Deep Learning for Station Keeping of AUVs

1st Kristoffer Borgen Knudsen
Dept. of Marine Technology
NTNU
Trondheim, Norway
kristobk@stud.ntnu.no

2nd Mikkel Cornelius Nielsen
Dept. of Marine Technology
NTNU
Trondheim, Norway
mikkel.cornelius.nielsen@ntnu.no

3rd Ingrid Schjølberg
Dept. of Marine Technology
NTNU
Trondheim, Norway
ingrid.schjolberg@ntnu.no

***Abstract*—Control of underwater vehicles remains an active research topic within the literature. Multiple challenges exists for controlling an underwater vehicle, including highly nonlinear effects due to hydrodynamics. Control based models seek to model the underlying dynamics but suffer from the balance between tractable computation and performance. Machine Learning (ML) control techniques show promise as an alternative to classical model-based approaches.**

This article investigates the application of a model-free deep reinforcement learning algorithm, Deep Deterministic Policy Gradient (DDPG), for station keeping in six degrees of freedom (DOF) for an underwater vehicle.

***Index Terms*—underwater robotics, station keeping, deep reinforcement learning, deep deterministic policy gradients**

I. Introduction

Station keeping denotes the act of maintaining a constant position and orientation (pose), relative to a reference object [1]. Many underwater operations rely on the underwater vehicles station keeping capabilities, and thus station keeping represents a fundamental control task.

The controller design is crucial for accomplishing station keeping. Unfortunately, the underwater environment is complex, making the autonomous control nonlinear since flow and hydraulic resistance governs the motion of AUVs [2]. The complexity of the underwater dynamics complicates the expirission of the physical models. The complication causes difficulties for the application of traditional model-based control. Multiple researchers have considered control for station-keeping of underwater vehicles. Smallwood and Whitcomb [3] investigated multiple model-based control designs including a PD-control, an exact linearizing feedback controller and an approximate version. The author note that the performance of the model-based controllers remain robust for sensor noise, but that the performance suffer severely for wrong model parameters. Azis *et al.* [4] reviews multiple control methods including LQR, PID, Sliding Modes, Fuzzy methods, and Neural Networks and states that each method comes at a disadvantage. [5] proposed to use a sliding mode estimator with fuzzy gains to estimate the thruster model and exploit the knowledge in a feed-forward strategy. However, the presented results relied on simulation. [6] applied a Hybrid-Neuro-Fuzzy-Network, which uses an indirect adaptive approach, to simultanuously identify and control an AUV. The article shows results in both simulation and in an actual system. The challenges of controlling AUVs, in combination with the rapid

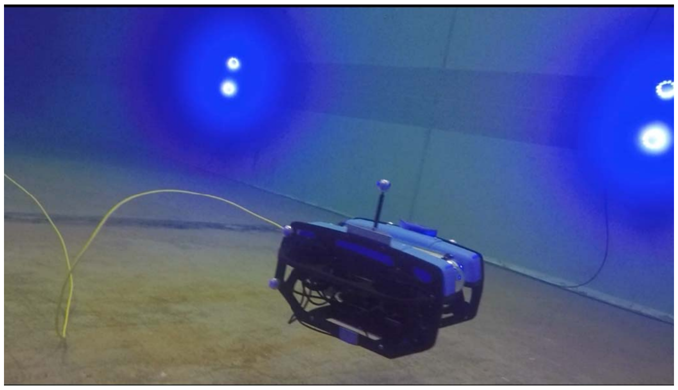


Figure 1: BlueROV2 with Qualisys markers attached for motion tracking in Marine Cybernetics Laboratory (MCLAB).

developments in artificial intelligence (AI), have triggered the interest of applying machine learning (ML) techniques in AUV control designs.

