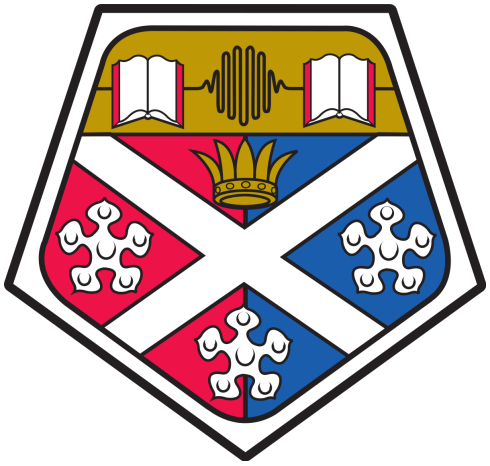
Deep Reinforcement Learning-based Longitudinal Optimization and Control of the Proton Synchrotron Booster at CERN



Lewis Kennedy

Department of Physics, University of Strathclyde

Submitted for the degree of

MPhys Physics with International Placement

Supervised by Dr. Simon Albright (CERN), Dr. Piotr Skowronski (CERN)

2023

CERN-THESIS-2023-438
01/06/2023

Acknowledgments

I would like to express my sincere gratitude to everyone who has contributed to the completion of my master's thesis both directly and indirectly. I want to thank my supervisors, Piotr Skowronski and Simon Albright, for their invaluable guidance throughout my project, and for granting me the opportunity to complete my project at such an amazing accelerator facility. I also want to thank everyone in BE-OP-PSB, without whom it would not have been possible to collect the data required for my studies. Last but certainly not least, I want to thank my friends and family for being pillars of support and for their never ending encouragement throughout my studies.

Abstract

This thesis presents the development and implementation of algorithmic methods for automated optimization software in the accelerator control room at CERN. Specifically, the problem of optimizing longitudinal profiles at PSB beam injection for improved luminosity in LHC-type beams is addressed using deep reinforcement learning methods. A soft-actor-critic RL agent was trained on the online accelerator, and demonstrated promising results by simultaneously reducing the transverse emittance while increasing beam transmission. A surrogate model of the problem was developed using data collected during online training, which was then used to train the agent offline in an attempt to mitigate the impact of low sample availability. The agent was able to consistently solve the offline optimization problem within 13 steps, indicating the potential effectiveness of a fully trained RL agent. However, there are still a number of limitations which need to be addressed, particularly in sample efficiency and the agent's performance in dynamic environments.

Contents

1 **Introduction** . . . 1
1.1 Background and Motivation . . . 1
1.2 Formulating the Problem . . . 2
1.2.1 Proton Synchrotron Booster . . . 2
1.2.2 Longitudinal Beam Dynamics . . . 3
1.2.3 Transverse Beam Dynamics . . . 6
1.3 Optimization Objectives . . . 9

2 **Deep Reinforcement Learning** . . . 9
2.1 Markov Decision Processes . . . 9
2.2 Deep Learning . . . 10
2.3 Policy Approximation . . . 11
2.4 Policy Evaluation . . . 13
2.5 Continuous Models . . . 14

3 **Methodology** . . . 15
3.1 Agent Development . . . 15
3.2 OpenAI Gym Framework . . . 16
3.3 Updating the Environment . . . 16
3.4 Machine Safety Constraints . . . 17
3.5 Reward Function . . . 18
3.6 Reward Saturation . . . 19
3.7 Surrogate Environment . . . 20
3.8 Accelerator Interaction Loop . . . 20

4 **Results** . . . 21
4.1 Online Training . . . 21
4.2 Offline Training . . . 22
4.3 Agent Deployment . . . 23

5 **Discussion** . . . 25
5.1 General Discussion . . . 25
5.1.1 Performance . . . 25
5.1.2 Limitations . . . 26
5.1.3 Mitigation . . . 26
5.2 Further Studies . . . 27
5.2.1 Model Validity . . . 27
5.2.2 Input and Observation Data . . . 27
5.2.3 Dynamic Environments . . . 27

6 **Conclusion** . . . 28

1 Introduction

1.1 Background and Motivation

The CERN accelerator complex comprises a series of particle accelerators which serve diverse experimental end-users, including those at the Large Hadron Collider (LHC) such as ALICE, ATLAS, CMS and LHCb, and other experiments such as the Advanced Proton Driven Plasma Wakefield Acceleration Experiment (AWAKE) and the Isotope mass Separator On-Line facility (ISOLDE). Particles for the proton-based experiments are accelerated by Linear Accelerator 4 (LINAC4), which feeds the chain consisting of the Proton Synchrotron Booster (PSB), the Proton Synchrotron (PS), the Super Proton Synchrotron (SPS) and the LHC. The complex is illustrated in Fig. 1.1, showing how each of the individual experiments receive protons.

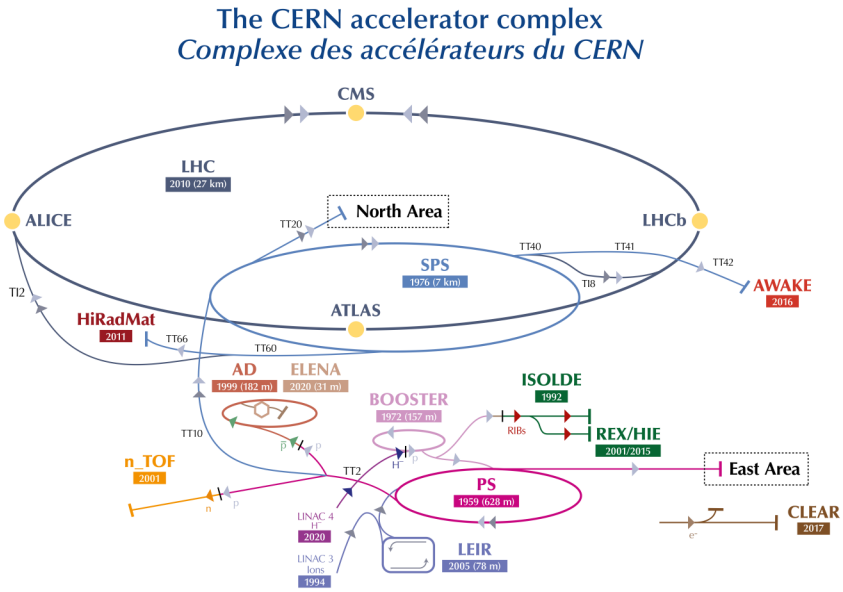


Figure 1.1: CERN's particle accelerator complex [1].

Variations in beam specifications required by the different experiments result in rapid accelerator configuration changes from shot to shot, motivating the need for automation in the accelerator control room. Recently, there has been a trend towards the adoption of data-driven approaches such as Reinforcement Learning (RL) to mimic human behaviour for control tasks [2, 3], which has accelerated discoveries across many scientific disciplines. With an abundance of real data and the presence of highly repetitive control tasks, accelerator facilities provide an excellent environment for applying RL methods. Accelerator physicists are looking to capitalize on the strengths of RL and similar optimization frameworks to not only offer increased levels of automation, but to be able to push the boundaries of beam performance using advanced statistical optimization methods.

Traditional numerical optimization and machine-learning based approaches have already shown promise in solving control tasks in similar settings [4, 5, 6]. However, traditional optimization algorithms lack memory, necessitating starting from scratch any time op-

timization is needed. Deep RL overcomes this limitation by incorporating a Deep Neural Network (DNN) to memorize past experience, allowing an RL algorithm to achieve human-like optimization [7, 8] while capitalizing on the strengths of algorithmic optimization. This thesis details how optimization of the PSB was tackled using Deep RL in an attempt to automate beam performance optimization.

1.2 Formulating the Problem

1.2.1 Proton Synchrotron Booster

The PSB is a synchrotron made from four superposed rings which receive protons from LINAC4. LINAC4 chops and accelerates H^- bunches to 160MeV [9] before they are passed through an electron stripping foil in the PSB leaving only bare protons. This is what is known as charge exchange injection and is illustrated in Fig. 1.2 [10].

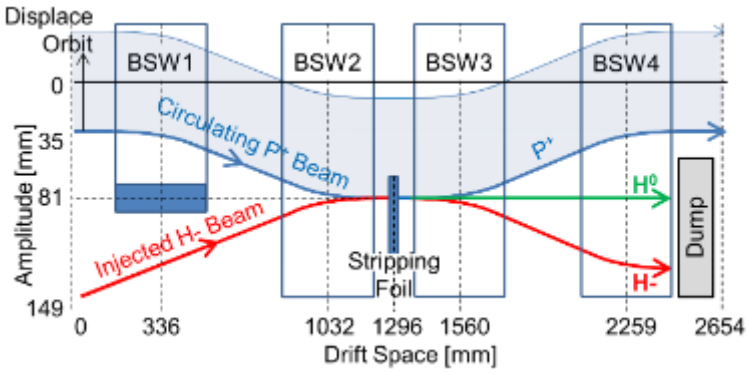


Figure 1.2: The injection interface between LINAC4 and the PSB [10].

The stripping process is not 100% efficient, so magnetic fields are used to deflect surviving particles using the Newton-Lorentz Force in Eq. (1.1).

$$\frac{d\hat{p}}{dt} = \hat{F} = q(\hat{E} + \hat{v} \times \hat{B}) \tag{1.1}$$

The magnetic field contribution $\hat{F} = q(\hat{v} \times \hat{B})$ acts to deflect particles in the transverse planes with a focusing strength $k \propto 1/B\rho$, where the magnetic rigidity $B\rho = p/q$. Hence depending on a particle's charge and momentum, they are either deflected towards the beam dump or accelerated in the ring. Charge exchange injection allows injected and circulating beams to be overlapped to increase the phase space density without increasing the beam size. As explained by Liouville's theorem, injecting protons would require a continually increasing emittance [11, 12], which is the area of the beam in phase space, and which defines the size of the beam in the given plane. Liouville's theorem states that the phase-state distribution is invariant along the trajectories of the system, hence the emittance is invariant under the effect of purely Hamiltonian forces [11]. However, there are cases where Liouville's theorem does not hold. Examples include during adiabatic damping where the beam emittance shrinks under acceleration, Coulomb scattering, and filamentation which can all lead to poor beam quality [13, 14, 15]. The emittance typically only grows throughout the accelerator complex [16], so the optimization of the PSB is crucial since the ceiling on beam quality seen by the downstream accelerators is

ultimately determined by the PSB. For LHC-type beams the ultimate goal is to reach higher luminosities meaning an increased phase space density and a higher probability and rate of collisions. Luminosity upgrades are therefore required for more efficient data generation and collection. The luminosity is given by Eq. (1.2) [17, 18].

$$\mathcal{L} = \frac{N_1 N_2 f N_b}{4\pi\sigma_x\sigma_y} \tag{1.2}$$

where N_1 and N_2 are the intensities of two colliding bunches, f is the revolution frequency, N_b is the number of bunches in one beam, and $\sigma_{x,y}$ the beam sizes in the transverse planes given by the beam emittance. It is therefore necessary to reduce the beam emittance through optimizing both the longitudinal and transverse beam dynamics.

1.2.2 Longitudinal Beam Dynamics

Longitudinal beam dynamics are a fundamental aspect of particle acceleration, as they determine key beam parameters such as the energy spread of the beam. The radiofrequency (RF) field contribution of the Lorentz Force $F = qE_z$, where $E_z = E_0 \sin(\omega t)$, accelerates particles in the direction of the beam axis. The energy gained per turn by an individual particle is given by $\Delta E_{turn} = q\hat{V}\sin(\phi - \phi_s)$, where $\phi_s = 2\pi q\rho R\dot{B}$ is the synchronous phase, ρ is the dipole bending radius and R is the physical radius of the machine. The synchronous phase is the phase at which the accelerating voltage matches the energy gain per turn, as shown by the stable fixed points P_1 and P_3 in Fig. 1.3, which all see constant acceleration.

