

# POLITECNICO DI TORINO

Master's Degree in Mechatronics Engineering



Master's Degree Thesis

## On the viability and effectiveness of Reinforcement Learning techniques for Wave-Energy converter control.

Supervisors

Prof. Giuliana MATTIAZZO

Prof. Edoardo PASTA

Prof. Sergej A. SIRIGU

Candidate

Leonardo GAMBARELLI

October 2022



# Summary

The aim of this essay is that of investigating if it is possible to apply Reinforcement Learning (RL) methods for controlling Wave Energy Converters (WEC) devices and how they can actually improve making more viable Wave Energy production in general.

At first, a review of the available literature about the reasons behind the WEC concept is presented along with the main types of devices used with their working principle. Moreover, their main points of strenght will be highlighted along with the reasons why this technology is still not commercially viable.

Successively, there will be an introduction to RL, a field of Machine Learning theory which overlaps with Control System theory. The most used RL algorithms will be presented and their suitability for WEC control will be pointed out.

After having defined the main RL algorithms, various control codes on MATLAB, based on different RL methods, will be developed at first for a simple ideal Mass-Spring-Damper system.

From this ideal case it will be possible to pass to real WEC devices by introducing some changes.

In order to simulate the environment and have a ready-to-be imported control scheme, Simulink software will be used jointly with MATLAB.

After having developed a control code for each of the main RL methods for WEC, they will be compared, highlighting the weaknesses and strenghts of each method as which of them is more suited in which case.

# Acknowledgements

To my family and friends.



# Table of Contents

<b>List of Tables</b>	VII
<b>List of Figures</b>	VIII
<b>Acronyms</b>	XII
<b>1 An Introduction to Wave Energy Converters</b>	<b>1</b>
1.1 Reasons behind recent interest in WECs . . . . .	1
1.2 Main types and working principles of WECs . . . . .	2
1.3 Wave Conditions . . . . .	6
1.4 Hydrodynamical modelling of the WEC and of the WSI . . . . .	8
<b>2 Traditional control of WECs</b>	<b>14</b>
2.1 History of WECs control . . . . .	14
2.2 Resistive control . . . . .	15
2.3 Reactive control . . . . .	16
2.4 Model Predictive Control . . . . .	17
2.5 Discrete Control-Latching and Declutching . . . . .	19
<b>3 Reinforcement Learning Algorithms</b>	<b>21</b>
3.1 Introduction to Reinforcement Learning . . . . .	21
3.2 Markov Decision Process . . . . .	23
3.3 Exploration Strategy . . . . .	24
3.4 Monte-Carlo methods . . . . .	25
3.5 Temporal Difference methods . . . . .	27
3.5.1 Sarsa . . . . .	28
3.5.2 Q-learning . . . . .	30
3.5.3 Function Approximation . . . . .	30
3.5.4 Sarsa with LFA . . . . .	33
3.5.5 Q-learning with LFA . . . . .	34
3.5.6 Neural Fitted Q-iteration . . . . .	35

3.5.7	Least Square Policy Iteration . . . . .	36
<b>4</b>	<b>Implementation of Reinforcement Learning algorithms on MAT-Lab</b>	<b>43</b>
4.1	MSD model: Off-line versions of the algorithms . . . . .	44
4.1.1	Off-line Q-learning . . . . .	52
4.1.2	Off-line Sarsa . . . . .	70
4.2	MSD model: On-line version of Q-learning . . . . .	72
4.2.1	Learning Parameters . . . . .	76
4.2.2	Results . . . . .	76
4.3	PeWEC model . . . . .	82
4.3.1	Control Variables . . . . .	85
4.3.2	Sea states . . . . .	85
4.3.3	Offline Q-Learning . . . . .	89
4.3.4	Online Q-Learning . . . . .	92
4.3.5	Towards irregular waves . . . . .	106
<b>5</b>	<b>Conclusions</b>	<b>109</b>
	<b>Bibliography</b>	<b>110</b>

# List of Tables

4.1	First parameters setting of interest . . . . .	59
4.2	Second parameters setting of interest . . . . .	59
4.3	Third parameters setting of interest . . . . .	71
4.4	Example of a parameter settings which explores very little and converge soon after initialization . . . . .	77
4.5	Example of a parameter settings which actually explores before convergence . . . . .	77
4.6	Another example of a parameter settings which actually explores before convergence, even if explores for a little time, and just around the convergence point. . . . .	78



# List of Figures

1.1	Typical shapes for one-body PAs. Taken from [7]	3
1.2	Typical shapes for two-bodies PAs. Taken from [7]	3
1.3	Typical types of OWCs. Taken from [7]	4
1.4	Typical types of Attenuators. Taken from [7]	4
1.5	Typical types of Terminators. Taken from [7]	5
1.6	Example of a scatter table. Taken from [7]	8
1.7	Schematic representation of the possible modelling choices. Taken from [7]	9
1.8	Example of a graph representing the efficiency variation of a PTO. Taken from [7]	11
1.9	PTO working scheme for the 4 main types. Taken from [13]	12
1.10	WEC operating modes with respect to sea state. Taken from [14]	12
2.1	Choice of a sub-optimal action which does not violate constraints. Taken from [25]	18
2.2	Action principle of latching (a) and declutching (b). Taken from [20]	19
2.3	Example of the evolution of the system variables in a latching control. Taken from [20]	19
3.1	General diagram of a RL process. Taken from [26]	22
3.2	Policy iteration in MC. Taken from [26]	26
3.3	Example of activation features of radial basis function for 2 dimensions. Taken from [19]	32
3.4	Example feed-forward NN used for function approximation of the Q function (with 1 state and 1 action only). Taken from [19]	33
3.5	Representation of the scheme of a Policy Iteration algorithm. Taken from [19]	38
3.6	Representation of the scheme of an LSPI. Taken from [19]	41
4.1	MSD model. Taken from Wikipedia	44
4.2	Simulink scheme used for simulating the system	46

4.3	Exciting wave function in time . . . . .	47
4.4	Power function in time example. Particular configuration with average power negative (the WEC gives energy to the sea) . . . . .	48
4.5	Power function with 1 as discrete step . . . . .	49
4.6	Power function with 1 as discrete step, zoomed in the central section	50
4.7	Power function with 5 as discrete step . . . . .	51
4.8	Particular combinations where the power is given from the WEC to the sea . . . . .	52
4.9	Flowchart of the action selection process . . . . .	54
4.10	Effect of the odd elevating power in the range $[-1,1]$ of the domain .	55
4.11	Policy obtained just from the preliminary initialization . . . . .	57
4.12	Actual values and representing action of the colours . . . . .	58
4.13	Flow chart of the offline Q-learning algorithm . . . . .	60
4.14	First policy obtained, with average number of visits per state of 6.59	61
4.15	Second policy obtained, with average number of visits per state of 7.14 . . . . .	62
4.16	Third policy obtained, with average number of visits per state of 7.69	63
4.17	First policy obtained, with parameters in Table 4.1, with 8.59 number of average visits. . . . .	64
4.18	Second policy obtained, with parameters in Table 4.1, with 11.53 number of average visits. . . . .	65
4.19	Third policy obtained, with parameters in Table 4.1, with 6.59 number of average visits. . . . .	66
4.20	First policy obtained, with parameters in Table 4.1, with 13.73 number of average visits. . . . .	67
4.21	Second policy obtained, with parameters in Table 4.1, with 12.08 number of average visits. . . . .	68
4.22	Third policy obtained, with parameters in Table 4.1, with 10.98 number of average visits. . . . .	69
4.23	Values of the power in the old optimal and in the 2 points nearby.	70
4.24	Policy obtained with the parameters in table 4.3, with average number of visits of 17.58 . . . . .	71
4.25	Policy obtained with the parameters in table 4.2, with average number of visits of 15.38 . . . . .	72
4.26	Simulink scheme used to simulate the system . . . . .	73
4.27	Workflow of the MATLAB function, which is run at each time step .	75
4.28	Control damping time evolution, with the parameters in table 4.4 .	77
4.29	Control stiffness time evolution, with the parameters in table 4.4 . .	78
4.30	Control damping time evolution, with the parameters in table 4.5 .	79
4.31	Control stiffness time evolution, with the parameters in table 4.5 . .	80
4.32	Control damping time evolution, with the parameters in table 4.6 .	81

4.33	Control stiffness time evolution, with the parameters in table 4.6 . .	82
4.34	PeWEC Scheme. Taken from [45] . . . . .	83
4.35	PeWEC graphical representation. Taken from [46] . . . . .	84
4.36	Simulink scheme used to simulate the PeWEC. . . . .	85
4.37	Powermatrix of the sea state 1. X axis represents control stiffness, Y axis represents control damping. . . . .	86
4.38	Powermatrix of the sea state 2. . . . .	87
4.39	Detail of the powermatrix of the sea state 2. . . . .	88
4.40	Power matrix of the sea state 3. . . . .	89
4.41	Policy obtained for the sea state 1, with an average number of visits per state of 11.24. . . . .	90
4.42	Policy obtained for the sea state 2, with an average number of visits per state of 7.10. . . . .	91
4.43	Policy obtained for the sea state 3, with an average number of visits per state of 7.37. . . . .	92
4.44	Simulink model used to simulate the system. . . . .	93
4.45	Control damping obtained with set 1 and sea state 1 (T=5) . . . .	95
4.46	Control stiffness obtained with set 1 and sea state 1 (T=5) . . . .	96
4.47	Control damping obtained with set 1 and sea state 2 (T=6) . . . .	97
4.48	Control stiffness obtained with set 1 and sea state 2 (T=6) . . . .	98
4.49	Control damping obtained with set 1 and sea state 3 (T=7) . . . .	99
4.50	Control stiffness obtained with set 1 and sea state 3 (T=7) . . . .	100
4.51	Control damping obtained with set 5 and sea state 1 (T=5) . . . .	101
4.52	Control stiffness obtained with set 5 and sea state 1 (T=5) . . . .	102
4.53	Control damping obtained with set 5 and sea state 2 (T=6) . . . .	103
4.54	Control stiffness obtained with set 5 and sea state 2 (T=6) . . . .	104
4.55	Control damping obtained with set 5 and sea state 3 (T=7) . . . .	105
4.56	Control stiffness obtained with set 5 and sea state 3 (T=7) . . . .	106
4.57	Example of an irregular wave . . . . .	107
4.58	Content in frequency of the wave . . . . .	108



# Acronyms

**RL**

Reinforcement Learning

**ML**

Machine Learning

**WEC**

Wave-Energy Converter

**LPFT**

Linear Potential Flow Theory

**CFD**

Computational Fluid Dynamics

**FNPF**

Fully Nonlinear Potential Flow

**MPC**

Model Predictive Control

**WSI**

Wave Structure Interaction

**PEWEC**

Pendulum Wave Energy Converter

**PTO**

Power Take Off

**MDP**

Marko Decision Process

**SARSA**

State Action Reward State Action

**LFA**

Linear Function Approximation

**NSE**

Navier Stokes Equations

**DOF**

Degree of Freedom

**PDE**

Partial Differential Equation

**TD**

Time Domain

**FD**

Frequency Domain

**SD**

Spectral Domain

# Chapter 1

## An Introduction to Wave Energy Converters



**POLITECNICO  
DI TORINO**

### 1.1 Reasons behind recent interest in WECs

Fossil fuels are still the main source of energy for our society worldwide, accounting for around the 80% of the entire energy consumption [1].

Even given the recent technological improvements of renewable energy resources, like solar and wind, it still is not sufficient to significantly replace fossil fuels.

Moreover, due to the increase in environmental damage brought in by fossil fuels, Europe aims at reducing by 55% the emission by 2030 and aim at becoming the world's first climate-neutral continent [2].

A possible help towards this goals may be Wave Energy: harvesting energy from

the motion of the sea waves.

The main reasons behind the interests in WECs are:

1. The high energy content of the sea, since it is estimated that being able to extract only the 0.2% of energy of the sea would provide sufficient energy worldwide [3]).
2. The high density of the sea energy: the solar energy passing to the wind and then to the sea, interestingly increases its density in each passage [4].
3. An higher availability (around 90%) with respect to solar and wind energy, which arrive at 30% of availability.
4. Little to no damage to the environment.
5. Easy integration with wind/solar systems.
6. Wave power is highly predictable (parameters slowly varying with time) [5].

All this reasons brought to a increasing interest in Wave Energy Conversion during the last years, but it still is not commercially feasible, as it can be noted from the very little number of devices working at the moment.

The main problem for the commercial viability of WECs is the way too high Levelised Cost of Energy (LCOE) given by the absence of a uniform convergence towards a unique design (indicating that there is no absolute best design for all the possible cases [6]) and by the difficulties encountered during the development of a suitable Control System for these devices.

In the following section there will be a presentation of the working principles of the main WECs along with the principal developed designs.

Successively an overview on how traditional Control Theory works on WEC will be made, underlining the main issues it faces and how they could be solved by passing to a RL based Control System.

## **1.2 Main types and working principles of WECs**

Given the fact that there are more than 1000 device designs reported for WECs [7], an unique categorization method does not exist.

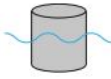
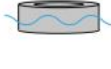
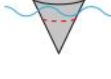
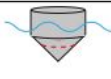
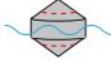
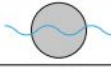
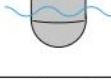
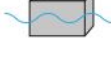

Nonetheless it is possible to distinguish in those devices 4 main types with respect to the shape of device:

1. Point Absorbers (PAs), the ones with characteristic dimensions much smaller than that of the incident sea wave. PAs can operate in heave, pitch or in multiple Degrees of Freedom (DoFs) and they can be further subdivided in 2

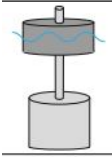
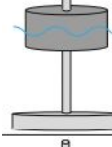
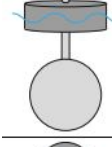
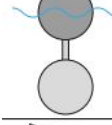
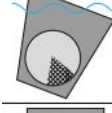
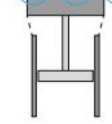


types: one-body PAs and two-bodies PAs.

One-body PAs use one body floating or partially submerging to have a Wave-Structure Interaction (WSI) and behave like a low pass filter in the frequency domain, while two-bodies PAs utilize the relative motion between the to 2 bodies for the interaction and behave like a band pass filter. For those device it is important to suitably tune their natural frequency to that of the incident wave in order to maximize the extracted power, but also to implement an end-stop function in case the motion exceeds the structural constraints

Schematic	Shape	Parameters
	Cylinder (Cyl)	Radius Height Draught
	Cylinder-moonpool (CylMpl)	Radii Height Draught
	Cone (Con)	Radius Cone Angle Draught
	Cylinder-cone (CylCon)	Radius Heights Cone Angle Draught
	Cone-Cylinder-cone (ConCylCon)	Radius Heights Cone Angle Draught
	Sphere (Sph)	Radius Draught
	Cylinder-sphere (CylSph)	Radius Height Draught
	Cuboid (Cub)	Length Width Height Draught
	Arbitrary shape (ArbShp)	Points draught freeboard

**Figure 1.1:** Typical shapes for one-body PAs. Taken from [7]

Schematic	Shape	Parameters
	Cylinder-cylinder (Cyl-Cyl)	Radii Height Draught
	Cylinder-plate (Cyl-Plt)	Radii Draught
	Cylinder-sphere (Cyl-Sph)	Radii Height Draught Submergence
	Sphere-sphere (Sph-Sph)	Radii Draught Submergence
	Hull-pendulum (Hul-Pdl)	Profile Draught
	Cylinder-piston (Cyl-Pst)	Radii Height Draught Submergence

**Figure 1.2:** Typical shapes for two-bodies PAs. Taken from [7]

2. Oscillating Water Column (OWC), which utilize a hollow structure to trap in it air that gets compressed by the oscillating water under it. Since there is no

oscillating linear motion, an end-stop function is not required.









Schematic	Shape	Parameters
	Universal fixed (UnivFix)	Chamber size Orifice size Submergence
	Bottom-shaped (BtmShp)	Chamber size Orifice size Submergence Bottom shape
	Front-shaped (FrntShp)	Chamber size Orifice size Submergence Front wall
	U-shape	Chamber size Orifice size Submergence
	Universal floating (UnivFlt)	Chamber size Orifice size Draught
	Spar-buoy	Chamber size Orifice size Draught
	Backward bent duct buoy (BBDB)	Height Length Width
	UGEN	Radii Height Draught Submergence

Figure 1.3: Typical types of OWCs. Taken from [7]

- Attenuators, which are floating WEC devices, usually composed of multiple bodies interconnected, orientated parallel to the direction of the wave in order to attenuate it, so extracting power.

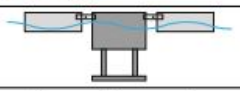
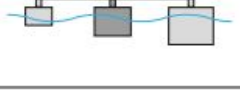
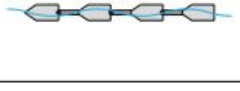
Schematic	Shape	Parameters
	Barge	Number Length
	M4	Radii Distance Submergence
	Seaweed	Length Draught Distance

Figure 1.4: Typical types of Attenuators. Taken from [7]

- Terminators, similar to Attenuators, but they are oriented perpendicular to the wave direction in order to actually terminate it


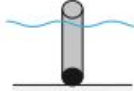
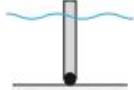
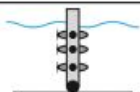
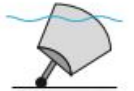
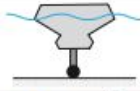
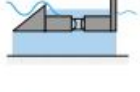

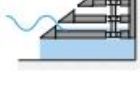
Schematic	Shape	Parameters
	Duck	Submergence Beak angle
	Cylinder	Radius Height Submergence
	Flap	Length Width Height Submergence
	Flap-vane	Flap size Vane size Vane number Submergence
	C-cell	C-cell profile Height Submergence
	Arbitrary shape (ArbShp)	Control points
	Wave dragon (WD)	Ramp angle Freeboard Submergence
	Catamaran-like WaveCat	Wedge angle Freeboard Draught
	Sea slot-cone generator (SSG)	Crest level Ramp angle Ramp draught

Figure 1.5: Typical types of Terminators. Taken from [7]

All these types of WECs convert the kinetic and potential energy of the sea waves thanks to the force/displacement applied to them. The Power-Take-Off (PTO) is the component which then converts this energy into electric energy.

PTOs can be fully electromechanic or hydraulic and electromechanic, with the last one more suited for high torques and low velocities, like in the oceans.

In a first approximation of the device (equivalent electric circuits), the PTO parameters, which can be seen as an equivalent electric impedance, are the ones that must be tuned instant-per-instant in order to give the maximum absorbed

power for that wave condition. For this reason, all WECs have a sensor for measuring the actual sea state. Moreover for an optimal Model Predictive Control (MPC) there are other sensors nearby the device, in order for it to know in advance the future wave conditions, since the controller is anticausal.

It is then needed to better define wave conditions and how they are quantified.

### 1.3 Wave Conditions

There are 3 possible ways to account for the wave conditions during the design of a WEC

1. Regular waves: simplest approximation, is that of a single wave with defined amplitude and period. If the wave height  $H$  is much smaller than the wavelength  $\lambda$ , by using Linear Wave Theory, the wave equation is

$$\eta(x, y, t) = \frac{H}{2} \cos(\omega t - kx \cos \beta - ky \sin \beta + \phi)$$

where  $\eta(x, y, t)$  is the wave elevation,  $\omega$  is the angular frequency,  $k$  is the wave number,  $\beta$  is the incident angle and  $\phi$  is the initial phase.

In deep water, the wave power per unit width of the wave front, or wave-energy flux, can be defined

$$J_r = \frac{\rho g^2}{32\pi} T H^2$$

where  $T$  is the wave period and  $g$  is the gravity constant.

Due to their simplicity, regular waves are easy to be account for but leave only simple performance criteria available, like the average power.

2. Irregular waves: where it is considered a wave spectra instead of a single amplitude-period pair. One of the most notable wave spectra is the JOint North Sea WAve Project (JONSWAP) [8]

$$S(\omega) = \frac{5H_s^2}{16} \frac{\omega_p^4}{\omega^5} \exp\left(-\frac{5\omega_p^4}{-4\omega^4}\right)$$

where  $H_s$ ,  $\omega_p = \frac{2\pi}{T_p}$  and  $T_p$  are the significant wave peak, peak angular frequency and period of the wave spectra.

In practice the significant wave height  $H_s$  and the energy period  $T_e$  are statistically estimated from the measurements of the PSD, using the spectral

moments [9].

By defining the general  $k^{th}$  spectral moment as:

$$m_k = \int_{\omega=0}^{\infty} \omega^k S(f) d\omega$$

Where  $\omega$  is the angular frequency of the signal, and  $S(\omega)$  is its PSD. The energy period can be defined as:

$$T_e = \frac{m_{-1}}{m_0}$$

and the significant height as:

$$H_s = 4\sqrt{m_0}$$

In this way the wave-energy flux can be written

$$J_{ir} = \frac{\rho g^2}{64\pi} T_e H_s^2$$

which becomes identical to that of the regular waves with  $H = H_s$  and  $T = T_p$ .

Due to their more complex nature, considering the irregular waves gives a more realistic design.

3. Wave climate: the most realistic case but also the most time and resource consuming. Annual wave observation are effectuated and statistically elaborated in order to output a scatter table (amplitude-period) like that in figure 1.7 [10].

Usually the most common scenario is that utilized for power extraction design, while for the safety and survival of the devices it is considered the most critical one, i.e. the upper right corner.

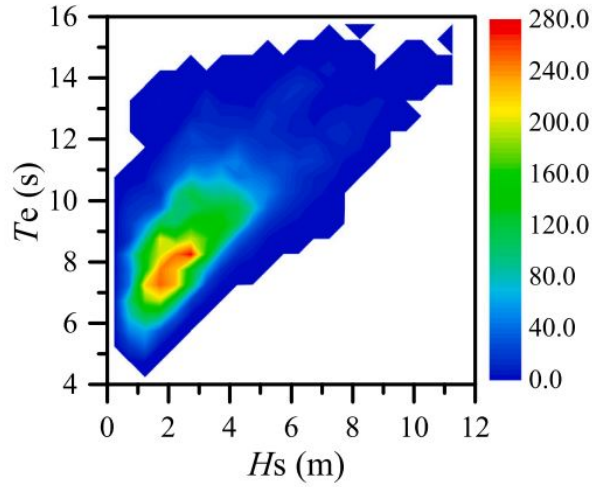


Figure 1.6: Example of a scatter table. Taken from [7]

## 1.4 Hydrodynamic modelling of the WEC and of the WSI

There exist various ways to model the dynamics of the system, with different initial assumptions and computational costs. It is important to address the contrast between the quality of the model and its computational heaviness.

By using Newton's second Law, it is possible to get the general equation of motion as:

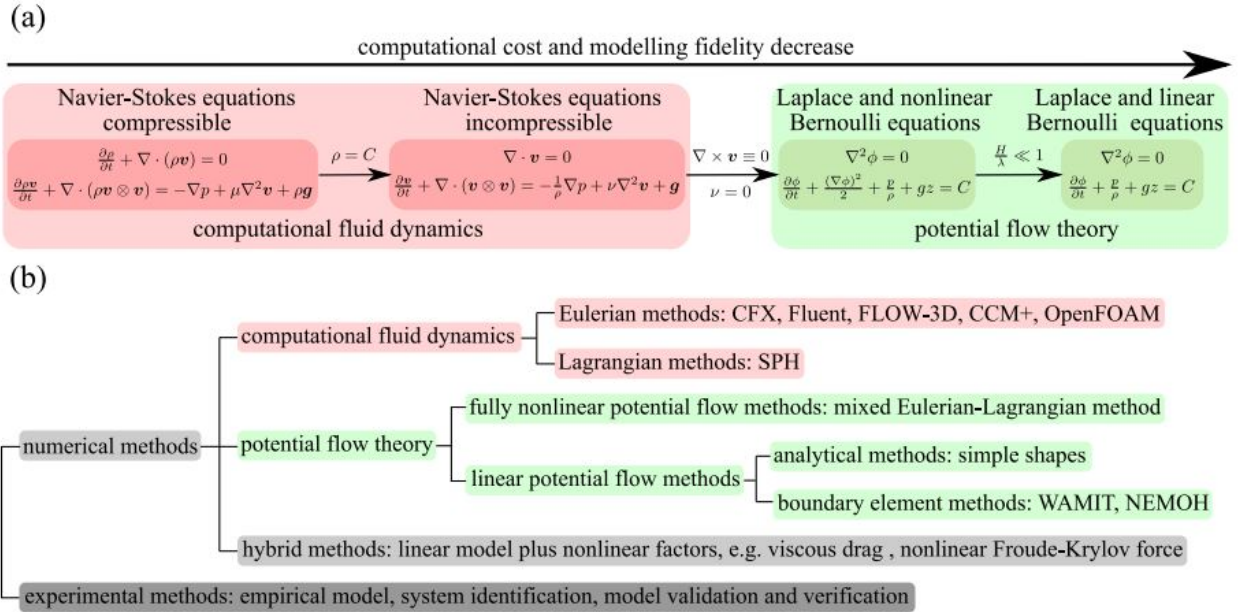
$$\mathbf{M}\ddot{\xi} = f_h(t) + f_g(t) + f_{PTO}(t) + f_m(t) + f_{add}(t)$$

whit  $\mathbf{M}$  the inertial mass of the device,  $\xi$  the displacement of the device,  $f_h$  the hydrodynamic force,  $f_g$  the gravity force,  $f_{PTO}$  the force on the PTO,  $f_m$  the mooring force and  $f_{add}$  any other possible additional force.

The number of equations is that of the DoFs of the system (max 6).

The most critical force to calculate is  $f_h$  since it is given for each DoF by a surface integral of the pressure, which may be impossible to calculate analytically for an arbitrary surface, moreover, it is exactly the force responsible for the WSI.

There are 2 possible solutions: that of computing those integrals numerically (and here is where different assumptions on the hydrodynamic behaviour of the wave make a difference); or estimating the values of this interaction through experimental data.



**Figure 1.7:** Schematic representation of the possible modelling choices. Taken from [7]

If the environment is highly non-linear (as we would like for energy conversion applications, since non-linear effects arise in high energy configurations), in order to have a simulation loyal enough to the reality, Navier-Stokes equations (NSEs) have to be used, with the eventual simplification for incompressible fluids. These simulations, even if highly accurate, are computationally demanding, generally requiring 1000s of computational time for 1s of simulation time. Another possible solution is that of using Potential Flow Theory with equations way easier to simulate, specially for the Linear Potential Flow Theory, but with lower fidelity in the results. In the end, there are 4 possible ways to go:

1. Computational Fluid Dynamics (CFD): Since NSEs have to be solved, and there is no analytical solution for those equations, numerical methods are used to approximate the solution. There are 2 main categories: Eulerian methods, which discretise the space and time domain in order to have a system of linear algebraic equations; Lagrangian methods, which actually discretise the fluid as a set of particles. Both can deal with compressible and incompressible fluids. Generally, Eulerian based methods are used for WEC dynamic analysis, even if Lagrangian method can more easily simulate extreme wave events like wave-breaking.

2. Potential Flow Theory (PFT): assuming an ideal fluid, not only incompressible, but also inviscid and irrotational, the NSEs equation can be simplified into the Laplace and nonlinear Bernoulli equations.

In this way the Partial Differential Equations (PDEs) on the velocity vector  $\mathbf{v}$  in the NSEs become PDEs of the potential flow scalar  $\phi$ . Moreover, small wave steepness ( $\frac{H}{\lambda} \ll 1$ ) scenario is assumed.

This equations can still account for nonlinearities, Fully Nonlinear Potential Flow (FNPF), or not, Linearl Potential Flow (LPF), even if FNPF, not considering viscosity, does not have the same accuracy of CFD for highly nonlinear environments.

3. Hybrid modelling methods: since both LFP and FNPF struggle for nonlinear environments and CFD requires a way too high computational time, an idea is that of adding "ad hoc" non linear terms in the dynamic equation of motion, in order to derive parametrised mathematical models of the WEC. This method requires an accurate knowledge of the kind of nonlinearities affecting the system, and so highly depends on the control and mooring systems. An example can be [11]:

$$(\mathbf{M} + \mathbf{M}_{\infty})\ddot{\xi} = f_e(t) + \mathbf{K}\xi(t) - \mathbf{k}_r(\tau) * \dot{\xi} + f_{PTO}(t) + f_m(t) + f_{nl}(t)$$

with  $f_{nl}(t)$  representing the nonlinear hydrodynamic forces.

4. Experimental methods: in the end, experimentation is still need to validate the simulations' results.

Generally, at the early stage of design, a small scale prototype is realized and tested.

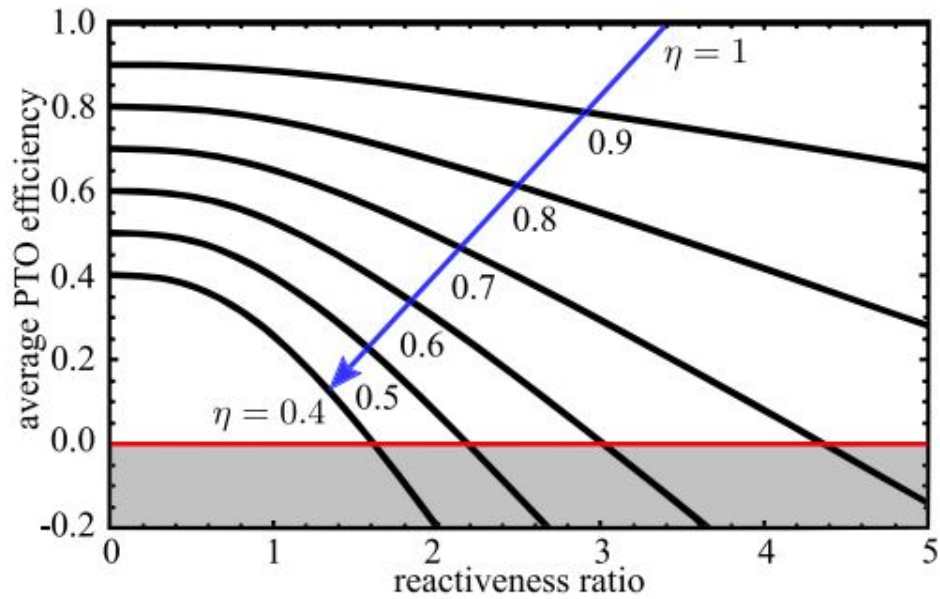
It is important to pay attention to the scaling, since quantities like hydrodynamic forces, body motion and captured power can be easily scaled up with the Froude number, but other effects like mechanical friction do not escalate accordingly and can disproportionately affect the experimental results.

Finally there is to model the power take-off, the component responsible for the transfer of energy from the motion of the WEC into a useful form, i.e. electricity. They can support only unidirectional flow of power (resistive power flow) or also bidirectional (reactive power flow).

The PTO has its own dynamic and non linear effects that directly affect the overall dynamic of the system. It is important to consider the energy losses due to those non idealities and there are more ways to do that, with different accuracy and computational efforts.

One of the most common ways is that of using a parametrised graph, like that in figure 1.9 [12].

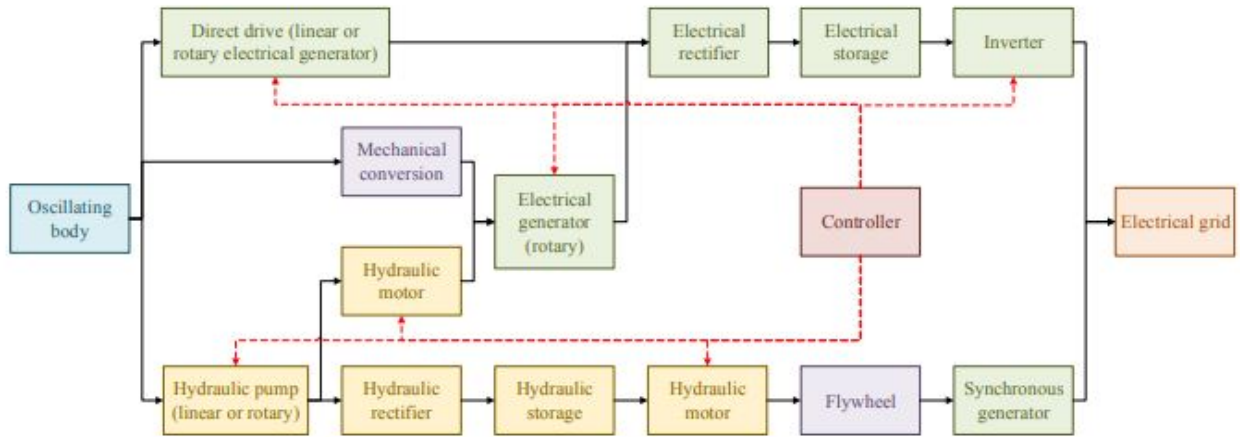




**Figure 1.8:** Example of a graph representing the efficiency variation of a PTO. Taken from [7]

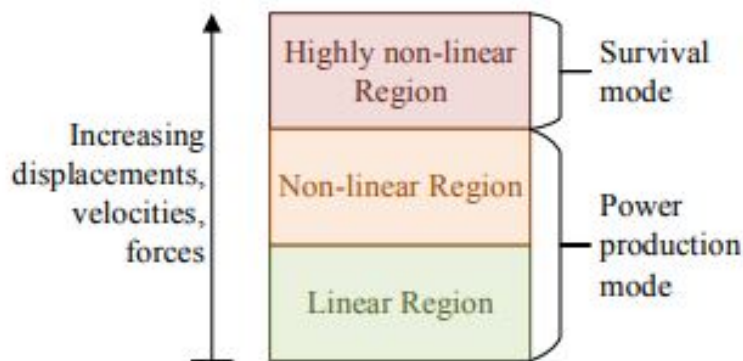
The most important aspect is that the efficiency decreases as the reactivity ratio increases, thus the PTO rapidly becomes inefficient if working for high reactivity ratios.

With respect to their working principle, it is possible to define 4 main types of PTO [13]: hydraulic with hydraulic rectifier, hydraulic with electrical rectifier, mechanical with mechanical rectifier and direct drive.



**Figure 1.9:** PTO working scheme for the 4 main types. Taken from [13]

From the WEC's point of view, there are 2 main operating modes with respect to the energy content and the amount of nonlinearities in the sea state: when the state is highly nonlinear, the device is not able to generate energy and just enters in a survival mode for avoiding any damage; when slightly nonlinear or totally linear, the device is able to generate energy.



**Figure 1.10:** WEC operating modes with respect to sea state. Taken from [14]

In the end, due to the way too high computational complexity of the CFD simulations, parametrised and simplified models are used for simulations and for the control of the entire system.

The resulting equation of motion can be written in time-domain (TD), in frequency domain (FD) and in spectral domain (SD).

