Mixed Precision Reinforcement Learning for Control Simulation of Unmanned Undersea Vehicles

Christopher Hixenbaugh

Alfa Heryudono

Eugene Chabot

1. ABSTRACT

Control systems for Unmanned Undersea Vehicles (UUVs) are typically implemented using Proportional Integral Derivative (PID) control systems. PID control systems for UUVs are resource intensive to tune because they require several engineers, marine operators, and ship crew to spend time offshore to tune the controller. Furthermore, PID controllers rely on complex dynamic system models that contain assumptions to reduce the computational complexity of the models but degrade performance of the controller if an environmental condition is encountered that conflicts with an assumption. In this study, a Deep Reinforcement Learning control system based on the Deep Deterministic Policy Gradient (DDPG) algorithm is studied for a UUV control system. The DDPG algorithm is model free, meaning that a complex dynamic system model is not needed to learn and provide optimal control performance. Secondly, Deep Reinforcement Learning control systems are tuned autonomously, so this greatly reduces the resources needed for controller tuning. The drawback to Reinforcement Learning is that it is computationally and resource intensive. To improve upon this, this study will investigate how mixed floating point precision with loss scaling can be used to reduce the time and computational resources needed to train the DDPG agent. Numerical case studies will be presented.

2. INTRODUCTION

Technologies that rely entirely on autonomous systems have played significant roles in advancing environmental research in the undersea domain. Unmanned Undersea Vehicles (UUVs) such as the Naval Post Graduate School UUV research platform have played a role in advancing the state of the art of autonomous systems for research purposes. Using autonomous systems for research is becoming more popular because autonomous systems can relieve humans from repetitive tasks and reduce the risk of injury. Additionally, UUVs can be manufactured in large quantities at relatively lower costs. Moreover, due to advances in computing and battery technologies, UUVs can undertake more extended missions without human interventions.

One of the essential parts of UUVs is the control system. UUV control system configuration may change based on the vehicle payload or environmental factors such as salinity. The control system is responsible for achieving and maintaining stable flight about at a target path. PID controllers are widely implemented on UUVs, although their use comes with a significant cost to tune the controller. The steep cost does not provide the benefits of a robust or intelligent solution because of two major problems.

The first problem is that PID controllers rely on complex dynamic system models to control the UUV. The dynamic system models have simplifying assumptions that allow the control problem to be solved efficiently. When the assumptions are not valid, a PID controller can provide sub-optimal control, or even complete loss of control can occur. The second problem is that PID

controllers are not intelligent and cannot learn autonomously. PID controllers require multiple engineers and other personnel to spend days collecting and analyzing data to tune the controller. Tuning a PID controller is a manual task that introduces the opportunity for human error.

There is much ongoing research in using Deep Reinforcement Learning methods for autonomous vehicle control systems and it has shown promising results [1,2]. Deep Reinforcement Learning controllers have been shown to outperform PID controllers for UUVs executing path-following missions [3]. Additionally, Deep Reinforcement Learning based controllers have been demonstrated to provide superior attitude control compared to PID controllers for Unmanned Aerial Vehicles (UAVs) [4-5]. Although this example is not specific to UUVs, this concept from the aerial domain can be translated to the undersea domain.

Some of the most popular Deep Reinforcement Learning algorithms being used for autonomous vehicle control system development are the Proximal Policy Optimization (PPO) [6] and Deep Deterministic Policy Gradient (DDPG) [7] algorithms. This study will focus on the DDPG algorithm. The DDPG algorithm is an Actor-Critic type Deep Reinforcement Learning algorithm. Actor-Critic algorithms learn both a policy and value function. The concept of an Actor-Critic algorithm is that the policy function (the actor) determines the actions of the system according to the current state, and the value function (the critic) critiques the actions. In Deep Reinforcement learning, the policy and value functions are approximated by DNNs, specifically Multi-Layer Perceptrons (MLPs) in this study.

There are two major benefits that a Deep Reinforcement Learning controller based on the DDPG algorithm provides compared to a traditional PID controller for UUVs. The first benefit is that the DDPG algorithm is model-free. It does not require any knowledge of the vehicle or environmental dynamics to provide optimal control. Therefore, it avoids the downfalls of simplifying assumptions needed to solve complex vehicle or environmental dynamic system models efficiently. Secondly, Deep Reinforcement Learning based control systems can be tuned (trained) autonomously. This will reduce the resources needed to tune a Deep Reinforcement Learning based control system compared to a PID control system.

The drawback to Reinforcement Learning is that it can be more computationally intensive than is situationally acceptable. Studies have shown that using mixed numerical precision [8] to improve computational efficiency of model training can provide computational improvements and power efficiencies while maintaining solution accuracy. Traditionally, either single-precision (8-digit accuracy) or double-precision (16-digit accuracy) is used, but not both. However, recently there is a trend to utilize mixed numerical precision in deep neural-network training [9-10] and it is shown that comparable single precision accuracy can be achieved by using a combination of single and half precision (4-digit accuracy) but with substantial computational speed up during training.

There is limited ongoing research in using reduced precision to improve the computational efficiency of Reinforcement Learning. The authors in [11] demonstrate how quantization techniques can be used to improve the system performance of Deep Reinforcement Learning. The authors of [12] show a strategy with 6 methods to increase numerical stability for low precision training of the Soft Actor Critic (SAC) algorithm. While the ongoing research focuses on the benchmark Reinforcement Learning problems, this concept is relatively unexplored for

scientific applications such as using a Deep Reinforcement Learning agent for continuous control of a UUV.

This study will demonstrate how training a DDPG agent for continuous control of a UUV with mixed precision and loss scaling does not affect control system performance while making the solution computationally more efficient in two ways. First, we will compare performance of the DDPG agent trained with both fixed and mixed numerical precisions to the performance of a PID controller for a 1 degree of freedom speed control problem. Training step times will be examined for training the DDPG agent with fixed and mixed precisions. Second, this study will investigate the DNN size and batch size thresholds where the benefits of training a DDPG agent with mixed precision outweigh the computational costs.

The rest of the paper is structured as follows. The Problem Formulation section will provide a brief background on the DDPG algorithm, NPSUUV dynamics, PID control, and mixed numerical precision. The Experimental Analysis section will describe the setup and results of the numerical experiments run in this study. Finally, the overall work and future planned work will be described in the Conclusions and Future Work section.

3. PROBLEM FORMULATION

3.1 DDPG ALGORITHM

This study will focus on using the DDPG algorithm for control of the UUV described in this section. The DDPG algorithm is an off-policy, model-free actor-critic Deep Reinforcement Learning algorithm. Model-free means that the algorithm does not rely on an environmental or UUV dynamic system model to achieve optimal performance. The DDPG algorithm is an off-policy algorithm. This means that the state-action function does not depend on the policy used to gather experiences. An off-policy algorithm considers the maximum Q value over all the potential actions available in a given a state [13]. This is different from an on-policy algorithm such as SARSA [14] which relies on the Q value calculated by the experience gathering policy.

A benefit of off-policy algorithms is that a large number of past experiences can be considered when computing the Q value. These experiences given by the tuple, (s_t, a_t, r_t, s_{t+1}) , are stored in an experience buffer, or replay buffer, in a First In First Out (FIFO) fashion. s_t is the state at time t, a_t is the action selected by the agent at time t, r_t is the reward at time t, and s_{t+1} is the next state. As the buffer becomes full, the oldest experiences are removed. Removing old experiences is important because older experiences tend to be less useful and it's preferable for the agent to learn from more recent experiences. Each time the agent trains, experiences are sampled from a uniform distribution to update the parameters of the critic network. The number of experiences used during training is called the batch size. The buffer size and batch size are tunable DDPG agent hyperparameters.

The DDPG algorithm is an actor-critic type algorithm. In Deep Reinforcement Learning the actor and critic are implemented as DNNs. The actor network, $\mu(s|\theta)$, learns a parameterized policy that computes an action according to the current state. The critic network, $Q(s, a|\phi)$, learns a value function given a state action pair and provides reinforcing information to the actor.

