2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Angler: An Autonomy Framework for Intervention Tasks with Lightweight Underwater Vehicle Manipulator Systems

Evan Palmer, Christopher Holm, and Geoffrey Hollinger

***Abstract*— Developing autonomous intervention capabilities for lightweight underwater vehicle manipulator systems (UVMS) has garnered significant attention within recent years because of the opportunity for these systems to reduce intervention operating costs. Developing autonomous UVMS capabilities is challenging, however, because of the lack of available standardized software frameworks and pipelines. Previous works offer simulation environments and deployment pipelines for underwater vehicles, but fall short of providing a complete UVMS software framework. We address this gap by creating *Angler*: a software framework for developing localization, control, and decision-making algorithms with support for sim-to-real transfer. We validate this framework by implementing a state-of-the-art control architecture and demonstrate the ability to perform station keeping with a mean error below 0.25 m and waypoint tracking with an average final error of 0.398 m.**

I. Introduction

Lightweight underwater vehicle-manipulator systems (UVMS)—which range from 25-50 kg in size—can be used to perform intervention tasks like scientific sampling and structural maintenance [1]. Performing intervention with lightweight UVMS currently requires at least one well-trained human operator, whose training can be resource expensive. This motivates the development of autonomous solutions. However, because of the lack of standardized open-source software frameworks and unified development pipelines, implementing and deploying autonomous task-based planning and control capabilities for lightweight UVMS is burdensome and acts as a significant barrier-to-entry for researchers hoping to work with these systems.

Existing approaches provide solutions to individual stages of the UVMS development and testing process. Specifically, Project DAVE [2], UUV Sim [3], UWSim [4], and Stonefish [5] all provide options for simulating UVMS but do not provide pipelines to transition implemented solutions to a hardware platform. Approaches that support deployment to hardware include ROS-MVP [6] and COLA2 [7]. Unfortunately, ROS-MVP is designed only for vehicle integration, and COLA2 is proprietary.

In an effort to standardize testing and to reduce development time needed to deploy lightweight UVMS, we propose *Angler*. *Angler* is an open-source[1,2] ROS 2 [8] framework designed to provide a plug-and-play solution for developing custom sensor interfaces, localization algorithms, control algorithms, and behavior trees for lightweight UVMS. In addition to offering interfaces for developing UVMS solutions, *Angler* provides a unified pipeline for testing these algorithms in simulation and transitioning them to hardware.

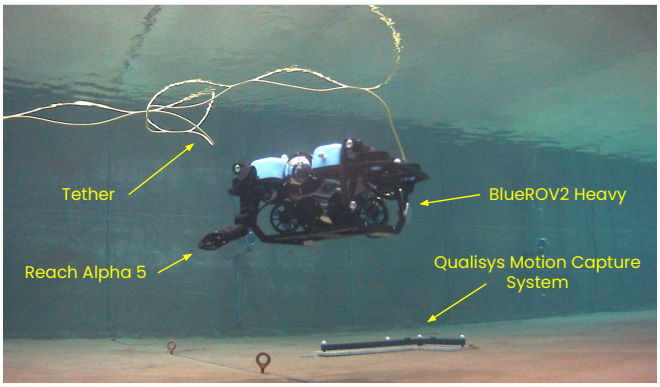

Fig. 1: We demonstrate our proposed framework and sim-to-real pipeline on the BlueROV2 Heavy with a mounted Reach Alpha 5 manipulator through hardware experiments in the O.H. Hinsdale Wave Research Laboratory.

We validate the proposed framework by creating a state-of-the-art UVMS control architecture using integral sliding mode control and set-based task priority inverse kinematic control. The control architecture accounts for the unique challenges associated with integrating lightweight UVMS (i.e., high manipulator-to-vehicle mass ratio and model uncertainty created by nonlinear hydrodynamic forces acting on the system) and achieves fundamental control capabilities required for autonomous intervention. Hardware experiments were performed using the framework with a BlueROV2 Heavy[3], a mounted Reach Alpha 5 manipulator[4], and a Qualisys motion capture system for localization[5] (see Fig. 1). The results of these experiments demonstrate the system's ability to perform successful end effector station-keeping with an average end effector tracking error below 0.25 m and waypoint tracking with the average final error between the end effector and waypoint being 0.398 m.

II. Related Work

Developing autonomous intervention capabilities for UVMS has garnered significant attention within recent years with several large-scale projects achieving success using

*The work of Evan Palmer was supported by the National Defense Science and Engineering Graduate (NDSEG) Fellowship.

Evan Palmer, Christopher Holm and Geoffrey Hollinger are with the Collaborative Robotics and Intelligent Systems (CoRIS) Institute, Oregon State University, Corvallis OR 97331, USA {palmeeva, christopher.holm, geoff.hollinger}@oregonstate.edu

[1]https://github.com/Robotic-Decision-Making-Lab/blue

[2]https://github.com/Robotic-Decision-Making-Lab/angler

[3]https://bluerobotics.com/

[4]https://reachrobotics.com/

[5]https://www.qualisys.com/

intervention-class UVMS (100-200 kg). The SAUVIM [9] and TRIDENT [10] projects demonstrate underwater target recovery. Valve turning is another common demonstration which has been shown by DexROV [11], TRITON [12], and Giergial et al. [13]. Finally, the TWINBOT project has successfully demonstrated autonomous cooperative object transportation [14]. While these projects have been successful, many take advantage of propriety software to implement their solutions [7] while the remaining leave their software frameworks closed-source.

