The Report Committee for Justin Chad Hart
Certifies that this is the approved version of the following Report:

# Reinforcement Learning for Adaptive Control of an Unmanned Surface Vehicle

APPROVED BY

SUPERVISING COMMITTEE:

Dongmei (Maggie) Chen, Supervisor

Raul G. Longoria

# Reinforcement Learning for Adaptive Control of an Unmanned Surface Vehicle

**by**

**Justin Chad Hart**

**Report**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Engineering**

**The University of Texas at Austin**

**May 2025**

# Acknowledgments

First, I would like to thank Dr. Chen for her support throughout the writing of this report and for taking the time to review its contents. I would also like to thank Dr. Longoria for agreeing to review this report. I would like to thank my family and friends for their love and support. Thank you to my wife, Vanessa, for enabling me to pursue my goals and for continually motivating me throughout this process.

# Abstract


## Reinforcement Learning for Adaptive Control of an Unmanned Surface Vehicle

Justin Chad Hart, M.S.E.
The University of Texas at Austin, 2025

Supervisor: Dongmei (Maggie) Chen

Unmanned Surface Vehicles (USVs) are being used across an array of industries, the automation of which contributes to their appeal. A critical component of their automation is the course keeping task which is the ability to adjust its heading angle to the desired heading and hold it. One method that can be applied to execute this task is a reinforcement learning (RL) - PID controller, which combines the advantages of the intuitiveness of a PID controller, and the robustness of RL in dynamic environments, while minimizing the difficulty in parameter tuning of a PID controller, and the black-box nature of RL. An adaptive RL-PID controller is presented in this report, utilizing the proximal policy optimization RL algorithm. Its performance is verified in both course keeping and path following evaluations and the adaptive controller accomplishes these tasks as well and slightly better than a fixed PID controller. More work should be done to fine tune the RL training and to conduct the training on higher end hardware to produce a policy that greatly exceeds the performance of a fixed PID controller.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1: Introduction

Autonomous oceanographic vehicles, in particular, unmanned surface vehicles (USVs), are being used across multiple industries as critical assets to extend capabilities and lessen the burden and responsibility placed on personnel. These include in the defense industry for shore patrol and situational awareness, the oil and gas as well as energy industries for inspection and exploration, and in oceanographic research for surveying and monitoring. One example is the Otter, developed by Maritime Robotics, that is designed for environmental monitoring and surveillance in shallow water, and has also been deployed to the Ukrainian Navy for data collection [8]. This limits the danger posed to Ukrainian Naval personnel while enhancing their security and defense capabilities.

Key systems that allow for the safe and effective autonomous operation of the USV include path planning, obstacle avoidance, navigation, and course following, elements of the guidance, navigation, and control system. The course following component is the focus of this report, especially in the presence of varying water current disturbances.

Course following control methods that have been implemented include the traditional proportional, integral, derivative (PID) controller (7), optimal methods such as the linear quadratic regulator (LQR), and nonlinear control methods like sliding mode control (SMC) and backstepping (8). Reinforcement learning (RL) methods have been implemented but ultimately face the challenge of the lack of insight into the workings of an RL policy, and the risk of the policy choosing unpredictable decisions in real world environments (3). To mitigate the shortcomings of the RL only controller, (9) and (3) combine PID and RL to form an RL-PID controller for the course following problem of a USV and autonomous underwater vehicle (AUV) respectively.

The objective of this report is to replicate the RL-PID course following controller presented in (9), using the proximal policy optimization (PPO) RL algorithm. This will be simulated on an Otter USV which is controller through two stationary propeller rotational velocities, whereas the USV in (9) is controlled through rudder angle commands.

The work presented in this report will first be related to previous research related to the problem, then specific USV modeling and the implemented guidance law connecting positional errors to angular errors are described. The general RL framework is next presented, and the PPO algorithm is explained. The RL environment and training setup are summarized and then results in course keeping and path following for the RL-PID course following controller are reviewed.

# Chapter 2: Literature Review

Path following control is a key function of any autonomous vehicle, and unmanned surface vehicles are no exception. Traditional methods, such as PID controllers, have been used frequently to fulfill this role, and generally perform well (7). The downfall of the PID controller is the difficulty in tuning for performance in complex environments. This can be time consuming and difficult, requiring extensive testing and iteration.

A summary of various AUV path following control methods that have been utilized as of the date of its publication (2021) are given in (8). These include optimal methods – linear quadratic regulator (LQR), state dependent Riccati equation (SDRE), and model predictive control (MPC); as well as nonlinear methods such as sliding mode control (SMC) and backstepping. There are also adaptive control methods like fuzzy control, neural network based control, and deep reinforcement learning for control. Some of the weaknesses of each method as described in (8) include a lack of robustness for LQR, a heavy computational burden for MPC, and a lack of systematic methods for fuzzy control.

Although various control methods have been implemented on autonomous at sea vehicles (both underwater and surface vehicles), the PID control method remains the industry standard due to a combination of ease of implementation, low compute requirements, and it's clear physical interpretation. The issue with PID control is that it can be time consuming and difficult to adapt to complex environments. This can take many iterations of coefficient tuning as well as real-world testing in varying conditions in order to dial in an adequate controller.

As reinforcement learning (RL) has garnered popularity within the field of machine learning, it has been explored as an optimal control solution for complex systems. One advantage of the use of RL for control is the elimination of a reliance

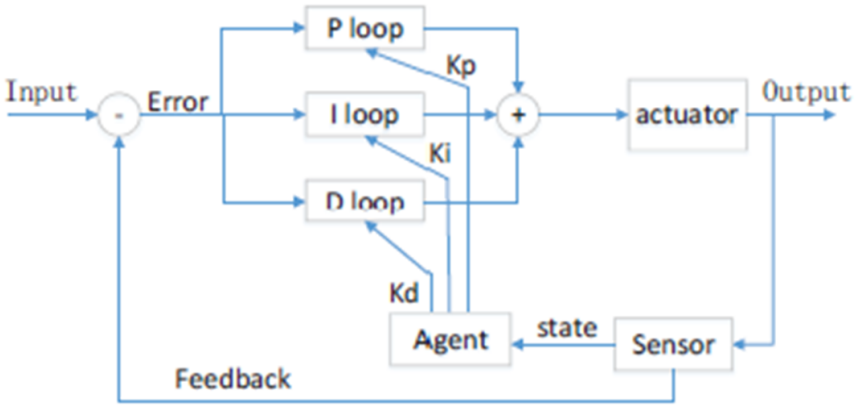


Figure 2.1: RL-PID Controller Diagram (1).

on dynamic models of the system you're trying to control (10). This is especially beneficial because it can be difficult to identify the model parameters and overall structure of complicated models. RL has been applied to path following control of a USV in (10) and the unknown nonlinear dynamics of the USV tested were addressed. One major roadblock to the use of RL control methods on actual hardware is the lack of understanding of the internal workings of an RL policy. This has a two-fold effect in the sense that the policy can make unpredictable decisions in the presence of real-world environments, and an understanding of the cause of these actions can be difficult to grasp.