θ and ϕ are the weights of the actor and critic networks respectively. The critic computes the temporal difference (TD) error that is used in both the actor and critic networks during the training process. A high level architecture of the actor-critic agent is found in **Figure 1.**

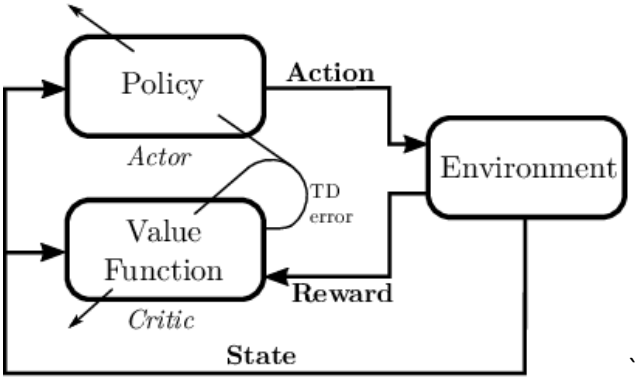

Figure 1: Actor-critic agent architecture [15]

There are two actor and critic networks in the DDPG algorithm. There is a trained network and a target network for both actor and critic. The target networks for the actor and critic are denoted by $\mu'(s|\theta')$ and $Q'(s,a|\phi')$ where θ' are the weights of the target actor network and ϕ' are the weights of the target critic network. The reason for using two networks is that it stabilizes training [16]. During training, the actor and critic networks are being updated frequently and this makes training difficult because there can be large changes to the actor and critic networks between each training step. To mitigate this, target networks are implemented that are updated at a slower rate. There are two strategies to update the weights of the target network. One strategy is to copy the weights from the actor and critic networks to their respective target networks periodically, or to use the Polyak averaging method to update the target actor and critic weights at a slower constant rate. The Polyak method will be used in this study because it eliminates the potential for large changes in the target actor and critic networks that may occur if the time between updates is too long. The Polyak update method for the target actor and critic networks are:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

τ is a tunable hyperparameter called the Target Smoothing Factor that controls how fast the target actor and critic network weights change with respect to the actor and critic networks. Typical values for τ are on the order of 10^{-3}. The target actor and critic networks are updated during each time step during training just like the actor and critic networks.

The actor network is updated by sampling the policy gradient:

$$\nabla_\theta \mathrm{J} \approx \left(\frac{1}{N}\sum_i \nabla_a Q(s_i, \mu(s_i|\theta) \mid \phi)\nabla_\theta \mu(s_i|\theta)\right)$$

The critic network is updated by minimizing the loss:

$$L = \frac{1}{N}\sum_{i}\left(y_i - Q(s_i, a_i|\phi)\right)^2$$

To promote exploration in the DDPG algorithm, an exploration policy is built by adding noise to the action selected by the actor network. The noise added to the policy is defined by the Ornstein-Uhlenbeck process [17]. The exploration policy is defined as

$$\mu'(s) = \mu(s|\theta) + OUnoise$$

The Ornstein-Uhlenbeck (OU) noise is implemented as [18]:

$$x_t = x_{t-1} + \vartheta(\mu - x_{t-1})dt + randn\big(size(\mu)\big)\sigma\sqrt{dt}$$

Where x_t is the noise to be added to the selected action at time t, ϑ is the Mean Attraction Constant which specifies how quickly the noise model output is attracted to the noise model mean, μ. σ is the standard deviation and is defined as a percentage of the action space range.

$$\sigma = 0.X(action\_space_{max} - action\_space_{min})$$

During each time step, the standard deviation is decayed by the decay rate ε. The decayed standard deviation is

$$\sigma_{decayed} = \sigma(1 - \varepsilon)$$

The standard deviation to be used in the next step is the calculated as

$$\sigma = \max\big(\sigma_{decayed}, \sigma_{min}\big)$$

where σ_{min} is the minimum standard deviation for the noise model. σ_{min} is a tunable hyperparameter.

3.2 NPSUUV DYNAMICS

The UUV numerical model used in this study is the Naval Post Graduate School Aries UUV model as described in [19]. The authors detail a six degree of freedom dynamic system model for this UUV. The equations of motion for this system are developed in the body fixed reference frame as described in [20]. The velocity components of the UUV in the body fixed reference frame are described as:

$$\dot{x} = [u(t), v(t), w(t), p(t), q(t), r(t)]$$

Where *u(t)* is the surge velocity, *v(t)* is the sway velocity, *w(t)* is the heave velocity, *p(t)* is the roll velocity, *q(t)* is the pitch velocity, and *r(t)* is the yaw velocity.

The body-reference frame for the NPSUUV is illustrated in **Figure 2.**

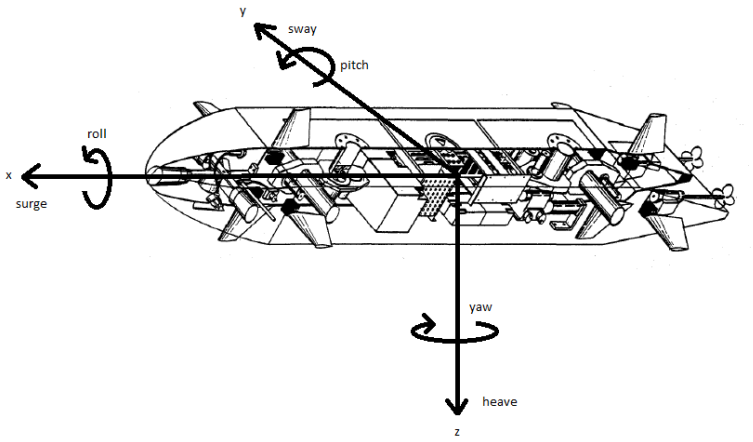

Figure 2: Body-reference frame for Aries UUV [19]

The six components describing UUV position are:

$$x = [x(t), y(t), z(t), \phi(t), \theta(t), \psi(t)]$$

Where *x(t)* is the position in the surge direction, *y(t)* is the position in the sway direction, *z(t)* is the position in the heave direction, $\phi(t)$ is the roll angle, $\theta(t)$ is the pitch angle, and $\psi(t)$ is the yaw angle.

To control the UUV, the model control inputs are:

$$ui = [\delta_r(t), \delta_s(t), \delta_b(t), \delta_{bp}(t), \delta_{bs}(t), n]$$

Where $\delta_r(t)$ is the rudder angle, $\delta_s(t)$ is the port and starboard stern plane angle, $\delta_b(t)$ is the top and bottom bow plane angle, $\delta_{bp}(t)$ is the port bow plane angle, $\delta_{bs}(t)$ is the starboard bow plane, and *n* is the propeller shaft speed.

The equation of motion for the UUV is described in terms of twelve non-linear systems of equations as described in [21]:

$$M(t)\frac{dx}{dt} = \boldsymbol{f}\big(x(t), z(t), c(t)\big) + \boldsymbol{g}\big(x(t), z(t)\big) * \boldsymbol{u}(t)$$

$$\frac{dz}{dt} = \boldsymbol{h}(z(t), x(t), u_c)$$

The mass matrix *M(t)* includes both the mechanical and hydrodynamic added mass. The functions **f** and **g** are mappings of the UUV motions into forces such as Coriolis, gravitational, centrifugal, hydrostatic, hydrodynamic, and moments acting on the UUV with coefficients *c*, and

the motion dependent influence of the control surfaces, propeller, and ballasting. The function **h** includes the kinematical relationships found in performing the coordinate system transformation between the body reference frame, inertial reference frame, and ocean current u_c.

The equations of motion for each degree of freedom as implemented in the Matlab MSS toolbox *npsauv.m* function can be found in the appendix section.

3.3 PID CONTROL

Proportional Integral Derivative (PID) controllers are simple and versatile controllers that are widely used in practice. Although PID controllers are easy to implement and tune, they are very sophisticated in that these controller types capture the history of the system through integration and can anticipate the future behavior of the system through differentiation [22].

The architecture of a feedback control loop with a PID controller is illustrated in **Figure 3.**

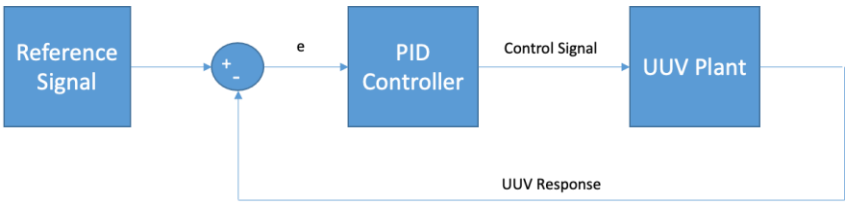

Figure 3: Feedback control loop with PID controller

In this model, the reference signal is the target response that we desire the UUV to achieve. The PID controller accepts the error (e) between the reference signal and the UUV response and computes a control signal for the UUV that will minimize the difference between the reference signal and UUV response.