For example, in FD, the equation on motion becomes:

$$\{-\omega^2[\mathbf{M} + \mathbf{M}_a(\omega)] + j\omega\mathbf{B}(\omega) + \mathbf{K}\}\boldsymbol{\Xi}(\omega) = \mathbf{F}_e(\omega) + \mathbf{F}_a(\omega)$$

where  $\boldsymbol{\Xi}(\omega)$  and  $\mathbf{F}_e(\omega)$  are the frequency domain representations of  $\boldsymbol{\xi}(\omega)$  and of  $\mathbf{f}_e(\omega)$ , while  $\mathbf{F}_a(\omega)$  represents the control force and eventual linearised nonlinear terms.

The parameters  $\mathbf{M}_a(\omega)$ ,  $\mathbf{B}(\omega)$ ,  $\mathbf{K}$ ,  $\mathbf{F}_e(\omega)$  are usually obtained through BEM methods. It is to note that, in the most general case, those parameters are matrices, where there are more DoFs, and both displacements and forces are vectors.

FD models are very computationally efficient but consider ideal PTO system, so that the entire PTO/control system can be approximated by a mass-spring-damper system. Therefore, it is difficult to consider nonlinear term, except some like the viscous drag [15]

In a similar way, SD models, obtained through Lorentz linearisation [16], are pretty computationally efficient, can consider some nonlinear terms, but fail to accurately describe the instantaneous dynamics and the captured power.

For this reasons, most control system utilize TD domain, since able to deal with nonlinear terms. Even if at first view TD domain models have a higher computational complexity, since they hide in them a convolution radiation force term, this term can be approximated by a finite order parametrised model, so reducing the computational complexity of the model. This can be done via system identification techniques [17].

## Chapter 2

# Traditional control of WECs

### 2.1 History of WECs control

The first study about the power maximizing control of a WEC dates back to 1975 [18], considering monochromatic waves, fully linear hydrodynamics and an ideal PTO.

Since then, a variety of different control approaches have been tried on WECs, with the main challenge encountered being the intrinsic anticausality of this control problem [19], i.e. for choosing the optimal action it is needed to know the past and the future states of the systems, thus requiring to have sensors nearby the WEC to measure the incoming waves and filtering those measurements from disturbing noises, like wind.

The simplest control is the resistive one, where the PTO force is proportional to the velocity at the PTO. This is a simple and practical way to implement control, avoiding negative power flows (instants where the WEC gives back the energy to the sea), but results in a suboptimal control in the end.

The natural evolution of resistive control is the reactive control, where a stiffness term is added in order to control not only the amplitude of the variables, but also their phase. This approach, known as complex-conjugate approach, is the optimal one from an ideal hydrodynamic point of view, maximizing the converted power. Its main problems are due to the bidirectional power flow needed, and the inability to deal with position or velocity constraints, actually trying to maximize them as much as possible (resonance).

Moreover, the optimal stiffness-damping pair are dependent on the excitation frequency [20] and so have to be tuned and adapted to sea state changes, and this can be challenging for irregular waves.

For these problems, people started to look out for different control methods, like adaptive control techniques.

Totally different control techniques are the discrete control methods, like the latching where the action is binary, stop moving part or do nothing, and so easy to implement and deal with constraints, but the overall controller results in a suboptimal one. The main advantage is that it is still a more efficient control of the resistive one, even without bidirectional power flows. A similar control is the declutching control where the action consist in disconnecting or reconnecting the control system.

Another promising control technique is the model predictive one, which selects each action after solving a quadratic optimization problem, and can in this way easily include displacement and velocity constraints.

All these controllers suffer from the inability to adapt to the model parameters changing in time as the device is used. Indeed WECs can undergo damages to their structures while in use and this can change their dynamical properties. This is the main reason that brought to the development of Machine Learning based control systems, like the ones using Reinforcement Learning algorithms to train the device how to act.

## 2.2 Resistive control

Resistive or passive control is the strategy where the PTO force is directly proportional to velocity of the PTO.

Considering a PA with one degree of freedom,  $z$ , and defining the damping coefficient of the PTO as  $B_{PTO}$ , the PTO force is given by:

$$f_{PTO} = B_{PTO}\dot{z}$$

which gives as absorbed power  $P$

$$P = f_{PTO}\dot{z}$$

With the dynamic model in the frequency domain, it is possible to optimize the system with respect to  $B_{PTO}$  in order to maximize the absorbed power, obtaining [21]:

$$B_{PTO, opt} = \sqrt{B(\omega)^2 + [\omega(M + M_a(\omega)) - \frac{K}{\omega}]^2}$$

which gives the maximum absorbed mean power:

$$\bar{P}_{opt} = \frac{1}{4} \frac{F_e(\omega)}{B(\omega) + \sqrt{B(\omega)^2 + [\omega(M + M_a(\omega)) - \frac{K}{\omega}]^2}}$$

This formulas could be extended to irregular waves, by using the superposition principle, but saturation force constraint can not be easily represented in the frequency domain. This is why usually it is preferred to use optimization based on non-linear TD models [22].

Discretising the sea state with discrete value of significant wave height and energy wave period, it is possible to calculate for each discrete state its optimal damping. Anyway it is clear that this approach is way too simple, considering all ideal, linear and stationary, not even needing to wave forecast.

## 2.3 Reactive control

Reactive control can be seen as an extension of resistive control, where the PTO force is given by a damping term proportional to the velocity (like in resistive control) plus a stiffness term proportional to the displacement at the PTO.

$$f_{PTO} = B_{PTO}\dot{z} + K_{PTO}z$$

The stiffness term enable the control also of the phase of the response and not only of its amplitude. An inertia term could be used instead of the stiffness one, but the stiffness proved to have a more robust control [23]. Having an additional stiffness term, the power may be negative in some parts of the cycle, and so the generator has to work for some instants as a motor, giving power to the waves, and so requiring a special design.

Considering a PA with one DoF  $z$ , the optimal coefficients are given by:

$$B_{PTO, opt} = B(\omega)$$

$$K_{PTO, opt} = \omega^2(M + M_a(\omega))$$

which give as maximum mean absorbed power

$$P = \frac{|F_e|^2}{8B_{PTO}}$$

Reactive control is also called impedance-matching control, since it comes down to setting the PTO impedance equal to complex-conjugate of the intrinsic device impedance.

Reactive control has a greater absorbed power with respect to resistive control, but the obtained values of the control force, body displacement, structural load, and

negative power flows make it hard to implement in practice.

It is then needed to actually increase the damping coefficient and decrease the stiffness coefficient.

Moreover, the optimal damping may be negative for some states, thus potentially leading to instability.

For this reasons, reactive control always needs TD simulations for selecting the most suitable impedance.

Finally it has to be considered the efficiency of the conversion of the PTO  $\eta$ , which decreases the overall absorbed power.

$$P(t) = \begin{cases} \eta f_{PTO}(t)\dot{z}(t), & \text{if } f_{PTO}(t)\dot{z}(t) > 0 \\ \frac{1}{\eta} f_{PTO}(t)\dot{z}(t), & \text{otherwise} \end{cases}$$

accounting for both positive and negative power.

## 2.4 Model Predictive Control

Model Predictive Control (MPC) refers to a variety of model-based control techniques. These techniques employ an optimization problem over a future horizon in order to derive the optimal control action [24].

The main features of an MPC are:

1. A mathematical model of system, so that it is possible to simulate it over a future time horizon. Usually discrete model are used, and so continuous-time problems have to be discretised.
2. An objective function to be maximized, that so gives the optimal control sequence along the optimization window.
3. A receding strategy that, at each step, applies only the first control action of the optimal control sequence obtained at that step.

The obtained MPC can be linear or non-linear with respect to the complexity of the mathematical model used.

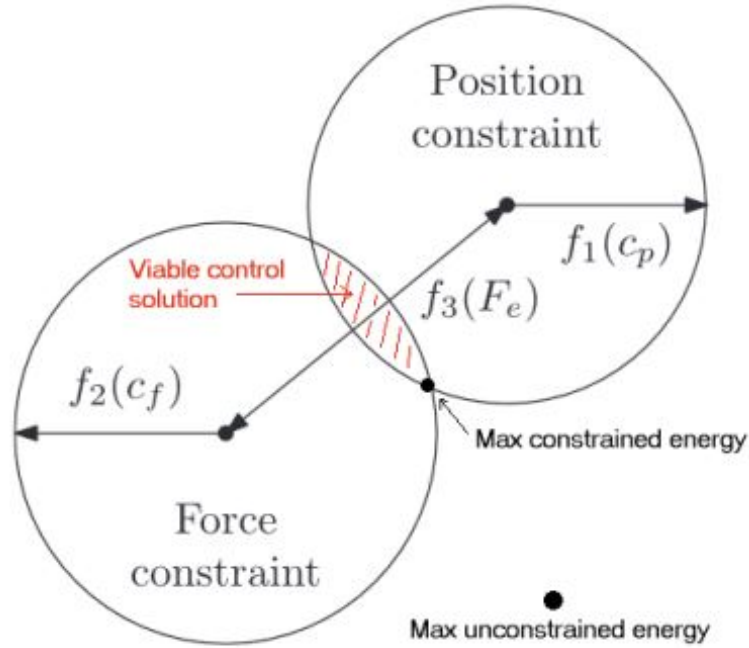
One of the main advantages of the MPC is that is naturally able to satisfy all the constraints the system is subject to, due to its intrinsic definition as an optimization problem. This means that, in the case the optimal action would violate the constraints, the controller can easily identify sub-optimal actions which maximize the objective with respect to all other actions allowed by the constraints.

In general, an MPC optimization problem can be stated as:

$$\min_{f_{PTO}} - \int_0^T f_{PTO}(t)v(t) dt$$

$$\begin{aligned} s.t. \xi(t) &\leq \xi_{max} \\ s.t. f_{PTO}(t) &\leq f_{PTOmax} \end{aligned}$$

For a WEC device, an MPC controller can easily restrict its research to the actions that do not violate position constraints and force constraints.



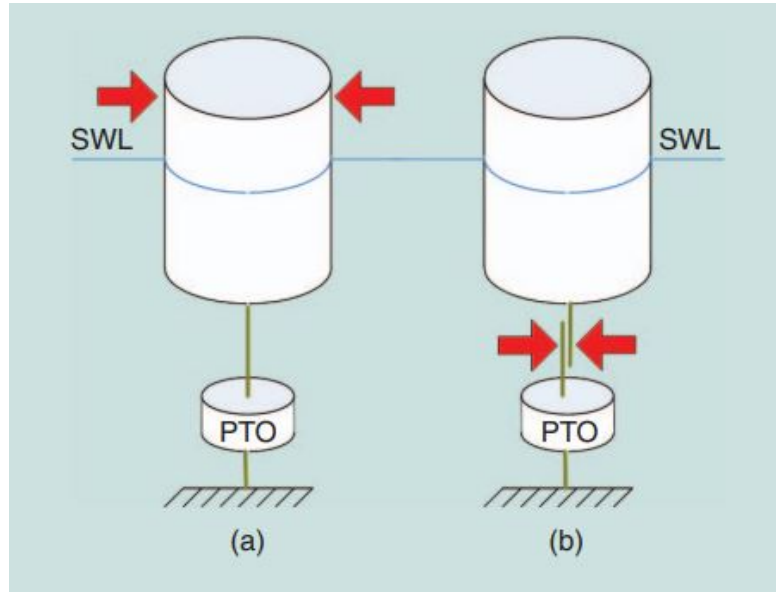
**Figure 2.1:** Choice of a sub-optimal action which does not violate constraints. Taken from [25]

The main problem of the MPC is that it is anti-causal. In order to solve the optimization problem, it needs information from the future regarding the exciting wave (excitation force estimation and forecasting). In order to do so, several sensors have to be placed nearby the WEC, measuring the incoming waves. The difficulties given by this problem pushed the research of a suitable control system for WEC into the Machine Learning methods, such as Reinforcement Learning.

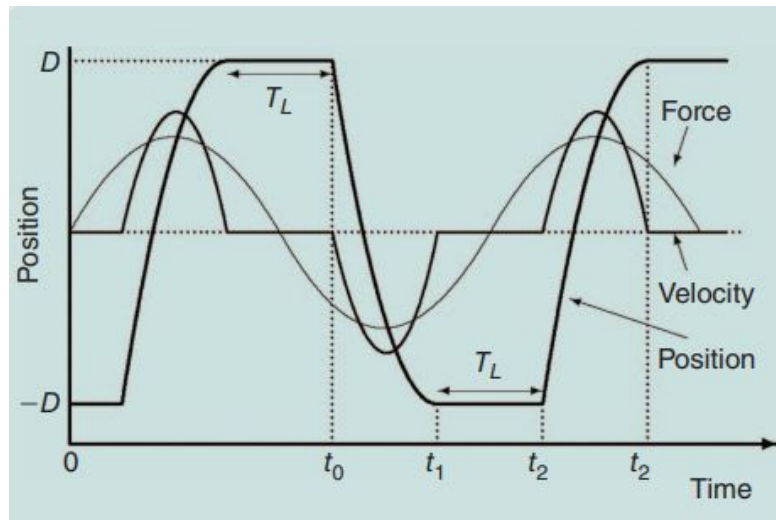


## 2.5 Discrete Control-Latching and Declutching

Discrete control algorithms are the ones implementing an on/off PTO force by a braking mechanism, i.e. latching, or a bypass, i.e. declutching.



**Figure 2.2:** Action principle of latching (a) and declutching (b). Taken from [20]



**Figure 2.3:** Example of the evolution of the system variables in a latching control. Taken from [20]

Both latching and declutching are sub-optimal (and not optimal) controllers, but they offer a simple way to achieve resonance when the natural frequency of the device is lower (declutching) or higher (latching) than the frequency of the exciting wave.

## Chapter 3

# Reinforcement Learning Algorithms

### 3.1 Introduction to Reinforcement Learning

Reinforcement Learning (RL) is a field of study in between Machine Learning and Control Theory. It is a class of nature-inspired algorithms where an intelligent agent interacts with an environment through actions in order to maximize a defined cumulative reward.

RL can be also seen as the third paradigm of Machine Learning, being actually different from both supervised and unsupervised learning.

For the general view of RL given in this thesis, [26] [19] have been taken as primary guides, where the same arguments are treated in greater detail.

In the general scheme [26] there is an agent, in a particular state  $s$ , which performs an action  $a$ .

This action moves the agent into the new state  $s'$  and is followed by a reward  $r$ . The selection of the action is modelled with a Markov Decision Process (MDP), based on the value function, i.e. an estimate of the future reward.



**Figure 3.1:** General diagram of a RL process. Taken from [26]

During the design of such an algorithm, it is important to keep in mind the need for an accurate trade-off between exploration and exploitation: if the agent aims only at maximizing the future reward (exploitation) it will never visit states different from the usual ones, thus not exploring well the environment. This is known as the problem of exploitation vs exploration.

It would be good if the agent aims more at exploring during the initial stages, so that it can collect information on the environment, and then it can start exploiting without having missed some important infos.

Before talking more in detail about Reinforcement Learning, it is needed to introduce some terminology and concepts:

1. The policy,  $\pi(s)$ , which is the behaviour of the agent with respect to the state he is in. Essentially is a function of the states whose output is the action the agent takes. A policy can be deterministic or stochastic and it is called greedy when it aims at maximizing the rewards, i.e. prefers exploitation.
2. The reward function,  $r(s, a)$ , which quantifies the amount of instantaneous incoming from doing a particular action in a particular state. A discount factor,  $\gamma \in [0,1)$ , is used to discount future rewards. This discount factor comes from an economic environment and moreover helps ensuring the convergence of the algorithms. It is to remark that the design of a suitable reward function is often the most delicate part of a RL algorithm, since in many scenarios the notion of reward is not well defined or the reward itself can be delayed with respect to when the action is taken. For example, for developing a RL algorithm for playing chess. the most natural definition of reward is winning/losing which is an event that happens only at the end, thus making it hard to relate to a

particular state-action. For these reasons more complex and instantaneous definition of rewards are needed [27].

3. The value function, which can be of 2 types, the state-value function ,  $V(s)$ , which quantifies how much good it is to stay in a particular state, and the state-action value function,  $Q(s, a)$ , which quantifies how much good it is to do a particular action in a particular state. The main difference from this functions and the reward functions is that the former ones actually take in account also the future rewards that will come from the states where the agent will move to.

Reinforcement Learning techniques can be grouped into 3 main categories: Dynamic Programming (DP), Temporal Difference (TD) and Monte-Carlo methods (MC). DP is a well developed mathematical tool, able to give analytical solutions, but it requires a model of the system in order to output a suitable policy [26] [28], thus not giving a model-free controller.

Instead MC and TD can learn directly from data. The difference between MC and TD is that MC needs to wait until the end of the episode/task in order to update the value function, while TD can update it after a number  $\lambda$  of time steps, so giving the various TD- $\lambda$  techniques.

## 3.2 Markov Decision Process

Markov Decision Processes (MDPs) are at the base of the working principle of every RL techniques.

MDP can be seen as a memoryless approximation of an evolving complex system. An MDP is defined by the tuple  $(S, A, P, \gamma)$ , where S is the set of all the states, A is the set of all the action and P is a transition model of the form  $P(s, a, s') = \text{prob}(\text{newstate} = s' | \text{originalstate} = s \ \& \ \text{action} = a)$  and  $\gamma$  is the already introduced discount factor.

It is needed also a reward function  $R(s,a,s')$ , corresponding to the reward obtained by performing action a in state s and moving to state s'. The reward notation can be simplified by considering:

$$R(s, a) = \sum_{s' \in S} P(s, a, s') R(s, a, s')$$

The policy  $\pi$  is defined as a mapping from the state space to action space, associating to each state the probability it selects a specific action in it, for all actions.  $\pi : S \rightarrow \Omega(A)$ , where  $\Omega(A)$  is the set of all possible probabilities distributions over A.

With  $\pi(a; s)$  it is indicated the probability that a given policy choose the specific action  $a$  in the state  $s$ .

A particular case is when this policy,  $\pi(s)$  is fully deterministic, having as output one and only one action  $\pi : S \rightarrow A$ .

The value-function  $Q^\pi(s, a)$  is defined over all possible state-action pairs, and represents the total expected reward of performing action  $a$  in state  $s$  and following  $\pi$  thereafter, considering also all the future (discounted) steps.

The exact value of the value function can be solved by solving the linear system of the Bellman equations [29]:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') \sum_{a' \in A} \pi(a'; s') Q^\pi(s', a')$$

which can be rewritten in matrix form as:

$$Q^\pi = R + \gamma P \Pi_\pi Q^\pi$$

where  $Q^\pi$  and  $R$  are vectors of size  $(|S|^*|A|)$ ,  $P$  is a stochastic matrix of size  $(|S|^*|A|, |S|)$ , and  $\Pi_\pi$  is a stochastic matrix of size  $(|S|, |S|^*|A|)$ .

This results in the linear system:

$$(I - \gamma P \Pi_\pi) Q^\pi = R$$

which can be solved analytically or iteratively, where  $I$  is the identity matrix.

For every MDP, there exist an optimal policy  $\pi^*$  which maximize the total reward obtained and can be casted as:

$$\pi^* = \arg \max_{\pi} Q^\pi \forall s \in S, a \in A.$$

TD and MC methods can avoid using a  $P$  matrix, and so a model of the noise, and estimate the transition probabilities a posteriori from the data.

### 3.3 Exploration Strategy

In order to ensure that the agent explores exhaustively the environment he is in, usually  $\epsilon$ -greedy policies are used.

In an  $\epsilon$ -greedy policy, when the agent is in state  $s$  and has to choose one action to perform, with probability  $\epsilon$  it chooses an action randomly and in the other cases, i.e. with probability  $1-\epsilon$ , it selects the action which, according to its knowledge, maximizes the reward coming from it.

$$a = \begin{cases} \arg \max_{a' \in A(s)} Q(s, a'), & \text{with probability } 1-\epsilon \\ \text{random action}, & \text{with probability } \epsilon \end{cases}$$

It is important to note that the set of admissible action  $A(s)$  directly depends on the particular state  $s$ . Indeed, not all action are admissible for all states.

If for example the actual state is at a boundary of the state-space (like the value of damping/stiffness being already at its maximum/minimum), the actions that would bring out of the state-space are not admissible (increase/decrease damping/stiffness).

For focusing on exploring during the first stages, and succesively starting preferring exploitation, the  $\epsilon$  is set such that it is greater for the first visits in a state, and then starts decaying for all subsequent visits.

$$\epsilon = \begin{cases} \epsilon_0, & \text{if } N \leq N_\epsilon \\ \epsilon_0 / \sqrt{N - N_\epsilon} & \text{if } N > N_\epsilon \end{cases}$$

Where  $\epsilon_0$  is the initial exploration rate, and  $N_\epsilon$  is the number of visits to a state, after which, the exploration rate in that state starts to decrease.

The number of visits to all possible discrete state-action pairs, has to be memorized in a matrix  $N(s, a)$ , of dimension  $(|S|, |A|)$ .

In the system for the  $\epsilon$  selection, the discriminating  $N$ , the number of visits to the particular state  $s$ , can be calculated for the matrix  $N(s,a)$  as:

$$N = \sum_{a' \in A(s)} N(s, a')$$

This particular type of decay used is that most commonly adopted for WEC control. For more traditional RL applications a slower decay is typically used, and usually depends on the number of episodes instead [30].

### 3.4 Monte-Carlo methods

With MC methods, the learning problem is solved by averaging sample rewards, which in this context are called returns.

These methods are well defined only when the notion of episode is well defined too: the experience has to be divisible into discrete episode with a defined end.

The policy is updated only when the current episode ends rather than online, making an actual implementation of an MC RL alghorithm hard to implement in real-time.

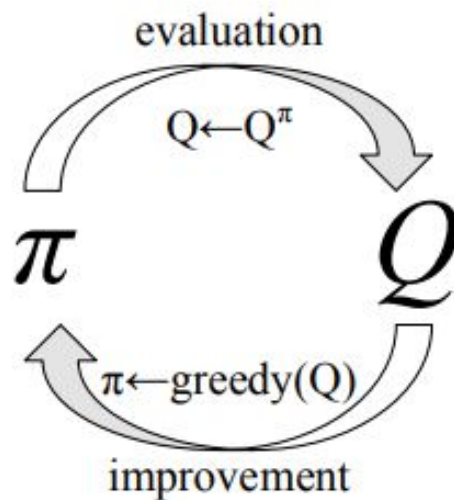
The state value function of a policy is simply obtained by averaging the returns experienced after visiting that state.

There are 2 main ways to implement that:

1. the every-visit Monte-Carlo approach, which approximates the value function  $V^\pi(s)$  by calculating the mean of the returns following all visits in a state  $s$  in a set of episodes, while following policy  $\pi$ .
2. the first-visit Monte-Carlo approach, which instead determines the value function from the average over only the first visits to the states.

Both approaches converge to the actually correct value function, for a number of visits approaching to infinity.

When a model of the environment is not available, MC methods have to alternate a stage of policy evaluation and a stage of policy improvement.



**Figure 3.2:** Policy iteration in MC. Taken from [26]

In the policy evaluation, the state-action value function is updated by using the returns stored during the episode.

In the policy improvement, the policy gets updated to the greedy policy with the newly computed value function.

Moreover, MC methods further subdivide into on-policy and off-policy schemes.

In on-policy schemes, the policy that is used to update the value-function is the same policy that is being used to select the actions.

In off-policy schemes instead, the policy that is used to update the value function can be any policy, also different from the one being evaluated.

An example of an MC algorithm is reported below from [26], with an  $\epsilon$ -greedy



policy, where the agent takes the greedy action with probability  $1 - \epsilon + \epsilon/|A(s)|$ , and all non-greedy actions with probability  $\epsilon/|A(s)|$ .

---

**Algorithm 1** On-policy, first visit MC algorithm with  $\epsilon$ -greedy exploration, from [26]

---

```

1: ▷ Input:  $\epsilon_0, N, S, A$ 
2: ▷ Output:  $\pi$ 
3: ▷ initialize  $Q(s,a)$  arbitrarily  $\forall s \in S, a \in A(s)$ 
4: ▷ initialize  $R(s,a)$  as a zero vector  $\forall s \in S, a \in A(s)$ 
5: ▷ initialize  $\pi$  as an arbitrary  $\epsilon$ -soft policy
6: while end time not reached do
7:   ▷ run episode following  $\pi$ 
8:   for each pair  $(s,a)$  in the episode do
9:     ▷ set  $R$  as the return following the first occurrence of  $(s,a)$ 
10:    ▷ append  $R$  to  $R(s,a)$ :  $R(s,a)=[R(s,a),R]$ 
11:    ▷  $Q(s,a)=\text{mean}(R(s,a))$ 
12:   end for
13:   for each  $s$  in the episode do
14:     ▷  $a^* = \arg \max_{a' \in A(s)} Q(s, a')$  greedy action
15:     for all  $a \in A(s)$  do
16:       if  $a = a^*$  then
17:         ▷  $\pi(s, a) = 1 - \epsilon + \epsilon/|A(s)|$ 
18:       else
19:         ▷  $\pi(s, a) = \epsilon/|A(s)|$ 
20:       end if
21:     end for
22:   end for
23: end while

```

---

Monte-Carlo methods have been the first RL methods historically developed, even if now the research is moving towards Temporal Difference and Dynamic Programs methods. Anyway, their averaging nature gives them an higher robustness with respect to other methods, and so they are used when other methods fail to converge.

### 3.5 Temporal Difference methods

Temporal Differences methods try to merge the positive things of Monte-Carlo methods (their ability to learn directly from data, without the need of a model)

and of Dynamic Programming methods (not needing a notion of episode, able to learn on-line without the need to wait for the completion of the task).

TD methods have been widely used in the robotics and computer industry.

TD methods further subdivide into On-line and Batch-mode methods algorithms. The on-line algorithms update the state-action value function at each step [30]. Of those algorithms some of the most important ones are the Sarsa (State-Action-Reward-State-Action) [31] and the Q-Learning [32] [33], which are on-line algorithms explained in greater detail in the next sections.

Batch-mode algorithms instead update the state-action value function off-line, using a number of stored sample of past interactions of the form  $(s,a,r,s')$ . Examples of those algorithms are Neaural Fitted Q-iteration (NFQ) [34], and Least Square Policy Iterarion (LSPI) [35], which will also be explained in the next sections with greater detail.

### 3.5.1 Sarsa

Sarsa is a TD RL method on-line and on-policy, which mean it can update the state-action value function in real time, and, to do the update of the state-action value function, uses the same policy that is being evaluated.

At each time step, the state-action value function of the current state-action pair is updated with:

1. the immediate reward experienced in the current state with the action given by the policy  $\pi$  that is being evaluated;
2. the state-action value function obtained in the future state, by choosing there the action related to this future state with  $\pi$ ;

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$$

The above equation can be seen as numerical approximation of the Bellman equation, where  $s$  is the current state,  $a$  is the associated action given by  $\pi$ ,  $s'$  is the future state in the next time step  $a'$  is the action associated to that future state always by  $\pi$ .

This is a way to consider not only the immediate reward after the pair state-action  $s$ - $a$ , but also all future, discounted rewards.

In this case both states and actions are discrete, and so the Q-function can be exhaustively represented by a table.

$\gamma$  represents the discount factor explained before, and  $\alpha$  represents the learning rate, which quantifies the amount of amount of old knowledge to retain during the update of the state-action value function.

For practical purposes, in a way analogue to the  $\epsilon$ , it is preferable to have greater

values of  $\alpha$  during the first visits to a specific state-action pair, and to start lower always more than a certain number of visits to that value, in order to decrease the possible negative effects from disturbances.

$$\alpha = \begin{cases} \alpha_0, & \text{if } N \leq N_\alpha \\ \alpha_0/(N - N_\alpha) & \text{if } N > N_\alpha \end{cases}$$

Where  $\alpha_0$  is the starting learning rate,  $N_\alpha$  is the number of visits to state-action pair after which start decreasing the learning rate, and  $N$  is just the number of visits to that state-action pair.

The specific structure of the decay of the learning rate has been chosen with respect to the application for WEC control, while for more general purposes a slower decay of the learning rate is preferred..

Given that the states and action are discrete, the discount factor is lower than 1, and a learning rate decaying in a proper way [36], the Sarsa algorithm is guaranteed to converge for a number of time steps tending to infinity.

The structure of the Sarsa algorithm is proposed below.

---

**Algorithm 2** Sarsa: on-line, on-policy RL algorithm with discrete and exact states and actions, from [26]

---

```

1: ▷ Input:  $S, A, \alpha_0, N_\alpha, \gamma, \epsilon_0, N_\epsilon$ 
2: ▷ Output:  $\pi$ 
3: ▷ initialize  $Q(s, a)$  arbitrarily  $\forall s \in S, a \in A(s)$ 
4: for each episode do
5:   ▷ Initialize  $N(s, a)$  as an empty matrix
6:   ▷ Initialize  $s$ , randomly or not
7:   ▷ get  $\epsilon$  from an  $\epsilon$ -greedy strategy
8:   ▷ choose a given  $s$ , from the actual policy  $\pi$ 
9:   for each step in the episode do
10:    ▷ take action  $a$  and observe  $s'$  and  $r$ 
11:    ▷ update  $N(s, a) = N(s, a) + 1$ 
12:    ▷ get  $\epsilon$  from an  $\epsilon$ -greedy strategy
13:    ▷ choose  $a'$  given  $s'$ , from the actual policy  $\pi$ 
14:    ▷ get  $\alpha$ , in accordance to the learning rate value strategy used
15:    ▷ update  $Q(s, a) == Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ 
16:    ▷ update  $s = s'$ 
17:    ▷ update  $a = a'$ 
18:   end for
19: end for

```

---

### 3.5.2 Q-learning

One of the most successful RL algorithm, Q-learning was proposed by [32] and [33]. From this algorithm, later were derived modification of it like the Sarsa and the NQF (Neural-Q-Fitting).

The main difference with Sarsa is that Q-learning is an off-policy algorithm, meaning that the policy used for choosing the actions is not necessary the policy used for update the Q function.

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a' \in A(s)} Q(s, a') - Q(s, a)]$$

Like Sarsa, Q-learning is guaranteed to converge for discrete state space and actions, for a given discount factor lower than 1 and for a properly decaying learning rate. The general scheme of the algorithm is reported in Algorithm 3.

---

**Algorithm 3** Q-Learning: on-line, off-policy RL algorithm with discrete and exact states and actions, from [26]

---

```

1: ▷ Input:  $S, A, \alpha_0, N_\alpha, \gamma, \epsilon_0, N_\epsilon$ 
2: ▷ Output:  $\pi$ 
3: ▷ initialize  $Q(s, a)$  arbitrarily  $\forall s \in S, a \in A(s)$ 
4: for each episode do
5:   ▷ Initialize  $N(s, a)$  as an empty matrix
6:   ▷ Initialize  $s$ , randomly or not
7:   for each step in the episode do
8:     ▷ Get  $\epsilon$  for an  $\epsilon$ -greedy strategy
9:     ▷ choose action  $a$  given  $s$  using the  $\epsilon$ -greedy policy
10:    ▷ tank action  $a$ , observe new state  $s'$  and obtained reward  $r$ 
11:    ▷ update  $N(s, a) = N(s, a) + 1$ 
12:    ▷ get  $\alpha$ , in accordance to the learning rate value strategy used
13:    ▷ update  $Q(s, a) == Q(s, a) + \alpha[r + \gamma \max_{a' \in A(s)} Q(s, a') - Q(s, a)]$ 
14:    ▷ update  $s = s'$ 
15:   end for
16: end for

```

---

### 3.5.3 Function Approximation

The simple structure of Sarsa and Q-learning algorithms is able deal only with discrete states and actions.

A lot of practical problems in control engineering instead deal with actions and states living in continuous spaces, like for example the voltage and the current of

an electric motor.

Function approximation is the way approximate the Q function on continuous spaces. This approximated Q function is usually denoted as  $\hat{Q}$ .

The main advantages of using function approximation are:

1. a reduction of the computation cost of the problem, by lowering the overall dimension of state space and of the action space;
2. the possibility to generalize for unseen states, extending the results from nearby states, so decreasing the convergence time.

### Linear Function Approximation

Linear Function Approximation (LFA) is a type of function approximation with all features linearly independent.

With these methods, the state-action value function is usually denoted as:

$$Q(s, a) = \hat{Q}(s, a) = \Phi(s)^T \Theta_{:,a}$$

where  $\Phi$  is the vector of arbitrary, linearly independent, usually nonlinear basis function and  $\Theta$  is the correspondent weight matrix.

The weight matrix has a number of columns equal to the number of actions, so that  $\Theta_{:,a}$  represents the  $a^{th}$  column of  $\Theta$ .

$\Phi$  and  $\Theta$  have the same number of rows,  $J$ . For practical purposes, usually  $J \ll ||S||$  in order to reduce the overall complexity of the problem.

Examples of basis functions are tabular and radial.

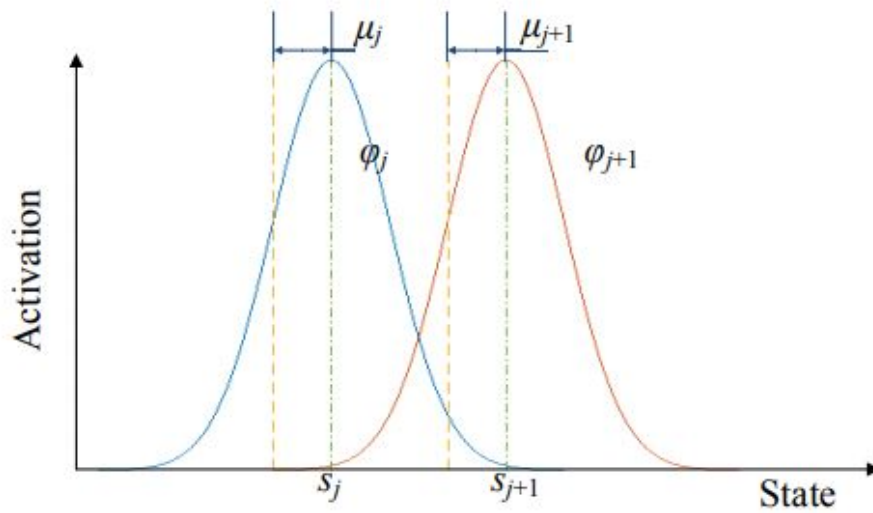
Tabular basis function is the simplest LFA and consist in a separate weight for each possible state-action pair, using as basis functions for the states:

$$\phi_j(s) = \begin{cases} 1, & \text{if } s == s_j \\ 0 & \text{if } s \neq s_j \end{cases}$$

The exact state  $s_j$  has correspondent weights for the basis functions equal to 1 for the  $j^{th}$  and equal to 0 for all the others. This means that if the states were discrete from the beginning, the approximated value function is equal to the exact one, since there has been no dimensionality reduction ( $J = ||S||$ ), thus not obtaining any positive effects on the computational cost.

Unlike tabular basis, with Gaussian radial basis functions (RBFs) [37] it is possible to have a continuous representation of the state space.

The activation features of the RBFs are  $J$  gaussian radial functions centered around the various  $s_j$  with given bandwidths  $\mu_j$ .

