AUV Position Tracking Control Using End-to-End Deep Reinforcement Learning

Ignacio Carlucho[a,b], Mariano De Paula[a], Sen Wang[b], Bruno V. Menna[a], Yvan R. Petillot[b], Gerardo G. Acosta[a]
ignacio.carlucho@fio.unicen.edu.ar, mariano.depaula@fio.unicen.edu.ar, s.wang@hw.ac.uk,
brunovmenna@fio.unicen.edu.ar, y.r.petillot@hw.ac.uk, ggacosta@fio.unicen.edu.ar
[a] INTELYMEC Group, Centro de Investigaciones en Física e Ingeniería del Centro
CIFICEN – UNICEN – CICpBA – CONICET, 7400 Olavarría, Argentina
[b] School of Engineering & Physical Sciences Heriot-Watt University, EH14 4AS, Edinburgh, UK

***Abstract*—In this article we consider the navigation problem for an autonomous underwater vehicle (AUV) for reaching a desired way-point. The navigation problem in underwater vehicles presents major problems, the highly coupled dynamics of the vehicles and the unknown parameters of the dynamic model, make the need for complex control architectures. However, current developments in reinforcement learning show promising results for robotics applications. In particular underwater autonomous vehicles could benefit from this new techniques, achieving adaptive behavior for real-time problem solving. Based on this developments the navigation problem is solved using deep reinforcement learning, in particular the deep deterministic policy gradient. In this proposal a model free approach is used, where the raw sensor information is used as inputs to a policy network, and the outputs of this network are directly mapped to the thrusters. In addition an adaptive goal driven architecture is used to allow the agent to reach variable way points consistently. The obtained simulated results show its capacity for successfully solving AUV navigation problems.**

***Index Terms*—reinforcement learning, deep reinforcement learning, AUV, navigation**

I. Introduction

Autonomous underwater vehicles (AUVs) have a prominent role in oceanic operations, such as in the oil industry, marine geoscience, biology, archeology and others. The autonomy provided allows the completion of multiple tasks without human involvement. However, the required control techniques to provide enough degree of autonomy are still an active area of research due to the non linearities of the AUVs as well as the variable and uncertain environmental conditions that are typical of oceanic environments. Many classical control architectures were developed to provide mission autonomy, as in [1] [2]. However, these proposal need a dynamic model of the vehicle, which often becomes in a cumbersome task. For this reason, researchers have been focusing on artificial intelligence techniques to develop autonomous control strategies.

Within the artificial intelligence field a branch with growing importance is the Reinforcement Learning (RL) paradigm. In this unsupervised learning framework, the agent learns an optimal control policy by its direct interaction with the environment [3]. This type of learning from experience mimics a common process in nature. Moreover there is currently high evidence that indeed reinforcement learning happens in mammalian brains [4].

Further developments in the RL field were recently obtained by utilizing deep neural networks as function approximators of the policy function. In [5], the deep Q-Network (DQN) algorithm was developed, where a deep neural network was used to parameterize the state action-value function of the well-known Q-learning algorithm. However, the DQN algorithm can only be applied to discrete problems, where the state and action spaces are discrete and finite. To solve this issue the Deep Deterministic Policy Gradient (DDPG) was proposed in [6] to obtain a continuous control policy. On the other hand, end-to-end deep RL (DRL) approaches have been developed to solve complex manipulation tasks in grasping applications of robotics arms. In these cases, the robot decision system must deal with a complex perception system and continuous control actions in a high dimensional space [7].

In the past some attempts were made to employ RL techniques to the control of AUVs [8]–[10]. However the new developments in DRL could mean a new avenue for the improvement of real-time operation of underwater vehicles. In this article we developed a technique for end-to-end position control of the AUV by using the DDPG. In our formulation, an agent learns a control policy function to reach a dynamic goal by directly interacting with the environment.

The algorithmic formulation uses an actor-critic architecture with a deep neural network to represent the policy function. In order to do so, the state of the system is determined by the low-level continuous data obtained from the sensory input. In the same way, the outputs of the policy are continuous control signals that can be directly mapped to the thrusters of the AUV. In addition, an adaptive goal driven architecture is used, that allows the agent to reach variable way points and thus successfully solving the navigation problem. The proposed algorithm was tested in simulation using the dynamic model of Nessie VII an AUV developed by the Heriot-Watt University, shown in Fig. 1. The simulated results show the feasibility of the proposal for different test cases, where the agent was able to successfully reach a different number of way-points. Moreover, the ability of the agent to adapt to unexpected scenarios was also tested, by performing a thruster

978-1-5386-4814-8/18/$31.00 ©2018 IEEE

fault simulation, showing the capabilities of the proposal.

The rest of the article is structured as follows. Section II presents a summary of related works. On section III a brief explanation of reinforcement learning is giving, follow by Section IV where the proposed algorithm is explained. In section V the obtained results are shown and finally Section VI presents some conclusions.

