

# Renovation of the beam-based feedback systems in the LHC

**Student:** Leander Grech

**Supervisor:** Dr. Ing. Gianluca Valentino

**Co-supervisor:** Dr. Diogo Alves



**L-Università  
ta' Malta**

**Faculty of Information and Communication Technology  
Department of Communications and Computer Engineering  
University of Malta**

September 2021

*A dissertation submitted in partial fulfilment of the requirements for the degree of  
Ph.D. in Communications and Computer Engineering.*



L-Università  
ta' Malta

## **University of Malta Library – Electronic Thesis & Dissertations (ETD) Repository**

The copyright of this thesis/dissertation belongs to the author. The author's rights in respect of this work are as defined by the Copyright Act (Chapter 415) of the Laws of Malta or as modified by any successive legislation.

Users may access this full-text thesis/dissertation and can make use of the information contained in accordance with the Copyright Act provided that the author must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the prior permission of the copyright holder.

**FACULTY OF INFORMATION AND COMMUNICATION  
TECHNOLOGY**

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines” (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I, the undersigned, declare that the dissertation submitted is my work, except where acknowledged and referenced.

I understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

Leander Grech

Student Name



Signature

Renovation of the beam-based feedback systems in the LHC

Title of work submitted

01/09/2022

Date



L-Università  
ta' Malta

**FACULTY/INSTITUTE/CENTRE/SCHOOL** Information & Communication Technology  
**DECLARATION OF AUTHENTICITY FOR DOCTORAL STUDENTS**

Student's Code 0348796M

Student's Name & Surname Leander Grech

Course PhD in Computer Engineering

Title of Dissertation/Thesis  
Renovation of the beam-based feedback systems in the LHC

**(a) Authenticity of Thesis/Dissertation**

I hereby declare that I am the legitimate author of this Thesis/Dissertation and that it is my original work.

No portion of this work has been submitted in support of an application for another degree or qualification of this or any other university or institution of higher education.

I hold the University of Malta harmless against any third party claims with regard to copyright violation, breach of confidentiality, defamation and any other third party right infringement.

**(b) Research Code of Practice and Ethics Review Procedure**

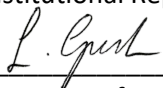
I declare that I have abided by the University's Research Ethics Review Procedures.  
Research Ethics & Data Protection form code ICT-2022-00076

As a Ph.D. student, as per Regulation 49 of the Doctor of Philosophy Regulations, I accept that my thesis be made publicly available on the University of Malta Institutional Repository.

As a Doctor of Sacred Theology student, as per Regulation 17 of the Doctor of Sacred Theology Regulations, I accept that my thesis be made publicly available on the University of Malta Institutional Repository.

As a Doctor of Music student, as per Regulation 26 of the Doctor of Music Regulations, I accept that my dissertation be made publicly available on the University of Malta Institutional Repository.

As a Professional Doctorate student, as per Regulation 55 of the Professional Doctorate Regulations, I accept that my dissertation be made publicly available on the University of Malta Institutional Repository.

  
Signature of Student

LEANDER GRECH  
Name in Full (in Caps)

01/09/2022  
Date

# Table of Contents

<b>Table of Contents</b>	<b>i</b>
<b>Abstract</b>	<b>iv</b>
<b>Acknowledgements</b>	<b>vi</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>xviii</b>
<b>Publications</b>	<b>xxi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 CERN and the LHC . . . . .	1
1.2 Beam-based feedback systems . . . . .	2
1.3 Machine learning . . . . .	4
1.4 Motivation and aims . . . . .	5
1.5 Dissertation structure . . . . .	5
<b>2 Background and literature review</b>	<b>7</b>
2.1 Overview of beam dynamics . . . . .	7
2.2 LHC accelerator and its magnets . . . . .	12
2.3 Beam Instrumentation . . . . .	18
2.3.1 Beam Position Monitors (BPMs) . . . . .	18
2.3.2 Tune Measurement and Feedback . . . . .	20

2.4	The LHC Beam-Based Feedback System State-of-the-Art . . . . .	29
2.4.1	Limitations . . . . .	35
2.5	Hardware acceleration . . . . .	38
2.6	Machine Learning . . . . .	40
2.6.1	Supervised Learning (SL) . . . . .	40
2.6.2	Universal Function Approximators . . . . .	42
2.6.3	Generative Deep Learning . . . . .	50
2.6.4	Reinforcement Learning (RL) . . . . .	52
2.6.5	Use of ML in particle accelerators . . . . .	63
2.7	Summary . . . . .	65
<b>3</b>	<b>Development of new tune estimation systems</b>	<b>67</b>
3.1	A review of the BQ algorithm . . . . .	68
3.2	Simulations . . . . .	69
3.3	An alternative approach . . . . .	75
3.3.1	Polynomial Fit . . . . .	77
3.3.2	Weighted Moving Average (WMA) . . . . .	78
3.3.3	Gaussian Process (GP) . . . . .	79
3.3.4	Measuring performance . . . . .	84
3.4	A Machine Learning approach . . . . .	85
3.4.1	Simple approach . . . . .	85
3.4.2	Improving the dataset with SimGANs . . . . .	90
3.4.3	Training . . . . .	92
3.5	Results comparison and analysis . . . . .	97
3.6	Summary . . . . .	102
<b>4</b>	<b>Application of Reinforcement Learning in a beam-based feedback controller</b>	<b>104</b>
4.1	QFB RL . . . . .	106
4.1.1	Environment setup . . . . .	107
4.1.2	Implementation details . . . . .	109

4.1.3	Training . . . . .	110
4.1.4	Evaluation . . . . .	126
4.2	OFC RL . . . . .	171
4.2.1	Environment Setup . . . . .	172
4.2.2	Training . . . . .	173
4.3	Summary . . . . .	175
<b>5</b>	<b>Renovation of the beam-based feedback systems</b>	<b>177</b>
5.1	Original BBFS design . . . . .	177
5.2	Feasibility of Hardware Acceleration . . . . .	186
5.2.1	Problem Definition . . . . .	186
5.2.2	Proposed Solution . . . . .	186
5.2.3	Benchmarking Tests . . . . .	187
5.2.4	SVD Accuracy statistics . . . . .	190
5.2.5	Memory and CPU usage . . . . .	191
5.2.6	Discussion . . . . .	194
5.3	Code renovation . . . . .	195
5.4	BFCLHC Testing framework . . . . .	198
5.5	Summary . . . . .	199
	<b>Conclusions</b>	<b>202</b>
	Achieved Aims and Objectives . . . . .	202
	Critique and Limitations . . . . .	203
	Future Work . . . . .	205
	<b>Glossary</b>	<b>208</b>
	<b>Bibliography</b>	<b>213</b>

# Abstract

The Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN) is the largest synchrotron built to date, having a circumference of approximately 27 km. The LHC is able to accelerate two counter-rotating proton and/or heavy-ion beams up to 7 TeV per charge. These highly energetic beams are contained inside a vacuum chamber with an inner diameter of 80 mm by means of strong magnetic fields produced by superconducting magnets. A beam cleaning and machine protection system is in place to prevent high-energy halo particles from impacting and heating the superconducting magnets.

Due to the tight tolerances on the beam trajectory imposed by the beam cleaning and machine protection system, the LHC was the first accelerator to require automatic beam-based feedback control. Until the LHC Run 2, this was implemented by the Beam-Based Feedback System (BBFS) which was mainly responsible for collecting beam measurements from various types of beam instrumentation and calculating the corrections in the current of corrector magnets throughout the whole length of the LHC.

Throughout the years of LHC operation, some of the original BBFS functionality became deprecated and fell into disuse. Various minor upgrades were performed, however, without a systematic redesign of the BBFS the code became difficult to maintain. The BBFS also suffered from a computational bottleneck during the calculation of the optics matrices used by the BBFS controllers. A feasibility study was performed to assess whether Hardware Acceleration (HA) in the form of Graphical Processing Units (GPUs) would improve the performance. It was found that the use of GPUs was not necessary and as a result the hardware upgrade for the BBFS could be finalised. Fol-



lowing this was the redesign and implementation of a more streamlined BBFS called BFCLHC. This work was done in collaboration with BE-OP-LHC and was designed with a more intuitive interface. A new feature called Function Players was implemented at the request of the LHC operators, which allows certain BFCLHC settings to be automated.

The Base-Band Q (tune) (BBQ) system is responsible for estimating the values of the horizontal and vertical tunes in both beams of the LHC. It was found that noise harmonics were perturbing the tune estimates and consequently causing the Tune Feedback (QFB) within the BBFS to behave erroneously. In this work, new tune estimation algorithms were developed to improve upon the accuracy of the BBQ system. To aid with this development, a simulation procedure was set up which mimics the frequency spectra obtained by the BBQ system. Two algorithmic approaches were proposed which take into consideration the location of the noise harmonics. A data-driven approach was also attempted where a combination of simulated and real frequency spectra were used to train a neural network to predict the value of the tune from a BBQ spectrum. It was observed that the data-driven approach improves upon the stability of the tune estimates in the presence of noise harmonics.

The use of Reinforcement Learning (RL) in beam-based feedback control systems was considered since it offers the possibility of optimal control, regardless of any dynamical changes in the machine. Since this work was performed during the LHC Long Shutdown 2 (LS2), the control problem present in the BBFS was formulated in a simulation environment called QFBEnv that allowed RL agents to be trained offline. The linear beam optics models used in the standard Proportional-Integral (PI) controllers of the BBFS were used to simulate the response of the LHC. In particular, the performance of several RL agents in the task of tune correction were assessed in detail and run through different failure scenarios. All the tests performed showed that certain RL agents can achieve a better performance than a standard PI controller, also in the presence of actuator failures. The possibility of using RL in the orbit feedback was also considered and discussed.

# Acknowledgements

I would like to express my sincere gratitude to my supervisor from the University of Malta, Gianluca Valentino, who has gifted me with the opportunity to start this work and the knowledge, guidance and tenacity to finish it. I am also grateful to my supervisor at CERN, Diogo Alves, who always made sure that my stay was welcoming and that I had the necessary skills to be productive.

Special thanks goes to all my colleagues in the beam instrumentation group. In particular I would like to thank Stephen Jackson and Thibaut Lefevre for welcoming me in their respective sections.

I am very grateful to Simon Hirländer for the discussions on Reinforcement Learning and to Verena Kain for introducing me to the Machine Learning community at CERN.

Heartfelt thanks goes to all the friends that I have made at CERN, whose care and joviality made my stay at CERN all the more worthwhile.

Finally, I want to thank my family. Their love and their support is what makes everything possible.

Leander Gresh

# List of Figures

2.1	Co-rotating local coordinate system of an ideal particle (black), circular design trajectory (red) and a perturbed orbit (blue) [1]. . . . .	8
2.2	LHC Tune diagram. The x-axis and y-axis are the horizontal and vertical tunes, respectively. The green circle shows an example of the tune working point at injection with the red dotted circle being the boundary of acceptable tunes. The blue circle shows an example of the tune working points during collisions. . . . .	10
2.3	The LHC accelerator layout [2]. . . . .	12
2.4	Simulated cross-section showing direction and magnitude of magnetic flux in a LHC dipole [2]. . . . .	15
2.5	The superconducting coils and the magnetic field lines (yellow) of a LHC dipole[3]. . . . .	16
2.7	Beam envelope inside a FODO cell [4]. . . . .	17
2.6	Forces acting on a particle moving through the quadrupole. Particle beam moves into the page [4]. . . . .	17
2.8	Cross-section showing the coils that provide the magnetic field in a superconducting quadrupole. . . . .	18
2.9	Cross-section of the button Beam Position Monitor (BPM) showing the 4 electromagnetic pick-ups (buttons) [5]. . . . .	19
2.10	Simple strip-line BPM configuration with a matched load impedance [6].	20
2.11	Principle of Direct Diode Detection (3D) [7] for the purpose of tune measurement. . . . .	21

2.12	A comparison of the BBQ and the EMA filtered spectra from operational data logged during the LHC Run 2 in 2018. . . . .	23
2.13	Present tune estimation algorithm diagram [8] . . . . .	24
2.14	BQ tune trace showing adjacent harmonic jumps. 6 points on this tune trace were chosen and their respective spectra are shown on the bottom plots. . . . .	25
2.15	BQ tune trace showing estimates coincident with a 50 Hz noise harmonic. 6 points on this tune trace were chosen and their respective spectra are shown on the bottom plots. . . . .	27
2.16	BQ tune trace showing large variance. 6 points on this tune trace were chosen and their respective spectra are shown on the bottom plots. . . . .	28
2.17	Schematic view of the Beam-Based Feedback System (BBFS) in the LHC complex. . . . .	29
2.18	Schematic view of the data paths and a more detailed view of the BBFS architecture. . . . .	30
2.19	Pseudo-identity matrix obtained by multiplying the pseudo-inverse calculated with the first 390 eigenvalues, to the response matrix . . . . .	34
2.20	Pseudo-identity matrix obtained by multiplying the pseudo-inverse calculated with the first 40 eigenvalues, to the response matrix . . . . .	35
2.21	A generic diagram of an ANN where the nodes/neurons consist of the input, hidden and output nodes. The number of hidden layers, $H$ must be $\geq 1$ otherwise the resultant effect is a simple linear transformation. The constant node represents the bias term, where when multiplied by the bias weights $b$ (shown in red), acts as the y-intercept for the activation function of the respective consecutive node. The length $\ell$ of each hidden layer is denoted by the same symbol however each hidden layer can have an arbitrary length where $\ell \geq 1$ . . . . .	42
2.22	Generic segment of a ANN. . . . .	43

2.23	Four of the most extensively used activation functions. Due to its simplicity and good empirical results, ReLU is often considered the best choice for hidden layers. . . . .	44
2.24	An example of a CNN used for image classification. . . . .	49
2.25	Generative Adversarial Network (GAN) architecture applied to image generation using the input data distribution [9]. . . . .	50
2.26	Top-level view of the interaction between the Agent and the Environment [10]. . . . .	52
2.27	Each episode consists of a repeated sequence of state-action pairs along with a respective reward outcome. . . . .	53
2.28	A non-exhaustive taxonomy of RL algorithms [11]. . . . .	54
2.29	Beam 1 vertical loss maps at 450GeV for nominal collimator settings (left) and a 200 $\mu m$ displacement in one vertical collimator (right) [12].	64
3.1	<b>BBQ spectrum</b> (blue) from: Beam 1, Plane V, Beam Mode INJPHYS, Fill 6847. <b>Simulated spectrum</b> (orange): Second-order system baseline spectrum. . . . .	71
3.2	<b>BBQ spectrum</b> (blue) from: Beam 2, Plane V, Beam Mode ADJUST, Fill 6874. <b>Simulated spectrum</b> (orange): Baseline spectrum with added Gaussian noise. . . . .	72
3.3	<b>BBQ spectrum</b> (blue) from: Beam 2, Plane V, Beam Mode FLATTOP, Fill 6881. <b>Simulated spectrum</b> (orange): Noisy baseline spectrum with injected 50 Hz noise harmonics. . . . .	73
3.4	An example of the operational frequency windowing used by the BQ algorithm during the LHC Run 2 to the new window length used in this work. The tunes were obtained from LHC fill 6890, horizontal plane of beam 1 in the FLATTOP beam mode. During FLATTOP the optics are changed and thus the tune shifts to a new frequency. . . . .	74

3.6	Traces of individual harmonics within a frequency window. Every column in the heat-map is a normalised BBQ spectrum obtained from a fill in Run 2. . . . .	76
3.5	An example of a spectrum from Fill 6890, beam 1, horizontal, in beam mode FLATTOP (blue). The red vertical bands correspond to the frequencies which are dropped from the spectrum. The resulting spectrum contains gaps (orange scatter). . . . .	77
3.7	Performance of Weighted Moving Average (WMA) with different window lengths on three samples of real BBQ spectra. . . . .	80
3.8	Performance of Gaussian Process (GPs) with different RBF kernel length scales on three samples of real BBQ spectra. . . . .	82
3.9	Error density histogram from 10000 simulated spectra for alternative approach algorithms. . . . .	83
3.10	Histograms showing the distribution of errors of the tune estimates from the resonance used to simulate the data. Tune estimates obtained using the fully-connected networks; Model #0 to Model #2. . . . .	87
3.11	Network using convolutional layers . . . . .	88
3.12	Histograms showing the distribution of errors of the tune estimates from the resonance used to simulate the data. Tune estimates obtained using the convolutional networks; Model #3 to Model #7. . . . .	89
3.13	Overview of the SimGAN architecture. . . . .	90
3.14	Refined spectrum generation scheme using a bank of trained SimGANs. . . . .	95
3.15	Results from a trained SimGAN. The green line represents the tune estimate from BQ algorithm on the refined spectrum. . . . .	96

3.16	(Background) Heat map obtained by post-processing BBQ spectra from LHC Fill 6768, Beam 1, Horizontal plane, in the transition from INJPHYS to PRERAMP. (Foreground) mean, $\mu$ , and scaled standard deviation, $3\sigma$ , of the tune evolution as estimated by different tune estimation algorithms and ML models: original algorithm (BQ), Gaussian Process (GP) with RBF kernel length scale 25 (GP25), GP70, GP130, Weighted Moving Average (WMA) with window length 5 (WMA5), WMA10, WMA20, best ANN model from Simple approach (ML#1), best 1D CNN model from Simple approach (ML#5) and ANN model trained using spectra refined by SimGAN. . . . .	99
3.17	(Background) Heat map obtained by post-processing BBQ spectra from LHC Fill 6768, Beam 1, Horizontal plane, in the transition from RAMP to FLATTOP. (Foreground) mean, $\mu$ , and scaled standard deviation, $3\sigma$ , of the tune evolution as estimated by different tune estimation algorithms and ML models: original algorithm (BQ), Gaussian Process (GP) with RBF kernel length scale 25 (GP25), GP70, GP130, Weighted Moving Average (WMA) with window length 5 (WMA5), WMA10, WMA20, best ANN model from Simple approach (ML#1), best 1D CNN model from Simple approach (ML#5) and ANN model trained using spectra refined by SimGAN. . . . .	100
3.18	(Background) Heat map obtained by post-processing BBQ spectra from LHC Fill 6768, Beam 1, Horizontal plane, in the transition from SQUEEZE to ADJUST. (Foreground) mean, $\mu$ , and scaled standard deviation, $3\sigma$ , of the tune evolution as estimated by different tune estimation algorithms and ML models: original algorithm (BQ), Gaussian Process (GP) with RBF kernel length scale 25 (GP25), GP70, GP130, Weighted Moving Average (WMA) with window length 5 (WMA5), WMA10, WMA20, best ANN model from Simple approach (ML#1), best 1D CNN model from Simple approach (ML#5) and ANN model trained using spectra refined by SimGAN. . . . .	101

3.19	Probability distribution of the tune estimation stability. Obtained from Fill 6768, beam 1, horizontal plane using spectra from INJPHYS to FLATTOP. Slow and Fast correspond to stability measures having time constants of 10 s and 2 s respectively (Equation (3.15)). Original algorithm (BQ), Gaussian Process (GP) with RBF kernel length scale 25 (GP25), GP70, GP130, Weighted Moving Average (WMA) with window length 5 (WMA5), WMA10, WMA20, best ANN model from Simple approach (ML#1), best 1D CNN model from Simple approach (ML#5) and ANN model trained using spectra refined by SimGAN. Threshold=0.005, chosen by LHC operators. . . . .	103
4.1	Performance statistics of NAF2 during training. Five NAF2 agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.2. . . . .	113
4.2	Performance statistics of PPO during training. Five PPO agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.3. . . . .	115
4.3	Performance statistics of TD3 during training. Five TD3 agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.4. . . . .	116
4.4	Performance statistics of the best TD3 agents during the hyperparameter search. . . . .	119
4.5	Performance statistics of SAC during training. Five SAC agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.7. . . . .	121
4.6	Performance statistics of the best SAC agents during the hyperparameter search. . . . .	123
4.7	Performance statistics of SAC-TFL during training. Five SAC-TFL agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.9 . . . . .	124



4.8	Snapshot of the training performance of AE-DYNA-SAC. Batch denotes the information about the data collected from the real environment. Sim denotes information about the agent performance over the models. Tests denotes information about the agent performance over the real environment. . . . .	126
4.9	Best NAF2 agent. Action is deterministic. . . . .	128
4.10	Best NAF2 agent. Action Gaussian noise with zero mean and standard deviation 10% of action range. . . . .	129
4.11	Best NAF2 agent. Action Gaussian noise with zero mean and standard deviation 25% of action range. . . . .	129
4.12	Best NAF2 agent. Action Gaussian noise with zero mean and standard deviation 50% of action range. . . . .	130
4.13	Effect of action noise on the episode length due to varying action noise on the best NAF2 agent and the PI controller. . . . .	131
4.14	Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best NAF2 agent and the PI controller. . . . .	131
4.15	Effect of action noise on the last action used in the episode due to varying action noise on the best NAF2 agent and the PI controller. . . . .	132
4.16	Best PPO agent. Action is deterministic. . . . .	133
4.17	Best PPO agent. Action Gaussian noise with zero mean and standard deviation 10% of action range. . . . .	133
4.18	Best PPO agent. Action Gaussian noise with zero mean and standard deviation 25% of action range. . . . .	134
4.19	Best PPO agent. Action Gaussian noise with zero mean and standard deviation 50% of action range. . . . .	134
4.20	Effect of action noise on the episode length due to varying action noise on the best PPO agent and the PI controller. . . . .	136

4.21	Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best PPO agent and the PI controller. . . . .	136
4.22	Effect of action noise on the last action used in the episode due to varying action noise on the best PPO agent and the PI controller. . . .	137
4.23	Best TD3 agent. Action is deterministic. . . . .	138
4.24	Best TD3 agent. Action Gaussian noise with zero mean and standard deviation 10% of action range. . . . .	138
4.25	Best TD3 agent. Action Gaussian noise with zero mean and standard deviation 25% of action range. . . . .	139
4.26	Best TD3 agent. Action Gaussian noise with zero mean and standard deviation 50% of action range. . . . .	139
4.27	Effect of action noise on the episode length due to varying action noise on the best TD3 agent and the PI controller. . . . .	141
4.28	Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best TD3 agent and the PI controller. . . . .	141
4.29	Effect of action noise on the last action used in the episode due to varying action noise on the best TD3 agent and the PI controller. . . .	142
4.30	Best SAC agent. Action is deterministic. . . . .	143
4.31	Best SAC agent. Action Gaussian noise with zero mean and standard deviation 10% of action range. . . . .	143
4.32	Best SAC agent. Action Gaussian noise with zero mean and standard deviation 25% of action range. . . . .	144
4.33	Best SAC agent. Action Gaussian noise with zero mean and standard deviation 50% of action range. . . . .	144
4.34	Effect of action noise on the episode length due to varying action noise on the best SAC agent and the PI controller. . . . .	146

4.35	Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best SAC agent and the PI controller. . . . .	146
4.36	Effect of action noise on the last action used in the episode due to varying action noise on the best SAC agent and the PI controller. . . . .	147
4.37	Best SAC-TFL agent. Action is deterministic. . . . .	148
4.38	Best SAC-TFL agent. Action Gaussian noise with zero mean and standard deviation 10% of action range. . . . .	148
4.39	Best SAC-TFL agent. Action Gaussian noise with zero mean and standard deviation 25% of action range. . . . .	149
4.40	Best SAC-TFL agent. Action Gaussian noise with zero mean and standard deviation 50% of action range. . . . .	149
4.41	Effect of action noise on the episode length due to varying action noise on the best SAC-TFL agent and the PI controller. . . . .	150
4.42	Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best SAC-TFL agent and the PI controller. . . . .	151
4.43	Effect of action noise on the last action used in the episode due to varying action noise on the best SAC-TFL agent and the PI controller. . . . .	152
4.44	Best AE-DYNA-SAC agent. Action is deterministic. . . . .	153
4.45	Best AE-DYNA-SAC agent. Action Gaussian noise with zero mean and standard deviation 10% of action range. . . . .	153
4.46	Best AE-DYNA-SAC agent. Action Gaussian noise with zero mean and standard deviation 25% of action range. . . . .	154
4.47	Best AE-DYNA-SAC agent. Action Gaussian noise with zero mean and standard deviation 50% of action range. . . . .	154
4.48	Effect of action noise on the episode length due to varying action noise on the best AE-DYNA-SAC agent and the PI controller. . . . .	155

4.49	Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best AE-DYNA-SAC agent and the PI controller. . . . .	156
4.50	Effect of action noise on the last action used in the episode due to varying action noise on the best AE-DYNA-SAC agent and the PI controller.	157
4.51	Effect of 50 Hz harmonics on the best NAF2, PPO and TD3 agents, and PI controller. . . . .	159
4.52	Effect of 50 Hz harmonics on the best SAC, SAC-TFL and AE-DYNA-SAC agents, and PI controller. . . . .	160
4.53	Effect of different number of actuator failures occurring at different episode steps. The best NAF2 policy is compared with the PI controller.	162
4.54	Effect of different number of actuator failures occurring at different episode steps. The best PPO policy is compared with the PI controller.	165
4.55	Effect of different number of actuator failures occurring at different episode steps. The best TD3 policy is compared with the PI controller.	166
4.56	Effect of different number of actuator failures occurring at different episode steps. The best SAC policy is compared with the PI controller.	167
4.57	Effect of different number of actuator failures occurring at different episode steps. The best SAC-TFL policy is compared with the PI controller. . . . .	168
4.58	Effect of different number of actuator failures occurring at different episode steps. The best AE-DYNA-SAC policy is compared with the PI controller. . . . .	169
4.59	Undiscounted episode returns for four PPO agents trained on OFCEnv, using the hyperparameters tabulated in Table 4.18. . . . .	174
5.1	Flowchart of the main program of the Orbit Feedback Controller (OFC).	179
5.2	Flowchart of the INIT block in Figure 5.1. . . . .	180
5.3	Flowchart of the LISTEN OT block in Figure 5.1, which waits for the reception of the Orbit Trigger (OT) TCP packet from the OFSU. . . .	181

5.4	Flowchart of the <code>CONC</code> block in Figure 5.1, which concentrates the incoming data from the BPMs, BBQ and CODs. . . . .	182
5.5	Left flowchart shows the <code>COMP</code> block in Figure 5.1, middle flowchart shows the energy (RF frequency) correction and the right flowchart shows the orbit correction. . . . .	183
5.6	Continuation of Figure 5.5 showing the flowchart of the tune and coupling correction. . . . .	184
5.7	SVD execution time on different devices . . . . .	189
5.8	Average absolute error obtained from the different configurations of HA libraries and devices using Equation (5.2). . . . .	191
5.9	Standard deviation of the absolute error obtained from the different configurations of HA libraries and devices using Equation (5.2). . . . .	192
5.10	CPU usage percentage during SVD calculation using different libraries. . . . .	192
5.11	Details from Figure 5.10. . . . .	193
5.12	Memory consumption during SVD calculation using different libraries. . . . .	194
5.13	Function Player states (encircled) and events (connections) as implemented in Beam Feedback Controller LHC (BFCLHC). . . . .	196
5.14	Testing framework schematic, showing the use of Docker containers in the GitLab Continuous Integration (CI) framework. . . . .	199
5.15	Screenshot of the BFCLHC testing framework test results using CI on GitLab. . . . .	200

# List of Tables

2.1	Types of corrector magnets used by the Orbit Feedback Controller (OFC) and Tune Feedback (QFB). H and V denote horizontal and vertical, respectively. F and D denote focusing and defocusing, respectively [2]. . . . .	14
3.1	Error statistics from 10000 simulated spectra for alternative approach algorithms (GP and WMA). . . . .	84
3.2	Number of nodes per hidden layer used for ANN tune estimation models.	86
3.3	Model architectures presented for CNNs . . . . .	88
3.4	SimGAN Refiner network architecture . . . . .	94
3.5	SimGAN Discriminator network architecture . . . . .	94
4.1	Actuators and Sensors of the beam-based feedback systems considered for RL. . . . .	105
4.2	Hyperparameters used for NAF2. . . . .	112
4.3	Hyperparameters used for PPO. . . . .	114
4.4	Original hyperparameters used for TD3. . . . .	116
4.5	Hyperparameter search of TD3. . . . .	117
4.6	Hyperparameters of the five successful agents and the corresponding best mean episode length. The best agent, TD3#0, obtained an average episode length of 41.3 steps after 15000 training steps. . . . .	118
4.7	Original hyperparameters used for SAC. . . . .	120
4.8	Hyperparameters of the five best performing agents. . . . .	122

4.9	Hyperparameters used for SAC-TFL. . . . .	122
4.10	The statistics (mean±std.) for the episode length obtained by the best NAF2 agent and PI controller with respect to the amplitude of Gaussian action noise. . . . .	128
4.11	The statistics (mean±std.) for the episode length obtained by the best PPO agent and PI controller with respect to the amplitude of Gaussian action noise. . . . .	135
4.12	The statistics (mean±std.) for the episode length obtained by the best TD3 agent and PI controller with respect to the amplitude of Gaussian action noise. . . . .	140
4.13	The statistics (mean±std.) for the episode length obtained by the best SAC agent and PI controller with respect to the amplitude of Gaussian action noise. . . . .	145
4.14	The statistics (mean±std.) for the episode length obtained by the best SAC-TFL agent and PI controller with respect to the amplitude of Gaussian action noise. . . . .	150
4.15	The statistics (mean±std.) for the episode length obtained by the best AE-DYNA-SAC agent and PI controller with respect to the amplitude of Gaussian action noise. . . . .	152
4.16	Episode length statistics (mean±std) under the effect of actuator failures.	170
4.17	Distance to Optimal point (DTO) statistics (mean±std) under the effect of actuator failures. Underlined cells show where it is likely to observe successful termination. . . . .	171
4.18	Hyperparameters used for different PPO agents on OFCEnv. . . . .	173
5.1	Specifications of the Techlab machines available. Giga Floating-point Operations per Second (GFLOPS) quantify the processor performance in the order of $1 \times 10^9$ 32-bit floating point operations per second (FP32). The two GPUs also show the performance on 64-bit double floating point (FP64) precision. . . . .	188

5.2	Host CPUs of the two GPUs used for the benchmarking tests. . . . .	188
-----	--	-----



# Publications

- L. Grech, D. Alves, S. Jackson, G. Valentino, and J. Wenninger, “Feasibility of Hardware Acceleration in the LHC Orbit Feedback Controller,” in 17th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’19), New York, NY, USA, 05-11 October 2019, 2020, pp. 584–588.
- L. Grech et al., “An Alternative Processing Algorithm for the Tune Measurement System in the LHC,” in 9th International Beam Instrumentation Conference (IBIC’20), Virtual, 14-18 September 2020.
- L. Grech, G. Valentino, and D. Alves, “A Machine Learning Approach for the Tune Estimation in the LHC,” *Information*, vol. 12, no. 5, p. 197, Apr. 2021.
- L. Grech, D. Alves and G. Valentino, ”Renovation of the beam-based feedback controller in the LHC,” in 18th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’21), Virtual, 14-22 October 2021.

## Future Publications

- L. Grech and G. Valentino, ”Application of Reinforcement Learning in the LHC Tune Feedback,” to be submitted to *Frontiers: Application of Artificial Intelligence and Machine Learning to Accelerators*.

# Chapter 1

## Introduction

### 1.1 CERN and the LHC

The European Organization for Nuclear Research (CERN) houses the Large Hadron Collider (LHC), which is the largest particle accelerator of its type built to date [13]. The LHC was constructed in the tunnel of the Large Electron Positron (LEP) collider, and is a 27 km long synchrotron capable of accelerating protons and heavy ions up to 7 TeV per charge and storing them in two counter-rotating beams [2]. Each beam in the LHC can store up to 362 MJ and is thus able to cause irreparable damage to equipment inside the tunnel if it comes into contact with the sides of the vacuum chamber [14]. It is also important that the beam is kept far from the sides of the beam pipe, since loss of particles in the superconducting material of the magnets may cause a phase-transition to normal conductivity. This sudden increase of resistivity in the superconducting magnet coils would transform all the electrical energy into heat and cause irreparable damage to the coils themselves.

To prevent this, the LHC is equipped with a state-of-the-art machine cleaning and protection system which would dump the beam into an external absorber made of graphite, situated at a safe distance from all sensitive equipment [2, 15]. The purpose of the cleaning system is to control the particle losses which subsequently ensures a maximal luminosity. Luminosity is one of the two main performance measurements in particle colliders, with the other being beam energy. Simply stated, the luminosity,  $\mathcal{L}$  is

the proportionality factor between the number of events (particle collisions) per second and the beam cross-section at the Interaction Point (IP). It can be approximated by the following formula:

$$\mathcal{L} = \frac{N_1 N_2 f N_b}{A} \cdot F$$

Where  $N_1$  and  $N_2$  are the number of particles (protons or heavy ions) per bunch for the two beams, respectively,  $N_b$  is the number of bunches in each beam,  $f$  is the revolution frequency of the LHC which is 11.245 kHz, and  $A = 4\pi\sigma_x\sigma_y$  is the cross-sectional area of two colliding Gaussian beams.  $A$  is expressed as the area within one sigma of a two-dimensional Gaussian, where the horizontal and vertical standard deviations are  $\sigma_x$  and  $\sigma_y$ , respectively.  $F$  represents a reduction factor which includes reduction in luminosity due to a non-zero crossing angle between the beams and an imperfect beam overlap [16].

The Beam-Based Feedback System (BBFS) relies on measurements from beam instrumentation, e.g. beam positions within the beam pipe, in order to control the performance of the LHC. The BBFS is also traditionally reliant on the use of linear beam optics models in order to reliably control the beam. The early design of the BBFS implemented feedback systems for various beam parameters including: a) the Orbit Feedback Controller (OFC) which mitigates the effects of beam orbit perturbation sources and ensures optimal beam position and stability within the vacuum chamber and; b) the Tune Feedback (QFB) which corrects the tune shifts from the reference tune [1]

## 1.2 Beam-based feedback systems

The LHC was designed to handle particle momenta between one and two orders of magnitude higher than previous accelerators. More energetic beams imply more energetic halo particles, which run the risk of damaging sensitive equipment in the LHC complex. From the early design stages of the LHC it was already understood that the state-of-the-art beam cleaning system, also known as the collimation system could

only effectively clean the particle beam if it imposed tightened tolerances on the design and operational parameters of the LHC [17].

As a direct consequence of these tightened tolerances, the LHC was the first particle accelerator which required a feedback control system on the orbit stability for safe and reliable operation. It was found that if some beam orbit perturbation sources were not compensated by adjustments to the LHC magnetic lattice, the orbit stability would decrease by one order of magnitude and subsequently increase beam losses [1].

The orbit feedback control system was implemented by the OFC, which measured the beam orbit and calculated the corrections required in magnetic lattice. This was achieved by a 2D Response Matrix (RM), a linear model which takes as input the magnetic deflections produced by the magnets and outputs the the response on the beam orbit. This model can either be found by executing a series of calibration tests on the LHC during commissioning or by using tools such as Methodical Accelerator Design (MAD-X), which iteratively calculate the RM from the beam optics [18].

The BBFS was also equipped with feedback systems for the energy, tune, coupling and chromaticity. The tune is defined as the number of transverse oscillations that a particle makes in one turn around the LHC. These oscillations are obtained when quadrupole magnets are used to confine the beam. Coupling is the phenomenon when betatron oscillations are no longer confined to one plane, but have components on two orthogonal planes. Finally there is chromaticity, which is a measurement of the dependency of the tune on the particle's momentum [19]. This is synonymous to chromatic aberration which is the effect created by the varying refraction of different wavelengths of light. Following this analogy, the distortion in an image caused by chromatic aberration, is called chromaticity and results in an unwanted tune spread in the particle beam [4, 20].

The tune and coupling feedback systems went hand in hand, since the values of the tunes in both planes were responsible for the coupling between the two planes. These feedback systems were incorporated within the QFB, where the skewed betatron oscillations caused by coupling are mitigated with the aid of skew quadrupoles, which are quadrupole magnets rotated by  $45^\circ$ . Studies have also shown that if coupling is

left unattended, it could push the tune in either plane to a resonant state which results in the loss of beams [21].

Quadrupole magnets are used to confine the beam, however, each quadrupole focuses the beam on one plane and defocuses it in an orthogonal plane in the direction of the beam. These magnets directly influence the betatron function of the beam and thus have an impact on the value of the tune. As a consequence any inherent magnetic imperfections at specific locations of the quadrupoles' magnetic fields also have an effect on the tune of the beam. If the tune was at a value which resulted in the same transverse oscillations occurring in the same longitudinal location, a resonance would ensue. The amplitude of the betatron motion would monotonically increase over many turns and finally results in beam loss. Therefore, the ideal value of the tune is an irrational number, since the same transverse oscillations could not occur in the same location of the magnetic lattice [19].

### 1.3 Machine learning

Machine Learning (ML) is a subset of Artificial Intelligence and emerged from the fields of computer science, mathematics and statistics. It concerns itself with the study of algorithms, designed to solve particular tasks without the use of explicit instructions, which are tailor-made for the task at hand. Rather, ML aims to perform tasks based only on previous experiences [22]. Due to this remarkable property, ML tools have a potential use on systems which produce large amounts of data, e.g. particle accelerators.

The most relevant ML fields to this work are Supervised Learning (SL), Generative Adversarial Networks (GANs), and Reinforcement Learning (RL). SL is the study of algorithms which given a set of correctly labelled data, can learn to accurately predict the labels of potentially unseen data. GAN is a framework, where two networks are trained to solve a minimax problem. The resultant effect is that one of the networks learns to generate new data from an approximation of the real world distribution which produced the training data. RL is the study of algorithms that can learn a

behavioural policy by interacting with an environment which produces a scalar signal called reward. The ultimate purpose of the learned policy is to maximise the reward given by the environment at each step.

## 1.4 Motivation and aims

The BBFS comprised of a standalone C++ program using ROOT libraries and a FESA class which communicates with it. The BBFS included a multitude of unused functionality left over from early versions and throughout the years, various efforts were made in order to improve its maintainability and to fix any bugs found during operation. Long Shutdown 2 (LS2) allows enough time for major upgrades to take place and one of the aims of this work is the renovation of both the OFC and Orbit Feedback Service Unit (OFSU).

Another important sub-system of the BBFS was the QFB, which only had perturbed tune estimates as input for tune control. Therefore, one of the aims of this work is to develop alternatives to the current tune estimation algorithm, which can alleviate the effect of tune perturbations.

During LS2, some of the hardware running the BBFS was also upgraded. In particular, more powerful machines are planned to be installed, which potentially could harness Hardware Acceleration (HA). One of the aims of this work is to perform a feasibility study on the use of HA in the BBFS sub-systems.

RL has become increasingly popular at CERN and other particle accelerator communities. One of the aims of this work is to assess the feasibility of applying RL within the BBFS. Various state-of-the-art RL algorithms need to be taken into consideration, as well as the formulation of a beam-based feedback system into an RL environment.

## 1.5 Dissertation structure

Following the introduction, Chapter 2 provides the necessary theory and definitions for the work presented in this thesis. This chapter starts by going through the relevant

beam dynamics, which explain the movement of the beam inside the LHC. Following is a review of the hardware which comprises both (a) the instrumentation which provides beam measurements and (b) the magnets which allow for the control of said parameters. Consequently, the design of the BBFS until Run 2 is introduced and discussed. This is followed by an overview of HA, which introduces the work done to analyse the feasibility of HA in the OFC. The background research ends by giving an introduction to the various branches of ML relevant to this work.

Chapter 3 takes a closer look at the source of the QFB controller failures, which is inaccurate tune estimates. Work was done to improve the tune estimate of the LHC, which in turn helps the QFB improve its operational performance. Two alternative tune estimation processes are presented in this work, one which attempts an improved algorithmic approach, whilst the other uses machine learning tools that learn to estimate the tune by training on offline data. The results of both approaches are compared and discussed.

In Chapter 4, the results of the first attempt at using RL for the tune feedback control in a particle accelerator are presented. This chapter offers a comparative study of using different state-of-the-art RL algorithms to train a data-driven QFB. The possibility of applying state-of-the-art RL algorithms to the OFC is also considered and discussed.

Chapter 5 goes further through the design of the OFC until Run 2 and explains the operation of OFC. Following this is a feasibility study on the use of HA to compute optics calculations faster. Then, the design choices that were made in the renovation of the OFC code are introduced and discussed. Finally, the development of a new testing framework by BE-OP-LHC is summarised.

# Chapter 2

## Background and literature review

This chapter will provide an appropriate background for the various topics explored in this work. A literature review of the state-of-the-art is also provided. Primarily, Section 2.1 will introduce the physics required to accurately model the beam dynamics relevant to the Beam-Based Feedback System (BBFS). Section 2.2 and Section 2.3 will delve into the hardware which is installed in the LHC, that ultimately produced all the data that was accrued during Run 2. Section 2.4 will describe the design, implementation and limitations of the BBFS which was used during Run 2. Section 2.5 will introduce a possible solution to one of the limitations of OFC in terms of applying Hardware Acceleration (HA) to accelerate the Singular Value Decomposition (SVD) operation. Finally Section 2.6 will provide a concise summary of ML with special focus on the topics most relevant to this work, such as GANs and RL.

### 2.1 Overview of beam dynamics

The design of the BBFS in the LHC is built on a number of key beam dynamics principles for circular accelerators. Firstly, consider the co-rotating local coordinate system as shown in Figure 2.1 where by definition  $x(s+C) = x(s)$ ,  $y(s+C) = y(s)$ ,  $\rho(s+C) = \rho(s)$  and  $b_n(s+C) = b_n(s)$ , where  $C$  is the accelerator circumference,  $x$  is the horizontal local axis pointing outward,  $y$  is the vertical local axis pointing upward,  $s$  is the longitudinal axis pointing in the direction of motion of the ideal particle,  $\rho(s)$



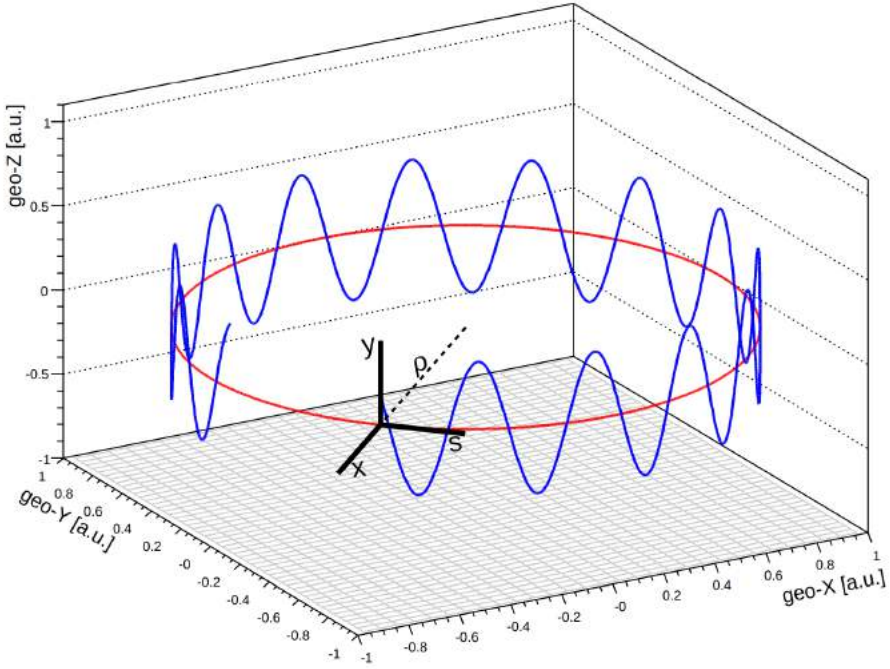


Figure 2.1: Co-rotating local coordinate system of an ideal particle (black), circular design trajectory (red) and a perturbed orbit (blue) [1].

is the local bending radius of the dipole magnetic fields at longitudinal position  $s$ , and  $b_n$  represents the  $n^{\text{th}}$  magnetic multi-pole strength for  $n > 1 \in \mathbb{N}$

To derive a system of equations which describe the particle dynamics in a generalised magnetic field is a cumbersome task. For the purpose of obtaining an analytical representation of the beam dynamics, linearity can be assumed. This implies that only dipoles with bending radius  $\rho(s)$  and quadrupoles with a linear magnetic field  $b_2(s, x, y)$  are considered (Dipoles and quadrupoles are discussed in more detail in Section 2.2). Following this assumption a Hill type system of equations as shown in Equation (2.1) can be obtained by considering a particle which traverses the blue path on Figure 2.1 [1, 23].

$$\begin{cases} x'' + \left( k(s) + \frac{1}{\rho^2(s)} \right) \cdot x & = \frac{1}{\rho(s)} \cdot \frac{\Delta p}{p} \\ y'' - k(s) \cdot y & = 0 \end{cases} \quad (2.1)$$

where  $k(s)$  is the quadrupole strength normalised by the momentum of the moving

particle and  $\frac{\Delta p}{p}$  is the particle momentum offset with respect to the ideal particle following the red path.

The following ansatz, along with Floquet's Theorem is used to solve the homogeneous part of Equation (2.1):

$$z_\beta(s) = \sqrt{\epsilon_z \beta_z(s)} \cdot \cos(\Phi_z(s) + \phi_z) \quad (2.2)$$

where  $z$  can be either the horizontal or vertical coordinate ( $x, y$ ), and  $\epsilon_z$  and  $\phi_z$  are the particle's initial conditions. Equation (2.2) describes the particle oscillations known as the *Betatron oscillations*, which derive their name from the Betatron accelerator in which they were first observed [24]. Consequently, the amplitude modulation due to the changing focusing strengths of the quadrupoles,  $\beta_z(s)$ , is known as the *betatron function*.  $\Phi_z(s)$  is known as the *betatron phase advance* and is obtained from  $\beta_z$ :

$$\Phi_z(s) = \int_{s_0}^s \frac{1}{\beta_z(s')} ds' \quad (2.3)$$

where  $s_0$  is an arbitrary longitudinal reference point on the ideal particle trajectory.  $\Phi$  is also related to the number of transverse oscillations that a particle performs in one full turn around the machine. The latter can be measured for any circular accelerator and is called *tune*,  $Q$ :

$$Q_z = \frac{1}{2\pi} \int_0^C \Phi_z(s) ds \quad (2.4)$$

It is important to note that the choice of both  $Q_x$  and  $Q_y$ , which denote the tunes in the horizontal and vertical planes respectively, is critical to the stability of a circular machine. The inevitable small imperfections in the quadrupole magnets can lead to the growth of a particle's transverse oscillations after many turns. Uncontrolled, this effect leads to large betatron amplitudes which cause energetic particles to come in contact with the beam pipe. This contact creates a cascade of particle losses which ultimately results in the loss of the beam itself. In particular,  $Q_z$  has to be a non-integer number and ideally an irrational number to avoid resonance. In addition, any choice of  $Q_x$  and  $Q_y$  which satisfies the resonance condition in Equation (2.5) should

also be avoided, where for  $m, n, p \in \mathbb{Z}$ .

$$mQ_x + nQ_y = p \quad (2.5)$$

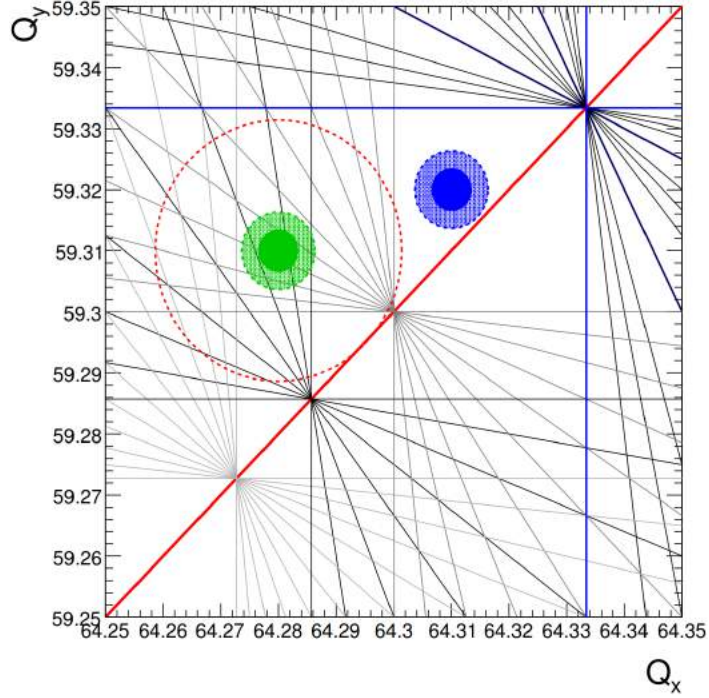


Figure 2.2: LHC Tune diagram. The x-axis and y-axis are the horizontal and vertical tunes, respectively. The green circle shows an example of the tune working point at injection with the red dotted circle being the boundary of acceptable tunes. The blue circle shows an example of the tune working points during collisions.

Figure 2.2 shows an enlarged tune diagram, with an example of injection and collision tune working points (respectively the green and blue circles). The red dotted circle around the injection working point indicates the allowed tune values during injection. The red straight line signifies the second order resonance and the blue orthogonal straight lines the third order resonance. The other grey lines are higher order resonances which have little effect on the tune with respect to the BBFS. Regardless, the collision tune reference values are shifted to a relatively calm region of the tune diagram after injection.

The inhomogeneous part of Equation (2.1) describes a shift in the equilibrium beam position which is due to the particle momentum offset,  $\frac{\Delta p}{p}$ , and which can be

found from the bending radii,  $\rho(s)$ , of the dipole magnets. The special solution of the inhomogeneous part of Equation (2.1) is called the *Dispersion function*,  $D(s)$ . Normally  $D(s)$  is defined for  $\frac{\Delta p}{p} = 1$  and is periodic. The shift of the equilibrium beam position due to dispersion is given by Equation (2.6).

$$z_d(s) = D(s) \cdot \frac{\Delta p}{p} \quad (2.6)$$

where  $z_d(s)$  is known as the *dispersion orbit*. Dispersion is an intrinsic property of dipole magnets and is evident when considering the definition of *beam rigidity*,  $B\rho$ :

$$B\rho = \frac{p}{q} \quad (2.7)$$

where  $p$  and  $q$  are the momentum and charge of a particle, respectively, and  $B$  is the dipolar field acting on the particle. Equation (2.7) shows that under the same  $B$ , the bending radius,  $\rho$ , experienced by a particle is proportional to its momentum,  $p$ . Therefore, the natural presence of momentum spread within a particle bunch is the source of the dispersion orbit.

The general solution to Equation (2.1) can be written as the superposition of Equation (2.2) and Equation (2.6):

$$\begin{aligned} z(s) &= z_\beta(s) + z_d(s) \\ z(s) &= \sqrt{\epsilon_z \beta_z(s)} \cdot \cos(\Phi_z(s) + \phi_z) + D(s) \cdot \frac{\Delta p}{p} \end{aligned} \quad (2.8)$$

The equations of motion alone however are not enough to fully design the beam optics of circular machine with potentially thousands of individual components that need to be taken into consideration. It is not possible to find the optimal betatron function and phase advance analytically and therefore numerical approaches have to be used which attempt to iteratively find the best configuration of parameters given the constraints of the machine. The parameters used to describe the beam optics are called *Twiss parameters*: a) the betatron function,  $\beta(s)$ , introduced in Equation (2.2), b)  $\alpha(s) \triangleq -\frac{1}{2}\beta'(s)$  and c)  $\gamma(s) \triangleq \frac{1+\alpha^2(s)}{\beta(s)}$  [25]. A general-purpose tool specifically

intended for optics design is MAD-X, distributed freely by CERN, and which is actively used by LHC operators to design new optics for the LHC [26].

## 2.2 LHC accelerator and its magnets

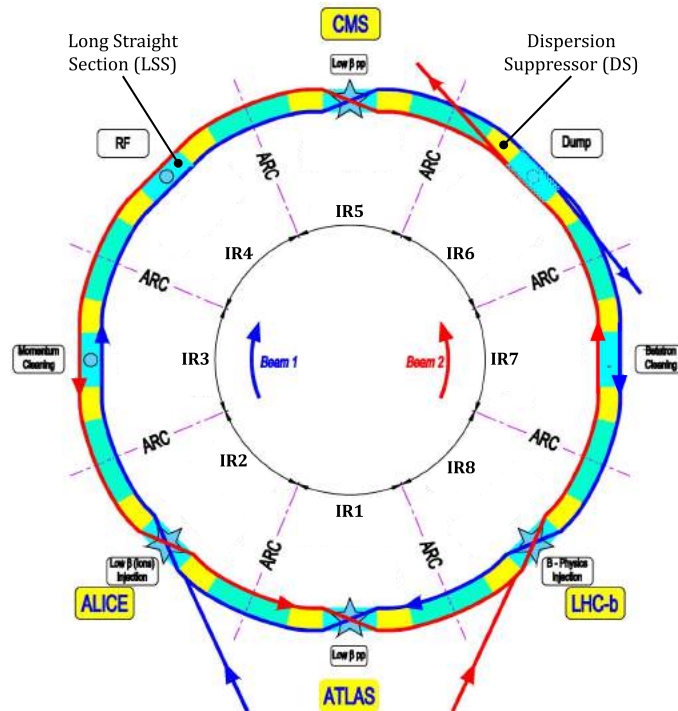


Figure 2.3: The LHC accelerator layout [2].

The LHC accelerator layout is not a perfect circle, instead it has 8 arcs joined by 8 straight sections. Figure 2.3 illustrates this layout where the arcs are coloured green and the Long Straight Sections (LSSs) are coloured light blue. The yellow regions are called the Dispersion Suppressors (DSs) and their role is threefold: 1) to adapt the reference orbit of the beams to the geometry of the LHC; 2) to cancel the horizontal dispersion; 3) and to help match the insertion optics. The two beams are shown by the blue and red lines, where beam 1 moves clockwise and beam 2 counter-clockwise. The beams themselves are not a continuous stream of particles but consist of bunches. These bunches are accelerated close to the speed of light by the Radio-Frequency (RF) system, which are also spaced by a minimum of 25 ns: this is equivalent to minimum distance between the bunches of  $\approx 7.5$  m. By June 2017 the maximum

number of bunches per beam was 2556, which marked a significant increase in the peak luminosity [27]. The bunched beams themselves can be both entirely made up of either protons or heavy-ions, as well as one proton beam and one heavy-ion beam at the same time.

An Insertion Region (IR) contains a LSS and its two adjacent DSs. The eight IRs of the LHC are balanced between experiments and accelerator utilities. It can be seen in Figure 2.3 that the beams swap places at four Interaction Points (IPs), which is where the beams are collided in the four main experiments of the LHC. The two high-luminosity general-purpose experiments, A Toroidal LHC ApparatuS (ATLAS) and Compact Muon Solenoid (CMS) are placed symmetrically on two ends of the accelerator at IR1 and IR5, respectively. The other two experiments, A Large Ion Collider Experiment (ALICE) and Large Hadron Collider beauty (LHCb) are placed in IR2 and IR8, respectively. IR3 and IR7 are dedicated to the beam cleaning systems and IR4 holds the two independent RF systems for the two beams. The two beams are injected from below the reference plane in the same LSSs as ALICE and LHCb, and they are inserted inside the beam pipes via vertical kicks from insertion dipoles. Lastly, there is also an independent beam dump system per beam, where fast-pulsed horizontal kickers and vertical septum magnets are used to extract the bunched beams and deposit them in 7.7 m long graphite blocks located in IR6.

The arcs are made up of 23 arc cells and each cell is made up of two symmetrical half-cells, each 53.45 m long. A cell contains the required magnets to form one period of the betatron oscillations described by Equation (2.2). Each half-cell contains one 6.63 m long cryostat known as the Short Straight Section (SSS), and three 14.3 m long cryo-dipoles, also known as the main dipoles (MB). One SSS contains one arc quadrupole (MQ) which can be either focusing (QF) or defocusing (QD), along with several corrector magnets. In particular, one horizontal arc orbit corrector (MCBH) is installed at each QF and similarly, one vertical arc orbit corrector (MCBV) is installed at each QD. The tuning quadrupole correctors (MQT) are also installed in some of the SSSs, which together can adjust the tune in the order of  $\pm 0.1$ . The MB, QF and QD are grouped by sector, respectively, and powered in series. On the other hand, all

the corrector magnets can be independently controlled.

The experimental IRs also contain what are known as inner triplet assemblies that mainly consist of quadrupoles which adjust the beta function of the beams around the IP. The value of the beta function at the IP, also known as the Beta star, must be as small as possible to ensure maximal luminosity. The inner triplets also consist of several corrector magnets, in particular to this work is a set of orbit correction dipoles (MCBX).

Outside the inner triplets, the IRs contains other types of quadrupoles, namely quadrupoles of similar construction as MQ (MQM) and quadrupoles with a larger aperture which are required for high beta values of the beam (MQY). Both MQM and MQY are adjoined with two different sets of orbit correctors, named MCBC and MCBY, respectively. The beam cleaning IRs contain collimators which strip halo particles from the beams. As a result this process creates higher beam losses in the immediate vicinity of the collimators. Since the probability of quenches are proportional to the beam losses, IR3 and IR7 contain the only normal conducting orbit correction dipoles (MCBW).

Name	Description	Count	$I_{max}$ [A]	$I_{rate}$ [As <sup>-1</sup> ]
MCBH	Hor. arc orbit corrector	375	55	0.5
MCBV	Ver. arc orbit corrector	375	55	0.5
MCBCH	Hor. orbit corrector	84	80/100	0.667/1
MCBCV	Ver. orbit corrector	84	80/100	0.667/1
MCBXH	Hor. orbit corrector	24	400	2.5
MCBXV	Ver. orbit corrector	24	400	2.5
MCBYH	Hor. orbit corrector	38	72	0.667
MCBYV	Ver. orbit corrector	38	72	0.667
MCBWH	Hor. single aperture warm orbit corrector	8	500	5
MCB WV	Ver. single aperture warm orbit corrector	8	500	5
MQTF	Focusing tuning quadrupole corrector	8	550	1.5
MQTD	Defocusing tuning quadrupole corrector	8	550	1.5

Table 2.1: Types of corrector magnets used by the Orbit Feedback Controller (OFC) and Tune Feedback (QFB). H and V denote horizontal and vertical, respectively. F and D denote focusing and defocusing, respectively [2].

Table 2.1 summarises the magnets which are relevant throughout the later chapters. The dipole corrector magnet names are suffixed with either H or V to signify whether they act on the horizontal or the vertical plane, respectively. The magnets MCBH(V), MCBCH(V), MCBXH(V), MCBYH(V) and MCBWH(V) are collectively called Orbit Corrector Dipoles (CODs) in the Beam-Based Feedback System (BBFS) nomenclature. The tuning quadrupole magnet name is suffixed with either F or D to indicate whether they are focusing or defocusing magnets, respectively. Other higher order multipole corrector magnets are required for stable operation of the LHC, however, the main focus of this work primarily relates to dipole and quadrupole corrector magnets.

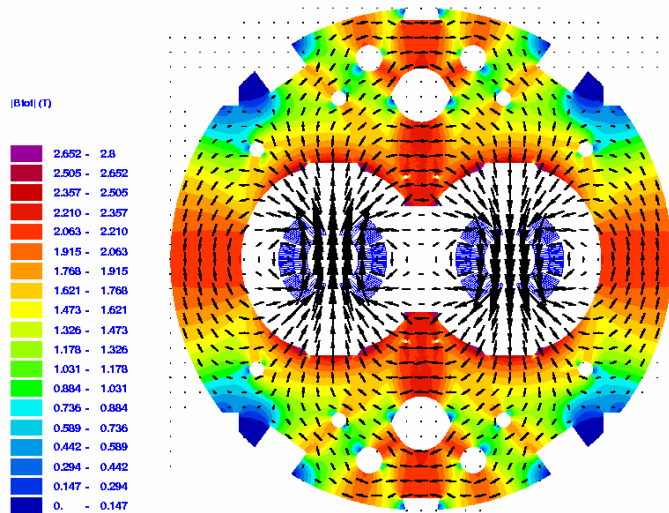


Figure 2.4: Simulated cross-section showing direction and magnitude of magnetic flux in a LHC dipole [2].

Dipoles provide a homogeneous, quasi-unidirectional magnetic field within the beam pipes as illustrated in Figure 2.4. The field in each beam pipe,  $\mathbf{B}_z$ , is perpendicular to the direction of motion of the ideal particle and serves to guide the beams around the accelerator. Most of the dipole magnets in the LHC constitute of superconducting coil configured as shown in Figure 2.5.

Quadrupoles provide a linearly varying magnetic field,  $B_z = -gz$ , where  $g$  is the magnetic field gradient. It is important to note from Equation (2.1) that if the quadrupole strength is positive in one plane, it must be negative in the orthogonal plane. As seen in Figure 2.6, if the poles are configured to focus the beams in the hor-



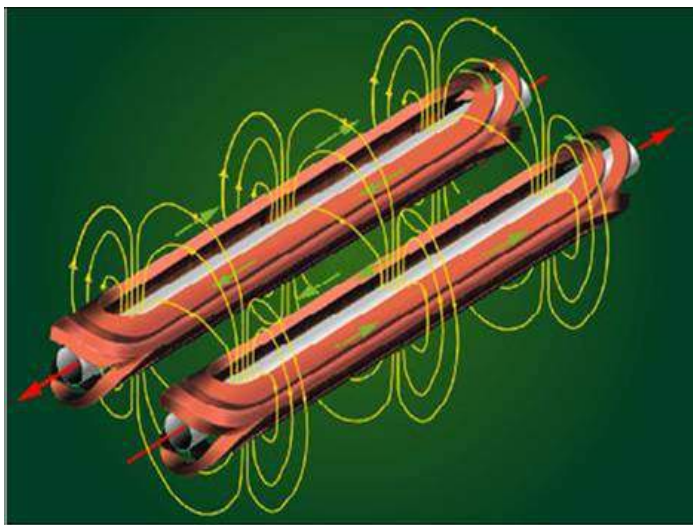


Figure 2.5: The superconducting coils and the magnetic field lines (yellow) of a LHC dipole[3].

horizontal plane, the orthogonal force components from the magnetic field would defocus in the vertical plane [4]. From Figure 2.6, it is also evident that if the quadrupole is perfectly symmetric, there would be no magnetic field on the central axis, and so the ideal particle would remain virtually unaffected by a quadrupole. Figure 2.7 shows a standard arrangement of QF and QD quadrupoles called a FODO cell. Two arc half-cells make up one FODO cell, which ultimately confines the beam envelope to within the beam pipes. Finally, Figure 2.8 shows a cross-section of an LHC superconducting quadrupole.

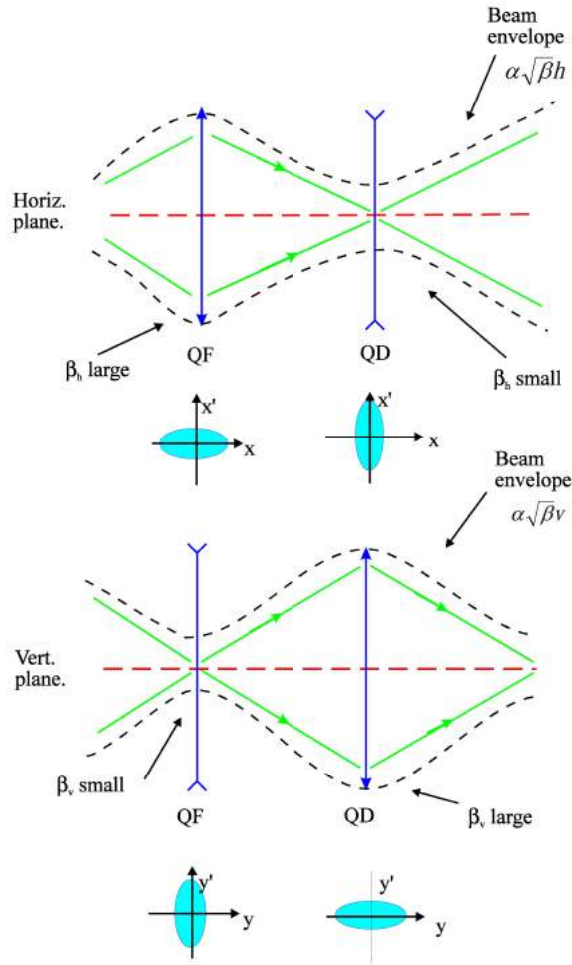


Figure 2.7: Beam envelope inside a FODO cell [4].

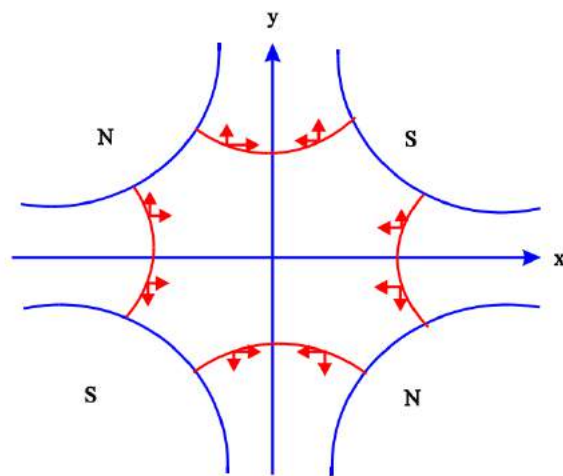


Figure 2.6: Forces acting on a particle moving through the quadrupole. Particle beam moves into the page [4].

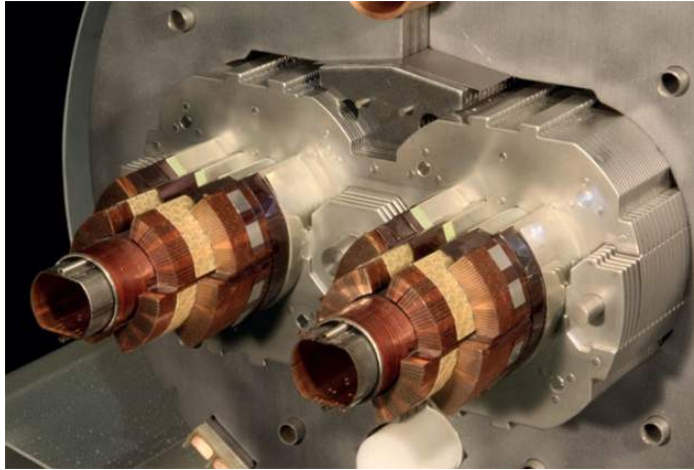


Figure 2.8: Cross-section showing the coils that provide the magnetic field in a superconducting quadrupole.

## 2.3 Beam Instrumentation

Beam instrumentation can be considered as a field of study within particle accelerator physics. The aim is to design, build and maintain the necessary hardware to perform measurements on the beam. Any upgrade that is done to a particle accelerator, will generally require more accurate measurements of key parameters in order to allow for effective controllers to be implemented. This also makes beam instrumentation an active field of study which seeks to improve the measurements done on the beam [28].

### 2.3.1 Beam Position Monitors (BPMs)

Beam Position Monitors (BPMs) are some of the most important beam instrumentation devices with respect to the needs of the OFC. BPMs are a non-destructive type of instrumentation which can measure the position of the center of mass of a bunched beam, through the induced electromagnetic field by the particle as it moves past the BPM. Various designs of BPMs exist and operate within a specific design objective, such as a trade-off between accuracy and acquisition rate. The LHC uses two different kinds of BPMs which are the button-type and the strip-line BPMs.

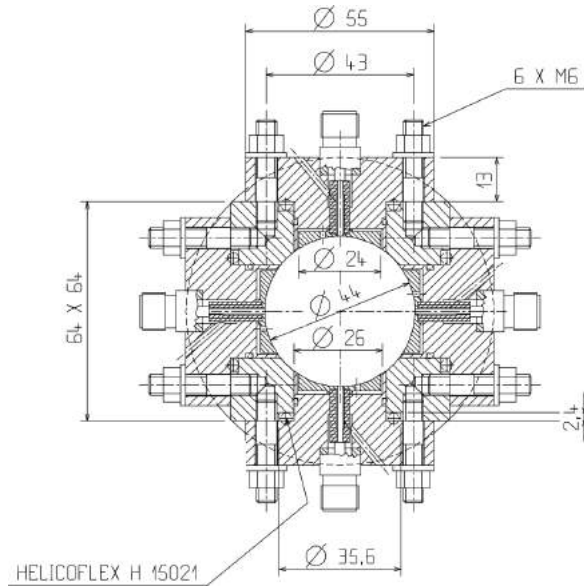


Figure 2.9: Cross-section of the button BPM showing the 4 electromagnetic pick-ups (buttons) [5].

The first type of BPM is one of the most compact designs for beam position measurement and is also a relatively simple design as shown in Figure 2.9. Therefore this means that it is not expensive to manufacture, which became a very important factor during the design phase of the LHC, as over 900 button BPMs were eventually installed in the accelerator. Its operating principle is relatively simple: the position of the beam in a plane is measured from the image current that is induced in two opposing buttons when a charged particle bunch passes by them through the vacuum. If the bunch is not passing exactly from the center of the beam pipe, the two opposite buttons will produce different image currents. The two readings are processed together by the BPM electronics, which output the position of the center of mass on the axis of the buttons. Another pair of buttons are placed orthogonally, to measure the center of mass on the other plane [6, 5].

The second type of BPM to be found in the LHC is the strip-line BPM. This kind of beam instrumentation is used when the two counter-rotating beams are in the same vacuum chamber in the experiment regions of the LHC. The strip-line BPM can discriminate between the two beams and find their positions respectively. The strip-line BPM operates as follows: The bunch enters the BPM and induces an image

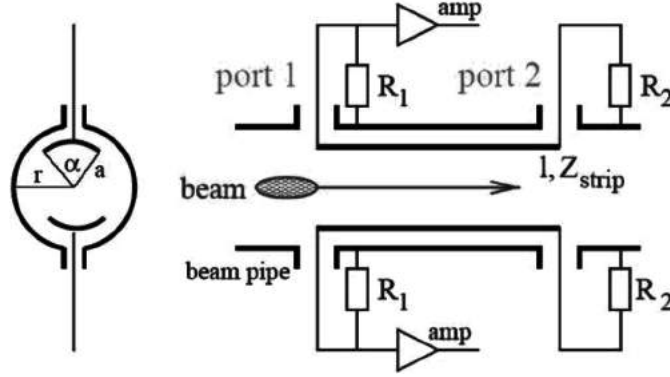


Figure 2.10: Simple strip-line BPM configuration with a matched load impedance [6].

signal at the first edge of the strip-line on port 1. The signal and the beam travel in phase to the direction of port 2. At the second edge of the strip-line another image signal is formed, but this time it travels in the opposite direction. Therefore half of the signal is cancelled by the propagating signal from port 1 and the other half of the signal propagates back to port 1. If a second bunch is seen at that time, the reflected signal from the first bunch is cancelled by the signal from the second bunch, and so on. The magnitude of the signal coming out of port 1 is then amplified, and passes through the electronics and the numerical methods needed to find the measurement for the beam position [6, 29]. A diagram illustrating the design of a strip-line BPM can be seen in Figure 2.10.