As mentioned above, the main reason for investigating the possibilities of ML in underwater control is that identifying and modeling a real-life system is challenging and in some cases infeasible due to unobservability and highly nonlinear effects, which is the case in the underwater environment [7]. Reinforcement Learning (RL) posits a potential avenue of research within ML control. Model-free RL (MFRL) does not assert any prior knowledge about the underlying system. Instead, MFRL seeks to learn the system through interaction and apply that knowledge to maximize a reward. In other words, MFRL constitute an adaptive optimal control strategy. Due to the potential benefits, multiple authors have investigated the application in various dynamic systems. De Paula and Acosta [8] proposed a trajectory tracking algorithm for autonomous vehicles using *Adaptive Reinforcement Learning*, but this was too general for usage in AUV control. One of the more significant difficulties, which is also one of the primary goals of AI in control, is to solve complex tasks from unprocessed, high-dimensional and sensory input [9]. Based on this, David Silver et al. proposed a *Deterministic Policy Gradient* (DPG) algorithm, for performing complex tasks with high dimensionality and perceptible input [10]. This algorithm showed significantly better performance than using *Stochastic Policy Gradients* (SPG), as well as having usage within nonlinear optimization problems as well. Yu *et al.* [2] based their research on this when they in 2017 proposed a *Deep Deterministic Policy Gradient* (DDPG) for trajectory tracking control of AUVs, which showed significant improvements compared to traditional methods.

This article investigates the application of Deep Reinforcement Learning to enable station keeping for AUVs by proposing a dual controller design. The dual controller design encompasses a DDPG algorithm, based on the work of Silver *et al.* [10] and Yu *et al.* [2], in conjunction with a PD controller.

The article starts by describing the DDPG algorihm in Section II, then Section III describes the simulation and real-life experiments. Section IV presents the results and discusses the validity before Section V presents the conclusions and future work.

II. Method

Reinforcement Learning (RL) aims at solving tasks through an agent, who seek to maximize a reward signal obtained from the environment in which the agent acts. The reward signal informs the agent on how good the action taken in a particular state was. The agent progressively learns the environment through rewards and tries to maximize the accumulated rewards over all states in the environment by finding an optimal policy to follow.

A. Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradients (DDPGs) is a state-of-the-art DRL algorithm, which utilizes Artificial Neural Networks (ANNs) to learn a policy. DDPG has shown remarkable results on both benchmark computer games [9] and trajectory tracking control of AUVs [2].

The DDPG algorithm belongs to the category of poly gradients that exhibit model-free off-policy learning in an actor-critic framework. The policy gradient nature allows the algorithm to continuously compute noisy estimates from the gradient of the expected reward and optimize the reward using the estimated gradient. The algorithm also belongs to the acto-critic family meanin that the policy function, the actor, remains independent of the value function, the critic. The separation of the two means that the critic estimates the value of the actions produced by the actor, and the actor learns a better policy based on the input of the critic. Off-policy learning refers to the notion that the algorithm does not need to control the system to learn better policies. Feeding the algorithm old data still allows DDPG to optimize the networks and produce better policies. The benefit of having this separation is that the policy might be deterministic, e.g greedy, while the policy function can continue to sample all possible actions in every state. The result from this is that the agent does not convergence towards a sub-optimal policy.

Finally, the model-free characteristics of the algorithm means that the algorithm does rely on assumptions of the environment.

B. Exploration vs. Exploitation

RL algorithms operate under the concept of exploration versus exploitation. Exploration allows the system to learn about the mapping between state, actions, and rewards, while exploitations utilize the learned knowledge to maximize the reward through the actions at a given state. Only exploiting knowlege would lock the agent to a potential sub-optimal return of rewards, and in contrast by only exploring, the agent never seeks actual reward. Thus, the exploration versus explotaition represents a trade-off between learning about potentially higher rewards in the system and exploiting the known system for maximum return of reward. A simple, yet effective solution is to introduce an noise parameter ϵ, which represents a probability that the next action taken will be random exploration.

