the moment of inertia in yaw, $N_{\dot{r}}\dot{r}$ is the added mass in yaw, N_r is the linear viscous damping in yaw, $N_{r|r|}$ is the quadratic damping in yaw, r is the vehicle's yaw velocity, and $\dot{r}$ is the vehicle's yaw acceleration.

Control gains were calculated using this strategy, initially based on hydrodynamic coefficients found in [16,38] for vehicles similar to the BROV2. Using the hydrodynamic coefficients found in the literature, initial results of real-world testing of the yaw step response produced large discrepancies between the settling times predicted by SITL, where actual settling times were much longer. Therefore, a preliminary round of water tank testing for system identification was performed that determined a more accurate model for simulations and the hybrid autopilot.

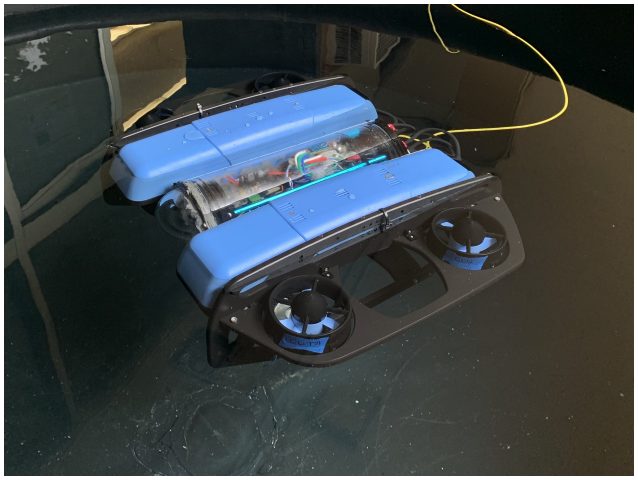

Figure 7. Real-world testing was performed in a small tank approximately two meters in diameter and a meter and a half in depth.

Open-loop testing of the yaw degree of freedom of the BROV2 Heavy was performed in a water tank (Figure 7). During this, six different yaw torques were applied to the

BROV2 and the resulting yaw velocities were measured, with at least five trials performed per torque tested. These tests were intended to observe inertia and drag behavior. The particular BROV2 Heavy used in these experiments had additional ballast in the form of metal washers attached to the front corners of the vehicle to level out its resting floating state. It was slightly positively buoyant, and before each trial was allowed to come to rest near the surface of the water, mostly submerged so that less than a quarter inch of the buoyancy floats was exposed. An experimenter also held the vehicle's tether above and out of the way, careful not to disturb the vehicle. At the start of each trial at time $t = 0$, a fixed yaw torque input command was sent to the vehicle and held constant. Data collection recorded yaw velocity r for times $t \geq 0$. Yaw torques were held constant for several seconds to give the velocity output ample time to stabilize before ceasing the trial. Each open-loop response had two distinct regions of the velocity output: an initial unsteady region while the vehicle was accelerating from rest and a steady-state region where the vehicle was no longer accelerating and maintained a constant velocity r_f. The unsteady region of the velocity output of the open-loop response was analyzed to determine inertia characteristics and the steady-state region to determine drag characteristics.

3.1. Viscous Drag Analysis

Given a constant yaw torque, the BROV2 starting from rest eventually reaches an equilibrium state after enough time has elapsed, where the drag torque balances with the thruster torque. While in equilibrium, acceleration largely ceases and a steady-state velocity is maintained. Equation (10) is reduced to Equation (A2) or

$$N(\dot{r} = 0) = N_r r + N_{r|r|} r^2. \tag{11}$$

Previous work in [38] tested a slightly smaller ROV similar to the BROV2 at low torques; in this work, a variety of test torques above this range were tested.

It was found that all open-loop responses at the various torques reached a steady state within two and a half seconds after the input torque was initiated. Because the experimental velocity data collected were noisy, a two-second interval spanning the range $2.5\,\text{s} \leq t \leq 4.5\,\text{s}$ was averaged to obtain the steady-state velocity. Mean and variance measurements of the interval were taken and are recorded in Figure 8. Using the experimental results, a least squares curve was fit to Equation (A2) and plotted with the open-loop experimental data as well as the data from [38]. It can be seen that there is good agreement between the fitted curve across the entire range of experimental torques and velocities. The resulting hydrodynamic coefficients for this curve found that the linear term N_r was zero and the quadratic term $N_{r|r|} = 2.42\,\text{Ns}^2/\text{rad}^2$.

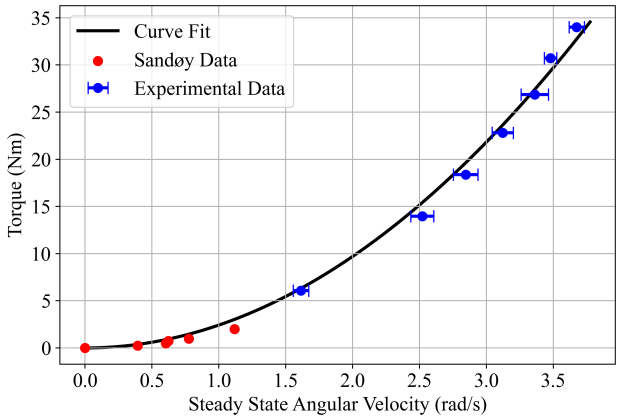

Figure 8. Experimental results from the drag analysis of the open-loop yaw testing; the complete drag function is actually symmetric about the origin. Error bars were calculated as two sample standard deviations to the left and right of the mean.

3.2. Inertia Analysis

Every BROV2 has unique inertia properties resulting from their payload, which may consist of custom modules or different sensors. In the case of the BROV2 that underwent open-loop testing, the ballasts stabilized its buoyancy altered its angular moments of inertia. Having determined that there is no linear term N_r, and constraining the value of $N_{r|r|}$ according to the analysis just described, the acceleration term $\dot{r}$ may be reintroduced to study the unsteady regime and determine the remaining inertia terms. If added mass $N_{\dot{r}}$ assumes the values suggested in [16,38], and the N controlled by the experimenter is also considered constrained, then the only remaining unknown quantity in Equation (10) is the moment of inertia I_{zz}.

Stated mathematically, Equation (10) reduces to the ordinary differential equation

$$\dot{r}(N) = \frac{N - N_{r|r|}r^2}{I_z + N_{\dot{r}}} \tag{12}$$

where $\dot{r}(N)$ is the vehicle's yaw acceleration given an applied input yaw torque N, $N_{r|r|}$ is the quadratic damping in yaw, I_{zz} is the moment of inertia in yaw, $N_{\dot{r}}\dot{r}$ is the added mass in yaw, and r is the measured output yaw velocity. Hence, for any of the six yaw torques tested Equation (12) has a one-parameter family of solutions

$$r(N) = f(t, I_z) \tag{13}$$

where r is yaw velocity, which is a function of time t, respective control torque N, and moment of inertia I_{zz}.

The solutions proposed above are applicable for constant torques N. Referencing Figure 9, it can be seen that in the moments immediately after the yaw torque input initiates there is a concave up region where the yaw torque is ramping up and has not yet reached its intended magnitude. Every trial has this initial thruster ramp-up time when yaw torque increases with time; the velocity at the time when constant torque is reached describes the left-hand boundary conditions for the added mass fitting. While a constant torque is maintained, velocity continues to increase with time. Eventually, in every trial the vehicle reaches an equilibrium steady state as defined in the previous viscous drag analysis. Therefore, the intervals and boundary conditions of Equation (13) are

$$t_0 \leq t \leq t_f, \quad r(t_0) = r_0, \quad r(t_f) = r_f. \tag{14}$$

where t is time, t_0 is the thruster ramp-up time while the vehicle is accelerating, t_f is the settling time when steady-state velocity is reached, r_0 is the velocity when constant torque is reached, and r_f is the steady-state velocity.

At the time t_0, where the curve transitions from concave up to concave down, full respective constant torque is reached; the corresponding velocity at the inflection point also gives the initial conditions $r(t_0) = r_0$ for the particular solution. Extracting t_f from the experimental data required other processing. To account for noise and the overshoot present at higher torques, a sliding window that calculated a moving average starting from the initial overshoot moving to the right was utilized which converged when the sliding window average absolute error from r_f was within a 5% tolerance, choosing the left-hand limit of the sliding window as t_f.

Now that the differential Equation (12) is well defined, with all other constants constrained, and the boundary conditions are established, the ordinary differential Equation (13) poses initial value problems for $r(N)$ dependent on the choice of I_{zz}.

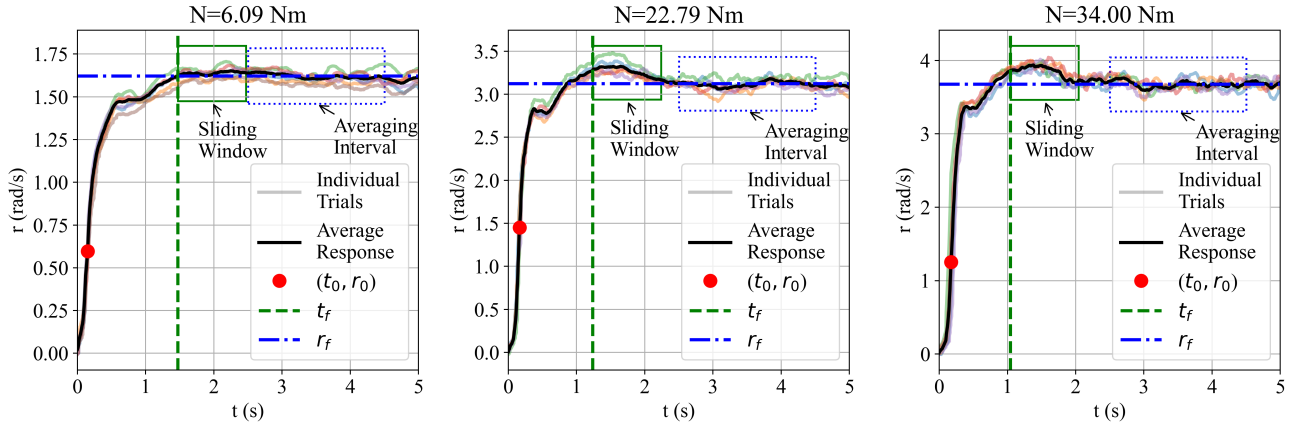

Figure 9. The yaw heading data contained in the dotted blue box is the averaging interval to determine steady-state yaw velocity r_f first, shown with the dash-dotted horizontal blue line. Then a sliding window moving average shown with a solid green box was used to determine settling time t_f, shown by the dashed vertical green line.

