CEORL

Cost of Energy Optimised by Reinforcement Learning

WES Control Systems Stage 3

Public Report

MaxSim Ltd.

This project has been supported by Wave Energy Scotland

Contents

1. Introduction

This report summarises the CEORL[1] project funded by Wave Energy Scotland[2] [9]. The CEORL project uses reinforcement learning (RL) to discover optimal[3] control policies for heaving buoy wave energy converters (WECs) operating in irregular seas.

A control policy of a WEC is a function that specifies how to operate the directly-controllable aspects of the WEC, for example, what force the power take-off (PTO) mechanism should apply under what conditions. The PTO applies forces to the body of the WEC that can oppose or enhance those applied by the waves. Depending on the magnitude and direction of the applied PTO forces in relation to the wave forces, the WEC can either absorb energy from, or dissipate energy to, the surrounding water. A good control policy is one that controls the PTO in such a way that it applies forces that maximise a specified metric in the long term. A simple metric could be the mean power absorbed by a WEC operating within the force and stroke limits of its PTO. More advanced metrics could be defined that measure the levelised cost of energy (LCOE) or the return on investment (ROI).

At the start of Stage 3 our goal was to demonstrate that RL could find better control policies than a suitably chosen baseline control in a scale model in a wave tank. However, by the middle of 2020 the UK was suffering mandatory restrictions and lock-downs because of the COVID-19 pandemic. This caused considerable uncertainty about the availability and access to wave tank facilities. In response, we changed our work plan by replacing the wave tank experiments with another RL approach called "offline RL" [3]. The project did not, therefore, demonstrate our original goal that RL could find better control policies than a baseline control in a scale model in a wave tank. We did, however, show that RL can find better control policies than the baseline in simulations and we showed that RL can learn directly on a real device, where the real device was our rapid control prototyping (RCP) rig configured to mimic a heaving buoy WEC. At the start of Stage 3 we did not expect to be able to use RL to train policies directly on a real device, instead we thought we would need to learn policies in very accurate numerical simulations of the target real device. Showing that RL can learn directly on a real device in a relatively short training time is arguably more beneficial than training policies in simulations prior to testing them in a tank. This is because the approach of learning directly on a real device is more versatile and could be applied directly to WECs for which very accurate numerical models are not available.

In the following sections we summarise the main findings from Stage 3. The summary is at a high level and does not review the technical details in depth. An appendix contains a list of abbreviations.

2. Background

2.1. Rapid Control Prototyping Rig

At the start of Stage 3 we had planned to build the rapid-control-prototyping (RCP) rig and a scale WEC model as shown in Figure 1. The RCP rig, on the left of the figure, consists of two servomotors that react against each other by pulling on the two ends of a tether that connects them. The upper motor, labelled as "PTO winch" in the image, is the PTO motor that extracts power from the lower motor, labelled as "HIL actuator" in the image. The lower motor applies a force to the tether that simulates the force that the scale WEC would apply to the PTO in the image on the right of Figure 1.

The scale WEC, depicted on the right in Figure 1, is 1/35 scale and has dimensions of roughly $(r_0, d_0) = (7.57\,\text{m}, 6.14\,\text{m})$ at full scale, however, it is relatively easy to change the scale of the rig by changing the mass of the flywheel attached to the actuator. For all the results presented in this report, we configured the rig to represent a 1/15 scale WEC with a geometry of $(r_0, d_0) = (1.75\,\text{m}, 5\,\text{m})$. We also used a simplified hydrodynamics model based on the relative motion hypothesis (see Newman [5, page 303]). Using this hydrodynamics model allowed us to compute theoretical upper bounds for the power absorbed by the WEC. This was helpful when assessing the quality of learnt policies.

[1] CEORL stands for Cost of Energy Optimised by Reinforcement Learning; we pronounce it as "coral".

[2] This report is work commissioned by Wave Energy Scotland. The views expressed in this publication are those of the author and not necessarily those of Wave Energy Scotland.

[3] We use the word "optimal" as a short-hand for "close to optimal" or "near-optimal" because there are no guarantees that the truly optimal controls can be found in finite time for stochastic systems with complex non-convex cost functions.

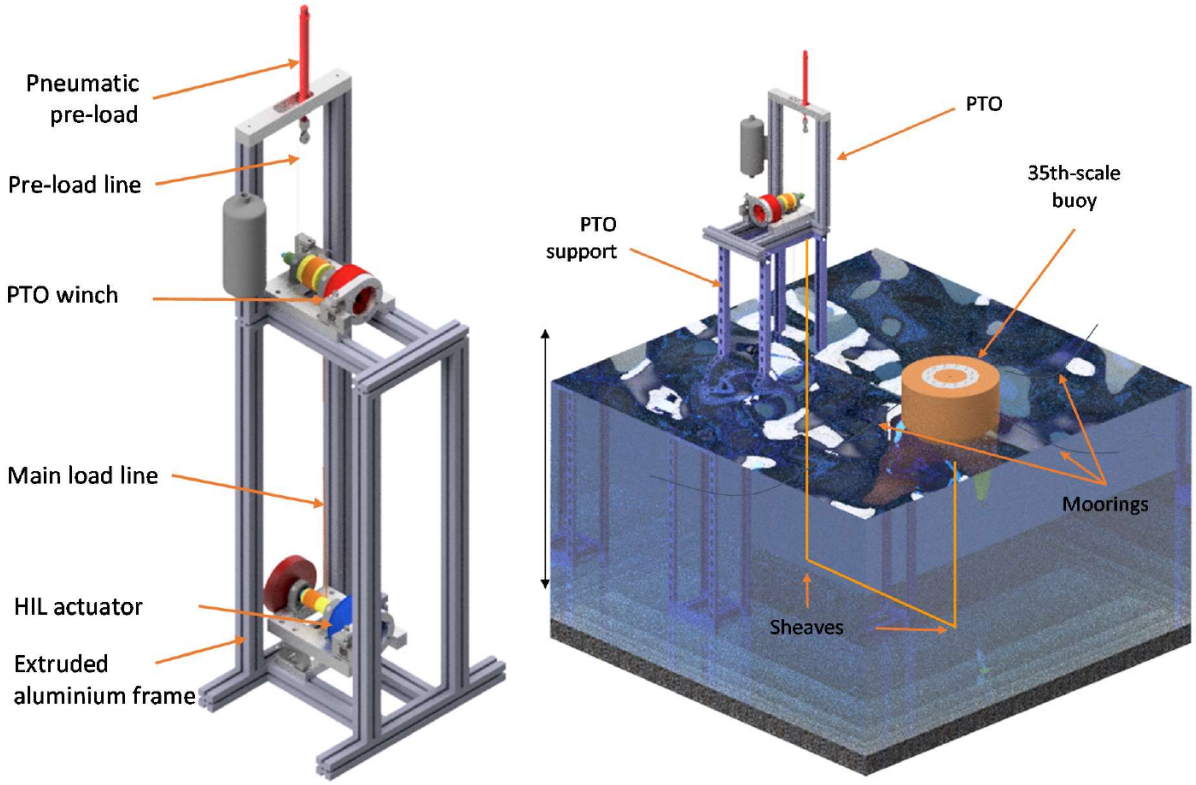

Figure 1: RCP rig design (right) and tank test set-up (left).

2.2. Rationale and Overview of Approach

During Stage 3 we focused entirely on point absorber WECs, also referred to as heaving buoy WECs. We had a baseline control that we used as a benchmark to compare with control polices learnt by RL. Our baseline control consisted of the class of control policies defined by constant spring, k_c, and damping, b_c, values applied by the PTO. This is a suitable baseline because, provided one has access to data from a WEC operating in various sea-states defined by their significant wave height, H_s, and energy period, T_e, it is relatively easy to group seas by the similarity of their (H_s, T_e) and then find optimal (k_c, b_c) for each group by applying a simple grid-search or hill-climbing search in the 2D space of available (k_c, b_c) values. With some care, this approach could be applied to find optimal (k_c, b_c) values for a real WEC operating in sea-states with time-varying (H_s, T_e). A simple application of this approach could yield a small look-up-table (LUT) that would map the cell containing the current sea state, as defined by its (H_s, T_e), to the optimal values of (k_c, b_c). This is a common, simple, and effective approach for optimising heaving buoy WECs. Using this approach, the baseline control force can be written as

$$F_{c,i}(t) = -k_{c,i} z(t) - b_{c,i} \dot{z}(t), \tag{1}$$

where $F_{c,i}(t)$ is the PTO control force, i indexes the discrete cell of the LUT, $z(t)$ is a position measurement of the PTO, and $\dot{z}(t)$ is the rate of change of that position measurement with time. For a simple heaving buoy WEC moving only in heave, one can think of $z(t)$ as being the heave displacement of the WEC.

When finding an optimal control policy using RL, the control force can be written as

$$F_{c,i}(t) = \pi_{\theta_i}\big(s(t)\big), \tag{2}$$

where $s(t)$ is an arbitrary state vector, and π_{θ_i} denotes the policy function which is represented by a deep neural network with θ_i representing many hundreds of constant policy parameters that depend on the (H_s, T_e) cell of the LUT in the same way as $(k_{c,i}, b_{c,i})$. The state vector $s(t)$ can include $z(t)$ and $\dot{z}(t)$ as components. It is easy to see that the class of baseline control policies expressed by (1) is a subset of the

class of control policies expressed by (2). Because of this, successful application of RL should find control policies given by (2) that are no worse, and potentially substantially better, than the optimal baseline control policy given by (1).

2.3. On-Policy RL and Offline RL

By the end of Stage 3 we had pursued two parallel, and largely independent, workstreams that culminated in:

1. Training and testing control policies using an on-policy PPO [7] agent learning and acting directly on the real RCP rig.
2. Training control policies using an offline CQL [2] agent learning from data generated by the real RCP rig, then applying the learnt policy to the real RCP rig.[4]

The breakthrough that allowed us to incorporate Item 1 above in the change of scope was the large reduction, by a factor of about 16, in the number of learning steps required to train a good policy with an on-policy PPO agent. This made it feasible to learn directly on the real rig in around one or two hours of training as opposed to a day or so of continuous training (such long training periods would have had to be split over several days because the rig cannot be left to run unattended).

Being able to train directly on a real rig or a real WEC has, potentially,[5] the great benefit of totally avoiding the need to solve the problem of the sim-to-real gap [10]. The sim-to-real gap describes the situation where discrepancies between a simulation of the target environment and the real target environment result in control policies that perform well in simulation but poorly in the real environment. Solving this problem is considered to be one of the hardest things in robotics. Clearly, if an RL agent is training directly on the target environment, such as a real WEC, there can be no sim-to-real gap, so there is no need to learn policies in a simulated environment before applying them to the real target environment.

The second workstream, i.e. Item 2 above, was enabled by the rapid progress made in offline RL [3] and the availability of the d3rlpy codebase [8] that implemented many offline RL algorithms. The offline RL workstream also avoided the problem of the sim-to-real gap because we planned to learn from data generated by our real RCP rig, which was our target environment. The drawback of the offline RL workstream is that offline RL was developed to take advantage of large volumes of data that already exist, however, for our real RCP rig large volumes of data did not already exist. This meant that we would have to generate large enough volumes of data from our rig to successfully train offline agents.