For lightweight UVMS, most work has been limited to simulation. Marais et al. present an anisotropic disturbance rejection technique [15] and an energy optimal trajectory planner [16] for a simulated lightweight UVMS. Barbălată et al. explore the dynamic coupling effect and use of coupled integral sliding mode control on a simulated lightweight UVMS in [17], [18]. Wang et al. provides an exception to the use of simulation by demonstrating successful target recovery using a custom-built manipulator [19] and a bioinspired lightweight UVMS [20]. Though, these works focus on the development of custom hardware platforms and do not provide an accompanying open-source software framework.

III. Software Framework

Angler provides a complete framework for developing and testing lightweight UVMS, and has been informed by the following goals:

1) *Modularity*: Ensure researchers can implement sensor interfaces, control algorithms, and localization algorithms without needing to re-write existing interfaces.
2) *Task-Oriented*: Provide an architecture that allows researchers to leverage existing algorithms and capabilities to perform complex intervention tasks.
3) *Flexibility*: Provide interfaces to support using a variety of hardware platforms ensuring that our framework is hardware agnostic.
4) *Sim-To-Real*: Create a pipeline that enables prototyping solutions in simulation and deploying the same solutions on hardware without significant modification.
5) *Leverage existing open-source software*: *Angler* uses ROS 2 and ArduPilot [21] to create a familiar API and toolset for underwater roboticists.

Angler accomplishes these objectives by implementing a collection of ROS 2 packages that enable development of sensor and hardware interfaces as well as algorithms for control, localization, and task planning. *Angler* also integrates ArduPilot to enable software-in-the-loop (SITL) testing and deployment to hardware. In the remainder of this section, we provide additional details regarding the interfaces made available through *Angler*.

A. Localization

At the core of any underwater robotics platform is the ability to perform localization. Unfortunately, most lightweight UVMS offer minimal sensing capabilities out-of-the-box, making it necessary for researchers to integrate their own sensor interfaces. Furthermore, once a sensor reading has been received, that reading is not always sent directly to a state estimator; sometimes it's used as an input to a localization algorithm (e.g., sonar scans used for SLAM [22]). Recognizing these factors, *Angler* provides a localization package with helper classes to support integrating sensors into the system and creating localization algorithms using received sensor data. The package can be used to send any identified velocity or pose states to the ArduPilot Extended Kalman Filter (EKF) for state estimation.

B. Controllers

UVMS control architectures can be separated into two categories: coupled and decoupled. In a coupled control architecture, the manipulator(s) and vehicle are controlled by a single control algorithm. Whereas, in a decoupled control architecture, a high-level kinematic controller is used in conjunction with separate manipulator and vehicle dynamic controllers. *Angler* provides two interfaces to support developing both types of control architectures:

1) Vehicle Control: Robust control techniques like sliding mode control are generally used for vehicle control when implementing decoupled control algorithms for lightweight UVMS [15], [18]. Unfortunately, ArduPilot only implements PID control for vehicle control, which has been demonstrated to be insufficient in the presence of underwater disturbances [23]. *Angler* enables development and integration of more complex controllers by providing a custom interface for sending commands to individual thrusters.

2) Whole-Body Control: Angler provides an additional package for developing specifically whole-body controllers. Whole-body controllers can be implemented for use in either coupled or decoupled control schemes according to the researcher's needs.

C. Behavior Tree

Behavior trees are a popular solution to enabling high-level decision-making and task execution in robotic systems, and have seen preliminary adoption within marine robotics applications [24], [25]; however, they have not yet been integrated into a UVMS framework. Behavior trees can be characterized as rooted, directed trees comprised of control flow nodes and execution nodes. Control flow nodes are responsible for enabling decision-making within a behavior tree. The types of control flow nodes are *fallback*, *sequence*, *parallel*, and *decorator* nodes. Execution nodes perform some task within the behavior tree as either an *action* node or a *condition* node. For additional information regarding behavior tree design, we refer readers to [26].

Angler takes advantage of behavior trees in the framework to create reusable primitive behaviors, or subtrees, from which more complex behaviors can be developed. The implemented primitive behaviors include arming/disarming, querying a planner, and moving to a waypoint.

D. Hardware Interface

One of the main strengths of *Angler* is that it gives researchers the flexibility to choose their hardware platform.

This flexibility is made possible through the *mux/demux* interfaces:

1) Mux: A *mux* acts as the primary entry point to *Angler* for vehicle and manipulator state information. The gathered state information is combined into a generic state object that can be used by *Angler*. Researchers can implement their own *mux* to collect state information from custom sources. For example, a *mux* could be implemented to gather state information regarding the vehicle from the ArduPilot EKF and state information regarding a manipulator from a custom manipulator driver.