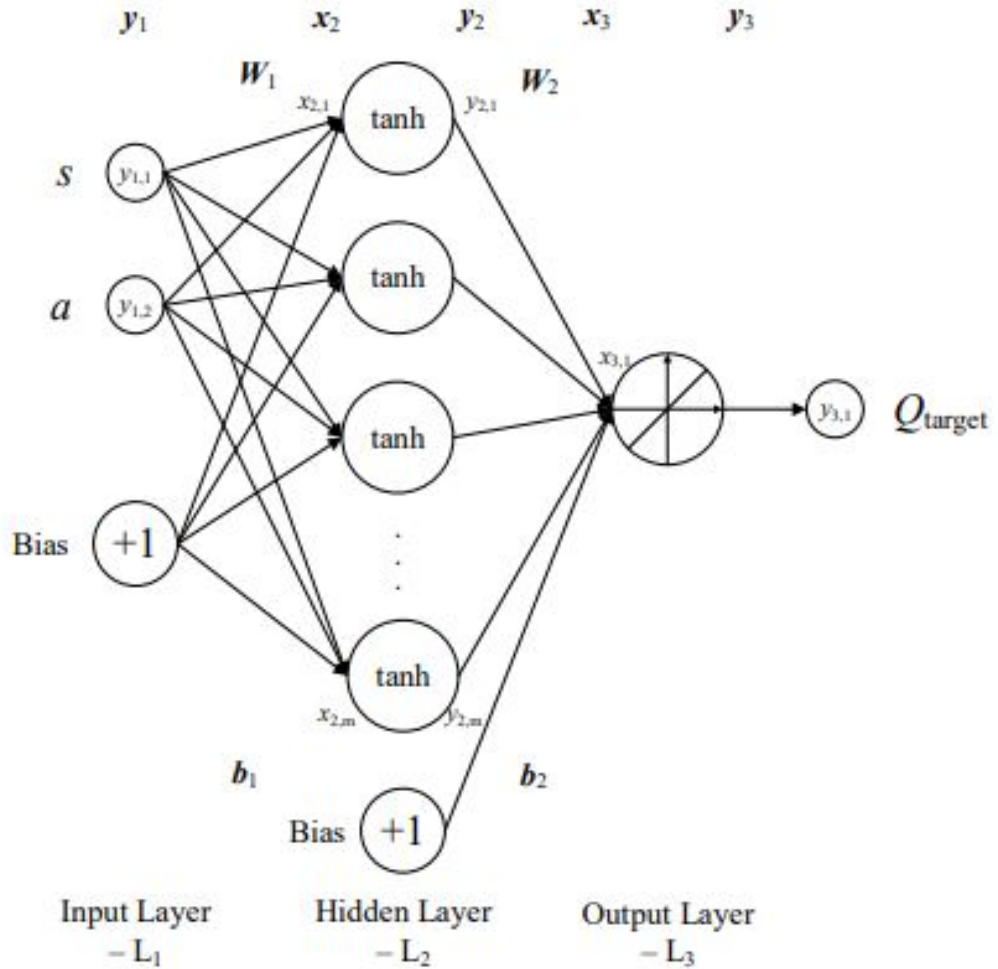


**Figure 3.3:** Example of activation features of radial basis function for 2 dimensions. Taken from [19]

With multidimensional state spaces, the bandwidths can be different for each direction, thus generating an elliptical basis function [28]. In some cases, it may be useful to add a constant bias term, in order to lift up the activation function of a given amount [19].

## Neural Function Approximation

In recent years, the research focused on the application of non-linear basis functions, for example the ones coming from machine learning techniques, like the neural fitted Q iteration (NFQ) from [34].



**Figure 3.4:** Example feed-forward NN used for function approximation of the Q function (with 1 state and 1 action only). Taken from [19]

In figure 3.4 is presented an activation neural network (ANN) with a single hidden layer with  $m$  neurons.

The final output is obtained with forward propagation of the inputs, denoting the single neuron activation function as  $o = f(i)$ , where  $o$  is the output, and  $i$  is the input, and the training is done by setting the weight matrices  $W_1, W_2$

### 3.5.4 Sarsa with LFA

The general structure of Sarsa algorithm with LFA is presented below.

It is to note that even if the state space is continuous, for the updating of the

exploration rate and of the learning rate, a tabular approach for updating the matrices of the number of visits  $N(s_d, a)$  is used, where  $s_d$  is the closest discrete state to the actual state  $s$ .

---

**Algorithm 4** Sarsa: on-line, on-policy RL algorithm with LFA, from [26]

---

```

1: ▷ Input:  $A, \alpha_0, N_\alpha, \gamma, \epsilon_0, N_\epsilon, S_d, \phi$ 
2: ▷ Output:  $\pi$ 
3: ▷ initialize  $\Theta$  arbitrarily
4: for each episode do
5:   ▷ Initialize  $N(s, a)$  as an empty matrix
6:   ▷ Initialize  $s$ , randomly or not
7:   for  $\forall a \in A(s)$  do
8:     ▷ Get  $\hat{Q}(s, a) = \phi(s)^T \Theta_{:,a}$ 
9:   end for
10:  ▷ Get  $\epsilon$  with an  $\epsilon$ -greedy strategy
11:  ▷ Select  $a$  from the given  $s$ , following an  $\epsilon$ -greedy policy
12:  for each step in the episode do
13:    ▷ Get the closest discrete state  $s_d$  to the actual state  $s$ 
14:    ▷ Update  $N(s_d, a) = N(s_d, a) + 1$ 
15:    ▷ Get the values of  $\epsilon$  and  $\alpha$  with the number of visit to the state and
state-action pair
16:    ▷ Take action  $a$ , observe  $r$  and  $s'$ 
17:    ▷ Initialize  $\delta = r - \hat{Q}(s, a)$ 
18:    for  $\forall a \in A(s')$  do
19:      ▷ Get  $\hat{Q}(s', a) = \phi(s')^T \Theta_{:,a}$ 
20:    end for
21:    ▷ Get  $a'$  given  $s'$ , by following the  $\epsilon$ -greedy policy
22:    ▷ Update  $\delta = \delta + \gamma \hat{Q}(s', a')$ 
23:    ▷ Update  $\Theta_{s,a} + 1 = \Theta_{s,a} + \alpha \delta$ 
24:    ▷ Update  $s = s'$ 
25:    ▷ Update  $a = a'$ 
26:  end for
27: end for

```

---

### 3.5.5 Q-learning with LFA

Similarly to Sarsa, also Q-learning can be casted with LFA, with the same approximation of discretising the state space in order to have a tabular update of the learning and exploration rates.



---

**Algorithm 5** Q-learning: on-line, off-policy RL algorithm with LFA, from [26]

---

```

1: ▷ Input:  $A, \alpha_0, N_\alpha, \gamma, \epsilon_0, N_\epsilon, S_d, \phi$ 
2: ▷ Output:  $\pi$ 
3: ▷ initialize  $\Theta$  arbitrarily
4: for each episode do
5:   ▷ Initialize  $N(s, a)$  as an empty matrix
6:   ▷ Initialize  $s$ , randomly or not
7:   for each step in the episode do
8:     for  $\forall a \in A(s)$  do
9:       ▷ Get  $\hat{Q}(s, a) = \phi(s)^T \Theta_{:,a}$ 
10:    end for
11:    ▷ Get  $\epsilon$  with an  $\epsilon$ -greedy strategy
12:    ▷ Select  $a$  from the given  $s$ , following an  $\epsilon$ -greedy policy
13:    ▷ Get the closest discrete state  $s_d$  to the actual state  $s$ 
14:    ▷ Update  $N(s_d, a) = N(s_d, a) + 1$ 
15:    ▷ Get the value of  $\alpha$  with the number of visit to the state-action pair
16:    ▷ Take action  $a$ , observe  $r$  and  $s'$ 
17:    ▷ Initialize  $\delta = r - \hat{Q}(s, a)$ 
18:    for  $\forall a \in A(s')$  do
19:      ▷ Get  $\hat{Q}(s', a) = \phi(s')^T \Theta_{:,a}$ 
20:    end for
21:    ▷ Update  $\delta = \delta + \gamma \hat{Q}(s', a)$ 
22:    ▷ Update  $\Theta_{s,a} + 1 = \Theta_{s,a} + \alpha \delta$ 
23:    ▷ Update  $s = s'$ 
24:  end for
25: end for

```

---

### 3.5.6 Neural Fitted Q-iteration

Neural Fitted Q-iteration (NFQ) was firstly described by Riedmiller [34], who wanted to combine both the positive aspect of Q-learning and Neural Networks (NN) [19]: a robust and efficient RL algorithm, with a powerful tool for representing non-linearities.

The output of the NN is the actual Q function that is being estimated:

$$\hat{Q}_{target}(s, a) = r + \gamma \hat{Q}(s', a')$$

In this way, it is possible to directly define the squared error of the NN as:

$$e = (\hat{Q}(s, a) - \hat{Q}_{target}(s, a))^2$$

which is used to apply the backward propagation technique (i.e. adjusting iteratively 1 parameter vector of the NN at the time with the goal of minimizing the error [38]).

Since solving in real time the training of the NN is not feasible, an algorithm in batch mode is usually applied: the update of the Q function is made offline, after having stored a certain number of samples of the form  $(s, a, s', r)$ . This is repeated for a certain number of episodes.

Below is schematized the general scheme of a NFQ algorithm, where  $\mathbf{S}_{:,j}$  represents the  $j^{th}$  column of the sample matrix  $\mathbf{S}$  and  $\mathbf{W}$  is the matrix containing the neural network's parameter weights.

### 3.5.7 Least Square Policy Iteration

Least Square Policy Iteration, LSPI, is an off-line, on policy, RL method developed by [35].

This technique derives from the Policy Iteration, a technique used to find the optimal policy for an MDP [40].

In LSPI, not having a model for the MDP, the algorithm has to rely on the stored data, of the form  $(s, a, r, s')$  and can so approximate the actual solution to the policy iteration.

In Least Square Temporal Difference Q, LSTDQ, the state-action value function or Q-function is approximated for each state-action pair by using the stored data.

#### Policy Iteration

In the standar Policy Iteration, unlike in the TD methods, the policy is stored in a memory structure independent of the Q-function [26].

Convergence to the optimal policy is given by alternating a stage of policy evaluation, where the Q-function is evaluated from the current policy by solving a linear system of Bellman equations, and a stage of policy improvement, where the current policy is updated in accordance to the new Q-function [35].

This process, shown in the figure below, is repeated until the policy converges, which is guaranteed for a tabular representation of the Q-function.

The stage of policy evaluation is referred to as the critic and the stage of policy improvement as the actor.

**Algorithm 6** Neural Fitted Q-iteration: off-line, off-policy RL algorithm with NFA, from [34]

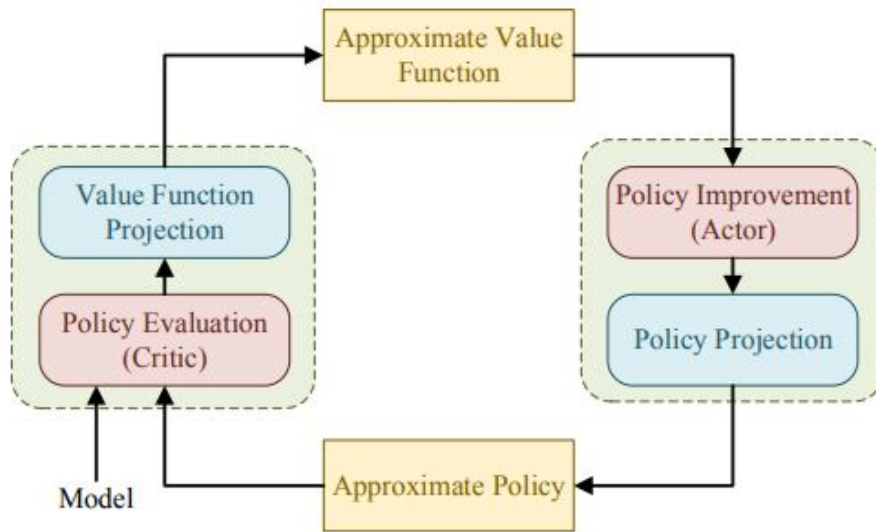
---

```

1: ▷ Input:  $A, N_\alpha, \gamma, \epsilon_0, N_\epsilon, S_d$ 
2: ▷ Output:  $\pi$ 
3: ▷ Initialize  $\mathbf{W}$  arbitrarily
4: ▷ Initialize  $\mathbf{S}$  as an empty matrix
5: ▷ Get the number of actions  $|A|$ 
6: for each episode do
7:   for each step in the episode do
8:     ▷ Observe current state  $s$ 
9:     ▷ Get corresponding nearest state  $s_d$ 
10:    ▷ Get  $\epsilon$  with an  $\epsilon$ -greedy strategy
11:    for  $\forall a \in A$  do
12:      ▷ Get  $\hat{Q}(s, a) \approx f(s, a)$  (input-output relation of the NN)
13:    end for
14:    ▷ Select the action  $a$  with the  $\epsilon$ -greedy policy
15:    ▷ Apply action  $a$ , observe  $r, s'$ 
16:    ▷ Store in  $\mathbf{S}$  the sample  $(s, a, r, s')$ 
17:    if end episode then
18:      ▷ Get the number of samples  $n_s$ 
19:      ▷ Initialize  $\hat{Q}_{target} = 0$  with size  $(n_s, |A|)$ 
20:      for  $k = 1 : k_{max}$  do
21:        for each sample  $i$  in  $\mathbf{S}$  do
22:          for  $\forall a \in A$  do
23:            ▷ Get  $\hat{Q}(s', a) \approx f(\mathbf{S}(i, 4), a)$ 
24:          end for
25:          ▷ Get  $\hat{Q}_{target}(i) = \mathbf{S}(i, 3) + \gamma \max_{a' \in A} \hat{Q}(s', a')$ 
26:        end for
27:        ▷ Train the NN as in [39]
28:      end for
29:    end if
30:  end for
31: end for

```

---



**Figure 3.5:** Representation of the scheme of a Policy Iteration algorithm. Taken from [19]

The exact state-action function  $Q^\pi(s, a)$  is approximated by  $\hat{Q}^\pi(s, a, w)$  where  $w$  represents the adjustable parameters of the state-action value function approximator.

The same is done for the actual policy  $\pi(s)$  which gets approximated by  $\hat{\pi}(s, \theta)$  where  $\theta$  represents the adjustable parameters of the policy approximator.

Since only the adjustable parameters have to be stored, this is less memory demanding than storing the entire policy or the entire state-action value function.

[41] and [42] show how the approximated policy converges to the actual correct policy when the errors of the approximators are bounded and decreasing to zero.

### Least-squares fixed point approximation

Even if the state-action value function can be approximated with the Bellman equations, and doing so the function would be more stable and predictable, in practice it is preferred to use the least-squares fixed point method in the context of learning [42].

This method was developed by [43] and is based on the observation that the correct state-action value function  $Q^\pi$  of the policy  $\pi$  is a fixed point of the Bellman operator  $T_\pi$  [35].

$$T_\pi Q^\pi = Q^\pi$$

For this reason, an approximator in order to be good must be a fixed point under the Bellman operator, moreover from the construction it also has to lie in the space spanned by the basis functions.

The latter condition is always true by definition for  $Q^\pi$ , but not for  $T_\pi Q^\pi$ , which must then be projected onto the space spanned by the the basis functions, for example ([35]) with an orthogonal projection which minimizes the  $L_2$  norm  $(\Phi^T(\Phi\Phi)^{-1}\Phi^T)$ . The problem then becomes that of finding an approximated state-action value function  $\hat{Q}^\pi$  that is invariant under an application of the Bellman operator followed by an orthogonal projection.

$$\hat{Q}^\pi = (\Phi^T(\Phi\Phi)^{-1}\Phi^T)T_\pi\hat{Q}^\pi$$

Which expanding the Bellman operator becomes:

$$\hat{Q}^\pi = (\Phi^T(\Phi\Phi)^{-1}\Phi^T)(R + \gamma P\Pi_\pi\hat{Q}^\pi)$$

Using linearly independent features for the approximator, the column space of  $\Phi$  is well defined and so the problem can be casted into a  $K \times K$  linear system, where  $K$  is the total number of basis functions.

$$\Phi^T(\Phi - \gamma P\Pi_\pi\Phi)\omega^\pi = \Phi^T R$$

For a finite number of any  $\gamma$  and for any  $\Pi_\pi$ , the solution is guaranteed to exist [44].

$$\omega^\pi = (\Phi^T(\Phi - \gamma P\Pi_\pi\Phi))^{-1}\Phi^T R$$

For controlling the distribution of the approximation error, usually a weighted projection is used.

Indicating as  $\mu$  is the probability distribution over the pair (s,a) and  $\Delta_\mu$  is the diagonal matrix with the projection weights  $\mu(s, a)$ .

The final weighted least-square approximator is given by [35]:

$$\omega^\pi = (\Phi^T\Delta_\mu(\Phi - \gamma P\Pi_\pi\Phi))^{-1}\Phi^T\Delta_\mu R$$

## Least-squares temporal difference learning for the state-action value function

Like with all TD methods, there is no model of the process and it has to be learned with samples.

Considering  $K$  linearly independent basis functions, the problem requires to find the parameters  $\omega^\pi$  of  $\hat{Q}^\pi = \Phi\omega^\pi$ . The exact values of the parameters can be obtained by following the linear system of equations [35]:

$$A\omega^\pi = b$$

where  $A = \Phi^T \Delta_\mu (\Phi - \gamma P \Pi_\pi \Phi)$  and  $b = \Phi^T \Delta_\mu R$ .

In order to learn  $A$  and  $b$ , samples of the form  $(s, a, r, s')$  have to be stored, and, after having collected  $N$  samples, the values of  $A$  and  $b$  can be approximated with  $\tilde{A}$  and  $\tilde{b}$  as [35]:

$$\tilde{A} = \frac{1}{N} (\tilde{\Phi}^T (\tilde{\Phi} - \gamma \tilde{P} \tilde{\Pi}_\pi \tilde{\Phi}))$$

$$\tilde{b} = \tilde{\Phi}^T \tilde{R}$$

with

$$\tilde{\Phi}^T = [\Phi(s_1, a_1)^T \dots \Phi(s_n, a_n)^T \dots \Phi(s_N, a_N)^T]$$

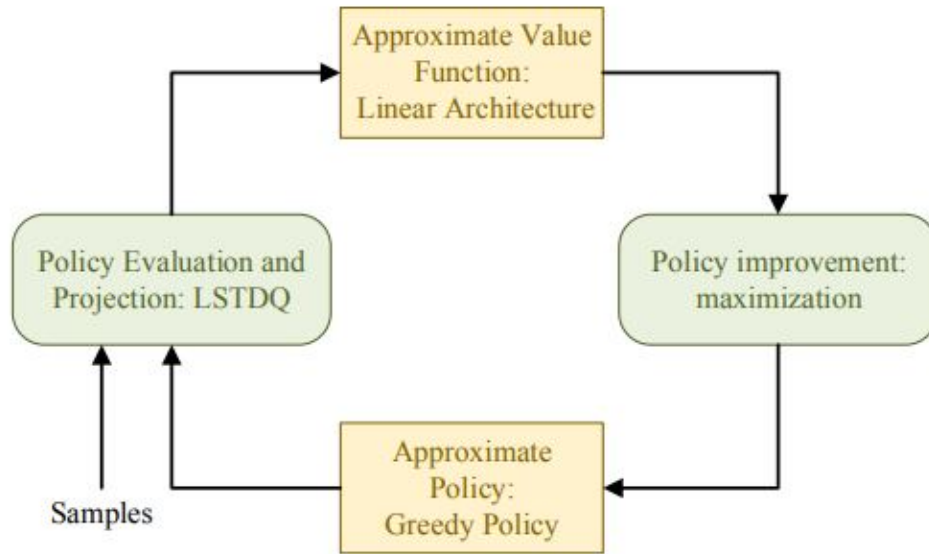
$$\tilde{P} \tilde{\Pi}_\pi \tilde{\Phi} = [\Phi(s'_1, \pi(s'_1))^T \dots \Phi(s'_n, \pi(s'_n))^T \dots \Phi(s'_N, \pi(s'_N))^T]$$

$$\tilde{R} = [r_1 \dots r_n \dots r_N]$$

With an infinite number of samples ( $N \rightarrow \infty$ ), the sample approximated  $\tilde{A}$  and  $\tilde{b}$  are guaranteed to converge to the actual  $A$  and  $b$ .

The policy  $\pi$  used is usually the classic  $\epsilon$ -greedy policy.

It is to be noted that the algorithm requires the inversion of the matrix  $\tilde{A}$  which, specially at the beginning of the algorithm, is not full rank, and so recursive least-squares techniques are needed to compute the inverse of  $\tilde{A}$  recursively [35].



**Figure 3.6:** Representation of the scheme of an LSPI. Taken from [19]

Using together linear function approximation, approximate policy iteration, least-squares fixed-point approximation and least-squares temporal difference learning for the state-action function it is possible to develop an LSPI algorithm, which is described below.

---

**Algorithm 7** LSPI: off-line, on-policy RL algorithm with LFA, from [35]
 

---

```

1: ▷ Input:  $A, \alpha_0, N_\alpha, \gamma, \epsilon_0, N_\epsilon, S_d, \phi, \delta$ 
2: ▷ Output:  $\pi, \Theta$ 
3: ▷ Initialize  $\Theta$  arbitrarily
4: ▷ Initialize  $\Theta_0 = 0$ 
5: ▷ Initialize  $\mathbf{S}$  as an empty matrix
6: ▷ Get the number of actions  $|A|$ 
7: for each episode do
8:   for each step in the episode do
9:     ▷ Observe current state  $s$ 
10:    ▷ Get corresponding nearest state  $s_d$ 
11:    ▷ Get  $\epsilon$  with an  $\epsilon$ -greedy strategy
12:    for  $\forall a \in A$  do
13:      ▷ Get  $\hat{Q}(s, a) = \phi(s)^T \Theta_{:,a}$ 
14:    end for
15:    ▷ Select the action  $a$  with the  $\epsilon$ -greedy policy
16:    ▷ Apply action  $a$ , observe  $r$  and the new continuous state  $s'$ 
17:    ▷ Store in  $\mathbf{S}$  the sample  $(s, a, r, s')$ 
18:    if condition for off-line weight matrix update met then
19:      ▷  $\Theta_1 \leftarrow \Theta_0$ 
20:      while  $\|\Theta - \Theta_1\| \geq \delta$  do
21:        ▷  $\Theta_1 \leftarrow \Theta$ 
22:        ▷  $\tilde{A} \leftarrow \mathbf{0}$ 
23:        ▷  $\tilde{b} \leftarrow \mathbf{0}$ 
24:        for each sample  $(s, a, r, s')$  in  $\mathbf{S}$  do
25:          ▷  $\tilde{A} \leftarrow \tilde{A} + \phi(s)(\phi(s) - \gamma\phi(s', \pi(s')))^T$ 
26:          ▷  $\tilde{b} \leftarrow \tilde{b} + \phi(s)r$ 
27:          for  $\forall a \in A$  do
28:            ▷  $\Theta_{:,a} \leftarrow \tilde{A}^{-1}\tilde{b}$ 
29:          end for
30:        end for
31:      end while
32:    end if
33:  end for
34: end for

```

---



## Chapter 4

# Implementation of Reinforcement Learning algorithms on MATLAB

In the following chapters, various RL algorithms will be developed for the WEC control problem in more ways using MATLAB codes.

For this implementations, an ideal situation will be considered, with the WSI being totally linear and simulated in Simulink environment. Moreover the WSI will be considered as an equivalent Mass-spring-damper (MSD) model with parameters independent from the frequency of excitation, in this way the control problem reduces to the choice of the equivalent damping coefficient and stiffness coefficient of the WEC, which will be the actual controllable states, that should be chosen such that they maximize the converted energy.

At first an off-line version of the algorithms will be developed, where the power is computed before for each possible state, and the algorithms just uses this stored data. This first versions will prove the viability in principle of the algorithm and show it converges in a reasonable amount of time.

Next, an on-line version of the algorithms will be implemented through a Simulink scheme, so that the algorithm computes by himself the power of the state he is in with the information he receives from the sensors of its own position and velocity. For this first versions, a single exciting wave is considered with a period of 5 second and an amplitude of 1 meter, remembering that also non monochromatic states of excitation can be dealt with an equivalent monochromatic wave with an equivalent period and an equivalent amplitude.

## 4.1 MSD model: Off-line versions of the algorithms

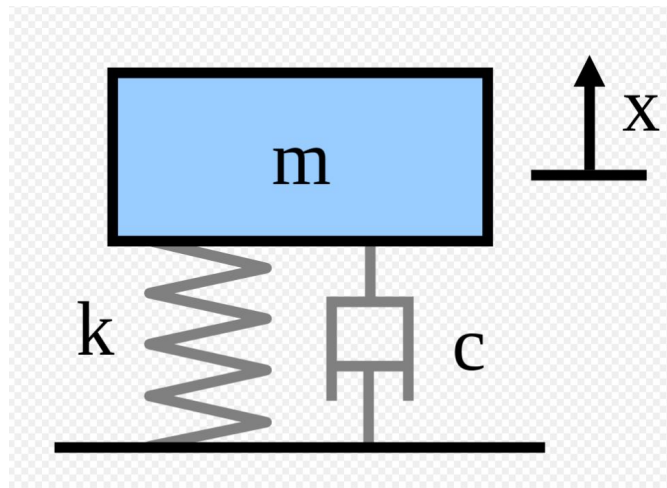
The first thing to do for creating an off-line version of the algorithms is to create the data the algorithms will be trained on.

To do so, the system is defined in MATLAB and Simulink environments and it is simulated in all possible states. The parameters of the MSD representing the WEC are fixed and with values:

$m = 20kg$  mass of the MSD

$k = 10N/m$  spring coefficient of the MSD

$c = 15Ns/m$  damping coefficient of the MSD



**Figure 4.1:** MSD model. Taken from Wikipedia

By introducing in MATLAB the variable  $s$  as the Laplace variable, the differential equation of the system in the time domain becomes a polynomial equation in the Laplace domain and the system can be modeled with a transfer function (II order prototype) in the Laplace domain.

ODE in time domain:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = f_{ext}(t)$$

Polynomial equation in Laplace domain:

$$ms^2X(s) + csX(s) + kX(s) = F_{ext}(s)$$

with  $X(s) = L[x(t)]$  and  $F_{ext}(s) = L[f_{ext}(t)]$ , where  $L$  represents the Laplace operator.

Defining the transfer function of the MSD:  $TF(s) := \frac{X(s)}{F_{ext}(s)}$

$$TF(s) = \frac{1/m}{s^2 + cs/m + k/m}$$

The external force  $f_{ext}(t)$  is given by the force the wave exerts on the WEC minus the control force.

$$f_{ext}(t) = f_{wave}(t) - f_{ctrl}(t)$$

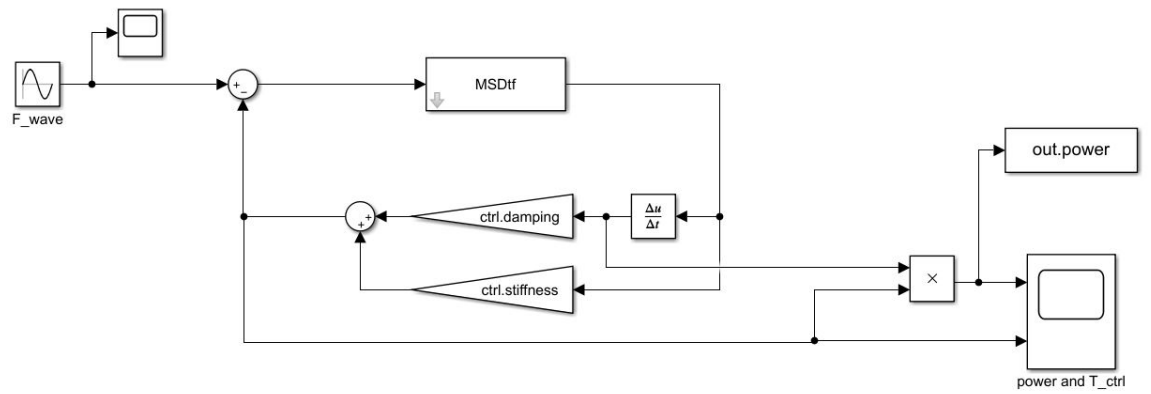
And the control force is given by:

$$f_{ctrl}(t) = c_{ctrl}\dot{x}(t) + k_{ctrl}x$$

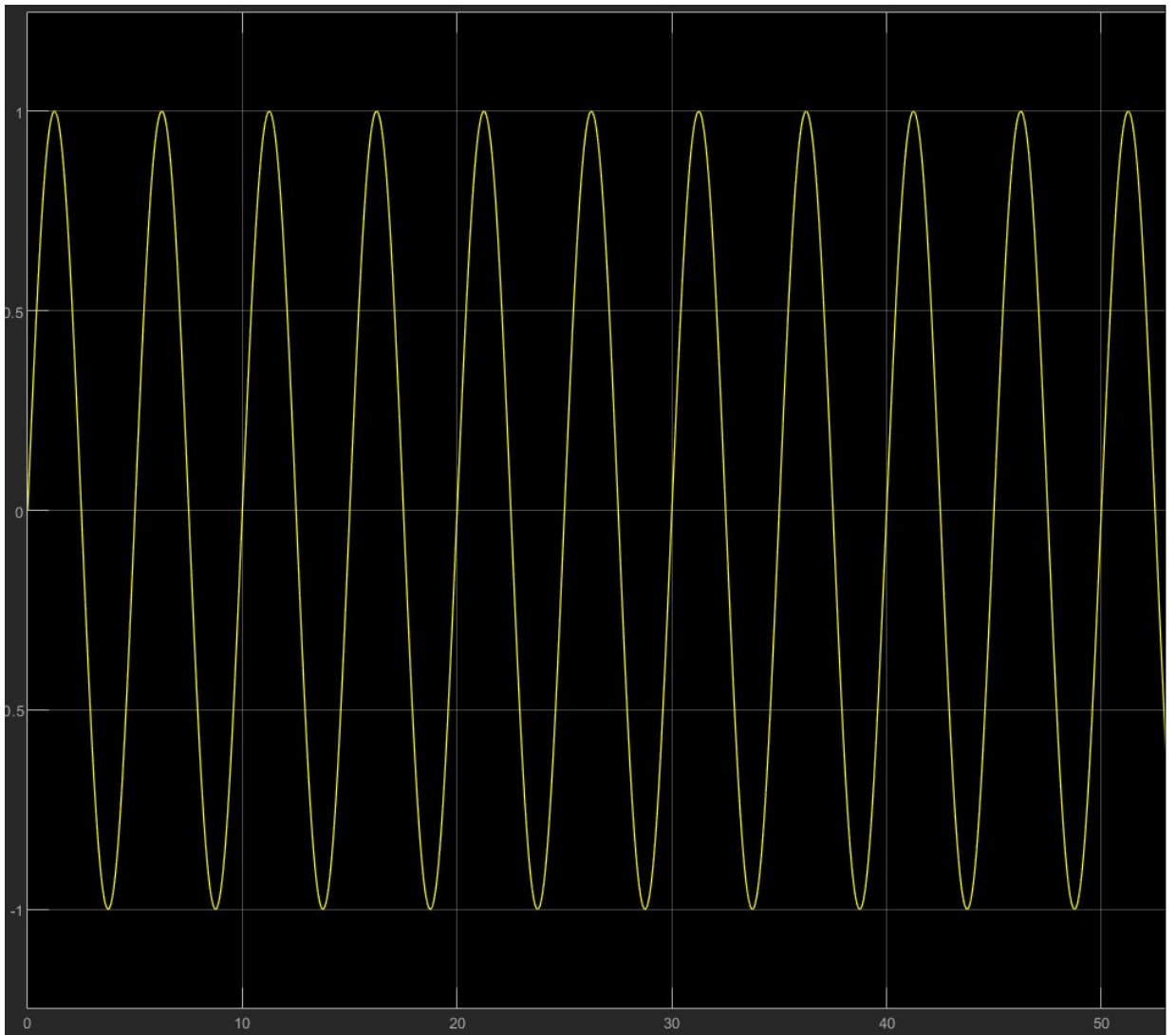
where  $c_{ctrl}$  and  $k_{ctrl}$  are the parameters to be controlled in order to maximize the average power taken off by the PTO.

Considering that the period of the exciting wave is 5 seconds, it has been chosen a simulation time of 100 seconds, time over which the instant power is averaged to get an average power. In this way the average power is calculated over 20 cycles to avoid any possible error due to discrete nature of the simulation.

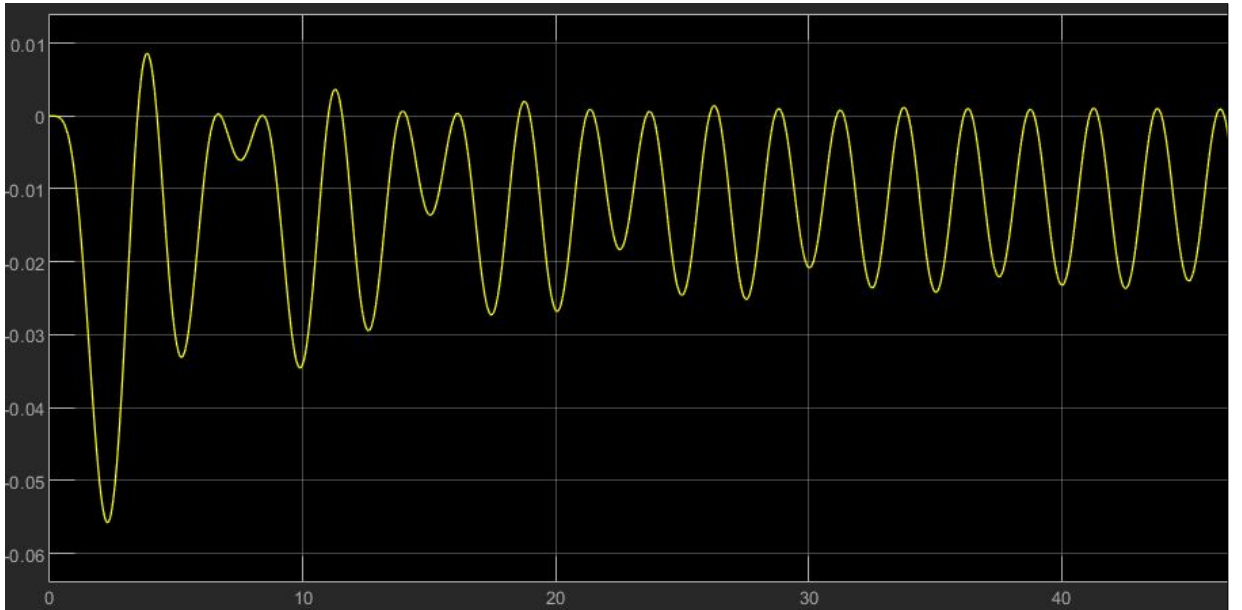
For the time step, which instead is needed much lower than the exciting period, a fixed-step size value of 0.001 seconds has been set, which may be way too low and could potentially increase too much the computational effort of the simulator, but considering it is a totally linear system and here is only averaging the power, the total computational effort remains reasonably bounded.



