RL Project Report

Team Offset

Abhinav Lodha, Anshuman Sharma, Nikhil Henry, Subham Jalan

Abstract: Teaching a BlueROV2 how to do cool underwater maneuvers.

April 30, 2025

1. Probem Statement

BlueROV2 is an underwater remotely operated vehicle (ROV) developed by Blue Robotics with **5 DOF** (surge, sway, heave, roll, yaw). This configuration allows for agile maneuvering in complex underwater environments. However, manually controlling such a high-DOF system becomes increasingly challenging due to a large state space and the inherent uncertainties in underwater dynamics. Manual control under these conditions is often inadequate for performing **complex trajectories.** In hydrobatics, where the ROV is expected to perform advanced maneuvers like **barrel rolls, corkscrews, spirals or sudden rotations,** a precise control is critical. This is useful in areas where agile control is required, like emergency rescue, inspection, monitoring, etc.

Given these complexities, we aim to develop a reinforcement learning based **control system** capable of learning to **maneuver the BlueROV2** autonomously and robustly under dynamic conditions.

Our goal is to experiment with by training various RL agents in simulation environment on such **non-trivial trajectories** with the aim of understanding whether a generalizable policy is feasible or a tailored solution is required.

2. RL Formulation

- **State Space**: We have 2 different state spaces -
  - An *unmasked state*, which is the complete description of the agent, for our dynamics model to compute next state, and a *masked state* called observation, used only for training the RL agent. The unmasked state (s') has the components: $\left[x, y, z, \theta, v_x, v_y, v_z, \omega\right]$. The ranges of the cartesian axes, orientation and corresponding rates come from the simulation bounds and time step ($\partial t = 0.1$).
  - A ***masked*** **observation space (s),** is one in which we replace (x, y, z, θ) by $(offset_x, offset_y, offset_z, offset_\theta)$, that are computed by taking the difference between current pose and desired pose. [*Section 3*]. We are not using the absolute pose of the robot in the observation space because we want the RL agent to learn movements with respect to a waypoint, and not in the general un-anchored space. Additionally, these values are normalized inherently during training in the Gymansium environment.
- **Action Space**: In our simulation, we are using only 4DOF - Surge, Yaw, Heave, Sway (pitch not included) because of the absence of its dynamics in literature and difficulty of computing accurate numbers in experiments. We have PWM values in range [1000, 2000] normalized to [−1, 1] for each RC/DOF. Hence actions come from $[-1, 1]^4$.

3. Methodology

3.1. Approach

In order to make the robot follow a trajectory, there are **2 approaches** that we are simultaneously experimenting with after studying the literature. In both, the methodology is to move the robot from one waypoint to another, and then reset s, or the state of the robot, once it is within a certain threshold of position and angle. This means that the robot is always motivated to keep **minimizing its positional and angular errors** w.r.t to the waypoint just ahead and the absolute pose in unmasked state s' becomes irrelevant.

- **Generalizable Policy** : The idea is to train a robust policy by initializing the robot at s = $[0, 0, 0, \theta, 0, 0, 0, 0]$ where $\theta \sim Uniform(-2\pi, 2\pi)$ and asking the robot to reach random poses, where the position is sampled from a 3D sphere of radius $R \sim Uniform(0.5, 2)$ and target angle $\phi \sim Uniform(-2\pi, 2\pi)$. The observation state s is computed using the random pose as a waypoint. The RL agent is then tasked to reach these states (waypoints) by minimizing the offsets. Our belief is that such a **generalizable policy** can then be **iterated over waypoints of a trajectory** to go from each point to the other by repeatedly applying the policy. By teaching the robot how to reach random

observation states, we aim to learn a good control model that can move the robot from one random state (s_1') to another (s_2') by the grace of s_1 and s_2.

- **Specific Policy**: In this, we are taking 2 different approaches. One is to **sample two adjacent points** from one trajectory and use them as s_1' and s_2'. This should train the RL agent on all small goals of an identified trajectory and allow the robot to traverse it. This exposes the robot to vectorized offsets between the points of a known trajectory and **reduces the effect of generalizing**. The only difference in this approach as compared to the generalizable policy is that now we restrain the ranges from which the random one-look ahead waypoint is sampled. Another way is to frame the reward function to take into account **more than one waypoint ahead (possibly all)**. This would mean that s would not be w.r.t a certain waypoint but the entire trajectory itself. Framing this reward function is difficult, and we are experimenting with that right now.