The control signal is computed by the PID controller by considering the error, the integral of the error, and the derivative of the error as follows:

$$Control\ Signal\ = C_p e(t) + C_i \int e(t)dt + C_d \frac{de(t)}{dx}$$

The coefficient C_p scales the control signal proportionally to the error and influences how quickly the system reacts to minimizing the error. C_i scales the integral of the error and helps minimize the steady state error. C_d scales the derivative of the error and allows the controller to predict the future error. Coefficients C_p, C_i, and C_d are determined empirically through experimentation.

The empirical nature of determining (tuning) PID controller coefficients is both advantageous and detrimental. While the controller easy to understand, it is very resource intensive to tune. Tuning a UUV PID controller requires several engineers to spend time offshore with a physical UUV to manually derive the coefficients through experimentation. After the resource intensive tuning effort, a PID controller is model based, meaning that a dynamic system model of the UUV is required to provide control. This type of dynamic system model is very complex and requires

simplifying assumptions to be solvable in a computationally acceptable manner. If the UUV encounters an environmental condition that conflicts with a simplifying assumption, it can cause the controller to provide sub-optimal control, or even complete loss of control.

To improve control system technology for UUVs, there is ongoing research on using Deep Reinforcement Learning methods for UUV control. Key benefits of using Deep Reinforcement Learning methods for UUV control have been described in the introduction section of this paper.

3.4 FLOATING POINT PRECISION – MIXED PRECISION TRAINING WITH LOSS SCALING

The goal of mixed precision strategies is to decrease computation time and reduce the amount of memory required by roughly half. By applying this to Deep Reinforcement Learning methods we will make strides toward fielding these computationally intensive models on resourced constrained computing systems such as those implemented onboard modern UUVs.

Floating point numbers are represented by 3 components according to IEEE standard 754. A sign bit which signifies if a number is positive or negative. A number of bits that represent the integer part of the number called the exponent, and bits that represent the fractional part of the number called the mantissa. This can be visualized as illustrated in **Table 1.**

0	01101….	00110…
Sign Bit	Exponent	Mantissa

Table 1: Floating point number representation according to IEEE standard 754

The number of bits used to represent half precision (FP16), single precision (FP32), and double precision (FP64) are illustrated in **Table 2**

Precision	**Sign**	**Exponent**	**Mantissa**	**Total Bits**
Half	1	5	10	16
Single	1	8	23	32
Double	1	11	52	64

Table 2 – Bit representation of of Half, Single, and Doule floating point precisions

The smallest representable number in the previously mentioned floating point precisions are specified by the machine epsilon, ε. Machine epsilon values for each precision are illustrated in **Table 3.**

Precision	**ε value**
Half	$9.77\ x\ 10^{-4}$
Single	$1.19\ x\ 10^{-7}$
Double	$2.22\ x\ 10^{-16}$

Table 3: Machine epsilon values

Reinforcement Learning models are very complex models that require large amounts of time and computational resources to train. Scientific computations are traditionally carried out using FP64, however in practice many Machine Learning applications can be solved using FP32. Using less than FP32 (FP16) is problematic in Machine Learning applications because of numerical underflow and overflow. Underflow occurs when a number is too small to be represented by a given floating point precision and the number is cast to 0. Conversely, overflow occurs when a number is too large to be represented by a floating point precision and the number is cast to infinity.

Efforts to reduce the time and computing resources needed to train Machine Learning models is an active area of research. Parallel computing methods such as those described in [23] can be used to reduce the amount of time needed to train a model. Mixed precision methods such as [9,10] address both the time and computing resources needed to train Machine Learning models. Memory requirements are reduced by using a combination of FP32 and FP16 to represent numbers and time requirements are lessened by leveraging GPUs that are designed for lower precision mathematical operations. Using a combination of FP32 and FP16 is referred to as mixed float 16 (MF16).

This study employs model training with MF16 with loss scaling according to [10]. In this approach, the authors store the network weights, activations, and gradients in FP16 but maintain a copy of and update these values in FP32 during each optimizer step. During training, the network weights and activations are converted from FP32 to FP16 and the forward and backward passes are executed using FP16. The loss values are scaled so that the gradients are representable in FP16 and underflow is avoided. The scaled losses are then used to compute the gradients. The gradients are scaled by the same amount as the loss because of the chain-rule. The scaled gradients are unscaled immediately after the backward pass but prior to gradient operations to avoid changes to the model hyperparamters. The network weights are then updated in FP32 . For this study, mixed precision with loss scaling is implemented as in Tensorflow [24] in our numerical experiments.

To briefly illustrate how the gradients are scaled by the same amount as the loss via the chain rule, consider updating weight w_1 in the simple supervised learning example for the neural network in **Figure 4.**

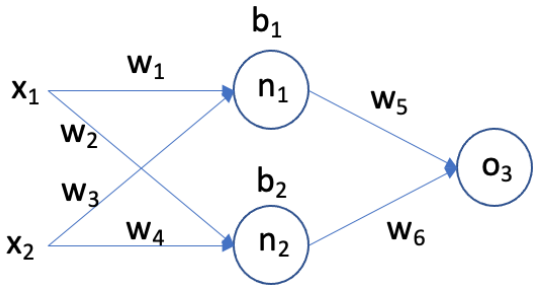

Figure 4

First define the ground truth as y_{true} and the predicted output as y_{pred}. For simplicity, let $y_{true} = 1$. The loss function will be defined as the mean squared error between y_{true} and y_{pred}.

$$L = \frac{1}{n}\sum_{1}^{n}(1 - y_{pred})^2$$

Letting n=1 for this simple example gives

$$L = (1 - y_{pred})^2$$

We can now scale L by a constant C giving

$$L = C(1 - y_{pred})^2$$

To determine how L will change if we changed the weight w_1 we can take the partial derivative of L with respect to w_1.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}}\frac{\partial y_{pred}}{\partial w_1}$$

Using the chain rule, we can re-write $\frac{\partial y_{pred}}{\partial w_1}$ as

$$\frac{\partial y_{pred}}{\partial w_1} = \frac{\partial y_{pred}}{\partial h_1}\frac{\partial h_1}{\partial w_1}$$

Which gives

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}}\frac{\partial y_{pred}}{\partial h_1}\frac{\partial h_1}{\partial w_1}$$

We can write y_{pred} and h_1 as:

$$y_{pred} = f(w_5 x_1 + w_6 h_2 + b_3)$$

$$h_1 = f(w_1 x_1 + w_2 h_2 + b_1)$$

The function f is the activation function. Taking the partial derivatives of L, y_{pred}, and h_1 gives:

$$\frac{\partial L}{\partial w_1} = 2C(1 - y_{pred})[(w_5 f'(w_5 x_1 + w_6 h_2 + b_3)][\, x_1 f'(w_1 x_1 + w_2 h_2 + b_1)]$$

This process is carried out for all the weights in the network to compute the gradient. So we can see that the gradient is scaled by the same factor as the loss, C. The gradients are then unscaled and the weights are updated using an appropriate optimization method such as Stochastic Gradient Descent (SGD), RMSprop, or Adam [25-26].

The process explained above can be applied to the DDPG algorithm for updates to the actor and critic networks. The difference with reinforcement learning compared to supervised learning is that there is no constant ground truth to compute the loss from. Training the deep neural

networks in deep reinforcement learning is like trying to hit a moving target compared to supervised learning. For the DDPG algorithm, the loss value of the critic network is approximated as:

$$L_{critic} = \frac{1}{N}\sum_{i}\left(y_i - Q(s_i, a_i|\phi)\right)^2$$

Where y_i is the target value and is computed as:

$$y_i = r_i + \gamma * Q(s_{i+1}, a_{i+1}|\phi')$$

And r_i is the reward batch used for the training step, γ is the discount factor, and $Q(s_{i+1}, a_{i+1}|\phi')$ is the value of the batch of next state action pairs used for the training step.

