Underwater Robotic Arm Control using Deep Reinforcement Learning

X.Xu[a], **S.Travadel**,[b]

a. Mines Paris PSL - xindong.xu@etu.minesparis.psl.eu
b. MINES Paris PSL, Centre for Research on Risks and Crises - sebastien.travadel@minesparis.psl.eu

Keywords : Robot Arm Control, Reinforcement Learning, Deep Learning, Deep Q Networks (DQN), Real world RL training;

Abstract : *This paper proposes a deep reinforcement learning-based controller for a flexible underwater robotic arm. Due to the complex and harsh underwater environment, traditional robot control methods face technical challenges such as high pressure, reduced visibility, and limited communication. To address these challenges, we apply deep reinforcement learning to train the robotic arm to make optimal decisions based on its environment without constant human intervention. Our experiments demonstrate that our approach can achieve an accuracy rate of* 82.8% *in reaching operator-specified locations. However, our research still faces some limitations such as imprecise position detection and mechanical instability. Future work will focus on addressing these limitations and extending our approach to more complex underwater operations.*

1 Introduction

The control of underwater robots is an evolving field that presents many technical challenges due to the complex and harsh underwater environment [1]. The underwater environment is subject to extreme conditions such as high pressure, reduced visibility, and unpredictable weather conditions, which make the control of these robots very difficult. In addition, communication between the underwater robot and the operator is often limited due to distance and signal attenuation, making it difficult for data and control commands to be transmitted in real time, thus complicating the control task.

Furthermore, in our experiments, the arm is a flexible, low-cost robot that is constructed from 3D printed parts and driven by two servo motors, making it more difficult to meet the high demands in terms of control accuracy and the requirement to model it in a simulation.

Traditional robot control systems and modelling methods are well established, in which robot arm control has been solved by hand-crafting solutions in a modular fashion, for example, 3D reconstruction, scene segmentation, object recognition, object pose estimation, robot pose estimation, and finally trajectory planning [2]. However, with the above two realistic factors, it is difficult to have satisfactory results.

In this case, our task is to develop a control algorithm for an underwater robotic arm that overcomes these complex challenges by using reinforcement learning techniques to guide the end of the arm to an operator-specified location. Using this approach, we hope to train the robotic arm to make optimal decisions in real time based on its environment, without the need for constant human intervention [3]. This could lead to greater efficiency and accuracy in underwater and complex operations.

2 Deep Reinforcement Learning

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a sub field of machine learning that deals with the problem of learning how to make decisions in an environment [3]. It is a type of learning in which an agent learns

to interact with an environment by taking actions and receiving feedback in the form of rewards or punishments. The goal of the agent is to learn a policy that maximizes its cumulative reward over time. This interaction is commonly depicted as the feedback structure in the following figure (Fig.1).

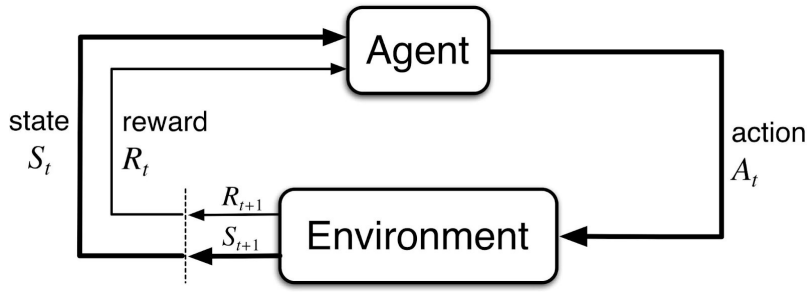

Figure 1: *Agent–environment interaction in reinforcement learning*

In RL, the agent interacts with an environment in discrete time steps. At each time step, the agent observes the current state of the environment and selects an action to take. The action then leads to a new state and a reward signal from the environment, which the agent uses to update its policy. The agent's policy is a mapping from states to actions, and its goal is to learn a policy that maximizes the expected cumulative reward.

One of the most popular RL algorithms is Q-learning, which is a model-free, off-policy, value-based algorithm that learns the optimal action-value function (also called the Q-function) for a given policy [3]. The Q-function estimates the expected cumulative reward of taking an action in a given state and following the optimal policy thereafter.

