École Polytechnique Fédérale de Lausanne

Hierarchical Reinforcement Learning to control the AWAKE Electron Beamline

by Borja Rodriguez Mateos

Master Thesis

Approved by the Examining Committee:

Prof. Dr. Sc. Mike Seidel
Thesis Advisor

Dr. Snuverink Jochem
External Expert

Dr. Verena Kain & Dr. Tatiana Pieloni
Thesis Supervisor

EPFL PH LPAP
BSP 610 (Cubotron)
Rte de la Sorge
CH-1015 Lausanne

September 19, 2023

If the first person at the table takes the napkin to their right, then there is no other choice but for the others to take the right napkin. The same goes for the left. This is called a society.

Acknowledgments

I would like to thank Pr. Dr. Sc. Mike Seidel and Dr. Tatiana Pieloni for introducing me to the incredible domain of particle accelerators and letting me finalize my master with such a fascinating topic.

My utmost gratitude goes to Dr. Verena Kain without whom this thesis would not have been possible. Her academic and non-academic guidance throughout this semester has been invaluable. She marked the trail to follow and integrated me in her team from day one. Exchanges and advice from Dr Michael Schenk have given a remarkable added value to the work. I am thankful to him.

Furthermore, to my mentors along my academic journey, Juanjo, Antonio, Jake, Ian, Sungjoong and Daeho: thank you for lighting the path, I aspire to be able to stand next to you all.

Finally, my heart goes to my wonderful family and friends, your support and being by your side is a true blessing.

Lausanne, September 19, 2023 Borja Rodriguez Mateos

Contents

List of Figures

Abstract

New generation particle accelerators are increasingly more complex and have more stringent requirements on parameter quality and efficiency. The current control algorithms for accelerators mostly rely on linearization of accelerator dynamics and/or manual tuning of parameters. In order to meet the expectations of future particle accelerators, control systems are evolving and increasingly embrace machine learning techniques. Deep Reinforcement Learning (DRL) is a promising approach to develop smart and efficient controllers that can potentially run particle accelerators processes autonomously. In this thesis an autonomous controller based on Hierarchical Deep Reinforcement Learning was developed and applied to learn how to correct the AWAKE electron line trajectory. The proposed algorithm consists of a goal-based two level hierarchy of policies that learn optimal control without model of the control problem. It uses so-called off-policy experience replay to be sample-efficient. By abstracting decision making into two layers of policies, studies have shown that the controller manages to solve more complex tasks with longer time horizons. The algorithm, developed in the course of this master projected, is evaluated on the AWAKE electron line steering task, popular for testing novel control methods due to its high repetition rate and low risk of damage to the machine. The final implementation of the hierarchical controller manages to score an average success rate of 99.85% for AWAKE electron line trajectory correction and is able to solve the task in two steps on average. The possibility to use pretrained agents as part of a hierarchical control structure is particularly appealing as it allows to re-use learned low-level "skills". When using a classically pretrained controller for the lower-level agent, it turned out that resulting algorithm can make good use of the available "skills". The success rate with this set-up was 80.3%.

This thesis gives a brief introduction of the concepts of Reinforcement Learning, before describing in detail the AWAKE trajectory steering problem. The implementation of the chosen Hierarchical Reinforcement Learning HIRO for the AWAKE steering problem will also be summarised. The achieved performance with this RL algorithm will be shown at the end of this thesis.

Chapter 1

Introduction

In this chapter the evolution of charged particle accelerators, from their advent to their current designs, will be summarised. Then, motivated by the increase in the accelerators' complexity, the future of the control systems of these machines will be discussed with a focus on model-based and model-free control. The subject of this thesis is the development of a novel control algorithm which will be tested on the task of trajectory steering at the AWAKE facility. The final part of this chapter will therefore give an overview of the AWAKE trajectory steering problem.

1.1 History of particle accelerators

Particle accelerators were originally developed for particle physics, but in the course of time had great beneficial impact on many other fields. Examples are medicine, biology and material sciences. The first experiment that could be considered as linear accelerator was built by Wilhelm Röntgen in 1896 to produce X-rays with an electron electrostatic accelerator for the first radiography. The design of the linear accelerator was then much improved in 1924 by Gustav Ising when he made the first proposal for a linear accelerator using repetitive application of voltages along the beam line with RF fields instead of a simple anode. An accelerator based on this principle was built some years later by Rolf Widerøe.

Cockcroft and Walton designed their eponymous electrostatic accelerator using a voltage diode multiplier in 1932. It managed to accelerate protons up to 1 MeV and was considered as the first "high energy" accelerator. It was used for break-through experiments for nuclear fission.

The idea of circular designs for accelerators was at first rejected as it was thought that the particles would loose as much energy in a turn as they would gain in the accelerating cavity. Using Maxwell-Faraday's equation in its integral form $\int_\Gamma \mathbf{E}\cdot\mathbf{ds} = -\frac{\partial}{\partial t}\int_S \mathbf{B}\cdot\mathbf{n}da$ which states that with a time varying magnetic field, one could in theory accelerate particles in a closed loop. Widerøe used the induction principle to make the first design of a circular accelerator with constant radius called a Betatron. The idea relied on varying the magnetic flux through a core, which would induce an accelerating longitudinal electrical field for the electrons residing in a vacuum chamber orthogonal to the magnetic field. On paper this device could accelerate electrons up to a few tens of MeVs,

much more than any electrostatic accelerator or any particle produced by a radioactive decay. A machine accelerating electrons through magnetic induction was built successfully almost 20 years afterwards by Kerst [32].

Reading about Widerøe's discovery of the linac, Lawrence was prompted to invent the cyclotron - an accelerator consisting of two half discs separated by an accelerating gap. Ions accelerated by this device would be produced in the center of it and would spiral out of the machine horizontally due to a constant vertical magnetic field. This accelerator has a constant revolution frequency in the classical regime up to 50 MeV. The first concepts of transverse focusing, pulsed beam or phase stability were developed when optimizing the performances of this machine to be operational in the relativistic regime.

It was during a night shift supervising the industrial separation of Uranium isotopes that Oliphant made the first proposal for a pulsed magnetic ring, fundamental to the synchrotron [76]. The concept was inspired from the phase stability of the cyclotron where the frequency of the alternating accelerating field was increased with the energy of the particle. A parallel would be drawn with the bending magnetic field in order to keep the magnetic rigidity constant ($B\rho \propto p$) and the design of a circular accelerator where the particles trajectory would assume a constant radius. Moreover, the particles are accelerated with an RF voltage in cavities or gaps along their trajectory. It was then formalized in 1945 by MacMillan and Veskler independently, stating the conditions for the magnetic field and the accelerating frequency to maintain the orbital period in a cyclotron. This relation would go on and be called **phase stability** and would be used in all RF accelerators except the fixed frequency cyclotron.

The first electron synchrotron was built in 1946 by Goward and Barnes [20] modifying an old betatron that was used for X-raying unexploded bombs in the streets of London. This accelerator was able to accelerate the particles up to 8 MeV. The next year, Oliphant, Gooden and Hyde made the first proposal for a proton synchrotron that could produce protons up to 1 GeV. Nevertheless, the first team to complete the construction of their machine was the Brookhaven National Laboratory in 1952, naming their 3 GeV accelerator the "Cosmotron" [8].

The transverse focusing on the early designs of synchrotrons relied on the mechanism of "weak focusing", where the guide field in the magnets decreases slightly when the radius increases and its gradient is constant at a fixed radius around the machine. This constrained the aperture of the beam vacuum chambers to be large and correspondingly made the magnets bulky and costly. It was in the development of the Cosmotron, that Courant, Livingston and Snyder, found that by alternating some of the yokes of the bending magnets, the focusing of the beam would be improved. This enabled them to propose the strong focusing mechanism, where consecutive focusing and defocusing magnetic lenses are alternated in order to alternatively focus in one plane while defocusing in the orthogonal one. With this mechanism, the synchrotron overshadowed the synchro-cyclotron and betatron in the race for higher energies.

In order to obtain higher energies for particle production in physics experiments, the center-of-mass energy is what needs to be maximized. In the quest for the energy frontier, *Fixed target* experiments were gradually replaced by particle colliders as the amount of energy available follows

the relation $\gamma_F = 2\gamma_C^2$ with γ_F and γ_C being respectively the amount of energy available in a fixed target experiment and in a collision. The first collider built was a Princeton-Stanford e-e experiment that stored its first beam in 1962. This experiment was still using weak focusing and had an energy of 500 MeV per beam, it was used for e-e scattering experiments. Storage ring colliders still dominate the energy frontier in high energy physics nowadays.

The use of superconductive magnets in proton machines permitted to have higher magnetic fields and thus being able to control hadron beams at the highest energies. These advancements coupled with innovations in vacuum technology, RF cavities and control systems permitted to build more and more powerful particle accelerators. The increase in beam energy so far was culminated by the Large Hadron Collider (LHC) - a 27 km long circular hadron collider at CERN that performs collisions of protons and Pb ions up to 7 and 1150 TeV respectively.

Since the advent of the first accelerators the increase of beam energy has been exponential. A fragment of this rise is shown in Fig. 1.1 for both Hadron and Lepton colliders.

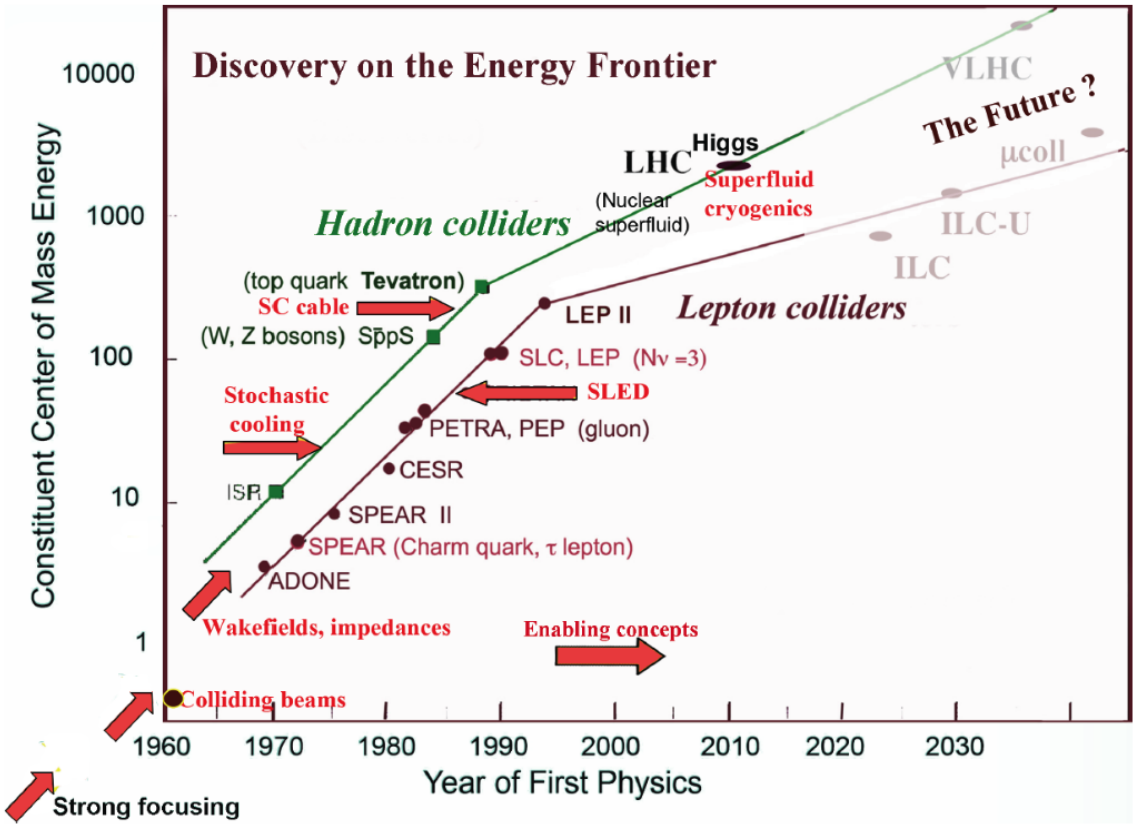

Figure 1.1: Livingstone chart [3] for colliders. It shows the maximum reach in center of mass energy in GeV for the various accelerators as function of time.

With the accelerators becoming larger and larger with more and more components, the complexity of the control system needs to increase as well to guarantee operability while keeping cost down.

The control systems need to provide for more and more sophisticated operational scenarios with parameters pushed to the limit and strict reproducibility requirements. Due to practical limitations for maximum reachable magnetic fields, the race for higher beam energies goes through having larger and thus more complex accelerators. The LHC at CERN for example has a circumference of 27 km and its potential successor, the FCC, might have a circumference of 100 km. The LHC consists of approximately 8000 superconducting magnets alone, with many hundreds of control knobs. A sophisticated control system with many abstraction layers is indispensable for its reliable operation.

1.2 Towards more efficient and autonomous accelerators

Similarly to the beam energy in accelerators, the computing power has been exponentially increasing since the 1970s and the cost per transistor in a microchip has steadily dropped. Coupled with the increasing complexity of novel particle accelerators and thus the need for increased abstraction or automation through the control system to allow operability, the introduction of more advanced control methods that incorporate artificial intelligence (AI) has become inevitable. In this new control paradigm for particle accelerators, manual optimization and stabilisation of parameters is increasingly replaced by sophisticated, autonomous optimisation algorithms [14]. These algorithms are able to solve multi-parameter optimisation problems while adhering to safety constraints. Together with classical control algorithms, AI for enhanced diagnostics, preventive maintenance systems and other, they will eventually allow for autonomous accelerators and hence re-define the operational model of the next generation particle accelerators.

In the past decade or so, a wide range of different AI algorithms have been applied to particle accelerators. Genetic algorithms have been used to optimize the properties of particle beams [79], Bayesian optimization with Gaussian Processes is used for machine tuning [33], artificial neural networks are used for improving the prediction and optimization of the output radiation of Free Electron Lasers[64] and better ML control policies for sub-systems of accelerators are being developed[31].

1.3 Controlling CERN's particle accelerators

Digitalisation and advances in information technology allowed to replace analog knobs, ampermeters and oscilloscopes with remote front-end interfaces and computer controls. Large accelerators can possess up to tens of thousands of different components, from accelerating units and magnets to vacuum pumps and gauges, power supplies as well as position, intensity and profile monitors. Multi-tiered control systems together with timing and triggering systems are necessary to orchestrate the various processes from injection to acceleration and finally extraction in a particle accelerator[2]. The CERN accelerators have increasingly adopted model-based control, where the settings of the various systems are derived from accelerator physics considerations, which are then translated into hardware parameters through transfer functions stored in a data base. In fact, the CERN accelerators are typically controlled with abstract high level physics parameters that can be defined 1-to-1 with

simulations [29]. As such, most accelerator settings nowadays are pre-calculated, or so-called "generated", for a given physics scenario and stored in a settings database. As long as the dynamics can be simulated, inverted and written down in closed-form for one-shot corrections, the LHC software architecture (LSA), that is used by almost all CERN accelerators today, has all the means to drive and regulate the particle beams. The parameters and operational scenarios of the LHC have been designed with great care to fall into this category (i.e. accurate simulations, analytical solutions for control problems, ...), thus the success of this collider. Still, it cannot be run autonomously and this is even more true for the older and lower energy CERN machines. The older accelerators often lack sufficient modelling, are multi-cycling (the LHC is set up for only one acceleration program during a given year) and frequently miss crucial beam instrumentation. Yet, also these machines have increasingly more complex physics programs with more and more stringent stability requirements.

Model-based control has obvious advantages. Recently control algorithms have been proposed that learn the dynamics or transition models from data either implicitly or even explicitly without prior knowledge while trying to solve the control problem. They learn through reward and are called reinforcement learning algorithms. Reinforcement learning (RL) algorithms will certainly be part of the AI portfolio for future accelerator control. RL has already shown remarkable performance in the domain of nuclear fusion, where it managed to find stable plasma configurations in the Tokamak à Configuration Variable (TCV) [13]. Similarly to particle accelerators, fusion reactor control requires high-dimensional and high-frequency closed-loop control. And it turned out that autonomous controllers based on deep RL are advantageous [13]. While the problem of trajectory or orbit correction for accelerators is completely solved for constant optics, it will be used as use case in this study to implement Hierarchical Reinforcement Learning that bears the prospect of solving more complex tasks with improved sample-efficiency.

1.3.1 Reinforcement learning for optimal control

Reinforcement learning deals with mapping situations (or so-called *states*) to optimal *actions*. It finds these mappings by maximizing a numerical reward signal [71]. The untrained so-called agent or controller does not know which actions are optimal at the start and must discover which actions yield the highest reward. During the learning process it updates its so-called policy. The basic terms and definitions of reinforcement learning will be presented in the following:

- **Agent:** Also called the "learner". It is one of the two main building blocks of the reinforcement learning setting along with the so-called environment (see below for the definition), in which the agent acts. The agent is the decision maker that learns what is the best action to take at each time step.

- **Environment:** The other fundamental element in reinforcement learning. It represents the problem definition or the "world" and the interactions in which the agent "lives". All problem specific information is contained in the environment, none of it resides in the algorithm itself. This allows for necessary modularity.

- **Action:** The setting or optimisation parameter by which the agent interacts with the environment, it cannot influence the rules or dynamics of the environment by it.
- **State:** Or "observation". It is a representation of the current version of the environment that the agent is in. Ideally, it has all the information that the agent needs in order to make the next decision. It is modified through the actions of the latter.
- **Reward signal:** Defines the goal of a reinforcement learning problem. It is given to the agent by the environment at every time step after applying the action as a single number called reward. Through its value, the agent learns which actions are advantageous or disadvantageous in a given state in the pursuit of its goal.
- **Value function:** Contrary to the reward which indicates what is good or bad immediately at every time-step, the value function specifies what is beneficial in the long term. It is related to the expected value of the cumulative reward starting at a given state.
- **Policy:** Map between the perceived states by the agent and a resulting action to be taken in those states. Most of the time, policies are stochastic and output actions with a certain probability. If the agent is fully trained it will know the "optimal policy", which at each state will issue the action that maximises the cumulative reward.
- **Model:** The model is a surrogate of the dynamics of the environment. For example, the model could predict the next state and next reward for a given state and action. It is used for planning, that is deciding on a set of consecutive actions before they are experienced by the environment. The types of algorithms to learn the policy observation-optimal action mapping can be divided into *model-based* approaches and *model-free* ones. With the former class of methods, the dynamics model is learned explicitly as part of the training and is used for e.g. planning. In the latter case the model is not explicitly available and the agent can only improve its policy by interacting with the environment. Both approaches will be briefly introduced below.

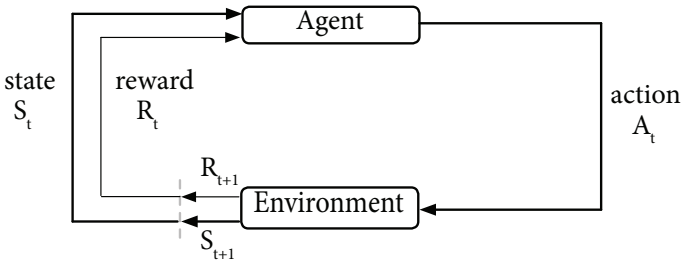

Figure 1.2: Schematic of the control loop for the agent-environment interactions that allows the reinforcement learning agent to learn the optimal policy. The environment informs the agent at every learning step about its current state S_t and the associated reward R_t. Based on this information, the agent acts on the environment through the action A_t and modifies the environment. This process is recursively performed until the agent performs the desired actions.