The loss value of the actor network is approximated as:

$$L_{actor} = -\frac{1}{N}\sum_{i} Q(\, s_i, \mu(s_i|\theta) \mid \phi)$$

The weights of the actor and critic networks are updated according to the method just described.

If one would like to incorporate improved numerical stability, one can refer to the methods used in [12]

4. EXPERIMENTAL ANALYSIS

This section will describe the configuration of the control system architecture, numerical simulation environments, modifications to the DDPG algorithm for training in MF16 with loss scaling, and DDPG agent configuration used to execute numerical experiments. We will also present the results of our experiments in this section.

4.1 EXPERIMENTAL SETUP

The PID control system was implemented in Simulink 2020a and the Reinforcement Learning control system was implemented in using Tensorflow 2.5.0. The PID control system architecture was implemented as a feedback control loop as illustrated in **Figure 3.**

For the PID control system architecture, the reference signal block defines the set point surge speed that the UUV is to achieve. The PID controller block provides the control signal to the UUV Plant which is the NPSUUV model as described in the previous section. The PID controller was implemented using the Simulink PID control block and the UUV Plant is implemented using the NPSUUV model (npsauv.m) from the Marine Systems Simulator Toolbox [27].

The Reinforcement Learning control system architecture was implemented as illustrated in **Figure 5.**

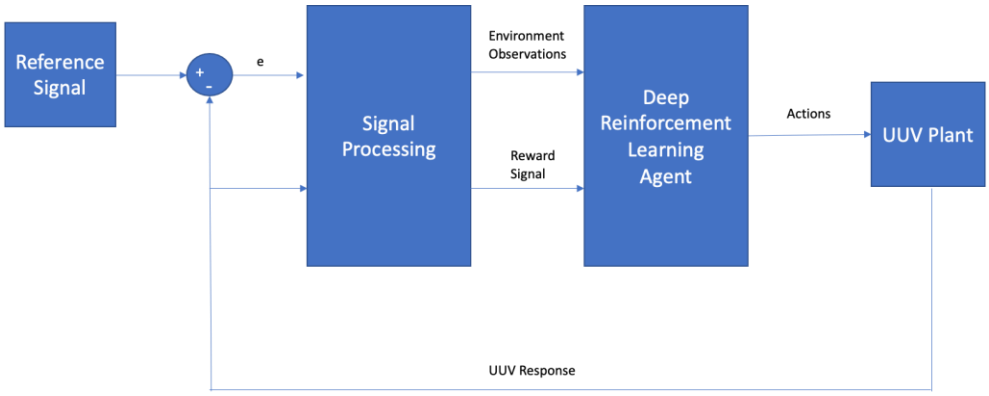

Figure 5: Feedback control loop for DDPG Reinforcement Learning control system for the NPSUUV.

For the Reinforcement Learning control system, the reference signal and UUV plant blocks provide the same functionality and are implemented the same as the PID control system. The Signal Processing block computes the environment observations (states) and reward signal. The Deep Reinforcement Learning Agent (DDPG agent) selects an action given the states and reward. The observations generated by the signal processing block are:

$$s_t = \{\mathrm{e}_t\,, \int \mathrm{e}_t\, dt\,, \frac{d\mathrm{e}_t}{dt}\,, u\,\}$$

Where e_t is the error between the setpoint surge speed and the surge speed at time t,

$$\mathrm{e}_t \;=\; \mathrm{u}_{ref} \;-\; \mathrm{u}_t\,.$$

The reward signal generated by the signal processing block is:

$$r_t = -(c_1 \mathrm{e}_t{}^2 + \mathrm{c}_2(N_t - N_{t-1})^2) + A_t$$

Where N_t is the control signal and $A_t = 1$ when $e_t \leq 0.1\ m/s$, $A_t = 0$ otherwise.

The constants c_1 and c_2 are used to scale components of the reward signal are derived empirically.

The Deep Neural Network architectures used to approximate the policy and value functions for the DDPG agent are illustrated in **Figure 6**.

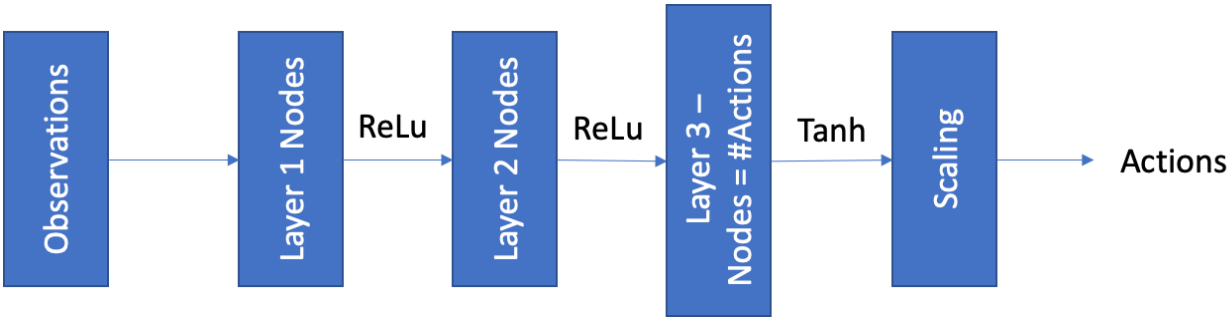

Figure 6(a) – Actor network architecture

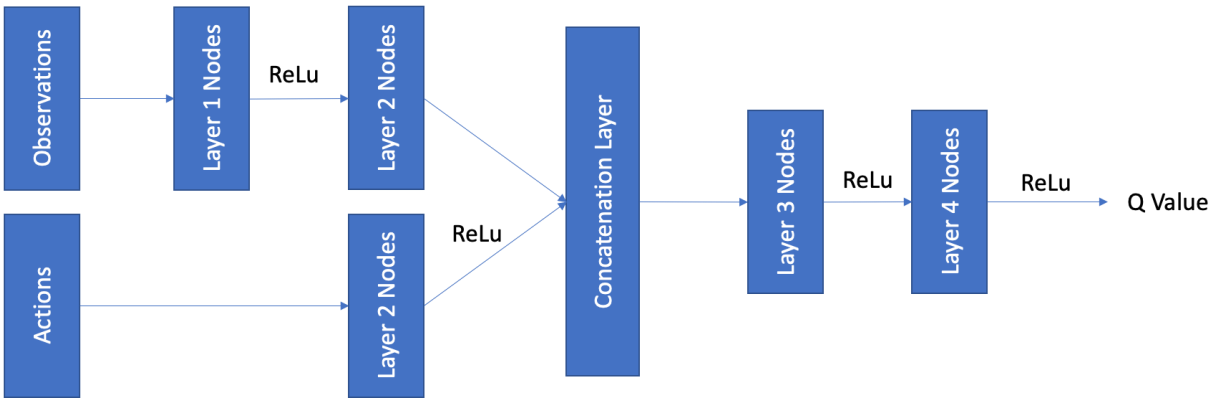

Figure 6(b) – Critic network architecture

For this study, 64 nodes were used in each layer of both the actor and critic network except for the next to last layer of the actor network which has the same number of nodes as dimensions of the action space. In this study that layer has 1 node because the only action is the UUV propeller speed.

The DDPG agent hyperparameters are illustrated in **Table 4**

Parameter Name	**Value**
Critic Learning Rate	0.001
Actor Learning Rate	0.0001
Target Smoothing Factor	0.001
Batch Size	64
Buffer Size	100000
Discount Factor	0.99
OU Noise Variance	$0.06\ (Action\ Space_{max} - Action\ Space_{min})$

Noise Decay Rate	0.000005
Noise Mean Attraction Constant	0.15
Noise Mean	0

Table 4 – Hyperpameters for DDPG agent

The control system architecture and hyperparameters are the same for the DDPG agent trained using FP32 and MF16.

The numerical experiments were run using an Nvidia GPU with compute capability of 7.0 or higher. The GPU is used to update the weights of the DDPG agent's actor and critic networks. All other computations are executed on the CPU. The DDPG algorithm with MF16 and loss scaling is illustrated in **Algorithm 1**. This algorithm is a modified version of the algorithm found in [7]. The same algorithm was used to train the DDPG agent with fixed FP32 precision but with MF16 and loss scaling steps removed.