2) Demux: A *demux* is responsible for splitting up a unified control command and distributing that command to its respective end points. In the case of decoupled control, a *demux* might split a command from a kinematic controller into separate control commands for the vehicle and manipulator controllers. Alternatively, a *demux* for a coupled controller may simply proxy control commands to the hardware.

E. Sim-to-Real

Angler uses Gazebo for hydrodynamics simulation and ArduPilot for SITL testing and deployment to hardware. The benefit in providing a SITL testing environment is that any implemented algorithms and interfaces can be tested using the same software that provides integration with hardware, helping to minimize integration issues during sim-to-real transfer. With regards to the transfer process itself, computers used for simulation and topside operation often have different processor architectures and operating systems than devices onboard the vehicle. Consequently, performing sim-to-real transfer can introduce cross-compilation issues and dependency mismatches.

Angler overcomes this problem by integrating Docker. *Angler* builds and releases Docker images targeting *amd64* and *aarch64* architectures which allows researchers to run *Angler* from a topside computer or on a compute platform located on the UVMS. Furthermore, the use of Docker ensures that the *Angler* framework is completely sand-boxed, limiting differences between the simulation environment and the hardware environment. The complete sim-to-real pipeline may be observed in Fig. 2.

IV. System Model

This section establishes a mathematical model for an underwater vehicle manipulator system (UVMS) and other preliminary information needed to understand the implemented control architecture.

A. Kinematic Model

A UMVS is a multi-body system consisting of an underwater vehicle and one or more manipulators affixed to the vehicle. The vehicle's pose η with respect to an inertial, earth-fixed frame I and velocities v with respect to a body-fixed frame B are defined as [27]:

$$\begin{aligned} \eta &= \begin{bmatrix}\eta_1 & \eta_2\end{bmatrix}^T, \quad \eta_1 = \begin{bmatrix}x & y & z\end{bmatrix}^T, \quad \eta_2 = \begin{bmatrix}\phi & \theta & \psi\end{bmatrix}^T \\ v &= \begin{bmatrix}v_1 & v_2\end{bmatrix}^T, \quad v_1 = \begin{bmatrix}u & v & w\end{bmatrix}^T, \quad v_2 = \begin{bmatrix}p & q & r\end{bmatrix}^T \end{aligned} \tag{1}$$

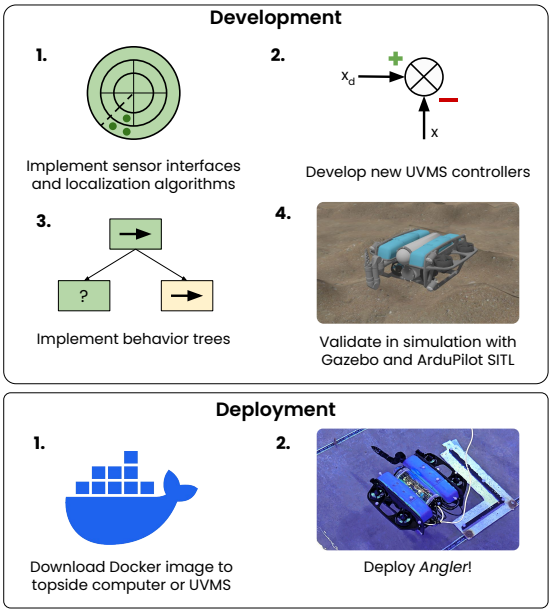

Fig. 2: The *Angler* sim-to-real pipeline uses Docker to facilitate deployment of software tested in simulation to a topside computer or the UVMS itself.

where x, y, and z capture the linear position of the vehicle; ϕ, θ, and ψ denote the vehicle's roll, pitch, and yaw angles, respectively; u, v, and w denote linear velocities; and p, q, and r denote angular velocities. The joint angles of an attached n degree of freedom manipulator are represented using the joint matrix $q = \begin{bmatrix}q_1 & q_2 & \dots & q_n\end{bmatrix}^T \in \mathbb{R}^n$. The respective joint velocities are captured as $\dot{q} = \begin{bmatrix}\dot{q_1} & \dot{q_2} & \dots & \dot{q_n}\end{bmatrix}^T \in \mathbb{R}^n$. Finally, the end effector's pose $\eta_{ee} = \begin{bmatrix}\eta_{ee,1} & \eta_{ee,2}\end{bmatrix}^T \in \mathbb{R}^6$ with respect to the inertial frame may be determined using forward kinematics.