C. Reward function

The reward function r defines the framework of the control objective, and optimal design of the reward function is essential for accomplishing sufficient behavior of the agent. Unfortunately, the reward function design also needs to take convergence of the DDPG algorithm into account. In DDPG algorithms and policy gradient (PG) methods, in general, the steepest ascent (or descent) direction, g, of the reward is computed. g is used to update the policy θ by taking a gradient step αg in the steepest direction. The problem with this step is that it is a first-order derivative, which essentially means that the reward function is assumed to be flat. However, if the reward function is not flat, the agent might take a too large step resulting in a long time needed to reach convergence. The fact that the algorithm is a PG puts a constraint on our reward function design, which means that accomplishing a sufficient reward function is difficult.

Furthermore, the number of states controlled by the DDPG algorithm also impacts the time needed to reach convergence towards an optimal policy. Due to this, in order to limit the impact on convergence time, a dual controller design is used. The dual controller splits splits the fully actuated BlueROV2 by using a DDPG algorithm to control the (NED) position in surge x and sway y, and a PD algorithm to control the (NED) position in heave z and the (NED) orientations in pitch ϕ, roll θ and yaw ψ. The dual controller design architecture is visualized in Fig. 2.

III. Experiments

A. Modeling

This article applies DDPG to the station keeping problem on a BlueROV2 platform. The BlueROV2 is a small and low-cost remotely-operated-vehicle (ROV) shown in Fig. 1.

A DRL algorithm requires many training samples for the algorithm to converge. Therefore, simulation-based training provided the foundation for the algorithm. The Gazebo simulator with the UUV_SIMULATION [11] plugin and a dynamic model of the BlueROV2 [12] provided a simulation environment for the training. Real-life experiments conducted in Marine Cybernetics Laboratory (MCLab), in Trondheim,

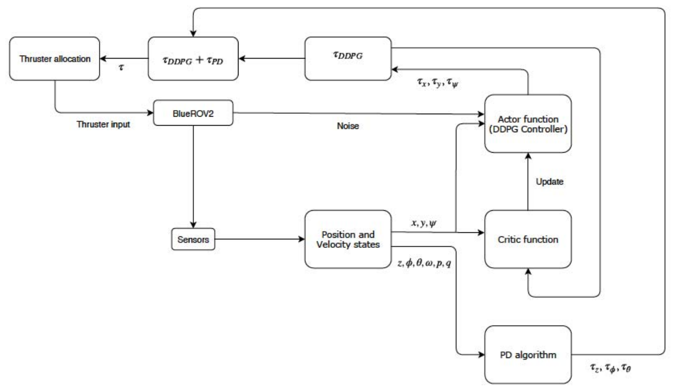

Thruster allocation
$\tau_{DDPG} + \tau_{PD}$
τ_{DDPG}
τ
$\tau_x, \tau_y, \tau_\psi$
BlueROV2
Thruster input
Noise
Actor function (DDPG Controller)
Update
Sensors
Position and Velocity states
x, y, ψ
$z, \phi, \theta, \omega, p, q$
Critic function
PD algorithm
$\tau_z, \tau_\phi, \tau_\theta$

Figure 2: Control system architecture for the BlueROV2.

provided the validation for the training. Fig. 4 shows the two reference frames utilized in the article; the local body-fixed frame attached to the BlueROV2, denoted by $\{b\}$, and the global earth-fixed North-East-Down (NED) reference frame, denoted by $\{n\}$. Transforming the velocity vector from body-fixed frame to NED frame requires a transformation matrix $\boldsymbol{R}_b^n$, such that,

$$\mathbf{v}_{b/n}^n = \mathbf{R}_b^n(\Theta_{nb})\mathbf{v}_{b/n}^b. \tag{1}$$

The transformation matrix $\mathbf{R}_b^n$ from the local body-fixed frame to the NED frame relies on the Euler angles. Assuming the pitch rotation ϕ and the roll rotation θ *small*, the rotation matrix from (Body) to (NED) can be expressed as,

$$\boldsymbol{R}(\psi) = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2}$$

where $c = cos$ and $s = sin$, and ϕ, θ, and ψ represents the roll, pitch and yaw, respectively [13].

