Deep Learning for Station Keeping of AUVs

Kristoffer Borgen Knudsen
kristobk@stud.ntnu.no

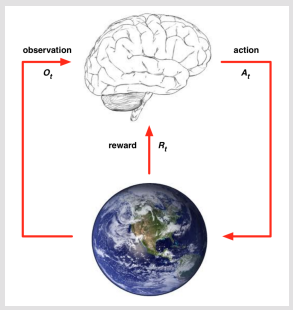
NTNU
Norwegian University of Science and Technology

Problem

The process of doing underwater control design is complicated and in some cases infeasible. This is due to unobservability and highly nonlinear effects in the underwater environment, which makes the autonomous control nonlinear since flow and hydraulic resistance easily influence AUVs motions [1].
These disadvantages have triggered the interest of using artificial intelligence, more precisely machine learning, in design of underwater control systems. This thesis investigates the possibilities of using deep reinforcement learning to accomplish 6 DOF station keeping capabilities of AUVs.

Basic Concepts

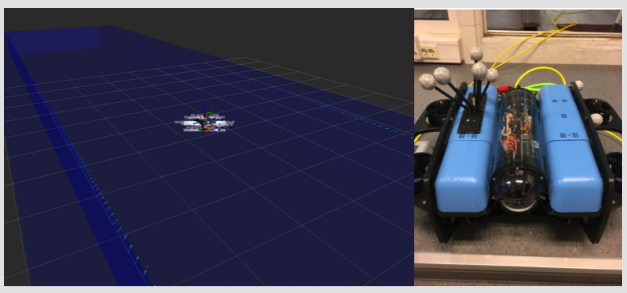
The branch of machine learning used in this thesis is denoted deep reinforcement learning (DRL). In reinforcement learning (RL) the overall goal is to train an agent, or algorithm, to perform correct actions in an environment. This is done by giving the agent a reward based on how good it was to take a particular action in a specific state. The overall goal is to maximise the total cumulative reward over all states, such that the optimal policy, or strategy, to follow is found [2]. The RL architecture is presented in the figure below.



DRL are a family of methods within RL that use the principles of artificial neural networks (ANNs), which are based on on the idea of how biological nervous systems, such as the human brain, operates [3]. In the DRL family three state-of-the-art methods are significant: Deep Deterministic Policy Gradients (DDPGs), Trust Region Policy Optimisation (TRPO) and Proximal Policy Optimisation (PPO).

Models

The algorithm is trained and validated using a dynamic model of the BlueROV2, in combination with the simulation environments Gazebo and Robot Operating System (ROS). Station keeping capabilities are also validated on the real-life system in the marine cybernetics (MC) lab at Tyholt.



References

[1] R. Yu, Z. Shi, C. Huang, T. Li, Q. Ma: *Deep Reinforcement Learning Based Optimal Trajectory Tracking Control of Autonomous Underwater Vehicles*, Proceedings of the 36th Chinese Control Conference, 2017

[2] D. Silver: *Introduction to Reinforcement Learning*, University of London lecture notes, 2015

[3] K. O'Shea: *An Introduction to Convolutional Neural Networks*, Research gate - Aberystwyth University, 2015

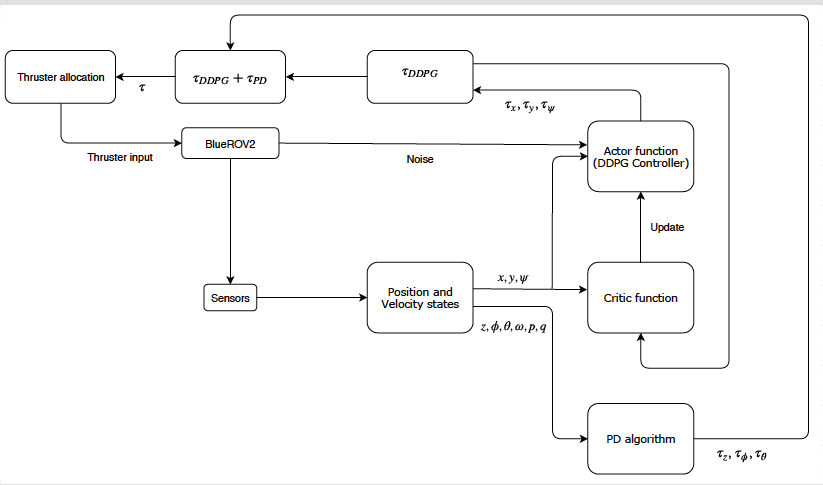
Acknowledgements

I would like to express gratitude towards my supervisor, Professor Ingrid Schjoelberg for guidance throughout the project period. Furthermore, I want to give a special thanks to Postdoctoral Fellow Mikkel Cornelius Nielsen for developing the simulation environment, as well as valuable discussions throughout the project period.