The Q-learning algorithm updates the Q-values based on the Bellman equation, which expresses the relationship between the Q-values of a state-action pair and the Q-values of its successor states and actions [4]. The Bellman equation (Eq.1) for the optimal Q-function is given by:

$$Q^*(s,a) = \mathbb{E}[r + \gamma max_{a'} Q^*(s',a')|(s,a)] \tag{1}$$

where $Q^*(s,a)$ is the optimal Q-value of state-action pair (s,a), and the expectation is taken over all possible next states s' and the probability of transitioning to them. The optimal policy can be derived from the optimal Q-function by selecting the action with the highest Q-value in each state.

Q-learning iteratively updates the Q-values based on the Bellman equation using the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \tag{2}$$

where α is the learning rate that determines the step size of the update. The update rule aims to make the Q-value of the current state-action pair closer to the expected value of the Bellman equation. By iteratively updating the Q-values, Q-learning converges to the optimal action-value function, which can be used to derive the optimal policy.

While Q-learning is a powerful algorithm for solving RL problems, it has some limitations. One limitation is that it assumes a discrete and finite state and action space, which may not be the case in real-world problems. In addition, Q-learning is known to suffer from the "curse of dimensionality," meaning that its computational complexity grows exponentially with the number of state variables.

To overcome these limitations, researchers have developed variants of Q-learning, such as Deep Q-learning (DQL), which uses deep neural networks to represent the Q-values and can handle high-dimensional state spaces [5]. DQL has been successfully applied to a variety of problems, including game playing, robotics, and control. We will describe DQL in more detail in the next section.

2.2 Deep Q Learning

Deep Q Learning (DQL) is a variant of Q-learning that combines RL with deep neural networks to approximate the Q-values of a large state space. DQL was introduced in the seminal paper by Mnih et al. [6], which demonstrated the successful application of DQL to learn to play Atari games from raw pixels.

In Deep Q Learning, a deep neural network is used to represent the Q-function, which takes as input the state of the environment and outputs the Q-value for each action. The network is trained to minimize the difference between the predicted Q-values and the target Q-values, which are computed using the equation:

$$y_i = r_i + \gamma Q(s_{i+1}, argmax_a Q(s_{i+1}, a; \theta_i); \hat{\theta}) \tag{3}$$

where y_i is the target Q-value for the i-th transition, r_i is the immediate reward, s_{i+1} is the next state, γ is the discount factor, and $\hat{\theta}$ are the weights of a separate target network that is updated less frequently than the online network, where the parameters θ_i are updated by the following way :

$$\mathcal{L}_i = \frac{1}{2}(y_i - Q(s_i, a_i; \theta_i))^2, \tag{4}$$

$$\theta_{i+1} = \theta_i - \alpha \frac{\partial \mathcal{L}_i}{\partial \theta_i} = \theta_i - \alpha(y_i - Q(s_i, a_i; \theta_i)) \frac{\partial Q(s_i, a_i; \theta_i)}{\partial \theta_i} \tag{5}$$

In our research, the Deep Q Learning algorithm also uses experience replay to decorrelate the data and stabilize the training. During each episode, the agent stores the transitions (s_i, a_i, r_i, s_{i+1}) in a replay buffer, and randomly samples a mini-batch of transitions from the buffer to update the network weights. By using a random sample of transitions, Deep Q Learning can break the correlation between consecutive updates and learn from a diverse set of experiences. In addition, in our training of Deep Q Learning uses a separate target network to compute the target Q-values, which reduces the variance of the updates and stabilizes the training. The target network is a copy of the online network with fixed weights, which are updated periodically with the weights of the online network. By using a separate target network, DQL can stabilize the Q-value estimates and prevent oscillations or divergence.