Algorithm 1 DDPG Algorithm with Mixed Precision and loss scaling

Set data type policy to mixed_float16
Randomly initialize actor network $\mu(s|\theta)$ and critic network $Q(s,a|\phi)$ with weights θ and ϕ
Initialize target actor and critic networks $\mu'(s|\theta')$ and $Q'(s,a|\phi')$ with weights $\theta' \leftarrow \theta$ and $\phi' \leftarrow \phi$
Initialize experience buffer R
For episode 1:E, **do**
Initialize OU Noise process N for action exploration
Receive initial observation state s_1
For t = 0: episode length T, **do**
Select action $a_t = \mu(s_t|\theta) + N_t$ according to the current policy and exploration noise (FP32)
Execute action a_t and observe reward r_t and observe new state s_{t+1} (FP32)
Store transition tuple $(s_t,\ a_t,\ r_t,\ s_{t+1})$ in R (FP32)
Sample a random minibatch of N transitions $(s_i,\ a_i,\ r_i,\ s_{i+1})$ from R (FP32)
Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta')|\phi')$ ** (MF16)
Calculate critic loss $L_{critic} = \frac{1}{N}\sum_i \left(y_i - Q(s_i,a_i|\phi)\right)^2$ ** (MF16)
Scale loss by dynamically determined loss factor C_{critic} ** (MF16)
Compute scaled critic gradients $\frac{dL_{critic}}{dw_{\phi,i}}$ **. (MF16)
Unscale critic gradients ** (FP32)
Update critic network weights ** (FP32)
Approximate the actor loss $L_{actor} = -\frac{1}{N}\sum_i Q(\ s_i, \mu(s_i|\theta)\ |\ \phi)$ ** (MF16)
Scale the actor loss by dynamically determined loss factor C_{actor} ** (MF16)
Compute scaled actor gradients $\frac{dL_{actor}}{dw_{\theta,i}}$ ** (MF16)
Unscale the actor gradients ** (FP32)
Update the actor network weights ** (FP32)
Update target actor and critic network weights (FP32)

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

end for
end for
** Denotes computations executed on the GPU. All other computations are executed on the CPU. //

4.2 EXPERIMENTAL RESULTS

In this numerical experiment, the UUV is starting from rest and is tasked to achieve a setpoint surge speed of 1.4 m/s.

The DDPG agent was trained until and average reward value of 250 was achieved with a scoring window of 5 episodes. The problem is considered solved when that average reward value is achieved. Each training episode was a 600 second simulation with a timestep size of 1 second. Average reward curves for the best trained DDPG agents trained with FP32 and MF16 are illustrated in **Figure 7**.

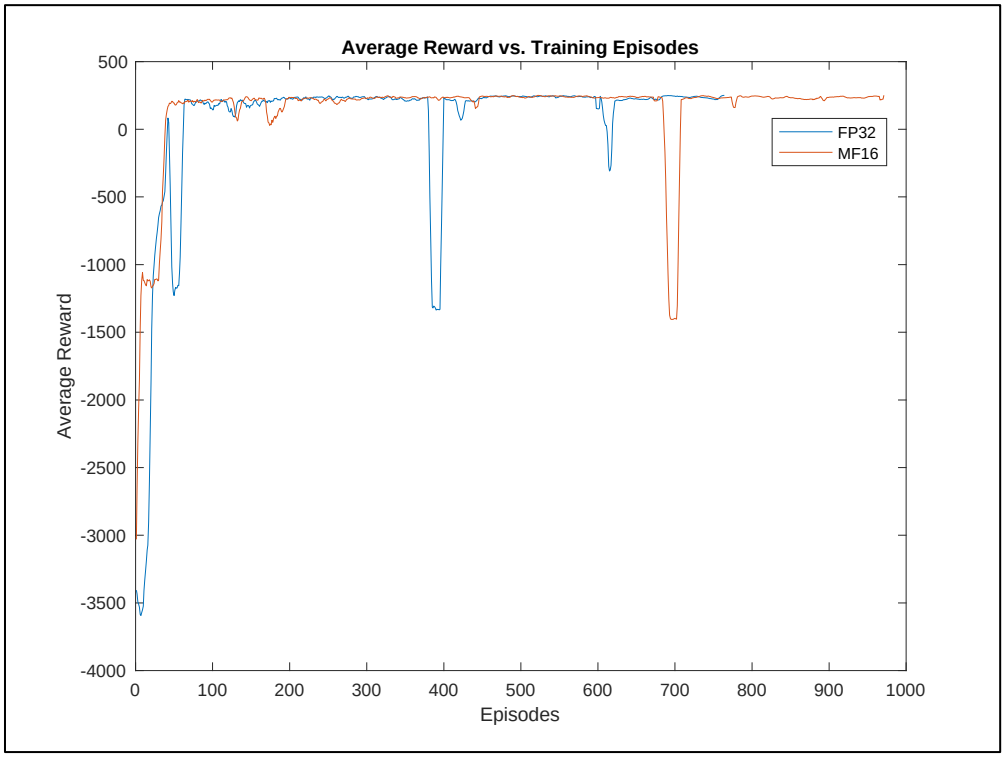

Figure 7: Average reward training data for the best performing Reinforcement Learning control system trained with FP32 and MF16

The PID control system was tuned using the Ziegler-Nichols method [28] and then manually adjusted to improve performance. The gains computed are $C_p = 70{,}000$, $C_i = 50$, $C_d = 0.0005$.

The performance of the best trained Reinforcement Learning control system trained using FP32 and MF16 compared to the performance of the PID control system are illustrated in **Figure 8.**

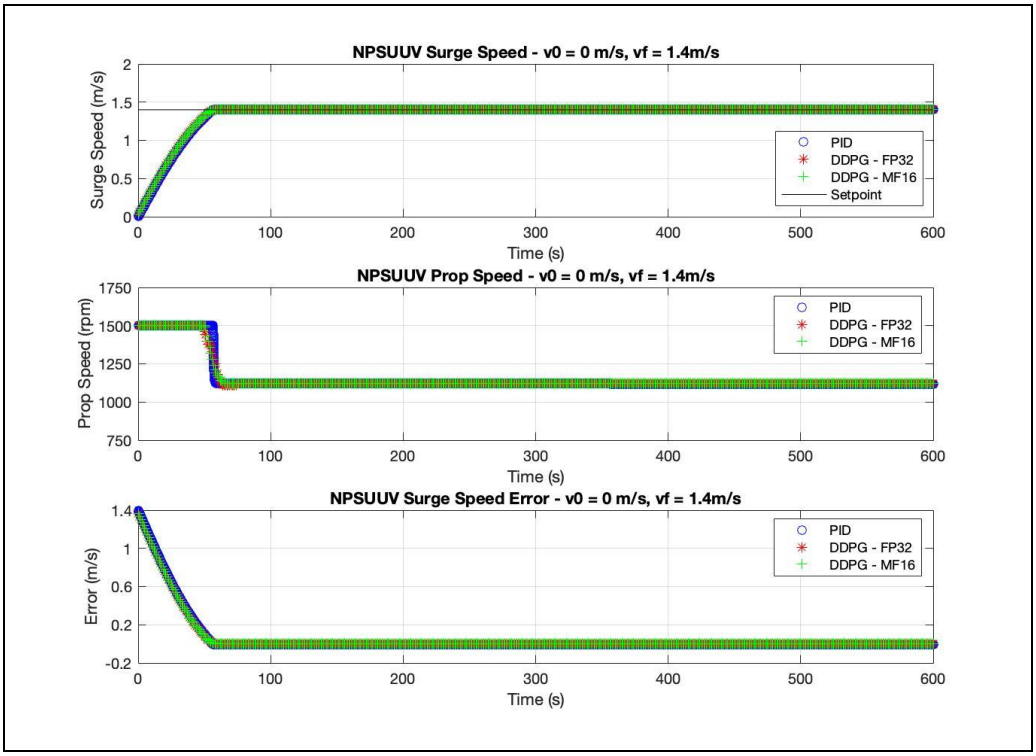

Figure 8 – Performance of best trained Reinforcement Learning control system trained using FP32, MF16, and the PID controller.

