NUWC-NPT Technical Report 12,432
8 August 2023

Mixed Precision Deep Reinforcement Learning for Continuous Control of Complex Dynamic Systems

Christopher J. Hixenbaugh
Eugene J. Chabot
Undersea Warfare Platforms and Payload Integration Department
NUWC Division Newport

Alfa R. Heryudono
University of Massachusetts Dartmouth

Naval Undersea Warfare Center Division
Newport, Rhode Island

ADMINISTRATIVE INFORMATION

This report was prepared under NUWC Division Newport Network Activity No. 300000179238/0010, "A Model Free Reinforcement Learning Approach to UUV Control," principal investigator Christopher J. Hixenbaugh (Code 4532). The sponsoring activity is the NUWC Division Newport Internal Investment Program, program manager Derek K. Potvin (Code 00X).

The technical reviewer for this report was John DiCecco (Code 4533).

Reviewed and Approved: 13 March 2024

Rebecca I. Chhim
Acting Head, Undersea Warfare Platforms and Payload Integration Department

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
08-08-2023	Technical	01-10-2022	30-09-2023

4. TITLE AND SUBTITLE

Mixed Precision Deep Reinforcement Learning for Continuous Control of Complex Dynamic Systems

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)

Christopher J. Hixenbaugh
Eugene J. Chabot
Alfa R. Heryudono

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Undersea Warfare Center
Division Newport
1176 Howell Street
Newport, RI 02841-1708

8. PERFORMING ORGANIZATION REPORT NUMBER

TR 12,432

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Undersea Warfare Center
Division Newport
1176 Howell Street
Newport, RI 02841-1708

10. SPONSOR/MONITOR'S ACRONYM(S)

NUWC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Deep reinforcement learning (RL) shows promising results for control problems in continuous action spaces. Deep RL is disadvantaged in that it can be very computationally intensive, which is of particular concern when considering fielding deep RL applications on autonomous vehicle platforms with computational and power constraints. Mixed numerical precision methods are an area of active research where progress is being made on improving the computational efficiency of deep learning methods. While mixed-precision approaches are well understood for supervised learning tasks, this area is relatively unexplored for deep RL. The aim of this project is to fill this gap in the research by presenting a method to improve the computational efficiency of the Deep Deterministic Policy Gradient (DDPG) algorithm by using mixed numerical precision and loss scaling. Numerical cases that require continuous control of a complex dynamic system model are presented that quantify the performance and computational improvements of DDPG agents trained with mixed precision as compared with single precision.

15. SUBJECT TERMS

deep reinforcement learning, data-driven control, autonomous systems, deep learning

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
(U)	(U)	(U)	SAR	25

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
CHRISTOPHER J. HIXENBAUGH	401-832-1227

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AUV	autonomous underwater vehicle
Avg	average
DDPG	Deep Deterministic Policy Gradient (algorithm)
DOF	degrees of freedom
FP	floating point
FP16	half-precision floating point
FP32	single-precision floating point
FP64	double-precision floating point
GPU	graphics processing unit
max	maximum
MDP	Markov decision process
MF	mixed float
MF16	mixed float 16
min	minimum
MSE	mean squared error
NPSAUV	Naval Postgraduate School autonomous underwater vehicle (model)
NUWC	Naval Undersea Warfare Center
OU	Ornstein-Uhlenbeck
PID	proportional integral derivative
POS	position
PPO	Proximal Policy Optimization (algorithm)
Prop	propeller
Q value	expected reward computed by the algorithm for an action taken in a given state
RL	reinforcement learning
ReLU	rectified linear unit
SAC	soft actor-critic
SARSA	State–Action–Reward–State–Action (algorithm)
Tanh	hyperbolic tangent function
TD	temporal difference

LIST OF ABBREVIATIONS AND ACRONYMS (Cont'd)

TD3	Twin-Delayed Deep Deterministic Policy Gradient
UAV	unmanned aerial vehicle
W	weight

1. INTRODUCTION

The considerable amount of ongoing research into the use of deep reinforcement learning (RL) methods for autonomous vehicle control systems has shown promising results [1], [2]. Deep RL controllers have been shown to outperform linear quadratic Gaussian integral controllers for autonomous undersea vehicles (AUVs) that are executing path-following missions [3]. Additionally, deep RL-based controllers have been demonstrated to provide superior attitude control compared to proportional derivative integral (PID) controllers for unmanned aerial vehicles (UAVs) [4], [5]. Although these examples are not specific to AUVs, this concept from the aerial domain can be applied to the undersea environment. Current state-of-the-art deep RL algorithms that are showing promising results for control of autonomous vehicles are the DDPG [6], the Twin Delayed Deep Deterministic Policy Gradient (TD3) [7], the Soft Actor-Critic (SAC) [8], and the Proximal Policy Optimization (PPO) [9].

However, the benefits of deep RL algorithms for controlling unmanned vehicles come with a steep computational cost. It will be necessary to improve the computational efficiency of deep RL algorithms to field a control system powered by deep RL on an unmanned vehicle platform. The drawback to deep RL is that it can be more computationally intensive than is situationally acceptable. A significant contributor to the computational cost of deep RL is the large number of matrix and vector mathematical operations required to update the neural networks used to approximate the deep RL policy and value functions. Studies have shown that using mixed numerical precision [10] to improve the computational efficiency of deep learning 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 has been a trend to utilize mixed numerical precision in deep learning tasks [11], [12]. It is shown that comparable single-precision accuracy can be achieved using a combination of single- and half-precision (4-digit accuracy) but with substantial computational speedup during training.

There is limited ongoing research into using reduced precision to improve the computational efficiency of RL. In [13], the authors demonstrate how quantization techniques can improve the system performance of deep RL. In [14], the authors describe a strategy with six methods to increase numerical stability for low-precision training of the SAC algorithm. A benefit of improving the computational efficiency of deep RL algorithms is that the agent can be trained online. In online training, the agent updates itself in situ from recently collected data.

A relatively unexplored area in this research is the connection between training deep RL models with mixed precision to control dynamic models with higher complexity. The study will demonstrate experimentally that implementing mixed precision with loss scaling as presented in [12] improves computational efficiency while not degrading system performance for the DDPG algorithm. The study will demonstrate this result by applying this approach to a high-complexity dynamic system model—the Naval Postgraduate School AUV (NPSAUV) model [15].

Section 2 gives high-level background on deep RL, the DDPG algorithm, mixed numerical precision, and the NPSAUV dynamics. Section 3 describes implementation of mixed precision in the DDPG algorithm. Section 4 describes the experiment goals, setups, and results. Section 5 summarizes the results of this study and describes additional avenues for future research.