The linear and angular velocities of the manipulator's end-effector $\dot{\eta}_{ee} = \begin{bmatrix}\dot{\eta}_{ee,1} & \dot{\eta}_{ee,2}\end{bmatrix}^T$ expressed in the earth-fixed frame can be determined using differential kinematics and are given as:

$$\dot{\eta}_{ee} = \begin{bmatrix}J_{pos} \\ J_{ori}\end{bmatrix}\zeta = J(\eta, q)\zeta \in \mathbb{R}^6 \tag{2}$$

where $\zeta = \begin{bmatrix}v_1 & v_2 & \dot{q}\end{bmatrix}^T$ is the vector of system velocities, J_{pos} represents the UVMS position Jacobian, and J_{ori} represents the UVMS orientation Jacobian. Let R^β_α denote a rotation matrix from frame α to frame β. Furthermore, let $r^\alpha_{\alpha,\beta}$ denote a vector defining the translation from frame α to frame β with respect to frame α. We now define J_{pos} and J_{ori} as:

$$J_{pos} = \begin{bmatrix}R^I_B & -S_t R^I_B & J^I_{pos,man}\end{bmatrix} \in \mathbb{R}^{3\times(6+n)} \tag{3}$$

$$J_{ori} = \begin{bmatrix}O & R^I_B & J^I_{ori,man}\end{bmatrix} \in \mathbb{R}^{3\times(6+n)} \tag{4}$$

where $S_t = S(R^I_B r^B_{B,0}) + S(R^I_0 r^0_{0,ee})$, 0 represents the manipulator base frame, ee represents the end-effector frame, O denotes the null matrix, and $S(x)$ defines the skew-symmetric matrix for a vector x [27]. Finally, $J^I_{pos,man}$ and $J^I_{ori,man}$ represent the manipulator position and orientation Jacobian matrices with respect to the inertial frame.

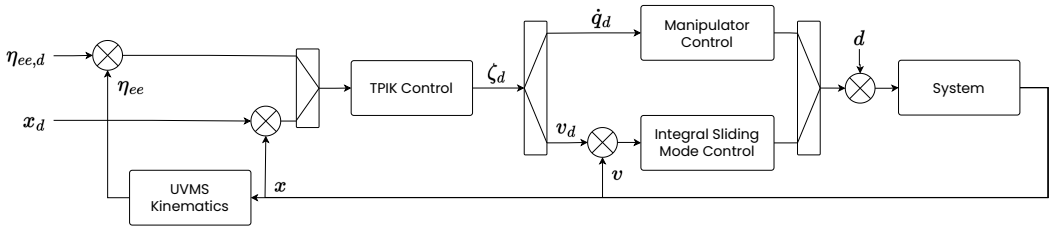

Fig. 3: Block diagram depicting the proposed control architecture for a lightweight UVMS. TPIK control is used to enable tracking high-level control tasks while the ISMC and manipulator controller are responsible for tracking the desired system velocities.

B. Dynamic Model

The dynamics for an underwater vehicle are given in Newton-Euler form as

$$M\dot{v} + C(v)v + D(v)v + g(\eta) + d(\eta, t) = \tau \in \mathbb{R}^6 \quad (5)$$

where M is the inertia matrix with added mass terms, $C(v)$ is the matrix of Coriolis and centripetal terms, $D(v)$ is the damping matrix, $g(\eta)$ is the vector of restoring forces, $d(\eta, t)$ represents the unmodeled forces, and τ is the vector of vehicle forces and moments [28].

V. Control Architecture

We demonstrate the capabilities of *Angler* by creating a state-of-the-art, decoupled control architecture designed to achieve fundamental control capabilities required for autonomous intervention with lightweight UVMS. Set-based task priority inverse kinematic (TPIK) control has been implemented as the whole-body kinematic controller. The formulation for set-based TPIK control and a description of the implemented control tasks are given in Section V-A. We also design an integral sliding mode controller for robust vehicle velocity control to ensure accurate tracking of desired system velocities provided by the set-based TPIK controller. The integral sliding mode controller is further explored in Section V-B. The complete control architecture is summarized in Fig. 3.

A. Set-Based Task Priority Inverse Kinematic Control

Task priority control methods [27] seek to achieve a desired system state through the composition tasks, where a task is defined as a control objective $\sigma(x) \in \mathbb{R}^m$ which acts as a function on the system state $x = \begin{bmatrix}\eta & q\end{bmatrix}^T$. The desired system velocities ζ_d required to achieve a desired task value σ_d can be calculated with the following closed-loop inverse kinematics equation [29]:

$$\zeta_d = J^\dagger(\dot{\sigma}_d + K\sigma_e) \quad (6)$$

where $J^\dagger$ represents the Moore-Penrose pseudoinverse of the task Jacobian, $K \in \mathbb{R}^{m\times m}$ is a positive-definite matrix of gains, and $\sigma_e = \sigma_d - \sigma$ is the task error.

If the UVMS is kinematically redundant (i.e., the system has more degrees of freedom than the number of degrees of freedom required by the task), then Equation 6 will contain a null projection operation:

$$\zeta_d = J^\dagger(\dot{\sigma}_d + K\sigma_e) + (I - J^\dagger J)\zeta_{null} \quad (7)$$

where $\zeta_{null} \in \mathbb{R}^{n+6}$ is an arbitrary velocity matrix projected into the null space of the Jacobian [30]. Using the null space projection, multiple tasks may be defined and organized as a hierarchy in terms of priority for execution such that lower priority tasks are projected into the null space of the Jacobian of a higher priority task. The resulting system velocity given a hierarchy of tasks is computed as $\zeta_d = \zeta_1 + N_1\zeta_2 + \cdots + N_{1,i-1}\zeta_i$ where ζ_i are the calculated system velocities for the i^{th} task and $N_{1,i-1}$ is the null space of the augmented task Jacobian $J_{1,i-1} = \begin{bmatrix}J_1 & J_2 & \dots & J_{i-1}\end{bmatrix}^T$.