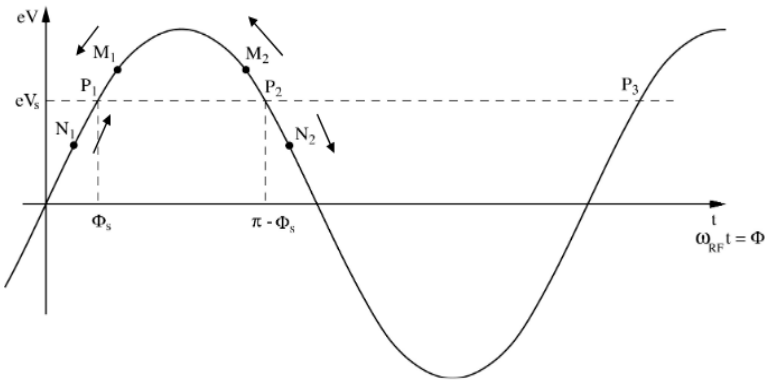


Figure 1.3: Stability conditions as a function of phase [19].

Particles with shifted momentum relative to the synchronous particle have a different orbit path length and velocity, hence a different revolution frequency and phase slip on each turn. $\eta = 1/\gamma^2 - \alpha$ gives the phase slip factor, where γ is the relativistic factor and $\alpha = \frac{\Delta L/L}{\Delta p/p}$ is the momentum compaction factor or relative change in orbit length with momentum [19]. Therefore above transition, where $\eta > 0$, the velocity gains are limited and path length dominates whilst below transition velocity gains dominate. For $\eta < 0$, low energy particles move to the tail and should be accelerated, whilst high energy particles move to the head and should be decelerated. For $\eta > 0$, the opposite is true. In both cases, particles are restored towards the stable fixed points and will oscillate in bound motion. Particles not meeting the stability conditions will arrive out of phase and

will be lost to the system in unbound motion. The longitudinal phase space representation showing the separation of bounded and unbounded motion is given by the so-called separatrix as shown in Fig. 1.4.

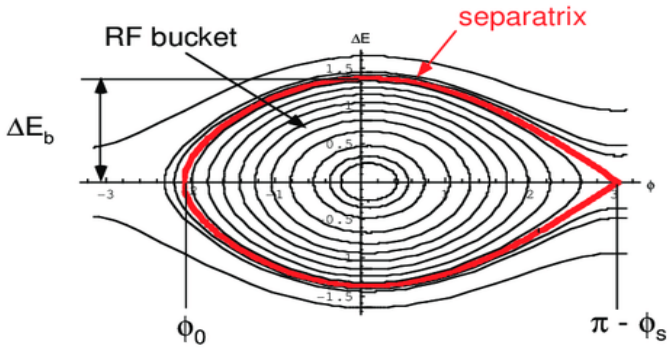


Figure 1.4: Longitudinal phase space separatrix showing the separation between bounded motion (RF bucket) and unbounded motion [20].

The RF bucket of the system is defined as the area of bound motion and is given by Eq. (1.3) [21].

$$\Delta E_{sep} = \sqrt{\frac{qV_{max}\beta^2 E_0}{\pi h \eta}[\cos\phi_s + \cos\phi + (\phi - \pi + \phi_s)\sin\phi_s]} \tag{1.3}$$

where β is the relativistic factor v/c, h is the harmonic number of the RF system, V_{max} the maximum RF voltage, E_0 the energy, ϕ the particle phase, and ϕ_s the synchronous phase. If the particle energy is at transition, i.e $1/\gamma^2 = \alpha$, $\eta = 0$ and from Eq. (1.3) $\Delta E \rightarrow \infty$ hence the particle is lost. For a given B and dB/dt, the voltage can be adapted to give the desired RF bucket and shape from Eq. (1.3).

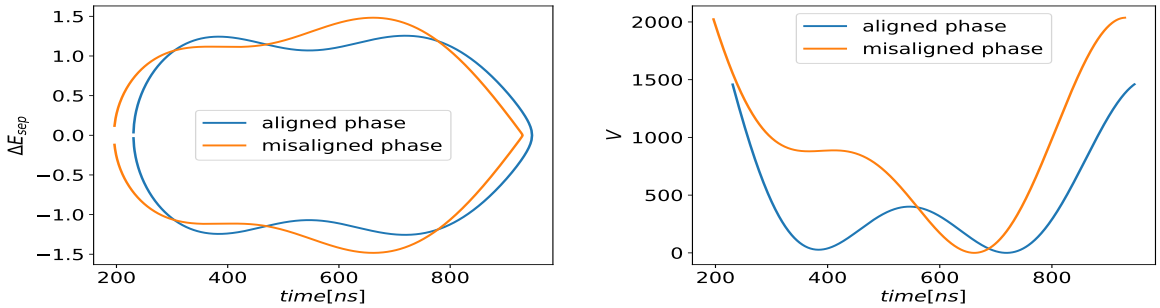
In the PSB, this is typically done using two harmonics in bunch lengthening mode which are used to increase the bucket area. If new voltage programs are simply set without making the corresponding phase corrections, phase offset errors will be induced which will result in potential well asymmetries. When changing the primary harmonic voltage from V_1 to V_2, the synchronous phase will change according to Eq. (1.4), and the phase must therefore be corrected with the phase correction given by Eq. (1.6).

$$\phi_s = \arcsin\left(\frac{\Delta E}{qV}\right) \tag{1.4}$$

$$\Delta\phi = \phi_{s_1} - \phi_{s_2} \tag{1.5}$$

$$\Delta\phi_{h=2} = \arcsin\frac{\Delta E}{qV_1} - \arcsin\frac{\Delta E}{qV_2} \tag{1.6}$$

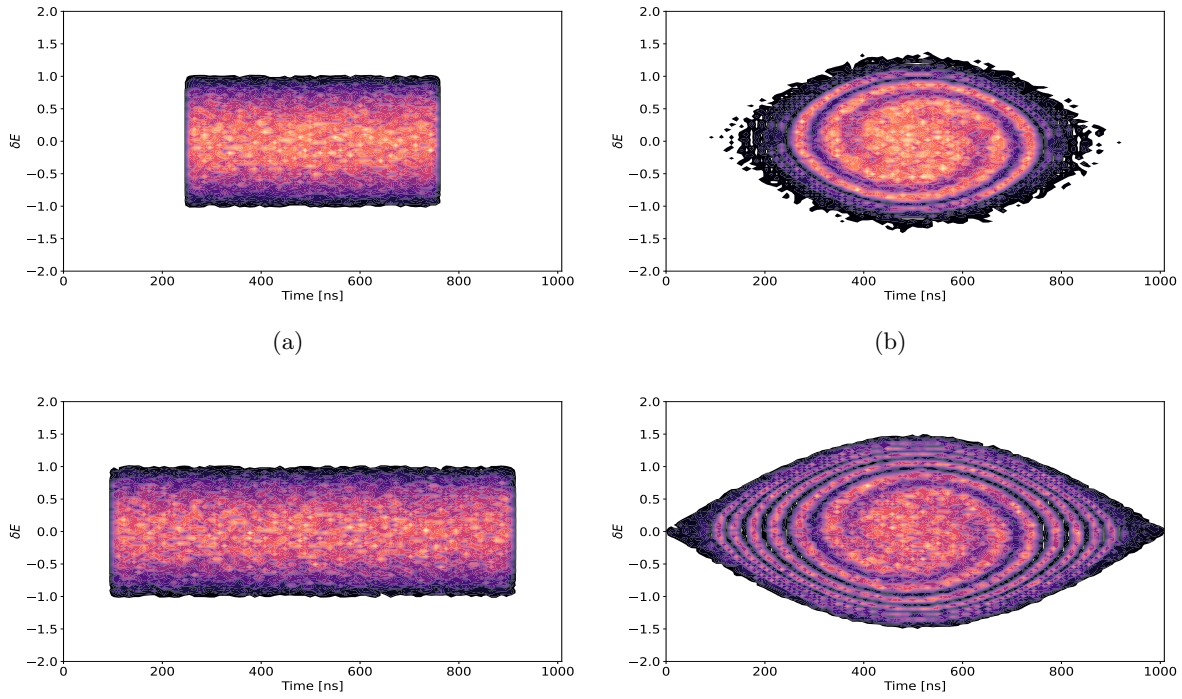
By correcting the phase offset of h=2, the asymmetry of the potential well is corrected as shown in Figs. 1.5(a) and 1.5(b).



(a) (b)

Figure 1.5: Potential wells and buckets for in-phase and out of phase harmonics 1 and 2.

A beam is matched when the distribution of particles is uniform along the iso-Hamiltonian contours in Fig. 1.4. If a beam is mismatched and there is a spread of synchrotron frequencies, there will be some level of filamentation. Filamentation occurs when a beam has an energy spread causing the bunch to decohere and fill the available acceptance. This leads to oscillations in linear density λ_{max}, as shown by the high charge density regions or filaments illustrated in Figs. 1.6(b) and 1.6(d).



(a) (b)

(c) (d)

Figure 1.6: BLonD simulation of beam profiles showing various levels of filamentation for injected beams of (a) length 400ns and (b) 800ns.

In Fig. 1.6(a), a short bunch length is injected, which in this case is better matched, leading to less filamentation in Fig. 1.6(b). In Fig. 1.6(c), a longer bunch is injected hence an increased level of filamentation and higher peak charge density in Fig. 1.6(d).

Filamentation can cause various unwanted effects, such as emittance growth and beam loss, and mitigating these is a significant challenge in the design and operation of particle accelerators.

1.2.3 Transverse Beam Dynamics

In particle accelerators, the transverse beam dynamics refer to the motion of particles perpendicular to the direction of the beam. The transverse motion of particles can be described in terms of their position and angular divergence, giving the phase space representation of the beam. The beam emittance is a measure of the area covered by the particles in the transverse phase space as illustrated in Fig. 1.7, where the area $A = \pi\epsilon$ [12].

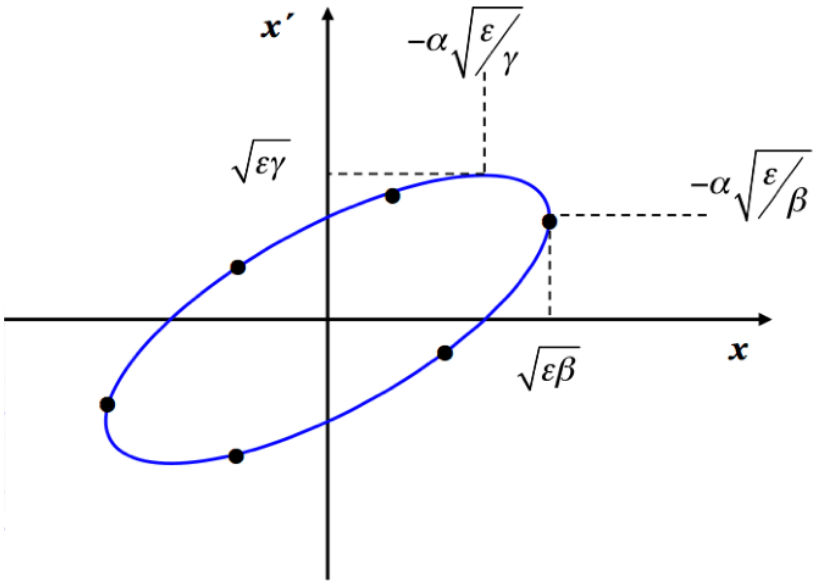


Figure 1.7: phase space ellipse representation of a particle bunch [22].

The projection of the phase space ellipse onto the x and y axis characterizes the beam width and angular divergence respectively. The position and divergence of an individual particle is governed by Hill's equation, $\frac{d^2x}{ds^2} + K(s)x = 0$, which has the solutions:

$$x(s) = \sqrt{\epsilon\beta(s)}\cos(\phi) \tag{1.7}$$

$$x'(s) = -\alpha\sqrt{\epsilon/\beta(s)}\cos(\phi) - \sqrt{\epsilon/\beta(s)}\sin(\phi) \tag{1.8}$$

where ϵ is the beam emittance, and α, β and γ are the Twiss parameters [23]. The Twiss parameters play a critical role, as they determine the size of the beam for a given emittance and define the trajectory of the beam. The Twiss parameters are set by the machine optics, and govern key beam qualities such as the betatron oscillation amplitude, the bunch rotation, and the angular divergence alongside beam size [24]. The betatron oscillation frequency, which is governed by the quadrupole focusing strengths in the ring,

is defined as the particle tune Q [25]. Since the focusing strength seen by a particle is dependent on it's momentum and magnetic rigidity $B\rho = p/q$, a momentum spread will lead to particles with different energies experiencing different focusing strengths. A spread in focusing strength Δk will therefore be seen and the particles with higher momentum will experience a reduced focusing strength. Illustrated in Fig. 1.8 is the chromatic effects as a result of momentum spread leading to different focal lengths.