Essentially, Equation (12) is an initial value problem with the boundary conditions shown in Equation (14) having a one-parameter family of solutions Equation (13) dependent on I_{zz}. To trim down the search space of the infinite number of solutions, I_{zz} was assumed to be greater than the values suggested in [16,17,38] and less than a solid rectangle with the dimensions of the BROV2. A set of values within this range was iteratively tested against the actual unsteady response recorded in the experimental data using the root mean square error along the $t_0 \leq t \leq t_f$ interval. Results of this analysis are shown in Figure 10. The best agreement was found at $I_{zz} = 1.0\,\mathrm{kg\,m^2/rad}$.

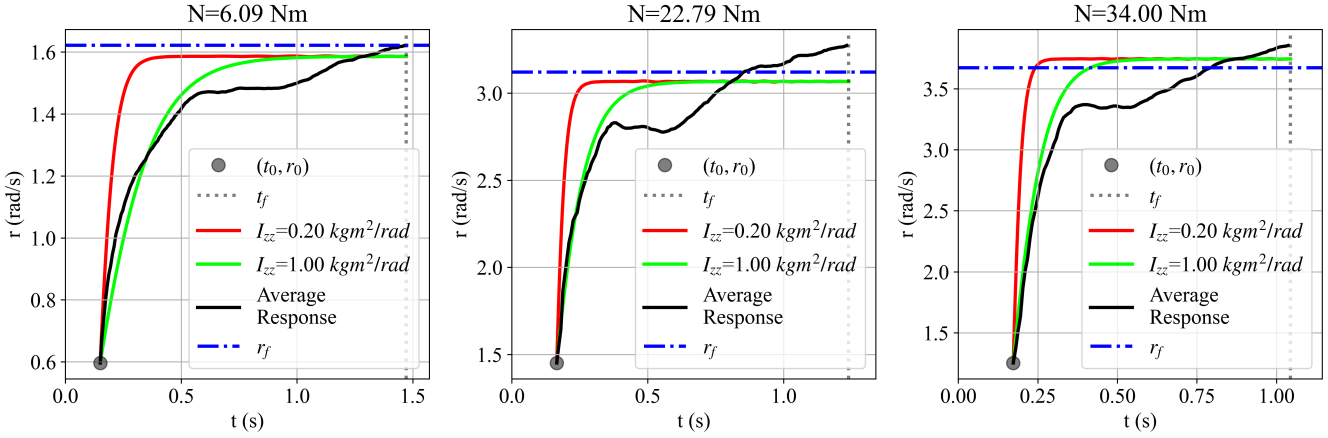

Figure 10. Comparison of the solutions of the lower moment of inertia in the yaw direction I_{zz} suggested by [16,17,38] alongside the chosen $I_{zz} = 1.0\,\mathrm{kg\,m^2/rad}$.

An alternative to the experimental strategy described here for estimating inertial properties is the moment of inertia calculator included with Gazebo Harmonic [39]. If users do not constrain inertia parameters themselves ahead of runtime, Gazebo can estimate the moment of inertia matrix of a robot while loading its 3D mesh model. The simulator does this with a type of finite element analysis and a density parameter given by the user. This method provides a convenient way for users to estimate inertia. However, this method does rely on the accuracy of the 3D models and the comprehensiveness of the configuration file containing joints, links, and density information. In other words, this type of estimation is reliable and convenient for simple homogeneous shapes, but becomes more cumbersome

when the shapes get more complex and consist of multiple materials. This work has opted not to use Gazebo, but encouraged readers to explore their options.

4. Designing Autopilot Performance

Appendix B provides a methodology for calculating the gains of a critically damped closed-loop linear system. In this section, the specifications of the PD control law of this particular ROV are described. Because the proposed error feedback control only applies to angular states, the generic variable state variables γ_d and γ used to derive control gains in Appendix B are replaced with the specific state vectors $\boldsymbol{\eta_{2d}}$ and $\boldsymbol{\eta_2}$. Desired angular states are designated in the inertial frame and passed into the autopilot as $\boldsymbol{\eta_{2d}}$. The rate of change of the desired state $\dot{\boldsymbol{\eta}}_{\mathbf{2d}}$ is calculated as a moving average from $\boldsymbol{\eta_{2d}}$, because the pilot input for the desired state is provided in real time rather than as a pre-defined trajectory. The FMU tracks the vehicle's velocity in the body frame $\boldsymbol{\nu_2}$ and this is transformed into the inertial frame with the transformation matrix

$$\boldsymbol{J}_{\boldsymbol{2},\boldsymbol{\eta_2}} = \begin{pmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi)/\cos(\theta) & \cos(\phi)/\cos(\theta) \end{pmatrix}, \tag{15}$$

by left multiplying to result in

$$\dot{\boldsymbol{\eta}}_{\mathbf{2}} = \boldsymbol{J}_{\boldsymbol{2},\boldsymbol{\eta_2}}\boldsymbol{\nu_2}. \tag{16}$$

These velocity state vectors are compared to determine the derivative error

$$\dot{\tilde{\boldsymbol{\eta}}}_{\mathbf{2}} = \dot{\boldsymbol{\eta}}_{\mathbf{2d}} - \dot{\boldsymbol{\eta}}_{\mathbf{2}}. \tag{17}$$

Proportional gains $\boldsymbol{k_p}$ and derivative gains $\boldsymbol{k_d}$ are calculated in Equation (A17) and Equation (A18), respectively, as diagonal matrices and are applied to the errors through the control law as seen in Equation (A5). For the proposed autopilot for AR pilots, only angular torques are generated; hence, the specific control law is

$$\boldsymbol{\tau}_{4:6} = \boldsymbol{k_p}\boldsymbol{J}^{-1}_{\boldsymbol{2},\boldsymbol{\eta_2}}\tilde{\boldsymbol{\eta}}_{\mathbf{2}} + \boldsymbol{k_d}\boldsymbol{J}^{-1}_{\boldsymbol{2},\boldsymbol{\eta_2}}\dot{\tilde{\boldsymbol{\eta}}}_{\mathbf{2}} + \left(\boldsymbol{r_B} \times \boldsymbol{J}^{-1}_{\boldsymbol{1},\boldsymbol{\eta_{2d}}}\boldsymbol{F_B}\right), \tag{18}$$

where the inertial-to-body frame transformation matrix of angular components in the current angular state is

$$\boldsymbol{J}^{-1}_{\boldsymbol{2},\boldsymbol{\eta_2}} = \begin{pmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \sin(\phi)\cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{pmatrix}, \tag{19}$$

where $\boldsymbol{r_B}$ is the constant vector from the center of gravity to the center of buoyancy in the body frame, $\boldsymbol{F_B}$ is the constant buoyancy force of the completely submerged vehicle in the inertial frame, and $\boldsymbol{J}^{-1}_{\boldsymbol{1},\boldsymbol{\eta_{2d}}}$ is the inertial-to-body frame transformation matrix for linear components at the *desired* angular state (not shown here in its entirety due to space limitations, please see [32] for the full form).

The cross-product term in Equation (18) not present in Equation (A5) is a feed-forward term that accounts for the hydrostatic righting moment $\boldsymbol{\tau_{b,4:6}}$. Without the feed-forward term, the angular control law converges to a steady-state with an offset error resulting from the unaccounted for torque bias. In Appendix B, it is claimed that buoyancy effects can be ignored during calculations of PD gains. This is allowable because when the desired state is achieved, i.e., $\boldsymbol{\eta_2} = \boldsymbol{\eta_{2d}}$, the feed-forward term exactly cancels out the hydrostatic righting moment such that

$$\boldsymbol{\tau_{b,4:6}} = \left(\boldsymbol{r_B} \times \boldsymbol{J}^{-1}_{\boldsymbol{1},\boldsymbol{\eta_2}}\boldsymbol{F_B}\right) = \left(\boldsymbol{r_B} \times \boldsymbol{J}^{-1}_{\boldsymbol{1},\boldsymbol{\eta_{2d}}}\boldsymbol{F_B}\right). \tag{20}$$

Thus, when equating the control forces from Equation (18) with the hydrodynamic model Equation (A6) while calculating gains, the formula reduces to the same form as Equation (A13), meaning that the calculations that follow in Equations (A17) and (A18) are unchanged.

5. Controller Tuning Methodology

In this section, the procedure followed for tuning one degree of freedom, specifically the yaw direction, shall be reviewed. The equation for control forces in the yaw direction is

$$N_C = k_p(\omega_d)[\psi_d - \psi] + k_d(\omega_d)[\dot{\psi}_d - \dot{\psi}], \tag{21}$$

which is the third element of the vector represented in Equation (18). Notice that there are no buoyant torques in the yaw direction, which simplifies calculations. In the case of control forces in the other directions, an additional feed-forward term would appear in K_C and M_C, as seen in Equation (20).