The above task-priority framework is designed to achieve a specific task value (i.e., an equality task). However, it is valuable to also constrain a task variable to a range. Such tasks are identified as set-based tasks. We integrate set-based tasks into the task-priority framework by implementing the method proposed by Moe et al. [31], which constructs the task hierarchy at each control iteration according to the current system state. If the current system state violates a set-based task's limits, the task in violation is embedded into the hierarchy as an equality task. Once the system no longer violates the set-based task constraints, the task is removed from the task hierarchy. For additional information regarding the hierarchy construction and selection algorithm, we refer readers to the derivation by Moe, et al. [31].

We implement three types of control tasks for execution within the task hierarchy. The first is a joint limit set-based task which constrains the range-of-motion of a manipulator joint, helping to prevent over-extension of the joint beyond its mechanical limits. The task Jacobian for this constraint is the row vector

$$J_{\text{lim},i} = \begin{bmatrix}0 & \dots & \underbrace{1}_{i} & \dots & 0\end{bmatrix} \in \mathbb{R}^{1\times n} \quad (8)$$

where i denotes the i^{th} manipulator joint. The task Jacobian for the vehicle yaw equality task is of the same form as that identified in Equation 8 with the exception being that index i is equivalent to that of the vehicle yaw column within the system Jacobian defined in Equation 2. Finally, the end-effector position task Jacobian is defined in Equation 3.

We assign the following task priority within the hierarchy:

1) Joint limits
2) End-effector position tracking
3) Vehicle yaw tracking

Joint limits have been assigned the highest priority in the hierarchy to ensure that the controller adheres to the

system safety constraints. End-effector position tracking is performed with second priority to accomplish accurate trajectory tracking. Finally, vehicle yaw tracking has been integrated as an optimization task to ensure that the vehicle faces its operational workspace.

B. Integral Sliding Mode Control

We design an integral sliding mode controller (ISMC) for vehicle control which provides robustness to matched uncertainties during the full system response [32]. To design the ISMC, we begin with the control law

$$\tau = \tau_0 + \tau_1 \tag{9}$$

where τ_0 is a controller responsible for stabilizing the nominal system and τ_1 is a controller responsible for rejecting the forces induced through $d(\eta, q, t)$. For this work, assume that $d(\eta, q, t)$ meets the definition of matched uncertainty such that $d(\eta, t)$ has a known upper bound and fulfills the matching condition (i.e., $d(\eta, t) \in span\{B\}$, where B represents a control selection matrix).

We design τ_0 using the nominal vehicle dynamics obtained from Equation 5 and computed torque control as follows:

$$\tau_0 = Mp + C(v)v + D(v)v + g(\eta) \tag{10}$$

where $p = \dot{v}_d + K_p v_e$ represents the auxillary control input, $v_e = v_d - v$ represents the vehicle velocity error, and $K_p \in \mathbb{R}^{6\times6}$ is a positive-definite matrix of gains. We now design τ_1 to reject uncertainties unhandled by τ_0. Let $s = \sigma + z$ be the sliding variable in which

$$\sigma = Cv_e, \tag{11}$$

$$\dot{z} = \dot{v}_e + K_p v_e - C\dot{v}_e \tag{12}$$

with C representing a positive-definite matrix. To ensure disturbance rejection during full system response, assume that the initial error state dynamics exist on the manifold describing the desired error dynamics (i.e., $s(0) = 0$). $z(0)$ can then be given as $z(0) = -Cv_e(0)$. After integrating $\dot{z}$, s and its respective time derivative are determined to be

$$s = v_e + K_p \int_0^t v_e \, dt - K_p v_e(0) \tag{13}$$

$$\dot{s} = \dot{v}_e + K_p v_e \tag{14}$$

When the desired error dynamics are achieved, $s = \dot{s} = 0$ and the disturbances are rejected throughout the system response. τ_1 is now designed to enforce sliding motion:

$$\tau_1 = \rho \; sign(s) \tag{15}$$

where ρ is a positive-definite matrix of switching gains and $sign(s)$ is the sliding mode control sign function.

To prove the stability of the controller, take the Lyapunov candidate

$$V = \frac{1}{2} s^T M s \tag{16}$$

$$\dot{V} = s^T M \dot{s} \tag{17}$$

which, upon substituting Equations 5, 14, and 9 and rearranging terms, yields

$$\begin{aligned} \dot{V} &= s^T (d(\eta, t) - \rho \; sign(s)) \\ &= ||s||(-\rho + ||d(\eta, t)||) \end{aligned} \tag{18}$$

When ρ is selected such that $\rho \geq ||d(\eta, t)|| + \epsilon$ where ϵ is a positive scalar, we obtain the inequality $\dot{V} \leq -\epsilon||s||$ and the controller is stable.