3. Main Findings

3.1. Sim-to-Real Gap in the RCP Rig

As we mentioned in Section 2.3, dealing with the sim-to-real gap is considered to be one of the hardest problems in robotics, and how to deal with it comprised a large part of the Stage 3 application. We were going to start with a basic attempt at system characterisation, and extend that as necessary, possibly culminating in using a so-called "actuator network" as described in Hwangbo et al. [1]. Actuator networks are neural networks that are trained to estimate force output values from various input values. They are trained by supervised learning on a real device, and as such, they provide accurate mappings between the inputs and force outputs on the real device. These accurate mappings are then used inside the simulations to reduce the sim-to-real gap.

[4] We ended up applying the offline RL to our realistic Simscape numerical model of our RCP instead of to the real RCP rig directly. The two reasons for this: (i) we could not fix the free-energy problem (see Section 3.3) in the real rig but we could fix it in the Simscape model; (ii) the long time (36 hours) required to run the rig to gather enough data for offline learning.

[5] Initially we thought that being able to train directly on the real rig would definitely avoid the problem of the sim-to-real gap, however, instead of a sim-to-real gap with the "sim" representing the numerical simulation code, we had a sim-to-real gap where the "sim" expresses the fact that the real RCP rig is a "simulation" that is not a perfect model of a real scale model WEC operating in a wave tank.

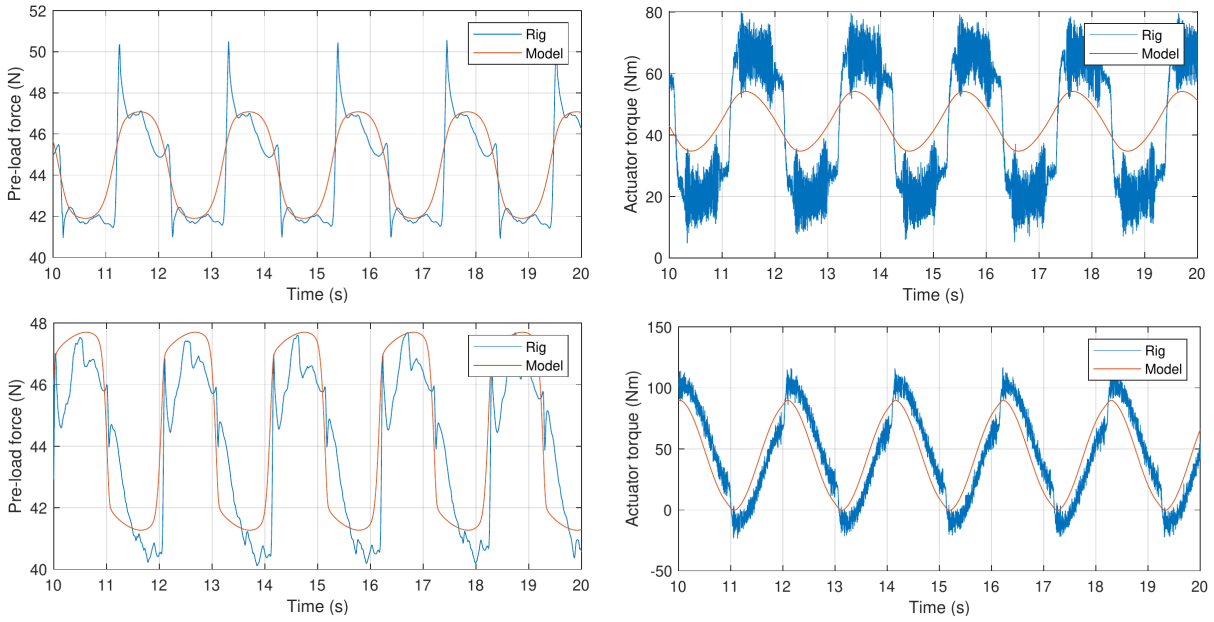

Figure 2: Comparison of measured and simulated pre-load force and actuator torque in regular waves with $T = 2.1$ s, $A = 0.033$ m, Upper plots: for Mod(Z) control with $(b_c, k_c) = (80.8\,\text{Ns/m}, 0\text{N/m})$. Lower plots: for Budal limit control with $(b_c, k_c) = (20.2\,\text{Ns/m}, -278.2\text{N/m})$.

Examples of the sim-to-real gap in our RCP rig are shown in Figure 2. Measured values are blue and labelled as "Rig", simulated values using our basic attempt at system characterisation are red and labelled as "Model". All the plots show results for regular $T = 2.1$ s, $A = 0.033$ m waves. The upper row shows results for Mod(Z) control (i.e., the damping-only control that gives most power), and the lower row shows results for control settings that give Budal's upper bound for power. The plots illustrate the size of the sim-to-real gap in our RCP rig. In these plots, the pre-load force is provided by the pneumatic pre-load cylinder. We expected this cylinder to provide a force that was close to constant and independent of its position, as it was supposed to simulate the constant buoyancy force, $-g\rho\pi V_0$ where $V_0 = \pi\, r_0^2 d_0$, of the WEC on still water. It is clear from the plots how difficult it is for basic system characterisation methods to replicate the highly complex functions of the real measurements.

It is possible that actuator networks could have been trained to give a far better match to the real rig measurements, however, we did not need to attempt this because of the move away from training in simulations to training directly on a real device. This is fortunate because a method that uses actuator networks would require at least the following three phases if being applied to a wave tank model:

1. Train actuator networks by long runs of supervised learning in a wave tank.
2. Incorporate the trained actuator networks into the simulation software and use RL to train control policies.
3. Test the learnt control policies on the real WEC in the wave tank.

It could require many iterations of these three phases to develop a working system. This would have been possible on the rig, but it would have been very expensive to do in a wave tank, and probably would have required multiple campaigns. As explained in Section 2.3, the approach of attempting to reduce the sim-to-real gap and train in simulations was replaced in favour of training directly on a real device, which could be a real RCP rig or a scale WEC operating in a wave tank.

3.2. Reduced Training Time

As mentioned in Section 2.3, the important improvement in RL training that made it feasible to train directly on a real device was the large reduction, by a factor of about 16, in the number of learning

steps required to train a good policy using an on-policy PPO agent. This reduction in training time was first obtained in the idealised simulations using our custom Python WEC environment and the Stable Baselines3 (SB3) [6] RL codebase. The main change that led to faster learning was the change from the 2D action space of control spring and damping parameters, (k_c, b_c), to the 1D action space of control force, F_c. This change was also implemented in our idealised Simulink and realistic Simscape versions of the WEC environment using the MathWorks RL Toolbox [4], and then in the RL code that trained directly on the RCP rig. In all cases the large increase in training speed was observed. The factor of 16 reduction in the number of training steps required to learn meant that training on the real rig took less than 2 hours. We did not test the training in a wave tank, but it is likely that when training in a wave tank we would want to increase the dimensions of the state space because there will be additional degrees of freedom that are not present in the rig. However, the factor of 16 reduction in training time is still likely to apply if we use F_c as the only action.

3.3. The Problem of "Free Energy"

3.3.1. "Free Energy" in Simulations

Another benefit of changing the learnt actions from (k_c, b_c) to F_c was that we discovered a particularly troubling phenomenon whereby RL could learn to extract energy from still water (which would obviously be impossible in a real WEC). We called this "the free-energy problem". It had previously gone unnoticed because, when training RL on a heaving buoy environment with no waves and (k_c, b_c) as the controllable actions, the RL exploration of the actions always leads to zero control force if z and $\dot{z}$ are zero in (1). In Stage 2, we had concerns that something like the free-energy problem was present in our simulations. We attempted to test for it by starting simulations with the buoy's height set to above its equilibrium position so that z and $\dot{z}$ were initially non-zero. However, we did not observe RL extracting energy from still water in those systems. In retrospect, this was presumably because the motion of the buoy damped down to zero faster than the time required for the RL agent to learn policies that could extract energy from still water. The RL agent's exploration of actions involved it continuously trying different values of (k_c, b_c) which would all return a zero control force once the buoy's initial motion had damped down to where z and $\dot{z}$ were both zero.

When we made the change to using F_c as the action, it was easy to operate the buoy on still water for long periods of time during which the control force would never be zero, even when z and $\dot{z}$ were both zero, because, in these systems the RL agent's exploration of actions involves it continuously trying different values of F_c. In these systems the RL agent had enough time to learn to exploit unrealistic aspects of the numerical simulation to increase its reward, and, therefore, generate mean power from still water.

Once the problem of free energy in the simulations was identified, we discovered a number of changes to the simulation code that could fix the problem. These change were enough to prevent the free energy problem on still water and in waves, however, they did not prevent F_c from being very discontinuous with the learnt policy instructing large changes to F_c over very short time periods. This discontinuous behaviour of F_c could be fixed by so-called continuity costs. These are costs that are added to the reward function to penalise discontinuity in the actions, which in this case is F_c. Examples of simulations using learnt control policies that provide smooth control forces with no free energy are presented in Section 3.5.1.

3.3.2. "Free Energy" in the Real RCP Rig

The problem of free energy was first noticed, and solved, in the Python code of the idealised WEC environment. The same, or very similar, solutions were found to work in the Simulink and Simscape versions of the simulation code. However, the same solutions did not work when applied to the real RCP rig. In fact, we discovered that on-line RL can exploit an artefact of the rig that was not present in our simulations and would not be present in a tank test. The artefact appears as an undamped heave oscillation that is apparent when the rig simulates the WEC floating on still water. The undamped heave mode is shown between $t = 188.5$ s and $t = 191.5$ s in Figure 3. It appears as a 2.5 Hz heave oscillation with a very small amplitude of about 0.5 mm. In a wave tank this would not occur because such a heave

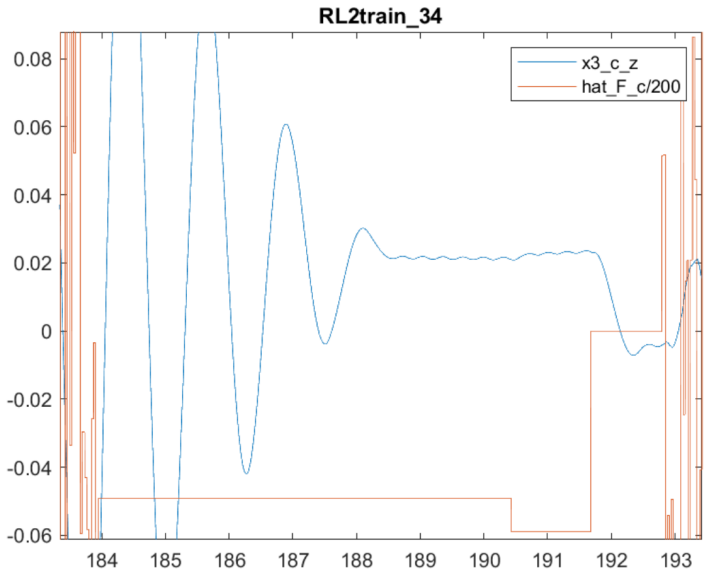

Figure 3: Two types of oscillations in a policy training on still water. The close-up shows a gap between training episodes where the control force has a constant value. The x-axis are time in seconds. The blue line shows large oscillation in heave position, `x3_c_z` (units are metres), decaying rapidly due to radiation damping. It also shows small 2.5 Hz oscillations between about $t = 188.5$ s and $t = 191.5$ s that do not decay and, therefore, appear as undamped modes of oscillation.