### 2.3.2 Tune Measurement and Feedback

The LHC Base-Band Q (tune) - (BBQ) system was the first quasi non-intrusive tune measurement in a synchrotron machine, which could increase the betatron signal levels by order of magnitudes whilst still observing the beam in its naturally excited form for short bunch lengths.

This system works on the principle of Direct Diode Detection (3D), which as shown in Figure 2.11, is comprised of an envelope detector which is normally used for demodulating Amplitude Modulation (AM) signals. The latter works by having the diode (D) letting only signals above its threshold voltage through and consequently filtering the low frequency components of the resulting signal  $s_d(t)$  via an RC circuit with the

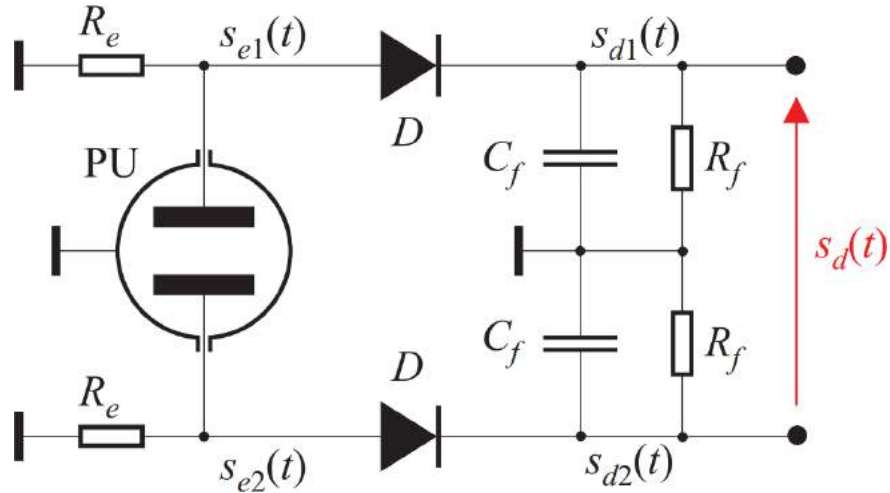


Figure 2.11: Principle of Direct Diode Detection (3D) [7] for the purpose of tune measurement.

time constant in Equation (2.9).

$$\tau = R_f C_f \quad (2.9)$$

The resultant effect of the circuit in Figure 2.11 is the time-stretching of the short bunch length signal spectra to baseband frequencies. As a result the betatron oscillations are conserved and can be observed in baseband with higher Signal-to-Noise Ratios (SNRs) with respect to its classical counterpart. This concept alone makes the Base-Band Q (tune) (BBQ) capable of accurately measuring the tune of a bunched beam without external excitation.

Classical tune measurement techniques did not use the 3D principle and instead directly estimated the tune from the  $s_e(t)$  signals obtained by the pick-up (PU) as shown in Figure 2.11. As shown in the work of Gasior et al., the BBQ system had outperformed the classical tune measurement systems by more than an order of magnitude in terms of the SNR [7].

Ultimately, the value of the tune is equal to the frequency of the largest mode of oscillation within  $s_d(t)$ . After converting the signal into frequency domain, this mode can be easily found by finding the position of the largest magnitude within the spectrum. It is also important to note that the general location of the tune in the

spectrum is inferred from the optics of the LHC, therefore only a small frequency window is considered when estimating the tune.

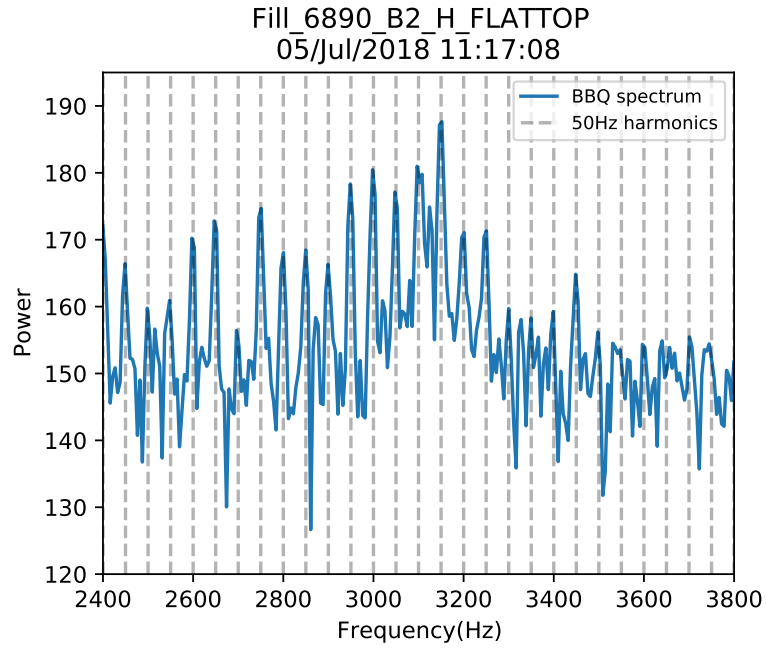
### Present tune estimation algorithm

The present tune estimation system, which in this text will be referred to as the BQ algorithm, was used in the LHC until the end of Run 2 as the main source of tune estimates. BQ uses the signal  $s_d(t)$  as shown in Figure 2.11 sampled at 11 245.55 Hz; the LHC revolution frequency. Every 2048 samples, a Fast Fourier Transform (FFT) is performed to obtain a spectrum containing 1024 frequency bins. Following this, the spectrum is clipped to a specific frequency window chosen by the LHC operators, roughly centered around the expected tune. Figure 2.12a shows an example of a resultant spectrum after these operations. The resultant spectral resolution,  $f_{res}$  can then be found by considering that the Nyquist frequency of the BBQ spectrum is half the LHC revolution frequency. Equation (2.10) finds the spectral resolution of a BBQ spectrum.

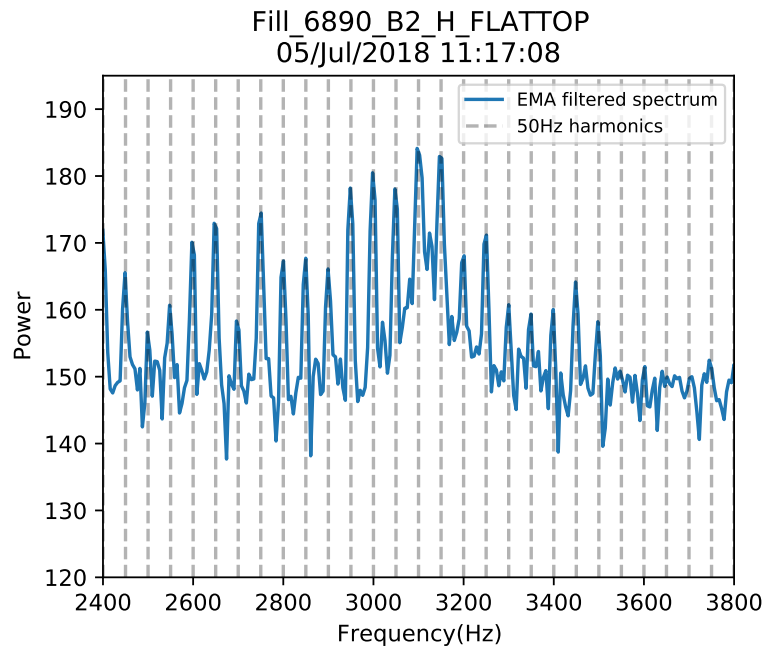
$$f_{res} = \frac{F_s}{2N_f} = \frac{11245.55}{2 \times 1024} \approx 5.5 \text{ Hz} \quad (2.10)$$

The BQ algorithm has been in operation since the BBQ system was developed and throughout the years it has had some small design upgrades which were put in place to improve its performance. BQ is comprised of a set of sequential processing blocks as depicted in Figure 2.13. First each frequency spectrum obtained from the BBQ system is passed through a bank of independent Exponential Moving Average (EMA) filters, where each filter works on a single frequency bin. This pre-processing stage is essentially used to reduce spectral white noise. Figure 2.12b is the same spectrum in Figure 2.12a, after the EMA smoothing operation.

Median and average filters are subsequently applied to the EMA filtered spectrum to increase its smoothness with respect to the tune peak. At this stage the frequency corresponding to the maximum value of the processed spectrum is taken. This frequency is then refined by going back to the output of the bank of exponential moving



(a) Spectrum obtained directly from BBQ signal which is picked up by PU.



(b) Spectrum with filtered frequency bins which is used by BQ.

Figure 2.12: A comparison of the BBQ and the EMA filtered spectra from operational data logged during the LHC Run 2 in 2018.



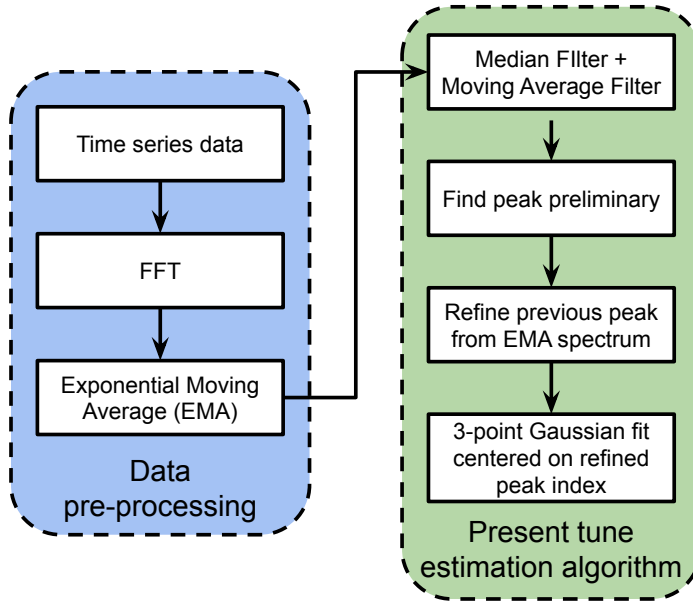


Figure 2.13: Present tune estimation algorithm diagram [8]

averages and performing a Gaussian fit of the spectral region in its immediate vicinity. The Gaussian fit essentially corrects the location of the maximum by fitting the maximum found earlier, and its 2 adjacent points onto a Gaussian distribution and after set the new maximum location at the mean of the Gaussian [19, 30].

### Performance issues

Since the first time that the BBQ system was switched on, spectral components at mains frequency have been observed and confirmed by independent diagnostic systems. The origin of these 50 Hz noise harmonics has been identified by recent studies to be the main dipoles and therefore these harmonics are present in the beam itself and are not a product of the instrumentation [31]. Figure 2.12a provides an example of BBQ spectrum polluted with 50 Hz noise harmonics where it can be easily understood why these harmonics might affect the tune estimation process as described in Section 2.3.2.

The top plot in Figure 2.14 shows the tune trace of the resultant tune estimates of the BQ algorithm from EMA filtered spectra during an LHC fill in 2018. Here we can observe that BQ is estimating a tune value which is coincident with two adjacent 50 Hz noise harmonics. First we can observe Points A and B which are providing a relatively

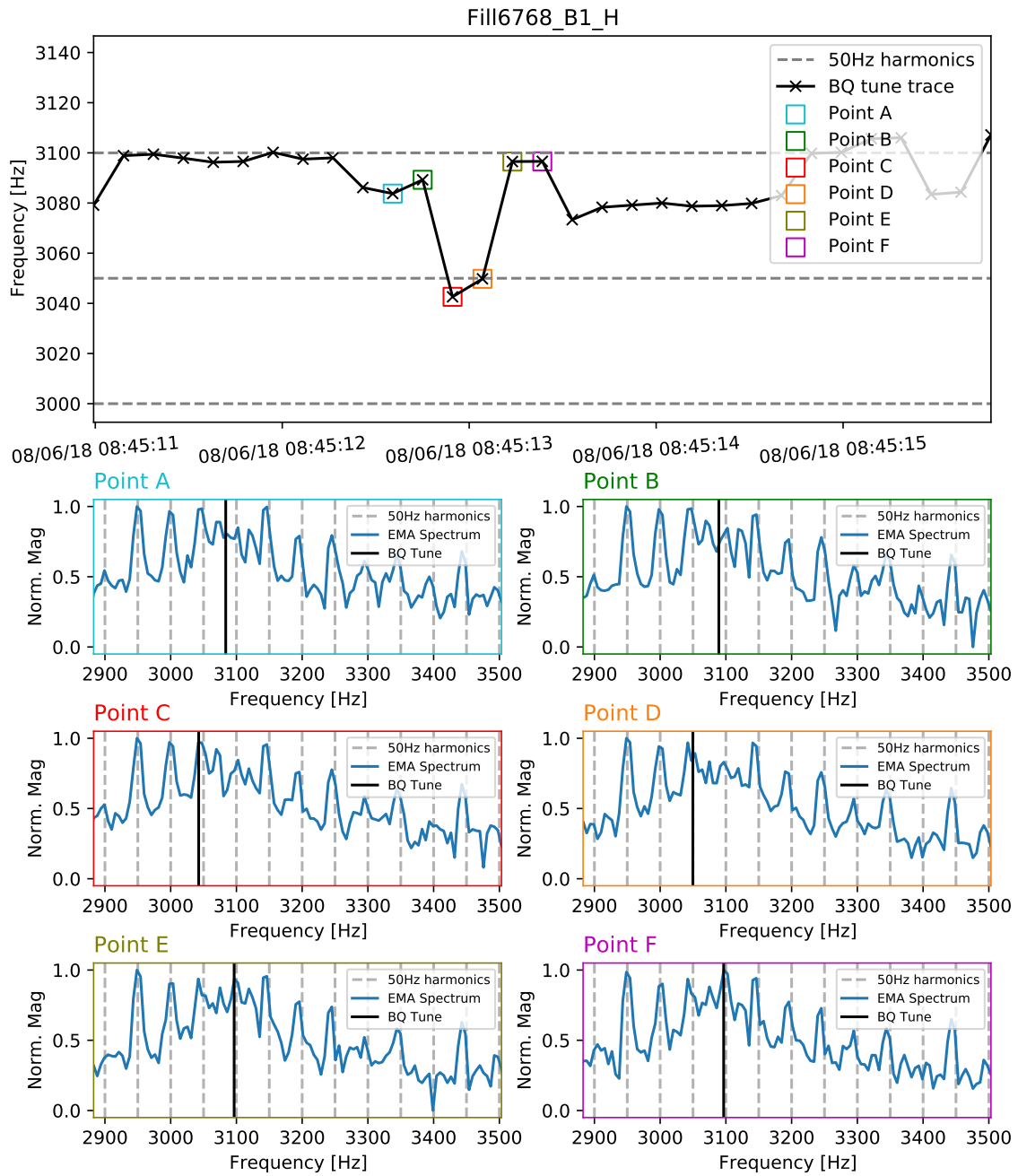


Figure 2.14: BQ tune trace showing adjacent harmonic jumps. 6 points on this tune trace were chosen and their respective spectra are shown on the bottom plots.

good estimate, however the tune estimate jumps at Point C and the tune estimate is coincident with the 50 Hz harmonic at 3050 Hz until Point D. After one time step at Point E, the tune estimate becomes coincident with the next 50 Hz harmonic at 3100 Hz. The spectra of Points E and F show the magnitude growth of the harmonic at 3100 Hz which caused the error in the tune estimate.

Figure 2.15 shows a similar plot from the same fill, after approximately one minute of real operation. Now Points A to C are following the tune trace which is coincident with the 3100 Hz harmonic that was noted in Figure 2.14. Following this we can see from the respective spectra of Points D and E that the tune peak has somewhat shifted to left by a few frequency bins, which finally allowed the BQ algorithm to avoid the harmonic peak. Point F shows the tune estimate jumping again to the same harmonic. This example shows the sensitivity of BQ to small changes in the tune peak which might cause it to jump to a harmonic and no longer measure the tune. From this example it can also be seen that since the tune peak itself is coinciding with a harmonic, the behaviour of BQ becomes undefined since then the estimate directly comes from the position of the peak of the harmonic.

Figure 2.16 shows the tune trace from BQ on a different fill where the focus is on a sample of the estimates which exhibits a large variance. Here we can observe that initially, Points A and B provide very similar tune estimates, however at Points C and D the tune estimate drops by approximately 70 Hz. After another time step we can see at Point E that the tune estimate has jumped by approximately 120 Hz. The latter is equivalent to almost 22 frequency bins of variation from consecutive estimates. At point F the tune estimate falls closer to Points A and B again. This example illustrates that when the tune peak is wide within the spectrum, BQ suffers from large variance in adjacent tune estimates.

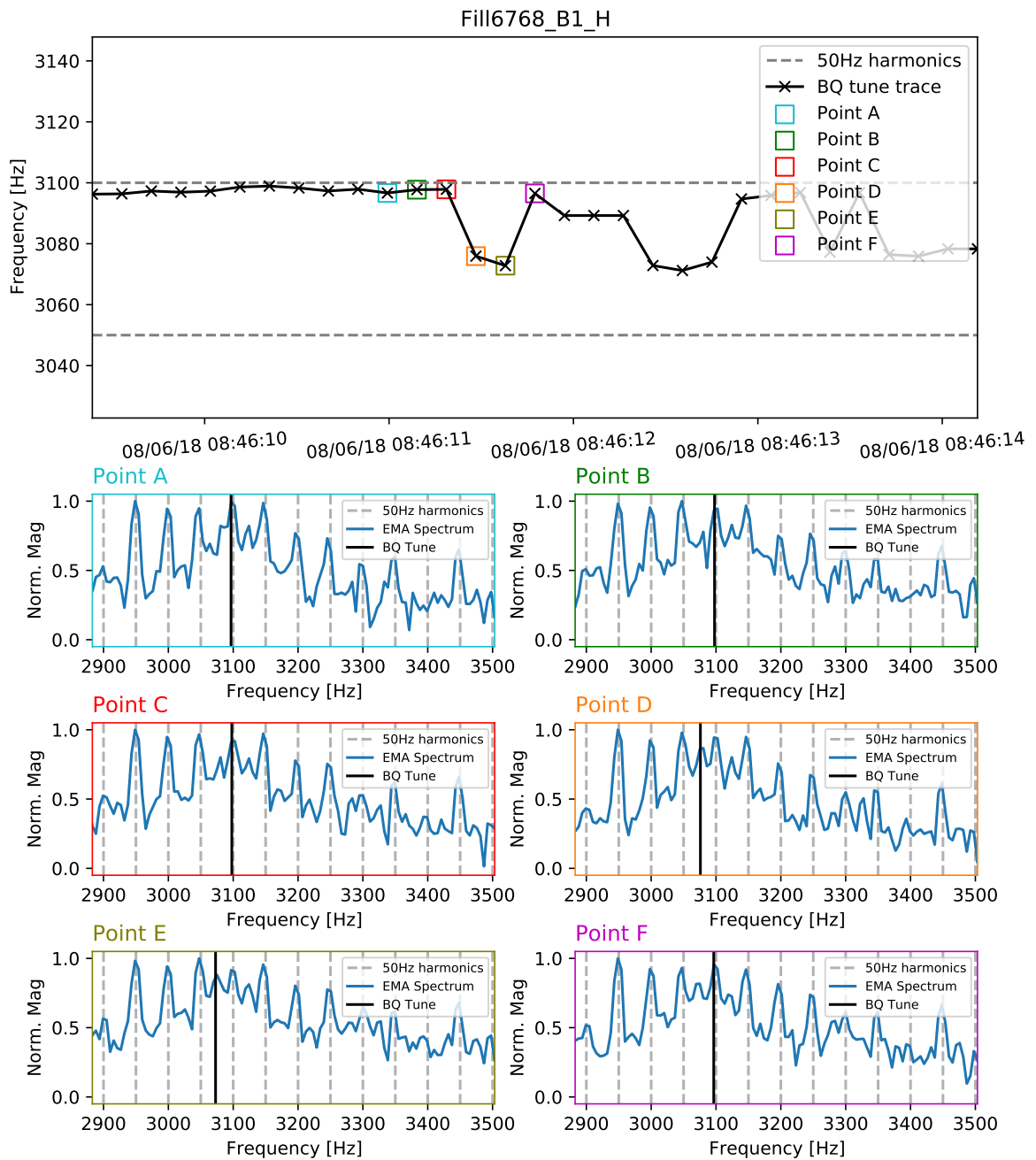


Figure 2.15: BQ tune trace showing estimates coincident with a 50 Hz noise harmonic. 6 points on this tune trace were chosen and their respective spectra are shown on the bottom plots.

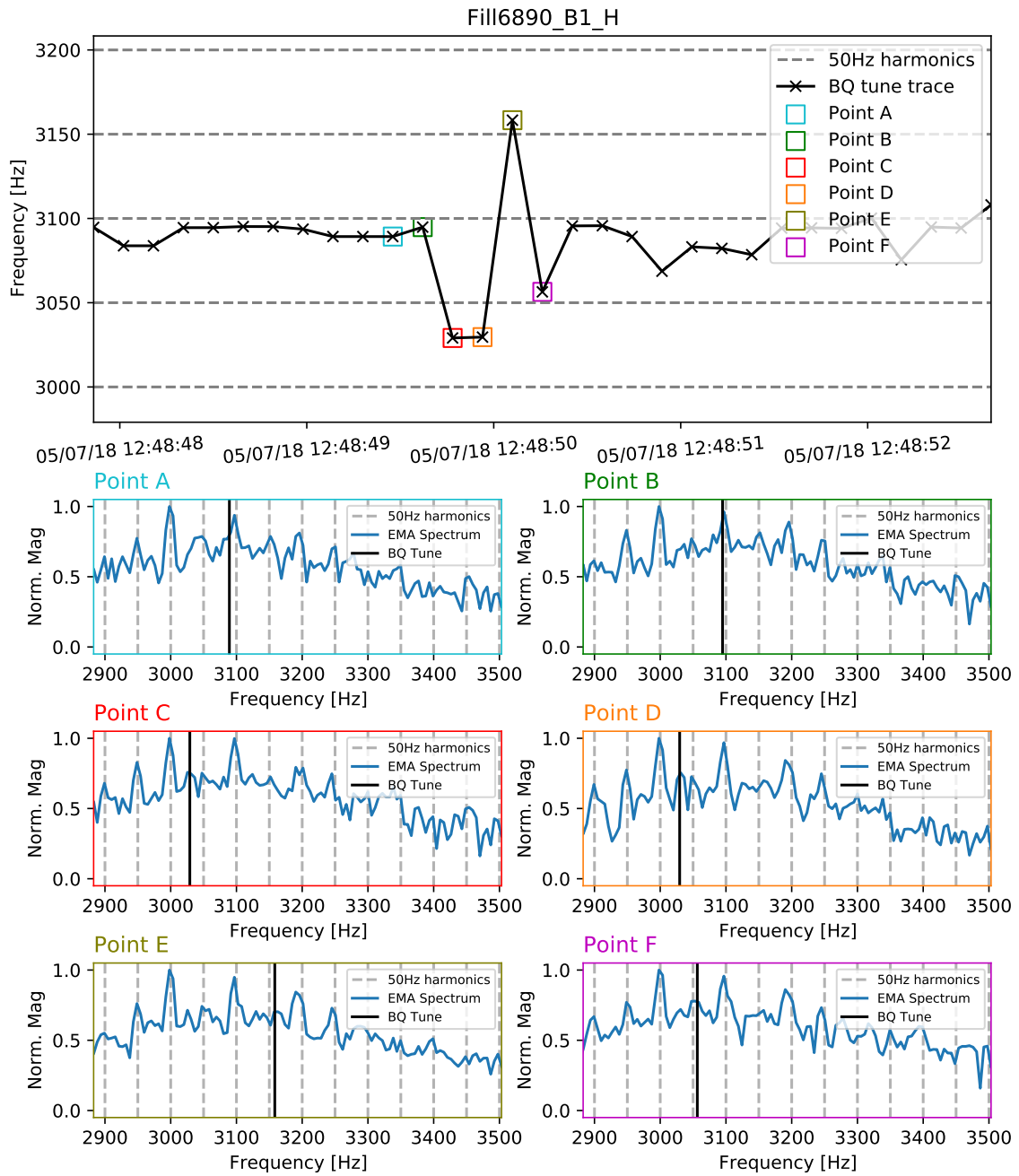


Figure 2.16: BQ tune trace showing large variance. 6 points on this tune trace were chosen and their respective spectra are shown on the bottom plots.

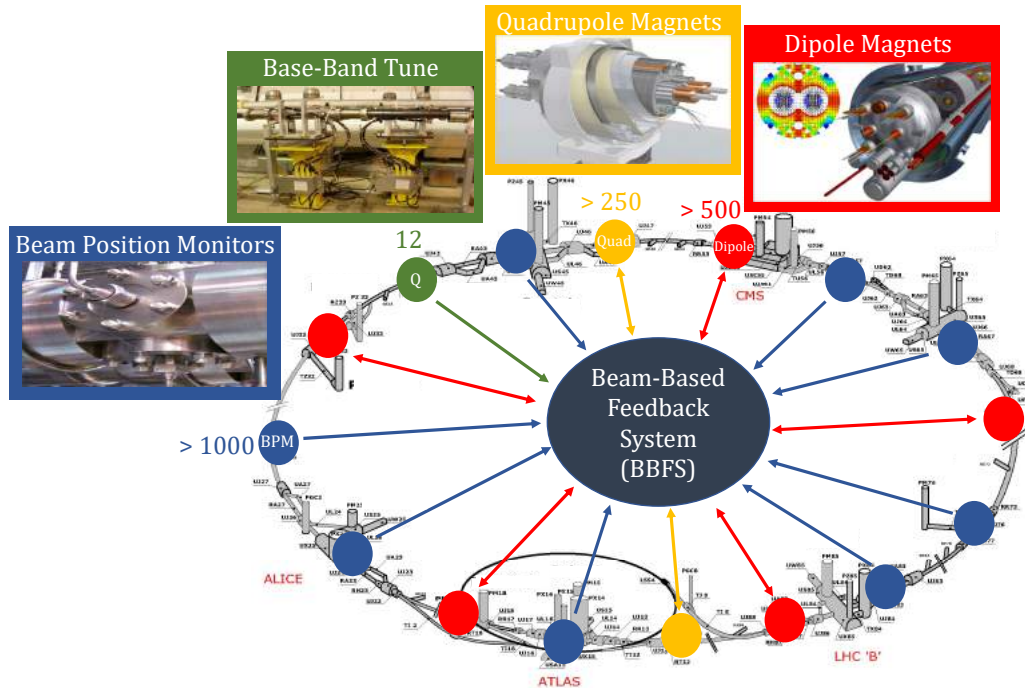


Figure 2.17: Schematic view of the Beam-Based Feedback System (BBFS) in the LHC complex.

## 2.4 The LHC Beam-Based Feedback System State-of-the-Art

Figure 2.17 is a schematic overview of the Beam-Based Feedback System (BBFS), which represents all the beam-based feedbacks as they were implemented within the LHC complex up until Run 2. The basic layout of the LHC is shown in the background along with the four main experiments labelled in red text. Four essential components for the BBFS are also shown which are the Beam Position Monitors (BPMs), Base-Band Tune (BBQ), quadrupole magnets and the dipole magnets. The direction of the arrows linking these components to the BBFS indicates the direction of the data flow to and from the feedback system.

Figure 2.18 illustrates a more detailed view of the BBFS architecture which is comprised of the Orbit Feedback Service Unit (OFSU) and the Orbit Feedback Controller (OFC). The OFSU is connected to the OFC via a private Ethernet connection

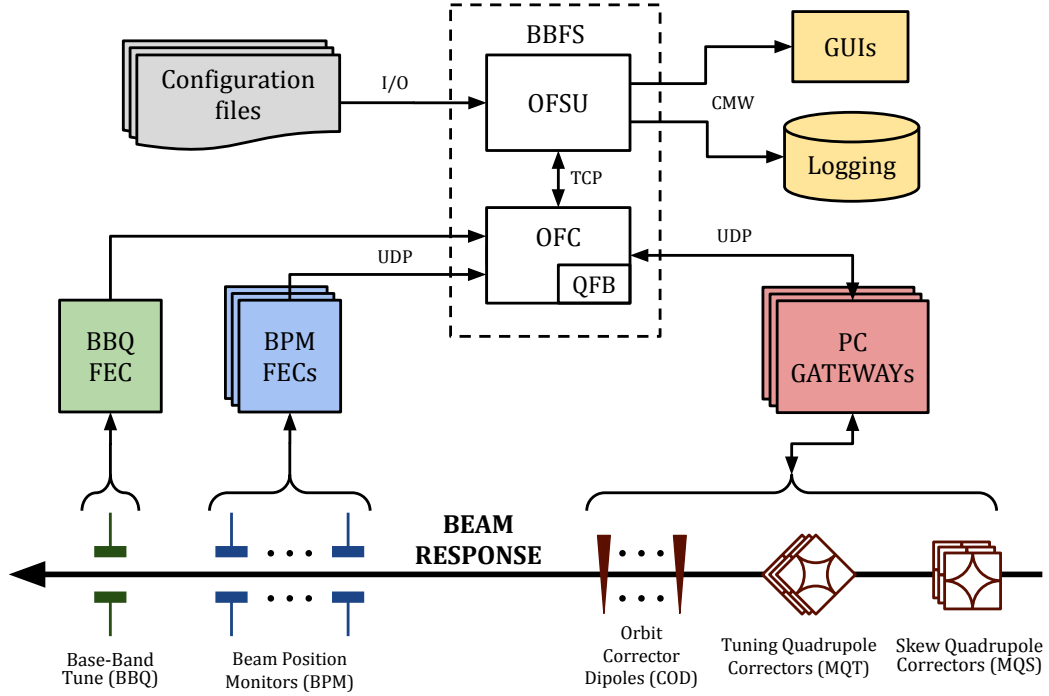


Figure 2.18: Schematic view of the data paths and a more detailed view of the BBFS architecture.

and Transmission Control Protocol (TCP) is used for lossless communication. The Tune Feedback (QFB) is enclosed within the OFC and is virtually hidden from the outside. Data from the BPMs and the BBQ is received by the OFC in the form of User Datagram Protocol (UDP) packets. The BPM packets arrive from 118 Front-End Computers (FECs) which can interface up to 18 BPMs each. The BPMs are not connected to FECs in a sequential order, rather they are interlaced to increase robustness in the presence of FEC failures. Similarly, the Orbit Corrector Dipole (COD) and tuning quadrupole packets arrive from, and are sent to the Power Converter (PC) gateways. The PC gateways are connected to the Function Generator/Controllers (FGCs) which control the magnets themselves [32].

This configuration allowed the Orbit Feedback Controller (OFC) to collect information about the beam tune and orbit in real-time, whereby it can calculate the required corrections to the magnets at a rate of 25 Hz [1]. The OFC was designed in C++ with the aid of ROOT libraries. ROOT is a scientific software framework originating from CERN with the purpose in mind to be used for the processing, analysis, visualisation

and storage of big data related to high energy and nuclear physics [33]. The limited responsibility of the OFC with respect to all the tasks required to be performed by the BBFS was prompted by the computational limitations of the hardware in place at the start of the LHC operation. By keeping the program structure of the OFC simple, the designers could ensure a deterministic execution.

Figure 2.18 also shows that the OFSU served as an interface for the OFC and was used by the LHC operators to monitor and control the OFC [34]. The OFSU was also responsible for loading the configuration files, which contained the settings and beam optics required by the OFC. This information was communicated to the OFC upon initialisation as well as by request of the operators. The OFSU also served as a hub for the beam orbit data collected by the OFC and as a result a myriad of Graphical User Interfaces (GUIs) used by the LHC operators depended on the data relayed by the OFSU. Finally the OFSU was responsible for logging the data collected by the OFC along with the settings of the BBFS itself. The OFSU was designed in CERN's own Front-End Software Architecture (FESA) and the sheer number of tasks described above made the OFSU the largest FESA class in CERN until the LHC Run 2 [35, 36, 37].

In its initial implementation the BBFS comprised of a series of nested feedback loops, each controlling a specific parameter of the LHC. First and foremost was the orbit feedback loop, which calculated the change in COD deflections necessary to kick the beam trajectories back to the reference orbit. Second was the energy feedback loop, which calculated the change in frequency required in the RF cavities to counteract any systematic momentum offset introduced by the CODs during orbit correction. The other three nested feedback loops controlled the tune, coupling and chromaticity simultaneously. It is important to note that in the latest implementation of the OFC, only the QFB remained. Due to the instabilities observed in the tune estimates, subsequent derived measurements, e.g. coupling and chromaticity, were not reliable enough to merit a feedback system. Therefore when adjustments on the coupling and chromaticity were required, manual operator-calculated corrections had to be uploaded to the OFC [38]. The QFB was used to correct the tune in critical operational modes



of the LHC; however, it was only used sparingly since frequent feedback instabilities caused it to shut itself off. It was suspected that erroneous tune estimates from the BBQ system were the cause for these failures.

The response of the currents in the CODs on the BPM measurements is estimated by a RM. The main principle behind the orbit correction performed by the OFC is to (a) measure the beam position using the BPMs, (b) calculate the error with respect to the reference orbit, (c) use a Proportional-Integral (PI) controller as a feedback loop (d) calculate the change in current needed in each COD to correct the beam position and (e) finally send the current corrections to the CODs. A similar procedure is used by the QFB to correct the tunes in both planes for both beams with the main difference from the OFC being that the currents of tuning quadrupoles are changed to correct the tunes.

Several methods exist which calculate the required COD currents and the most popular are the MINimisation des Carrés des Distortions d'Orbite (MICADO) and SVD [39]. The MICADO algorithm optimises the global orbit of the beam by finding a set of the most effective CODs in the lattice which minimise the Root Mean Square (RMS) by using as few corrector kicks as possible. The SVD algorithm is used in a different approach, by which the RM is decomposed into three more manageable matrices from which the so called Pseudo-Inverse (PInv) is found. The PInv directly relates the required COD magnetic field strengths as a function of the measured orbit error. During the early design of the OFC it was decided that the SVD algorithm was a more robust and faster orbit correction algorithm than MICADO for use in the OFC [1].

The following is a basic description of the PInv and the SVD algorithm. The pseudo-inverse of a matrix  $A$ , is the matrix which when multiplied to  $A$  produces an identity matrix. It is similar to an inverse of a square matrix with the difference being that  $A$  can be any rectangular matrix. The PInv is calculated after applying SVD on  $A$ , which allows us to express  $A$  as a multiplication of three sub-matrices with special properties, that make it possible to find the PInv matrix. Below is the result of SVD

on  $A$ :

$$A = U\Sigma V^*$$

where  $U$  and  $V$  are unitary matrices, meaning that their complex conjugate is their own inverse and  $\Sigma$  is a rectangular diagonal matrix, with non-negative real numbers on its diagonal and zero padding to allow for matrix multiplication. The diagonal values of  $\Sigma$  are also known as the singular values of  $A$  and the columns of  $U$  and  $V$  are known as the left singular vectors and the right singular vectors of  $A$ , respectively [40]. Therefore:

$$\begin{aligned} \text{PInv} &\triangleq A^{-1} \\ A \cdot A^{-1} &= I \\ (U \cdot \Sigma \cdot V^*) \cdot A^{-1} &= I \\ A^{-1} &= V \cdot \Sigma^{-1} \cdot U^* \end{aligned} \tag{2.11}$$

where  $I$  is an identity matrix and  $\Sigma^{-1}$  is the inverse of  $\Sigma$ .

In the OFC, the number of singular values is purposefully reduced and corresponds to COD deflections which are less localised. This is an attractive property of the SVD algorithm since that during collisions, it is preferable to correct the beam orbit with slow changing COD deflections [1]. For this reason, depending on the LHC beam mode, a PInv with different qualities is required. In standard operation, the number of eigenvalues used to build PInv from Equation (2.11) is reduced from 550, down to 40. Figure 2.19 illustrates the Pseudo-Identity (PI<sub>d</sub>), which is the result of the multiplication of a PInv using 390 singular values and the RM. The change in PI<sub>d</sub> is more pronounced in Figure 2.20, obtained by a PInv using 40 eigenvalues, and used during collisions. It can be seen that the PI<sub>d</sub> matrix obtained with 40 singular values has a wider diagonal which corresponds to a less localised response and gentler changes in the COD deflections.

The SVD algorithm has to be employed whenever there is a change in the RM of the LHC as well as when there is a COD or BPM failure. When a COD does not

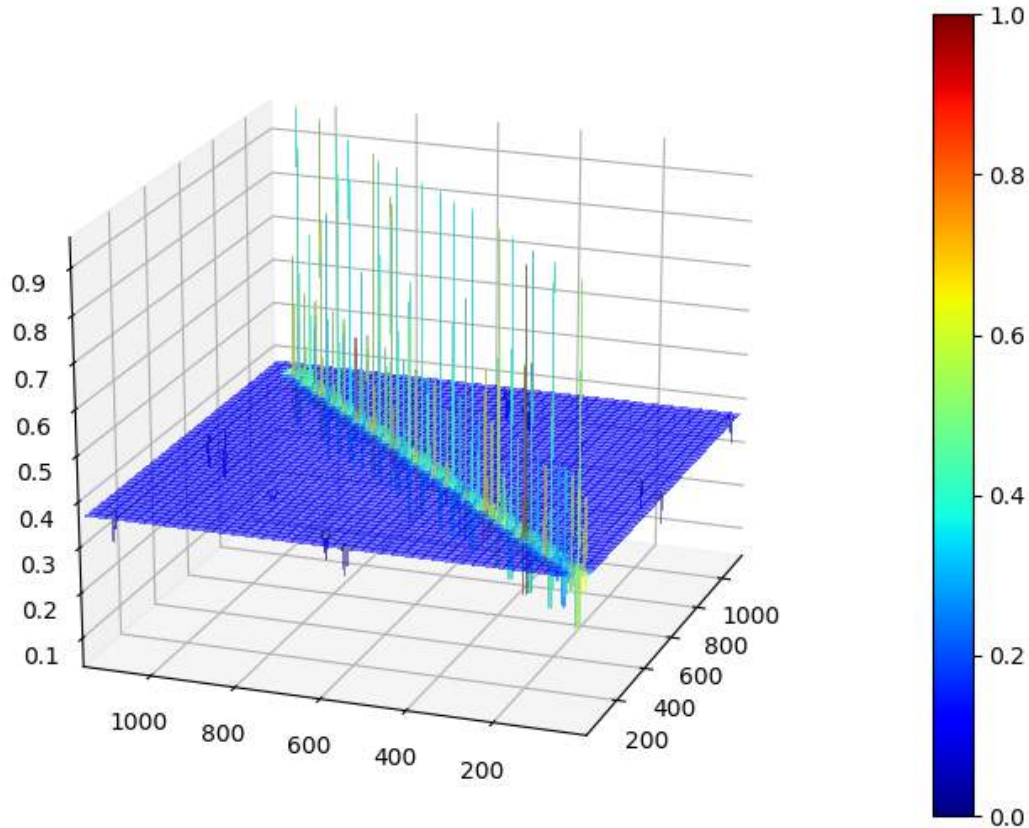


Figure 2.19: Pseudo-identity matrix obtained by multiplying the pseudo-inverse calculated with the first 390 eigenvalues, to the response matrix

provide the magnetic field that the OFC has requested, the beam position will start to deviate from the reference orbit. As a result, the PI controller keeps increasing the integral error to no effect, which may lead to a beam dump due to the beam shifting too close to the sides of the vacuum chamber. This problem can be mitigated if the RM is altered to account for the malfunctioning COD and having the PInv recalculated.

If the current corrections send to the CODs and tuning quadrupoles are too large, it could be the case that the beam response become non-linear. This arises due to a magnet not being supplied enough current to apply the corrections requested by the controller. To avoid this problem, both the OFC and QFB were designed to scale down all the corrections by the same factor to accommodate the magnet with the largest current delta relative to its maximum allowable current rate. The resultant effect is a sub-optimal correction, however, convergence is still ensured.

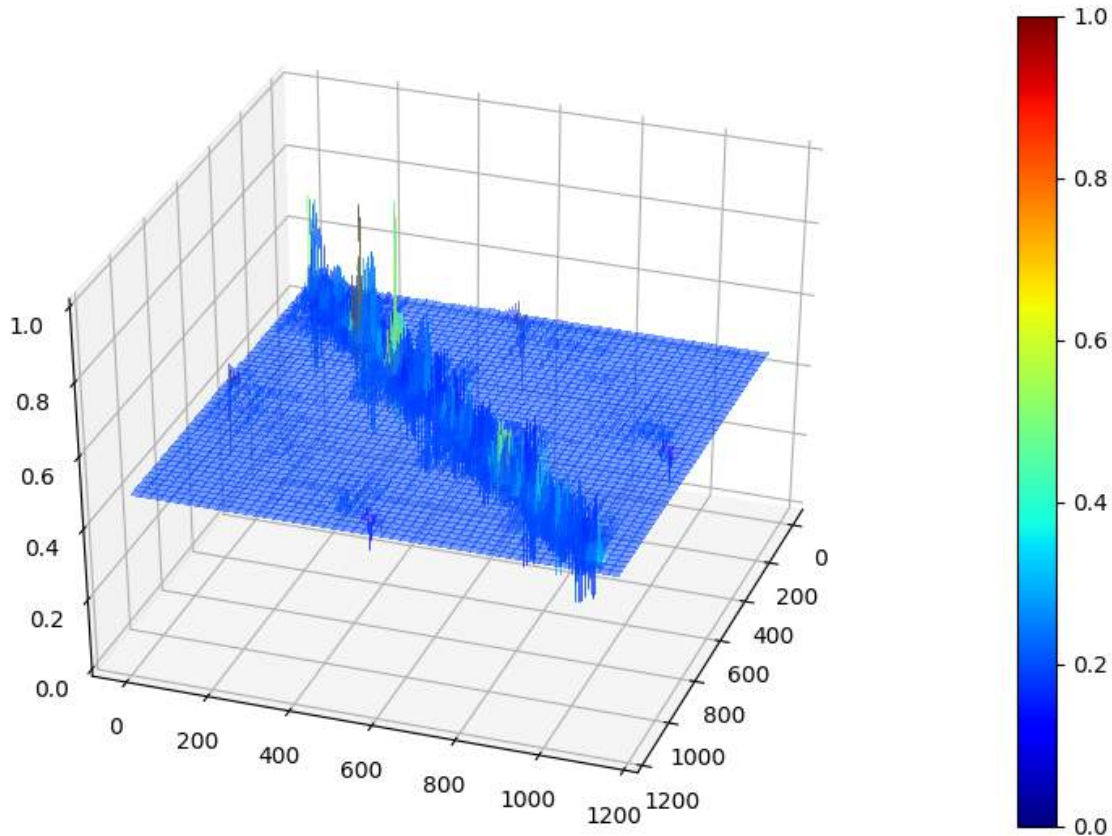


Figure 2.20: Pseudo-identity matrix obtained by multiplying the pseudo-inverse calculated with the first 40 eigenvalues, to the response matrix

### 2.4.1 Limitations

When the positional error vector is multiplied with the PInv, the entire set of corrector magnets included in the RM is used. Applying a correction involving the use of a malfunctioning COD may lead to significant distortions in the entire beam orbit. This occurs due to the absence of a kick at the location of the malfunctioning COD. The PInv attempts to restore the missing kick as given by the optics model. As a consequence, in the event of a corrector failure the feedback must be stopped and the SVD recomputed without that corrector.

In the current implementation of the OFC, the SVD algorithm is not used during operation since its calculation takes in the order of tens of seconds to be executed, making it unfeasible to be used in real-time. As a workaround, when a malfunctioning COD is found, the LHC operators mask the current corrections sent to it. Conse-

quently, the calculation of a new PInv has to be manually started offline on the OFSU then uploaded to the OFC. This procedure has been shown to work in the past with non-critical CODs however with a time limit of around one minute. This is not a sustainable solution mainly due to the possible unavailability of expert operators to complete the task [37, 38].

In the case of a BPM malfunctioning, the LHC operators mask the measurements of the respective BPM and set the beam position at that location to be the reference orbit. This may also be done automatically by the OFC itself and is accomplished by reading the hardware state given to it by the BPM UDP packets. Ideally, following the masking of a BPM the PInv is recomputed immediately, however, due to the same limitations for malfunctioning CODs and the lessened impact of BPM failures on operation, this is never done in the OFC until LHC Run 2.

Ultimately, in the current OFC implementation, the required COD deflections are calculated using the original PInv without taking into consideration the malfunctioning BPM. The result is that the beam position at the location of the BPM malfunction is left relatively unaffected due to the assumption that the position at that location is at reference orbit [37, 38]. This is a reasonable solution when a small number of BPMs are malfunctioning in sparse locations, however should it be the case that adjacent BPMs are defective, the OFC might induce significant beam distortions. Thus it would be a significant improvement to recalculate the PInv immediately and in real-time following any failure.

It is important to note that for an efficient and accurate correction, the OFC relies on the quality of the RM. Two factors may affect the quality, the status of the elements (BPMs or CODs) and the accuracy of the optics (machine versus model). The typical optics errors (beta-beating) are in the range of 1-10% which translates into errors of similar amplitude on the elements of RM. During the commissioning of a new machine configuration or optics, the errors may reach up to 30-40%. When the gain of the PI controller was not too high and the number of singular values in the PInv was small, the orbit corrections have always converged and this has been confirmed by simulations [1]. Non-functioning BPMs or CODs must be removed from the matrix

before applying the SVD, for example by zeroing the corresponding column or row of RM.

The LHC correctors are grouped into 2 main categories. Correctors with maximum currents between 70A and 600A ( $\approx 300$ ) are installed in the Long Straight Sections (LSS) and in the beginning of the arc sections. They are connected by hardware link to the PIC powering interlock system. In case of a powering failure a dump trigger is generated immediately by the PIC. The  $\approx 750$  arc correctors with a maximum current of 60A are surveyed by the Software Interlock System (SIS). As soon as SIS detects a failure of a corrector, a dump trigger is generated if the corrector deflection is larger than  $5 \mu rad$ . For smaller deflections, it is possible for the OFC to handle the problem.

Since the RMS kick strength of the arc correctors is around  $12 \mu rad$ , the beam is dumped for the majority of corrector failures. The beam dump due to corrector failure occurs with a frequency of 5 to 10 events per year, resulting in a few hours of downtime each time and is therefore undesirable. The circuit time constants of those arc corrector ranges between 20 and 60 seconds depending on the normal conducting cable length. This time is sufficiently long to open a “beam rescue window” provided the OFC detects the failure immediately using the state information stored in the FGC UDP packets and is able to update the PInv accordingly within no more than 2 seconds. The missing corrector will leave a bump on the orbit, but it is possible to continue operation in those conditions [41].

It has also been observed during the LHC operation, that the QFB becomes unstable multiple times per fill and as a result is automatically turned off. The source of the problem was not immediately clear, however after analysis of logged data it was found that erroneous tune estimates from the BBQ system were causing the QFB instability. The erroneous tune estimate examples shown in Figure 2.14, Figure 2.15 and Figure 2.16 are relevant when viewed from the perspective of the QFB. Since the QFB is essentially a Proportional-Integral (PI) controller, the error in the estimate will result in a large control error within the QFB. The Integral part of a PI controller accumulates past errors in order to compensate for any residual errors, the estimation errors will be accumulated within the QFB resulting in stochastic behaviour.

To maintain determinism within the QFB, it was decided that a tune estimate stability metric had to be set up in order to automatically switch off the QFB in the presence of instability. This is done by keeping two Exponential Moving Averages (EMAs) with time constants of 2 and 10 seconds respectively, of the difference between adjacent tune estimates (See Equation (3.15) in Section 3.5).

During operation, when both the slow and fast stability measurements are above a certain threshold, the tune estimates are considered to be too unstable to be used for control. As a result, the QFB is automatically switched off. Also note that large variances, such as that shown in Figure 2.16, will cause the stability measurement to get a kick and consequently marking the estimate as more unstable. The automatic switching off of the QFB was not an uncommon event, and could occur multiple times per fill. Due to the increased frequency of the feedback failures, constant operator supervision was required.

Chromaticity feedback in the OFC was based and designed on chromaticity values derived from tune estimates, which created a significant margin of error. This made it difficult to effectively implement the chromaticity feedback in previous years and thus it is not currently in use in the LHC. Instead, operators monitor the chromaticity and correct it manually if necessary through offline calculations.

## 2.5 Hardware acceleration

Hardware Acceleration (HA) is defined as the use of specialised computing hardware, that can execute certain typed of programs more efficiently than general-purpose Central Processing Units (CPUs). Initially, HA was intended for faster graphics rendering for computer displays and introduced by Graphical Processing Units (GPUs). Originally GPUs were introduced in the market by NVIDIA for the sole purpose of rendering graphics on computer monitors. The unusual architecture of such devices created their potential to be used to accelerate general-purpose computing [42]. While today there exist many GPU manufacturers, with the main two brands being NVIDIA and AMD, NVIDIA is often considered at the forefront of the competition [43, 44].

NVIDIA also developed a parallel computing platform specialised for their GPUs called CUDA. CUDA gives the programmer the possibility to exploit the parallel architectures of both modern CPUs and GPUs for mathematical calculations [45]. There exists another parallel computing platform, OpenCL, which unlike CUDA is non-proprietary and royalty-free [46]. In this work only NVIDIA devices were considered, primarily due to their higher availability as offered by Techlab at CERN [47]. This made it paramount to understand the difference among each of the three families of NVIDIA GPUs.

The first and most popular one being the GeForce family of GPUs which mainly focus on the gaming industry as well as optimised hardware specifically for real-time game rendering. Consider as an example their most recent GeForce RTX line of GPUs which focus on optimising ray tracing, a heavily-used concept in game engines.

The second family is called Quadro, which is more focused on the professional sector such as companies using Computer Aided Design (CAD) and photo-realistic rendering. These GPUs offer highly accurate vectorised images to the computer's monitor and are normally installed in professional workstations rather than in a personal gaming computer. This is done through a significant increase in double precision Floating Point Operations (FLOPs) when compared to their GeForce counterparts. Their price tag also sets them apart from the GeForce family, with the Quadro family usually being one order of magnitude more expensive [48].

The third and last family is called Tesla, which is the General-Purpose Graphical Processing Unit (GPGPU) family from NVIDIA. This is a special kind of GPU, where its purpose is not to render images for a computer monitor, but solely to aid in mathematical calculations over a large amount of data. GPGPU are conventionally used for simulations, such as in the engineering sector however their parallel architecture made them ideal for ML applications as well [49].

Several HA libraries exist today, however, ArrayFire [50] was chosen as it provides well documented and updated methods for the calculation of the SVD algorithm. ArrayFire abstracts the hardware drivers from the programmer as well as provides a straightforward interface. Furthermore, ArrayFire provides a myriad of hardware



accelerated mathematical functions ranging from simple array manipulations to multi-dimensional matrix manipulation and linear algebra. ArrayFire also provides the user with an API to easily switch the computational device during program execution, as well as different programming paradigms to compile the program itself.

In particular ArrayFire gives the programmer the ability to choose from OpenCL, CUDA and asynchronous multi-threaded execution on multi-core CPUs. Both OpenCL and CUDA offer the possibility to interface with massively parallel architectures including, but not only limited to GPUs whereas the latter CPU implementation uses optimised thread and cache manipulations. As a final note, the main difference between OpenCL and CUDA is that the former is a royalty-free standard, maintained by an open-source community, whilst CUDA is the NVIDIA standard and uniquely optimised for NVIDIA hardware [46, 51].

## 2.6 Machine Learning

### 2.6.1 Supervised Learning (SL)

SL concerns itself with learning a mathematical model which maps some input to an output. An introductory example is the house pricing problem where the aim is to learn a model which can accurately predict the price of a house given a set of attributes that describe it; e.g. total area, sound pollution levels and garage option. Given many examples of house prices and their attributes, SL methods would obtain a model that achieves a high level of accuracy in predicting the price of a new house on sale [52, 53].

Consider that the price of a house is exclusively proportional to its total area. Given this assumption, a linear model can be set, called the hypothesis,  $h$ , which is parameterised by a parameter vector,  $\theta$ :

$$h_{\theta}(x) = \theta_0 + \theta_1 x \quad (2.12)$$

where  $h_{\theta}$  denotes the parameterised model and  $\theta_0$  and  $\theta_1$  are the model parameters which are initialised randomly. Model training starts by formulating the "goodness"

of the model with respect to the training data. This is done through the *cost function*, and for this simple example the cost function can be the average squared difference between each entry of the model prediction  $x^{(i)}$ , and the training data  $y^{(i)}$ :

$$J(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \frac{1}{2m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)})^2 \quad (2.13)$$

where  $\min_{\boldsymbol{\theta}}$  indicates that the optimisation occurs with respect to the model parameters and  $\frac{1}{2}$  is introduced to simplify the mathematics further on. By using Equation (2.13) as a metric, a model which better fits the data would have a lower cost. The current hypothesis can be improved by using *Gradient Descent*, an optimisation algorithm which can iteratively reach the local minimum of any arbitrary continuous function. Gradient descent is based on the simple principle that a minimum in a function has a gradient of zero. Analytically this converts to finding the first-order derivative of the function, setting it equal to zero and solving for the parameter which is being optimised. In practice this converts to an update function, which changes the sub-optimal parameters in the direction which minimises  $J(\boldsymbol{\theta})$ :

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) \quad (2.14)$$

where  $\alpha$  is the learning rate. The partial derivatives of the hypothesis in Equation (2.12) with respect to  $\theta$  are expressed as follows:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (2.15)$$

It is important that all parameters are updated simultaneously over the same batch of data to ensure that the hypothesis is wholly updated by the end of each iteration [54].

## 2.6.2 Universal Function Approximators

The previous section made a hypothesis that the relationship between house prices and their area is linear, however, this is rather stringent. One can always choose a hypothesis with more degrees of freedom in order to introduce some complexity in the predictions, however, as it is often the case, the data supplied to the model has an unknown and multi-dimensional distribution which makes it harder to design a well-behaving hypothesis. That is where the idea of a universal function approximator comes in, which is also parameterised by some parameter vector  $\theta$ , and is able to model any type of continuous function given that  $\theta$  is chosen accordingly.

### Artificial Neural Networks

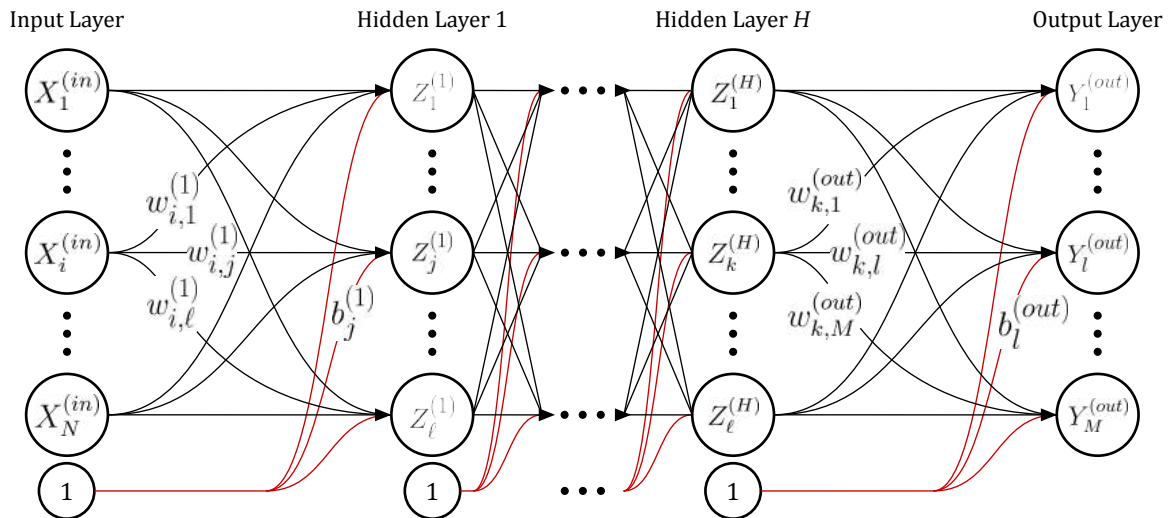


Figure 2.21: A generic diagram of an ANN where the nodes/neurons consist of the input, hidden and output nodes. The number of hidden layers,  $H$  must be  $\geq 1$  otherwise the resultant effect is a simple linear transformation. The constant node represents the bias term, where when multiplied by the bias weights  $b$  (shown in red), acts as the y-intercept for the activation function of the respective consecutive node. The length  $\ell$  of each hidden layer is denoted by the same symbol however each hidden layer can have an arbitrary length where  $\ell \geq 1$ .

Artificial Neural Network (ANN) are such universal function approximators. Figure 2.21 shows a generic diagram of an ANN structure. An ANN should contain at least one hidden layer, otherwise, the output is restricted to a linear combination of

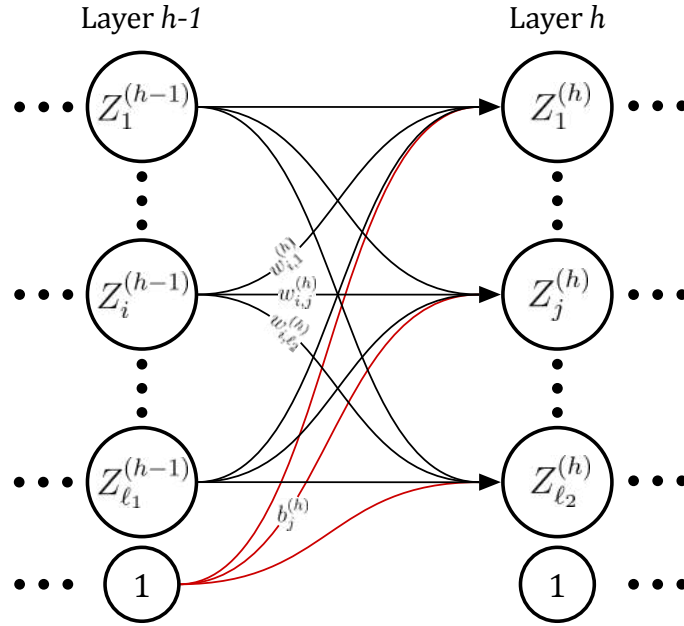


Figure 2.22: Generic segment of a ANN.

the inputs. With one or more hidden layers, an ANN can approximate any continuous real-valued function. ANNs with less than three layers are referred to as shallow networks, and those with three or more hidden layers are called deep networks.

From Figure 2.21 it can be observed that every hidden node  $Z$ , is connected to all the nodes in the previous and the succeeding layers. The first layer is supplied with an input vector  $X$  and the last layer produces the output. A constant node with the value of 1 is also added to all layers except the input layer, and it is multiplied by the bias weights,  $b$ , to act as the threshold of the activation function used in the connected node. The activation function, denoted by  $\sigma(\cdot)$ , is applied to the sum of the weighted inputs to a node. The forward propagation through one node at index  $j$  of layer  $h$  will produce an output  $Z_j^{(h)}$ . The connections to one generic node is shown in Figure 2.22. Equation (2.16) shows the calculation of the total input  $v_j^{(h)}$  and the relationship between the connections in forward propagation:

$$v_j^{(h)} = \sum_{i=0}^{\ell_1} Z_i^{(h-1)} \cdot w_{i,j}^{(h-1)} + b_j^{(h)} \quad (2.16)$$

$$Z_j^{(h)} = \sigma \left( v_j^{(h)} \right) \quad (2.17)$$

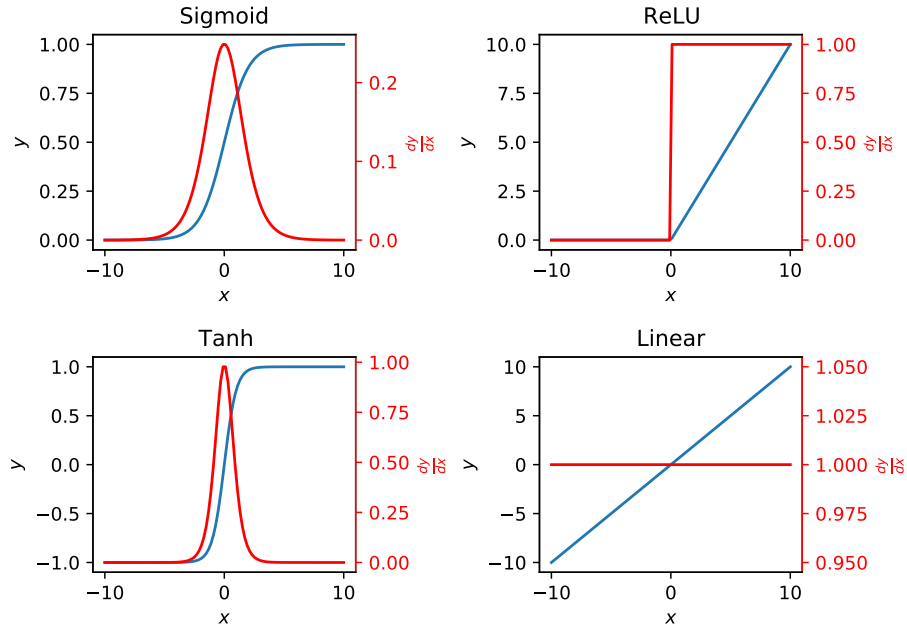


Figure 2.23: Four of the most extensively used activation functions. Due to its simplicity and good empirical results, ReLU is often considered the best choice for hidden layers.

The first formal analysis and proof of the universal function approximation property of ANNs can be seen in [55] and is known as the *Universal Approximation Theorem (UAT)*. The main results of the latter work prove that functions of the form:

$$G(x) = \sum_{j=1}^N \alpha_j \sigma(w_j^\top \cdot x + b_j)$$

where  $w \in \mathbb{R}^N$  and  $\alpha, b \in \mathbb{R}$  are fixed throughout one forward pass, and  $\sigma(\cdot)$  is a bounded differentiable discriminatory function known as the activation function which can take various forms; can approximate any real-valued continuous function.

Throughout the years many more accompanying theorems were developed which generalised over some assumptions made in [55], as well as to strengthen the mathematical understanding of UATs in general. As a result, some conditions required by [55] for the *Universal Approximation Property* to remain valid were found to not be necessary, e.g. a bounded activation function. A condensed and comprehensive overview of the various mathematical advances of the UAT can be seen in [56].

Figure 2.23 illustrates some of the most used activation functions. The sigmoid function is expressed by:

$$\sigma_{sigmoid}(x) = \frac{1}{1 + e^{-x}} = \begin{cases} 1 & \text{as } x \rightarrow \infty \\ 0 & \text{as } x \rightarrow -\infty; \end{cases}$$

the tanh function is the hyperbolic tangent which is expressed by:

$$\sigma_{tanh}(x) = \frac{2}{1 + e^{-2x}} - 1 = \begin{cases} 1 & \text{as } x \rightarrow \infty \\ -1 & \text{as } x \rightarrow -\infty; \end{cases}$$

the ReLU function rectifies its input:

$$\sigma_{relu}(x) = \max(0, x);$$

and the linear function is normally used in the output layer and is simply given by:

$$\sigma_{linear}(x) = x$$

Many other variants have been developed and each created as a possible solution for specific drawbacks present in all activation functions in Figure 2.23. The default activation function used in the hidden layers of a ANN is ReLU, which is owed to its simplicity and good performance on various kinds of tasks [57]. As it can be seen from the plot of ReLU, its gradient is either 0 or 1 where standard machine learning libraries assume that the gradient at  $x = 0$  is 0. Notwithstanding the unlikely occurrence that  $x$  is exactly equal to 0, this choice of default value of  $x$  alludes to sparse networks, where most nodes have a value of 0. Sparse networks have desirable properties, such as more efficient storage and computations as well as being a better model of the cortical neurons in brains [58].

## Back-propagation

Learning vis-à-vis ANNs is the process of adjusting the weights on the edges of Figure 2.21 in order for the network to better approximate the real-world *function* that created the training data. The same procedure as the housing problem in Section 2.6.1 is used to train neural networks, and it is called back-propagation. To aid with the derivation of the training procedure, the small segment of a network in Figure 2.22 is considered where the ReLU activation function is used. Let  $v_j^{(h)}$  denote the total input to a node  $j$  in layer  $h$ . For simpler notation, the bias can be omitted from the derivation of back-propagation since it can be treated like any other weight. Therefore:

$$v_j^{(h)} = \sum_{i=0}^{\ell_1} Z_i^{(h-1)} \cdot w_{i,j}^{(h)} \quad (2.18)$$

and,

$$Z_j^{(h)} = \max\left(0, v_j^{(h)}\right) \quad (2.19)$$

First a cost signal has to be set up, similar to Equation (2.13), where it is assumed that a desired output state for node  $j$ , denoted by  $D_j^{(h)}$ , exists:

$$J(\mathbf{Z}^{(h-1)}, \mathbf{w}_{:,j}^{(h)}) = J = \frac{1}{2} \left( Z_j^{(h)} - D_j^{(h)} \right)^2 \quad (2.20)$$

Second, the derivative of Equation (2.20) with respect to the output of node  $j$  is found:

$$\frac{\partial J}{\partial Z_j^{(h)}} = Z_j^{(h)} - D_j^{(h)} \quad (2.21)$$

Using the chain rule on Equation (2.21), the effect of a change to the total input of the node to the cost is obtained:

$$\frac{\partial J}{\partial v_j^{(h)}} = \frac{\partial J}{\partial Z_j^{(h)}} \cdot \frac{dZ_j^{(h)}}{dv_j^{(h)}} \quad (2.22)$$

where  $\frac{dZ_j^{(h)}}{dv_j^{(h)}}$  depends on the type of activation function. Since ReLU is used in this

network segment,  $\frac{dZ_j^{(h)}}{dv_j^{(h)}}$  is given by:

$$\frac{dZ_j^{(h)}}{dv_j^{(h)}} = \begin{cases} 1 & \text{if } v_j^{(h)} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.23)$$

By applying the chain rule to Equation (2.22) and differentiating Equation (2.18) with respect to a weight  $w_{i,j}^{(h)}$  to obtain  $\frac{\partial v_j^{(h)}}{\partial w_{i,j}^{(h)}}$ , the effect of a change in  $w_{i,j}^{(h)}$  is the chain of derivatives shown in Equation (2.25):

$$\frac{\partial v_j^{(h)}}{\partial w_{i,j}^{(h)}} = Z_i^{(h-1)} \quad (2.24)$$

$$\frac{\partial J}{\partial w_{i,j}^{(h)}} = \frac{\partial J}{\partial Z_j^{(h)}} \cdot \frac{dZ_j^{(h)}}{dv_j^{(h)}} \cdot \frac{\partial v_j^{(h)}}{\partial w_{i,j}^{(h)}} \quad (2.25)$$

By substituting Equation (2.21), Equation (2.23) and Equation (2.24) in Equation (2.25), the effect of changing a weight  $w_{i,j}^{(h)}$  on the cost  $J$  can be found:

$$\frac{\partial J}{\partial w_{i,j}^{(h)}} = \begin{cases} \left( Z_j^{(h)} - D_j^{(h)} \right) \cdot \left( Z_i^{(h-1)} \right) & \text{if } v_j^{(h)} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.26)$$

To continue propagating backwards it is required to re-formulate Equation (2.21), to take into account the change in cost due to a change in the outputs of layer  $h - 1$ . This can also be derived by the chain rule and summing up the contributions of  $Z_i^{(h-1)}$ , to layer  $h$ :

$$\frac{\partial J}{\partial Z_i^{(h-1)}} = \sum_{j=0}^{\ell_2} \frac{\partial J}{\partial v_j^{(h)}} \cdot \frac{\partial v_j^{(h)}}{\partial Z_i^{(h-1)}} \quad (2.27)$$

where  $Z_i^{(h-1)}$  is some node  $i$  in layer  $h - 1$  and  $\ell_2$  is the length of layer  $h$ , as shown in Figure 2.21. To obtain  $\frac{\partial v_j^{(h)}}{\partial Z_i^{(h-1)}}$ , Equation (2.18) is differentiated with respect to  $Z_i^{(h-1)}$ :

$$\frac{\partial J}{\partial Z_i^{(h-1)}} = \sum_{j=0}^{\ell_2} \frac{\partial J}{\partial v_j^{(h)}} \cdot w_{i,j}^{(h)} \quad (2.28)$$



Now if layer  $h$  is the output layer then it is possible to obtain  $D_j^{(h)}$  from the training label. Hence back-propagation is required to start from the right-most layer of Figure 2.21. Following this, by storing the gradients calculated in the last layer, Equation (2.28) is used to calculate the gradients in the penultimate layer, and so on.

## Convolutional Neural Networks

ANNs are universal, meaning that any continuous function can be approximated. However, as the input size increases, so does the training time. This was especially a problem when ANNs were being applied to images, since increasing the image resolution meant a quadratic increase in training time, as well as other issues such as over-fitting. A more efficient network architecture was developed specifically for images, the Convolutional Neural Network (CNN) [59].

CNNs were inspired by the same mechanism observed in a mammalian visual cortex, which is specialised to work with images [60]. Local regions in images are highly correlated, i.e. to form any shape, adjacent pixels need similar values. Other types of data where local correlation is high are therefore compatible with CNNs. One such example of data used in this work are BBQ spectra, where the tune peak is formed by adjacent frequencies having similar amplitudes.