A control method to combine the advantages of PID and RL control, while mitigating the disadvantages of each, is the RL-PID controller as proposed in (9). In this paper, RL is used to train an agent to output PID parameters based on the current state as shown in Figure 2.1. The RL-PID algorithm is used to control an inverted pendulum in OpenAI gym and performs better than a fixed PID controller in the same environment.

RL-PID controllers have recently been explored for use on autonomous oceanographic vehicles. A path following controller combining RL and PID methods for AUVs is developed in (3) that shows promising performance over fixed PID controllers. They are also applied to path following controllers for USVs in (9), specifi-

cally for a rudder controlled USV model.

# Chapter 3: Methods

## 3.1 USV Modeling

The coordinate system reference frames utilized in the modeling of a USV are the North-East-Down (NED) coordinate system, the body-fixed reference frame, and the path frame. The NED system is the inertial frame fixed to the earth. The body-fixed reference frame is the moving coordinate frame fixed to the USV. The path frame is the coordinate frame that is fixed to the desired path of the USV, and changes as each consecutive waypoint becomes the new starting point for the intermediate path. Visualizations of each reference frame are shown in Figure 3.1.

Equations of motion for a 6 degree of freedom (DOF) marine craft expressed in body coordinates are given in (2) as:

$$\dot{\eta} = J_{\Theta}(\eta)v_r \tag{3.1}$$

$$M\dot{v}_r + C(v_r)v_r + D(v_r)v_r + g(\eta) + g_0 = \tau \tag{3.2}$$

Where $\eta = [x, y, z, \phi, \theta, \psi]^T$ is the vector comprising the position and orientation of the USV, and $\nu = [u, v, w, p, q, r]^T$ is the vector of linear and angular velocities. The relative velocity vector, $\nu_r = \nu - \nu_c$, is the difference between the velocity of the USV and the water current velocity, $\nu_c$. The relative velocity vector can be used in the equations of motion only when the water current velocity is irrotational, i.e., possessing only linear velocity components, which is the case for this report. $J_{\Theta} \in R^{6x6}$ is the Euler angle coordinate system transformation matrix. $M \in R^{6x6} = M_{RB} + M_A$ is the system inertia matrix which is the combination of the rigid body mass matrix and the added mass matrix (virtual mass added to the system due to a body moving through a fluid). $M_{RB}$ is calculated based on the rigid-body forces acting on the USV due to it's inherent mass, whereas $M_A$ is typically calculated using a hydrodynamic computer simulation. $C(\nu_r) \in R^{6x6} = C_{RB}(\nu_r) + C_A(\nu_r)$ is the coriolis-centripetal

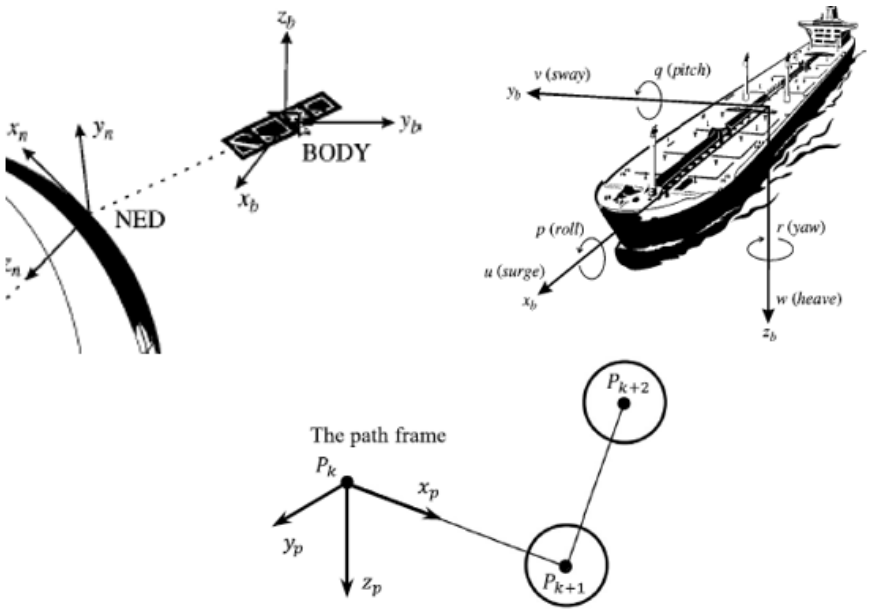


Figure 3.1: Reference frames used in modeling of a USV (2),(3).

matrix, accounting for the forces due to the rotation of the body fixed frame relative to the NED inertial frame. Similar to the system inertia matrix, this includes both rigid body and added mass terms. $D(\nu_r) \in R^{6x6}$ is the damping matrix, and just like the added mass terms, is typically calculated using hydrodynamic simulations. It usually consists of potential damping, skin friction effects, vortex shedding, and lifting forces. These are all described in greater detail in (2). $g(\eta) \in R^6$ is the vector of gravitational/buoyancy forces and moments and $g_0 \in R^6$ is the vector of ballast forces if applicable. $\tau = [X, Y, Z, K, M, N]^T$ is the vector of control inputs.

## 3.2 Path Following Control

### 3.2.1 Incremental PID Controller

In an effort to replicate the results of (9), the incremental PID controller is applied to the USV control problem as in (9). The advantage of the incremental PID controller over the positional PID controller is the zero accumulation of error, as the output is the incremental control output from the current state relative to the desired

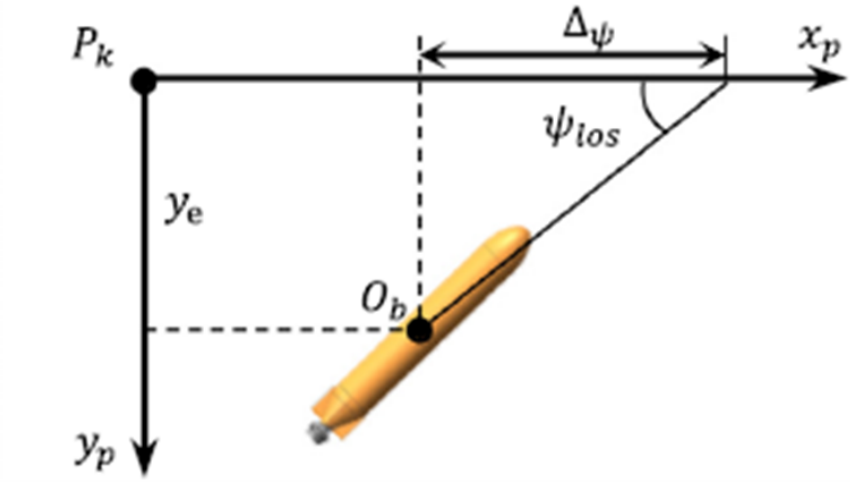


Figure 3.2: 2-D LOS diagram (3).

setpoint. The incremental controller equation is given below:

$$\begin{aligned} \Delta u(t) &= K_P[e(t) - e(t-1)] + K_I e(t) + K_D[e(t) - 2e(t-1) + e(t-2)] \\ u(t) &= u(t-1) + \Delta u(t) \end{aligned} \tag{3.3}$$

$K_P, K_I$ and $K_D$ are the proportional, integral, and derivative coefficients of the controller respectively. The control output is $u(t)$, while $\Delta u(t)$ is the incremental control output. $e(t)$ is the error term and is the difference between the current state and the desired setpoint.