B. Implementation

The DDPG controller controls the two horizontal states, x and y using two neural networks. The two neural networks forms the actor network and the critic network as presented in Yu *et al.* [2]. In order to define the neural networks, the equation of motions for underwater vehicles [13] is used.

$$M\dot{v} + C(v)v + D(v)v + g(\mu) + \delta = \tau. \tag{3}$$

By assuming non-singularity of the mass matrix, $M = M_{RB} + M_A$, Equation 3 is modified as follows,

$$\dot{v} = M^{-1}(\tau - D(v)v - g(\eta) - C(v)v - \delta), \tag{4}$$
$$\Re(\psi)v = \dot{\eta}. \tag{5}$$

Applying the first-order Taylor expansion to this results in

$$v(t+1) = M^{-1}(\tau + G(t)), \tag{6}$$
$$\eta(t+1) = \boldsymbol{R}(\psi(t))v(t), \tag{7}$$
$$G(t) = (-D(v(t))v - g(\eta(t)) - C(v(t))v(t) - \omega), \tag{8}$$

where ω is noise from the environment, and t is a certain moment of the system. Then, the controller τ is a defined as a function of the position and velocity of the previous moment, such that,

$$\tau(t) = \mu(v(t), \eta(t)), \tag{9}$$

which is simplified to,

$$\tau(t) = \mu(s_t) \tag{10}$$
$$s_t = [v(t), \eta(t)]^T. \tag{11}$$

The controller τ applies a thrust matrix input to the system in the given state, which means that it can defined as an action (a_t) that the vehicle executes in the given state (s_t). Since the goal is to accomplish station keeping of the AUV at a desired state s_d, this means that the actor function is designed to minimize the state error through minimizing the reward function, given by,

$$R(s_t, a_t) = \int_t^\infty \gamma^{-(k-t)} r(s_t, a_t) dk. \tag{12}$$

where $\gamma \in (0, 1)$ is a discount factor, which is used to reduce the influence from possible future states. Conceptually, the control problem becomes an optimization which is defined as follows,

$$\arg\min_{s_t, a_t}\{R(s_t, a_t)\} \tag{13}$$
$$\text{s.t. } a_{min} \le a_t \le a_{max} \tag{14}$$
$$s_{min} \le s_t \le s_{max}. \tag{15}$$

In Equation 15, a_{min} and a_{max} represents the minimum and maximum thrust input from the controller, respectively. The BlueROV2 has a maximum velocity of 2 [m/s], but in simulation this is set to half of this, resulting in $a_{min} = -1[N]$ and $a_{max} = +1[N]$.

1) Critic Function: To solve the **argmin** optimization problem, the critic function is defined as

$$Q(s_t, a_t) = R(s_t, a_t) = \int_t^\infty \gamma^{-(k-t)} r(s_t, a_t) dk \tag{16}$$

$$\text{Discretize } \longrightarrow R(s_t, a_t) = \sum_{i=t}^{\infty} \gamma^{(i-t)} r(s_t, a_t) \tag{17}$$

The **Bellman Equation** states that,

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) \tag{18}$$

This gives the **optimal** critic function as,

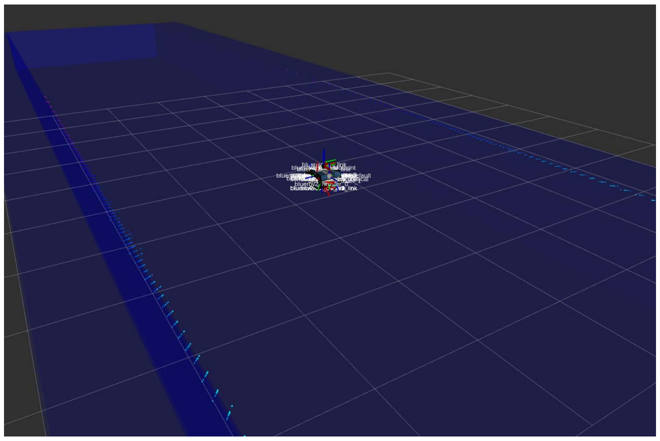
$$\hat{Q}(s_t, a_t) = \arg\min_{s_t, a_t}(Q(s_t, a_t)). \tag{19}$$