The network architecture of any CNN contains at least one convolutional layer. A convolutional layer is a set of small filters, also called kernels, which are slid over the input data to create what is known as a feature map. Specifically, each kernel is parameterised by a set of trainable weights, which are used during the convolution of a subset of input data with the same size as the kernel. Each operation produces a feature which is sorted in a feature map respective to the position of the kernel over the input data. When one step of the filter is done, it is then moved to the adjacent subset of input data by distance called a stride length, which defines the number of pixels to slide over when shifting the kernel [57].

Stacking convolutional layers also means that more complex features can be extracted while still using a small number of kernels. Figure 2.24 shows an example of a CNN being applied to an image classification task. One kernel having only 9 weights

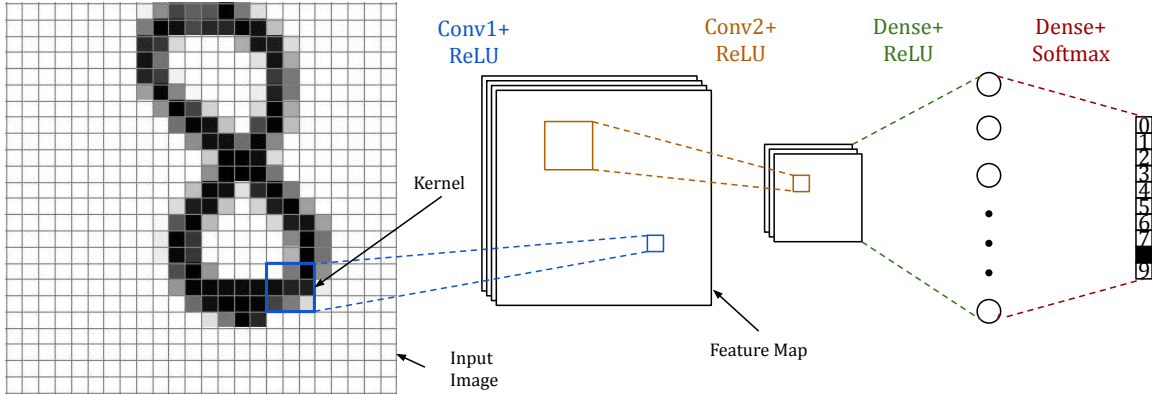


Figure 2.24: An example of a CNN used for image classification.

(blue), is shifted over the input image to create one feature map. The addition of more kernels creates more feature maps. When another convolutional layer is stacked on top of the feature maps, another set of kernels (orange) are shifted over the feature maps to create another set of feature maps themselves. After the elements in the last feature map are flattened into one dimension, they are then connected to a dense layer. For the task of image classification, the last dense layer uses the Softmax activation function, where the output is the probability distribution of the image label.

CNNs offer the advantage of a significantly smaller number of parameters needed to achieve the same performance as a ANN when applied to data types such as images. However, CNNs can also be used for 1D time series data, e.g. real-time electrocardiogram (ECG) monitoring [61], as well as for 3D data, e.g. prediction of macro-scale mechanical properties of a microscopically heterogeneous material [62].

Back-propagation as shown in Section 2.6.2 can still be applied to CNNs, however, with slight adjustments. Equation (2.18) defines the total input to one node in a ANN. For CNNs, the total input to one feature,  $j$ , in a map in layer,  $h$ , becomes:

$$v_j^{(h)} = \sum_{i=0}^{\ell_k} Z_i^{(h-1)} \cdot k_i^{(h)}$$

,

where  $\ell_k$  is the size of the kernel, and  $Z_i$  corresponds to the segment of the previous layer output, which is being convolved with the kernel  $k$ . Therefore,  $k_i$ , denotes the  $i^{th}$

weight of the kernel. The rest of the steps required for back-propagation follow suit.

### 2.6.3 Generative Deep Learning

Generative Deep Learning (GDL) is a subset of machine learning with the aim of training deep networks that can approximate a probabilistic model of a dataset. These networks can in turn be used to provide more samples of the input dataset [63]. This technique is useful for various applications, such as in reduced data representation in Variational Auto-Encoders (VAEs) and data generation in Generative Adversarial Networks (GANs).

VAEs have an Auto-Encoder network structure, where the hidden layers in the Encoder reduce in size to create a latent space representation of the input data. This encoded data is then used by the Decoder, which recreates the input data. Auto-Encoder structures require a representational loss, which compares the reconstructed input with the real input. In addition, VAEs include a regularisation loss on the latent space variables. This loss relates to the capability of the Decoder to generate new data from the latent space [64].

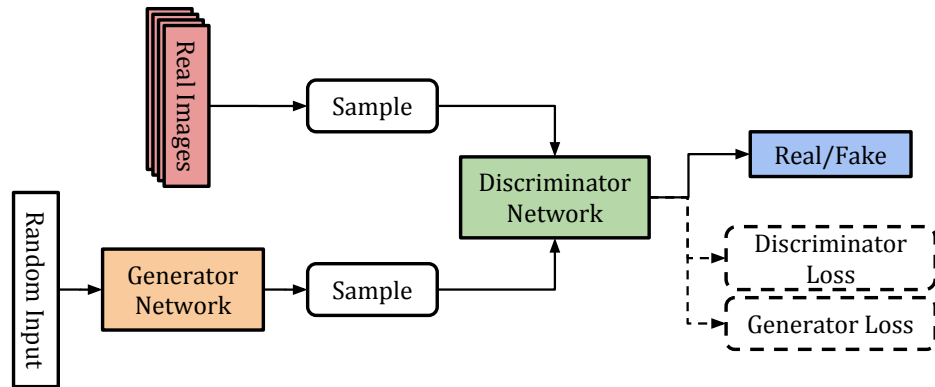


Figure 2.25: Generative Adversarial Network (GAN) architecture applied to image generation using the input data distribution [9].

Figure 2.25 shows the basic architecture used in GANs, where two networks are set up in an adversarial manner. The generator is a function which transforms a random initial distribution into an approximation of the data distribution. The discriminator is a function that accepts a data sample, and predicts whether it is coming from the real

data distribution or from the generator. To train the GAN, the generator needs access to the network parameters of the discriminator. Since GANs are an optimisation of a minimax problem, the generator is set up to maximise the loss of the discriminator, while the discriminator is set up to minimise its classification error. Ideally, GANs converge to a generator which is able to accurately mimic the data distribution, and a discriminator which cannot distinguish between true and fake samples [65].

## SimGAN

The SimGAN is a modified version of the GAN, where the goal is to refine a synthetic or simulated image to look more realistic [66]. Since the role of the GAN generator is now changed to refine an input synthetic image, SimGAN nomenclature refers to the generator network as a *Refiner Network* (R).

In the original work, SimGANs were used to refine simulate images to look more realistic. The problem was that in order to train an accurate model which can estimate the eye gaze from the image of an eye, a large set of real images were required. Since data acquisition of this scale are time consuming, a game simulator was used to produce a large set of simulated images of eyes with different gaze directions. By using adversarial training, the refiner could alter the image to appear obtained from the real data distribution. This allowed for an improved refined dataset that could train models which perform well on real data.

The generating process within GANs is thus transformed into a controlled refinement process, where the annotation information stored in the image is conserved. This allowed for an arbitrarily large dataset to be created, which was used to train a more robust model when used on real data. Section 3.4.2 goes into more detail with regards to the application of SimGANs in the tune estimation system.

## 2.6.4 Reinforcement Learning (RL)

### Introduction

Reinforcement Learning (RL) is the study of a class of algorithms which learn from their interactions with the *environment* through what is called an *agent*. The environment can be anything that is measurable and interacted with such as the following examples: A cart-pole with an inverted pendulum with control of the forces on the cart and the current angle of the pole per time step; a game of chess with the positions of the chess pieces being observable after every move; and a robot moving in a maze where it can only detect if it bumped into a wall.

The environment is responsible for giving a reward, which is a measure of the *goodness* of the previous *action* passed to the environment. The agent is responsible for interacting with the environment, assessing the *value* of the current state of the environment, and choosing an action accordingly which would maximise the next reward. Every state has a value and it signifies the potential of future rewards when starting from said state. The reward is the only external performance metric to the agent and all RL algorithms essentially try to maximise the future rewards through various techniques. Another important term in RL is the *policy* which is a function within the agent that maps the states to the actions. The agent uses the policy to choose the next action that will maximise the future reward.

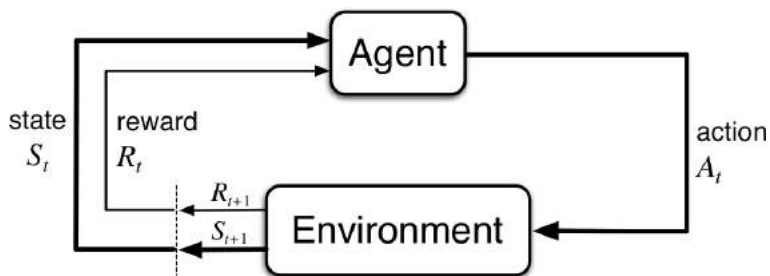


Figure 2.26: Top-level view of the interaction between the Agent and the Environment [10].

Figure 2.26 shows a top-level view of the essential RL components and how they interact. The various signals between the agent and environment are labelled with the mathematical notation used in RL nomenclature.  $S_t$  and  $A_t$  refer to the state of the

environment and the action chosen at time  $t$ .  $R_t$  refers to the reward being given to the agent for the action taken at time  $t - 1$ , hence  $R_{t+1}$  is defined as the reward given to the agent for choosing action  $A_t$  when in state  $S_t$ .

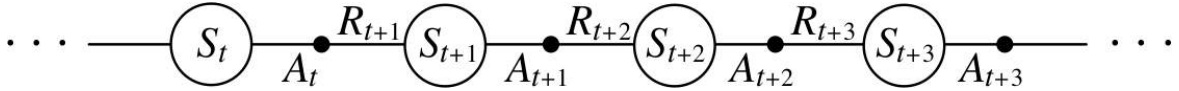


Figure 2.27: Each episode consists of a repeated sequence of state-action pairs along with a respective reward outcome.

An introductory RL algorithm is called State-Action-Reward-State-Action (SARSA), which is an *on-policy* Temporal Difference (TD) prediction algorithm aimed at solving the control problem [10]. SARSA gets its name from the sequence of state-action pairs which make an *episode* as seen in Figure 2.27. An episode is an agent-environment interaction trajectory which ends in a terminal state that can also have a reward, e.g. one tic-tac-toe game is an episode and its terminal state reward can be a win, lose or draw. SARSA uses the *action value*,  $Q(s, a)$ , which is simply a measure used by the agent to estimate how good a specific action is when in a specific state. SARSA is also known as an *on-policy* method, meaning that  $q_\pi$  is continually being estimated from the actions chosen from the behavioural policy,  $\pi$ , where the latter will in turn be updated with the new estimate of  $q_\pi$ . The TD nature of SARSA also means that the action values of the visited states are updated as soon as they have been visited and awarded a reward. The action value update function for SARSA is defined as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (2.29)$$

In SARSA every transition to a non-terminal state,  $S_{t \neq T}$ , calls the update function in Equation (2.29), from which the estimate of  $q_\pi$  for the behavioural policy,  $\pi$ , is continuously refined. Effectively this means that whenever the agent finds itself in  $S_t$  again, it would *remember* how good the previously selected action was in the long run, and either selects or discards this action depending on its past performance. A standard behavioural policy is greedy in  $Q(S_t, a)$ , meaning that it will choose the action with the highest estimated action value.

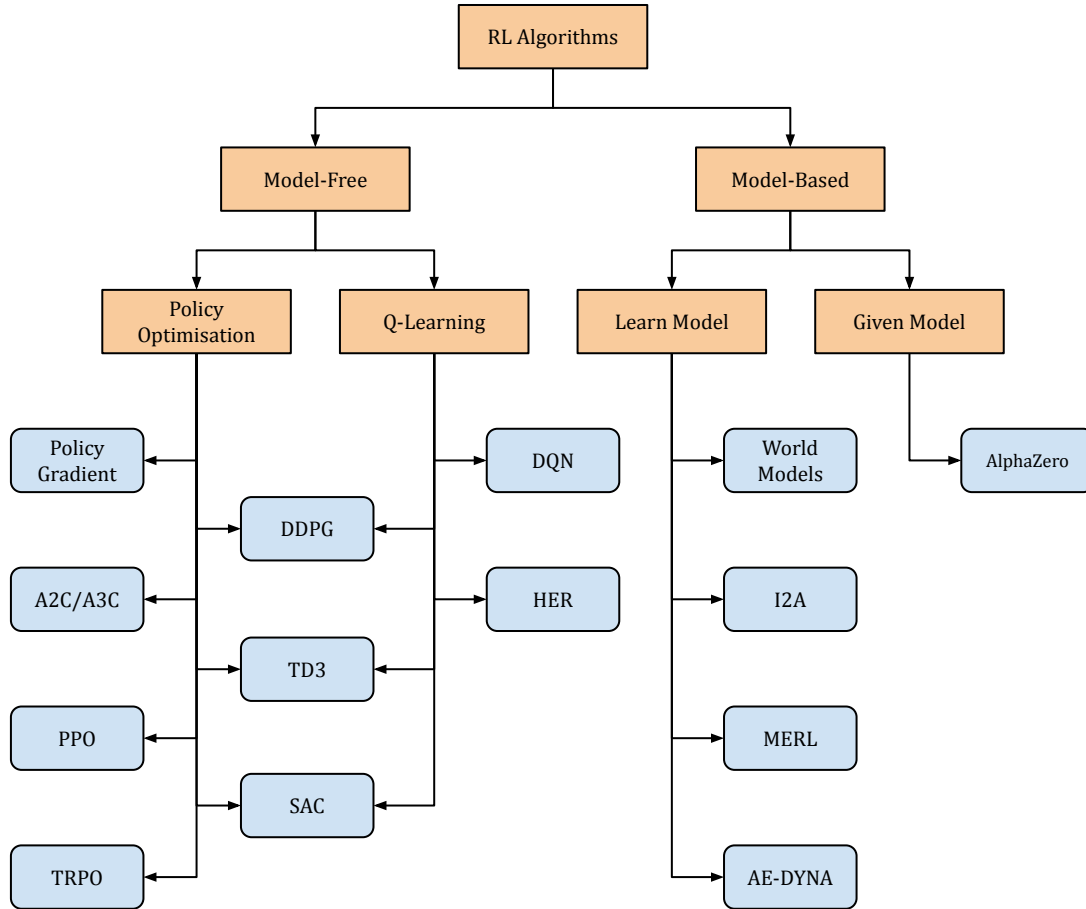


Figure 2.28: A non-exhaustive taxonomy of RL algorithms [11].

## The state-of-the-art in Reinforcement Learning

RL is a vast field of study which ranges from the development of its foundational theory, the exploration of its practical applications, and the development of new techniques that implement the algorithms themselves. Self-driving cars, surveillance drones and machine tuning for optimal resources usage are just some of the usages of RL [67, 68, 69, 70]. This section looks at the development of RL algorithms throughout the years and highlights difference between various algorithms.

Figure 2.28 shows a non-exhaustive taxonomy of the various branches of RL algorithms (orange cells), along with several examples of RL algorithm (blue cells). RL algorithms can be split into two categories where: a) those that explicitly use an internal world model are called Model-Based (MB) and; b) those that do not are called Model-Free (MF). Model-based range from algorithms that have complete access to the

world model, such as AlphaGo [71], to algorithms that approximate the world model from previous experiences, such as Anchored Ensemble DYNA-style (AE-DYNA) [72]. Model-free algorithms consist of algorithms that: a) train a policy network to choose the next action directly, e.g Proximal Policy Optimisation (PPO) [73]; b) train a separate q-network to predict the expected action-value of the current state-action pair and act greedily on it, e.g. Deep Q-Network (DQN) [74] and; c) a combination of both by using the actor-critic architecture, e.g. Deep Deterministic Policy Gradient (DDPG) [75].

One common set of benchmark environments is provided by OpenAi’s Atari 2600 emulator which challenges RL algorithms to reach a maximum high score in all games [76]. Discrete environments embody a strict set of rules, however the real world is made up of continuously varying parameters. This prompts the shift to Deep Reinforcement Learning (DRL) where instead of tabular methods, universal function approximators are trained to predict the action which maximises the reward from the environment. One example of a fully continuous environment is called Humanoid, where the aim is to train an agent to control the joints of a humanoid structure learn complex tasks such as walking, without external guidance [77].

### *Q-Learning*

SARSA was introduced as an on-policy method using TD learning to train the policy. An equally simple method which was developed in the early days of RL is called Q-Learning [78]. Q-Learning is very similar to SARSA with the difference that the current policy is not used to find the next best action but instead Q-Learning chooses the action which will maximise the Q-value of the next state as much as possible. It is for this reason that Q-Learning is called an off-policy method. The equivalent action-value update function for Q-Learning as to the update function Equation (2.29) for SARSA, is shown in Equation (2.30).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \quad (2.30)$$

### *Neural Fitted Q Iteration (NFQ)*



Q-learning has been shown to always converge making it a very reliable algorithm, however, akin to SARSA, it is still considered as a tabular method. This means that all the state-action pairs are saved in a table with their respective Q-value. In the case that a state has never been visited during training, the agent does not know which action to choose. Early attempts at solving this problem involved training an Artificial Neural Network (ANN) to fit the Q-function and this is known as the Neural Fitted Q Iteration (NFQ). The proposed algorithm by Riedmiller involved storing previous state-action-next-state triplets encountered during training as a set of experiences, which are known as the sample set [79]. This method involved updating the ANN offline using the sample set with the aim of updating the ANN globally. In other words, the ANN is not updated on every state transition. Instead, NFQ collects experiences and trains the ANN offline in batches. This was done to better harness the generalisation property of ANNs [79].

#### *Deep Q-Network (DQN)*

However as the number of actions increase, the required number of samples stored in experience makes NFQ too sample inefficient and thus making it unfeasible to be used with large ANNs. This limitation motivated the work of Mnih et al. with the development of the DQN algorithm [74]. The main idea of DQN is to approximate the Q-function using an Artificial Neural Network (ANN), given the agent's previous experiences within the environment. Given adequate training, the ANN is able to approximate the Q-value of even unseen state-action pairs. Thus DQN introduces a new level of generalisation in RL which is able to compress very large state-spaces via a function approximator [74]. This also means that the Q-function,  $Q(s, a)$ , now becomes a parametric function,  $Q(s, a, \theta)$ , where  $\theta$  is a parameter vector signifying the weights of all the nodes in the ANN. It has been shown that the parametrised Q-function allows for the agent to learn different policies for different environments with the same ANN. For this reason, DQNs are considered an important milestone in deep RL [10, 74].

#### *Double Deep Q-Network (DDQN)*

It has been shown that DQN has a tendency to have over-optimistic Q-values due

to the use of the max operator as used in Equation (2.30). Using the same Q value to both select the action and to value it will further bias DQN to over-optimism. Optimism in this context is the proclivity of the agent to choose the actions which have been shown to work in the past, and solely relying on them to find the optimal policy. Van Hasselt et al. proposed an algorithm that split the maximum operation in two. An online ANN which is used with a greedy policy to choose the next best action, and another ANN to estimate the state-action values. From these two deep ANNs arises the name Double Deep Q-Network (DDQN). It is also important to note that experiences are assigned randomly to any one of the two networks [80].

### *Stochastic Policy Gradient (PG)*

Another class of RL algorithms aims at parametrising the policy directly without the need to parametrise the Q-function itself. In Stochastic Policy Gradient (PG) methods the parametrised policy is used to choose the next best action given a state. A performance measurement,  $J(\theta)$ , of the agent is used to achieve this. This family of algorithms express the policy differently from other algorithms such as Q-learning, where the policy,  $\pi(a|s)$ , is expressed as  $\pi(a|s, \theta)$  or  $\pi_\theta(a|s)$ . An important analytic result known as the *policy gradient theorem* is used extensively in PG methods. It specifically addresses how the performance measurement of an agent can be improved, if both the state and action distributions depends on the policy parameters,  $\theta$ . The policy gradient theorem establishes that the gradient of the performance measurement does not depend on the gradient of the state distribution and that:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_S \rho^\pi(s) \int_A \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) da ds \\ &= \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)] \end{aligned} \tag{2.31}$$

where  $\rho^\pi(s)$  is the (on-policy) state distribution when following policy  $\pi$ . Following this the update function of the policy parameter can be defined as a small change in the direction of the gradient of the performance measure [81]:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_{\theta_t}) \tag{2.32}$$

A common architecture in which PG methods are used is the actor-critic algorithm, where the *actor* is responsible for tuning the policy parameter,  $\theta$ , with respect to the stochastic policy,  $\pi_\theta(s)$ , as shown in Equation (2.32), whilst the *critic* estimates the state-action value function by using proper policy evaluation techniques. The Q-function estimate is expressed by the critic as  $Q_w(s, a)$  where given enough samples it would converge to the optimal Q-function under policy  $\pi$  [10]:

$$Q_w(s, a) \approx Q_\pi(s, a)$$

### *Normalised Advantage Functions (NAF)*

Normalised Advantage Functions (NAF) was designed to work in the continuous action domain and is considerably simpler than DDPG. The main idea is to represent the Q-function in a way such that the  $\arg \max_a Q(s, a)$  can be computed analytically and quickly [82]. To better explain the NAF algorithm, the idea of an advantage function has to be introduced:

$$\begin{aligned} V^\pi &= \mathbb{E}_{r_{i \geq t}, s_{i \geq t} \sim E, a_{i \geq t} \sim \pi} [R_t | s_t, a_t] \\ A^\pi &= Q^\pi(s_t, a_t) - V^\pi(s_t) \end{aligned} \tag{2.33}$$

From Equation (2.33) it can be seen that the advantage function measures the value of an action,  $a_t$ , with respect to the average value that can be taken in a state  $s_t$ . NAF, as introduced by Gu et al. is based on a neural network which can separately output both the value and the advantage of a specific state-action pair. Gu et al. parametrised the advantage function as a quadratic function of the nonlinearities of the state as well as represented the Q-function as follows:

$$Q(s, a | \theta^Q) = A(s, a | \theta^A) + V(s | \theta^V) \tag{2.34}$$

$$A(s, a | \theta^A) = -\frac{1}{2}(a - \mu(s | \theta^\mu))^T P(s | \theta^P)(a - \mu(s | \theta^\mu)) \tag{2.35}$$

$$P(s | \theta^P) = L(s | \theta^P)L(s | \theta^P)^T \tag{2.36}$$

Equation (2.34) shows how to obtain the state-action value from the advantage and

value of a specific state-action pair and comes from Equation (2.33). Equation (2.35) is the parametrised version of the advantage function in Equation (2.33). Note that  $P(s|\theta^P)$  in Equation (2.36) is a state-dependent, positive-definite square matrix and  $L(s|\theta^P)$  is a lower triangular matrix constructed from the linear output layer of the value network.

Another important milestone of NAF was the acceleration in the learning rate through *imagination rollouts*, which generate synthetic experiences from an iteratively refitted, time-varying linear model. This approach varies from a naive attempt at creating an off-line behavioural policy based purely on a learned model by mixing the synthetic experience with real experience obtained from interaction with the environment. Tests by Gu et al. concluded that teaching an off-policy agent by only using a learned model as a behavioural policy, will not be effective at training the agent since only good action will be chosen by the behavioural policy. To learn an effective policy, the agent has to be allowed to make mistakes and learn on its own the impact of each action [82].

#### *Trust Region Policy Optimisation (TRPO)*

Unconstrained PG methods are known to suffer from policy collapse if the learning rate is too high. This is due to the sensitivity of the policy output to seemingly small changes in the network parameters. To solve this problem, Trust Region Policy Optimisation (TRPO) was introduced, which sets a limit to how much a policy can change after each update. Consider an update being done to a policy, and let the old policy be denoted by  $\pi_\theta$ , and the new policy be denoted by  $\pi_{\theta'}$ . Then let  $J$  denote the objective function that is maximised by TRPO:

$$\begin{aligned}
 J(\theta) &= \mathbb{E}_{s \sim \rho^{\pi_\theta}, a \sim \pi_\theta} \left[ \frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)} \hat{A}_\theta(s, a) \right] \\
 \text{s.t. } &\mathbb{E}_{s \sim \rho^{\pi_\theta}} [D_{KL}(\pi_\theta(\cdot|s) || \pi_{\theta'}(\cdot|s))] \leq \sigma
 \end{aligned}
 \tag{2.37}$$

where  $\hat{A}_\theta$  denotes the estimate of the advantage when in state  $s$  and choosing action  $a$ .  $\rho^{\pi_\theta}$  denotes the state visitation frequency when using policy  $\pi_\theta$ . The condition of Equation (2.37) measures the Kullback-Leibler divergence between the old and new

policies and ensures that it is smaller than a threshold parameter  $\sigma$  [83].

### *Proximal Policy Optimisation (PPO)*

TRPO is a notoriously complex algorithm to implement, however, the idea to limit the maximum change to a policy is relatively simple to understand. PPO also maximises Equation (2.37), however, the Kullback-Leibler constraint is not used. Instead, PPO imposes constraints directly to the probability ratio,  $r(\theta, \theta')$ :

$$\begin{aligned}
 r(\theta) &\triangleq \frac{\pi_{\theta'}(a|s)}{\pi_{\theta}(a|s)} \\
 g(\epsilon, A) &\triangleq \begin{cases} (1 + \epsilon)A & A \geq 0 \\ (1 - \epsilon)A & A < 0 \end{cases} \\
 J(\theta) &= \mathbb{E}_{s \sim \rho^{\pi_{\theta}}, a \sim \pi_{\theta}} \left[ \min \left( r(\theta) \hat{A}^{\pi_{\theta}}(s, a), g \left( \epsilon, \hat{A}^{\pi_{\theta}}(s, a) \right) \right) \right] \quad (2.38)
 \end{aligned}$$

where  $\epsilon$  is a small hyperparameter and controls how much a policy can change after each update. It has been shown that the performance of PPO is similar to that of TRPO on a majority of benchmark environments [73, 84].

### *Deterministic Policy Gradient (DPG)*

So far, only stochastic policies were considered, which have the form of  $\pi_{\theta}(a|s)$ . For every state,  $s$ , a stochastic policy outputs an action distribution. The next action,  $a$ , is randomly selected from this distribution.  $\theta$ , is the vector containing the network parameters that make up the policy. However, another approach, known as Deterministic Policy Gradient (DPG), is to choose an action from a state in a deterministic manner, where the conditional parametric distribution,  $\pi$  is replayed by a deterministic policy described by:

$$a = \mu_{\theta}(s)$$

The deterministic policy gradient has an equivalent theorem to Equation (2.31):

$$\begin{aligned}
 \nabla_{\theta} J(\pi_{\theta}) &= \int_S \rho^{\mu}(s) \nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a)|_{a=\mu_{\theta}(s)} ds \\
 &= \mathbb{E}_{s \sim \rho^{\mu}} \left[ \nabla_{\theta} \mu_{\theta}(s) \nabla_a Q^{\mu}(s, a)|_{a=\mu_{\theta}(s)} \right] \quad (2.39)
 \end{aligned}$$

In the work of Silver et al. it has also been shown that if the stochastic policy gradient is parameterised by a deterministic policy,  $\mu_\theta$ , and a variance parameter,  $\sigma$ , such that the stochastic policy becomes  $\pi_{\mu_\theta, \sigma}$ , then the latter approaches the deterministic policy as the variance tends to zero [85]:

$$\lim_{\sigma \rightarrow 0} \nabla_\theta J(\pi_{\mu_\theta, \sigma}) \equiv \nabla_\theta J(\mu_\theta)$$

### *Deep Deterministic Policy Gradient (DDPG)*

The problem of high-dimensional state-spaces was solved by DQN by fitting a deep ANN to the Q-function to approximate the state-action value at any state. However, this only works on discrete and relatively small action-spaces. Large action-spaces become unfeasible to explore efficiently with a DQN. DDPG builds upon the deterministic policy gradient in Equation (2.39) to create a model-free, off-policy actor-critic algorithm. In DDPG, separate copies of the main actor and critic networks are created and are called the target networks. Thus DDPG uses a total of four networks [86]:

- $Q(s, a|\theta^Q)$  is the main critic Q-network and is initialised randomly
- $Q'(s, a|\theta^{Q'})$  is the target Q-network and is initialised to  $Q(s, a|\theta^Q)$
- $\mu(s|\theta^\mu)$  is the main deterministic policy network of the actor and is initialised randomly
- $\mu'(s|\theta^{\mu'})$  is the target deterministic policy network and is initialised to  $\mu(s|\theta^\mu)$

Subsequently the agent chooses actions on every step made in the environment and stores these experiences in a replay buffer, R. Mini-batches of length N are sampled from R and are used to train the critic Q-network and the policy network. The Q-network of the critic is optimised by minimising the loss  $L(\theta^Q)$ :

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E} \left[ \left( Q(s_t, a_t|\theta^Q) - y_t \right)^2 \right] \quad (2.40)$$

where  $\beta$  is a separate stochastic behaviour policy used to learn  $Q^\mu$  and  $y_t$  is called the target. Lillicrap et al. reported that directly applying Q-learning using ANNs was

unstable in many environments. However, by using the target critic and actor networks to estimate  $y_t$ , the learning is stabilised. The target  $y_t$  for DDPG is formulated by:

$$y_t = r_t + \gamma Q' \left( s_{t+1}, \mu'(s_{t+1} | \theta^{\mu'}) | \theta^{Q'} \right) \quad (2.41)$$

The better learning stability, albeit at a slower rate, is due to the Polyak averaging of the main and target network parameters. Equation (2.42) describes the target networks parameter update rule, where  $\tau$  is usually a small value, e.g. 0.005.

$$\left. \begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \end{aligned} \right\} \tau \ll 1 \quad (2.42)$$

### *Twin-delayed Deep Deterministic Policy Gradient (TD3)*

While it is possible to successfully train DDPG on many environments, it is often very sensitive to hyperparameter tuning. One common problem with DDPG is shared by DQN, where the predictions of the Q-values are over-optimistic. Similar to the solution introduced by DDQN, Twin-delayed Deep Deterministic Policy Gradient (TD3) implements two Q networks and chooses the smaller between them as the estimate for the state-action value. The second improvement that TD3 introduced was the delayed target policy update, which was reported to increase training stability. The final improvement was the addition of clipped smoothing noise added to all target actions, which was reported to help the agent assign higher values to actions which are robust to noise [87].

### *Soft Actor-Critic (SAC)*

Soft Actor-Critic (SAC) was the first RL algorithm to combine a stochastic policy with a DDPG-style actor-critic agent. This was done by introducing the idea of entropy maximisation. Entropy, in information theory stands for the measure of randomness present in a variable. The following equation shows how the entropy for a random variable,  $x$ , drawn from probability distribution,  $P$ , is calculated:

$$H(P) = \mathbb{E}_{x \sim P} [-\log P(x)] \quad (2.43)$$

Since the policy is stochastic, Equation (2.43) can be used on the action distribution of the policy for any given state. SAC changes the objective of standard RL algorithms to include a maximisation of entropy. Therefore:

$$J(\pi) = \sum_{t=0}^T \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi^i}} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot | s_t))] \quad (2.44)$$

where  $\alpha$  is called the temperature parameter which controls the relative importance of entropy with respect to the reward. This augmented objective incentivises the policy to explore regions of the state-action space which were not visited and also helps the policy to stop exploring unpromising paths. Similar to TD3, SAC also uses two Q networks to reduce over-estimation [88].

### 2.6.5 Use of ML in particle accelerators

There has been an interest in the application of ML in particle accelerators in recent years and some of the key points of interest are highlighted in [89]. Here several ML-based techniques were discussed and their potential to provide a significant improvement over the current systems in a particle accelerator was analysed.

The sheer size of a particle accelerator such as the LHC synchrotron, requires a high level of precision from all of the thousands of components which are needed for its correct operation. This inevitably demands highly complex and data-intensive operations which might be sensitive to even the smallest failure in any component making the particle accelerator work. To solve these issues, ML has previously been used to detect anomalies in critical components as they occur, in turn strengthening the LHC machine protection systems.

One such example is the anomaly detection for the beam loss maps, which are used to validate the LHC collimator settings for beam cleaning and machine protection. In this work Beam Loss Monitors (BLMs) from IR7 were used as features for anomaly detection. Principal Component Analysis (PCA) can orthogonally transform a set of potentially correlated variables into a smaller set of uncorrelated variables. This was used to reduce the feature dimensionality of all the BLMs onto a two dimensional plane



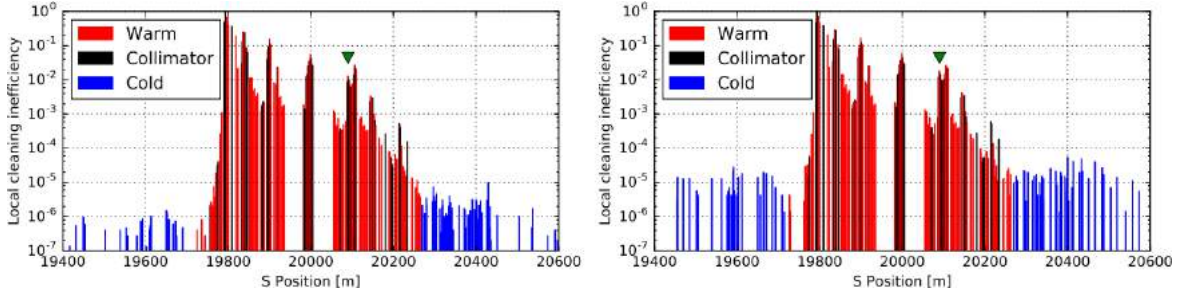


Figure 2.29: Beam 1 vertical loss maps at 450GeV for nominal collimator settings (left) and a  $200\mu\text{m}$  displacement in one vertical collimator (right) [12].

corresponding to the two largest principal components of the BLM signals. Thus, any small change in the beam loss maps would result in a shift of the point on the principal component plot. Adding the fact that after each set of measurements, the loss maps are not expected to change a lot due to the same optics being used, loss maps done at different times would result in a cluster in the principal component plot. Any loss map which contains erroneous measurements would thus manifest itself as an outlier in the principal component plot. Finally these outliers were detected using Local Outlier Factor (LOF), which is an Unsupervised Learning (UL) algorithm and which works based on the idea of measuring the density of the local region around a point, and declaring it an outlier if its local region has a lower density than its neighbouring points [90]. Figure 2.29 illustrates the small change (at position of green markers) in the measured loss map as detected by this anomaly detection scheme and corresponds to a  $200\mu\text{m}$  shift in one vertical collimator [12].

The use of RL in particle accelerators has also been quite recently explored and successfully implemented on relatively small environments. In particular, Kain et al. used NAF with Prioritised Experience Replay (PER) to control the electron beam position in AWAKE [91]. PER is a replay buffer sampling strategy used in off-policy RL algorithms, which revisits trajectories with a high Temporal Difference error (TD-error). This error measures the difference between the estimated and the actual returns when following a policy,  $\pi$ .

Another work by Hirländer et al. saw the development of two RL algorithms called NAF2 and AE-DYNA, which were subsequently used to maximise the intensity

of a seeded Free-Electron Laser (FEL). NAF2 is the NAF algorithm with some of the improvements introduced by TD3: double Q-learning and smoothing target action noise. AE-DYNA is a model-based RL algorithm, which trains a model that captures both the aleatoric and epistemic noise of the environment. This uncertainty-aware model is subsequently used to train a model-free RL algorithm, which performs well in the real world [72].

## 2.7 Summary

This chapter provides the necessary background for all the work presented in this thesis. The first topic to be considered in Section 2.1 introduces the physics of the systems within the Large Hadron Collider (LHC) which are related to the Beam-Based Feedback System (BBFS). Sections 2.2 and 2.3 introduce the hardware which serve as output and input of the BBFS, respectively. Section 2.4 goes through the software architecture of the BBFS as it existed by the end of the LHC Run 2. Section 2.5 introduces the basics of Hardware Acceleration (HA) to serve as a possible solution to the computational bottleneck within the BBFS. Finally, Section 2.6 goes through the different Machine Learning (ML) tools employed in this work in a bottom-up approach. All these topics are then linked together in the proceeding chapters to explain a certain aspect of the results obtained in this work.

In particular, Chapter 3 delves into the development of new tune estimation systems in the presence of the 50 Hz harmonics introduced in Section 2.3.2. The development of the data-driven solution in this chapter is built upon the literature introduced in Sections 2.6.1 to 2.6.3.

The information about the LHC magnets provided in Section 2.2 becomes more relevant in the feasibility study done in Chapter 4. The work in this chapter aims to develop a control system using a Reinforcement Learning (RL) agent, which would ultimately interact with the beam via the corrector magnets. Section 2.6.4 serves to understand the differences among the various types of RL agents attempted in Chapter 4.

The last chapter in this thesis, Chapter 5, draws upon the design of the BBFS as introduced in Section 2.4, to explain the work done during the code renovations. The feasibility study on the application of HA is a direct continuation of the background research provided by Section 2.5.

# Chapter 3

## Development of new tune estimation systems

This chapter starts with a recapitulation of the BQ algorithm in Section 3.1, which provided tune estimates from Base-Band Q (tune) (BBQ) spectra until Long Shutdown 2 (LS2). Section 2.3.2 looked at some of the effects of 50 Hz noise harmonics on the BQ tune estimates. During the LHC LS2, it was decided to look into potential solutions to more reliably estimate the tune from a harmonically noisy spectrum.

Early on during the development of new tune estimation algorithms, the need for ground truth values of the tune within a spectrum became apparent. Ground truth values were not available since the logged tune estimates from real operation were labelled by BQ tune estimates. To compare the performance of BQ and any alternative tune estimation algorithm, Section 3.2 introduces a method to simulate spectra with known tune values and artificially injected 50 Hz harmonics.

In Section 3.3, a series of algorithms were developed, which explicitly removed the harmonics from a spectrum, and estimated the tune from a spectrum with gaps. Throughout this work, the new algorithms are collectively called the *Alternative approach* and in total, three different algorithms were proposed. At the end of this section, a Monte Carlo simulation is performed to compare the two best approaches.

Following this work, the first attempts at solving the tune estimation problem with ML tools are called the Simple Approach, where the tune estimation task was

delegated to an Artificial Neural Network (ANN). Due to the lack of ground-truth tune estimates in the real BBQ data, the training data in the simple approach was simulated using the method in Section 3.2. Different network architectures were attempted and their performances were analysed and compared. It was found that over-fitting to the simulated spectra might have caused a sub-standard performance in the simple approach.

As a potential solution, it was proposed to use a variant of Generative Adversarial Network (GAN) called SimGAN. A 1D version of SimGAN was used to refine the simulated spectra to make them look more realistic, whilst still retaining the spectrum information from the simulator, i.e. ground-truth tune value. It was found that many SimGANs, with slightly different loss functions were required to create a varied training dataset for a tune estimation network. To create a refined dataset, a simulated dataset is created first, then a trained SimGAN is randomly chosen from disk, and finally the simulated batch is fed to the SimGAN. By repeating this process, an arbitrarily large training dataset could be created. The best model architecture from the simple approach was then trained with the improved dataset.

The performance of the salient algorithms and models attempted in this work was compared on unseen BBQ spectra. The impact on the Tune Feedback (QFB) following these enhancements is also discussed at the end of this chapter.

### **3.1 A review of the BQ algorithm**

Throughout the years in operation, the design of the BQ algorithm has evolved through a series of upgrades guided by the LHC operators. BQ worked with a sequence of exponential moving averages, median filters, average filters and Gaussian fits in order to estimate the position of the dominant peak within a spectrum. By LS2, BQ had become a complex algorithm, however, its performance was still heavily affected by the noise harmonics.

Section 2.3.2 discussed the decrease in performance of BQ, in the presence of 50 Hz noise harmonics. This was due to the nature of the tune estimation algorithm, which

allowed a tune estimate to be made from a spectrum with harmonic spikes. The probability that the maximum of the spectrum occurs at a harmonic spike is proportional to largest harmonic amplitude. This was observed by the LHC operators where in the presence of strong noise harmonics, the stability of the estimate degrades significantly. In the presence of erroneous estimates, the use of the Proportional-Integral (PI) controller within QFB created numerical problems, which at times crashed the BBFS. In the end, the QFB was fitted with its own stability metric to discern whether the tune estimates fed into it were stable enough to be used for control.

A lot stands to be gained from a more robust tune estimation algorithm. In line with the goal of this work, which is to improve the beam-based feedback systems, a more stable tune estimate would directly affect the performance of the QFB. This would also allow the operators to have more control on the tune and consequently gain more control on the beam. A reduction in the risk of beam instabilities also improves the long-term performance of the LHC.

## 3.2 Simulations

The performance of the alternative algorithms developed in Section 3.3 could not be reliably measured and compared to the BQ algorithm. This also meant that if there were any improvements detected through simulated spectra, they could not be justified on real data.

As shown in Equation (2.1), the transverse beam dynamics can be expressed by a Hill's type equation. Due to Floquet's Theorem, the homogeneous solutions to these differential equations can be expressed by Equation (2.2). This manifests as transverse oscillations in the beam called betatron oscillations. It was found that this motion can be modelled by a damped second-order system with added Gaussian noise. The frequency response of a damped second-order system with a resonant frequency,  $\omega_{res}$ , and a damping factor,  $\zeta$ , is given by:

$$G(\omega) = \frac{\omega_{res}^2}{\sqrt{(2\omega\omega_{res}\zeta)^2 + (\omega_{res}^2 - \omega^2)^2}} + \mathcal{N}(0, \sigma), \quad (3.1)$$

Damping causes the true resonant frequency of the system to be different from,  $\omega_{res}$ . The true system frequency,  $\omega_{res}^{true}$ , is calculated by:

$$\omega_{res}^{true} = \sqrt{1 - 2\zeta^2}\omega_{res}. \quad (3.2)$$

Therefore the peak in the simulated spectrum occurs at  $\omega_{res}^{true}$  and can be considered equivalent to the value of the tune in a BBQ spectrum. By using this simulation, any tune estimate can be compared to the actual resonance in a spectrum. Thus a reliable metric for the accuracy of the tune estimates could now be obtained. Equation (3.1) simulates the shape of the baseline by using  $\omega_{res}$  and  $\zeta$ , as well as adds Gaussian noise,  $\mathcal{N}(0, \sigma)$ . Simulated spectra are shown superimposed on two examples of BBQ spectra where: Figure 3.1 shows the baseline simulated spectrum and Figure 3.2 shows the baseline with added Gaussian noise.

Some of the new tune estimation approaches developed in this chapter require a fixed length input, e.g. ANNs. Since the LHC operators use different frequency windows during a fill, it was decided to pre-define the size of all simulated spectra to be 100 frequency bins long. This value was chosen to be slightly larger than the frequency windows chosen during real machine operation using the BQ algorithm. The average operational frequency window obtained from a sample of parameters used in the BBQ system for the beam during FLATTOP is around 80 frequency bins long. It was empirically observed however that sometimes the dominant peak lay close to the edges of the chosen window, which subsequently limited the performance of the BQ algorithm. Figure 3.4 compares the new window length to that used in operation in LHC Run 2. It can be observed that the BQ tune estimates around the 15s were close to edge of the operational window (grey). This example shows that the bounds set on the inputs to all tune estimation approaches developed in this chapter are realistic.

It is important to note that the absolute y-values of the spectrum are not important for tune estimation. Therefore, each spectrum created with Equation (3.2) was normalised to the range  $[0, 1]$ . To recreate the 50 Hz harmonics, it was decided to randomly choose a normalised amplitude per harmonic from  $\mathcal{U}(0.5, 2)$ , and after adding

Fill 6847, Beam 1, Plane v, Mode INJPHYS  
Mode starts at: 26/06/18 07:39:07 AM, ends at: 26/06/18 08:07:31 AM  
Spectrum data starts at 26/06/18 07:53:16 AM, end at: 26/06/18 07:53:22 AM

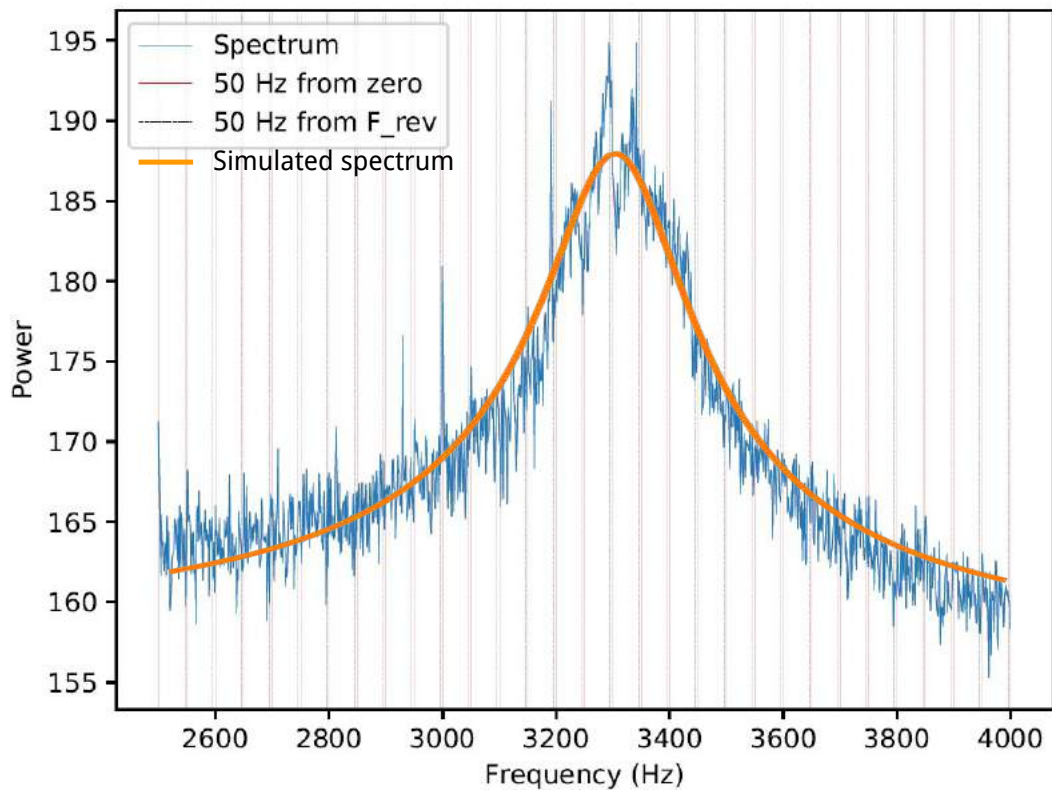


Figure 3.1: **BBQ spectrum** (blue) from: Beam 1, Plane V, Beam Mode INJPHYS, Fill 6847. **Simulated spectrum** (orange): Second-order system baseline spectrum.



Fill 6874, Beam 2, Plane v, Mode ADJUST  
Mode starts at: 02/07/18 01:17:05 AM, ends at: 02/07/18 01:26:35 AM  
Spectrum data starts at 02/07/18 01:21:47 AM, end at: 02/07/18 01:21:54 AM

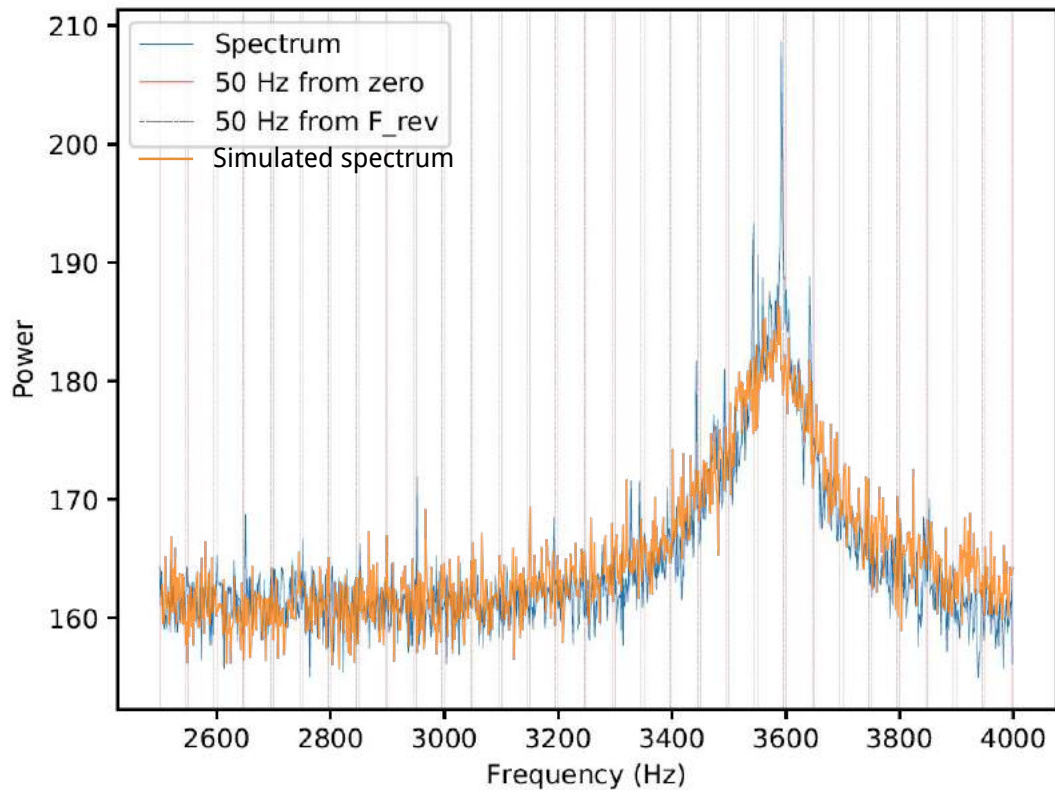


Figure 3.2: **BBQ spectrum** (blue) from: Beam 2, Plane V, Beam Mode ADJUST, Fill 6874. **Simulated spectrum** (orange): Baseline spectrum with added Gaussian noise.

Fill 6881, Beam 2, Plane h, Mode FLATTOP  
Mode starts at: 03/07/18 06:12:02 AM, ends at: 03/07/18 06:14:54 AM  
Spectrum data starts at 03/07/18 06:13:25 AM, end at: 03/07/18 06:13:31 AM

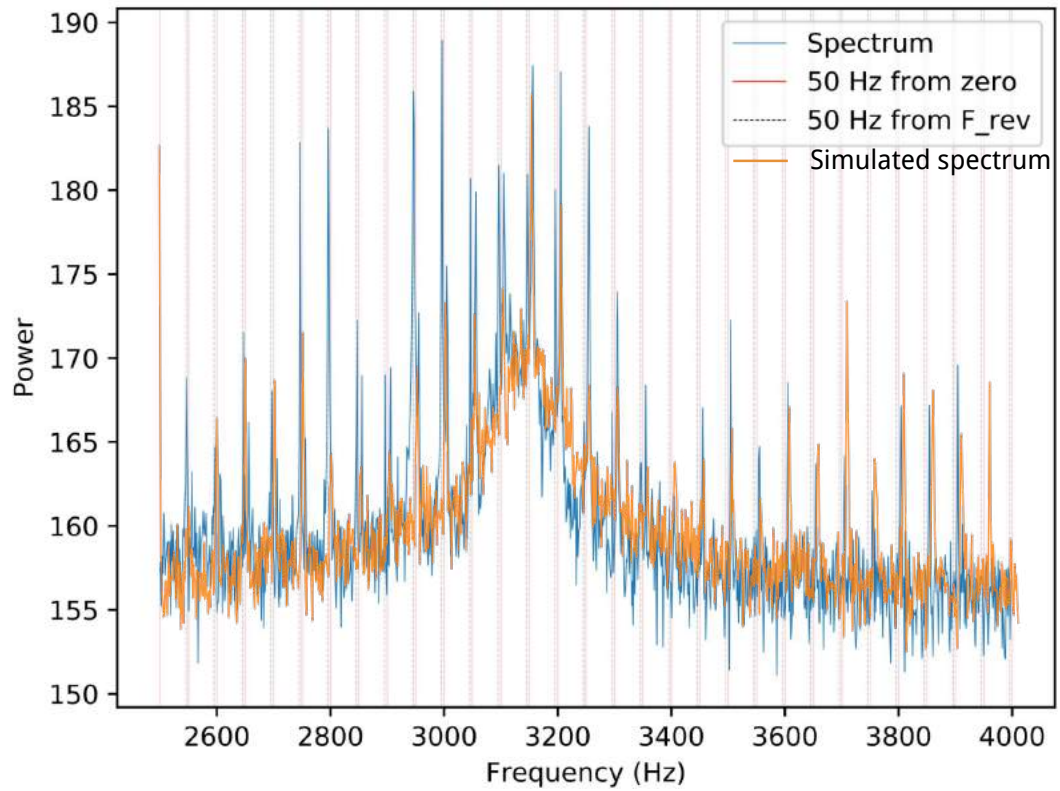


Figure 3.3: **BBQ spectrum** (blue) from: Beam 2, Plane V, Beam Mode FLATTOP, Fill 6881. **Simulated spectrum** (orange): Noisy baseline spectrum with injected 50 Hz noise harmonics.

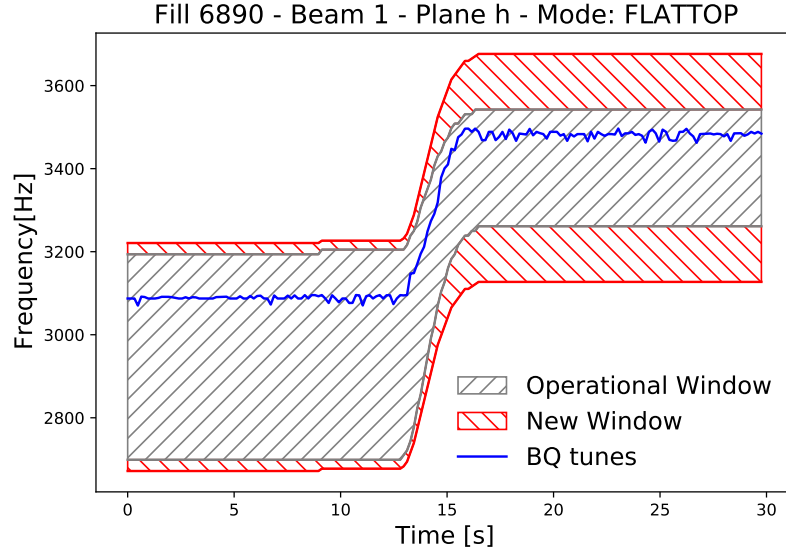


Figure 3.4: An example of the operational frequency windowing used by the BQ algorithm during the LHC Run 2 to the new window length used in this work. The tunes were obtained from LHC fill 6890, horizontal plane of beam 1 in the FLATTOP beam mode. During FLATTOP the optics are changed and thus the tune shifts to a new frequency.

the harmonics to the simulated spectrum. Moreover, a simple linear digital filter of size 3 was passed forward and backward to the spectrum in order to give width to the harmonics. The spectrum was then normalised again, and  $\omega_{res}^{true}$  was found in terms of the normalised frequency range with respect to frequency window limits. Figure 3.3 shows another example of a BBQ spectrum with a superimposed simulated spectrum, created with the approximate  $\omega_{res}^{true}$  and  $\zeta$  required to mimic the shape of the tune peak. Note that all simulated spectra shown in Figures 3.1 to 3.3 were scaled appropriately to match with the BBQ spectra for these examples.

Ultimately, all the tests comparing the performance of alternative algorithms in Section 3.3 and the creation of large datasets of simulated spectra in Section 3.4, use a Monte Carlo (MC) simulation of the second-order system spectrum to create spectra with different shapes and peak positions relative to the frequency window. To avoid the creation of spectra with the tune peak at the edge of the frequency window, all normalised resonant frequencies were sampled from  $\mathcal{U}(0.1, 0.9)$  and the damping factor,  $\zeta \sim 10^{\mathcal{U}(-2.5, -1.8)}$ .

### 3.3 An alternative approach

During LS2 it was decided to attempt to mitigate the impact of the 50 Hz harmonics on the tune estimates by upgrading the software side of the tune estimation to be more robust to harmonic noise. An observation was made that the noise harmonics are relatively stable when viewed from the point of view of the spectral resolution of BBQ spectra at 5.5 Hz (Equation (2.10)). Figure 3.6 shows an example of the evolution of the 50 Hz harmonics (red), superimposed on a heat-map made up of spectra from Fill 6890, beam 1, horizontal, during beam modes PRERAMP to SQUEEZE. The red traces correspond to mean,  $\mu$ , and standard deviation,  $\sigma$ , of the peak frequencies of each respective harmonic frequency. It can be observed that the position of the tune peak in the spectra changes around 05/07/18 13:17:00, however, the 50 Hz harmonic frequencies remain relatively stable.

Following this observation it made sense to assume that the general position of all the possible harmonics within the frequency spectrum could be precalculated and removed from every spectrum. The number of points that would have to be removed was sufficiently low as to still retain the shape of the spectrum [8].

Figure 3.5 shows an example of a BBQ spectrum. The red bands show the regions of the frequency axis which are ignored during the tune estimation. The spectrum with gaps (orange) was used by all the alternative algorithms attempted in this work. The following sections will explore three different algorithmic approaches to improving the BBQ tune estimates.

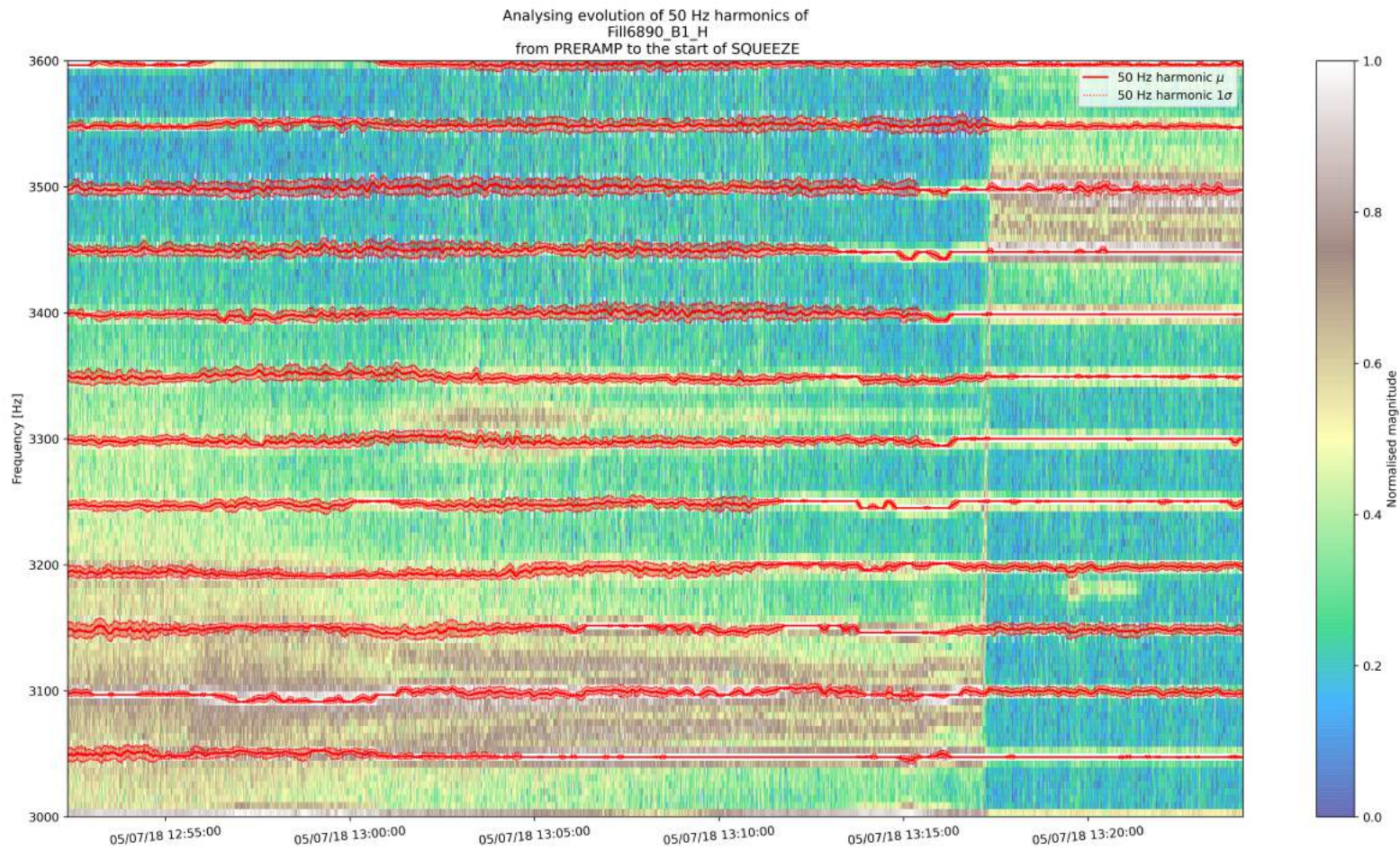


Figure 3.6: Traces of individual harmonics within a frequency window. Every column in the heat-map is a normalised BBQ spectrum obtained from a fill in Run 2.



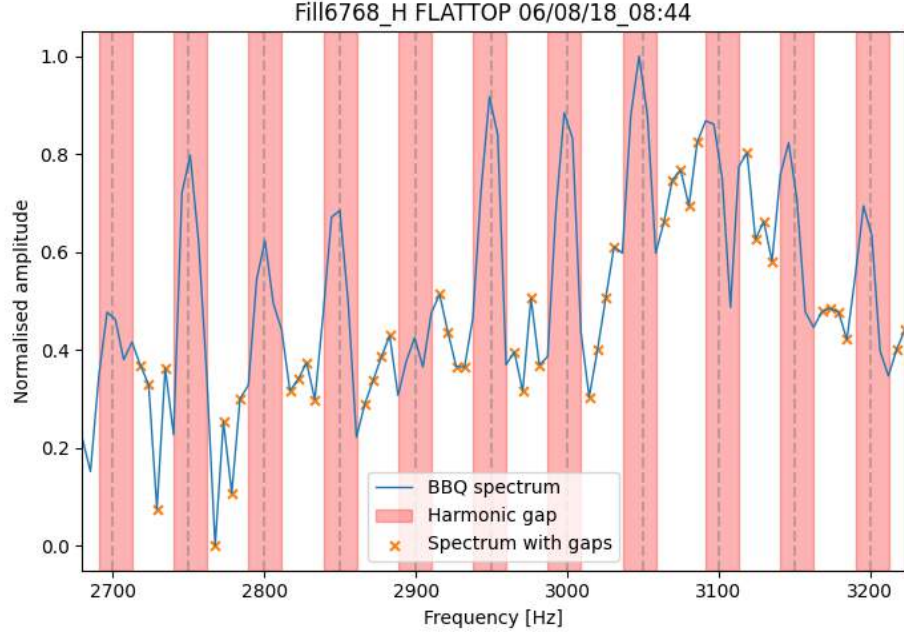


Figure 3.5: An example of a spectrum from Fill 6890, beam 1, horizontal, in beam mode FLATTOP (blue). The red vertical bands correspond to the frequencies which are dropped from the spectrum. The resulting spectrum contains gaps (orange scatter).

### 3.3.1 Polynomial Fit

In the first attempt, polynomial fits were used on the spectra with gaps. The fits used a finer frequency axis, which could capture the tune estimate with a higher resolution. The initial idea was to capture the tune peak by fitting a quadratic curve, and if it converges well, its maximum should coincide with the tune value. Therefore, the tune estimate can be found with respect to the parameters of the quadratic fit,  $F_{poly}$ :

$$\begin{aligned}
 F_{poly}(\omega) &= a\omega^2 + b\omega + c \\
 \frac{d(F_{poly}(\omega))}{d\omega} &= 2a\omega + b \\
 \text{At maximum: } \frac{d(F_{poly}(\omega))}{d\omega} &= 0 \\
 f_{max} &= \frac{-b}{4\pi a} \tag{3.3}
 \end{aligned}$$

While this approach seemed to work on spectra with a symmetrical and centered

tune peak, it suffered over spectra with an asymmetrical tune peak. A quadratic fit over an asymmetrical peak would not necessarily have its maximum coincide with the value of the tune. Another limitation of quadratic fits is that the spectrum has to be clipped so that only the tune peak is present. To decide where to clip the frequency spectrum, various smoothing techniques have to be applied prior to fitting the quadratic curve, to ensure that Equation (3.3) is equivalent to the tune value.

By using higher-order ( $> 10^{th}$  order) polynomials the tune peak asymmetries can be modelled better. However, it was observed that higher-order polynomial fits were susceptible to bad fitting at the edges of the spectrum window. This causes certain spectra to obtain polynomial fits whose edges are higher than the tune peak. This makes it difficult to retrieve the value of the tune from the fit and would need to introduce yet more logic in the algorithm. Due to these problems, this approach is not considered as a viable candidate to replace BQ.

### 3.3.2 Weighted Moving Average (WMA)

The second attempt implemented a Weighted Moving Average (WMA), which slides a window of length  $2L + 1$  over the spectra to produce a smoothed fit of the spectrum with gaps. As an example, consider a window centered around frequency bin  $i$  in the spectrum, then the smoothed value for point  $i$ ,  $\hat{y}$ , can be obtained by:

$$\hat{y}_i(L) = \frac{\sum_{j=-L}^L w(j, L) \times y_{i+j}}{\sum_{j=-L}^L w(j, L)} \quad (3.4)$$

where  $y$  is the spectrum with gaps. The weights,  $w$ , are proportional to the distance of the bin from the center frequency bin of the window :

$$w(k, L) = \begin{cases} L - |k| & , \text{ if } \exists y_{i+j} \\ 0 & , \text{ if } \nexists y_{i+j} \end{cases} \quad (3.5)$$

where the second case occurs at the edges of the spectrum and in the 50 Hz gaps. Therefore, the length of the window must be larger than the number of contiguous frequency bins removed for each 50 Hz noise harmonic. This is to ensure a non-zero denominator in the smoothing operation of Equation (3.4).

Figure 3.7 shows three examples of BBQ spectra (dashed black) with three reconstructed spectra each, obtained from Equation (3.4) with  $L$  set to 5, 10 and 20, respectively. It can be seen that the majority of the frequency bins contained within a harmonic spike have been successfully removed. The reconstruction of the spectrum with gaps is smoother as  $L$  increases, however, there is a probability that this also affects the position of the tune peak.

### 3.3.3 Gaussian Process (GP)

The third and final approach uses a type of Bayesian modelling called Gaussian Processes (GPs), to fit over the spectra with gaps. GPs can be thought of as functions of functions, where given a set of noisy data over domain  $X_1$ , a multi-variate Gaussian distribution is created to sample over another domain,  $X_2$ . The advantage is that  $X_2$  can have a higher resolution than  $X_1$  which makes it possible to easily interpolate over a spectrum with gaps and produce a finer and smoother fit. The multi-variate Gaussian distribution obtained is written as:

$$f(X) \sim \mathcal{N}(\mu = m(X), \Sigma = k(X, X)) \quad (3.6)$$

$m(X)$  in Equation (3.6) obtains the means of  $f(X)$  whilst  $k(X, X)$  is the kernel function, which calculates the covariance matrices. Given some observed data,  $(X_1, Y_1)$ , a covariance matrix,  $\Sigma$ , is obtained from the kernel. One such kernel is the exponentiated quadratic kernel, or the Radial Basis Function kernel. This uses the squared Euclidean distance between points  $x_a$  and  $x_b$  as a measure of correlation between the points:

$$k(x_a, x_b) = e^{-\frac{1}{2\sigma^2} \|x_a - x_b\|^2} \quad (3.7)$$



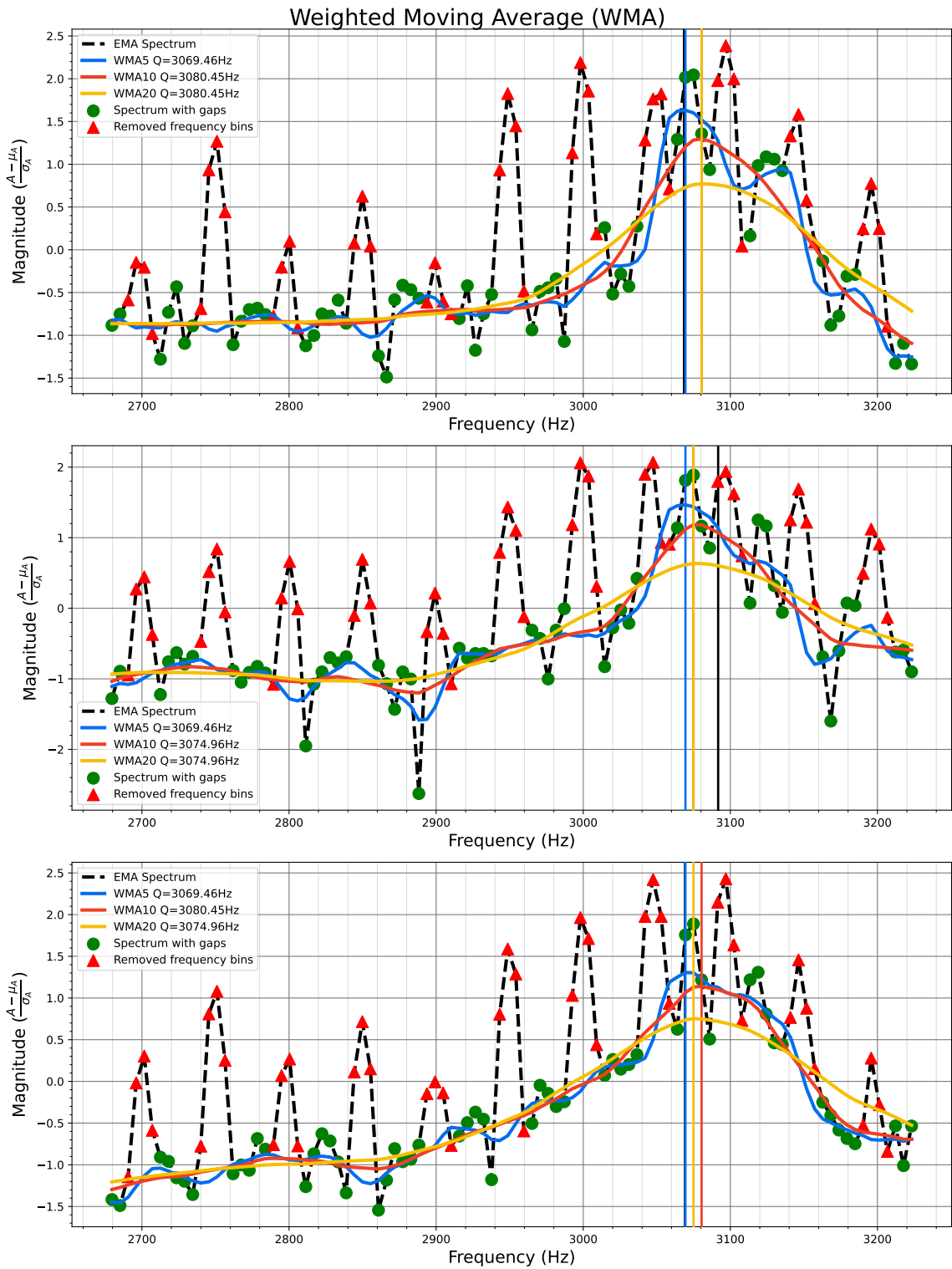


Figure 3.7: Performance of WMA with different window lengths on three samples of real BBQ spectra.