2. TECHNICAL BACKGROUND

2.1 DEEP REINFORCEMENT LEARNING

Reinforcement learning can be used to solve sequential decision-making problems such as control problems. An exampleof a control problem is adaptive cruise control for autonomous vehicles [16], [17]. Sequential decision-making problems can be abstracted and formalized as a Markov decision process (MDP). MDPs consist of three parts: states, actions, and rewards. At each time step of an MDP, the agent observes a state, $s_t \in S$, selects an action, $a_t \in A$, and receives a numerical reward, $r_t \in R \subset \mathbb{R}$. Following that sequence, the agent observes the environment's next state, s_{t+1}. The set of all nonterminal states is defined by S, A is the set of all actions available in a state s, R is the set of possible rewards, and $\mathbb{R}$ is the set of real numbers.

This information can be used to compute the state-transition probabilities defined by $p(s'|s,a) = \Pr\{s_{t-1} = s'|s_{t-1} = s, a_{t-1} = a\}$, where Pr is probability and s' is the next state [18]. The agent's goal is to select actions that affect the environment in a way that maximizes the numerical reward. The reward is a metric that is commonly used to quantify agent performance. The MDP is illustrated in Figure 1.

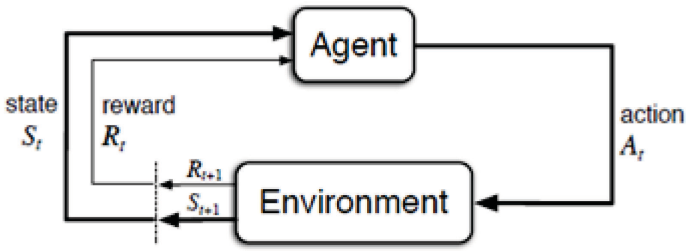

Figure 1. Markov Decision Process

2.2 DEEP DETERMINISTIC POLICY GRADIENT ALGORITHM

The DDPG algorithm is an off-policy, model-free, actor-critic deep RL algorithm. "Model-free" means that the algorithm does not rely on an environmental model to achieve optimal performance. In the context of a DDPG algorithm, "off-policy" means that the state-action function does not depend solely on the current policy used to gather experiences. A benefit of off-policy algorithms is that many past experiences can be considered when computing the Q value.[1] These experiences are given by the tuple (s_t, a_t, r_t, s_{t+1}) and are stored in a buffer. An off-policy algorithm considers the maximum Q value over all the potential actions available in a given state for a number of experiences [19]. This is different from an on-policy algorithm such as State–Action–Reward–State–Action (SARSA) [20], which relies on the Q value calculated by the experience-gathering policy.

[1] Q denotes the expected reward computed by the algorithm for an action taken in a given state.

The DDPG algorithm is an actor-critic type of algorithm. In deep RL, the policy and value functions are approximated using deep neural networks. The actor network, $\mu(s \mid \theta)$, learns a parameterized policy that computes an action according to the current state. The critic network, $Q(s, a \mid \phi)$, 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 used in both the policy and value functions during the training process. Figure 2 shows a diagram of the high-level actor-critic agent architecture.

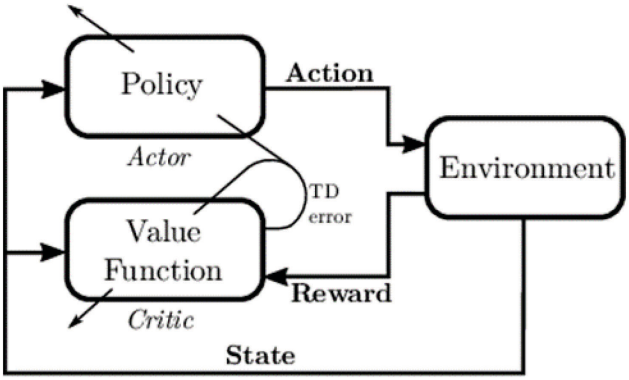

Figure 2. Actor-Critic Agent Architecture

There are two actor and critic networks in the DDPG algorithm. There is a trained network and a target network for both the actor and critic. The target networks for the actor and critic are denoted by $\mu'(s \mid \theta')$ and $Q'(s, a \mid \phi')$, respectively, where θ' are the weights of the target actor-network and ϕ' are the weights of the target critic network. The reason for the two networks is that it stabilizes training [21]. During training, the actor and critic networks are updated frequently. Training is difficult because there can be large changes to the actor and critic networks between each training step. To mitigate this issue, target networks are implemented that are updated at a lower rate. This study uses the Polyak averaging method [22] to update target actor and critic networks. The Polyak averaing method for the target actor and critic networks are given by Equations (1) and (2), respectively:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' , \tag{1}$$

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi' , \tag{2}$$

where τ is a tunable hyperparameter called the target smoothing factor that controls how the target actor and critic network weights change with respect to the networks.

The critic network is updated by minimizing the loss using the gradient descent:

$$\nabla_{-}\phi \, \frac{1}{N}\sum_{i}\left(y_i - Q(s_i, a_i|\phi)\right)^2 , \tag{3}$$

where y_i is the target value, computed as

$$y_i = r_i + \gamma Q(s_{i+1}, a_{i+1}|\phi') \,, \tag{4}$$

and γ is the discount factor.

The actor network is updated with the gradient ascent to maximize $Q_\phi\ (s, a)$:

$$\max_\theta \mathbb{E}\left[Q_\phi\left(s_i, \mu_\theta(s_i)\right)\right] . \tag{5}$$

To promote exploration in the DDPG algorithm, an exploration policy is built by adding noise, N, to the action selected by the actor network. The noise added to the policy is defined by the Ornstein-Uhlenbeck process [23]. The exploration policy is defined as:

$$\mu_s' = \mu(s|\theta) + N \,. \tag{6}$$

2.3 MIXED NUMERICAL PRECISION

Mixed-precision strategies aim to improve computational efficiency by decreasing computation time and reducing the memory required for floating-point (FP) mathematical operations. Applying this to deep RL methods will make strides toward fielding computationally intensive deep RL models on the resource-constrained computing systems implemented onboard modern AUVs, with the goal of enabling online training.

Deep RL models are very complex and require large amounts of time and computational resources to train. Scientific computations are traditionally carried out using double precision (FP64); however, many deep learning applications can be implemented using single precision (FP32). Using less precision than FP32 (e.g., half precision (FP16)) is problematic in deep learning applications because of numerical underflow and overflow.

Efforts to reduce the time and computing resources needed to train deep learning models are an active area of research. Parallel computing methods such as those described in [24] can reduce the time necessary to train a model. Mixed-precision and low-precision methods such as [11]–[13], address the time and computing resources required to train deep learning models. Mixed-precision methods reduce memory by using a combination of FP32 and FP16 to represent numbers, and time requirements are lessened by leveraging NVIDIA graphics processing units (GPUs) [25] that are designed for lower-precision mathematical operations. A mixed-precision combination of FP32 and FP16 is referred to here as "mixed float 16" (MF16).