We are currently working on sampling $v_x, v_y, v_z,$ and ω from a Uniform distribution (for s_1 or the start state of the vehicle) on their respective ranges that occur in the simulation. It will enable the robot to follow the trajectories more seamlessly.

3.2. Experiments

To evaluate our reinforcement learning approach for underwater ROV trajectory tracking, we conducted a series of experiments using different control strategies and learning algorithms. The progression of experiments followed a systematic approach from traditional control methods to advanced reinforcement learning techniques.

3.2.1. Ultra-Short Base Line, (USBL - Hardware)

We used Cerulean Sonar's ROV Locator Mk III°, which is an Ultra-Short Base Line to record the velocities and positions of an underwater ROV in real-time.

We equipped the BlueROV2 with USBL but the readings were too noisy and unsuitable for use. This was expected as it is designed typical usage range of ($\sim 100m$) is well above our lab's tank size. This motivated us to take a software-first approach and before the deployment of RL algorithms to real world.

3.2.2. Trajectory Generation

We have created different trajectories -

- **Spiral**: A helical path with constant radius and linear depth descent.
- **Lemniscate**: A figure-eight pattern with sinusoidal x, y variations and linear z progression.
- **Square**: A four-sided path with straight segments and sinusoidal depth oscillation.
- **Straight Line**: A linear path along the x-axis with constant y, z coordinates.

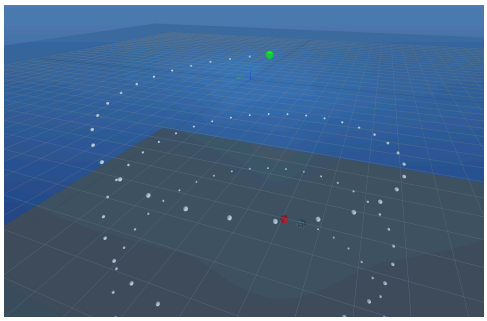

Figure 1: Spiral

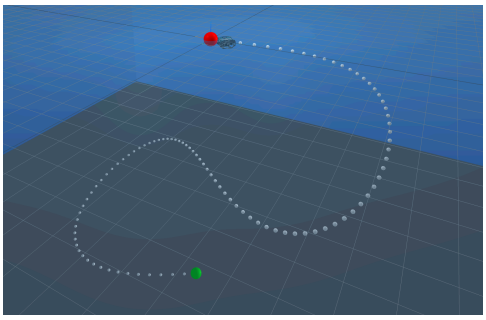

Figure 2: Lemniscate

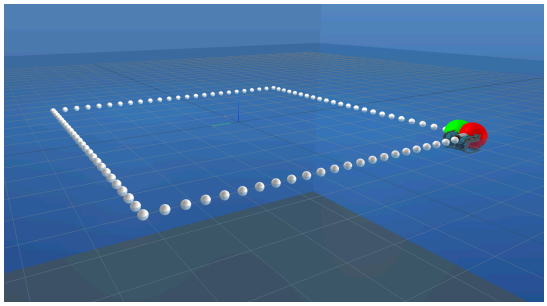

Figure 3: Flat Square

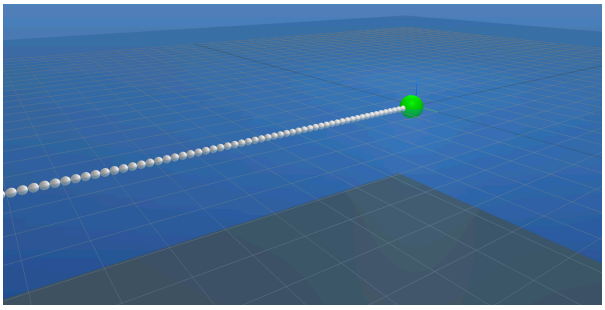

Figure 4: Straight Line

The equations for these can be found in the codebase.

3.2.3. PID Controller

We first established a baseline performance using a **traditional PID** (Proportional-Integral-Derivative) controller to assess the difficulty of the task and provide a reference point for our RL approaches.