**Figure 4.2:** Simulink scheme used for simulating the system



**Figure 4.3:** Exciting wave function in time



**Figure 4.4:** Power function in time example. Particular configuration with average power negative (the WEC gives energy to the sea)

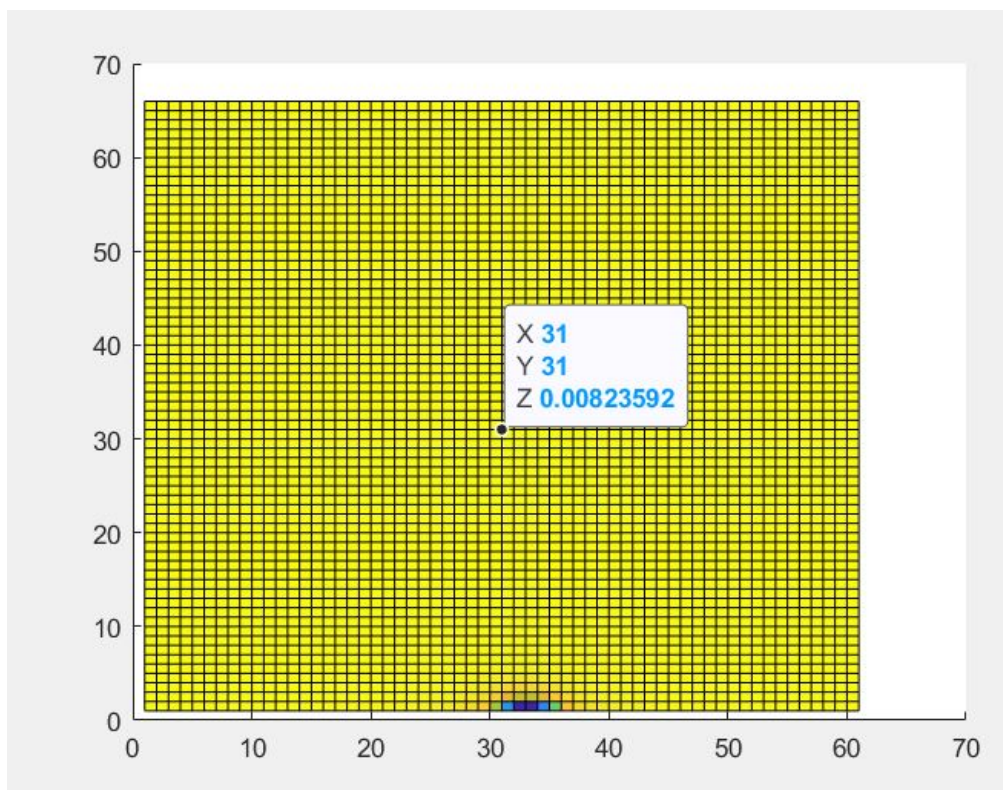
Next is to be chosen how to discretize the range of values of the control damping and of the control stiffness. The control damping value is bounded between  $[-14.9; 50]$   $Ns/m$ , while the stiffness damping value is bounded between  $[-9.9; 50]$   $N/m$ .

For both values a fixed step size has been used for the discretization. This fixed value will affect a lot the training time of the future algorithm: considering that the control damping and the control stiffness will be the actual states of the RL algorithms, their discretization determines how many possible states there are to learn.

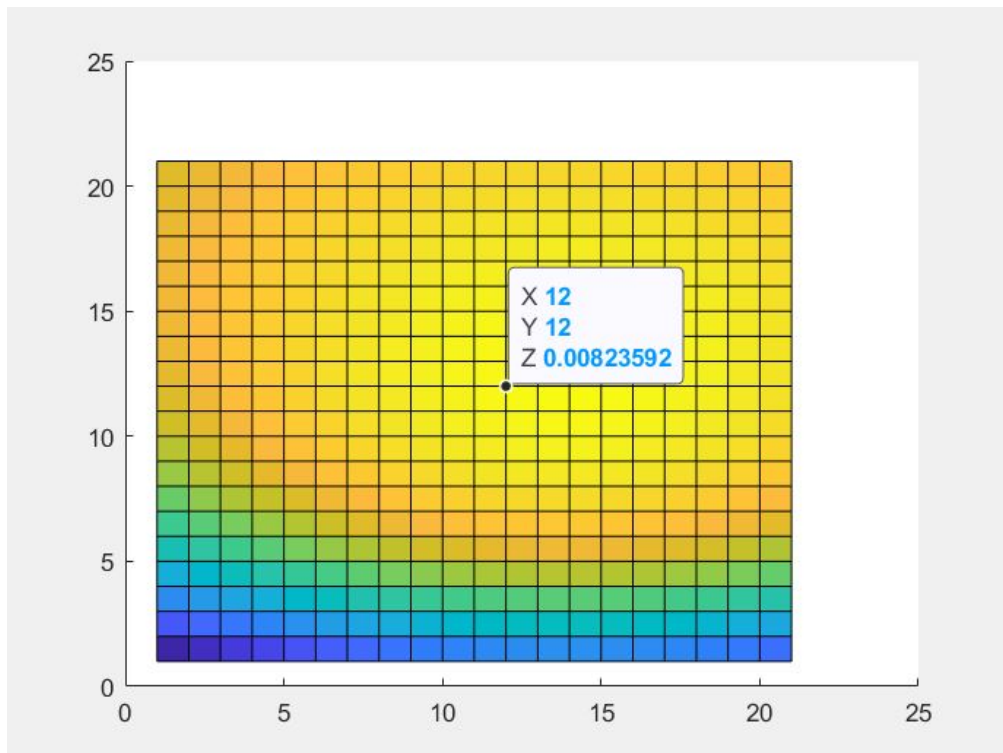
At first it was chosen a discrete step of 1 for those values, giving 66 possible values for the control damping and 61 possible values for the control stiffness, and so a total of 4026 states. This gave no problem for the calculation of the power in each state, but when passing to the actual Q-learning algorithm, the time for convergence to an optimal policy resulted way too high to be of a practical interest. Having considered later on also the time for convergence of the algorithms, in order to have a feasible time for convergence, a discrete step of 5 has been chose, thus having 14 possible values for the control damping and 13 possible values for the control stiffness, in total 182 possible states.

For each of the 182 combination of states (control damping and control stiffness), the average power has been calculated from the simulation, and its value has been stored in the power.mat file.

After having generated the data, a first preliminary analysis was made directly on this data by just plotting on MATLAB the average generated power as a function of the states.

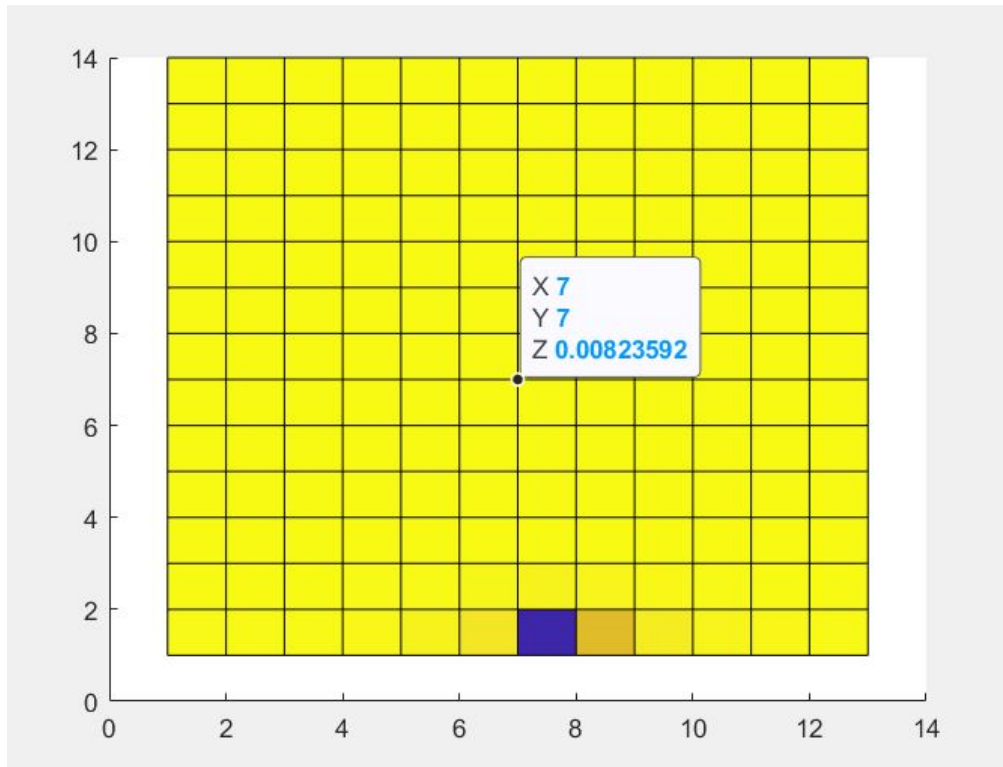


**Figure 4.5:** Power function with 1 as discrete step



**Figure 4.6:** Power function with 1 as discrete step, zoomed in the central section

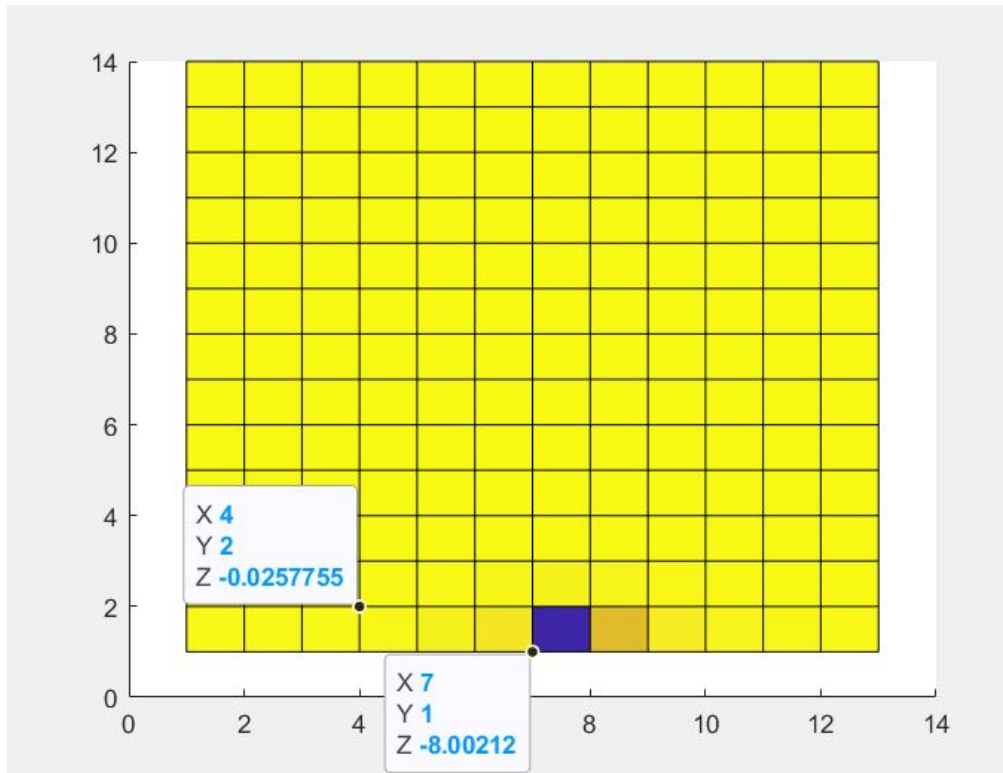




**Figure 4.7:** Power function with 5 as discrete step

As it can be seen from the figures, the optimal point (maximum absorbed power) coincided in both the 2 discretization, with a value of absorbed power of around  $0.0082W$  and in correspondance of a control damping (Y value) of  $15Ns/m$  and of a control stiffness (X value) of  $20N/m$ . Between the values of the parameters and the X-Y coordinate of the plots there is just a linear mapping to rescale their ranges.

It is interesting to note the the power function is actually a concave function with respect to the control variables (as can be seen clearly in figure 4.6), thus indicating that in theory the convergence of an RL algorithm can be achieved pretty fast, specially if with a preliminar initialization. Moreover for some particular combinations of the control variables the net power results to be negative (the WEC gives energy to the sea).



**Figure 4.8:** Particular combinations where the power is given from the WEC to the see

The obtained values of the power have been stored inside 'power.mat' a .mat file that will then be loaded by the off-line algorithms to see the power in each state instead of simulating the state scenario and then calculating the power.

### 4.1.1 Off-line Q-learning

With the data already available, it has been possible to develop a first off-line RL algorithm, in particular a Q-learning one.

#### State space

The state space used has been the same one used to calculate in the previous step all the values of the power in each state, a 2 dimensional (control damping and control stiffness) discrete state, discretized with a common step value of 1 for both dimensions. This space is been defined on MATLAB creating 2 vectors with the

'linspace' command:

1.  $s1$  ranging from 1 to 14, representing the values of control damping ranging from  $-14.9Ns/m$  to  $50Ns/m$
2.  $s2$  ranging from 1 to 13, representing the values of control stiffness ranging from  $-9.9N/m$  to  $50N/m$

For each possible pair  $(s1, s2)$  the absorbed power is already available in the 'power.mat' file, in position  $(s1, s2)$ , due to how the discretization of the states has been made up.

### Action Space

The action space has been defined a discrete, state dependent set. The state dependency is used to implicitly define the constraints: the set of all possible actions reduces in a state at the border of the state space ( $s1 = 1$  or  $s1 = 14$ ,  $s2 = 1$  or  $s2 = 13$ ) in order to avoid the selection of an action that would bring the actual states, and so the control variable, outside of the allowed range.

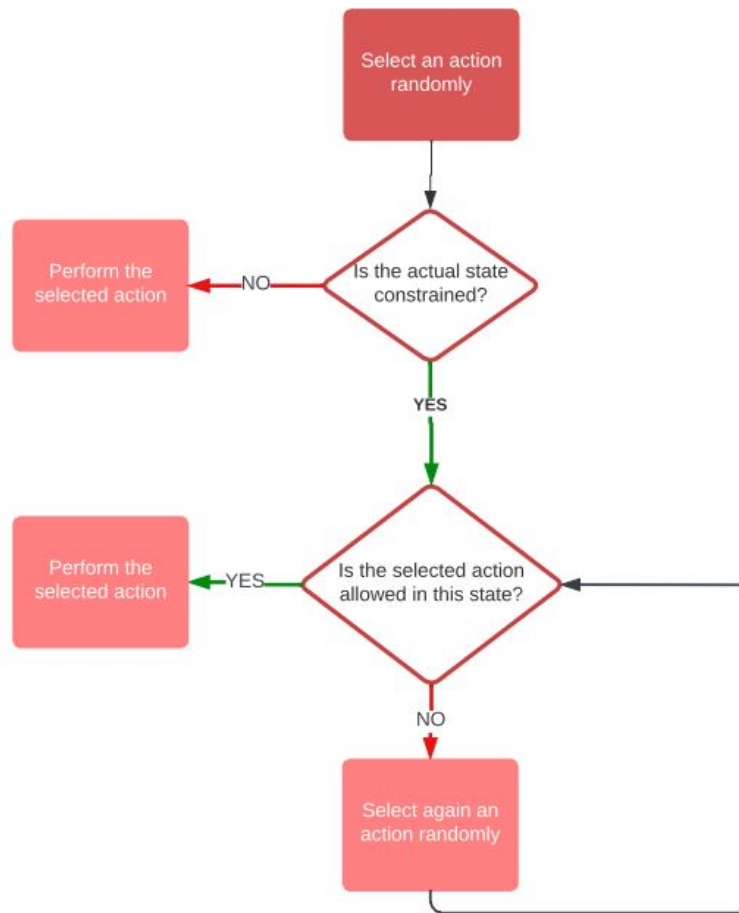
In a constraint-free point of the state space, 5 actions can be taken:

1. Do nothing, and so the new states will be exactly the actual states.
2. Increase the control damping, so that the new  $s1$  will be the old  $s1$  plus 1.
3. Decrease the control damping, so that the new  $s1$  will be the old  $s1$  minus 1
4. Increase the control stiffness, so that the new  $s2$  will be the old  $s2$  plus 1.
5. Decrease the control stiffness, so that the new  $s2$  will be the old  $s2$  minus 1

Instead in a constrained point (points lying on the sides of the state space), one action is not allowed (for example, if  $s1=1$ ,  $s1$  cannot be decreased). The 4 points at the corners are the only ones where 2 actions cannot be taken.

For the MATLAB implementation, a discrete vector 'a' has been used, with 5 values ranging from 1 to 5.

Since a constraint violating action can be taken only when the controller is choosing the action to perform randomly (exploration phase) using an uniform probability distribution for all actions, the constraints can be easily implemented by an if-while combination.

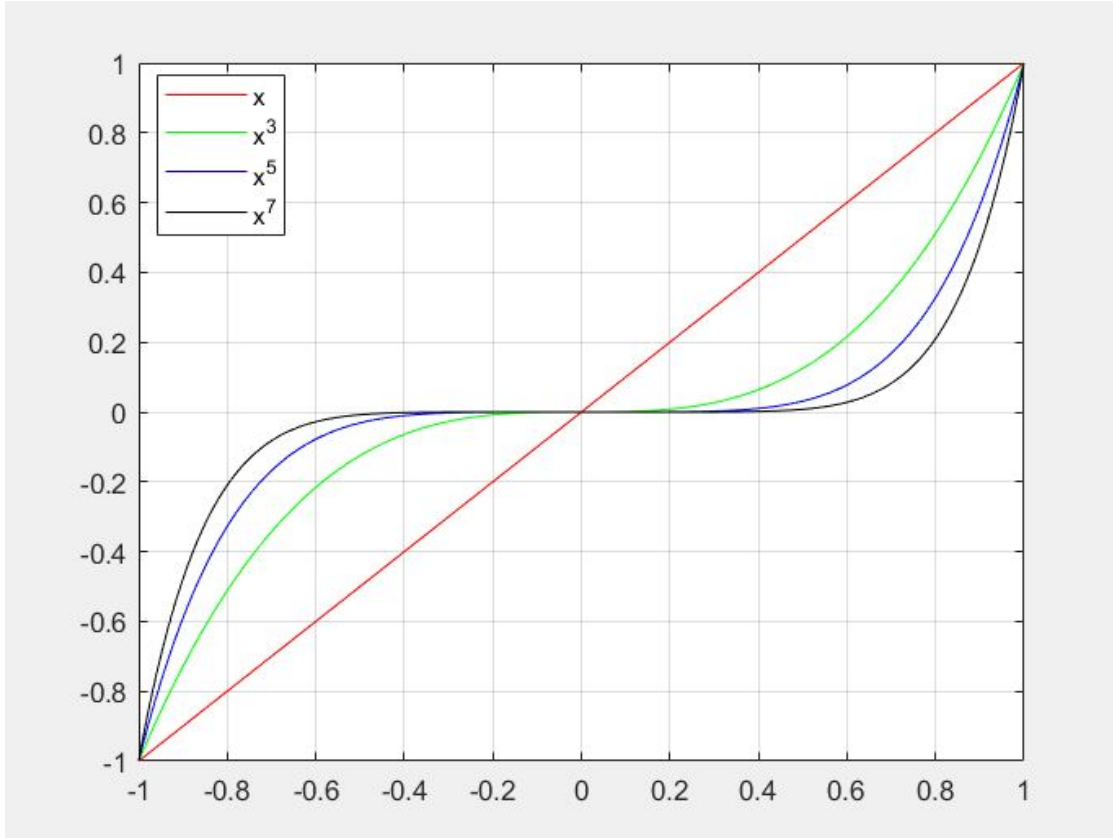


**Figure 4.9:** Flowchart of the action selection process

### Power normalization

From the literature [19] it can be seen that a normalization of the power can greatly increase the convergence speed of an RL algorithm. The standard choice is that of dividing each single power by the maximum absolute value possible of the power, so to have a number between -1 and 1, then the obtained number is elevated to an odd power. It is important to note that if the power is even, this obtained number loses the information about whether the power is being absorbed or given by the WEC. This power elevation helps distinguishing more the powers with an higher magnitude, while distinguishing less the powers with a lower one.

From now on, when talking about power, it is actually meant the normalized power (which has the same information of the non-normalized one)



**Figure 4.10:** Effect of the odd elevating power in the range  $[-1,1]$  of the domain

### Reward definition

The actual power in a state is not a suitable candidate to be the reward function, mainly due to the fact that it depends only on the newly reached state and not on the previous state and neither on the action taken in the previous state.

It is important to remember that the reward is related to the performing of an action in a particular state.

A more suitable definition of the reward for this learning process is the difference between the power absorbed in the previous state, and the power that it is absorbing in the newly reached state (reached performing the selected action in the previous state).

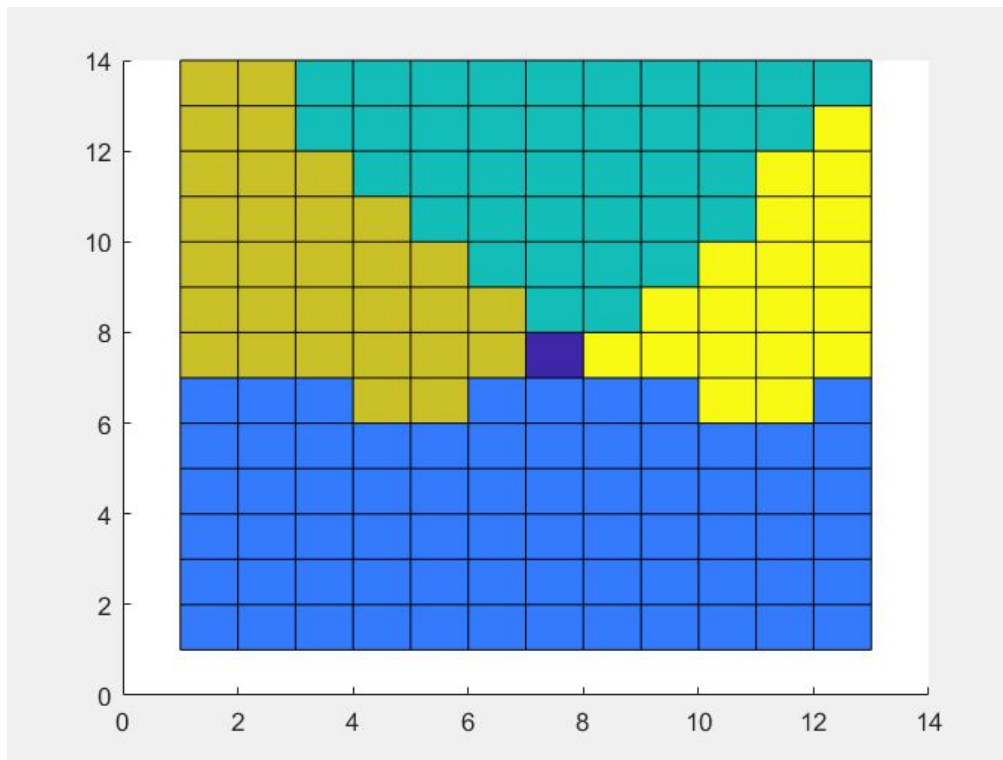
## Q Initialization

In order to both greatly speed up the convergence time and assure that the convergence is to a correct policy, a preliminary initialization of the Q matrix helps a lot.

Before the actual learning phase, where the algorithm will actually select the actions and perform them, a first exploration to all possible states is forced and the power there is saved.

After doing that it is possible to initialize the Q matrix with 3 nested 'for' cycle (one for each dimension of the Q matrix, which is a  $14 \times 13 \times 5$  matrix,  $\dim(S1) * \dim(S2) * \dim(A)$ ). Actually not all the values of the Q matrix are filled: since constraints-violating actions are never performed, their value in the Q matrix remains forever 0. This would give problems if in a particular state all the allowed action gave a negative reward, but in this case, due to the concavity of the power function, there is no state where no action gives a positive reward.

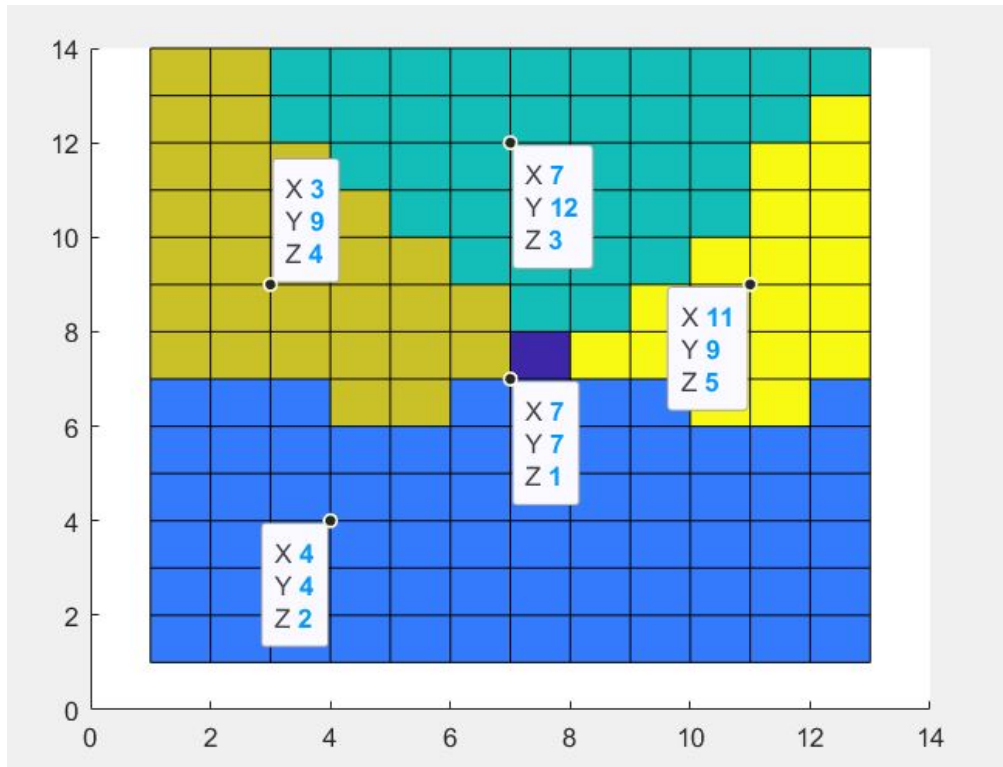
Interestingly, just the initialization alone is enough to find the actual ideal policy and one may think that this is enough. The problem is that even if the identified policy is the correct one, the parameter of the WEC may change in time and consequently the power function, so the actual correct policy is dynamical in real application. For these nonidealities it is not enough to just check before all possible states and see from there which policy is the best, but an actual periodical training is needed to redirect the algorithm towards the new correct policy.



**Figure 4.11:** Policy obtained just from the preliminary initialization

In figure 4.8 the coloured squares represent:

1. Dark blue colour, 'Do nothing' action.
2. Blue colour, 'Increase control damping' action (move up)
3. Turquoise colour, 'Decrease control damping' action (move down)
4. Brown colour, 'Increase control stiffness' action (move right)
5. Yellow colour, 'Decrease control stiffness' action (move left)



**Figure 4.12:** Actual values and representing action of the colours

### Setting of the experience

The algorithm starts by defining the state space and the action space, then it loads the 'power.mat' file and initializes with it the Q matrix, after having normalized the powers.

Then it defines the learning parameters and the learning experience can start.

The learning starts with a while cycle and gets into it if the previous policy is different from the newly obtained one. If they are different a new episode starts.

An episode is defined as 100 experiences, where an experience consist in:

1. Choose the action  $a$  to perform with the respect to the actual state, following the  $\epsilon$ -greedy policy.
2. Perform that action, move the the next state in accordance to the action and see the reward obtained.
3. Update the Q matrix and the various N matrices (the ones containing the number of visits to each state, and the number of times a particular action has been performed in each state)



At the end of each episode the states are changed to random ones, and the N matrices (needed for the calculation of the actual  $\epsilon$  and  $\alpha$ ) are reset to 0.

### Convergence criteria

The policy derived from the Q-matrix is considered as the convergence criteria: after the first 10 episodes, at each end of an episode, the "at the moment" policy is stored and compared with the policy stored at the end of the previous episode. If this 2 policies are exactly the same (which means that, after an episode, the optimal actions in all the states remained the same) the algorithm stops.

### Learning Parameters

Various combinations of the learning parameters  $\epsilon_0$   $\alpha_0$   $N_\epsilon$   $N_\alpha$  have been tested. Considering the already correctly initialized Q matrix, almost all combinations converge to a policy equivalent to the correct one, noting that there are multiple policies equally correct (if in the upper right corner the algorithm decides to first down and then left, it is equivalent to going first left and then down).

What can help in discriminating good parameters setting from bad parameters setting is actually the total time needed for the convergence of the policy.

Even if in this case nothing is being simulated, and so there is no indicator of the time convergence, the average number of visit per state has been considered as a replacement for the convergence time. After having seen the results, 2 particular settings have been particularly interesting:

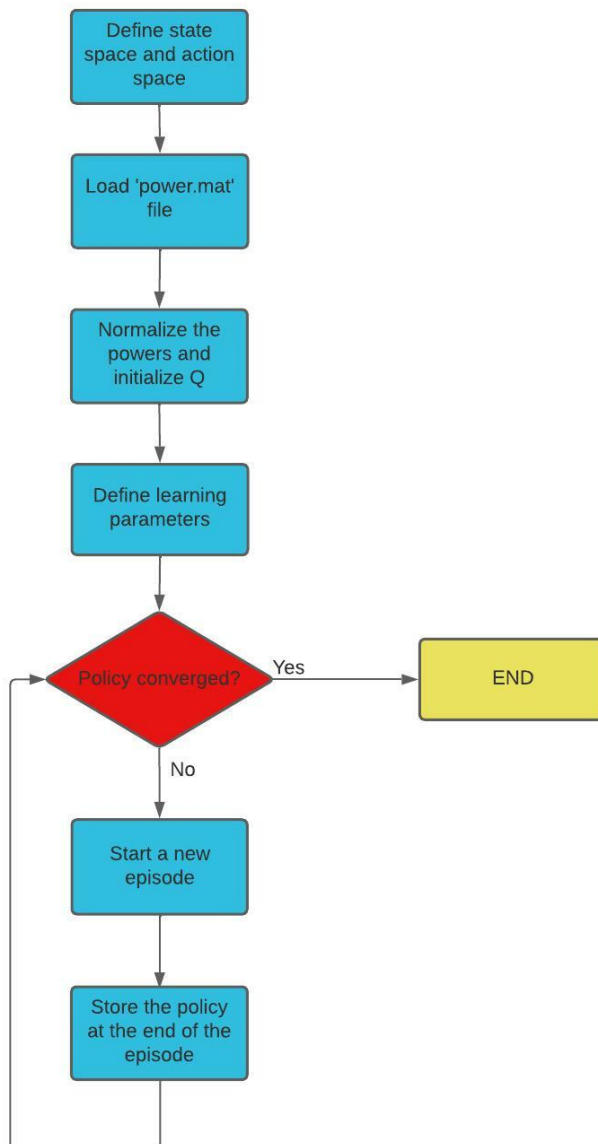
$\epsilon_0$	0.7
$\alpha_0$	0.99
$N_\epsilon$	5
$N_\alpha$	1

**Table 4.1:** First parameters setting of interest

$\epsilon_0$	0.5
$\alpha_0$	0.9
$N_\epsilon$	3
$N_\alpha$	30

**Table 4.2:** Second parameters setting of interest

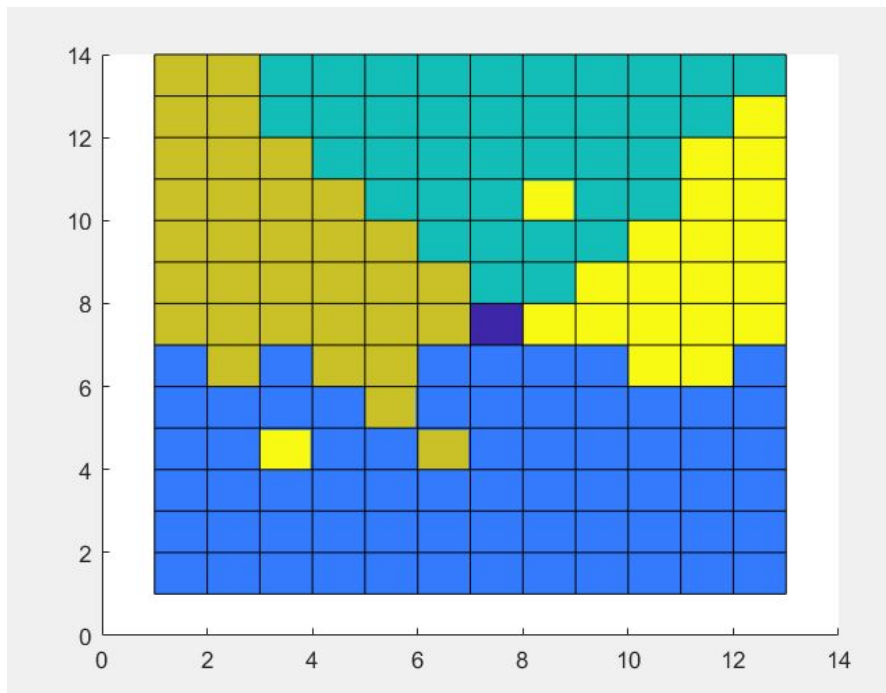
The parameter  $\gamma$  has been set to 0.9 in all cases. Considering future rewards are in practice equivalent to present rewards in an energy production scenario, lowering  $\gamma$  is highly unrecommended.