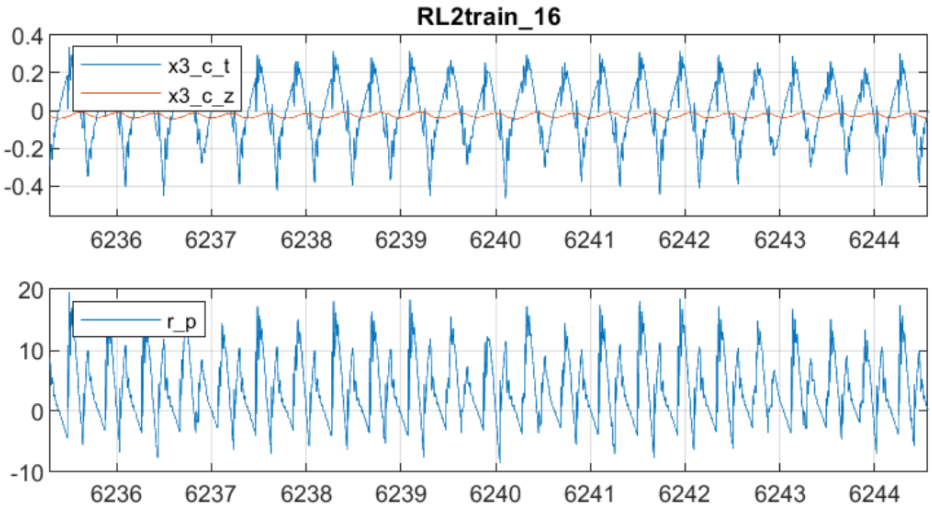

Figure 4: Results from controlling the rig with a policy learnt directly on the rig by a PPO RL agent trained on still water. The x-axes are time in milliseconds. Upper: the heave position, `x3_c_z` (units are metres) and the heave velocity, `x3_c_t` (units are metres/second), of the WEC. Lower: the instantaneous power, `r_p` (units are Watts), which can be seen to have a mean value greater than zero.

oscillation would soon damp to zero on still water. It occurs in our rig because the actuator attempts to move the mass to the demand positions computed by the simulation code running on the Speedgoat target computer; and, the low-level PID controller of the actuator was configured to minimise delays in reaching the demanded positions by being highly responsive to changes in position demands. The consequence of these configuration choices is that the actual position tends to repeatedly overshoot and undershoot the demanded position by a small amount. This is what is shown in Figure 3 between $t = 188.5$ s and $t = 191.5$ s. During this time range, the red line shows a constant force on the WEC (apart from a single step in the force), and blue line shows what appear to be undamped heave oscillations of the WEC.

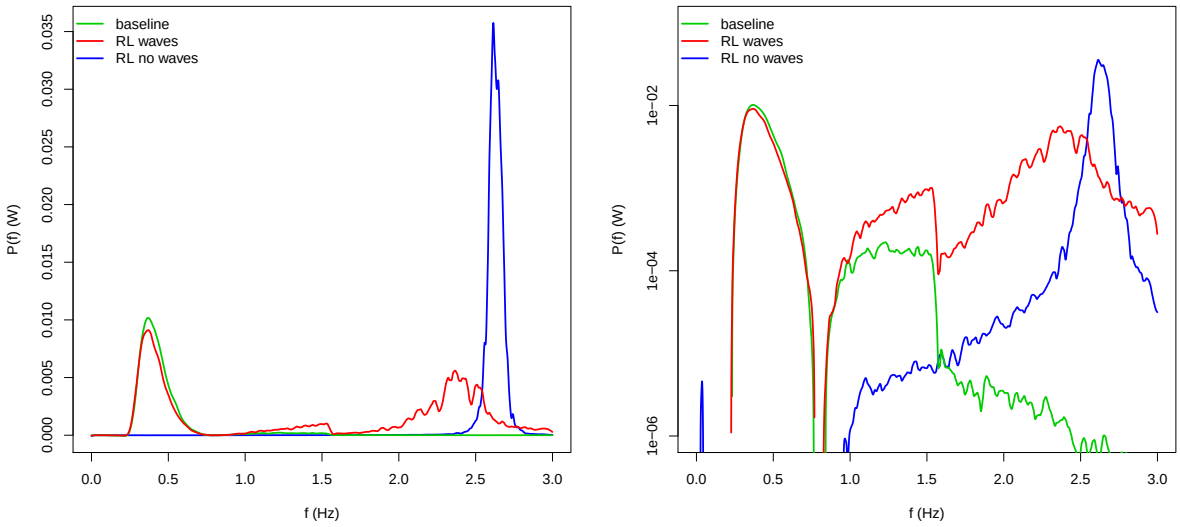

Figure 5: Power as a function of frequency for various control policies run on the real RCP rig. Green: the baseline control in waves. Red: RL-learnt control in waves. Blue: RL-learnt control on still water.

Using a position demand is common in RCP rigs as it allows them to model variable inertia (e.g., inertia that changes with frequency, as it does in a WEC with frequency-dependent added mass). Usually, such a small oscillation would not adversely affect the operation of the rig, however, we found that when applying RL to the rig the RL agent learns to exploit this undamped mode by applying PTO control forces that increase the mode's amplitude from about 0.5 mm (as shown by `x3_c_z` in Figure 3) to about 1.5 mm (as shown by `x3_c_z` in Figure 4), while also timing the application of negative and positive PTO control forces to transfer energy from the actuator to the PTO (as shown by the mean of the power `r_p` being greater than zero in the lower plot Figure 4). Because of the high PTO forces (10 N) and the relatively high frequency (2.5 Hz, compared with the far lower peak frequency, 0.37 Hz, of the sea state), this effect transfers substantial amounts of energy from the actuator to the PTO. The transfer rate is about 3 W, which is larger than the 2.3 W which is the highest power achieved when optimising constant spring and damping controls, (k_c, b_c), by the grid-search method. In standard rig tests this undamped mode can be safely ignored as insignificant, however, the mode becomes a dominant feature of the tests when applying RL to maximise energy transfer between the actuator and the PTO.

Figure 5 shows plots of mean power absorbed as a function of frequency for the three cases of the optimised baseline control in waves (green), an RL-optimised learnt control in waves (red), and RL-optimised learnt control in still water (blue). In the experiments with waves, the waves were generated from a Pierson–Moskowitz (PM) spectrum with $(H_s, T_e) = (0.294\text{ m}, 2.329\,\text{s})$ at rig-scale, which is 1/15 of full-scale. This gives an "energy frequency" of $f_e = 1/T_e = 0.429$ Hz. The peak period was $T_p = 2.711$ s, which gives a peak frequency of $f_p = 1/T_p = 0.369$ Hz. The wave spectrum was truncated for frequencies greater than about 1.55 Hz, so there was no wave energy with frequencies above about 1.55 Hz. The plot for the baseline control (green line), shows that the bulk of the power absorbed is from frequencies around f_e or f_p, however, for both the learnt control policies we see significant power absorbed at frequencies around the 2.5 Hz value of the unphysical undamped mode present on the rig.

Table 1 gives the mean powers from each of the experiments shown in Figure 5. It can be argued that our RL system is successfully learning on the real rig because we are asking it, through the reward function, to maximise the power transferred from the actuator to the PTO, and it is successfully doing that in both the cases with and without waves. This demonstrates the ability of RL to perform well when learning directly on a real device, but, unfortunately it is not learning policies that would be useful in a real wave tank because it is exploiting a phenomenon that would not be present in a real wave tank, that is, the undamped heave oscillations at around 2.5 Hz. This is yet another good illustration of the importance of reducing any "gaps" between the environment the RL is trained on and the target environment that we

On-Policy, Real RCP Rig		
Control	Mean Power (W)	Free-Energy Present
Baseline	2.1507	No
RL waves	4.7105	Yes
RL still water	3.6906	Yes

Table 1: Mean powers from various control policies run on the real RCP rig.

are trying to learn about through the simulations. In Sections 2.3 and 3.1 we discussed the sim-to-real gap, which represented the gap between numerical simulation and a real device. In the example in this section we have a sim-to-real gap where the simulation is the real RCP rig and the real device would be a real scale model WEC operating in a wave tank. In this example, the "gap" is between the real RCP rig, (with its undamped heave oscillations at around 2.5 Hz) and a real scale model WEC operating in a wave tank (which would have no undamped heave oscillations). One can take the idea of the sim-to-real gap a step further and suggest that "gaps" will be present between the "simulation" provided by a real scale model WEC operating in a real wave tank and the target environment of a real WEC operating in the open ocean. An example of such a "gap" could be that RL learns to exploit wave reflections in a relative narrow wave tank that would not be present in the open ocean. In this example, the power measured from RL learning in the simulated environment (i.e., the real scale model in a real wave tank) may give an unrealistic estimate of the power uplift RL could achieve in the real environment (i.e., a real full-scale WEC operating in the open ocean). This illustrates the importance of being able to learn directly on the target environment, and it also highlights the caution required when interpreting the results of RL experiments run on simulated environments (numerical or real) that are not the final target environment.

We have a possible solution to the free-energy problem in our RCP rig that we plan to implement in future. It is to run the actuator in force-demand mode, instead of position-demand mode, so that there will be no actuator delays in the force demand like there currently are in the position demand. This is a valid approach within our numerical approximation that uses the relative motion hypothesis to approximate the fluid force at the peak period of the wave spectrum, and, therefore, allows us to use a constant mass in our tests. If we were to use a more realistic model of the fluid forces that included a frequency-dependent added-mass we would not be able to use a constant mass and we would not be able to use a force-demand to control the actuator. In fact, the reason we initially chose to operate the actuator in position-demand mode is because we anticipated that we would need to run more accurate models of the fluids forces with frequency dependent inertia, but this is not necessary if learning directly on a real WEC.

3.4. On-Policy RL Agents Learning on the Real Rig

During the course of Stage 3, the CEORL team developed their own software to allow learning on our real rig. The software was written to communicate between the real rig and an RL agent (from MathWorks RL Toolbox) in such a way that the real rig appeared as a standard simulated environment from the perspective of the RL agent. Unfortunately, the software suffered some problems with the communication causing large variances in the learning time steps used by RL. This variance is illustrated in Figures 6 and 7. In Figures 6 all the "Delta t(s)" (on the y-axis) should be 0.02 s, which is marked by the red lines, however, the black points show the large variance in the measured values. Figure 7 shows probability mass plots of time step sizes from the first and last episodes of a training session on the real rig. For a numerically simulated environment there would be no variance in the values, but for our real rig exchanging data with the RL agent through our communication software there is a large variance as shown in the figures.

Even though the communication code is far from perfect, it is encouraging that RL could still learn to optimise a reward function when learning directly on a real device, as discussed in Section 3.3.2.