Replacing the critic function with a *neural* network results in,

$$Q(s_t, a_t) \longrightarrow Q(s_t, a_t|\omega). \tag{20}$$

To evaluate the policy it is first necessary to find the optimal critic function in Equation 19. This is done by defining a Loss function

$$Loss = \frac{1}{2}(y_t - Q(s_t, a_t|\omega))^2 \tag{21}$$
$$y_t = r(s_t, a_t) + \gamma Q(s_t, a_t|\omega) \tag{22}$$



(a) The figure shows ROS Gazebo simulation of BlueROV2 during training.

(b) Marine Cybernetics Laboratory (MCLAB) at Norwegian University of Science and Technology (NTNU).

Figure 3: Figure shows the ROS training simulation, as well as the actual MCLAB where experiments were conducted.

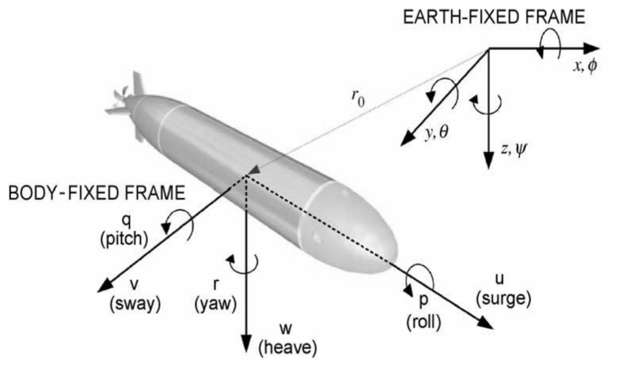


Figure 4: The two reference frames utilized in the article; body-fixed and earth-fixed frame.

By then using the policy gradient algorithm from Silver *et al.* [10] with a sampled batch from (s_t, a_t), the average Loss function, and its gradient, are given as

$$\bar{Loss} = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\omega))^2 \tag{23}$$

$$\nabla_\omega \bar{Loss} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\omega))\frac{\partial Q(s_i, a_i|\omega)}{\partial \omega} \tag{24}$$

The weight, ω, is updated by

$$\omega_{t+1} = \omega_t + \alpha \nabla_w Loss \tag{25}$$

where α is the learning rate.

2) Actor Function: The critic function is now used to update the actor function by replacing $\mu(s_t)$ with a neural network, $\mu(s_t|\theta)$. By substituting $a_t = \mu(s_t|\theta)$ into $Q(s_t, a_t|\omega)$ the result becomes,

$$J = Q(s_t, \mu(s_t|\theta)|\omega) \tag{26}$$

The differentiation of J yields,

$$\nabla_\theta J = \frac{\partial Q(s_t, \mu(s_t|\theta)|\omega)}{\partial a_t}\frac{\partial \mu(s_t|\theta)}{\partial \theta} \tag{27}$$

The algorithm applies the stochastic optimizer **Adam** [14] to update the network weights iteratively as follows,

$$m_{t+1} = \wp \times m_t + (1 - \wp)\nabla_\theta \mu \tag{28}$$

$$\Im_{t+1} = \beta \times \Im_t + (1 - \beta)\nabla_\theta \mu \tag{29}$$

$$\hat{m}_t = \frac{m_t}{1 - \wp^t} \tag{30}$$

$$\hat{\Im}_t = \frac{\Im_t}{1 - \beta^t} \tag{31}$$

$$\theta_{t+1} = \theta_t - \eta \frac{1}{\sqrt{\hat{\Im}_t}}\hat{m}_t \tag{32}$$

Here, $\wp$ and β are the Adam learning rates.