This study is particularly interested in MF16 with loss scaling according to [12], [26]. During training, the network weights and activations are converted from FP32 to FP16, and backpropagation is executed using FP16. The loss values are scaled so that the gradients are representable in FP16, and underflow is avoided. The scaled losses are used to compute the gradients. The scaled gradients are unscaled after the backward pass but before gradient operations to prevent changing the model hyperparameters. This is possible because of the chain rule. The network weights are then updated in FP32. For this study, mixed-precision with loss scaling is implemented in the numerical experiments by using the Tensorflow Automatic Mixed-Precision library [27].

2.4 NPSAUV DYNAMICS

The AUV numerical model used in this study is the NPSAUV model as described in [15]. The authors detail a 6-degree-of-freedom (DOF) dynamic system model for this AUV. The equations of motion for this system are developed in the body-fixed reference frame as described in [28]. The velocity components of the AUV in the body-fixed reference frame are defined as

$$\dot{x} = [u(t), v(t), w(t), p(t), q(t), r(t)]\,, \tag{7}$$

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 six components describing the AUV position are given by

$$x = [x(t), y(t), z(t), \phi(t), \theta(t), \psi(t)]\,, \tag{8}$$

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 AUV, the model control inputs are given by

$$ui = \left[\delta_r(t), \delta_s(t), \delta_b(t), \delta_{bp}(t), \delta_{bs}(t), n\right], \tag{9}$$

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, $\delta_{\mathrm{bp}}(t)$ is the port bow plane angle, $\delta_{\mathrm{bs}}(t)$ is the starboard bow plane angle, and n is the propeller shaft speed.

The equation of motion for the AUV is described in terms of 12 nonlinear systems of equations as described in [29].

3. PROPOSED METHOD

In the method proposed here, mixed precision with loss scaling is successfully implemented in the DDPG algorithm without causing degradation in agent performance. The method described in [12] and [26] for mixed precision with loss scaling can be integrated as part of the DDPG algorithm updates to the actor and critic networks. The approach here uses a mixture of FP32 and FP16 precisions in the DDPG algorithm. FP16 is used to compute the gradients from dynamically scaled actor and critic loss values, while FP32 is used for all other computations in the DDPG algorithm. In [30], the authors show that the use of mixed precision for iterative methods and variants thereof is promising in terms of computational efficiency, and the convergence rates, depending on the condition of the problem, do not degrade as dimensionality increases. FP16 is used in these specific computations because it has been shown to reduce the computation time and the memory needed to execute the FP operations [10], [31]–[33] required for deep learning.

The DDPG algorithm with mixed numerical precision acceleration is given here as algorithm 1, inserted below in the text (boxed). Note that the steps in the algorithm marked with double asterisks are computations executed on the GPU and the data types used in each step are in parentheses.

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 $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta')|\phi')$ ** (FP32)
        Calculate critic loss $L_{\text{critic}} = \frac{1}{N}\sum_i \left(y_i - Q(s_i,a_i|\phi)\right)^2$ ** (MF16)
    Scale loss by dynamically determined loss factor C_{critic} ** (MF16)
    Compute scaled critic gradients $\frac{dL_{\text{critic}}}{dw_{\phi,i}}$ **. (MF16)
    Unscale critic gradients ** (FP32)
    Update critic-network weights ** (FP32)
    Approximate the actor loss $L_{\text{actor}} = -\frac{1}{N}\sum_i Q(\ s_i, \mu(s_i|\theta)\ |\ \phi)$ ** (MF16)
    Scale the actor loss by dynamically determined loss factor C_{actor} ** (MF16)
        Compute scaled actor gradients $\frac{dL_{\text{actor}}}{dw_{\theta,i}}$ ** (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

4. NUMERICAL EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Our evaluation of the DDPG algorithm with mixed numerical precision aims to answer two questions. The first question is whether training with MF16 yields computational efficiencies on a par with training with FP32 (the baseline). The second question is whether an agent trained with MF16 exhibits the same performance as an agent trained with FP32.

The two questions will be answered through numerical experiments that consider a complex dynamic system model in which a DDPG agent is trained to control the NPSAUV model in 5 DOF for speed, depth, and sway control.

Answering the first question will require an examination of GPU and memory resource consumption and the time per training step for agents trained with either MF16 or FP32. To answer the second question, the control performance of the NPSAUV model will be quantified by examining agent learning curves, step function characteristics [33] for the degrees of freedom being controlled, and the mean squared error (MSE) comparing the actual performance of the NPSAUV compared with the set points.

The MSE metric is defined as

$$\text{MSE} = \frac{1}{N}\sum_{t=0}^{N}\left(x_{\text{ref},t} - x_t\right)^2 , \tag{10}$$

where x_{ref} is the set point at time t and x_{t} is the actual value of the NPSAUV at time t.

The environmental state observations input to the deep RL agent are given by

$$\begin{aligned} s_t = (&\Delta z_t, \int \Delta z_t dt, \Delta y_t, \int \Delta y_t\, dt, \Delta u_t, \int \Delta u_t dt, \Delta\theta_t, \int \Delta\theta_t\, dt, \sin\Delta\theta_t, \cos\Delta\theta_t, \Delta\phi_t, \\ &\int \Delta\phi_t\, dt, \sin\Delta\phi_t, \cos\Delta\phi_t, \frac{dz}{dt}, \frac{dy}{dt}, \frac{d\theta}{dt}, \frac{d\phi}{dt}, \\ &\Delta z_{t+1,2,3,4}, \Delta y_{t+1,2,3,4}, \Delta\theta_{t+1,2,3,4}, \Delta\phi_{t+1,2,3,4}, \delta_{s,t}, \delta_{r,t}, w, q, r) , \end{aligned} \tag{11}$$

where $\Delta z = z_{\text{ref}} - z_t$, $\Delta y = y_{\text{ref}} - y_t$, $\Delta u = u_{\text{ref}} - u_t$, $\Delta\theta = \theta_{\text{ref}} - \theta_t$, and $\Delta\phi = \phi_{\text{ref}} - \phi_t$.

In addition to observing the differences in set point pitch and sway angles, the angle differences in Equation (11) are divided into sine and cosine components to avoid issues with periodicity.

The time derivatives of the depth and sway positions and the pitch and yaw angles are included to help the agent anticipate future tendencies of the path to be followed.

