A Generalizable Entity-Component-System Architecture for Underwater ROV Control

Eoghan McIvor, George Sklivanitis, and Dimitris A. Pados
Center for Connected Autonomy and AI, Florida Atlantic University, Boca Raton, FL, USA
E-mail: {emcivor2021, gsklivanitis, dpados}@fau.edu

***Abstract*—We present a novel software architecture for underwater Remotely Operated Vehicles (ROVs) based on Entity Component System (ECS), a software architectural pattern mostly used in video game development. We introduce two key innovations: 1) A modular, cross-platform architecture that seamlessly synchronizes state between the ROV and surface station, and 2) a generalized thruster control system capable of adapting to arbitrary thruster configurations. Unlike existing ROV software solutions that tend to be tightly coupled to specific hardware configurations, our approach enables flexible reconfiguration and robust control across different physical setups. We experimentally validate our system on a modified BlueROV2 platform in a swimming pool, demonstrating successful control across four different thruster configurations with attitude errors remaining below 2° for yaw and below 4° for roll and pitch control. The results confirm that our architecture can effectively maintain control performance even when thrusters are disabled or reconfigured, providing a foundation for adaptable underwater robotic systems.**

I. Introduction

Underwater Remotely Operated Vehicles (ROVs) operate in challenging environments that demand robust control systems. Traditional ROV software architectures often couple control algorithms tightly to specific hardware configurations, limiting adaptability when hardware changes are required due to failures or varying mission requirements [1].

This paper addresses two fundamental challenges in underwater ROV software design:

1) **Rigid Software Architectures:** Many existing ROV software implementations are cumbersome to extend and tightly coupled to the electromechanical configuration of a specific ROV. As such, many current solutions require substantial expertise to customize, especially in the case of novel research vessels.
2) **Hardware-Specific Control:** Existing thruster control approaches typically employ fixed matrices designed for specific thruster arrangements, requiring significant code changes when the physical configuration changes.

Our work contributes:

- An Entity Component System (ECS) based architecture that provides automatic state synchronization between the ROV and control station.

This work was supported in part by the National Science Foundation Grant CMMI-2408117.

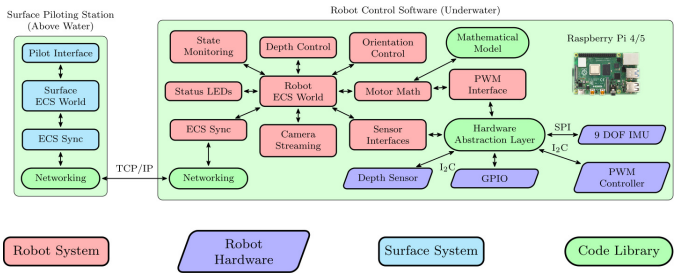

Fig. 1. System architecture showing the distributed ECS implementation. The surface station (left) and robot control software (right) each maintain their own ECS world with automatic synchronization over TCP/IP. The robot side integrates with hardware through a hardware abstraction layer, while both sides share a common software foundation.

- A generalized thruster control system capable of adapting to arbitrary thruster configurations through matrix pseudoinverse techniques.

Experimental validation of our developed system was carried out on a modified BlueROV2 platform in a swimming pool demonstrating effective control across four different thruster configurations with attitude errors remaining below 2° for yaw and below 4° for roll and pitch control.

II. Related Work

We focus our review on two key areas relevant to our contributions: 1) ROV software architectures, and 2) thruster control approaches.

ROV Software Architectures: Commercial ROV platforms typically employ proprietary software with limited flexibility. ArduSub [2] is a fully-featured open-source solution for ROVs but it remains difficult to extend and is tightly coupled to specific thruster configurations. MOOS, short for "Mission Oriented Operating Suite", is a suite of software modules that organizes different software processes running on an autonomous vehicle and its original use was for the Bluefin Odyssey III vehicle owned by MIT. IvP stands for "Interval Programming" which is a mathematical programming model for multi-objective optimization. The MOOS-IvP software design is based on three architecture philosophies, (a) the backseat driver paradigm, (b) publish and subscribe autonomy middleware, and (c) behavior based autonomy. ROS [3] is a widely-used open-access software framework for robot software development. While ROS may be open source and

have an architecture adaptable to any robot configurations, it has several shortcomings regarding usability, portability, and platform dependency. Additionally, both ROS and MOOS-IvP lack native support for underwater inter-vehicle communication among multiple marine vehicles [4].

Thruster Control Systems: Most underwater vehicles employ predetermined thruster configurations with fixed control matrices. Although theoretical approaches for arbitrary thruster configurations exist, practical implementations that can dynamically adapt to changing thruster arrangements remain rare [5], [6]. Recent work has explored optimization-based approaches, but these typically operate as offline planning tools rather than real-time control systems [7].