Algorithm 1 Deep Q Learning with Experience Replay

Initialize replay memory D to capacity N
Initialize online network function Q with random weights θ
Initialize target network function $\hat{Q}$ with random weights $\hat{\theta} = \theta$
for $episode = 1 \rightarrow episode = M$ **do**
    Initialize sequence $s_1 = x_1$
    for $t = 1 \rightarrow t = T$ **do**
        With probability ϵ, select a random action a_t
        otherwise $a_t = argmax_a Q(s_t, a_t; \theta)$
        Execute action a_t to the two servo motors and observe reward r_t and environment state s_{t+1}
        Store information (s_t, a_t, r_t, s_{t+1}) in D
        Sample random mini-batch of sets (s_t, a_t, r_t, s_{t+1}) from D
        Set $y_j = \begin{cases} r_j, & \text{if episode terminates at step } j+1 \\ r_i + \gamma \hat{Q}(s_{i+1}, \text{argmax}\, aQ(si+1, a; \theta); \hat{\theta}), & \text{otherwise} \end{cases}$
        Perform a gradient descent step for on $\mathcal{L}_j = \frac{1}{2}(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameters θ
        Every C t time steps, reset $\hat{Q} = Q$
    end for
end for

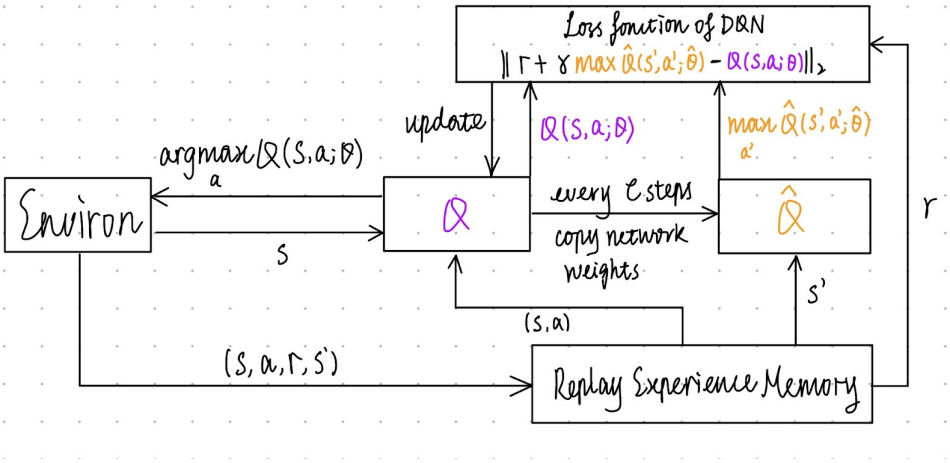

Figure 2: *Deep Q Learning training process*

In our experiment, as well as other application situation, Deep Q Learning has been successfully applied to a variety of RL problems, including game playing, robotics, and control [7]. However, DQL has some limitations, such as the tendency to overestimate the Q-values in the presence of function approximation and the sensitivity to hyper parameters such as learning rate and exploration rate, thus, we need to continue studying them in the following section by finishing much work of experiment.

2.3 Preprocessing and Model Architecture

Preprocessing the input data is an important step in deep reinforcement learning. In our experiments, we preprocess the images captured by the camera mounted on the underwater robotic arm using several techniques to improve the performance of the deep neural network.

In our experiments, we used a low-cost, flexible underwater robotic arm constructed from 3D printed parts and driven by two servo motors. The task of the robotic arm was to move its end to a specified location, as indicated by a green toothpick which is marked at the end of the arm in the camera view (Fig.3).