Finally, terms are incorporated for the future errors of the depth and sway positions and the pitch angle and yaw angles. The future errors of depth are defined in Equation (12). The other future errors are defined similarly. The future states are included because of the intention to incorporate forward-looking image data in future work that will provide this capability. The state observation values are scaled to be in the range of ±1.

$$\begin{aligned} \Delta z_{t+1} &= z_{\text{ref},t+1} - z_t , \\ \Delta y_{t+2} &= y_{\text{ref},t+2} - y_t , \\ \Delta\theta_{t+3} &= \theta_{\text{ref},t+3} - \theta_t , \\ \Delta\phi_{t+4} &= \phi_{\text{ref},t+4} - \phi_t . \end{aligned} \tag{12}$$

The reward function is given by

$$r_t = -\Big(c_1\Delta z_t^2 + c_2\Delta u_t^2 + c_3\Delta y_t^2 + c_4\Delta\theta_t^2 + c_5\Delta\phi_t^2 + c_6\big(\delta_{s,t} - \delta_{s,t-1}\big) + c_7\big(\delta_{r,t} - \delta_{r,t-1}\big) + c_8(n_t - n_{t-1})\Big) + S_t + A_t + Y_t + P_t\,, \tag{13}$$

where n_t is the propeller speed at time t, $\delta_{r,t}$is the rudder plane angle at time t, and $\delta_{s,t}$is the stern plane angle at time t; $A_t = 1$ when $|\Delta z_t| \leq 3.0$ meters, $A_t = 2$ when $|\Delta z_t| \leq 2.0$ meters, $A_t = 3$ when $|\Delta z_t| \leq 1.0$ meter, $A_t = 0$ otherwise; $S_t = 1$ when $|\Delta u_t| \leq 0.2$ m/s, $S_t = 2$ when $|\Delta u_t| \leq 0.1$ m/s, $S_t = 3$ when $|\Delta u_t| \leq 0.5$ m/s, $S_t = 0$ otherwise; $Y_t = 1$ when $|\Delta y_t| \leq 3.0$ meters, $Y_t = 2$ when $|\Delta y_t| \leq 2.0$ meters, $Y_t = 3$ when $|\Delta y| \leq 1.0$ meter, $Y_t = 0$ otherwise.

$P_t = -5000$—and the training episode is terminated—if $u_t > 1.885$ m/s or $u_t < 0.0$ m/s (the NPSAUV is going faster than maximum speed or is going in reverse), or $z_t < 0.0$ meter (the NPSAUV is above the water surface), or $|\Delta z_t| < 500.0$ meters, or $|\Delta y_t| < 500.0$ meters (for both $|\Delta z_t|$ and $|\Delta y_t|$, 500 meters is the error threshold set to improve training efficiency), $P_t = 0$ otherwise. When computing the reward signal, the values for Δu_t, Δz_t, Δy_t, n_t, $\delta_{s,t}$, and $\delta_{r,t}$ are absolute values, i.e., not scaled. The constants c_1, c_2, c_3, c_4, c_5, c_6, c_7, and c_8 are used to scale the components of the reward signal and are derived empirically.

The actor and critic neural network architectures are illustrated in Figure 3 and Figure 4, respectively.

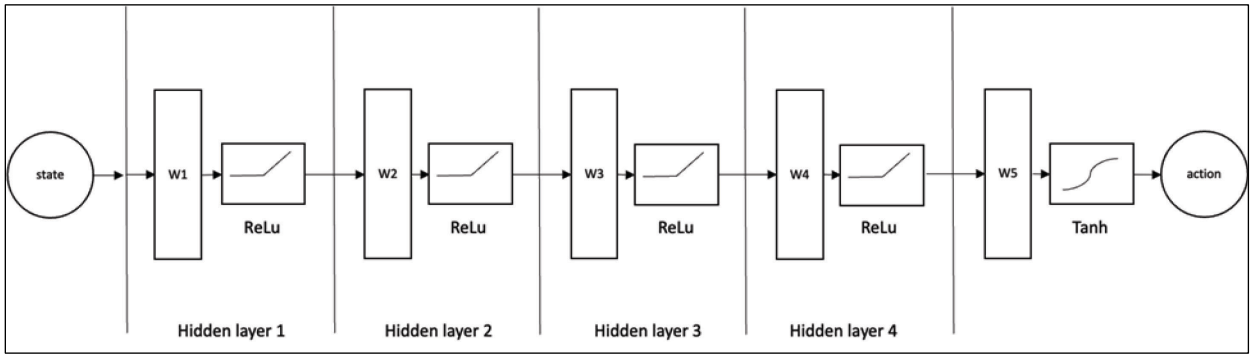

Figure 3. Actor Neural Network Architecture

For the actor deep neural network illustrated in Figure 3, $W1 = W2 = W3 = W4 = 1024$, and $W5 = 3$ weights (W), which corresponds to the number of actions. For the critic deep neural network illustrated in Figure 4, $W1 = W2 = W3 = W4 = W5 = 1024$ weights. A batch size of 1024 was used.