II. Related works

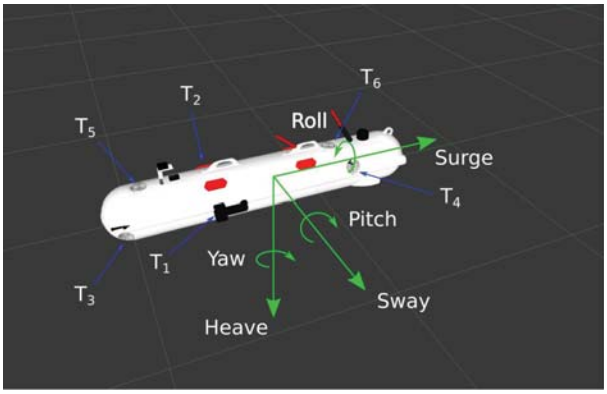
In [9], the authors used Q-learning, together with an interpolator to achieve continuous actions, to try to reach a position goal with an AUV. The approach controlled only 2DOF and was tested in a simulator. A path planning algorithm was developed in [11] to navigate under oceanic currents. A Q-learning algorithm was used for planning a local path following a navigation chart. In [12] model based policy search was used to train the agent to recover from thruster failures. The proposed approach was tested in simulation using the dynamic model of an AUV.

In [13] SARSA was compared with actor-critics methods for a pipe following task of an AUV. The analysis was made using a simulator of the AUV. Further developments were achieved in [8], where an AUV was able to successfully follow an underwater cable using the natural actor-critic algorithm. A two step learning approach was taken, meaning the algorithm was trained first in simulation and then continued in the real vehicle. In this problem the state space was obtained using a vision-based system, that gives as output a vector with 4 components, defining therefore the position of the cable with respect to the AUV. While the output space was a vector of two components, meaning the control algorithm only controls 2DOF. A variant of the actor-critic algorithm, namely the episodic natural actor-critic (ENAC), was used in [14] to generate an adaptive path planner for AUVs. The navigation module was implemented in ROS, controlling yaw and sway movements, while the state was a Tile Coded representation of the state of the AUV.

As previously detailed new developments in the RL field were recently obtained in [5] by utilizing deep neural network. Particularly, convolutional neural networks (CNN) were used, to directly extract the agent state from raw pixel data [15]. The stabilization of the policy was obtained by means of the experience replay and the introduction of a target network [16]. In this seminal contribution it was shown how an agent is able to learn a control policy directly from raw pixel data. Some improvements were proposed to the DQN algorithm such as in [17] were experience was prioritized, replaying important transitions more often, thus optimizing learning. In [18] proposed to use double Q-learning taking advantage of the target network as a natural replacement for the double Q-learning original proposal. By using two Q estimators the overestimation problem in Q-learning is solved, achieving higher results. However, this methods are not directly applicable to problems were the state-action space is too large or continuous.

Other methods are proposed to solve this shortcomings, such as the Deep Deterministic Policy Gradient (DDPG) [6].

(a) Nessie VII



(b) AUV reference system

Fig. 1. AUV platform used

This algorithm is based on the deterministic policy gradient [19] and uses the actor critic architecture, together with ideas of batch normalization and experience replay, to obtain a continuous control policy. The DDPG algorithm is thus able to solve a multiple number of problems using both CNN to extract the states from raw pixel data, as well as low level information from sensors.

This new developments show a promising avenue of future research for the development of control algorithms for AUVs. Indeed, DDPG algorithms have been already tested in AUVs for a different number of applications [20]–[22]. However, further research is still required for goal seeking problems.

III. Reinforcement Learning

In this control paradigm the agent is able to learn a control policy π by successive interactions with the environment. RL is a special case of a Markov Decision Process (MDP), therefore it can be represented by the tuple : S, A, P, r(.). Where S is the State Space, A the action space, P the state transition probability and r(.) the reward function. Therefore to solve the RL problem the agent must find the optimal control policy π^* that defines the optimal action u^*, to in turn maximize the total expected reward:

$$J^* = \max J_\pi = \max_\pi \mathbb{E}\{R_t | \boldsymbol{x}_t = \boldsymbol{x}\} \quad (1)$$

where J^* is the maximum expected reward, x is the robot state, R_t is the total return from a state defined as $R_t = \sum_{k=0}^{\infty} \gamma^k r_{k+t+1}$, and γ is a discount factor.

It is possible to classify RL methods in three large groups: actor-only, critic-only and actor-critic methods [23]. In this classification critic-only methods are those that use only a value function to obtain the policy. Therefore the policy is directly derived from the obtained state value function (V) or state-action value function ($Q(s, a)$). The disadvantage with these methods is that a discretization of the action space is usually needed (such as in the Q-learning algorithm).