The performance of the control systems can be quantified by computing the step response characteristics as defined at [29] as well as the mean squared error (MSE) of the surge speed compared to the set point surge speed. The mean squared error is defined as:

$$MSE = \frac{1}{N}\sum_{t=0}^{N}\left(u_{ref,t} - u_t\right)^2$$

The step response characteristics and mean squared error for the Reinforcement Learning based control system trained with both FP32 and MF16 and PID control system can be found in **Table 5.**

	Steady State Error (m/s)	MSE $(m/s)^2$	Rise Time (s)	Settling Time (s)	Settling Min (m/s)	Settling Max (m/s)	Overshoot (%)	Undershoot (%)	Peak (m/s)	Peak Time (s)
FP32	0.00	29.44	42.42	54.26	1.27	1.40	0.00	0.00	1.40	58.00
MP16	0.00	29.44	42.42	54.26	1.27	1.40	0.00	0.00	1.40	59.00
PID	0.00	31.56	43.19	54.77	1.26	1.41	0.71	0.00	1.41	56.87

Table 5 – Step response characteristics for Reinforcement Learning controllers trained using FP32, mixed precision, and the PID controller.

Visual inspection of **Figure 8** suggests that all three control systems perform about the same. The step response characteristics and MSE quantitatively show that the Reinforcement Learning

control system trained with either FP32 or FP16 is more efficient than the PID controller because there is no overshoot. Off all three controllers, the Reinforcement Learning control system trained using FP32 is the most efficient. The Reinforcement Learning control system trained with MF16 performs practically the same as the system trained with FP32 except being 1 second slower to reach the peak value. Depending on the performance needs of the UUV, it may be situationally acceptable to accept a slightly slower peak time to realize the computational efficiencies of MF16

To investigate the effects of training the Reinforcement Learning control system with FP32 compared to MF16 we will quantify the results using average time per training step for all Reinforcement Learning agents trained in this experiment. The results are displayed in **Table 6.**

Floating Point Precision	Time / training step (s)	Training Steps
FP32	0.007121672	8,135,400
MF16	0.010380572	8,083,200

Table 6: Average time per training step for Reinforcement Learning control system trained with FP32 and MF16

For the reduced degree of freedom problem being considered here, the deep neural network size is not large enough for the benefits of training a model with MF16 to outweigh the computational cost. This follows the insight published at [24] which suggest that the deep neural network size needs to be larger than those typically used for toy problems such as this. Considering this, we set out to determine how many nodes need to be in each layer of the actor and critic deep neural network architectures implemented in this study to realize benefits of training with MF16. While exploring this it was observed that the benefits of training a DDPG agent with MF16 is a function of both the number of nodes per layer and the batch size. The results of this investigation are illustrated in **Figure 8.**

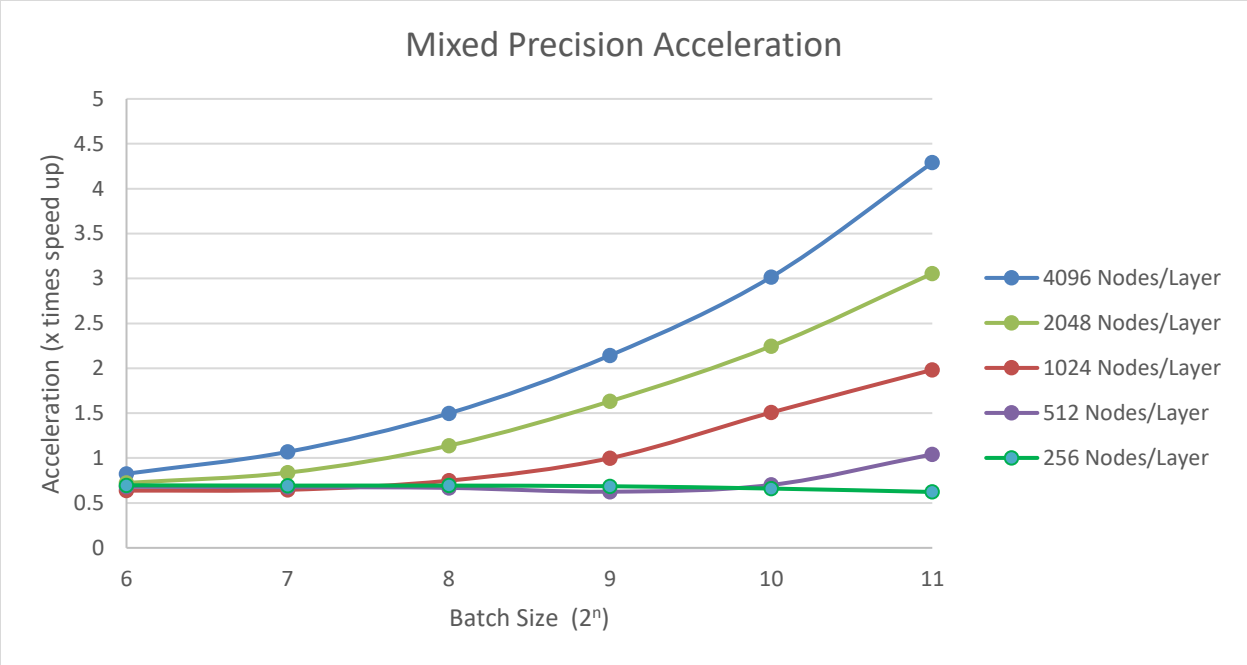

Figure 8 – Acceleration of training DDPG agent with mixed precision compared to training with FP32 for combinations of nodes/layer and training batch size

5. CONCLUSION AND FUTURE WORK

In this study we demonstrated that a Reinforcement Learning control system based on the DDPG algorithm can be trained using FP32 and MF16 and perform practically equivalent for continuous control of a UUV for the reduced degree of freedom problem considered here. Additionally, we showed that the Reinforcement Learning control system trained with MF16 and FP32 outperforms a PID controller tuned using the Ziegler-Nichols method. Although the cost of training the Reinforcement Learning control system with MF16 outweighs the benefits for the toy control problem considered in this study, results were presented that illustrate the neural network layer size and batch sizes where the benefits of MF16 outweigh the costs. As we progress in our research through increasing the degrees of freedom of our control system model and begin to incorporate simulated UUV sensor data, we expect to have more complex actor and critic deep neural networks that will realize the benefits training with MF16.

6. REFERENCES

1. Hsu, Y., Wu, H., You, K., & Song, S. (2019). A selected review on reinforcement learning based control for autonomous underwater vehicles. *arXiv preprint arXiv:1911.11991*.
2. Yu, R., Shi, Z., Huang, C., Li, T., & Ma, Q. (2017, July). Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle. In *2017 36th Chinese Control Conference (CCC)* (pp. 4958-4965). IEEE.
3. Wu, H., Song, S., You, K., & Wu, C. (2018). Depth control of model-free auvs via reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *49*(12), 2499-2510.
4. Bøhn, E., Coates, E. M., Moe, S., & Johansen, T. A. (2019, June). Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)* (pp. 523-533). IEEE.
5. Koch, W. (2019). Flight controller synthesis via deep reinforcement learning. *arXiv preprint arXiv:1909.06493*.
6. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
7. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
8. Higham, N. J. (2017, July). The rise of multiprecision arithmetic. In *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)* (pp. 1-1). IEEE.
9. Nandakumar, S. R., Le Gallo, M., Boybat, I., Rajendran, B., Sebastian, A., & Eleftheriou, E. (2018, May). Mixed-precision architecture based on computational memory for training deep neural networks. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1-5). IEEE.
10. Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., ... & Wu, H. (2017). Mixed precision training. arXiv preprint arXiv:1710.03740.Krishnan, S., Chitlangia, S., Lam, M., Lam, Z., Faust, A., & Reddi, V. J. (2019). Quantized Reinforcement Learning (QUARL). arXiv preprint arXiv:1910.01055.
11. Krishnan, S., Chitlangia, S., Lam, M., Lam, Z., Faust, A., & Reddi, V. J. (2019). Quantized Reinforcement Learning (QUARL). *arXiv preprint arXiv:1910.01055*.
12. Bjorck, J., Chen, X., De Sa, C., Gomes, C. P., & Weinberger, K. Q. (2021). Low-precision reinforcement learning. *arXiv preprint arXiv:2102.13565*.
13. Graesser, L., & Keng, W. L. (2019). Foundations of deep reinforcement learning: theory and practice in Python. Addison-Wesley Professional.
14. Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Vol. 37, p. 20). Cambridge, UK: University of Cambridge, Department of Engineering
15. Rubí, B., Morcego, B., & Pérez, R. (2020, May). A Deep Reinforcement Learning Approach for Path Following on a Quadrotor. In 2020 European Control Conference (ECC) (pp. 1092-1098). IEEE.
16. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