For obvious reasons it is extremely beneficial if the environment can be simulated and the agent trained "off-line" and only the trained agent is then applied to the real environment [31]. If the simulation is not accurate the "sim to real" transfer can be challenging, albeit techniques are available to help under these conditions, particularly if it is roughly known what is typically different without knowing exactly by how much [5].

1.3.2 Hierarchical Reinforcement learning for control

Hierarchical implementations in reinforcement learning subdivide a goal into several sub-tasks at different levels of abstraction. The agents at the higher levels plan for longer time scales on more abstract tasks, whereas in lower levels they plan at short time scales and interact directly with the environment.

By subdividing the task and having several agents solving them, it is possible to create more specialized learners that solve simpler tasks in cooperation with longer time horizons. It has been shown [51] [52] that hierarchical architectures are a promising approach to solve complex problems.

The task that will be solved in this study will be autonomous steering of the electron line in the AWAKE experiment using Hierarchical RL. The AWAKE electron source and beam line have been a popular test bed to for many novel control algorithms in the past [31][73] [65].

Chapter 2

Theoretical Background

This chapter will introduce the essential theoretical elements needed to understand and formulate how autonomous controllers learn in the setting of reinforcement learning. As RL will be deployed to solve a trajectory steering control problem later on, the classical trajectory steering algorithms in particle accelerators based on SVD using the so-called response matrix will also be discussed. A short excursion to Model Predictive Control (MPC) with data-driven models will follow the accelerator physics part. For the remaining chapter the reinforcement learning formalism will be in the center. The building blocs of Optimal Decision Making starting from Markov Decision Processes will be introduced as well as the basics of deep learning as it is required for some of the modern RL algorithms as in Deep Reinforcement Learning.

2.1 Classical model-based trajectory correction in particle accelerators

The classical model-based trajectory correction methods rely primarily on computing the beam transport matrices via the Twiss parameters between Beam Position Monitors (BPMs) and corrector dipole magnets. The transport matrix models the single particle phase-space coordinate transport in the horizontal or vertical plane from one location to another as the beam travels through a storage ring or a transport line. The β-function and other related Twiss parameters are defined by the focusing properties of the lattice and can be calculated for circular machines or in case of having the initial conditions in case of beam transfer by various accelerator modelling programs e.g. MAD-X [57].

2.1.1 From the transport matrix to the response matrix

In order to correct for trajectory imperfections, the two lattice elements that are used are Beam Position Monitors (BPMs) and corrector dipole magnets. BPMs are non-destructive diagnostics used widely in all types of particle accelerators. A BPM measures the charges induced by the electric field of the beam on two opposite insulated metal plates. The difference is used to determine the position of its center of mass [16]. Corrector dipole magnets are used to adjust the trajectory of the charged

particles in the vertical and horizontal plane wrt the design trajectory by providing "kicks" $\Delta\theta$ of typically mrad. To determine the deflection angle required to correct the position at a BPM by a certain Δx or Δy, the relation between the beam position measured at the BPMs and the correction angle at a corrector needs to be formulated. The transfer or transport matrix between the corrector and the BPM linearly models the transport of phase-space coordinates from the corrector location to the BPM, i.e. it linearly propagates x and x' from location (0) to location (1) and is a function of the lattice.. It can be derived in several manners (e.g. using the Twiss parameters):

$$\begin{pmatrix} x^{(1)} \\ x'^{(1)} \end{pmatrix} = \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{R}_{21} & \mathbf{R}_{22} \end{pmatrix} \begin{pmatrix} x^{(0)} \\ x'^{(0)} \end{pmatrix} \tag{2.1}$$

Our interest lies in describing the influence of a deflection angle $\Delta\theta$ at the location of a corrector magnet on the position Δx measured at a BPM. By developing the first equation of (2.1) with initial conditions $(x^{(0)}, x'^{(0)}) = (0, \Delta\theta)$, the following relation is obtained:

$$\Delta x = \mathbf{R}_{12}\Delta\theta \tag{2.2}$$

For an arbitrary number of BPMs measuring the position and trajectory correctors inducing deflection angles, the equation (2.2) can be extended to2.3. For a system of M BPMs and N trajectory correctors, the resulting set of equations reads:

$$\begin{pmatrix} \Delta x^{(1)} \\ \vdots \\ \Delta x^{(\mathrm{M})} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{00} & \dots & \mathbf{A}_{\mathrm{N0}} \\ \vdots & & \vdots \\ \mathbf{A}_{\mathrm{M0}} & \dots & \mathbf{A}_{\mathrm{MN}} \end{pmatrix} \begin{pmatrix} \Delta\theta^{(1)} \\ \vdots \\ \Delta\theta^{(\mathrm{N})} \end{pmatrix} \tag{2.3}$$

where $\Delta x^{(\mathrm{i})}$ is the position change measured at the i$^{\mathrm{th}}$ BPM and $\Delta\theta^{(\mathrm{j})}$ the induced deflection angle of the j$^{\mathrm{th}}$ trajectory corrector. The matrix $\mathbf{A} \in \mathbb{R}^{\mathrm{M}\times\mathrm{N}}$ is the so-called *response* matrix that relates the measured positions to the deflection angles. Its coefficients $\mathbf{A}_{ij}$ can be derived from Twiss parameters (see below) or data-driven methods. By measuring Δx at a fixed BPM i as a function of a varyingdeflection angles at a trajectory corrector j, linear regression allows to obtain the response matrix coefficient $\mathbf{A}_{ij}$.

2.1.2 SVD on response matrix

The most widely used linear control method for trajectory steering is based on Singular Value Decomposition (SVD) of the response matrix. Let M be the number of BPMs in the accelerator and N the number of correctors used for correction, the resulting elements of the response matrix $\mathbf{A}$ $\in \mathbb{R}^{\mathrm{M}\times\mathrm{N}}$ based on the TWiss parameters are [44]:

$$\mathbf{A}_{ij} = \frac{\sqrt{\beta_i \beta_j}}{2 \sin(\pi\nu)} \cos(\phi_{ij} - \pi\nu) \qquad \text{In a ring} \tag{2.4}$$

$$\mathbf{A}_{ij} = \sqrt{\beta_i \beta_j} \sin(\phi_{ij}) \qquad \text{In a transfer line} \tag{2.5}$$

with β_i and β_j the beta functions of the i^{th} corrector and j^{th} BPM, ν the betatron tune and $\phi_{ij} = |\psi_i - \psi_{cj}|$ the phase difference between location i and j. In the case of transport lines, the response matrix is sparse (i.e. triangular) as the position at every BPM can only be influenced by upstream correctors yielding $\mathbf{A}_{ij} = 0$ for $i < j$.

In order to correct the trajectory to a reference trajectory (i.e. find $\Delta\theta$ to correct by a given Δx), equation 2.3 needs to be inverted. In the case where there are as many BPMs as trajectory correctors, i.e. M=N, the response matrix is square, such that $\Delta\theta = \mathbf{A}^{-1}\Delta x$. In this case standard matrix inversion algorithms can be used which have a complexity of approximately $\mathcal{O}(N^{2.373})$ to $\mathcal{O}(n^3)$. When there are more BPMs than trajectory correctors, i.e. M > N, or for a non-full rank square matrix, a "pseudo-inverse" needs to be determined to find the correct deflection angels $\Delta\theta$. Singular Value Decomposition of the response matrix **A** is used for that purpose. It is a generalization of the eigen-decomposition of a square normal matrix into an orthonormal basis for any rectangular matrix.

Theorem 1 (Singular Value Decomposition (SVD))**.** *Let **A** be an (m × n) matrix with m≥n. Then there exist orthogonal matrices **U** (m × n) and **V** (n × n) and a diagonal matrix* $\Sigma = diag(\phi_1, ..., \phi_n)$ *(m × n) with* $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$ *such that*

$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^T$$

If $\sigma_r < 0$ *is the smallest singular value greater than zero, then rank(**A**) = r [70]. (Note that for orthogonal matrices* $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{1}$*.)*

According to theorem 1, the response matrix can be decomposed into two orthogonal matrices **U** and **V** and its corresponding diagonal form Σ, yielding $\mathbf{R} = \mathbf{U}\Sigma\mathbf{V}^T$. With the two orthogonal matrices, the BPM space and corrector space vectors are projected onto the eigenbasis of the response matrix. The transformed vectors are written as follows:

$$\Delta x^* = U^T \Delta x$$
$$\Delta\theta^* = V^T \Delta\theta$$

Equation (2.3) can be rewritten as $\Delta x^* = \Sigma\Delta\theta^*$ with $\Sigma \in \mathbb{R}^{m\times n}$ being a rectangular diagonal representation of the response matrix, which can be easily inverted to define the corrector strengths to correct to a certain reference trajectory. This method is particularly attractive for accelerator control because SVD minimizes in addition the root mean square (RMS) value of the solution vector $\Delta\theta$, thus minimizing the current used in the corrector magnets.

SVD is the state-of-art method for orbit correction and is widely used in the CERN accelerator complex. For example, the LHC orbit feedback system is based on SVD. The LHC has 1150 BPMs and 550 dipole correctors per plane, thus the response matrix of this system $A \in \mathbb{R}^{1150\times550}$ is very large [21]. This method is also available in the control system of the AWAKE electron line to correct the trajectory [31]. There are 11 beam position monitors and 11 trajectory correctors along the line. More on this topic will be discussed in section 3.1.

2.1.3 Model Predictive Control

Model Predictive control (MPC) is a powerful optimization strategy for feedback control. It uses a dynamics model to forecast the system's behaviour and optimize the control sequence to produce the best decision at the current time step [62]. It relies on the availability of a sufficiently accurate model - analytical or data-driven from simulation or the real environment. For the MPC formalism, the dynamics or transition model has the following form :

$$\begin{aligned} \dot{x} &= f(x,u,t) \\ y &= h(x,u,t) \\ x(t_0) &= x_0 \end{aligned}$$

where $x \in \mathbb{R}^n$ is the state of the system, $u \in \mathbb{R}^m$ the control input, $y \in \mathbb{R}^p$ the output (often related to loss, cost or objective) and $t \in \mathbb{R}$ is time. The initial condition x_0 is the initial state at $t = t_0$ and f can be any type of linear or non-linear function. In this setting, a solution to the differential equation for time greater than t_0 is sought. Most commonly, the model are linearised for the time varying and time invariant case, such that the time derivative of state and output are linear transformations of state and control input with coefficients that can either vary in time or not. These models can also be discrete in time if the system of interest is sampled discretely.

The solution to this type of control problem is the control law with the best closed-loop properties that solves an infinite horizon, constrained optimal control problem [62]. For this purpose the cost (or value function) is defined as :

$$V_\infty(x,u(.)) = \int_0^\infty l(x(t),u(t))\,dt \tag{2.6}$$

where $x(t)$ and $u(t)$ satisfy the dynamics $\dot{x} = f(x,u)$ and $l(.)$ being the stage cost function. This is then used to formulate an optimization problem $\min_{u(.)} V_\infty(x,u(.))$ with the constraints $\dot{x} = f(x,u)$ and $x(t_0) = x_0$. The solution to it is then be optimal $u^0_\infty(.,x)$ and a resulting optimal value function V^0_∞. In practice, directly finding an open-loop solution to the problem does not yield satisfying results for complex problems as it cannot plan for inconveniences that the controller might stumble upon. A feedback solution is usually the preferred way to solve these types of control problems, where the control value of only the first time step in the optimal control sequence is applied for the current measured value of state x and then another optimal control sequence is calculated for the

next time step. As the cost function $V(x, u(.))$ is usually not a convex function of $u(.)$, the solution to the above optimization problem is not easily found.

A practical example for a feedback solution using model predictive control for particle accelerators with data-driven dynamics models was developed by N.Bruchon for the FERMI free electron laser in Trieste [7]. The goal was to align the seed laser properly with the electron beam in the modulator undulator and obtain an intensity greater than a given threshold. To do so, the system was modelled in the following manner:

$$x_{k+1} = Ax_{(k)} + Bu_{(k)}$$
$$I_{(k)} = f(x_{(k)})$$

with $x_{(k)}$ being a concatenation of all the servo motors voltages controlling the angles of the mirrors for the laser at time k and $u_{(k)}$ being the the control input at that time measured in incremental voltages. The output $I_{(k)}$ is the intensity that needs to be maximized above a given threshold. In order to invert the dynamics and define the best control sequence, the iterative linear quadratic regulator (iLQR) was used. By discretizing the system in time, the iterative process can be formulated as :

$$x_{(k+1)} = g\left(x_{(k)}, u_{(k+)}\right)$$

with $g(.,.)$ a non linear function. This process results in a locally-optimal state-feedback control law able to minimize, in a finite time horizon, a quadratic cost function. MPC has become increasingly popular in controlling particle accelerators due to its ability to handle complex and non-linear systems with multiple related variables. It also allows to easily integrate operational constraints to ensure safe and stable operation. Furthermore, as shown in the example above, MPC can be data-driven where the non-linear function f describing the mapping between the detected intensity I and the state x is represented through a Neural Network trained on data collected before launching the controller. As the function f can be learnt using past experience, it improved the sample efficiency of the procedure. These concepts will be further explained in the subsection 2.4.1 and section 2.4.

2.2 Reinforcement Learning

Complex high dimensional control systems as the ones present in today's particle accelerators usually have to deal with non-linear processes where frequently the model of their dynamics cannot be written in closed form. Very often these processes are manually tuned if needed, or theoretical settings are applied which are (rarely) touched. Reinforcement learning could be a useful addition in these cases. RL agents learn control policies model-free. The dynamics can be time-varying, non-linear and multi-dimensional. In this section, the principals behind reinforcement learning i.e. Markov Decision Processes (MDPs) will be introduced followed by the base algorithms to obtain

optimal control.

2.2.1 Markov Decision processes

For reinforcement learning to solve the optimal control problem, it needs to be formulated as a Markov Decision Process (MDP). Optimal control primarily deals with continuous MDPs. In this setting the environment is called to be fully observable and an optimal policy will have all the information from the state to define the optimal action. An MDP is a classical formalization of sequential decision making, where actions influence not only the immediate reward at the end of the time step, but also subsequent possible future states and rewards which arise from having chosen said action [71]. Partially observable problems can be converted into MDPs or Bandits, such that they are solvable with RL. MDPs can deal with delayed reward, which results in the need to trade off between immediate and delayed reward during policy training. This is commonly called the exploration versus exploitation dilemma. Through exploration it is possible to find more information about the environment whereas with exploitation known information is straight away exploited to maximize the reward.

As seen in subsection 1.3.1, reinforcement learning problems are composed of two elements: an agent that is the decision maker and learns how to make better and better choices and an environment that contains the MDP. The agent interacts with the environment. And the environment models the system and its dynamics and adheres to the Markov property. Agent and environment interact dynamically as described in the Fig. 1.2 with the agent selecting actions and the environment judging them through the reward and presenting a new situation (a.k.a. state). These interactions are discretized in time. At each time step t, the agent receives some representation of the environment state $s_t \in \mathcal{S}$ and with this observation it chooses an action $a_t \in \mathcal{A}$. At the next time step, the agent receives a reward $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and the resulting next state $s_{t+1} \in \mathcal{S}$. $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ are the State, Action and Reward sets. This interaction can be described as a sequence or a trajectory in these spaces as :

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$$

Where A_t, R_t and S_t are respectively the observed state at time t, the action selected at that time and its corresponding reward. (Note the difference with respect to their lower capital counterparts which are the random variables and not the observed value.) In a finite MDP, the sets $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ have a finite number of elements. The remainder of this section will deal with finite MDPs. The developed formalism can be extended to infinite or continuous MDPs. The following definition 2.2.1 is paramount to the forumulation of MDPs:

Definition 2.2.1. A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t]$$

Being Markov means that all the relevant information from the history of the process is captured by the current state, such that the next state only depends on the current state and action. This

means that in the finite Markov case, S_t and R_t have well defined finite probability distributions that only depend on the previous action and state. The probability of observing the state s' and the reward r at time t is fully described by these "transition" probabilities:

$$p(s', r|s, a) = \mathbb{P}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a) \tag{2.7}$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \qquad \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{2.8}$$

This is valid for all s, $s' \in \mathcal{S}$, $r \in \mathcal{R}$ and $a \in A$. The deterministic function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ fully describes the dynamics of the MDP which in turn describes the environment's dynamics. The state-transition probability can be computed by summing over all possible rewards as:

$$p(s'|s, a) = \mathbb{P}(S_t = s'|S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r|s, a) \tag{2.9}$$

which describes the probability of being in state s' at the time t in function of the previous state and action. Other two useful functions are the expected reward function for state-action pair and state-action-next state triplet:

$$r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a) \tag{2.10}$$

$$r(s, a, s') = \mathbb{E}\left[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)} \tag{2.11}$$

In the case of a finite MDP, a transition matrix $\mathbf{P}$ as in equation (2.12) can be formulated. It defines all the transition probabilities between states and their successor $s, s' \in \mathcal{S}$ for a fixed action $a \in \mathcal{A}$.

$$\mathbf{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \middle| S_t = s, A_t = a\right] \tag{2.12}$$

where the sum of each row of the matrix equals 1.

2.2.2 Discounts in Markov Decision Processes

The task that an RL agent needs to learn can either be to maximize its performance over a fixed period or an indefinite amount of time steps, where a time limit is enforced in order to diversify its experience [55]. The goal of the agent is formalized through the reward signal from the environment. The optimal control sequence corresponds to maximizing the the cumulative reward in the long run called the return at time t, G_t instead of the immediate reward. It is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \tag{2.13}$$

where γ is the so-called discount factor, $0 \leq \gamma \leq 1$. The discount factor enforces an exponential decay of contributions of future rewards to the return and ensures in this way that it remains bounded [1]. This definition is also valid for fixed time episodes, where the discounted rewards are summed over a finite number of episodes commonly noted as T. The discount rate determines the present value of future rewards, as a reward obtained for example k steps into the future will only be worth γ^{k-1} times what it would be worth if gained immediately. If $\gamma = 0$ the agent is considered "myopic" and only considers immediate rewards. If $\gamma \to 1$ the agent becomes more farsighted.

With the addition of the discount factor, the definition of a finite Markov Decision Processes is complete and can be written as:

Definition 2.2.2 (Finite Markov Decision Process)**.** A Markov Decision Process is a tuple $< \mathcal{S}, \mathcal{A}, \mathbf{P}, r, \gamma >$ where :

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $\mathbf{P}_a$ is a state transition probability matrix as defined in equation (2.12), $\mathbf{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \middle| S_t = s, A_t = a\right]$.
- r is the reward function as defined in equation (2.10), $r(s,a) = \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)$.
- γ is the discount factor $\gamma \in [0,1]$

2.2.3 Value functions, policies and Bellman equation

The sequence of states or state-action pairs which the agent follows, defines the cumulative reward that it will obtain. The value function is a description of the long-term "value" of the present state and as it depends on the sequence to of state-action pairs to become it depends on the agent's policy (i.e. which actions are chosen for a given state).