In Equation (3.7),  $\sigma$  is a free parameter which controls the smoothness of the resultant sampled functions from  $f(X)$ . Using this formalism, a multi-variate Gaussian can be formed, which given the observed data  $(X_1, Y_1)$ , can predict  $Y_2$  over a new input dataset,  $X_2$ :

$$P(y_2 | y_1, X_1, X_2) \sim \mathcal{N}(\mu_{2|1}, \Sigma_{2|1}) \quad (3.8)$$

where:

$$\Sigma_{ab} \triangleq k(X_a, X_b) \quad (3.9)$$

$$\mu_{2|1} = (\Sigma_{11}^{-1} \Sigma_{12})^\top y_1 \quad (3.9)$$

$$\Sigma_{2|1} = \Sigma_{22} - (\Sigma_{11}^{-1} \Sigma_{12})^\top \Sigma_{12} \quad (3.10)$$

One advantage of GPs is that the reconstructed fit can be computed over an  $X_2$  with more frequency bins. Therefore, in theory, the location of the tune peak can be found with a higher accuracy. On the other hand, the main disadvantage of GPs is their computational complexity, where  $\Sigma_{11}$  is a  $100 \times 100$  matrix and is constructed for every GP fit calculation. In both Equation (3.9) and Equation (3.10), the inverse of  $\Sigma_{11}$  needs to be calculated, which involves relatively expensive computations. However, when considering the processing power that will be available to the BBQ system in LHC Run 3, the high computational cost can be handled deterministically.

Figure 3.8 shows three examples of BBQ spectra (dashed black) with three reconstructed spectra each, obtained from Equation (3.9), while using an RBF kernel with a length scale of 25, 70 and 130, respectively. It can be seen that the majority of the frequency bins contained within a harmonic spike have been successfully removed and the reconstruction of the spectrum with gaps is smoother as the length scale increases. However, this also slightly affects the position of the tune peak, similarly to WMA.

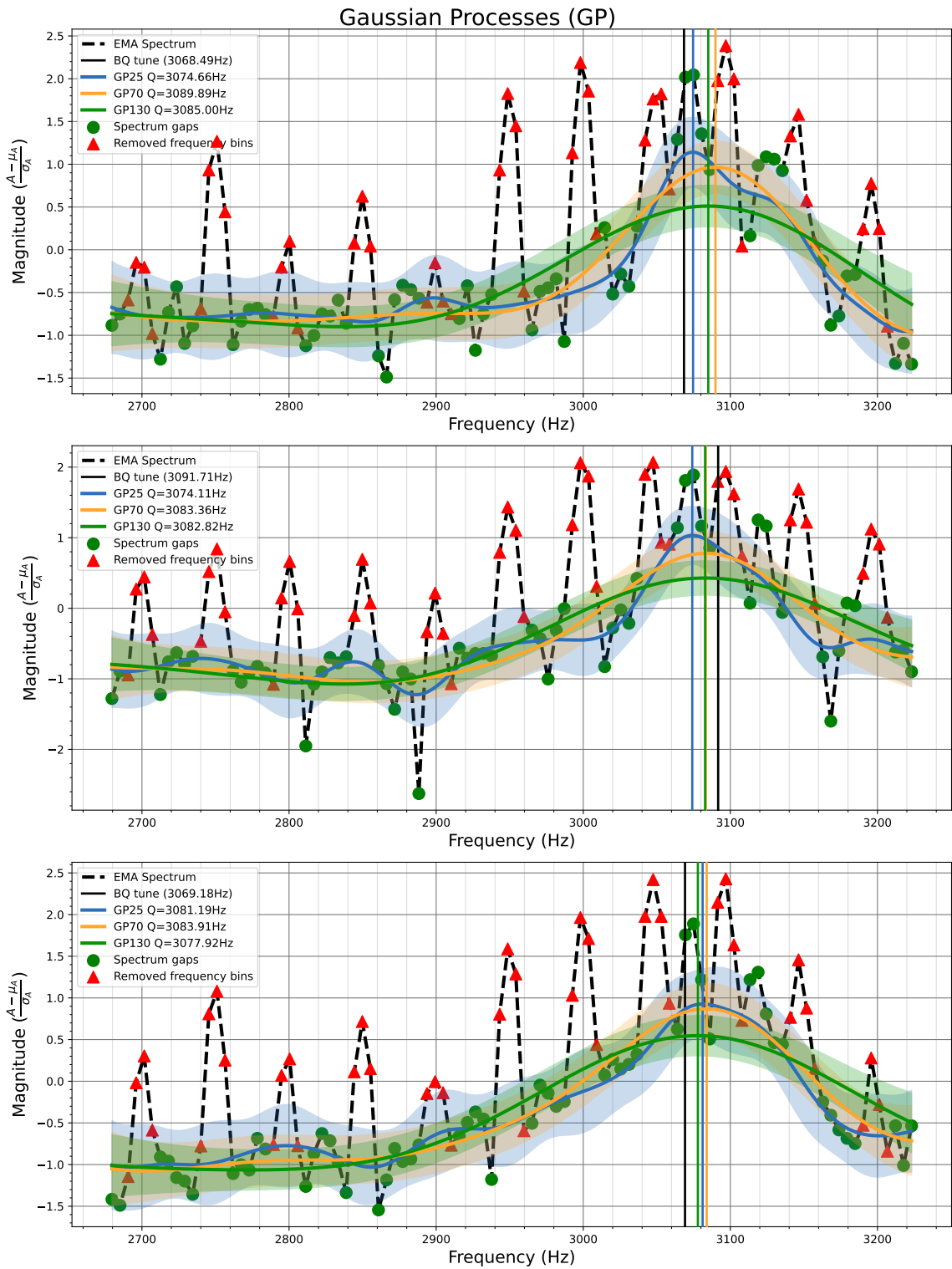
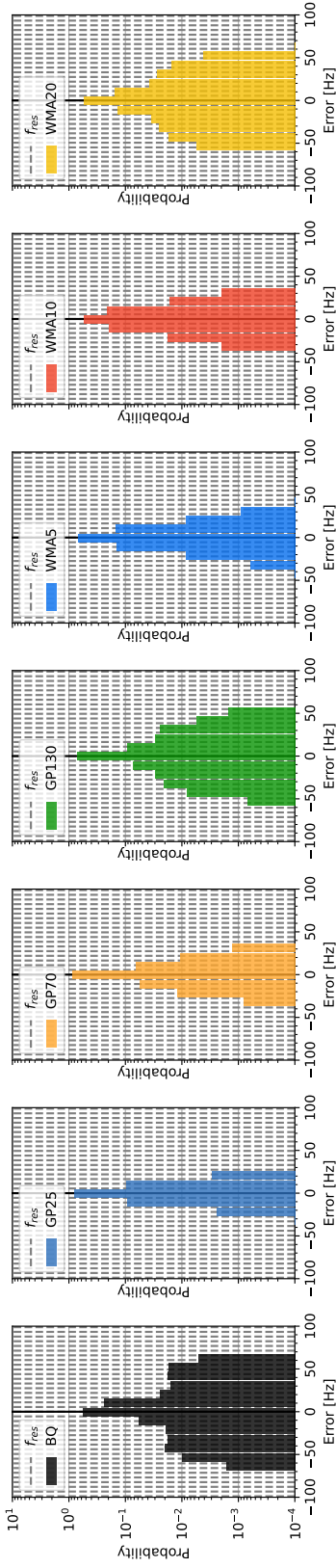
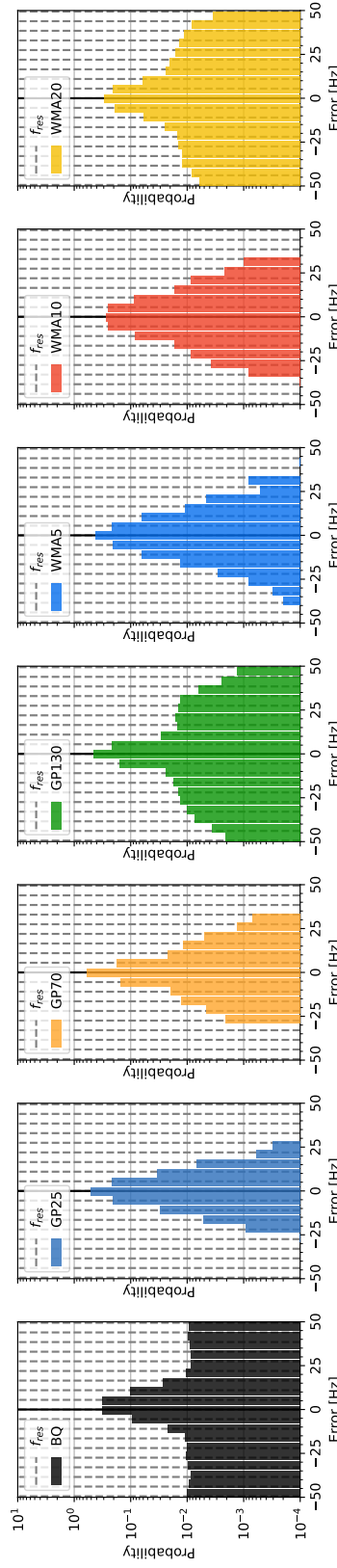


Figure 3.8: Performance of GPs with different RBF kernel length scales on three samples of real BBQ spectra.

Alternative Approach:  $-100 \text{ Hz} < \Delta Q < 100 \text{ Hz}$



Alternative Approach:  $-50 \text{ Hz} < \Delta Q < 50 \text{ Hz}$



Alternative Approach:  $-25 \text{ Hz} < \Delta Q < 25 \text{ Hz}$

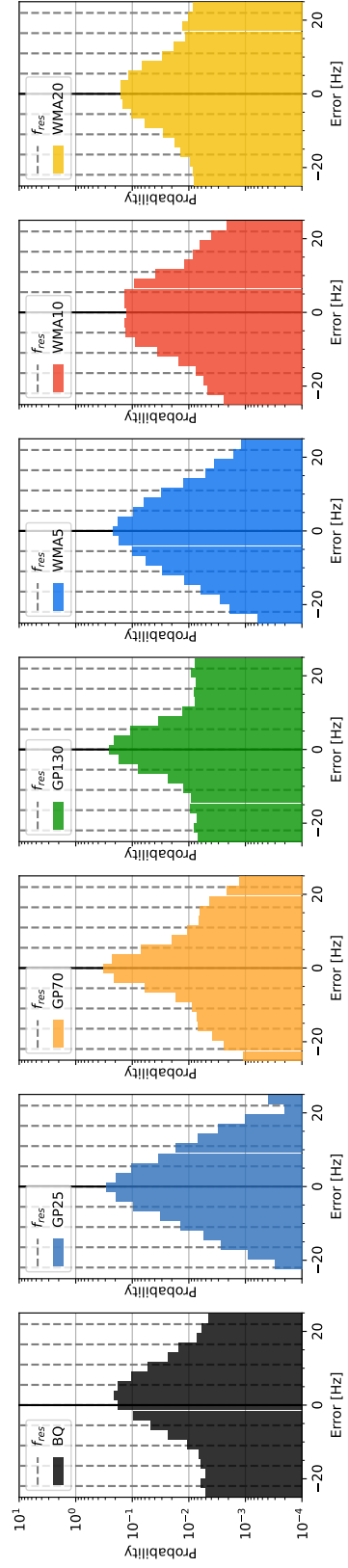


Figure 3.9: Error density histogram from 10000 simulated spectra for alternative approach algorithms.

### 3.3.4 Measuring performance

A Monte Carlo simulation was performed, where 10000 spectra were simulated using the distribution of parameters listed in Section 3.2. GPs and the WMA were used to obtain tune estimates while using different hyperparameters. In particular, the approach using a GP used an RBF kernel with a length scale of 25, 70 and 130, respectively. The approach using using a WMA uses window lengths of 5, 10 and 20, respectively.

Algorithm	$\mu$ [Hz]	$\sigma$ [Hz]	Max. abs. error [Hz]
BQ	2.68	15.45	65.73
GP25	0.11	4.49	27.09
GP70	0.16	4.98	33.58
GP130	0.36	10.77	55.09
WMA5	0.07	5.93	42.49
WMA10	0.06	7.21	34.28
WMA20	0.16	13.92	61.01

Table 3.1: Error statistics from 10000 simulated spectra for alternative approach algorithms (GP and WMA).

The error obtained between each respective tune estimate, and the resonance frequency used to create the spectrum, was used to compare the performance of the alternative approaches. As a reference, the BQ algorithm was also tested on the simulated spectra. Table 3.1 tabulates the average,  $\mu$ , standard deviation,  $\sigma$ , and the maximum absolute error, in Hertz. Figure 3.9 shows the distributions of the errors from each respective algorithm configuration attempted. The results show that BQ was outperformed by all the alternative approaches. In particular, WMA5 and WMA10 obtained the best accuracy while GP25 and GP70 obtained the best precision. The smallest maximum absolute error was obtained by GP25 which suggests that it is the most stable tune estimation algorithm from those attempted. It can also be observed, that the performance of GPs degrades at a slower rate than WMA with respect to the scale of their hyperparameters. This suggests that GP is easier to tune than WMA.

Regardless, the good performance of WMA5 and WMA10, and the simplicity of the WMA algorithm, make WMA the most ideal candidate for replacing BQ within the BBQ system.

## 3.4 A Machine Learning approach

The alternative approaches presented in Section 3.3 provide a possible upgrade which incorporates the observation that the 50 Hz harmonics are relatively stable when compared to the resolution of the spectrum. Despite the removal of points from the spectrum and interpolation by using WMA or GP, it is observed in Section 3.5 that tune estimates over real BBQ spectra still suffered severe inaccuracies.

It is not possible to quantitatively measure the accuracy of the alternative algorithms over real spectra since ground-truth tune values do not exist. Due to this limitation, when the tune estimates are suspected to be inaccurate, operators directly look at the real spectrum and try to estimate the tune empirically.

Machine Learning (ML) provides a set of tools which can be used to build, train and validate a model, over some set of training data. The type of model chosen in this work was an Artificial Neural Network (ANN), which can be trained to predict the tune estimate directly from an EMA spectrum.

### 3.4.1 Simple approach

Section 2.6 introduced and discussed Supervised Learning (SL) where provided a set of correctly labelled data, a model can be trained to predict a label for any data coming from the same distribution as the training data. SL can be used to directly predict the value of the tune for any real BBQ spectrum passed as input to the network. First, a sensible network architecture has to be chosen, and then trained using BBQ spectra and the most reliable tune estimate found using either BQ, WMA or GP. Through this approach however, the tune estimation ANN model can never give predictions that are better than the labels used during training.

As a solution to this problem, a dataset of second-order spectra was simulated as

Table 3.2: Number of nodes per hidden layer used for ANN tune estimation models.

	<b>Layer 1</b>	<b>Layer 2</b>	<b>Layer 3</b>	<b>#<sup>1</sup></b>
<b>Model #0</b>	150	50	10	23,221
<b>Model #1</b>	300	100	20	62,441
<b>Model #2</b>	500	250	50	188,351

<sup>1</sup> Number of trainable parameters

shown in Section 3.2 and the resonant frequencies of the simulated spectra were used as labels. A subset of the dataset was used as training data for the ML tune estimation models and another subset was used for validation.

### **Fully-connected layers**

The first tune estimation model attempted used a fully-connected layer neural network design. Three ANN models were attempted and their respective parameters are tabulated in Table 3.2. These models all contained three hidden layers, the same activation function per hidden layer, but different number of nodes. For the models shown in Table 3.2, Rectified Linear Unit (ReLU) was used for all hidden layers. The last layer contained only one node, densely connected to the penultimate layer. Its activation function was linear.

Figure 3.10 shows the distribution of the errors between the tune estimates obtained by each respective ANN model and the resonant frequencies used by the simulator to create the respective input spectrum. It can be observed that Model #1 and Model #2 have the best performance, in terms of accuracy and precision. It can also be seen despite Model #2 having much more parameters than Model #1, the latter still exhibits a similar performance. By comparing Figure 3.9 and Figure 3.10, it can be seen that the error density plot of the three ML models drops to zero below  $-20$  Hz and above  $20$  Hz, which is already better than the results obtained by the alternative approaches in Figure 3.9.

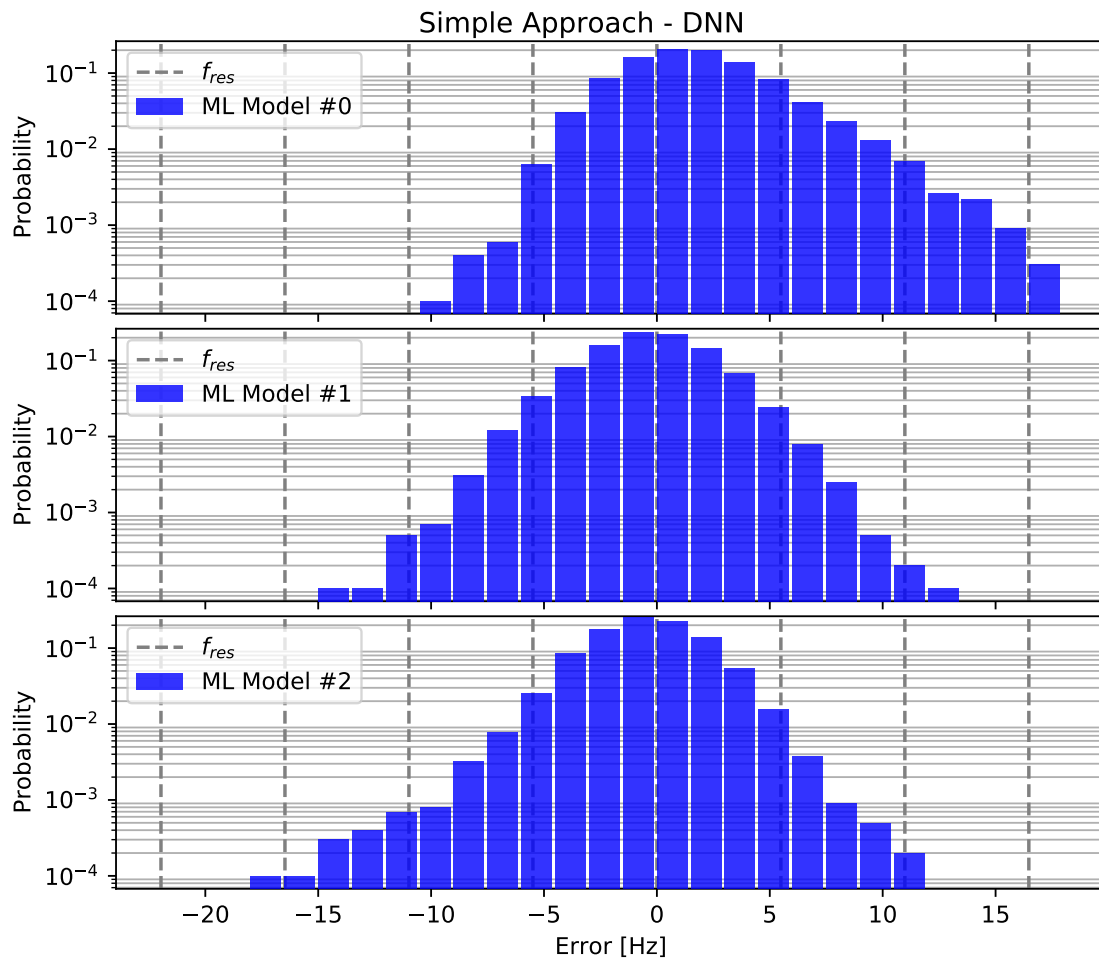


Figure 3.10: Histograms showing the distribution of errors of the tune estimates from the resonance used to simulate the data. Tune estimates obtained using the fully-connected networks; Model #0 to Model #2.



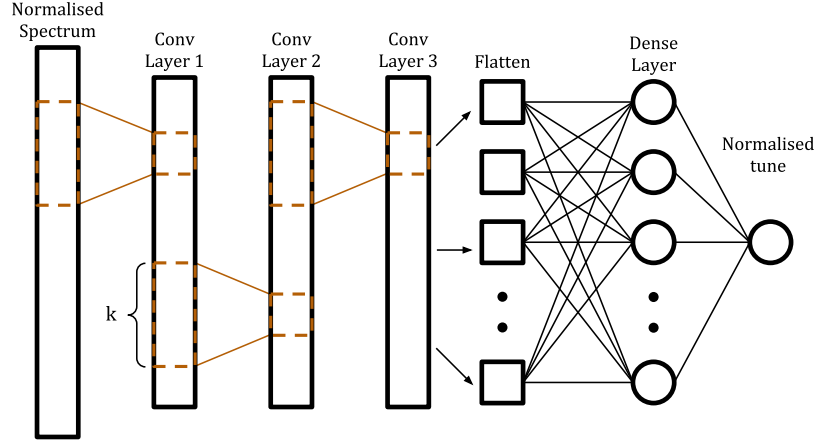


Figure 3.11: Network using convolutional layers

Table 3.3: Model architectures presented for CNNs

	Layer 1			Layer 2			Layer 3			Dense	Parameters
	$f^1$	$k^2$	$s^3$	$f$	$k$	$s$	$f$	$k$	$s$		
<b>Model #3</b>	32	3	3	16	3	3	8	3	3	20	2,753
<b>Model #4</b>	32	3	1	16	3	1	8	3	1	20	18,113
<b>Model #5</b>	64	3	3	32	3	1	16	3	1	20	18,905
<b>Model #6</b>	128	3	3	64	3	3	16	3	3	20	29,561
<b>Model #7</b>	64	3	1	32	3	1	16	3	1	20	40,025

<sup>1</sup> Number of filters    <sup>2</sup> Kernel size of convolution    <sup>3</sup> Stride, shift size of kernel

### Convolutional layers

The next type of neural network design attempted was a Convolutional Neural Network (CNN) where its basic architecture is shown in Figure 3.11. Five CNN models were attempted and their respective parameters are tabulated in Table 3.3. These models all contained three convolutional layers, ReLU activation function in the hidden layers, and one dense layer connected to the flattened feature maps of the last hidden layer, and with one output node having a linear activation. The convolutional layers that were used are adapted to one dimensional data, where a kernel operates on adjacent frequency bins.

Figure 3.12 shows the distribution of the errors between the tune estimates obtained by CNN models and the resonant frequencies used by the simulator to create the respective input spectrum. The best CNN model was ML Model #5, and it can be seen that its error drops to zero below  $-30$  Hz and above  $20$  Hz. This makes its

Simple Approach - CNN

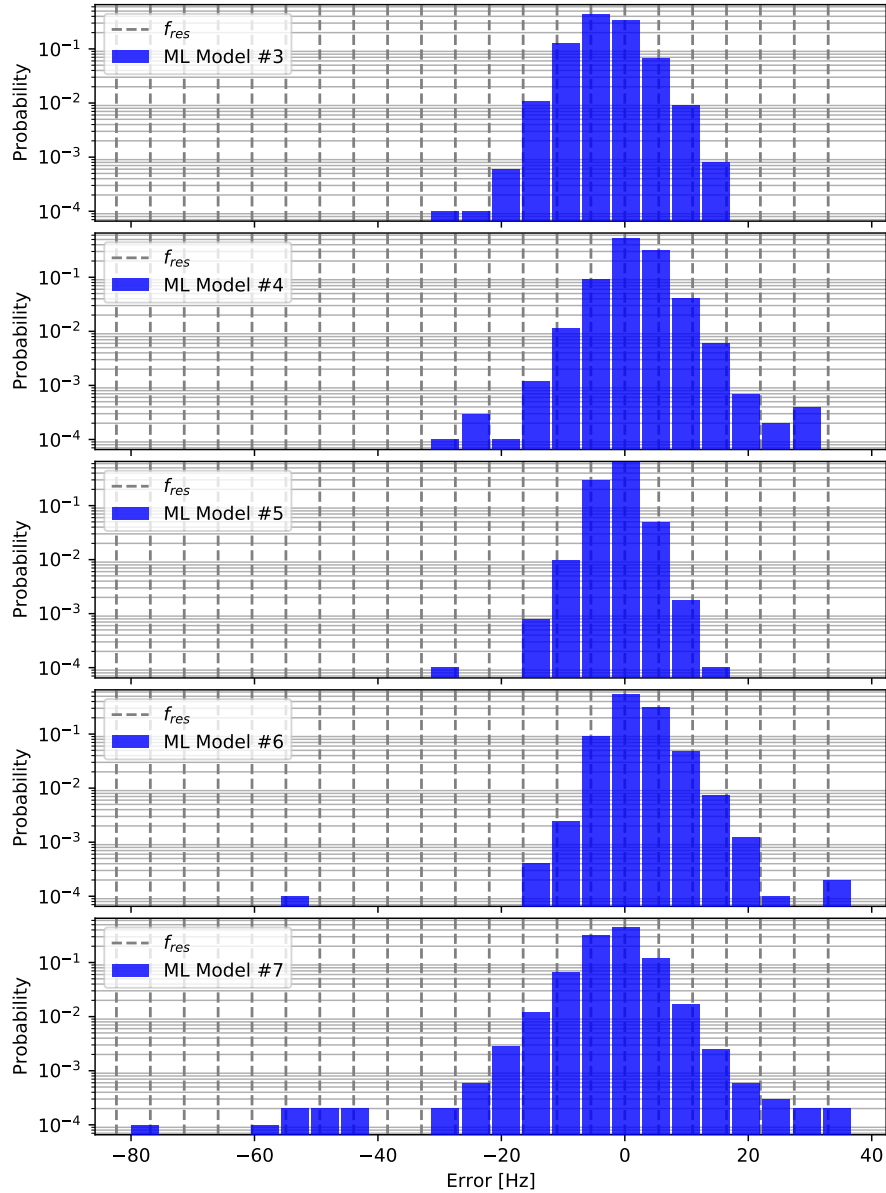


Figure 3.12: Histograms showing the distribution of errors of the tune estimates from the resonance used to simulate the data. Tune estimates obtained using the convolutional networks; Model #3 to Model #7.

performance slightly worse the average performance of the ANN models in Figure 3.10.

### 3.4.2 Improving the dataset with SimGANs

It was observed that regardless of the model architecture used, the performance of the model trained on simulated spectra was sub-optimal in estimating the tune from a real spectrum (Section 3.5 illustrates the sub-optimal tune estimates obtained by ML Model #1 (ML#1) and ML Model #5 (ML#5) when using BBQ spectra instead of simulated spectra). There was a possibility that the models trained so far were over-fit to some features only present in simulated spectra, which would explain the sub-optimal performance on real data. Following this hypothesis, another source of realistic training data was needed to train another model which is more robust on unlabelled real data.

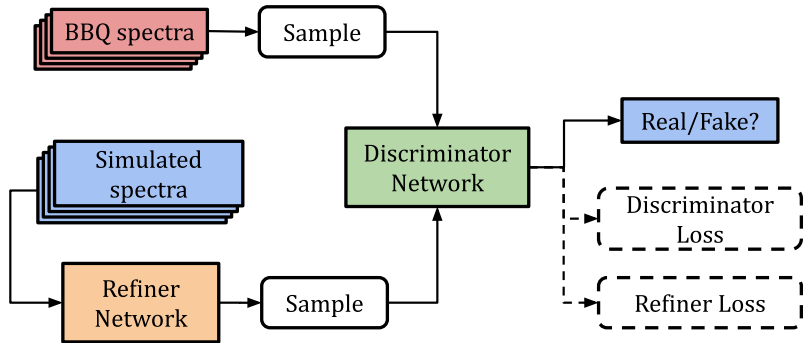


Figure 3.13: Overview of the SimGAN architecture.

Figure 3.13 illustrates the architecture of the SimGAN used in this work. Two notable differences which distinguish SimGANs from GANs are: The input of the refiner is now coming from a simulator; the addition of a regularisation loss to the refiner loss, which restricts the output of the network to remain similar to the input. The discriminator is continuously trained to distinguish between refined and real spectra. This ultimately helps the refiner to create more realistic spectra in an adversarial manner.

SimGANs were originally developed to work with images via 2D convolutional networks. Since the aim is to refine simulated spectra to look more like spectra obtained from the BBQ system, all the networks in Figure 3.13 use 1D convolutional networks.

The theory presented below remains valid since the same losses are minimised, regardless of the input shape and the network type.

The goal is to use a set of unlabelled real data,  $y_i \in \mathcal{Y}$ , to learn a refiner network,  $R_\theta(x)$  that refines simulated data  $x$ , with  $\theta$  as the function parameters. Therefore we can define refined data,  $\tilde{x}$  as:

$$\tilde{x} := R_\theta(x)$$

The key requirement is that  $\tilde{x}$  should look similar in appearance to the real data in the set  $\mathcal{Y}$ , while still preserving the annotation information from the simulator. The training procedure of  $\theta$  involves the minimisation of a combination of two losses:

$$\mathcal{L}_R(\theta) = \sum_i \ell_{real}(\theta; x_i, \mathcal{Y}) + \lambda \ell_{reg}(\theta; x_i) \quad (3.11)$$

where  $x_i$  is the  $i^{th}$  simulated spectrum.  $\ell_{real}$  adds realism and  $\ell_{reg}$  regulates the preservation of the annotation information of the input.  $\ell_{reg}$  is scaled by a scalar  $\lambda > 0$ , which balances the effect of regularisation over realism. Since a refiner makes it difficult to classify spectra as real or refined, an adversarial discriminator  $D_\phi$  is trained to classify spectra as real or refined. Thus the output of the discriminator can be considered as the probability that the input spectrum is real on a scale  $[0, 1]$ . The discriminator updates  $\phi$  by minimising the following cross-entropy loss:

$$\mathcal{L}_D(\phi) = -\sum_i \log(D_\phi(\tilde{x}_i)) - \sum_j \log(1 - D_\phi(y_j)) \quad (3.12)$$

Also note that  $\ell_{real}$  can be formed by using  $D_\phi$  to update  $\theta$ :

$$\ell_{real}(\theta; x_i, \mathcal{Y}) = -\log(1 - D_\phi(R_\theta(x_i))) \quad (3.13)$$

By minimising Equation (3.13),  $\theta$  is updated in the direction that makes  $D_\phi$  classify refined spectra as real, thus improving the performance of the refiner by using the discriminator as a metric. Apart from this,  $\ell_{reg}$  is introduced as a self-regularisation measure to preserve the annotation information of the input:

$$\ell_{reg} = \|\psi(\tilde{x} - x)\|_1$$

where  $\psi(\cdot)$  modifies how the regularisation loss is obtained. For example  $\psi$  could be an identity matrix which maps the input to itself. In this work,  $\psi$  was configured to satisfy an empirical observation that the 50 Hz noise harmonics in real spectra are always additive artefacts and never manifest as dips. Due to this observation  $\psi(\cdot)$  was designed as shown in Equation (3.14) so that any artefacts introduced below the baseline have a higher cost on the model for  $\eta > 1$ .

$$e \triangleq \tilde{x} - x$$

$$\ell_{reg} = \sum_i \max(0, e_i) + \eta \sum_i \max(0, -e_i) \quad (3.14)$$

### 3.4.3 Training

The second-order spectrum simulator that was used to create the spectra to train the models in the Simple Approach was modified to not inject the 50Hz noise harmonics. Instead the simulator would now only create a spectrum from Equation (3.1) and add Gaussian noise. Note that the effect of noise on the input is twofold: 1) It helps the refiner to generalise the refinement process, as in any update process utilising backward propagation [92]; 2) It is the only source of randomness which the refiner can use to generate somewhat different artefacts in its output.

The real dataset,  $\mathcal{Y}$ , used in Equation (3.12) was obtained from the BBQ data logged during Fill 6890, Beam 1, horizontal plane, during FLATTOP. Since the input size of all networks considered in this work was 100, each real spectrum was truncated by a frequency window having a randomised position, while guaranteeing that the dominant peak of the spectrum lies within said window. This was done to ensure that the discriminator does not over-fit to the dominant peak always occurring at the same relative location with respect to the frequency window. This ensures stabler and more generalised training of all the constituent components of SimGAN.

The network architectures used for both the refiner and discriminator are a 1D

version of the networks used in [66]. All convolutional layers used were 1D and unless otherwise specified, all stride lengths were 1. The refiner was a Residual Network (ResNet) with an input size of 100 which was fed to a first convolutional layer having 64 filters and a kernel size of 3. The output of this layer was fed to 4 ResNet blocks connected in sequence. Each block consisted of two convolutional layers having 64 filters each with a kernel size of 3 and a connection merging the input of the block with the output of the block's second convolutional layer. The output of the last ResNet block was then fed to a final convolutional layer with 1 filter having a kernel size of 1, producing an output of size 100. All hidden layers of the refiner used the ReLU activation function and the output was linear. The discriminator was a 1D convolutional neural network with an input layer size of 100 which was fed to a convolutional layer having 96 filters with a kernel size of 3 and a stride length of 2, followed by another convolutional layer having 64 filters with a kernel size of 3 and a stride length of 2. Following the second hidden layer, Max pooling was applied with a pool size of 3. The output of the pooling was fed to a convolutional layer having 32 filters, kernel size of 3 and a stride length of 3 followed by two convolutional layers having 16 and 2 filters respectively and a kernel size of 1. All hidden layers of the discriminator used the ReLU activation function and the last layer had an output size of 2 with the softmax activation function. The two outputs of the discriminator correspond to the probabilities that the input spectrum is real or refined, respectively. All layer weights were initialised using the Glorot normal initialiser. Tables 3.4 and 3.5 summarise the network architectures used for SimGAN.

Table 3.4: SimGAN Refiner network architecture

<b>Refiner Network Architecture</b>					
<b>Layer type</b>	<b>Number filters</b>	<b>Kernel size</b>	<b>Stride length</b>	<b>Activation type</b>	
Conv1D	64	3	1	ReLU	
ResNet block	Conv1D	64	3	1	ReLU
	Conv1D	64	3	1	ReLU
ResNet block	Conv1D	64	3	1	ReLU
	Conv1D	64	3	1	ReLU
ResNet block	Conv1D	64	3	1	ReLU
	Conv1D	64	3	1	ReLU
ResNet block	Conv1D	64	3	1	ReLU
	Conv1D	64	3	1	ReLU
Conv1D	1	1	1	Linear	

Table 3.5: SimGAN Discriminator network architecture

<b>Discriminator Network Architecture</b>				
<b>Layer type</b>	<b>Number filters</b>	<b>Kernel size</b>	<b>Stride length</b>	<b>Activation type</b>
Conv1D	96	3	2	ReLU
Conv1D	64	3	2	ReLU
MaxPool	Pool size = 3			
Conv1D	32	3	3	ReLU
Conv1D	16	1	1	ReLU
Conv1D	2	1	1	SoftMax

The initial value of  $\eta$  in Equation (3.14) was set to 5 and the initial value of  $\lambda$  in Equation (3.11) was set to  $1 \times 10^{-3.8}$ . These values were found empirically to achieve the best balance between realism and preservation of the annotation information from the simulated spectra to the refined spectra. During the training of one SimGAN it was observed that it is possible for a refiner to learn to add valid artefacts which are able to trick its respective discriminator into classifying the refined spectra as real. However, the ultimate goal was to generate spectra with enough variety in their shapes to reliably train a tune estimation model. Equation (3.11) only constrains the refiner

network to make simulated spectra look similar in appearance to spectra obtained from the BBQ system. An individual refiner is not incentivised to learn different types of artefacts and in fact using one refiner to generate realistic spectra caused the tune estimation model to over-fit to the type of artefacts that one refiner produced.

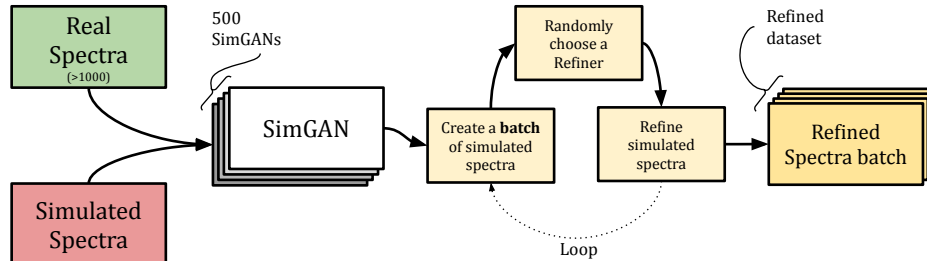


Figure 3.14: Refined spectrum generation scheme using a bank of trained SimGANs.

To overcome this problem, the scheme shown in Figure 3.14 was developed, to obtain a new set of labelled spectra (Refined dataset) to train a new tune estimation model. In this scheme, 500 SimGANs were initialised with different values of  $\lambda$  in Equation (3.11), and trained individually. A good distribution for  $\lambda$  was found empirically to be;

$$\lambda \sim 10^{\mathcal{U}(-4, -3.5)}$$

Due to Equation (3.11), a refiner with  $\lambda = 10^{-4}$  introduces somewhat less realism than a refiner with  $\lambda = 10^{-3.5}$ . In turn, this variety allowed the tune estimation model to become more generalised and perform better over real BBQ spectra. Results from randomly selected refiners can be seen in Figure 3.15. It can be seen that from a baseline simulated spectrum (red), a trained refiner is able to add artefacts to create a refined spectrum (blue). The refined spectra look similar to real BBQ spectra, such as the one shown in Figure 2.12b.

The results shown in Section 3.4.1 were bootstrapped to obtain the best performing network architecture for the tune estimation problem. The same architecture as ML#1 was used for ML-refined, a model trained by a Refined dataset of 10000 synthetic spectra.



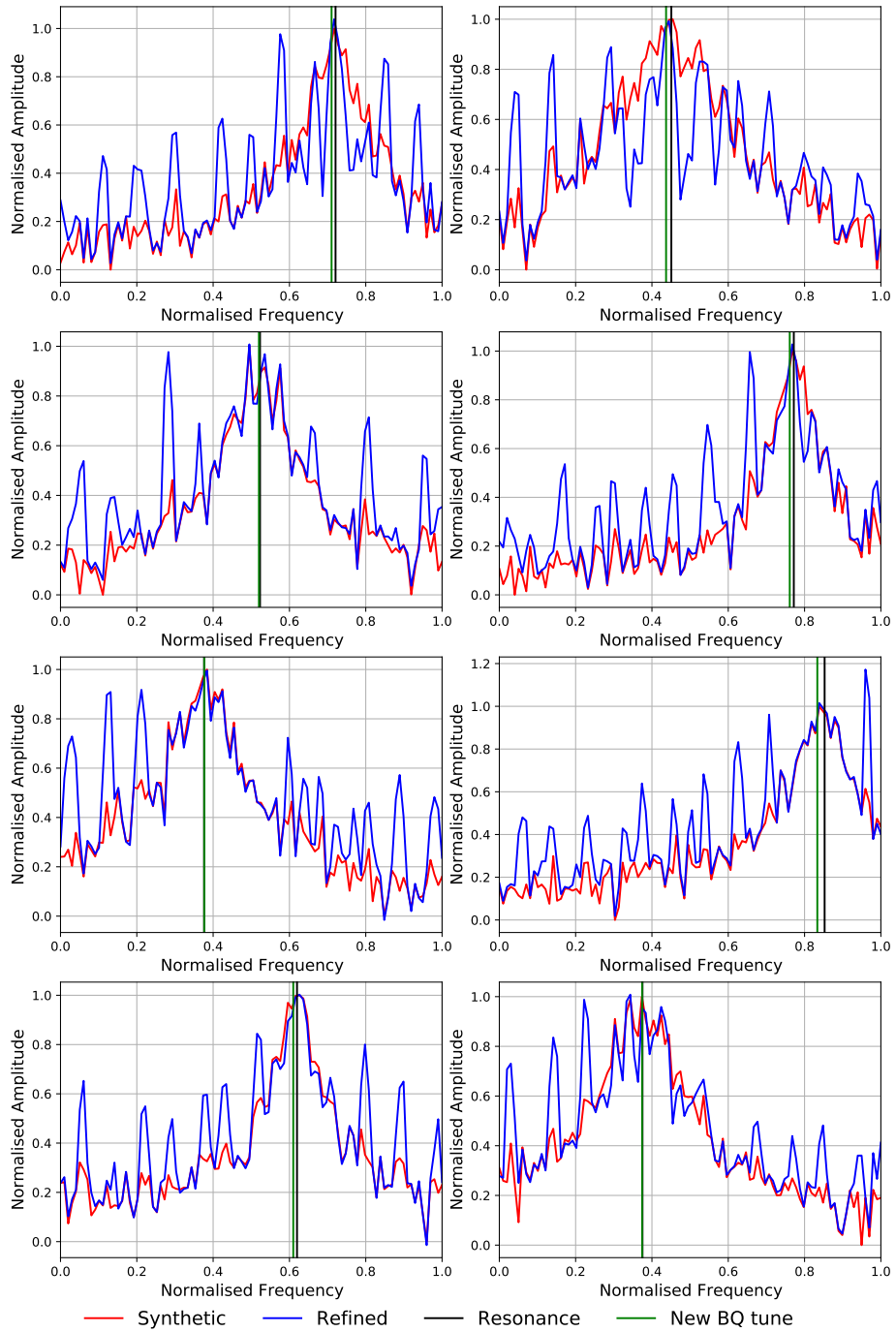


Figure 3.15: Results from a trained SimGAN. The green line represents the tune estimate from BQ algorithm on the refined spectrum.

### 3.5 Results comparison and analysis

When using real BBQ spectra, it is difficult to show any objective difference among the performances of the original tune estimation algorithm BQ, the new ML tune estimation models and the alternative using WMA and GPs. The accuracy and precision of any tune estimation system can only be assessed visually by superimposing the tune estimates over the real spectra.

A set of spectra obtained in the LHC Run 2 during the transition from INJPHYS to ADJUST were used to evaluate the performance of the tune estimation systems. It was made sure that the spectra used during evaluation were not used during the SimGAN training in order to remove any bias. Specifically, BBQ spectra from Fill 6768, Beam 1, horizontal plane were used. A tune evolution estimate was calculated from each spectrum by BQ, GP25, GP70, GP130, WMA5, WMA10, WMA20, ML#1, ML#5 and ML-Refined, respectively. The BBQ spectra were converted to heat maps by filtering out the 50 Hz harmonics and smoothing in time and frequency axes. Moving averages ( $\mu$ ) and moving standard deviations ( $\sigma$ ) of the tune estimates were performed using a centered window of length 10. Subsequently, the corresponding  $\mu$  and  $3\sigma$  were superimposed on the heat maps.

Figure 3.16 shows the BBQ spectral evolution from the start of INJPHYS to the end of PRERAMP. Figure 3.17 shows the BBQ spectral evolution from the start of RAMP to the end of FLATTOP. Figure 3.18 shows the BBQ spectral evolution from the start of SQUEEZE to the end of ADJUST. A perfect tune estimate is assumed to coincide with the peaks of the smoothed spectra and would appear to be centered within the regions of the heatmap coloured white. Upon inspection of the tune estimates until FLATTOP it is clear that ML-Refined obtained the most accurate tune estimates from all approaches attempted. Beyond FLATTOP, the shapes of the spectra degrade, with momentary instances of a peak forming. It can be observed, that only BQ and ML-Refined always manage to obtain tune estimates coinciding with these momentary peaks

Figure 3.12 shows that ML#5 performed well over simulated data, however, all the

results on BBQ spectra show otherwise. ML#1 obtained a comparable performance to the BQ algorithm, however, beyond FLATTOP ML#1 is not as successful as ML-Refined. These results also prove that simply training with simulated data does not obtain a robust tune estimation model to real data.

As mentioned in Section 2.3.2, the tune estimates are used in the QFB to control the quadrupoles and correct the tune of the LHC. In order to ensure a stable operation, the QFB measures the stability of the tune estimates with the following formulas:

$$\begin{aligned}\Delta q_t &= q_t - q_{t-1} \\ S_t &= S_{t-1} * (1 - \alpha) + \Delta q_t * \alpha\end{aligned}\tag{3.15}$$

where  $q_t$  is the tune estimate at time  $t$ ,  $\alpha$  is the time constant of the exponential moving average and  $S_t$  is the stability measure at time  $t$ . The evaluation of both the long and short term tune estimate stability is done via two independent stability measures with a different  $\alpha$ . Each  $\alpha$  corresponds to a fast (2 s) and slow (10 s) time constant. To ensure a stable tune estimate, both stabilities are required be below a certain threshold, which is by default 0.005 ( $\approx 56$  Hz). It is important to note that the tune estimates such as those shown in Figure 2.15, would obtain a relatively low value of instability.

The stability metric shown in Equation (3.15) was used to quantitatively assess the performance of the trained models with regard to the requirements of the QFB. Figure 3.19 shows the stability probability distribution of each model and algorithm used in this work. Tune estimates were obtained from the unseen BBQ spectra of Fill 6768, beam 1, horizontal plane, from INJPHYS to FLATTOP and amounting to a total of approximately 21000 real BBQ spectra. Note that the initial spectra in INJPHYS (until  $t \approx 180s$  in Figure 3.16), were discarded for the calculation of the stability. This was done since the stability metric only makes sense in the presence of well defined tune peaks. The blue and orange histograms correspond to the fast and slow stability measurements, respectively. It is shown that the BQ algorithm is

Fill 6768 - Beam 1 - Plane H

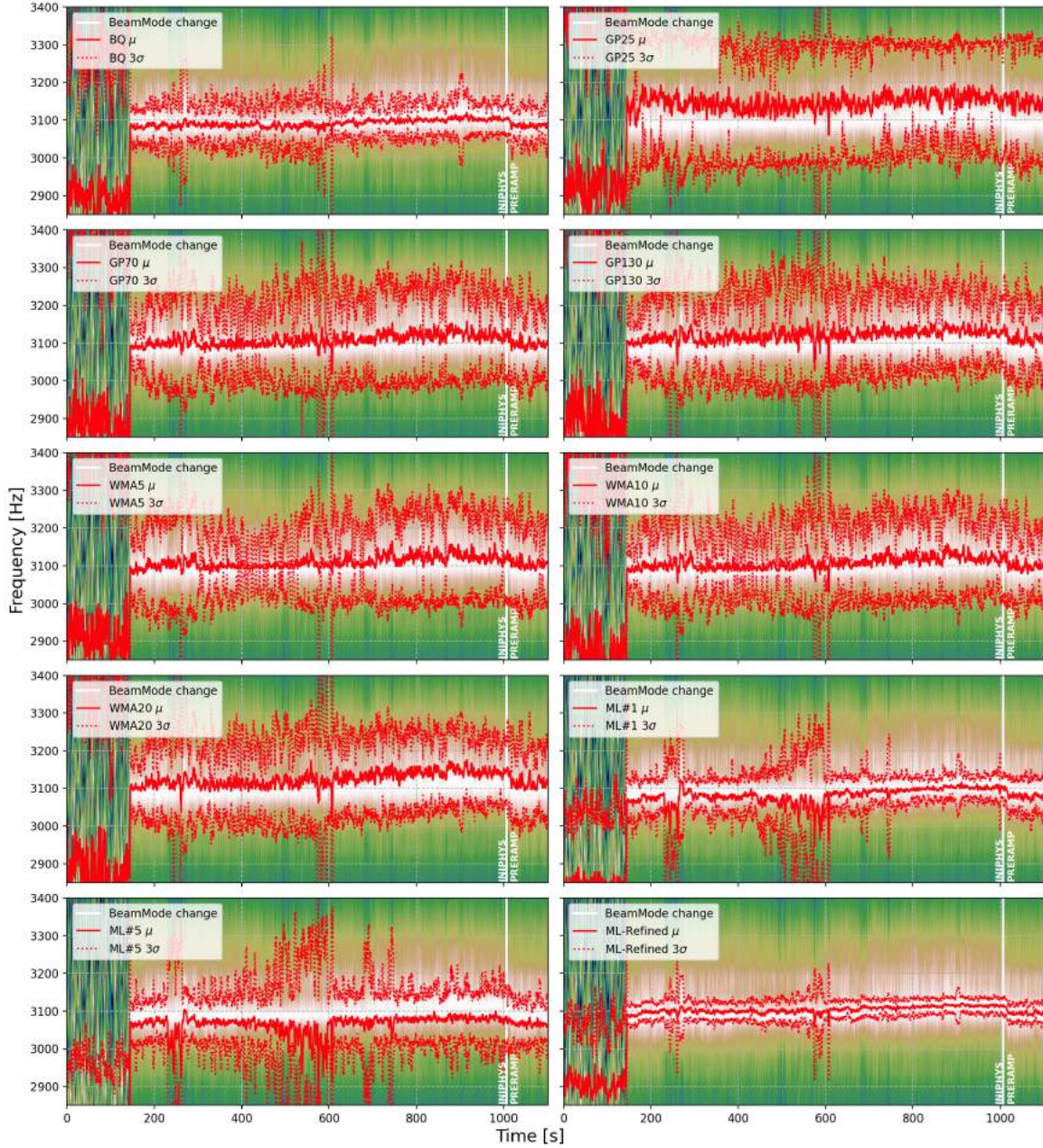


Figure 3.16: (Background) Heat map obtained by post-processing BBQ spectra from LHC Fill 6768, Beam 1, Horizontal plane, in the transition from INJPHYS to PRE-RAMP. (Foreground) mean,  $\mu$ , and scaled standard deviation,  $3\sigma$ , of the tune evolution as estimated by different tune estimation algorithms and ML models: original algorithm (BQ), Gaussian Process (GP) with RBF kernel length scale 25 (GP25), GP70, GP130, Weighted Moving Average (WMA) with window length 5 (WMA5), WMA10, WMA20, best ANN model from Simple approach (ML#1), best 1D CNN model from Simple approach (ML#5) and ANN model trained using spectra refined by SimGAN.



Fill 6768 - Beam 1 - Plane H

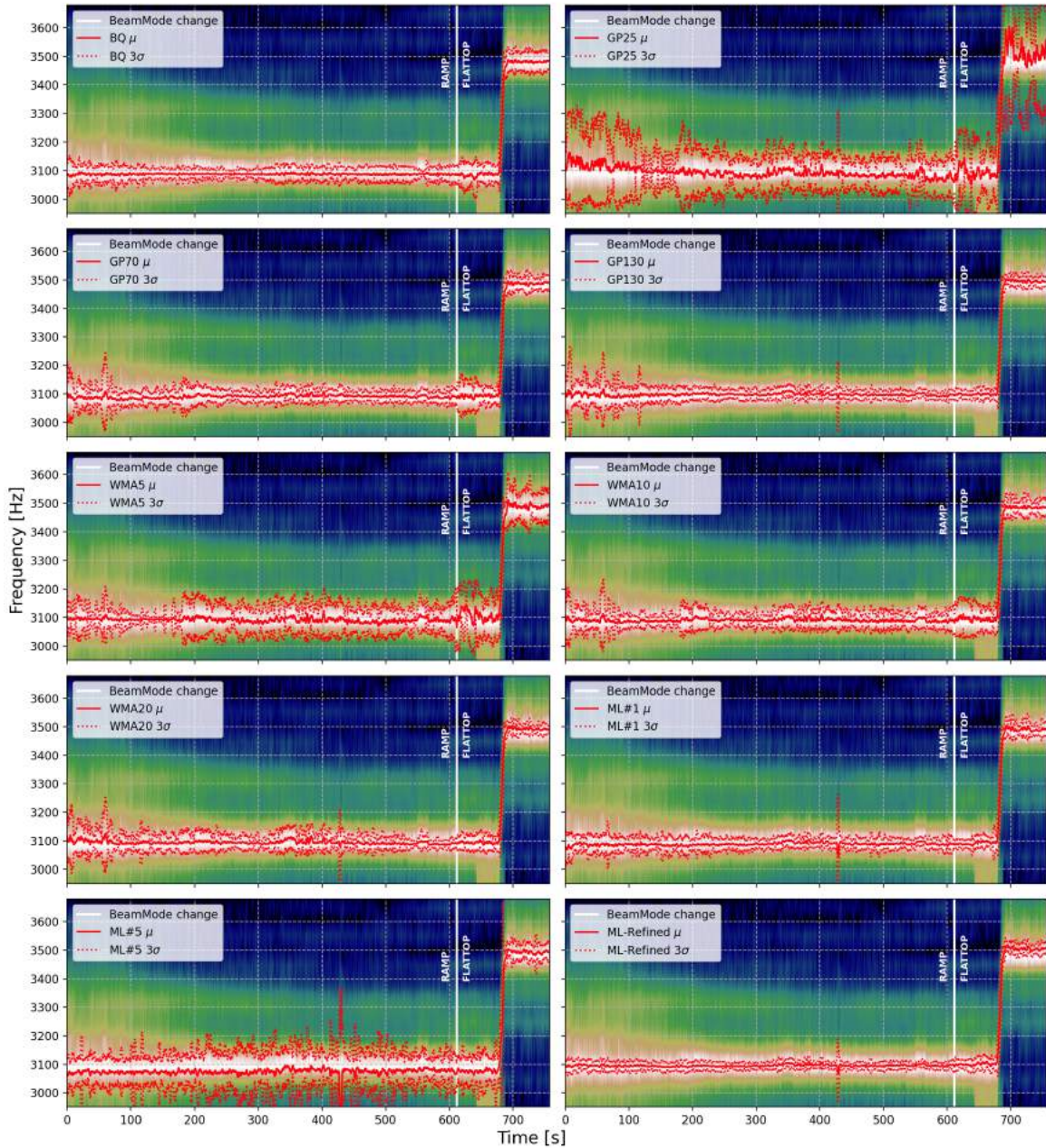


Figure 3.17: (Background) Heat map obtained by post-processing BBQ spectra from LHC Fill 6768, Beam 1, Horizontal plane, in the transition from RAMP to FLATTOP. (Foreground) mean,  $\mu$ , and scaled standard deviation,  $3\sigma$ , of the tune evolution as estimated by different tune estimation algorithms and ML models: original algorithm (BQ), Gaussian Process (GP) with RBF kernel length scale 25 (GP25), GP70, GP130, Weighted Moving Average (WMA) with window length 5 (WMA5), WMA10, WMA20, best ANN model from Simple approach (ML#1), best 1D CNN model from Simple approach (ML#5) and ANN model trained using spectra refined by SimGAN.



Fill 6768 - Beam 1 - Plane H

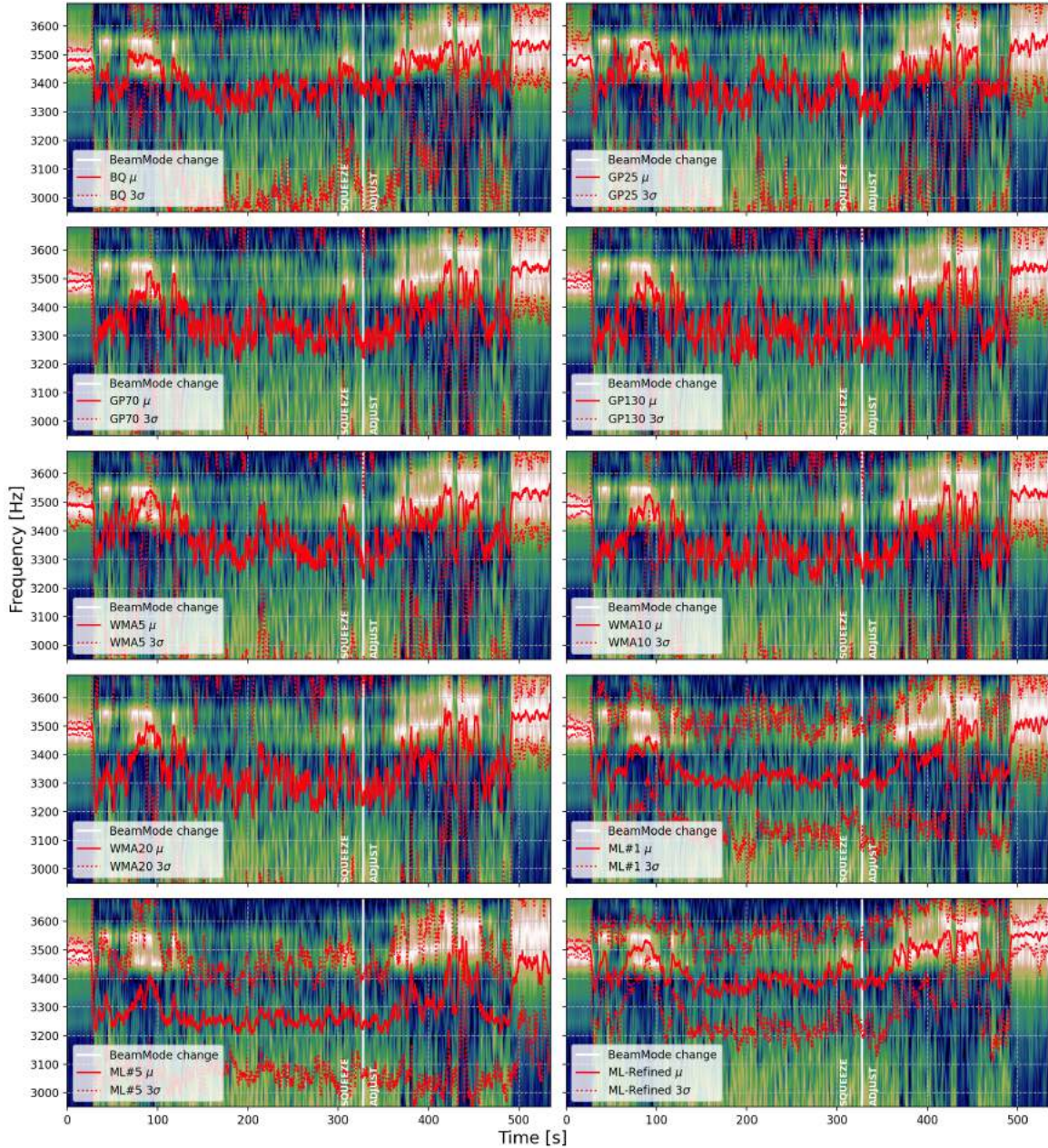


Figure 3.18: (Background) Heat map obtained by post-processing BBQ spectra from LHC Fill 6768, Beam 1, Horizontal plane, in the transition from SQUEEZE to ADJUST. (Foreground) mean,  $\mu$ , and scaled standard deviation,  $3\sigma$ , of the tune evolution as estimated by different tune estimation algorithms and ML models: original algorithm (BQ), Gaussian Process (GP) with RBF kernel length scale 25 (GP25), GP70, GP130, Weighted Moving Average (WMA) with window length 5 (WMA5), WMA10, WMA20, best ANN model from Simple approach (ML#1), best 1D CNN model from Simple approach (ML#5) and ANN model trained using spectra refined by SimGAN.

marginally stable, while all alternative approach algorithms exhibit a wide distribution of both fast and slow stabilities. This indicates a high risk that in real operation the QFB switches off due to instability. ML#5 is shown to provide the most unstable tune estimates from the simple approach while ML#1 is somewhat stabler. Upon examination of the BQ tune estimates, it was observed that the effect of the 50 Hz harmonics may improve stability due to the tune estimates being stuck to one noise harmonic closest to the tune peak. Finally ML-Refined is shown to provide the most stable tune estimate with a zero probability of the QFB switching off, at least for the BBQ spectra used for evaluation.

## 3.6 Summary

Section 3.1 starts with an overview of the systems responsible for tune estimation as existing the LHC by the start of LS2. Following this is the introduction of a new simulation procedure in Section 3.2, which was extensively used during the development of the new tune estimation algorithms. Section 3.3 provides more information about the tools used during the development of the novel algorithmic approaches to tune estimation. This section concludes with a Monte Carlo simulation to compare the performance of the different algorithmic approaches. Section 3.4 goes through the development cycle of the data-driven approach to tune estimation. Finally, results from the original and the new tune estimation algorithms are collated and discussed in Section 3.5.

Considering that this work was carried out during LS2, the results presented in Section 3.5 were obtained using real data logged during the LHC Run 2. Specifically, an unseen dataset was used to qualitatively compare the tune estimates obtained by the various methods proposed. This dataset was also used to deduce the resultant effect of the new tune estimation algorithms on the stability of the QFB within the BBFS. It was found that the data-driven approach outperforms both the original and the alternative algorithmic approaches to tune estimation.

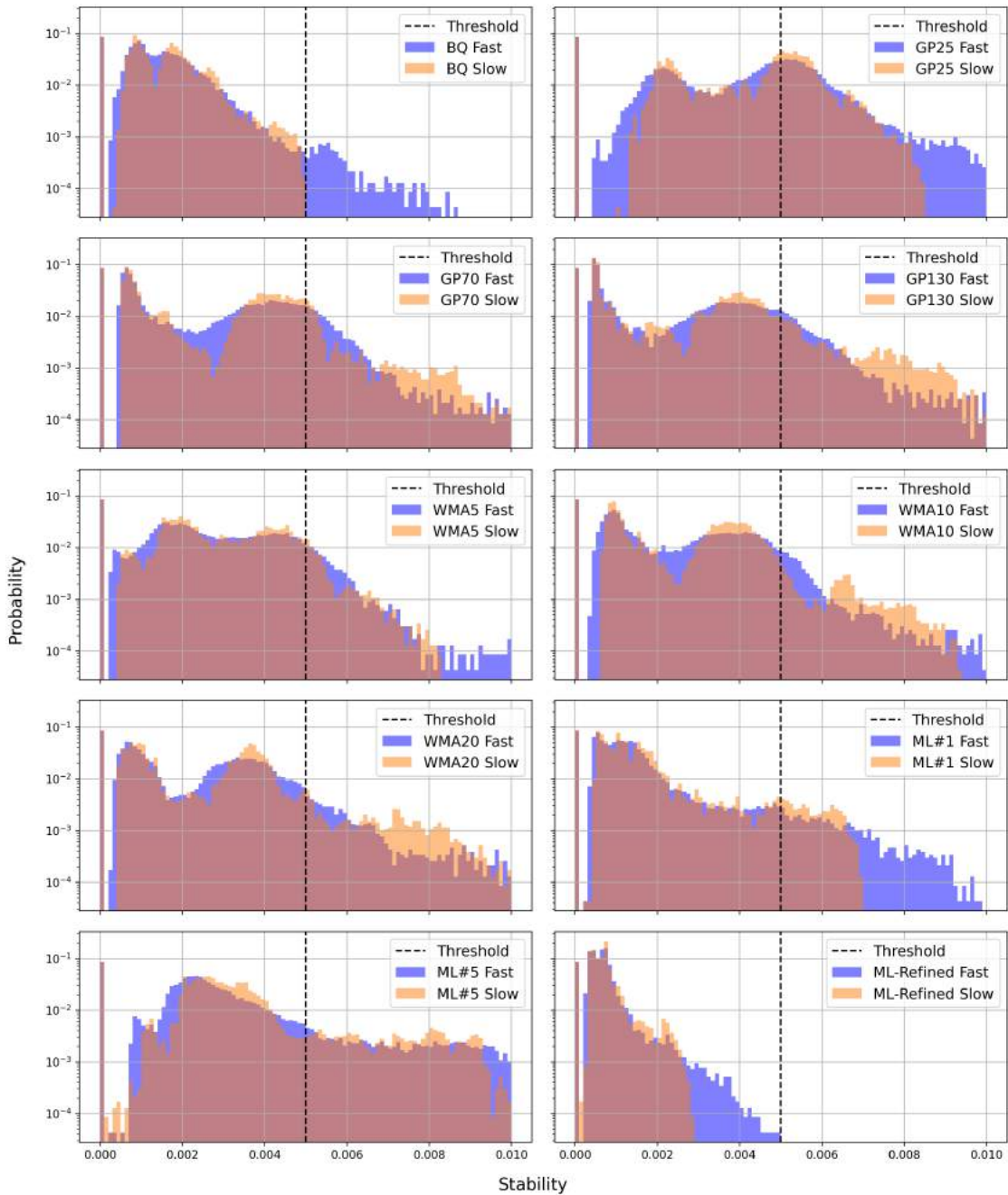


Figure 3.19: Probability distribution of the tune estimation stability. Obtained from Fill 6768, beam 1, horizontal plane using spectra from INJPHYS to FLATTOP. Slow and Fast correspond to stability measures having time constants of 10 s and 2 s respectively (Equation (3.15)). Original algorithm (BQ), Gaussian Process (GP) with RBF kernel length scale 25 (GP25), GP70, GP130, Weighted Moving Average (WMA) with window length 5 (WMA5), WMA10, WMA20, best ANN model from Simple approach (ML#1), best 1D CNN model from Simple approach (ML#5) and ANN model trained using spectra refined by SimGAN. Threshold=0.005, chosen by LHC operators.



# Chapter 4

## Application of Reinforcement

### Learning in a beam-based feedback controller

All beam-based feedback controllers until LS2 rely on a linear beam optics model in the form of a  $n \times m$  matrix. This matrix is also known as a Response Matrix (RM) where  $n$  is the number of actuators and  $m$  is the number of sensors. Table 4.1 tabulates the actuators and sensors used by the Orbit Feedback Controller (OFC) and Tune Feedback (QFB). For simplicity, only the number of actuators and sensors for Beam 1 are listed. The inclusion of Beam 2 in the OFC would also include the Orbit Corrector Dipoles (CODs) acting on both beams simultaneously in the Insertion Regions (IRs) housing the 4 main experiments of the LHC. The number of actuators and sensors of the OFC in Table 4.1 are listed for a controller acting on beam 1 on the horizontal plane. On the other hand, the QFB was always split in two. The two beams do not interact with one another with respect to the tune and therefore, the tunes on different beams are controlled via two independent controllers. The influence of one plane on another with respect to the tune is handled by a separate coupling feedback. For simplicity, the number of actuators and sensors considered for the QFB as shown in Table 4.1 are listed for a controller acting on beam 1, on the horizontal tune and only using MQT type correctors.

Feedback System on Beam 1	Actuator	Sensor
OFC	Orbit Corrector Dipole (COD) $n = 276$	Beam Position Monitor (BPM) $m = 544$
QFB	Tuning Quadrupole $n = 16$	Base-Band Tune (BBQ) $m = 2$

Table 4.1: Actuators and Sensors of the beam-based feedback systems considered for RL.

The RM assumes that the interface between the controller and the beam is ideal, and that the currents sent to the actuators/magnets are always faithfully applied. This assumption is incorrect when considering the boundary conditions set by the BBFS until Run 2. In particular, the set of current corrections were scaled by a factor which ensured that all the magnets can be supplied current by the power converters. E.g. if the current rate limit for a magnet,  $M$ , is  $0.5 \text{ A s}^{-1}$  and a correction is sent that requires  $1 \text{ A s}^{-1}$  to apply, the currents of *all* the magnets would be scaled by a factor  $k \leq 0.5$ , to accommodate  $M$ . Moreover, the magnet which requires the smallest  $k$ , determines the scaling of the current corrections that are sent to the magnets. This approach always ensures convergence given that the model is representative of the LHC magnetic lattice. Regardless, the scaling of the corrections results in sub-optimal behaviour.

Reinforcement Learning (RL) offers the possibility to train an agent by interacting with the LHC and producing an optimal correction without access to the beam optics. One major drawback of current vanilla RL algorithms is the large amount of interactions needed with the environment for the agent to converge to an optimal policy. In general, the larger the environment the more interactions are needed. The difficulty of standard RL toy-problems in the same class of problems as the beam-based controllers can be normally ranked by the number of continuous actions available to the agent. As an example, the simple *Humanoid* environment in [77] contains 21 continuous actions and 67 observables and is considered a hard problem for current state-of-the-art RL algorithms. Considering that the OFC has hundreds of observables and hundreds of continuous actions, it is unlikely that present-day RL algorithms offer a tractable solution. The standard control approach taken by the OFC is at worst

$O(\min(m^2n, mn^2)) \equiv O(mn^2)$ , and this occurs when a change in beam optics due to malfunctions is recomputed through Singular Value Decomposition (SVD). RL offers the possibility to obtain a linear complexity given enough examples and/or exploration of the agent itself. A change in the environment due to malfunctions can cause the agent to adapt to a more optimised policy quicker than the standard approach. Ultimately, when considering the relative large state and action spaces of the OFC, it is expected that state-of-the-art RL algorithms will suffer from poor sample efficiency.

As shown in Table 4.1, the control problem present in the QFB acting on one beam contains 2 sensors and 16 actuators. In RL nomenclature this results in an environment with 2 observables and 16 continuous actions. Such an environment falls within the bounds of current state-of-the-art RL algorithms. Due to the prospect of the QFB being solvable using vanilla RL algorithms, the first part of this chapter will formulate, train and examine the policies obtained for the QFB problem. The second part will focus on the application of RL in the OFC problem.

## 4.1 QFB RL

The QFB on one beam relies on a Tune Response Matrix (QRM), which is a  $16 \times 2$  matrix modelling the change in the vertical and horizontal tunes due to a change in the deflections of 16 (de)focusing tuning quadrupoles of type MQT. Therefore, a vector containing the delta quadrupole deflections,  $\Delta\vec{\delta}_Q$  is multiplied by the QRM:

$$\Delta\vec{\delta}_Q \cdot Q_{RM} = \Delta\vec{Q}$$

where  $\Delta\vec{Q}$  is the modelled change in tune due to  $\Delta\vec{\delta}_Q$  being applied to the quadrupoles.