On the other hand, actor-only methods are those that can obtain a policy directly, for instance by means of a neural network [24]. Having a explicit policy means that a discretization of the action space is not necessarily and a continuous policy function can be obtained. Actor methods, such as the policy gradient, are known to converge. However, this methods are prone to variance in the estimation of the gradient and tend to learn slower.

Finally actor-critic methods combine the advantages of the previous methods. By having a parameterized policy they can use continuous actions while the critic reduces the variance by giving information of the performance. Within these methods is the deterministic policy gradient, that will be explained in the following subsection

A. Deterministic policy gradient

The deterministic policy gradient utilizes a actor-critic architecture to solve the RL problem. In this case, the critic is the state-action function Q, defined as:

$$Q^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathbb{E}\{R_t | x_t, u_t\} = \mathbb{E}\{\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} | \boldsymbol{x}_t, \boldsymbol{u}_t\} \quad (2)$$

where $\boldsymbol{u}_t$ is the action taken at time t. The state-action function can be learned using transitions from state $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$. If the target policy is deterministic, Q can be learned off-policy, by following trajectories generated by different policy $\beta(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Then, the critic is updated as in the regular Q-learning algorithm.

$$Q^w(\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathbb{E}\{r_{x_t, u_t} + \gamma Q^w(\boldsymbol{x}_{t+1}, \boldsymbol{u}_{t+1})\} \quad (3)$$

where Q^w is a differentiable parameterized function, so that $Q^w \approx Q^\pi$

Now, if we consider an actor that parameterizes states directly into actions with parameters θ, thus $\mu(x_t|\theta)$. And we define a performance objective function $J(\mu_\theta) = \mathbb{E}\{r^\gamma | \mu\}$ and a probability distribution, then the performance as an expectation can be written as:

$$J(\mu_\theta) = \int \rho^\mu r(\boldsymbol{x}_t, \mu) dx = \mathbb{E}[r(\boldsymbol{x}_t, \mu_\theta(\boldsymbol{x}_t))] \quad (4)$$

and by applying the chain rule to the expected return, we can then write:

$$\nabla_\theta J = \int \rho^\mu \nabla_\theta \mu_\theta(x) \nabla_u Q^\mu(x, u) \quad (5)$$

$$\nabla_\theta J = \mathbb{E}[\nabla_\theta \mu_\theta(x) \nabla_u Q^\mu(x, u)] \quad (6)$$

in [19] was proven that this is the deterministic policy gradient and its convergence properties. In the following subsection we will describe with more detail the function aproximators that are used for representing both the actor an the critic.

B. Deep Neural Networks as function approximators

Previously, the utilization of neural networks as function approximators for the policy and value functions were unsuccessful. However, in recent developments it is common to see networks with many layers and different architectures. Constitutional neural networks have been use in multiple applications to solve classification as well as reinforcement learning problems. The use of rectifier linear activations units (ReLu) activation layers as well as new stochastic optimization algorithms, such as Adam [25], allowed researchers to train networks with bigger architectures.

In particular, for the RL problem, deep neural networks are used as function approximators for both the critic and actor, since it allows generalization over large state spaces. However, the training of the network cannot be applied directly during training since during exploration the data sets are correlated causing instabilities on the learning. So, during training phase small changes of the network weights could cause major changes to the policy and thus make it diverge from the optimal.

To address this issues two major solutions were proposed in [6], that is the use of a replay buffer and the utilization of a target network. The replay buffer is used to store all the transitions tuples of the form (x_t, u_t, r_t, x_{t+1}). Instead of using the last transitions, random samples are obtained from the replay buffer to train the agent in each step, thus reducing the correlation between samples. In addition, a copy of the actor and critic networks is created (μ' and Q'), which are called target networks. The weights of these target networks are updated as:

$$\theta' = \tau\theta + (1 - \tau)\theta' \quad (7)$$

where $\tau << 1$. It was demonstrated that using this updating strategy greatly improve the stability of the learning process.

C. Deep Deterministic Policy Gradient

With the use of deep neural networks together with the ideas of the replay buffer and target networks, it is possible to represent the actor and the critic of the DPG algorithm presented in the previous subsection.