Method

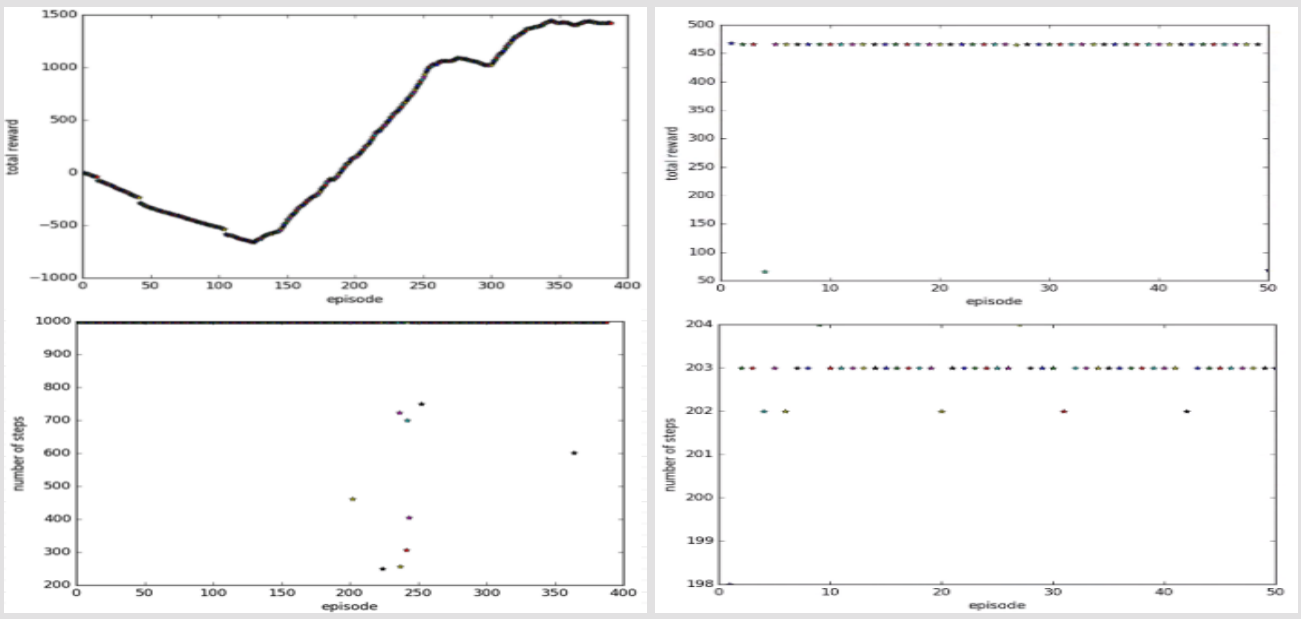
The proposed method is to develop a dual control design, which combines a proportional-derivative (PD) controller with a deep deterministic policy gradient (DDPG) algorithm. DDPGs are the type of DRL algorithms that are characterised as off-policy, model-free and actor-critic policy gradients. The benefit of using these is that no assumptions about the environment or dynamic model are needed, which means that the problems related to classical control designs are removed.
The goal is to accomplish station keeping, which is the ability to maintain a constant position and orientation (pose), with regard to a reference object. To do this, a DDPG algorithm is used to control the states: surge x and sway y, while the PD algorithm is used to control the states: heave z, pitch ϕ, roll θ and yaw ψ. The reason for not using DRL techniques on all 6 states is that difficulties related to convergence towards an optimal policy increases exponentially with the number of DRL states. The developed DDPG algorithm and controller architecture are displayed in the figure below.