To alleviate chatter in the controller, we approximate the sign function using the hyperbolic tangent function: $sign(s) = tanh\left(\frac{s}{\lambda}\right)$, where $\lambda > 0$ is a scalar value defining the sign function boundary thickness. Furthermore, we perform thruster-level control by applying the formula

$$\tau = B_t K u \tag{19}$$

where $B_t \in \mathbb{R}^{6\times t}$ represents the thruster allocation matrix for a vehicle with t thrusters, $u \in \mathbb{R}^t$ represents a vector of thruster control signals, and $K \in \mathbb{R}^{t\times t}$ is a diagonal matrix whose elements represent the nonlinear relationship between a thruster's control signal and resulting thrust output.

VI. Experimental Setup

We validated our proposed framework using a BlueROV2 Heavy with a mounted Reach Alpha 5 manipulator (see Fig. 1). This platform can be easily launched by a single person without the use of launch and recovery systems. *Angler* was deployed to a Raspberry Pi 4 onboard the BlueROV2 to reduce system latency and to avoid bandwidth limitations imposed by the tether. We integrated the Reach Alpha 5 manipulator into the system by creating an open-source[6] ROS 2 driver the the Reach Alpha 5. Finally, we designed the system behavior tree with the ability to load a sequence of waypoints for the system to navigate to and the ability to execute that plan using the control architecture. A disarm fallback state was also integrated into the behavior tree, giving users the option to manually override the system in the event of unexpected behavior. The behavior tree is presented in Fig. 4.

All experiments were performed in the O.H. Hinsdale Wave Research Laboratory with a water depth of 1.2 m.

[6]https://github.com/Robotic-Decision-Making-Lab/alpha

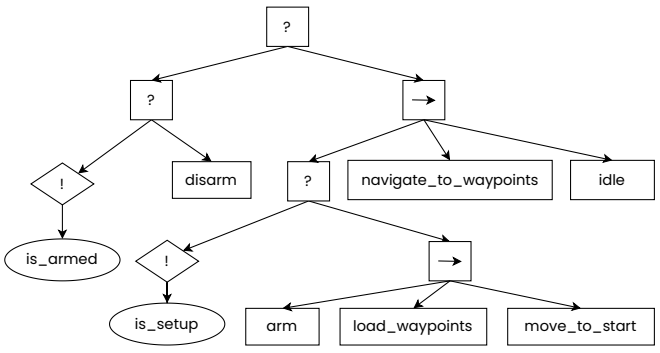

Fig. 4: Our proposed behavior tree has been design to enable UVMS task execution. Here, square nodes represent actions, ovals are conditions, ? represents a fallback node, → represents a sequence node, and ! denotes a not-decorator.

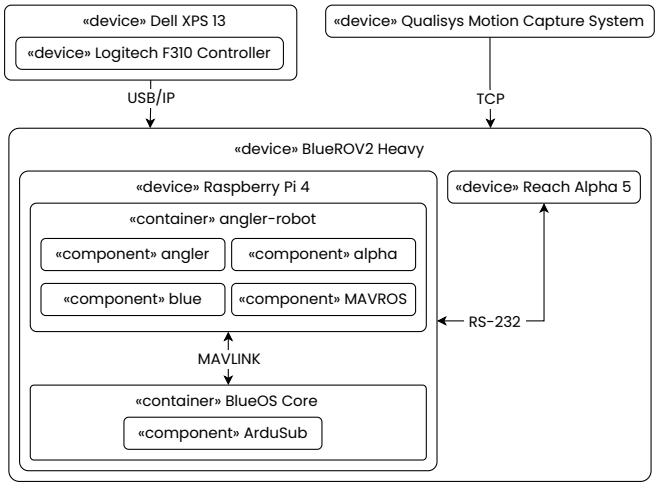

Fig. 5: We deploy our framework using Docker onto a Raspberry Pi 4 hosted by the BlueROV2 Heavy. This helps reduce latency between the software framework and the onboard devices.

Due to the minimal onboard sensing capabilities, we utilize a Qualisys underwater motion capture system for localization. The motion capture capture system provided an accuracy of 4 mm at the time of calibration, making it suitable for ground-truth measurements. In the case of real-world deployments, the motion capture system would be replaced by additional sensors, such as a Doppler Velocity Log and acoustic beacons. Our goal in these trials is to characterize the error inherent in the implemented control architecture given accurate localization. The complete deployment diagram is presented in Fig. 5.

VII. Experiments and Results

A majority of intervention tasks (e.g., control panel operation and valve turning) require a UVMS to maintain a stationary pose (i.e., station-keeping) for extended periods of time, making station-keeping one of the most critical capabilities for an autonomous platform. Therefore, we evaluate the ability of our system to perform end-effector station-keeping across five consecutive trials, each lasting one minute in duration. The end effector tracking error observed across each trial may be observed in Fig. 6. The mean error across all trials was 0.223 m $\pm$ 0.031 m, respectively, showing that the system was able to maintain a consistent position across all trials. The observed error can be attributed to steady-state error in the ISMC controller and inaccuracies in the nonlinear thrust-to-PWM mapping used for direct thruster control.