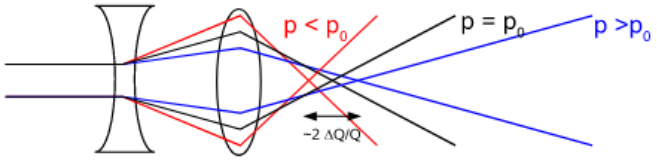


Figure 1.8: Chromatic effects leading to tune spreads due to beam mismatching and momentum spread [25].

These deviations in focusing trajectories due to momentum and focusing strength spread leads to a tune spread. The tune spread as a function of momentum spread is given by Eq. (1.9), where the chromaticity ξ is the tune's dependence momentum, showing that an increase in particle momentum leads to a reduction in tune.

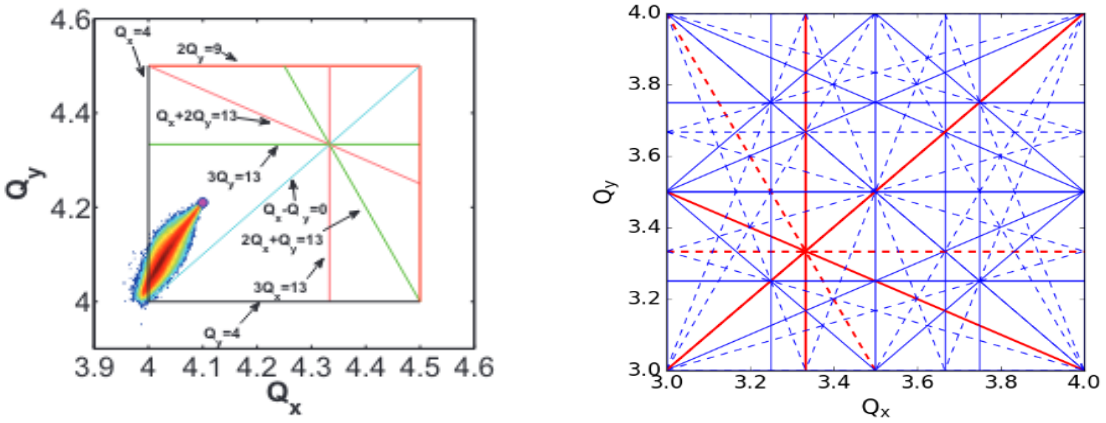
$$\Delta Q = \xi \frac{\Delta p}{p} \tag{1.9}$$

By injecting beams with large energy spreads, chromatic effects induce a tune spread which can lead to instability due to interaction of particles with resonances. The resonance condition occurs when tune shifts modify the oscillation frequency of individual particles such that their frequency matches that of an external perturbation. These resonances can act to excite the particles by giving a kick, causing repeated energy gain and eventually leading to instability and incoherence. Injecting shorter bunches with reduced energy spread would mitigate the presence of filamentation and chromatic effects, but the problem here is four-fold. An increase in the average charge density would mean space charge effects start to dominate. Space charge effects are a result of the collective interaction between charged particles [26]. This occurs when there are a high density of like-charged particles in a confined space, creating Coulombic repulsion and interaction with self fields. Space charge effects drive many performance limiting dynamics, and are one of the main limiting factors in achieving high luminosity [27]. In the presence of space charge effects, all particles will experience some level of tune shift leading to a tune footprint proportional to λ_{max} in accordance with Eqs. (1.10) and (1.11).

$$\Delta Q_x = \frac{\lambda_{max} r_p}{2\pi\beta^2\gamma^3} \oint \frac{\beta_x(s)}{\sigma_x(s)[\sigma_x(s) + \sigma_y(s)]} \tag{1.10}$$

$$\Delta Q_y = \frac{\lambda_{max} r_p}{2\pi\beta^2\gamma^3} \sqrt{\frac{1}{\epsilon_y}} \oint \frac{\beta_y(s)}{\sigma_x(s) + \sigma_y(s)} \tag{1.11}$$

with $\sigma_x(s) = \sqrt{\beta_x(s)\epsilon_x + D_x^2(s)(\frac{\Delta p}{p})^2}$ and $\sigma_y(s) = \sqrt{\beta_y(s)\epsilon_x + D_y^2(s)(\frac{\Delta p}{p})^2}$ the horizontal and vertical beam sizes. In the vertical plane where the dispersion factor D_y is practically zero, $\sigma_y(s) \approx \sqrt{\beta_y(s)\epsilon_y}$. Here, $\beta_{x,y}(s)$ correspond to the horizontal and vertical beta functions, $\epsilon_{x,y}$ the horizontal and vertical emittances, $\Delta p/p$ the rms momentum spread, r_p the proton radius, and β and γ the relativistic factors [28]. This tune footprint growth risks causing interaction of particles with PSB resonances, which are illustrated in Fig. 1.9(a), where the working point is the design tune that all particles should see if not for collective intensity effects.



(a) (b)

Figure 1.9: (a) PSB tune footprint diagram with relevant resonance lines and the lattice working point (purple dot) [29] and (b) up to 4th order resonances in the PSB [30].

While not guaranteed that a particle with a resonant tune will see excitation, it is evident that a reduced tune footprint minimizes the probability of these interactions. These resonant excitations induce higher energy spreads leading to further tune spread enlargement and potentially instability and beam loss. This is particularly problematic in cases where a large number of particles have tune values that could see resonant excitation, since a higher proportion of the beam could become unstable. Fig. 1.10 shows two cases where the λ_{max} evolution differs based on bunch length at injection.

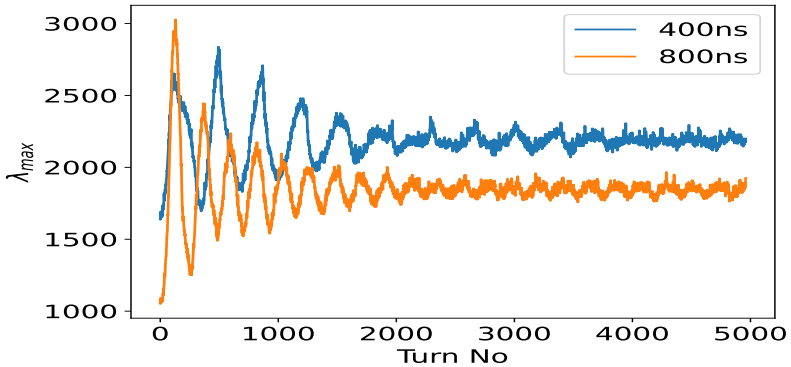


Figure 1.10: λ_{max} evolution as a function of turns for (a) 400ns injected bunch length and (b) 800ns injected bunch length.

However, it is not yet clear which leads to worse beam performance: short bunches and high *average* charge density, or longer bunches with more filamentation, chromatic tune spread, and higher *peak* charge density. Luckily, Eqs. (1.10) and (1.11) are sufficiently coupled such that a tune spread reduction in one plane should see almost the same reduction in the other. This is key, since it permits omitting the horizontal components from all emittance measurements, reducing the optimization to the vertical plane only where dispersion can be neglected since no vertical bending magnets are used.

1.3 Optimization Objectives

The primary objective is to optimize the longitudinal profiles at PSB injection to mitigate the impact of beam degrading factors. Beam degradation can be caused by a number of phenomena: high λ_{max} driving space charge tune spread, and beam energy spreads leading to chromatic tune spread, filamentation and beam loss. This presents a relatively complex, multi-objective optimization problem with multiple competing phenomena.

This project will therefore address the task of developing data-driven, algorithmic means of automating beam shaping at the PSB injection for improved beam quality. The algorithm will be responsible for changing the RF voltage programs to modulate the bucket shape given by Eq. (1.3), with performance measured in beam emittance and transmission improvements. Deep Reinforcement Learning will be investigated as a method for mimicking human-like optimization performance in live particle accelerator settings.

2 Deep Reinforcement Learning

2.1 Markov Decision Processes

Deep Reinforcement Learning is a subset of Artificial Intelligence (AI) concerned with developing algorithms to replicate human-level control behaviour in a wide range of tasks [31, 32]. This is achieved through repeated interaction with an environment in the form of taking actions, making observations and receiving rewards as illustrated in Fig. 1.11.

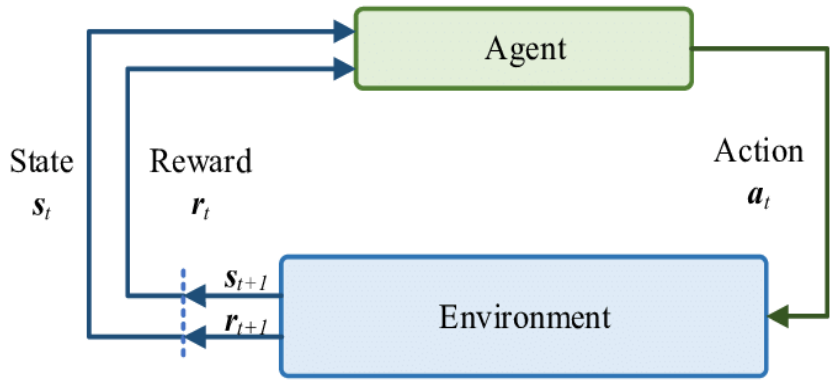


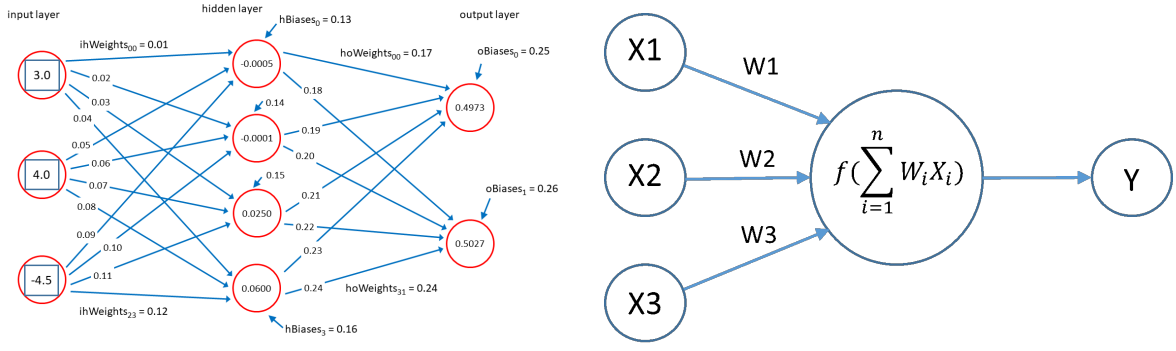
Figure 1.11: Reinforcement Learning interaction loop showing the process of an agent interacting with an environment to update the state and receive a reward. Adapted from [33].

Much of the cutting-edge research in RL is tested on video games, where the player is the agent taking actions, the environment is the game in which the player acts, and the reward can be formulated as some scoring mechanism. In this case, the actions the agent takes will be based on an observation of the current state of the environment, and their perception of what action should be taken to receive a reward.

The relationship mapping these environment states to actions is known as the policy π, which is essentially a Markov Decision Processes (MDP) used for modelling the sequential decision making processes in optimization algorithms [34]. An MDP is defined as any process which consists of a set of states $\mathcal{S}$, a set of possible actions $\mathcal{A}$, a transition policy $\pi_a(s, s')$ which is the probability of transitioning from one state s to the next state s' given an action a, and finally a reward $r_a(s, s')$ received after transitioning. Therefore an MDP is any process described by the tuple $(\mathcal{S}, \mathcal{A}, \pi_a(s, s'), r_a(s, s'))$, and is one which forms the basis of RL. An interesting feature of an MDP is the Markov property, which essentially states that the agent's decision depends *only* on the current state observation.

2.2 Deep Learning

Of course, if the policy π is already known then the problem becomes very straightforward and easy to solve. In environments governed by thousands of parameters and complex relationships such as particle accelerators, an analytical description of the system can often be too complex to describe and must instead be learned. Employing "black box" optimizers in the form of Deep Neural Networks (DNNs) [7] to learn the underlying control function therefore becomes a viable option, despite offering very little knowledge of what has been learned. The DNN learns the problem by minimizing the gradient of some cost function using the backpropagation algorithm [35, 36].