A key challenge was mapping position errors to the BlueROV2's control inputs. We implemented the following transformation:

Surge control: $a_1 = e_x \cos(\theta) + e_y \sin(\theta)$

Sway control: $a_2 = -e_x \sin(\theta) + e_y \cos(\theta)$

Heave control: $a_3 = e_z$

Yaw control: $a_4 = e_\theta$

where e_x, e_y, e_z, e_θ are the errors in the three axes and heading.

The controllers were tuned with the following parameters:

- Position controller ($x \mid y \mid z$): $K_p = 1.0$, $K_i = 0.0$, $K_d = 0.0$
- Heading controller (θ): $K_p = 0.8$, $K_i = 0.0$, $K_d = 0.0$

The PID controller performed well after tuning, but deriving the mapping from error to control inputs was challenging due to nonlinear DOF coupling and complex underwater dynamics. Each trajectory required manual, non-generalizable tuning, and performance dropped on sharp or complex paths.

3.3. Reinforcement Learning Approaches

Based on the limitations of traditional control methods, we proceeded with two popular on-policy RL algorithms: *Proximal Policy Optimization (PPO)* and *Advantage Actor-Critic (A2C)*.

We selected **PPO** and **A2C** for the following reasons:

- **Sample Efficiency:** On-policy methods like PPO and A2C are generally more **sample-efficient** than off-policy methods for continuous control tasks with complex dynamics.
- **Stability:** Both algorithms provide good stability during training through trust region optimization (PPO) or synchronized advantage estimation (A2C).
- **Continuous Action Space:** Both algorithms handle continuous action spaces naturally, which is essential for precise ROV control.
- **Parallelization:** Both models benefit from parallel environment execution, allowing more effective training with our available computational resources.

3.3.1. Training, Evaluation, Termination, Success

3.3.1.1. Training

Our training configuration was standardized across experiments:

- Total timesteps (`n_steps`): 1,000,000

- Parallel environments (`n_envs`): 16
- `MAX_EPISODE_LENGTH` : 50
- Neural network architecture: `MultiInputPolicy` (default in Stable Baselines3)
- Batch size: 64 (for PPO)
- Discount factor (γ) = 0.99
- Normalization: `VecNormalize` for running averages of observations and rewards

3.3.1.2. Evaluation

Evaluation episodes `n_eval_episodes` : 20

For evaluation, we consider three metrics -
- *Success Rate* - #(robot reached waypoint(s_2'))/#(episodes)
- *Mean Episode* Return - Average return obtained by the agent per episode over the evaluation runs.
- *Mean Episode Length* - Average number of timesteps taken per episode during evaluation.

The *best* model is the one with the highest *Mean Episode Reward*. However, we aim to use the model with the lowest $\mu(\textit{episode length}) + \sigma(\textit{episode length})$ for testing. Intuitively, the fastest model should be the one that takes the shortest path.

3.3.1.3. Termination & Success

Some defined constants:
- $\textit{position\_threshold} = 0.1m$
- $\textit{angle\_threshold} = 0.1rad$
- $\textit{world\_bounds} = (-15m, 15m)$ in x, y and $(-10m, 10m)$ in z
- $\textit{away\_bound} = 0.5m$

Termination Condition:
1. Episode rollout length becomes equal to `MAX_EPISODE_LENGTH` .
2. Robot leaves the world's bounds.
3. $\|(s_1, s_2, s_3)\|_2 > d + \textit{away\_bound}$, where d is distance from goal at $t = 0$.

Success Condition:

$$\left(\|(s_1, s_2, s_3)\|_2 < \textit{position\_threshold}\right) \wedge (s_4 < \textit{angle\_threshold}) \tag{1}$$

3.3.2. Reward Function Shaping and terrors of agent reward hacking!

The most challenging aspect of our RL formulation was designing an effective reward function. We experimented with several reward structures (listed in increasing order of performance):

- **Test 1**: Negative L2 Norm Reward

  The negative squared distance term penalizes the agent more the farther it is from the goal, encouraging spatial convergence.The angular penalty ensures alignment with the desired final heading. The high bonus rewards the agent for fully completing the task, which helps it recognize the value of goal-reaching behaviors. Most importantly, this intrinsically modelled time and speed which theoretically pushed agent to finish/terminate the run as soon as possible.