### 3.2.2 Line of Sight (LOS) Guidance Law

The LOS guidance law as presented in (2) is used to convert a path following task to a control task, by comparing the position and orientation of a vehicle with respect to the desired path between waypoints and sending a desired heading term. This can then be used to calculate an angular error by taking the difference of the desired and current heading. This error term is then used as the input into the control framework to guide the vehicle along a desired path.

Referencing Figures 3.1 and 3.2, the starting point of the current path is given as $P_k = (x_k, y_k)$ and the next waypoint in the desired path is $P_{k+1} = (x_{k+1}, y_{k+1})$.

The LOS guidance yaw angle can then be calculated using the following:

$$\psi_{LOS} = \arctan\left(\frac{\Delta y}{\Delta x}\right) \tag{3.4}$$

Where $\Delta y = y_{k+1} - y_k$ and $\Delta x = x_{k+1} - x_k$. If the current position of the USV is given as $O_b = (x_0, y_0)$, then the position error of the USV with respect to the desired path is given as:

$$y_e = -(x_0 - x_k) * sin(\psi_{LOS}) + (y_0 - y_k) * cos(\psi_{LOS}) \tag{3.5}$$

Using the calculated LOS yaw angle and position error, the desired yaw angle for the USV is found using:

$$\psi_d = \psi_{LOS} - arctan(y_e/\Delta\psi) \tag{3.6}$$

Where $\Delta\psi$ is a lookahead distance, and is a parameter set at the discretion of the control system designer. Using the desired yaw angle, the angular error term between the current USV yaw angle and desired yaw angle can be calculated and used as input into the PID controller:

$$e_\psi = \psi_d - \psi \tag{3.7}$$

## 3.3 Reinforcement Learning

### 3.3.1 Overview

RL can be broken down simply into the process of interacting within an environment and learning from this interaction how to act in order to achieve a goal. An introduction and comprehensive summary of RL is given in (4) and is used as the reference for the following explanation. As shown in Figure 3.3, the main process within RL is the interaction of an agent within an environment. The agent (learner and decision maker) observes the state of the current environment (everything outside

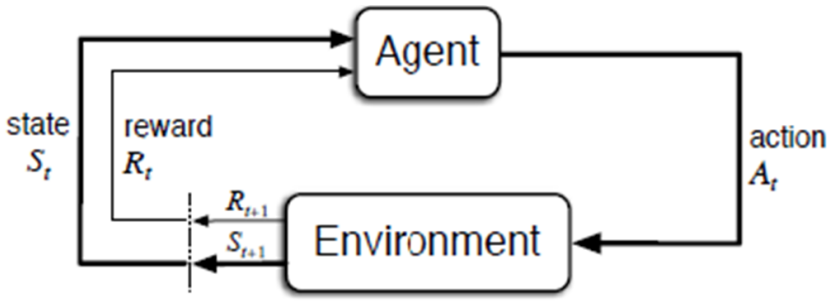


Figure 3.3: Agent-environment interaction loop (4).

of the agent), along with a reward signal based on how good the state is and then decides on an action to take which changes the environment.

An observation is the partial description of the state, and the set of all feasible actions within an environment is the action space. The policy is the rule the agent uses to select actions based on the observed state, and trajectories are the sequence of states and actions within the environment. The return is the sum of all rewards obtained by the agent, where the reward is a function of the trajectory of the environment. The goal of RL is to find a policy that when an agent follows this policy, the expected return is maximized. To find the expected return, the probability of a T-step trajectory is determined first:

$$P(\tau \mid \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t)\pi(a_t \mid s_t) \tag{3.8}$$

Where $\rho_0$ is the start state distribution, $s_0$ is the initial state, and $s_t$ and $s_{t+1}$ are the states at time $t$ and $t+1$ respectively, and $a_t$ is the action at time $t$. Next, the expected return can be found using the following equation:

$$J(\pi) = \int_{\tau} P(\tau \mid \pi)R(\tau) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \tag{3.9}$$

Where $\pi$ is the stochastic policy, $\tau$ is the trajectory, and $R(\tau)$ is the reward function. The optimization problem of RL is summarized as:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{3.10}$$

Where $\pi^*$ is the optimal policy.

### 3.3.2 Proximal Policy Optimization

The PPO algorithm, first proposed in (11), aims to solve the RL problem by improving a policy with current data, without degrading the performance of the policy (keeping policy updates 'reasonable'). Clipping is implemented within the objective function to prevent large jumps in policy. PPO trains the policy using an on-policy method, choosing actions based on the current version of its policy.

Policies are updated according to the following equation, using stochastic gradient descent to maximize the objective:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{(s,a)\sim\pi_{\theta_k}} \left[ L(s, a, \theta_k, \theta) \right] \tag{3.11}$$

Where $\theta$ are the policy parameters, and $L$ is the surrogate advantage function. This measures the performance of the new vs. old policy. The surrogate advantage function is given by:

$$L = \min \left( \frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s, a), g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right) \tag{3.12}$$

Here $A$ is the advantage function, corresponding to policy $\pi$, which quantifies how much better it is to take a certain action in the current state, over randomly selecting an action according to the policy, $\pi$. The gradient of the surrogate advantage function, $g(\epsilon, A)$ is as follows:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & \text{if } A \geq 0 \\ (1-\epsilon)A & \text{if } A < 0 \end{cases} \tag{3.13}$$

Where $\epsilon$ is a parameter that controls the allowable distance of the new policy from the old. Pseudo code for the PPO algorithm is given in Figure 3.4.

The specifics of how each of the previous methods described are used in this report are outlined in the following chapter.

**Algorithm 1** PPO-Clip

Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
**for** $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.
**end for**

Figure 3.4: PPO Pseudo code (5).

# Chapter 4: Implementation

## 4.1 USV Modeling

The USV used for this project is the Otter, developed by Maritime Robotics based out of Trondheim, Norway. The Otter is a catamaran hulled vessel, propelled by two fixed electric thrusters (propellers). An image of the vehicle, along with basic specifications are shown in Figure 4.1.

A dynamic model of this vehicle is provided by Fossen in his Python Vehicle Simulator code on GitHub (12). Fossen models the vehicle with a 25kg payload and determines the respective matrices and vectors for the equations of motion to include the system inertia matrix, Coriolis-centripetal matrix, the damping matrix, and the restoring and ballast forces vectors.

The disturbance model used to specify the magnitude of the water current velocity over time is a first-order Gauss-Markov process as presented in (2):

$$\dot{V}_c + \mu V_c = w \tag{4.1}$$

Where $w$ is Gaussian white noise and $\mu \geq 0$ is a constant. An example of what this looks like over time is shown in Figure 4.2.