(a) (b)

Figure 1.12: Deep Neural Network showing (a) a typical network architecture [37] and (b) the computation carried out at each node.

In Fig. 1.12(a), the first (input) layer takes some input variables which are then passed through the hidden layers containing an activation function f, which computes a scalar value as a function of the inputs X_i and associated weights W_i as shown in Fig. 1.12(b). The last layer (output) returns the sum of the computations carried out at each node in the network. If the network weights are known, the calculation is straight forward and for any input we have a direct mapping from input to output.

If the network weights are not known, ground-truth output data can be used to find the weights through backpropagation. Here, inputs and initial weights are taken to compute the output which is compared against the expected value, giving an error or cost value. Then, moving backwards, the weights in each hidden layer are updated by an amount proportional to the error and the process is repeated until the error is minimized through gradient descent and the weights are known. Once weights are known, the problem is solved and memorized with the network acting as a function mapping inputs to outputs.

2.3 Policy Approximation

Similar to how a human learns a game through experience and positive or negative reinforcement when interacting with an environment, an algorithmic agent can learn via the RL interaction loop illustrated in Fig. 1.13. Here, the agent receives the state s_t and an associated value of being in that state given by the reward r_t, and must consider a distribution of the possible actions it can take to improve the reward received at the next time step. It then selects an action a_t from the distribution of possible actions to update the accelerator environment and to generate a newly updated state-reward pairing s_{t+1} and r_{t+1}, completing the interaction loop.

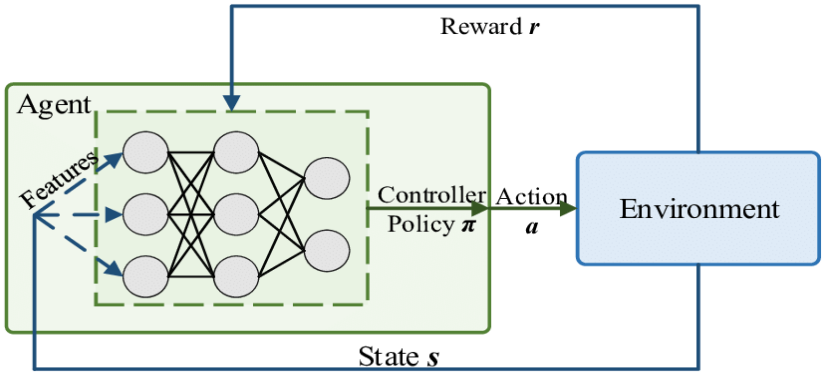


Figure 1.13: Deep Reinforcement Learning interaction loop between agent and environment. Adapted from [33].

The interaction loop is then iterated over until the policy is learned by the agent using policy search methods. Policy search methods directly optimize the policy by searching for the optimal set of parameters that maximize the expected reward, which is typically parameterized by a set of weights in a neural network. The optimization goal is to therefore search for the best set of weights that maximizes the expected reward. One common technique for policy approximation is the policy gradient method, which uses the gradient of the expected reward with respect to the policy parameters to update the policy [38]. The policy update $\Delta\theta$ is given as follows:

$$\Delta\theta = \alpha \nabla_\theta J(\theta) \tag{1.12}$$

where θ represents the policy parameters or neural network weights, α is the learning rate or step size parameter, and $J(\theta)$ is the expected reward. The policy gradient $\nabla_\theta J(\theta)$ is estimated using reward samples $R(S, A)$ collected via Monte Carlo sampling of data collected in the environment using the following:

$$\nabla_\theta J(\theta) = \nabla_\theta \, \mathbb{E}[R(S, A)] \tag{1.13}$$

But since the rewards have no direct dependence on θ, there is no way to generate samples by varying the policy parameters. Instead, the policy gradient theorem [39] given in Eq. (1.14) is used to take an expected gradient of the log of the policy, which does have dependence on θ, and which can therefore be used to generate samples. The policy update is then given by Eq. (1.15).

$$\nabla_\theta \, \mathbb{E}[R(S, A)] = \mathbb{E}[\nabla_\theta \log \pi_\theta(A|S) R(S, A)] \tag{1.14}$$

$$\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta \log \pi_{\theta_t}(A_t|S_t) \tag{1.15}$$

Hence, for any policy $\pi_\theta(a|s)$, where the goal is to find the best set of parameters θ to maximize the expected reward J(θ), policy gradients are computed and gradient ascents of the policies are taken such that the average expected reward is increasing at each time step. The policy is then updated, and iterations are repeated to ensure the average reward increases until the policy has been fully approximated, i.e when the best set of neural network weights are found, and the problem has been learned. This makes deep learning an extremely powerful tool when used to support RL, since by using DNNs, agents are capable of learning high-dimensional representations of the state and action spaces, allowing the agent to learn complex policies and save the underlying physics in memory in the form of neural network weights.

However, Eq. (1.14) only considers the transition from the current step at s_t to the next at s_{t+1} based on the instantaneous reward $r(s_t, a_t)$. This is not always ideal, since an action leading to the highest instantaneous reward might not always lead to the best trajectory of actions when considering long-term future rewards. The instantaneous reward should therefore be replaced by a long-term state-value function $Q^\pi(s, a)$, known as the Bellman function given by Eq. (1.17), to provide a means to consider future rewards when computing policy gradients in Eq. (1.16).

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)] \tag{1.16}$$

$$\text{where } Q^\pi(s, a) = \sum_{s', r} \pi_\theta(s', r|s, a)[r(s, a) + \gamma_{A'} Q^\pi(s', a')] \tag{1.17}$$

Here, the discount factor $\gamma \in [0, 1]$ defines the level of importance placed on future rewards versus immediate rewards. Lower γ acts to place less importance on future rewards by lowering the contribution of $\gamma_{a'} Q^*(s', a')$ in Eq. (1.17), while higher γ increases this importance to increase the likelihood of taking exploratory steps to avoid becoming stuck in local minima.

2.4 Policy Evaluation

For a known policy π, we can begin to evaluate the policy by assigning and providing some metric of value to the agent. This is done using a state-value function, which provides a measure of the expected return when starting in state s and following the policy π:

$$V^{\pi}(s) = \mathbb{E}[R|s, \pi] \tag{1.18}$$

where the best policy is the one which maximizes the return of the state-value function [38]. The question then becomes: given the state-value function V^{π}, what is the policy which maximizes the return? The relationship between states, actions, reward and policy are not deterministic and may have some drift in time. Therefore, when rolling out a sample trajectory to compute the expected returns, the actual return seen on each step may have some variance. How to define this value function is therefore not always clear, and in dynamic systems may have to be learned using the process in Eq. (1.19), which is similar to the policy update algorithm.

$$V(s_{t+1}) = V(s) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)) \tag{1.19}$$

This is what's known as "bootstrapping", where a function is rolled out to give a trajectory but at each step the function is re-evaluated and compared to the initial prediction, to then provide an update to the initial function [40]. For RL, it essentially functions as reward prediction since once learned, the state-value function will be able to predict the total reward return from a trajectory of states given an initial state and policy. This is what's known as Temporal Difference (TD) Learning, which seeks to minimize the error between predicted reward and actual reward seen [41]. The updated interaction loop when combining both the policy network (actor) and a value network (critic) to create a single actor-critic agent is illustrated in Fig. 1.14.

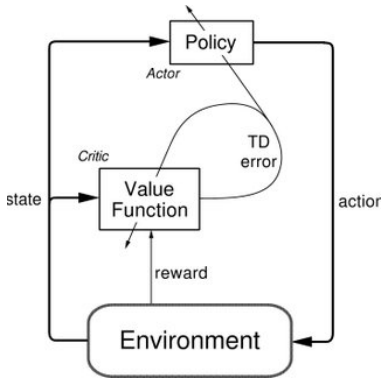


Figure 1.14: Actor-critic reinforcement learning interaction loop.

While the actor still takes actions on the environment to receive an updated state, it no longer sees the reward associated with being in that state. Instead, the critic network is introduced and receives both the current state and associated reward which can be used to compute a TD error, which is then used to provide feedback to the actor at each time step. Considering the role of each network: there is an actor concerned with approximating the policy π_θ to determine the best action to take given the current state, and a

critic which provides feedback on the long-term trajectory or performance of the actor. The idea then is that by exposing both the actor and critic to more data, they will act in tandem to learn the underlying control problem which will be stored in the form of weights in each neural network.

Once the problem is learned and memorized, a direct mapping from state inputs to the actions needed to maximize the reward signal is fully described and can be deployed in future to provide optimization with fast inference times. Relating this to the PSB injection optimization problem - at inference, the agent would be provided with a state returned by measuring the current transverse emittance and transmission values for the given machine configuration. Using this it would be able to adjust the voltage programs of the system to find the optimal cutoff between transverse emittance and transmission to improve luminosity with minimal human intervention.

2.5 Continuous Models

There are a small handful of state-of-the-art models commonly used for problems with continuous action spaces, with the 2 best performers typically being the Soft-Actor-Critic (SAC) [42] and Twin Delayed DDPG (TD3) [43]. SAC is an off-policy, maximum entropy stochastic actor-critic algorithm which is known for having improved sample efficiency compared with it's competitors.

The SAC algorithm seeks to maximize a combination of expected reward (i.e., the expected return over time) and the entropy of the policy. The entropy term encourages exploration and helps prevent the policy from getting stuck in a suboptimal local maximum. By maximizing both reward and entropy, the algorithm is said to be "soft" as it allows for a trade-off between exploration and exploitation. By incorporating the entropy regularization term, the SAC algorithm has been shown to be effective for continuous control problems, where the policy space is large and continuous.

The entropy factor between 0 and 1 essentially determines the level of randomness of the actor when selecting the best action. Considering a scenario where an actor is faced with a choice of actions with extremely similar expected value, there are cases where the best action at the current time step may not be the best long term action. With an entropy factor of 0, the actor chooses the next best action with confidence that it will be the action leading to the best immediate reward. However, there is a level of uncertainty surrounding this decision and there is a non-zero probability one of the other actions may have lead to better long-term reward. The entropy factor can therefore be used to encourage some level of randomness when choosing actions, such that the agent can enter exploration periods to explore the action space fully.

TD3 is a model-free, off-policy reinforcement learning algorithm that extends the Deep Deterministic Policy Gradient (DDPG) algorithm [44]. It was proposed as an improvement to the DDPG algorithm to address some of its limitations, such as sensitivity to hyperparameters, overestimation of Q-values, and instability during training. The TD3 algorithm has several benefits over the SAC algorithm. One advantage is that it is relatively simple to implement and has fewer hyperparameters to tune. The TD3 algorithm also includes several modifications to the DDPG algorithm that help reduce overestima-

tion of Q-values, making it more stable during training.

On the other hand, the SAC algorithm has some benefits over the TD3 algorithm as well. For example, the SAC algorithm can be more sample-efficient and can explore the environment more effectively. This is because the SAC algorithm includes an entropy term in the objective function, which encourages policy exploration. The choice between TD3 and SAC depends on the specific requirements and the nature of the problem being solved. TD3 is a good choice when the focus is on simplicity and stability, while SAC is a good choice when sample efficiency and exploration are critical.

3 Methodology

3.1 Agent Development

The DNNs used for both the actor and critic network were implemented using the PyTorch-based Stable-Baselines3 (SB3) [45], which is a Python package containing the algorithms needed for RL problems. SAC was chosen due to it's sample efficient performance. An example of how to construct a SAC model in Python is shown in the code snippet below.

```
import numpy as np
from stable_baselines3 import SAC
from stable_baselines3.common.noise import NormalActionNoise
from environments.optimization_environment import RFOptimization

environment = RFOptimization(max_steps_per_episode=25)
n_actions = environment.action_space.shape[-1]
net_architecture = dict(net_arch=[64, 64])
action_noise = NormalActionNoise(mean=np.zeros(n_actions), sigma=0.01 *
↪  np.ones(n_actions))

model = SAC(
        'MlpPolicy',
        environment,
        gamma=0.99,
        learning_rate=3e-4,
        tau=0.005,
        buffer_size=50000,
        ent_coef='auto',
        policy_kwargs=net_architecture)

model.learn(total_timesteps=1e5, tb_log_name=f"sac_log", log_interval=5)
```

Here, the model and it's hyper-parameters along with the policy type, DNN architecture, training lengths and any additional functionality via callbacks are defined. Due to the

tabular nature of the data, an Mlp (multi-layer perceptron) policy was used [35]. The hyper-parameters `gamma`, `learning_rate`, `tau`, `buffer_size` and `ent_coef`, along with the `net_architecture` are crucial for agent performance. Defining the hyper-parameters is in itself a significant optimization problem, so the work done in [42] was used as a reference point.