Algorithm 1: DRL algorithm
Initialize the networks of $Q(s_t, a_t|\omega)$ and $\mu(s_t|\theta)$
with weights ω and θ.
Initialize the copy networks of $Q'(s_t, a_t|\omega')$ and $\mu'(s_t|\theta')$
with weights ω' and θ'.
Initialize the replay buffer R
while $\left(\sqrt{\frac{\sum_{i=M-100}^{M}\sum_{i=1}^{T}|r_j(s_i,a_i)-r(s_t,a_t)|^2}{100T}} > \varepsilon_r\right)$ **do**
(M is the current training episode)
  Initialize the state s_0
  for t=1, T **do**
    Choose actor $a_t = \mu(s_t|\theta)$
    Get the state s_{t+1} according to the environment
    compute $r(s_t, a_t)$
    store in transition $R(s_t, a_t, r_t, s_{t+1})$
    Randomly select N arrays from R
    compute $y_i = r_i + \gamma Q'(s_i, a_i|\omega')$
    compute $Loss = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\omega))^2$
    compute $\nabla_\omega Loss = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\omega))\frac{\partial Q(s_i,a_i|\omega)}{\partial \omega}$
    update weight $\omega_{t+1} = \omega_t + \alpha \cdot \nabla_\omega Loss$
    compute $\nabla_\theta J = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial Q(s_i,a_i|\mu(s_i|\theta))}{\partial a_i}\cdot\frac{\partial \mu(s_i|\theta)}{\partial \theta}$
    compute $m_t = \varphi \cdot m_{t-1} + (1-\varphi)\nabla_\theta \mu$
    compute $\Im_t = \beta \cdot \Im_{t-1} + (1-\beta)\nabla_\theta \mu$
    compute $\hat{m}_t = \frac{m_t}{1-\varphi^t}$
    compute $\hat{\Im}_t = \frac{\Im_t}{1-\beta^t}$
    update weight $\theta_{t+1} = \theta_t - \eta\frac{1}{\sqrt{\hat{\Im}_t}}\hat{m}_t$
    update weight $\omega' = \rho\omega + (1-\rho)\omega'$
    update weight $\theta' = \rho\theta + (1-\rho)\theta'$
    (ρ is learning rate)
  end for
end while
end

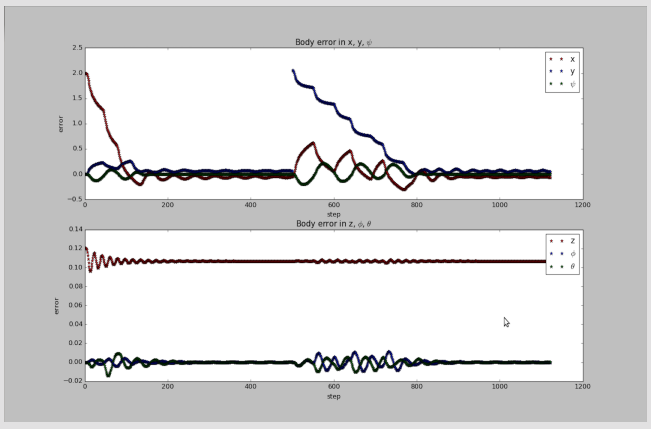


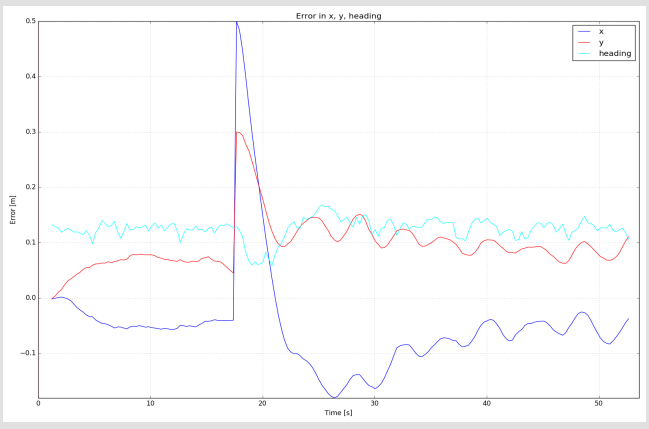
Results and Discussion

The algorithm was trained for ≈ 600 episodes in the simulated environment, where each episode is ran for 1000 simulation steps or until the vehicle satisfies station keeping capabilities (< 1000). Training is restarted from episode 400 due to problems with the software when the number of episodes increased. The right figures displays the last 50 episodes, and as shown the agent has converged towards and optimal policy, using approximately 203 steps each simulation to accomplish station keeping.



The performance of the controller was evaluated in both simulation and real-life. The left figure below display the validation results from simulation, where the agent received arbitrary desired poses for station keeping. As shown, the vehicle is accomplishing station keeping with error values in the order of $10^{-2}m$. The right figure displays the results from real-life experiments, where the agent is given a new arbitrary desired pose at $t = 18s$. The agent is able to sufficiently reduce the error, but the values are now in the order of $10^{-1}m$. The main reason for the differences in performance was due to the flaws in the real-life pose measurement system, Qualisys.





The problems related to Qualisys made it difficult to measure the performance of the controller to its full extent. Further work on the topic should try to resolve this issue, and one method could be to implement an observer, such that observer estimates can be used when the pose is lost. Furthermore, DDPG algorithms suffers from convergence issues. However, this is not the case for TRPO and PPO methods, which could be a possible implementation for further work.