## 4.2 Path Following Control

The control input available to the Otter is the rotational velocity of the rear thrusters. The control input vector looks like the following, as the only inputs are the surge and yaw components:

$$\tau = \begin{bmatrix} X & 0 & 0 & 0 & 0 & N \end{bmatrix}^T \tag{4.2}$$

Where $X$ is the surge control force and $N$ is the yaw control moment.

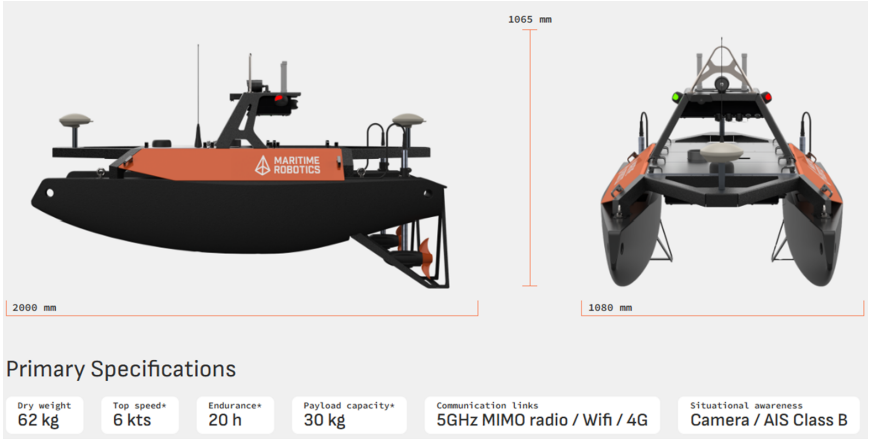


Figure 4.1: Otter USV from maritime robotics (6).

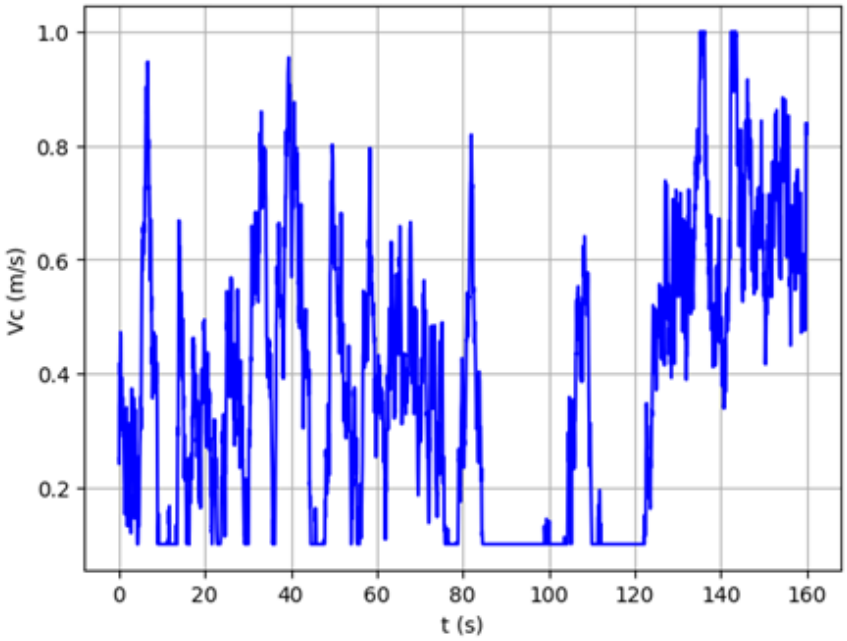


Figure 4.2: Gauss-Markov process modeling of water current velocity magnitude.

The PID controller is used to control the yaw angle (course heading) of the USV. The error term used is the difference between the actual and desired yaw angle. The output of the PID controller used for this design is the change in the yaw control moment. The surge control force is kept at a constant value to simplify the controller design for the reinforcement learning problem.

To get propeller rotational velocity of the two propellers from the surge and yaw control force and moment, the following relationship is used as presented in (2):

$$f = Ku \tag{4.3}$$

Where $f$ is the actuator (propeller) force, $K \in \mathbb{R}^{2x2}$ is the force coefficient matrix specific to the actuator, and $u = |n|^T n$ is the control input where $n = [n_1, n_2]$ is the vector of propeller rotational velocities. As shown in this equation, the propeller force has a quadratic relationship with respect to the propeller rotational velocities.

The control input vector is related to the actuator force through the following relationship:

$$\tau = Tf \tag{4.4}$$

Where $T \in \mathbb{R}^{2x2}$ is the thrust configuration matrix. Expanding the equations above, we get the following:

$$\tau = TKu \tag{4.5}$$

The T and K vectors can be combined into a single vector, B, since they are constants. The complete relationship from the control input vector to control input then becomes:

$$\tau = Bu \tag{4.6}$$

To solve for the propeller rotational velocities, the relationships below are used:

$$u = |n|^T n \tag{4.7}$$

$$n = \text{sgn}(u)\sqrt{|u|} \tag{4.8}$$

Where:

$$u = B^{-1}\tau \tag{4.9}$$

These propeller rotational velocities can be used as inputs into Fossen's Otter USV simulator, (12), to propagate the state of the vehicle forward in time using the equations of motion.

## 4.3 Reinforcement Learning

The reinforcement learning problem is initialized by determining an observation space (state), action space, and reward functions. For this problem, the observation space consists of the course angle error, which is a key component of the course following task, and the velocity and acceleration vectors of the USV, which help the agent understand the dynamics of how these interact with the disturbances and affect the course keeping performance. Also the propeller rotational velocities and the incremental output of the PID controller (yaw control moment), which inform the agent on how these are linked and impact course keeping, and the individual PID terms, which, when considered with the action values determine the controller output, are included. The observation space vector is shown below:

$$S = \begin{bmatrix} e(t) & u & v & r & \dot{u} & \dot{v} & \dot{r} & n_1 & n_2 & \Delta\tau_n \\ e(t) - e(t-1) & e(t) & e(t) - 2e(t-1) + e(t-2) & & & & & & & \end{bmatrix} \tag{4.10}$$

The action space, or what the agent is being trained to choose, consists of the PID coefficients for the course keeping PID controller limited to the ranges shown below:

$$A = \{(K_P, [0, 1000]) \mid (K_I, [0, 4]) \mid (K_D, [0, 100])\} \tag{4.11}$$

The advantage of the limitations on the action space is the prevention of extreme action values being chosen, which could inhibit the course keeping performance.

The same reward function used in (9) is implemented in this design and is presented as:

$$R = \begin{cases} \frac{250\exp(-|e(t)|)}{n} & \text{if } |e(t)| \leq e_{\min} \\ \frac{-|e(t)|}{n\pi} & \text{otherwise} \end{cases} \tag{4.12}$$

Where $n$ is the number of steps in one episode, $e_{min}$ is the minimum course angle error in the current episode, and $e(t)$ is the course angle error at time $t$. The first term in the reward function rewards small course angle errors and increases with a decrease in the error. It is normalized in the sense that the maximum reward value is 1. The second term in the reward function penalizes larger course angle errors (especially if the course angle error fails to decrease over time) and is normalized because the maximum course angle error is $\pi$, so the maximum penalty is –1.