  This is defined as

$$r = -d^2 - 0.1\theta_{\text{offset}}^2 + r_{\text{completion}} \tag{2}$$

  where:
  - $d = \|(s_1, s_2, s_3)\|_2$
  - $\theta_{\text{offset}} = s_4$ is the angular offset from the goal orientation.

‣ $r_{\text{completion}} = 1000$ if success condition is met, 0 otherwise. [Equation 1]

The agent struggled to learn meaningful policies because the large completion bonus only kicks in at success, creating a sparse reward signal that the agent may rarely experience during early training. While the L2 norm penalizes distance, it doesn't tell the agent how to move toward the goal. This lead to slow convergence.

- **Test 2**: Exponential Positive Reward
  After struggling with the sparse and largely negative signal of the previous reward, we redesigned it to be positively shaped and more informative at all stages of the task:

$$r = e^{-d^2} + 0.1e^{\theta^2_{\text{offset}}} + r_{\text{completion}} \tag{3}$$

  This formulation gave the agent a reward signal that always stayed positive and increased smoothly as it got closer to the goal — both spatially and in orientation. This reward function was inspired by Cai et al. (2025).

  Initially, the reward function incentivized the agent to remain close to the goal without efficiently reaching it. The vehicle would linger near the goal area and receive substantial cumulative reward despite not completing the task optimally. Apart from that, the exponential terms flatten out near zero offset, giving diminishing returns. This caused the agent to stop improving fine-grained control once it's close enough. To address this issue, we implemented a negative time penalty that reduced the reward for each timestep the agent wasn't at the goal position. Our intention was to encourage more direct, efficient paths to the goal. However, this modification produced unexpected behavior and didn't make the agent learn the controls.

  This outcome suggests our reward function may need further refinement to properly balance proximity rewards with time efficiency incentives.

- **Test 3**: Progress Reward

$$r_t = r_{\text{completion}} + e^{-d^2} + e^{-\theta^2_{\text{offset}}} + 20(d_{t-1} - d_t) + r_{\text{termination}} \tag{4}$$

  $r_{\text{termination}} = -500$ is a new reward which is given when termination occurs due to condition 2 or 3.

  This reward function should work because we are incentivizing the RL agent to make positive progress *towards* the goal at every time step. This is giving us better results as compared to the last reward function. However, we still need better time penalties and fine-tuning of `MAX_EPISODE_LENGTH` to make the robot take the shortest route towards the goal.

3.4. Results

Using the reward mentioned in Equation 4 on single waypoint sampling according to generalizable policy -

	PPO	**A2C**
Training Time	$\sim$ 240s	$\sim$ 90s
Success Rate	100%	100%
Average Episode Return	902.19 $\pm$ 16.19	1032.17 +/- 8.88
Average Episode Length	25.45 $\pm$ 9.03	18.35 +/- 5.58

All training was conducted on an Apple Silicon M2 processor with 16GB RAM . Notably, utilizing GPU resources (CUDA/Metal) did not significantly improve training time, as the Stable Baselines3 implementations of these algorithms are highly optimized for CPU execution.

We limited training to 1,000,000 timesteps after observing clear convergence in both algorithms. Learning curves indicated that performance plateaued after approximately 800,000 steps, with only marginal improvements thereafter. This convergence behavior suggests that the policies had effectively learned the optimal control mapping for the underwater navigation task within the given environmental constraints.

We tried other algorithms like **SAC**, **TD3**, and **DDPG** but those did not show promising results. This was primarily because of slower training times and slower convergence of the reported metrics.