To give a high-level overview of how all the parts covered previously go together, Figure 11 shows a block diagram of the information flow throughout the portion of the hybrid autopilot that controls angular DOF. "Pilot Input" during the tests discussed in this section originated from prerecorded csvs outputting desired angular states $\boldsymbol{\eta_{2d}}$. These desired states are input into the "Control Law", which uses gains calculated in Appendix B and produces the desired angular body torques $\boldsymbol{\tau_{4:6}}$. These torques are parsed with the "Thruster Allocation" block, implementing logic described in Section 2.2.1 and Algorithm 2 to determine thruster forces $\boldsymbol{m}$ that achieve the desired body torques. The collective effect of the eight thruster forces drive the "State/SITL" block, outputting the angular orientations $\boldsymbol{\eta_2}$ of the actual/virtual vehicle, respectively. Ultimately, the angular states are fed back to the "Control Law" block, establishing the closed loop of the angular controller.

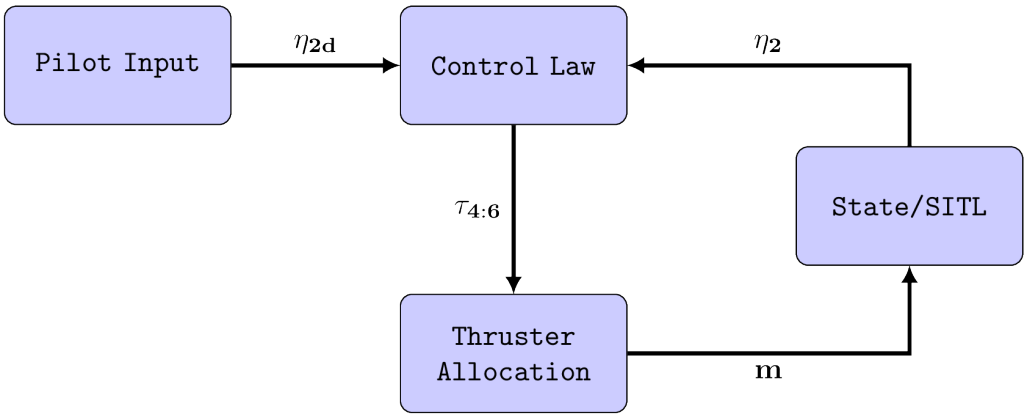

Figure 11. A block diagram of the information flow throughout the closed-loop portion of the hybrid autopilot that controls the angular degrees of freedom; arrows are labeled with the important vectors that pass between the nodes.

In Appendix B, the BROV2 is approximated as an LTI system in order to calculate control gains as a function of desired closed-loop natural frequency ω_d, but in actuality drag behaves nonlinearly, scaling quadratically with velocity as explained in Section 2.2. Therefore, an effective linear drag value $\check{N}_r$ was determined with trim conditions, which was used to set the linear PD controller. The step response of the nonlinear system under Equation (21) was tested in SITL to verify behavior similar to a critically damped system. It was observed that the step response exhibited underdamped behavior, where the level of oscillation was inversely proportional to ω_d^2, so the controller was modified to include an additional damping term inversely proportional to ω_d^2. With the additional damping term, the step response achieved critically or close-to-critically damped behavior for a wide range of desired closed-loop natural frequencies. Using the yaw PD control law tuned in SITL, the closed-loop step response was determined through water tank testing,

demonstrating agreement with simulation results. Following the step response tuning, a series of frequency response tests documented in Appendix C were performed in SITL for low ($\omega_d = 2.0\,\text{rad/s}$), medium ($\omega_d = 3.0\,\text{rad/s}$), and high ($\omega_d = 4.0\,\text{rad/s}$) desired closed-loop natural frequency settings to produce a Bode plot. The final sequence of testing generalized the yaw control law across all angular degrees of freedom and activated the open-loop linear controls to test the performance of the total hybrid autopilot in STIL given a simulated dive involving all six degrees of freedom.

5.1. Determining Trim Conditions

While the standard technique to linearize the drag would be a first-order Taylor-series approximation at the center of the expected operational range of velocities, this would be unusable because the line would not have any slope. The idea was to choose a line trimmed about the origin so that there was no y-intercept allowing the slope to be used as $\check{N}_r$. This trimmed line should approximate the expected drag torques as well as possible over the expected range of velocities. Considering that most ROV movements are typically low-amplitude minor adjustments, the value of $\check{N}_r = 5.97\,\text{Nms/rad}$ was selected. The linear fit of this approximation compared to the nonlinear yaw model determined earlier in Section 3 is shown in Figure 12.

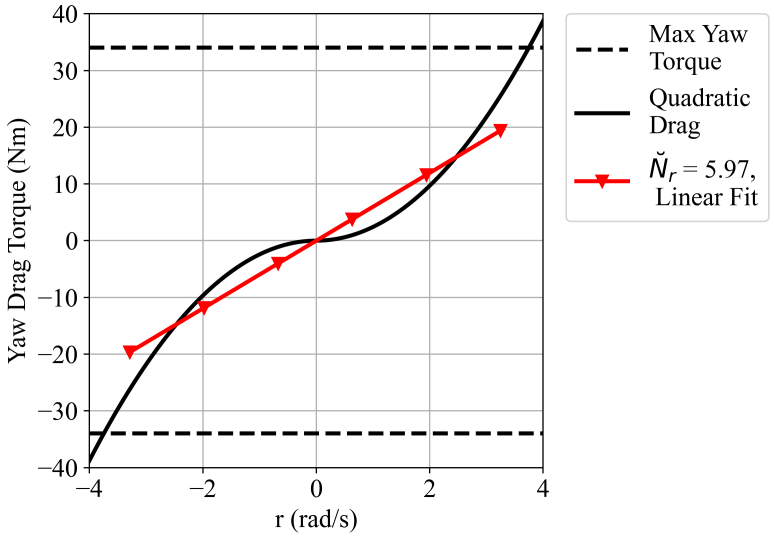

Figure 12. Nonlinear drag torque vs. velocity along with the trimmed linear fit; the line designated by $\check{N}_r = 5.97\,\text{Nms/rad}$ fit well in the possible yaw torque range.

5.2. Additional Damping Term to Correct for Nonlinear Behavior

With the effective linear damping approximation selected, step response tests were performed and compared to the theoretical LTI system step response to assess how the controller handled a step input to rotate from a 0° heading to a heading of 90° and hold the position. A simple step input was generated such that at $t < 0$, $\psi_d = 0$ and at $t \geq 1$, $\psi_d = 90°$, with all other inputs null and put into comma-separated value (csv) file format. The step input was fed through a ROS `csv_reader` node into a separate ROS `control_law` node, which monitored vehicle state and calculated the force outputs according to the PD control law as described in Section 4. The resulting step responses were recorded to csv files with a ROS `logger` node. This was then graphed against the theoretical step response.

The theoretical step response was calculated using the transfer function of the control law in the yaw direction which can be derived from Equation (A13) by taking the Laplace transform and rearranging. The resulting transfer function is

$$H_L = \frac{\Psi(s)}{\Psi_d(s)} = \frac{k_p(\omega_d) + sk_d(\omega_d)}{s^2(I_{zz} + N_{\dot{r}}) + s(\check{N}_r + k_d(\omega_d)) + k_p(\omega_d)} \tag{22}$$

where H_L is the closed-loop LTI transfer function, $\Psi(s)$ is the Laplace transform of yaw heading, $\Psi_d(s)$ is the Laplace transform of the desired yaw heading, k_p is proportional gain determined by the ω_d setting, k_d is derivative gain determined by the ω_d setting, and $\check{N}_r$ is the chosen characteristic linear damping. Setting $\psi_d = H(0)$, the Heaviside step function becomes $\Psi_d(s) = 1/s$ and performing the inverse Laplace transform on Equation (22), a function for $\psi(t)$ is obtained, which was used to generate the theoretical step response for various gains.

According to the linear function of the LTI system, the chosen ω_d and resulting k_p and k_d gains theoretically produce a critically damped response, which are shown as the dashed lines in Figure 13. Of note is the theoretical low $\omega_d = 2.0\,\text{rad/s}$ step response; there is an initial undershoot due to $k_d < 0$ causing the system to become a nonminimum-phase system [40]. The actual response of the nonlinear system, shown with solid lines, produced underdamped step responses, that steadily improved with increasing ω_d, but did not reach critically damped behavior. As the goal is to attain critically damped behavior for the entire range of ω_d, the solution is to increase the damping via the artificial damping enacted by k_d. Observing how low ω_d needs more damping than higher ω_d, the amount of additional damping necessary is inversely related to ω_d. Hence, a small δk_d term was added to the original calculated k_d, which scales inversely with ω_d, with the form

$$\delta k_d = \frac{\kappa}{\omega_d^2} \tag{23}$$

where κ is a constant of proportionality and ω_d is the desired bandwidth that k_d was originally calculated for. The ω_d term is squared because of the relatively low values of ω_d for this system. The new control law becomes

$$N_C = k_p(\omega_d)[\psi_d - \psi] + [k_d(\omega_d) + \delta k_d(\omega_d)][\dot{\psi}_d - \dot{\psi}], \tag{24}$$

A range of various constants of proportionality were tested across a wide range of ω_d. It was found that the value for the constant of proportionality that nullified oscillations best resulting in a critically damped or nearly critically damped step response was $\kappa = 20.0$. The resulting step response with the added δk_d term is shown in Figure 13 with dash–dotted lines.