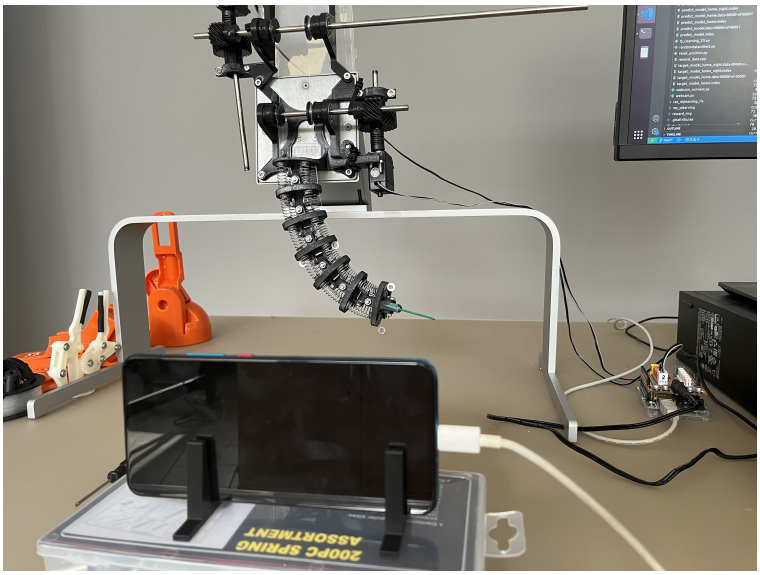

Figure 3: *Robotic arm position capture with a web camera*

To achieve this, we employed a preprocessing step to detect the marked section of the arm in the camera view. Specifically, we used OpenCV in Python to perform color detection by converting the image captured by the camera from RGB to the HSV color space. This allowed us to better target a particular range of colors by selecting an interval of hue, saturation, and value. A mask was then created using the CV2.inRange function with the help of CV2 python library, which returned a binary image with only the pixels that fell within the selected interval. Finally, we applied the RANSAC (Random Sample Consensus) algorithm from python scikit-learn library to perform straight line detection in the green part of the image. Thus, we can point out precisely the position of the end of the green toothpick. These positions were then used as the environment state input for the reinforcement learning algorithm.

For the model architecture, we used a deep neural network with three hidden layers and the hyperbolic tangent (tanh) activation function. The input layer consisted of the state space of the robotic arm, and the output layer consisted of the Q-values for each possible action. We used the Double DQL algorithm [8] to train the network, as described in the previous section.

Layer (type)	Output shape	Parameters	Activation
Batch normalization layer 1	(batch-size, 6, 1)	4	-
Dense layer 1	(batch-size, 6, 6)	12	tanh
Batch normalization layer 2	(batch-size, 6, 6)	24	-
Dense layer 2	(batch-size, 6, 4)	28	tanh
Batch normalization layer 3	(batch-size, 6, 4)	16	-
Dense layer 3	(batch-size, 6, 1)	5	-

Table 1: Summary of the architecture

The architecture used in our experiments is based on the popular Deep Q-Network (DQN) architecture proposed by Mnih et al. [6]. However, we made some modifications to the original architecture to adapt it to our task and reduce the computational cost. We use a deep fully connected network (FCN) architecture instead of the origin convolutional neuron network (CNN) to process the preprocessed images and estimate the Q-values, because we chose not to use the camera captured image as input of the network, but more directly and efficiently, the position of some key points on the image.

The network consists of three dense layers with batch normalization layers in between. The first dense layer has 6 units and uses the hyperbolic tangent (tanh) activation function. The second dense layer has 4 units and also uses the tanh activation function. The final dense layer has 1 unit and no activation function. The number of units and the activation function were chosen to strike a balance between expressiveness and over-fitting. The use of batch normalization layers helps to stabilize and speed up the training process. The output of the neural network is the optimal action to take, which is then translated into the rotation angle values of the servo motors to control the robotic arm.

Overall, the preprocessing and model architecture were designed to overcome the challenges posed by the complex and harsh underwater environment and the flexibility of the robotic arm. By using color detection and RANSAC to detect the marked section of the arm and a deep neural network with Double DQL to learn the optimal control policy, we were able to achieve greater efficiency and accuracy in underwater and complex operations. In the next section, we will describe our experimental method and details, where we will explain deeply the choice of the input and output of the network, especially their dimension and encoding techniques, we will evaluate the performance of the underwater robotic arm control algorithm based on deep reinforcement learning.

3 Training and Experiment

3.1 Experimental Method

To evaluate the effectiveness of our proposed deep reinforcement learning algorithm for controlling the movement of an underwater robotic arm, we designed and constructed a highly flexible underwater robotic arm using springs and 3D printed parts (Fig.4). The robotic arm is controlled by two servo

motors that rotate to adjust the position of the end of the arm. Actually, we designed a simple control system to send the rotation angle values to the servo motors. The control system consists of an Arduino micro-controller and a motor driver board. The micro-controller reads the rotation angle values from the computer and sends them to the motor driver board, which in turn drives the servo motors to rotate to the desired positions.