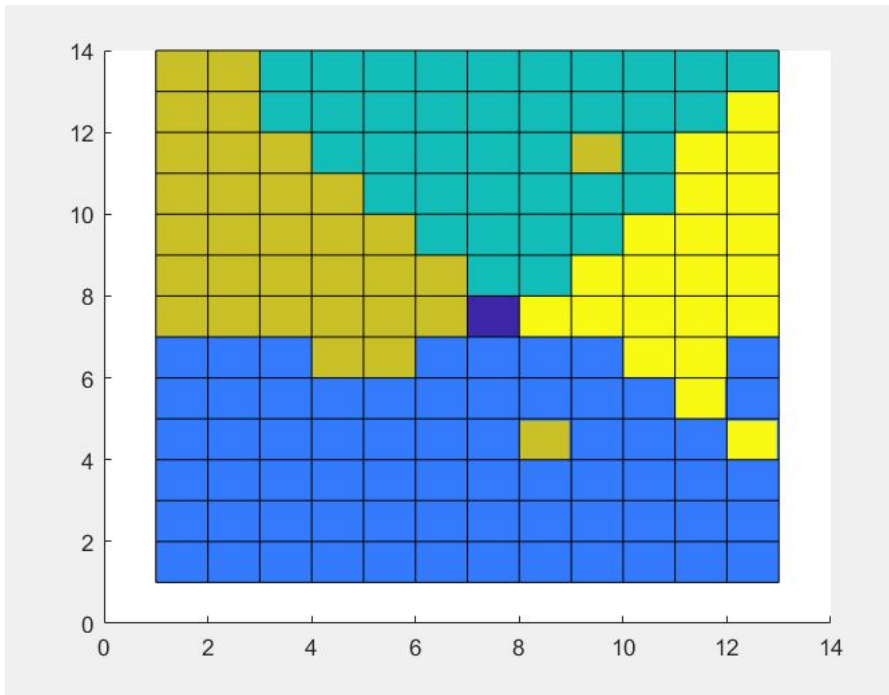
**Figure 4.13:** Flow chart of the offline Q-learning algorithm

## Results

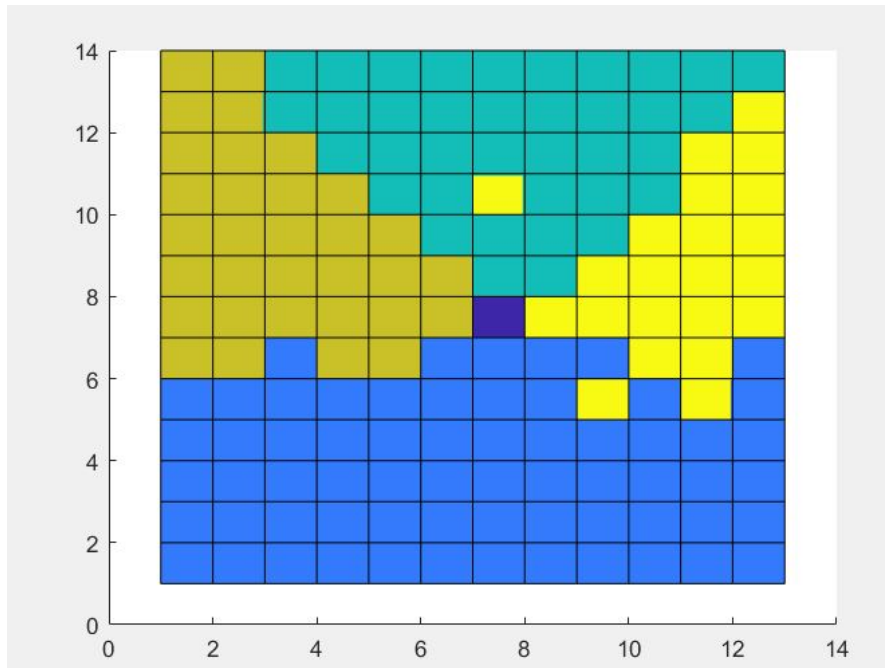
Considering the learning parameters of Table 4.1, the algorithm with that setting has been run multiple times, obtaining results a little bit different each time but satisfactory anyway. 3 of the final policies obtained along with the average number of visits per state effectuated are presented below:



**Figure 4.14:** First policy obtained, with average number of visits per state of 6.59



**Figure 4.15:** Second policy obtained, with average number of visits per state of 7.14



**Figure 4.16:** Third policy obtained, with average number of visits per state of 7.69

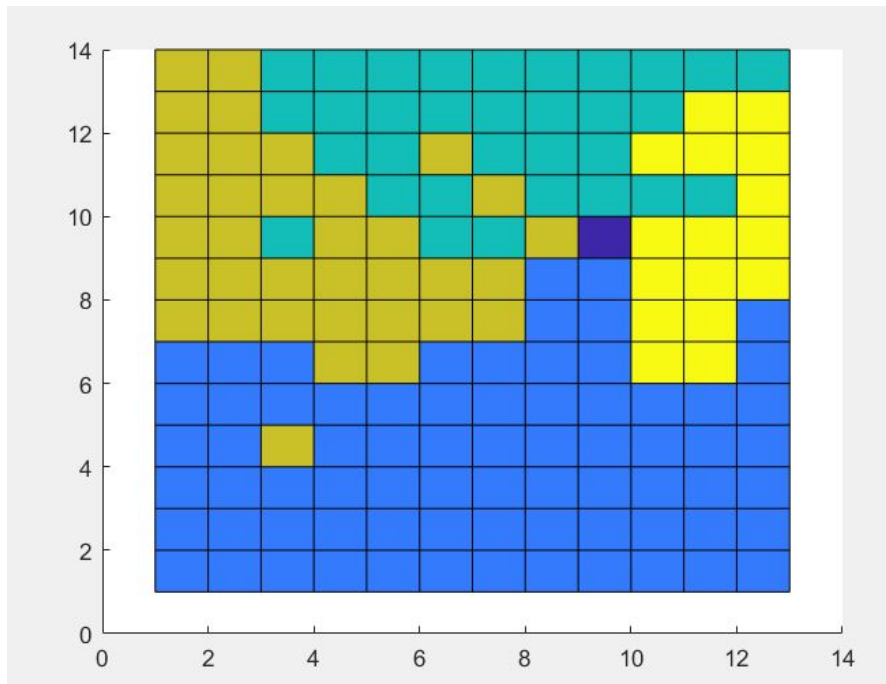
As it can be seen from the figures, the WEC ends up in the desired position (7,7) and there are no other steady points or loops, so the algorithm converges to policies optimal at steady state: following those policies, the WEC at the end reaches the position with maximum absorbed power (7,7) for all possible starting positions, but in moving to the optimum it sometimes follows a non optimal trajectory (for example in figure 4.14, in the point in (3,5) it moves to the left, while the desired destination is to the right, thus delaying the reaching of the optimal position). This is actually not a problem, since the WEC is designed for operating for prolonged times, so delaying of 1 action time is not a big deal. In the end, the algorithm is able to learn the sea state in a reasonable amount of time.

### Adaptability check

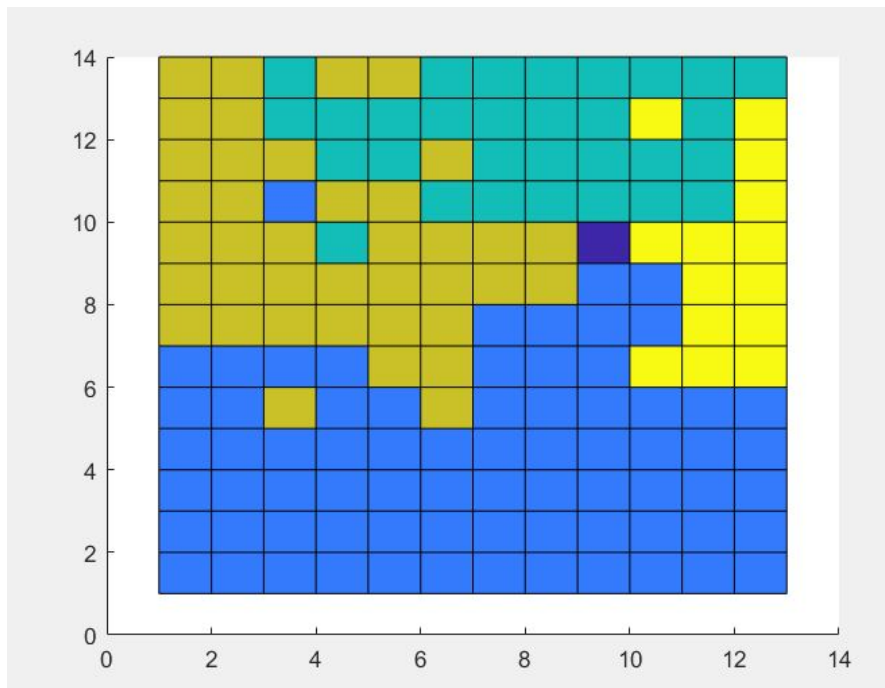
In order to verify that the algorithm makes the WEC control actually adaptable, the code has been run with the introduction of a "disturb": after 10 episode a change in the 'powermatrix' is directly introduced.

In the first case, the point in position (9,9), pretty far from the optimal one (7,7), sees its power increase suddenly (reaching a normalized power of 2, higher than the optimal one of 1) after only 7 episodes. Even if this is not a realistic scenario,

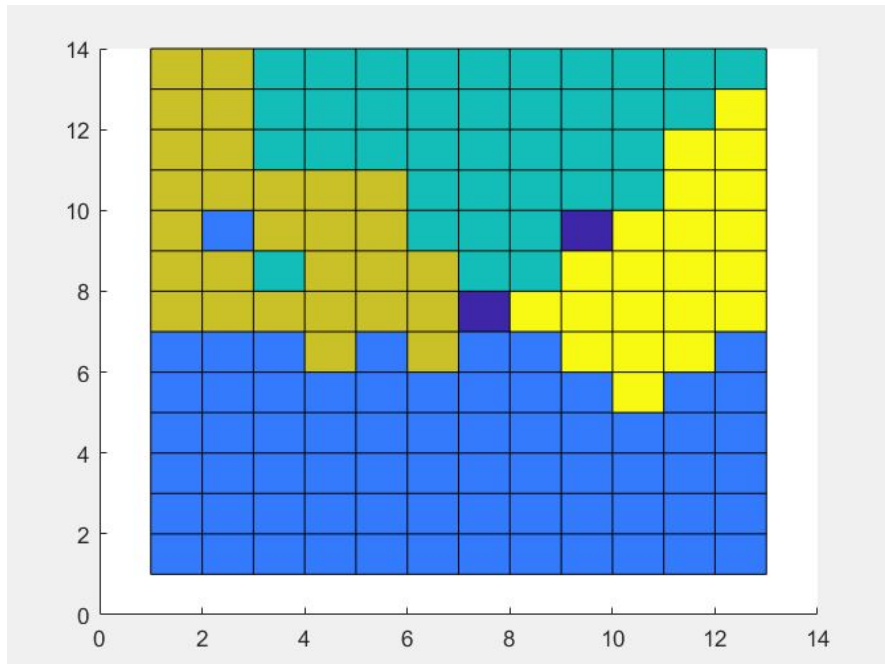
it can be helpful to use it to check how much awareness of the other points the algorithm has even after convergence.



**Figure 4.17:** First policy obtained, with parameters in Table 4.1, with 8.59 number of average visits.



**Figure 4.18:** Second policy obtained, with parameters in Table 4.1, with 11.53 number of average visits.



**Figure 4.19:** Third policy obtained, with parameters in Table 4.1, with 6.59 number of average visits.

As it can be seen from the figures, the algorithm acknowledges the change, and it adapts to it, changing the policy to converge to the new optimal point in figures 4.17 and 4.18, but increasing the time for convergence, as can be seen from the increased average number of visits.

The third run actually failed due to the stochasticity of this process: it converged too early (the number of average visits actually decreased), acknowledges the change in the point, but did not completely overwrite the previous policy and so it ended with 2 different convergence points.

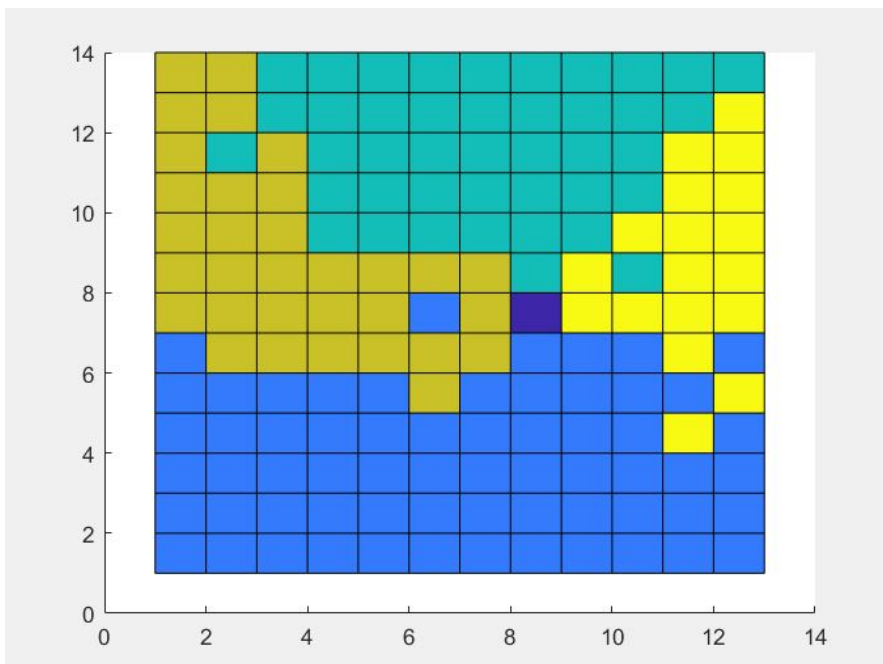
A change in the learning parameters (forcing more randomic exploration before convergence) can help reducing this problem, and make the algorithm find the right changed optimal in a more robust way, but this may increase the learning time of the algorithm.

Successively, a second type of change is analyzed: where the current optimal point (7,7) sees his power suddenly lowered to zero, after 10 episodes. This is a more realistic case, since it may happen that in time the WEC starts violating some constraints in the optimal position (es: change in the system due to fatigue), and this can be translated in a penalty that brings the total reward to zero or even lower.

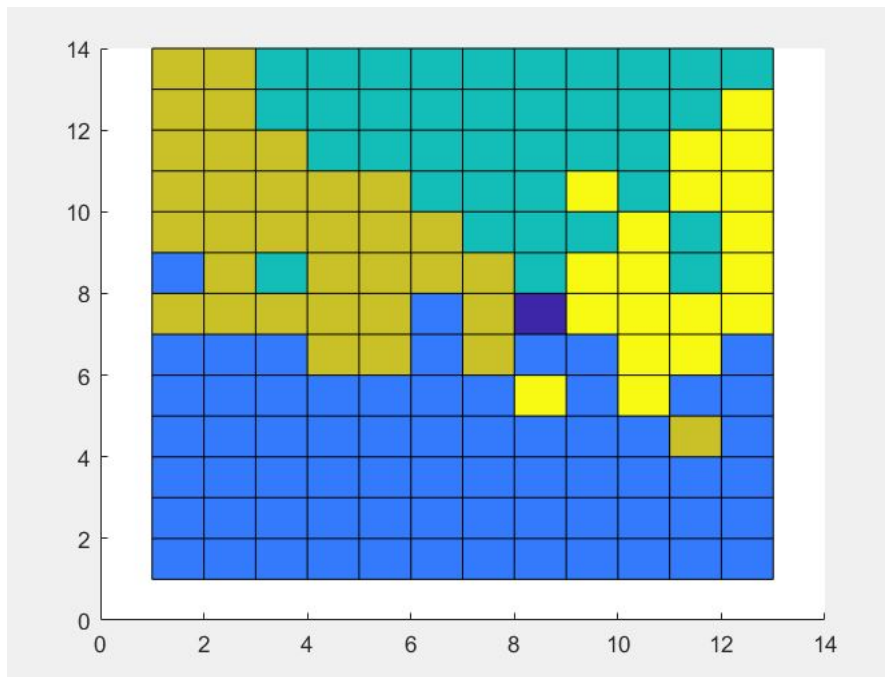
The new optimal becomes the point at the right of the previous optimal (8,7) and



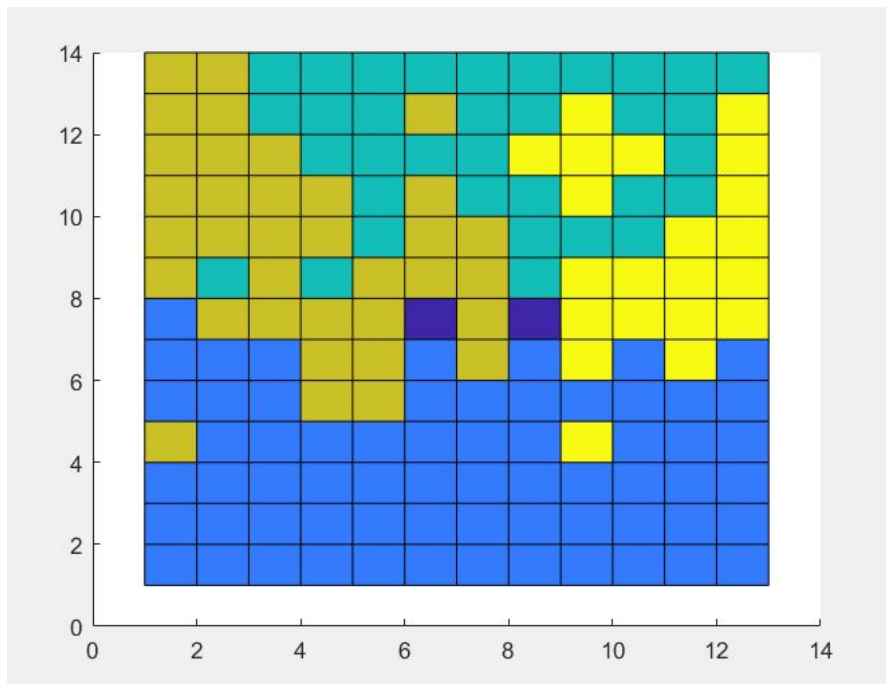
in theory the algorithm should converge to it now.



**Figure 4.20:** First policy obtained, with parameters in Table 4.1, with 13.73 number of average visits.



**Figure 4.21:** Second policy obtained, with parameters in Table 4.1, with 12.08 number of average visits.



**Figure 4.22:** Third policy obtained, with parameters in Table 4.1, with 10.98 number of average visits.

Again the first 2 policies (the ones with higher number of average visits per state) get correctly modified and converge only the new optimal point. The third policy (the one with lower number of average visits per state) instead, again due to the stochasticity, converges to both the new optimal and the other point next to the old optimal (6,7), which is a sub-optimal point. If this convergence to a sub-optimal point is a problem, a possible solution to make the algorithm more robust with respect this kind of change is still again searching for learning parameters more suitable for a changing power matrix (more exploration), or just to re-initialize the Q matrix after such a change has been detected. Both the options are easily implementable, but they will almost surely increase the learning time.

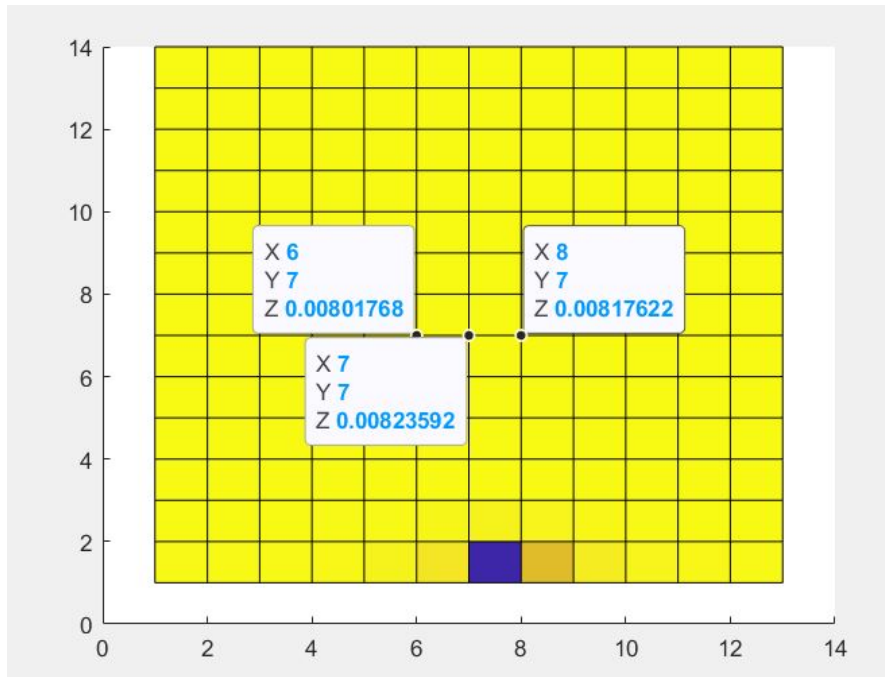


Figure 4.23: Values of the power in the old optimal and in the 2 points nearby.

### 4.1.2 Off-line Sarsa

Following the same procedure of the Q-learning algorithm, a Sarsa algorithm has been developed.

The only difference in the algorithm structure, from the Q-Learning one, is in the part of code regarding the policy evaluation: Q-Learning chooses as future action in the new state the one that maximizes the "at the moment" Q-matrix in that state, while Sarsa chooses this action with an epsilon policy.

Due to the determinicity of the WSI here considered, this ulterior stochasticity introduced in the evaluation will actually slow down the algorithm.

#### Learning Parameters

The different learning strategy requires different learning parameters to work better. For this algorithm, again 2 sets of parameters of before will be considered, the one used before, in table 4.2, which will actually be unsuccessful, and a new one, described in table 4.3

The main peculiarity of the this third set is that it has a low  $\epsilon$  but an high  $N_\alpha$  (and consequently high  $N_\epsilon$ ).

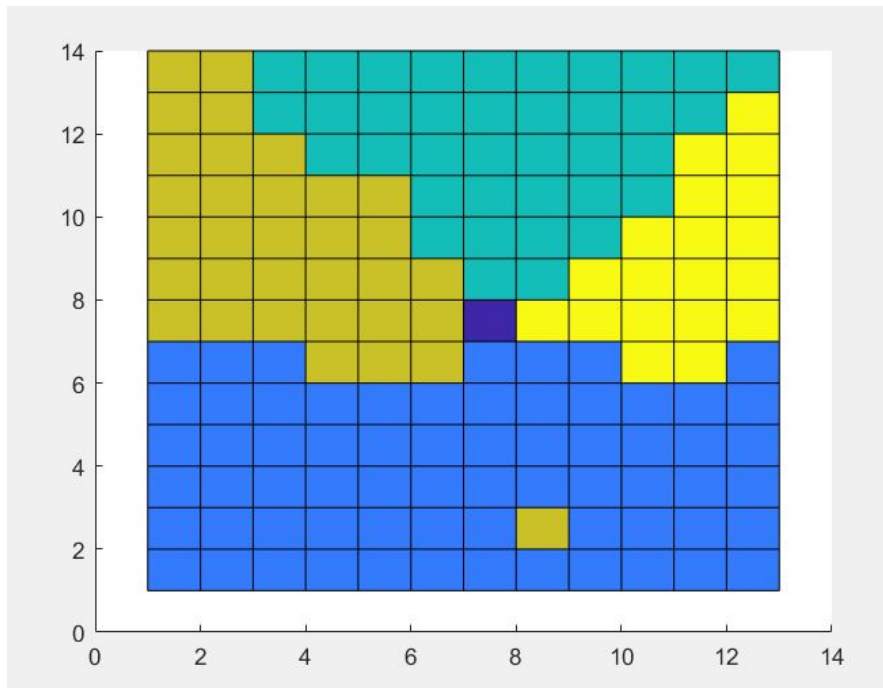
This change has been introduced in order to be able to lower  $\epsilon$  since it has been seen that an high  $\epsilon$  makes the convergence pretty hard and slows down a lot the

$\epsilon_0$	0.2
$\alpha_0$	0.9
$N_\epsilon$	50
$N_\alpha$	10

**Table 4.3:** Third parameters setting of interest

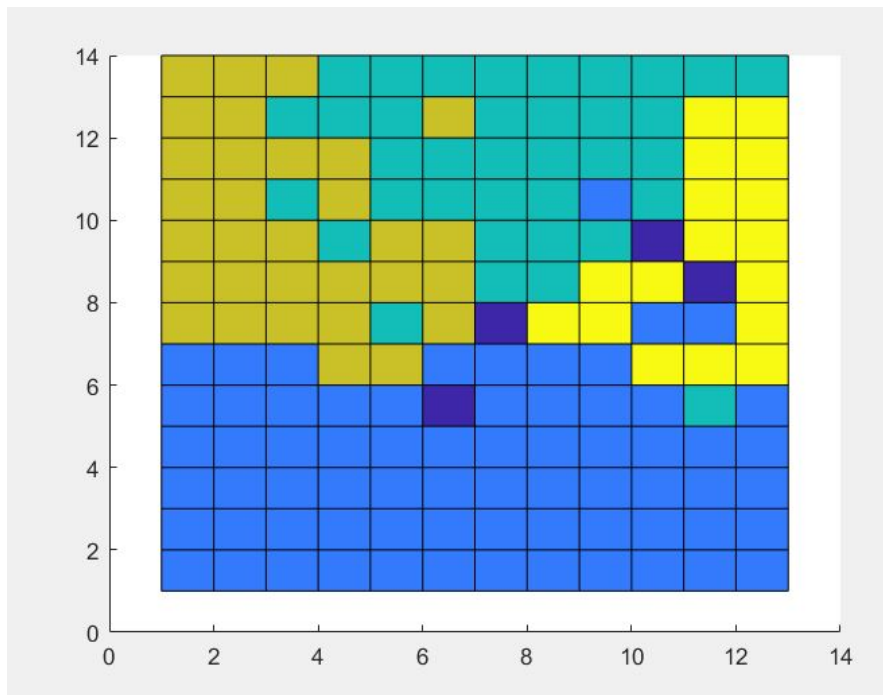
entire algorithm due to the high stochasticity in the evaluation stage.

## Results



**Figure 4.24:** Policy obtained with the parameters in table 4.3, with average number of visits of 17.58

Sarsa with the parameters in table 4.3 is able to converge to the correct policy, but it takes more than double the time of that used by Q-learning (even more time than Q-learning with changing power matrix).



**Figure 4.25:** Policy obtained with the parameters in table 4.2, with average number of visits of 15.38

Using the set of parameters in table 4.2, the algorithm converges to the wrong policy, converging to more points and not only to the optimal one, while still requiring more time than a correct Q-learning.

The results obtained have shown that due to the fact that the interaction between the WEC and the sea here is considered totally deterministic, Q-learning performs better than Sarsa, due to the higher simplicity in the policy evaluation stage.

Sarsa is still able to converge to the correct policy, but to do so it needs way more time than Q-learning and "ad hoc" parameters.

For this reasons, moving to a on-line control algorithm in the next section, only Q-learning algorithms have been considered there, since the overall computation effort of the algorithm and so the time needed for convergence will be an important aspect there .

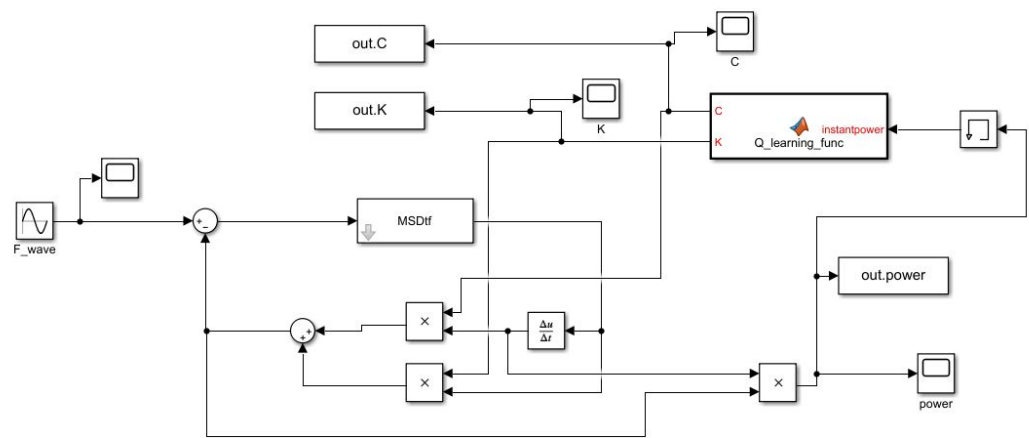
## 4.2 MSD model: On-line version of Q-learning

After having tested the feasibility of a Q-learning algorithm for the WEC control off-line, an on-line version has been developed, which is way more similar to the

real application.

This time, the power will not be calculated with various simulations before the learning, but all the learning will happen in just one very long simulation, where the algorithm will explore the state space and converge to a policy. To do so, an initial MATLAB code will define the environment and simulation parameters, then start it will start the Simulink model.

The Q-learning algorithm is scripted in the Simulink thanks to a MATLAB Function block, a Simulink block that can be defined as a MATLAB function with its own workspace and parameters.



**Figure 4.26:** Simulink scheme used to simulate the system

A total simulation time of 100 000s has been used, overall time the algorithm has at disposal to learn the policy.

A simulation fixed time-step of 0.05s has been used, giving a good compromise of numerical accuracy and overall speed of the simulation.

The learning experience has been structured in a analogous way to the off-line algorithm: an first initialization forces to explore 1 time all possible C-K combinations, and after that a free exploration divided in episodes is made.

Due to the online nature of this algorithm, the power has to be calculated in between the simulation: the algorithm starts with an initial control C and control K, then it waits in this state and keeps simulating it for 30 seconds (600 time steps, 6 forcing wave periods) and store all the 600 instantaneous power values and,

after this 30 seconds, it computes the average power, decides which action to take, updates the states and repeats until the end of the simulation.

The number of episodes will also be a changing learning parameter, like  $\epsilon_0$ ,  $\alpha_0$ , and  $N_\epsilon$ , but  $N_\alpha$  will no more be a parameter and will be decided by  $N_\epsilon$  with the relation  $N_\epsilon = 5N_\alpha$ . An episode will last for 6000 seconds (200 power calculations) and at the start of a new episode the states will be randomized and the matrices for the computation of the actual  $\epsilon$  and  $\alpha$  will be resetted.

The MATLAB function receives at each time step the instant power as input, and gives as output the actual control C and control K for the next time step.



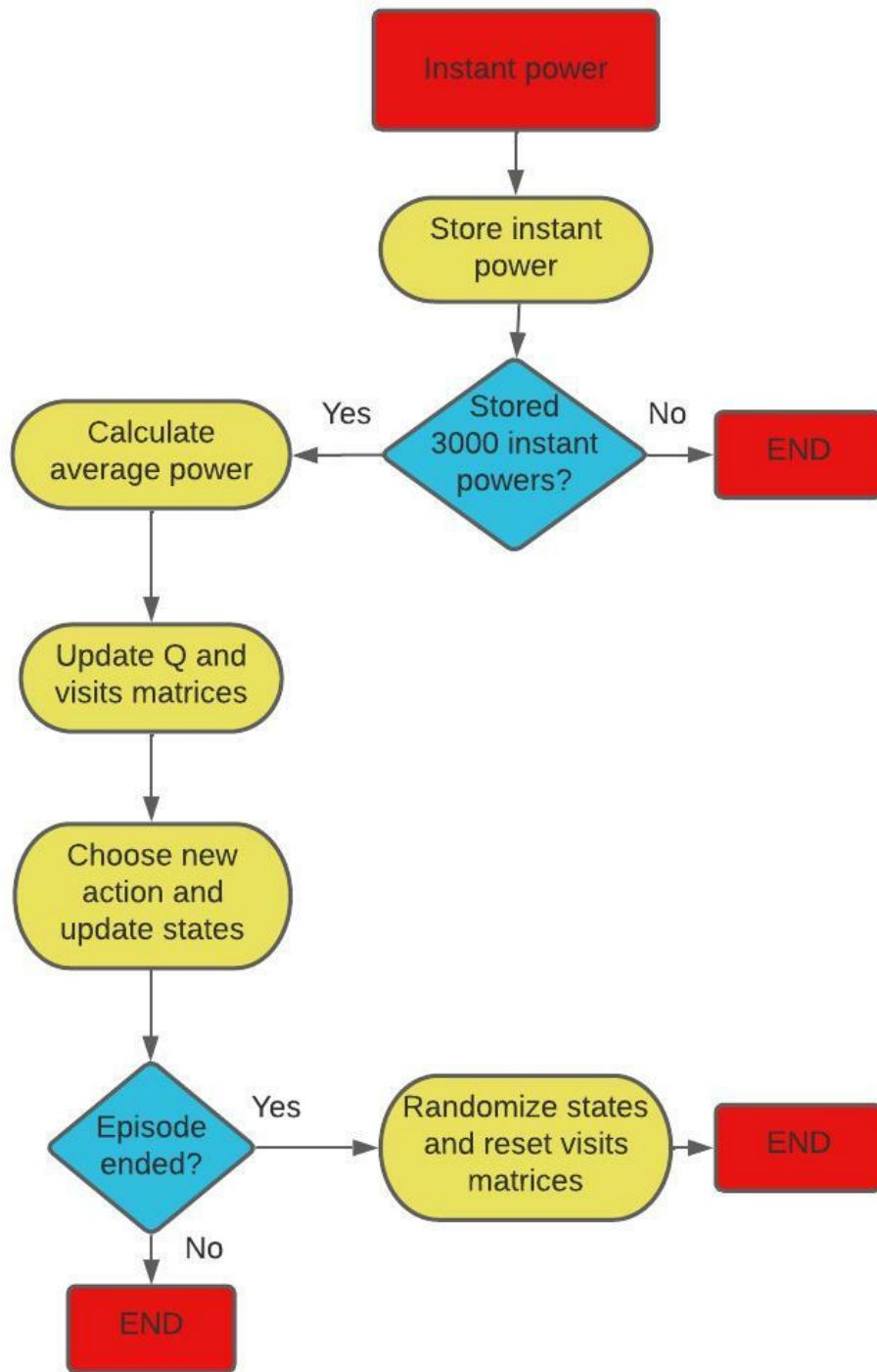


Figure 4.27: Workflow of the MATLAB function, which is run at each time step

### 4.2.1 Learning Parameters

Given the higher computational complexity of this online algorithm, the choice of suitable learning parameter is crucial to find the right settings, which enable both a right convergence and both a reasonable convergence time.

To find this parameters, multiple simulations have been run, each one testing a different combination and saving the results.

4 changing parameters have been used:

1.  $\epsilon = [0.1 - 0.3 - 0.5 - 0.7 - 0.9 - 0.95 - 0.99]$

2.  $\alpha = [0.1 - 0.3 - 0.5 - 0.7 - 0.9 - 0.95 - 0.99]$

3.  $N_\alpha = [1 - 2 - 3 - 5 - 7 - 9]$

4. number of episodes =  $[1 - 2 - 3 - 5 - 7 - 9]$

for a total of 1764 number of simulations, of which all the results have been stored in a .mat file containing a structure with all the data of each simulation (learning parameter settings, control C and control K history, power history).

### 4.2.2 Results

After having collected all the data from all the simulations, these data have been searched for the right combinations of parameters.

The algorithm considers "correct" a simulation where the control damping and the control stiffness in the last 10000 seconds, are in average close to the optimal ones, and where the average power in the last 10000 seconds is close to the average power between 50000-60000 seconds of the simulation.

In an other struct, only the correct combinations have been stored.