If a Q function is used as critic, then to learn an optimal policy we need to obtain an optimal critic. If the neural network has parameters w then the optimal critic can be found by minimizing the loss function $L(\cdot)$, of the form:

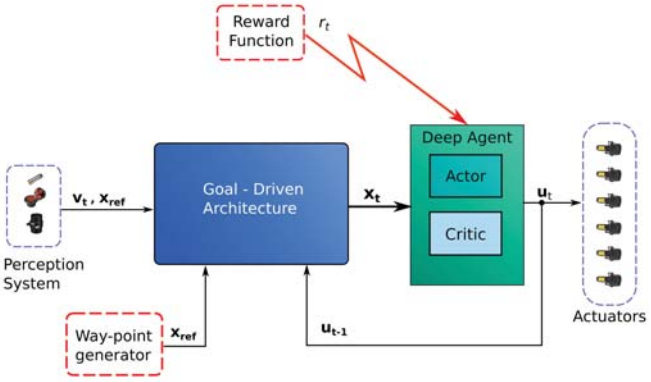


Fig. 2. Goal driven architecture

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q^w(x_i, u_i))^2 \tag{8}$$

where L is simple a mean squared error function, N represents a time horizon of N sampling times and y_i is the target state-action values obtained from the target deep neural network Q', such that:

$$y_i = r(x_i, u_i') - \gamma Q'^w(x_i, u_i) \tag{9}$$

where u_i' is given by the target actor network:

$$u_i' = \mu(x_i|\theta') \tag{10}$$

Then the gradient of the loss function $L(\cdot)$ is defined as:

$$\nabla_w L(w) = -\frac{2}{N}\sum_{i=1}^{N}(y_i - Q^w(x_i, u_i))\frac{\partial Q^w(x_i, u_i)}{\partial w} \tag{11}$$

Once the loss and gradient of the critic is defined we can define the updating rule for the actor, by following the policy gradient theorem. If the actor is represented by a neural network with parameters θ, and $J(\mu_\theta) = Q^w(x_i, \mu(x_i|\theta)|w)$, then by following the deterministic policy we obtain:

$$\nabla_\theta J = \frac{\partial Q^w(x_i, u_i)}{\partial u} \cdot \frac{\partial \mu(x_t|\theta)}{\partial \theta} \tag{12}$$

and thus the optimal policy can be found. Note that the updates to the target networks must be done as shown in Eq. (7).

IV. End-to-End Deep RL

We aim to develop a technique that allows us to solve the navigation problem end-to-end for AUV. Current underwater vehicles have four, six or even more thrusters to control the 6DOF of the AUV. This means that the agent must outputs the same number of continuous control signals to achieve the dynamic goal. The RL techniques previously explained can help the agent to solve the particular end-to-end problem, i.e. reaching a desired goal. However, RL lacks of generalization and therefore the agent should be retrained if the target goal (in this case the way-point) changes. Therefore, we propose an adaptive goal driven architecture which will be explained in the next subsection.

Algorithm 1 Algorithm 1

Initialize/Load Q and μ networks
Initialize target networks $Q' \leftarrow Q$ and $\mu' \leftarrow \mu$
Initialize/Load replay buffer R
while true **do**
  for j = 1 to M **do**
    Initialize a random noise process for exploration
    Get desired way-point x_{ref}
    Set x_0 from AUV initial measurements and x_{ref}
    for t = 1 to T **do**
      Select action $u_t = \mu_t(x_t|\theta)$ + noise
      Execute action u_t
      if $|R| > m$ **then**
        Sample a random minibatch S of m transitions
        for i=1 to N **do**
          Obtain $u'_{i+1} = \mu'(x_{i+1}|\theta')$
          Set $y_i = r_i + \gamma Q'(x_{i+1}, u'_{i+1}|w')$
        end for
        Obtain the critic parameterization w by minimizing the loss function $L(w)$
        Obtain the actor policy parameterization θ using the deterministic policy gradient
        Update the actor target network μ'
        Update the critic target network Q'
      end if
      Get AUV dynamic measurements and set state x_{t+1}
      Observe the reward r_t
      Store the transition (x_t, u_t, r_t, x_{t+1}) in R
      Set $x_t = x_{t+1}$
    end for
  end for
end while
$Q(\cdot, \cdot|w), u(\cdot|\theta), R$

A. Goal driven architecture

In order to avoid re-training the network every time a new goal is required, an architecture to help the neural network agent to generalize over different way-point goals is created. In order to do so, we include information of the desired goal into the system state configuration. Therefore, if a change in the goal occurs the current state (x_t) changes as a consequence. The information of the goal is provided to the agent as the vector $\boldsymbol{x_e}$ obtained from the current position ($\boldsymbol{x_{pos}}$) and the desired goal ($\boldsymbol{x_{ref}}$):

$$\boldsymbol{x_e = x_{ref} - x_{pos}} \tag{13}$$

In Fig. 2 the architecture of the system is shown. The deep agent is shown together with the goal drive architecture

forming the end-to-end agent. As it can be seen in the figure the agent receives information from the navigation system and the goal driven architecture. The current state is thus defined as $x_t = [x_{pos}, x_e, v_t, u_{t-1}]^T$. The information from the sensory system is the current position, the longitudinal speed. In addition the past thrusters commands are also feed to the end-to-end control system module giving information about the system behavior. Finally, the current goal brings an extra information dimension for the end-to-end control system.