Waypoint navigation is another core capability that an autonomous framework should be able to demonstrate. This capability is required for intervention tasks like target retrieval and object transportation. We test the proposed framework's ability to navigate to the waypoint $\eta_{ee,1,d} = [1.0, 1.0, 0.5]^T$ from an initial position across five consecutive trials. The trajectories executed by the system are presented in Fig. 7. The average error between the end effector and desired waypoint at the time of convergence across all trials was 0.398 m $\pm$ 0.099 m.

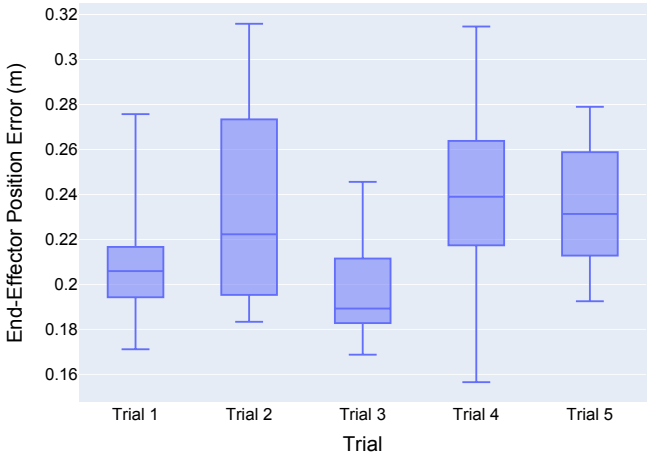

Fig. 6: The average end-effector position error across station keeping trials was observed to be 0.223 m $\pm$ 0.031 m, demonstrating the stability of the control framework for extended periods of time.

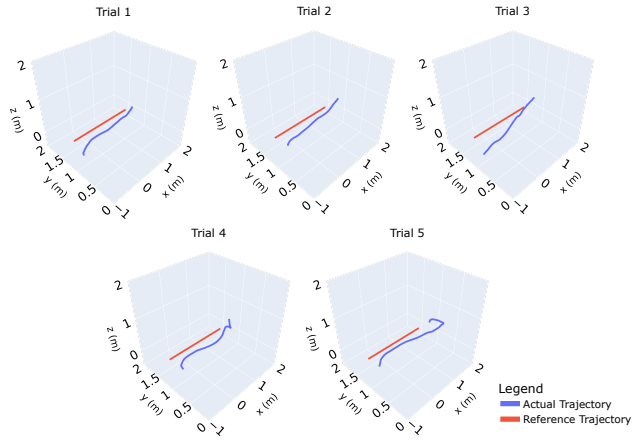

Fig. 7: The UVMS was instructed to navigate from an approximate initial position of $\eta_{ee,1} = [-1.0, 1.0, 0.5]^T$ to the waypoint $\eta_{ee,1,d} = [1.0, 1.0, 0.5]^T$. This shows the ability of the framework to converge on a distinct waypoint over an extended distance.

VIII. Discussion

We have presented *Angler*, a complete software framework for lightweight underwater vehicle manipulator systems (UVMS) that makes it easier to develop and deploy UVMS algorithms. *Angler* provides interfaces to enable integration of sensors, localization algorithms, controllers, and behavior trees. *Angler* emphasizes flexibility with regards to the hardware platform and provides a pipeline for testing systems in simulation and deploying those systems to hardware. We have validated *Angler* by creating a state-of-the-art control architecture for lightweight UVMS and successfully demonstrated core autonomy capabilities. This work is a first step in improving the accessibility of underwater robotics research to the general robotics community.

IX. Acknowledgement

We would like to thank Hannah Kolano and Scott Chow for their help throughout the experimentation process.

REFERENCES

[1] F. Nauert and P. Kampmann, "Inspection and maintenance of industrial infrastructure with autonomous underwater robots," *Frontiers in Robotics and AI*, vol. 10, 2023.

[2] M. M. Zhang, W.-S. Choi, J. Herman, D. Davis, C. Vogt, M. McCarrin, Y. Vijay, D. Dutia, W. Lew, S. Peters, and B. Bingham, "DAVE Aquatic Virtual Environment: Toward a General Underwater Robotics Simulator," in *IEEE Autonomous Underwater Vehicles Symposium (AUV)*, 2022, pp. 1–8.

[3] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS*, 2016, pp. 1–8.

[4] J. J. Fernández, J. Pérez, A. Peñalver, J. Sales, D. Fornas, and P. J. Sanz, "Benchmarking using UWSim, Simurv and ROS: An autonomous free floating dredging intervention case study," in *OCEANS*, 2015, pp. 1–7.

[5] P. Cieślak, "Stonefish: An Advanced Open-Source Simulation Tool Designed for Marine Robotics, With a ROS Interface," in *OCEANS*, 2019.

[6] E. C. Gezer, M. Zhou, L. Zhao, and W. McConnell, "Working toward the development of a generic marine vehicle framework: ROS-MVP," in *OCEANS*, 2022, pp. 1–5.

[7] N. Palomeras, A. El-Fakdi, M. Carreras, and P. Ridao, "COLA2: A Control Architecture for AUVs," *IEEE Journal of Oceanic Engineering*, vol. 37, no. 4, pp. 695–716, 2012.