214 parameters combination pass this test, and they can be divided into 2 types:

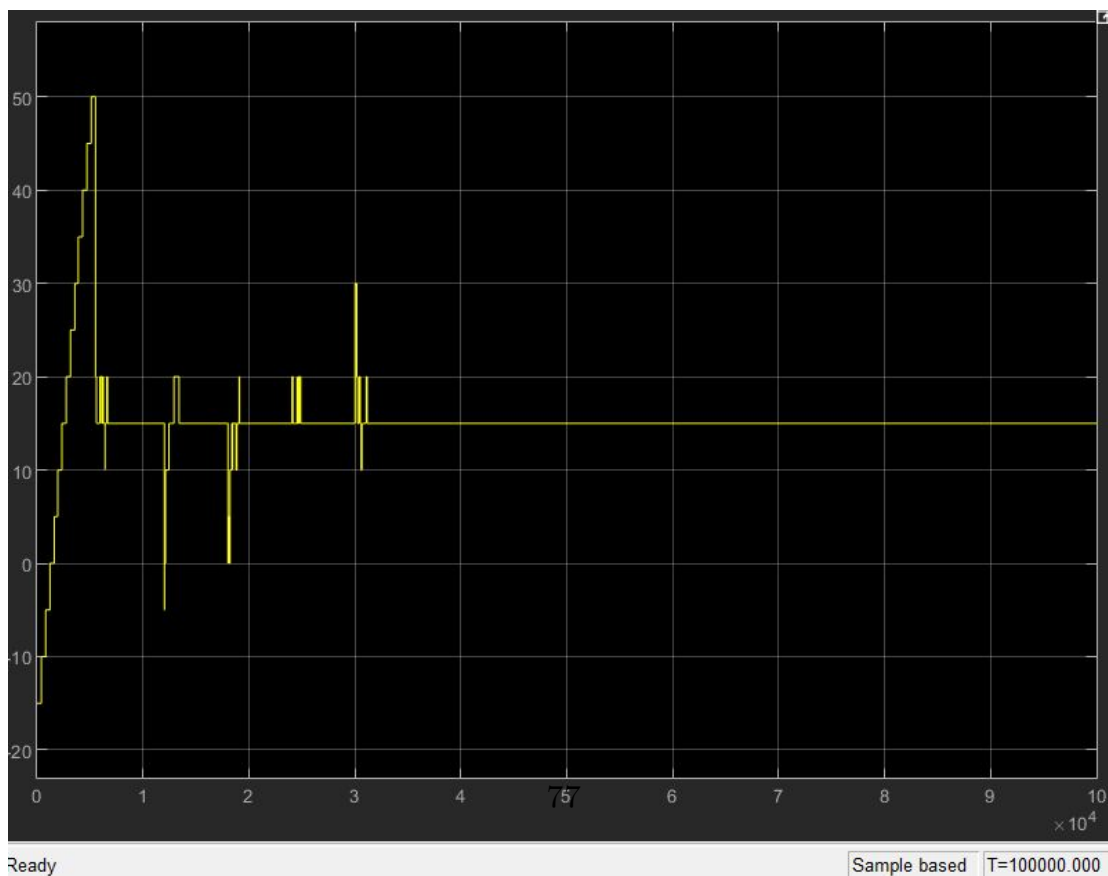
1. Learning settings that in practice only do initialization with little to no exploration.
2. Actual learning algorithm, where after the initialization there is serious exploration and then convergence.

$\epsilon_0$	0.1
$\alpha_0$	0.3
$N_\alpha$	3
number of episodes	5
convergence time	around 10000 seconds

**Table 4.4:** Example of a parameter settings which explores very little and converge soon after initialization

$\epsilon_0$	0.5
$\alpha_0$	0.95
$N_\alpha$	2
number of episodes	5
convergence time	around 30000 seconds

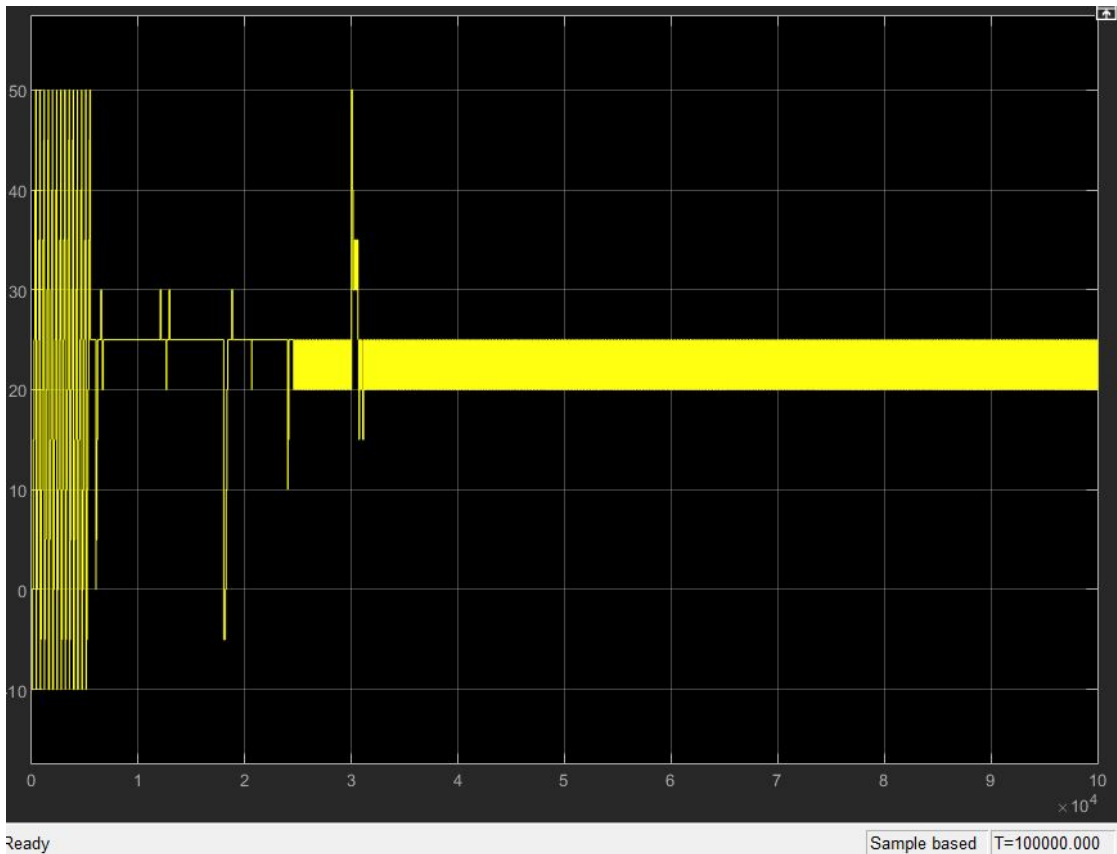
**Table 4.5:** Example of a parameter settings which actually explores before convergence



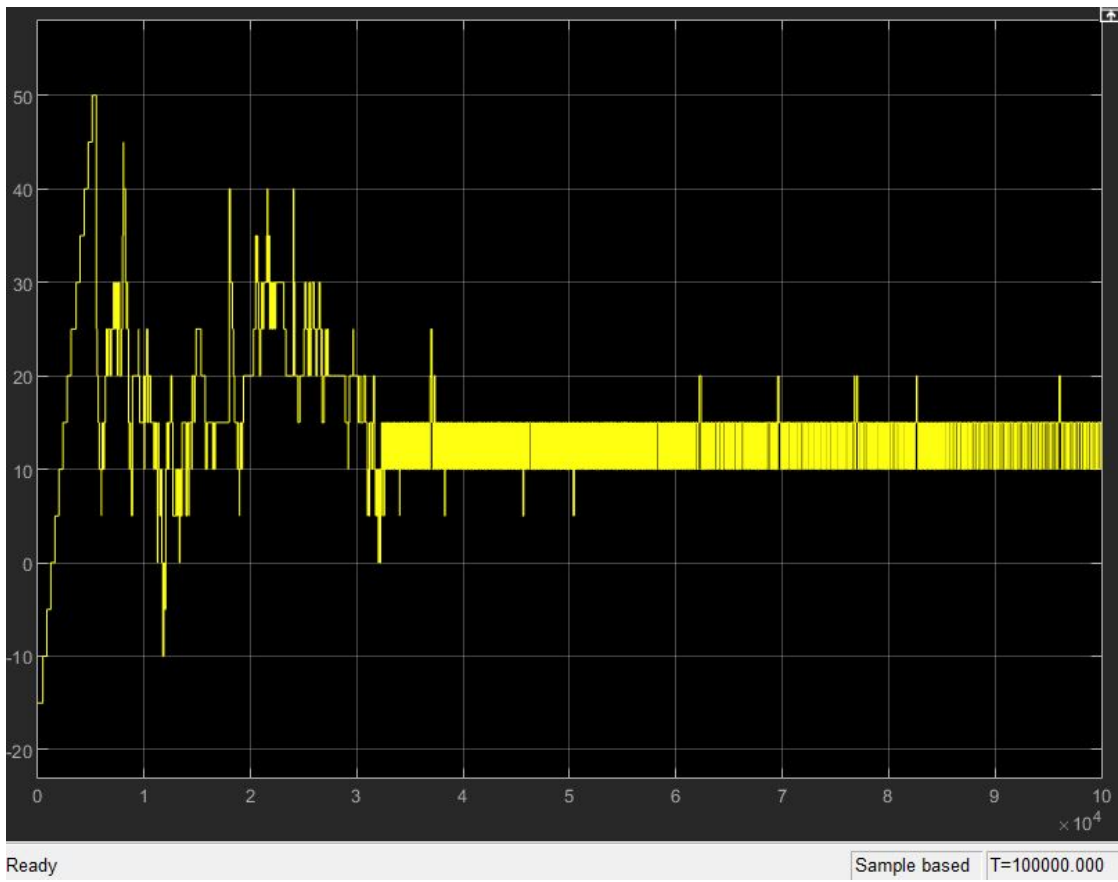
**Figure 4.28:** Control damping time evolution, with the parameters in table 4.4

$\epsilon_0$	0.7
$\alpha_0$	0.99
$N_\alpha$	1
number of episodes	1
convergence time	around 20000 seconds

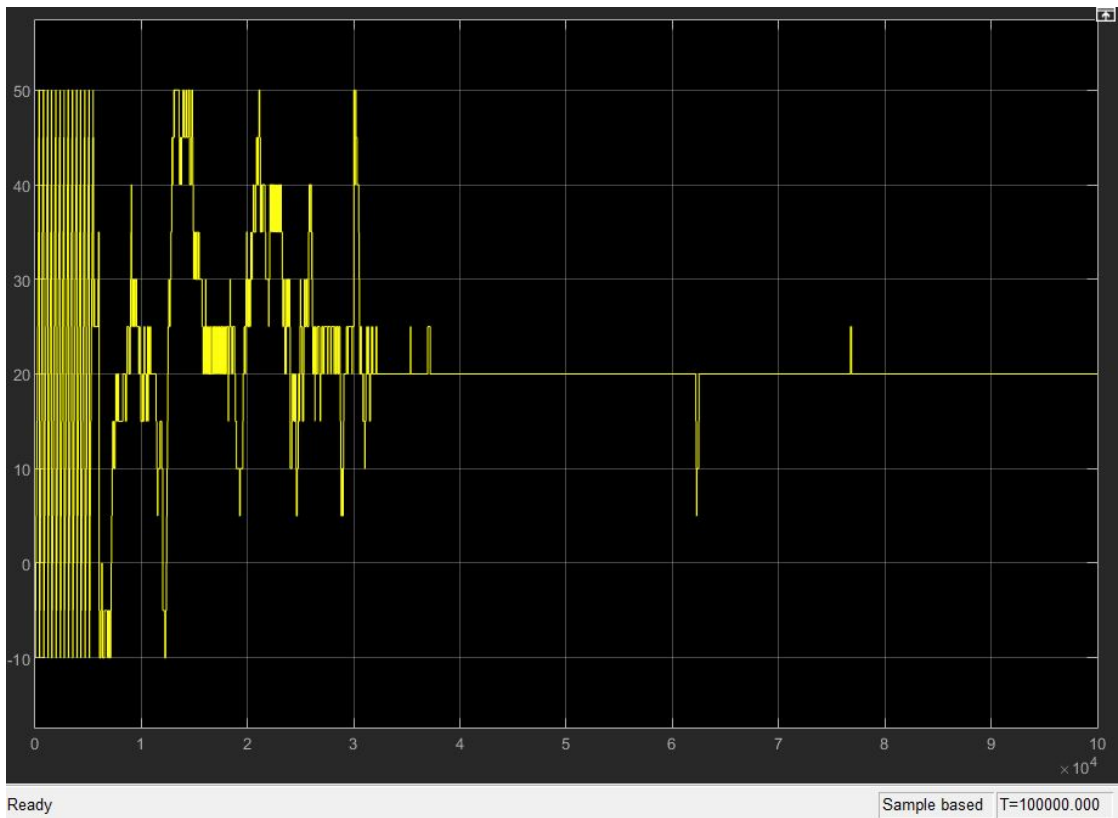
**Table 4.6:** Another example of a parameter settings which actually explores before convergence, even if explores for a little time, and just around the convergence point.



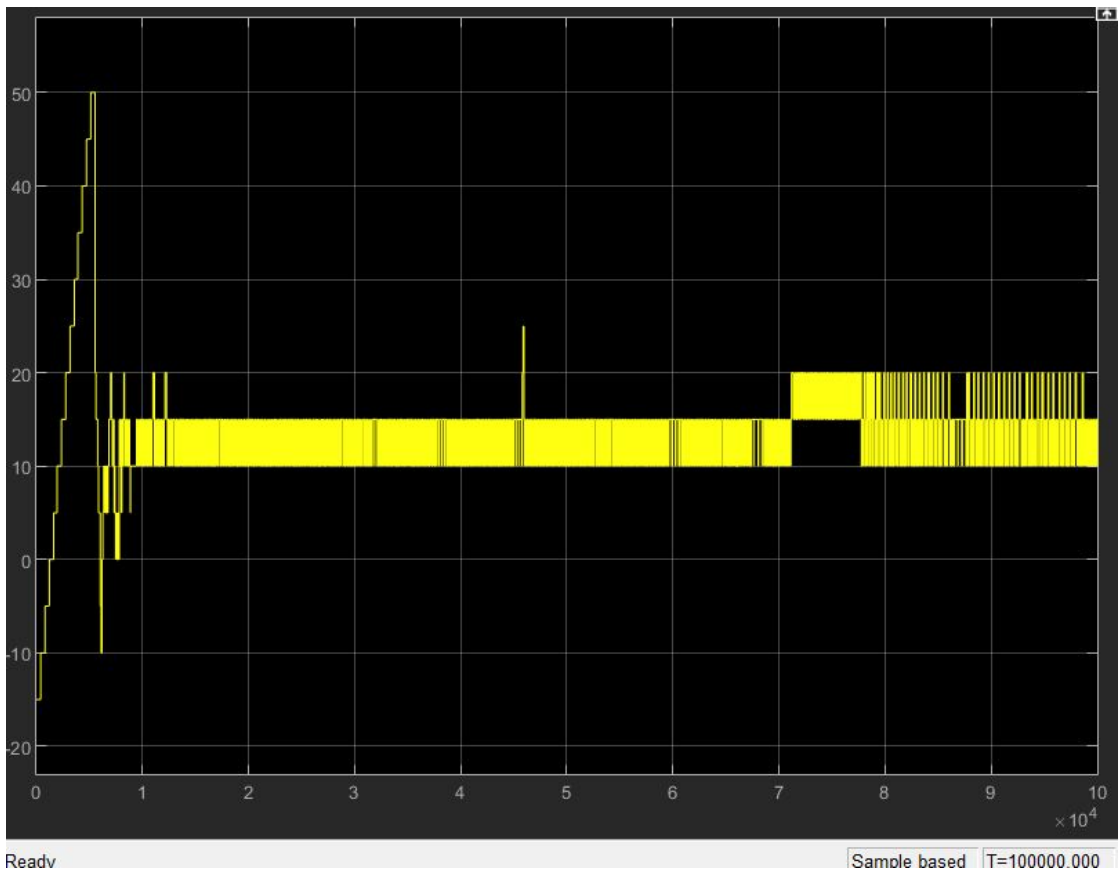
**Figure 4.29:** Control stiffness time evolution, with the parameters in table 4.4



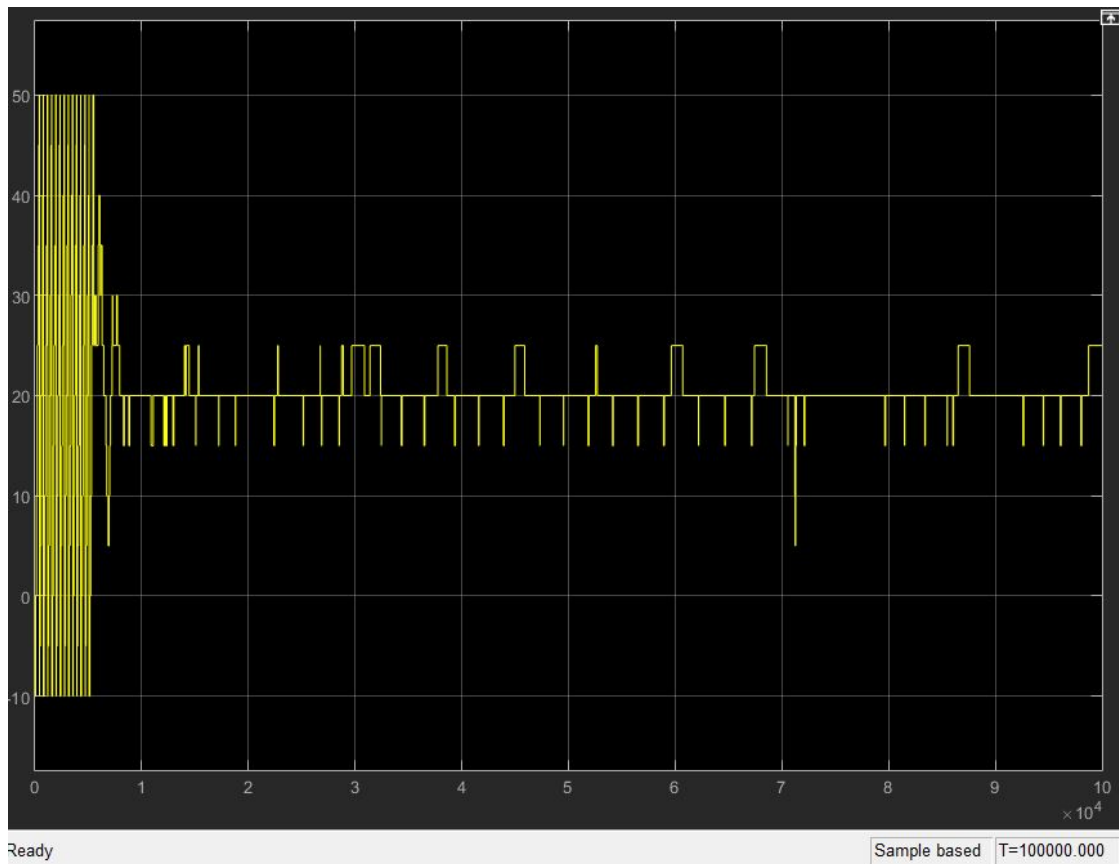
**Figure 4.30:** Control damping time evolution, with the parameters in table 4.5



**Figure 4.31:** Control stiffness time evolution, with the parameters in table 4.5



**Figure 4.32:** Control damping time evolution, with the parameters in table 4.6



**Figure 4.33:** Control stiffness time evolution, with the parameters in table 4.6

As it can be seen from the figures, it is possible to have convergence to the optimal control variables with multiple sets of learning parameters, differing from each other with respect to the convergence time and the "amount" of exploration (exploration-exploitation trade-off).

Anyway almost all learning parameters combination do not just converge to the optimum, but, even after 90 000 seconds of simulation, keep exploring the surrounding states, thus potentially being able to detect easily continuous variation of the optimal point.

### 4.3 PeWEC model

After having tested the algorithms on a simplified MSD model, the successive step has been that of passing to a real WEC model.

The Pendulum Wave Energy Converter (PeWEC) is a WEC designed for the



Mediterranean waves, characterized by a relatively long period [45].

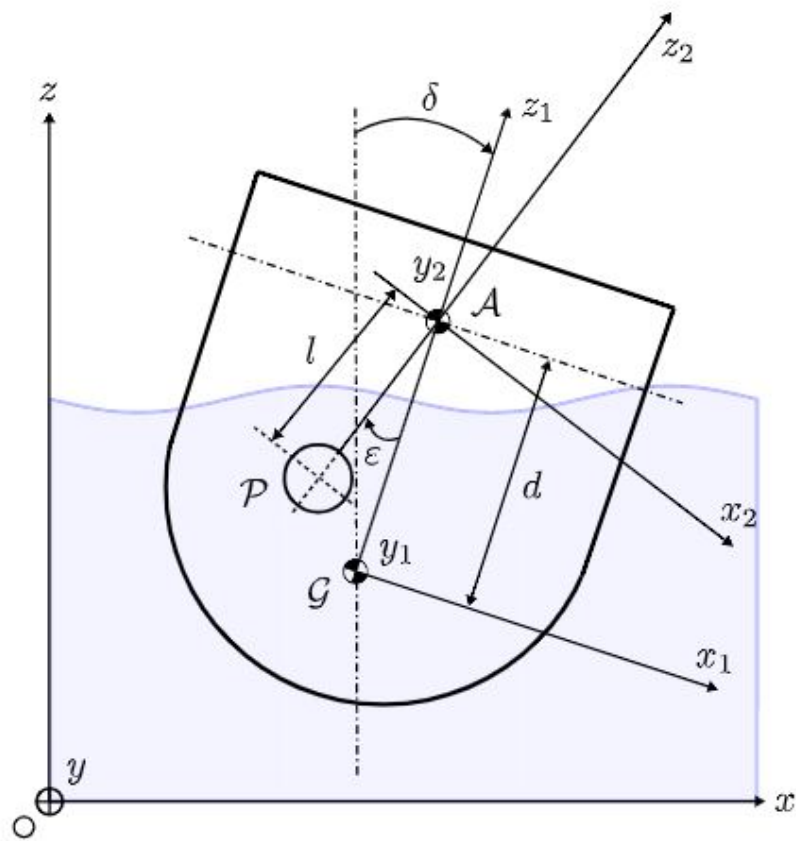
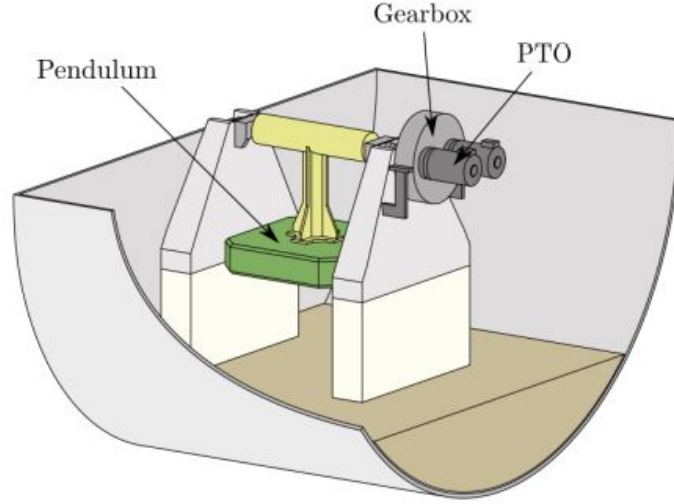


Figure 4.34: PeWEC Scheme. Taken from [45]



**Figure 4.35:** PeWEC graphical representation. Taken from [46]

In a totally linear situation, the equation describing the evolution of a PeWEC system is the Cummins' one [46]:

$$(M + A_\infty)\ddot{X}_f(t) + \int_0^t K_r(t - \tau)\dot{X}_f(\tau) d\tau + B_\nu|\dot{X}_f(t)|\dot{X}_f(t) + K_h X_f(t) = F_{ext}(t)$$

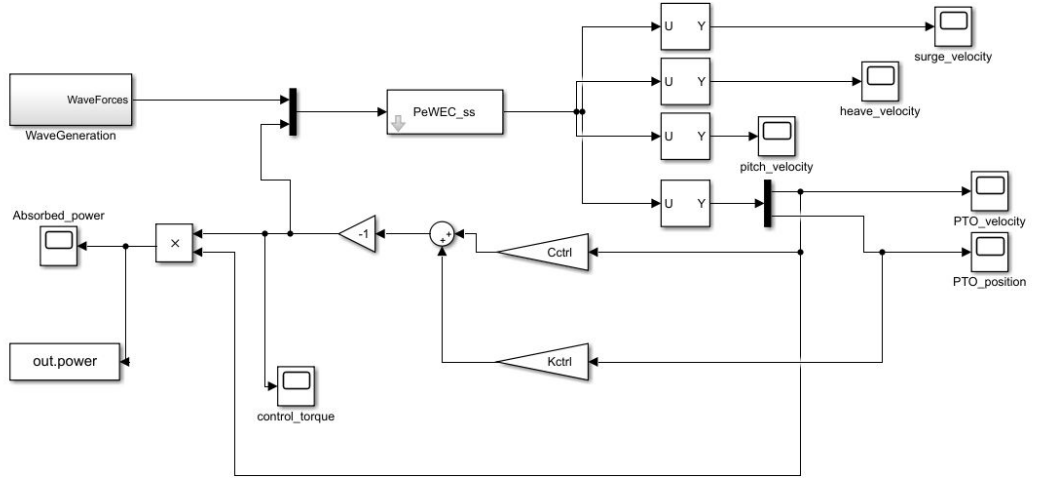
where  $M$  is the inertial mass of the device,  $A_\infty$  is the added mass at infinite frequency,  $K_r$  is the radiation impulse response functions matrix,  $K_h$  is the hydrostatic stiffness matrix and  $F_{ext}(t)$  are the generalized external forces.

The  $B_\nu$  coefficients, since are related to nonlinear viscous forces, have to be computed through CFD simulations.

Since the convolution integral has an high computational complexity, usually it is replaced with its approximation through a state space representation:

$$\int_0^t K_r(t - \tau)\dot{X}_f(\tau) d\tau = F_r(t) \approx \begin{cases} \dot{\zeta}_r(t) = A_r\zeta_r(t) + B_r\dot{X}_f(t) \\ F_r(t) = C_r\zeta_r(t) + D_r\dot{X}_f(t) \end{cases}$$

Due to this mathematical details of the model, for simulating the system with Simulink, the WEC system will no more be represented by an LTI TF, but with a state-space representation, where the matrices of the system are frequency-dependent, as they represent also the convolution term which is strongly frequency dependent.



**Figure 4.36:** Simulink scheme used to simulate the PeWEC.

### 4.3.1 Control Variables

The control variables as usual will be the control damping and the control stiffness of the PTO, even if this time they represent rotational mechanics ones (indeed the control action is a torque and not a linear force).

The range considered for the control damping has been  $[8000; 850000] Kg\ m^2/s$ , while the range for the control stiffness has been  $[-844770; 0] Kg\ m^2/s^2$ . Both the control variable have been linearly discretized such that they each have 13 elements.

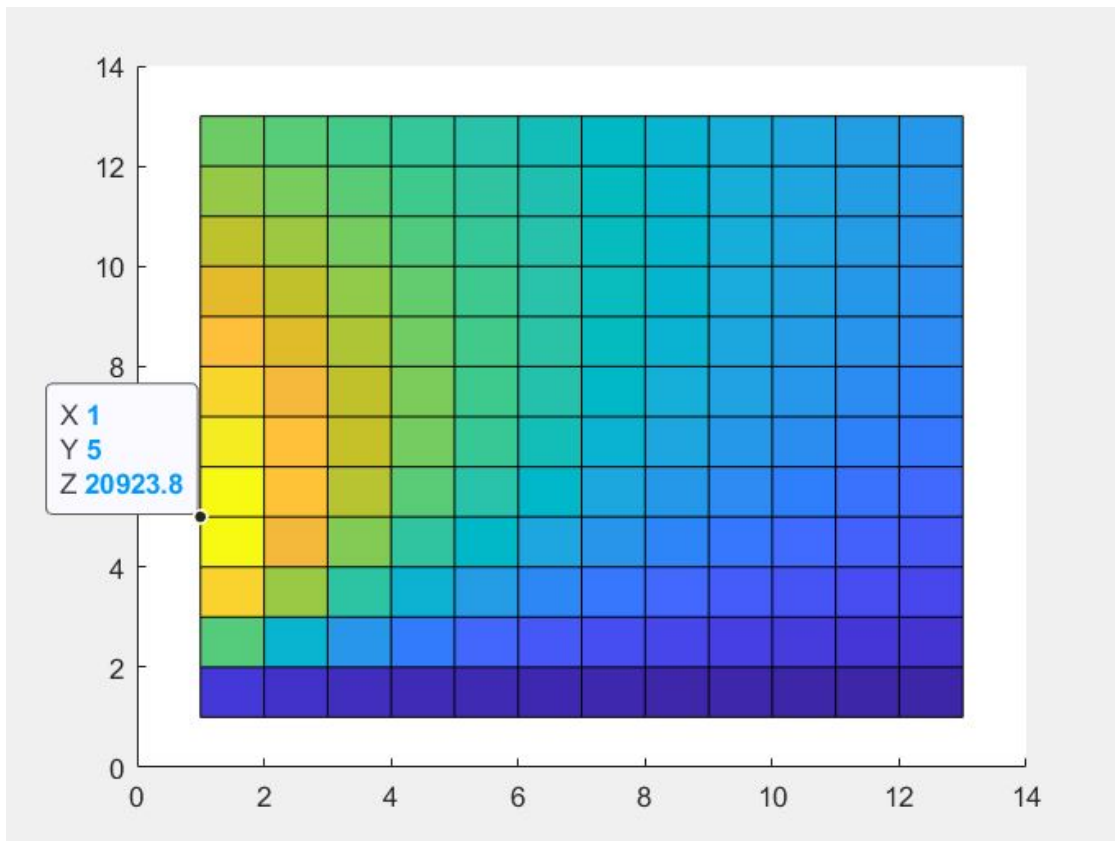
### 4.3.2 Sea states

In order to develop an algorithm able to be physically implemented on a device, more sea states will be considered and the ability to converge of the algorithm will be tested on all of them. 3 sea states will be considered:

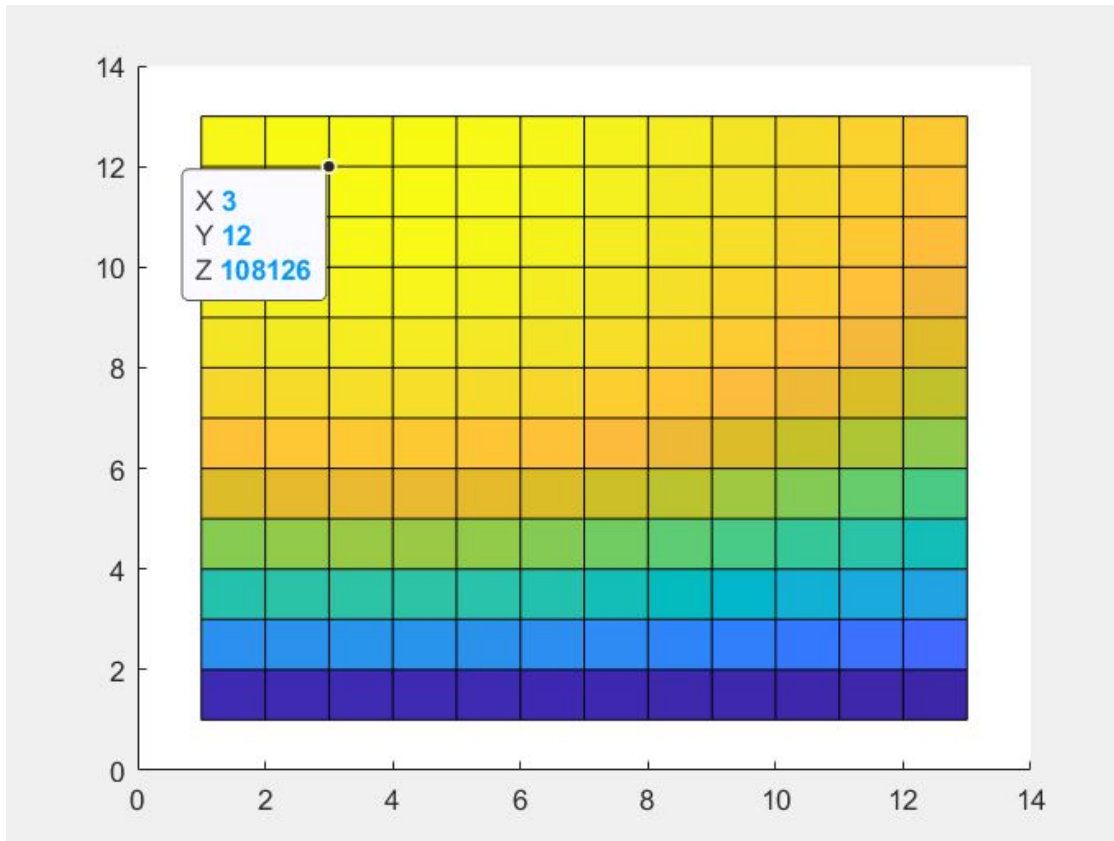
1. State 1:  $T_e = 5$  and  $H_s = 1$
2. State 2:  $T_e = 6$  and  $H_s = 1$
3. State 3:  $T_e = 7$  and  $H_s = 1$

A first simulation of all possible control states in all the sea states has been performed to assess the optimal control values in each sea state. For this simulation

a step time of 0.05 s and a simulation time of 1800 s (overall time for computing the power) have been selected, in order to have accurate preliminary values of the powers.



**Figure 4.37:** Powermatrix of the sea state 1. X axis represents control stiffness, Y axis represents control damping.



**Figure 4.38:** Powermatrix of the sea state 2.

For the sea state 2, it needs to be pointed out that there are a lot of points near the optimal one with almost the same power (the optimal stiffness can change in a range where there is almost no power variation).

This will make it hard for the algorithm to individuate the exact optimum, since it uses power-based rewards.

Anyway this should not be a problem, since the objective is that of absorbing power, those sub-optimal solutions are practically indistinguishable from the optimal one.