IV. Results and Discussion

A. Training results

The training utilizes a simulated environment, which is safer and can be completed at a faster rate than in a real-life environment. Two parameters are measured; the total reward over the last 100 episodes and number of steps. A step denotes one computer step, and an episode denotes one training interval, which ends when the number of steps reaches 1000 or the agent reach the terminal state. The terminal state indicates that the agent has sufficiently done station keeping. The agent trained for approximately 600 episodes. The left figures in Fig. 5 shows the result from the first 400 episodes. ROS and Gazebo experienced difficulties when the number of episodes increased. Therefore, the training simulator restarted after 400 episodes. The total reward converges towards 450 as the agent reaches approximately 600 episodes. The right figures in Fig. 5 shows the results where the last 50 episodes remain approximately 450 in the returned reward. From the bottom right figure, the agent uses approximately 203 steps

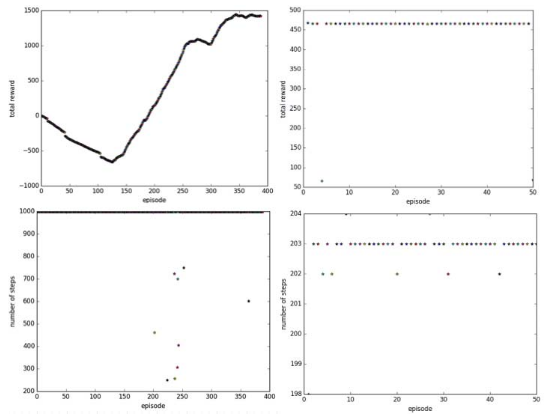


Figure 5: Training results: left figures: 0 < episode < 400, right figures: 550 < episode < 600.

in each episode, meaning that it has sufficiently satisfied the station keeping criteria in each episode.

B. Validation results

The validation of the training involves both simulation and real-life testing on the actual vehicle.

1) Simulation: Fig. 6 visualizes the body-fixed error for each DOF. The agent received an initial desired pose of $[x_d, y_d, z_d, \phi_d, \theta_d, \psi_d] = [2, 0, 0, 0, 0, 0]$, and at $t = 350$ a new desired pose was given as $[x_d, y_d, z_d, \phi_d, \theta_d, \psi_d] = [2, 2, 0, 0, 0, 0]$. The desired pose change was done to evaluate how the agent responds to change in pose, as well as to validate that the solution was universal in the state space. As shown in

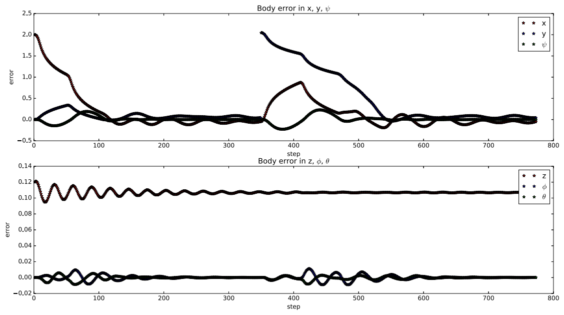


Figure 6: Simulated validation results showing convergence of the ROV to the desired states.

Fig. 6, the agent has sufficiently accomplished station keeping capabilities, with error values in the order of $10^{-2}m$.

2) Real-life experiments: The real-life experiments were conducted on the actual BlueROV2 in the MCLab at NTNU, Trondheim. The experiments consisted of three separate tests; two with different desired pose definitions, as well as a DP 4-corner test.