The QFB uses two QRM per beam and each QRM is set up by default to contain 6 outputs: the horizontal and vertical tune, the horizontal and vertical chromaticity, and the real and imaginary components of the coupling coefficient. To separate the QFB from the coupling and chromaticity control, the QRM is truncated to have only two outputs; the tunes. The tune control sequence occurs at 12.5 Hz and a prede-

fined sequence of steps is performed where: a) The tune error,  $\Delta\vec{Q}$ , is obtained by subtracting the current tune estimate and the reference tune; b) A velocity form PI controller is applied by using  $\Delta\vec{Q}_t$  and  $\Delta\vec{Q}_{t-1}$ , where  $t$  denotes the time step; c) The PI output is multiplied by the pseudo-inverse matrix of QRM (Tune Pseudo-Inverse (QPI)) to obtain a set of quadrupole currents; d) The currents are multiplied by -1; e) The currents are globally scaled down to accommodate the slowest quadrupole; f) Finally the corrections are sent to the quadrupole power converters via UDP packets.

### 4.1.1 Environment setup

An OpenAI Gym environment [93] was set up to mimic the response of the LHC to a varying quadrupolar magnetic field. This environment will hereon be referred to as QFBEnv [94]. QFBEnv was used within the RL framework as illustrated in Figure 2.26. QFBEnv has two continuous states as output and uses 16 bounded continuous actions as input. Both the states and actions were normalised to the range  $[-1, 1]$ .

QFBEnv implemented two important functionalities: a) the *reset* function, which initialised an episode with random initial tune error and; b) the *step* function, which first converted the normalised actions to Amperes and then multiplied the resulting vector to the QRM to obtain the resulting tune delta due to the action. The tune delta was then added to the current tune error to obtain the new state. The new state is then normalised to the range  $[-1, 1]$ . The reward was chosen to be the negative average quadratic of the state, as shown in Equation (4.1).

$$r_{t+1} = -\frac{1}{n} \sum_i^n \left( s_{t+1}^{(i)} \right)^2 \quad (4.1)$$

Since the goal of any RL agent is to maximise the reward, a perfectly trained agent would thus have a policy which reduces the tune error to zero. It is also important to note that the gradient of quadratic reward increases, the farther the state is from the optimal point. This reflects the importance of controlling a larger tune error with respect to a smaller error in the QFB, e.g. if  $\Delta Q_H \gg \Delta Q_V > \mathbf{Goal}$ , the agent would

put more importance on the correction of  $\Delta Q_H$ . From trial and error, this shape of reward was observed to produce more stable training.

In addition, the PI controller used by the QFB was also re-implemented as a separate method within QFBEnv. This was done to provide a reference for the performance of a trained agent. The PI controller was implemented with the global current rate limiting as done in the QFB. This was done in order to better compare the change in performance due to trained RL agents.

The proportional,  $K_p$ , and integral,  $K_i$ , gains were set to low values by default as a conservative measure during initialisation of the QFB. To ensure a fair comparison, the PI controller was tuned using the Ziegler-Nichols method [95]: a)  $K_i$  was set to 0; b)  $K_p$  was increased until state oscillations were observed; c) the latest value of  $K_p$  was halved; d)  $K_i$  was increased until state oscillations were observed; e)  $K_i$  was halved. The final gains obtained were  $K_p = 1000$  and  $K_i = 2000$ .

A baseline episode length was obtained by running 1000 episodes using the PI controller to choose the actions. The measured average episode length was approximately 28 steps with a standard deviation of 6 steps. Due to these results, the maximum allowable episode length in QFBEnv was chosen to be 70 steps long, approximately double of the maximum baseline episode length observed.

The normalised state space represented a range of  $[-25 \text{ Hz}, 25 \text{ Hz}]$  of tune error. The goal of QFBEnv was to reduce the tune error of both planes below a threshold of 1 Hz. The normalised action space represented the fraction of the total allowable current rate in the magnets. Every step was also assumed to occur every 80 ms, which corresponds to the QFB controller frequency of 12.5 Hz. Therefore, a normalised action of 1 on a magnet with a maximum current rate of  $0.5 \text{ A s}^{-1}$  is equivalent to a current change of  $0.5 \times 0.08 = 40 \text{ mA}$ .

Current rate limiting was thus implicitly implemented by QFBEnv when clipping the actions to the range  $[-1, 1]$ . The main difference from the QFB being that now each magnet was scaled with respect to its own current rate. The global scaling scheme used by the QFB was therefore not enforced by QFBEnv when training RL agents. Instead, each magnet could be used to its full potential.

An episode is defined as starting from the first state initialisation until a terminal state is reached. The initial state is sampled from a uniform distribution where  $s_i \sim \mathcal{U}(-\infty, \infty)$ . A terminal state could be reached either by a successful early termination or after 70 steps were made without success. Successful early termination is defined as the latest 5 rewards being above a threshold. Since the threshold was equivalent to the states being equal to 1 Hz, or  $\frac{1}{25} = 0.04$  in normalised space, the threshold reward can be obtained from Equation (4.1):

$$r_{thresh} = -\frac{1}{2} (0.04^2 + 0.04^2) = -0.0016$$

This early termination criterion was chosen in order to allow for more examples of the state below the threshold. This ultimately leads to the agent learning a better policy close to the threshold boundary.

### 4.1.2 Implementation details

State-of-the-art Deep RL algorithms are difficult to implement from scratch due to their complexity. Normally, the code used to obtain the published results is provided by the authors, however, the code might not always be guaranteed to work as expected. RL algorithms started to become somewhat standardised with the introduction of OpenAI Spinning Up [96]. However, the RL implementations found in Spinning Up are designed for educational purposes. A more robust library is OpenAI Baselines, which offers tested implementations of the main algorithms, with the main two purposes of comparing their performance and improving the algorithms [97].

Stable Baselines is a fork of OpenAI Baselines, and it provides a RL library of reliable algorithm implementations with an improved performance by using HA [98]. Stable baselines was initially used to train Proximal Policy Optimisation (PPO), Twin-delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC) agents on QFBEEnv. However, a more stable implementation of SAC from [99] was used to obtain the best policy for the SAC algorithm and will hereon be called SAC-TFL.

Stable Baselines supports Tensorflow [100], which allows the use of Graphical Pro-

cessing Units (GPUs) for faster training. Furthermore, Tensorflow provides a visualisation toolkit called Tensorboard. This tool proved to be essential for logging purposes during RL training. Considering that RL algorithms have many moving parts, Tensorboard condensed training information and made it easier to compare different RL agents.

Normalised Advantage Functions with Double-Q Learning (NAF2) and Anchored Ensemble DYNA-style (AE-DYNA) were implemented using a fork of the original code used in [72]. NAF2 was upgraded to Tensorflow 2 for a more efficient training and logging, however, difficulties were found when using the original implementation of AE-DYNA. With a code upgrade provided in [99], a newer version of AE-DYNA was attempted and the results shown were obtained from this version.

### 4.1.3 Training

In summary, the five RL algorithms that were trained on QFBEnv were:

- SAC - *off-policy and model-free*
- TD3 - *off-policy and model-free*
- NAF2 - *off-policy and model-free*
- PPO - *on-policy and model-free*
- AE-DYNA - *off-policy or on-policy and model-based*

During training of the model-free agents, two callback functions were called: a) every 1000 training steps to save the network parameters of the most recent agents to disk and; b) every 100 training steps to evaluate and log the performance of the most recent agent. Therefore, the evolution of the agent throughout the training process could be analysed off-line, and the various policies trained by the RL algorithms could be visualised. SAC and TD3 required the most hyperparameter tuning to obtain a satisfactory result. NAF2 and PPO were less susceptible to hyperparameter tuning. AE-DYNA was more complex to set up correctly and also required some network adjustments in order for it to learn a successful policy on QFBEnv. The performance

of the most recent agent in b), was evaluated on a separate instance of QFBEnv. 20 episodes were played in sequence and the actions were chosen by the most recent agent. Some of the training metrics were the average episode length, average undiscounted episode return and the average success rate. These values were logged with Tensorboard and are used in the remaining part of this section to describe the training process of each algorithm.

All the agents used the same network architecture for their policies. The final architecture was chosen through a grid-search to be an Artificial Neural Network (ANN) with 2 hidden layers having 50 neurons each and using the ReLU activation function. Owing to its successful training on NAF2 and PPO, all algorithms were set up with the same policy architecture. This network architecture was used for the value function networks of the off-policy agents as well.

Training off-line with QFBEnv is easier than training on the real QFB in the LHC. At episode termination during off-line training, QFBEnv is simply reset to a zero action and a random initial state. This allows up to 100 frames per second to be played with the environment. In the real QFB, all interactions occur at 12.5 frames per second which makes it inherently slower.

In on-line training on the QFB, the worst case episode length is 70 steps and every step is taken at a rate of 12.5 Hz. Therefore the maximum time of one episode on the QFB is:

$$\frac{70 \text{ steps}}{12.5 \text{ Hz}} = 5.6 \text{ s}$$

At episode termination, the actuators need to be re-adjusted to their initial current settings. The worst case scenario is when an action is saturated and remains constant throughout the entire maximum episode length of 70 steps. Therefore, the worst case readjustment time for one episode is also 5.6 s. All time estimates for off-line training shown in this work must be doubled to obtain the worst case training time.



## NAF2

Name	Value
learning rate	0.001
$\gamma$	0.9999
batch size	100
buffer size	5000
$q_\sigma$	0.02
$q_{clip}$	0.05
$\tau$	0.001

Table 4.2: Hyperparameters used for NAF2.

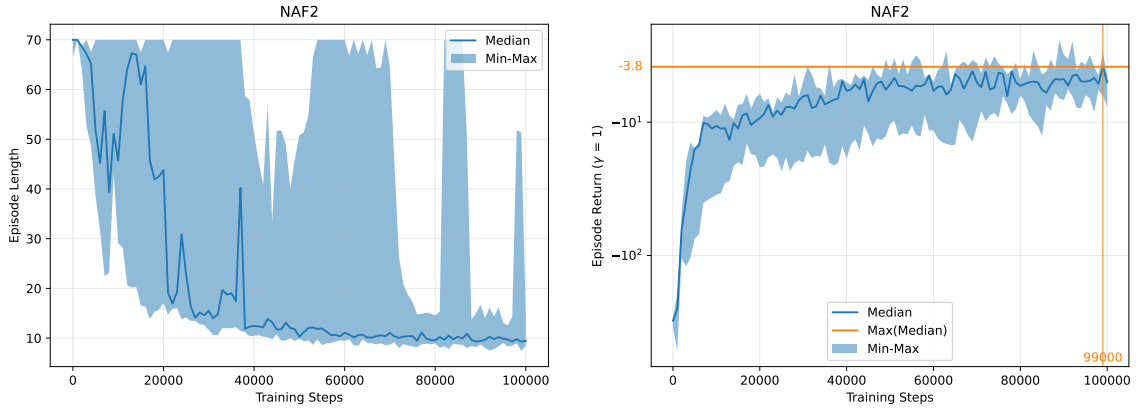
$\gamma$  is the discount factor and  $\tau$  is the time constant for the Polyak averaging done between the target and main networks.  $q_\sigma$  and  $q_{clip}$  were chosen by trial and error, and the other hyperparameters were left at their original values [72].  $q_\sigma$  set up the standard deviation of the action smoothing noise applied at each step in QFBEnv when acquiring data.  $q_{clip}$  clipped the action smoothing noise to a range  $[-q_{clip}, q_{clip}]$ .

In addition to the smoothing noise, a decaying action noise was also used during training. The following noise function was used:

$$a_i := a_i + \mathcal{N}(0, 1) \times \max\left(1 - \frac{\text{ep}_{idx}}{40}, 0\right)$$

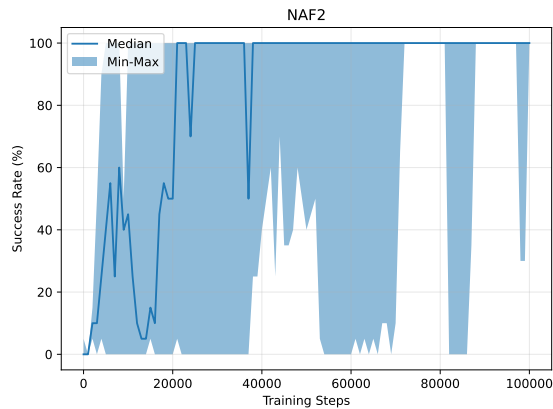
where  $a_i$  denotes the  $i^{th}$  action,  $\mathcal{N}(0, 1)$  is a standard Gaussian, and  $\text{ep}_{idx}$  is the number of the current episode. Therefore the action noise decayed to zero after 40 episodes. Note that the choice to decay the action noise relative to the number of episodes was arbitrary. The action noise can be made to decay relative to the number of steps taken so far during training.

Figure 4.1 shows the performance statistics of the NAF2 agents. In Figure 4.1a it can be seen that the episode length decreases below 20 after 20000 steps. In Figure 4.1c it can be seen that the success rate goes to 100% after 20000 steps as well. From Figure 4.1b it can be seen that the highest average undiscounted return was



(a) Median episode length

(b) Median undiscounted episode return



(c) Median success rate

Figure 4.1: Performance statistics of NAF2 during training. Five NAF2 agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.2.

$-3.4$  and occurred at the end of training at almost 100,000 steps. This shows the monotonic improvement in performance, regardless of the Min-Max bounds shown in Figure 4.1a and Figure 4.1c. The large Min-Max boundaries are explained by partially solved episodes. In these episodes, the policy manages to increase the reward until a local minimum is reached, without satisfying the successful early termination criterion. However, slight improvements in the policy push the states closer to the threshold, subsequently increasing the return.

Successful policies using NAF2 were relatively sample efficient to train when compared to other four algorithms attempted. The monotonic improvement shown in Figure 4.1b implies that a successful agent can be expected relatively early in terms

of training steps taken on the environment. Some agents during the training performed well with an average episode length of 20 steps after approximately 20,000 steps. In real LHC operation on the QFB, the worst case training time of 20,000 steps is calculated by:

$$\frac{20,000 \text{ steps}}{12.5 \text{ Hz}} \times 2 \approx 53 \text{ min}$$

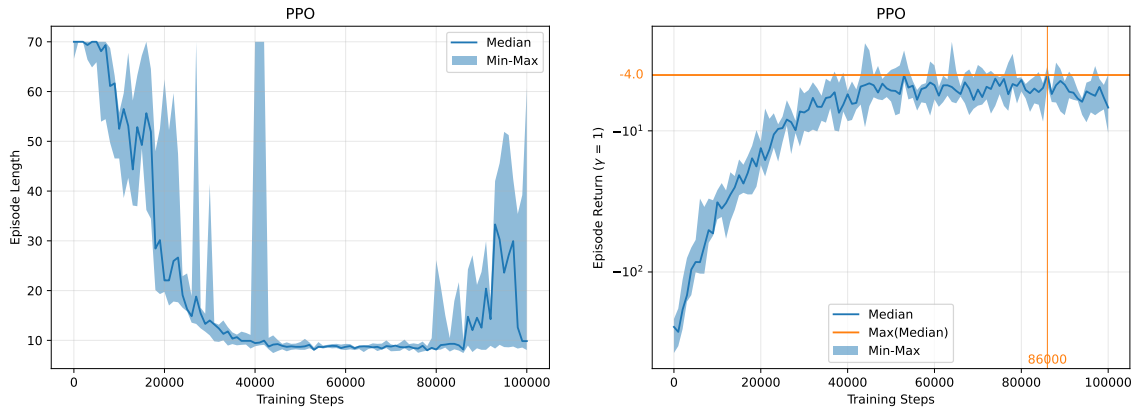
## PPO

Name	Value
learning rate	0.00025
$\gamma$	0.99
$\epsilon$	0.2

Table 4.3: Hyperparameters used for PPO.

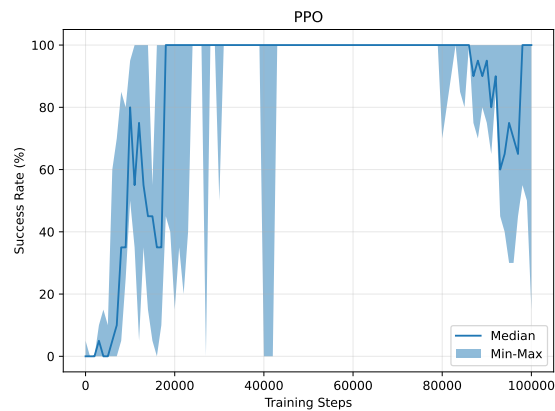
The hyperparameters shown in Table 4.3 are the default parameters of the PPO algorithm as implemented in Stable Baselines. PPO proved to be the easiest algorithm to apply to QFBEnv in terms of hyperparameter tuning and usage.

Figure 4.2 shows the performance statistics during training. As shown in Figure 4.2a, PPO converges below an episode length of 20 after around 20000 training steps. A solution which drops the episode length to below 10 steps is found after approximately 40000 steps. This remains stable until catastrophic forgetting of the policy occurs after 80000 steps. Figure 4.2c also shows that some agents expected a 100% success rate between 20000 and 80000 steps. Figure 4.2b shows that at approximately 86000 steps, the best median performance was reached, with an undiscounted episode return of -4.



(a) Median episode length

(b) Median undiscounted episode return



(c) Median success rate

Figure 4.2: Performance statistics of PPO during training. Five PPO agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.3.

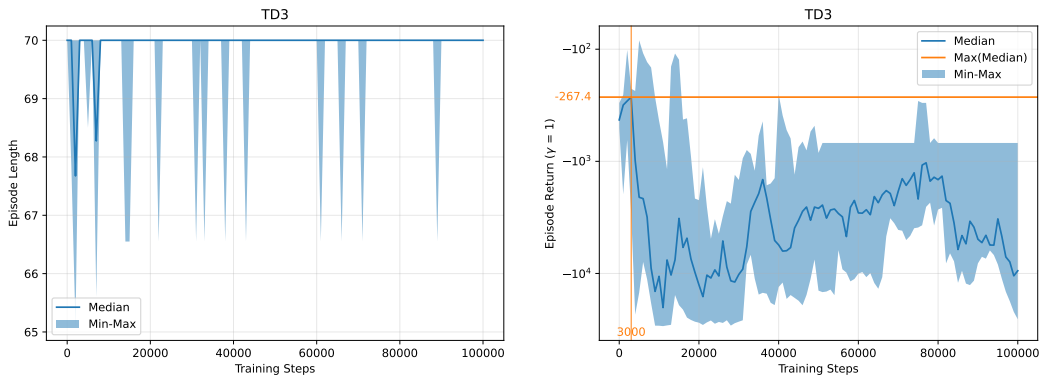
The catastrophic forgetting of the policy, however, did not occur quickly. At around 90000 steps, the expected episode length had increased to 20 steps again. Therefore, a callback function can easily halt training in the case that the policy starts forgetting, and freeze the network parameters to obtain the best performing agent. This predictability is important if the agent training occurs in real LHC operation. Similarly to NAF2, approximately 20000 steps are required to learn a good policy which is equivalent to a worst case on-line training time of 53 min.

## TD3

Name	Value
learning rate	0.0003
$\gamma$	0.99
batch size	128
buffer size	50000
$\tau$	0.005

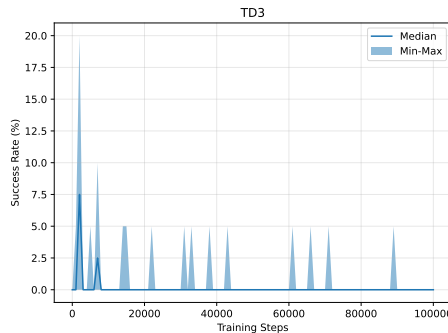
Table 4.4: Original hyperparameters used for TD3.

The hyperparameters shown in Table 4.4 are the original parameters of the TD3 algorithm as introduced in [87].  $\gamma$  is the discount factor and  $\tau$  is the time constant for the Polyak averaging done between the target and main networks.



(a) Median episode length

(b) Median undiscounted episode return



(c) Median success rate

Figure 4.3: Performance statistics of TD3 during training. Five TD3 agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.4.

Figure 4.3 shows the performance statistics during training. Figure 4.3a, Figure 4.3b and Figure 4.3c all show a failure by the algorithm to converge to a good solution in under 100000 steps. Figure 4.3b shows that the episode return decreases by two orders of magnitude after approximately 10000 steps. After 70000 steps the expected episode return increased by one order of magnitude, however, this was undone in the remaining steps where the return decreased again. This shows the training instability of TD3 when attempted on QFBEnv.

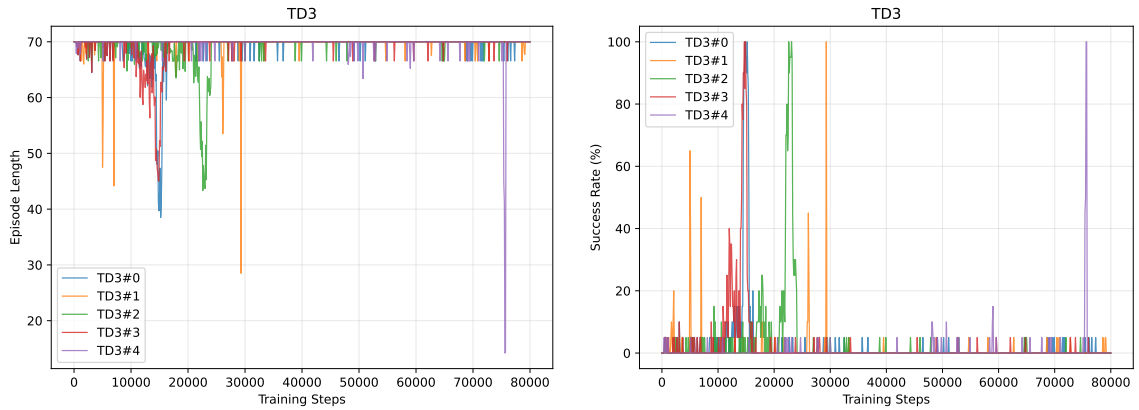
<b>Learning rate</b>	<b>Batch size</b>	<b>Buffer size</b>	$\tau$
$3 \times 10^{-4}$	32	1000	0.05
$3 \times 10^{-4}$	64	10000	0.005
	128	50000	
	512		

Table 4.5: Hyperparameter search of TD3.

As a result of the unsuccessful first attempts using TD3, a hyperparameter grid-search was performed. A permutation of the parameters shown in Table 4.5 was attempted and the agents which obtained an average success rate of 100% at least once during their training, were recorded. The different agents were sorted by the best average episode length. 5 configurations out of a total of 48 obtained a 100% success rate at least once and Table 4.6 shows the hyperparameters of the successful agents.

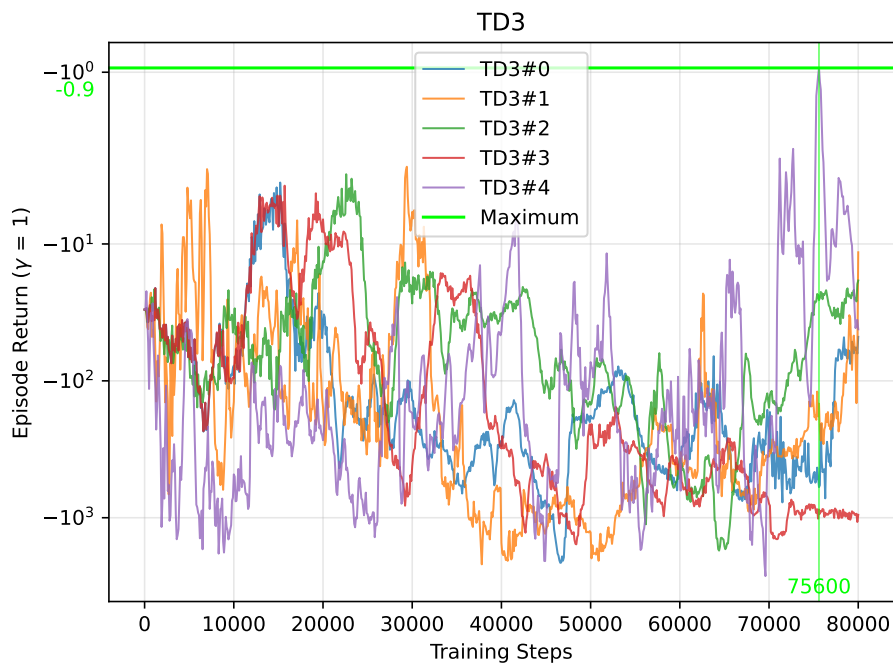
Index	Learning rate	Batch size	Buffer size	$\tau$	Best mean episode length (@ training step)
TD3#0	$3 \times 10^{-5}$	64	50000	0.005	41.3 (15k)
TD3#1	$3 \times 10^{-4}$	512	50000	0.05	45.92 (7k)
TD3#2	$3 \times 10^{-5}$	128	10000	0.05	46.4 (23k)
TD3#3	$3 \times 10^{-5}$	64	10000	0.005	49.38 (15k)
TD3#4	$3 \times 10^{-4}$	32	1000	0.005	64.96 (59k)

Table 4.6: Hyperparameters of the five successful agents and the corresponding best mean episode length. The best agent, TD3#0, obtained an average episode length of 41.3 steps after 15000 training steps.



(a) Episode length

(b) Success rate



(c) Undiscounted episode return

Figure 4.4: Performance statistics of the best TD3 agents during the hyperparameter search.

Figure 4.4 shows a similar training plot for the five best agents trained in the hyperparameter search. It can be seen that TD3#4 obtained the highest return of  $-0.9$ , however, upon inspection of the policy at steps 75000 and 76000, it was found that both had failed to converge. This suggests that the 100% successful policy obtained on step 75600 was forgotten after a few hundred steps.



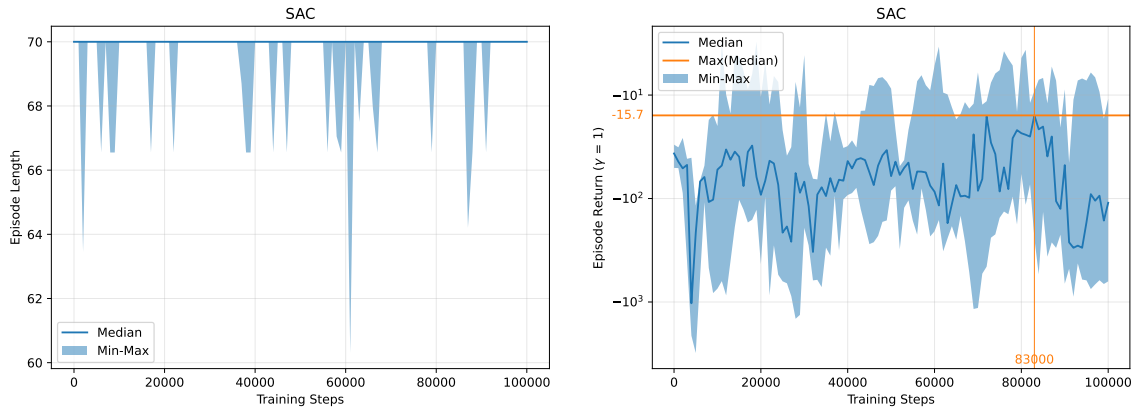
## SAC

Name	Value
learning rate	0.0003
$\gamma$	0.99
batch size	64
buffer size	50000
$\tau$	0.005

Table 4.7: Original hyperparameters used for SAC.

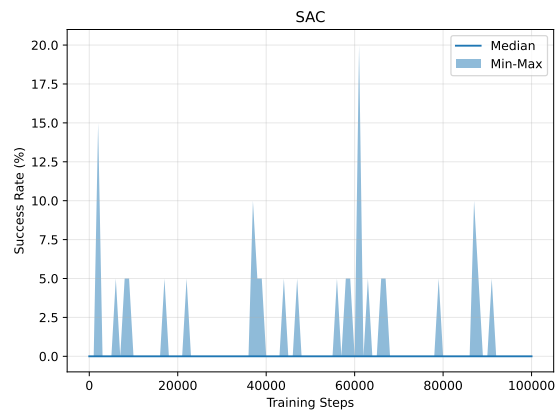
The hyperparameters shown in Table 4.7 are the original parameters of the SAC algorithm [88].  $\gamma$  is the discount factor and  $\tau$  is the time constant for the Polyak averaging done between the target and main networks.

Figure 4.5 shows the performance statistics of SAC during training when using the hyperparameters shown in Table 4.7. Figure 4.5a, Figure 4.5b and Figure 4.5c all show the failure of the SAC algorithm to converge to a good policy in under 100000 steps. On the contrary to the episode returns of TD3, Figure 4.5b shows that the all the agents obtained an episode return of the same order of magnitude as the initial value. It can also be seen that at approximately 83000 steps, the best performing agent obtained an undiscounted episode return of -15.7. This is significantly higher than the best episode return of TD3, albeit SAC still did not produce a successful policy.



(a) Median episode length

(b) Median undiscounted episode return



(c) Median success rate

Figure 4.5: Performance statistics of SAC during training. Five SAC agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.7.

A hyperparameter search of SAC was performed. A permutation of the same parameters used for TD3 in Table 4.5 was attempted to find a configuration which produced a result with a success rate of 100% at least once. The hyperparameters were also sorted by the average episode length. 11 configurations out of a total of 48 obtained a 100% success rate at least once. This shows that it is somewhat easier to train SAC rather than TD3. Table 4.8 shows the hyperparameters of the 5 agents with the lowest average episode length during training.

Index	Learning rate	Batch size	Buffer size	$\tau$	Best mean episode length (@ training step)
SAC#0	$3 \times 10^{-5}$	32	50000	0.05	48 (50k)
SAC#1	$3 \times 10^{-4}$	128	50000	0.005	19 (72k)
SAC#2	$3 \times 10^{-4}$	64	50000	0.005	21 (65k)
SAC#3	$3 \times 10^{-5}$	64	50000	0.005	21 (39k)
SAC#4	$3 \times 10^{-5}$	128	50000	0.05	17 (67k)

Table 4.8: Hyperparameters of the five best performing agents.

Compared to the best agents obtained for NAF2 and PPO, the performance of SAC is significantly worse. For this reason, another SAC implementation used in [99] was attempted on QFBEnv. Since this implementation uses TensorLayer, an open-source Deep Learning (DL) and RL library extended from TensorFlow, it will be denoted by SAC-TFL.

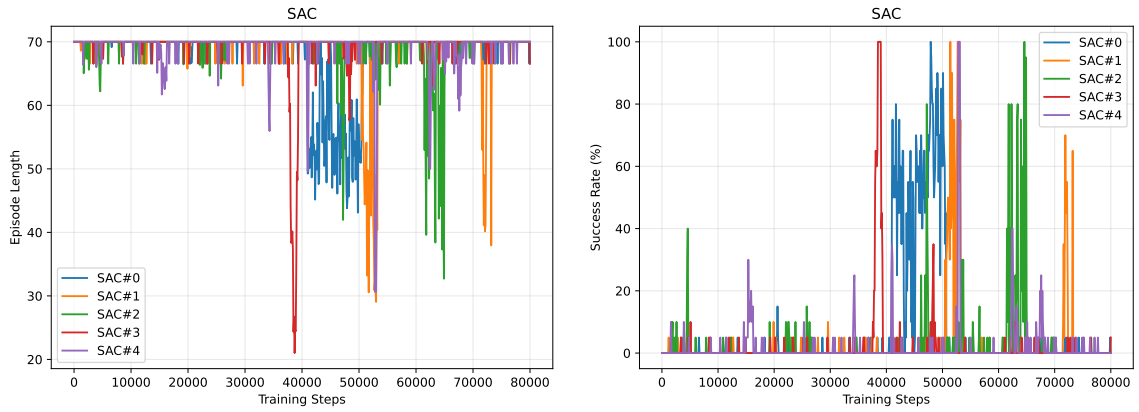
Name	Value
learning rate	0.001
$\gamma$	0.99
batch size	256
buffer size	500000
$\tau$	0.005

Table 4.9: Hyperparameters used for SAC-TFL.

The hyperparameters shown in Table 4.9 are the parameters as implemented in [99]. Figure 4.7 shows the performance statistics of five SAC-TFL agents initialised with different random seeds. Figure 4.5a, Figure 4.5b and Figure 4.5c show that some agents managed to obtain episode lengths smaller than 20 after approximately 45000 steps. A success rate of 100% was maintained for approximately 5000 steps, after which the policy maintains an episode return on the same order of magnitude. The highest episode return was -1.4 at 96000 training steps, however, at 45000 steps it was almost as high. In real LHC operation on the QFB, the worst case training time of

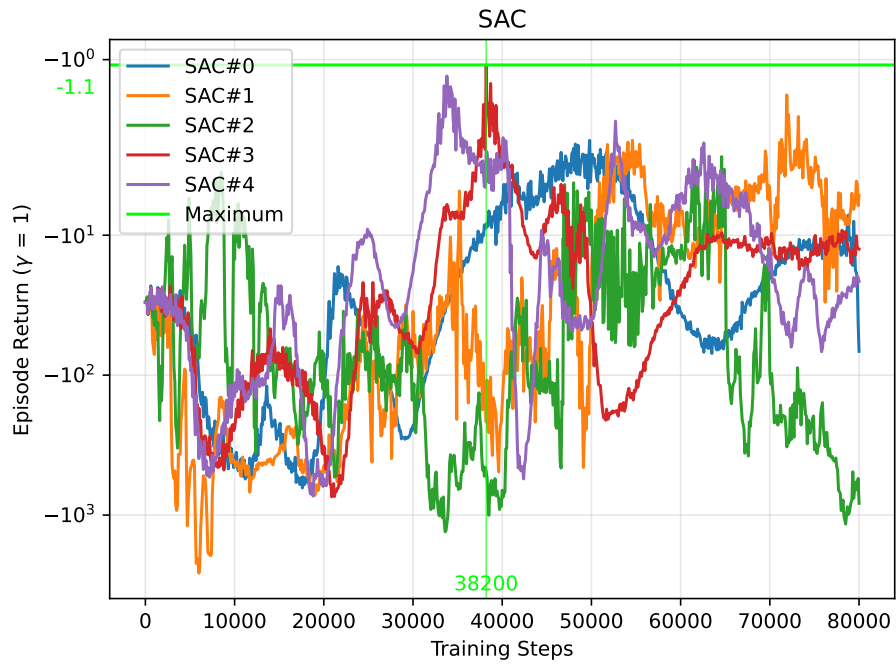
45,000 steps is calculated by:

$$\frac{45,000 \text{ steps}}{12.5 \text{ Hz}} \times 2 = 2 \text{ h}$$



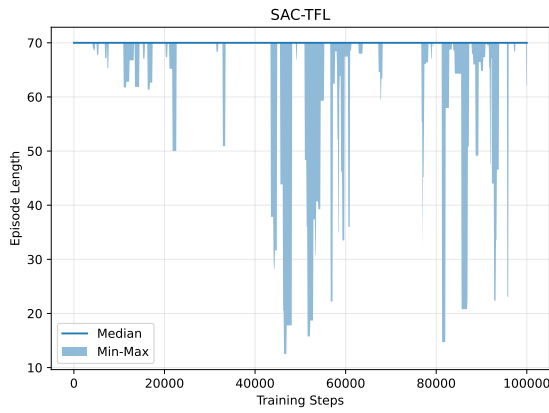
(a) Episode length

(b) Success rate

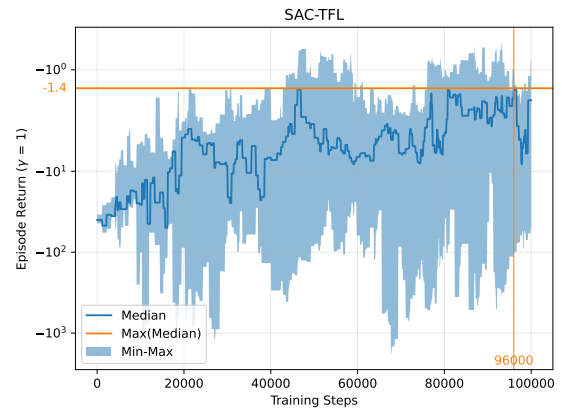


(c) Undiscounted episode return

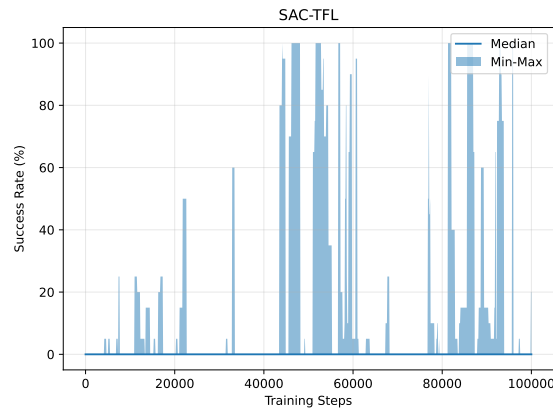
Figure 4.6: Performance statistics of the best SAC agents during the hyperparameter search.



(a) Median episode length



(b) Median undiscounted episode return



(c) Median success rate

Figure 4.7: Performance statistics of SAC-TFL during training. Five SAC-TFL agents were initialised with different random seeds and set up with the hyperparameters shown in Table 4.9

## AE-DYNA

Training AE-DYNA on QFBEnv consists of performing a sequence of three tasks iteratively. The tasks are: a) collecting a batch of data by interacting with the real environment; b) training a set of models from the collected data independently, where each model can accept a state-action pair and predict the next-state and reward; c) train a standard Model-Free (MF) RL algorithm by interacting with the trained models instead of the real environment.

The use of multiple models allow the sampling of the next-state, reward pair from different, randomly initialised networks. This has been shown to approximate the stochasticity observed in a real environment, while still capturing the dynamics of the system. In this work, SAC-TFL was trained from scratch on every epoch. An epoch is defined as the time between data collections from the real environment. The trained agent is consecutively used to obtain new batches of data from the real environment, in order to diversify the variety of trajectories in the buffer. Following this, the agent is re-initialised, and the AE-DYNA training loop repeats.

An example of training AE-DYNA-SAC on QFBEnv is shown in Figure 4.8. Batch denotes information about the data collected from the real environment. Sim and Test denote information about the interactions of the RL agent with the models and the real environment, respectively. An ideal training would consist of Sim behaving similarly to Test (training = evaluation), however, a contrast between the bottom right plot and the center right plot can be seen. Agent success is detected in Test while the models predict no success as shown in Sim. Therefore, the models cannot be used to guide training and constant testing on the real environment is required to successfully train AE-DYNA.

A 100% successful policy was obtained around 155000 training steps with an average episode length of 35 steps. At this training step, a total of 6 batches of data, equal to 4500 real steps in the environment are recorded. It is equivalent to 6 min of beam time and was the only data available for the models and subsequently for the agent to successfully train a policy. The downside is the unsuccessful early termination when using models to train the agent. If one evaluation episode is done

on the real environment every 1000 steps, 155000 steps would take approximately:  
 $\left(\frac{155000}{1000} * 70\right) * \frac{1}{12.5 \text{ Hz}} \approx 14.5 \text{ min}$ . The total worst case training time for AE-DYNA while using these evaluation parameters becomes:  $(\text{Batch time} + \text{Test time}) \times 2 \approx 41 \text{ min}$

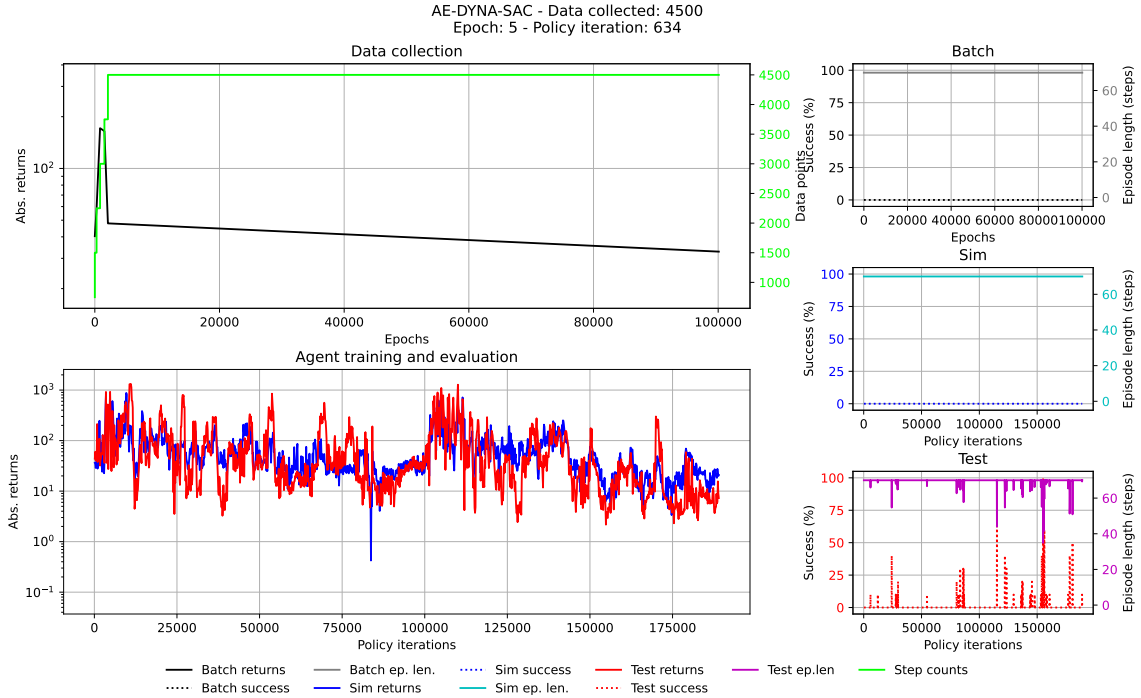


Figure 4.8: Snapshot of the training performance of AE-DYNA-SAC. Batch denotes the information about the data collected from the real environment. Sim denotes information about the agent performance over the models. Tests denotes information about the agent performance over the real environment.

#### 4.1.4 Evaluation

This section looks at the performance of the best policies obtained by each respective RL algorithm. These evaluations were performed by loading the agent with the best performance and playing multiple episodes in QFBEnv. For each episode trajectory obtained, a reference episode is created by using the actions from the PI controller. To ensure a fair comparison, the reference episode is also subjected to the test, e.g. Gaussian noise is applied to the PI action in Figure 4.10.

## Effect of Gaussian noise

QFBEnv implements a deterministic model and an action is deterministic by default. However, by adding Gaussian noise to the action chosen by the policy, stochasticity can be introduced to QFBEnv. By subjecting each agent to a stochastic environment, the general robustness of each agent can be empirically verified. During this test, the initial state per episode was ensured to be sufficiently randomised to be able to show more coverage of the state-action space.

The best RL agents trained in Section 4.1.3 were used to obtain the first set of evaluation plots which show a set of three episodes each on QFBEnv. As an example consider Figure 4.9 where the top plots (blues) correspond to the evolution of  $\Delta Q_H$  and  $\Delta Q_V$  in time, with markers denoting the start and end of each episode. A boundary (dashed green) is also drawn to indicate the threshold state. The bottom plots (reds) correspond to the evolution of the 16 actions of each respective episode until a terminal state is reached. Each set is obtained by applying Gaussian action noise with a zero mean and a standard deviation equal to 10%, 25% and 50% of half the action range  $[0,1]$ , respectively.

To aid with the analysis of the trained agents under the effect of Gaussian action noise, the action noise was varied with a finer interval and more statistics were taken on the performance of the trained agents and the PI controller e.g. Figure 4.13. For each action noise value on the x-axis, 1000 episodes were executed and used to obtain the statistics shown in the respective figures. These plots illustrate more information on the distribution of key measurements linked with the performance of the agents (in blue) and the PI controller (in red). In particular, the distributions of the following measurements are presented: a) Episode length (one scalar per episode); b) Distance of the terminal state from the optimal state of  $[\Delta Q_H = 0, \Delta Q_V = 0]$ , calculated by Equation (4.2) and is referred to as Distance To Optimal (DTO) (one scalar per episode);

$$\text{DTO} = \sqrt{\Delta Q_H^2 + \Delta Q_V^2} \quad (4.2)$$

and c) Concatenated values of the last actions applied in the episode before termina-



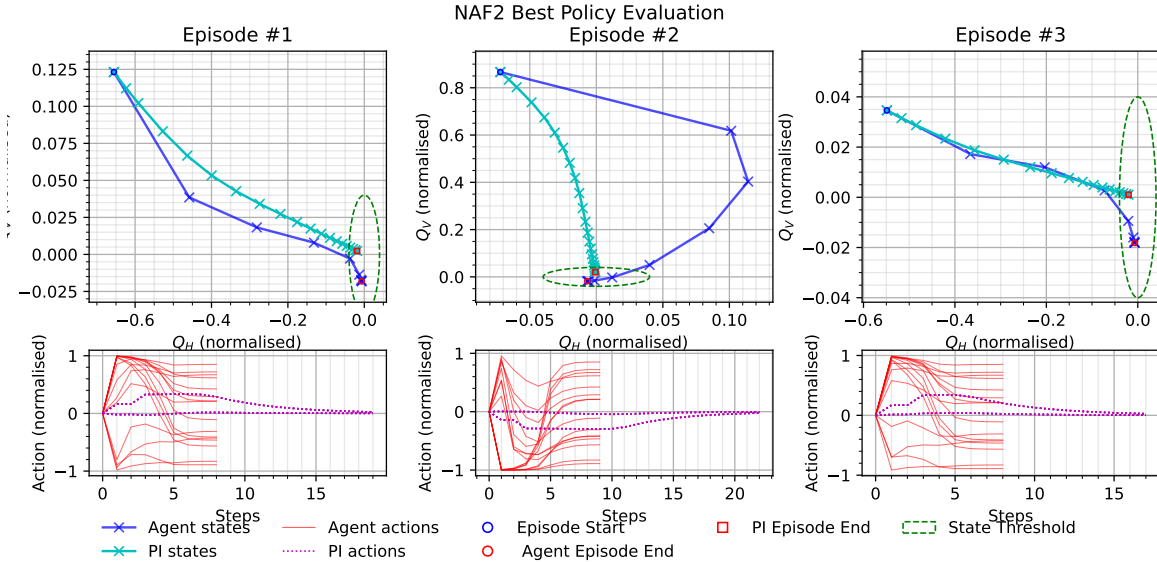


Figure 4.9: Best NAF2 agent. Action is deterministic.

tion, be it successful or otherwise (array of size 16 per episode).

Figure 4.9 shows three episodes obtained with the best NAF2 agent trained in Section 4.1.3 and using deterministic actions. It is important to note that Episode #2 of Figure 4.9 shows the agent to be only slightly worse than the PI controller. Upon comparison with the other two episodes, it can be seen that the policy trained by NAF2 has learned to converge to the optimal state.

Figures 4.10 to 4.12 show a set of three evaluation episodes each obtained from the best NAF2 policy. It can be seen that the agent managed to satisfy the early termination criterion in each scenario. The longest episode can be observed in Figure 4.12 to be approximately 20 steps long. Moreover, Episode #3 of Figure 4.12 shows that the PI controller failed to satisfy the successful early termination criterion.

Action noise	0%	10%	25%	50%
<b>NAF2</b>	$9.06 \pm 1.35$	$9.16 \pm 1.41$	$9.82 \pm 1.91$	$17.89 \pm 9.24$
<b>PI controller</b>	$20.05 \pm 3.78$	$20.41 \pm 3.99$	$24.81 \pm 7.47$	$53.32 \pm 18.37$

Table 4.10: The statistics (mean $\pm$ std.) for the episode length obtained by the best NAF2 agent and PI controller with respect to the amplitude of Gaussian action noise.

Table 4.10 tabulates the episode length statistics collected from 1000 episodes, for

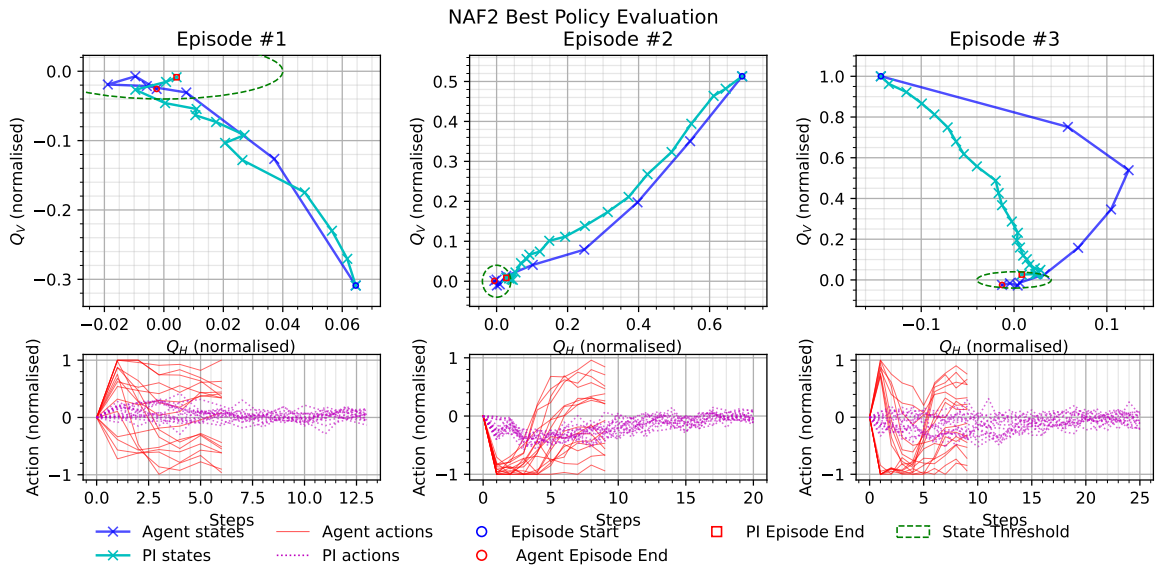


Figure 4.10: Best NAF2 agent. Action Gaussian noise with zero mean and standard deviation 10% of action range.

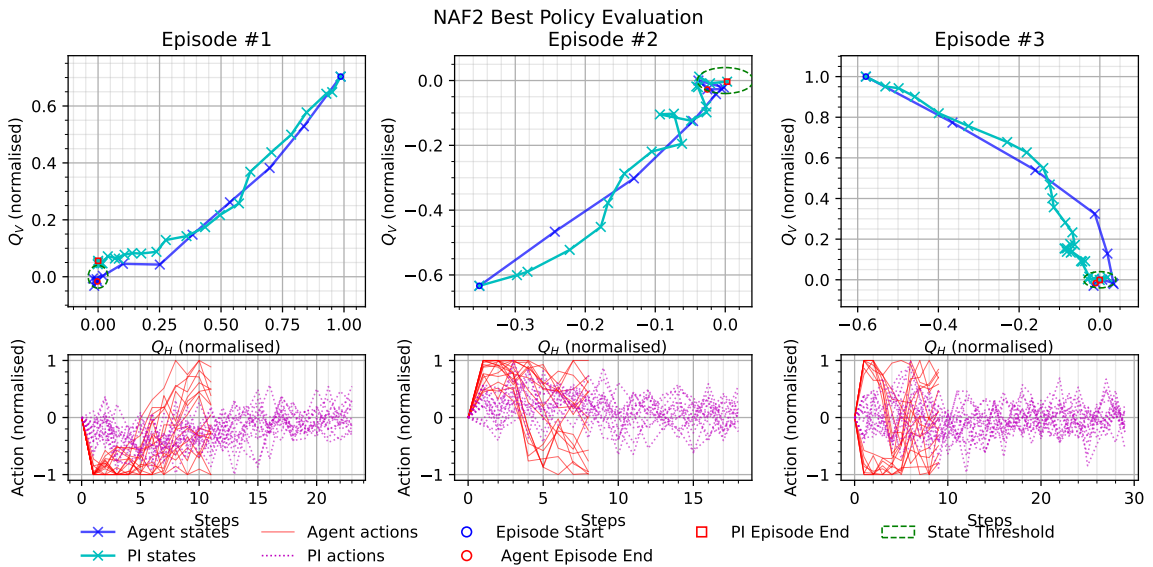


Figure 4.11: Best NAF2 agent. Action Gaussian noise with zero mean and standard deviation 25% of action range.

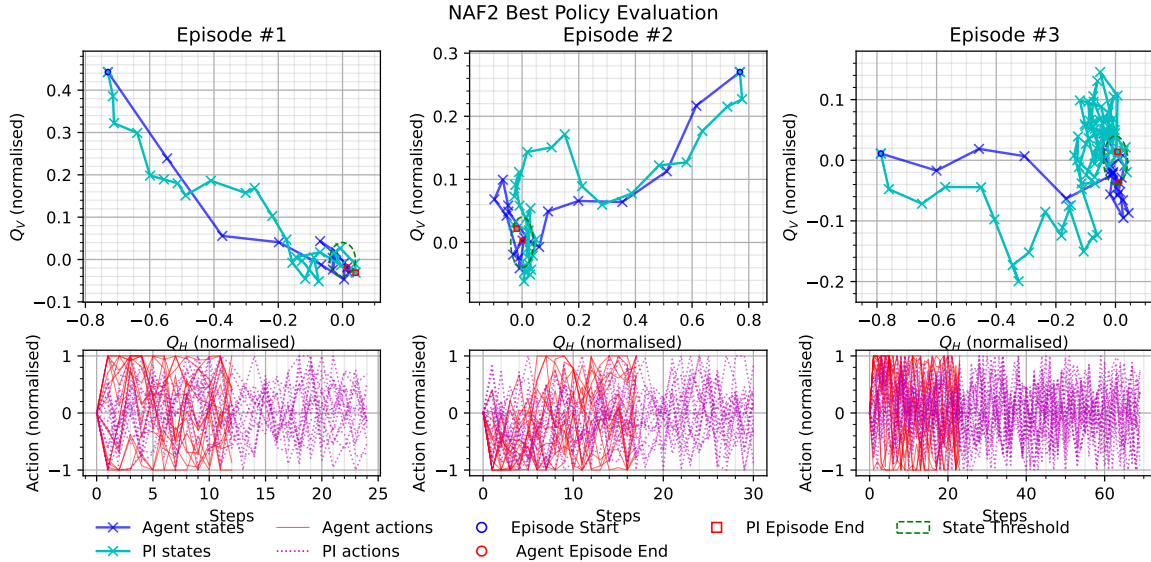


Figure 4.12: Best NAF2 agent. Action Gaussian noise with zero mean and standard deviation 50% of action range.

the values of Gaussian action noise considered in the episode evaluation plots. Both the mean and the standard deviation of the episode lengths obtained by the NAF2 agent outperformed those of the PI controller. Furthermore, Figure 4.13 illustrates that the upper bound of the PI controller episode lengths reaches 70 steps at 25% action noise; this indicates that the PI controller starts to fail to successfully terminate episodes. This corresponds with the results shown in Figure 4.14 where at approximately 25% action noise, the upper bound of the DTO of the PI controller at the end of an episode moves past the Goal threshold set in QFBEnv (green dashed line). The NAF2 agent maintains an upper bound DTO below the threshold until approximately 45% action noise.

Figure 4.15 illustrates the statistics of the last action chosen in each episode. This is where the weakness of the NAF2 algorithm is exposed, where it can be observed that the values of the last actions are unpredictable even without action noise, i.e. at 0% action noise, the PI controller actions are below  $\pm 0.05$  while the NAF2 action values populate almost all the action range. These results signify that NAF2 has trained a policy which outperforms the PI controller in a noisy environment, albeit the policy is sub-optimal.

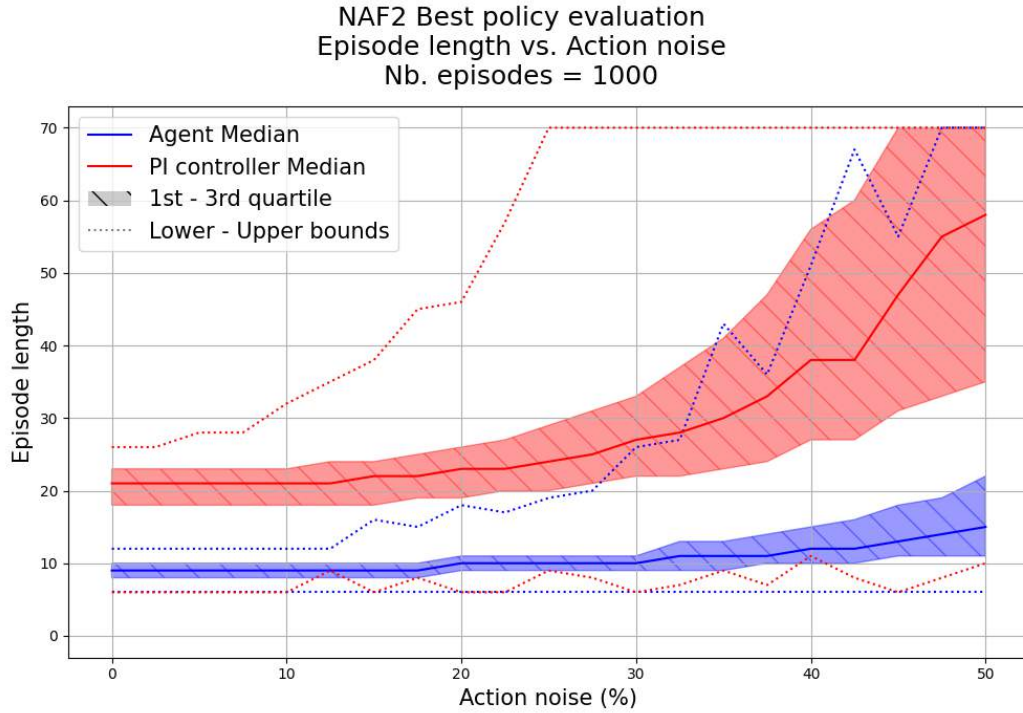


Figure 4.13: Effect of action noise on the episode length due to varying action noise on the best NAF2 agent and the PI controller.

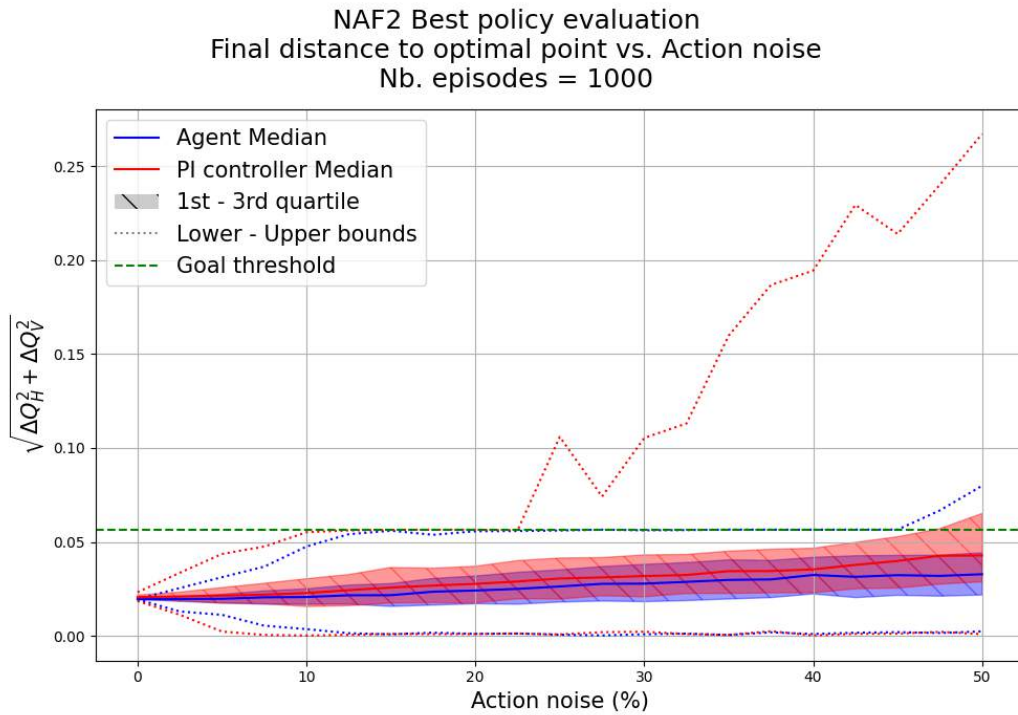


Figure 4.14: Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best NAF2 agent and the PI controller.

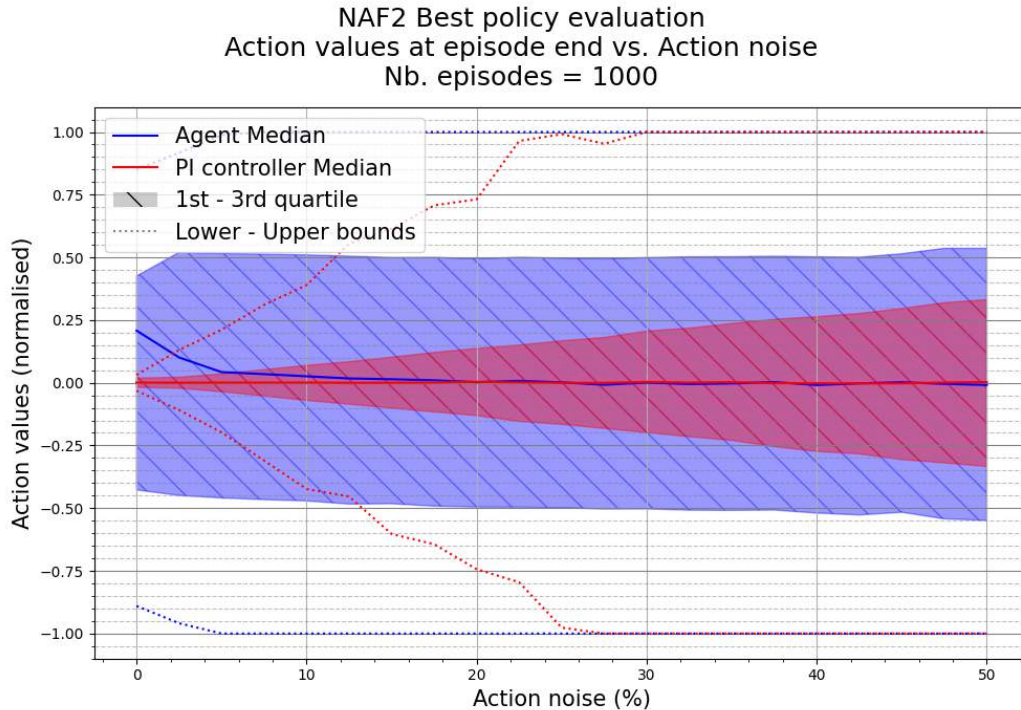


Figure 4.15: Effect of action noise on the last action used in the episode due to varying action noise on the best NAF2 agent and the PI controller.

Figure 4.16 shows three episodes obtained by the best PPO agent trained in Section 4.1.3 and using deterministic actions. Similarly to NAF2, PPO learned to optimally converge to the optimal state. Furthermore, the actions of PPO start to converge back to zero at the end of the episode. This does not occur in Figure 4.9, where the actions spread out rather than converge.

Figures 4.17 to 4.19 show a set of three evaluation episodes each obtained from the best PPO policy. It can be seen that the agent managed to satisfy the early termination criterion in each scenario. PPO also shows a wider range of episode lengths as the action noise is increased. During the 50% action noise tests in Figure 4.19, the episode length varied more than for NAF2. Therefore, the policy obtained by NAF2 is slightly more robust to action noise than PPO.

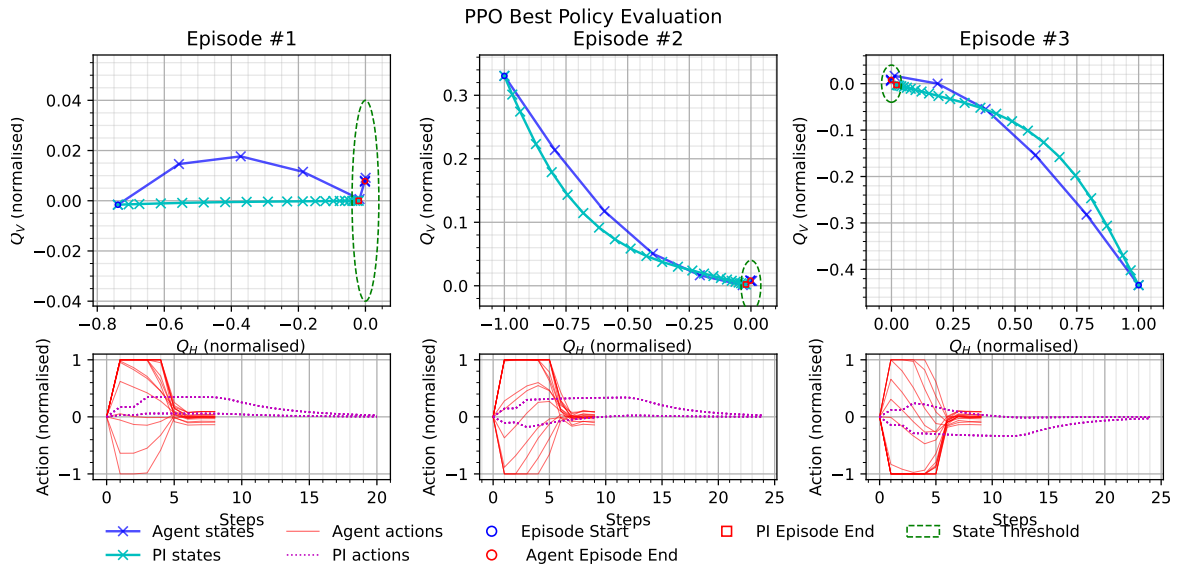


Figure 4.16: Best PPO agent. Action is deterministic.

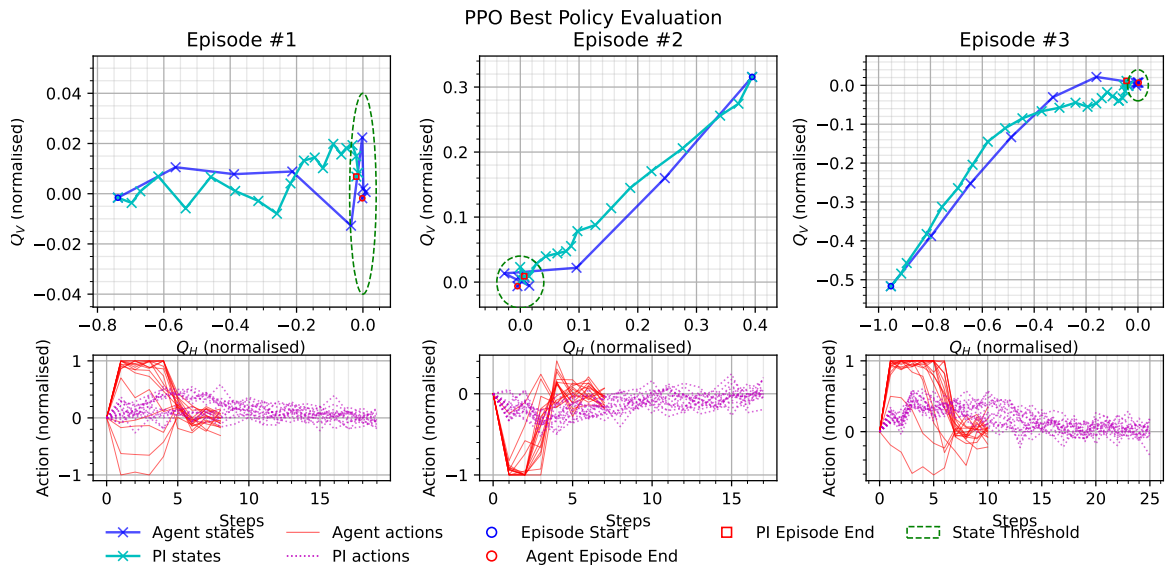


Figure 4.17: Best PPO agent. Action Gaussian noise with zero mean and standard deviation 10% of action range.



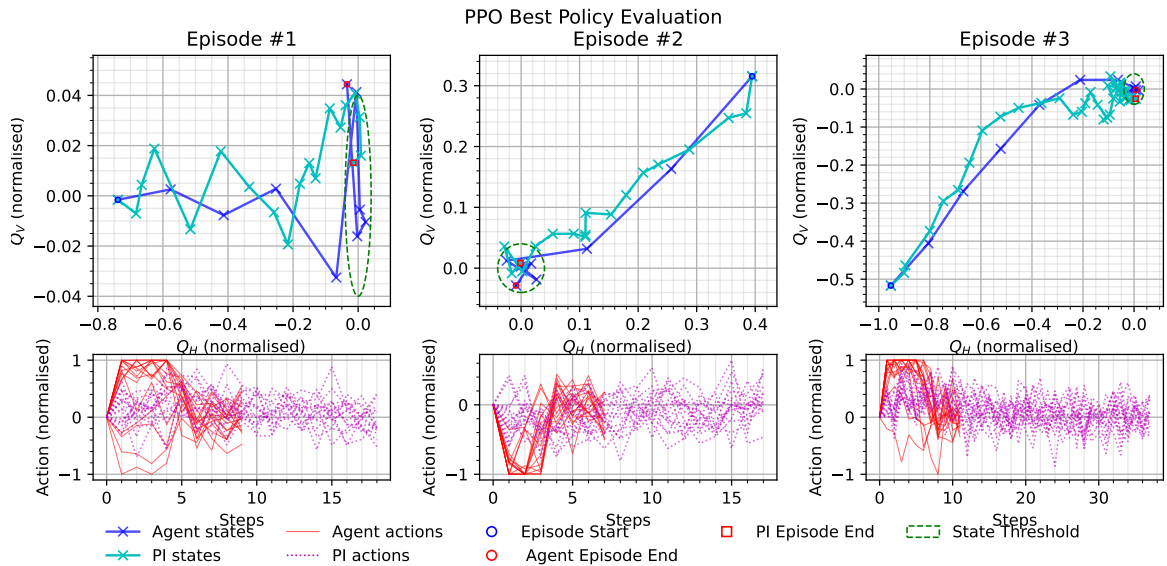


Figure 4.18: Best PPO agent. Action Gaussian noise with zero mean and standard deviation 25% of action range.