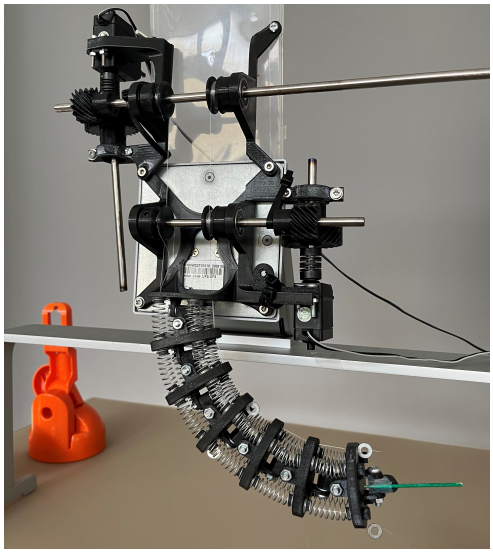

Figure 4: *2D moving robotic arm in our experiment*

In our experiments, we focused on controlling the 2D movement of the robotic arm in a horizontal plane. To extract the position of the end of the robotic arm, we marked a section of the robotic arm with a green toothpick, which can be captured by a camera. By inputting the rotation angle values of the servo motors, we can obtain all potentially reachable positions of the end of the robotic arm (Fig.5).

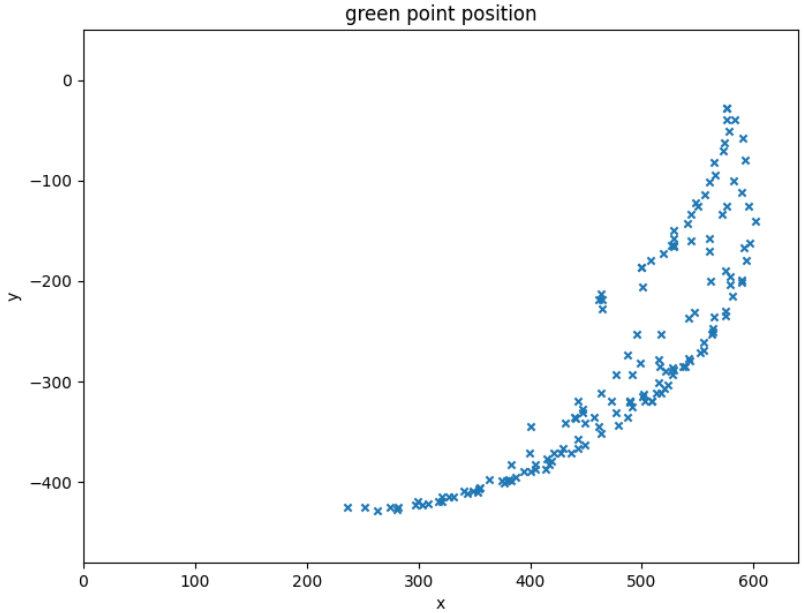

Figure 5: *Potentially attainable positions of the robotic arm end*

These positions were then used as the state space for the reinforcement learning algorithm, thus, the

input of the Deep Reinforcement Learning algorithm consists of the two-dimensional coordinate positions of the target point and the current end point and the optimal current action based on the given situation, which contains two values, respectively for the two servo motors. Taking into account the response time of the servo motors and the fact that the control algorithm runs at each small step interval, we have coded the command actions input to the servo motors as 10 forward, 10 backward and hold still, making a total of nine input command possibilities for the two motors. To avoid the monotonicity of the distribution of this parameter affecting the prediction of the Q-value, we coded the action input to the algorithm as a_1 and a_2, with $a_1 \in [-10, 0, 10]$ and $a_2 \in [-10, 0, 10]$.

To evaluate the performance of the robotic arm in different scenarios, we set up various target positions in the environment and tested the robotic arm's ability to reach these targets. We measured the success rate of the robotic arm in reaching the target position within a certain tolerance, as well as the time it took to reach the target position. We also evaluated the robustness of the control system by testing it in different water conditions, such as varying water currents and depths.