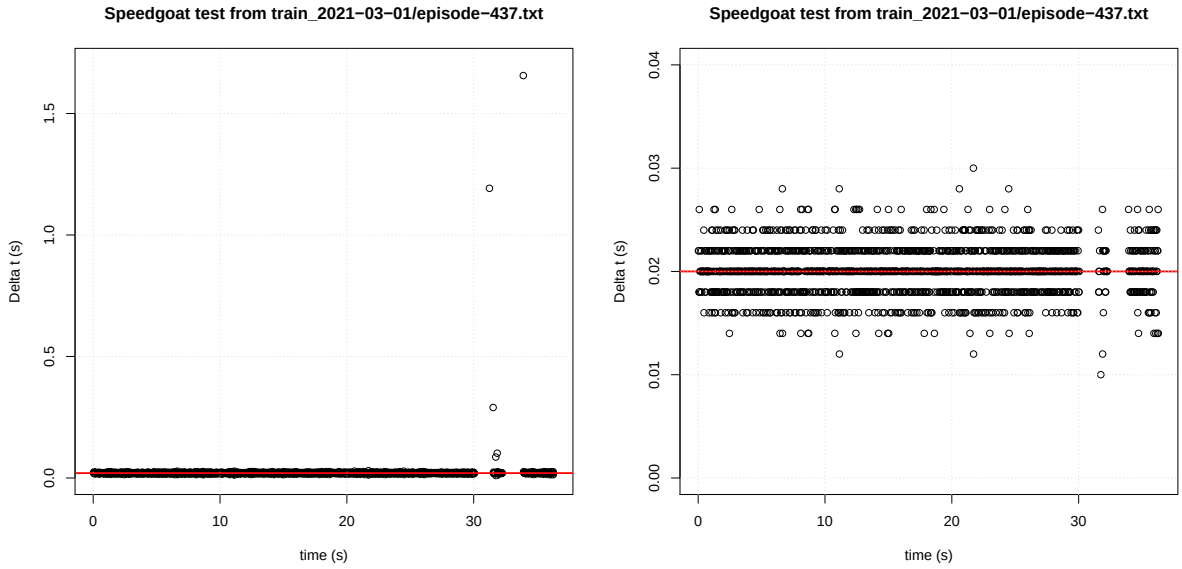

Figure 6: Scatter plots of actual simulation time step size between reported states and rewards during RL training using the communication software in the real RCP rig. The values should all be 0.02 s. Right: full view. Left: zoomed in view.

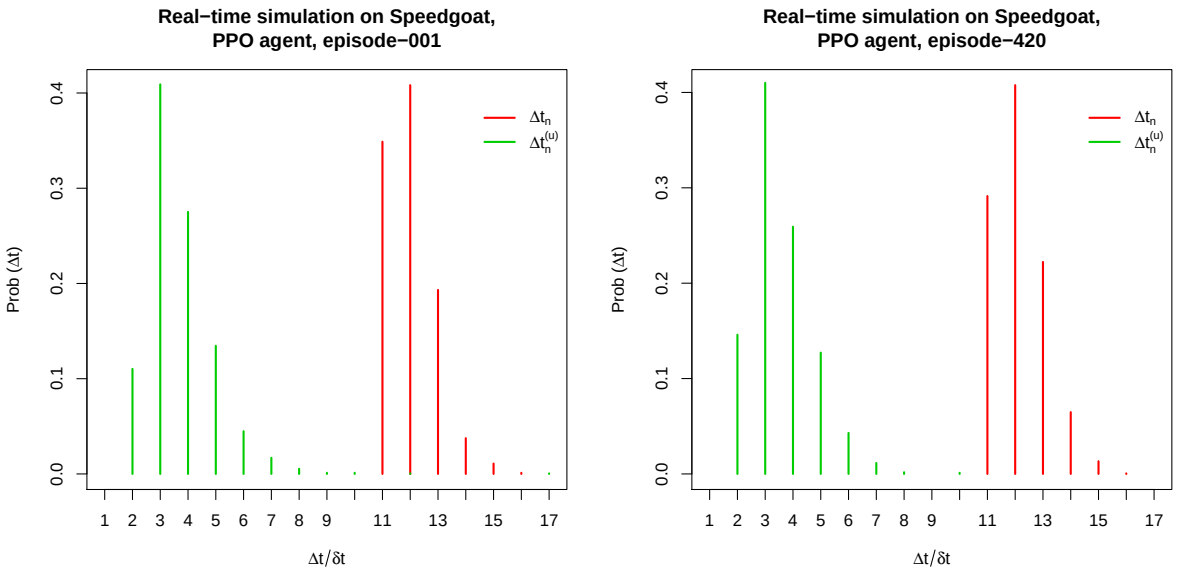

Figure 7: Probability masses for the duration of the learning time steps, Δt_n, and the unused time steps, $\Delta t_n^{(\mathrm{u})}$, as a function of the number of dynamic time steps, δt, they contain. The data is from the first and last episodes of an RL training session using the an idealised Simulink environment running on the real-time target computer.

3.5. On-Policy RL In Simulations

3.5.1. Idealised WEC in Python with Stable Baselines3

Most of the on-policy RL-related code development throughout Stage 3 was initially done using the Stable Baselines3 (SB3) [6] RL codebase applied to a custom idealised WEC environment written in Python. When satisfactory progress was made in this environment the newly developed methods where applied to

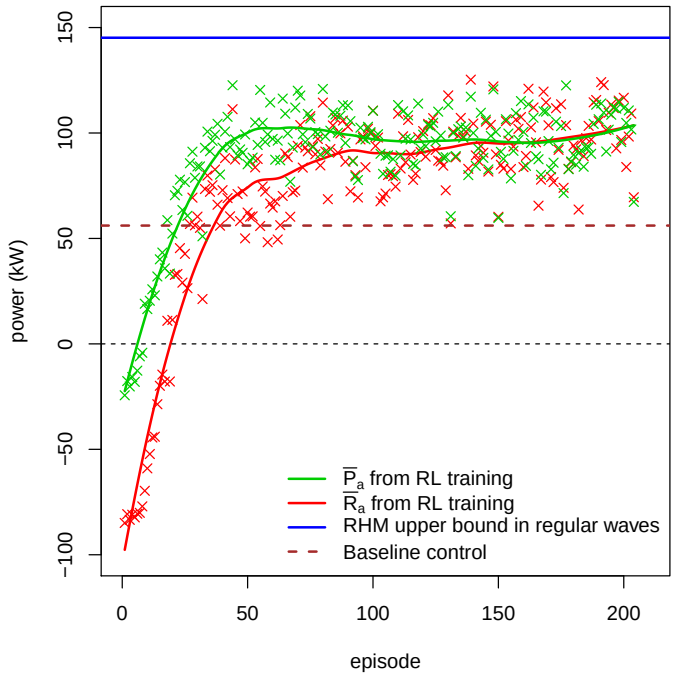

Figure 8: Learning curves for $\overline{P}_a$ and $\overline{R}_a$ as a function of the number of training episodes for a WEC of size of $(r_0, d_0) = (1.75\,\text{m}, 5\,\text{m})$ operating in PM sea states with $T_p = 10.5$ s. Each episode is 128 s long at full scale. The dashed brown line at $\overline{P}_a = 56.1$ kW is the mean power from the baseline control using the best constant values of control spring and damping. The blue line at $\overline{P}_a = 145.2$ kW is the mean power absorbed from regular waves with the same energy flux as irregular waves generated from the PM spectra with $T_p = 10.5$ s.

our Simulink and Simscape models that used on-policy RL agents from MathWorks RL Toolbox [4]. Early in Stage 3 we thoroughly checked that the different idealised WEC environments (Python and Simulink) yielded identical results, however, by the end of Stage 3 it seems that our idealised WEC environments had diverged somewhat as they gave slightly different results for experiments that should have yielded the same results. In this section we summarise the results from the Stable Baselines3 [6] codebase applied to an idealised WEC environment written in Python.

Figure 8 shows learning curves for mean absorbed power, $\overline{P}_a$, and mean return, $\overline{R}_a$, as a function of the number of training episodes for a WEC of size of $(r_0, d_0) = (1.75\,\text{m}, 5\,\text{m})$ operating in PM sea states with $T_p = 10.5$ s. Each episode is 128 s long at full scale. The dashed brown line at $\overline{P}_a = 56.1$ kW is the mean power from the baseline control using the best constant values of control spring and damping. The blue line at $\overline{P}_a = 145.2$ kW is the mean power absorbed from regular waves of the same energy flux as irregular waves generated from the PM spectra with $T_p = 10.5$ s.

The training occurred for 200 episodes, which equates to a training time of 200×128 seconds, or 7.1 hours, at full scale. At the end of Stage 2, the training time for such an experiment would have been about 5 days at full scale. This experiment included measures that prevented the free-energy problem and a continuity cost that penalised discontinuities in the control forces, and therefore encouraged smooth control forces.

We computed results for different maximum PTO control forces, F_{max}. The results are summarised in Table 2. They show that, for the same maximum PTO load, RL can find control policies that give 83% more power than the baselines control. Similarly, when constrained to use only 61% of the maximum PTO load used by the baseline control, RL can find control policies that give 18% more power. These two results correspond to the first two bullet points stated in the original objectives in Section 1, however, they have only been demonstrated satisfactorily in the Python simulations of an idealised WEC. Attempts were not made to run these experiments in the Simulink environment or on the RCP rig directly.

On-Policy Power Full-Scale, Python Idealised WEC				
Control	$F_{\max}$ (MN)	$\overline{P}_a$ (kW)	$\overline{R}_a$ (kW)	Power uplift (%)
baseline	0.327	56.1	55.4	—
learnt	0.425	102.0	104.6	81.8
learnt	0.327	102.6	103.9	82.9
learnt	0.200	66.3	79.5	18.2

Table 2: Comparisons of full-scale mean powers, $\overline{P}_a$, and mean returns, $\overline{R}_a$, from the baseline control and the learnt controls with different limits on the maximum PTO control forces, $F_{\max}$.