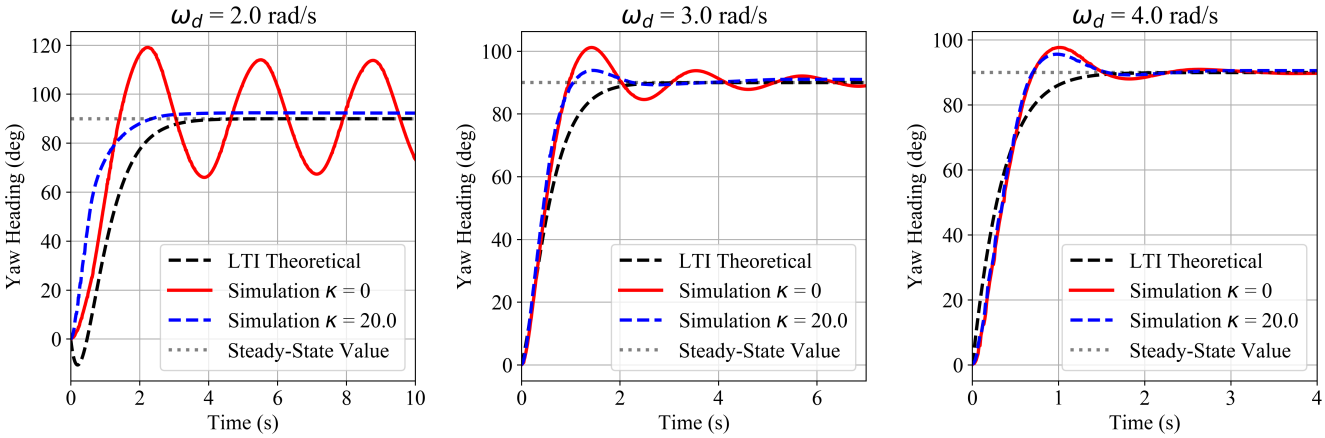

Figure 13. Step response testing in the nonlinear simulation; a tolerance of ±5% was needed to enable convergence because the thruster deadzone prevented slight adjustments when very close to the steady-state value of 90°.

It can be seen that with this additional term, the controller creates the desired behavior for the nonlinear vehicle system. At $\omega_d = 2.0\,\text{rad/s}$, the $k_d + \delta k_d > 0$ modified derivative gain addresses the initial undershoot, and is large enough to prevent any overshoot having an even better settling time than the LTI theoretical settling time. For the $\omega_d = 3.0\,\text{rad/s}$

and $\omega_d = 4.0\,\text{rad/s}$ cases, there are slight overshoots, but they are acceptable for practical purposes.

A range of experimental step response settling times of the nonlinear system with δk_d were tested in 0.25 rad/s increments in the range of $1.5\,\text{rad/s} \leq \omega_d \leq 5.0\,\text{rad/s}$ and compared with their theoretical linear settling times shown in Figure 14. In the case of $\omega_d = 3.0\,\text{rad/s}$, the overshoot falls within the $\pm 5\%$ tolerance and may be neglected. In the case of $\omega_d = 4.0\,\text{rad/s}$, the initial overshoot exceeds the tolerance, but after one oscillation the step response converges; this is why there is a jump in the settling times for $\omega_d \geq 3.5$.

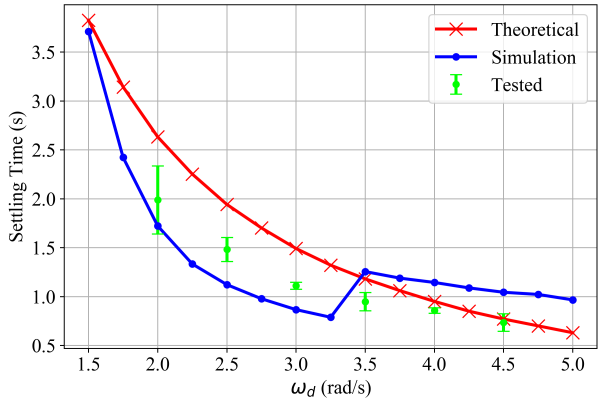

Figure 14. Settling time was defined as the amount of time it took the vehicle given a step input to stabilize within $\pm 5\%$ of the steady-state value of 90° yaw heading; results from the real-world test show error bars of two sample standard deviations.

Also shown in Figure 14 are the settling times for the water tank step response tests at $\omega_d = \{2.0, 2.5, 3.5, 4.0, 4.5\}\,\text{rad/s}$ using the tuned yaw controller of Equation (24) with $\kappa = 20.0$. Every ω_d was tested seven times each during water tank testing. Like the open-loop tests described in Section 3, each step response test was performed with the BROV2 nearly completely submerged, floating near the surface of the water, and initiated once while the BROV2 was totally at rest. Figure 14 demonstrates that the SITL simulation better approximates the actual response of the BROV2 than the LTI model, because the simulation predicts faster settling times at lower ω_d and the simulation results mirror this pattern. Settling times progress in a continuous fashion because they do not oscillate before converging; yet, there is a discontinuity in the simulation settling because the SITL model, while taking into account hydrodynamic nonlinearity, does not account for thruster unsteady behavior and other physical complexities beyond the scope of this work. Tuned step responses of $\omega_d = \{2.0, 3.0, 4.0\}\,\text{rad/s}$ of the water tank experiments are shown with their accompanying simulation step responses in Figure 15. The $\omega_d = 2.0\,\text{rad/s}$ experimental response is nearly identical to the simulation; however, as ω_d increases the responses diverge slightly. This is in part due to the thruster ramp up time causing lag and otherwise resulting from imperfections in the model. It appears that the SITL model predicts sharper overshoots and faster convergence, which may be due to un-modeled higher dimensional drag terms.

Despite these shortcomings, the updated simulation model is a vast improvement over the old simulation model, as can be seen in Figure 16. The old hydrodynamic model predicts a much slower step response than what was actually observed during the water tank experiments. This is due to the numerous errors outlined in Section 2.2.4. Further quantification is presented in Table 1, where the average normalized error of the simulated step responses of the old and new SITL models are compared to what was observed experimentally across a range of ω_d settings. Average normalized error, $\bar{e}$, is defined as

$$\bar{e} = \frac{1}{T}\int_0^T \frac{|e|}{\psi} dt, \tag{25}$$

where T is the length of time that the averaging interval is measured, $|e|$ is the absolute error between the simulated and experimental heading, ψ is the experimental heading, and t represents time, of which both $|e|$ and ψ are functions. The columns labeled $\bar{e}_{old}$ and $\bar{e}_{new}$ characterize the error of the old and new SITL models, respectively. It can be seen in Table 1 that the old model predicts significantly more errors than the our updated model. Figure 16 also displays that the updated model captures the higher order behavior well. This proves that the methodologies for system identification described in Section 3 and controller tuning described in Section 5 are effective tools for designing BROV2 performance.

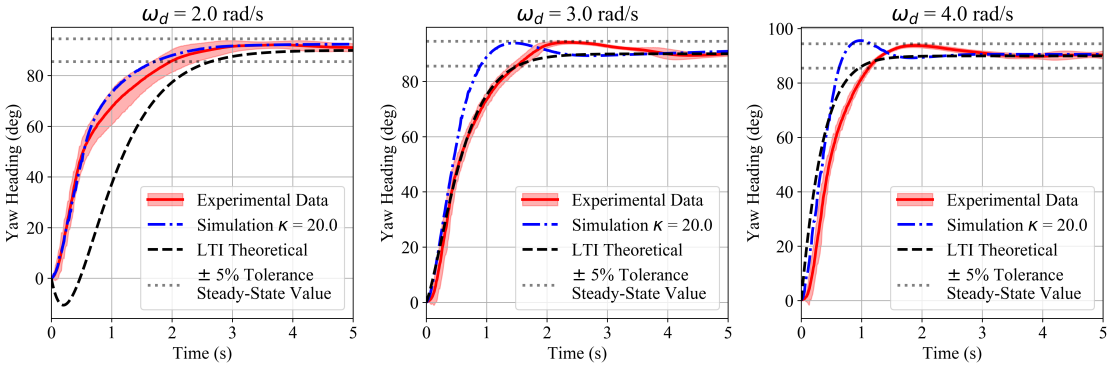

Figure 15. Experimental data from water tank step response testing are shown with a shaded region, where each vertical slice of time represents two sample standard deviations around the average response of seven trials; also plotted are simulation results and linear approximation predictions.

Table 1. Average normalized error of old and new SITL models with respect to experimental step responses for five-second window.

ω_d (rad/s)	$\bar{e}_{old}$ (deg)	$\bar{e}_{new}$ (deg)
2.0	63.80	3.66
2.5	45.53	8.26
3.0	37.53	5.20
3.5	34.18	4.32
4.0	26.36	5.27
4.5	24.60	4.10

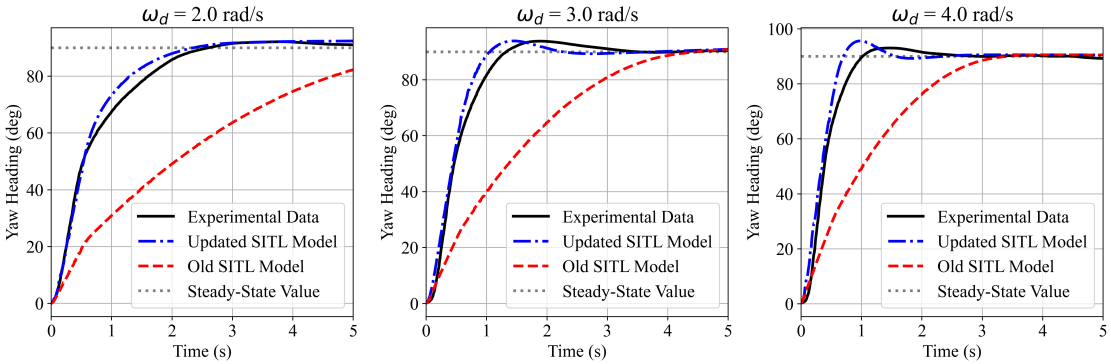

Figure 16. Step responses comparing simulation results of the original software-in-the-loop (SITL) hydrodynamic model, the new SITL model from this work, and the water tank results.