As defined in subsection 1.3.1, a policy describes the behaviour of an agent. Formally, it is a mapping from states to probabilities of selecting possible actions. For an agent that follows a policy π at time t, $\pi(A_t = a | S_t = s) = \pi(a|s)$ is the probability that an action a is selected if in state s for $a \in \mathcal{A}$ and $s \in \mathcal{S}$. The *state value* and *action-state value* of respectively a state s and a state-action pair (s,a) are the expected *return* when starting in s or (s,a) and following the policy π. It can be

[1] Consider a constant bounded reward $R_t = \alpha \ \ \forall t$, the sum of infinite bounded terms is non zero and constant if $\gamma < 1$:

$$G_t = \sum_{k=1}^{\infty} \alpha \gamma^{k-1} = \frac{\alpha}{1-\gamma}$$

written formally as :

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \middle| S_t = s\right] \qquad \forall s \in \mathcal{S} \tag{2.14}$$

$$q_\pi(s,a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \middle| S_t = s, A_t = a\right] \qquad \forall (s,a) \in (\mathcal{S}, \mathcal{A}) \tag{2.15}$$

where $\mathbb{E}_\pi[\cdot] = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[\cdot]$ is the expected value of a random variable given that the agent follows the policy π. The function $v_\pi(s)$ and $q_\pi(s,a)$ are called respectively the "state-value function" and "action-value function" for the policy π. These value functions can be estimated from past experience through classical Monte Carlo methods. Furthermore, they are widely used in dynamical programming as they satisfy the following recursive equation :

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[G_t | S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \middle| S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \middle| S_t = s\right] \end{aligned} \tag{2.16}$$

This relation is the *Bellman equation* for v_π and is valid for any state s and policy π. It is a consistency condition between the state s and its successor s'. All actions a, states s and s' and rewards are sampled from their respective sets. A similar relation can also be derived for the state-action value function $q_\pi(s,a)$:

$$q_\pi(s,a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a\right] \tag{2.17}$$

It is important to note that the value function $v_\pi(s)$ and state-value function $q_\pi(s,a)$ are unique solutions to their respective Bellman equations. With the transition matrix defined in equation (2.12), the Bellman equation can be linearized into matrix form. So for a finite MDP with n states $s_1, s_2, \ldots, s_n \in \mathcal{S}$ and their respective rewards $r_1, r_2, \ldots, r_n \in \mathcal{R}$ following the policy π, the Bellman equation (2.16) can be rewritten as :

$$\begin{pmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{pmatrix} = \begin{pmatrix} r_1 \\ \vdots \\ r_n \end{pmatrix} + \gamma \begin{pmatrix} \mathbf{P}_{11} & \ldots & \mathbf{P}_{1n} \\ \vdots & & \\ \mathbf{P}_{n1} & \ldots & \mathbf{P}_{nn} \end{pmatrix} \begin{pmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{pmatrix} \tag{2.18}$$

This system of equations has a unique solution $\vec{v}_\pi^* = (\mathbb{1} - \gamma \mathbf{P})^{-1} \vec{r}$ and can be computed in $\mathcal{O}(n^3)$ operations (which makes it impractical for large MDPs). The Bellman equation is central to the agent training procedure. In the following, methods will be presented to approximate its solution.

2.2.4 Optimality in MDPs

Reinforcement learning is an optimization problem with the goal to find the optimal policy, which maximizes the cumulative reward in the long run. Using the functions defined in subsection 2.2.3, the optimal policy can be derived for finite Markov Decision Processes. The value function defines a partial ordering between policies. That is, a policy π is defined as better or equal to a policy π' if its expected return is higher or equal for all possible states. Formally, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_\pi(s') \ \ \forall s \in \mathcal{S}$. Theorem 2 defines the optimal value and action-value functions and asserts the existence of the optimal policy:

Theorem 2 (Optimal policy)**.** *For any Markov Decision Process [71]:*

- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi$, $\forall \pi$*
- *All optimal policies achieve the optimal value function : $v_{\pi_*}(s) = v_*(s) = \max_\pi v_\pi(s)$, $\forall s \in \mathcal{S}$*
- *All optimal policies achieve the optimal action-value function : $q_{\pi_*}(s,a) = q_*(s,a) = \max_\pi q_\pi(s,a)$, $\forall s \in \mathcal{S}$, $a \in \mathcal{A}$*

Theorem 2 defines the properties of the optimal state-value function and the optimal action-value function. The optimal state-value function v_* also satisfies the self-consistency condition given in equation (2.16). (Note it is not needed to reference the policy, as the optimal one is considered in case of the optimal state-value function.) This yields the **Bellman optimality equation**, which is defined for both state and state-action value functions in equation (2.19).

$$v_*(s) = \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \middle| S_t = s, A_t = a\right] \tag{2.19}$$

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a q_*(S_{t+1}, a) \middle| S_t = s, A_t = a\right] \tag{2.20}$$

where $\mathbb{E}[\cdot] = \sum_{s',r} p(s', r|s, a)[\cdot]$, the expectation value over possible states and rewards. The Bellman optimality equation simply states that the value of a state under an optimal policy is equal to the expected return for the best action starting from that state, i.e $v_*(s) = \max_{a \in \mathcal{A}} q_*(s,a)$. As this equation (2.19) has a unique solution, it allows to solve the reinforcement learning problem and thus obtain the optimal policy. Nevertheless, the solution is not usually easy to find as similarly to (2.18) it is a system of equations of the size of the number of states with the same amount of unknowns, but non-linear. Therefore explicitly solving the Bellman optimality equation is rarely directly useful. In the following methods will be described that approach the optimal solution for reasonable compute power (for reference, the game of Backgammon has already 10^{20} states).

2.3 Model-free Reinforcement Learning

As mentioned in the introduction, RL methods can be divided into model-based and model-free algorithms. In both cases, the model is not available upfront. Model-based methods explicitly learn

a dynamics model and have many advantages. Still, in the simplest form of RL, it is implemented model-free and also for the test case of AWAKE electron line trajectory steering with hierarchical RL, model-free agents were be deployed. Hence a summary of how to find the Bellman optimal state or state-action value functions described in equation (2.19) with model-free RL algorithms will be given. (Note methods based on Monte-Carlo sampling will not be introduced here. Details on these methods can be found in [71])

The focus of this section will be on the estimation of the value function of an unknown Markov Decision Process with so-called *Temporal difference learning*. How to apply it for control π_* will be introduced in subsection 2.3.2.

2.3.1 Model-free prediction

Temporal difference (TD) learning is a combination of Monte Carlo and dynamic programming ideas to estimate the value function of an unknown Markov Decision process. Formally, it is equivalent to the case where the transition probabilities $p(s', r|s, a)$ are unknown. It is similar to Monte Carlo methods as TD methods can learn directly from raw experience without a model of the environment's dynamics. On the other hand, it is similar to dynamic programming as it does not need to wait for a final outcome to update its estimates. Given some experience (i.e. tuples of (s, a, s', r)) following a policy π, temporal difference methods update their estimate V of v_π for the non-terminal state S_t occurring in that experience as follows:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right] \tag{2.21}$$

with R_{t+1} being the reward obtained at the state S_{t+1} and $\alpha \in (0, 1]$ the learning rate. Note that with TD methods the approximation of the value function can be updated at every step. At time $t+1$ the $R_{t+1} + \gamma V(S_{t+1})$ is used as "target" for the updated rule with R_{t+1} as the observed reward and the estimate $V(S_{t+1})$. The update rule in equation (2.21) is called "TD(0)". As TD(0) bases its update in part on an existing estimate, it is also called a "bootstrapping" method. Algorithmically, for a tabular situation method, $V(s) \forall s \in \mathcal{S}$ is initialized and iteratively updated using experienced rewards. The quantity in the brackets of the TD update rule in (2.21) can also be seen as an "error", measuring the difference between the estimated value at S_t and the better estimate $R_{t+1} + \gamma V(S_{t+1})$. This "error" is called the TD error in literature :

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.22}$$

For any fixed policy π, it has been shown that TD(0) converges towards v_π provided that the constant step size α is small enough[2]. This update rule can be used to learn the state value function. For optimal control, it is however more useful to learn the state-action value function instead of the state

[2]Stochastic approximation theory yields the following conditions to assure convergence with probability 1 :

$$\sum_{t=1}^{\infty} \alpha_t(a) = \infty \text{ and } \sum_{t=1}^{\infty} \alpha_t^2(a) < \infty$$

value function. This allows to not only know the value of visited states, but also those associated with the chosen actions in a given state in order to yield the highest return.

2.3.2 Model-free control

Previously it has been shown how to estimate value functions V in a deterministic procedure with a complexity in $\mathcal{O}(n)$ with n being the number of states. Nevertheless, for model-free control it is particularly useful to estimate the values of state-action pairs, or commonly called action values. With a model, state values alone are enough to determine a policy as one can simply look ahead one step and choose whichever action leads to the best combination of reward and next state. Without a model however state values alone are not sufficient. One must also explicitly estimate the value of each action in order for the values to be useful in suggesting a policy. Thus our goal in model-free control is to consider methods that estimate q_* (see equation (2.20)). There are two approaches to estimate q_* in model-free reinforcement learning, on-policy and off-policy. Each of the approaches has their advantages and disadvantages.

On and off-policy learning

Model-free algorithms have no guidance on which actions to choose, instead these algorithms need to explore. A trade-off between exploration and exploitation must be struck and on and off-policy methods solve this dilemma differently. The former one improves the policy directly during training (i.e. uses the action obtained with the current state-value function for the function update and continues also with that action), whereas off-policy methods improve a different policy than used to generate the data (i.e. in the state-value function update a different action is used). The simplest and most common methods for both of these approaches will be presented in the following.

On-policy TD learning : SARSA

SARSA is an on-policy RL algorithm that approximates the action-state value function Q. As it is an on-policy method, $q_\pi(s,a)$ is estimated and iteratively updated for the current behaviour policy π for all actions a and states s. Instead of state transitions, state-action pair to state-action pair are considered. TD learning is transferable from state to state-action pairs as both deal with Markov chains with a reward process, using the the state-action value function leads to the same convergence properties as TD(0). Equation 2.23 shows the update rule for the iterative update in the case of on-policy SARSA:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)\right] \tag{2.23}$$

The update occurs after every transition from a non-terminal state S_t (if S_{t+1} is terminal, then $Q(S_{t+1},A_{t+1}) = 0$). The convergence properties of the SARSA algorithm rely on the nature of the policy's dependence on Q. ϵ-greedy or ϵ-soft policies [18] [67] are used during training to enhance exploration. SARSA converges with probability 1 to an optimal policy and state-action value function if all state-action pairs are visited an infinite number of times. In practice, for simple cases a few

iterations on every state-action pair are sufficient to yield a good approximation of q_π. For more complex problems this poses a considerable challenge as the set of all possible state-action pairs can be large. Note that SARSA, as most on-policy methods, does not necessarily converge to the optimal state-action value function q_* as it can be trapped in local minima. An off-policy counterpart to SARSA will be presented next.

Off-policy TD : Q-learning

The state-value function is also often referred to as Q-function - and hence the name of the off-policy TD algorithm Q-learning. The update rule in the case of Q-learning looks like:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{2.24}$$

The learned state-action value function Q directly approximates the optimal one q_* independently of the policy being followed. The behavioural policy that is used to explore however still has an effect in that it determines which state-action pairs are visited and updated. Nevertheless, all that is required for correct convergence is a sufficient exploration of the state-action space. Off-policy methods can be "misguided" specifically if they favour exploitation (on-policy methods are more conservative and hence potentially safer). This bias in exploration can be minimized by learning two Q-functions at the same time with random updates. A method adopting this approach will be discussed in section 2.4. Q-learning is however a simple yet powerful method to learn an optimal action-value function q_* in a sample efficient manner.

The data driven learning methods for the controllers for AWAKE will be based on Q-learning as Q-learning manages to learn with a relatively small amount of policy updates. After a short excursion to Deep Learning, Q-learning will be adapted for large states, state-action spaces and specifically for continuous actions. The section section 2.4 will merge the off-policy model-free control methods shown here with Feedforward Neural Networks used for function approximation in order to derive a control method that can be used for trajectory control at the AWAKE electron line.

2.4 Deep Reinforcement Learning

In this section the tabular methods for model-free off-policy reinforcement learning introduced earlier will be extended to problems with large action and state spaces. Even with the computational resources available nowadays, it is not possible to find an optimal policy or value function for most real-world problems with simple tabular methods as discussed so far. The memory needed store the action-state pairs and the computing time in order to fill the tables would be prohibitively large. It is inevitable to come up with algorithms that are capable to output sensible actions for states that have not been necessarily seen before. This generalization is achieved by non-linear function approximation methods such as Artificial Neural Networks. Supervised learning methods from Deep Learning will be combined for example with the update rules from above. The first part of this section will therefore introduce the subject of "Deep Learning".

After the introduction to "Deep Learning", some of the modern Deep Reinforcement Learning algorithms will be presented with the focus on the methods applied to the AWAKE steering problem.

2.4.1 Deep learning

Deep Learning (DL) is a branch of machine learning aiming to learn abstract representations of the input space through complex function approximators. This class of methods rely on the construction of multiple hierarchical layers of representations. Each layer consists of simple non linear modules that progressively transform the initial input into increasingly abstract representations at higher levels. In this section a brief overview of deep learning methods will be given with an emphasis on function approximation, for an extended overview the reader is referred to [19].

Artificial Neural Networks

The earliest learning algorithms recognized today were intended to be computational models of biological learning, creating a model of the brain. The model of a single neuron, named a "perceptron", is the building block of artificial neural networks. A neuron is supposed to be able to extract critical information from an input signal and produce a meaningful output response. To simulate this behaviour, the output response of the ith perceptron is given by equation (2.25). It takes an input vector $\vec{x}$ of n values and outputs a scalar $y \in \mathbb{R}$.

$$y_i = f\left(\sum_j \mathbf{w}_{ij} x_j + b_i\right) \tag{2.25}$$

f is a non-linear function - historically a step function $f(x) = 1$, if $x > 0$ and 0 if not, and was used for classification. Nowadays, there is a wide variety of used non-linear functions, as for example the linear rectifier function $f(x) = \max(0, x)$ or the hyperbolic tangent. The intrinsic parameters of the neuron $\vec{w}_i \in \mathbb{R}^d$ and $b_j \in \mathbb{R}$ are called respectively the weights and the bias. They are the parameters that are learnt in order to maximize the log likelihood of the training data distribution given the input and function approximator.

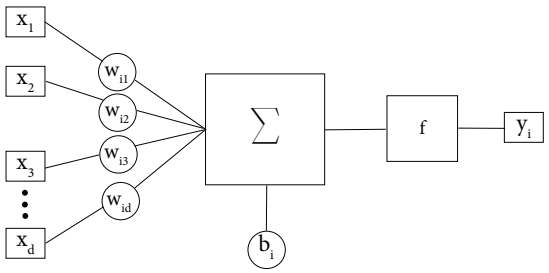

Figure 2.1: Schematic of the perceptron.

Similarly to the brain, an Artificial Neural Network (ANN) consists of a set of interconnected perceptrons arranged in layers. Neurons in the same layer are not connected between themselves

but are connected to all the outputs of the neurons of the previous layer. In the next part, the most common neural network used for function approximation will be presented.

(Vanilla) Feedforward Networks

Deep feedforward networks also called multilayer perceptrons (MLPs), are the quintessential deep learning models. The goal of a feedforward network is to approximate some function f^*. It defines a mapping $\vec{y} = f(\vec{x}, \vec{\theta})$ and learns the value of the parameters $\vec{\theta} = (\{w\}, \{b\})$ that result in the best function approximation. The term "feedforward" comes from how the information flows in the network. It flows from $\vec{x}$ through the intermediate layers called "hidden layers" and finally to the output $\vec{y}$. It does not possess any feedback connections in which outputs of the model are fed back to the input layer.

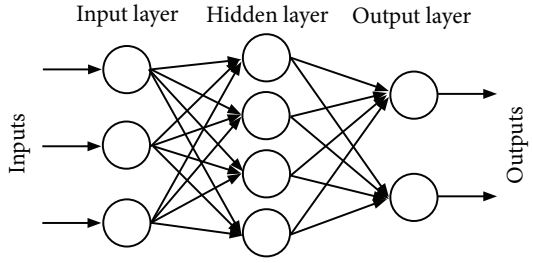

Figure 2.2: Schematic of a feedforward neural network with one hidden layer.

The general supervised learning procedure will be described in the next part of this section. It is not specific to feedforward neural networks. Any model that needs to adjust its internal parameters to match a known and desired output can use the following learning procedure.

Gradient based learning

When the desired output of the neural network is known for a finite number of samples, it is possible to use so-called supervised learning for training. In most cases the training of a feedforward neural network (FNN) is done in this fashion. The output function of the neural network $f(\vec{x}, \vec{\theta})$, where θ are the function approximator parameters, is compared to the desired observation $\vec{y} = f^*(x)$ through a loss function $l(\vec{x}, \vec{\theta})$ that needs to be minimized. For example, maximizing the log likelihood of the labels $\vec{y}$ via the parameters of the FNN and the input $\vec{x}$ and assuming a normal distribution with standard deviation $\boldsymbol{\sigma}$ of the observation $p(\vec{y}|\vec{x}) = \mathcal{N}(\vec{y}|f(\vec{x}, \vec{\theta}), \boldsymbol{\sigma})$ is equivalent to the minimization of the mean squared error cost :

$$l(\{\vec{x}\}, \vec{\theta}) = \frac{1}{m} \sum_{i=1}^{m} ||\vec{y}^{(i)} - f(\vec{x}^{(i)}, \vec{\theta})||^2 \tag{2.26}$$

$$l(\{\vec{x}\},\vec{\theta}) = -\log \prod_{i=1}^{m} p(\vec{y}^{(i)}|\vec{x}^{(i)}) \tag{2.27}$$

$$= \sum_{i=1}^{m} \log p(\vec{y}^{(i)}|\vec{x}^{(i)}) \tag{2.28}$$

$$\propto ||\vec{y}^{(i)} - f(\vec{x}^{(i)},\vec{\theta})||^2 \tag{2.29}$$

as $\mathcal{N}(\vec{x}|\vec{\mu},\boldsymbol{\sigma}) = \frac{1}{|\boldsymbol{\sigma}|\sqrt{2\pi}} e^{\left(\frac{\vec{x}-\vec{\mu}}{|\boldsymbol{\sigma}|\sqrt{2}}\right)^2}$. Furthermore, m is the number of known samples where the ground truth function f^* is evaluated. Note that by assuming other prior distributions on the observations other loss functions are obtained. This minimization of the loss function or maximization of the log likelihood is done by updating the parameters $\vec{\theta}$ of the neural network. And the update of the parameters can be done in a plethora of different ways. The most popular is stochastic gradient descent where the parameters are recursively updated as :

$$\vec{\theta} \leftarrow \vec{\theta} - \epsilon\vec{g}$$

with $\epsilon \in \mathbb{R}$ being the learning rate and $\vec{g} = \frac{1}{m'}\nabla_\theta \sum_{i=1}^{m'} ||\vec{y}^{(i)} - f(\vec{x}^{(i)},\vec{\theta})||^2$, where $m' < m$ a smaller subset of the known samples. The gradient of the cost function for each sample $\nabla_\theta ||\vec{y}^{(i)} - f(\vec{x}^{(i)},\vec{\theta})||^2$ can be computed with the chain rule and software frameworks are available that offer efficient implementations of this so-called back-propagation.