III. System Architecture

Our architecture employs the Entity Component System (ECS) pattern, which fundamentally separates data (components) from logic (systems) and identity (entities). This separation enables a uniquely flexible approach to ROV control that also lends well to modeling systems of multiple ROVs. Figure 1 shows our system architecture, illustrating how the ECS world on both the surface piloting station and the underwater robot are manipulated with systems and synchronized over the network. Examples of systems include `ECS Sync` on the robot and surface, and `Motor Math` on the robot. Examples of components include the target movement speed and the last reading from the depth sensor, to name a few.

A. ECS Architecture Overview

In our implementation, each ROV is represented as an entity with attached data components representing its state (position, orientation, desired movement, etc.). Systems operate on these data components in a three-phase update cycle:

1) **Pre-Update:** All sensor data is collected and processed. Control inputs are received over the network from operator.
2) **Update:** Control algorithms compute desired actions.
3) **Post-Update:** Actuator commands are calculated and transmitted.

This phased approach ensures consistent data flow while allowing parallel execution of non-conflicting systems. The architecture is implemented as two main packages—`robot` for the onboard controller and `surface` for the operator interface—with a shared `common` library.

To summarize with examples:

- **Control Modules:** Force order control intents (a component defined in `Motor Math`) are emitted by the `Orientation Control`, `Depth Control`, and `Pilot Interface` systems and the `Motor Math` system processes these intents to determine the appropriate thruster commands.
- **Sensor Integration:** New observations are read from the ROV's sensors over serial interfaces via the `Hardware Abstraction Layer` and then are stored as components in the ECS world, becoming visible to all other systems.
- **Communication:** `ECS Sync` and `Networking` systems process mutations made to the local ECS world and manage synchronizing these state changes with the surface and/or other nodes.

B. Automated State Synchronization

A key innovation in our system is the automated state synchronization between the ROV and the surface station. Any entity containing state relevant to the operation of other nodes (robot state, pilot movement intents, etc) can be tagged with a marker component named "Replicate", which enables change tracking and synchronization for that entity. When such an entity is modified, the change is:

1) Serialized into a binary protocol;
2) transmitted over the network (currently TCP);
3) deserialized and applied to the corresponding entity in the receiving node's ECS world.

This approach enables transparent distributed operation. Changes made on either node are automatically propagated to the others, creating a single logical ECS world despite physical separation.

IV. Generalized Thruster Control

Our second key contribution is a generalized thruster control system that can adapt to arbitrary thruster configurations without code changes.

A. Mathematical Foundation

For an ROV with n thrusters, we define:

$$m = Tf \quad \text{where} \tag{1}$$

- $m \in \mathbb{R}^6$ represents the desired movement in 6 DOF (surge, sway, heave, roll, pitch, yaw);
- $T \in \mathbb{R}^{6\times n}$ is the thruster configuration matrix;
- $f \in \mathbb{R}^n$ is the vector of thruster forces.

Each column in T represents a thruster's contribution to all six degrees of freedom when it produces one unit of thrust. This matrix is determined by each thruster's direction vector ($\hat{d}$) and each thruster's position vector ($\vec{r}$) relative to the ROV's center of mass. Namely, the first three rows of each column in T represent force and are defined by the components of $\hat{d}$. The last three rows represent torque and are defined by the components of $\vec{r} \times \hat{d}$. This setup follows from the fact that when the thruster is producing unit thrust, the force vector it imparts on the ROV is equivalent to the thruster's normal vector.

When controlling the ROV, we solve for f given m using the Moore-Penrose pseudoinverse for T as follows:

$$f = T^{+}m. \tag{2}$$

This approach provides the minimum-norm solution when $n > 6$ (overdetermined) and the least-squares solution when $n < 6$ (underdetermined), making it applicable to any thruster configuration.