3.2 OpenAI Gym Framework

The agent then needs an environment in which it can take actions to update the state. This interaction interface was implemented using a standard OpenAI Gym environment [46], which provides a general framework for implementing RL problems. The way in which the agent interacts with the particle accelerator is defined in the environment based on problem specifics, and the rest of the heavy lifting is taken care of by SB3/OpenAI under the hood.

The OpenAI Gym environment class contains the key methods for updating the system and computing the necessary metrics. First, the `action_space` is implemented to define the domain of allowed actions by the agent, typically normalized to an action $a_n \in$ [-1, 1] for each degree of freedom. Next, the `observation_space` is set to define the boundary of observations given by $T \in [0, 1]$ and the beam emittance $\epsilon \in [1.0, 1.7]$. Next, the methods `get_initial_params` which measures the initial state, `get_reward` which computes the reward, and `reset` which resets the training episode are defined. `step` then takes actions as input and these actions are processed to update the environment, calling on the other methods and returning the observation and reward to the agent.

3.3 Updating the Environment

Using `step`, the environment receives actions from the agent to update the state via the PyJapc API, which is a Pythonic implementation of the Java API for Parameter Control (JAPC), providing an API to control device parameters. The agent actions are in the form of deltas used to update the RF voltage programs, illustrated in Fig. 1.15, to change the bucket shape.

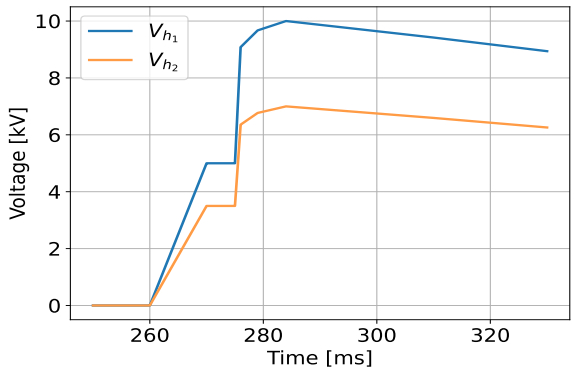
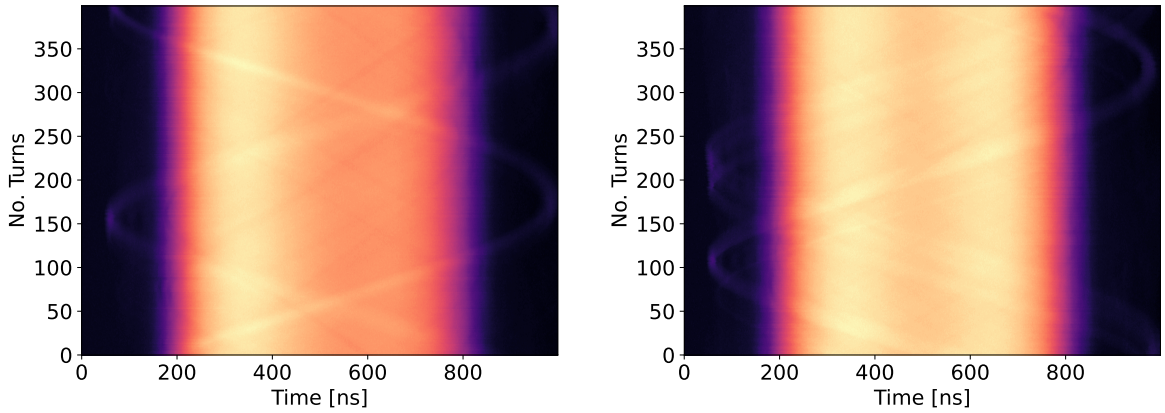


Figure 1.15: Example of a typical voltage function for the primary and secondary harmonic voltages.

The actions returned by the agent are in the range [-1, 1], which are then multiplied by a scaling factor of 0.05 and are used to give a percentage increase/decrease on the voltage

where $\Delta V \in [-5\%, +5\%]$. Using the phase correction in Eq. (1.6), phase offsets such as the one shown in Fig. 1.16(a) are corrected to maintain beam symmetry after changing voltages. This ensures the particles are distributed evenly amongst the two potential wells, reducing the λ_{max} and any space charge effects in any single bucket.



(a) (b)

Figure 1.16: (a) Phase offset leading to beam asymmetries and increased density in h=1 (left portion) and (b) corrected phase leading to a symmetric beam.

Observations are then made in the form of transverse emittance and beam transmission measurements and the reward is calculated as a function of both. The transverse emittance is calculated using Eq. (1.7), where for a known β and beam size σ the emittance can be computed. The beam size is measured using the scanning electron miscroscope grid (SEM grid) devices [47]. The SEM grid consists of a grid of thin metal wires typically with a size of a few microns which are inserted into the beam to create non-destructive secondary electron emission (SEE). SEE creates an image of the beam profile which can be analyzed to determine the beam size and therefore the emittance.

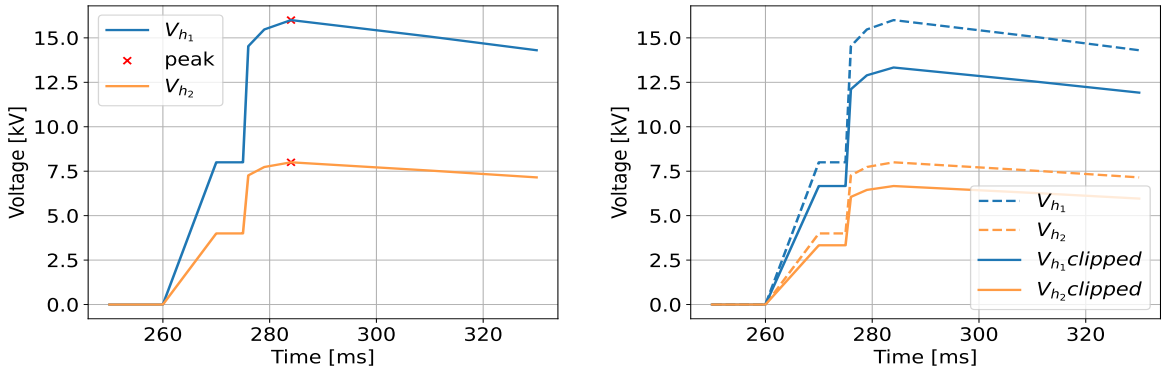
Beam intensity measurements are given by the Beam Loss Monitoring (BLM) system [48]. BLMs are placed in strategic positions around the accelerator, and consist of a set of metal plates used to monitor current to provide a real-time measurement of the beam intensity. The beam transmission T is then computed as the ratio of intensity at extraction I_E vs the injected beam intensity I_I, hence $T = I_E/I_I$.

3.4 Machine Safety Constraints

It is crucial to perform safety checks to ensure any potentially damaging environment updates are dealt with correctly. The individual voltage programs can therefore not exceed 20kV at any point during the cycle, whilst the sum of both voltages at any point in the cycle must also not exceed 20kV. To maintain acceleration, both the primary and secondary harmonic voltages $V_{h=1}$ and $V_{h=2}$ must not fall below lower bounds. A set of boundary conditions are introduced giving $5kV \leq V_{sum} \leq 20kV$ and $0 < V_{ratio} \leq 0.5$ where V_{sum} is the voltage sum at the peak and V_{ratio} is given by the ratio of V_{h_2}/V_{sum} at the peak.

To enforce the constraints, each voltage function is simplified to a single voltage point taken at the ramp peak as shown in Fig. 1.17(a), where a check is carried out to ensure this

point lies within the boundary. If the the new V_{sum} violates the boundary conditions, the V_{sum} is clipped and V_{ratio} is used to compute new ramp peak positions for each harmonic. The remaining voltage points for $V_{h=1}$ and $V_{h=2}$ are then reconstructed, ensuring no point exceeds the peak point. An example of a clipped voltage is illustrated in Fig. 1.17(b), where the clipping function essentially shifts the whole voltage program downwards.



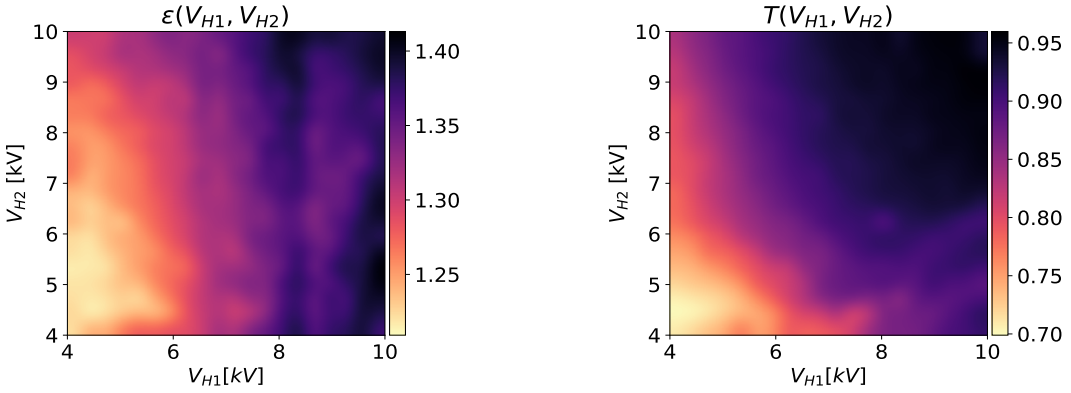
(a) Illegal voltage

(b) Clipped voltage

Figure 1.17: (a) voltage boundary violation and (b) clipping function shifts the voltage programs to be within the boundary.

3.5 Reward Function

Defining the objective function is typically the main challenge in any optimization problem, and should be designed based on the specifics of the underlying problem. To gain a better understanding of the shape of the parameter space and to inform reward function design, a parametric scan was carried out and used to determine the convexity of the problem. The results of the scan on a relatively coarse 10x10 grid are shown in Fig. 1.18.



(a) Emittance

(b) Transmission

Figure 1.18: Parametric scan results for emittance and transmission.

There is a clear correlation between low transmission and low emittance with the two scans tracing each other closely. The aim of reward design is therefore to identify solutions where this does not hold true. The base reward function takes the ratio of the two scans

as indicated in Eq. (1.20), which is also roughly proportional to the luminosity given by Eq. (1.2).

$$R(\epsilon, T) \propto \left(\frac{T}{\epsilon}\right)^2 \tag{1.20}$$

Taking this as a baseline reward function, extra constraints are then placed on the observations made by setting key performance thresholds. The reward thresholds were determined based on the reference configuration values, where $\epsilon_0 = (1.45 \pm 0.02)\mu m$ and $T_0 = (93.97 \pm 0.29)\%$. The following reward function conditions are set, where $r(\epsilon, T)$ is the baseline reward function, α is a positive scaling factor > 1, and S_c is a clipped state.

$$r'(\epsilon, T) = \begin{cases} -1 & \text{if } s_t = S_c \\ 0 & \text{if } T < 85\% \text{ and } \epsilon > 1.47 \\ \alpha r(\epsilon, T) & \text{if } T \geq 92\% \text{ and } \epsilon \leq 1.40 \\ r(\epsilon, T) & \text{if } 85\% \leq T \leq 92\% \text{ and } 1.40 \leq \epsilon \leq 1.47 \end{cases}$$

This allows the agent to see some level of continuous reward for acceptable solutions, no reward for poor solutions, a negative penalty for illegal actions, and finally a boosted reward if the solution outperforms the desired thresholds. The final reward scheme is illustrated in Fig. 1.19.