A custom environment following the Gymnasium interface as described in (13) is used, which allows for the implementation of the observation and action spaces as well as the reward function within an RL framework. A step method is defined that contains the logic of the environment and implements Fossen's otter model. This method iterates through each timestep of the environment's dynamics in the RL episode using the agent's chosen actions for that timestep. The reset method initiates a new training episode after the previous episode has been deemed complete and resets the training environment to an initial state. The action and observation spaces are normalized across the range $[-1, 1]$ using the known limits of the observations and actions within this environment which helps the training process.

The PPO RL algorithm for learning is implemented using the Stable Baselines3 (SB3) implementation (14). SB3 deploys many common RL algorithms within PyTorch and provides a unified structure for these algorithms for the generation of policies through agent learning. Weights and Biases (wandb) is used to log all training in real time, and assist in the model performance evaluation (15). The extent of the training process is summarized in Figure 4.3.

At first, the environment is initialized and the USV is set at a random initial course angle on $[-\pi, \pi]$. The water current velocity initial magnitude and direction

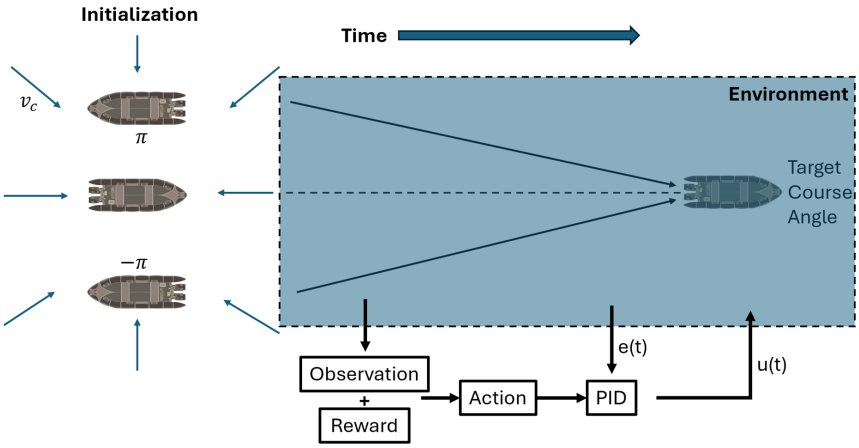


Figure 4.3: RL Training Process

is initialized over $[0, 1]m/s$ and $[-\pi, \pi]$ respectively, and the magnitude varies as described in section 4.1. The target course angle is set to zero. For each time step, the RL agent determines an action based on the reward produced from the observation of the current state. The agent updates the policy parameters using the PPO RL algorithm and the history of actions, rewards, and states. The actions (PID coefficients) are input into the PID controller, and this, combined with the course angle error, determines the next control input to the USV (propeller rotational velocity). Based on this control input, the dynamics are propagated forward in time, and the process is repeated until the episode ends (when a designated number of steps is reached). At that point a new episode is initialized and executed until the entire RL training process is completed based on the desired number of total training episodes. Ideally, an optimal policy is produced for an adaptive course keeping PID controller at the end of the training. The parameters used for training are shown in Table 4.1.

The PPO specific parameters used in the SB3 implementation of the algorithm are used for this training process. The number of rollout steps is the number of training steps completed before the policy is updated. This, combined with the number of environments, determines the batch size, or the size of the experience data used to update the RL policy. The number of environments is the number of parallel

| Parameter | Value |
|---|---|
| Number of environments | 32 |
| Rollout steps | 128 |
| Minibatch size | 32 |
| Batch size | 4096 |
| Episode steps | 3200 |
| Time step | 0.05s |
| $\tau_X$ | 120N |

Table 4.1: RL Training Parameters

environments executing the training process. The minibatch size is the segment of the full batch that is used for each individual policy gradient update. Using minibatches improves learning of the agent by allowing for more frequent policy updates (14). The episode step count is the number of time steps completed before terminating the episode and starting a new one. Results of the training are presented in the next chapter.

# Chapter 5: Results

## 5.1 Training Results

Plots of the average reward over episode, standard deviation, clip fraction, and explained variance for the PPO-PID training are shown in Figure 5.1.

Ideally, the average reward should show a general trend toward increasing over the iteration of steps indicating the agent is able to choose actions based on the current state that maximize the reward, approaching an optimal policy. The standard deviation should decrease and level out approaching zero, which shows that over time the policy is more certain of what good action values are. The clip fraction should increase then stabilize. This shows the fraction of times that the clip range hyperparameter is used to cap a new policy within the clip range of the old policy to promote stable learning. The explained variance should approach 1 as this calculates how accurate the value-function estimation of the reward is.

For this training, the rewards increase quickly and then slowly increase until about 45 million steps, at which point there is a large increase and subsequent decline in the returns. This may be due to a divergence in policy (increase in exploration) as can be shown in the increase of the clip fraction curve at this same point. Before then the clip fraction was plateauing as desired. The standard deviation had also been stabilized near zero prior to this point. The explained variance remains close to one for the duration of training indicating the value-function estimation is accurate.

## 5.2 Fixed PID Controller

All training results are compared to the performance of a fixed PID controller. The PID coefficients used for the fixed controller are shown below:

$$K_P = 414, K_I = 0.001, K_D = 50$$

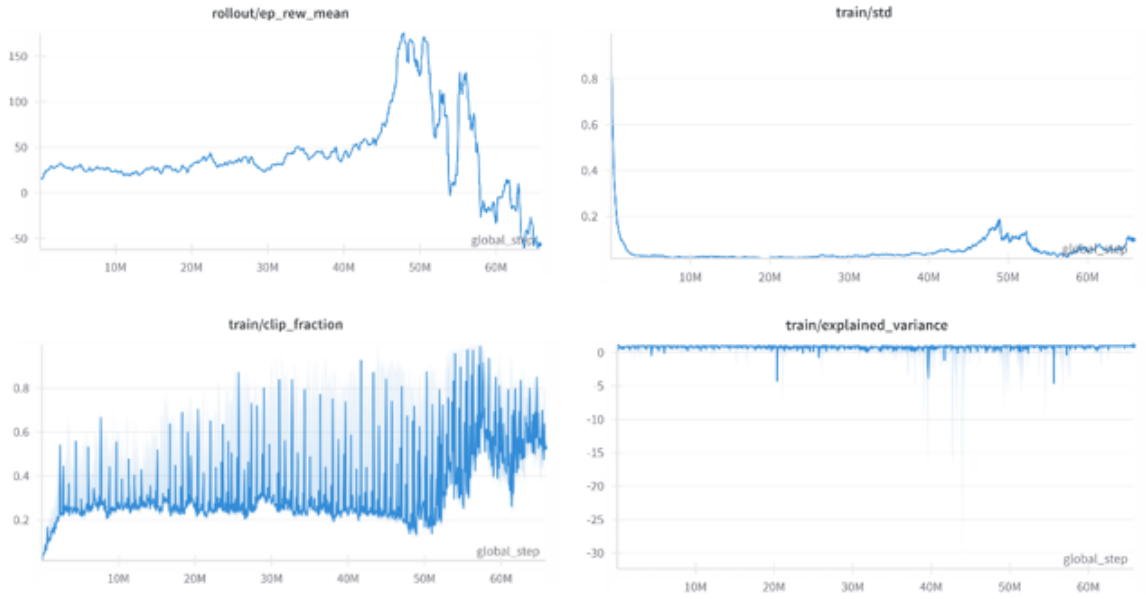


Figure 5.1: PPO-PID training results

The performance of the fixed PID controller when subjected to a course angle offset of 180 degrees with a disturbance water current velocity magnitude of zero is shown in Figure 5.2.