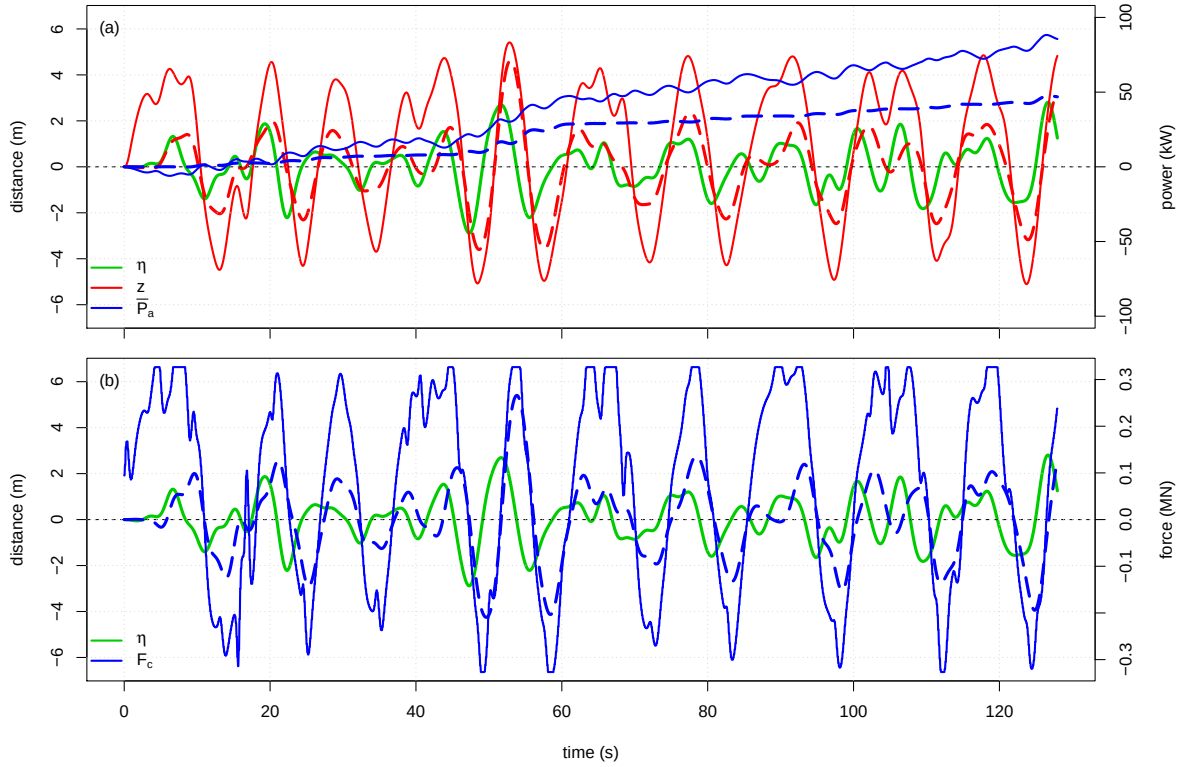

Figure 9: Examples of time series plots from the baseline control (using the best constant values of control spring and damping) and the learnt control policy $\pi_\theta(s_t)$ for a WEC of size of $(r_0,\, d_0) = (1.75\,\text{m},\, 5\,\text{m})$ operating in PM sea states with $T_p = 10.5$ s. In both plots the green line is the surface elevation, $\eta(t)$. Upper: the red lines are the heave positions, $z(t)$, for the baseline (dashed line) and the learnt control (solid line), the blue lines represent the total energy absorbed to time t divided by the full 128 s of the episode so the values at $t = 128$ s are the mean powers of the baseline control (dashed line) and the learnt control (solid line). Lower: the blue lines are the control forces, $F_c(t)$, for the baseline (dashed line) and the learnt control (solid line).

Figure 9 shows examples of time series of the motion of the WEC when controlled by the baseline control and the learnt policy $\pi_\theta(s_t)$. Examination of these plots can clarify why the learnt policy performs significantly better than the baseline control. In both the subfigures, dashed lines are used for variables from the baseline control, whereas solid lines are used for variables from learnt policy $\pi_\theta(s_t)$. Lines that are the same in both cases, i.e., η and F_c, are drawn as solid.

In Subfigure 9(a), the wave elevation, η, is plotted in green and the WEC heave, z, in red. The axis on the left-hand side gives the distance scale for these variables; that on the right-hand side gives the power scale for the mean power, $\overline{P}_a$, which is plotted in blue. Note that $\overline{P}_a$ is plotted as the mean of the absorbed energy over the 128 s of the episode, not the mean computed to time t. This way of plotting has the following advantages: (i), the mean computed to time t would fluctuate wildly at low times; (ii), the mean value of absorbed power computed over the whole episode can be read directly from the plot

On-Policy Power
1/15 Model Scale, Simscape

Environment	Baseline (W)	RL (W)	Uplift (%)
Simscape Idealised WEC	5.74	6.78	18.1
Simscape Realistic WEC	5.01	5.40	7.78

Table 3: Comparisons of rig-scale mean powers for the baseline control and control policies learnt by on-policy RL for both idealised and realistic WECs modelled by Simulink. The realistic WEC included viscous drag forces, non-linear mooring forces, as well as realistic delays and noise in the state vector, the action vector, and the reward scalar.

as the value at $t = 128$ s; (iii), the unit of the axis label is power, which allows for more convenient and easier comparison than if total absorbed energy had been used. In the example plot, we can see that $\overline{P}_a$ for the baseline control is slightly less than 50 kW, whereas that for learnt policy is slightly less than 100 kW. The plot also illustrates that using the learnt control causes far larger heave motions than the baseline control. Higher heave, at the same period, means higher velocities, and therefore, given the similar phases, more energy is absorbed during each oscillation of the WEC.

Subfigure 9(b) shows the wave elevation, η, and the control force, F_c. The learnt control policy is producing control forces that are consistently high throughout the episode, whereas the baseline control gives a relatively high peak control force at around $t = 55$ s and relatively low peak values at other times. The learnt control policy applies its maximum control force of $|F_c| = 0.327$ MN at many points in the time series.

3.5.2. Realistic WEC in Simscape with MathWorks RL Toolbox

The Simscape environment was used to model both idealised and realistic WECs, and the on-policy RL agents were provided by MathWorks RL Toolbox [4]. The difference between the realistic and idealised WEC models was that the realistic model included viscous drag forces, non-linear mooring forces, as well as realistic delays and noise in the state vector, the action vector, and the reward scalar. The noise and delays are "realistic" in the sense they were chosen based on values measured from the real RCP rig. The results are summarised in Table 3. There is a large difference between the power uplifts for the idealised cases reported in Tables 3 (82.9%) and 3 (18.1%). This suggests that the environment codes are different, or that the RL agents are not performing equally well. For the realistic WEC modelled by Simulink, RL gives only a small uplift of about 18.1% in mean power over the baseline control.

3.6. Offline RL in Realistic Simulations with d3rlpy

It was in a deliverable from the Offline RL Workstream, that we first switched from using control spring and damping as the action vector, i.e., $a = [k_c, b_c]$, to using the control force, i.e., $a_t = [F_{c,t}]$. The change was necessary for offline learning because RL could not learn how to improve $a = [k_c, b_c]$ from data generated by the best known control policy (also called the best known behaviour policy) if that behaviour policy always applied the same action (i.e., the optimal constant values of k_c and b_c) as this meant that the generated data would include only state transitions and rewards associated with a single action.

We experimented with offline datasets produced from data generated by five different combinations of behaviour policy, including the best known control policy which was the baseline control with the best constant spring and damping values. Of the five datasets, we found one that enabled offline RL to learn policies that gave significant improvements over the baseline control policy (i.e., 69% more power) and four that showed no improvement. On one hand, this result was encouraging because it was the first time we demonstrated successful offline learning on a dataset from a WEC environment; on the other hand it was discouraging because our attempts at offline RL did not succeed on data from the best known policy alone. This is discouraging because the purpose of offline RL is to learn from large historical datasets, and the only large volumes of data likely to exist are those from running the best known control policies

on particular WEC devices. For offline RL to work on our WEC environment, we had to supplement data from the best known policy with data from a random policy. This is not ideal because large quantities of data from a random policy will not already exist, so they would have to be created, which would defeat the object of offline RL. This is an area that would benefit from further research.

As part of the offline RL workstream we had planned to test offline RL on data gathered from the real RCP rig, however, because we could not fix the free-energy problem on the rig (see Section 3.3.2) and other reasons such as running out of time to gather and process large quantities of data from the rig, we instead used data generated from our realistic Simscape model of the rig. For the realistic Simscape model we had a working solution to the free-energy problem, and we could generate data far more quickly than we could from running the real rig in real time. The work on applying offline RL to the realistic Simscape model was carried out in two stages, with an important difference that in the first stage we allowed ourselves access to the environment while the offline RL was training policies from offline data. This meant that we could evaluate the learnt policies at regular intervals during training to assess the quality of the policies. This option to evaluate polices during training is useful because of a phenomenon in RL called policy-collapse, which is where an RL agent can gradually learn a good control policy, only for the quality of that policy to suddenly collapse as training continues. The only way to be certain that policy collapse has occurred is to evaluate the policy by applying it to the environment. However, when applying offline learning to a real WEC it would not be possible to test learnt policies during the training, therefore, for a realistic implementation of an offline RL process we applied this same restriction of not allowing ourselves access to the environment to evaluate policies during training.

This meant that we had to assess the quality of different policies without running the policies in the environment. Offline RL is implemented like a supervised learning problem, where the data is split into a train dataset and a test dataset. Like in supervised learning, a model is fitted to the train data and evaluated on the test data. To aid offline policy assessment, the process of running CQL from d3rlpy outputs a number of performance metrics that can be used to estimate the performance of a policy without testing the policy on the environment. There are two types of metrics that can be used for offline evaluation: those associated with the training of the neural networks to fit the offline train data, and those associated with the trained policy to assess how well it performs on the portion of the offline data that is reserved for testing.

To gain experience on how to interpret these new metrics from offline RL, we applied offline RL to the idealised WEC environment while allowing ourselves unrestricted access to the environment for evaluation purposes. Once we felt that we had an adequate understanding of the new offline RL metrics we used that knowledge to access the results of offline RL applied to the realistic Simscape data. This enabled us to choose policies from the offline learning process that we thought would work well in the realistic Simscape environment before we performed any checks of those policies on the Simscape environment.

While working on the offline RL workstream, it became clear that the research cycle associated with changing the WEC environment, or the parameters of RL algorithms, is far slower for offline RL than it is for online RL. For testing a change to an online RL system all one needs to do is make the change and rerun the experiment for about 200 episodes (which takes about 1 hour of simulation time); however, for offline RL one needs to perform the following five-stage process:

1. Create two datasets from the behaviour policies, each consisting of about 1000 episodes (which takes 10 hours of simulation time).
2. Combine the two datasets to create the replay buffer for offline learning.
3. Train a model (which takes about 3 hours of computer time).
4. Perform a manual analysis of the new metrics for policy evaluation in offline RL.
5. Evaluate the trained model on the target environment (which, unlike for on-policy RL, is a separate process from training).

The results from applying these stages are summarised in Table 4. We regard the 14.4% uplift in power as a good result since it was the first time we had applied offline RL in the same "blind" manner as it would need to be applied to a real WEC, and the offline policy evaluation methods seemed to work reasonably well.

Offline RL, Power
1/15 Model Scale, Simscape

Environment	Baseline (W)	RL (W)	Uplift (%)
Python Idealised WEC	4.23	7.13	68.5
Simscape Realistic WEC	5.55	6.35	14.4

Table 4: Comparisons of rig-scale mean powers for the baseline control and control policies learnt by offline RL for the realistic WEC modelled by Simulink. The realistic WEC included delays and noise in the state vector, action vector, and the reward scalar. It also included viscous damping forces and non-linear mooring forces.

4. Conclusions

In Stage 3 of the CEORL project we have gained valuable understanding and made significant progress that will provide a strong foundation for the continuation of the project.

The most significant advancements made during Stage 3 were:

- Very large reductions in training time for on-policy RL (by a factor of about 16), which make training directly in real-time on a real devices a feasible option (model-scale or full-scale).
- Implemented Matlab code to learn directly on a real environment (as opposed to a simulation), albeit with some communication problems that hindered learning.
- In simulations:
  - Demonstrated the two main objectives for Stage 3, i.e., that RL can give more power for the same force, and use less force to provide the same power.
  - Identified and fixed the free-energy problem in simulations.
  - Designed a number of reward functions that yield control policies that give relatively smooth control forces, however, the same functions did not work on the real rig because of unavoidable delays in the PID control of the actuator (causing repeated overshooting and undershooting of the demand position).
  - Demonstrating that offline RL can learn better policies than those used to generate the offline data, albeit using data that is unlikely to be available in large quantities prior to training. This is not to say that offline RL could not learn from the type of data that might be available from previous WEC deployments, just that we did not succeed in showing that it could. More research would need to be done in this area.
- In the real RCP rig:
  - Identified the likely cause of the free-energy problem and have a good candidate to fix the problem (i.e., change the actuator from running with position demand to running with force demand).
  - Demonstrated that on-policy RL can learn safely while acting within the constraints of a supervisory control that prevents RL implementing its chosen actions if those actions could damage the hardware.
  - Demonstrated that on-policy RL can learn polices that gave high returns on real devices, albeit with RL taking advantage of artefacts of the simulation environment (in this case, the real RCP rig) that would not exist in the target environment (i.e., scale model in a wave tank).