5.3. Simulated Pilot Testing

The final set of testing used a simulated pilot dive containing all 6DOF provided by collaborators from the UF. Like the previous tests, inputs were parsed by a `csv_reader` ROS node and logged with a `logger` node. What was different for this test was that the

`control_law` node was extended to include all three angular degrees of freedom, as shown in Equation (18). The equations for K_C and M_C are identical to Equation (21) except for additional feed-forward buoyancy terms that were discussed in Section 4. The presence of the feed-forward terms directly counteract the hydrostatic righting moment at the desired orientation, preventing persistent steady-state error due to false convergence. As the simulation and control law models for roll and pitch are identical to yaw, nonminimum phase behavior at low ω_d and underdamped behavior were observed, so $\delta k_d = \kappa/\omega_d^2$, with $\kappa = 20$, was also applied.

Linear controls in the surge, sway, and heave directions were treated like direct inputs from a joystick and did not have error feedback control laws associated with them. This meant that the magnitude of a linear signal required the percentage of the associated elements of $\boldsymbol{\tau_{max,1:3}}$, as discussed in Section 2.2. The following results of the orientation trajectory are shown row-wise for $\omega_d = \{2.0, 3.0, 4.0\}$ rad/s in Figure 17. For every increase of ω_d, there are marginal improvements in satisfying the desired trajectories, most apparent in the yaw column. To represent these improvements more clearly, Figure 18 displays the integral of the absolute error between the desired and actual trajectories. Of note is how the pitch error does not improve significantly by increasing ω_d from 2.0 rad/s to 3.0 rad/s. This could be a result of managing buoyancy stability while carrying out challenging roll commands simultaneously. In general, it can be seen that error decreases with increasing ω_d.

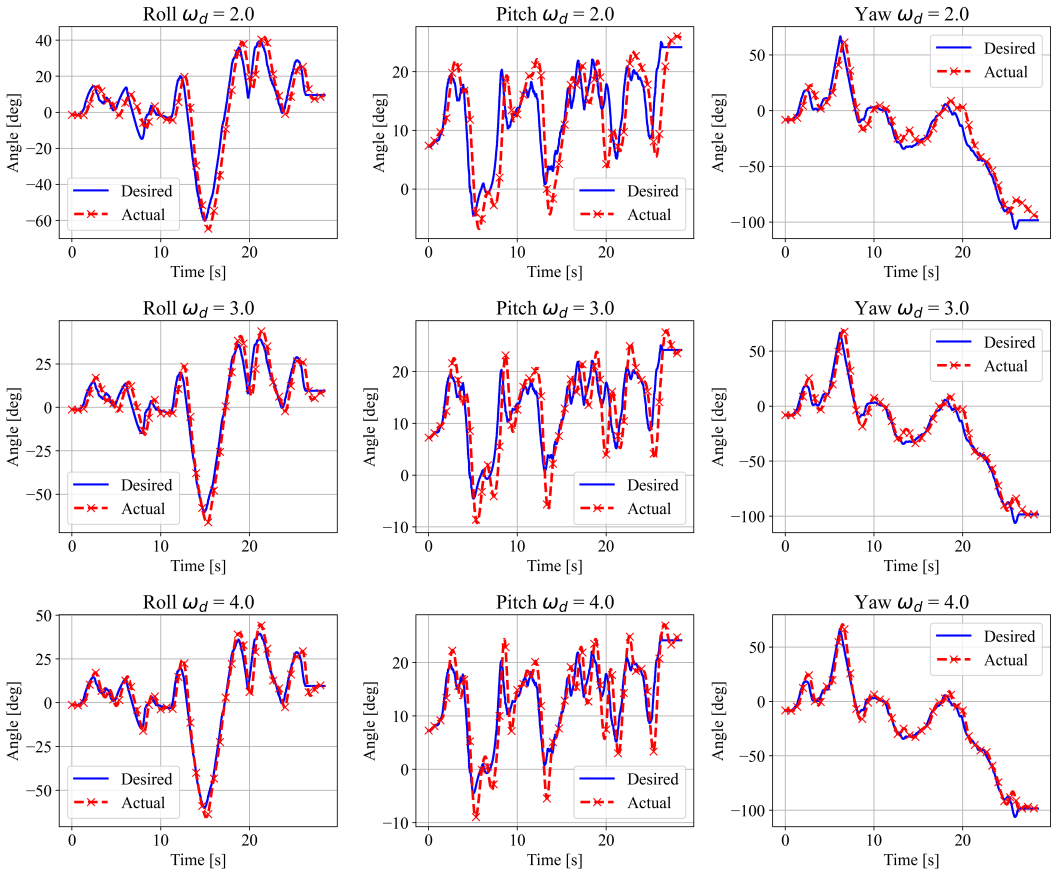

Figure 17. Each row graphs the desired and actual trajectories of different ω_d; every column displays a different angular degree of freedom.

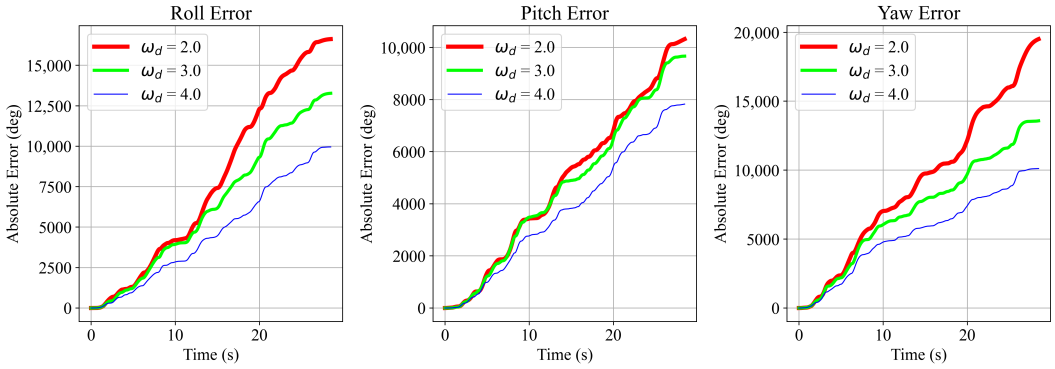

Figure 18. Shown are the integrals of absolute error with respect to time at the different ω_d. The same general pattern of error is present for all ω_d, but there are decreases in magnitude with increases in ω_d.

Figure 19 shows the linear open-loop command signals of the simulated dive in the top row, and graphs representative of the energy costs of the angular controllers are in the lower row. The values for the energy costs are summed and normalized to the PWM signal from their respective controllers or

$$\sum_{i=1}^{n}\left(\frac{|PWM - 1500|}{400}\right) \tag{26}$$

where PWM is the control signal sent for the respective degree of freedom, i is the index of the autopilot update, and n is the total autopilot iterations of the dive. An absolute value is taken in the numerator to group the magnitudes of positive and negative control signals because the PWM signals are centered on 1500 µs. This value is normalized with the maximum range ±400 µs of control signals. It can be seen that energy costs increase with increasing ω_d.

As a whole, the hybrid autopilot performed very well in tracking the angular states prescribed by the simulated pilot dive. A noticeable improvement in pitch error in ω_d, changing from 3.0 rad/s to 4.0 rad/s, indicates the nuance in selecting appropriate hybrid autopilot settings that the pilot must grow accustomed to. Energy costs are also another consideration that should be considered by pilots when planning their missions. Broadly speaking, the BROV2's performance will depend in equal parts on the choice of ω_d and the pilot's handling via the AR/VR interface.

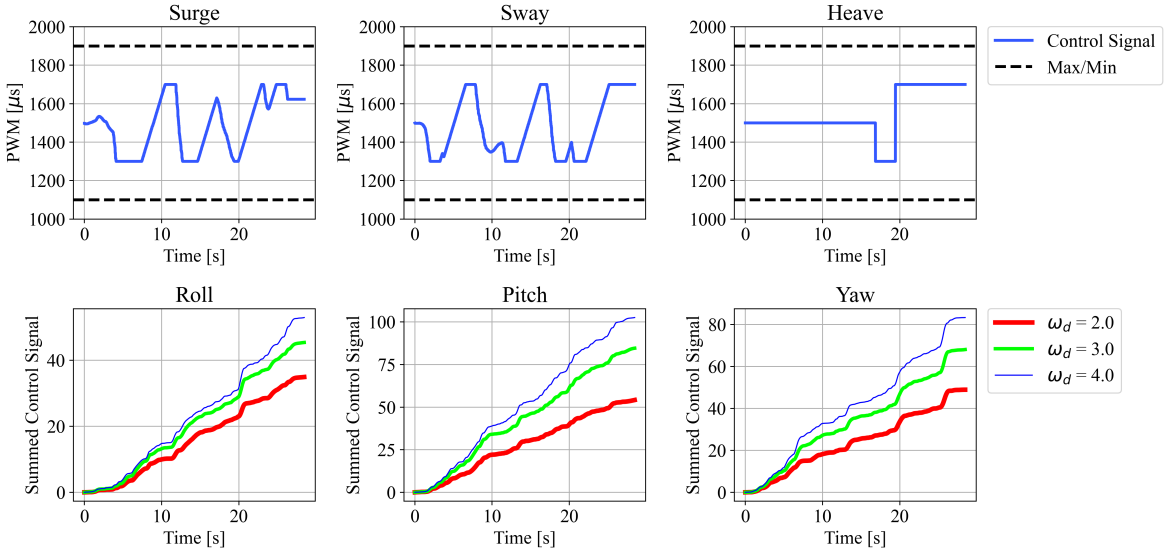

Figure 19. First row shows the open-loop control PWMs of linear degrees of freedom with respect to time, and second row shows summed PWMs of angular degrees of freedom defined in Equation (26), correlating energy costs of different ω_d.