The PID controller shows a fast response to the initial course angle offset and a limited overshoot with good steady state performance as there are only minute fluctuations about the target course angle.

## 5.3 Course Keeping Evaluation

This evaluation compares the development of the course-keeping task for the best performing policy from training vs. the fixed PID controller. The initial course angle is set to $180^o$, and the target course angle is set to $0^o$. A total of 36 iterations of the evaluation are completed using a varying disturbance magnitude over time on the range of $[0, 1]m/s$, with the disturbance angle broken into $10^o$ increments from $[-\pi, \pi]$. The average course angle and standard deviation of the 36 runs are plotted using a solid line and shaded areas respectively in Figure 5.3. A violin plot of average

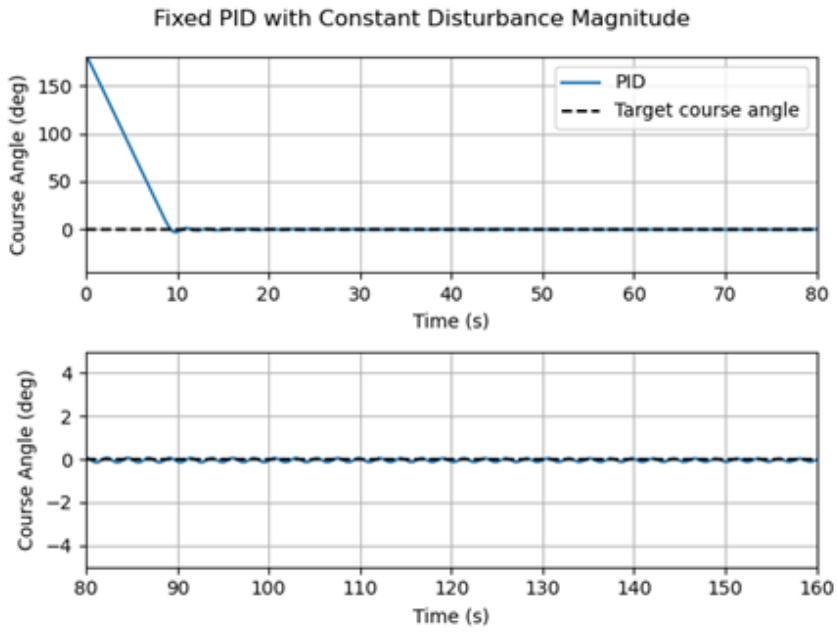


Figure 5.2: Path following fixed PID controller performance.

course angle error from time 80s to the end of the iteration is compared between RL-PID and fixed PID controllers as well. The RL-PID policy with the best performance when compared to a fixed PID controller is the policy at training step 2million, and is the policy used for this evaluation.

The RL-PID controller displays a much slower settling time and a larger overshoot when compared to the fixed PID controller. The steady state performance appears to be better than the fixed PID controller, though, when observing the oscillations on the plot from 80s to 160s. This is confirmed in the violin plot as the RL-PID controller shows a smaller range of course angle deviation about the steady state course angle, although there is some steady state error in the RL-PID controller.

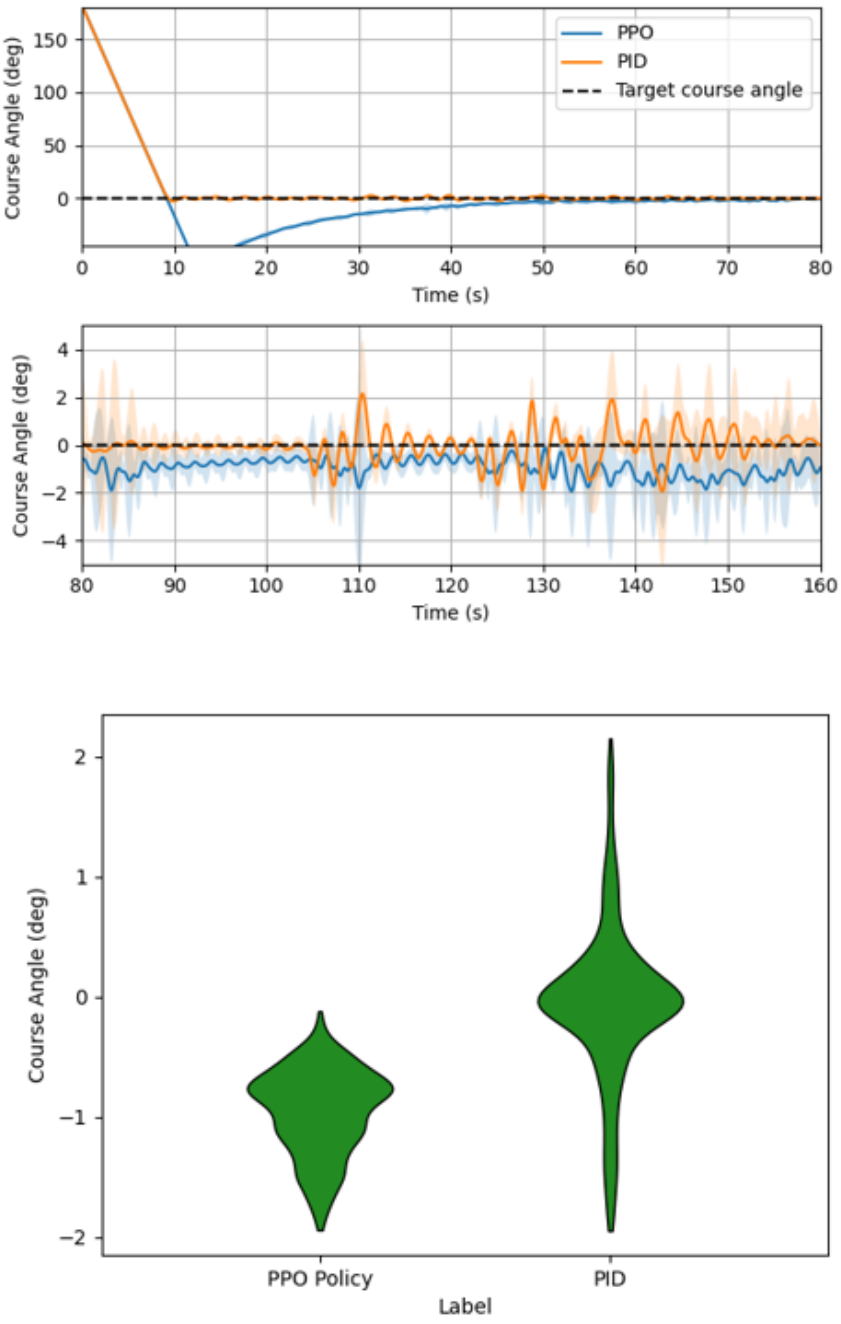


Figure 5.3: Course keeping results.