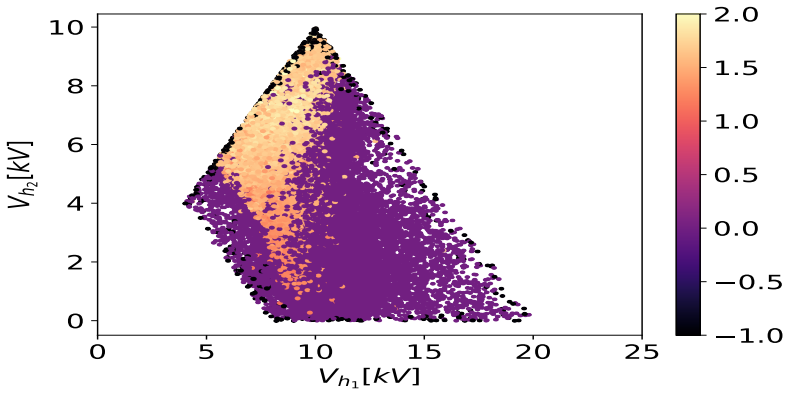


Figure 1.19: Reward plot as a result of reward shaping, take through exploration over 35k steps.

3.6 Reward Saturation

While clipping voltages seems straightforward, how to deal with reward returns for clipped states is difficult due to saturation around the boundary. There were instances where the agent would execute an illegal action and be clipped, receiving a reward based on the observation *after* clipping since no out of bounds observation took place. This can lead to positive reward feedback despite an illegal action taking place. If a negative penalty is applied instead, a good-state negative-reward pairing is created which leads to confusion. For the exact same state observation, the agent could see either a penalty or a reward depending on how it arrived at the given state - either through an illegal and clipped action or a perfectly legal action. This reward saturation stunted the agent's learning

process, and the agent quite often became stuck taking illegal actions, which it perceived to be good actions, at the boundary.

This was negated by introducing a"dummy state", where the illegal update is not sent to the machine and the observation return can be some fake, arbitrary state. The dummy state was defined to contain the upper limit of transverse emittance observations and the lower limit on transmission, giving the dummy state $S_d = [1.7, 0]$. Now the agent perceives any illegal action as an action leading to a maximum emittance return with transmission of 0%. This allows the agent to freely explore outside of the boundary without ever sending damaging settings to the machine, and with mitigated reward saturation the agent was able to quickly learn where the boundary lay. Furthermore, this approach significantly reduces the training time required for online observations since there is no real interaction with the accelerator during out of bounds updates.

3.7 Surrogate Environment

To mitigate the impact of having no beam time during the technical shutdown, a combination of online and offline training methods were employed. Due to the complexities of simulating space charge effects, it was not possible to simulate the beam dynamics in a reasonable time frame using available tools. Instead, online data campaigns were conducted to gather real data which was used to interpolate and create a new, offline surrogate environment. Using SciPy's `LinearNDInterpolator`, piecewise linear interpolation of the dataset was used to convert the sparse data in Fig. 1.19 to the fine grid shown in Fig. 1.20.

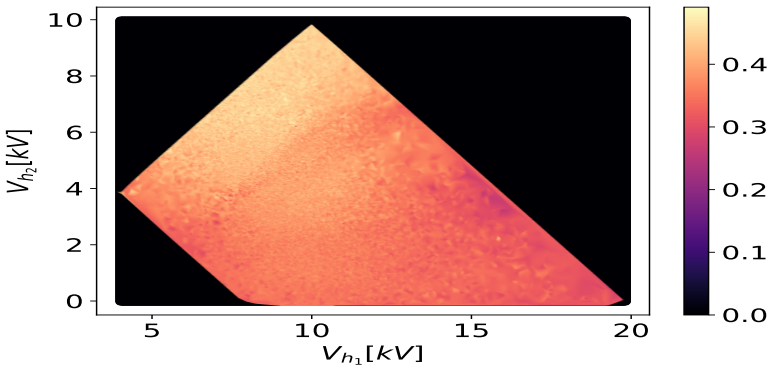


Figure 1.20: Offline surrogate model of the accelerator environment where each element on the grid is given by the baseline reward signal $R(\epsilon, T) = (T/\epsilon)^2$.

The new environment approximation replaced the online accelerator during the shutdown period so that the agent could continue to be trained offline. The new environment is finite and therefore not continuous whilst the agent has a continuous action space output. Hence a nearest neighbor search is taken on the grid and the reward at this location is returned.

3.8 Accelerator Interaction Loop

Finally, Fig. 1.21 illustrates the RL interaction loop when applied in an accelerator physics context. It demonstrates how the agent takes action a_t in the form of voltage deltas to update the RF voltage program via PyJapc. It then receives a state update S_{t+1} in the

form of SEMgrid (emittance) and BLM (transmission) measurements which are then used to compute a scalar reward based on the reward function. The loop is then repeated iteratively, with policy parameters of the networks being updated until the agent reaches stability.

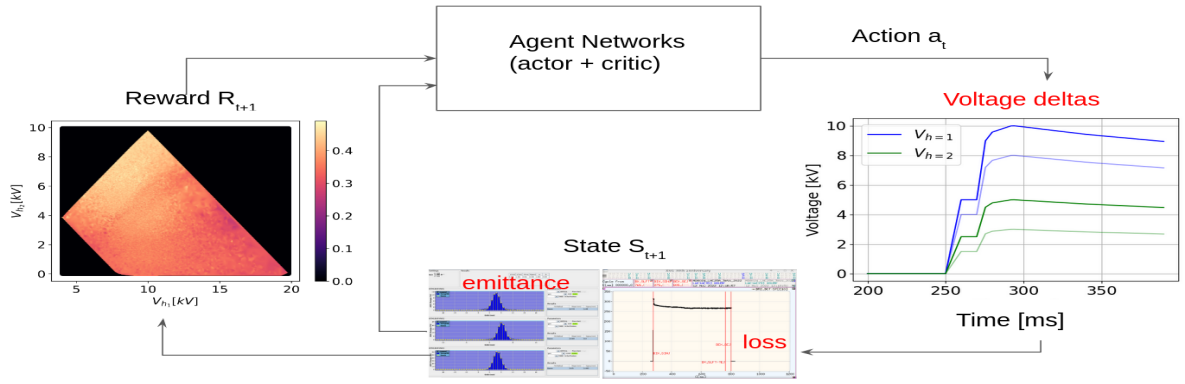


Figure 1.21: The RL interaction loop applied in a real setting.

4 Results

4.1 Online Training

An initial training run of 35k steps yielded promising results and indicated the control problem is solvable by the agent. In Fig. 1.22, any solution where both thresholds were at least matched is shown. One solution with an emittance reduction of 4% was found while increasing transmission by 2%, showing that better solutions than the reference configuration exist.

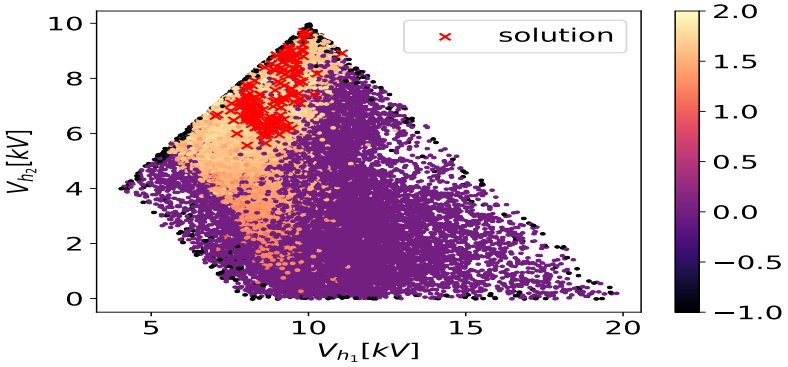
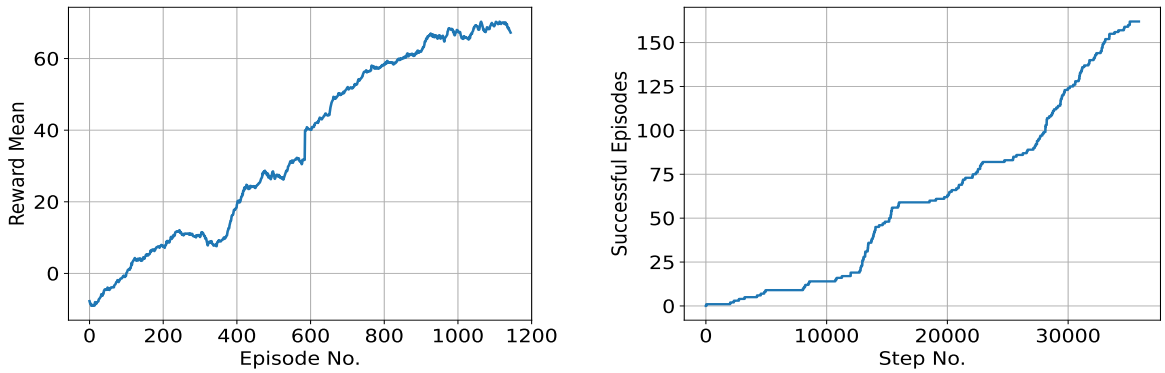


Figure 1.22: reward received during agent exploration in training over 35k steps.

Fig. 1.23(a) shows that the mean reward per episode is increasing indicating the agent is progressively finding better solutions with increased exposure to data. In Fig. 1.23(b), the rate of solving an episode appears to increase with time, indicating that the agent is not only finding better solutions but it does so with increasing frequency. This is especially apparent after around 10k steps where the rate of finding solutions begins to accelerate.

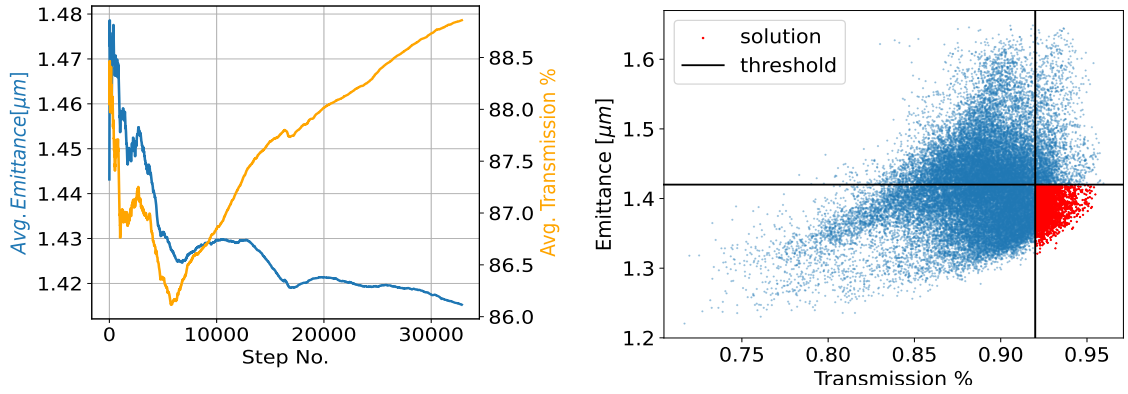


(a) Episode Reward Mean

(b) Cumulative No. of Successful Episodes

Figure 1.23: Raw training metrics indicating the agent's learning progress through episode reward means and number of times the problem is solved.

The increasing reward mean is reflected in the beam performance metrics as shown in Fig. 1.24(a). While the agent understands the primary objective of minimizing the transverse emittance, it does so at the expense of transmission with an almost identical trajectory initially. This is not surprising, since by reducing the beam intensity, space charge effects and therefore transverse emittance is reduced. After roughly 8-9k steps, the two trajectories begin to diverge and the agent begins identifying solutions where emittance is reduced whilst transmission is increased.



(a)

(b)

Figure 1.24: (a) Shows the average transverse emittance and transmission trend across the whole training period, and (b) shows the individual solutions, where a solution is defined as any sample outperforming both thresholds.

4.2 Offline Training

An agent was trained on an offline surrogate model of the accelerator environment. Fig. 1.25 shows the reward distribution seen by the agent during offline training and demonstrates a number of local minima spread throughout a particular area in parameter space.