3.2 Training and Stability

To train the deep reinforcement learning algorithm for controlling the underwater robotic arm, we used a Double Deep Q Learning algorithm, which is a variant of the Q-learning algorithm. We implemented the Double DQL algorithm using the Keras and TensorFlow deep learning library in Python, whose source code is fully available in this GitHub repository.

This Double DQL algorithm can improve the accuracy of the Q-value estimates and stabilize the training [8]. During the training process, we used an experience replay buffer to store the past experiences of the robot and randomly sampled mini-batches from this buffer to train the network. We also used two separate networks, an online network and a target network, to estimate the Q-values and compute the target Q-values, respectively. We updated the target network less frequently than the online network to prevent over-fitting and improve the stability of the training (Fig.1).

To prevent over-fitting and improve the generalization of the learned policy, we also applied batch normalization to the hidden layers of the neural network to stabilize the learning process and improve the convergence speed. We used a mean squared error loss function to train the neural network, and the Adam optimizer with a learning rate of 0.001.

During the training process, we set the exploration rate ϵ to decay exponentially over time to balance between exploration and exploitation. The discount factor γ was set to 0.99, and the replay buffer size was set to 1,000. We trained the network for a total of 50 episodes, with each episode consisting of up to 100 time steps.

To ensure the stability of the training process, we monitored several metrics, including the average reward per episode (Fig.6) and the loss of the neural network (Fig.7). As shown in the following figures, we can clearly find out that with the process of our training going on, the action deciding agent performed increasingly well. The average reward rises gradually during the trials, where reward is defined in our study as the negative of the Euclidean distance from the current vertex position of the robot arm to the target position. As for the loss of the neural network, we did not directly select the variation of the loss function as an observation of the convergence of the algorithm, because as the deep neural network of Q-value approaches optimality, the average value of the output of the network becomes larger and larger, which is of course directly determined by our choice of gamma (0.99). Therefore, we have chosen the neuron network loss function over the average Q-value output as a metric observation, and in the Fig.7 we clearly see that this measure eventually converges gradually as the training episodes increase. To sum up, our Deep Reinforcement Learning algorithm training yields a relatively good and stable model.

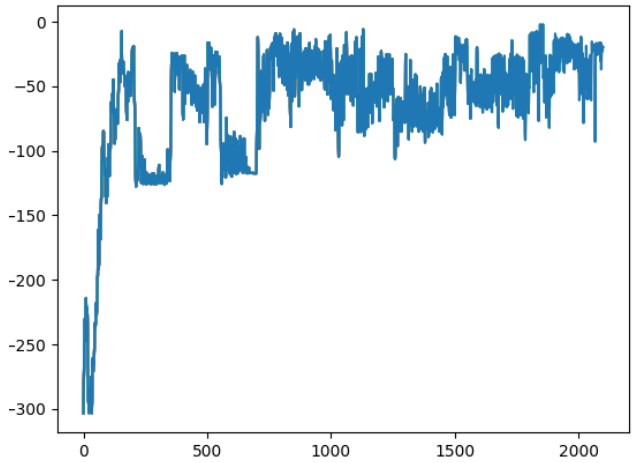

Figure 6: *Evolution of average reward per training episode*

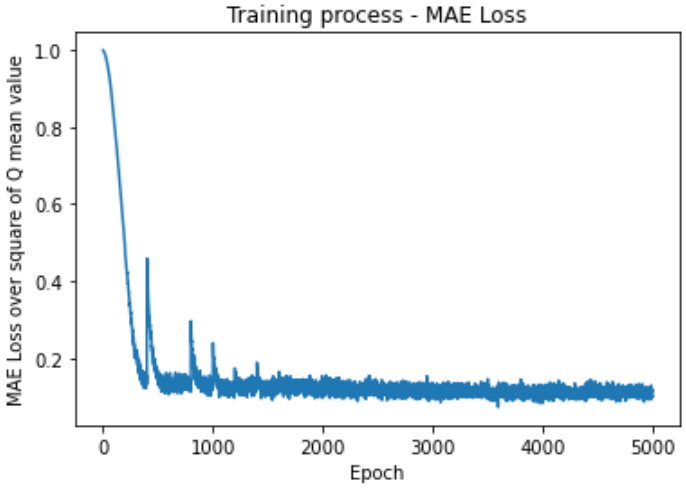

Figure 7: *Evolution of loss function over the average Q-value per training episode*

Furthermore, we also evaluated the performance of the learned policy periodically by testing the robotic arm in various environments with different target positions, which will be mostly presented and introduced in the next section.