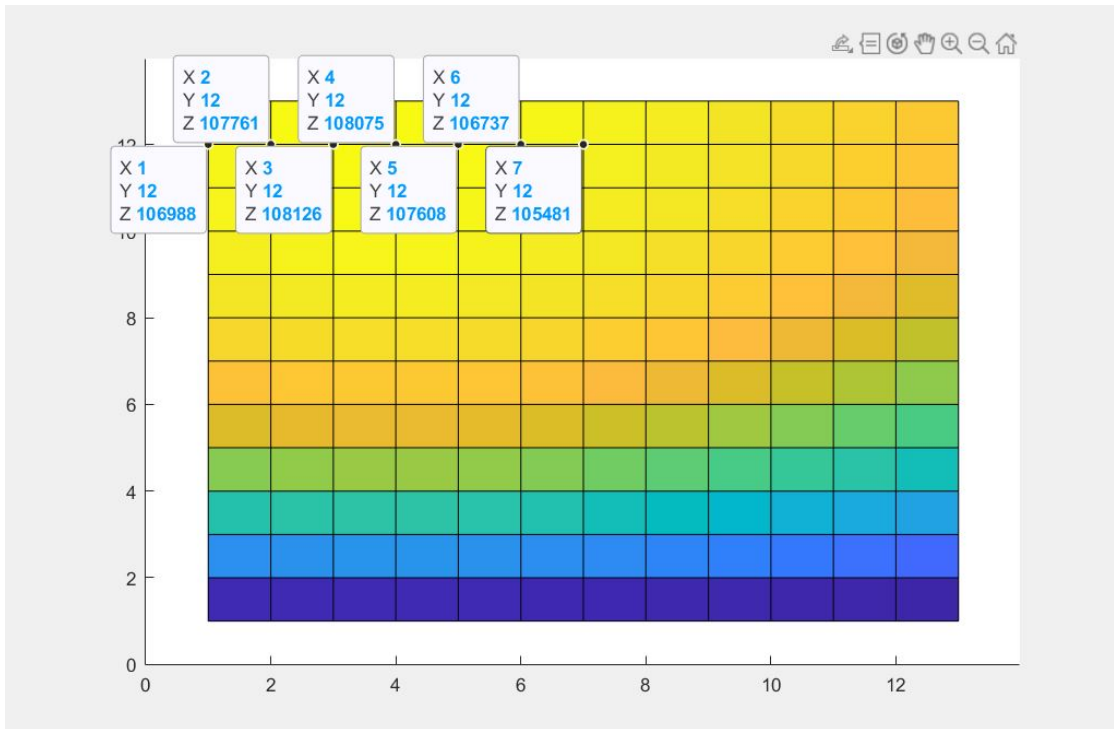
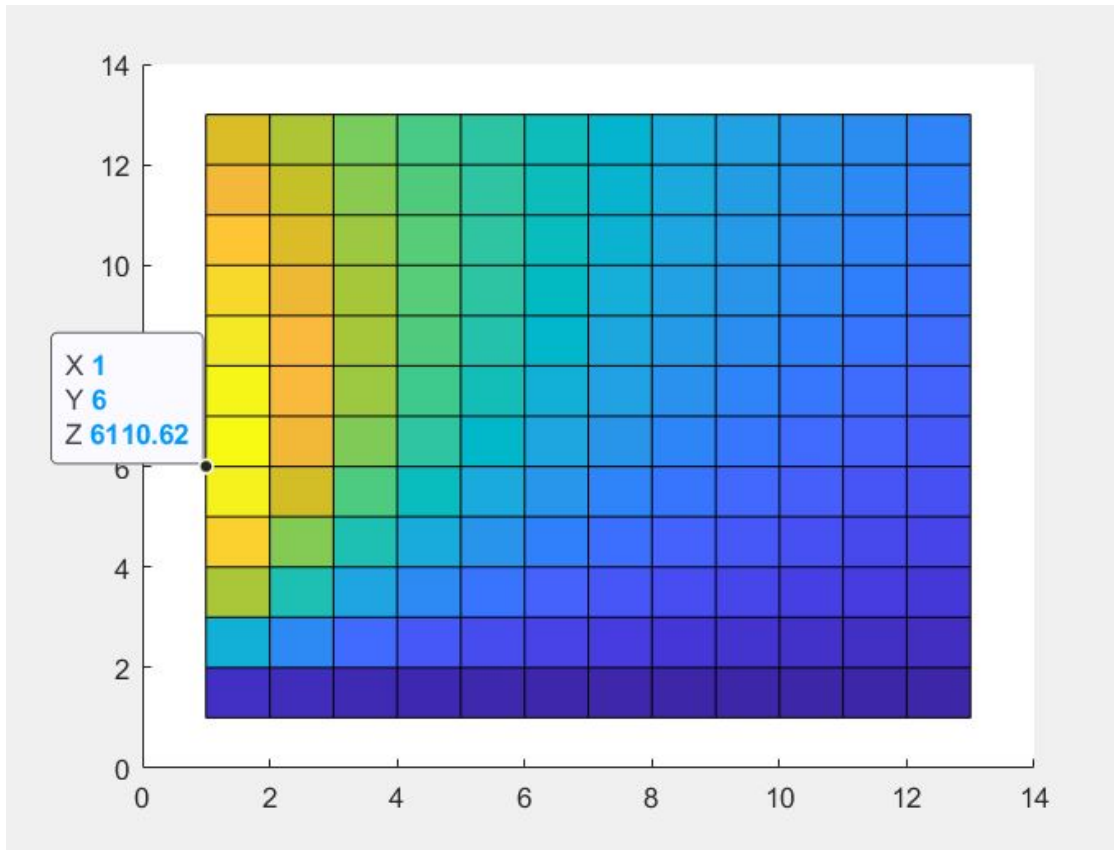


Figure 4.39: Detail of the powermatrix of the sea state 2.



**Figure 4.40:** Power matrix of the sea state 3.

Optimal control values and absorbed power:

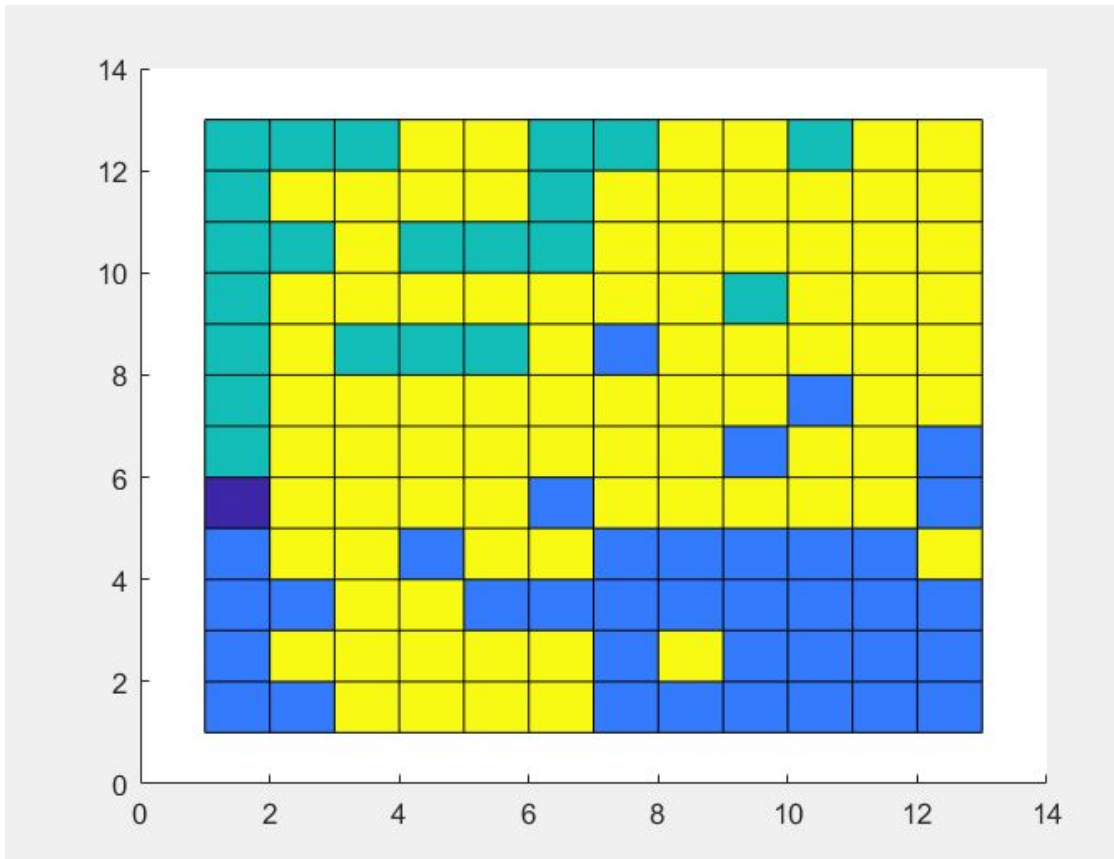
1. State 1:  $C_{ctrl}=288670 \text{ Kg m}^2/s$ ;  $K_{ctrl}=-844770 \text{ Kg m}^2/s^2$ ; Absorbed power= $20923 \text{ W}$
2. State 2:  $C_{ctrl}=779830 \text{ Kg m}^2/s$ ;  $K_{ctrl}=-703980 \text{ Kg m}^2/s^2$ ; Absorbed power= $108126 \text{ W}$
3. State 3:  $C_{ctrl}=358830 \text{ Kg m}^2/s$ ;  $K_{ctrl}=-844770 \text{ Kg m}^2/s^2$ ; Absorbed power= $6110 \text{ W}$

### 4.3.3 Offline Q-Learning

At first an offline version of a Q-Learning algorithm was created, Using the same algorithm used for the MSD, but with the new power matrix related to the PeWEC instead of the one related to the MSD model.

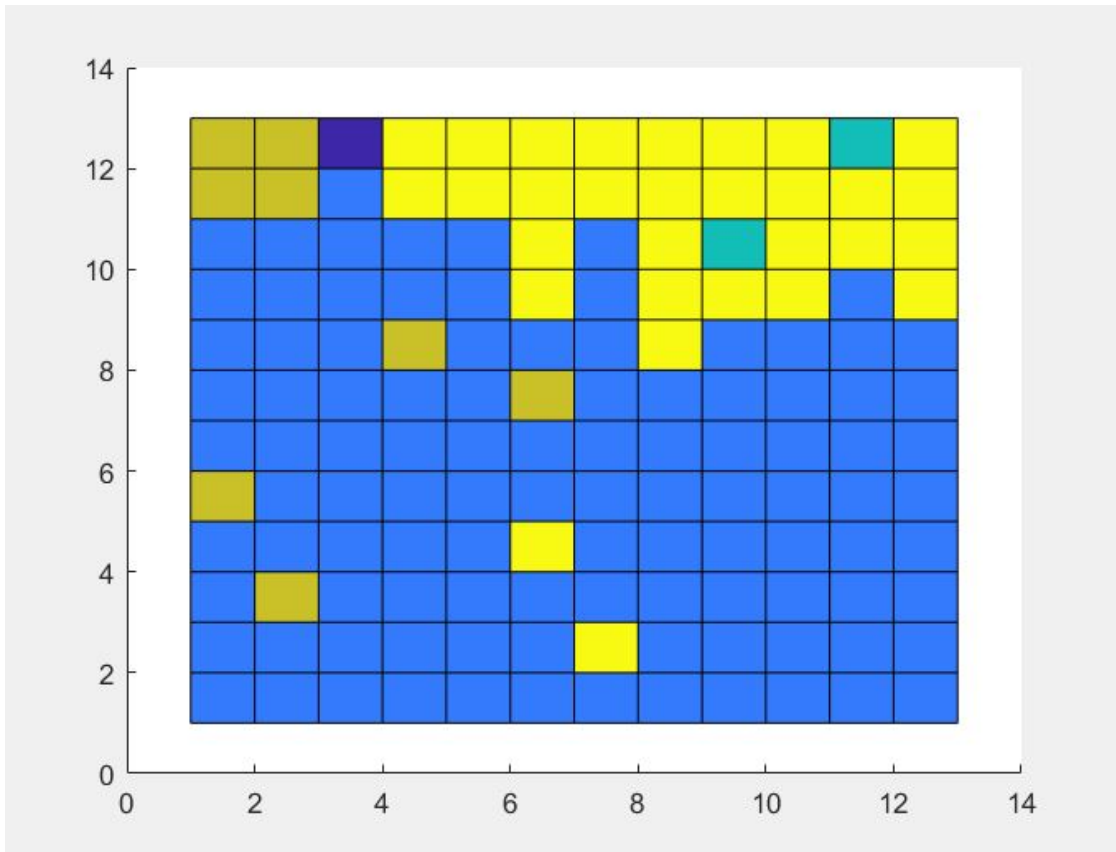
No differences showed up with respect to what obtained with the MSD, given the identical structure of power matrix input function (concave function).

Using as Learning parameters  $\epsilon_0 = 0.7$ ,  $\alpha_0 = 0.9$ ,  $N_\epsilon = 5$ ,  $N_\alpha = 1$  and a number of 100 actions per episode, the policies in the following figures have been obtained:

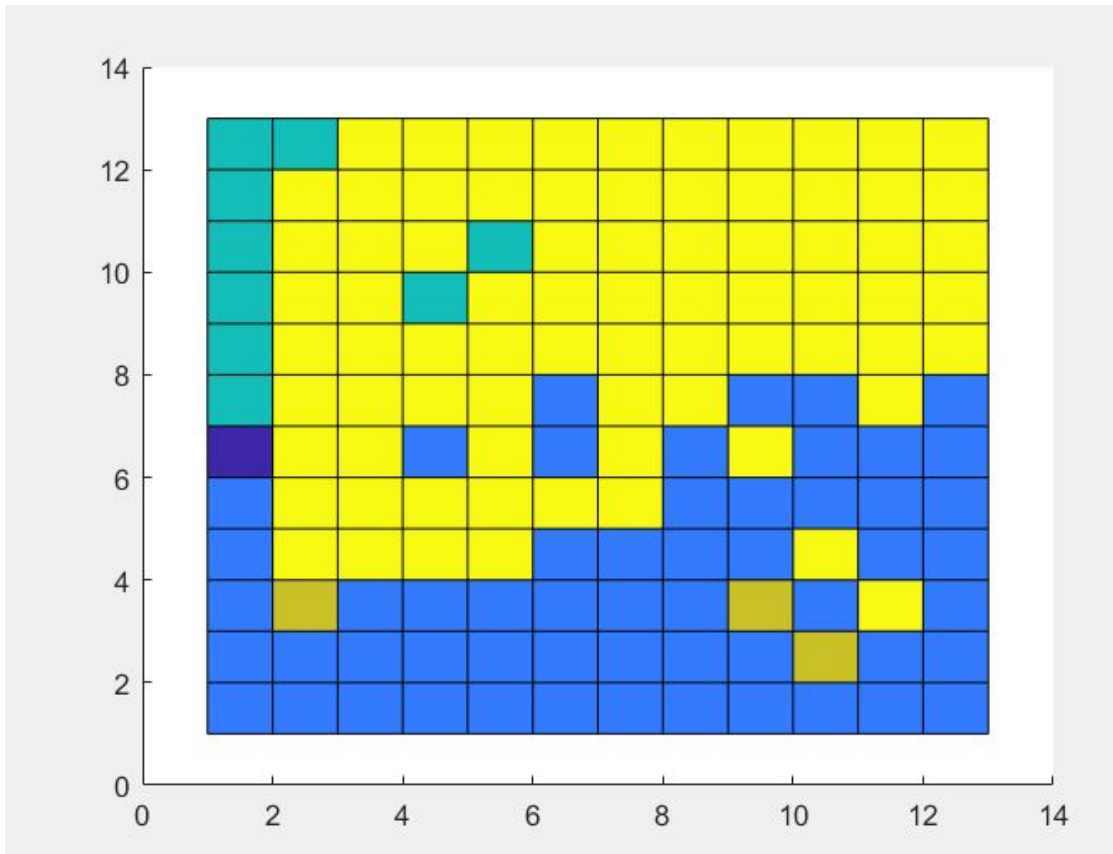


**Figure 4.41:** Policy obtained for the sea state 1, with an average number of visits per state of 11.24.





**Figure 4.42:** Policy obtained for the sea state 2, with an average number of visits per state of 7.10.



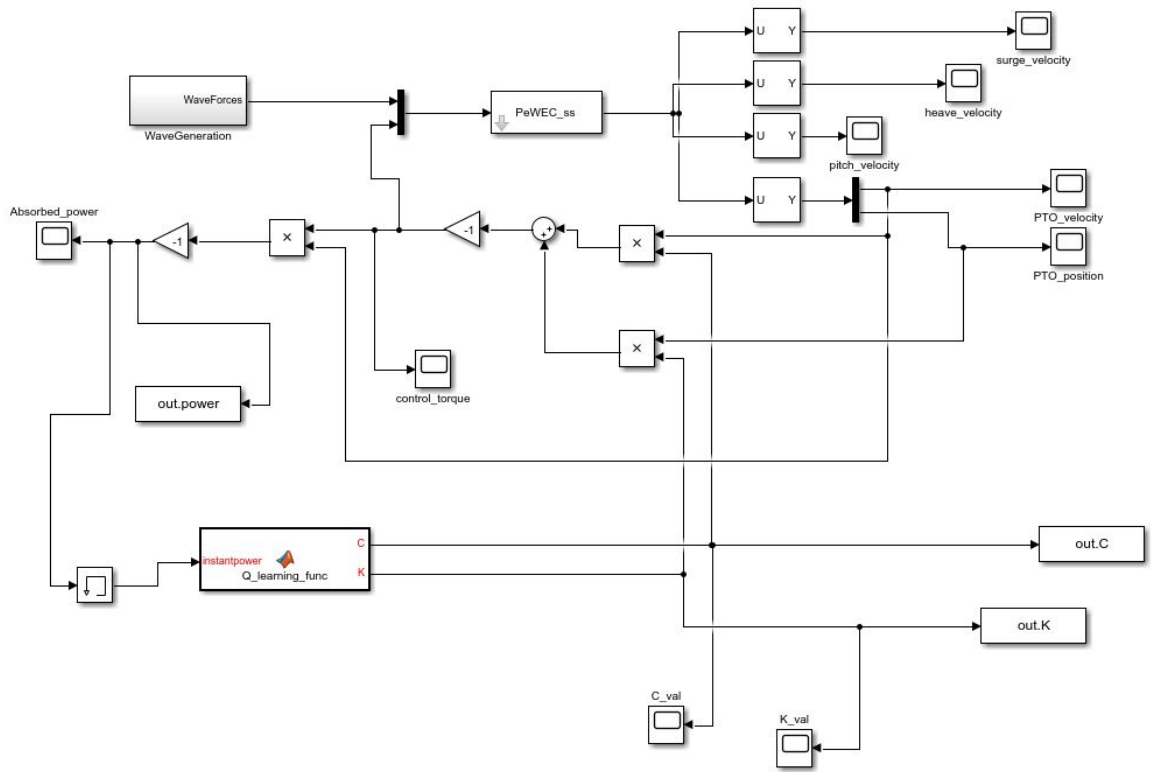
**Figure 4.43:** Policy obtained for the sea state 3, with an average number of visits per state of 7.37.

All the policies obtained converged to the correct state in a reasonable amount of time.

### 4.3.4 Online Q-Learning

Like what has been done with the MSD model, next step has been that of developing an online version of the algorithm with the PeWEC.

This algorithm has the same structure of the MSD one, but different parameters and a more complex model of the system.



**Figure 4.44:** Simulink model used to simulate the system.

Also in this case, a total simulation time of 100 000 seconds has been taken, and a fixed step time of 0.05 seconds.

The time for averaging the power was initially set to 300 seconds and successively was brought to 100 seconds in order to decrease the time needed for convergence. Like with the MSD algorithm, in order to find the optimal learning parameters, multiple simulations have been run with different learning parameters, their results were saved and finally compared.

Again, as learning parameters have been considered  $\epsilon_0$ ,  $\alpha_0$ ,  $N_\alpha$  and the number of episodes, with values:

1.  $\epsilon = [0.5 - 0.7 - 0.8 - 0.7 - 0.9 - 0.95 - 0.99]$
2.  $\alpha = [0.5 - 0.7 - 0.8 - 0.7 - 0.9 - 0.95 - 0.99]$
3.  $N_\alpha = [1 - 2 - 3 - 5 - 7]$

4. number of episodes = [1 - 2 - 3 - 5 - 7]

Each combination of the learning parameters has been simulated in each of the 3 sea states.

In the end, 5 combinations resulted to converge to the optimal states in a reasonable amount of time for all the 3 sea states:

1. Set 1:  $\epsilon = 0.5$   
 $\alpha = 0.5$   
 $N_\alpha = 5$   
*number of episodes = 1*
  
2. Set 2:  $\epsilon = 0.5$   
 $\alpha = 0.7$   
 $N_\alpha = 5$   
*number of episodes = 1*
  
3. Set 3:  $\epsilon = 0.5$   
 $\alpha = 0.7$   
 $N_\alpha = 5$   
*number of episodes = 2*
  
4. Set 4:  $\epsilon = 0.5$   
 $\alpha = 0.8$   
 $N_\alpha = 7$   
*number of episodes = 2*
  
5. Set 5:  $\epsilon = 0.7$   
 $\alpha = 0.7$   
 $N_\alpha = 2$   
*number of episodes = 2*

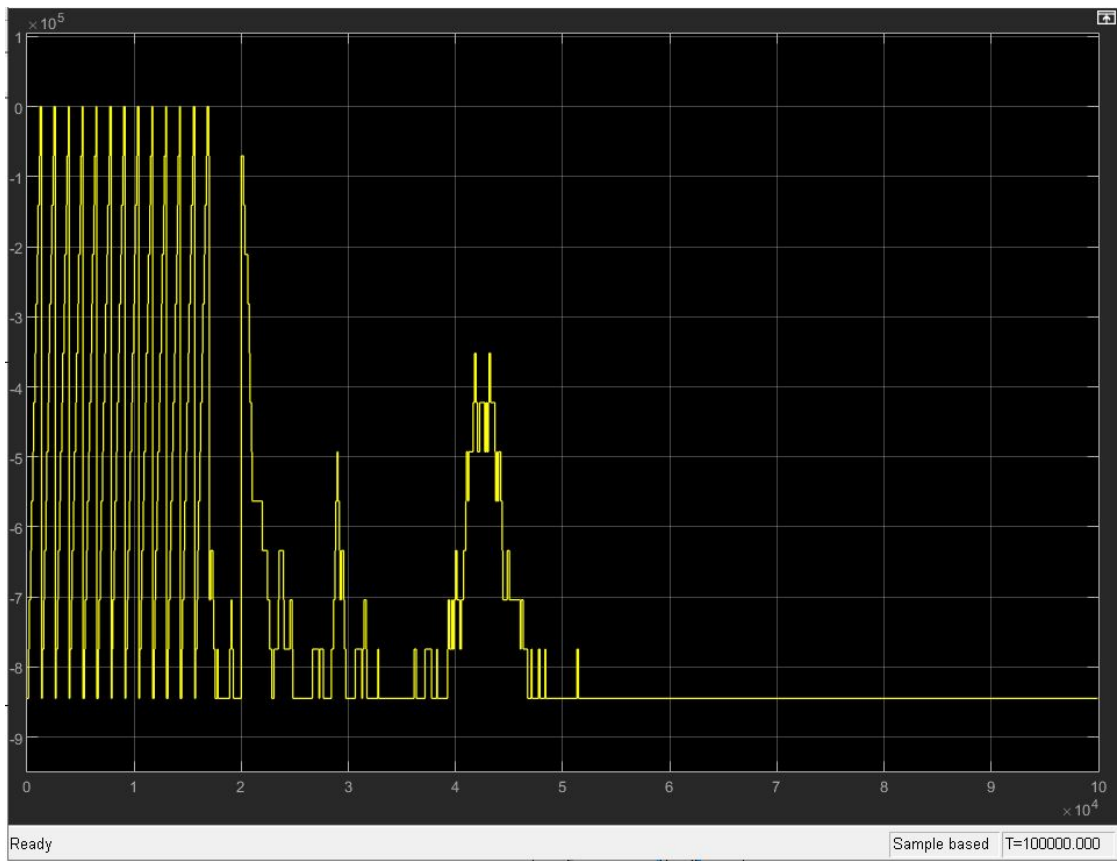
All these combinations converged to the optimal control states in all sea states in between 20 000 - 60 000 seconds.

Like what obtained with the MSD, the optimal learning parameters are those who find the optimal trade-off between exploration and exploitation, as can be seen from the fact that the first 4 sets have an  $\epsilon = 0.5$  and a  $N_\alpha = 5$  or 7, while the last combination has an higher  $\epsilon = 0.7$  but a lower  $N_\alpha = 2$ .

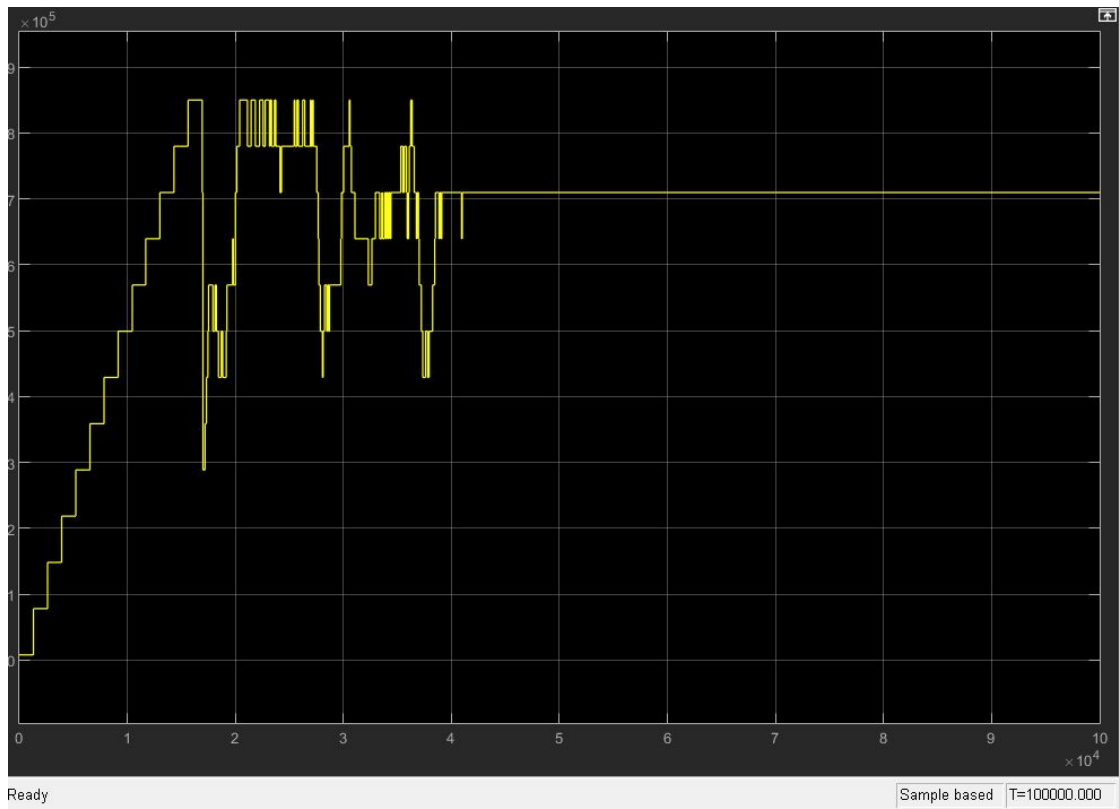
All the obtained results have a low number of episodes, since they have to converge relatively fast.



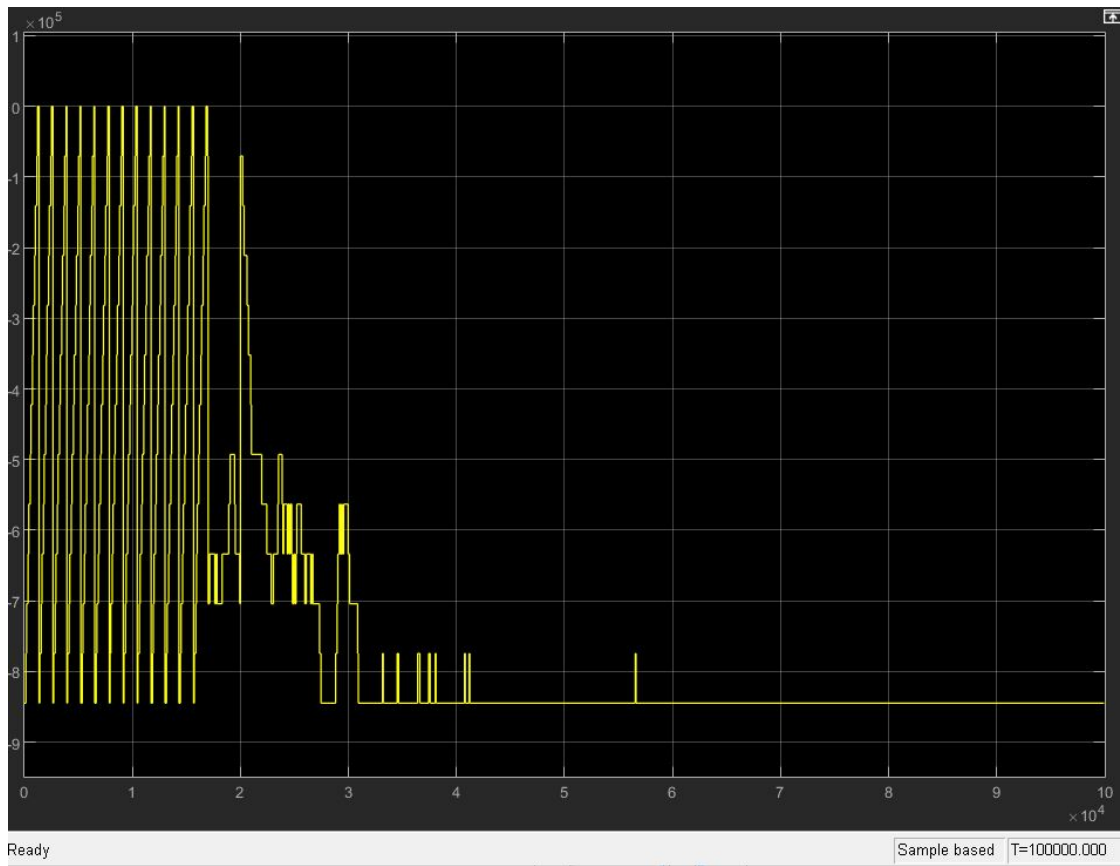
**Figure 4.45:** Control damping obtained with set 1 and sea state 1 ( $T=5$ )