Universal approximation theorem

Formally, it is possible to describe the class of functions representable by a one-hidden layer neural network from $\mathbb{R}^d$ to $\mathbb{R}$ as :

$$\mathcal{F}_{w_1,w_2,b}(x) := \left\{ f_{w_1,w_2,b} = w_1^T \sigma(w_2 x + b) \text{ for } w_2 \in \mathbb{R}^{D\times d}, b \in \mathbb{R}^D, w_1 \in \mathbb{R}^D \right\} \tag{2.30}$$

here $\sigma(\cdot)$ is the activation function, i.e. the non-linear function of each perceptron of the same layer. The parameters d and D are respectively the dimensions of the input and the number of hidden neurons. $x \in \mathbb{R}^d$ is the input, $w_2 \in \mathbb{R}^{D\times d}$ are the weights of the first layer, $w_1 \in \mathbb{R}^D$ are the weights of the hidden layer and $b \in \mathbb{R}^D$ are the biases of the first layer. For this theorem there is no need of biases in the hidden layer nor an activation function. The universal approximation theorem is as follows :

Theorem 3 (Universal Approximation Theorem)**.** *Let $f^* : \mathbb{R} \to \mathbb{R}$ be a continuous bounded target function. It can be approximated by a 1-hidden layer feed forward fully connected neural network $f_{w_1,w_2,b} \in \mathcal{F}_{w_1,w_2,b}$ with an activation function σ that is not a polynomial function, such that $\forall \epsilon > 0$ [27]:*

$$\sup_{X\in\mathbb{R}^d} ||f^*(X) - f_{w_1,w_2,b}(x)|| < \epsilon$$

With $\mathcal{F}_{w_1,w_2,b}$ as described in (2.30)*.*

The original proof of this theorem was done for a sigmoid non-linear function and can be found in [12]. It has then been adapted to any non-polynomial function in [59]. This theorem only states the existence of a good network that allows to approximate any continuous function f^* as precisely as needed, but it does not mean that it can be found effectively. The optimization problem (i.e. the loss function) for neural networks is usually non-convex, thus it is not guaranteed to always find the optimal function approximation through stochastic gradient descent. Moreover, depending on the function needed to approximate, the number of hidden neurons D can be exponentially large (it usually scales exponentially with respect to the input dimensions d).

2.4.2 Value function approximation

On-policy value function approximation

In the case of on-policy methods v_π or q_π are approximated via experience generated following a policy π. Instead of using a table where the value function is stored, a parameterized functional form with a weight vector $\vec{w} \in \mathbb{R}^d$ is used as approximator. In the most common implementation, feedforward neural network as described earlier play the role of these function approximators. The parameterized value and state-value functions are hence written as $\hat{v}(s, \vec{w}) \approx v_\pi$ or $\hat{q}(s, a, \vec{w}) \approx q_\pi$ with weight vector $\vec{w}$. In the supervised learning framework the best $\vec{w}$ is found reducing the Mean Squared Value Error noted $\overline{VE}$ of either the state or action-state value function with respect to their true values :

$$\overline{VE}_v(\vec{w}) = \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \vec{w}) \right]^2 \tag{2.31}$$

$$\overline{VE}_q(\vec{w}) = \sum_{s \in \mathcal{S}} \mu(s) \left[q_\pi(s, a) - \hat{q}(s, a, \vec{w}) \right]^2 \tag{2.32}$$

$$\tag{2.33}$$

where $\mu(s) \geq 0$, $\sum_{s \in \mathcal{S}} \mu(s) = 1$ is the state distribution representing which states are visited more frequently. The global optimum is reached for a weight vector $\vec{w}^*$ where $\overline{VE}(\vec{w}^*) \leq \overline{VE}(\vec{w})$, $\forall \vec{w}$. Stochastic Gradient Descent (SGD) can be used to find the global optimum in an iterative manner. The weight vector is adjusted after each observation by a small amount in the direction that reduces the mean squared error $\overline{VE}$ according to:

$$\vec{w}_{t+1} = \vec{w}_t - \frac{1}{2} \alpha \nabla_{\vec{w}} \left[v_\pi(S_t) - \hat{v}_\pi(S_t, \vec{w}) \right]^2 \tag{2.34}$$

$$\vec{w}_{t+1} = \vec{w}_t - \frac{1}{2} \alpha \nabla_{\vec{w}} \left[q_\pi(S_t, A_t) - \hat{q}_\pi(S_t, A_t \vec{w}) \right]^2 \tag{2.35}$$

In practice, the actual value functions v_π and q_π are not available. They are therefore replaced by target outputs $U_t \in \mathbb{R}$ following very much the arguments before with the TD learning.(For example for an equivalent of a TD(0) of state value function : $U_t \leftarrow R_{t+1} + \gamma \hat{v}(S_{t+1}, \vec{w})$). Note that

this procedure does not necessarily converge to $\vec{w}^*$ as it can get trapped in local minima. This is a frequent issue with on-policy methods.

Off-policy value function approximation

Applying function approximation to off-policy reinforcement learning is significantly more complicated than it is for on-policy. If the tabular value functions are simply replaced by neural networks, the resulting RL algorithms do not converge as robustly as if trained on-policy. The reason for this issue comes from the fact that the algorithm seeks to learn the value function for an unkown target policy π given data due to a different actual policy, often called the behaviour policy b and hence subsequent actions.This needs to be taken into account for Stochastic Gradient Descent when updating the weights for the neural networks approximating the value function $\hat{v}_\pi$ or $\hat{q}_\pi$ by introducing *importance sampling*.

$$\vec{w}_{t+1} = \vec{w}_t - \frac{1}{2}\alpha\rho_t\nabla_{\vec{w}}\left[U_t^v - \hat{v}_\pi(S_t,\vec{w})\right]^2 \tag{2.36}$$

$$\vec{w}_{t+1} = \vec{w}_t - \frac{1}{2}\alpha\nabla_{\vec{w}}\left[U_t^q - \hat{q}_\pi(S_t,A_t\vec{w})\right]^2 \tag{2.37}$$

where ρ_t is the importance sampling ratio. The update targets for the state and state-action value functions U_t^v and U_q^t are:

$$U_t^v = R_{t+1} + \gamma\hat{v}(S_{t+1},\vec{w}) \tag{2.38}$$

$$U_t^q = R_{t+1} + \gamma\sum_a \pi(a|S_{t+1})\hat{q}(S_{t+1},a,\vec{w}_t) \tag{2.39}$$

Importance sampling is a general technique for estimating expected values under one distribution given samples from another:

$$\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \tag{2.40}$$

As can be seen in equation 2.37, importance sampling is only required if learning the state value function directly. For Q-learning where the state-action value function is approximated, it is not needed. Still the fact that the behaviour policy does not equal the target policy makes off-policy learning prone to divergence.

Sample-efficiency and promises with Off-policy learning

If the training is typically more unstable with off-policy methods when using function approximation, why are many of the most performing RL algorithms relying on off-policy algorithms?

Off-policy methods free behaviour from the target policy and this allows to learn from or "replay" past experience generated by humans, other optimisation or control algorithms or other historical data. It also allows parallelization when it comes to training. This so-called "Experience replay"

is what makes off-policy learning truly sample-efficient and so appealing. Some of the remaining issues with off-policy Q-learning in DRL can be overcome by clever tricks. And some of these tricks will be discussed next.

Double Q-learning with Experience Replay

The deep Q-network (DQN) agent proposed by Mnih. et al. [48] uses deep neural networks to approximate state action values that are then updated off-policy by Q-learning. It uses Experience Replay [43] to sample transitions with a mixture of past policies. However, the algorithm sufferd from overestimating the action value function due to the maximisation steps in the updates (see equation (2.24)). This issue was mitigated by Silver et al. by introducing *Double Q-learning* [24].

In the Double DQN (DDQN) algorithm two value functions are learned by assigning each experience to randomly update one of the two of them, parameterized with weights $\vec{w}$ and $\vec{w}'$. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. The targets in the DDQN algorithm are written as :

$$U_t^{Dq} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a, \vec{w}), \vec{w}') \tag{2.41}$$

Greedy policy is used with the weights $\vec{w}$. The second set of weights $\vec{w}'$ is used to evaluate the value of the policy. The roles of the two value functions (i.e. networks) are switched periodically. DDQN uses experience replay like the original DQN algorithm and stores the experience as (S_t, A_t, R_{t+1}, S_{t+1}) in replay memory. For weight updates a random mini-batch of transitions is sampled from the replay buffer to calculate the gradients.

This algorithm improved the state-of-the-art performance on Atari games when introduced and has since been applied to a wide range of control problems. Some of the ideas in DDQN were also used for the algorithm to solve the AWAKE trajectory steering problem. Still DDQN or Q-learning in its simplest is restricted to discrete action space. The methods described next will allow also for continuous action space.

2.4.3 Policy Gradient Methods

Instead of defining the policy through maximising a value function, the policy can be learned directly. The following notation for the parametric policy will be used: $\pi(a|s,\vec{\theta}) = \mathbb{P}\left\{A_t = a \middle| S_t = s, \vec{\theta}_t = \vec{\theta}\right\}$, which corresponds to the probability of selecting an action a at time t given that the environment is in state s. $\vec{\theta} \in \mathbb{R}^{d'}$ is the policy's parameter vector. Policy gradient methods still need the value function as performance measure in the parameter or weight update rule:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \nabla J(\vec{\theta}_t) \tag{2.42}$$

where $J(\theta_t) = \mathbb{E}v_{\pi_{\theta_t}}(s)$. The gradient $\nabla J(\theta_t)$ can be estimated with the *Policy Gradient Theorem*:

Theorem 4 (Policy Gradient Theorem)**.** *Let $\pi(a|s,\vec{\theta})$ be a differentiable policy, then for any policy performance measure J, the policy gradient is [71]:*

$$\nabla_{\vec{\theta}} J(\theta) = \mathbb{E}_{\pi_{\vec{\theta}}} \left[\nabla_{\vec{\theta}} \log \pi(a|s,\vec{\theta}) q^{\pi_{\vec{\theta}}}(s,a) \right]$$

Actor-Critic algorithms

Actor-Critic algorithms merge function approximation for value functions as in Q-learning with policy gradient methods to approximate the policy. This class of algorithms was originally presented by Konda and Tsitsiklis in 1999 [34]. The parameterized policy is called "actor" and the value function approximation "critic". The actor $\pi(a|s,\vec{\theta})$ learns its parameters $\vec{\theta}$ by stochastic gradient ascent as shown in (2.42). Following Theorem 4, .instead of using the true value function $q^{\pi_{\vec{\theta}}}(s,a)$, the value function $\hat{q}^{\pi_{\vec{\theta}}}(s,a,\vec{w})$ approximated by the critic is used, see Fig. 2.3. Thus the policy gradient becomes:

$$\nabla_{\vec{\theta}} J(\theta) = \mathbb{E}_{\pi_{\vec{\theta}}} \left[\nabla_{\vec{\theta}} \log \pi(a|s,\vec{\theta}) \hat{q}^{\pi_{\vec{\theta}}}(s,a,\vec{w}) \right] \tag{2.43}$$

The critic estimates the state-action value function and uses standard TD learning methods to update its parameters.

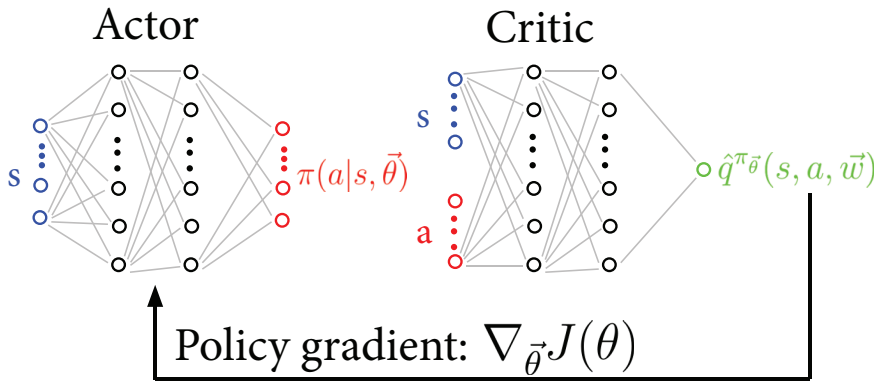

Figure 2.3: Actor-Critic structure.

As the Actor-Critic algorithm does not require the maximization step in the update rules or for selection of the next action, it is also suitable for continuous action space. The TD method for updating the state-value function is usually Q-learning and hence the training off-policy with the additional advantage of experience replay, see [41, 74] for some of the famous Actor-Critic algorithms. (Note depending on how the value function targets are defined it is also possible to train on-policy.)

2.4.4 The Twin Delayed Deep Deterministic policy gradient algorithm (TD3)

The *Twin Delayed Deep Deterministic Policy Gradient Algorithm (TD3)* is an actor-critic method that uses clipped double Q-learning to deal with the overestimation bias when approximating the Q-function. It was developed by Fujimoto et al [17]. It also mitigates function approximation errors by delaying policy updates to reduce the error generated by each update and improve the overall performance.

Clipped Double Q-learning is similar to the standard double Q-learning presented in subsection 2.4.2, where two approximations of the action value function are alternatively learnt with two sets of parameters w and w'. However, instead of using one or the other q-function to choose an optimal action, the two functions are compared to chose the smaller of the two for the update. In clipped double Q-learning the target for the weight update becomes:

$$U_t^{CDq} = R_{t+1} + \gamma \min_{\phi=\tilde{w},\tilde{w'}} q(S_{t+1}, \pi_{\tilde{\theta}}(S_{t+1}), \phi) \tag{2.44}$$

where $\tilde{\theta}$, $\tilde{w}$ and $\tilde{w'}$ are the target parameters that differ from their counterparts θ, w and w' - the parameters of the estimation of the policy and value functions. While the issue of overestimation is removed, underestimation bias might occur. This is however preferable to overestimation as it does not propagate through the policy update to subsequent targets. Another issue that vanilla Actor-Critic methods encounter, that TD3 attempts to mitigate, is the high variance in its estimates. High variance in estimates produce noisy gradients for policy updates which reduces learning speed. The variance is inherent to the Temporal Difference learning and is due to the fact that in a function approximation setting, the Bellman equation is not exactly satisfied and has a residual TD error in every target (see (2.22)). In [17] it was shown that the variance can be reduced by delaying the update through a "slow update" of the target network as :

$$\tilde{w} \leftarrow \tau w + (1-\tau)\tilde{w} \tag{2.45}$$

where $\tilde{w}$ are the new parameters of the target, w the parameters of the current network and $\tau \in [0,1]$. Note that this update rule is used for all three sets of parameters, approximating the target actor network and the two target critic networks. By delaying the policy updates, the likelihood of repeating updates with respect to an unchanged critic is reduced. The resulting algorithm is given in **Algorithm 1**.

Algorithm 1 TD3 algorithm

Initialize actor network π_θ and critic networks q_w and $q_{w'}$ with random parameters θ, w and w'
Initialize target networks with current parameters of actor and critic networks $\tilde{\theta} \leftarrow \theta$, $\tilde{w} \leftarrow w$ and $\tilde{w'} \leftarrow w'$
Initialize replay buffer $\mathcal{B}$
for t=1 **to** T **do**
  Select action with exploration noise $a \sim \pi_\theta(s) + \epsilon$,
$\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward R and new state s'
  Store transition tuple (s, a, R, s') in $\mathcal{B}$
  Sample mini-batch of N transitions (s, a, R, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\tilde{\theta}}(s') + \epsilon, \quad \text{clip}(\mathcal{N}(0,\sigma), -c, c)$
  $U^{CDq} \leftarrow R + \gamma \min_{\phi=\tilde{w},\tilde{w'}} q(s', \tilde{a}, \phi)$
  Update critics : $\phi \leftarrow \underset{\phi}{\text{argmin}} \frac{1}{N} \sum (U^{CDq} - q(s', \tilde{a}, \phi))^2, \quad \text{for } \phi = \tilde{w}, \tilde{w'}$
  if t mod $d = 0$ **then**
    Update θ by deterministic policy gradient : $\nabla_\theta J(\theta) = \frac{1}{N} \sum \nabla_a q(s, a; w)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$
    Update target networks :
    $\tilde{\theta} \leftarrow \tau\theta + (1-\tau)\tilde{\theta}$
    $\tilde{w} \leftarrow \tau w + (1-\tau)\tilde{w}$
    $\tilde{w'} \leftarrow \tau w' + (1-\tau)\tilde{w'}$
  end if
end for

In order to mitigate the deterministic policy to overfit narrow peaks in value estimate, clipped noise $\epsilon = \text{clip}(\mathcal{N}(0,\sigma), -c, c)$ is added to the action as $\tilde{a} \leftarrow a + \epsilon$. Where c is a symmetric action bound. This noise smooths the target policy and promotes exploration. Furthermore, it can also improve the resulting controller in stochastic domains.

The TD3 algorithm was used as the base algorithm to implement Hierarchical Reinforcement Learning for AWAKE trajectory steering.

2.5 Hierarchical Reinforcement Learning

In this final section, the hierarchical reinforcement learning (HRL) will be briefly summarised. More on the actual implementation will come in later chapters.

As mentioned already in the introduction chapter, the core idea of HRL is to subdivide a long horizon reinforcement task into a hierarchy of subproblems and the result can be seen as learning or approximating a hierarchy of policies. The long-horizon problem is divided into shorter tasks to be solved by the lower policies, referred to as temporal abstraction. It was shown that this allows to solve complex tasks that cannot be solved by simpler algorithms [28][4]. Several studies explained that the improved performance comes from an improved exploration strategy induced by the higher-level policy [52] [30].

2.5.1 Formalism

The hierarchical structure can be implemented in several manners, but all of them contain one agent per policy that learns an approximation of its optimal policy. Each agent attempts to maximize the cumulative reward over its trajectories in state and action space. The expected length of these trajectories is what is called commonly the "time-horizon". If the various trajectory spaces at different levels are large with a long time horizons, then exploration can be difficult for methods using the standard reinforcement learning approach with a single policy. A practical example of how a long-term horizon task can be decomposed into simpler tasks will be given with the description of tasks for baking a cake. The long-term horizon task would be "Bake a cake". This task can be broken down into "Buy ingredients", "Bring them to the kitchen", "measure quantity needed" and so on. The level of hierarchical decomposition is arbitrary and varies from problem to problem (for example one could go to another abstraction level where moving each limb is a task).

To simplify the notation, only two levels of policy will be considered. (It can however easily be extended to an arbitrarily large number of policies depending on the design of the problem.) A higher-level policy noted hi directs one or more lower-level policies noted lo that directly interact with the environment. In the most common and simplest framework, the higher-level policy chooses a new high level action (i.e. new task) every c steps. The tunable hyperparameter c permits the lower level agents to accomplish the task in this discrete amount of steps. There are two hierarchical reinforcement learning frameworks, the option and the goal-conditioned framework. In the options hierarchy, the high-level action is a discrete choice between lower policies that selects which of the n trained lower-level policies is used for the next c steps. The goal-conditioned framework trains a single goal-conditioned lower policy and the higher level action is a state of the lower-level policy that the latter is incentivised to reach. The AWAKE electron line steering controller was implemented with goal-conditioned hierarchy RL. Thus the rest of this introduction will only consider the goal-conditioned framework.