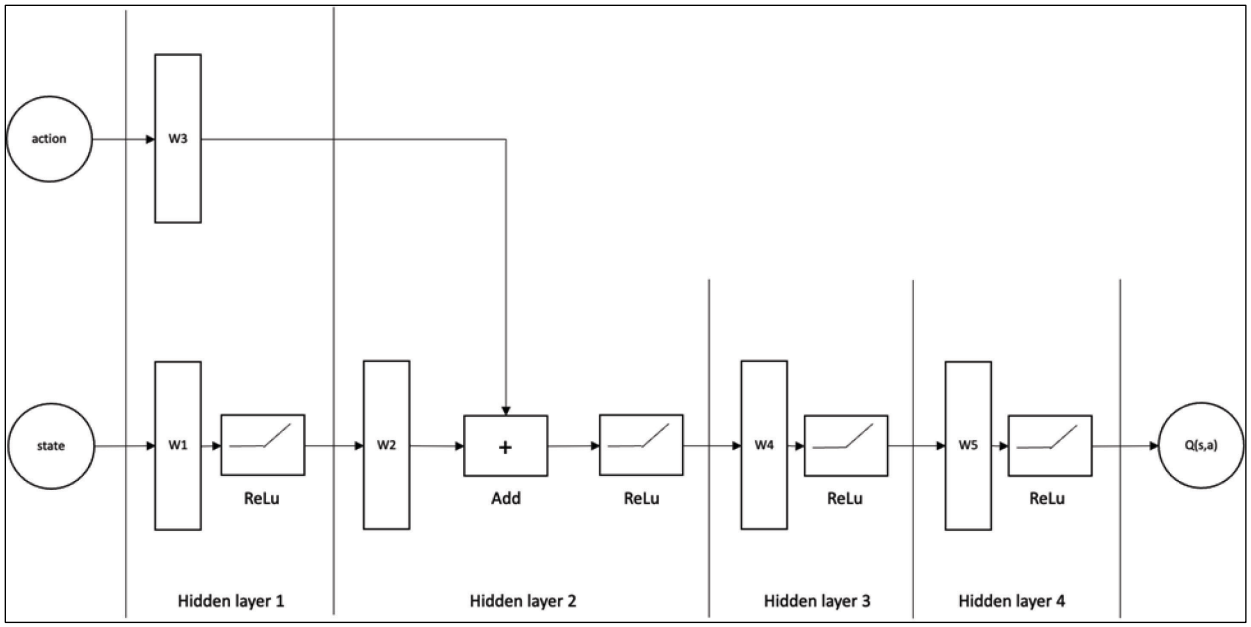

Figure 4. Critic Neural Network Architecture

In addition to the evaluation criteria stated earlier, the performance of the DDPG agents trained using MF16 and FP32 was compared with the performance of a PID controller by considering the step response characteristics and MSE metrics stated earlier. The PID controllers in this study were tuned using the Simulink Automatic PID Tuning utility and then fine-tuned manually.

The coefficients for the depth controller are $C_p = 0.025$, $C_i = 0$, $C_d = 0.001$, and $\text{FilterCoeficient} = 0$. The coeffients for the sway controller are $C_p = 0.015$, $C_i = -4.19^{e^{-4}}$, $C_d = -1.3$, and $\text{FilterCoeficient} = 20$. The coefficients for the speed controller are $C_p = 0.05$, $C_i = -4.19^{e^{-4}}$, $C_d = 0.001$, and $\text{FilterCoeficient} = 0$.

The DDPG agent was trained for 7,000 episodes with an average scoring window of 10 episodes. Training episodes are 600 time steps (seconds) in duration unless a termination condition is met.

The simulation environments and deep RL algorithms were implemented using Tensorflow 2.1 and the Tensorflow Automatic Mixed-Precision library [27]. The numerical experiments were performed using an NVIDIA RTX 5000 GPU.

4.2 EXPERIMENTAL RESULTS

Here, results were examined from experiments in which the best-trained DDPG agents are trained with FP32 and MF16 to control the depth, sway, and speed of the complex NPSAUV dynamic system model. In these experiments, the DDPG agent is trained to start from rest and achieve a set point surge speed while simultaneously starting at initial depth and sway positions and following set point depth and sway step function paths. This type of task was used for this comparison because it is convenient to quantify results using step function characteristics. The

results are also compared with the performance of a PID controller for the NPSAUV model. These experiments illustrate the performance of the NPSAUV model, controlled by DDPG agents trained with both FP32 and MF16, in which the modeled AUV

- starts from rest and achieves a set point speed of 1.2 m/s,
- starts from an initial depth of 1000 meters and dives to a set point depth of 1015 meters, and
- starts from an initial sway position of 0 meters and maneuvers to a set point sway position of −10 meters.

Figure 5 shows the average reward learning curves of DDPG agents trained to control the NPSAUV using FP32 and MF16.

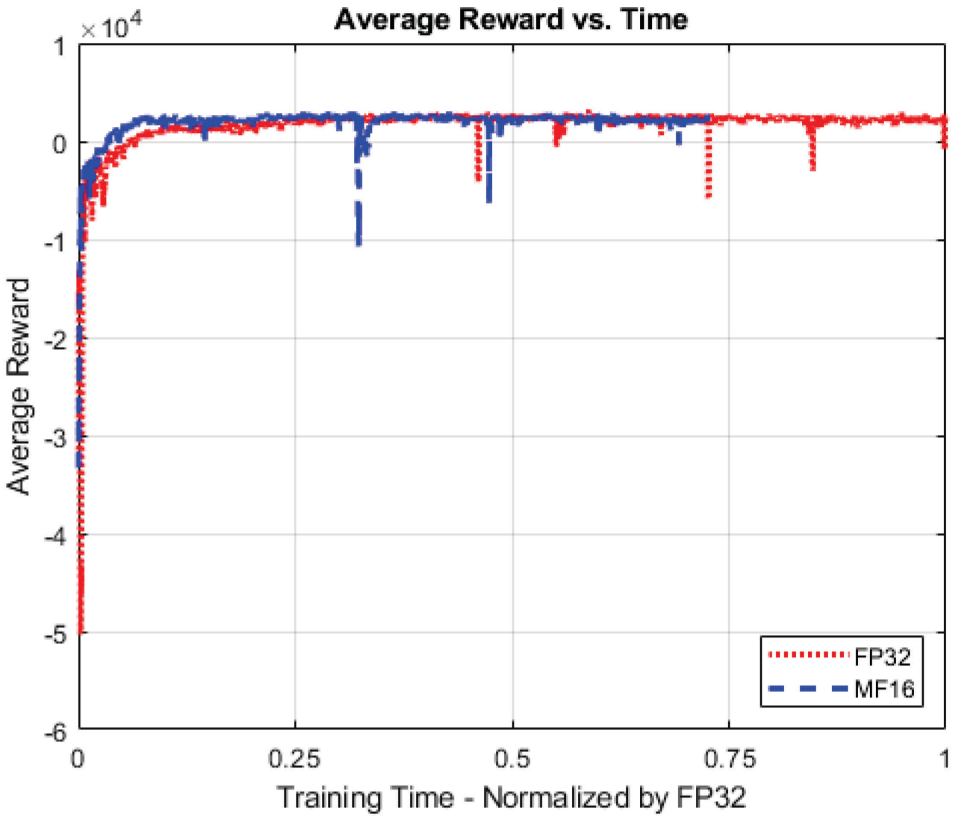

Figure 5. Average Reward Learning Curve Comparison: DDPG Agents Trained with FP32 Versus MF16

Figure 6, Figure 7, and Figure 8 illustrate the performance of the best-trained DDPG agents trained with FP32 and MF16 and the PID controller in controlling the NPSAUV surge velocity, depth, and sway, respectively. These performance characteristics—surge velocity, depth, and

sway—are quantified using step function characteristics and MSE in Table 1, Table 2, and Table 3, respectively.