Fig. 7 shows the results of the first experiment. The initial error state is defined as $[x_e, y_e]$ = [0.3, 0.3] and at $t = 15$ seconds a new error state is given to x, such that, $[x_e^{t=15}, y_e^{t=15}]$ = [-0.5, 0]. The control objective is initially defined as this to evaluate the performance by only changing one error state. Fig. 8 shows the results of the second experiment, where

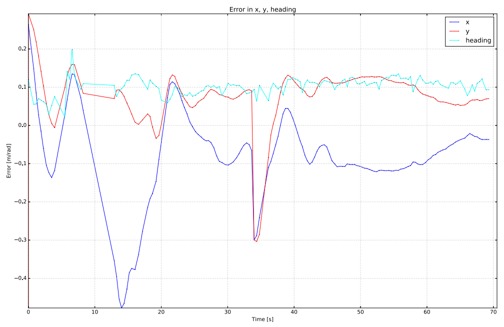


Figure 7: Real-life validation: Experiment 1, showing the error states for $[x, y, \psi]$.

the initial error state is defined as $[x_e, y_e]$ = [0, 0] and at $t = 18$ seconds a new error state is given as $[x_e^{t=18}, y_e^{t=18}]$ = [0.5, 0.3]. The control objective is defined as this such that a change in both error states at the same time can be evaluated. Fig. 9 shows the results of the final experiment,

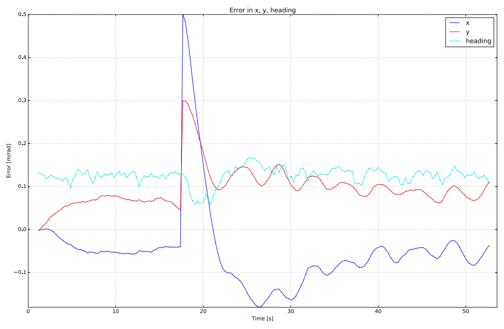


Figure 8: Real-life validation: Experiment 2, showing the error states for $[x, y, \psi]$.

which consisted of a a DP 4-corner test. The DP 4-corner test is a benchmark test in validation of dynamic positioning (DP) systems. The test defines the following four error states; 1) $[x_e, y_e]$ = [0, 0] at $t = 0$s, 2) $[x_e, y_e]$ = [0.5, 0] at $t = 15$s, 3) $[x_e, y_e]$ = [0, 0.5] at $t = 38s$s, 4) $[x_e, y_e]$ = [-0.5, 0] at $t = 70$s, 5) $[x_e, y_e]$ = [0, -0.5] at $t = 90$s. From the results

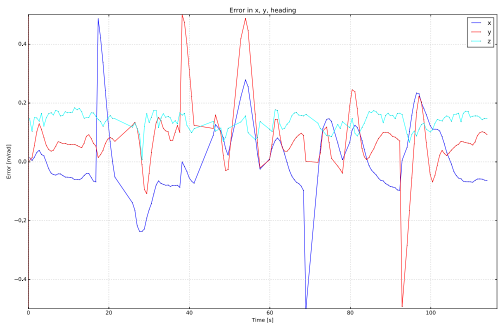


Figure 9: Real-life validation: Experiment 3, showing the error states for $[x, y, \psi]$.

in Fig. 7, 8 and 9, the agent is sufficient in making sure that $[x_e, y_e] \longrightarrow [0, 0]$ when receiving a new error state, but the performance is slightly worse than simulation, especially when performing the DP 4-corner test. The reason for this discrepancy between the simulated and real-life trials are two-fold. Firstly, the dynamic model used in training does not exactly match the actual vehicle. Secondly, the flaws related to the real-life pose measurement equipment have a large impact on performance.

The real-life pose measurement equipment was not robust, resulting in the agent having access to its pose in a very limited area, which makes the DP 4-corner test particularly challenging. Unfortunately, these flaws made it very difficult to test the ability of the DDPG based controller design to its full extent.