6. Conclusions and Future Work

The SITL simulation suite offered by ArduSub is a potentially useful tool for designing systems for the BROV2 Heavy, because it allows for rapid testing without requiring actual water or even the vehicle itself. The main advantage of using ArduSub SITL over other simulations is that the exact same programs, like those written in the ROS for example, and customized firmware can be used both in SITL and in the field. Software frameworks which implement control can be iterated on at a much higher rate with SITL, be in a much better state by the time of the first field test, and have greater improvements between subsequent deployments. However, in its current configuration available to the public, these potential advantages cannot be capitalized on because the vehicle model and hydrodynamic model are so inaccurate. Proper control applications also require a quantitative model that can work with physical quantities like force and torque; the current manual control is a heuristic model which caters to more casual users. In order to access the full potential of ArduSub for our particular project, a few steps were taken. First, on the firmware side of ArduSub, a rework of the manual control flight mode was implemented which incorporated inverse thruster allocation, determined from BROV2 Heavy vehicle specifications found in the available literature, to enable precise control applications by allowing requested forces and torques to be interpreted by the vehicle. Second, open-loop experimental testing was performed to build a reliable model for the specific configuration of the BROV2 that the hybrid autopilot was being developed for. Third, on the simulation side of ArduSub SITL, the thruster allocation complementing the firmware's inverse thruster allocation that updates the vehicle state with respect to thruster inputs was incorporated, as well as a new hydrodynamic model including added mass parameters, updated drag calculations, and corrections to physics.

The development process of a hybrid autopilot for assisting AR ROV pilots was started on this new and improved ArduSub platform. A PD control law for critically damped yaw trajectory following was derived using an effective linear drag $\check{N}_r = 5.97\,\mathrm{Nms/rad}$, which approximated the nonlinear drag. Through thee ROS, the error feedback control mechanism was implemented and could be set to different sensitivities with the selection ω_d. During simulation step response testing, it was found that the nonlinear system behaved in an underdamped manner and was corrected with the addition of a δk_d term creating nearly critically damped response. It was found in the water tank tests that the controller tuned in SITL performed as expected in the real world and demonstrated the improved ArduSub SITL's value as a design tool. In continued simulation testing, recorded in Appendix C, a linear regime of the closed-loop yaw frequency responses occurring withinn gains of $0.71 \leq |H| \leq 1.0$ was identified, providing a working definition for the control bandwidth of the nonlinear system.

Testing of the hybrid autopilot proceeded with extending the tuned control law for yaw to roll and pitch terms with additional feed-forward terms. The control laws for all three angular degrees of freedom were implemented simultaneously and linear controls were passed through directly without error feedback. A dive was simulated with prerecorded simulated inputs containing low-frequency angular trajectories and open-loop linear control followed the desired states well. Simulated dive testing highlighted the challenges that pilots are bound to face in managing autopilot settings, energy costs, and mastering the AR/VR pilot's interface. It is hard to draw definite conclusions from the simulation data alone because ultimately the future human pilots' experience of the hybrid autopilot will be the deciding factor in performance.

The immediate next steps for this project are having live AR pilots interact with the hybrid autopilot in SITL and real water dives. Future work in gathering experimental data to model roll and pitch hydrodynamics of the BROV2 is also planned. Conducting tests similar to the ones in this study and tow tank tests in the real world could assist in this process of refining the hydrodynamic model used to calculate PD gains. Improving the accuracy of the hydrodynamic model will reduce errors and speed up convergence in practical applications and increase the value of SITL testing and improve the hybrid

autopilot to better assist pilots. Some updates to the SITL model are unique to our particular vehicle, but many fundamental errors were corrected in this work. Readers are encouraged to adopt our methodologies and use our results as reference points for their own vehicles. Moving forward, as our ArduSub SITL model of the BROV2 Heavy continually improves, exploration of complicated topics like motor saturation, other nonlinear strategies, and fully autonomous operations are more worthwhile to conduct in simulation and now have a way to be implemented thanks to our updates to ArduSub that unlock quantitative control.

Author Contributions: Conceptualization, P.N. and M.K.; methodology, P.N. and M.K.; software, P.N.; validation, P.N.; formal analysis, P.N. and M.K.; investigation, P.N.; resources, P.N.; data curation, P.N.; writing—original draft preparation, P.N.; writing—review and editing, P.N. and M.K.; visualization, P.N.; supervision, M.K.; project administration, M.K.; funding acquisition, M.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Science Foundation (NSF) grant number 2128924.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: Thanks to University of Florida (UF) collaborators Pengxiang Xia and Eric Jing Du for the test trajectories simulating an AR piloted dive, and for consultation on controller strategy. ChatGPT and Github Copilot assisted with coding tasks associated with this work.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Angular Drag Coefficients

The simulation code uses a classic drag model for calculating the drag forces or torques acting on the vehicle, which for any single degree of freedom is

$$\tau_d = -\frac{1}{2} V|V| C_d S \rho, \tag{A1}$$

where τ_d is the drag force or torque, V is the signed linear or angular velocity to preserve the direction of movement, C_d is the drag coefficient, S is the wetted area, and ρ is the density of the surrounding fluid. The original code approximated the vehicle as a sphere to determine S, chose arbitrary C_d, and applied Equation (A1) to calculate instances of linear drag. However, for drag torque the value of τ_d was then multiplied by an approximate equivalent sphere radius. Depending on if the drag being calculated is a force or torque, C_d will be dimensionless for force or have units of meters for torque; in the case of angular drag torques, by multiplying τ_d by a radius an additional length dimension multiplies the torque, producing major errors.

To fix the erroneous implementation and improve accuracy, a different model with values backed by the research literature is proposed to replace the Equation (A1) model. Research conducted by Sandøy in [38] characterized a smaller version of the BlueROV named uDrone. The uDrone had only five thrusters and could not be controlled in the roll and pitch degrees of freedom, so only the yaw direction was analyzed. Researchers had exerted constant moments at various torques and recorded the angular velocity of the vehicle at those points; then, a least squares curve fit was used to find

$$N_D = N_r r + N_{r|r|} r|r| \tag{A2}$$

where N_D is the viscous torque due to drag in the yaw direction, N_r is the linear damping coefficient, $N_{r|r|}$ is the quadratic damping coefficient, and r is the signed yaw velocity. The yaw degree of freedom is also presented here because it is the focus of this study; later, the testing and tuning of the yaw autopilot control shall be discussed. Before delving

into the autopilot development, determination of the yaw hydrodynamic model through experimental water tank tests shall be covered in the following section.

Appendix B. Calculating Control Gains for a Critically Damped System

A simple PD control law was carried out via a ROS layer. Given a generic state variable γ containing pose and γ_d containing desired pose, the difference between the two

$$\tilde{\gamma} = \gamma_d - \gamma, \tag{A3}$$

is the proportional error. There is also an associated rate over time

$$\dot{\tilde{\gamma}} = \frac{d\tilde{\gamma}}{dt}, \tag{A4}$$

known as the derivative error. Using the error terms, a corrective force $\boldsymbol{\tau}$ in the body-fixed frame is calculated by applying proportional gain $\boldsymbol{k_p}$ to $\tilde{\gamma}$ and derivative gain $\boldsymbol{k_d}$ to $\dot{\tilde{\gamma}}$,

$$\boldsymbol{\tau} = \boldsymbol{k_p}\tilde{\gamma} + \gamma_d\dot{\tilde{\gamma}}. \tag{A5}$$

The full 6DOF hydrodynamic model relates the system inertia matrix $\boldsymbol{M}$, added mass matrix $\boldsymbol{M_A}$, and acceleration $\ddot{\gamma}$, as well as the damping matrix $\boldsymbol{B}$ and velocity $\dot{\gamma}$, to produce the net external force

$$\boldsymbol{\tau} = (\boldsymbol{M} + \boldsymbol{M_A})\ddot{\gamma} + \boldsymbol{B}\dot{\gamma} + \boldsymbol{\tau_B}. \tag{A6}$$

The additional term $\boldsymbol{\tau_B}$ is forces/torques due to buoyancy that will not effect the PD gain calculation, which is addressed in Section 4.

Simplified models for the vehicle parameters are used in which only diagonal elements are nonzero. The system inertia matrix takes the form

$$\boldsymbol{M} = \boldsymbol{diag}(m, m, m, I_{xx}, I_{yy}, I_{zz}) \tag{A7}$$

where m is mass of the vehicle and I_{jj} is the moment of inertia about the j-axis. The added mass matrix also takes a simplified diagonal form

$$\boldsymbol{M_A} = \boldsymbol{diag}(X_{\dot{u}}, Y_{\dot{v}}, Z_{\dot{w}}, K_{\dot{p}}, M_{\dot{q}}, N_{\dot{r}}). \tag{A8}$$

The damping matrix consists of

$$\boldsymbol{B} = \boldsymbol{B_L} + \boldsymbol{B_{NL}} \tag{A9}$$

where the linear terms are grouped in

$$\boldsymbol{B_L} = \boldsymbol{diag}(X_u, Y_v, Z_w, K_p, M_q, N_r) \tag{A10}$$

and the nonlinear terms are grouped in

$$\boldsymbol{B_{NL}} = \boldsymbol{diag}(X_{u|u|}|u|, Y_{v|v|}|v|, Z_{w|w|}|w|, K_{p|p|}|p|, M_{q|q|}|q|, N_{r|r|}|r|). \tag{A11}$$

The nonlinear terms present in Equation (A11) are functions of the absolute values of the corresponding velocities. To proceed with calculating control gains, it is necessary to linearize the terms in $\boldsymbol{B}$; therefore, a new matrix

$$\check{\boldsymbol{B}} = \boldsymbol{diag}(\check{X}_u, \check{Y}_v, \check{Z}_w, \check{K}_p, \check{M}_q, \check{N}_r), \tag{A12}$$

consisting of characteristic drag terms linearized with respect to expected operating conditions is introduced. In the following Section 5.1, the determination of the characteristic linear yaw drag term $\check{N}_r$ used to calculate control gains is detailed, and the remaining degrees of freedom are calculated with a similar process.