The policies in HRL have a strong time dependence, formalised with Semi-Markov Decision Processes (SMDP). SMDPs are similar to Markov Decision Processes as discussed in subsection 2.2.1 with the addition of the concept of time over which an action is executed. For an initial state $s_t \in \mathcal{S}$ at time t, the transition probabilities of the SMDP are joint distribution:

$$\mathbb{P}(s_{t+c}, c|s_t, a'_t) = \mathbb{P}(s_{t+c}|s_t, a'_t, c)\mathbb{P}(c|s_t, a'_t) \tag{2.46}$$

where c is the number of timesteps for which the action a'_t is executed. Note that for the AWAKE goal-conditioned case c is fixed so $\mathbb{P}(c|s_t, a'_t) = 1$, but that is not necessarily the case for more complex problems or other implementations of HRL. The higher and lower-level agents for the AWAKE case are Actor-Critic agents, implemented as TD3. The training of both high and lower policies is done by minimizing the TD error(2.22) for both the high and low policy simultaneously. The resulting TD errors become (note that in reality Double Q-learning is used. For practical reasons only simple Q-learning is shown below):

$$\delta_t(s_t, g_t, a_t, R_t, s_{t+1}, g_{t+1}) = \left(Q_{lo}(s_t, g_t, a_t) - R_t - \gamma Q_{lo}(s_{t+1}, g_{t+1}, \pi_{lo}(s_{t+1}, g_{t+1}))\right)^2 \tag{2.47}$$

$$\delta_t(s_t, g_t, R_{t:t+c-1}, s_{t+c}) = \left(Q_{hi}(s_t, g_t) - R_{t:t+c-1} - \gamma Q_{hi}(s_{t+c}, \pi_{hi}(s_{t+c}))\right)^2 \tag{2.48}$$

where $Q_{lo}(s_t, g_t, a_t)$ and $Q_{hi}(s_t, g_t, a_t)$ are respectively the lower and higher-level state-action value functions. Here the goal g_t is the output action of the higher-level policy $g_t = \pi_{hi}(s_t)$ and is part of the state space $g_t \in \mathcal{S}$ of the lower-level agent. Furthermore, $R_{t:t+c-1} = \sum_{i=0}^{c-1} R_{t+i}$ is the sum of the environment rewards obtained by the lower-level policy over c steps.

Chapter 3

Formulating Hierarchical Reinforcement Learning for AWAKE trajectory steering

In this chapter a description of the real-world problem of automatic trajectory correction at the AWAKE electron line with a data-driven hierarchical controller will be given with the focus on involved equipment and layout. Specificities of how to formulate the AWAKE steering control problem to fit into the Hierarchical Reinforcement Learning framework will also be discussed. The obtainable performance with such a controller will follow in the next chapter.

3.1 Trajectory correction at the AWAKE electron line

AWAKE is a facility at CERN for proton-driven plasma wakefield acceleration experiments [22] [50] [61]. It receives protons at 400~GeV from the CERN SPS. The AWAKE electrons are generated in a photocathode RF gun that produces bunches of approximately $1.2 \cdot 10^9$ electrons at 5 MeV/c, which are then accelerated up to 10-20 MeV/c by a linac booster and transported via a 12 m transfer line to the accelerating plasma cell [66], where their trajectory joins the proton trajectory. The AWAKE electron line, when run in standalone, offers an ideal set-up for testing advanced control algorithms due to the high repetition rate of 10 Hz and the low damage potential in case of beam loss with the low energy and intensity electron beams. HRL was introduced for electron trajectory control. An overview of the electron line and its equipment is given in Fig. 3.1.

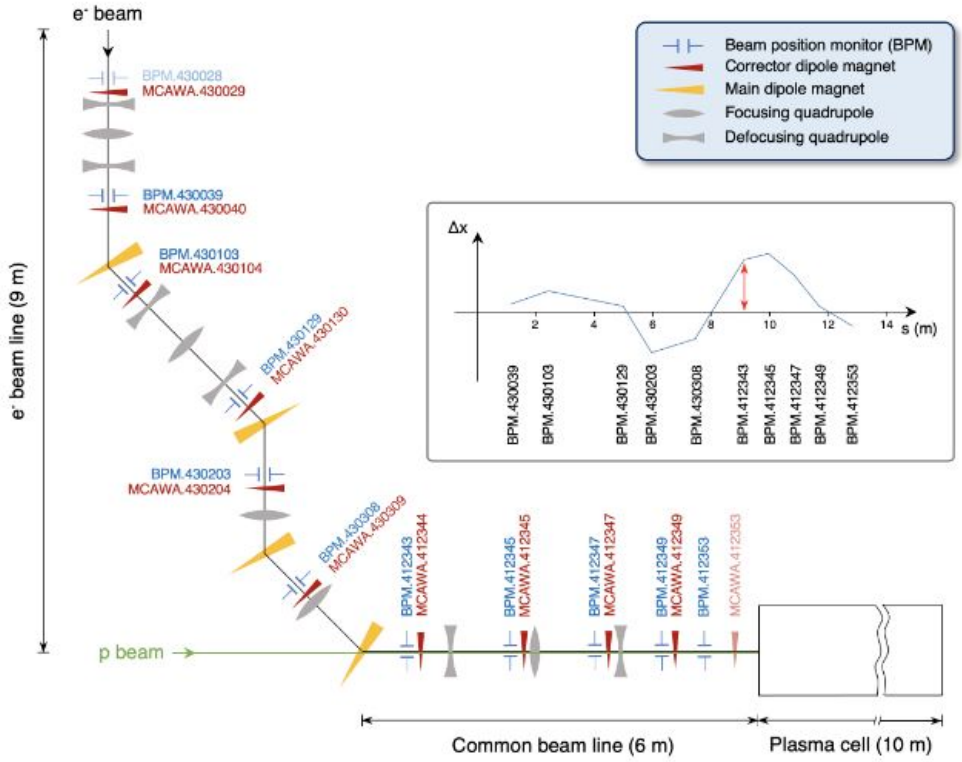

Figure 3.1: Layout of the electron line of the AWAKE facility. It contains 11 Beam Position Monitors and 11 trajectory correctors. An example trajectory measured at the BPMs (as relative position to the design trajectory) is shown as well [65].

The horizontal and vertical electron beam position is measured by 11 Beam Position Monitors (BPMs) along the electron line. These monitors are able to measure the position of a single electron bunch with a charge of 0.1 to 1 nC and a resolution of less than 10 μm rms [45]. To compensate for static errors such as field or alignment errors of the magnets along the line, trajectory correctors are placed after every BPM. The corrector magnets can be used up to a strength of $4.34 \cdot 10^{-4}$ Tm corresponding to an angle maximum of 924.6 mrad per magnet [66]. The HRL controller was limited to use maximum ± 300 μrad per time step.

3.1.1 Description of the AWAKE environment

To fit with the most common RL frameworks, the control problem needs to be formulated as so-called environment using e.g. the framework OpenAi Gym [6]. This framework defines an interface between RL agents and RL problems. Implementing the methods of a Gym environment allows full separation of problem specific information (e.g. state and reward definition and how to apply the action) and algorithm logic. The main method in Gym that the defines the interaction between agent and problem is the method `step(a)`, which returns the next state and reward by applying the

action a. For test purposes and specifically tuning of algorithms, a simulated environment for the AWAKE trajectory steering problem is available. The simulated environment uses response matrices calculated with MAD-X [57] based on a user-selected optics. Previous studies have demonstrated that the agents trained in simulation with this particular environment transfer well to the real hardware [31], testifying to the excellent status of the AWAKE model.

Adequate state or "observable" information needs to be provided. The state needs to have the Markov property, i.e. be memoryless, as discussed inchapter 2 and section subsection 2.2.1. The obvious multi-dimensional state for the task of trajectory steering is the horizontal or vertical position measured at each BPM noted respectively $\{x_t^{(i)}\}$ or $\{y_t^{(i)}\}$, where (i) indicates the BPM index. The goal of the control task is to steer the electrons such that their position at the various BPMs matches a reference trajectory given by a user and which is written in a similar fashion $\{x_{\text{ref},t}^{(i)}\}$ and $\{y_{\text{ref},t}^{(i)}\}$. (Note the position and reference positions here are relative positions with respect to a design trajectory in the horizontal and vertical plane) The multi-dimensional states are written as vectors:

$$s_t^h = [x_t^{(0)}, x_t^{(1)}, \ldots, x_t^{(9)}, x_{\text{ref},t}^{(0)}, x_{\text{ref},t}^{(1)}, \ldots, x_{\text{ref},t}^{(9)}] \tag{3.1}$$

$$s_t^v = [y_t^{(0)}, y_t^{(1)}, \ldots, y_t^{(9)}, y_{\text{ref},t}^{(0)}, y_{\text{ref},t}^{(1)}, \ldots, y_{\text{ref},t}^{(9)}] \tag{3.2}$$

Note that in the formulation of the state in (3.1) and (3.2) there are only 10 out of the 11 BPMs of the transfer line. The missing BPM corresponds to the first one named "BPM.430028", which does not have any upstream correctors.

The reward signals for horizontal and vertical trajectory steering tasks are simply the root mean square (RMS) errors between the measured and the reference position times (-1). The factor (-1) is coming from the fact that RL needs to maximise the reward, where in this task we would like to minimise the RMS, i.e $r(s_t) \to 0$. The reward for trajectory steering for n BPMs is defined as:

$$r_t^h(s_t) = -\sqrt{\frac{1}{n}\sum_{i=0}^{n-1} ||x_t^{(i)} - x_{\text{ref},t}^{(i)}||^2} \tag{3.3}$$

$$r_t^v(s_t) = -\sqrt{\frac{1}{n}\sum_{i=0}^{n-1} ||y_t^{(i)} - y_{\text{ref},t}^{(i)}||^2} \tag{3.4}$$

where $n = 10$ in the AWAKE case. The actions are 10-dimensional vectors in the horizontal or vertical plane with bounds on the relative deflection angle of ± 300 μrad:

$$a_t^h = [\theta_t^{h,(0)}, \ldots, \theta_t^{h,(9)}] \tag{3.5}$$

$$a_t^v = [\theta_t^{v,(0)}, \ldots, \theta_t^{v,(9)}] \tag{3.6}$$

The actions are relative settings, i.e. at each time step the actions defined by the agent are added to the existing magnet settings.

3.2 Hierarchical RL agent

The previous paragraphs introduced the set-up suitable for basic RL agents. The extensions to hierarchical reinforcement learning (HRL) for the case of the AWAKE trajectory steering problem will follow in this section. As introduced insection 2.5, HRL divides the main task into a hierarchy of subtasks learned by different agents. The higher level task is more abstract and has a longer time horizon. The policies of the higher and lower level agents interact with each other, as it is the higher level policy that defines the subtasks or goals of the lower ones to solve. The chosen HRL algorithm for the AWAKE agent was inspired by the implementation of O.Nachum, who proposed together with others a data efficient off-policy hierarchical reinforcement learning algorithm called HIRO [51]. An important ingredient will be so-called *off-policy correction* to tackle the initially non-optimal policies and associated rewards of the lower level agents for potentially correct goals from the higher level agent. *Off-policy correction* and its implementation will be described in detail at the end of the section.

3.2.1 HIRO architecture for AWAKE

The proposed hierarchical structure consists of a two layer architecture where two policies are learnt at the same time. Only the lower level agent interacts with the AWAKE environment through the actors. The higher level agent attempts to solve the problem of planning the optimal trajectory in state space $\mathcal{S}$, which the lower controller needs to go through via subtasks. In practice, the goals given by the higher agent are *reference trajectories*, noted as $\{g_t^{(i)}\}$, with i indicating the position at the i^{th} BPM. These *goals* are the output of the higher level policy or its actions. And these *goals* become additional input of the lower level agent or a part of the observables. The 10 dimensional goal vector replaces the original reference trajectory of the lower level agent as described above in the state space. Effectively, it means that the lower agent has no information on the ultimate goal (i.e. the user-defined reference trajectory). It only follows the goals indicated by the higher agent. The states of the higher and lower agent are shown respectively in equations (3.7) and (3.8) (For simplicity, only the horizontal steering problem will be considered in the text. It can however be easily extended to the vertical plane):

$$s_t^{\text{H}} = [x_t^{(0)}, x_t^{(1)}, \ldots, x_t^{(9)}, x_{\text{ref},t}^{(0)}, x_{\text{ref},t}^{(1)}, \ldots, x_{\text{ref},t}^{(9)}] \tag{3.7}$$

$$s_t^{\text{L}} = [x_t^{(0)}, x_t^{(1)}, \ldots, x_t^{(9)}, g_t^{(0)}, g_t^{(1)}, \ldots, g_t^{(9)}] \tag{3.8}$$

This behaviour of the lower level agent of not having the knowledge of the actual reference trajectory and the higher agent not being able to act directly on the environment is what constitutes the core of HRL. It permits that the higher level agent only focuses on longer time horizons without needing to take care of how to operate the trajectory correctors, whereas the lower agent can specialize on controlling the trajectory correctors.

The actions of the higher and lower level agent are written respectively in equations (3.9) and

(3.10), where the $\theta_t^{h,(i)}$ is the deflection angle of the i^{th} trajectory corrector as discussed in the previous section:

$$a_t^{\mathrm{H}} = [g_t^{(0)}, g_t^{(1)}, \ldots, g_t^{(9)}] \tag{3.9}$$

$$a_t^{\mathrm{L}} = [\theta_t^{h,(0)}, \ldots, \theta_t^{h,(9)}] \tag{3.10}$$

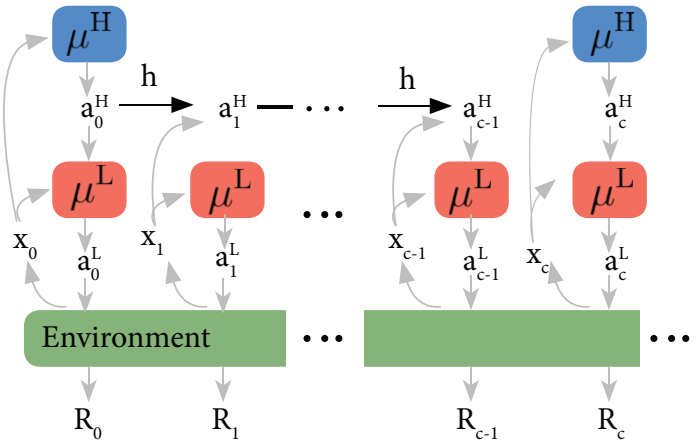

Figure 3.2: Temporal evolution during c steps of the architecture of the two level hierarchical policies learning off-policy.

Not both agents operate at every timestep. The higher level agent outputs an action every c timesteps only. This allows the lower agent to complete a subtask given by the higher agent between two of its actions. As mentioned in the section 2.5, c is a hyperparameter that needs to be tuned. A schematic of the interactions of the HIRO algorithm are given in Fig. 3.2.

3.2.2 Goal and transfer function

The higher level agent's observation are the beam position at every BPM $\{x_t^{(i)}\}$ and the user-defined reference trajectory $x_{\text{ref},t}^{(i)}$. Based on these observations, it outputs a goal every c steps $\vec{g}_t \in \mathbb{R}^n$, with the dimension n corresponding to the number of BPMs. This goal can either be relative or absolute depending on the design of the controller. A relative goal - most common in the field of robotics control - corresponds to for example the relative position of the beam with respect to a target value and is updated every time that the beam position changes. On the other hand, the absolute goal is a "quasi-"reference trajectory in the same space as the measured beam position and does not change as the position of the beam evolves in time. The relation between the relative goal at the i^{th} BPM $g_{\text{rel},t}^{(i)}$ and his absolute counterpart noted $g_{\text{abs},t}^{(i)}$ is as follows:

$$g^{(i)}_{\text{abs},t} = x^{(i)}_t + g^{(i)}_{\text{rel},t} \tag{3.11}$$

Depending on the control task, either the relative or absolute goal is more suitable. For the AWAKE trajectory steering case, the performance with both goal formulations is shown in section A.3. A so-called *transfer function* $h(x^{(i)}_t, g^{(i)}_t, x^{(i)}_{t+1})$ transforms the higher level goals from one time step to the next such that also in the case of a relative goal the lower level agent does not have to deal with a moving target while solving the tasks during c steps.

$$h(x^{(i)}_t, g^{(i)}_t, x^{(i)}_{t+1}) = x^{(i)}_t + g^{(i)}_t - x^{(i)}_{t+1} = g^i_{t+1} \tag{3.12}$$

When working with an absolute goal h becomes the identity function $h(x^{(i)}_t, g^{(i)}_t, x^{(i)}_{t+1}) = g^{(i)}_t = g^{(i)}_{t+1}$. In summary, every c steps a new goal is sampled from the higher level policy $\vec{g}_t = \pi^{\text{H}}(a^{\text{H}}_t | s^{\text{H}}_t)$ that is transmitted to the lower level policy. Then, during the $c-1$ remaining timesteps, this goal is propagated in time by the transition function $h(x^{(i)}_t, g^{(i)}_t, x^{(i)}_{t+1}) = g^{(i)}_{t+1}$.

3.2.3 Reward signal

The reward formulation differs for the two agents as they specialize in different tasks. The lower level reward is given in equation (3.14) and corresponds to the RMS error between the goal and the next measured trajectory at the different BPMs in case of an absolute goal (with equation (3.12) it can easily be formulated for relative goals).

$$r^{\text{H}}_t(\vec{x}_t, \vec{x}^{\text{ref}}_t) = \sum_{j=t}^{t+c-1} R_j \tag{3.13}$$

$$r^{\text{L}}_t(\vec{x}_t, \vec{g}_t, \vec{x}_{t+1}) = -\sqrt{\frac{1}{n}\sum_{i=0}^{n-1} ||g^{(i)}_{\text{abs},t} - x^{(i)}_{t+1}||^2} \tag{3.14}$$

The higher level agent receives the reward feedback every c steps and it is simply the environment reward accumulated during those c steps, see equation (3.13), where R_t is the environment reward obtained at time t as shown in (3.3).

3.2.4 Experience replay

Both agents, the higher and lower level ones, are implemented as algorithms based on the state-of-the-art RL off-policy actor-critic algorithm TD3 described in detail in subsection 2.4.4. The higher and lower level agents have different transitions at different update frequencies and therefore also need separate replay buffers. The experience for the lower agent is stored every timestep t and consists of $(\vec{x}_t, \vec{g}_t, \vec{a}^{\text{L}}_t, r^{\text{L}}_t, \vec{x}_{t+1}, h(\vec{x}_t, \vec{g}_t, \vec{x}_{t+1}))$, where a^{L}_t is the action of the lower level agent (3.10). For the higher level agent, the experience is stored every c steps and contains all the c transitions, that is $(\vec{x}_{t:t+c-1}, \vec{g}_{t:t+c-1}, \vec{a}^{\text{L}}_{t:t+c-1}, R_{t:t+c-1}, s_{t+c})$. The lower level actions are needed in the high level replay buffer for off-policy correction.