Another improvement implemented in our architecture is the goal decay. The goal in the navigation problems means reaching a neighborhood of the desired x_{ref}. In a three dimensional space, we regard an spheric neighborhood, i.e. a sphere with radius β. Initially, when the agent has a poor policy, reaching a goal with low β is extremely difficult, which means that the agent will not be able to experience enough successful episodes to learn an adequate policy. On the other hand, if β is too big, even if the agent learns a policy to reach this way-point, the navigation problem is not considered to be successfully solved since the distance to the x_{req} is too large for practical applications.

Therefore by decaying β as the agent learns and gains experience, it allows it to experience enough successful episodes. And once the policy is learn, the β is small enough so that the navigation problem can be consider as solved.

With this modifications we were able to enhance the deep agent allowing it to solve more complex tasks and, more importantly, generalize over different goals.

B. End-to-End RL algorithm

The final end-to-end proposal is based on an RL agent using the deterministic policy gradient together with deep neural networks as policy approximators. This approach is expanded by means of the goal driven architecture.

Algorithm 1 outlines the pseudo code of our proposal. In the first three lines the networks and replay buffer are initialized. Note that these can also be load, with information about collected experience in previous training, and this may be an important issue for deployments in the real vehicles.

In our formulation, reaching each way-point can be seen as an episodic task. Therefore we define an outer loop that circles trough episode. The episode ends if the agent reaches the goal or if a certain amount of time (T) passes without reaching the goal. Following an initial desired way-point is defined, during training this is simply a randomly generated point into a certain region. With this information and the initial AUV position, the state vector x_0 is generated.

In line 9 the inner loop is defined. This loop cycles from time to a final time T, if the agent is unable to reach the way-point in this time lapse, the episode is considered as ended and a penalization is given. Inside the loop, in line 10, the agent chooses an action to then execute it. If the number of transitions stored in the replay buffer ($|R|$) are bigger than a number m (m is the size of the minibatch used for training) then a training step is carried out. During the training a minibatch of the stored transitions is taken from buffer R. This minibatch S is used to update the critic network by minimizing the loss function L presented in Eq. 8, using stochastic optimization, and to update the actor network following the deterministic policy gradient as in Eq. 11. The training is then completed by updating the target networks as previously explained.

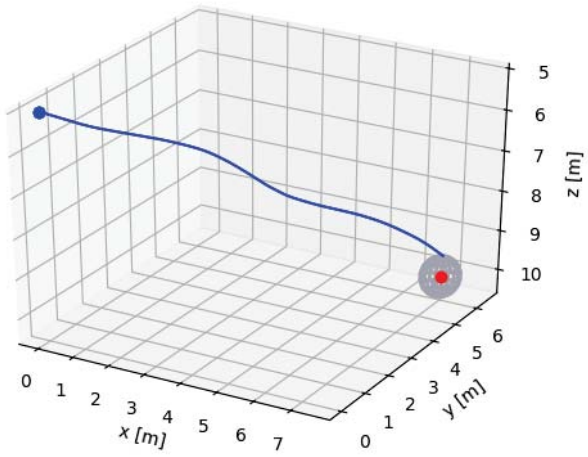


Fig. 3. Test 1: 3D position of AUV

Once the training step has been completed and after the action u_t was applied on the robot, the system evolves to the state x_{t+1} and it is observed (line 23). The reward of taking the action is observed and the transition step is stored in buffer R. Then the current state x is updated (line 26), and the loop continues until the training has been completed. The outputs of the algorithm are the policy networks and the replay buffer.

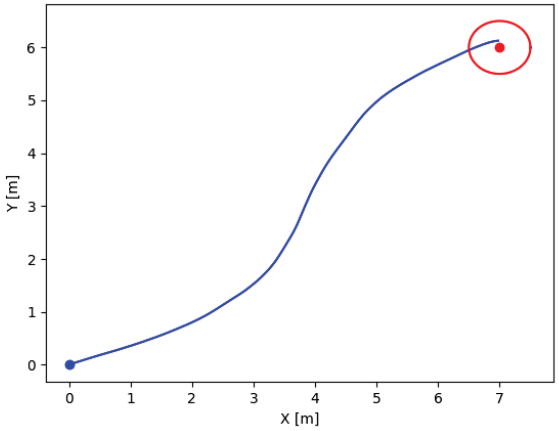
V. Results

The proposed approach was tested in simulation using the model of Nessie VII, developed by the Heriot-Watt University, Fig. 1. This AUV has 6 thrusters, indicated as T1 to T6 in Fig. 1b, that allow for a 5DOF control (surge, heave, sway, pitch and yaw).