As shown in Figure 1, the `Motor Math` system interfaces with both the mathematical model of the thruster configuration

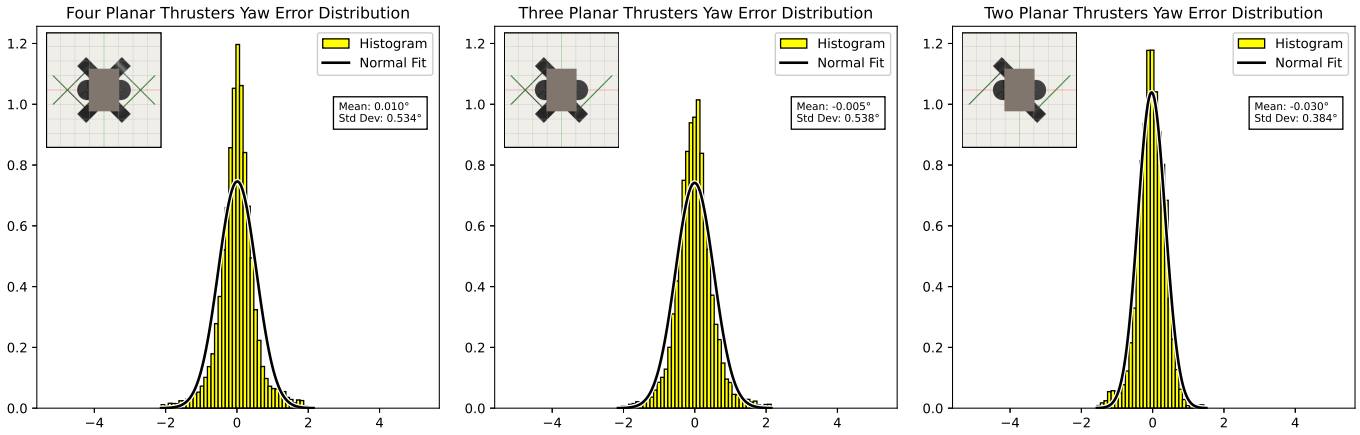

Fig. 2. Error distributions across different thruster configurations demonstrating yaw control with decreasing numbers of horizontal thrusters (from left-to-right: 4, 3, and 2 thrusters). X-axis is degrees, Y-axis is probability density.

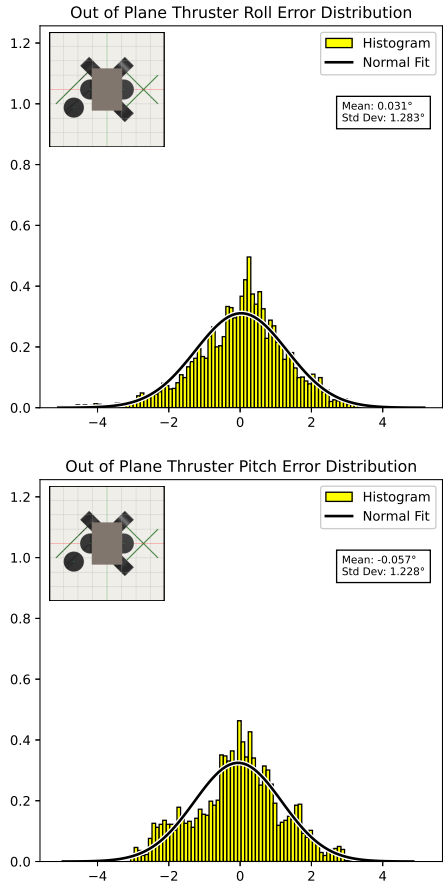

Fig. 3. Error distributions across different thruster configurations demonstrating roll and pitch control using the same modified configuration with a repurposed out-of-plane thruster. X-axis is degrees, Y-axis is probability density.

and the `PWM interface` that drives the actual thrusters. This separation allows the control system to adapt to different configurations without code changes by simply updating the parameters of the mathematical model and allows the system to be elegantly extended to support other thruster control interfaces, such as CAN Bus, in the future.

B. Practical Implementation

Our implementation accounts for real-world constraints through:

Current Limiting: We use a binary search algorithm to find the maximum scalar α such that:

$$\sum_{i=1}^{n} I(\alpha \cdot f_i) \leq I_{\text{max}} \quad \text{where} \tag{3}$$

- $I(f_i)$ is a function that returns the current required for thruster i to produce force f_i, obtained from the thruster performance lookup table (see next section);
- α is a scalar multiplier applied to all thruster forces equally;
- I_{max} is the maximum allowable total current draw.

Thruster Performance Modeling: Thruster dynamics are modeled through lookup tables that relate the output signal, current draw, and thrust. Such data is readily available from the manufacturers of common ROV thrusters. This approach allows our system to adapt to:

- Nonlinearities between the output signal and the force produced by the thruster;
- asymmetric forward and reverse thrust curves inherent to typical propeller designs;
- thruster failures or reconfiguration by updating the T matrix to exclude failed thrusters;
- and enables us to control the ROV using absolute units (e.g. newtons) instead of relative units (e.g. percent of maximum thrust output).

V. Experimental Validation

We validated our system in a swimming pool using a modified BlueROV2 platform across four different thruster configurations. The robot was deployed about $30cm$ below the water surface.