3.2.5 Off policy correction

During training, the change in the behavioural policy of the lower level agent creates a non-stationary problem for the higher level policy to solve. Old off-policy experience from the replay buffer might not be consistent anymore with lower level transitions for the same goal but an updated policy (e.g. the lower level agent's untrained policy might not reach the high level goals initially and the respective reward for the high level agent would not be representative for the goals it had issued). To still profit from the replay buffer and solve the issue of non-stationarity, Nachum et al. [51] and concurrently Levy et al. [40] proposed to relabel the goals $\vec{g}_t$ given by the higher agent when replaying experience. The goal relabelling or off-policy correction method that gave the best performance for the AWAKE case follows the approach by Nachum et al. [51].

The goals $\vec{g}_t$ in the replay buffer of the higher level agent are relabelled every time a mini-batch is sampled. The relabeled goal, noted $\tilde{g}_t$, is chosen to be the one that best fits the past low level behaviour $a_{t:t+c-1} \sim \mu^{\mathrm{L}}(\vec{x}_{t:t+c-1}, \vec{g}_{t:t+c-1}))$ but using the current low level policy. In other words, a new goal $\tilde{g}_t$ must be found that maximizes the likelyhood or log likelyhood of the played actions following the probability function $\mu^{\mathrm{L}}(a_{t:t+c-1}|\vec{x}_{t:t+c-1}, \tilde{g}_{t:t+c-1}))$. Once $\tilde{g}_t$ is determined, it can be propagated to $c-1$ other learning steps with the transition function (3.12). Maximising the log likelihood corresponds to maximising the negative MSE assuming a Normal distribution with constant variance:

$$\log \mu^{\mathrm{L}}(a_{t:t+c-1}|\vec{x}_{t:t+c-1}, \tilde{g}_{t:t+c-1})) \propto -\frac{1}{2} \sum_{i=t}^{t+c-1} ||a_i - \mu^{\mathrm{L}}(\vec{x}_i, \tilde{g}_i))||_2^2 + \text{ const} \tag{3.15}$$

In practice, (3.15) is maximised by computing it for a number of sampled candidate goals $\{\tilde{g}_{\mathrm{c},t}\}$ and then choosing the goal maximizing the probability instead of truly optimising (3.15). Candidate goals are intuitively situated near $\vec{x}_{t+c} - \vec{x}_t$ for a relative goal and around g_t for an absolute one. Thus, the candidate goals are sampled from a Normal distribution centred on these values. To this list of candidates the goal g_t of the past version of the lower behavioural policy is added. The optimisation problem is formulated as:

$$\begin{aligned} \tilde{g}_t &= \underset{g \in [g_t, \{\tilde{g}_{\mathrm{c},t}\}]}{\arg\max} -\frac{1}{2} \sum_{i=t}^{t+c-1} ||a_i - \mu^{\mathrm{L}}(\vec{x}_i, g))||_2^2 \\ \tilde{g}_{\mathrm{c},t} &\sim \mathcal{N}(\vec{x}_{t+c} - \vec{x}_t, \sigma) \qquad \text{if goal is relative} \\ \tilde{g}_{\mathrm{c},t} &\sim \mathcal{N}(g_t, \sigma) \qquad\quad \text{if goal is absolute} \end{aligned} \tag{3.16}$$

where $\tilde{g}_{\mathrm{c},t}$ are the sampled candidate goals. For the AWAKE problem, the number of candidates sampled was limited to 9, in order to have a total of 10 candidates counting the past goal g_t in addition. The standard deviation σ was chosen to be half the action range of the higher level agent.

Chapter 4

Results with Hierarchical Reinforcement Learning for AWAKE Steering

This chapter will summarise the results achieved for AWAKE electron line steering with a Hierarchical Reinforcement Learning algorithm implemented as part of this master project. The obtained performance will also be compared to a standard RL algorithm not using hierarchical policies. The code used to generate all the results can be found at `https://gitlab.cern.ch/borodrig/hrl_awake`.

4.1 AWAKE electron line steering with TD3

The training of RL agents is episodic. An episode corresponds to new initial conditions and hence a new problem to solve. An episode ends when either the maximal amount of steps is reached or when the task is solved (i.e. the reward reaches a certain threshold).The maximum allowed number of steps per episode for the AWAKE study was set to 50. An episode was considered completed successfully if the reward signal reached the threshold value of negative RMS $r_t^{h,v} > -1.6$ mm wrt a reference trajectory. Throughout the training, the agent was evaluated every 2 episodes for 1000 episodes. During evaluation the action noise, that helps exploration during training, noted as ϵ in **Algorithm 1**, is set to 0 and no policy parameter updates occur. The important performance metric for the trained agent are success rate of reaching the target (i.e. obtaining a reward above threshold) and number of steps the agent needs to reach the target. For the AWAKE electron line steering case, the final agent is supposed to reach the reference trajectory within a tolerance window with a success rate of ideally 100 % within one or maximum two iterations. During training itself, the number of training steps or iterations to reach a "good-enough" agent are of importance. The number of samples to train an agent should be as low as possible. The AWAKE trajectory steering problem can be efficiently solved with TD3 (or similar) as shown in [31]. The achievable performance with TD3 will be the bench mark for HRL.

4.1.1 Success rate with TD3 for AWAKE trajectory correction

The agent's success rate is computed as :

$$\frac{\text{Number of successful episodes}}{\text{Total number of evaluation episodes}}$$

For the AWAKE case, the success rate corresponds to the ratio of episodes out of 1000 episodes where the agent reaches a negative RMS error from the reference trajectory superior to -1.6 mm. Fig. 4.1 shows the results for the training of TD3 correcting the horizontal and vertical plane. After about 25 episodes the agents can correct to reference for any random initial trajectory.

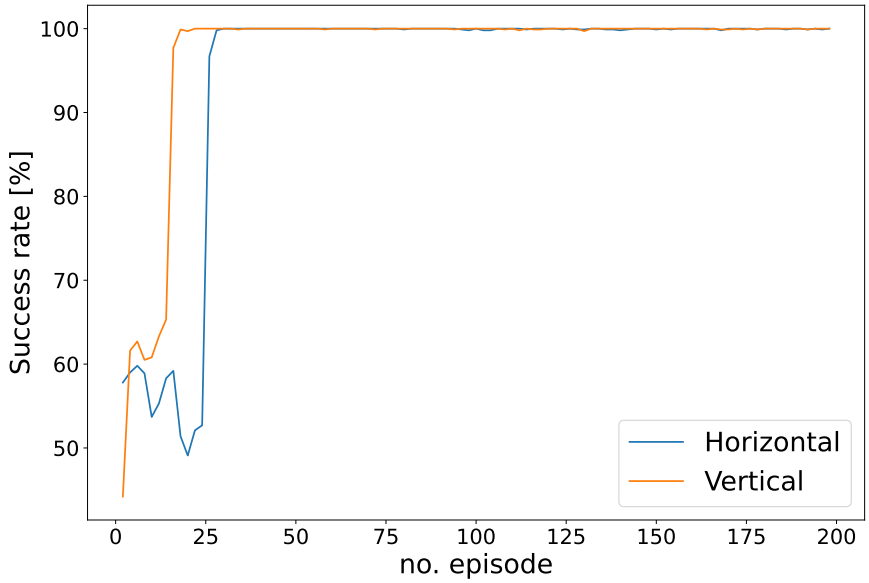

Figure 4.1: Success rate during training of TD3 for AWAKE electron line steering in the vertical and horizontal plane with evaluation on 1000 episodes per data point.

4.1.2 Mean number of iterations with TD3 for trajectory correction

For the agent to be useful when controlling the trajectory of the AWAKE electron line it needs to be able to complete the task in the minimum amount of time. The maximum possible is set to 50 iterations as defined in the environment. The evolution of number of iterations required to solve a task during TD3 training noted n_{step} is shown as average for a 1000 evaluation episodes in Fig. 4.2.

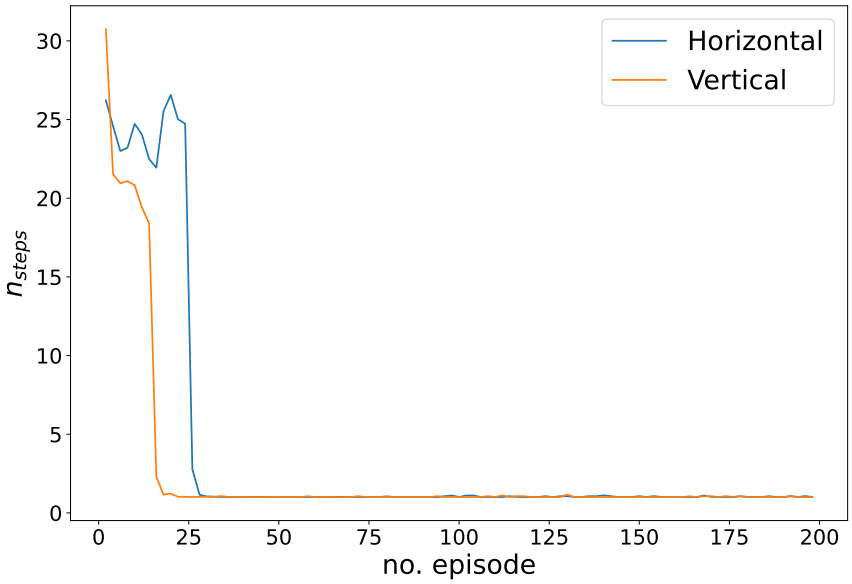

Figure 4.2: Mean number of steps during training of TD3 electron line steering in the vertical and horizontal plane with evaluation on 1000 episodes per data point.

As training evolves, the number of steps that are needed in order to solve the task decreases and reaches eventually $n_{step} = 1$, like the classical SVD method for the same problem introduced in subsection 2.1.2. The advantage of this method compared to SVD is however that this algorithm can also be used for non-linear systems.

4.2 Results with Hierarchical Reinforcement Learning

The same performance metrics as for TD3 will now be applied to HRL. For the following result plots, the agents were trained 10 times to test reproducibility of the results. Thus the plots show an average over 10 training runs. As mentioned before, the algorithms to implement the lower and higher level RL agents are based on TD3. TD3 comes with its own set of hyperparameters. The hyperparameters for the lower and higher level TD3 algorithms are the same as in the non-hierarchical set-up described earlier. To help convergence, the action noise ϵ follows a schedule such that is reduced during training as a function of the episode number. More information on the action noise can be found in the appendix A.2. The sensitivity to the HIRO hyperparameter c and the number of parameters for the actor and critic networks as well as the importance of off-policy correction were studied. Finally also using pre-trained lower level agents was tested. The results in the following are only shown for the vertical plane. HIRO behaves similarly on the horizontal plane.

4.2.1 Performance of HRL for AWAKE trajectory steering

HIRO was implemented from scratch as part of this master thesis. The resulting algorithm could be successfully tuned to learn trajectory steering for 10 degrees of freedom with acceptable performance. Fig. 4.3 shows the training evolution for the most performing hyperparameters for an example training run. The number of required iterations to reach the target as function of episode number (which is hear the tolerance window of acceptable negative RMS value) is shown on the upper plot and the initial and final negative RMS values per episode on the lower plot.

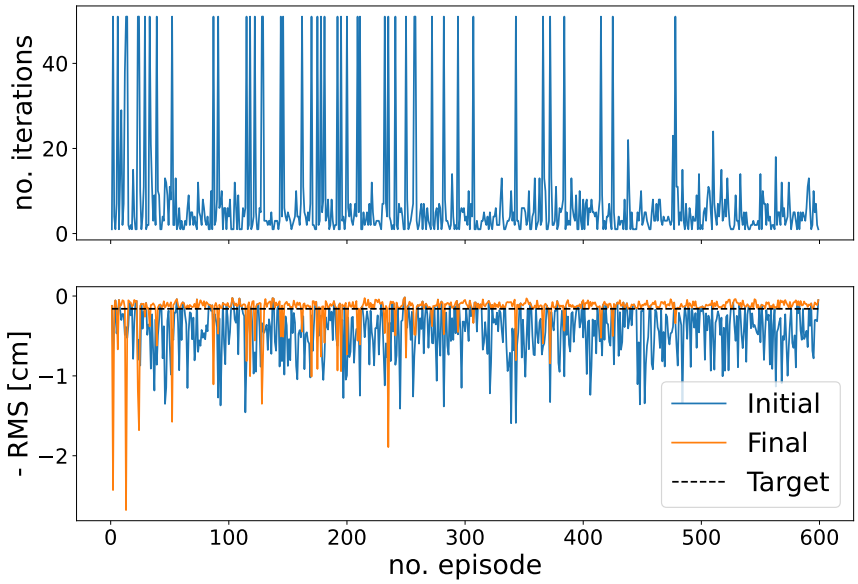

Figure 4.3: Number of training steps and initial and final negative RMS at every episode.

Figure 4.4 shows a comparison between TD3 and HIRO for the training evolution of trajectory correction at AWAKE. HRL successfully learns to correct the vertical plane of the AWAKE electron line. Nevertheless, TD3 learns to correct the line faster and more reliably (i.e. for TD3 the success rate remains at 100 % after the initial exploration phase). Also, the required number of iterations for successful correction remains at $n_{step} = 2$, which is acceptable but not as good as TD3. Note that the comparison between TD3 and HIRO was carried out with the optimum hyperparameters for HIRO. Relative goals were used instead of absolute ones, see appendix A.3 for details on goal configuration.

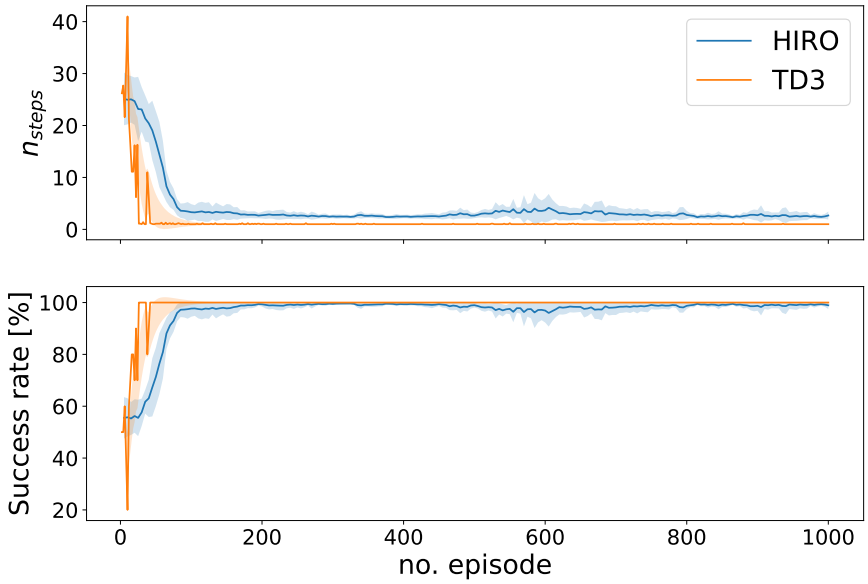

Figure 4.4: Performance during training of HIRO and TD3 on vertical trajectory correction at AWAKE. The results are the average of 10 training runs. Both agents learn to correct the AWAKE electron line trajectory successfully. TD3 is however faster and trains more consistently. Due to the hierarchical set-up for HIRO with parameter c, the required number of iterations for successful correction remains at minimum $n_{step} = 2$.

4.2.2 The HIRO hyperparameter c

As discussed previously, the lower level agent issues new actions at every environment time step, whereas the higher level one only every c time steps. The optimal value for parameter c is task dependent and, as will be seen, has a high impact on performance. The parameter c overrides to some extend the episode length limit set in the environment and defines how long the lower level agent is allowed to take to solve the task given by the higher level agent. Figures 4.5 and 4.6 show the comparison of performance for different parameters c as training evolves. The highest performance is achieved with $c = 2$, which intuitively fits with the observation of TD3 being able to correct the line within 1 iteration after training. Figure 4.6 would suggest that potentially $c = 1$ would further reduce the number of iterations per episode. However, the current implementation of HIRO does not allow to set $c = 1$ to enforce different latencies between the lower and higher level interactions. In a future release it will be possible to test also this configuration.

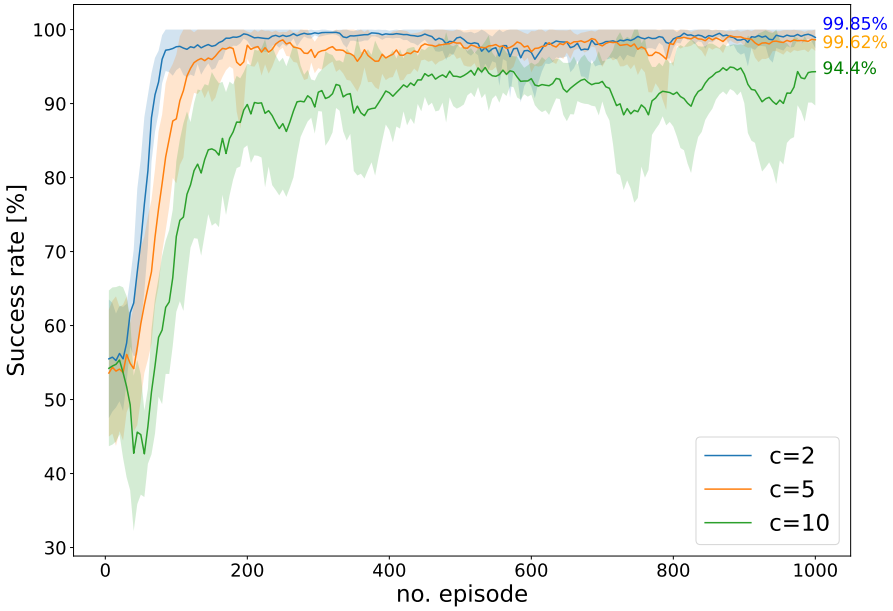

Figure 4.5: Comparison of success rate for different values of c as training evolves for HIRO. The results are an average over 10 training runs.

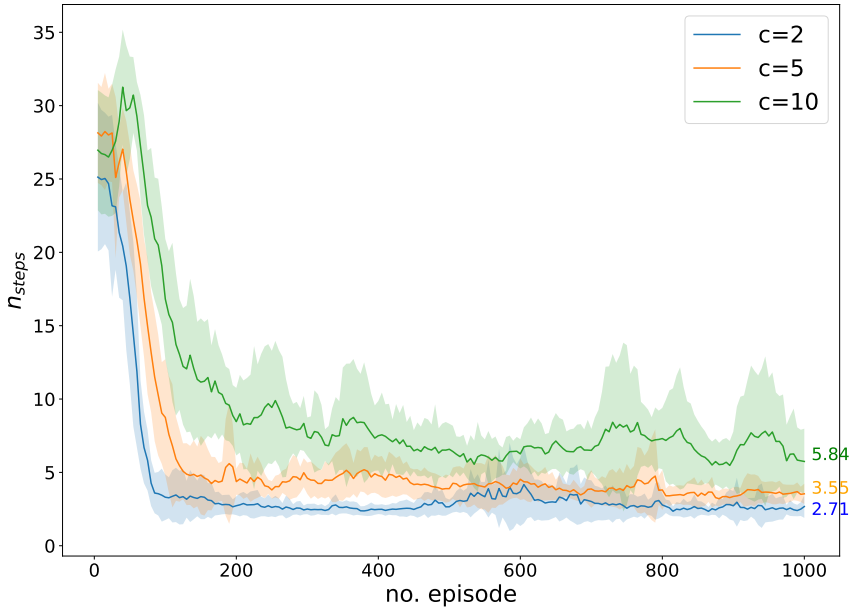

Figure 4.6: Comparison of number of iterations during training to correct the AWAKE electron line in the vertical plane for different values of c with HIRO. The results are an average over 10 training runs.