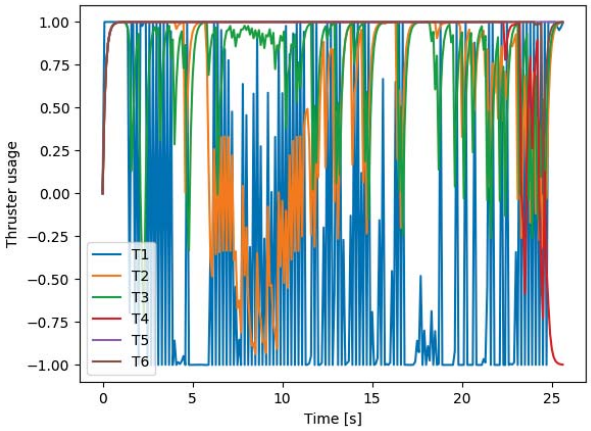
The goal of the algorithm is to reach a way-point (x_{ref}) in space. A way-point is defined as a vector of the $(north, east, down)$ coordinates in meters. During training, a new x_{ref} is generated randomly each episode. The AUV starting point x_0 is the same from episode to episode. We define the reward function as follows:

$$r_t = \begin{cases} 10, & \text{if } d(x, x_{ref}) \leq \beta \\ -1 - f(Tr), & \text{if } d(x, x_{ref}) > \beta \\ -10, & \text{if } d(x, x_{ref}) > \beta_{max} \end{cases} \quad (14)$$

where β is the radius of a sphere in space that defines a neighborhood to x_{ref} in which we consider the goal as reached, β_{max} is the radius of the biggest admitted sphere in which the agent is allowed to move so that $\beta >> \beta_{max}$, d is the euclidean distance function, and $f(Tr)$ is a function that weights the thruster usage. Therefore the reward function is as



(a) Test 1: Position results



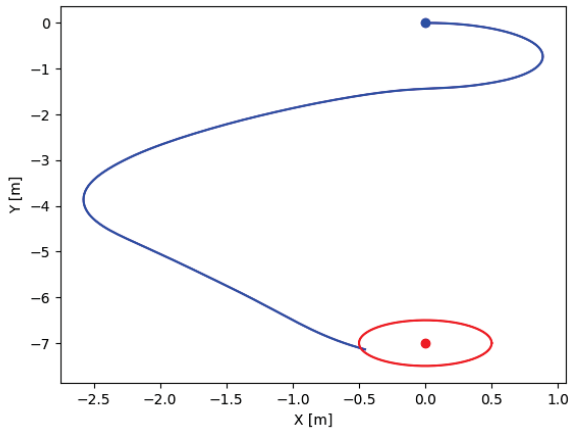
(b) Test 1: Thruster output

Fig. 4. Test 1

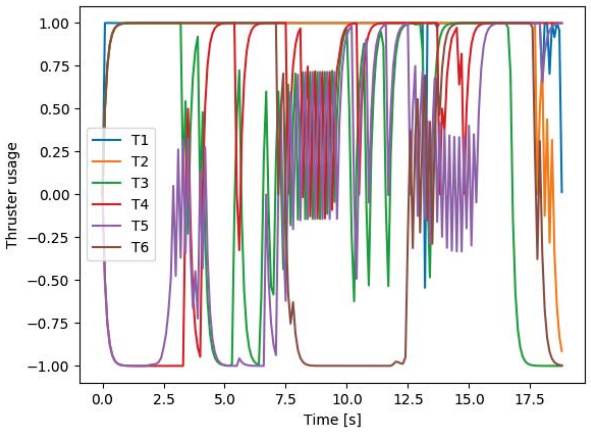
follows, if the distance of the AUV with respect to the way-point is less than β a positive big number is given as reward. On the contrary if the distance is bigger a negative number is given as reward (punishment) discounting also for the amount of thruster usage. Finally, if the distance of the AUV to the the way-point is bigger than a safe value β_{max} then a big negative reward is given and the episode is ended. This is done to consider physical restrictions that may occur during the deployment of the algorithm on the real vehicle. Initially β is fixed as $\beta = 1.5$m and as the number of episodes progresses it decays to $\beta = 0.5$m after 5000 episodes. In this way, we assure that along the initial episodes the goal is sufficiently large so that the agent will be able to achieve it. Therefore, the agent could have some successful experience from which the learning process can progress.

A. Test case scenarios

With the explained architecture we obtained the results shown below. For this we trained the agent for 6000 episodes each episode consisting of 700 time steps long, with a time step equal to 0.1 s. The agent was trained using Tensorflow, on an AMD eight-core processor with 16 GB of memory and a Nvidia GPU, the Titan X. The resulting actor and critic policy were tested in different scenarios, called test 1 and test 2.



(a) Test 2: Position results



(b) Test 2: Thruster output

Fig. 5. Test 2

In test 1 the desired way point was defined as $x_{ref} = [7., 6., 10.]$. The starting position was $x_0 = [0., 0., 5.]$, according to the reference system defined as in Fig. 1b. The goal of the agent is to reach the way point as fast as possible, following the reward function defined in Eq. 14.