3.3 Implementation and Evaluation

To evaluate the performance of the learned policy, we tested the robotic arm in various scenarios with different target positions. We measured the success rate of the robotic arm in reaching the target position within a certain tolerance in which a distance difference between the current position and the target point is smaller than 10 pixel will be seen as an accomplished mission, as well as the time it took to reach the target position.

Fig.8 shows the evolution of the reward over time for one of the successful trials where the robotic arm was able to reach the target position. We can observe that under the control of the reinforcement learning algorithm, the reward decreases over time, indicating that the algorithm is learning to control the robotic arm more effectively.

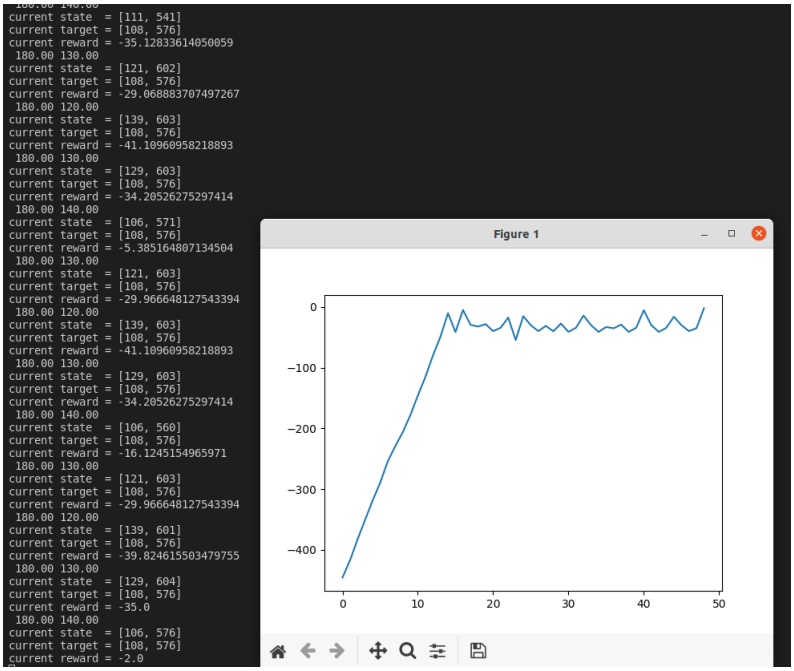

Figure 8: *Evolution of the reward over time for one of the successful trials*

We also randomly sampled 20 target positions and let the neural network control the robotic arm to move towards the specified position. Fig.9 shows the reward achieved in each of these trials. We can observe that in most cases, the algorithm was able to accomplish the control task successfully.

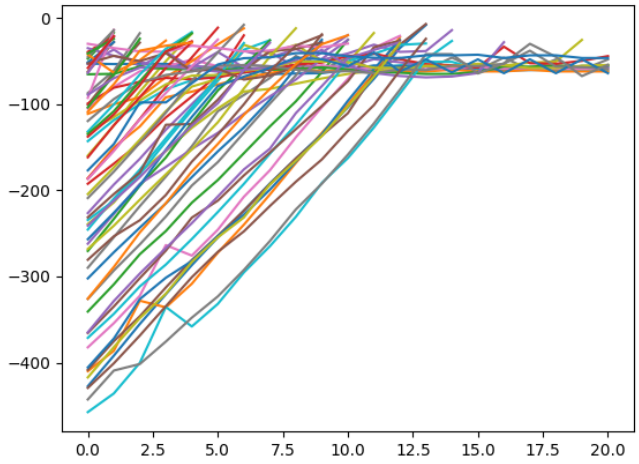

Figure 9: *Evolution of the reward over time for 20 successful trials*

To further evaluate the performance of the learned policy, we let the algorithm search for all possible target positions that we obtained at the beginning of the training process. We defined a final reward of less than 15 pixels as a successful task completion. We found that our current model can achieve an accuracy of 82.8% in reaching the target positions.