4.2.3 Off Policy Correction

Off-policy correction presented in subsection 3.2.5 proposed a relabelling of past high level actions $\vec{a}_t^{\,\mathrm{H}} = \vec{g}_t$ when sampled for experience replay. These relabelled goals $\tilde{g}_t$ maximize the log likelihood of the sequence of low level actions for the current behavioural policy $\mu^{\mathrm{L}}(a_{t:t+c-1}|\vec{x}_{t:t+c-1}, \tilde{g}_{t:t+c-1})$. Fig. 4.7 shows the success rate during training for vertical AWAKE trajectory steering with and without off-policy correction (OPC) and confirms the importance of relabelling past goals. Not only does the training with OPC reach a higher overall performance (99.85 % instead of 94 %) it also trains more consistently, indicated by the lower spread.

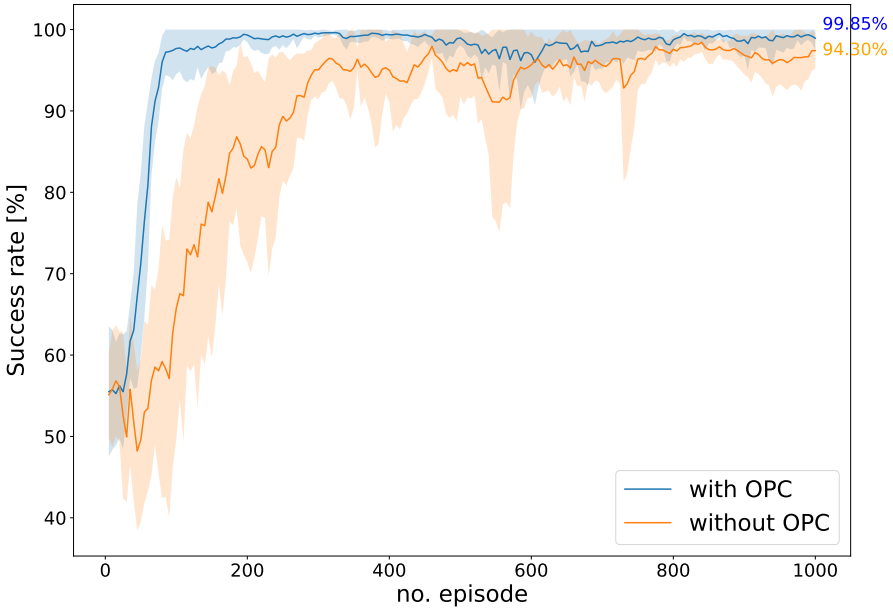

Figure 4.7: Success rate during training with and without off-policy correction. The results are an average over 10 training runs.

4.2.4 Pretraining Lower Level Agents

One of the appealing aspects that motivated the investigation of hierarchical reinforcement learning for CERN accelerators was the potential to reuse pre-trained agents for very specific tasks and allow for sophisticated hierarchies to run complicated accelerator systems. To test whether this could in principle work with HIRO, a pre-trained lower level agent (trained with standard TD3) was joined with an untrained higher level agent. Figures 4.8 and 4.9 show the performance evolution during training. Surprisingly, the overall performance does not profit at all from pre-trained lower level agents. Preliminary investigations suggest that the goals issued by the higher level agent do not sufficiently well resemble the user-defined reference trajectories (which a stand-alone low level agent would get as part of the state information), but the goals rather exaggerate the differences of the current trajectory to the reference one. And hence the pre-tained mapping from state to action in the lower level agent, does not bring any advantage rather the opposite. A future study will investigate this further and potentially improve the algorithm to benefit from pre-trained agents.

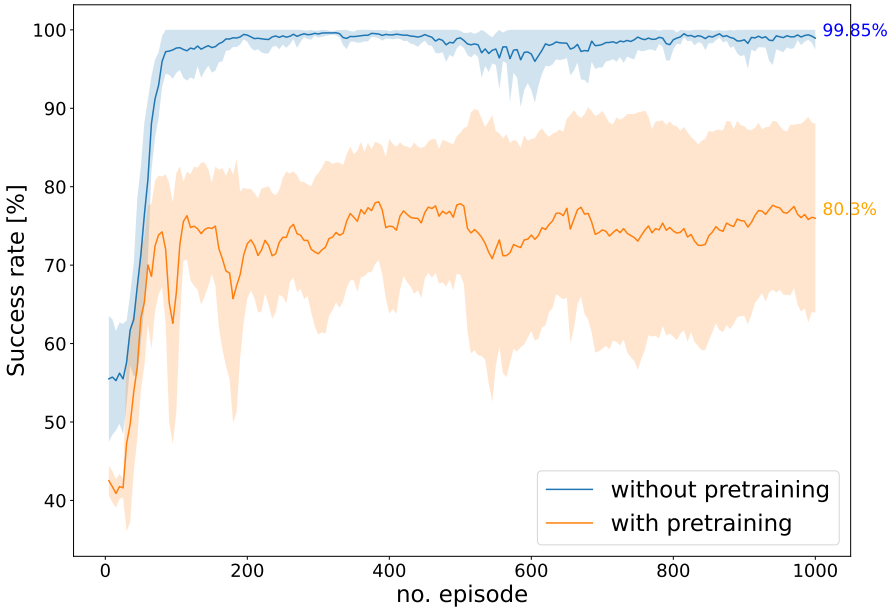

Figure 4.8: Success rate during training with and without pre-trained lower level agent for AWAKE vertical trajectory correction in the electron line. The results are an average over 10 training runs.

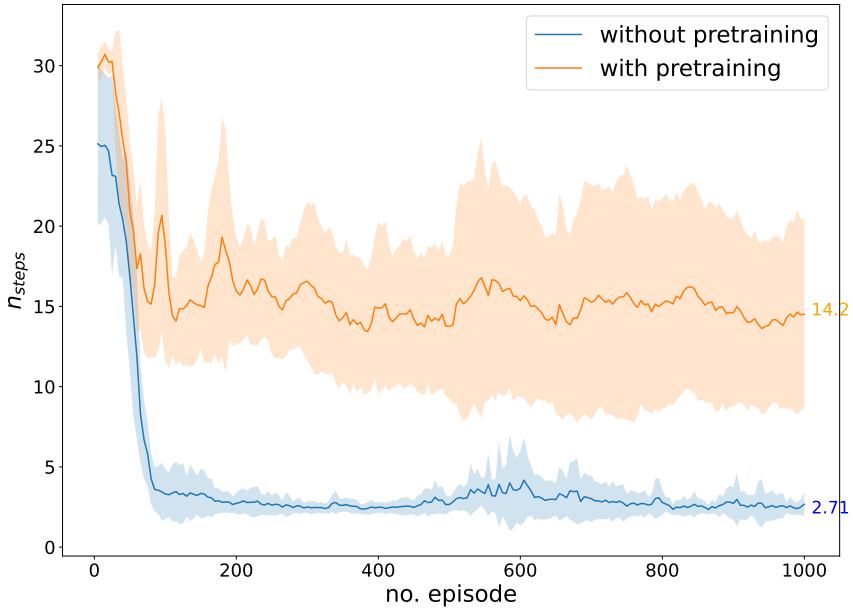

Figure 4.9: Mean number of iterations during training with HIRO to correct vertical AWAKE trajectory with and without a pre-trained lower level agent. The performance with pre-training is significantly poorer. The results are an average over 10 training runs.

4.2.5 Impact of Neural Network Size on HIRO performance

As final investigation and also for completeness, the impact of the number of neurons of actor and critic networks in the higher and lower level agents was investigated. It has been suggested that the neural network size should scale exponentially with respect to the dimension of the output in order to reduce the generalization error [25] [63]. Fig. 4.10 shows the results for the success rate as function of episode during training for number of neurons per hidden layer of n_{hw} = 300, 900 and 1500. Whereas the needed computing power for training the largest size agents increases significantly, the performance rather degrades. All the results above were therefore obtained with 300 neurons per hidden layer in the TD3 actor and critic networks with 2 hidden layers each.

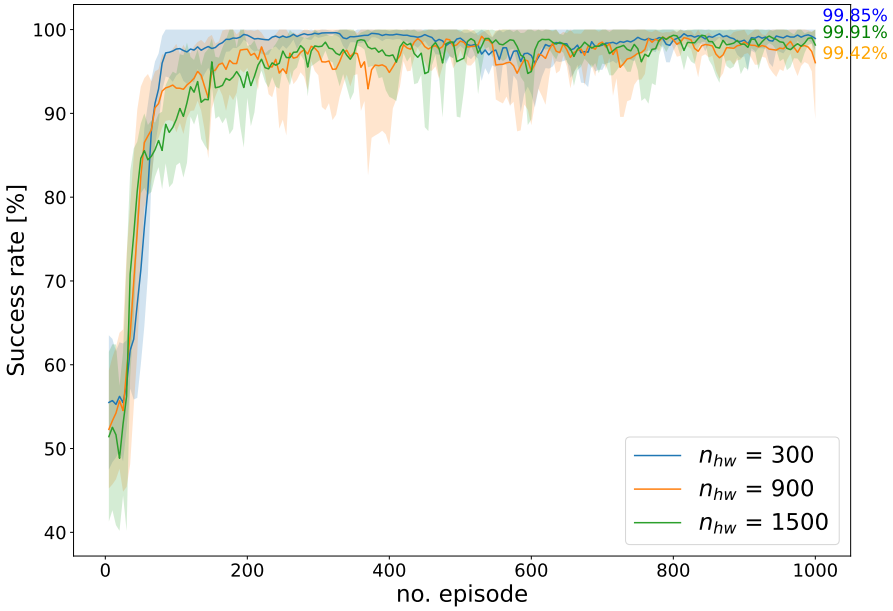

Figure 4.10: Success rate during training HIRO for AWAKE electron line steering with $n_{hw} = 300$, 900 or 1500 neurons per hidden layer in the actor and critic networks. The results are an average over 10 training runs.

4.3 Discussion

The complex concepts of hierarchical reinforcement learning could be successfully implemented and tested on a high dimensional use case of accelerator control. The test case was trajectory steering of the AWAKE electron line. State-of-the-art techniques such as experience replay and off-policy correction were used following the algorithm HIRO. Our HIRO derivative was implemented as an agent, with specific adaptations of TD3, replay buffers etc. Another possibility could be to implement the HIRO concepts as set of hierarchical wrapped Gym environments interacting with two (or more) standard RL agents. Both approaches have their advantages and disadvantages. The latter approach could have the benefit of rather easily swapping in and out new base off-policy RL algorithms. Further development and potentially moving to hierarchical wrapped environments will be part of a future study along with understanding the limitations and eventually boosting the performance in case of pre-trained low-level agents.

Chapter 5

Conclusion

The goal of this master project was to implement and test hierarchical reinforcement learning (HRL) for AWAKE electron line trajectory steering. The concepts of HRL were introduced in the thesis with the focus on the algorithm HIRO. With HRL, at least two control policies are learned at different abstraction levels. The tasks that the two resulting RL agents have to solve also have different time horizons. The higher level policy learns to plan in long time horizons and defines goals for the lower policy. The lower level agent interacts with actual hardware at short time horizons. The main result of the thesis is a fully functional HRL code based on HIRO that includes state-of-the-art techniques such as off-policy correction to help convergence. In a series of bench mark tests its performance was evaluated in comparison to standard RL algorithms (i.e. TD3). The HRL algorithm learns to correct the AWAKE electron trajectory to reference within roughly 100 episodes and shows a success rate of more than 99.8% when evaluated. The impact of a number of hyperparameters has been studied and an optimal set-up could be found. The overall performance of the HRL algorithm presented in this thesis is acceptable, but still poorer than standard state-of-the-art off-policy RL. Using pre-trained lower level agents could be of particular interest to re-use agents, that were trained previously on very specific tasks. This concept fits particularly well with HRL and could reduce the number of training iterations in the control room considerably. However, the obtained performance in the set-up with pre-trained agents did not meet expectations yet. Also of interest will be to test the HRL algorithm with a more complicated control problem, where longer time horizons are involved by design. A number of possible extensions or potential improvements are already planned and will hopefully eventually unleash the full power of hierarchical reinforcement learning for CERN accelerator controls.

Bibliography

[1] E. Adli, A. Ahuja, O. Apsimon, R. Apsimon, A.-M. Bachmann, D. Barrientos, F. Batsch, J. Bauche, V. K. Berglyd Olsen, M. Bernardini, T. Bohl, C. Bracco, F. Braunmüller, G. Burt, B. Buttenschön, A. Caldwell, M. Cascella, J. Chappell, E. Chevallay, M. Chung, D. Cooke, H. Damerau, L. Deacon, L. H. Deubner, A. Dexter, S. Doebert, J. Farmer, V. N. Fedosseev, R. Fiorito, R. A. Fonseca, F. Friebel, L. Garolfi, S. Gessner, I. Gorgisyan, A. A. Gorn, E. Granados, O. Grulke, E. Gschwendtner, J. Hansen, A. Helm, J. R. Henderson, M. Hüther, M. Ibison, L. Jensen, S. Jolly, F. Keeble, S.-Y. Kim, F. Kraus, Y. Li, S. Liu, N. Lopes, K. V. Lotov, L. Maricalva Brun, M. Martyanov, S. Mazzoni, D. Medina Godoy, V. A. Minakov, J. Mitchell, J. C. Molendijk, J. T. Moody, M. Moreira, P. Muggli, E. Öz, C. Pasquino, A. Pardons, F. Peña Asmus, K. Pepitone, A. Perera, A. Petrenko, S. Pitman, A. Pukhov, S. Rey, K. Rieger, H. Ruhl, J. S. Schmidt, I. A. Shalimova, P. Sherwood, L. O. Silva, L. Soby, A. P. Sosedkin, R. Speroni, R. I. Spitsyn, P. V. Tuev, M. Turner, F. Velotti, L. Verra, V. A. Verzilov, J. Vieira, C. P. Welsch, B. Williamson, M. Wing, B. Woolley, and G. Xia. "Acceleration of electrons in the plasma wakefield of a proton bunch". In: *Nature* 561.7723 (Aug. 2018), pp. 363–367. DOI: 10.1038/s41586-018-0485-4. URL: https://doi.org/10.1038%2Fs41586-018-0485-4.

[2] O. Barbalat. "TECHNOLOGY OF PARTICLE ACCELERATORS". In: 1990.

[3] W. A. Barletta, M. Bai, M. Battaglia, O. Bruning, J. Byrd, R. Ent, J. Flanagan, W. Gai, J. Galambos, G. Hoffstaetter, M. Hogan, M. Klute, S. Nagaitsev, M. Palmer, S. Prestemon, T. Roser, L. Rossi, V. Shiltsev, G. Varner, and K. Yokoya. *Planning the Future of U.S. Particle Physics (Snowmass 2013): Chapter 6: Accelerator Capabilities*. 2014. arXiv: 1401.6114 [hep-ex].

[4] Andrew G. Barto and Sridhar Mahadevan. "Recent Advances in Hierarchical Reinforcement Learning". In: *Discrete Event Dynamic Systems* 13.4 (2003), pp. 341–379. ISSN: 1573-7594. DOI: 10.1023/A:1025696116075. URL: https://doi.org/10.1023/A:1025696116075.

[5] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. *A Survey of Meta-Reinforcement Learning*. 2023. arXiv: 2301.08028 [cs.LG].

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. arXiv: 1606.01540 [cs.LG].

[7] Niky Bruchon, Gianfranco Fenu, Giulio Gaio, Simon Hirlander, Marco Lonza, Felice Andrea Pellegrino, and Erica Salvato. "An Online Iterative Linear Quadratic Approach for a Satisfactory Working Point Attainment at FERMI". In: *Information* 12.7 (2021). ISSN: 2078-2489. DOI: 10.3390/info12070262. URL: https://www.mdpi.com/2078-2489/12/7/262.

[8] P. J. Bryant. "A Brief history and review of accelerators". In: *CERN Accelerator School: Course on General Accelerator Physics*. 1992, pp. 1–16.

[9] Y Chung, G Decker, and K Evans. "Closed Orbit Correction Using Singular Value Decomposition of the Response Matrix". In: (1993). URL: https://cds.cern.ch/record/901051.

[10] J. D. Cockcroft and E. T. S. Walton. "Experiments with High Velocity Positive Ions. (I) Further Developments in the Method of Obtaining High Velocity Positive Ions". In: *Proc. Roy. Soc. Lond. A* 136 (1932), pp. 619–630. DOI: 10.1098/rspa.1932.0107.

[11] J. D. Cockcroft and E. T. S. Walton. "Experiments with High Velocity Positive Ions. II. The Disintegration of Elements by High Velocity Protons". In: *Proceedings of the Royal Society of London Series A* 137.831 (July 1932), pp. 229–242. DOI: 10.1098/rspa.1932.0133.

[12] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

[13] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. "Magnetic control of tokamak plasmas through deep reinforcement learning". In: *Nature* 602.7897 (2022), pp. 414–419. ISSN: 1476-4687. DOI: 10.1038/s41586-021-04301-9. URL: https://doi.org/10.1038/s41586-021-04301-9.

[14] A. L. Edelen, S. G. Biedron, B. E. Chase, D. Edstrom, S. V. Milton, and P. Stabile. "Neural Networks for Modeling and Control of Particle Accelerators". In: *IEEE Transactions on Nuclear Science* 63.2 (Apr. 2016), pp. 878–897. DOI: 10.1109/tns.2016.2543203. URL: https://doi.org/10.1109%2Ftns.2016.2543203.

[15] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. *Revisiting Fundamentals of Experience Replay*. 2020. arXiv: 2007.06700 [cs.LG].

[16] P Forck, P Kowina, and D Liakin. "Beam position monitors". In: (2009). DOI: 10.5170/CERN-2009-005.187. URL: https://cds.cern.ch/record/1213277.

[17] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. arXiv: 1802.09477 [cs.AI].

[18] Michael Gimelfarb, Scott Sanner, and Chi-Guhn Lee. *epsilon-BMC: A Bayesian Ensemble Approach to Epsilon-Greedy Exploration in Model-Free Reinforcement Learning*. 2020. arXiv: 2007.00869 [cs.LG].

[19] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[20] F. K. GOWARD and D. E. BARNES. "Experimental 8 Mev. Synchrotron for Electron Acceleration". In: *Nature* 158.4012 (1946), pp. 413–413. ISSN: 1476-4687. DOI: 10.1038/158413a0. URL: https://doi.org/10.1038/158413a0.

[21] Leander Grech, Diogo Alves, Stephen Jackson, Gianluca Valentino, and Jorg Wenninger. "Feasibility of Hardware Acceleration in the LHC Orbit Feedback Controller". In: (2019), MOPHA151. DOI: 10.18429/JACoW-ICALEPCS2019-MOPHA151. URL: https://cds.cern.ch/record/2772240.