17. Uhlenbeck, G. E., & Ornstein, L. S. (1930). On the theory of the Brownian motion. Physical review, 36(5), 823.
18. Options for DDPG agent, 2020, Mathworks, https://www.mathworks.com/help/reinforcement-earning/ref/rlddpgagentoptions.html
19. Healey, A. J., & Lienard, D. (1993). Multivariable sliding mode control for autonomous diving and steering of unmanned underwater vehicles. IEEE journal of Oceanic Engineering, 18(3), 327-339.
20. Fossen, T. I. (2011). Handbook of marine craft hydrodynamics and motion control. John Wiley & Sons.
21. Yuh, J. (2000). Design and control of autonomous underwater robots: A survey. Autonomous Robots, 8(1), 7-24.
22. Control Tutorial With Matlab and Simulink, https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID
23. Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., ... & Stoica, I. (2018). Ray: A distributed framework for emerging {AI} applications. In 13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18) (pp. 561-577).
24. Mixed Precision, https://www.tensorflow.org/guide/mixed_precision
25. Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.
26. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
27. Perez, T., Smogeli, O., Fossen, T., & Sorensen, A. J. (2006). An overview of the marine systems simulator (MSS): A simulink toolbox for marine control systems. Modeling, identification and Control, 27(4), 259-275.
28. Ziegler, J. G., & Nichols, N. B. (1942). Optimum settings for automatic controllers. trans. ASME, 64(11).
29. Stepinfo, https://www.mathworks.com/help/control/ref/lti.stepinfo.html;jsessionid=677c385b5003506aff375b873e85

7. APPENDIX

Equations of Motion for Aires UUV for each degree of freedom as described in [19] and implemented in [8]

The equation of motion in the surge direction is:

$$
\begin{aligned}
&m[\dot{u} - vr + wq - x_G(q^2 + r^2) + y_G(pq - \dot{r}) + Z_G(pr + \dot{q})] \\
&\quad = \frac{\rho}{2}L^4\left[X_{pp}p^2 + X_{qq}q^2 + X_{rr}r^2 + X_{pr}pr\right] \\
&\quad + \frac{\rho}{2}L^3\left[X_{\dot{u}}\dot{u} + X_{wq}wq + X_{vp}vp + X_{vr}vr + u\dot{q}\left(X_{q\delta_s}\delta_s + X_{\frac{q\delta b}{2}}\delta_{bp} + X_{\frac{q\delta b}{2}}\delta_{bs}\right)\right. \\
&\quad \left. + X_{r\delta r}ur\delta_r\right] + \frac{\rho}{2}L^2[X_{vv}v^2 + X_{ww}w^2 + X_{v\delta r}uv\delta_r \\
&\quad + uw\left(X_{w\delta s}\delta_s + X_{w\delta b/2}\delta_{bs} + X_{w\delta b/2}\delta_{bp}\right) \\
&\quad + u^2\left(X_{\delta s\delta s}{\delta_s}^2 + X_{\delta b\delta b/2}{\delta_{bs}}^2 + X_{\delta r\delta r}{\delta_r}^2\right) - (W - B)sin\theta \\
&\quad + \frac{\rho}{2}L^3X_{q\delta sn}uq\delta_s\epsilon(n) + \frac{\rho}{2}L^2\left(X_{w\delta sn}uw\delta_{sn} + X_{\delta s\delta sn}u^2{\delta_s}^2\right)\epsilon(n) + \frac{\rho}{2}L^2u^2X_{prop}
\end{aligned}
$$

The equation of motion in the sway direction is:

$$
\begin{aligned}
&m[\dot{v} + ur - wp + x_G(pq + \dot{r}) - y_G(p^2 + r^2) + z_G(qr - \dot{p})] = \frac{\rho}{2}L^4\left[Y_{\dot{p}}\dot{p} + Y_{\dot{r}}\dot{r} + Y_{pq}pq + Y_{qr}qr\right] + \frac{\rho}{2}L^3\left[Y_{\dot{v}}\dot{v} + Y_pup + Y_rur + Y_{vq}vq + Y_{wp}wp + Y_{wr}wr\right] + \frac{\rho}{2}L^2[Y_vuv + Y_{vw}vw + Y_{\delta r}u^2\delta_r] - \int_{x_{tail}}^{x_{nose}}\left[C_{dy}h(x)(v + xr)^2 + C_{dz}b(x)(w - xq)^2\right]\frac{v + xr}{U_{cf}(x)}xdx + (W - B)cos\theta sin\phi
\end{aligned}
$$

The equation of motion in the heave direction is:

$$
\begin{aligned}
&m[\dot{w} + uq - vp + x_G(pr - \dot{q}) + y_G(qr + \dot{p}) - z_G(p^2 + q^2)] \\
&\quad = \frac{\rho}{2}L^4\left[Z_{\dot{q}}\dot{q} + Z_{pp}p^2 + Z_{pr}pr + Z_{rr}r^2\right] + \frac{\rho}{2}L^3\left[Z_{\dot{w}}\dot{w} + Z_quq + Z_{vp}vp + Z_{vr}vr\right] \\
&\quad + \frac{\rho}{2}L^2\left[Z_wuw + Z_vv^2 + u^2(Z_{\delta s}\delta_s + Z_{\delta b/2}\delta_{bs} + Z_{\delta b/2}\delta_{bp})\right] \\
&\quad + \frac{\rho}{2}\int_{x_{tail}}^{x_{nose}}\left[C_{dy}h(x)(v + xr)^2 + C_{dz}b(x)(w - xq)^2\right]\frac{w - xq}{U_{cf}(x)}xdx \\
&\quad + (W - B)cos\theta sin\phi + \frac{\rho}{2}L^3Z_{qn}uq\epsilon(n) + \frac{\rho}{2}L^2[Z_{wn}uw + Z_{\delta sn}u^2\delta_s)\epsilon(n)
\end{aligned}
$$

The roll equation of motion is:

$$
\begin{aligned}
I_x\dot{p} + \left(I_z - I_y\right)qr - I_{xy}(pr - \dot{q}) - I_{yz}(\dot{q}^2 - r^2) + I_{xz}(pq + \dot{r}) \\
+ m[y_G(\dot{w} - uq + vp) - z_G(\dot{v} + ur - wq)] \\
= \frac{\rho}{2}L^5\left[K_{\dot{p}}\dot{p} + K_{\dot{r}}\dot{r} + K_{pq}pq + K_{qr}qr\right] \\
+ \frac{\rho}{2}L^4\left[K_{\dot{v}}\dot{v} + K_p up + K_r ur + K_{vq}vq + K_{wp}wp + K_{wr}wr\right] \\
+ \frac{\rho}{2}L^3\left[K_v uv + K_{vw}vw + u^2\left(K_{\delta b/s}\delta_{bp} + K_{\delta b/2}\delta_{bs}\right)\right] + (y_G W - y_B B)\cos\theta\cos\phi \\
- (z_G W - z_G B)cos\theta sin\phi + \frac{\rho}{2}L^4 K_{pn} up\epsilon(n) + \frac{\rho}{2}L^3 u^2 K_{prop}
\end{aligned}
$$