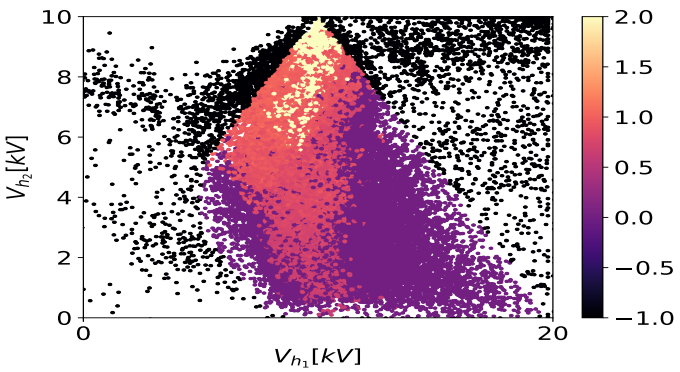
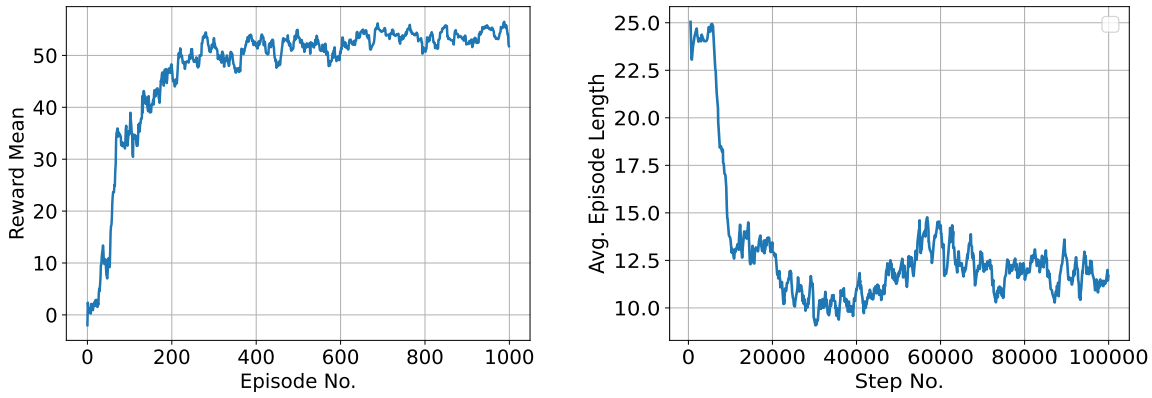


Figure 1.25: Shows the actual reward received by the agent during training and with reward shaping.

Illegal actions are recorded and included here to illustrate the extent of illegal actions taken over 100k steps, demonstrating how time expensive and potentially damaging these steps would be if not treated with dummy states. It was noted during training that the rate of attempting such actions reduces to almost zero once the problem is sufficiently learned. The gain in reward mean in Fig. 1.26(a) appears to slow at around 80k steps, indicating the problem is fully learned by the agent some time around 80-100k steps. Fig. 1.26(b) shows that as the problem is learned, the agent not only finds better overall solutions but it becomes more efficient at finding such solutions. This is reflected in the average episode length, which reduces to an average of around 13 steps per episode.



(a) Episode reward mean

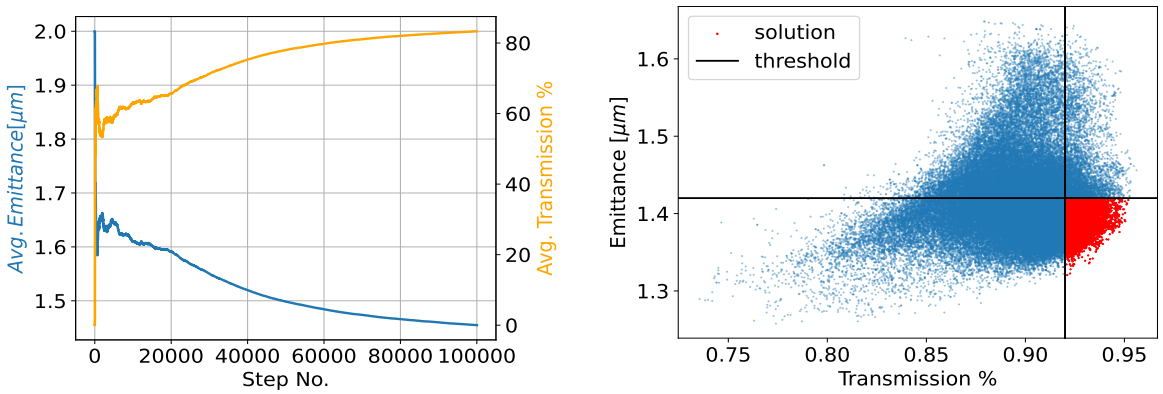
(b) Episode length mean

Figure 1.26: Raw training metrics indicating the agent's learning progress through episode reward means and number of times the problem is solved.

Again, similar to the online training, we see that the offline training in terms of beam performance metrics are positive as illustrated in Figs. 1.27(a) and 1.27(b), where there is a continuously reducing average emittance with increasing average transmission.

4.3 Agent Deployment

To begin optimizing in the offline environment, the agent is switched to behave in a deterministic manner such that it always takes the best perceived next action with 100%



(a) (b)

Figure 1.27: (a) Average transverse emittance and transmission trend across the whole offline training period. (b) individual solutions, where a solution is defined as any sample outperforming both thresholds.

probability and no longer looks to take exploratory actions. The agent was then deployed and performance measured over 100 optimization episodes with a maximum episode length of 50 steps, and where the initial state of each episode is completely random. Of the 100 optimization attempts in Fig. 1.28, the agent successfully solved the problem within 50 steps with a success rate of 100%.

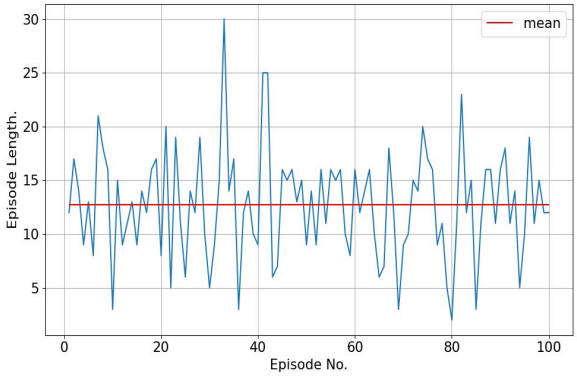


Figure 1.28: Episode length of each of the optimization episodes, showing the agent solved all episodes, with an average of 13 steps per episode.

Fig. 1.28 shows the episode length for the 100 episodes, showing that on average the agent will take 13 steps to solve the optimization problem starting from any given initial state. The longest recorded episode consisted of 30 steps, whilst a few episodes took less than 5 steps. The individual length of an episode depends on the initial conditions of the episode, so there is a chance that some episodes started with a state already close to a solution. In any case, optimization in around 13 steps is already very good and comes without encouraging the agent to perform fast inference, which could be achieved by adding penalties for time spent in episodes. An example of an optimization episode is given in Fig. 1.29.

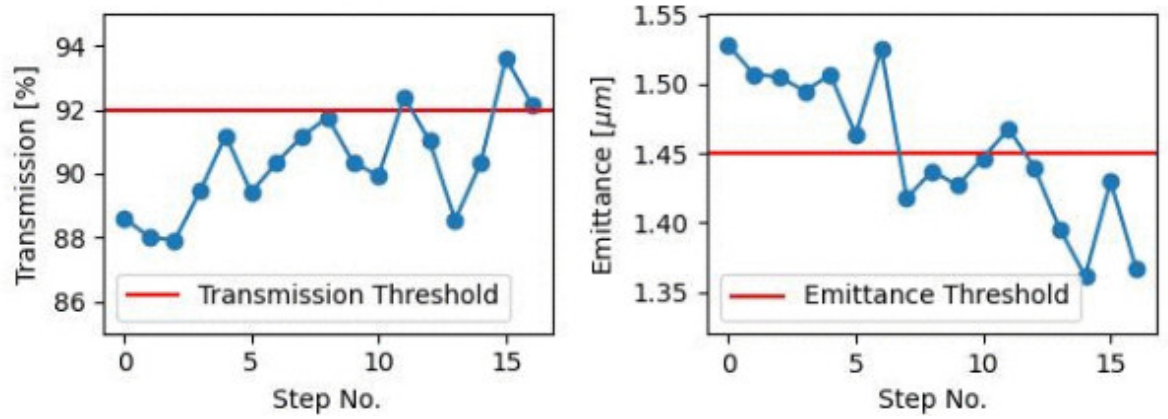


Figure 1.29: Optimization episode where the agent is used to correct a poor state in the surrogate model of the environment.

Here, the initial state is one with an emittance of $1.53\mu m$ and a transmission of 89%. This is a relatively poor initial state, but the agent solves the episode to give an emittance reduction of roughly 11% while improving transmission by 3% within 16 steps. Any episode is considered solved when both thresholds are met, which at this point can be set depending on desired performance targets.

5 Discussion

5.1 General Discussion

5.1.1 Performance

The performance of the agent demonstrates a positive trend when subjected to online training, showing that over time there is an increasing number of good solutions found for both the transverse emittance and the beam transmission. In the best case scenario, the agent found solutions with roughly 4% emittance reduction with 1-2% increases in beam transmission relative to the reference configuration, indicating solutions better than the reference exist and that the agent is capable of finding them. The agent also begins finding solutions with increasing frequency and performance with exposure to more data, but the full extent was not seen due to the technical shutdown for maintenance.

While it is not yet possible to deploy a trained agent on the live accelerator to gauge performance, an agent was trained and deployed on an offline, surrogate model where it showed great promise in consistently solving the control problem with an average of 13 steps. One example included an episode length of 16 steps where the agent corrected a very poor state, giving an 11% emittance and 3% transmission improvement. This is a clear indication of the potential strengths and fast optimization times achievable by an agent when fully trained. If similar performance could be seen online, RL could be considered a very strong option for automated optimization of longitudinal beam profiles at injection.

5.1.2 Limitations

Despite the promise shown by RL for live optimization, there are several limitations to be addressed. One of the main limitations of RL in the current form is low sample efficiency, which is especially problematic in particle accelerators with relatively low repetition rates. The full extent of optimization was not seen during training, hence exposing the agent to more data will allow the full potential to be realised. Predicting the amount of data required to fully train the agent to stability is difficult, but the results of offline training may allow an estimated to be inferred. The offline agent demonstrated sufficient learning after approximately 80k steps, indicating a potential lower bound to reach stability. Assuming at least one shot of the beam approximately every 30 seconds, it would require an upper limit of one month to collect 80k samples, making online training very time-consuming. A question over the agent's long term performance in dynamic environments also remains unanswered. The current training environment is practically fixed for the current training period but, over longer time scales, machine configuration drifts make the environment dynamic. The agent could therefore need frequent re-training to compensate for this drift.

Another limitation is seen when tuning the algorithm hyper-parameters. Typically, several agents with various hyper-parameter configurations would be trained in parallel before selecting the agent that yields optimal performance. However, multi-agent methods are not feasible as there is only one PSB, and sequential deployment would be excessively time consuming. Given the time-intensive nature of training an agent, determining the optimal configuration is not possible and it is not possible to know how optimal a trained agent is.

5.1.3 Mitigation

In simulation, training an agent can be performed for as long as required without incurring the same expenses as online training. However, due to the complexity of simulating space charge effects, acquiring live data is less time-expensive than simulated data. An offline surrogate model was used to mitigate the impact of low sample availability, reducing the training times and costs. The surrogate model consists of a mixture of real data and offline data generated through n-dimensional linear interpolation of live data. The online agent required around one week of training on a training effort with 35k samples, but with the surrogate model it was possible to train an agent over 100k steps in just a few minutes.

The validity of the surrogate model is difficult to gauge since it has yet to be tested in a live setting. The offline agent is also trained on a fixed deterministic grid, which does not fully represent the true accelerator environment. Furthermore, using a continuous output agent on a discrete grid necessitated the use of a nearest neighbor search, introducing more uncertainty. It is very possible there is a significant amount of error in the surrogate model and that the true underlying physics of the problem is lost. However, if the approximation is good, the time and cost limitations of online training can be significantly reduced by training and hyper-parameter tuning offline.