Table 5 gives an overall summary of the important numbers from Stage 3. Despite the mixed outcome of the CEORL project so far, we maintain that RL can be applied successfully to real WECs. Rewards from WECs are not sparse so on-policy RL agents can easily and quickly learn improvements to the control policy by making many small incremental changes to the policy in the direction of those improvements.

Environment	Agent	RL Code	Baseline (W)	RL (W)	Uplift (%)	Notes
Python Idealised	PPO	SB3	4.29	7.85	83.0	
Real RCP Rig	PPO	SB3	0.92	1.15	25.0	Used PT, Inc. FE
Simscape Idealised	PPO	RL Toolbox	5.74	6.78	18.1	
Simscape Realistic	PPO	RL Toolbox	5.01	5.40	7.8	Noise scale 0.02
Real RCP Rig	PPO	RL Toolbox	2.15	4.71	119.1	Inc. FE
Python Idealised	CQL	d3rlpy	4.23	7.13	68.6	
Simscape Realistic	CQL	d3rlpy	5.55	6.35	14.4	Noise scale 0.01

Table 5: Summary of mean powers from baseline and learnt controls at 1/15 model scale. (Inc. = Includes, FE = free energy, PT = policy transfer)

However, regarding the problem of the sim-to-real gap, RL has an amazing ability to exploit flaws in any simulation model that is not an extremely good representation of the target environment. Applying RL to such models can give misleading results if RL learns to exploit these flaws. We have found examples of the adverse effects of the sim-to-real gap in both simulations performed by numerical models and simulations carried out on a real device such as our RCP rig. In both cases we have seen how RL does exactly what we asked of it, i.e., it finds control policies that give higher returns than the baseline control, but in both cases the resulting policies did not perform as we expected or as we desired because they exploited aspects of the simulated environment what would not be present in the real target environment. We expect that the sim-to-real gap could even be a problem when using RL to control a scale model in a real wave tank model (an example is that of RL learning to use reflections from the side of a narrow tank). When using RL in inadequate simulation environments, it can seem like RL finds ways to "cheat" to give the high returns it is seeking. This emphasises the importance of training directly on the real target environment. It is our opinion that RL could not "cheat" in this way if it was trained directly on the real target environment.

The demonstration that RL can learn in a few hours in WECs indicates that having RL learn directly on the real target environment of a WEC is feasible, and the observation that RL can exploit sim-to-real gaps in unexpected and unforeseen ways tells us that learning directly on the real target environment is preferable. We will, therefore, continue the CEORL project with the following next steps:

- Fix the free-energy problem in the real rig (the first thing to try is to switch the actuator from running in position demand to running in force demand).
- Develop hardware and software that will train using on-policy RL directly on a real WEC without the communications problems that hindered our efforts during Stage 3.

With these two improvements in place, we should be able to demonstrate RL learning directly on the real RCP rig without the free-energy problem, and show that it can learn control policies that outperform the best baseline control, as we see in simulations and as we hypothesise in Section 2.2.

A. List of Abbreviations

CEORL	Cost of Energy Optimised by Reinforcement Learning
COVID	Coronavirus disease 2019
CQL	Conservative Q-learning
HIL	Hardware in the loop
LCOE	Levelised cost of energy
LUT	Look-up table
PID	proportional–integral–derivative
PM	Pierson-Moskowitz
PPO	Proximal Policy Optimisation
PTO	Power take-off
RCP	Rapid control propotyping
RL	Reinforcement learning
ROI	Return on investment
SB3	Stable Baselines3
UK	United Kingdom
WEC	Wave energy converter
WES	Wave Energy Scotland

References

[1] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *CoRR*, abs/1901.08652, 2019. URL `http://arxiv.org/abs/1901.08652`.

[2] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning, 2020. URL `https://arxiv.org/abs/2006.04779`.

[3] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020. URL `https://arxiv.org/abs/2005.01643`.

[4] MathWorks. Reinforcement learning toolbox, 2020. URL `https://uk.mathworks.com/products/reinforcement-learning.html`.

[5] J. N. Newman. *Marine Hydrodynamics*. MIT Press, 1977. ISBN 0-262-14026-8 hardback.

[6] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, `https://stable-baselines3.readthedocs.io`, 2019.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

[8] Takuma Seno. d3rlpy: A data-driven deep reinforcement library as an out-of-the-box tool. `https://github.com/takuseno/d3rlpy`, `https://d3rlpy.readthedocs.io/en/latest/index.html`, 2020.

[9] Wave Energy Scotland. URL `https://www.waveenergyscotland.co.uk`.

[10] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. *CoRR*, abs/2009.13303, 2020. URL `https://arxiv.org/abs/2009.13303`.

CEORL

Wave Energy Scotland

Control Systems Stage 3 Follow-On Project

Reinforcement Learning System to Learn Directly on Real WECs

Public Summary Report

Paul Stansell

MaxSim Ltd

2023-03-30

Contents

1 Introduction

This project builds on the work carried out in Stage 3 of the CEORL project. The goal is to create a CEORL control system that WEC developers will easily be able to use with their current control hardware to apply reinforcement learning (RL) to learn directly on their real WEC devices. The real devices could be RCP rigs, scale models tested in wave tanks or full-scale WECs deployed at sea.

As documented in the conclusion of the public summary report from WES Stage 3 [1], the work in Stage 3 significantly reduced the time required for RL to learn good control policies in simulation. These shortened training times opened up the possibility of training RL agents directly on real hardware as opposed training on a simulation of that hardware. Training directly on real hardware has the great advantage of circumventing the problem of the sim-to-real gap. The sim-to-real gap makes it difficult to train a good policy in simulation that also works well on the real hardware. We wrote code to train directly on the real rig in Stage 3, however, our it suffered from communication delays that prevented the RL from performing well because there was a high variance in the times the actions would be applied to the real system. The method implemented in Stage 3 required the system to send a control action from the development computer to the real-time target controller, and also send the state and reward information from the real-time controller to the development computer, all in a single learning time step of about 0.02 s. Moreover, the control action was computed on the development computer and transmitted over TCP/IP to the controller running on the real-time target computer. Small delays in these data transfers prevented the actions from being applied at the correct times and in the correct states on the real rig, and this hindered the RL agent's ability to learn.

In this latest work we overcame this problem by using a different implementation of the communications and by computing the action locally on the real-time target computer, instead of remotely on the development computer, so that it is not affected by communication speed. This is done by sending a copy of the control policy parameters (i.e., the weights and bias of the neural network) to real-time target computer so the policy can be run there in real-time and perfectly synchronised with the states being read in real-time from the environment.

The second problem, documented in the conclusion of the public summary report from Stage 3, that relates to learning directly on the RCP rig was the problem of so-called "free-energy".[1] In this stage we have fixed this problem by changing the rig actuator so that it runs in force-demand mode instead of position-demand mode.

2 Implementing Remote RL in MATLAB and Simulink

This stage of the project was greatly facilitated by working closely consultants from The MathWorks. The outcome of the collaboration was MATLAB and Simulink code, nicknamed RLDeploy, that can be used to implement both on and off-policy RL algorithms from the MATLAB RL Toolbox to train control policies directly on CEORL's RCP rig[2].

2.1 Code for Remote Policy Updates and Rollouts

Part of RLDeploy consists of four Simulink blocks that perform the communications and other essential functions for RL. These blocks are compiled to run in real-time on a Raspberry Pi or a Speedgoat. They are:

1. A TCP/IP Server, listening on Port 50100, that sends rollout experiences to the a TCP/IP Client running on the development computer.
2. A TCP/IP Server, listening on Port 50101, that receives policy parameters (neural network weights and biases) from a TCP/IP Client running on the development computer.

[1] Recall that we gave the name "free-energy" to the energy that an RL-trained control policy could absorb by taking advantage of imperfections in our numerical models and our real RCP rig model that would not be present in the real WEC operating in real waves.

[2] For a diagram of the RCP rig, see Figure 1 in the public summary report from Stage 3, [1].

3. A block that uses the policy parameters to compute deterministic actions, and standard deviations of actions, from observations measured in real-time by the process running on the target computer (a Raspberry Pi or a Speedgoat).
4. A random number generator that adds noise, with the appropriate standard deviations, to the deterministic actions to enable the action exploration necessary for learning.

At the same time as the above code is running in real-time on the target computer, two TCP/IP Clients are running on the development computer. They listen on the same ports and send the latest policy parameters when they are updated, and receive the rollout experiences as they are made available by the environment.

2.2 Enhanced Code Functionality

We took the major rewrite of much of the code from Stage 3 as an opportunity to enhance the code with additional functionality that was not previously present. The code improvements are as follows:

1. The code is significantly simplified and fully modularised. It uses reference subsystems that can be reused in different models.
2. A Simulink model of the real rig has been written that allows all parts of the code to be tested entirely in simulation (the Stage 3 code could only run directly on the real rig via a Speedgoat).
3. Extensive use has been made of Simulink Variant Subsystems. The main options are:
    - Control variants
        - PI Control
        - RL using the RL Toolbox
        - RL over TCP/IP using RLDeploy
    - Environment variants
        - Simulink simulation of rig
            - RL from the RL Toolbox
                - Run on a development computer
            - RL over TCP/IP using RLDeploy
                - Run in real-time on a Raspberry Pi
                - Run in real-time on a Speedgoat
        - Real rig
            - RL over TCP/IP using RLDeploy run in real-time via a Speedgoat

The versatility of the options in 3. above allows for the following workflow:

1. Rapid development and testing of RL-related ideas at high computational speeds by running all the code entirely on a development computer.
2. Safe development and testing of real-time RL over TCP/IP using RLDeploy to control the rig simulation in real-time on a Raspberry Pi or a Speedgoat.
3. Development and testing of learning directly on the real rig hardware by running RL over TCP/IP using RLDeploy to control the real rig in real-time via a Speedgoat.

2.3 Running Both Rig Motors in Force Demand

The code from Stage 3 was changed to allow the actuator to operate in force-demand mode in response to the wave excitation forces being simulated by the code running of real-time computer. This removed the need for a PID controller to continuously move the actuator motor into the position computed by code running the WEC simulation. Previously, the use of a position-demand and PID controller resulted in the controller continuously overshooting and undershooting the target position by a very small amount that RL could learn exploit to gain "free energy". In this stage both the PTO and the actuator motors were operated in force-demand mode during the WEC simulation. One component of the force from each motor was the constant (equal and opposite) tension force from the tether. The ensure that the opposing forces from the PTO and the actuator balanced exactly at a specified position, a constant force was applied by the PTO motor and a hydrodynamic spring force (linearly proportional to actuator position) was applied by the actuator.

3 Applying RL in Simulations of the Real Rig

3.1 Effect of Rig Frictions

The values of the friction coefficients used in the simulation of the rig have a large affect on the returns. Figure 1 shows a comparison of the mean rate of return, mean(R/T), from simulations with different values of the friction coefficients used to model the friction in the motors of the rig. The changes in the friction parameters not only resulted in large changes to the mean rate of return, but they also changed the location of the (k_c, b_c) values for optimal PI control. Each grid-point value of mean(R/T) is the mean from the same random sea state, with a repeat period of 1000 s, that will be used on the real rig. The sea state was generated from a Pierson–Moskowitz spectrum with a peak period and significant wave height of $(H_s, T_e) = (0.294\text{ m}, 2.71\text{ s})$ at rig scale, or, at full scale, from 4.17 hours of sea state with $(H_s, T_e) = (4.41\text{ m}, 10.5\text{ s})$.