In Fig. 4 a 2D representation of the trajectory is shown, together with the thruster usage. As it can be seen the AUV successfully reaches the goal in around 25s. In Fig. 6 a 3D plot of the trajectory followed by the AUV, according the obtained policy, is shown. It is worth noting that the dynamics of the AUV are highly coupled and so it is the thrusters torque applied to the systems.

A second test is shown in Fig. 5, where the goal is set as $x_{ref} = [0., -7., 8.]$. In this test case the AUV reaches the goal in almost 17s. In Fig. 5a it can be seen that since the AUV starts with a position facing away from the way-point

a more aggressive maneuverer is needed in order to reach x_{ref}. However the agent has no difficulty reaching the desired reference point.

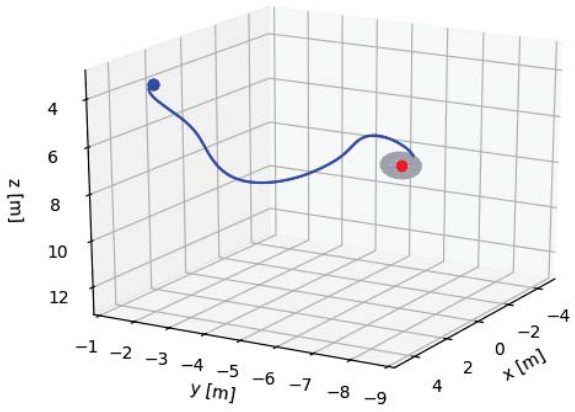


Fig. 6. Test 2: 3D position of AUV

B. Fault recover scenario

After proving that the agent is able to reach way-points in a normal scenario a fault recovery test case was perform in order to analyze the abilities of the agent to recover from unexpected scenarios. To this end a thruster failure is simulated on Thruster one (T1). The failure simulates a thrust reduction of 90%. Results from running test case 1 incorporating this failure are shown in Fig. 7. It can be seen that, due to the lack of thrust, the AUV undershoots the goal, having to perform a heavy turn in order to compensate for the failure. However, the agent is able to reach the goal. In Fig. 8 both the thruster usage and the xy position can be seen. In Fig. 8b the T1 output is shown clipped to display the actual thrust obtained from this thruster due to the simulated failure. It is important to note that even with a failing thruster the agent was able to reach the goal in around 30s.

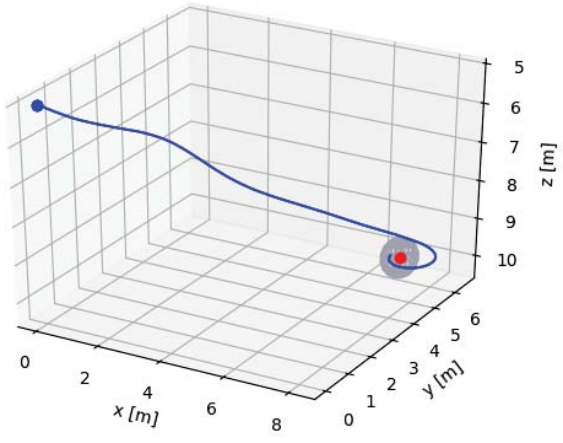
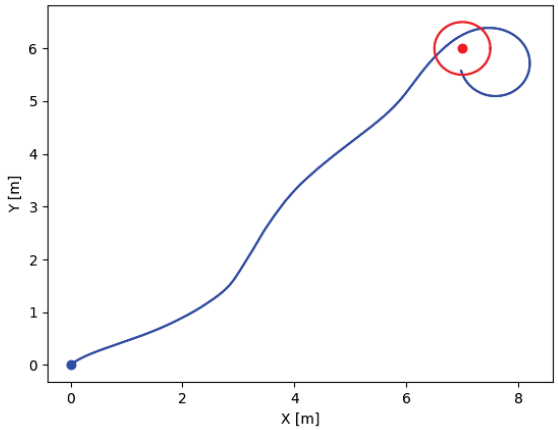
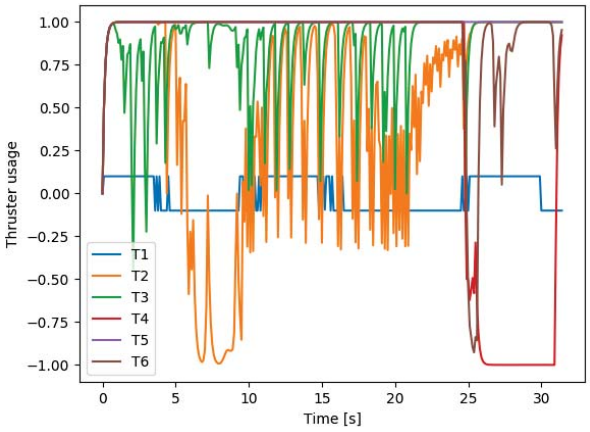


Fig. 7. Fault recover: 3D position of AUV



(a) Fault recover: Position results



(b) Fault recover: Thruster output

Fig. 8. Fault recover

VI. Conclusion

In this article an end-to-end reinforcement learning agent, capable of solving the navigation problem, was presented. For this end, the deep deterministic policy gradient method was used in an actor-critic architecture. In order to solve the lack of generalization problem of the RL agent, a goal oriented architecture was implemented. With this inclusion the agent was able to learn how to successfully navigate to different required way-points without the need of a continuous re-training from scratch.