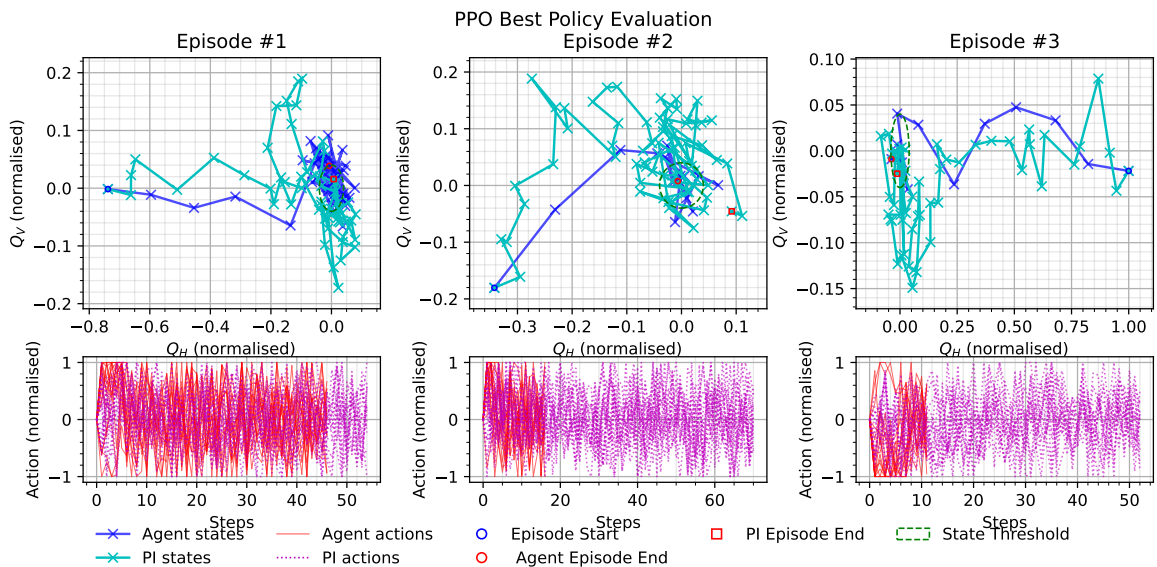


Figure 4.19: Best PPO agent. Action Gaussian noise with zero mean and standard deviation 50% of action range.

<b>Action noise</b>	<b>0%</b>	<b>10%</b>	<b>25%</b>	<b>50%</b>
<b>PPO</b>	$8.80 \pm 1.28$	$8.89 \pm 1.32$	$9.77 \pm 2.11$	$20.69 \pm 12.60$
<b>PI controller</b>	$20.09 \pm 3.70$	$20.50 \pm 3.92$	$24.91 \pm 7.66$	$53.30 \pm 18.26$

Table 4.11: The statistics (mean $\pm$ std.) for the episode length obtained by the best PPO agent and PI controller with respect to the amplitude of Gaussian action noise.

Table 4.11 tabulates the episode length statistics collected from 1000 episodes, for the values of Gaussian action noise considered in the episode evaluation plots. Both the mean and the standard deviation of the episode lengths obtained by the PPO agent outperformed those of the PI controller. Figures 4.20 and 4.21 show that the policy trained by PPO performs similarly to that of the NAF2 agent with regards to the episode length and the final distance to the optimal point.

Figure 4.22 shows the distribution of the values of the last action to be chosen at the end of an evaluation episode. It can be observed that the the PPO policy chooses actions which behave similarly to the PI controller in the presence of Gaussian action noise. At 0% action noise, distribution of action values of the agent are slightly larger than those of the PI controller. However, as the action noise is increased, the distribution widths of both PPO and the PI controller increase at the same rate. This consolidates what was observed in Figures 4.16 to 4.19, where the final values of the actions are dispersed with respect to the amplitude of the action noise applied. This concludes that PPO successfully trained a policy which is close to the optimal policy.



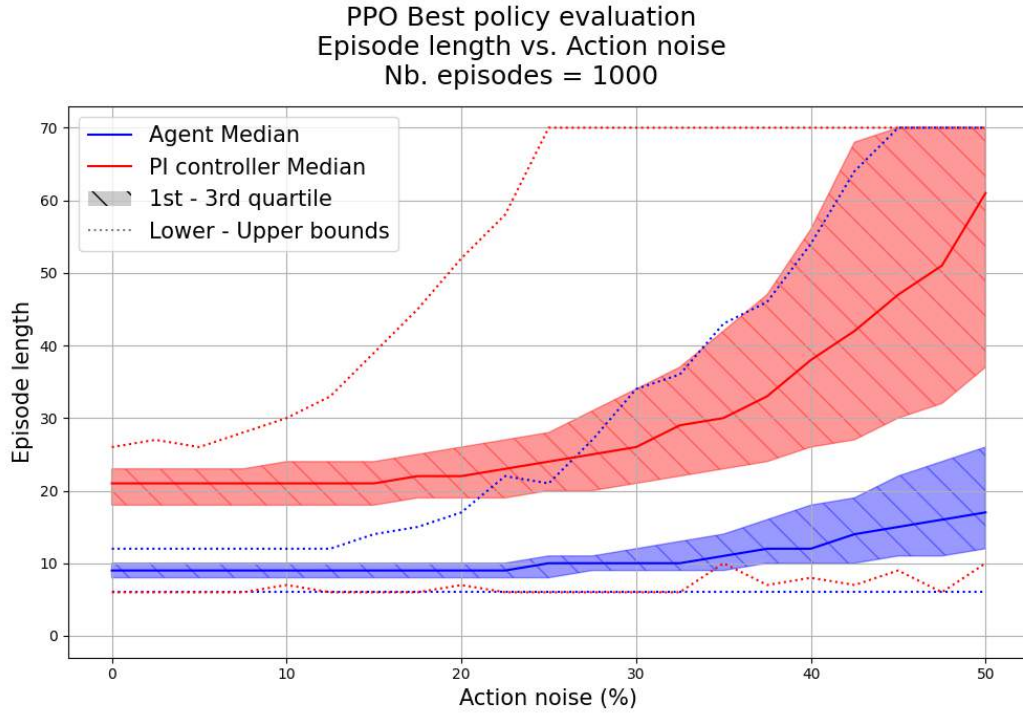


Figure 4.20: Effect of action noise on the episode length due to varying action noise on the best PPO agent and the PI controller.

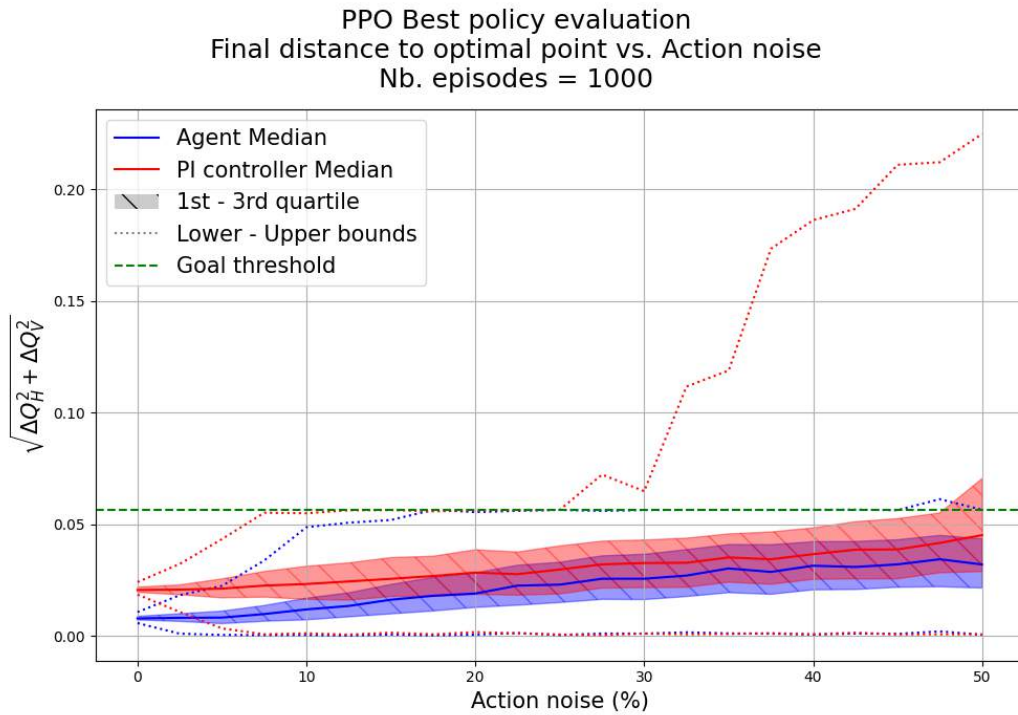


Figure 4.21: Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best PPO agent and the PI controller.

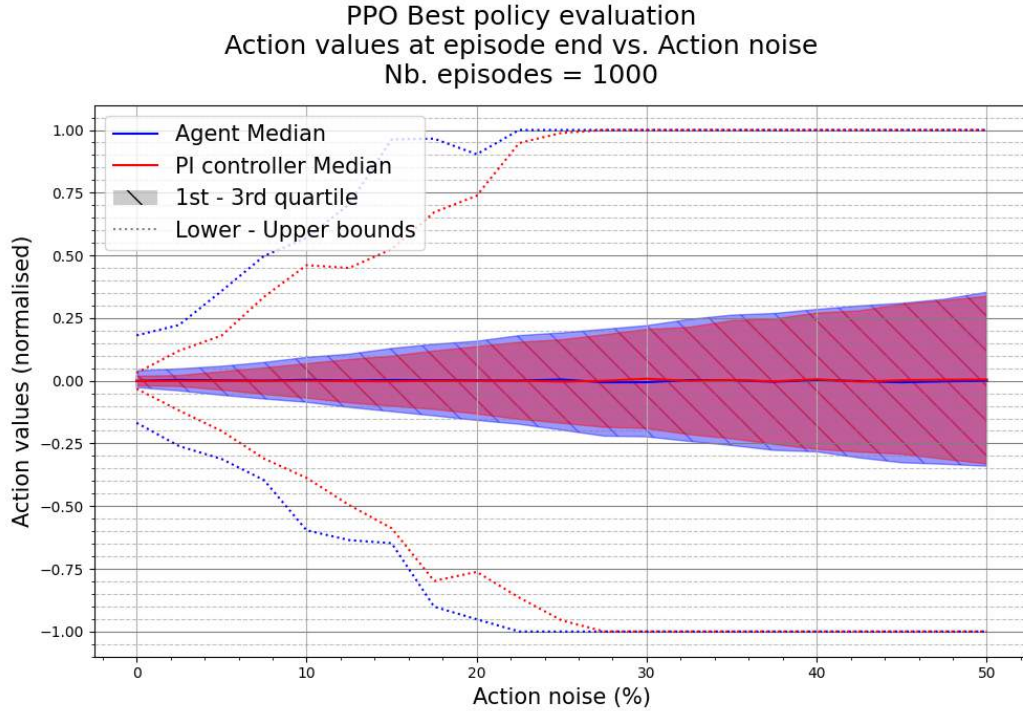


Figure 4.22: Effect of action noise on the last action used in the episode due to varying action noise on the best PPO agent and the PI controller.

Figure 4.23 shows three episodes obtained by the best TD3 agent trained in Section 4.1.3 and using deterministic actions. The top plots show that the policy does not take the most optimal path to the terminal state. In particular, the state evolution of Episode #1 shows that the agent takes smaller steps than the PI controller and as a result the episode length approximately doubles. The bottom plots show that similarly to PPO, the actions of TD3 start to converge back to zero towards the end of the episode. It can also be observed that the change in subsequent actions is gentler than both NAF2 and PPO.

Figures 4.24 to 4.26 show a set of three evaluation episodes each obtained from the best TD3 agent. It can be seen that the TD3 agent managed to satisfy the early termination criterion until 10% action noise, beyond which the episode length increases to 70 steps. The top plots of Figure 4.25 also show that the state excursions become longer for the agent, especially in Episode #3. Meanwhile, the worst episode length of the PI controller was approximately 40 steps.

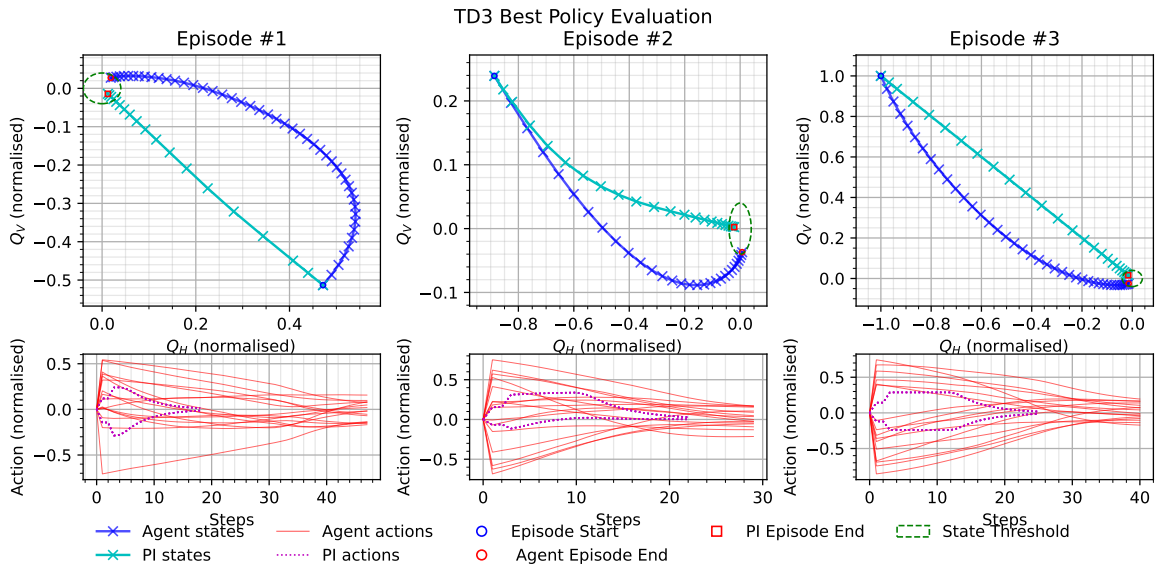


Figure 4.23: Best TD3 agent. Action is deterministic.

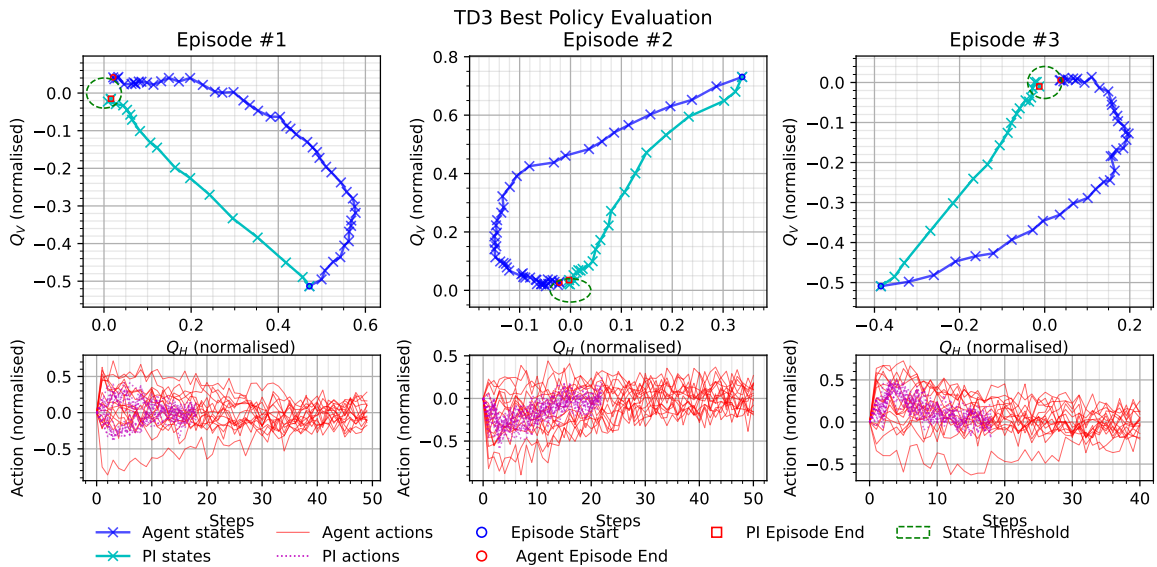


Figure 4.24: Best TD3 agent. Action Gaussian noise with zero mean and standard deviation 10% of action range.

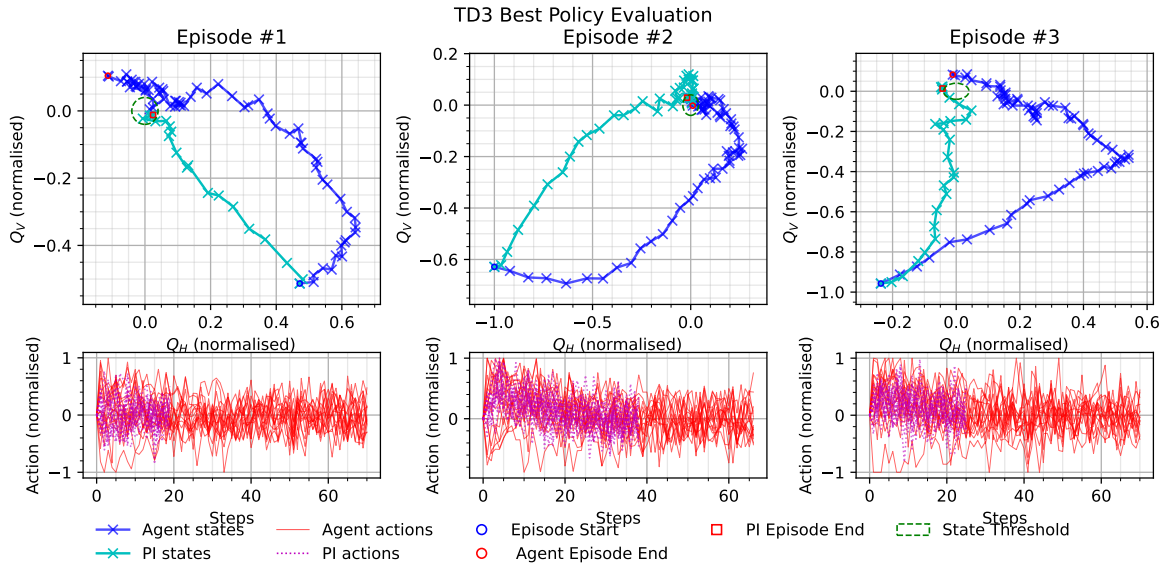


Figure 4.25: Best TD3 agent. Action Gaussian noise with zero mean and standard deviation 25% of action range.

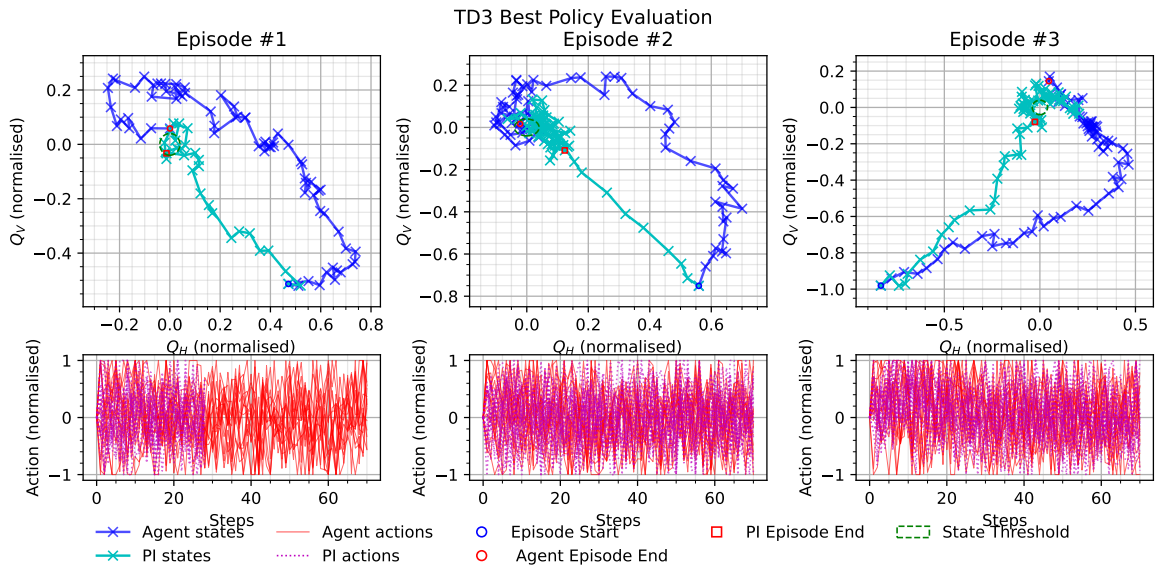


Figure 4.26: Best TD3 agent. Action Gaussian noise with zero mean and standard deviation 50% of action range.

<b>Action noise</b>	<b>0%</b>	<b>10%</b>	<b>25%</b>	<b>50%</b>
<b>TD3</b>	39.06 ± 12.08	40.34 ± 12.43	47.06 ± 15.24	63.36 ± 13.21
<b>PI controller</b>	20.32 ± 3.60	20.66 ± 3.82	25.20 ± 7.33	53.12 ± 18.92

Table 4.12: The statistics (mean±std.) for the episode length obtained by the best TD3 agent and PI controller with respect to the amplitude of Gaussian action noise.

Table 4.12 tabulates the episode length statistics collected from 1000 episodes, for the values of Gaussian action noise considered in the episode evaluation plots. The mean episode length of the best TD3 agent is approximately double that of the PI controller which corresponds to the results shown in Figure 4.29. It can also be observed that the upper bound TD3 episode length is 70 steps also at 0% action noise. These results correspond to the DTO results shown in Figure 4.28 where the upper bound exceeds the Goal threshold for all action noise amplitudes. On the other hand, Figure 4.29 shows that the distribution of the last actions chosen in the episodes increases in width proportional to the action noise applied to the episodes. This corresponds with the decaying action values observed in the deterministic episode evaluations in Figure 4.23.

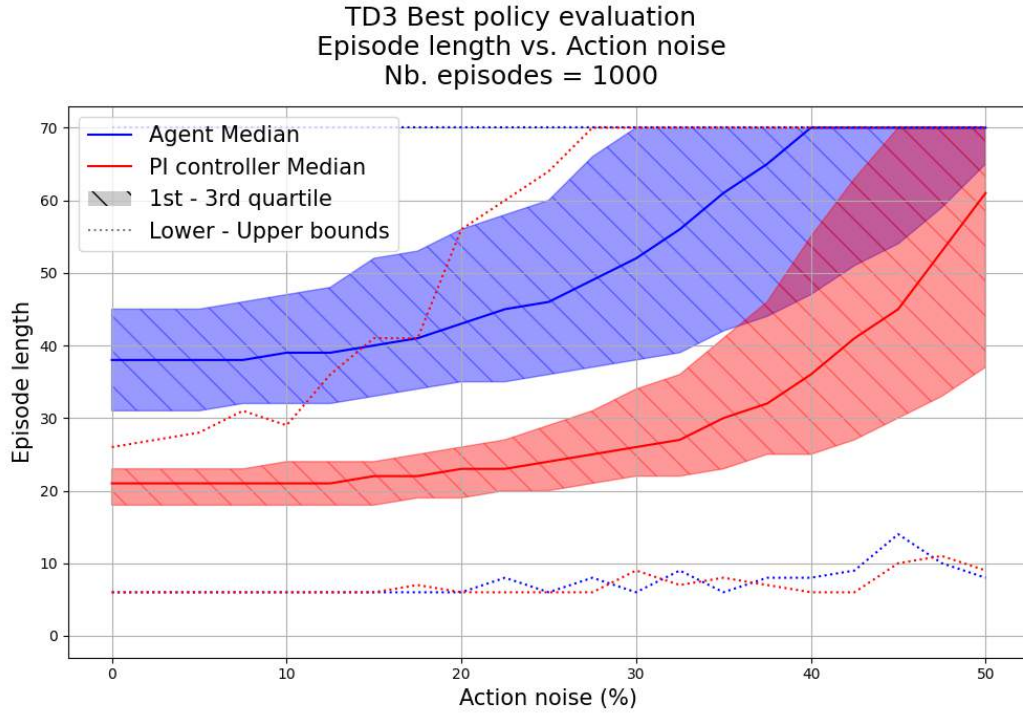


Figure 4.27: Effect of action noise on the episode length due to varying action noise on the best TD3 agent and the PI controller.

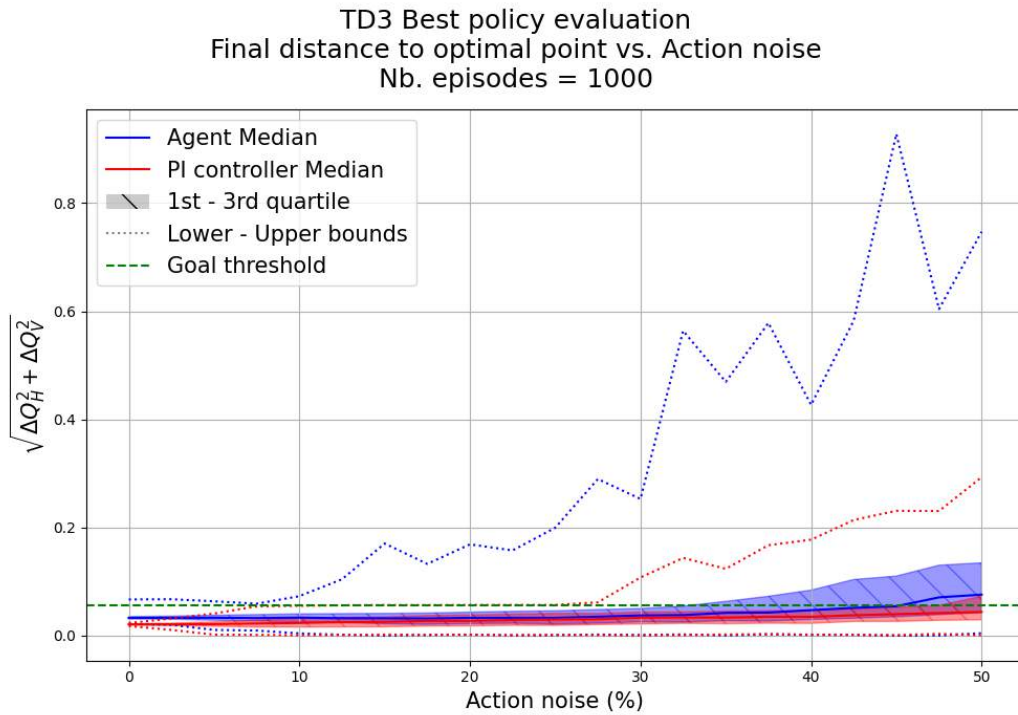


Figure 4.28: Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best TD3 agent and the PI controller.



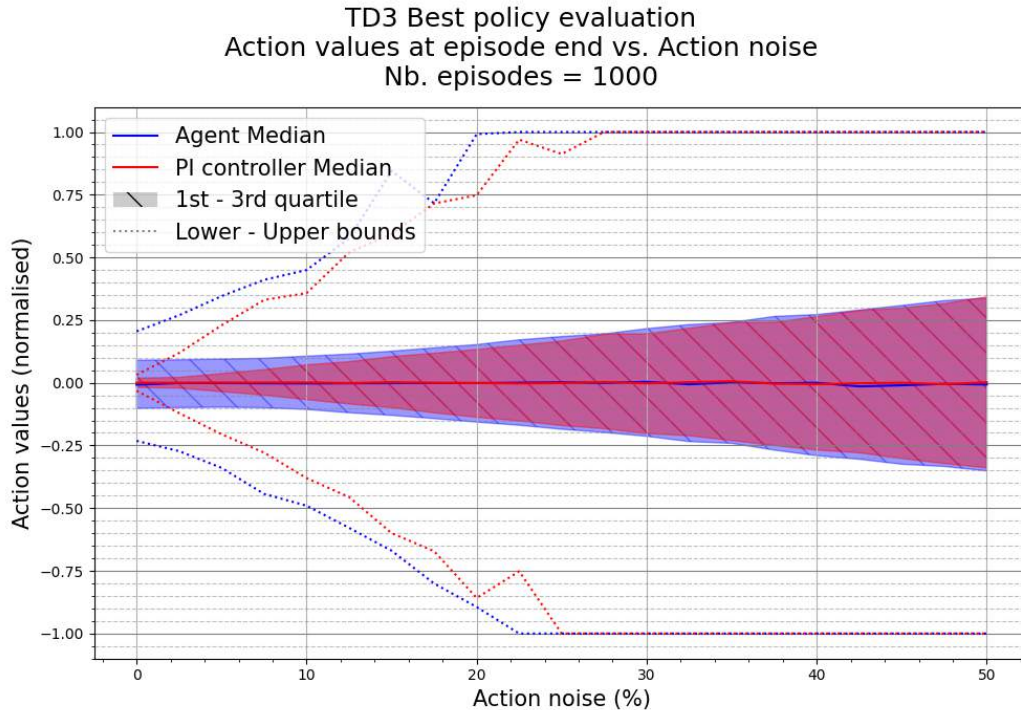


Figure 4.29: Effect of action noise on the last action used in the episode due to varying action noise on the best TD3 agent and the PI controller.

Figure 4.30 shows three episodes obtained with deterministic actions chosen by the best SAC agent trained in Section 4.1.3. Episode #3 of Figure 4.30 shows that the agent failed to converge to the terminal state and the bottom plot shows that the actions do not change after approximately 30 steps.

Similarly to the other agents, Figures 4.31 to 4.33 show a set of three evaluation episodes each. Unsurprisingly SAC started to fail the early termination criterion at 10% action noise. However, at 25% there were no failures and the number of steps required by the agent to reach the threshold boundary is on average, smaller than the PI controller. This can be attested to a sub-optimal policy which deterministically converges to a local minima. By introducing action noise, the performance of the policy seemingly improves. However, the agent is not able to maintain the state within the threshold and thus the final episode length is longer than that of the PI. At 50%, the performance of the agent does not degrade and outperformed the PI. Regardless, the best policy trained with SAC is shown to be sub-optimal when compared to the best policies of PPO and NAF2.

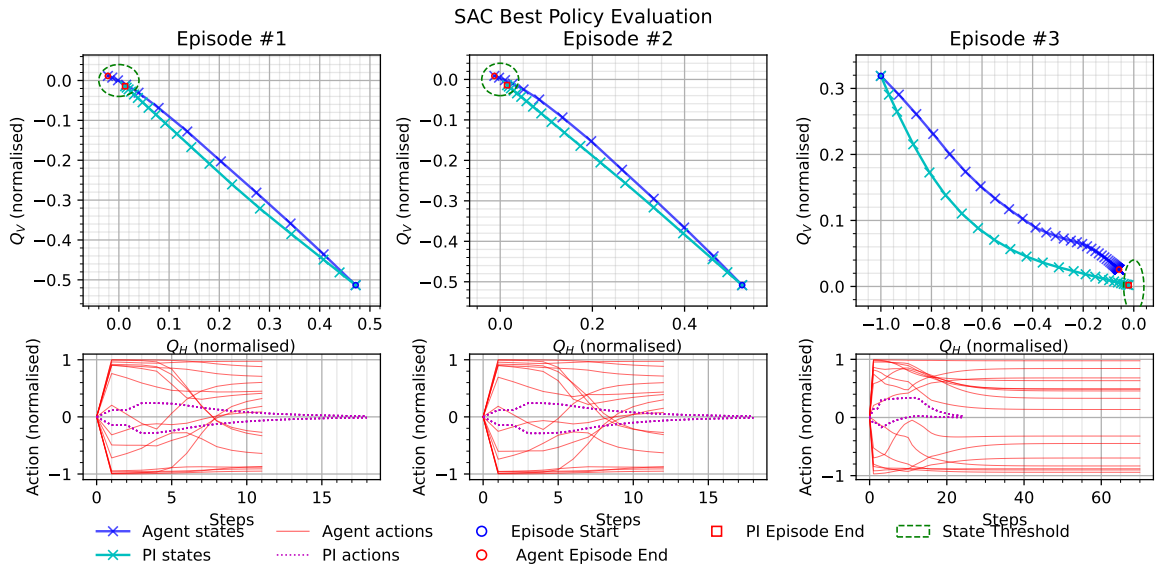


Figure 4.30: Best SAC agent. Action is deterministic.

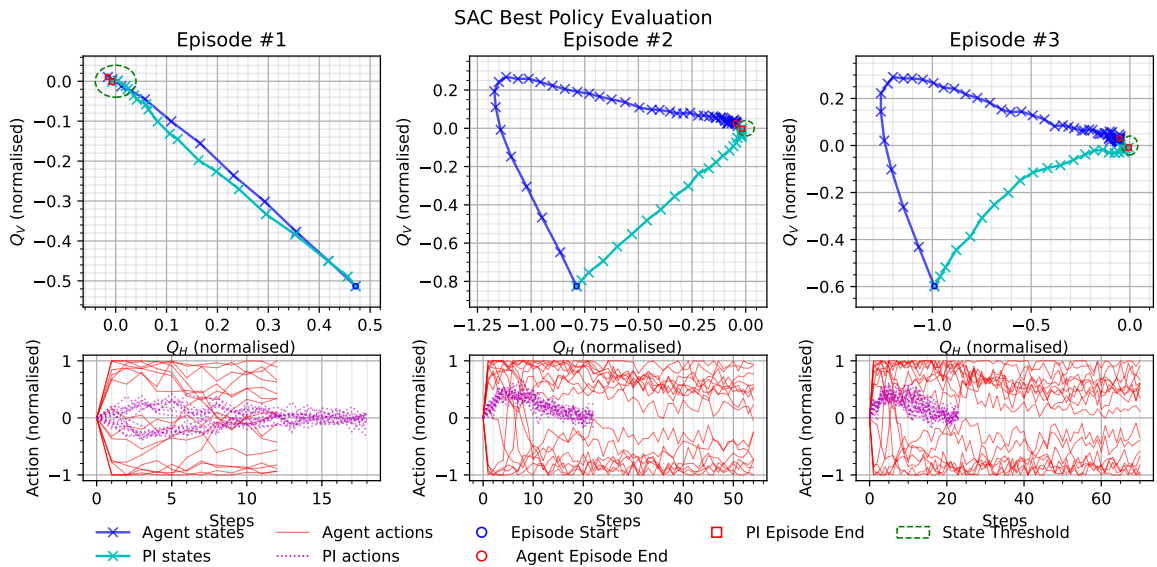


Figure 4.31: Best SAC agent. Action Gaussian noise with zero mean and standard deviation 10% of action range.



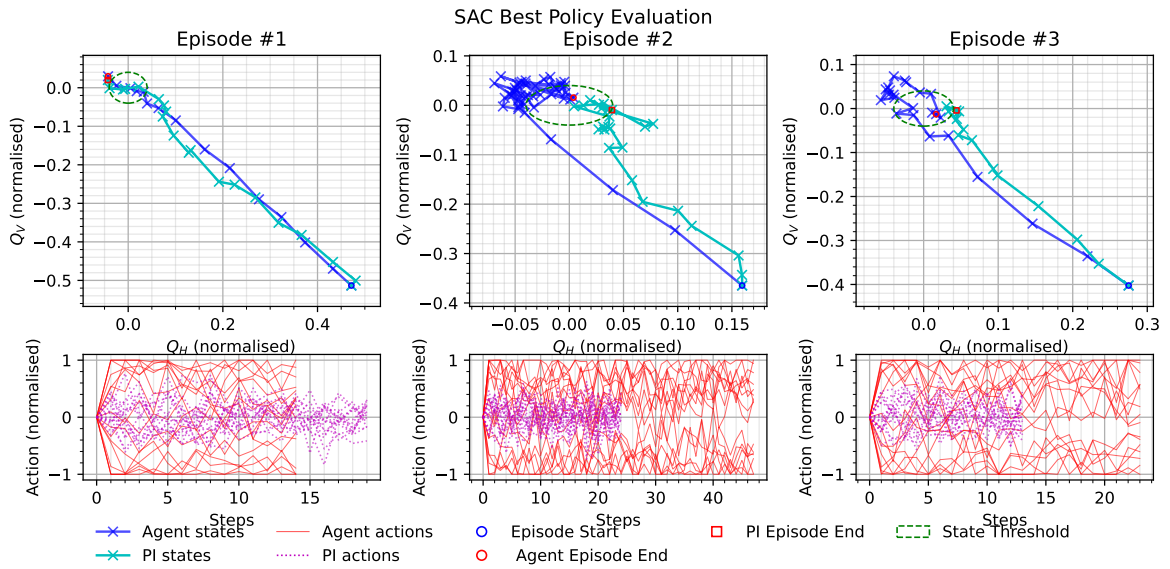


Figure 4.32: Best SAC agent. Action Gaussian noise with zero mean and standard deviation 25% of action range.

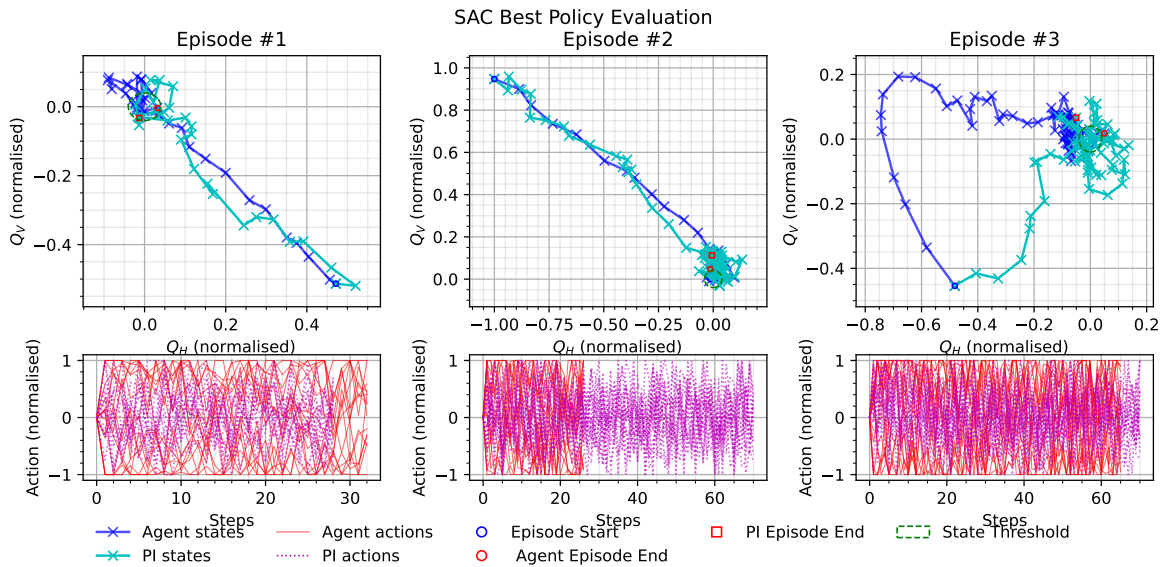


Figure 4.33: Best SAC agent. Action Gaussian noise with zero mean and standard deviation 50% of action range.

<b>Action noise</b>	<b>0%</b>	<b>10%</b>	<b>25%</b>	<b>50%</b>
<b>SAC</b>	43.09 $\pm$ 25.49	37.61 $\pm$ 21.99	33.35 $\pm$ 16.86	46.27 $\pm$ 18.41
<b>PI controller</b>	20.32 $\pm$ 3.60	20.72 $\pm$ 3.80	25.28 $\pm$ 7.90	52.56 $\pm$ 18.49

Table 4.13: The statistics (mean $\pm$ std.) for the episode length obtained by the best SAC agent and PI controller with respect to the amplitude of Gaussian action noise.

Table 4.13 tabulates the episode length statistics collected from 1000 episodes, for the values of Gaussian action noise considered in the episode evaluation plots. Figure 4.34 clearly shows the improved performance of the SAC agent as the action noise is increased, where beyond 35% action noise the median episode length of the SAC agent goes below that of the PI controller. Figure 4.35 shows that the upper bound DTO of the SAC agent never goes below the Goal threshold if QFBEnv, indicating that for all action noise values there were episodes which did not terminate successfully. Regardless, the lower bound DTO of the SAC agent is similar to the that of the PI controller for all action values, while the distribution of DTO up to the third quadrant of the SAC agent starts to match that of the PI controller after approximately 20% action noise. Figure 4.36 clearly shows the wide distribution of last actions chosen by the SAC agent for all action noise values, however, as the action noise increases the distribution width decreases or remains constant. These results show a policy which is sub-optimal, however, which behaves better under the effect of noise.



Figure 4.34: Effect of action noise on the episode length due to varying action noise on the best SAC agent and the PI controller.

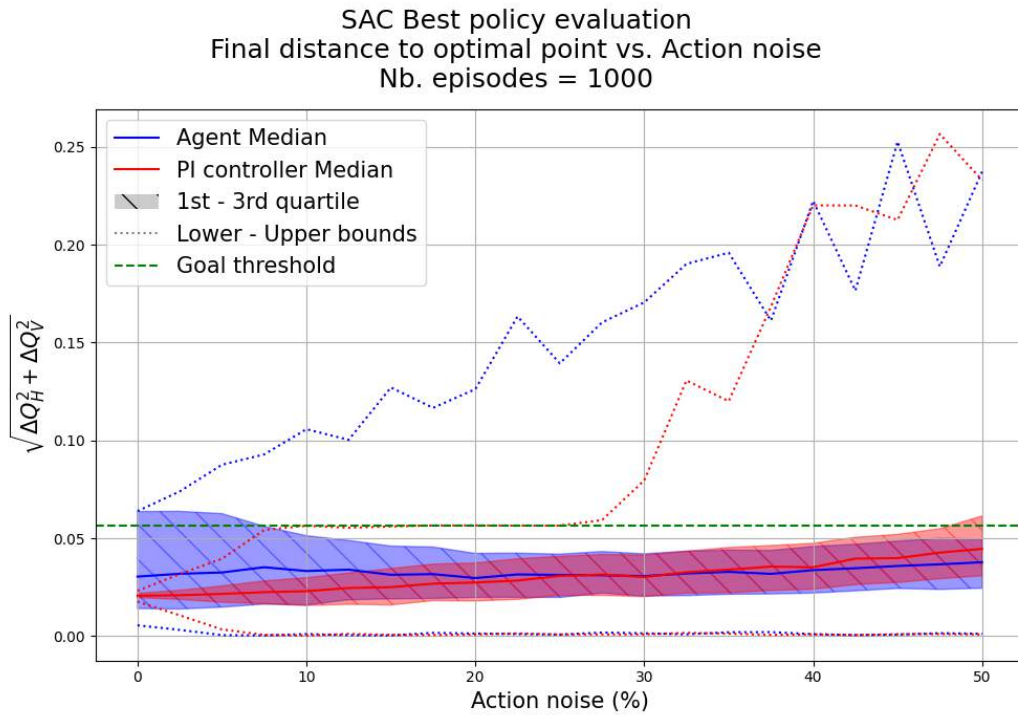


Figure 4.35: Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best SAC agent and the PI controller.

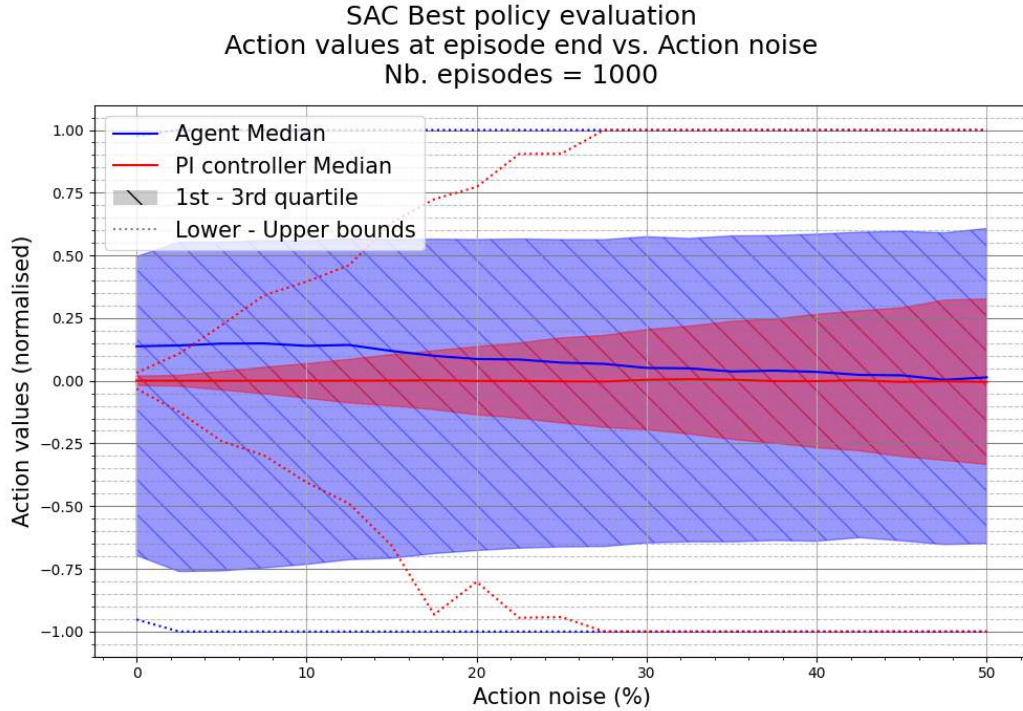


Figure 4.36: Effect of action noise on the last action used in the episode due to varying action noise on the best SAC agent and the PI controller.

Figure 4.37 shows three episodes obtained with deterministic actions chosen by the best SAC-TFL agent trained in Section 4.1.3. It can be seen that the episode length of the agent is approximately half that of the PI controller and is less than 15 steps long. However, it can be observed from the bottom plots that the deterministic actions selected by SAC-TFL are the most chaotic of all the agents discussed so far in this section.

Similarly to the other agents, Figures 4.38 to 4.40 show a set of three evaluation episodes each using different action noise. Until 25%, the agent obtains a shortest episode length than the PI controller, at approximately 20 steps. At 50% action noise, SAC-TFL fails to satisfy the early termination criterion in Episode #2, however, obtained an episode length below 10 steps in Episode #3. Considering that in each episode the agent managed to converge close to the threshold boundary in approximately 10 steps, the agent has successfully learned a policy that converges on QFBEnv.

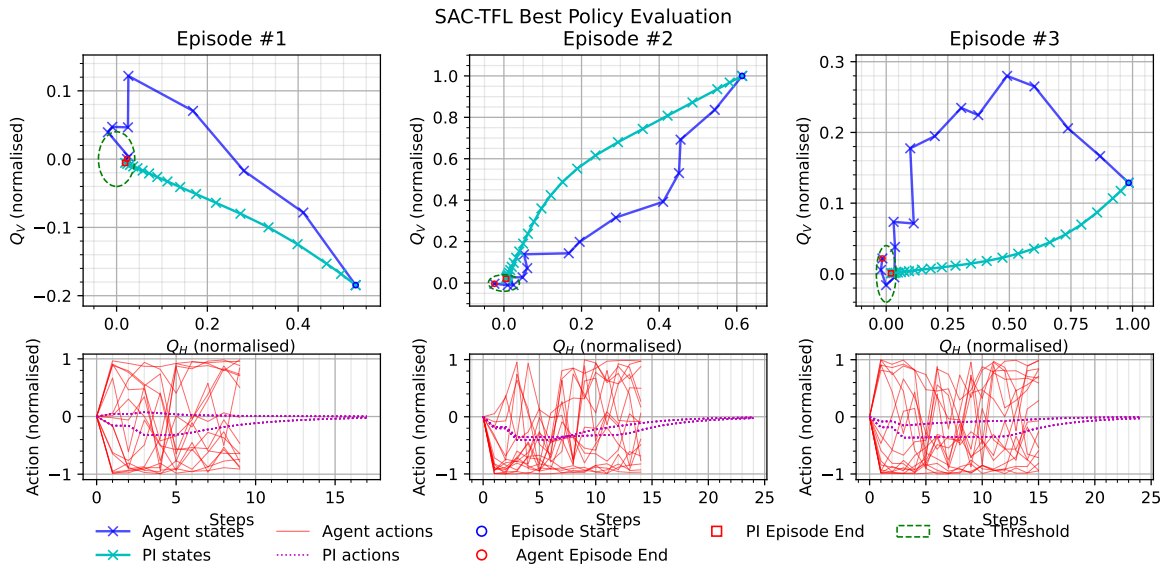


Figure 4.37: Best SAC-TFL agent. Action is deterministic.

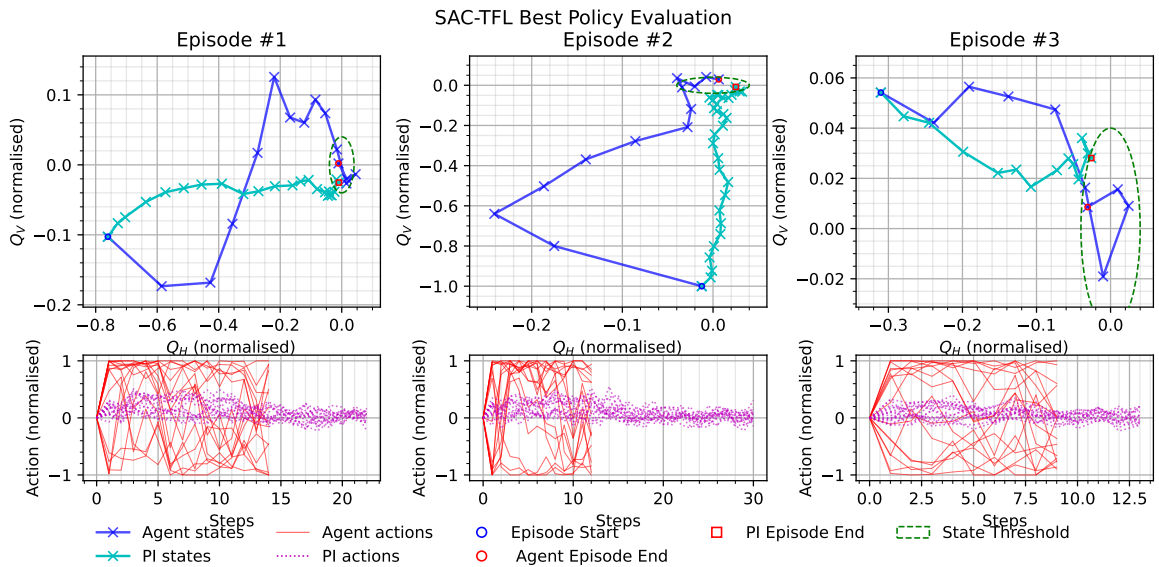


Figure 4.38: Best SAC-TFL agent. Action Gaussian noise with zero mean and standard deviation 10% of action range.



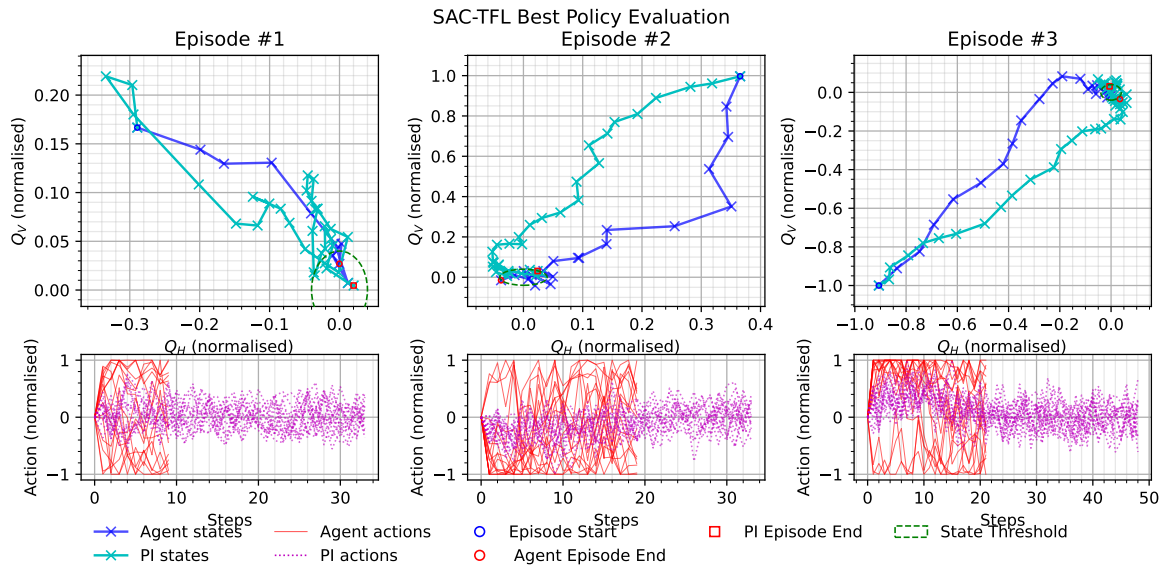


Figure 4.39: Best SAC-TFL agent. Action Gaussian noise with zero mean and standard deviation 25% of action range.

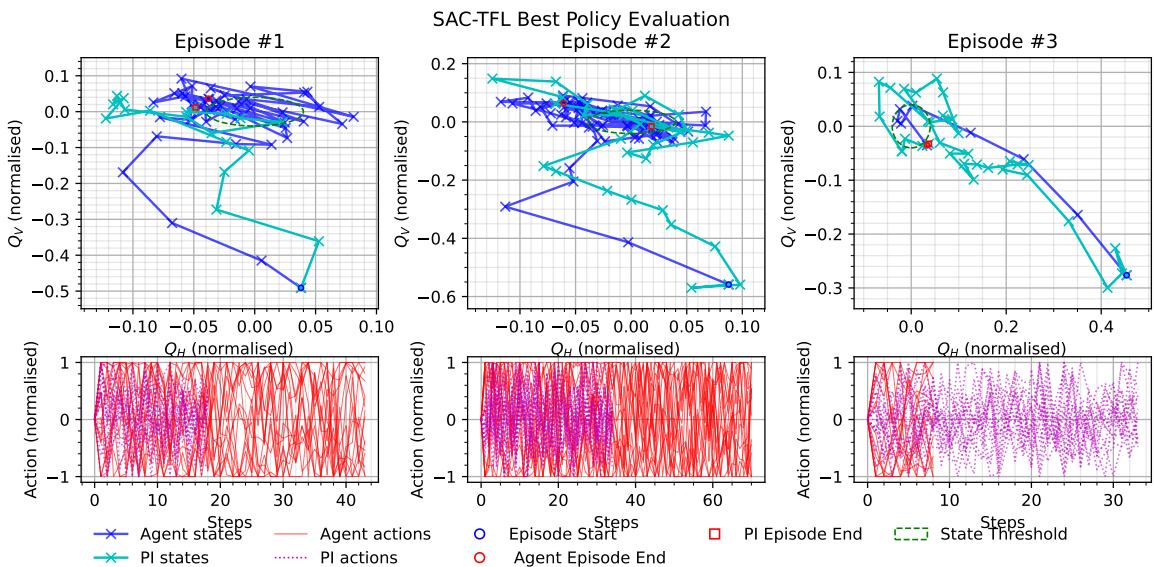


Figure 4.40: Best SAC-TFL agent. Action Gaussian noise with zero mean and standard deviation 50% of action range.

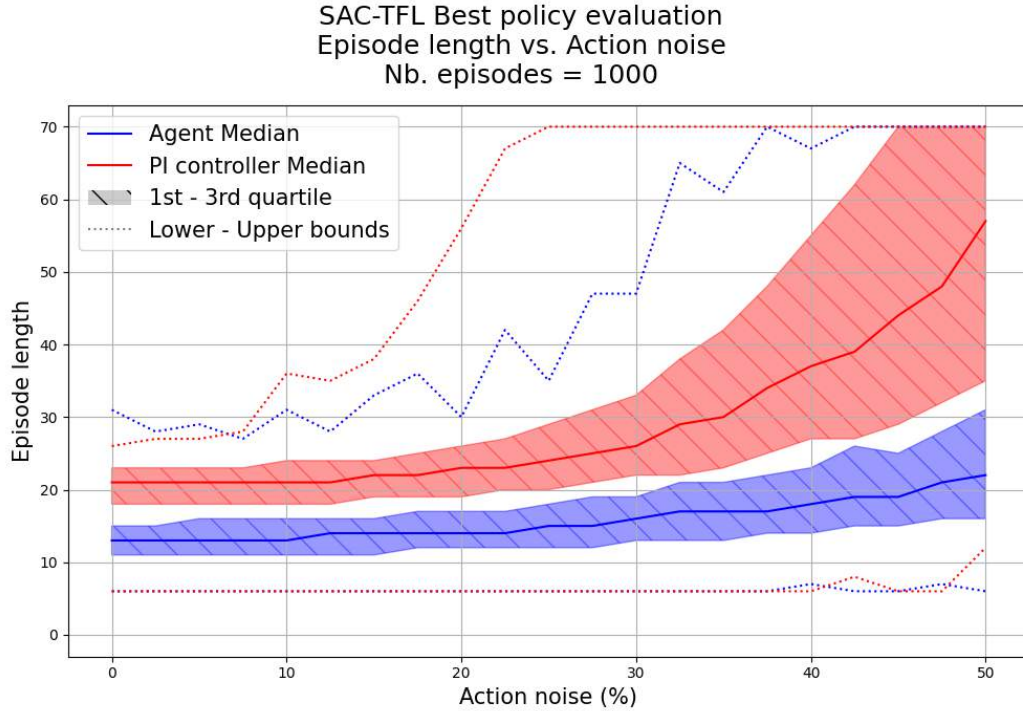


Figure 4.41: Effect of action noise on the episode length due to varying action noise on the best SAC-TFL agent and the PI controller.

Action noise	0%	10%	25%	50%
<b>SAC-TFL</b>	13.34 ± 3.19	13.64 ± 3.43	15.60 ± 4.53	24.81 ± 12.27
<b>PI controller</b>	20.29 ± 3.62	20.58 ± 3.91	25.15 ± 7.86	51.95 ± 18.56

Table 4.14: The statistics (mean±std.) for the episode length obtained by the best SAC-TFL agent and PI controller with respect to the amplitude of Gaussian action noise.

Table 4.14 tabulates the episode length statistics collected from 1000 episodes, for the values of Gaussian action noise considered in the episode evaluation plots. It can be seen that the best policy trained by SAC-TFL outperformed the PI controller. This can also be observed in Figures 4.41 and 4.42, where the performance of the best SAC-TFL agent is comparable to that of NAF2. Similarly to the best SAC agent, Figure 4.43 shows that the width of the action value distribution of the best SAC-TFL agent decreases as the action noise increases.

SAC-TFL Best policy evaluation  
 Final distance to optimal point vs. Action noise  
 Nb. episodes = 1000

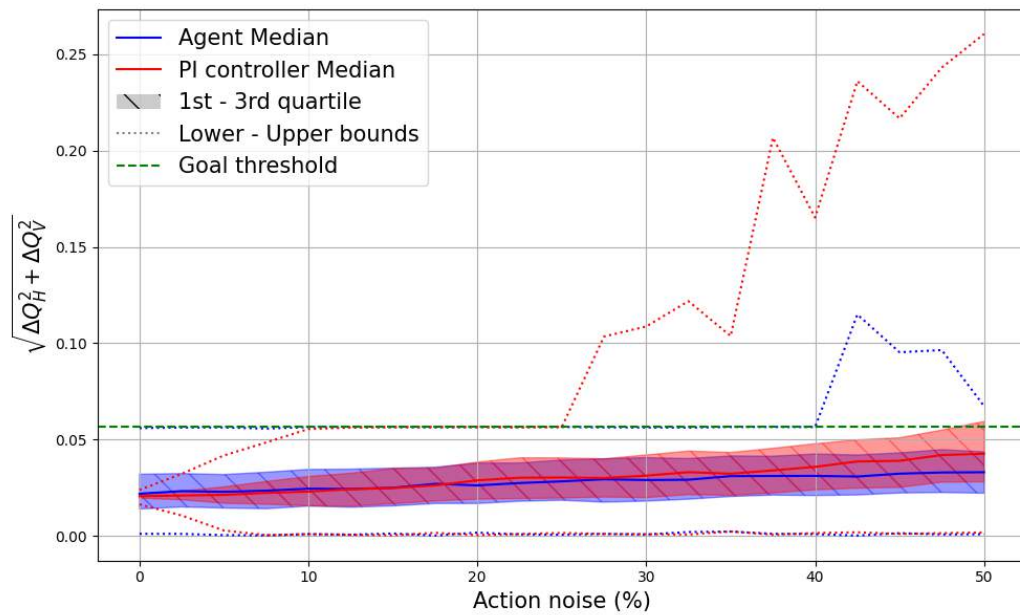


Figure 4.42: Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best SAC-TFL agent and the PI controller.



SAC-TFL Best policy evaluation  
 Action values at episode end vs. Action noise  
 Nb. episodes = 1000

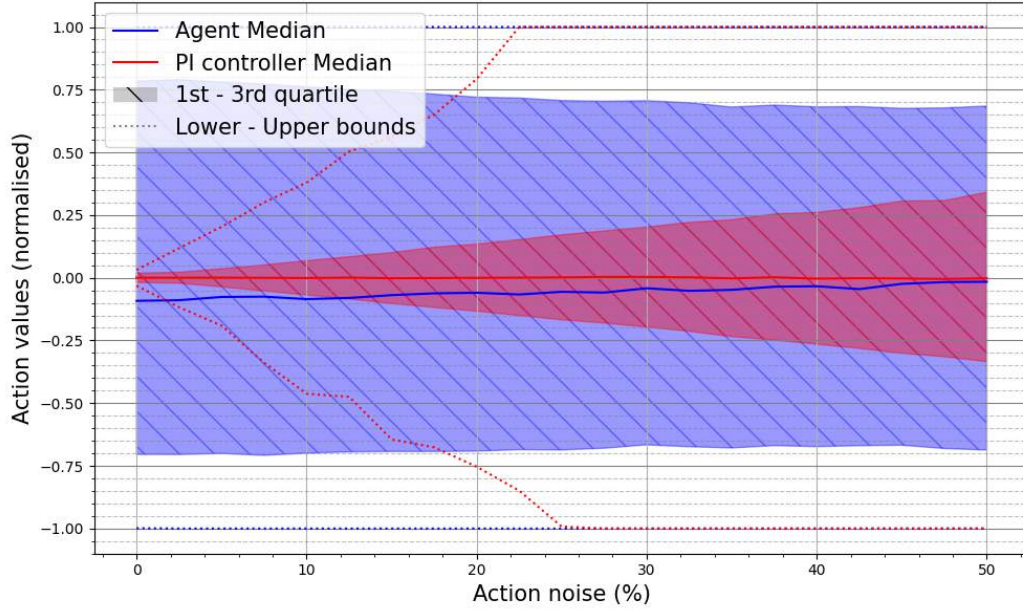


Figure 4.43: Effect of action noise on the last action used in the episode due to varying action noise on the best SAC-TFL agent and the PI controller.

Figure 4.44 shows three episodes obtained with deterministic actions chosen by the best AE-DYNA-SAC agent trained in Section 4.1.3. Similarly to the other agents, Figures 4.45 to 4.47 correspond to an action noise of 10%, 25% and 50%. In the deterministic case, the policy sometimes fails to converge under the threshold boundary through the most optimal route. At 10% action noise, the performance of the agent is almost the same as the PI controller, however, at 25% the performance degrades to around double the PI episode length. When the noise is increased to 50%, some episodes diverge from the optimal point, while the PI controller always succeeds.

Action noise	0%	10%	25%	50%
<b>AE-DYNA-SAC</b>	36.25 ± 17.50	37.22 ± 17.78	41.98 ± 18.24	54.51 ± 17.84
<b>PI controller</b>	20.20 ± 3.79	20.50 ± 4.03	25.36 ± 8.28	53.35 ± 18.41

Table 4.15: The statistics (mean±std.) for the episode length obtained by the best AE-DYNA-SAC agent and PI controller with respect to the amplitude of Gaussian action noise.

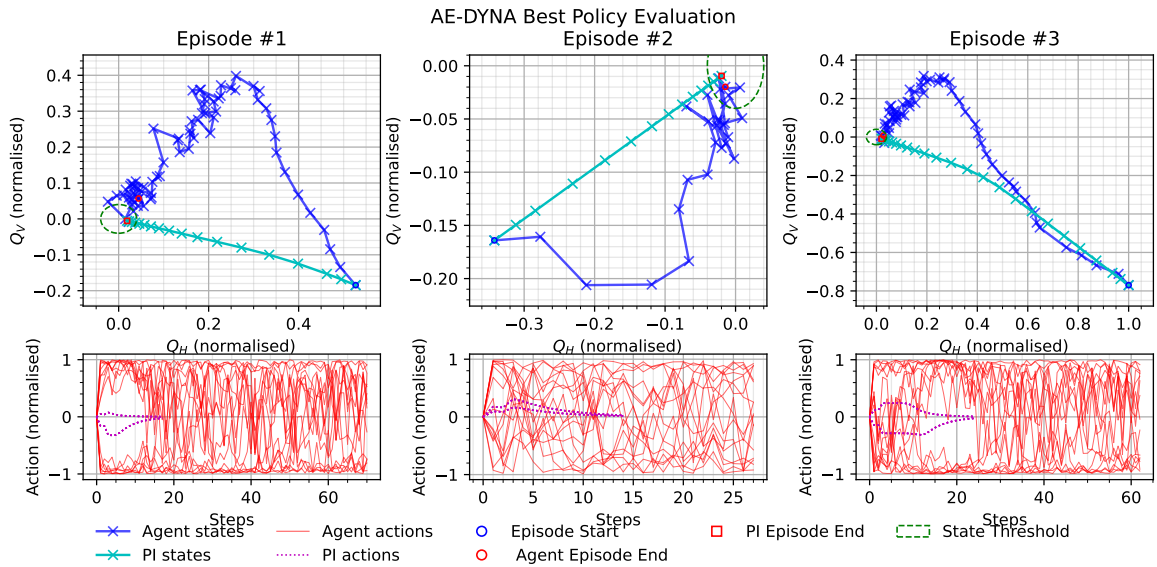


Figure 4.44: Best AE-DYNA-SAC agent. Action is deterministic.

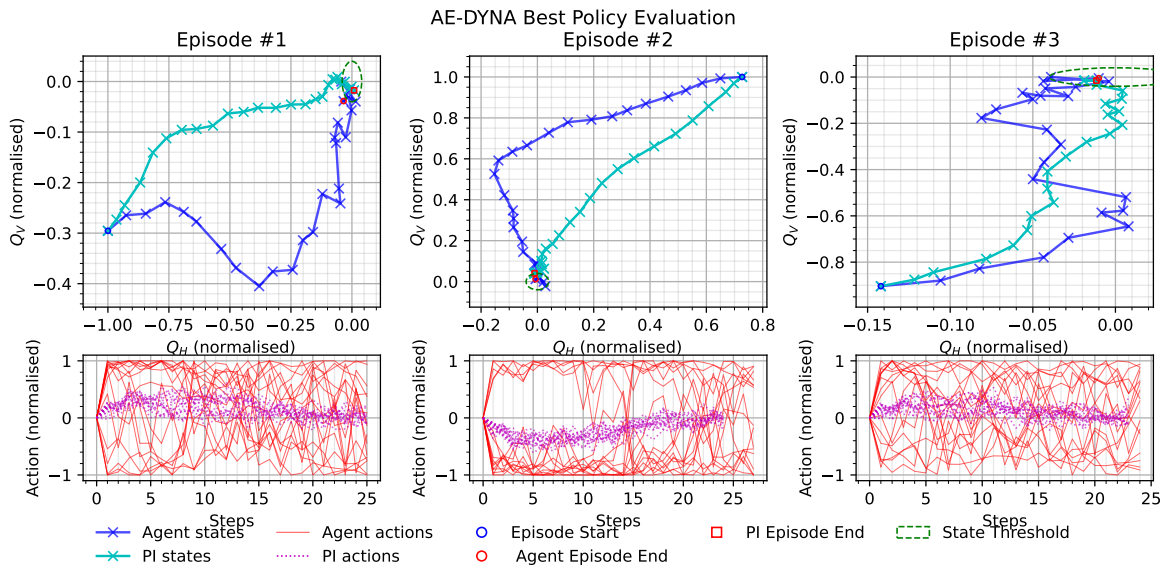


Figure 4.45: Best AE-DYNA-SAC agent. Action Gaussian noise with zero mean and standard deviation 10% of action range.

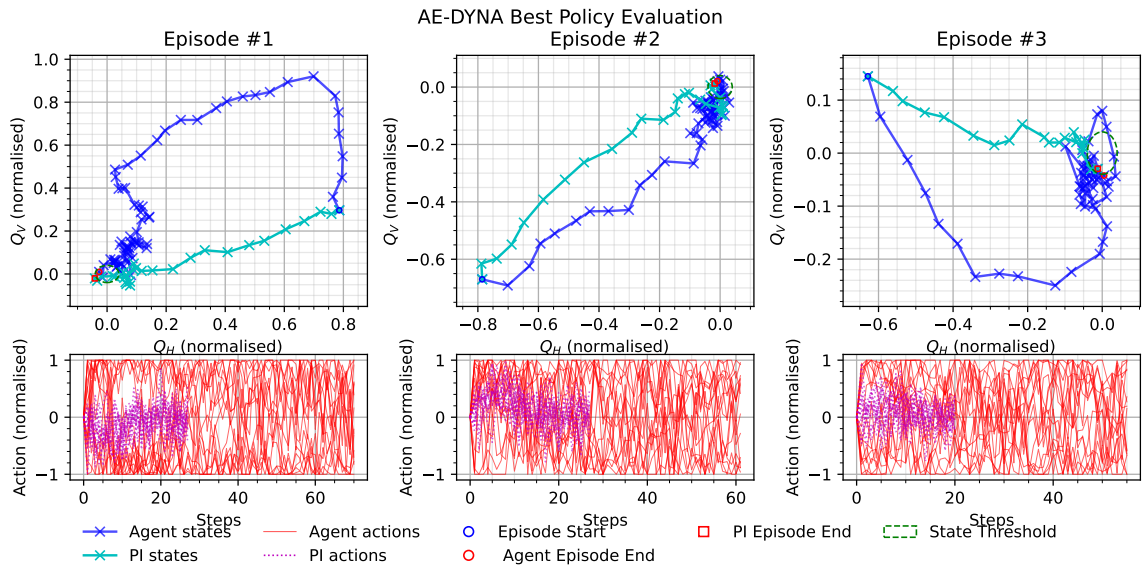


Figure 4.46: Best AE-DYNA-SAC agent. Action Gaussian noise with zero mean and standard deviation 25% of action range.