A. Experimental Setup

The standard BlueROV2 has six thrusters: Four horizontal and two vertical. It can move in five degrees of freedom (lacking independent pitch control).

In our experimental setup, we consider the following four configurations:

1) Standard BlueROV2 configuration with all four horizontal thrusters active.
2) Three horizontal thrusters are active (one is disabled).
3) Two horizontal thrusters are active (two are disabled).
4) Modified BlueROV2 configuration with one horizontal thruster repurposed as a vertical thruster on the side, enabling both roll and pitch control.

The fourth configuration was only tested for roll and pitch control capabilities since it shares the same planar thruster placement as configuration two. For each configuration, we commanded the ROV to follow an oscillating orientation setpoint and measured the error between the commanded and actual attitudes over time.

B. Experimental Results

Figures 2 and 3 shows the error distribution for each configuration and control axis. On the top left of each figure we show the thruster configuration on the BlueROV2. The results demonstrate that:

1) Yaw control remains robust even as horizontal thrusters are disabled, with control error remaining below 2° in all configurations tested.
2) Roll and pitch control with the modified thruster configuration shows higher variance than yaw with standard deviations of 1.283° and 1.228° respectively.
3) The error distributions closely follow normal distributions, indicating well-behaved control characteristics.

These results validate our system's ability to maintain effective control across different thruster configurations without requiring code changes. The significantly larger standard deviations in roll and pitch control (1.283° and 1.338° compared to around 0.5° for yaw) are expected due to the asymmetric thruster placement and competing buoyancy effects.

VI. Conclusions

This paper presented two key innovations for underwater ROV control: (i) An ECS-based architecture with automatic state synchronization; and (ii) a generalized thruster control system adaptable to arbitrary configurations. Our experimental results demonstrate that this approach enables effective control even when thrusters are disabled or repurposed, addressing a significant limitation of traditional ROV software architectures.

The ECS architecture shown in Figure 1 provides clear separation of concerns while maintaining system cohesion through automatic state synchronization. The generalized thruster control approach demonstrated in our experiments shows compelling robustness, with comparable control performance across a variety of modified configurations and expected increases in variability for degrees of freedom with less control authority.

Future work will focus on:

1) Extending our implementation to enable support for multi-ROV configurations.
2) Implementing visual-inertial odometry for improved localization.
3) Developing autonomous behaviors leveraging our modular architecture.

The flexibility demonstrated by our system provides a foundation for more robust and adaptable underwater robotics applications, particularly in scenarios where hardware reconfiguration or failures are likely.

VII. Software Availability

The source code for the software architecture described in this paper is available as an open-source project at https://github.com/Eoghanmc22/mate-rov-2025. The repository includes the control system implementation, thruster control algorithms, and configuration files used in our experiments.

References

[1] J. S. Willners, I. Carlucho, T. Łuczyński, S. Katagiri, C. Lemoine, J. Roe, D. Stephens, S. Xu, Y. Carreno, Èric Pairet, C. Barbalata, Y. Petillot, and S. Wang, "From market-ready rovs to low-cost auvs," *arXiv preprint arXiv:2108.05792*, 2021. [Online]. Available: https://arxiv.org/abs/2108.05792

[2] ArduSub Development Team, "ArduSub: Open-Source Submarine Control Software," https://www.ardusub.com/, 2024, accessed: March 9, 2025.

[3] V. B. Theja, P. Yachameni, S. Shakeera, and H. Venkataraman, "Integration of gazebo and ros for underwater vehicle environment," in *OCEANS 2022 - Chennai*, 2022, pp. 1–6.

[4] S. Mayberry, J. Wang, Q. Tao, F. Zhang, A. Song, X. Hong, S. Dong, C. Webb, D. Dugaev, and Z. Peng, "First step towards net: Open-access aquatic testbeds and robotic ecosystem," in *Proceedings of the 15th International Conference on Underwater Networks & Systems*, ser. WUWNet '21. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3491315.3491322

[5] M. Doniec, I. Vasilescu, C. Detweiler, and D. Rus, "Complete se3 underwater robot control with arbitrary thruster configurations," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 5295–5301.

[6] M. Doniec, C. Detweiler, and D. Rus, *Estimation of Thruster Configurations for Reconfigurable Modular Underwater Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 655–666. [Online]. Available: https://doi.org/10.1007/978-3-642-28572-1_45

[7] L. Cai, K. Chang, and Y. Girdhar, "Learning to swim: Reinforcement learning for 6-dof control of thruster-driven autonomous underwater vehicles," *arXiv preprint*, vol. arXiv:2410.00120, 2024. [Online]. Available: https://arxiv.org/abs/2410.00120