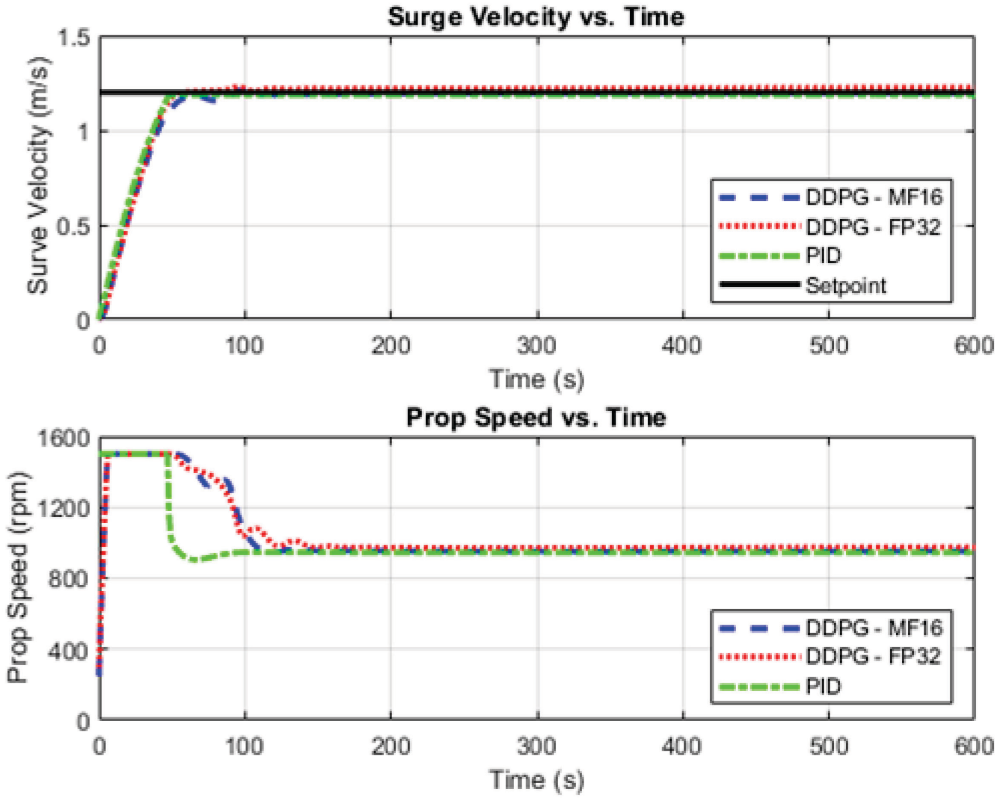

Figure 6. Performance: DDPG Agents Trained with MF16 or FP32 Versus PID—(top) Surge Velocity and (bottom) Propeller Speed Versus Time

Table 1. Step Function Characteristics of 5-DOF PSAUV Model Performance with PID, FP32, and MF16 Control: Surge Velocity

Controller	PID	FP32	MF16
Rise time (s)	37.73	36.01	39.48
Settling time (s)	38.12	39.41	41.72
Settling min (m)	1.09	1.09	1.09
Settling max (m)	1.19	1.23	1.21
Overshoot (%)	0	2.57	0.88
Undershoot (%)	0	0	0
Peak (m)	1.19	1.23	1.21
Peak time (s)	66	96	96
MSE	0.03	0.04	0.04

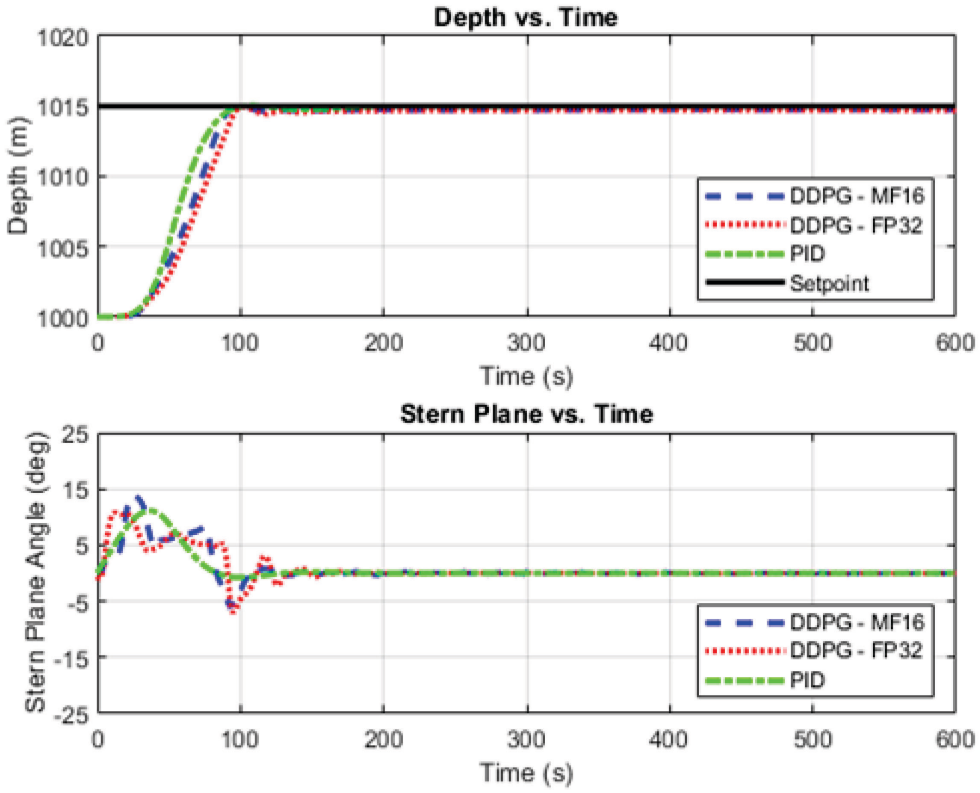

Figure 7. **Performance: DDPG Agents Trained with MF16 or FP32 Versus PID—(top) Depth and (bottom) Stern Plane Angle Versus Time**

Table 2. **Step Function Characteristics of 5-DOF PSAUV Model Performance with PID, FP32, and MF16 Control: Depth**

Controller	PID	FP32	MF16
Rise time (s)	43.8	52.12	48.09
Settling time (s)	86.26	94.41	88.84
Settling min (m)	1013.5	1013.68	1013.69
Settling max (m)	1015.04	1014.96	1015.07
Overshoot (%)	0	0	0
Undershoot (%)	0	0	0
Peak (m)	1015.04	1014.96	1015.07
Peak time (s)	107	102	98
MSE	18.07	21.23	19.98