The proposed system was tested in simulation using the dynamic model of Nessie VII, an autonomous underwater vehicle used as an experimental platform in multiple research works. The obtained simulation experiments demonstrated the feasibility of the proposed architecture for episodic navigation tasks, typical for AUV missions. Moreover, the agent was able to navigate to the way-point during a simulated thruster failure scenario, demonstrating the adaptive behavior of the proposed architecture. As future works, the authors propose

the deployment of the developed technique in the real robot to further explore the capabilities of the reinforcement learning techniques in underwater marine robotics. We also think that a more in depth analysis of the reward function could be performed in order to optimize the performance of the thruster output.

VII. Acknowledgement

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research. Particularly, we thank UNCPBA and CONICET for the financial support of Ignacio Carlucho at the Ocean System Laboratory at the Heriot-Watt University, Edinburgh, UK.

References

[1] L. Lapierre and B. Jouvencel, "Robust nonlinear path-following control of an auv," *IEEE Journal of Oceanic Engineering*, vol. 33, no. 2, pp. 89–102, April 2008.

[2] C. Barblat, V. D. Carolis, M. W. Dunnigan, Y. Ptillot, and D. Lane, "An adaptive controller for autonomous underwater vehicles," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 1658–1663.

[3] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[4] Y. Niv, "Reinforcement learning in the brain," *Journal of Mathematical Psychology*, vol. 53, no. 3, pp. 139 – 154, 2009, special Issue: Dynamic Decision Making.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] D. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, "Continuous control with deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2016.

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *Journal of Machine Learning Research*, vol. 17, pp. 1–40, 2016.

[8] A. El-Fakdi and M. Carreras, "Two-step gradient-based reinforcement learning for underwater robotics behavior learning," *Robotics and Autonomous Systems*, vol. 61, no. 3, pp. 271–282, 2013.

[9] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Reinforcement learning applied to the control of an autonomous underwater vehicle," in *In Proc. of the Australian Conference on Robotics and Automation (AUCRA99*, 1999, pp. 125–131.

[10] M. D. Paula and G. G. Acosta, "Trajectory tracking algorithm for autonomous vehicles using adaptive reinforcement learning," in *OCEANS 2015 - MTS/IEEE Washington*, Oct 2015, pp. 1–8.

[11] B. Liu and Z. Lu, "Auv path planning under ocean current based on reinforcement learning in electronic chart," in *2013 International Conference on Computational and Information Sciences*, June 2013, pp. 1939–1942.

[12] M. Leonetti, S. R. Ahmadzadeh, and P. Kormushev, "On-line learning to recover from thruster failures on autonomous underwater vehicles," in *2013 OCEANS - San Diego*, Sept 2013, pp. 1–6.

[13] S. A. Fjerdingen, E. Kyrkjeboe, and A. A. Transeth, "Auv pipeline following using reinforcement learning," in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, June 2010, pp. 1–8.

[14] G. Frost, F. Maurelli, and D. M. Lane, "Reinforcement learning in a behaviour-based control architecture for marine archaeology," in *OCEANS 2015 - Genova*, May 2015, pp. 1–5.

[15] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, Contour and Grouping in Computer Vision*. London, UK, UK: Springer-Verlag, 1999, pp. 319–.

[16] M. Riedmiller, "Neural fitted Q iteration - First experiences with a data efficient neural Reinforcement Learning method," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3720 LNAI, pp. 317–328, 2005.

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *CoRR*, vol. abs/1511.05952, 2015.

[18] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, pp. 2094–2100.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014.

[20] H. Wu, S. Song, K. You, and C. Wu, "Depth control of model-free auvs via reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–12, 2018.

[21] I. Carlucho, M. De Paula, S. Wang, Y. Petillot, and G. G. Acosta, "Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning," *Robotics and Autonomous Systems*, jun 2018.

[22] 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)*, July 2017, pp. 4958–4965.

[23] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, Nov 2012.

[24] A. El-Fakdi, M. Carreras, N. Palomeras, and P. Ridao, "Autonomous underwater vehicle control using reinforcement learning policy search methods," in *Europe Oceans 2005*. IEEE, 2005, pp. 793–798 Vol. 2.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.