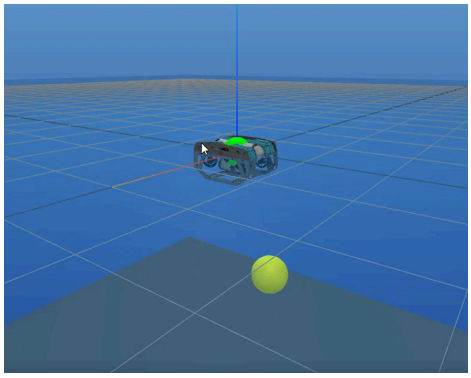

Figure 5: Testing learned policy on a single waypoint sampling according to generalizable policy

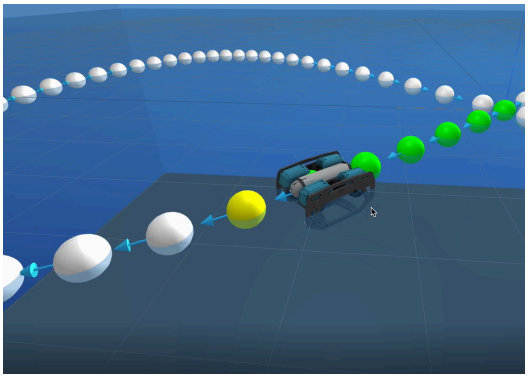

Figure 6: Testing learned policy on a flat lemniscate trajectory

Testing on full trajectories revealed suboptimal performance. While the agent was able to follow the first 5–10 waypoints reasonably well, it gradually lost control due to accumulated momentum and increasing drift. Over time, the deviation from the intended path compounded, ultimately causing the agent to veer off course and terminate upon reaching the boundary of the environment.

3.5. Conclusion

A2C with exponential rewards has given us good results till now. We plan to use the same framework and do the following before presentations:

1. Further refining the reward function for faster waypoint navigation.
2. Initializing s_1 (start state) with random velocities in position and orientation. This will help in the training of a robust generalizable policy.
3. Adding more disturbances in the hydro-dynamics model to make PID control difficult.
4. Do more experiments to mitigate the numerical stability issues of negative rewards.
5. Make trajectories denser and waypoints closer for a more complicated learning objective.

4. Contributions

Following is the split of work that we did.

- Anshuman Sharma
  - Tested the Gymansium environment and a sample training loop for BlueROV2 stability.
  - Optimized the ML training.
  - Ran experiments on reward shaping.
- Abhinav Lodha
  - Theorized the observation space.
  - Wrote the waypoint visualization and PID controller.
  - Ran **rigorous** experiments on reward shaping.
- Nikhil Henry
  - Added support for random goal points and trajectories to the environment.
  - Created foundation for training pipeline and generated trajectories.
  - Did literature survey and implemented optimal reward functions.
  - Handled the normalization of the spaces for model convergence.
- Subham Jalan
  - Assisted Abhinav Lodha in reward shaping and RL formulation with mathematical grounding.

- Wrote evaluation script for metrics and graphs.
- Ran initial experiments on the RL agent for training.

All of us were equally involved in the ideation of our approach and methodology.

5. Challenges

- The BlueROV2 Gymnasium was initially developed by Gokul° (a PhD student at the University of Illinois Urbana-Champaign (UIUC)). We had difficulty understanding the dynamics model, post which we reached out to him and he was very generous to share a document with us which contained the dynamics functions for each axis but did not present their methodology to create the hydrodynamics models.
- The Gymansium environment was set up on an old API, which is sometimes not compatible with SB3. Fixing these errors took time.
- It is difficult to train RL models for continuous action spaces because of their high computation, large search space and time cost.
- We had issues with the propoer normalization of our environments. Our results were not consistent before we fixed the issue.

6. References

- GitHub: https://github.com/Offset-official/bluerov_hydrobatics.git°
- BlueROV2: https://bluerobotics.com/store/rov/bluerov2/°
- Gymansium: https://gymnasium.farama.org/index.html°
- Stable Baselines3: https://stable-baselines3.readthedocs.io/en/master/°
- Multimedia: SharePoint link°

Bibliography

Cai, L., Chang, K., & Girdhar, Y. (2025,). *Learning to Swim: Reinforcement Learning for 6-DOF Control of Thruster-driven Autonomous Underwater Vehicles*. https://arxiv.org/abs/2410.00120°

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2006). Autonomous Inverted Helicopter Flight via Reinforcement Learning. In M. H. Ang & O. Khatib (Eds.), *Experimental Robotics IX: Experimental Robotics IX*.

Sutton, R. S., Barto, A. G., & others. (1998). *Reinforcement learning: An introduction* (Vol. 1, Issue 1). MIT press Cambridge.