A feed-forward term appended to Equation (A5) that cancels out the $\boldsymbol{\tau_B}$ in Equation (A6) shall be discussed in the immediately following subsection, for the yaw axis $\boldsymbol{\tau_B}$ is non-existent. Substituting Equations (A3) and (A4) into Equation (A5), setting this equal to Equation (A6), and grouping like terms together the resulting equation is obtained

$$\boldsymbol{k_p}\gamma_d + \boldsymbol{k_d}\dot{\gamma}_d = \boldsymbol{k_p}\gamma + (\boldsymbol{k_d} + \breve{\boldsymbol{B}})\dot{\gamma} + (\boldsymbol{M} + \boldsymbol{M_A})\ddot{\gamma}, \tag{A13}$$

where the left-hand side is a particular function $f(t)$ and the right-hand side is a second-order system of the standard form $f(t) = k\boldsymbol{x} + d\dot{\boldsymbol{x}} + m\ddot{\boldsymbol{x}}$. This makes the parameters

$$k = \boldsymbol{k_p}; \quad d = \boldsymbol{k_d} + \breve{\boldsymbol{B}}; \quad m = \boldsymbol{M} + \boldsymbol{M_A}. \tag{A14}$$

The natural frequency of a second-order system is $\omega_n = \sqrt{k/m}$, which can be reinterpreted to construct the diagonal matrix of closed-loop natural frequencies with

$$(\boldsymbol{\omega_n})_{ii} = \sqrt{\frac{(\boldsymbol{k_p})_{ii}}{(\boldsymbol{M})_{ii} + (\boldsymbol{M_A})_{ii}}}. \tag{A15}$$

In this paper, the notation $(\cdot)_{ii}$ refers to the i-th element of a diagonal matrix. The damping ratio of a second-order system in terms of its natural frequency is $\zeta = d/(2\omega_n m)$, which shall be used for the expression of the diagonal matrix whose elements are defined as

$$(\boldsymbol{\zeta})_{ii} = \frac{(\boldsymbol{k_d})_{ii} + (\breve{\boldsymbol{B}})_{ii}}{2(\boldsymbol{\omega_n})_{ii}(\boldsymbol{M} + (\boldsymbol{M_A})_{ii})}. \tag{A16}$$

All of $\boldsymbol{M}$, $\boldsymbol{M_A}$, $\breve{\boldsymbol{B}}$, $\boldsymbol{\omega_d}$, $\boldsymbol{\zeta}$, $\boldsymbol{k_p}$ and $\boldsymbol{k_d}$ are diagonal matrices, which allows for the convenient notation and manipulations in Equations (A15) and (A16). By setting the natural frequency ω_n to a desired value ω_d, we solve for proportional gain $\boldsymbol{k_p}$ with

$$\boldsymbol{k_p} = \boldsymbol{\omega_d^2}(\boldsymbol{M} + \boldsymbol{M_A}), \tag{A17}$$

and substituting this into Equation (A16) and setting $\boldsymbol{\zeta} = \boldsymbol{diag(1)}$ for a critically damped system, $\boldsymbol{k_d}$ becomes

$$\boldsymbol{k_d} = 2\boldsymbol{\omega_d}(\boldsymbol{M} + \boldsymbol{M_A}) - \breve{\boldsymbol{B}}. \tag{A18}$$

To be clear, each of the six elements of the respective diagonal matrices $\boldsymbol{k_p}$ and $\boldsymbol{k_d}$ have different values for each of the six degrees of freedom. Within Equations (A17) and (A18), only the desired closed-loop natural frequency $\boldsymbol{\omega_d}$ is variable, with all other terms being constant; therefore, $\boldsymbol{\omega_d}$ is the central parameter affecting the behavior of the control law from Equation (A5).

Appendix C. Frequency Response Testing

After achieving effectively critically damped step responses, the next sequence of testing for characterizing the performance of the yaw PD control law was closed-loop frequency response testing with SITL. Observations were made of the vehicle's ability to follow a $\pm 30^\circ$ desired yaw heading sine wave. A set of desired frequency yaw trajectories were generated from 0.05 Hz to 1.97 Hz in 0.04 Hz increments and saved as csv files. These trajectories were fed into the ROS `csv_reader` node which read the csv line-by-line in sync with the simulation such that the system had no knowledge of future headings (similar to the AR pilot commands). A separate ROS `control_law` node monitored vehicle state and calculated the force outputs according to the modified PD control law as described in Section 4 with the addition of δk_d, as discussed in the previous section. The resulting headings were recorded to csv files with a ROS `logger` node. Three desired closed-loop natural frequency settings were tested: $\omega_d = \{2.0, 3.0, 4.0\}$ rad/s.

These simulation results map the nonlinear frequency response H_{NL} using the tuned yaw controller of Equation (24) with $\kappa = 20.0$. The ratio of the amplitude of the output

(actual) to input (desired) is known as the gain $|H|$ or amplitudes of the system. Amplitudes were calculated by taking half the difference of the maximum and minimum heading from respective output and input data, and the gain is shown on the Bode gain plot. The amount that the output lags behind the control input is known as the phase shift. To quantify phase shift, a peak finding function was used on the respective output and input data to store the times of local maxima into vectors. This analysis was performed for the three ω_d and displayed in Figure A1, with the frequency presented in units of rad/s.

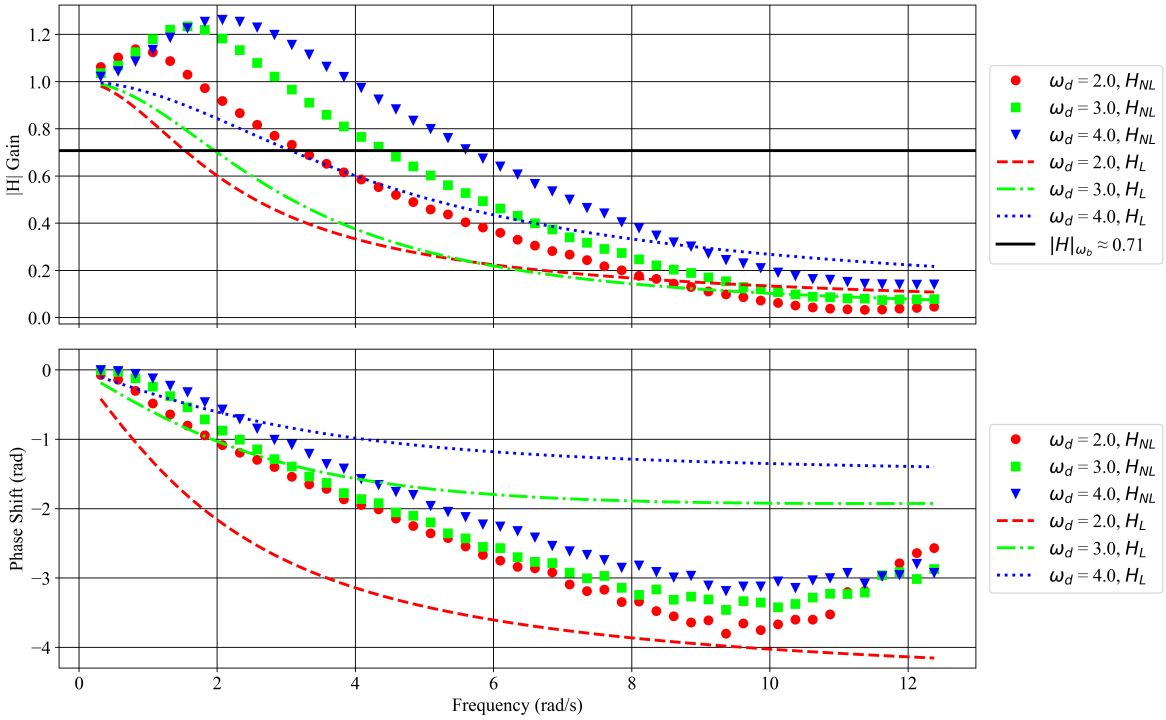

Figure A1. Bode plot gains and phase shifts of the closed-loop nonlinear frequency response simulation results are plotted with points and responses predicted by the linear approximation are plotted with lines. A solid line representing the -3 dB control bandwidth cutoff is drawn for reference on the gain graph. At higher frequencies where phase shift exceeded $-pi$, data were unwrapped to prevent discontinuities and for clarity.

The theoretical curves for the linear transfer function H_L shown on the Bode plot were calculated using the same transfer function as Equation (22), solving for the frequency response by replacing $s := j\omega$, where j is an imaginary number and ω is frequency, which can be expressed as

$$|H_L| = \left| \frac{\Psi(j\omega)}{\Psi_d(j\omega)} \right|. \tag{A19}$$

The theoretical phase shift is the angle of Equation (22) in the complex plane, which can be expressed as

$$\angle H_L = \arctan\left(\frac{Im(H_L)}{Re(H_L)} \right), \tag{A20}$$

which is also a function of ω.

One common metric of assessing a closed-loop controller's response is characterizing attenuation of the frequency response with control bandwidth. The definition given by [32] states that the -3 dB cutoff is

$$|H(j\omega)|_{\omega=\omega_b} = \frac{\sqrt{2}}{2} \tag{A21}$$

where $|H(j\omega)|$ is the gain of the frequency response of the transfer function, ω is the frequency of the desired heading, ω_n is the closed-loop natural frequency which is inter-

changeable with ω_d as per Equation (A15), and ω_b is the control bandwidth which marks the highest frequency the system can keep up with before the gain falls below -3 dB. In other words, the frequency ω where the gain falls to $|H| = \sqrt{2}/2 \approx 0.71$ is the so-called control bandwidth ω_b. Looking at Figure A1, in the region spanning $0.71 \leq |H| \leq 1.0$, the slope is constant for all three tested ω_d settings, causing an apparent linear relationship. Furthermore, unity gains $|H| = 1.0$ are reached at frequencies $\omega = \omega_d$, which could serve as a custom definition of "bandwidths" for this particular nonlinear system, which could assist pilots in understanding how the choice of ω_d can be expected to facilitate BROV2 performance.