3.2 PPO RL Agent Parameter Tuning Performed in Simulations

Prior to testing RL on the real rig we systematically tuned the parameters of the PPO agent and its actor and critic networks. This parameter tuning was performed on the frictionless Simscape model that was developed during WES Stage 3 of the CEORL project. The outcome of the tuning process was a set of parameters that could yield about 30% more return than the optimal clipped PI control. The mean rate of return for optimal clipped PI control is 4.55 W and that for the best tuned RL control is 5.59 W. The results are shown in Figure 2. The optimal values are (k_c, b_c) = (−220 N/m, 50 N s/m). The two middle panels show the mean power per episode, and a histogram of mean powers per episode, for the optimal PI control. The two lower panels show the learning curve, and histogram histogram of mean powers per episode, for the best RL control policy.

4 Applying RL to Learn Directly on the Real Rig

4.1 Optimal Clipped PI Control on the Real Rig

Using the simulation results in Section 3.1 as a guide to the shape of the surface of mean return, mean(R/T), as a function of the PI control parameters (k_c, b_c), we ran individual tests on the real rig to try to locate the optimal value of (k_c, b_c) on the real rig. Each value of mean(R/T) is the mean from the same 1000 s of sea states used in Section 3.1. The individual results are plotted in Figure 3, along with the full surface of values from simulations. The simulation results are plotted as the green mesh, the real results are plotted as connected points with different colours (red, magenta, cyan and blue). By this

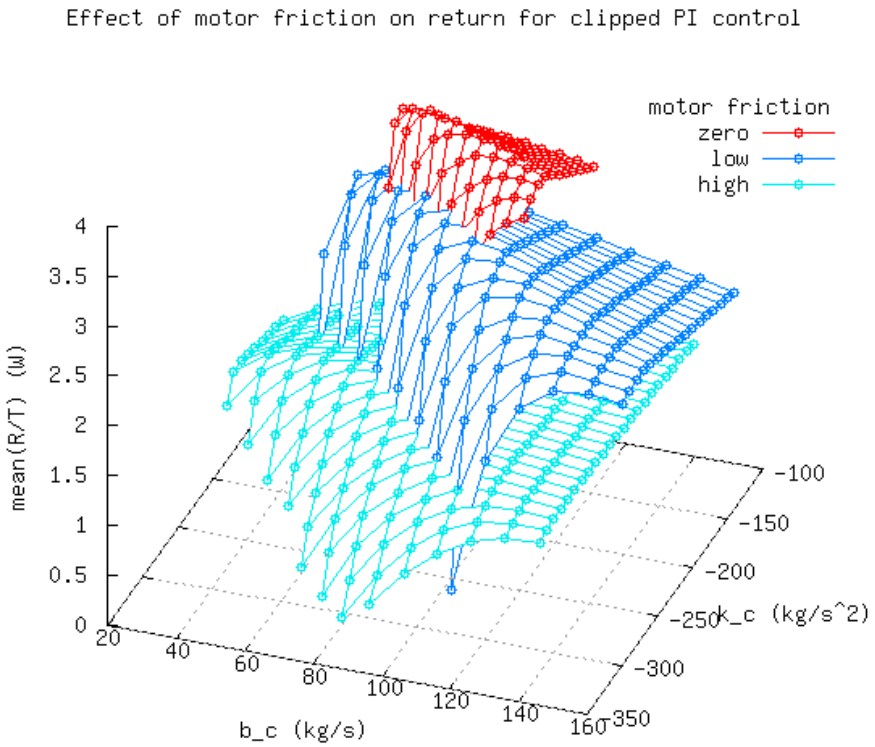

Figure 1: Plots showing the large effect changes in the friction coefficients have on the mean rates of return from the simulation of the rig. The values of the friction parameters for "zero", "low" and "high" friction show the effect of a range of frictions, with "high" being values indicative of the true friction in the rig's motors.

method we could the optimal PI control[3] on the real rig to be at $(k_c,\, b_c) = (-190 \pm 5\ \mathrm{N/m},\, 70 \pm 5\ \mathrm{N\,s/m})$ with a mean$(R/T) = 1.5183\,W$.

Some time-series plots of the optimal PI control are shown in Figures 4. The upper panel shows the control force applied by the PTO after the preload tension has been removed. The lower panel shows a detail of the time range $t = 155$ s to 180 s, which includes the point of maximum relative excursion. The lines plotted in the lower panel are the absolute heave excursion, $x_{3,\mathrm{act}}$, the water surface, η, and the maximum relative heave excursion, $x_{3,\mathrm{act}} - \eta$, along with horizontal lines marking the limits of the relative displacement above (or below) which a cost will be incurred during the optimisation of the control.

Comparing the plots from optimal PI control on the real rig in Figure 4 with those of optimal PI and RL controls for the Simscape simulation in 6, show the following three important differences:

1. On the real rig the magnitude of the allowed control force is insufficient for F_c to reach its maximum value at $F_{c,\max}$, above which it would be clipped.
2. On the real rig the WEC does not reach its volume limits as the relative heave displacement is always below the maximum relative displacement (`max_rel_disp`).
3. On the real rig the amplitude of oscillation, $x_{3,\mathrm{act}}$, is a lot less than that of the wave, η, whereas on the simulation of a frictionless ideal WEC the opposite is true.

[3] Actually, we found the $(k_c,\, b_c)$ location and mean(R/T) magnitude of the optimal PI control to depend on the temperature of the PTO motor, which in turn depended on the number of experiments that had been run prior to the test. The change in mean(R/T) was about 9%. The quoted value of mean$(R/T) = 1.5183\,W$ is the highest value measured when the PTO was relatively hot.

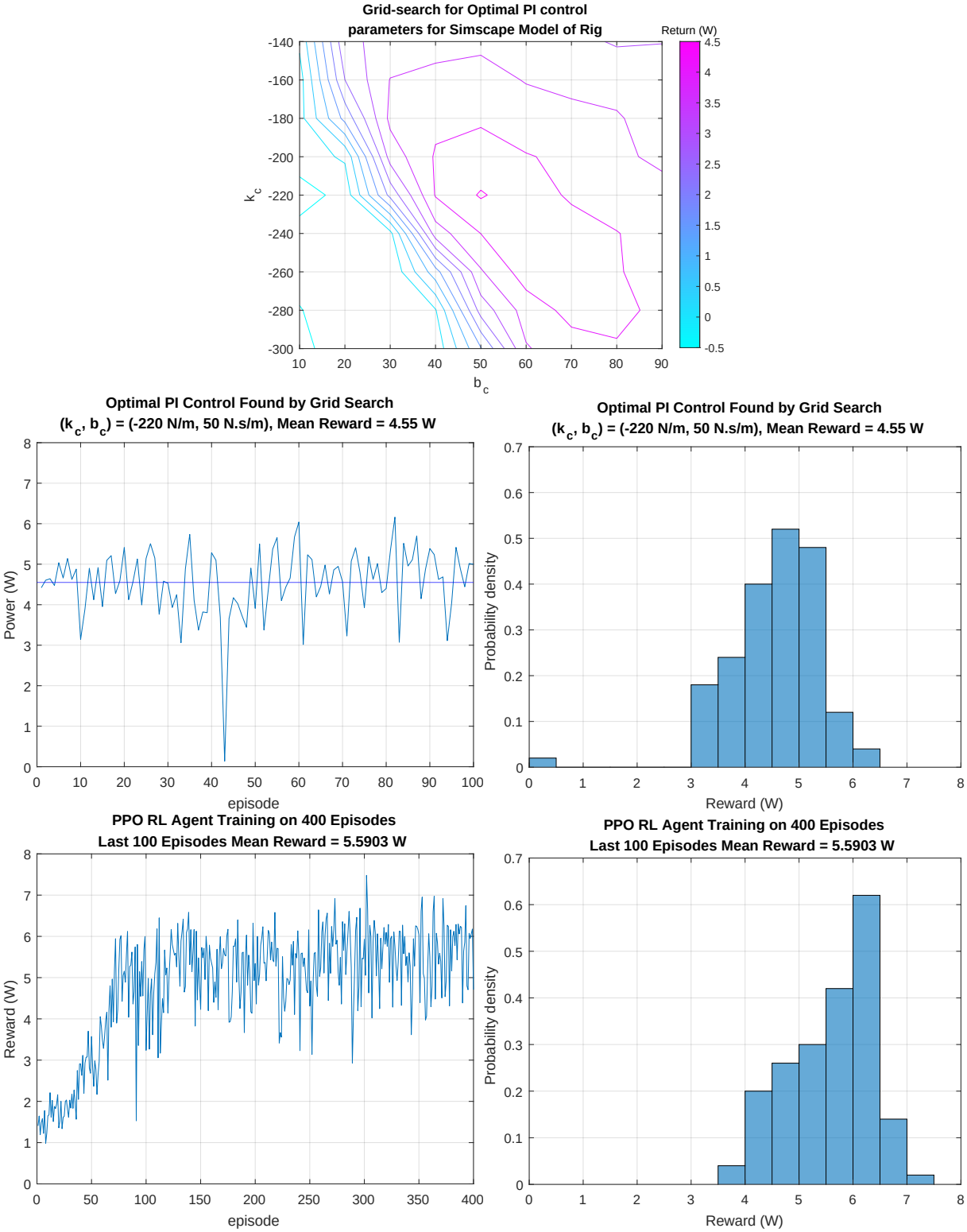

Figure 2: Optimal clipped PI and RL controls for Simscape model of rig developed during WES Stage 3 of the CEORL project. The WEC of size $(r_0, d_0) = (1.75\,\text{m}, 5\,\text{m})$ is operating in PM sea states with $T_p = 10.5$ s. Upper panel: contours of power from grid-search to find optimal clipped PI control parameters. Each point in the grid-search is the mean of 100 episodes, each of duration $T_{\text{episode}} = 128$ s at full-scale or $T_{\text{episode}} = 33.05$ s at model-scale. Middle panels: mean reward per episode and histogram of the reward per episode for optimal clipped PI control. Lower panels: Learning curve and histogram of the mean rewards per episode for best RL control policy.