[8] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022.

[9] G. Marani, S. K. Choi, and J. Yuh, "Underwater autonomous manipulation for intervention missions AUVs," *Ocean Engineering*, vol. 36, no. 1, pp. 15–23, 2009.

[10] P. J. Sanz, P. Ridao, G. Oliver, G. Casalino, Y. Petillot, C. Silvestre, C. Melchiorri, and A. Turetta, "TRIDENT An European project targeted to increase the autonomy levels for underwater intervention missions," in *OCEANS*, 2013, pp. 1–10.

[11] P. Di Lillo, E. Simetti, F. Wanderlingh, G. Casalino, and G. Antonelli, "Underwater Intervention With Remote Supervision via Satellite Communication: Developed Control Architecture and Experimental Results Within the Dexrov Project," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 1, pp. 108–123, 2021.

[12] N. Palomeras, A. Peñalver, M. Massot-Campos, G. Vallicrosa, P. L. Negre, J. J. Fernández, P. Ridao, P. J. Sanz, G. Oliver-Codina, and A. Palomer, "I-AUV docking and intervention in a subsea panel," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2014, pp. 2279–2285.

[13] P. Cieslak, P. Ridao, and M. Giergiel, "Autonomous underwater panel operation by GIRONA500 UVMS: A practical approach to autonomous underwater manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 529–536.

[14] R. Pi, P. Cieślak, P. Ridao, and P. J. Sanz, "TWINBOT: Autonomous Underwater Cooperative Transportation," *IEEE Access*, vol. 9, pp. 37 668–37 684, 2021.

[15] W. J. Marais, S. B. Williams, and O. Pizarro, "Anisotropic Disturbance Rejection for Kinematically Redundant Systems With Applications on an UVMS," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7017–7024, 2021.

[16] ——, "Go With the Flow: Energy Minimising Periodic Trajectories for UVMS," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 01–07.

[17] C. Barbălată, M. W. Dunnigan, and Y. Pétillot, "Dynamic coupling and control issues for a lightweight underwater vehicle manipulator system," in *OCEANS*, 2014, pp. 1–6.

[18] C. Barbălată, M. W. Dunnigan, and Y. Petillot, "Coupled and Decoupled Force/Motion Controllers for an Underwater Vehicle-Manipulator System," *Journal of Marine Science and Engineering*, vol. 6, no. 3, 2018.

[19] Y. Wang, S. Wang, Q. Wei, M. Tan, C. Zhou, and J. Yu, "Development of an Underwater Manipulator and Its Free-Floating Autonomous Operation," *IEEE Transactions on Mechatronics*, vol. 21, no. 2, pp. 815–824, 2016.

[20] Y. Wang, M. Cai, S. Wang, X. Bai, R. Wang, and M. Tan, "Development and Control of an Underwater Vehicle–Manipulator System Propelled by Flexible Flippers for Grasping Marine Organisms," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 4, pp. 3898–3908, 2022.

[21] ArduPilot Development Team, "ArduPilot," https://ardupilot.org/.

[22] J. Wang, S. Bai, and B. Englot, "Underwater localization and 3d mapping of submerged structures with a single-beam scanning sonar," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4898–4905.

[23] H. Tugal, K. Cetin, X. Han, I. Kucukdemiral, J. Roe, Y. Petillot, and M. S. Erden, "Sliding Mode Controller for Positioning of an Underwater Vehicle Subject to Disturbances and Time Delays," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 3034–3039.

[24] C. I. Sprague, Özkahraman, A. Munafo, R. Marlow, A. Phillips, and P. Ögren, "Improving the Modularity of AUV Control Systems using Behaviour Trees," in *Autonomous Underwater Vehicle Workshop (AUV)*, 2018, pp. 1–6.

[25] S. Bhat, I. Torroba, Özkahraman, N. Bore, C. I. Sprague, Y. Xie, I. Stenius, J. Severholt, C. Ljung, J. Folkesson, and P. Ögren, "A Cyber-Physical System for Hydrobatic AUVs: System Integration and Field Demonstration," in *IEEE Autonomous Underwater Vehicles Symposium (AUV)*, 2020, pp. 1–8.

[26] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[27] G. Antonelli, *Underwater Robots*. Springer International Publishing, 2018.

[28] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley Sons, Ltd, 2011, ch. 2, pp. 15–44.

[29] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 398–410, 1997.

[30] C. A. Klein and C.-H. Huang, "Review of pseudoinverse control for use with kinematically redundant manipulators," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 2, pp. 245–250, 1983.

[31] S. Moe, G. Antonelli, A. R. Teel, K. Y. Petterson, and J. Schrimpf, "Set-based tasks within the singularity-robust multiple task-priority inverse kinematics framework: general formulation, stability analysis, and experimental results," *Frontiers in Robotics and AI*, vol. 3, no. 16, 2016.

[32] V. Utkin and J. Shi, "Integral Sliding Mode in Systems Operating under Uncertainty Conditions," in *IEEE Conference on Decision and Control*, vol. 4, 1996, pp. 4591–4596 vol.4.