References

1. Hudson, I.; Jones, D.; Wigham, D. A review of the uses of work-class ROVs for the benefits of science: Lessons learned from the SERPENT project. *Underw. Technol.* **2005**, *26*, 83–88. [CrossRef]
2. Macreadie, P.I.; McLean, D.L.; Thomson, P.G.; Partridge, J.C.; Jones, D.O.B.; Gates, A.R.; Benfield, M.C.; Collin, S.P.; Booth, D.J.; Smith, L.L.; et al. Eyes in the sea: Unlocking the mysteries of the ocean using industrial, remotely operated vehicles (ROVs). *Sci. Total Environ.* **2018**, *634*, 1077–1091. [CrossRef] [PubMed]
3. Nautilus LIVE Ocean Exploration Trust. ROV Pilot. Available online: https://nautiluslive.org/career/rov-pilot (accessed on 20 May 2024).
4. Woods Hole Oceanographic Institution. ROV Jason/Medea. Available online: https://www.whoi.edu/what-we-do/explore/underwater-vehicles/ndsf-jason/ (accessed on 20 May 2024).
5. National Oceanic and Atmospheric Administration. Remotely Operated Vehicle SuBastian. Available online: https://oceanexplorer.noaa.gov/technology/subs/subastian/subastian.html (accessed on 20 May 2024).
6. School of Ocean and Earth Science and Technology at the University of Hawaii at Manoa. ROV Lu'ukai. Available online: https://www.soest.hawaii.edu/UHMC/Luukai.php (accessed on 20 May 2024).
7. Teague, J.; Allen, M.J.; Scott, T.B. The potential of low-cost ROV for use in deep-sea mineral, ore prospecting and monitoring. *Ocean. Eng.* **2018**, *147*, 333–339. [CrossRef]
8. Raber, G.T.; Schill, S.R. Reef Rover: A Low-Cost Small Autonomous Unmanned Surface Vehicle (USV) for Mapping and Monitoring Coral Reefs. *Drones* **2019**, *3*, 38. [CrossRef]
9. Wilby, A.; Lo, E. Low-Cost, Open-Source Hovering Autonomous Underwater Vehicle (HAUV) for Marine Robotics Research based on the BlueROV2. In Proceedings of the 2020 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV), St. Johns, NL, Canada, 30 September–2 October 2020; pp. 1–5. [CrossRef]
10. Zhou, M.; Gezer, E.C.; McConnell, W.; Yuan, C. Acrobatic Low-cost Portable Hybrid AUV (ALPHA): System Design and Preliminary Results. In Proceedings of the OCEANS 2022, Hampton Roads, VA, USA, 17–20 October 2022; pp. 1–5. [CrossRef]
11. Buscher, E.; Mathews, D.L.; Bryce, C.; Bryce, K.; Joseph, D.; Ban, N.C. Applying a Low Cost, Mini Remotely Operated Vehicle (ROV) to Assess an Ecological Baseline of an Indigenous Seascape in Canada. *Front. Mar. Sci.* **2020**, *7*, 669. [CrossRef]
12. Constine, J. Ocean Drone Startup Merger Spawns Sofar, the DJI of the Sea. Available online: https://techcrunch.com/2019/03/27/sofar-ocean-technologies/ (accessed on 20 May 2024).
13. Blue Robotics. BlueROV2 Heavy Configuration Retrofit Kit. Available online: https://bluerobotics.com/store/rov/bluerov2-upgrade-kits/brov2-heavy-retrofit/ (accessed on 20 May 2024).
14. Bell, T.W.; Nidzieko, N.J.; Siegel, D.A.; Miller, R.J.; Cavanaugh, K.C.; Nelson, N.B.; Reed, D.C.; Fedorov, D.; Moran, C.; Snyder, J.N.; et al. The Utility of Satellites and Autonomous Remote Sensing Platforms for Monitoring Offshore Aquaculture Farms: A Case Study for Canopy Forming Kelps. *Front. Mar. Sci.* **2020**, *7*, 520223. [CrossRef]
15. Zhao, L.; Zhou, M.; Loose, B.; Cousens, V.; Turrisi, R. Modifying an Affordable ROV for Under-ice Sensing. In Proceedings of the OCEANS 2021: San Diego–Porto, San Diego, CA, USA, 20–23 September 2021; pp. 1–5. [CrossRef]
16. Wu, C.J. 6-DoF Modelling and Control of a Remotely Operated Vehicle. Master's Thesis, Flinders University, Adelaide, Australia, 2018.
17. Einarsson, E.M.; Lipenitis, A. MPC Control for the BlueROV2—Theory and Implementation. Master's Thesis, Aalborg University, Aalborg, Denmark, 2020.
18. Shepherd, T. Pilots "Fly" Using ROVs. Available online: https://oceanexplorer.noaa.gov/edu/materials/rov-pilots-exploration-notes.pdf (accessed on 20 May 2024).
19. Xia, P.; McSweeney, K.; Wen, F.; Song, Z.; Krieg, M.; Li, S.; Yu, X.; Crippen, K.; Adams, J.; Du, E.J. Virtual telepresence for the future of ROV teleoperations: Opportunities and challenges. In Proceedings of the SNAME Offshore Symposium. SNAME, Houston, TX, USA, 22 February 2022; p. D011S001R001. [CrossRef]
20. Xia, P.; You, H.; Ye, Y.; Du, J. ROV teleoperation via human body motion mapping: Design and experiment. *Comput. Ind.* **2023**, *150*, 103959. [CrossRef]
21. Koubâa, A.; Allouch, A.; Alajlan, M.; Javed, Y.; Belghith, A.; Khalgui, M. Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey. *IEEE Access* **2019**, *7*, 87658–87680. [CrossRef]
22. ArduPilot. Choosing an Autopilot. Available online: https://ardupilot.org/copter/docs/common-autopilots.html (accessed on 20 May 2024).

23. Zhang, M.M.; Choi, W.S.; Herman, J.; Davis, D.; Vogt, C.; McCarrin, M.; Vijay, Y.; Dutia, D.; Lew, W.; Peters, S.; et al. DAVE Aquatic Virtual Environment: Toward a General Underwater Robotics Simulator. In Proceedings of the 2022 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV), Singapore, 19–21 September 2022; pp. 1–8. [CrossRef]
24. Cieślak, P. Stonefish: An Advanced Open-Source Simulation Tool Designed for Marine Robotics, with a ROS Interface. In Proceedings of the OCEANS 2019, Marseille, France, 17–20 June 2019. [CrossRef]
25. Gezer, E.C.; Zhou, M.; Zhao, L.; McConnell, W. Working toward the development of a generic marine vehicle framework: ROS-MVP. In Proceedings of the OCEANS 2022, Hampton Roads, VA, USA, 17–20 October 2022; pp. 1–5. [CrossRef]
26. Blue Robotics. BlueROV2 Assembly (R3 Version). Available online: https://bluerobotics.com/learn/bluerov2-assembly-r3-version/ (accessed on 20 May 2024).
27. PX4. Pixhawk. Available online: https://docs.px4.io/main/en/flight_controller/pixhawk.html (accessed on 20 May 2024).
28. Foote, T.; Purvis, M. REP 103 Standard Units of Measure and Coordinate Conventions. Available online: https://www.ros.org/reps/rep-0103.html (accessed on 20 May 2024).
29. Palmer, E.; Zhang, M. [REP-156] Define Coordinate Frame Conventions for Marine Robots #398. Available online: https://github.com/ros-infrastructure/rep/pull/398 (accessed on 9 August 2024).
30. Discourse, R. Maritime Robotics. Available online: https://discourse.ros.org/c/maritime/36 (accessed on 9 August 2024).
31. Blue Robotics. BlueROV2 Heavy Configuration Retrofit Kit Installation. Available online: https://bluerobotics.com/learn/bluerov2-heavy-configuration-retrofit-kit-installation/ (accessed on 20 May 2024).
32. Fossen, T.I. *Handbook of Marine Craft Hydrodynamics and Motion Control*; John Wiley & Sons: Hoboken, NJ, USA, 2021.
33. APM:Sub Release Notes. Available online: https://github.com/ArduPilot/ardupilot/blob/master/ArduSub/ReleaseNotes.txt (accessed on 14 August 2024).
34. Blue Robotics. T200 Thruster Polyfit. Available online: https://colab.research.google.com/drive/1CEDW9ONTJ8Aik-HVsqck8Y_EcHYLg0zK#scrollTo=yXoOCK3CvxoY (accessed on 20 May 2024).
35. Poubel, L.; Hamilton, A.; Anderson, M. Fluid Added Mass Proposal. Available online: http://sdformat.org/tutorials?tut=added_mass_proposal (accessed on 9 August 2024).
36. ArduSub. S.I.T.L. (Software in the Loop). Available online: http://www.ardusub.com/developers/sitl.html (accessed on 9 August 2024).
37. Orca4. Available online: https://github.com/clydemcqueen/orca4 (accessed on 15 August 2024).
38. Sandøy, S.S. System Identification and State Estimation for ROV Udrone. Master's Thesis, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2016.
39. Singh, J.; Taddese, A.; Dutia, D. Proposal for Automatic Moments of Inertia Calculations. Available online: http://sdformat.org/tutorials?tut=auto_inertial_params_proposal (accessed on 9 August 2024).
40. Franklin, G.F.; Powell, J.D.; Emami-Naeini, A. *Feedback Control of Dynamic Systems*, 6th ed.; Pearson: London, UK, 2010.