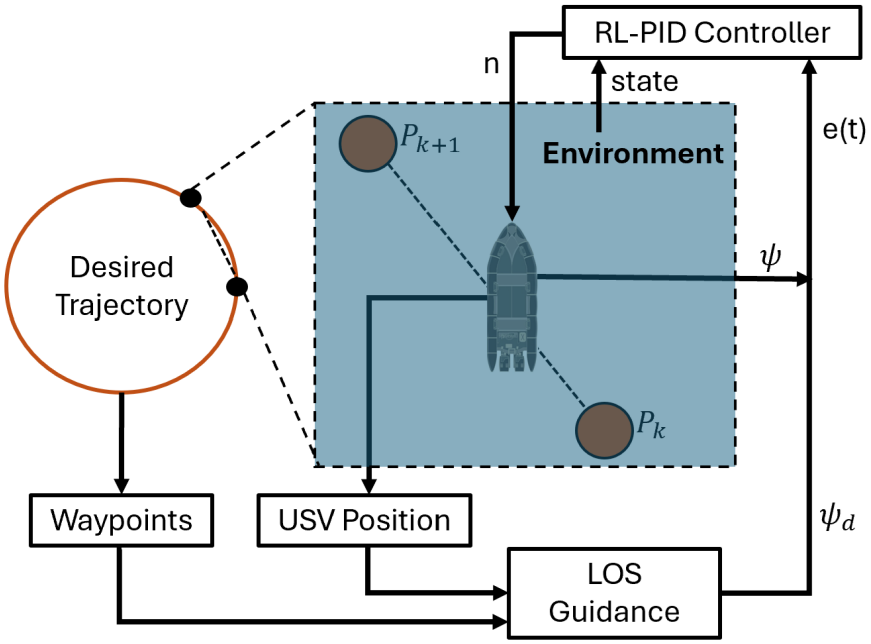


Figure 5.4: Path following evaluation setup.

## 5.4 Path Following Evaluation

### 5.4.1 Setup

The next evaluation completed combines the RL-PID controller with the LOS guidance law to show performance on a course following task. The same policy that was used for the course keeping evaluation is used in this case. An overview of the setup is shown in Figure 5.4.

Waypoints are generated from a desired path to follow, breaking the path into segments. The start and end points of these segments are the waypoints. The position of the USV and waypoints for the current segment it is traveling along are input into the LOS guidance law. A desired heading for the USV is returned and the error between this and the actual heading is fed into the RL-PID controller along with the current state, which outputs commanded propeller rotational velocities for each propeller. This is completed for each timestep until the goal waypoint is reached for that segment, and then the process repeats itself. Each segment is followed until

the final waypoint in the desired trajectory is reached. A disturbance water current velocity with varying magnitude is set to an initial magnitude of $0.25m/s$ and a fixed direction of $90^o$.

### 5.4.2 Circular Path

The first desired trajectory used in the evaluation is a circular trajectory with a radius of 500m, broken into 20 linear segments. The acceptance radius for the waypoints is set to 1.5m, meaning that the USV needs to come within 1.5m of the target waypoint for the waypoints to iterate to the next segment. If the USV cannot meet these acceptance criteria, then the next waypoint will not be given until the USV is 250m from the target waypoint of the segment it is currently traveling on. Results of this evaluation compared to the fixed PID controller are shown in Figure 5.5.

The X-Y positional plots of each course-following controller clearly shows that the RL-PID controller performs better in this scenario. The controller is able to reach each waypoint along the desired circular trajectory within the designated acceptance radius. The fixed PID controller could not do this and only switched to the next target waypoint after reaching the maximum distance from the previous waypoint. The violin plot confirms the RL-PID policy maintains a smaller range of course angle errors over the entire path. The superior performance of the RL-PID controller, despite the course keeping evaluation results, is likely due to the small initial course angle error of the USV at the start point of the path vs a large initial course angle error which was used in the course keeping evaluation.

### 5.4.3 M-Path

A second desired 'M' shaped path is used to evaluate the RL-PID controller performance in the course following task. The same setup is used as in the circular path, except that the waypoints of the desired path were those shown in Table 5.1.

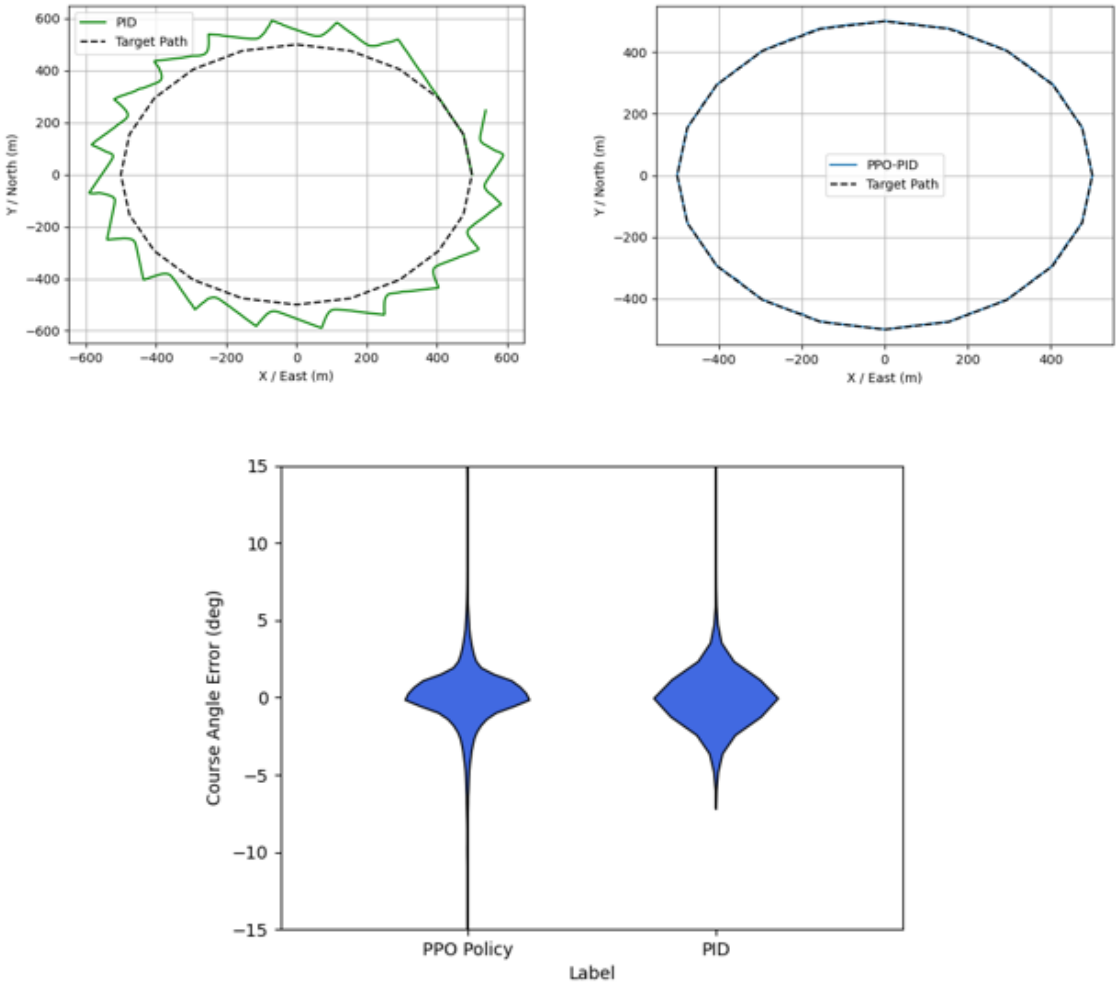


Figure 5.5: RL-PID circular path following results.