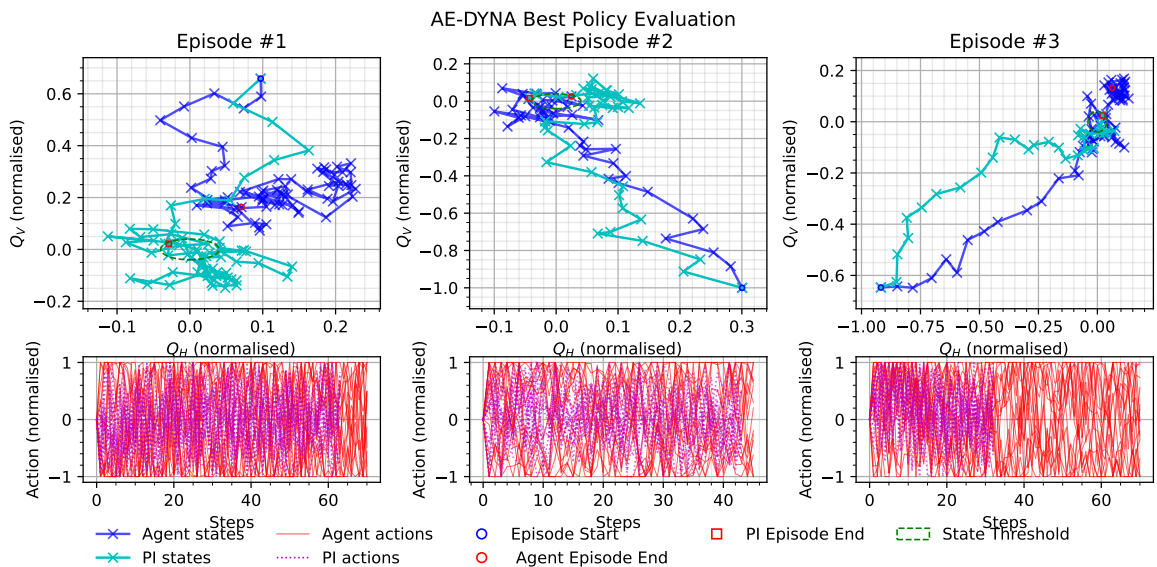


Figure 4.47: Best AE-DYNA-SAC agent. Action Gaussian noise with zero mean and standard deviation 50% of action range.

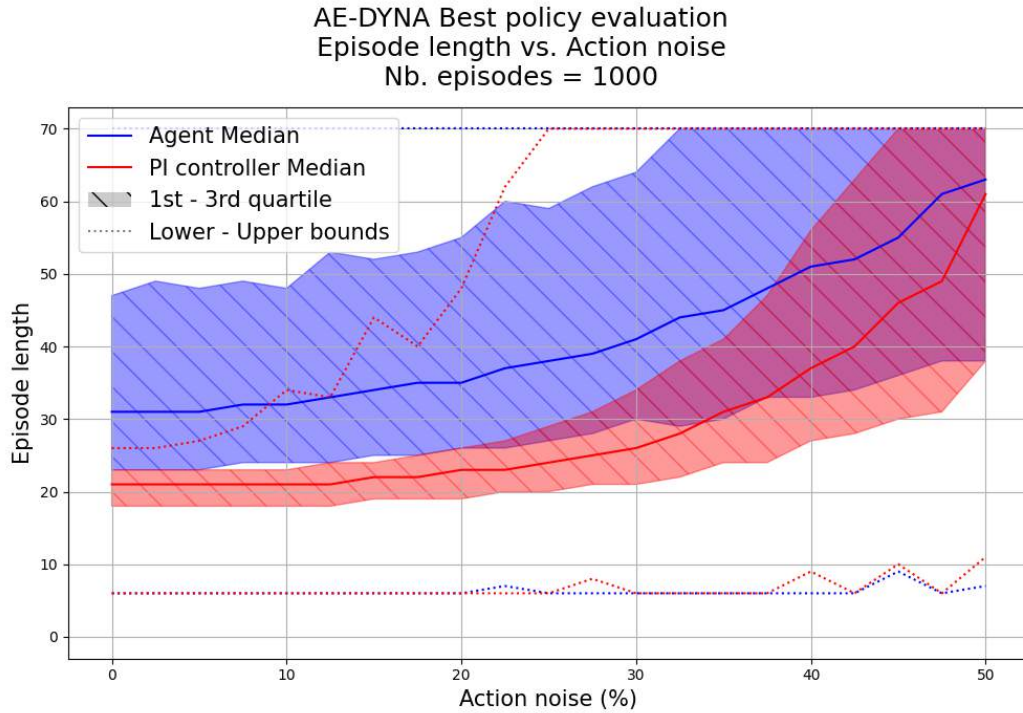


Figure 4.48: Effect of action noise on the episode length due to varying action noise on the best AE-DYNA-SAC agent and the PI controller.

Table 4.15 tabulates the episode length statistics collected from 1000 episodes, for the values of Gaussian action noise considered in the episode evaluation plots. Figure 4.48 illustrates the same information for more values of action noise. Figure 4.49 shows that the DTO of the best AE-DYNA-SAC agent is above the Goal threshold for all action values which implies that there were failed episodes even when using deterministic actions. Similarly to the best SAC and SAC-TFL agents, Figure 4.50 shows that the width of the action value distribution of the best AE-DYNA-SAC agent decreases as the action noise increases.

AE-DYNA Best policy evaluation  
 Final distance to optimal point vs. Action noise  
 Nb. episodes = 1000

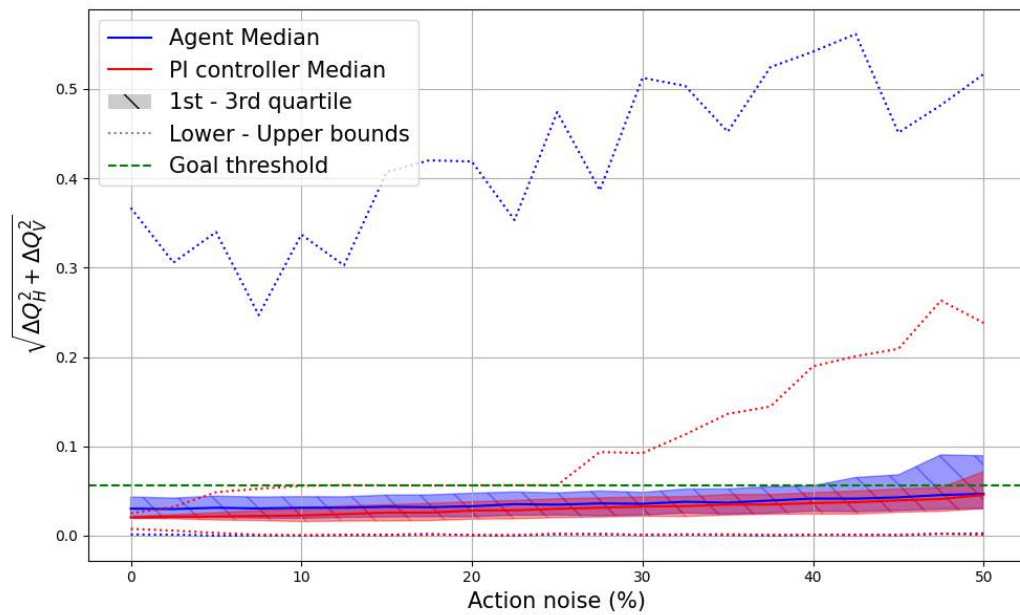


Figure 4.49: Effect of action noise on the distance to the optimal point at the end of the episode due to varying action noise on the best AE-DYNA-SAC agent and the PI controller.



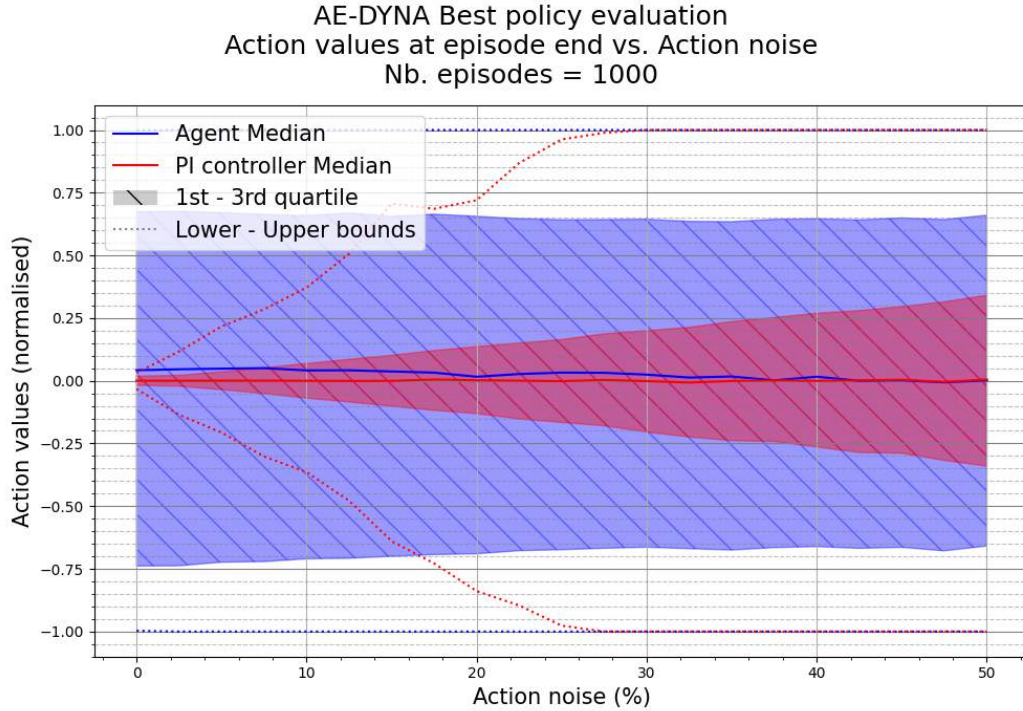


Figure 4.50: Effect of action noise on the last action used in the episode due to varying action noise on the best AE-DYNA-SAC agent and the PI controller.

In summary, NAF2 and PPO trained the best two agents, however, AE-DYNA-SAC was the most sample efficient and also obtained a policy which is stable in low action noise. The policies trained by TD3 and SAC were sometimes successful, however, their performance was significantly worse than NAF2 and PPO. On the other hand, the best SAC-TFL agent trained an adequate policy which works well on QF-BEnv.

### Effect of incorrect tune estimation

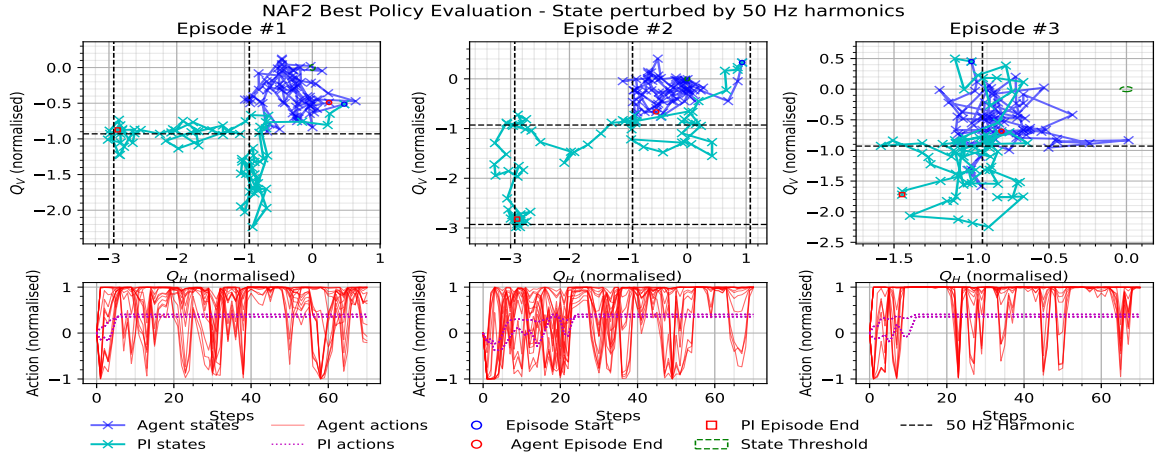
This test subjects the best agents trained in Section 4.1.3 to the effects of 50 Hz noise harmonics explored in Section 2.3.2. A similar procedure to the previous test is followed, where the best agent trained by each respective RL algorithm is loaded and is used to produce evaluation episodes. The only difference in this test is that after each step in the environment, the state is intercepted and a perturbation is added which simulates the effect of 50 Hz noise harmonic-induced perturbations. These

perturbations were obtained by the following steps: a) The state returned by the environment after applying an action,  $\Delta Q$ , was added to a random frequency,  $f_r$ ; b) The realistic spectrum simulation procedure in Section 3.2 was performed for a spectrum with a resonance frequency,  $f_{res}^{true} = f_r + \Delta Q$  and  $\zeta = 10^{\mathcal{U}(-2.5, -1.8)}$ ; c) The BQ algorithm is used to obtain perturbed tune estimates. All the plots shown in this section also show black dashed lines which mark the locations of the horizontal and vertical 50 Hz harmonics in normalised state space.

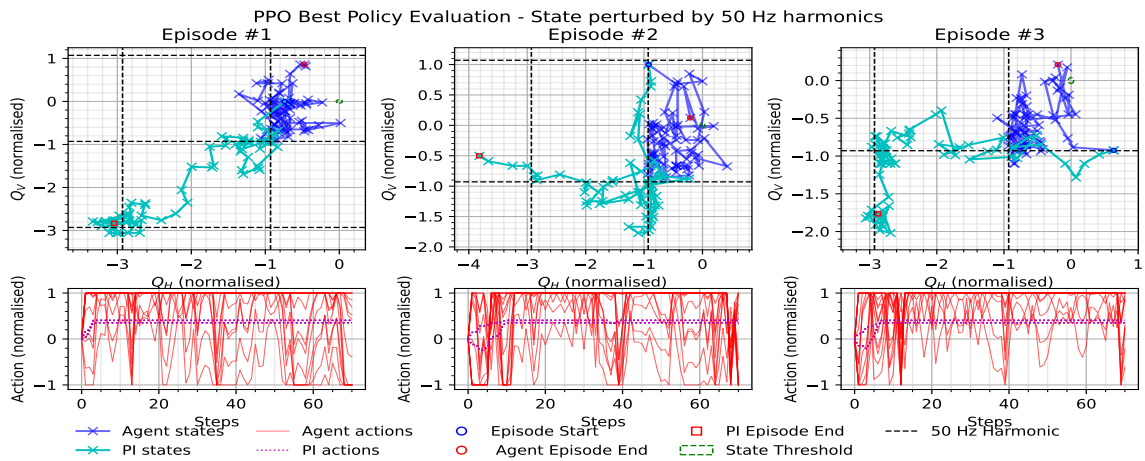
The best two policies so far were trained with NAF2 and PPO. Figure 4.51a and Figure 4.51b show three episodes, obtained by each respective policy. It can be observed that the states of the policies (dark blue) appear to be concentrated around an intersection of the horizontal and vertical 50 Hz harmonics. Similarly, the PI controller state (cyan) are concentrated around the same, or other intersections of the noise harmonics. It can be observed that the state of the two best policies remain on the closest intersection to the optimal point of the state space, while the PI controller states can sometimes extend up to two horizontal, or two vertical 50 Hz harmonics from the optimal point. This observation suggests that even without the tune estimation renovation discussed in Chapter 3, trained NAF2 and PPO agents can maintain the tune error as close as possible to the optimal point.

Figure 4.51c shows the best TD3 policy under the effect of 50 Hz harmonics. It can be observed that the policy behaves similarly to the response of the PI controller and extends up to the second harmonic away from the optimal point. Figure 4.52a shows the best SAC policy in this test and it can be observed that the PI controller maintains a similar performance to the other examples previously shown, while the SAC policy extends up to the fourth 50 Hz harmonics from the optimal point; this is a difference of more than 200 Hz from the optimal point.

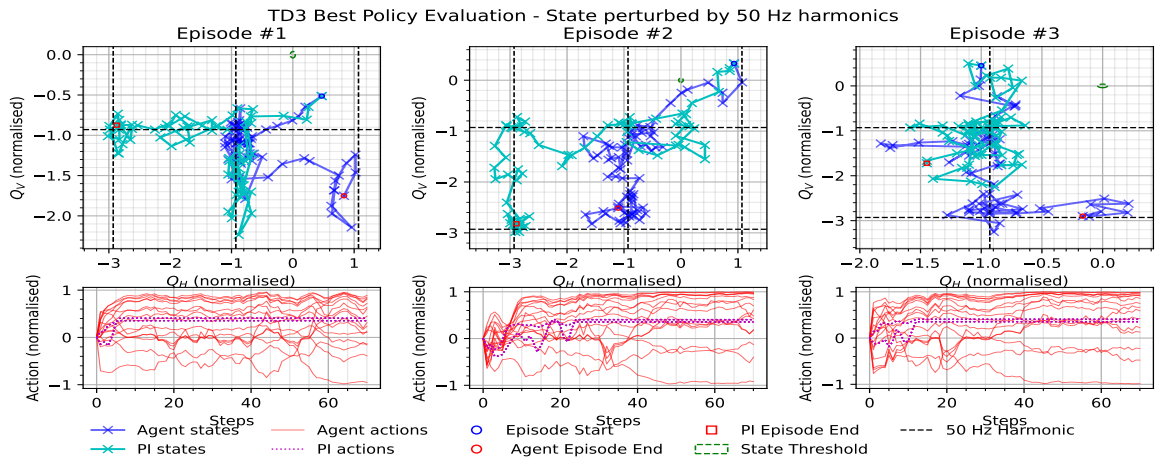
Figure 4.52b shows the results for the best SAC-TFL policy. It can be seen that the state was maintained around the closest 50 Hz harmonic intersection to the optimal point. This performance is similar to the best-performing agents, NAF2 and PPO. Figure 4.52c are the results corresponding to the best AE-DYNA-SAC policy and show that the policy obtained a performance similar to the PI controller.



(a) Best NAF2 agent.



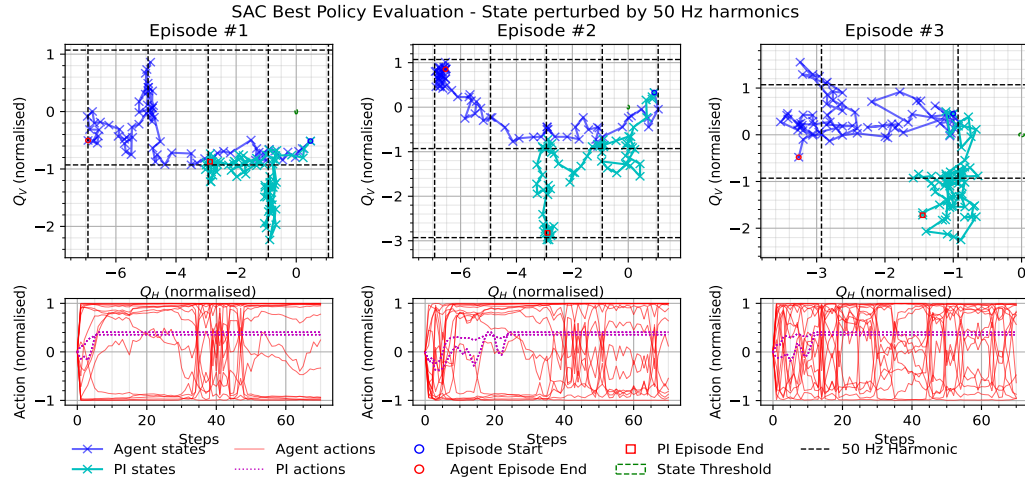
(b) Best PPO agent.



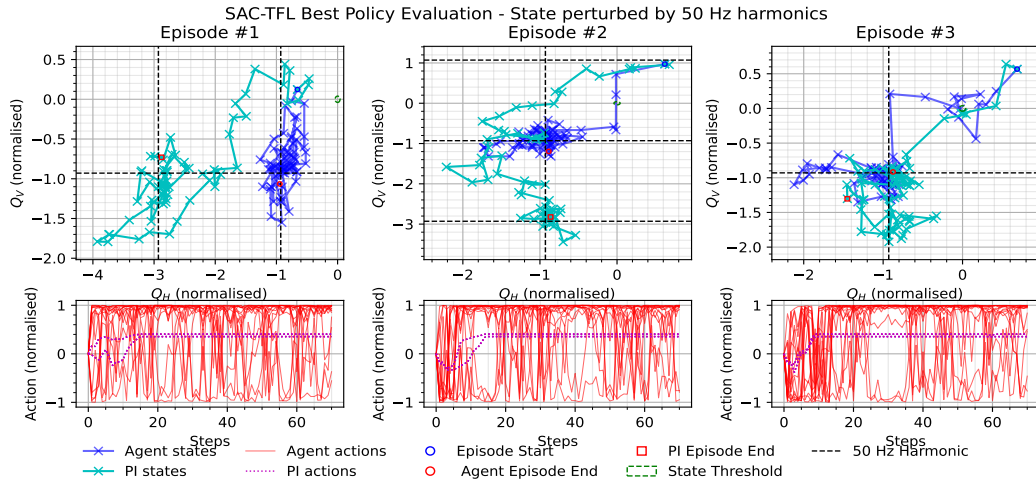
(c) Best TD3 agent.

Figure 4.51: Effect of 50 Hz harmonics on the best NAF2, PPO and TD3 agents, and PI controller.

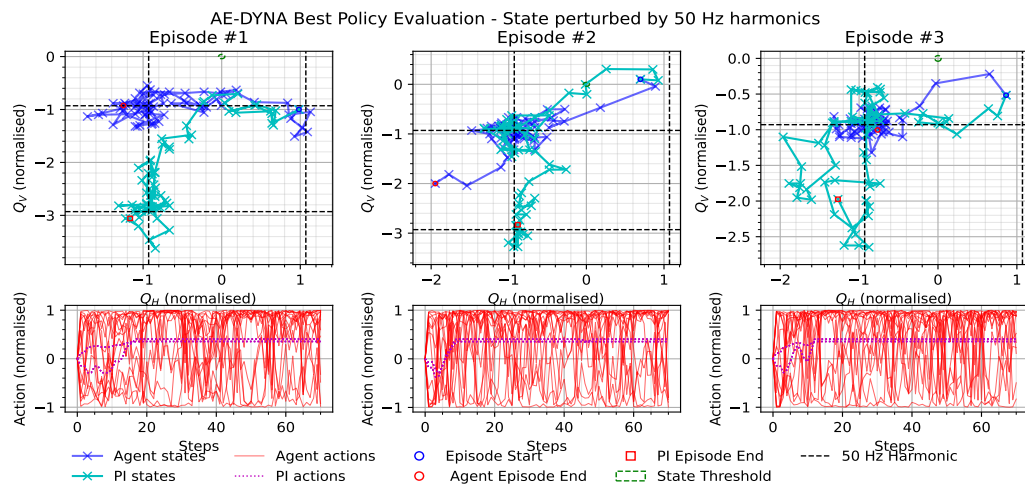




(a) Best SAC agent.



(b) Best SAC-TFL agent.



(c) Best AE-DYNA-SAC agent.

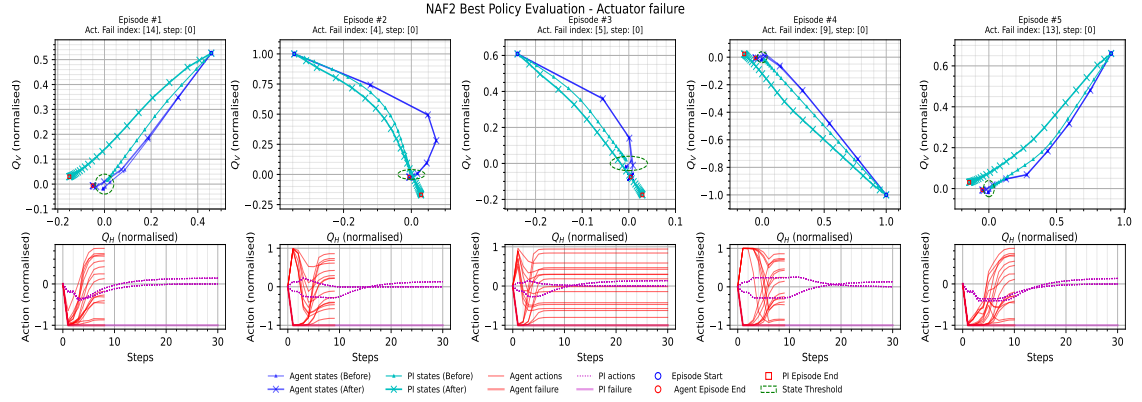
Figure 4.52: Effect of 50 Hz harmonics on the best SAC, SAC-TFL and AE-DYNA-SAC agents, and PI controller.

## Effect of actuator failure

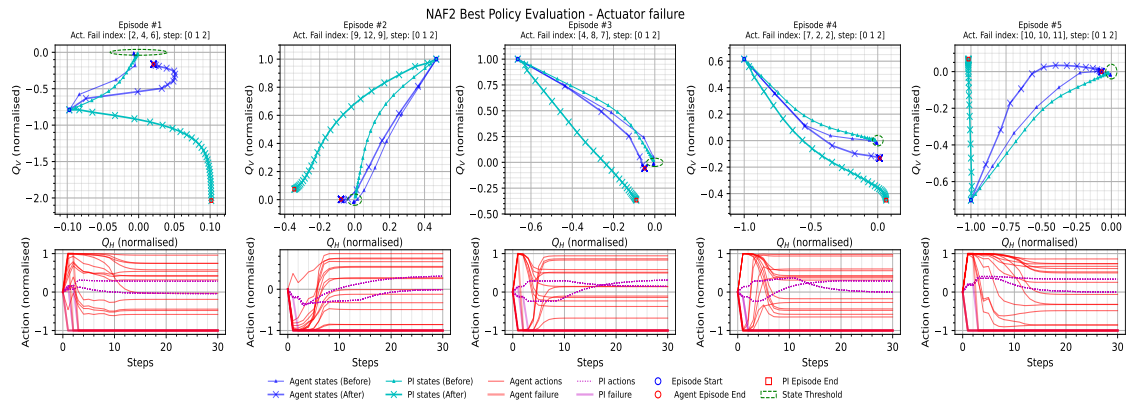
In Section 2.4.1, the possibility of magnet failures was discussed. In this test, the performance of best policies trained in Section 4.1.3 was analysed in the presence of such magnet failures. For each episode shown in this section, an action was chosen at random at a predetermined step in the episode. For the remaining steps until a terminal state, the action chosen was set to -1, to simulate a cool-down of the magnet after a circuit failure. The corresponding action obtained by the PI controller was set to the same value. While this test is not a perfect representation of magnet failures in the LHC, it is a worst case scenario which tests the performance of the policies and PI controller in unseen and unideal conditions.

For each policy, three scenarios with five episodes each are shown. In the first scenario, one actuator fails on step 1; in the second scenario, three actuators fail on steps 1, 2 and 3, respectively; in the third scenario five actuators fail in a similar sequence to the other scenarios. All the plots shown in this test also show an episode trajectory obtained by repeating the episode with the same initial conditions and with all actions functioning. The episode trajectories affected by the actuator failures use  $\times$  as a marker, while the episodes using all actions use  $\blacktriangle$  as a marker. It was also found that for certain scenarios, a maximum episode length of 70 obtained large state deviations. To aid the analysis of the results, the maximum episode length was set to 30.

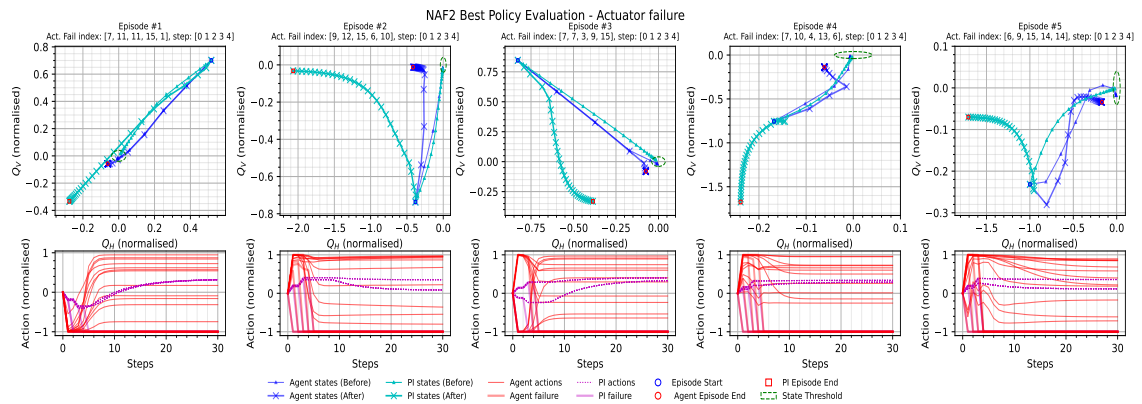
Figure 4.53 shows the effect of actuator failures on the best NAF2 agent. At one actuator failure in Figure 4.53a, the PI controller already diverges from the threshold boundary and fails on all episodes. However, NAF2 still manages to satisfy the early episode termination criterion in  $\frac{4}{5}$  of the episodes and all in under 10 steps. At three actuator failures in Figure 4.53b, both NAF2 and the PI controller fail to converge, however, NAF2 remained closer to the optimal point. This trend continues at five actuator failures in Figure 4.53c.



(a) Best NAF2 agent with 1 actuator failure.



(b) Best NAF2 agent with 3 actuator failures.



(c) Best NAF2 agent with 5 actuator failures.

Figure 4.53: Effect of different number of actuator failures occurring at different episode steps. The best NAF2 policy is compared with the PI controller.

Figure 4.54 shows the effect of actuator failures on the best PPO policy. Similarly to NAF2, PPO performs better than the PI controller in all scenarios. However, unlike

NAF2, PPO still converges in  $\frac{1}{5}$  of the episodes during five actuator failures, and also converges the closest to the threshold boundary in another  $\frac{2}{5}$  of the episodes. The effect of the number of actuator failures on the best PPO policy is also evident in the action evolution plots (red and magenta) of Figure 4.54. These figures along with bottom plots of Figure 4.16, show that when all actions are used, the actions decay the closest to zero at the end of the episode. They also show that the actions still decay during actuator failures, however, the actions remain separated by a range proportional to the number of actuators which failed during the episode. This observation suggests that the PPO policy has successfully generalised the optimal policy trained on one environment, to another environment with slightly different model dynamics.

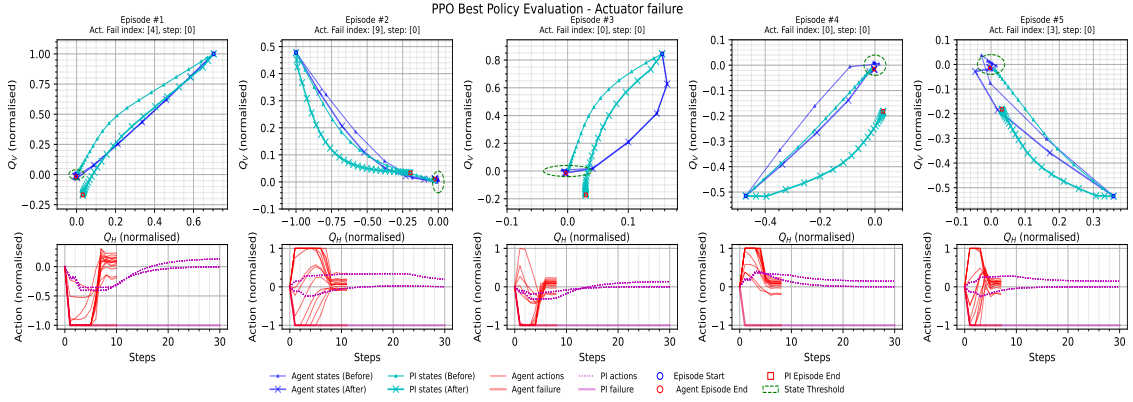
Figure 4.55 shows the results for this test when attempted with the best TD3 policy. It can be seen from Figure 4.55a, that with one actuator failure, TD3 already starts to fail to converge to the optimal point. The trend worsens as the number of actuator failures increase, as can be seen in Figure 4.55b and Figure 4.55c.

Figure 4.56 and Figure 4.57 show the results for the best SAC and SAC-TFL policies, respectively. Both policies appear to have similar performances with one actuator failure in Figure 4.56a and Figure 4.57a. However, at three actuator failures, SAC-TFL converges closer to the optimal point on average, whereas SAC diverges significantly, e.g. Figure 4.57b Episode #4. The performance of SAC-TFL degrades at five actuator failures, as shown in Figure 4.57c, however, at a much lower rate than the SAC policy. This observation suggests that the SAC-TFL implementation is able to generalise better over small model variations.

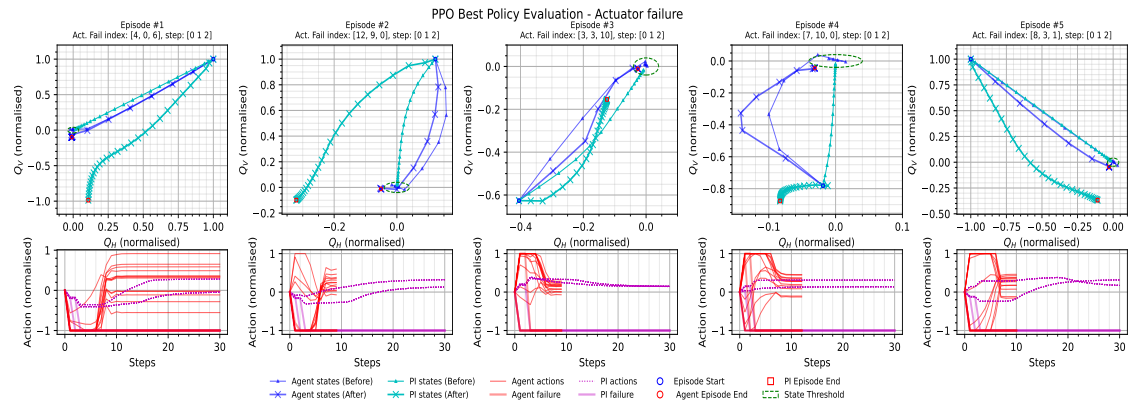
Finally, Figure 4.58 shows the results of the best AE-DYNA-SAC policy. Figure 4.58a shows that the policy can, at times, converge close to the optimal point, however, not enough to terminate the episode, e.g. Episode #4. The performance continues to degrade as more actuator failures are introduced, however, unlike SAC, the policy degradation is comparable to the performance of the PI controller.

In summary, the best two agents found for QFBEnv were the on-policy, model-free PPO agent, and the off-policy, model-free NAF2 agent. Both obtained a high performance in each test, however, the policy trained by PPO showed the desirable

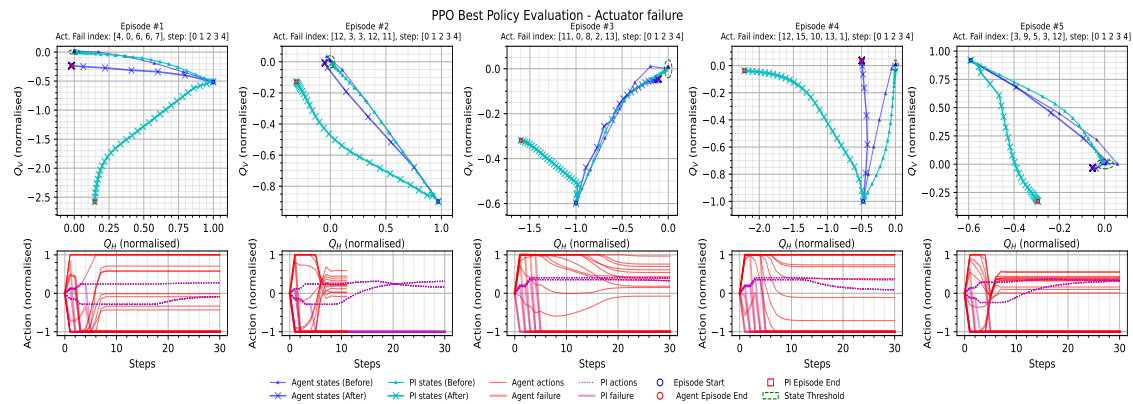
property of the actions decaying to zero, proportional to the reward and the slightly better generalisation observed during the actuator failure test. It was concluded that it was too difficult to tune the hyperparameters of stable-baselines SAC and TD3 and both their best policies suffered from sub-optimal performance even in the simplest, deterministic case. On the other hand, SAC-TFL improved over the training instability of stable-baselines SAC, and obtained a good policy overall. Finally, AE-DYNA-SAC was the most sample efficient agent attempted and the performance of the best policy trained, was comparable to that of the PI controller.



(a) Best PPO agent with 1 actuator failure.

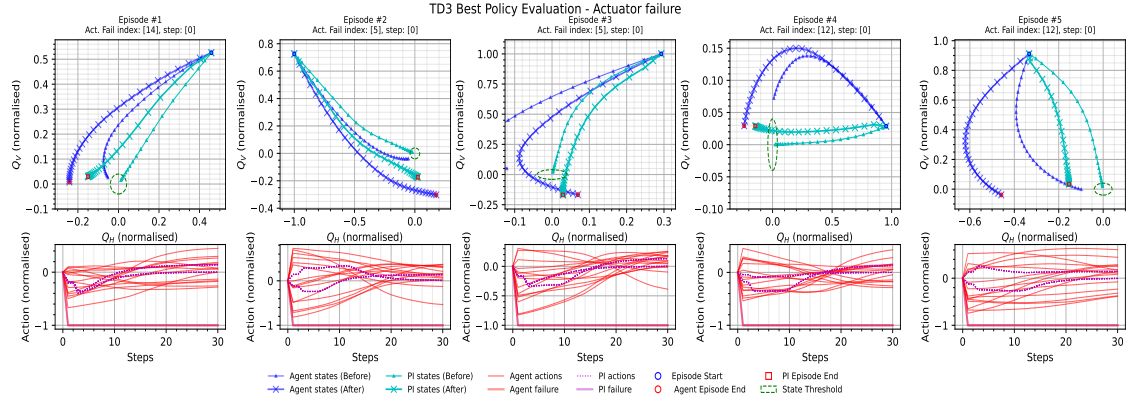


(b) Best PPO agent with 3 actuator failures.

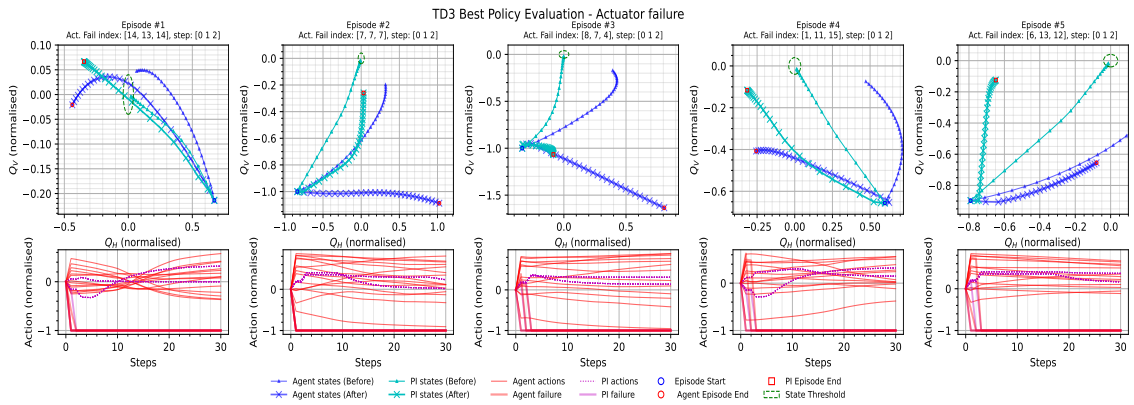


(c) Best PPO agent with 5 actuator failures.

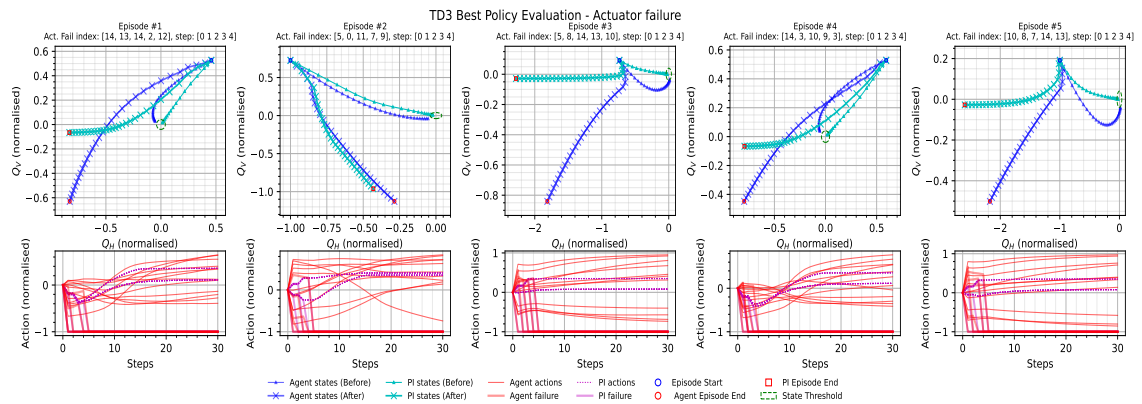
Figure 4.54: Effect of different number of actuator failures occurring at different episode steps. The best PPO policy is compared with the PI controller.



(a) Best TD3 agent with 1 actuator failure.



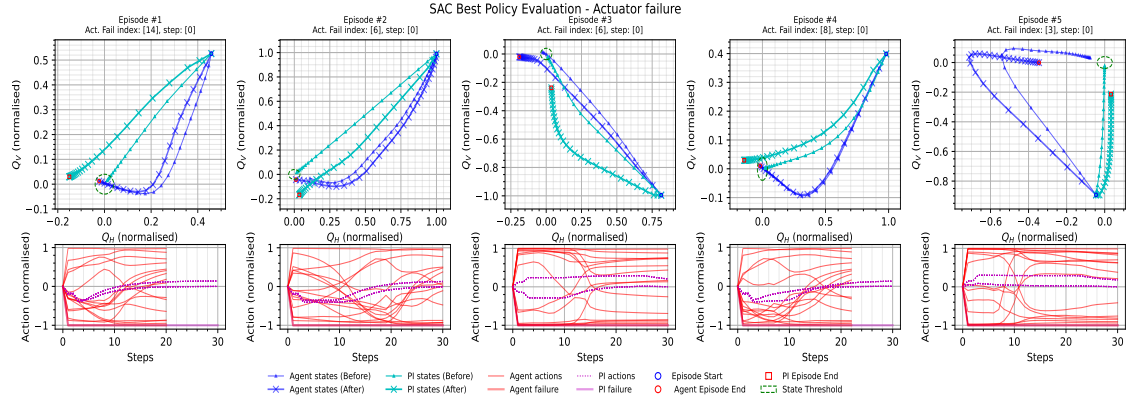
(b) Best TD3 agent with 3 actuator failures.



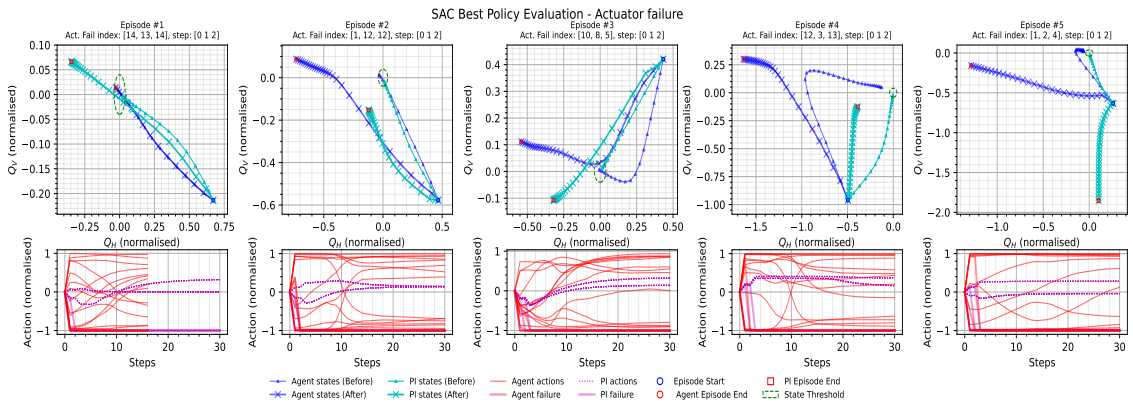
(c) Best TD3 agent with 5 actuator failures.

Figure 4.55: Effect of different number of actuator failures occurring at different episode steps. The best TD3 policy is compared with the PI controller.

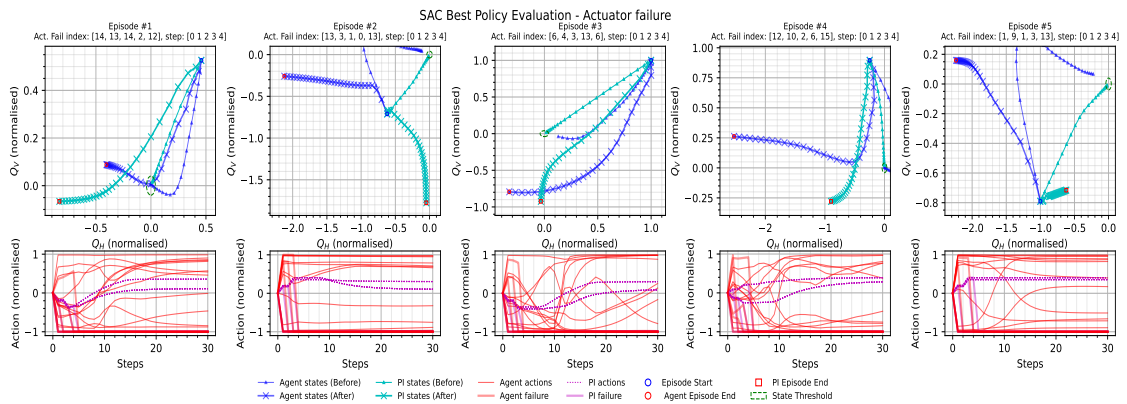




(a) Best SAC agent with 1 actuator failure.



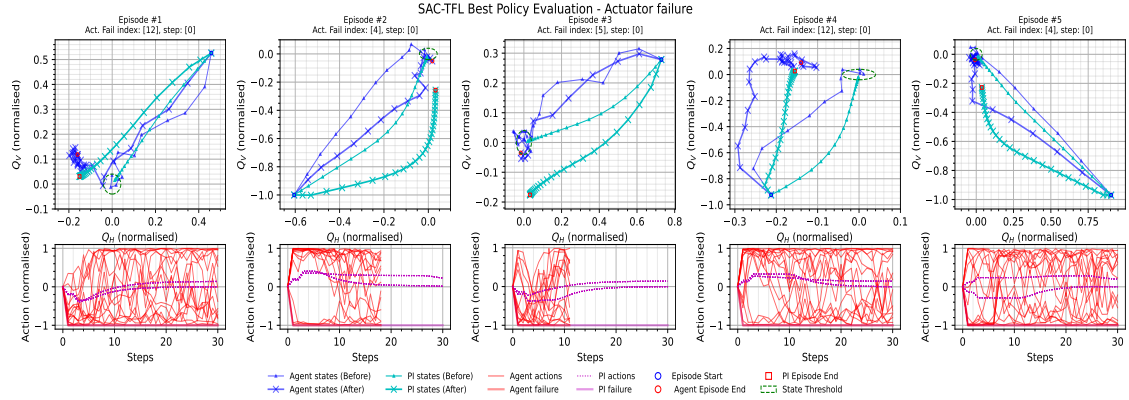
(b) Best SAC agent with 3 actuator failures.



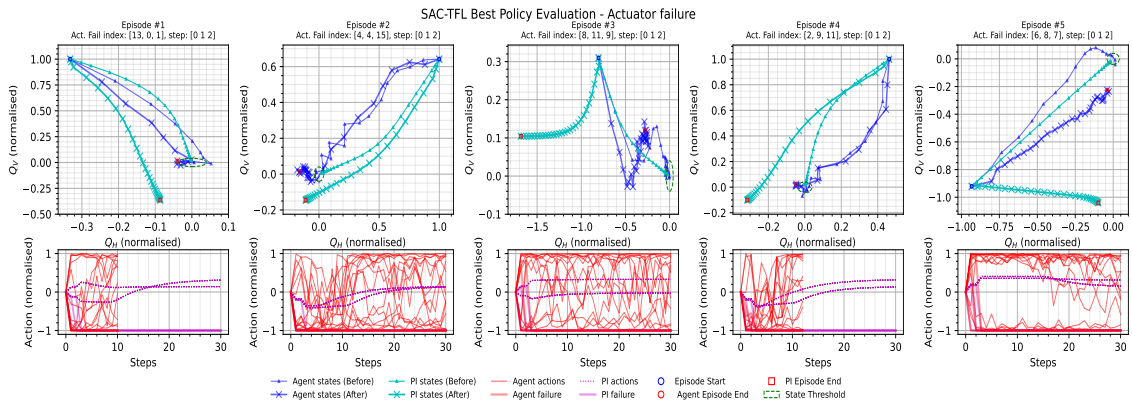
(c) Best SAC agent with 5 actuator failures.

Figure 4.56: Effect of different number of actuator failures occurring at different episode steps. The best SAC policy is compared with the PI controller.

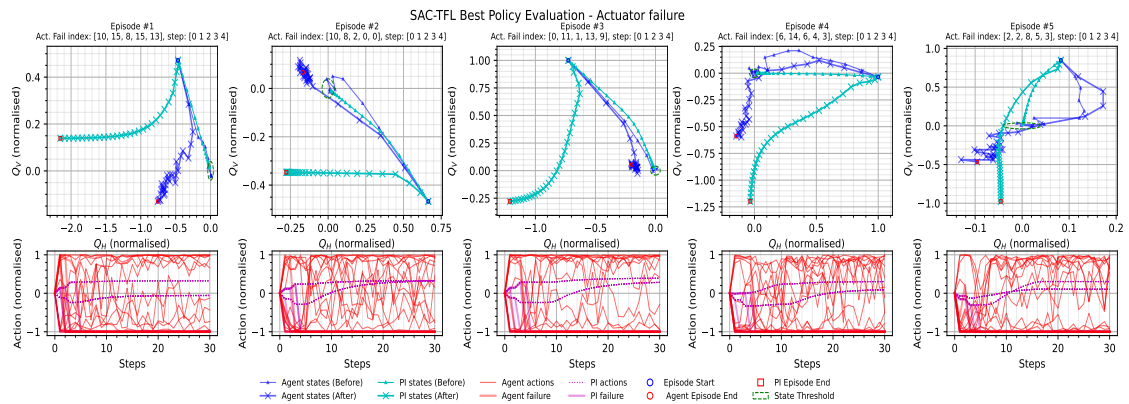




(a) Best SAC-TFL agent with 1 actuator failure.

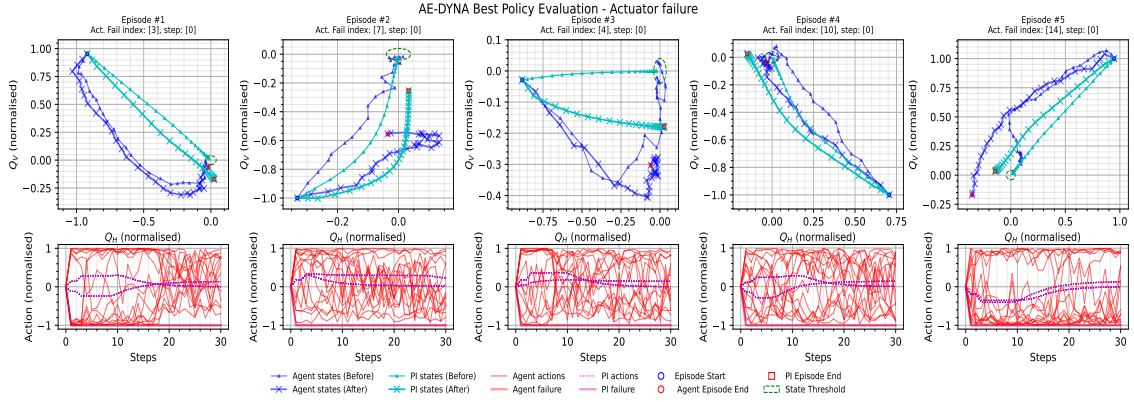


(b) Best SAC-TFL agent with 3 actuator failures.

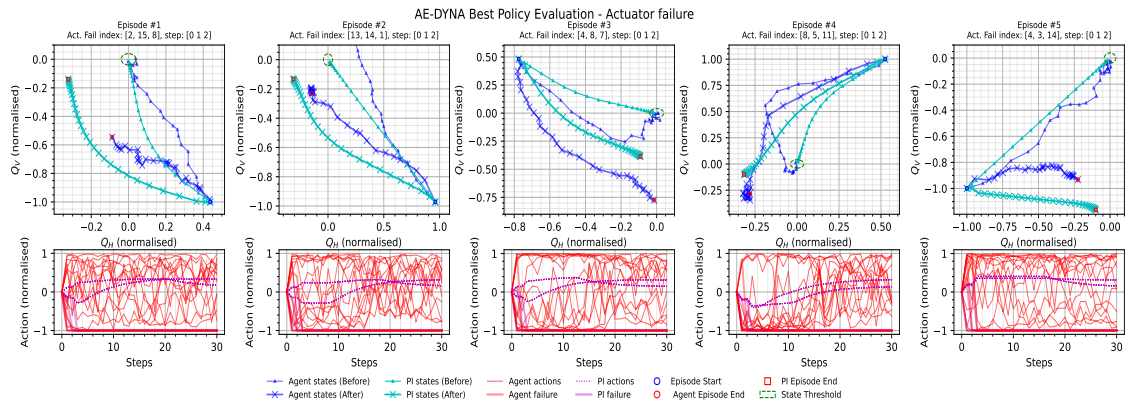


(c) Best SAC-TFL agent with 5 actuator failures.

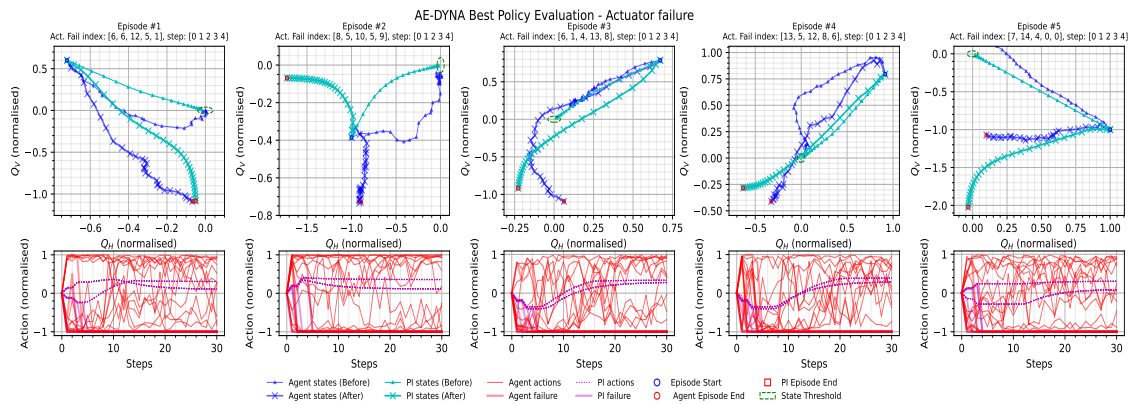
Figure 4.57: Effect of different number of actuator failures occurring at different episode steps. The best SAC-TFL policy is compared with the PI controller.



(a) Best AE-DYNA-SAC agent with 1 actuator failure.



(b) Best AE-DYNA-SAC agent with 3 actuator failures.



(c) Best AE-DYNA-SAC agent with 5 actuator failures.

Figure 4.58: Effect of different number of actuator failures occurring at different episode steps. The best AE-DYNA-SAC policy is compared with the PI controller.

The same procedure used to create the evaluation plots discussed above was used to obtain more statistics on the performance of the trained RL agents and the PI controller in the presence of actuator failures. The maximum episode length was set

Table 4.16: Episode length statistics (mean $\pm$ std) under the effect of actuator failures.

Algorithm	Nb. of actuator failures			
	0	1	3	5
NAF2	9.10 $\pm$ 1.39	33.01 $\pm$ 29.24	69.14 $\pm$ 7.01	70.00 $\pm$ 0.00
PPO	8.80 $\pm$ 1.28	9.14 $\pm$ 1.50	41.60 $\pm$ 29.85	70.00 $\pm$ 0.00
TD3	39.06 $\pm$ 12.08	69.05 $\pm$ 6.21	70.00 $\pm$ 0.00	70.00 $\pm$ 0.00
SAC	43.09 $\pm$ 25.49	62.97 $\pm$ 17.77	69.72 $\pm$ 3.93	70.00 $\pm$ 0.00
SAC-TFL	12.97 $\pm$ 3.03	29.44 $\pm$ 24.69	63.45 $\pm$ 17.40	69.90 $\pm$ 2.21
AE-DYNA-SAC	36.79 $\pm$ 17.55	61.58 $\pm$ 15.97	69.82 $\pm$ 2.72	70.00 $\pm$ 0.00
PI controller	20.09 $\pm$ 3.70	70.00 $\pm$ 0.00	70.00 $\pm$ 0.00	70.00 $\pm$ 0.00

back to 70 steps. The episode length and the final distance to the optimal point (DTO in Equation (4.2)) were used as performance metrics. The same three scenarios with one, two and three actuator failures were attempted, with a total of 1000 episodes per scenario per algorithm used to obtain the performance statistics.

Tables 4.16 and 4.17 tabulate the statistics for the episode length and DTO under the effect of actuator failures for all algorithms attempted in Section 4.1.3. The best PPO agent manages to maintain the smallest average episode length and a DTO which is close to the Goal threshold of QFBEnv at up to three actuator failures. NAF2 and SAC-TFL show a similar performance up to one actuator failure, however, it can be seen that SAC-TFL was the only agent which obtained successful termination at five actuator failures. The performances of TD3, SAC and AE-DYNA-SAC with respect to the average episode length are similar, however, AE-DYNA-SAC obtained slightly lower DTO. Ultimately, all the RL agents behaved better than the PI controller, which obtained no successful termination during actuator failures.

Table 4.17: Distance to Optimal point (DTO) statistics (mean $\pm$ std) under the effect of actuator failures. Underlined cells show where it is likely to observe successful termination.

Algorithm	Nb. of actuator failures			
	<b>0</b>	<b>1</b>	<b>3</b>	<b>5</b>
NAF2	<u>0.02 <math>\pm</math> 0.00</u>	<u>0.05 <math>\pm</math> 0.02</u>	0.13 $\pm$ 0.05	0.56 $\pm$ 0.87
PPO	<u>0.01 <math>\pm</math> 0.00</u>	<u>0.02 <math>\pm</math> 0.01</u>	<u>0.07 <math>\pm</math> 0.02</u>	0.52 $\pm$ 1.02
TD3	<u>0.03 <math>\pm</math> 0.00</u>	0.84 $\pm$ 0.76	1.96 $\pm$ 1.78	3.43 $\pm$ 2.03
SAC	<u>0.04 <math>\pm</math> 0.02</u>	0.25 $\pm$ 0.16	1.92 $\pm$ 1.04	5.02 $\pm$ 1.82
SAC-TFL	<u>0.02 <math>\pm</math> 0.01</u>	<u>0.07 <math>\pm</math> 0.07</u>	0.26 $\pm$ 0.17	0.94 $\pm$ 0.75
AE-DYNA-SAC	<u>0.04 <math>\pm</math> 0.04</u>	0.18 $\pm$ 0.14	1.03 $\pm$ 0.87	2.18 $\pm$ 1.46
PI controller	<u>0.02 <math>\pm</math> 0.00</u>	0.17 $\pm$ 0.01	0.98 $\pm$ 0.97	3.10 $\pm$ 1.34
Goal threshold	0.057			

## 4.2 OFC RL

The control of the orbit is more critical due to the larger number of actuators and sensors. In this section, the RL algorithm that obtained the best performance on QFBEnv was attempted on an OpenAI Gym environment called OFCEnv. Its training is used to demonstrate some shortcomings of applying state-of-the-art RL algorithms directly on large continuous action spaces.

The OFC on one beam and one plane relies on a RM, which is a  $276 \times 544$  matrix modelling the change in the horizontal or vertical beam orbit due to change in the CODs deflections [ $\mu\text{rad}$ ]. Therefore, a vector containing the delta COD deflections,  $\Delta \vec{\delta}_z$  is multiplied by the RM:

$$\Delta \vec{\delta}_z \cdot RM = \Delta \vec{z}$$

where  $\Delta \vec{z}$  is the modelled change in beam orbit due to  $\Delta \vec{\delta}_z$  being applied to the CODs. Note that  $z$  denotes either the horizontal or vertical planes.

The OFC uses two RM per beam and each matrix is obtained from the linear

beam optics of the LHC. The orbit control sequence occurs at 25 Hz and a predefined sequence of steps is performed where: a) The orbit error,  $\Delta \vec{z}$ , is obtained by subtracting the latest beam measurements from the reference orbit; b) A velocity form PI controller is applied by using  $\Delta \vec{z}_t$  and  $\Delta \vec{z}_{t-1}$ , where  $t$  denotes the time step; c) The PI output is multiplied by the pseudo-inverse matrix of RM (Pseudo-Inverse (PInv)) to obtain a set of COD deflections; d) The deflections are converted to current in Amperes; e) The currents are globally scaled down to accommodate the slowest acting COD; f) The current corrections are sent to the COD power converters via UDP packets.

Ultimately, the problem complexity increases linearly with every plane and beam added to the OFC. For this reason, any RL agent which can converge to an optimal policy on one beam and one plane, can cover the entire functionality of the OFC.

### 4.2.1 Environment Setup

The successful implementation of the smaller environment QFBEnv proved useful during the development of the OFCEnv. Both environments use the OpenAI Gym framework with the basic implementation of OFCEnv following that of QFBEnv closely [101]. The RM used by OFCEnv is a  $276 \times 544$  matrix, corresponding to the 276 horizontal orbit correctors and the 544 Beam Position Monitors (BPMs) available on beam 1.

The normalised state space represented a range of  $[-1.5 \text{ mm}, 1.5 \text{ mm}]$  of beam orbit error from the reference orbit. The initial state of each episode was sampled from  $\mathcal{U}(-1\text{mm}, 1\text{mm})$ . The goal of OFCEnv was to reduce the beam orbit error on a simulated beam response in the horizontal plane below an absolute threshold of 0.3 mm, which corresponds to the threshold orbit during injection [1]. Following the suggestions in [102], the state was also standardised by dividing by a running standard deviation of every state value.

The normalised action space represented the fraction of the total allowable current rate in the magnets. Similarly to QFBEnv, considering that the OFC operated at 25 Hz, the time between each step in the environment is 40 ms. Therefore, a normalised

action of 1 on a magnet with a maximum current rate of  $0.5 \text{ A s}^{-1}$  is equivalent to a current change of  $0.5 \times 0.04 = 20\text{mA}$ .

The reward function used for OFCEnv was the same as QFBEnv, Equation (4.1). The early successful termination criterion was limited to one reward above the threshold reward, in order to make a success more likely to occur. The PI controller functionality implemented on QFBEnv was also re-implemented in OFCEnv. In this work, it was primarily used to obtain an adequate episode length to use during training. Consequently, it was decided to set the maximum episode length to 500 steps.

### 4.2.2 Training

Due to the limited computing resources available during this work, a small number of agents were trained with a relatively large number of environment steps ( $> 1 \times 10^6$ ). A lecture by Schulman [102] highlighted the difficulties that can be faced when attempting new RL on new problems. In particular, when using any Stochastic Policy Gradient (PG) method such as PPO, varying,  $\epsilon$ , and the number of steps used for every policy update, greatly affects the training performance of the agent.

Name	PPO#0	PPO#1	PPO#2	PPO#3
Learning rate	$2.5 \times 10^{-4}$			
$\gamma$	0.99			
$\epsilon$	0.4	0.4	0.2	0.2
Batch size	1000	1000	2000	500

Table 4.18: Hyperparameters used for different PPO agents on OFCEnv.

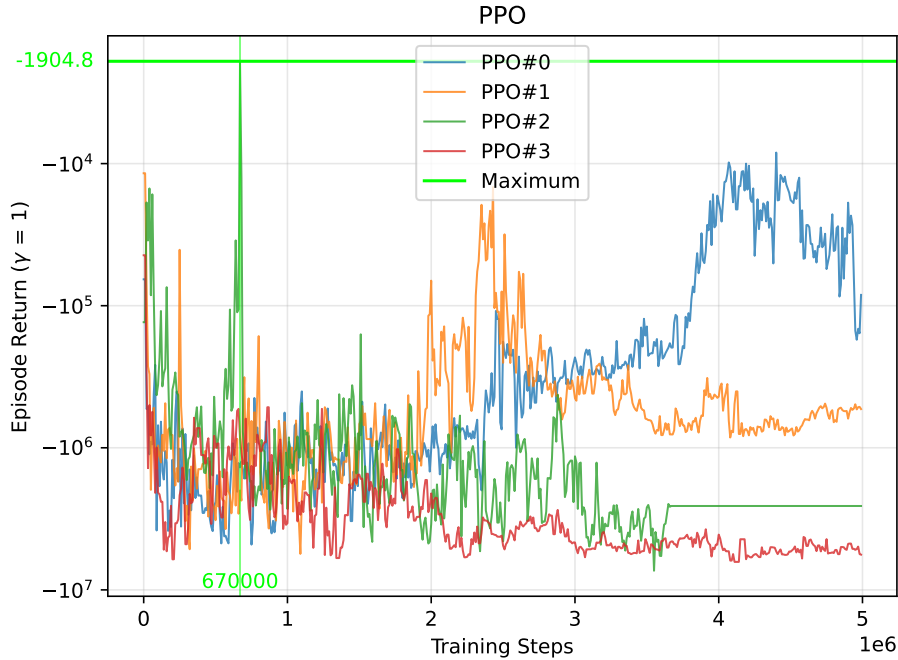


Figure 4.59: Undiscounted episode returns for four PPO agents trained on OFCEnv, using the hyperparameters tabulated in Table 4.18.

Table 4.18 tabulates the hyperparameters of four PPO agents attempted and their episode returns during training is shown in Figure 4.59. It can be observed that the maximum return was obtained by PPO#2 at 670,000 training steps. However, this policy is quickly forgotten and PPO#2 retains a poor performance throughout the next approximately 3 million training steps. PPO#2 and PPO#3 suggest that PPO is more susceptible to a change in  $\epsilon$ , and that varying the batch size at  $\epsilon = 0.2$ , did not improve the training performance. The highest performance agent was PPO#0, which exceeded PPO#1 by almost two orders of magnitude at 4 million time steps. Regardless, no policy was ever successful in satisfying the early termination criterion.

The training shown in this section illustrates the degradation of the sample efficiency of state-of-the-art RL algorithms in large state-action spaces. Other types of RL algorithms, e.g. NAF2 and SAC were also attempted on OFCenv, but similar results were obtained. It was also observed that training OFCEnv requires substantial computing resources and therefore, any future studies must be conducted on more powerful machines, e.g. cloud clusters.

### 4.3 Summary

This chapter presents a feasibility study on the application of Reinforcement Learning (RL) in the Beam-Based Feedback System (BBFS). The control problem present in the Tune Feedback (QFB) was converted into an RL environment called QFBEnv as presented in Section 4.1.1. The linear beam optics matrix (Tune Response Matrix (QRM)) used in the QFB controller was used to implement the forward dynamics of QFBEnv. The standard Proportional-Integral (PI) controller using the pseudo-inverse of the linear dynamics (Tune Pseudo-Inverse (QPI)) was re-implemented and re-tuned on QFBEnv to serve as a reference. A total of five types of RL algorithms were chosen and trained on QFBEnv as documented in Section 4.1.3.

The best agents trained by each RL algorithm along with the PI controller, were subjected to series of tests in Section 4.1.4. The first test in Section 4.1.4 tests the robustness of each agent to the effect of additive Gaussian action noise with zero mean. The second test in Section 4.1.4 looks at the effect of a systematic source of state perturbation, obtained by simulating the effect of 50 Hz noise harmonics as introduced in Section 3.2. The last test in Section 4.1.4 simulates a worst-case scenario to mimic malfunctioning magnets in the BBFS. Therefore, the last test subjects each agent to a QFBEnv with slightly different dynamics from the one used to train the agents themselves.

The best policy was trained by a Proximal Policy Optimisation (PPO) agent, which is a Model-Free (MF) and an on-policy algorithm. It was shown that a learned policy on QFBEnv is robust to action noise as well as to systematic perturbations to the state space. The results tabulated in Tables 4.16 and 4.17 also show that PPO can successfully terminate episodes until at least three actuator failures. This is in contrast to the standard PI controller, which never successfully terminates when the environment dynamics are changed. This concludes that it is possible to outperform the state-of-the-art beam-based controllers using RL.

Finally, Section 4.2 takes a preliminary look at the difficulties that arise when training with larger environments of the same type of QFBEnv. The control prob-



lem in the Orbit Feedback Controller (OFC) was implemented by OFCEnv, which is essentially a scaled up version of QFBEnv with a state-action space an order of magnitude larger. It was shown that the number of training steps required to converge using state-of-the-art RL algorithms would increase by at least two orders of magnitude. This is infeasible in a real-time scenario on the LHC, hence, more research into sample-efficient RL algorithms is required.