[22] E. Gschwendtner, M. Turner, E. Adli, A. Ahuja, O. Apsimon, R. Apsimon, A.-M. Bachmann, F. Batsch, C. Bracco, F. Braunmüller, S. Burger, G. Burt, B. Buttenschön, A. Caldwell, J. Chappell, E. Chevallay, M. Chung, D. Cooke, H. Damerau, L. H. Deubner, A. Dexter, S. Doebert, J. Farmer, V. N. Fedosseev, R. Fiorito, R. A. Fonseca, F. Friebel, L. Garolfi, S. Gessner, B. Goddard, I. Gorgisyan, A. A. Gorn, E. Granados, O. Grulke, A. Hartin, A. Helm, J. R. Henderson, M. Hüther, M. Ibison, S. Jolly, F. Keeble, M. D. Kelisani, S.-Y. Kim, F. Kraus, M. Krupa, T. Lefevre, Y. Li, S. Liu, N. Lopes, K. V. Lotov, M. Martyanov, S. Mazzoni, V. A. Minakov, J. C. Molendijk, J. T. Moody, M. Moreira, P. Muggli, H. Panuganti, A. Pardons, F. Pena Asmus, A. Perera, A. Petrenko, A. Pukhov, S. Rey, P. Sherwood, L. O. Silva, A. P. Sosedkin, P. V. Tuev, F. Velotti, L. Verra, V. A. Verzilov, J. Vieira, C. P. Welsch, M. Wendt, B. Williamson, M. Wing, B. Woolley, and G. Xia and. "Proton-driven plasma wakefield acceleration in AWAKE". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 377.2151 (June 2019), p. 20180418. DOI: 10.1098/rsta.2018.0418. URL: https://doi.org/10.1098%2Frsta.2018.0418.

[23] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. *Deep Reinforcement Learning and the Deadly Triad*. 2018. arXiv: 1812.02648 [cs.AI].

[24] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: 1509.06461 [cs.LG].

[25] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. *Deep Learning Scaling is Predictable, Empirically*. 2017. arXiv: 1712.00409 [cs.LG].

[26] Simon Hirlaender and Niky Bruchon. *Model-free and Bayesian Ensembling Model-based Deep Reinforcement Learning for Particle Accelerator Control Demonstrated on the FERMI FEL*. 2022. arXiv: 2012.09737 [cs.LG].

[27] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[28] Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. “Hierarchical Reinforcement Learning: A Survey and Open Research Challenges”. In: *Machine Learning and Knowledge Extraction* 4.1 (2022), pp. 172–221. ISSN: 2504-4990. DOI: 10.3390/make4010009. URL: https://www.mdpi.com/2504-4990/4/1/9.

[29] D Jacquet, R Gorbonosov, and G Kruk. “LSA - the High Level Application Software of the LHC - and Its Performance During the First Three Years of Operation”. In: (2014). URL: https://cds.cern.ch/record/1696961.

[30] Nicholas K. Jong, Todd Hester, and Peter Stone. “The Utility of Temporal Abstraction in Reinforcement Learning”. In: *The Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*. May 2008. URL: http://www.cs.utexas.edu/users/ai-lab?AAMAS08-jong.

[31] Verena Kain, Simon Hirlander, Brennan Goddard, Francesco Maria Velotti, Giovanni Zevi Della Porta, Niky Bruchon, and Gianluca Valentino. “Sample-efficient reinforcement learning for CERN accelerator control”. In: *Phys. Rev. Accel. Beams* 23 (12 2020), p. 124801. DOI: 10.1103/PhysRevAccelBeams.23.124801. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.23.124801.

[32] D. W. Kerst. “The Acceleration of Electrons by Magnetic Induction”. In: *Phys. Rev.* 60 (1 1941), pp. 47–53. DOI: 10.1103/PhysRev.60.47. URL: https://link.aps.org/doi/10.1103/PhysRev.60.47.

[33] Johannes Kirschner, Mojmir Mutný, Andreas Krause, Jaime Coello de Portugal, Nicole Hiller, and Jochem Snuverink. “Tuning particle accelerators with safety constraints using Bayesian optimization”. In: *Phys. Rev. Accel. Beams* 25 (6 June 2022), p. 062802. DOI: 10.1103/PhysRevAccelBeams.25.062802. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.25.062802.

[34] Vijay Konda and John Tsitsiklis. “Actor-Critic Algorithms”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[35] Ezgi Korkmaz. *Adversarial Robust Deep Reinforcement Learning Requires Redefining Robustness*. 2023. arXiv: 2301.07487 [cs.LG].

[36] Ezgi Korkmaz and Jonah Brown-Cohen. *Detecting Adversarial Directions in Deep Reinforcement Learning to Make Robust Decisions*. 2023. arXiv: 2306.05873 [cs.LG].

[37] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. *Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation*. 2016. arXiv: 1604.06057 [cs.LG].

[38] Guillaume Laurent, Laëtitia Matignon, and Nadine Fort-Piat. "The world of Independent learners is not Markovian." In: *KES Journal* 15 (Mar. 2011), pp. 55–64. DOI: 10.3233/KES-2010-0206.

[39] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[40] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. *Learning Multi-Level Hierarchies with Hindsight*. 2019. arXiv: 1712.00948 [cs.AI].

[41] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[42] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[43] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning* 8.3 (May 1992), pp. 293–321. ISSN: 1573-0565. DOI: 10.1007/BF00992699. URL: https://doi.org/10.1007/BF00992699.

[44] Chuyu Liu, A. Marusic, Michiko Minty, and Vadim Ptitsyn. "A SVD-based orbit steering algorithm for RHIC injection". In: (Jan. 2012), pp. 523–525.

[45] Shengli Liu, Paul Dirksen, Spencer Gessner, Franck Guillot-Vignot, David Medina, Lars Søby, and Victor Verzilov. "The Installation and Commissioning of the AWAKE Stripline BPM". In: (2019), TUPB01. DOI: 10.18429/JACoW-IBIC2018-TUPB01. URL: https://cds.cern.ch/record/2716026.

[46] Michiko Minty and Frank Zimmermann. *Measurement and Control of Charged Particle Beams*. Jan. 2003. ISBN: 978-3-642-07914-6. DOI: 10.1007/978-3-662-08581-3.

[47] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. arXiv: 1602.01783 [cs.LG].

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[49] Ted Moskovitz, Jack Parker-Holder, Aldo Pacchiano, Michael Arbel, and Michael I. Jordan. *Tactical Optimism and Pessimism for Deep Reinforcement Learning*. 2022. arXiv: 2102.03765 [cs.LG].

[50] Patric Muggli. “Physics to plan AWAKE Run 2”. In: *Journal of Physics: Conference Series* 1596.1 (2020), p. 012008. DOI: 10.1088/1742-6596/1596/1/012008. URL: https://doi.org/10.1088%2F1742-6596%2F1596%2F1%2F012008.

[51] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. *Data-Efficient Hierarchical Reinforcement Learning*. 2018. arXiv: 1805.08296 [cs.LG].

[52] Ofir Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. *Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning?* 2019. arXiv: 1909.10618 [cs.LG].

[53] M L Oliphant, J S Gooden, and G S Hide. “The acceleration of charged particles to very high energies”. In: *Proceedings of the Physical Society* 59.4 (July 1947), p. 666. DOI: 10.1088/0959-5309/59/4/314. URL: https://dx.doi.org/10.1088/0959-5309/59/4/314.

[54] Xiaoying Pang, Sunil Thulasidasan, and Larry Rybarcyk. *Autonomous Control of a Particle Accelerator using Deep Reinforcement Learning*. 2020. arXiv: 2010.08141 [cs.AI].

[55] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. *Time Limits in Reinforcement Learning*. 2022. arXiv: 1712.00378 [cs.LG].

[56] Stephen Peggs and Todd Satogata. *Introduction to Accelerator Dynamics*. Cambridge University Press, 2017. DOI: 10.1017/9781316459300.

[57] T H B Persson, Helmut Burkhardt, Laurent Deniau, Andrea Latina, and Piotr Skowroński. “MAD-X for Future Accelerators”. In: *JACoW IPAC* 2021 (2021), WEPAB028. DOI: 10.18429/JACoW-IPAC2021-WEPAB028. URL: https://cds.cern.ch/record/2804337.

[58] Juergen Pfingstner, M Hofbauer, Daniel Schulte, and J Snuverink. “SVD-based filter design for the trajectory feedback of CLIC”. In: (Jan. 2011).

[59] Allan Pinkus. “Approximation theory of the MLP model in neural networks”. In: *Acta Numerica* 8 (1999), pp. 143–195. DOI: 10.1017/S0962492900002919.

[60] R Ramjiawan, S Döbert, J Farmer, E Gschwendtner, F M Velotti, L Verra, G Zevi Della Porta, V Bencini, and P N Burrows. “Design and operation of transfer lines for plasma wakefield accelerators using numerical optimizers”. In: *Phys. Rev. Accel. Beams* 25.10 (2022), p. 101602. DOI: 10.1103/PhysRevAccelBeams.25.101602. URL: https://cds.cern.ch/record/2839258.

[61] Rebecca Ramjiawan, Vittorio Bencini, Steffen Doebert, John Farmer, Edda Gschwendtner, Francesco Velotti, Livio Verra, and Giovanni Zevi Della Porta. *Design of the AWAKE Run 2c transfer lines using numerical optimizers*. 2022. arXiv: 2203.01605 [physics.acc-ph].

[62] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. ISBN: 9780975937730. URL: https://books.google.ch/books?id=MrJctAEACAAJ.

[63] Jonathan S. Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. *A Constructive Prediction of the Generalization Error Across Scales*. 2019. arXiv: 1909.12673 [cs.LG].

[64] A. Sanchez-Gonzalez, P. Micaelli, C. Olivier, T. R. Barillot, M. Ilchen, A. A. Lutman, A. Marinelli, T. Maxwell, A. Achner, M. Agåker, N. Berrah, C. Bostedt, J. D. Bozek, J. Buck, P. H. Bucksbaum, S. Carron Montero, B. Cooper, J. P. Cryan, M. Dong, R. Feifel, L. J. Frasinski, H. Fukuzawa, A. Galler, G. Hartmann, N. Hartmann, W. Helml, A. S. Johnson, A. Knie, A. O. Lindahl, J. Liu, K. Motomura, M. Mucke, C. O'Grady, J.-E. Rubensson, E. R. Simpson, R. J. Squibb, C. Såthe, K. Ueda, M. Vacher, D. J. Walke, V. Zhaunerchyk, R. N. Coffee, and J. P. Marangos. "Accurate prediction of X-ray pulse properties from a free-electron laser using machine learning". In: *Nature Communications* 8.1 (June 2017), p. 15461. ISSN: 2041-1723. DOI: 10.1038/ncomms15461. URL: https://doi.org/10.1038/ncomms15461.

[65] Michael Schenk, Elías F. Combarro, Michele Grossi, Verena Kain, Kevin Shing Bruce Li, Mircea-Marian Popa, and Sofia Vallecorsa. *Hybrid actor-critic algorithm for quantum reinforcement learning at CERN beam lines*. 2022. arXiv: 2209.11044 [quant-ph].

[66] Janet Schmidt et al. "The AWAKE Electron Primary Beam Line". In: *6th International Particle Accelerator Conference*. 2015, WEPWA039. DOI: 10.18429/JACoW-IPAC2015-WEPWA039.

[67] John Schulman, Xi Chen, and Pieter Abbeel. *Equivalence Between Policy Gradients and Soft Q-Learning*. 2018. arXiv: 1704.06440 [cs.LG].

[68] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, June 2014, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html.

[69] Utsav Singh and Vinay P Namboodiri. *CRISP: Curriculum inducing Primitive Informed Subgoal Prediction for Hierarchical Reinforcement Learning*. 2023. arXiv: 2304.03535 [cs.LG].

[70] Gilbert Strang. *Introduction to Linear Algebra*. Fourth. Wellesley, MA: Wellesley-Cambridge Press, 2009.

[71] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

[72] T. Tajima and J. M. Dawson. "Laser Electron Accelerator". In: *Phys. Rev. Lett.* 43 (4 July 1979), pp. 267–270. DOI: 10.1103/PhysRevLett.43.267. URL: https://link.aps.org/doi/10.1103/PhysRevLett.43.267.

[73] Francesco Maria Velotti, Brennan Goddard, Verena Kain, Rebecca Ramjiawan, Giovanni Zevi Della Porta, and Simon Hirlaender. *Automatic setup of 18 MeV electron beamline using machine learning*. 2022. arXiv: 2209.03183 [physics.acc-ph].

[74] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. *Sample Efficient Actor-Critic with Experience Replay*. 2017. arXiv: 1611.01224 [cs.LG].

[75] R. Widerøe. *The Infancy of particle accelerators: Life and work of Rolf Wideroe*. Ed. by P. Waloschek. Braunschweig, Germany: Vieweg, 1994.

[76] E. J. N. Wilson. “Fifty years of synchrotrons”. In: *Conf. Proc. C* 960610 (1996). Ed. by S. Myers, A. Pacheco, R. Pascual, C. Petit-Jean-Genaz, and J. Poole, pp. 135–139.

[77] A. Wolski. *Beam Dynamics in High Energy Particle Accelerators*. Imperial College Press, 2014. ISBN: 9781783262779. URL: https://books.google.ch/books?id=0GwlmwEACAAJ.

[78] Jiachen Yang, Igor Borovikov, and Hongyuan Zha. *Hierarchical Cooperative Multi-Agent Reinforcement Learning with Skill Discovery*. 2020. arXiv: 1912.03558 [cs.LG].

[79] M. Yarmohammadi Satri, A. M. Lombardi, and F. Zimmermann. “Multiobjective genetic algorithm approach to optimize beam matching and beam transport in high-intensity hadron linacs”. In: *Phys. Rev. Accel. Beams* 22 (5 May 2019), p. 054201. DOI: 10.1103/PhysRevAccelBeams.22.054201. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.22.054201.

[80] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey”. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2020, pp. 737–744. DOI: 10.1109/SSCI47803.2020.9308468.

Appendix A

Numerical results

A.1 Response matrix of AWAKE electron line

The response matrix formalism in the control setting has already been shown in the subsection 2.1.2. This matrix is actively used in the environment describing the AWAKE electron line trajectory steering task described in section 3.1.

Figure A.1: Matrix coefficients of the AWAKE electron line response generated by the the MAD-X simulation.

In this study a surrogate model of the transfer line, this model is based on MAD-X simulations. It is possible to see the response matrix in Fig. A.1. As mentioned in the subsection 2.1.2, the response matrix of a transfer line is sparse as trajectory correctors further down the line have no influence on positions measured at BPMs before them, i.e $\mathbf{R}_{ij} = 0$ for $i < j$. This property is naturally verified for the simulation of the AWAKE electron line shown in Fig. A.1.

A.2 Action Noise Scheduling

In order to reduce the stochasticity of the interaction between the higher and lower agent during the learning of the two policies, the action noise ϵ (see **Algorithm 1**) is scheduled. By reducing the action noise as the agents learns their respective policy it is possible to reduce variability in the performances of said agents. Indeed, it is important for them to prioritize exploration in the initial learning steps to be able to learn a mapping of what state-action pairs produces a better reward. This exploration is further encouraged by the introduction additive noise to the action increasing the variability of the controller's behaviour. Nevertheless, an action noise with a magnitude wrongfully calibrated can be detrimental to the learning of the optimal policy especially when the learning is very stochastic.

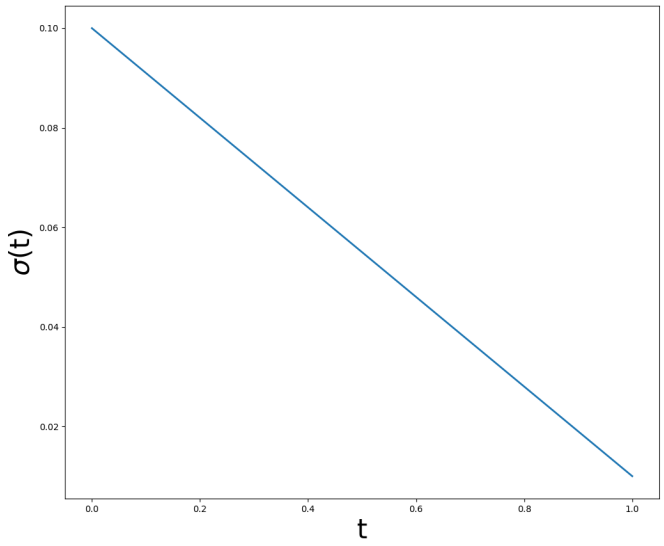

Figure A.2: Temporal dependency of action noise' standard deviation.

To reduce the stochasticity once the agent has already sufficiently learnt a good approximation of the state-action mapping, the standard deviation of the normally distributed action noise is reduced with respect to time, i.e. $\epsilon \sim \mathcal{N}(0, \sigma(t))$. The time dependency of the standard deviation is pictured in Fig. A.2. It can be seen that the function followed by the standard deviation is $\sigma(t) = 0.1(1-t) + 0.01t$

for a normalized time. It is simply a linear decay from its initial value of $\sigma(0) = 0.1$ to its final value $\sigma(1) = 0.01$.

A.3 Relative or absolute goal

As mentioned in subsection 3.2.2 the goals given by the higher agent to the lower one can be set in an absolute or relative framework. These goals are related as described in equation (3.11).

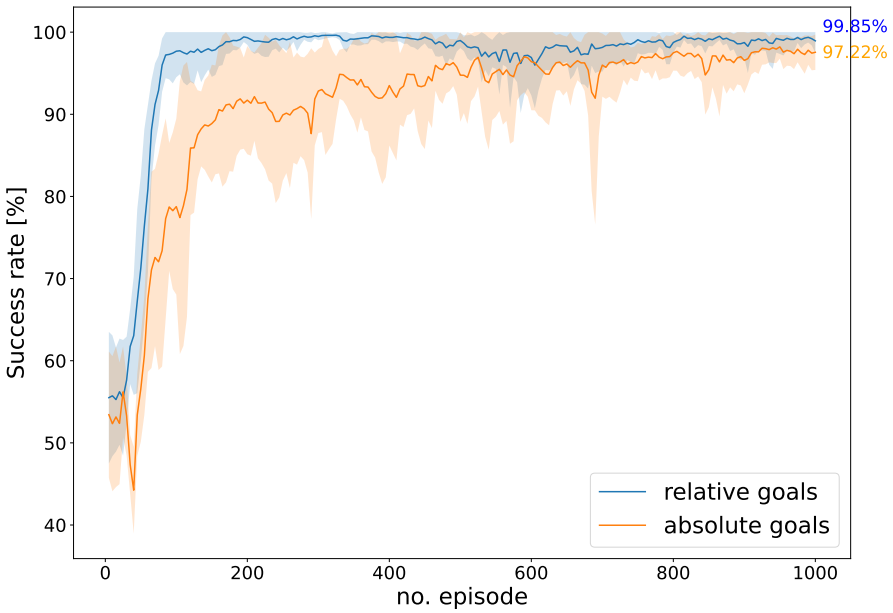

Figure A.3: Comparison of success rate for absolute or relative goals set by the higher behavioural policy μ^{H} as training evolves for HIRO. The results are an average over 10 training runs.

To determine how the setting the goals in an absolute or relative manner impacts the performances of the HIRO agent, their success rate is compared in Fig. A.3. It is possible to observe that the lower agent manages to perform more efficiently with relative goals. The optimal hierarchical agent in the AWAKE steering task should then use relative goals and not absolute ones.