**Figure 4.46:** Control stiffness obtained with set 1 and sea state 1 ( $T=5$ )

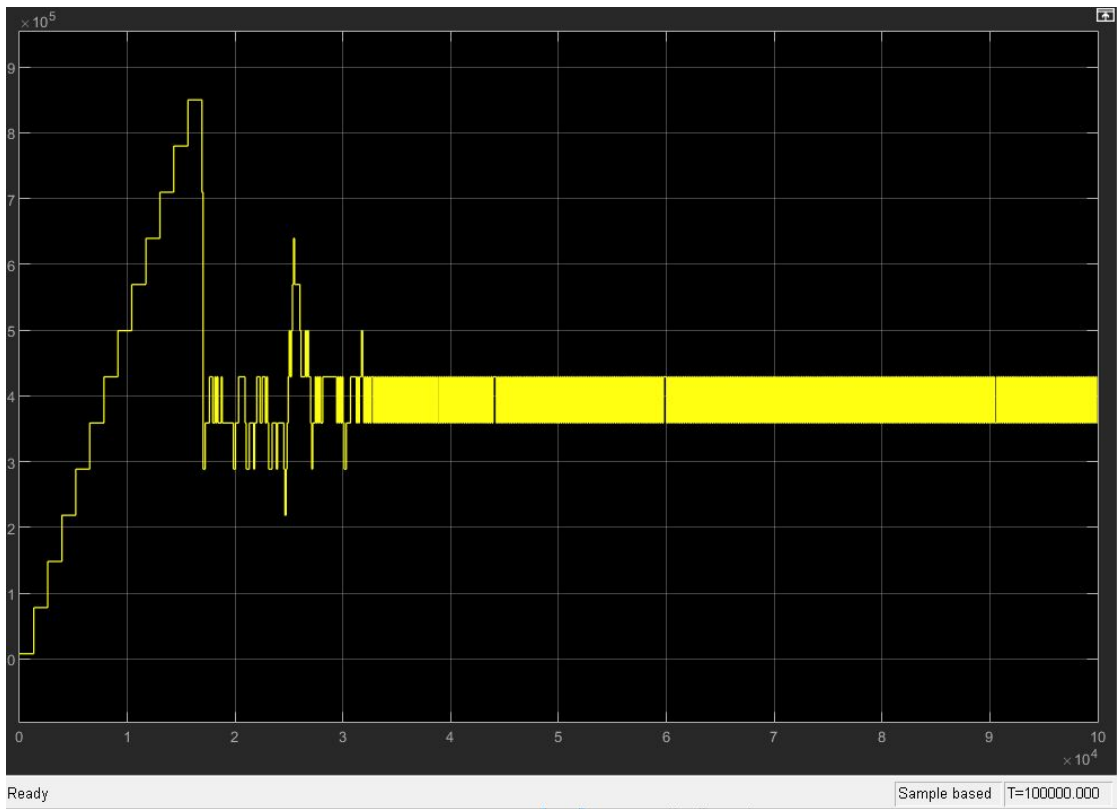


**Figure 4.47:** Control damping obtained with set 1 and sea state 2 ( $T=6$ )

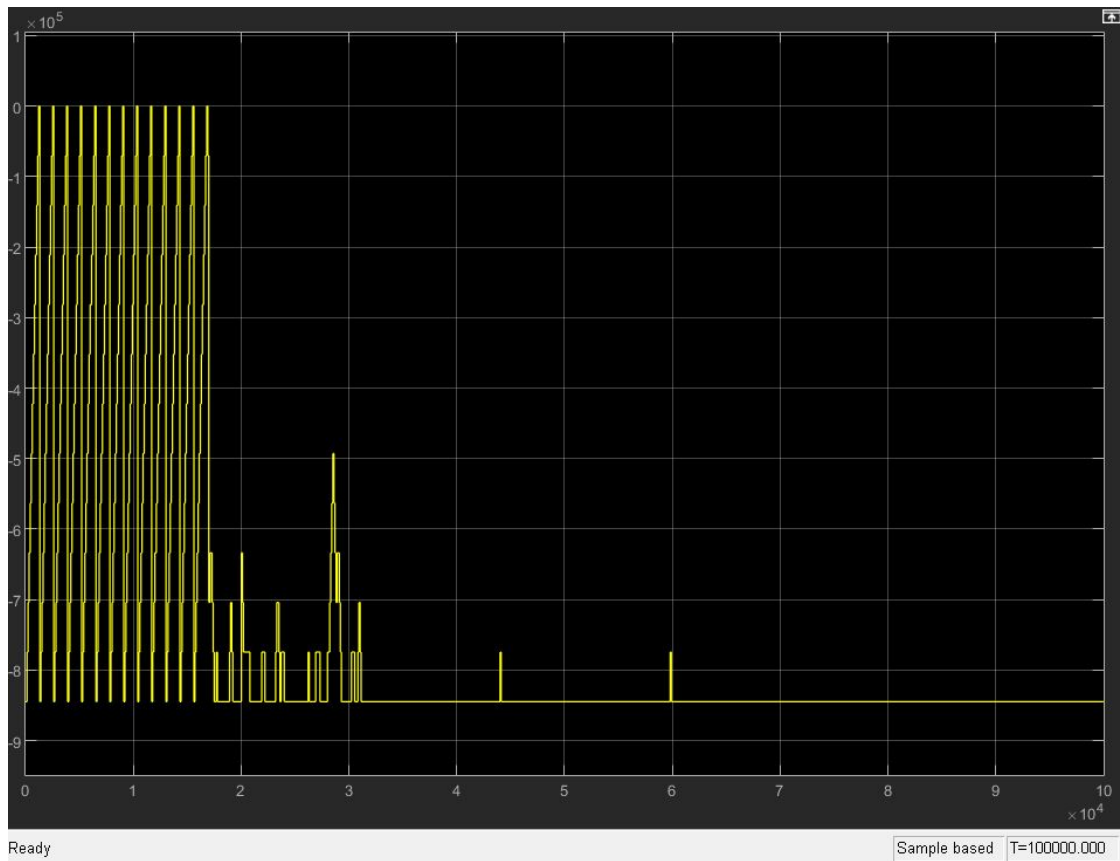


**Figure 4.48:** Control stiffness obtained with set 1 and sea state 2 ( $T=6$ )

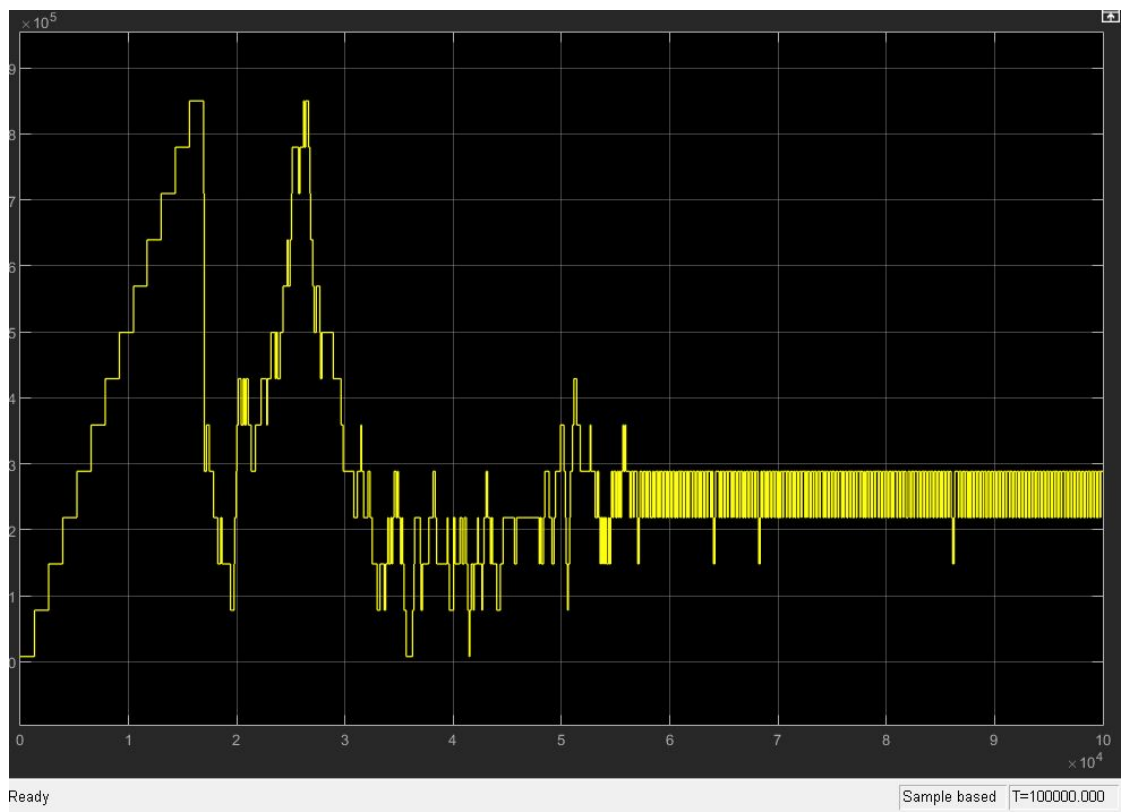




**Figure 4.49:** Control damping obtained with set 1 and sea state 3 ( $T=7$ )



**Figure 4.50:** Control stiffness obtained with set 1 and sea state 3 ( $T=7$ )



**Figure 4.51:** Control damping obtained with set 5 and sea state 1 ( $T=5$ )

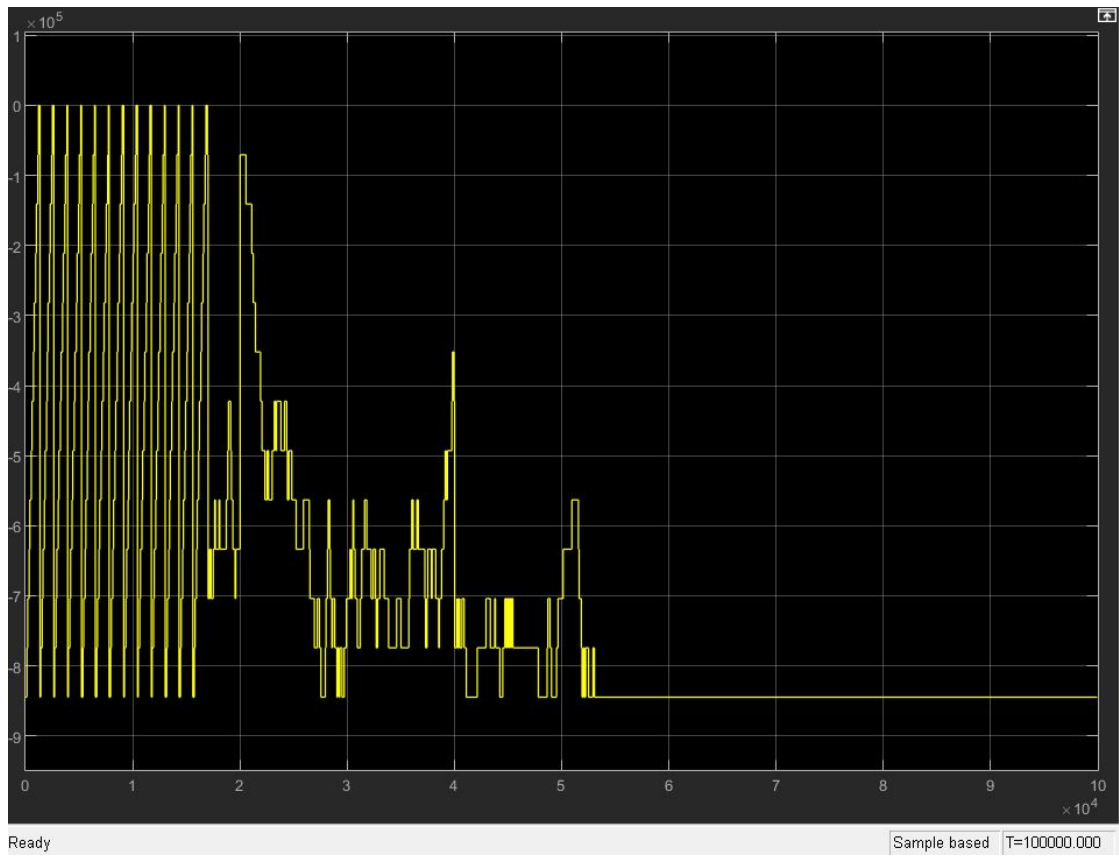
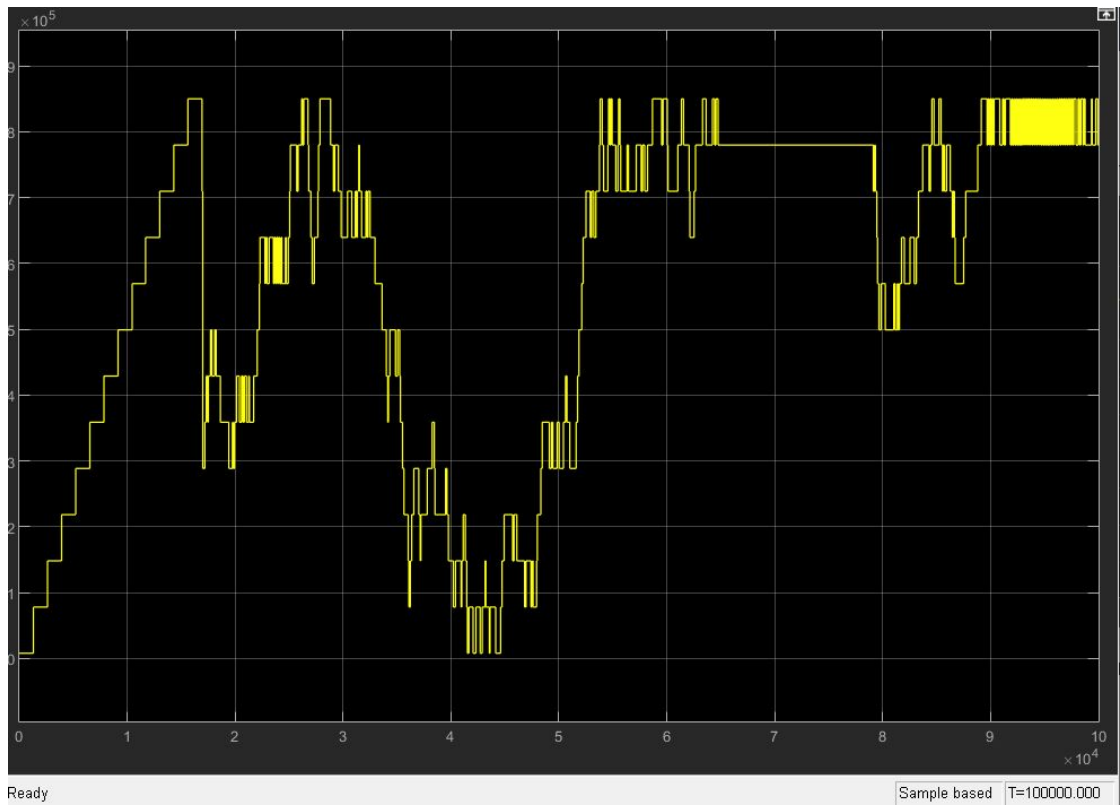
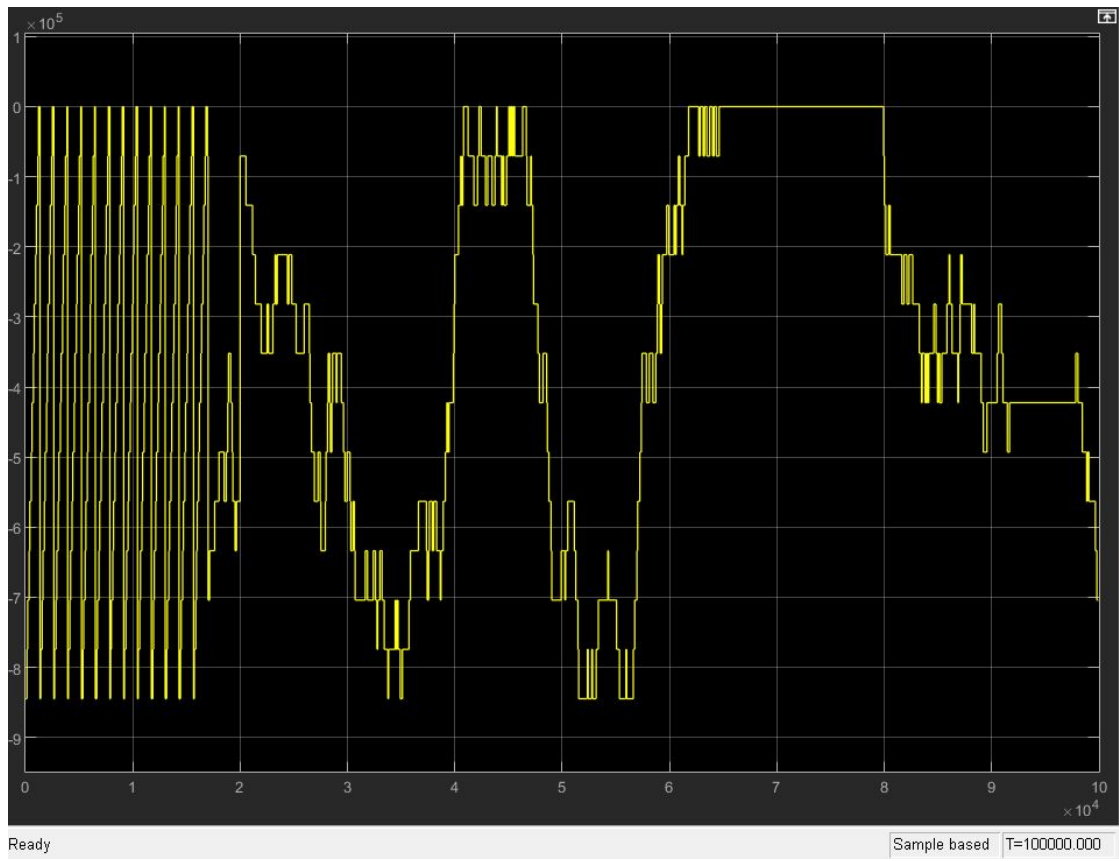


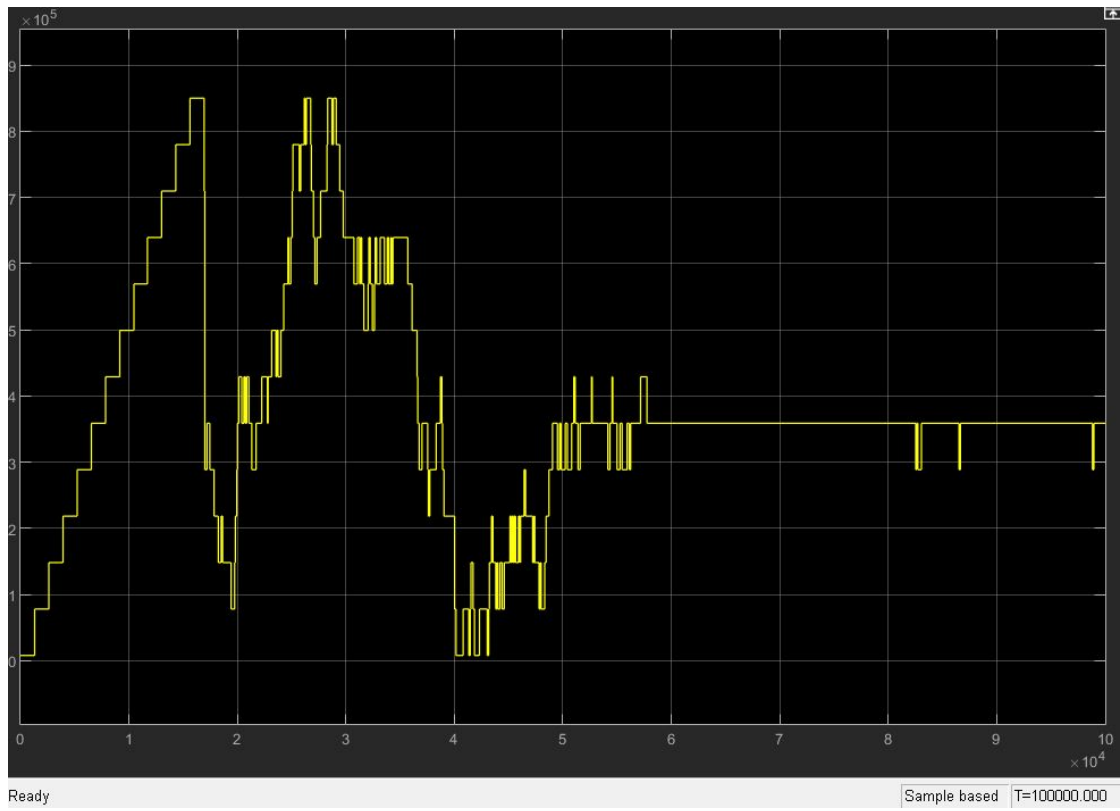
Figure 4.52: Control stiffness obtained with set 5 and sea state 1 ( $T=5$ )



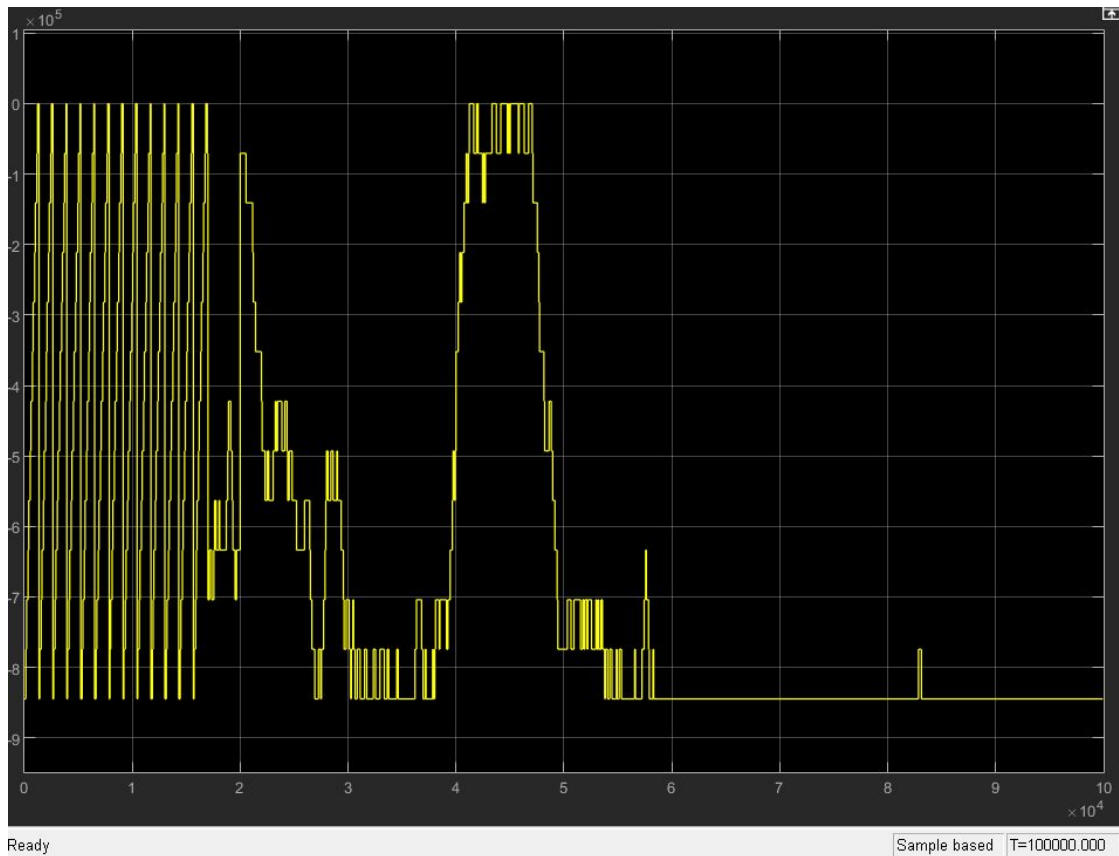
**Figure 4.53:** Control damping obtained with set 5 and sea state 2 ( $T=6$ )



**Figure 4.54:** Control stiffness obtained with set 5 and sea state 2 ( $T=6$ )



**Figure 4.55:** Control damping obtained with set 5 and sea state 3 ( $T=7$ )



**Figure 4.56:** Control stiffness obtained with set 5 and sea state 3 ( $T=7$ )

As it can be seen from the figures, multiple sets of learning parameters converge to the optimal state values (or almost equivalent ones for sea state 2) with different learning times.

The lowest learning time achieved is 30 000 seconds, including the first 18 000 seconds for the initialization.

This will help in passing to a real physical implementation, where the initialization can be done offline through simulations and then the actual learning can be done directly while working on the sea.

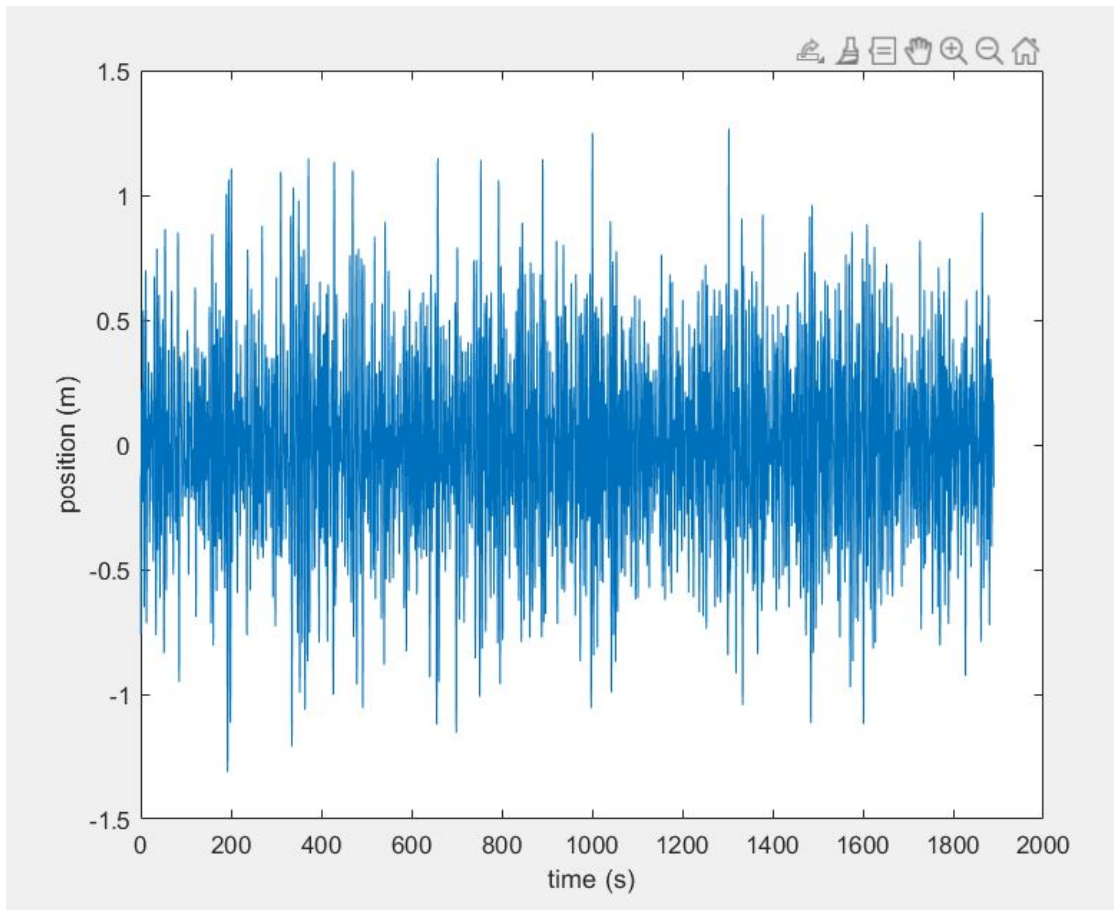
So a device using this algorithms would take around 3-4 hours to learn a sea state, making it a viable option.

### 4.3.5 Towards irregular waves

At the end, the capability of dealing also with irregular waves has been tested. Having 3 matlab data, with the data of 3 different irregular waves, an algorithm

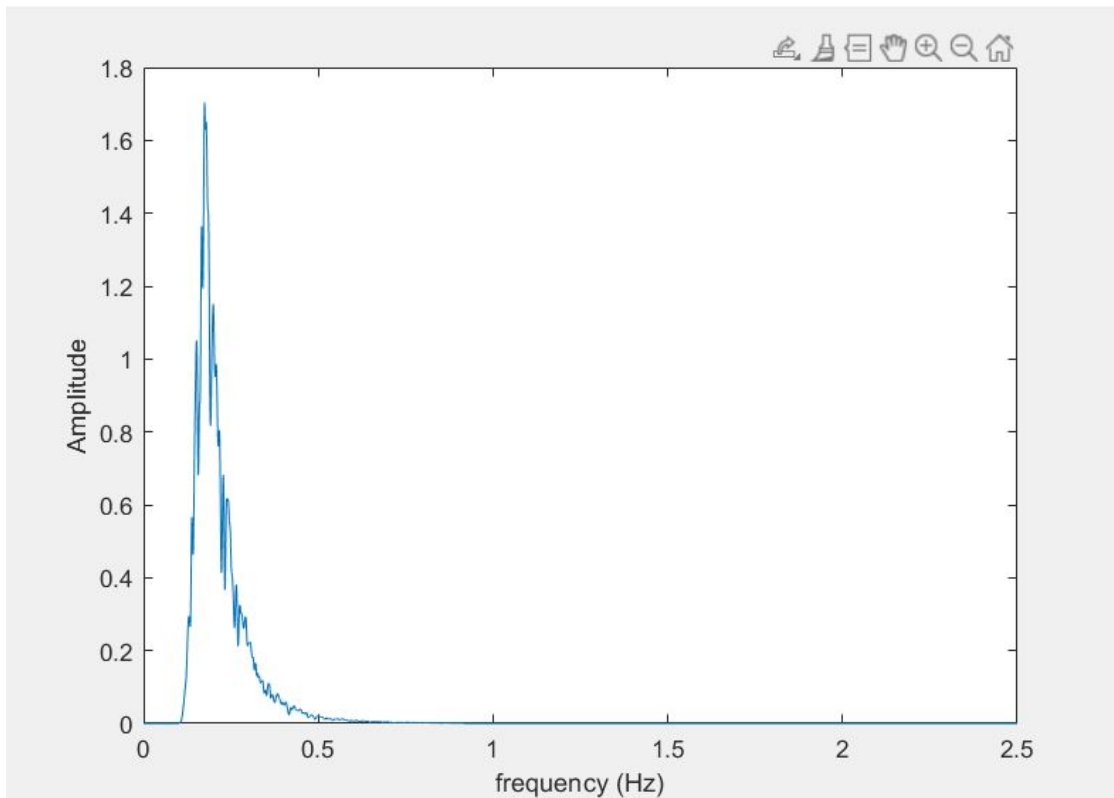


for extrapolating from this data the significant height and the equivalent period of the wave has been developed.



**Figure 4.57:** Example of an irregular wave

The signal in the time domain gets converted into frequency domain thanks to the 'pspectrum' command on MatLab.



**Figure 4.58:** Content in frequency of the wave

Having the spectrum of the wave, it is finally possible to calculate the spectral moments and use them to estimate the significant height and the equivalent period.

1. Wave 1:  
 $H_s = 1.5$  and  $T_e = 5$   
 and the estimated values resulted:  $H_s = 1.487$  and  $T_e = 4.994$
  
2. Wave 2:  
 $H_s = 1.5$  and  $T_e = 6$   
 and the estimated values resulted:  $H_s = 1.4923$  and  $T_e = 6.0358$
  
3. Wave 3:  
 $H_s = 1.5$  and  $T_e = 7$   
 and the estimated values resulted:  $H_s = 1.503$  and  $T_e = 7.0386$

# Chapter 5

## Conclusions

With the work done in this thesis, it has been possible to prove that Reinforcement Learning techniques are both effective and viable for control application of wave-energy converters: the use of states and actions helps in naturally translating the WEC's situation into a Markov decision process and the fact that the state transition is deterministic with respect to the actions taken speeds up the convergence of the policy, which needs around 1-2 visits per single state (100-200 seconds per single state) (for an entire sea state around:250 visits, 5 hours) in order to correctly converge, with the proper learning parameters and an offline initialization done with simulations.

Moreover, the real-time data-driven nature of the controller intrinsically deals with the adaptability problem: the controller is able to autonomously redirect its policy to the correct one, after a change in the system has happened, even if at the cost of an higher learning time for convergence increases a bit (of around 30 percent).

Further studies should be made on the usage of constraints: in this work only hard constraints have been used for avoiding that the controller goes out of the allowed range of damping and stiffness. The usage of soft constraints (i.e. penalties) should be investigated along with its effectiveness to avoid harmful situations to the device.

Finally, for the irregular waves, here it has been showed that they can be easily translated into regular waves by using the significant height and the equivalent period, but this still has to be implemented in a real time controller and test that it actually converges to a (sub)optimal policy.

# Bibliography

- [1] BP Statistical Review of World Energy 2020. In: (2020) (cit. on p. 1).
- [2] European Green Deal. In: (2019) (cit. on p. 1).
- [3] Johannes Falnes. «Ocean Waves and Oscillating Systems: Linear Interactions Including Wave-Energy Extraction by Johannes Falnes». In: (2002) (cit. on p. 2).
- [4] Michael E. McCormick. «Ocean wave energy conversion». In: (1981), p. 233 (cit. on p. 2).
- [5] John Ringwood Bingyong Guo. «A review of wave energy technology from a research and commercial perspective». In: (Oct. 2021) (cit. on p. 2).
- [6] Bryony DuPont Ali Trueworthy. «The Wave Energy Converter Design Process: Methods Applied in Industry and Shortcomings of Current Practices». In: (Nov. 2020) (cit. on p. 2).
- [7] John Ringwood Bingyong Guo. «Geometric optimisation of wave energy conversion devices: A survey». In: (May 2021) (cit. on pp. 2–5, 8, 9, 11).
- [8] Hasselman. «Measurements of wind-wave growth and swell decay during the joint North Sea wave project (JONSWAP)». In: (1973) (cit. on p. 6).
- [9] «Sergej Antonello Sirigu, Giovanni Bracco, Mauro Bonfanti, Panagiotis Dafnakis, Giuliana Mattiazzo. On-board sea state estimation method validation based on measured floater motion». In: (2018) (cit. on p. 7).
- [10] «AMETS berth B wave buoy. 2020, <http://www.oceanenergyireland.ie/> Observation. [Accessed 10 September 2020]». In: () (cit. on p. 7).
- [11] Alu Unim W. E. Cummins Wwllfil Iuhhl. «The Impulse Response Function and Ship Motions». In: (1962) (cit. on p. 10).
- [12] João C. C. Henriques António F. O. Falcão. «Effect of non-ideal power take-off efficiency on performance of single- and two-body reactively controlled wave energy converters». In: (2015) (cit. on p. 10).
- [13] Bacelli G. «Optimal control of wave energy converters.» In: (2014) (cit. on pp. 11, 12).

- [14] Giorgi Penalba Retes and Ringwood. «A Review of non-linear approaches for wave energy converter modelling». In: (2015) (cit. on p. 12).
- [15] Ringwood Merigaud. «A Nonlinear Frequency-Domain Approach for Numerical Simulation of Wave Energy Converters». In: (2017) (cit. on p. 13).
- [16] T. Whittaker M. Folley. «Spectral modelling of wave energy converters». In: (2010) (cit. on p. 13).
- [17] J. Falnes Z. Ju. «State-space modelling of a vertical cylinder in heave». In: (1995) (cit. on p. 13).
- [18] J. Falnes K. Budar. «A resonant point absorber of ocean-wave power». In: (1975) (cit. on p. 14).
- [19] Enrico Anderlini. «Control of wave energy converters using machine learning strategies». In: (2017) (cit. on pp. 14, 21, 32, 33, 35, 38, 41, 54).
- [20] Francesco Fusco John V. Ringwood Giorgio Bacelli. «Energy-Maximizing Control of Wave-Energy Converters». In: (2014) (cit. on pp. 14, 19).
- [21] J. Falnes. «Ocean waves and Oscillating systems». In: (2005) (cit. on p. 15).
- [22] Wave Energy Scotland. «Control Requirements for Wave Energy Converters Landscaping Study: Final Report. Technical report». In: (2016) (cit. on p. 16).
- [23] E. Vidal R. Hansen M. Kramer. «Discrete displacement hydraulic power take-off system for the wavestar wave energy converter.» In: (2013) (cit. on p. 16).
- [24] John V. Ringwood Nicolás Faedo Sébastien Olaya. «Optimal control, MPC and MPC-like algorithms for wave energy systems: An overview». In: (2017) (cit. on p. 17).
- [25] John V. Ringwood. «Wave energy control: status and perspectives». In: (2020) (cit. on p. 18).
- [26] A. G. Barto R. S. Sutton. «Reinforcement Learning». In: (1998) (cit. on pp. 21–23, 26, 27, 29, 30, 34–36).
- [27] supervisor: dr. Marco Wiering LEARNING TO PLAY CHESS USING REINFORCEMENT LEARNING WITH DATABASE GAMES Henk Mannen. In: (2003) (cit. on p. 23).
- [28] «Busoniu, Babuska, Schutter, Ernst. Reinforcement learning and dynamic programming using function approximators». In: (2010) (cit. on pp. 23, 32).
- [29] R. Bellman. «Dynamic Programming». In: (1957) (cit. on p. 24).
- [30] «Geramifard, Walsh, Sellex, Chowdari Roy, How. A Tutorial on Linear Function Approximators for Dynamic Programming and Reinforcement Learning.» In: (2013) (cit. on pp. 25, 28).

- [31] M. Niranjan G. A. Rummery. «On-Line Q-Learning Using Connectionist Systems.» In: (1994) (cit. on p. 28).
- [32] C. J. Watkins. «Models of Delayed Reinforcement Learning». In: (1989) (cit. on pp. 28, 30).
- [33] P. Dayan C. J. Watkins. «Machine Learning». In: (1992) (cit. on pp. 28, 30).
- [34] M. Riedmiller. «Neural fitted Q iteration - First experiences with a data efficient neural Reinforcement Learning method». In: (2005) (cit. on pp. 28, 32, 35, 37).
- [35] R. Parr M. G. Lagoudakis. «Least-squares policy iteration. The Journal of Machine Learning Research». In: (2003) (cit. on pp. 28, 36, 38–40, 42).
- [36] «Singh, S., Jaakkola, T., Littman, M. L., Szepes, C., and Hu, A. S. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms». In: (2000) (cit. on p. 29).
- [37] J. E. Moody and C Darken. «Fast Learning in Networks of Locally-Tuned Processing Units. Neural Computations». In: (1989) (cit. on p. 31).
- [38] Mark E. Parry Qing Cao. «Neural network earnings per share forecasting models: A comparison of backward propagation and the genetic algorithm». In: (2009) (cit. on p. 36).
- [39] M. T. Hagan and M. B. Menhaj. «Training Feedforward Networks with the Marquardt Algorithm». In: (1994) (cit. on p. 37).
- [40] R. A. Howard. «Dynamic Programming and Markov Processes». In: (1960) (cit. on p. 36).
- [41] D. P. Bertsekas and J. Tsitsiklis. «Neuro-Dynamic Programming». In: (1996) (cit. on p. 38).
- [42] R. Munos. «Error Bounds for Approximate Policy Iteration. Proceedings, Twentieth International Conference on Machine Learning». In: (2003) (cit. on p. 38).
- [43] «Bradtke, S. J., Barto, A. G., Kaelbling, P. Linear least-squares algorithms for temporal difference learning. Machine Learning». In: (1996) (cit. on p. 38).
- [44] D. Koller and R. Parr. «Policy Iteration for Factored MDPs. In Uncertainty in Artificial Intelligence». In: (2000) (cit. on p. 39).
- [45] «Fabio Carapellese, Edoardo Pasta, Bruno Paduano, Nicolás Faedo, Giuliana Mattiazzo. Intuitive LTI energy-maximising control for multi-degree of freedom wave energy converters. The PeWEC case». In: (2022) (cit. on p. 83).
- [46] Paolo Brandimarte Daniele Giovanni Gioia Edoardo Pasta and Giuliana Mattiazzo. Data-driven Control of a Pendulum Wave Energy Converter: a Gaussian Process Regression approach. In: (2022) (cit. on p. 84).