The equation of motion in the pitch direction is:

$$
\begin{aligned}
I_y\dot{q} + (I_x - I_z)pr - I_{xy}(qr + p) + I_{yz}(pq - \dot{r}) + I_{xz}(p^2 - r^2) \\
- m[x_G(\dot{w} - uq + vp) - z_G(u - vr + wq)] \\
= \frac{\rho}{2}L^5\left[M_{\dot{q}}\dot{q} + M_{pp}p^2 + M_{pr}pr + M_{rr}r^2\right] \\
+ \frac{\rho}{2}L^4\left[M_{\dot{w}}\dot{w} + M_{uq}uq + M_{vp}vp + M_{vr}vr\right] + \frac{\rho}{2}L^3[M_{uw}uw + M_{vv}v^2 \\
+ u^2\left(M_{\delta s}\delta_s + M_{\delta b/2}\delta_{bp} + M_{j\delta b/2}\delta_{bs}\right)] \\
- \int_{x_{tail}}^{x_{nose}} \left[C_{dy}h(x)(v + xr)^2 + C_{dz}b(x)(w - xq)^2\right]\frac{w + xq}{U_{cf}(x)}xdx \\
- (x_G W - x_B B)\cos\theta\cos\phi - (z_G W - z_B B)sin\theta + \frac{\rho}{2}L^4 M_{qn} uq\epsilon(n) \\
+ \frac{\rho}{2}L^3[M_{wn}uw + M_{\delta sn}u^2\delta_s)\epsilon(n)
\end{aligned}
$$

The yaw equation of motion is:

$$
\begin{aligned}
I_z\dot{r} + \left(I_y - I_x\right)pq - I_{xy}(p^2 - q^2) - I_{yz}(pr + \dot{q}) + I_{xz}(qr - \dot{p}) \\
+ m[x_G(\dot{v} - ur + wp) - y_G(\dot{u} - vr + wq)] \\
= \frac{\rho}{2}L^5\left[N_{\dot{p}}\dot{p} + N_{\dot{r}}\dot{r} + N_{pq}pq + N_{qr}qr\right] \\
+ \frac{\rho}{2}L^4\left[N_{\dot{v}}\dot{v} + N_p up + N_r ur + N_{vq}vq + N_{wp}wp + N_{wr}wr\right] \\
+ \frac{\rho}{2}L^3[N_v uv + N_{vw}vw + N_{\delta r}u^2\delta_r] \\
- \int_{x_{tail}}^{x_{nose}} \left[C_{dy}h(x)(v + xr)^2 + C_{dz}b(x)(w - xq)^2\right]\frac{v + xr}{U_{cf}(x)}xdx \\
+ (x_G W - x_B B)\cos\theta\sin\phi - (y_G W - y_B B)sin\theta + \frac{\rho}{2}L^3 u^2 N_{prop}
\end{aligned}
$$

The Euler angle rates, global positions, crossflow velocity, and propulsion terms used in the above equations of motion are:

$$\dot{\phi} = p + qsin\phi tan\theta + rcos\phi tan\theta$$

$$\dot{\theta} = qcos\phi - rsin\phi$$

$$\dot{\psi} = (qsin\phi - rcos\phi)/cos\theta$$

$$\dot{X} = u_{c0} + ucos\psi cos\theta + v[cos\psi sin\theta sin\phi - sin\psi cos\phi] + w[cos\psi sin\theta sin\phi - sin\psi sin\phi]$$

$$\dot{Y} = v_{c0} + usin\psi cos\theta + v[sin\psi sin\theta sin\phi - cos\psi cos\phi] + w[sin\psi sin\theta sin\phi - cos\psi sin\phi]$$

$$\dot{Z} = w_{c0} - usin\theta + vcos\theta sin\psi + wcos\theta sin\phi$$

$$U_{cf}(x) = [(v + xr)^2 + (w - xq)^2]^{1/2}$$

$$X_{prop} = C_{d0}(\eta|\eta| - 1);\ \eta = 0.012n/u$$

$$\epsilon(n) = -1 + sign(n)/sign(u)^*\ (\sqrt{C_t + 1} - 1)/(\sqrt{C_{t1} + 1} - 1)$$

$$C_t = 0.008L^2\eta|\eta|/2$$

$$C_{t1} = 0.008L^2/2$$

Constants uses in the equations of motion:

$W = 53.4kN$	$B = 55.4kN$	$L = 5.3m$	$I_x = 13587Nms^2$
$I_{xy} = -13.58\ Nms^2$	$I_{yx} = -13.58\ Nms^2$	$I_{xz} = -13.58\ Nms^2$	$I_y = 13587\ Nms^2$
$I_x = 2038\ Nms^2$	$x_G = 0$	$x_B = 0$	$y_G = 0$
$y_B = 0.0$	$z_G = 6.1cm$	$z_B = 0$	$g = 9.8\ m/s^2$
$\rho = 1000\ kg/m^3$	$m = 5454.54\ kg$		
$X_{pp} = 7.0e-3$	$X_{qq} = -1.5e-2$	$X_{rr} = 4.0e-3$	$X_{pr} = 7.5e-4$
$X_{\dot{u}} = -7.6e-3$	$X_{wq} = -2.0e-1$	$X_{vp} = -3.0e-3$	$X_{vr} = 2.0e-2$
$X_{q\delta s} = 2.5e-2$	$X_{q\delta b/2} = -1.3e-3$	$X_{r\delta r} = -1e-3$	$X_{vv} = 5.3e-2$

$X_{ww} = 1.7e-1$ $X_{v\delta r} = 1.7e-3$ $X_{w\delta s} = 4.6e-2$ $X_{w\delta b/2} = 0.5e-2$

$X_{\delta s\delta s} = -1e-2$ $X_{\delta b\delta b/2} = -4e-3$ $X_{w\delta s} = 4.6e-2$ $X_{q\delta sn} = 2e-3$

$X_{w\delta sn} = 3.5e-3$ $X_{\delta s\delta sn} = -1.6e-3$

$Y_{\dot{p}} = 1.2e-4$ $Y_r = 1.2e-3$ $Y_{pq} = 4e-3$ $Y_{qr} = -6.5e-3$

$Y_{\dot{v}} = -5.5e-2$ $Y_p = 3.0e-3$ $Y_r = 3.0e-2$ $Y_{vq} = 2.4e-2$

$Y_{wp} = 2.3e-1$ $Y_{wr} = -1.9e-2$ $Y_v = -1.0e-1$ $Y_{vw} = 6.8e-2$

$Z_{\dot{q}} = -6.8e-3$ $Z_{pp} = 1.3e-4$ $Z_{pr} = 6.7e-3$ $Z_{rr} = -7.4e-3$

$Z_{\dot{w}} = -2.4e-1$ $Z_q = -1.4e-1$ $Z_{vp} = -4.8e-2$ $Z_{vr} = 4.5e-2$

$Z_w = -3.0e-1$ $Z_{vv} = -6.8e-2$ $Z_{\delta s} = -7.3e-2$ $Z_{\delta b/2} = -1.3e-2$

$Z_{qn} = -2.9e-3$ $Z_{wn} = -5.1e-3$ $Z_{\delta sn} = -1.0e-2$

$K_{wp} = -1.3e-4$ $K_{wr} = 1.4e-2$ $K_v = 3.1e-3$ $K_{vw} = -1.9e-1$

$K_{\delta b/2} = 0.0$ $K_{pn} = -5.7e-4$ $K_{prop} = 0.0$

$M_{\dot{q}} = -1.7e-2$ $M_{pp} = 5.3e-5$ $M_{pr} = 5.0e-3$ $M_{rr} = 2.9e-3$

$M_{\dot{w}} = -6.8e-2$ $M_{uq} = -6.8e-2$ $M_{vp} = 1.2e-3$ $M_{vr} = 1.7e-2$

$M_{uw} = 1.0e-1$ $M_{vv} = -2.6e-2$ $M_{\delta s} = -4.1e-2$ $M_{\delta b/2} = 3.5e-3$

$M_{qn} = -1.6e-3$ $M_{wn} = -2.9e-3$ $M_{\delta sn} = -5.2e-3$

$N_{\dot{p}} = -3.4e-5$ $N_{\dot{r}} = -3.4e-3$ $N_{pq} = -2.1e-2$ $N_{qr} = 2.7e-3$

$N_{\dot{v}} = 1.2e-3$ $N_p = -8.4e-4$ $N_r = -1.6e-2$ $N_{vq} = -1.0e-2$

$N_{wp} = -1.7e-2$ $N_{wr} = 7.4e-3$ $N_v = -7.4e-3$ $N_{vw} = -2.7e-2$

$N_{\delta r} = -1.3e-2$ $N_{prop} = 0.0$

INITIAL DISTRIBUTION LIST

Addressee	No. of Copies