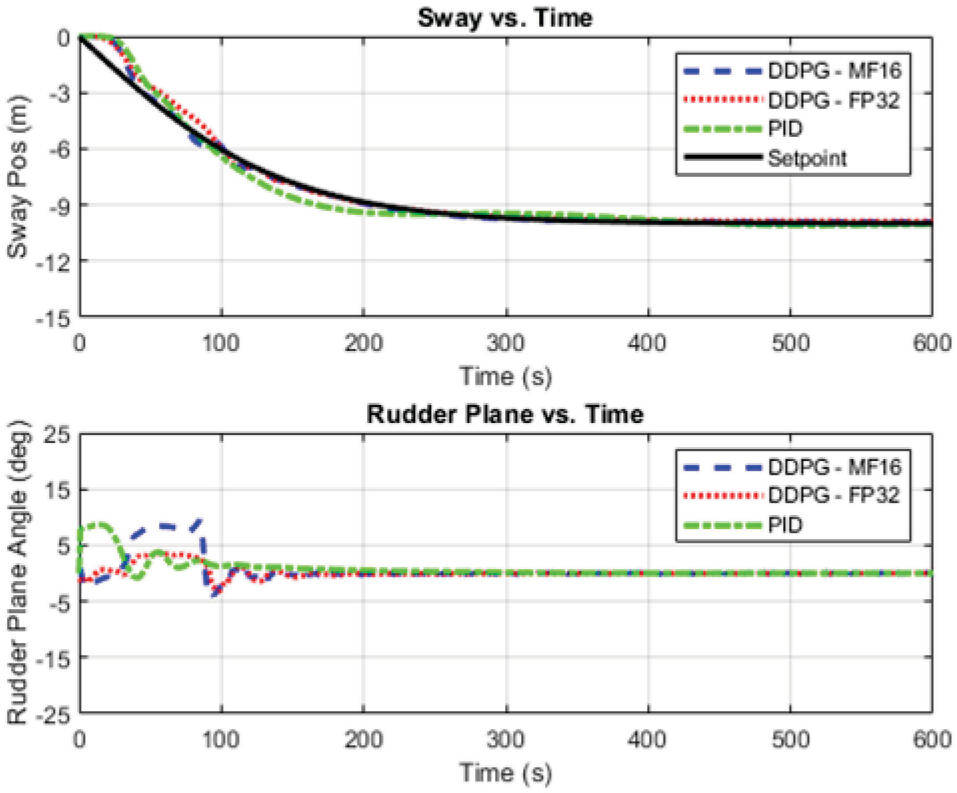

Figure 8. Performance: DDPG Agents Trained with MF16 or FP32 Versus PID—(top) Sway and (bottom) Rudder Plane Angle Versus Time

Table 3. Step Function Characteristics of 5-DOF PSAUV Model Performance with PID, FP32, and MF16 Control: Sway

Controller	PID	FP32	MF16
Rise time (s)	131.34	178.84	173.88
Settling time (s)	326.93	266.49	253.98
Settling min (m)	−10.09	−9.88	−9.92
Settling max (m)	−9	−9	−9
Overshoot (%)	0.98	0	0
Undershoot (%)	0.02	0	0
Peak (m)	10.09	9.88	9.92
Peak time (s)	518	600	593
MSE	10.5	10.98	10.16

Considering the results of these experiments, we suggest that the DDPG agents trained with MF16 or FP32 and the PID controller perform equivalently. If we exclude the PID controller and compare just the performance of the DDPG agents, the same observation can be made: the agent trained with FP32 and the agent trained with MF16 outperform one another for individual

parameters, but neither DDPG agent outperforms the other for all parameters being considered. The evidence shows that a DDPG agent trained with MF16 performs equivalently to an agent trained with FP32.

The effects on computation of training a DDPG agent with FP32 compared with MF16 for control of the NPSAUV are illustrated in Table 4—training time—and Table 5—consumption of computational resources.

Table 4. Agent Training Time: FP32 Versus MF16

Precision	FP32	MF16
Avg time/training step (s)	0.019	0.013
Environment steps	4,207,601	4,207,601

Table 5. Computational Resource Consumption: FP32 Versus MF16

Precision	FP32	MF16
Memory utilization (%)	40.50	32.60
GPU utilization (%)	70.28	48.13

Control of the NPSAUV dynamic system model utilizes deep neural networks that are complex enough to realize a 32% reduction in computation time when training a DDPG agent with MF16 compared with training with FP32. Additionally, training with MF16 compared with FP32 resulted in a 20% reduction in memory utilization and a 32% reduction in GPU utilization. The memory utilization and GPU utilization percentage metrics are measured using the NVIDIA System Management Interface (SMI).

Considering the significant benefits demonstrated in this example, training with mixed precision will be even more attractive as the problems become more complex and thus inherently more computationally intensive. Control that incorporates learning from pixel data to control complex dynamic systems is just one advance that could make the deep RL model more computationally demanding to the point at which training with MF16 will be even more computationally advantageous.

5. CONCLUSIONS AND FUTURE WORK

This study demonstrated that the DDPG algorithm can be trained using single-precision (FP32) accuracy or a combination of single-precision and half-precision (FP16) accuracy—called mixed numerical precision accuracy (or mixed float 16 (MF16))—to perform equivalently for continuous control of a complex dynamic system such as the NPSAUV model.

The study also demonstrated that DDPG agents trained with either single (FP32) or mixed precision (MF16) with loss scaling perform similarly to a PID controller in controlling the NPSAUV model, that is, with no degradation of system performance.

Further, the study showed that training a DDPG agent for the NPSAUV control problem using mixed precision provided significant computational efficiencies compared with training the agent using single precision. The mixed-precision method reduced memory by using a combination of FP32 and FP16 to represent numbers, and the time requirements were reduced by leveraging GPUs designed for lower-precision mathematical operations.

Future work in this area includes incorporating pixel-based data for control of the NPSAUV to further illustrate the attractiveness of training with mixed precision as problems in AUV control become more computationally intensive. Additional areas of interest include exploring the effects of training other deep RL algorithms—particularly the Twin Delayed Deep Deterministic Policy Gradient (TD3) and the Proximal Policy Optimization (PPO)—with the mixed-precision approach utilized in this study.

REFERENCES

[1] Y. Hsu, H. Wu, K. You, and S. Song, "A selected review on reinforcement learning based control for autonomous underwater vehicles," *Frontiers Inf. Technol. Electro. Eng.*, arXiv:1911.11991 [cs], Nov. 2019. [Online]. Available: http://arxiv.org/abs/1911.11991

[2] R. Yu, Z. Shi, C. Huang, T. Li, and Q. Ma, "Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle," in *2017 36th Chin. Control Conf. (CCC)*, Dalian, China, Jul. 2017, IEEE, pp. 4958–4965. doi: 10.23919/ChiCC.2017.8028138. [Online]. Available: http://ieeexplore.ieee.org/document/8028138/

[3] H. Wu, S. Song, K. You, and C. Wu, "Depth Control of Model-Free AUVs via Reinforcement Learning," *IEEE Trans. Systems, Man. Cybern.: Syst.*, vol. 49, no. 12, pp. 2499–2510, Dec. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8263166

[4] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen,"Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization," in *2019 Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, Atlanta, GA, USA: IEEE, Jun. 2019, pp. 523–533. doi: 10.1109/ICUAS.2019.8798254. [Online]. Available: https://ieeexplore.ieee.org/document/8798254/

[5] W. Koch, "Flight Controller Synthesis Via Deep Reinforcement Learning," arXiv:1909.06493 [cs, eess, stat], Sep. 2019. [Online]. Available: http://arxiv.org/abs/1909.06493

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D.Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv:1509.02971 [cs, stat], Jul. 2019. [Online]. Available: http://arxiv.org/abs/1509.02971

[7] S. Fujimoto, H. Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proc. 35th Int. Conf. Mach. Learn.,* PMLR, vol. 80, Jul. 2018, pp. 1587–1596.