V. Conclusions

This article applied a deep reinforcement learning algorithm to the station keeping problem for underwater vehicles. The results revealed that the DDPG algorithm was capable of conducting station-keeping for an AUV. Both simulation and real-life experiments validated the presented results. In the simulated environment station keeping capabilities achieved an error in the order of $10^{-2}m$, and likewise $10^{-1}m$ in real-life. The discrepancy between the real and simulated case is two-fold. Firstly, the dynamics of the simulated vehicle did not match the actual vehicle exactly. Secondly, the pose measurement system in the laboratory often lost tracking, which was detrimental to the results.

A. Future Work

The agent was successful in performing station keeping of the BlueROV2 through the use of a DDPG algorithm, which controlled the x and y states. However, the algorithm had difficulties in learning to control heading ψ as well. Furthermore, the real-life experiments exposed weaknesses compared to the simulated results.

To resolve this, further work should address convergence issues, which is a large part of why controlling ψ is difficult. DDPG algorithms and policy gradient methods, in general, suffers from the fact that it samples the whole trajectory for only one policy update, and that the policy cannot update at every time step. This design is not sample efficient, resulting in large training sessions needed to reach convergence.

Further work should also try to resolve the robustness of the real-life sensor measurements. One way to do this could be to include an observer into the controller architecture, such that unmeasured states can be estimated.

References

[1] J. S. Riedel and A. J. Healey, “Shallow water station keeping of AUVs using multi-sensor fusion for wave disturbance prediction and compensation”, *Oceans Conference Record (IEEE)*, vol. 2, pp. 1064–1068, 1998.

[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 Chinese Control Conference (CCC)*, IEEE, 2017, pp. 4958–4965.

[3] D. A. Smallwood and L. L. Whitcomb, “Model-Based Dynamic Positioning of Underwater Robotic Vehicles: Theory and Experiment”, *IEEE Journal of Oceanic Engineering*, vol. 29, no. 1, pp. 169–186, 2004.

[4] F. A. Azis, M. S. Aras, M. Z. Rashid, M. N. Othman, and S. S. Abdullah, “Problem identification for Underwater Remotely Operated Vehicle (ROV): A case study”, *Procedia Engineering*, vol. 41, no. Iris, pp. 554–560, 2012.

[5] W. M. Bessa, M. S. Dutra, and E. Kreuzer, “Dynamic positioning of underwater robotic vehicles with thruster dynamics compensation”, *International Journal of Advanced Robotic Systems*, vol. 10, 2013.

[6] O. Hassanein, S. G. Anavatti, H. Shim, and T. Ray, “Model-based adaptive control system for autonomous underwater vehicles”, *Ocean Engineering*, vol. 127, no. June 2015, pp. 58–69, 2016.

[7] S. Moe, A. M. Rustad, and K. G. Hanssen, “Machine Learning in Control Systems: An Overview of the State of the Art”, in, ser. Lecture Notes in Computer Science, M. Bramer and M. Petridis, Eds., vol. 11311, Cham: Springer International Publishing, 2018, pp. 250–265.

[8] M. De Paula and G. G. Acosta, “Trajectory tracking algorithm for autonomous vehicles using adaptive reinforcement learning”, *OCEANS 2015 - MTS/IEEE Washington*, pp. 1–8, 2016.

[9] 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”, 2015. arXiv: 1509.02971.

[10] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms”, *31st International Conference on Machine Learning, ICML 2014*, vol. 1, pp. 605–619, 2014.

[11] M. M. M. Manhaes, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, “UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation”, in *OCEANS 2016 MTS/IEEE Monterey*, IEEE, 2016, pp. 1–8.

[12] M. C. Nielsen, O. A. Eidsvik, M. Blanke, and I. Schjølberg, “Constrained multi-body dynamics for modular underwater robots — Theory and experiments”, *Ocean Engineering*, vol. 149, no. February 2018, pp. 358–372, 2018.

[13] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*, 1st. Trondheim: Wiley and son, 2011.

[14] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization”, pp. 1–15, 2014. arXiv: 1412.6980.