| Point | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $x_d$ | 500 | 450 | 300 | 250 | 100 | 50 | -100 | -150 |
| $y_d$ | 0 | 143 | 143 | 0 | 0 | 143 | 143 | 0 |

Table 5.1: M-Path waypoints.

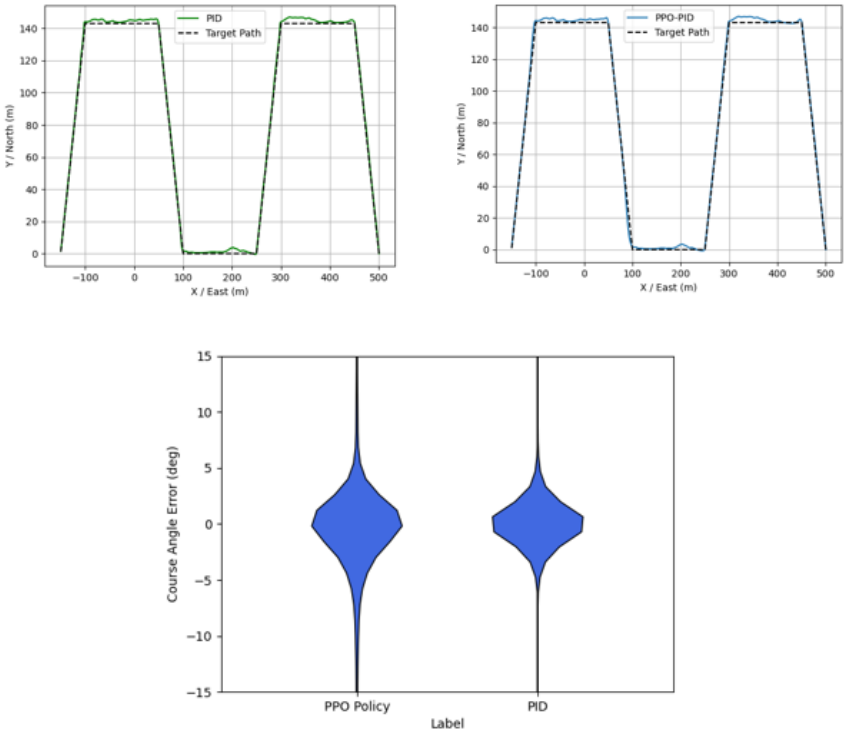


Figure 5.6: RL-PID M-path following results.

Results of this evaluation compared to the fixed PID controller are shown in Figure 5.6.

Reviewing the positional plots of the USV for the M shaped path shows that both controllers perform well, and their differences are negligible. They are both able to follow the set path within the given acceptance radius of the target waypoints, and the violin plot shows the fixed PID controller has the slight advantage in course angle error spread.

# Chapter 6: Conclusions and Future Work

In this report, an adaptive, RL-PID controller was developed for the course following task of a propeller controlled USV. This is done using the proximal policy optimization RL algorithm to learn a policy that can choose PID coefficients for an incremental PID controller based on the state of the environment. This report examines the detailed modeling of the Otter USV and ties together a LOS guidance law with a PID controller to turn path errors into course angle errors. A summary of the general RL problem is provided along with a detailed analysis of the PPO algorithm. The RL-PID controller performance is compared to a fixed PID controller over a course angle and two path following evaluations.

The RL-PID controller did not perform as well in the course keeping evaluation, but exceeded and matched performance of the fixed PID controller in the path following tests. Future work includes additional reward tuning and training environment refinement to improve the current RL-PID policy. The use of better hardware for training would allow for faster and longer training cycles to expedite this iterative process. The implementation of a more complex disturbance model to include wave and wind forces, as well as a more accurate dynamics model utilizing computational fluid dynamics would make the integration of an RL-PID controller onto actual hardware a more efficient and rewarding process. The comparison of performance using newer RL algorithms such as the Soft Actor Critic (SAC) and Twin Delayed DDPG (TD3) methods could provide improved performance. All code for this report is stored on GitHub for reference[1].

[1] https://github.com/jhart4593/RL_USV_Project

# Bibliography


[1] Y. Qin, W. Zhang, J. Shi, and J. Liu, "Improve pid controller through reinforcement learning," in *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*. IEEE, 2018, pp. 1–6.

[2] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[3] Y. Wang, Y. Hou, Z. Lai, L. Cao, W. Hong, and D. Wu, "An adaptive pid controller for path following of autonomous underwater vehicle based on soft actor–critic," *Ocean Engineering*, vol. 307, p. 118171, 2024.

[4] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction, 2nd edn. adaptive computation and machine learning," 2018.

[5] J. Achiam, "Spinning up in deep reinforcement learning," 2018, accessed: 2025-03-01. [Online]. Available: https://spinningup.openai.com/

[6] "Otter," https://www.maritimerobotics.com/otter, 2023, accessed: 2025-02-07.

[7] M. Caccia, M. Bibuli, R. Bono, and G. Bruzzone, "Basic navigation, guidance and control of an unmanned surface vehicle," *Autonomous Robots*, vol. 25, no. 4, pp. 349–365, 2008.

[8] D. Li and L. Du, "Auv trajectory tracking models and control strategies: A review," *Journal of Marine Science and Engineering*, vol. 9, no. 9, p. 1020, 2021.

[9] P. Lai, Y. Liu, W. Zhang, and H. Xu, "Intelligent controller for unmanned surface vehicles by deep reinforcement learning," *Physics of Fluids*, vol. 35, no. 3, 2023.

[10] Y. Wang, J. Cao, J. Sun, X. Zou, and C. Sun, "Path following control for unmanned surface vehicles: A reinforcement learning-based method with experimental validation," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[12] T. Fossen, "Python vehicle simulator," 2025, accessed: 2025-03-15. [Online]. Available: https://github.com/cybergalactic/PythonVehicleSimulator

[13] M. Towers *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," https://github.com/Farama-Foundation/Gymnasium, 2023, accessed: 2025-03-09.

[14] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of machine learning research*, vol. 22, no. 268, pp. 1–8, 2021.

[15] L. Biewald, "Experiment tracking with weights and biases," 2020, accessed: 2025-02-24. [Online]. Available: https://www.wandb.com/