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proc. 35th Int. Conf. Mach. Learn.,* PMLR, Jul. 2018, pp. 1861–1870.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv:1707.06347 [cs], Aug. 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[10] N. J. Higham, "The Rise of Multiprecision Arithmetic," in *2017 IEEE 24th Symp. Comput. Arithmetic (ARITH)*, Jul. 2017, pp. 1–1, doi: 10.1109/ARITH.2017.24. [Online]. Available: http://ieeexplore.ieee.org/document/8023056/

[11] S. R. Nandakumar, M. Le Gallo, I. Boybat, B. Rajendran, A. Sebastian, and E. Eleftheriou, "Mixed-precision architecture based on computational memory for training deep neural networks," in *2018 IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, IEEE, May 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8351656/

[12] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh. and H. Wu, "Mixed Precision Training," arXiv:1710.03740 [cs, stat], Feb. 2018. [Online]. Available: http://arxiv.org/abs/1710.03740

[13] S. Krishnan, M. Lam, S. Chitlangia, Z. Wan, G. Barth-Maron, A. Faust, and V. J. Reddi, "QuaRL: Quantization for Fast and Environmentally Sustainable Reinforcement Learning," arXiv:1910.01055 [cs], Nov. 2021. [Online]. Available: http://arxiv.org/abs/1910.01055

[14] J. Bjorck, X. Chen, C. De Sa, C. P. Gomes, and K. Q. Weinberger, "Low-Precision Reinforcement Learning: Running Soft Actor-Critic in Half Precision," arXiv:2102.13565 [cs], Jun. 2021. [Online]. Available: http://arxiv.org/abs/2102.13565

[15] A. J. Healey and D. Lienard, "Multivariable sliding mode control for autonomous diving and steering of unmanned underwater vehicles," *IEEE J. Ocean. Eng.*, vol. 18, no. 3, Jul. 1993, pp. 327–339. doi: 10.1109/JOE.1993.236372. [Online]. Available: https://ieeexplore.ieee.org/document/236372/

[16] Y. Zhang, P. Sun, Y. Yin, L. Lin, and X. Wang, "Human-like Autonomous Vehicle Speed Control by Deep Reinforcement Learning with Double Q-Learning," in *2018 IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 1251–1256.

[17] J. Laumônier, C. Desjardins, and B. Chaib-Draa, "Cooperative Adaptive Cruise Control: A Reinforcement Learning Approach," DAMAS Laboratory, Laval University, Canada, in *The Fourth Workshop on Agents in Traffic and Transportation*, Hakodate, Hokkaido, Japan, 2006.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[19] L. Graesser and W. L. Keng, *Foundations of Deep Reinforcement Learning: Theory and Practice in Python* (Pearson Addison-Wesley Data & Analytics Series), 1st ed. Boston, MA, USA: Addison-Wesley Professional, 2019, ISBN: 9780135172384.

[20] G. A. Rummery and M. Niranjan, "On-Line Q-Learning Using Connectionist Systems," Univ. Cambridge, Dept. of Eng., Cambridge, UK, CUE/F-INFE/TR-166, Sep. 1994.

[21] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, Feb. 2015, pp. 529–533. doi:10.1038/nature14236. [Online]. Available: http://www.nature.com/articles/nature14236

[22] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM J. Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992.

[23] G. E. Uhlenbeck, and L. S. Ornstein, "On the Theory of the Brownian Motion," *Physical Rev.*, vol. 36, no. 5, 1930, p. 823. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.36.823

[24] P. Moritz et al., "Ray: A Distributed Framework for Emerging AI Applications," in *Proc. 13th USENIX Symp. Operating Systems Design and Implementation (OSDI '18)*, Oct. 2018, Carlsbad, CA, USA, pp. 561–577. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/moritz

[25] N. Whitehead and A. Fit-Florea, "Precision & Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs," 2011, pp. 18749–19424. [Online]. Available: https://developer.download.nvidia.com/assets/cuda/files/NVIDIA-CUDA-Floating-Point.pdf

[26] "Train with Mixed Precision User Guide," NVIDIA Docs DA-08617-001_v001, NVIDIA Corporation, Santa Clara, CA, USA, March 2022, https://docs.nvidia.com/deeplearning/performance/pdf/Training-Mixed-Precision-User-Guide.pdf

[27] "Mixed precision," in TensorFlow Core, TensorFlow.org. [Online]. Available : https://www.tensorflow.org/guide/mixed_precision

[28] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*, Wiley, Hoboken, NJ, USA, 2011. ISBN: 9781119994138. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119994138

[29] J. Yuh, "Design and Control of Autonomous Underwater Robots: A Survey," *Auton. Robots*, 2000, vol. 8, no. 1, pp. 7–24. doi: 10.1023/A:1008984701078.

[30] C. T. Kelley, "Newton's Method in Mixed Precision," *SIAM Rev.*, vol. 64, no. 1, Feb. 2022 pp. 191–211.

[31] A. Abdelfattah, S. Tomov, and J. Dongarra, "Fast Batched Matrix Multiplication for Small Sizes using Half-Precision Arithmetic on GPUs," in *2019 IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2019, pp. 111–122.

[32] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," in *Proc. 32nd Int. Conf. Mach. Learn.*, PMLR, Jun. 2015, pp. 1737–1746.

[33] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, "Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers," in *SC18: Int. Conf. High Performance Computing, Networking, Storage and Analysis*, IEEE, 2018, pp. 603–613.

[34] "Stepinfo," The MathWorks, Inc., Natick, MA, USA, 2021. [Online]. Available: https://www.mathworks.com/help/control/ref/lti.stepinfo.html;jsessionid=677c385b5003506aff375b873e85/

[35] "Dense layer," Keras documentation, github.com. [Online]. Available: https://keras.io/api/layers/core_layers/dense/

INITIAL DISTRIBUTION LIST

Internal
Code(s): 1033 Corporate Research and Information Center (CRIC)

External
Defense Technical Information Center (DTIC)

Total: 2