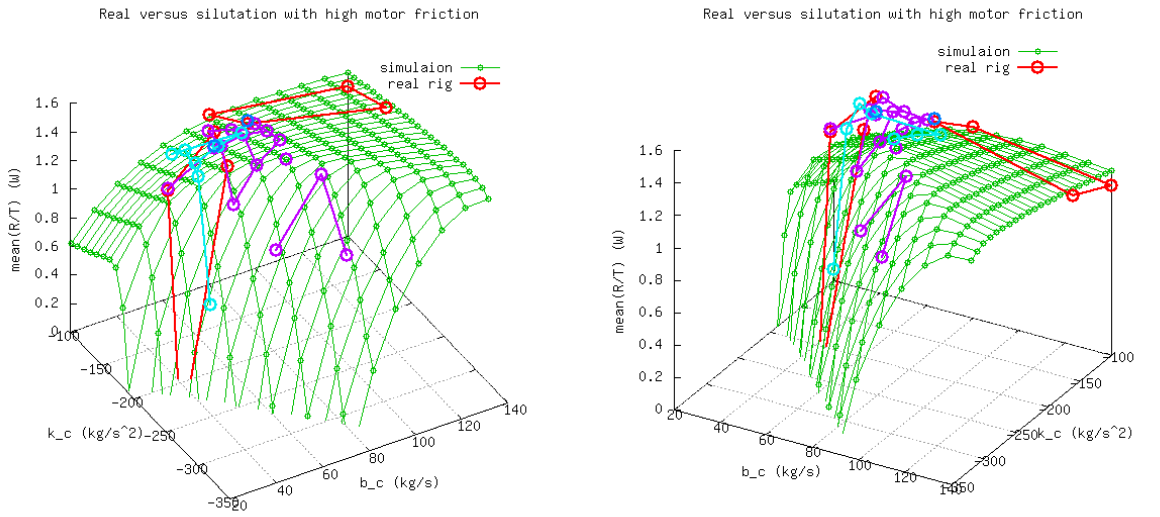

Figure 3: Two views of comparisons between the mean rates of return from "realistic", or "high friction", simulations of the rig and the real rig for clipped PI control. The simulation results are plotted in green and the real rig results are plotted in the other four colours (reg, cyan, magenta and blue), where each colour shows data gathered on different days in the lab.

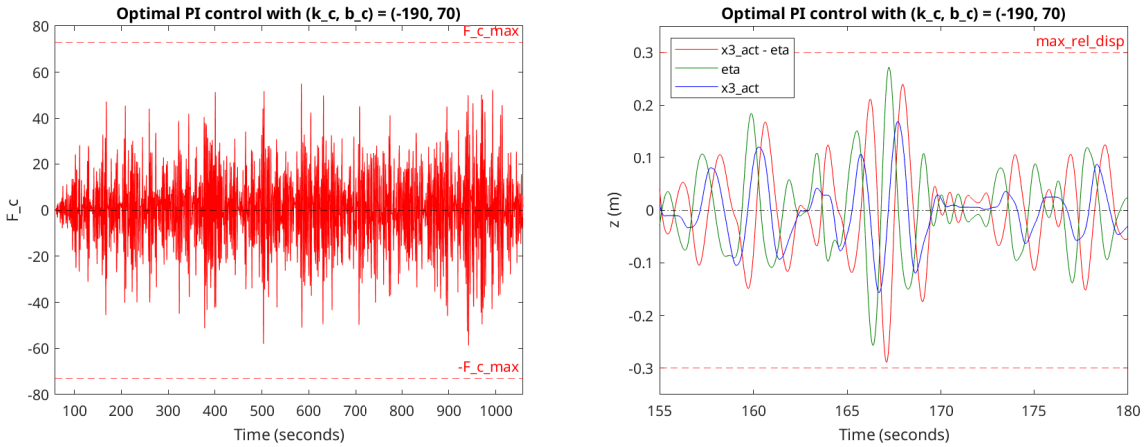

Figure 4: Plots of actuator, PTO and control forces on the real rig during the optimal PI run. The upper panel shows the control force, which is the force demand of the PI control policy. The lower panel shows the detail of a plot of the absolute heave excursion, $x_{3,\mathrm{act}}$, the water surface, η, and the maximum relative heave excursion, $x_{3,\mathrm{act}} - \eta$, along with horizontal lines marking the limits of the relative displacement above (or below) which a cost will be incurred during the optimisation of the control. The time range shows the point of maximum relative excursion.

4.2 RL Control on the Real Rig

This section reports on the results of using the RLDeploy code described in Section 2 to run RL on the real rig to train neural-network-based control policies. Unfortunately, the control policies learnt by RL did not produce as high a return as the optimal PI control found by grid-search. This is illustrated in Figure 5, which shows the time series of rewards from the optimal PI control and various RL tests. Also shown is a horizontal red line that marks the mean of the optimal PI control.

As indicated in the legend of the plot, the RL results include those being trained (i.e. with random exploration) and those that have been trained (i.e. without random exploration). There are also RL runs with and without the application of a low-pass filter to the RL actions, F_c. Applying a low-pass filter was

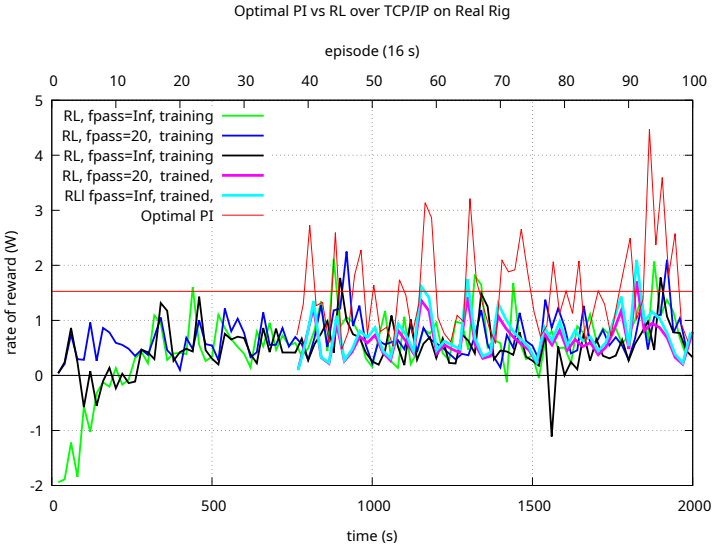

Figure 5: Time-series plots of the reward from the optimal PI control and various RL runs, including for policies being training (with exploration) and trained policies (without exploration). The plots include runs with and without a low-pass filter applied to the RL action to smooth the control force F_c. These low-passed variants are denoted by `fpass=Inf` and `fpass=20` in the legend, where `fpass=Inf` indicates no frequency-band filtering and `fpass=20` indicates low-pass filtering with cut-off frequency 20 rad/s.

tried because we feared that running the real rig with random force actions could damage the rig. For the first of these training runs the raw unfiltered values of the explorative actions, F_c, chosen randomly by the RL agent were applied to the rig, however, for the second run a first-order low-pass filter was applied to the actions to try to reduce the noise in F_c. These two variants are denoted by `fpass=Inf` and `fpass=20` in the legend, where `fpass=Inf` indicates no frequency-band filtering and `fpass=20` indicates the application of a low-pass filter with cut-off frequency 20 rad/s.

4.3 Comparison between Optimal PI and RL Control Policies

The left hand-side panel in Figure 6 shows comparisons between the cumulative reward with time for the optimal PI control and the training and trained policies for both `fpass=Inf` and `fpass=20`. As with Figure 5, it is evident that the optimal PI control (green solid line) produces by far the largest mean power. The left hand-side panel in Figure 6 shows a comparison between the instantaneous values of power, P_c, absorbed by the control force, F_c. From this plot it can be seen that the power from the optimal PI control (solid green line) regularly makes use of significant amounts of reactive power (i.e., when the instantaneous values of $P_c < 0$). However, neither of the trained RL policies are using significant amounts of reactive power. This means that they are learning control policies that have a damping component but no spring component.

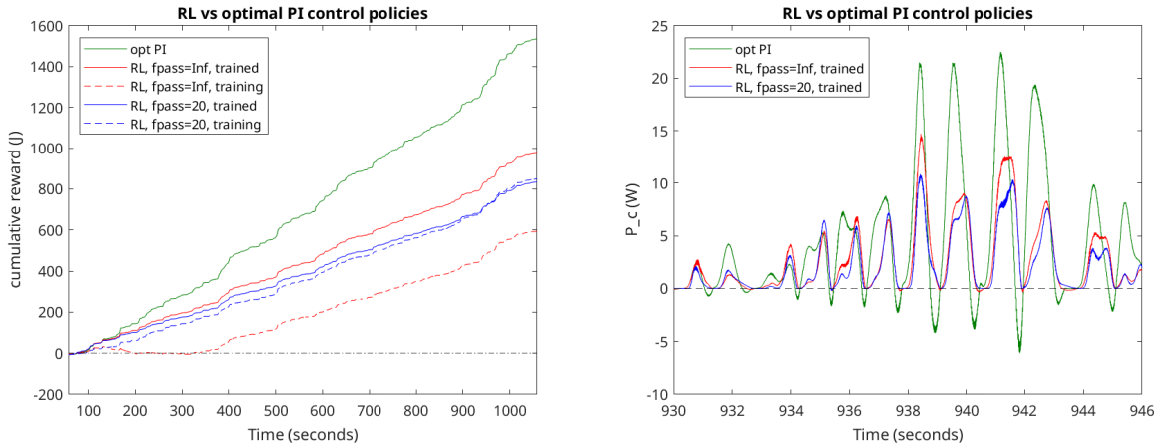

Figure 6: The left hand-side panel shows comparisons between the cumulative reward with time for the optimal PI control and the training and trained policies for both `fpass=Inf` and `fpass=20`. The right hand-side panel shows a detail of the instantaneous values power, P_c, absorbed by the control force, F_c.

5 Conclusions and Future Work

During this project we have implemented and tested code, called RLDeploy, that can use RL to learn directly on real hardware being controlled by a real-time host computer such as Speedgoat or a Raspberry Pi. RLDeploy is built to use the RL algorithms in the MATLAB RL Toolbox, which means all of the RL algorithms in the toolbox are available for use with RLDeploy. This includes on-policy RL agents such as PPO, and off-policy agents such as SAC.

The integration of RLDeploy, MATLAB and Simulink, along with a real-time host computer, such as Speedgoat or a Raspberry Pi, or any of the many embedded targets that are supported by Simulink Coder[4], provides a system with the following four options for a rapid and flexible product development and testing:

Option	RL	Environment	Running on
1	RL Toolbox	Simscape model	development computer
2	RL Toolbox	Simulink model	development computer
3	RLDeploy	Simulink model	real-time computer
4	RLDeploy	Real hardware	real-time computer

The main advantages of these options are faster and safer development and testing of the software and hardware. Options 1 and 2 allow for the rapid development and testing of RL-related ideas in simulations that run quickly and safely on the development computer. Option 3 allows for testing of the TCP/IP communications, essential for leaning on a real device with RLDeploy, in a safe simulated real-time environment. Finally, after one has gained confidence from experiments run in simulations, one can use Option 4 to safely run the same experiments on the real rig, with the assurance that the code has been comprehensively tested in simulation.

The biggest unsolved problem is why, compared with the optimal PI control, we see a 30% increase in power when RL is applied to the Simscape model from WES Stage 3, but we see a 50% decrease in power when RL is applied to the real rig. We have shown that RL can find better control policies than the optimal PI control policy in frictionless simulations, but we have not shown that the same is true in a real rig that has large amounts of friction. We will continue to work on solving the problems that remain, the main one of which is to develop algorithms for smooth exploration in RL. Smooth exploration could be a crucial requirement for an RL agent is tasked to extract as much energy as possible from an environment with high friction. This is because the act of exploring, which is required to learn good control policies, will use energy and therefore reduce the return. This could hinder learning and reduce the effectiveness of RL applied to these types of problems.

[4]Supported embedded real-time target computers can run the C/C++ code that is generated by Simulink Coder.

References

[1] Paul Stansell. Public Summary Report. Technical Report WES-CSC-S3-D33, MaxSim Ltd, Edinburgh, UK, September 2021.