5.2 Further Studies

5.2.1 Model Validity

Steps should be taken to determine the validity of the surrogate model by applying the offline-trained agent in a live setting. If valid, the method of mixed online and offline training can be considered for application to other beam objectives in future. A study into the required ratio of online to offline data could be done to find the minimum amount of real data needed to train a successful agent. If the current model is not valid, the agent may have learned enough of the underlying problem to be used in further training efforts where the current knowledge learned serves as a baseline intuition for learning the remainder of the true problem.

5.2.2 Input and Observation Data

Further work should also be done to provide the agent with more observational measurements to help facilitate learning through capitalizing on all available data, rather than being limited to a small subsection of observational data. The current observations are limited to tabular data in the form of only emittance and transmission measurements. However, using the data given by the tomography applications would allow the agent to benefit from images of the beam as illustrated in Figs. 1.16(a) and 1.16(b). These images are encoded with significantly more data than is currently being used, hence this data should be incorporated in the agent's learning. Feature extraction, i.e where one identifies key features from image data, could be done using simple data analysis techniques or by using computer vision methods such as Convolutional Neural Networks (CNNs) [49]. The advantage of using CNNs is that they can uncover extremely complex relationships in image data at the cost of complexity and extra data requirements. Setting up analysis routines to extract new observation features may therefore be more feasible.

Methods to change parameters such as bunch length and intensity have been developed during the shutdown and have been made available through PyJapc. Future studies will look at incorporating bunch length, intensity and energy spread as possible input parameters along with bucket shape and size changes, providing a more holistic approach to optimization. Including more degrees of freedom when making changes to the voltage programs should also be considered to allow more exploration when shaping the bucket.

5.2.3 Dynamic Environments

It is not yet clear how robust any trained agent will be to machine configuration drift. This is a known drawback of using RL in online environments, with research into the mitigation of the negative impacts of RL in dynamic environments an active research field. Aims of the project could therefore look at determining how capable an agent is in dealing with machine drift. This could be done by simulating or inducing this drift during training, which would involve identifying the key devices and parameters responsible for significantly impacting agent performance through drift. Finally, it should be investigated how applicable an agent trained on one beam specification is to other, similar beam specifications. If enough information can be learned on a single machine configuration to offer optimization in other environments, it may be possible to have a master agent capable of optimization in multiple beam environments.

6 Conclusion

This thesis presents studies into the development of algorithmic methods for automated optimization in the accelerator control room at CERN. In particular, it focuses on the optimization of longitudinal beam profiles at PSB beam injection for improved transverse beam dynamics and luminosity for LHC-type beam specifications. An RL agent was trained on the online accelerator with the goal of reducing the beam emittance while increasing transmission, and demonstrated promising results. The best solution found during training returned a 4% improvement in transverse emittance while increasing transmission levels by 2% relative to the reference configuration. This is an early indication that there are better solutions than the current reference configuration and that further exploration and training should be carried out to realize the full extent.

Data collected during online training was used to develop a surrogate model of the environment in a bid to mitigate the cost of high data sample requirements. The agent was trained over 100k steps on the offline environment and demonstrated stable performance after around 80k steps. When deployed in the offline environment, the agent was able to consistently solve the optimization problem with an average of 13 steps given any random initial state, indicating the effectiveness and potential fast optimization times of a fully trained agent. An example episode offered an 11% emittance and 3% transmission improvement from a relatively poor initial machine state in 16 steps. However, it was not possible to evaluate the performance of the model in a live setting due to the technical shutdown, and so it is not clear how the agent would perform when deployed online.

There are some concerns due to the volume of data required for the agent to learn the surrogate problem. If the offline data requirements for stable performance are to be taken as an estimate for online requirements, the feasibility of training an agent to stability in a reasonable timeframe is in question. Ongoing studies will look at capitalizing on all available data streams while improving problem description to facilitate good learning, and will investigate agent performance in dynamic environments. If the limitations in sample efficiency can be mitigated, RL looks very promising and with further scientific maturity and research, could be a strong tool for optimization and control in accelerator operations.

Bibliography

[1] URL: https://cern.ch/.

[2] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038/nature14236.

[3] Jonas Degrave et al. "Magnetic control of tokamak plasmas through deep reinforcement learning". *Nature* 602.7897 (Feb. 2022), pp. 414–419. ISSN: 1476-4687. DOI: 10.1038/s41586-021-04301-9. URL: https://doi.org/10.1038/s41586-021-04301-9.

[4] Verena Kain et al. "Sample-efficient reinforcement learning for CERN accelerator control". *Physical Review Special Topics - Accelerators and Beams* (2020).

[5] Nico Madysa and Verena Kain. "Automated intensity optimisation using reinforcement learning at LEIR". *13th Int. Particle Acc. Conf* (2022).

[6] Jason St. John et al. "Real-time artificial intelligence for accelerator control: A study at the Fermilab Booster". *Phys. Rev. Accel. Beams* (2021).

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[8] Kai Arulkumaran et al. "A Brief Survey of Deep Reinforcement Learning". *IEEE Signal Processing Magazine* 34 (Aug. 2017). DOI: 10.1109/MSP.2017.2743240.

[9] Corinne Pralavorio. "A new acceleration system for the PS Booster" (2017). URL: https://cds.cern.ch/record/2276508.

[10] W.J.M. Weterings et al. "The New Injection Region of the CERN PS Booster". *Proc. 10th International Particle Accelerator Conference (IPAC'19), Melbourne, Australia, 19-24 May 2019* (Melbourne, Australia). International Particle Accelerator Conference 10. Geneva, Switzerland: JACoW Publishing, June 2019, pp. 2414–2417. ISBN: 978-3-95450-208-0. DOI: doi:10.18429/JACoW-IPAC2019-WEPMP039. URL: http://jacow.org/ipac2019/papers/wepmp039.pdf.

[11] A Sørensen. "Liouville's theorem and emittance" (1989). DOI: 10.5170/CERN-1989-005.18. URL: https://cds.cern.ch/record/367259.

[12] Oscar R. Sander. "Transverse emittance: Its definition, applications, and measurement". *AIP Conference Proceedings* 212.1 (1990), pp. 127–155. DOI: 10.1063/1.39706. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.39706. URL: https://aip.scitation.org/doi/abs/10.1063/1.39706.

[13] Tanuja Dixit, Yoshito Shimosaki, and Ken Takayama. "Adiabatic damping during acceleration in the Induction Synchrotron". July 2007, pp. 4009–4011. ISBN: 978-1-4244-0916-7. DOI: 10.1109/PAC.2007.4439951.

[14] Neil Kirby et al. “Emittance growth from Multiple Coulomb Scattering in a plasma wakefield accelerator”. July 2007, pp. 3097–3099. DOI: 10.1109/PAC.2007.4440680.

[15] D Möhl. “Sources of emittance growth” (2006). DOI: 10.5170/CERN-2006-002.245. URL: https://cds.cern.ch/record/941314.

[16] A. Chancé, X. Li, and P.A.P. Nghiem. “Avoiding Emittance Degradation When Transferring the Beam From and to a Plasma-Wakefield Stage”. *Proc. 10th International Particle Accelerator Conference (IPAC'19), Melbourne, Australia, 19-24 May 2019* (Melbourne, Australia). International Particle Accelerator Conference 10. Geneva, Switzerland: JACoW Publishing, June 2019, pp. 3594–3597. ISBN: 978-3-95450-208-0. DOI: doi:10.18429/JACoW-IPAC2019-THPGW006. URL: http://jacow.org/ipac2019/papers/thpgw006.pdf.

[17] Burkhard Schmidt. “The High-Luminosity upgrade of the LHC: Physics and Technology Challenges for the Accelerator and the Experiments”. *Journal of Physics: Conference Series* 706.2 (Apr. 2016), p. 022002. DOI: 10.1088/1742-6596/706/2/022002. URL: https://dx.doi.org/10.1088/1742-6596/706/2/022002.

[18] W Herr, B J Holzer, and B Muratori. “Design and Principles of Synchrotrons and Circular Colliders: Concept of Luminosity”. *Landolt-Börnstein* 21C (2013), pp. 6.4-1 - 6.4–7. DOI: 10.1007/978-3-642-23053-0_9. URL: https://cds.cern.ch/record/1625184.

[19] Frank Tecker. *Longitudinal Beam Dynamics in Circular Accelerators*. Tech. rep. 17 pages, contribution to the CAS - CERN Accelerator School: Introduction to Accelerator Physics. arXiv admin note: substantial text overlap with arXiv:2004.11908, arXiv:1601.04901. 2020. arXiv: 2011.02932. URL: https://cds.cern.ch/record/2746987.

[20] G Robert-Démolaize and Ralph Assmann. “Design and Performance Optimization of the LHC Collimation System” (Jan. 2006).

[21] S Y Lee. *Accelerator Physics*. 4th. WORLD SCIENTIFIC, 2019. DOI: 10.1142/11111. URL: https://www.worldscientific.com/doi/abs/10.1142/11111.

[22] Verena Kain. “Beam Dynamics and Beam Losses - Circular Machines” (Aug. 2016). DOI: 10.5170/CERN-2016-002.21.

[23] Moses Chung, Hong Qin, and Ronald C Davidson. “Twiss parameters and beam matrix formulation of generalized Courant–Snyder theory for coupled transverse beam dynamics”. *Physics of Plasmas* (2010).

[24] Jean Buon. “Beam phase space and emittance; rev. version” (Feb. 1992).

[25] R J Steinhagen. “Tune and chromaticity diagnostics” (2009). DOI: 10.5170/CERN-2009-005.317. URL: https://cds.cern.ch/record/1213281.

[26] M Ferrario, M Migliorati, and L Palumbo. *Space Charge Effects*. en. 2014. DOI: 10.5170/CERN-2014-009.331. URL: https://cds.cern.ch/record/1982426.

[27] F. Asvesta et al. “Identification and characterization of high order incoherent space charge driven structure resonances in the CERN Proton Synchrotron”. *Physical Review Accelerators and Beams* 23 (Sept. 2020). DOI: 10.1103/PhysRevAccelBeams.23.091001.

[28] S Dawson et al. “Measurements of Emittance and Tune Spread at Linac2 and Linac4 Injection Energies for the Proton Synchrotron Booster (PSB)” (2012). URL: https://cds.cern.ch/record/1418902.

[29] V Forte et al. “Investigations on CERN PSB Beam Dynamics with Strong Direct Space Charge Effects Using the PTC-Orbit Code” (2013). URL: https://cds.cern.ch/record/1571185.

[30] Foteini Asvesta. “Space charge and lattice driven resonances at the CERN injectors. Φαινμεν χωρικυ φρτιυ και συντνισμι στυ επιταχυντε τυ CERN”. Presented 16 Sep 2020. 2020. URL: https://cds.cern.ch/record/2771289.

[31] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning” (Dec. 2013).

[32] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. DOI: 10.48550/ARXIV.1509.02971. URL: https://arxiv.org/abs/1509.02971.

[33] Rentao Gu, Zeyuan Yang, and Yuefeng Ji. “Machine Learning for Intelligent Optical Networks: A Comprehensive Survey”. *Journal of Network and Computer Applications* (Feb. 2020). DOI: 10.1016/j.jnca.2020.102576.

[34] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, 1999.

[35] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[36] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. *Neural Networks* 61 (Jan. 2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[37] James McCaffrey. *How To Code Modern Neural Networks Using Python and NumPy*. 2019.

[38] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[39] Richard S Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. *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/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[40] B. Efron. “Bootstrap Methods: Another Look at the Jackknife”. *The Annals of Statistics* 7.1 (1979), pp. 1–26. DOI: 10.1214/aos/1176344552. URL: https://doi.org/10.1214/aos/1176344552.

[41] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. *Machine Learning* 3.1 (Aug. 1988), pp. 9–44. ISSN: 1573-0565. DOI: 10.1007/BF00115009. URL: https://doi.org/10.1007/BF00115009.

[42] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[43] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. *CoRR* abs/1802.09477 (2018). arXiv: 1802.09477. URL: http://arxiv.org/abs/1802.09477.

[44] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: 1509.02971 [cs.LG].

[45] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[46] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[47] Daniel Belver et al. "Design and measurements of a test stand for the sem-grid system of the ess-bilbao". July 2012.

[48] K Wittenburg. "Beam loss monitors" (2009). DOI: 10.5170/CERN-2009-005.249. URL: https://cds.cern.ch/record/1213279.

[49] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].