Overall, these results demonstrate the effectiveness of our proposed deep reinforcement learning algorithm for controlling the movement of an underwater robotic arm. The algorithm is able to learn an effective control policy from raw sensory inputs and can generalize well to new and unseen scenarios. The success rate achieved by the algorithm shows that it has the potential to be applied to more complex and practical underwater operations in the future.

4 Discussion and Conclusion

4.1 Conclusion

In this study, we developed a 2D controller based on deep reinforcement learning for a flexible robotic arm that guides the end of the arm to a position specified by the operator at the beginning of each trial.

Our current model can achieve an accuracy of 82.8% in reaching the target positions, demonstrating the effectiveness of our proposed approach.

This control method, after training the neural network, can spontaneously achieve the task goals without requiring human intervention, and the computation time between each control decision is very short, making it an efficient and effective solution for controlling underwater robotic arms in complex situations.

In the future, we plan to extend our approach to 3D motion control and apply it to more complex and practical underwater operations. We believe that our proposed deep reinforcement learning algorithm has the potential to significantly improve the efficiency and accuracy of underwater operations, ultimately leading to new and exciting applications in various domains such as ocean exploration, deep-sea mining, and underwater construction.

4.2 Limits and Prospects

Our research still faces some challenges and limitations.

One limitation is the imprecise detection of the position. In each frame of the OpenCV video, the green toothpick used to mark the position is not a straight line but rather a cylinder, and there is always an error of a few pixels in identifying the tip of the toothpick.

In addition, motor control is not precise, and this mechanical structure is not stable enough. For example, for the same input of motor rotation angle, the result of the position detection algorithm is not necessarily the same every time. After each execution of the motor command, the robot arm undergoes considerable oscillation due to inertia, which can further increase the error of the OpenCV module.

Finally, we plan to apply the current deep reinforcement learning algorithm for 2D motion control to 3D motion control of the robotic arm in the future. We also need to introduce a second camera to finally achieve our control goals.

Despite these limitations, we believe that our proposed approach has great potential for improving the control of underwater robotic arms. Future work will focus on addressing these limitations and extending our approach to more complex and practical underwater operations. Ultimately, we hope that our research can lead to new and exciting applications in various domains such as ocean exploration, deep-sea mining, and underwater construction.

4.3 Acknowledgements

In closing, I would like to express my gratitude to several individuals who have provided invaluable assistance in the development of this research paper. First and foremost, I extend my heartfelt appreciation to my supervisors, Sébastien Travadel, Philippe Blanc and Samuel Olampi, all professors at Mines Paris, for their unwavering support, guidance, and expertise throughout the entire process. Their encouragement and feedback have been instrumental in the successful completion of this project.

I would also like to thank my colleagues and classmates who have contributed to this research with their insightful feedback and constructive criticism. Their input has been invaluable in shaping the direction of this work.

Together, these individuals have been instrumental in the success of this research, and I am grateful for their contributions.

References

[1] Side Zhao and J. Yuh. "Experimental study on advanced underwater robot control". In: *IEEE Transactions on Robotics* 21.4 (2005), pp. 695–703. DOI: 10.1109/TRO.2005.844682.

[2] Stephen James and Edward Johns. *3D Simulation for Robot Arm Control with Deep Q-Learning*. 2016. arXiv: 1609.03759 [cs.RO].

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[4] Shirin Joshi, Sulabh Kumra, and Ferat Sahin. *Robotic Grasping using Deep Reinforcement Learning*. 2020. arXiv: 2007.04499 [cs.RO].

[5] MD Muhaimin Rahman, SM Hasanur Rashid, and M. M Hossain. *Implementation of Q Learning and Deep Q Network For Controlling a Self Balancing Robot Model*. 2018. arXiv: 1807.08272 [cs.RO].

[6] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[7] A. Rupam Mahmood et al. *Setting up a Reinforcement Learning Task with a Real-World Robot*. 2018. arXiv: 1803.07067 [cs.LG].

[8] H. V. Hasselt. "Double Q-learning". In: *NIPS*. 2010.