# Chapter 5

## Renovation of the beam-based feedback systems

In this chapter, the renovation of the new Beam Feedback Controller LHC (BFCLHC) is introduced. The BFCLHC solves many of the problems encountered during the operation of the Beam-Based Feedback System (BBFS) until Run 2. In particular, the hardware, software and operational side of the BBFS were renovated. This renovation also saw the development of a hardware agnostic testing framework done in collaboration with BE-OP-LHC, which can be used to perform both static and dynamic tests. The first expected operation of the BFCLHC is in Run 3, however, tests are already being conducted on the on-line machines themselves.

### 5.1 Original BBFS design

The original implementation of the BBFS was comprised of two components, the Orbit Feedback Controller (OFC) and the Orbit Feedback Service Unit (OFSU). Figure 2.18 shows an overview of the internal structure of the original BBFS. The principal design objective during the implementation of the BBFS was to achieve a deterministic behaviour of the OFC. This was a problem at the time since the hardware posed a limitation. Consequently it was decided that in order to ensure a deterministic behaviour of the BBFS, the OFC and the OFSU would execute on two different machines. The

settings of the feedbacks were communicated between the OFSU to the OFC via a private Ethernet cable over a Transmission Control Protocol (TCP) connection while the acquisition data concentrated by the OFC was streamed to the OFSU using a User Datagram Protocol (UDP) connection. The OFC was the foremost concentrator for all beam position data coming from the Beam Position Monitor (BPM) systems and the measured tunes from the Base-Band Q (tune) (BBQ) systems. In addition, the OFC computed and drove the necessary corrections to the currents flowing in the Orbit Corrector Dipoles (CODs) and quadrupoles implemented in 2 independent feedback loops. All other tasks, such as data logging and access to the OFC settings, were performed via the OFSU.

The OFC was written in the C++ object-oriented programming language, and relied heavily on ROOT libraries. ROOT was developed and is still maintained by CERN and was originally intended for efficient calculations on very large data accumulated by high-energy physics experiments [103]. The programming paradigm of ROOT is object-oriented where all objects derive from what is called a TObject. All TObjects can use the various functionalities implemented in ROOT, such as I/O handling, memory management and error handling. The OFC consists of several TObject-derived classes, which implement all the required functionality. The core of the OFC was a loop-based program, where every iteration was one simultaneous pass through the control loops of the OFC and the QFB. The latest version of the OFC used ROOT Release 5.34/20 [104].

Figure 5.1 shows the flowchart of the OFC main program. `GLOBAL_RUN` was the variable which controlled the control loop. The `INIT` block is expanded in Figure 5.2 where it can be seen that several initialisation files were required to set up the OFC. The configuration file set up the thread parameters such as their priority and affinity and also contained two important parameters related to testing the OFC; `SIM_MODE` and `BASE_PORT`. `SIM_MODE` were used extensively throughout all the OFC code to change its behaviour during testing. Functionalities such as sending the UDP packets to the Function Generator/Controllers (FGCs) and Radio-Frequency (RF) cavities were suppressed when `SIM_MODE` was set in order to confine the output of the OFC to the

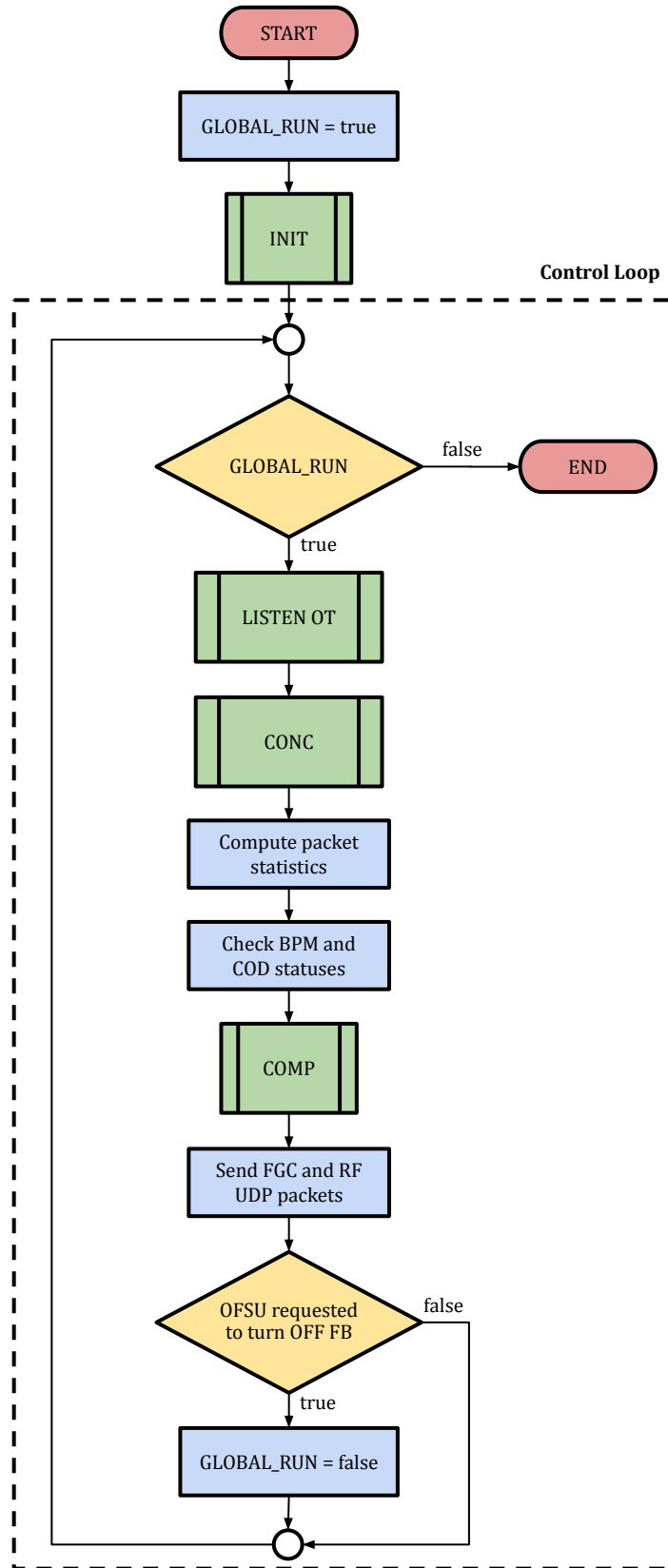


Figure 5.1: Flowchart of the main program of the Orbit Feedback Controller (OFC).

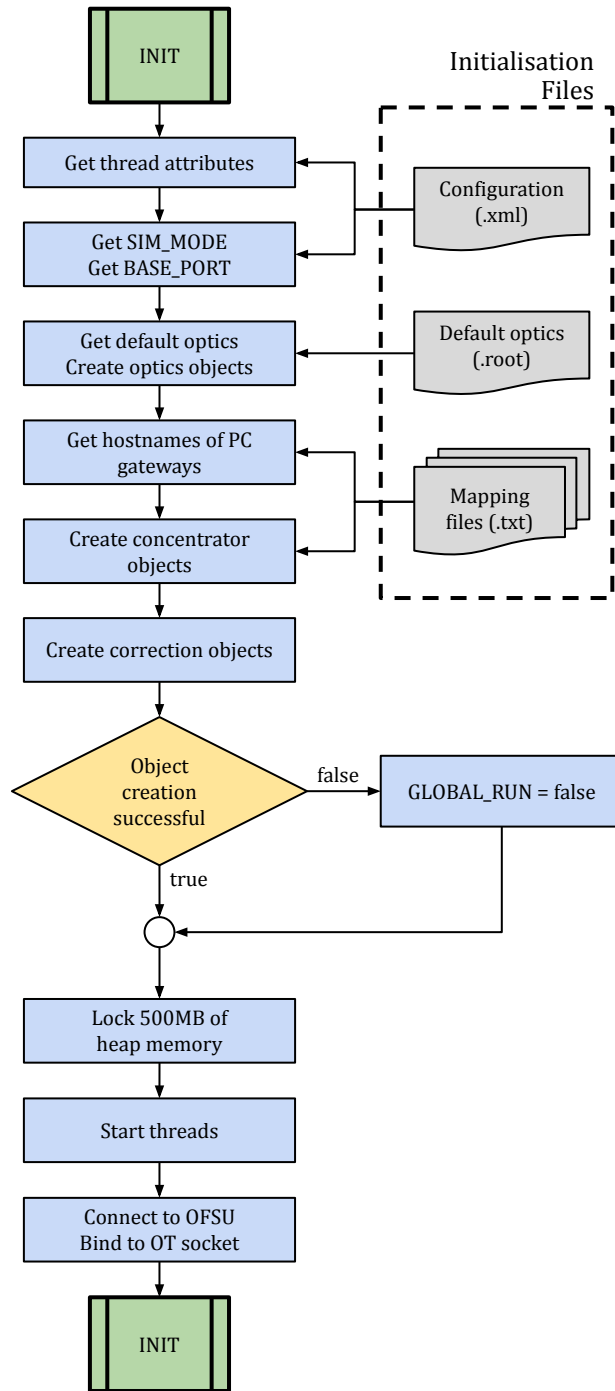


Figure 5.2: Flowchart of the INIT block in Figure 5.1.

testing framework. `BASE_PORT` served as an offset port number for all the network sockets used by the OFC. It was only changed during testing, when data was supplied by the testing framework. The configuration file contained other parameters, however, their use had become deprecated throughout the OFC lifetime.

The OFC also required a `ROOT` file upon initialisation which contained default optics stored in a `ROOT` object. This file existed since the first version of the OFC and due to its age, it proved difficult to verify its contents. Its removal was also deemed dangerous due to its ubiquitous use throughout the initialisation code and therefore it was left up to the next major renovation to be removed. Several other text files were also required upon initialisation which provided: a) the hostnames of all the Front-End Computers (FECs) that the OFC communicated with; b) the parameters of the magnets, e.g. maximum current rate; c) network routing information for all the BPMs, CODs, and quadrupoles required to create the network packets.

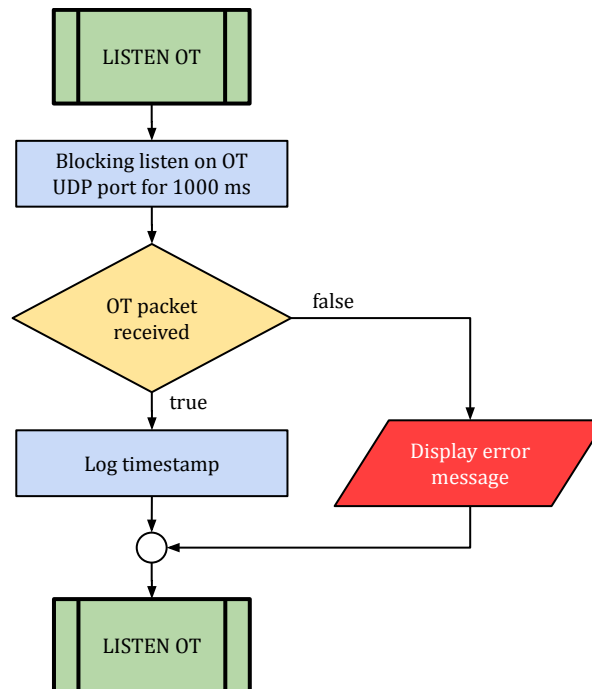


Figure 5.3: Flowchart of the `LISTEN OT` block in Figure 5.1, which waits for the reception of the Orbit Trigger (OT) TCP packet from the OFSU.

After a successful initialisation the control loop took over. The first task at the start of every loop was `LISTEN OT`. Figure 5.3 expands `LISTEN OT` which shows that

the control loop is blocked until the Orbit Trigger (OT) Beam Synchronous Timing (BST) signal is received (at a rate of 25 Hz). The blocking listen had a timeout of 1 s as a fail-safe measure. Therefore, in the event that the OT was not received the BBFS operated at 1 Hz.

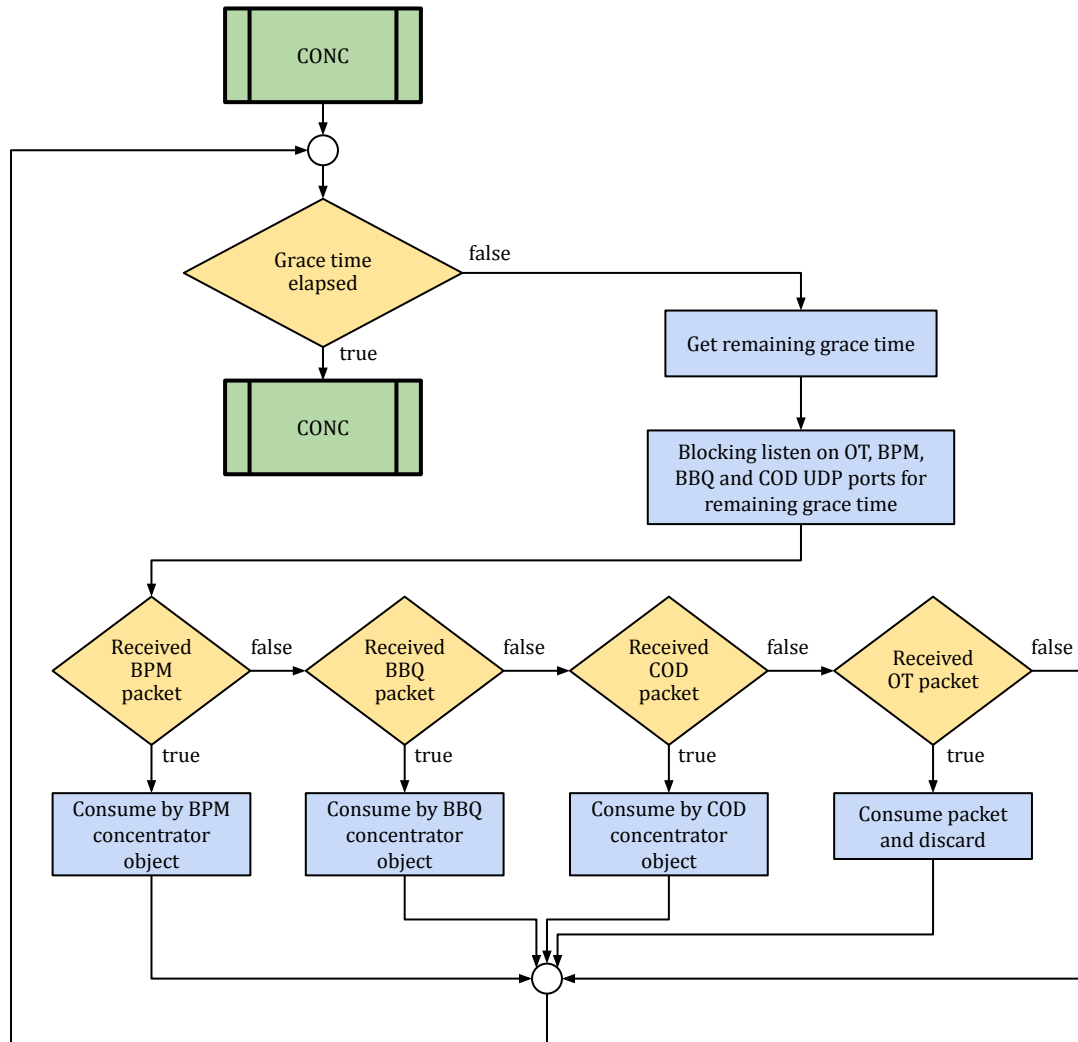


Figure 5.4: Flowchart of the CONC block in Figure 5.1, which concentrates the incoming data from the BPMs, BBQ and CODs.

The second task of the control loop was the data concentration in the CONC block. Figure 5.4 expands CONC and illustrates that the reception of any data depends whether a grace time has elapsed. The grace time is an amount of time dedicated to listening for new data. The grace time is set to be shorter than the OT period of  $\frac{1}{25\text{Hz}} = 40\text{ms}$ . Normally the grace time is set to 10 ms [1], which means that all the BPM, BBQ and

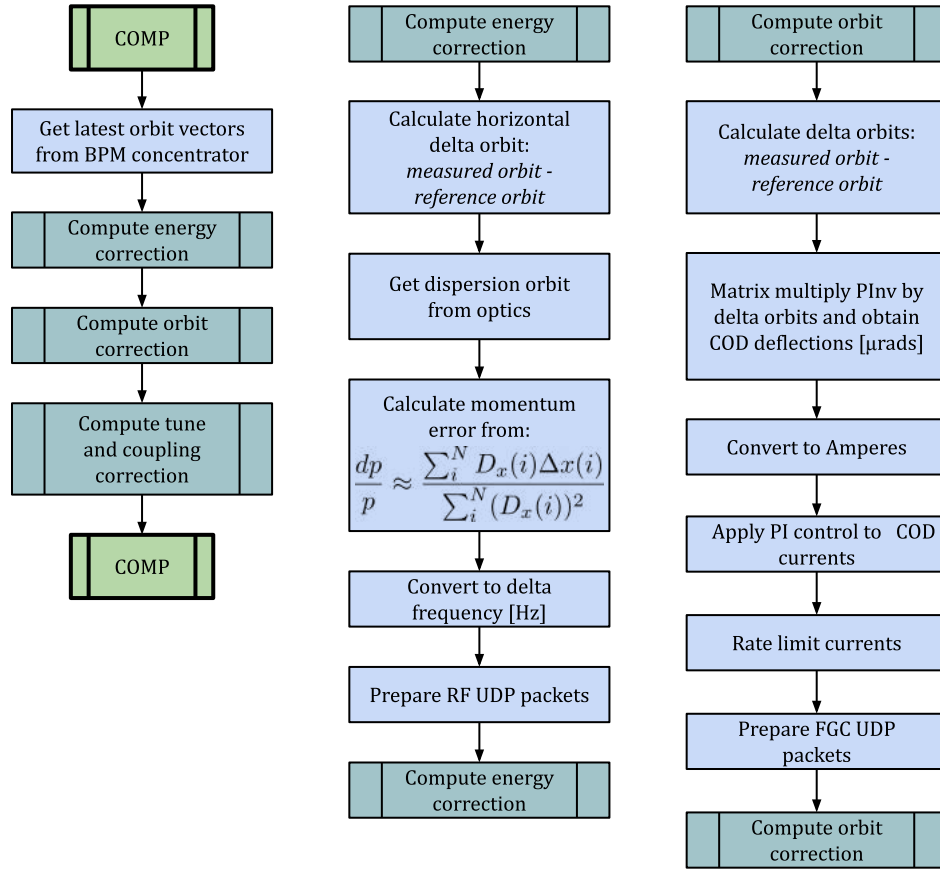


Figure 5.5: Left flowchart shows the COMP block in Figure 5.1, middle flowchart shows the energy (RF frequency) correction and the right flowchart shows the orbit correction.

magnet UDP packets are concentrated only during this time.

Following the data concentration during the grace time, some packet statistics are logged and the BPMs and CODs are masked according to the status bit contained within the UDP packets. For example, in the event that a BPM is marked as faulty, its position measurement is not taken into consideration. Instead the reference position is assumed at its longitudinal location in the LHC. Section 2.4.1 expanded on the operation of the BBFS during malfunctions of BPMs and CODs.

On the left of Figure 5.5, the COMP block is expanded which is responsible for calculating the required corrections to the magnets.. The first task is to update the internal memory with the latest measurements obtained from the CONC block. Following this, the energy corrections are calculated and the delta RF frequency is obtained and prepared in a UDP packet.



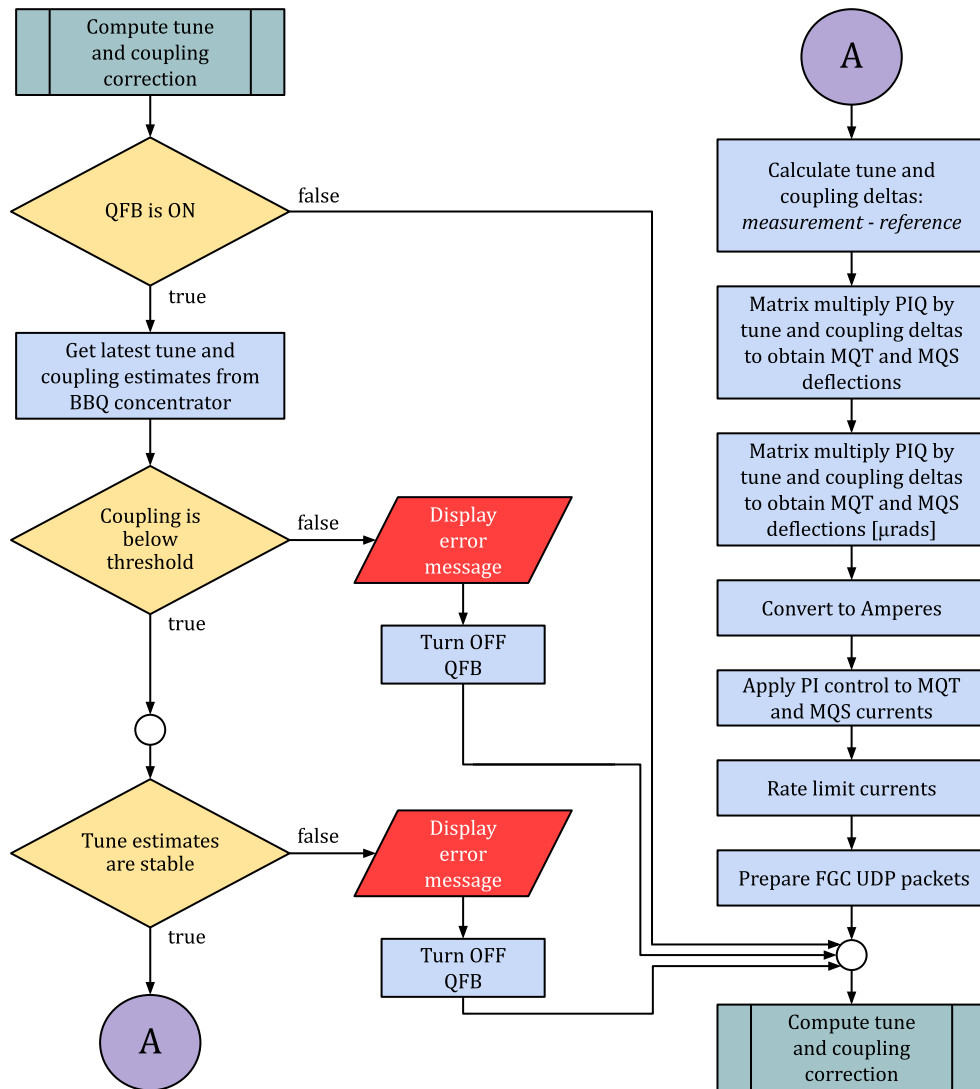


Figure 5.6: Continuation of Figure 5.5 showing the flowchart of the tune and coupling correction.

An overview of the process of orbit correction was given in Section 2.4. It is important to note that the required corrections to the COD deflections [ $\mu\text{rad}$ ] are calculated by multiplying the orbit error with the Pseudo-Inverse (PInv). Equation (5.1) is used to convert the calculated deflections to current:

$$I[\text{A}] = \frac{3 \cdot \theta[\text{rad}] \cdot p[\text{GeV}/c] \cdot I_{nom}[\text{A}]}{B \cdot L_{mag}[\text{Tm}]} \quad (5.1)$$

where  $p$  is the longitudinal momentum of the beam;  $I_{nom}$ ,  $B$  and  $L_{mag}$  are the nominal current, nominal magnetic field strength and the length of the dipole magnet, respectively; and  $\theta$  is the deflection. In the case that the calculated current corrections violate the maximum slew rate of the magnets, all the currents are scaled by a factor depending on the slowest magnet to act. Finally, the current corrections are prepared in UDP packets.

Finally, the Tune Feedback (QFB) follows in the `COMP` block, which calculates the corrections to the tuning quadrupole correctors. Figure 5.6 expands on the QFB. It can be seen that the QFB depends on the coupling and the tune estimate stability to be below a certain threshold to operate. The latter arose due to feedback instabilities incurred by the unstable tune estimates from the BQ algorithm. A similar procedure to the orbit corrections follows, where the current corrections to the quadrupoles are obtained by the Tune Pseudo-Inverse (QPI). The quadrupole currents are also slew rate limited and prepared in UDP packets.

After the `COMP` block, all the UDP packets prepared during the energy (RF frequency), orbit and tune feedbacks are sent through the LHC technical network. It is important to note that since the execution of the program is deterministic, all the UDP packets are synchronised with the OT BST signal as well, at 25 Hz. If the feedbacks are still left on at the end of a control loop, the next loop starts when the next OT signal is received in the `LISTEN OT` block.

## 5.2 Feasibility of Hardware Acceleration

### 5.2.1 Problem Definition

The BBFS was not internally equipped to detect failures in its sensors and actuators for the purpose of adjusting the beam optic Response Matrix (RM) models. As was previously mentioned, the failures were detected from the UDP packet status bits associated with a particular device. A failure in one of the magnets was more serious and the BBFS relied on the LHC operators on shift to detect that a magnet failure occurred. Then the operators calculated the new models off-line and uploaded them manually to the OFC through the OFSU.

One of the reasons why this functionality was not automated during the design of the BBFS had to do with the limited computing power available at the time. One Singular Value Decomposition (SVD) operation took in the order of  $1 \times 10^1$  seconds to execute. Considering that the controller operates at 25 Hz (40 ms) and that the OFC was designed for a deterministic execution, a full optics update would have been incompatible with the control loop if done sequentially.

Along the LHC cycle the optics change significantly and RM must be regularly updated to account for these changes. In practice up to 10 different RM matrices are used during a typical cycle. During ramp down or in preparation for injection, the SVD decomposition is applied to all RM matrices taking into account the known malfunctioning BPMs (typically 10-20 per beam and plane) and CODs (typically none). During the machine filling at injection all SVD decompositions are recomputed taking into account new malfunctioning elements. For 10 matrices the calculations required 2 to 3 minutes in the BBFS. Thus, the recalculation of the new PInv matrix through the SVD algorithm has proven to be a computational and operational bottleneck.

### 5.2.2 Proposed Solution

For the case when the COD is not in a critical location in the LHC, it was proposed that one of the heaviest computational elements in the OFC, the SVD optics inversion, could

be made faster by using Hardware Acceleration (HA). By calculating PInv from RM faster, optics model corrections can potentially be done on-line and help the operators save the beam in the event of a small corrector magnet malfunction. As discussed in Section 2.5, HA libraries allow the use of multi-core devices, such as modern-day conventional Central Processing Units (CPUs), much more efficiently through a careful and optimised use of the CPU cache memory. Moreover, such libraries facilitate the use of other devices such as Graphical Processing Units (GPUs), which have a massive number of computational units.

Techlab [47] provided access to machines equipped with GPUs as well as multi-threaded CPUs. Several machine configurations could be used to test the feasibility of using HA on the SVD. In particular, Table 5.1 lists the machines that were available to test SVD acceleration by HA libraries.

Table 5.1 clearly shows the difference between CPUs and GPUs. The number of cores and the amount of operations per second (GFLOPS) in GPUs is orders of magnitude higher than standard <10 core CPUs. The Intel Xeon Phi 7120P was a coprocessor board, designed to work within supercomputer clusters and was connected to the main board via PCIe. Intel Core i7-4790 and Intel Xeon E5-4650 had Hyper-threading and therefore 8 and 16 threads, respectively, could be executed at once. Intel Xeon Phi 7120P could execute 244 threads simultaneously

### 5.2.3 Benchmarking Tests

Benchmarking tests were done on each device, where the ArrayFire and the original implementation using ROOT objects of the SVD algorithm were tested. Specifically these tests focused on measuring the time it takes the device to compute the PInv of the RM, and finding the average execution time over many executions.

The benchmarking tests provided a good insight into the effect of hardware accelerated libraries on the computation time of the SVD algorithm. Figure 5.7 shows the average execution time of the SVD algorithm for different libraries, running on differ-

---

<sup>1</sup>From [105].

<sup>2</sup>From [106].

<sup>3</sup>From [107].

Processor name	Cores	GFLOPS (FP32)	
Intel Xeon E5-2640 2.5 GHz	6	9.3 <sup>1</sup>	
Intel Core i7-4790 3.6 GHz	4	13.1 <sup>1</sup>	
Intel Xeon E5-4650 2.7 GHz	8	12.9 <sup>1</sup>	
Intel Xeon Silver 4110 2.1 GHz	8	21.1 <sup>1</sup>	
Intel Xeon Phi 7120P 1.3 GHz	61	1,208 <sup>2</sup>	
Processor name	Cores	GFLOPS (FP32)	GFLOPS (FP64)
NVIDIA GTX1080Ti	3584	11,340 <sup>3</sup>	354.4 <sup>3</sup>
NVIDIA V100	5120	14,130 <sup>3</sup>	7,066 <sup>3</sup>

Table 5.1: Specifications of the Techlab machines available. Giga Floating-point Operations per Second (GFLOPS) quantify the processor performance in the order of  $1 \times 10^9$  32-bit floating point operations per second (FP32). The two GPUs also show the performance on 64-bit double floating point (FP64) precision.

ent machines. Note that it was not possible to use some libraries on specific machines, e.g. the ArrayFire CUDA library could not be used where there was no NVIDIA device. The ROOT and ArrayFire CPU libraries were executed on all machines. During their execution there is no GPU utilisation and instead their respective host CPUs are used. Table 5.2 shows the two GPUs used in this test and their respective host CPUs.

GPU	Host CPU
NVIDIA GTX1080Ti	Intel Xeon E5-2640 @ 2.5 GHz
NVIDIA Tesla V100	Intel Xeon Silver 4110 C@ 2.1 GHz

Table 5.2: Host CPUs of the two GPUs used for the benchmarking tests.

From the results shown on Figure 5.7, it is clear that the fastest libraries were the ArrayFire CPU and ArrayFire OpenCL which on average outperformed the traditional ROOT implementation by two orders of magnitude. The execution time on Intel Core i7-4790 was similar for both the ArrayFire CPU and OpenCL implementations however the small difference between the two execution times could not attest which one of the two libraries performs better. The execution times for both Intel Xeon Co-processors were also very similar to one another, however it is clear that the Xeon Phi 7120 outperformed the Xeon E5-4650 on all occasions. From the results below it can also

## SVD Benchmarking

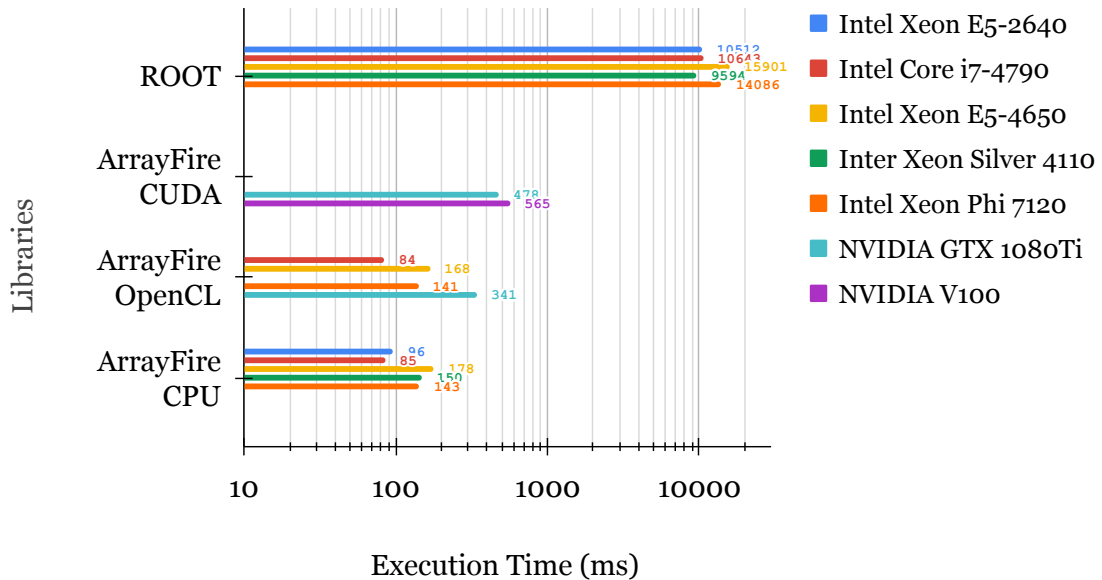


Figure 5.7: SVD execution time on different devices

be concluded that the ArrayFire CUDA implementation on both NVIDIA devices was slower than ArrayFire OpenCL and on average, approximately half an order of magnitude slower than the ArrayFire CPU implementation.

Another important conclusion drawn from this test was that the use of GPUs, namely the NVIDIA GeForce family: GTX1080Ti and the NVIDIA Tesla family: V100, is not suitable for the SVD calculation of the PInv of the LHC response matrix as all of the CPU machines outperformed them. The reason for the poor performance of the GPUs is that the speedup obtained from a massively parallel architecture is overshadowed by the time it takes to transfer data, to and from such devices. Should it have been the case where the SVD had to be performed on a much larger matrix, e.g. a high-definition image, the use of GPUs would have proved to be ideal. It is important to note that GPUs are mounted on the motherboard via a PCI port, which is inherently slower than the connection between the CPU and main memory. In addition, ArrayFire optimises the SVD algorithm through the use of the CPU cache, which is orders of magnitude faster than main memory [108].

### 5.2.4 SVD Accuracy statistics

This test addressed the hypothesis that different device architectures (namely in GPUs) would provide different results due to different optimisation techniques when applying HA the SVD algorithm. The different results might be caused by floating point rounding errors or different cut-off values used within the SVD algorithms themselves as they are implemented in each respective library.

A sample of the OFC beam optics used in LHC operation, containing a RM and its corresponding PInv, were obtained from the OFSU. The number of singular values used to create the PInv was also obtained. A test was devised which compared the PInv computed by the combination of libraries and devices available. In this test, the PInv matrices were multiplied with the original RM. Each multiplication obtained a Pseudo-Identity (PIId) matrix, where if the PInv was computed with all the singular values, PIId would be an identity matrix,  $I$ ; a zero matrix with diagonal values of one. Finally, the following calculation was performed:

$$\begin{aligned} \text{PIId} &\triangleq \text{PInv} \cdot \text{RM} \\ E &= |\text{PIId} - I| \end{aligned} \tag{5.2}$$

where  $E$  is the error analysed in this test. This calculation was also performed with the original PInv, and the results are annotated by OFSU.

Figure 5.8 shows that the average error obtained from each respective libraries is the same when executed on different machines. This suggests that the computation result of the PInv through the SVD algorithm is not hardware dependent. Figure 5.8 also shows a discrepancy in the error obtained from the OFSU and the ROOT libraries, however this should not be taken into consideration in this test. This discrepancy was due to calculation parameters which could not be perfectly matched from one library to the other. Such parameters include the number of singular values to be used in the calculation of the PInv as well as the cut-off value for such singular values, below which the singular value is zeroed out. In the BBFS, the SVD calculation was performed by user code built with ROOT libraries. Other custom changes in the SVD were made

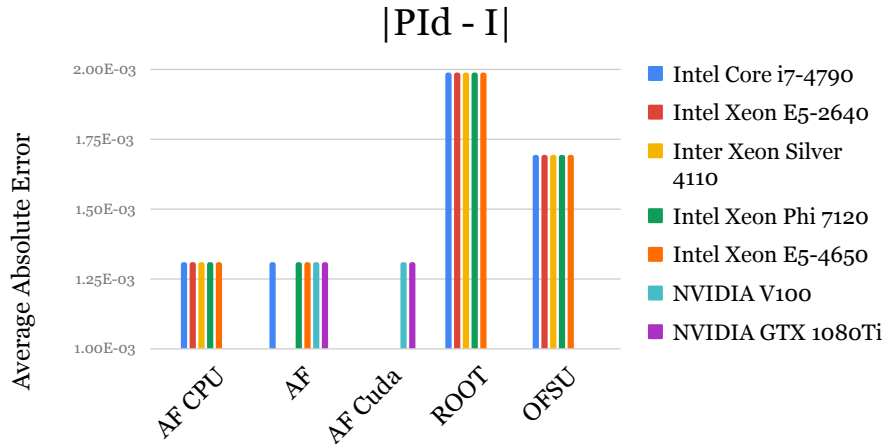


Figure 5.8: Average absolute error obtained from the different configurations of HA libraries and devices using Equation (5.2).

during the development of the BBFS to ensure singular values in descending order and to zero out singular values below a certain threshold. This created a truncated PInv, which could not be perfectly replicated with ArrayFire libraries.

The standard deviation of the error is shown on Figure 5.9 where it can be seen that this value is independent of the hardware where the computation is running on, as was the case for the average error. It can also be seen that the standard deviation of the error is approximately the same for all the libraries with which the PInv is calculated. This implies that the accuracy of the calculations is similar across libraries.

### 5.2.5 Memory and CPU usage

The final test performed focused on measuring the resource usage of the best libraries and machine illustrated in Figure 5.7; namely the ArrayFire OpenCL and ArrayFire CPU libraries executed on the Intel Core i7-4790 CPU. This was done by running the benchmarking test with one iteration of the PInv calculation and measuring the CPU usage as well as the memory consumption of the process. Then a python script was used to visualise the measured data from the running process.

The tests in Section 5.2.3 showed that CPUs are preferable to GPUs in terms of execution time. The use of ArrayFire OpenCL libraries is also not considered ideal.



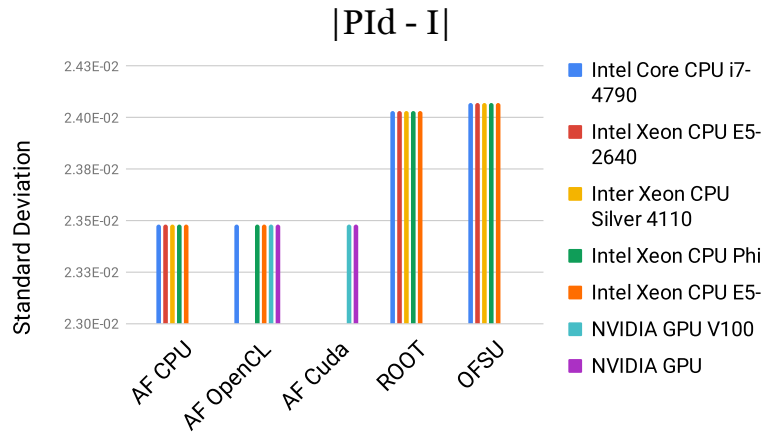


Figure 5.9: Standard deviation of the absolute error obtained from the different configurations of HA libraries and devices using Equation (5.2).

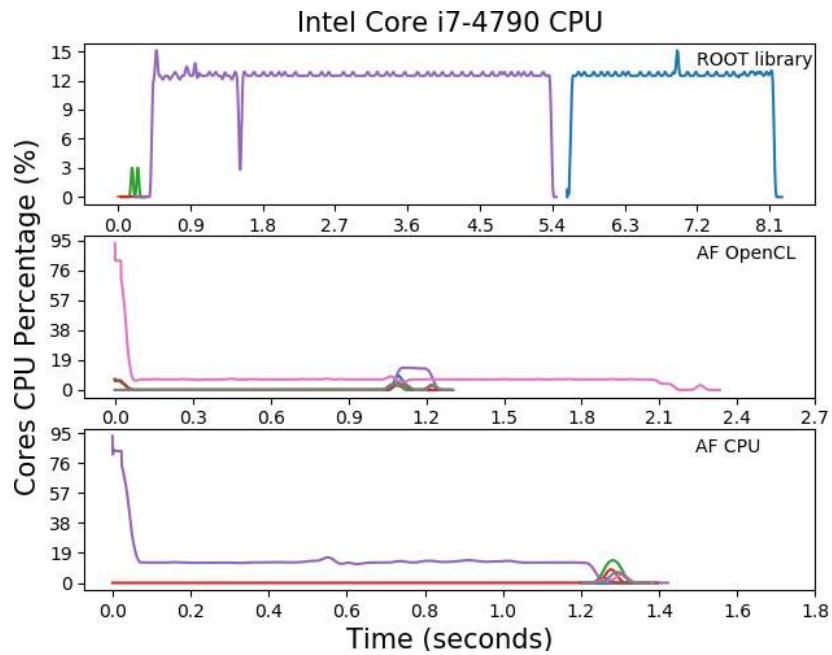


Figure 5.10: CPU usage percentage during SVD calculation using different libraries.

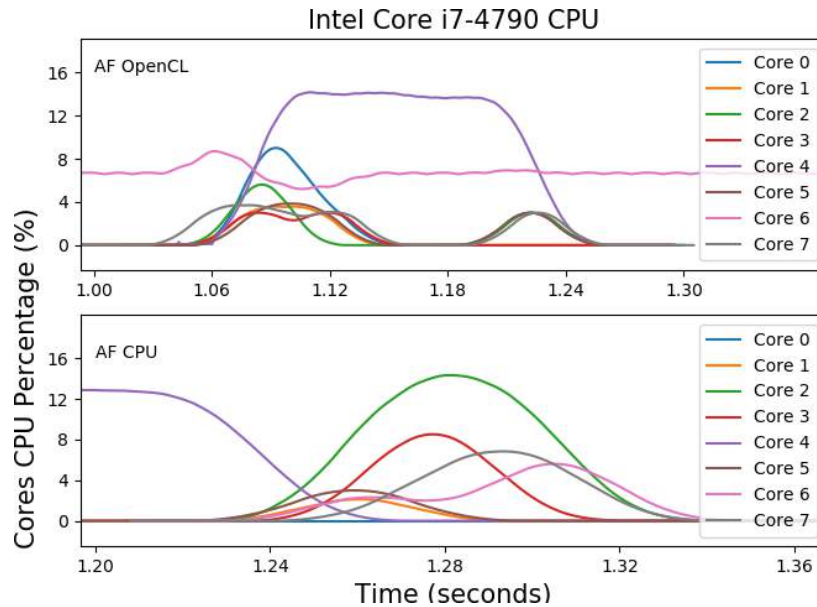


Figure 5.11: Details from Figure 5.10.

There were difficulties during the installation of OpenCL libraries on Intel Xeon E5-2640, which implies a risk of using OpenCL on operational machines. The simplest library to implement was ArrayFire CPU and coincidentally also obtained the best execution time on Intel Core i7-4790 CPU @ 3.60GHz.

The Intel Core i7-4790 CPU @ 3.60GHz and all the libraries except ArrayFire CUDA were chosen for this test. First was the implementation using the ROOT libraries, in the top plot of Figure 5.10. Multiple cores are being used throughout the execution of the program, however, they are not being used simultaneously at any point in execution.

The other plots in Figure 5.10 show the CPU usage of the ArraFire OpenCL and CPU libraries and Figure 5.11 shows an enlarged section. It can be seen that multiple threads are alive for approximately 80 ms, which is in accordance to the results shown in Figure 5.7.

Apart from the CPU utilisation, the machine's main memory (RAM) usage was also analysed. Figure 5.12 shows the memory usage for ROOT libraries (orange plot); ArrayFire using CPUs; and ArrayFire using OpenCL implementations respectively. The original implementation is quite standard, where after approximately 2 seconds, the response matrix is loaded and enough memory is allocated to hold the PInv after

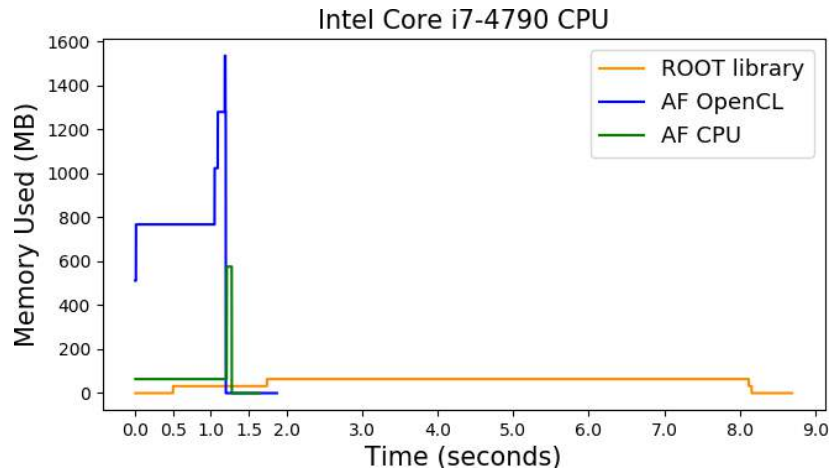


Figure 5.12: Memory consumption during SVD calculation using different libraries.

it is calculated. This memory consumption remains constant up until the end of the execution.

For ArrayFire CPU in Figure 5.12 (green plot), a different scenario can be seen, where after 1.2 seconds, the memory consumption increased by approximately 500 MB. This was due to ArrayFire copying the input response matrix over many threads, in order to parallelise the SVD. A similar approach can be seen for ArrayFire OpenCL (blue plot) which uses three times more memory.

This difference in memory consumption between ArrayFire’s CPU and OpenCL libraries can be attested to the specific programming paradigm used when opting to use OpenCL. However the latter was not further analysed due to it being proven to have a similar performance to the ArrayFire CPU implementation, yet consuming close to three times more memory.

### 5.2.6 Discussion

Three tests were performed which: a) compared the execution times of various libraries on different devices; b) checked the properties of the respective results; c) and analysed the hardware usage of various libraries on the best performing device in a). Both CUDA and OpenCL implementations were shown to be inadequate for the use on BFCLHC.

The use of GPUs was also shown to not be required. Since the hardware upgrade in

Long Shutdown 2 (LS2) introduced a 64-core CPU with 200 GB RAM, CPU parallelisation can be applied effectively to calculate multiple SVD calculations simultaneously. The time required to calculate 10 PInvs matrices is thus significantly reduced from  $1 \times 10^2$  seconds to the order of  $1 \times 10^0$  seconds.

### 5.3 Code renovation

The OFSU was the interface that the operators used during real operation in order to monitor the present status of the OFC, and changing parameters needed for operation. Two of the main problems with the OFSU were that it was bloated with unused functionality and black-box procedures which were required to change certain settings, e.g. fetching optics. The early design of the BBFS allowed access to more dedicated settings in the OFC, however, operator experience throughout the years has dictated which functionality to keep and improve, and which would become obsolete. Ultimately, the OFC had somewhat changed and became in discord with the OFSU. To avoid changing the major release of the BBFS during Run 2, any code renovations were delayed until LS2.

During LS2 it was decided to condense all the code in the BBFS, into one FESA-based application called BFCLHC. The hardware requirements were no longer an issue following a hardware upgrade of the OFC and OFSU machines. The hardware upgrade was from a 24-core, 32 GB RAM, 15 MB cache machine to a 64-core, 200 GB and 22MB cache machine [109]. This upgrade meant that the conservative measures taken in the initial design of the OFC could be relaxed. During LHC operation, the OFC never crashed due to insufficient hardware requirements, therefore, the hardware upgrade meant that it was safe to implement more advanced code and retain a deterministic execution.

At the same time, the FESA framework was undergoing development. With the introduction of new features, most of the basic operations done by the BBFS until Run 2 could be replaced by simple FESA commands. Some of the code used in the OFC and QFB was left intact: a) concentration of the COD, BPM and BBQ data;

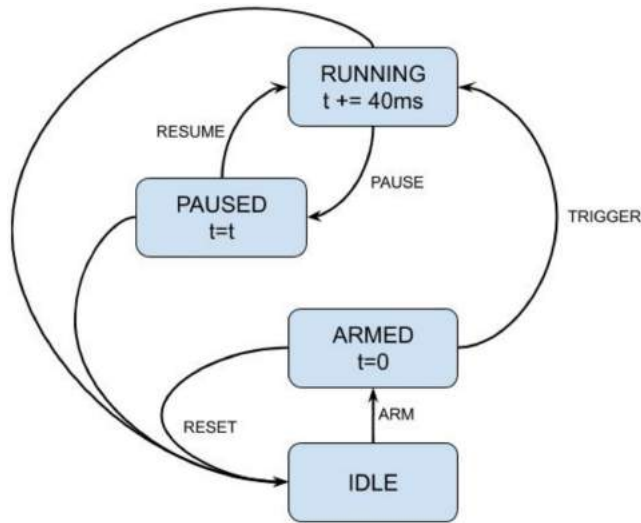


Figure 5.13: Function Player states (encircled) and events (connections) as implemented in BFCLHC.

b) computation of the corrections required on the CODs and the tuning quadrupoles;  
 c) Sending of the corrections to the respective gateways. The main program shown in Figure 5.1 and the OFSU FESA class, contained the majority of the code which was replaced. The FESA framework was used to re-implement the logic of the main program. One example is the relay of BPM orbit data from the OFC to the OFSU, which: a) underwent serialisation through ROOT TInterlink objects; b) sent via UDP to the OFSU; c) and finally set to the appropriate FESA property for orbit data.

To improve maintainability, the BFCLHC was designed to have minimal overhead user code by relying more on the FESA framework to implement the basic logic of the BFCLHC. For example, the data concentration is performed in a Real-Time Action (RTA), which updates the FESA device shared memory with the latest BPM orbit data. A standard FESA interface property is then set up for providing the most recent BPM orbit measurements directly from shared memory. Following this design approach, trivial tasks were easier to debug during implementation and to maintain in the future. Considering that back-compatibility is ensured by FESA, future versions of BFCLHC will be easier to maintain [110]. It is important to note that the design of the BFCLHC Application Programming Interface (API) was based on the API of the OFSU, however, it was adapted significantly to better fit the needs of the operators.

Function Players (FPs) was a new feature that was added to the BFCLHC, at the request of the LHC operators. Using FPs for automated settings control was introduced in [111]. In the BFCLHC, the FPs are used to automate the change of: a) reference values, e.g. reference orbit; b) control loop gains; and c) optic models. This will make it easier for the LHC operators to use more advanced optics in LHC operation. Presently, the BFCLHC can accept a list of reference values along with a list of time offsets when the references should be applied. Linear interpolation of reference values was also implemented, e.g. for the reference orbit. Each FP is used depending on its current state. Figure 5.13 shows the state machine of the FP as implemented in the BFCLHC. The state is stored in the FESA shared memory, and can be changed by a FESA command. To change the state of the FP, the operators must either manually send events, or synchronise them with the LHC timing. Some events are only available in specific states and a FESA exception is thrown when the user violates the state machine rules; e.g. when the FP is in the **ARMED** state, only a **TRIGGER** or **RESET** event can change the state. A function can only be set when the respective FP is in **IDLE** state.

The optics calculation procedures are now delegated to the BFCLHC FESA class. The approach to initialising and loading new optics to the feedbacks has therefore been changed. Any optics model must now be added to the BFCLHC memory via a FESA API set command. Subsequently, a FP must be used to instruct the feedbacks when to change to new optics. The operators rely on their own sub-routines which can obtain a set of optic models from LHC Software Architecture (LSA) settings database [112], and forward it to the BFCLHC.

Following the HA feasibility study in Section 5.2, it was determined that the use of GPUs did not improve computation times when calculating the PInv of the RMs when compared to multi-core CPU approach [113]. As a result, the multi-core CPU installed after the hardware upgrade was used for all calculations. In the new design, after an optics model is added, a new thread is spawned which calculates the RM and PInv. In the event of a sensor or magnet malfunction, a thread per optic model can be spawned to re-calculate the adjusted models without blocking the control loop. The

time to perform a full optics recalculation took in the order of minutes to complete on the BBFS. The BFCLHC reduces this time to the order of seconds, which makes it possible to attempt in real-time in the LHC Run 3.

## 5.4 BFCLHC Testing framework

The first formal testing of the OFC occurred at the start of the LHC Run 2 in 2014/15. A code renovation saw to the porting of the OFC and OFSU code from a 32-bit to a 64-bit architecture. Considering the maintenance and correct porting of more than 90,000 lines of code, BE-OP-LHC decided to build a separate testing framework, which mimicked the input signals of the OFC and OFSU by sending UDP packets disguised as coming from the BPMs. The original design of the OFC made this operation complex due to its global control loop architecture. A series of code changes had to be made to the BBFS simply to be able to test it. In particular, a global variable controlled key mechanisms within the OFC which determined whether the code was being used operationally or within a simulation. In addition, the output network ports were also tweaked in order to allow for a different base port to be used during testing. This was done as a protective measure against UDP packets actually being sent to the magnets during testing.

One of the main goals of the first testing framework was to verify that the renovated code respects all operational boundaries of the OFC. An identical hardware setup which was set up as a back-up to the operational hardware had to be used to run the tests. The tests were written in a Java-embedded Domain Specific Language (eDSL) and the JUnit framework was used to create the tests and generate reports. These design choices were prompted by need for non-programmer operators to easily write tests in a functional and intuitive approach. However, due to the inherent complexity of this testing framework it was quickly realised that it was difficult to maintain and to use efficiently. Despite these limitations, this testing framework was successful in detecting bugs which aided the code renovation of the BBFS prior to the LHC Run 2.

The new BFCLHC testing framework is still undergoing development, and will

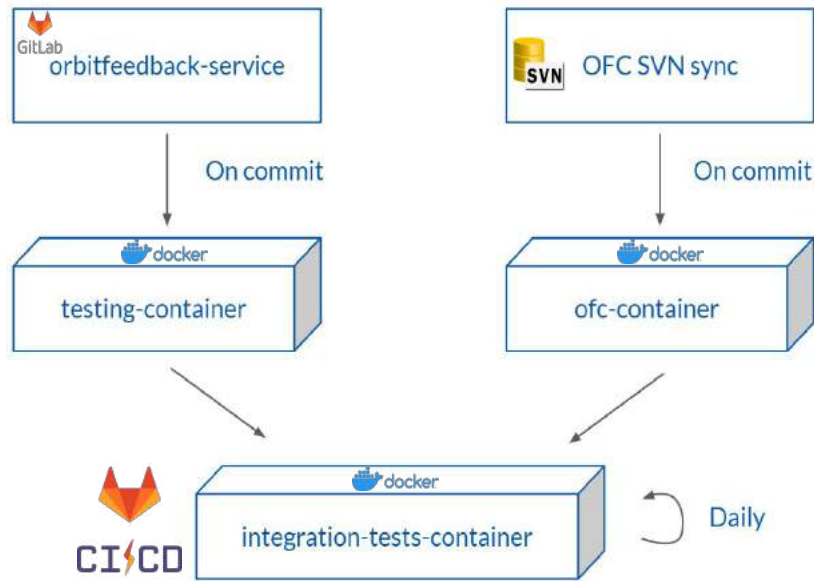


Figure 5.14: Testing framework schematic, showing the use of Docker containers in the GitLab Continuous Integration (CI) framework.

continue at least until the start of Run 3. Figure 5.14 shows the basic structure of the latest design. It was decided to make the testing framework hardware agnostic with the use of Docker containers [114]. This new design allowed tests to be performed on the BFCLHC during active development through the use of Continuous Integration (CI) tools in Gitlab [115]. As an example, Algorithm 1 shows the pseudo-code for a test that checks the limit on the number of optics that can be loaded to the BFCLHC. Figure 5.15 is a screenshot from GitLab CI which shows the results of the tests that were performed automatically after successful compilation and Docker image creation.

## 5.5 Summary

This chapter starts by providing more details on the design of the Beam-Based Feedback System (BBFS) in operation until the LHC Run 2. Section 5.1 uses flowcharts that capture the salient mechanisms of the BBFS, in particular, the concentration of beam-based measurement data and the calculation of the corrections to the corrector magnets currents.



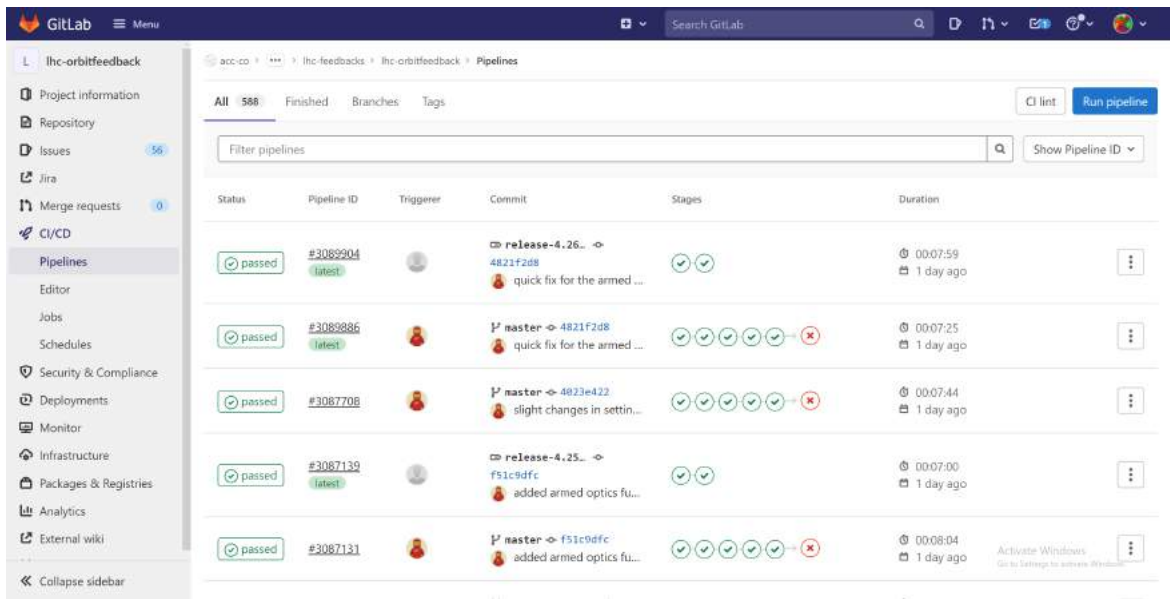


Figure 5.15: Screenshot of the BFCLHC testing framework test results using CI on GitLab.

---

**Algorithm 1** Test which asserts that only 30 optics can be loaded to BFCLHC.

---

```

optics_set                                ▷ Contains set of 30 optics
bfc                                         ▷ Running instance of BFCLHC
extra_optics                              ▷ 1 extra optics
res                                         ▷ Test result
for optics in optics_set do
  bfc.loadOptics(optics)
  if Allowable time expired then
    res ← Fail
    Exit
  else if not bfc.opticsLoadedSuccessfully() then
    res ← Fail
    Exit
  end if
end for
bfc.loadOptics(extra_optics)
if bfc.opticsLoadedSuccessfully() then
  res ← Fail
else
  res ← Success
end if

```

---

Section 5.2 defines and goes through the feasibility study of the application of Hardware Acceleration (HA) to mitigate a computational bottleneck in the BBFS due to the Singular Value Decomposition (SVD) of the Response Matrix (RM) matrices used in the Orbit Feedback Controller (OFC) to obtain the inverse dynamics. It was concluded that HA in the form of Graphical Processing Units (GPUs) would not improve the performance and thus, the hardware upgrade of the BBFS could be concluded in Long Shutdown 2 (LS2). A library called ArrayFire implements SVD with a more optimised use of the Central Processing Unit (CPU) cache memory was shown to reduce the computation time by two orders of magnitude.

Section 5.3 follows the code renovation done on BBFS and its redesign to the Beam Feedback Controller LHC (BFCLHC). The BFCLHC is a stand-alone FESA application which incorporates all the required features of the BBFS and the addition of new features requested by the LHC operators. In particular these are Function Players (FPs) which will be used during the LHC Run 3 to automate the change of specific settings in the beam-based controllers within the BFCLHC. Section 5.4 describes the hardware agnostic testing framework developed by BE-OP-LHC specifically to test BFCLHC during development. Trial runs performed during the LHC hardware commissioning in October 2021 show that the BFCLHC was successfully validated and performs as expected.

# Conclusions

The Beam-Based Feedback System (BBFS) was responsible for collecting real-time Beam Position Monitor (BPM) and Base-Band Q (tune) (BBQ) data, in order for the Orbit Feedback Controller (OFC) and the Tune Feedback (QFB) to calculate the required current correction to send to the magnets. This work was prompted by the need for better and improved functionality of the BBFS and took place during the LHC Long Shutdown 2 (LS2). Various aspects of the feedback systems were considerably upgraded and other studies were carried out which offer potential solutions to other problems within the BBFS during Run 2, e.g. alternative algorithms for tune estimation.

## Achieved Aims and Objectives

Various new techniques were developed to estimate the tune of the LHC more accurately, in the presence of 50 Hz harmonics. The methods developed in Section 3.3 provide a new algorithmic approach towards estimating the tune. This was done by first removing the frequency bins in the immediate vicinity of 50 Hz harmonics, and then interpolating over the spectrum with gaps. The best model trained in Section 3.4, ML-Refined, was trained by using a mixture of real BBQ and simulated spectra. It was demonstrated how a variant of Generative Adversarial Networks (GANs) called SimGAN, could be used to solve the problem of unlabelled datasets in an adversarial manner. Results in Section 3.5 show that ML-Refined can potentially obtain the best tune estimates.

A feasibility study was performed on the use of Hardware Acceleration (HA) in

the optics calculations. It was shown that Graphical Processing Units (GPUs) were not beneficial and that a modern Central Processing Unit (CPU) performs better on the matrix sized used in the BFCLHC. This ultimately lead to the final hardware architecture of the BFCLHC to be determined.

The feasibility study of the use of RL in the QFBEnv proved useful in determining the robustness of trained policies under various realistic scenarios. It was shown that RL has the potential to train policies which are more robust than the current feedback controller scheme present in BFCLHC. Preliminary tests have also shown state-of-the-art RL algorithms suffer from a lack of sample efficiency in the presence of large state-action spaces, e.g. in OFCEnv.

The new BFCLHC testing framework by BE-OP-LHC and the BFCLHC by SY-BI-SW were developed simultaneously. Initial tests and trial runs on the real machines provide successful preliminary results, where the BFCLHC behaved as expected, thus virtually the same as BBFS. BFCLHC also implemented new functionalities requested by the LHC operators, e.g. Function players which can automate setting changes. The handling of the optics calculations were delegated to the BFCLHC, allowing for parallelised calculation of the optics upon initialisation or request from a FESA command. The FESA API of BFCLHC was also significantly simplified, which meant that the BFCLHC can potentially be used by more operators if necessary, without the risk of black-box behaviour.

## Critique and Limitations

There were other potential solutions to improve the quality of the BBQ spectra. One was to increase the hardware acquisition rate of BBQ system, by which the BBQ spectra would have a higher frequency resolution. However, currently there are no plans to further analyse this change in the BBQ system settings until the end of beam commissioning in the LHC Run 3. The same effect can be achieved by performing the Fast Fourier Transform (FFT) on longer time signals to obtain BBQ spectra with a lower  $\Delta f$  between frequency bins.

The lack of variation in the refined spectra obtained by one trained SimGAN can be attributed to the relatively small amount of real BBQ data used in the Real dataset. It is possible that by using more examples of real BBQ spectra, one SimGAN would learn to add more types of artefacts in the refined spectra. Ultimately less SimGANs would have been required to build a robust dataset of refined spectra.

One major shortcoming of this work is the lack of testing and validation of the proposed upgrades on the LHC. In particular is the validation of the new tune estimation methods. Section 3.5 was limited to visual inspection to assess the accuracy of the tune estimates. It was also explained that the tune estimate stability plots could also be affected by 50 Hz harmonics. Therefore, the ultimate test can only be performed on-line.

The work in Chapter 4 was constricted to simulated environments, since the end of LS2 was delayed beyond the time available to perform the work on the LHC itself [116]. Previous work had already started to explore RL in the context of control in particle accelerators. However, experience has shown it to be difficult to deploy operationally beyond training and testing; e.g. the work of Kain et al. [91] on training RL for AWAKE (a plasma wake-field particle accelerator at CERN), required the development of a specialised FESA class which is capable of communicating with the Python scripts training the agent.

The Techlab machines used in Section 5.2.2 were also planned to be used to train RL on OFCEnv. However, when the work on RL started, the services offered by Techlab were no longer available. Therefore, the training was restricted by less powerful machines and obtained inconclusive results. The use of highly parallelised computing resources would have allowed for different types of RL to be attempted as well, e.g. multi-agent RL. Such algorithms are rather computationally expensive to train successfully, however, their effectiveness could not be analysed.

The last remark about the tests performed on the best policies trained on QFBEnv is that the actuator failure tests did not perfectly model magnet failures. It was explained that the purpose was to explore a worst case scenario and assess the behaviour of the policies under different environment conditions.

The code renovation in Chapter 5 mainly focused on the integration of all the functionality of the BBFS within one FESA class. The core code of the feedback systems was not changed in order to reduce the amount of new code that will be introduced at the start of Run 3. In order to make the BFCLHC more maintainable, it is proposed that a future version focuses on removing the dependence on ROOT libraries. The simpler architecture of the BFCLHC allows for an optimisation of the core code which uses faster linear algebra libraries.

During Run 2, a set of real-time COD, MQT, BPM and BBQ data were logged. However, it was found that some of the data was corrupted due to problems in the custom logging service. It is suggested that long-term real-time data is stored in LHC Run 3, to ensure a continuous development during technical stops. Previously, UDP packets were directly dumped to binary files since the hardware limitations posed a risk of any post-processing creating a bottle-neck. It was later found that some binary files were corrupted and that the logging service crashed often due to insufficient disk space on the machine running the logging service. Therefore, future real-time data logging systems should ideally prepare the data better prior to logging.

## Future Work

The attempts to design a better algorithmic approach for tune estimation in Section 3.3 will be tested in the LHC Run 3. In addition, it is proposed that the noise harmonic removal in the alternative approaches in Section 3.3 should have adjustable 50 Hz frequency rejection windows. In the presence of drifts in certain 50 Hz harmonics, expert operators can potentially obtain more stable tune estimates by fine tuning the position of individual harmonics.

Following the work done off-line in training RL agents on QFBEnv, is the implementation of a FESA class which can train an agent on-line. This can either be implemented alongside QFB within BFCLHC or in a separate FESA class which subscribes to the BFCLHC. The use of off-policy, or Model-Based (MB) RL normally allows the use of replay buffers which can be filled with real trajectories. Significant fail-safe

measures have to be taken within the FESA class responsible for training on-policy agents. The experiments performed on AWAKE consisted of an easier environment than QFBEnv, since AWAKE is tolerant to larger state excursions when acquiring new episode trajectories. A future implementation of any RL algorithm within the BFCLHC (developed in Section 5.3), needs to be equipped with proper fail-safe measures, e.g. automated early detection of large state excursions by using the linear beam optics model to predict the expected state excursion. The entire process of state and action sanitisation should be meant to avoid other machine protection systems from triggering. This can be achieved by playing episodes within a safe margin of operation and terminating any episode that obtains large state excursions. This implementation would also benefit from the experience and success achieved in applying RL in a particle accelerator setting, e.g. the work of Kain et.al [91] and Hirlander et al. [99]. The magnet cool-down periods also have to be respected as explained in the start of Section 4.1.3.

A more rigorous study will be conducted during Run 3 to improve the sample efficiency in RL in large continuous action spaces. The use of dimensionality reduction techniques, e.g. the latent space in VAEs, alongside the use of the attention mechanism in transformers [117], could solve some of the issues observed when training RL agents on OFCEnv. An RL algorithm which can train a policy that performs well on OFCEnv and which is trained with acceptable sample efficiency, can prove useful to other systems within the CERN complex which contain a large number of inputs and outputs. Considering that the main purpose of RL is behavioural training, by learning the behaviours of specific systems, an effective model is obtained simultaneously.

In theory, Model-Based (MB) RL promises better sample efficiency than Model-Free (MF) RL. Therefore, future work should also attempt other types of MBRL algorithms, e.g. ME-TRPO.

It is proposed that future work focuses as well on acquiring real-time COD, BPM and BBQ data. Information about the state visitation frequency can be obtained and might help the development of better RL algorithms for OFCEnv. Another use of the real-time data could be the modelling of beam orbit perturbation sources, that could

prove useful to making OFCEnv more realistic.

As mentioned in the previous section, the BFCLHC still relies on ROOT libraries. Future work should see the replacement of ROOT with standard linear algebra libraries. This would also be a major renovation task, however, it was planned that first, a series of unit tests should be written on the new BFCLHC testing framework and then the replacement code is written within a test.



# Glossary

**3D** Direct Diode Detection. vii, 20, 21

**AE-DYNA** Anchored Ensemble DYNA-style. 55, 64, 65, 110

**ALICE** A Large Ion Collider Experiment. 13

**AM** Amplitude Modulation. 20

**ANN** Artificial Neural Network. 42, 56, 57, 61, 68, 70, 111

**ATLAS** A Toroidal LHC ApparatuS. 13

**BBFS** Beam-Based Feedback System. iv, v, viii, 2, 3, 5–7, 10, 15, 29–31, 65, 69, 102, 105, 175, 177, 182, 186, 190, 195, 198, 199, 201, 202, 205

**BBQ** Base-Band Q (tune). v, 21, 24, 30, 32, 37, 67, 178, 182, 195, 202

**BFCLHC** Beam Feedback Controller LHC. xvii, 177, 194–199, 201

**BLM** Beam Loss Monitor. 63

**BPM** Beam Position Monitor. vii, 18–20, 30, 32, 33, 36, 172, 178, 182, 186, 195, 202

**CERN** European Organization for Nuclear Research. iv, 1, 30, 31

**CMS** Compact Muon Solenoid. 13

**CNN** Convolutional Neural Network. 48, 88

**COD** Orbit Corrector Dipole. 15, 30–36, 104, 105, 171, 172, 178, 185, 186, 195, 196

**CPU** Central Processing Unit. 38, 39, 187, 191, 201, 203

**DDPG** Deep Deterministic Policy Gradient. 55, 58, 61, 62

**DDQN** Double Deep Q-Network. 56, 57

**DL** Deep Learning. 122

**DPG** Deterministic Policy Gradient. 60

**DQN** Deep Q-Network. 55–57, 61

**DRL** Deep Reinforcement Learning. 55

**DS** Dispersion Suppressor. 12, 13

**EMA** Exponential Moving Average. 22, 38

**FEC** Front-End Computer. 30, 181

**FESA** Front-End Software Architecture. 31

**FFT** Fast Fourier Transform. 22, 203

**FGC** Function Generator/Controller. 30, 37, 178

**FP** Function Player. 197, 201

**GAN** Generative Adversarial Network. ix, 4, 7, 50, 68, 202

**GDL** Generative Deep Learning. 50

**GP** Gaussian Process. x–xii, 79, 81, 82, 84, 97, 99–101, 103

**GPGPU** General-Purpose Graphical Processing Unit. 39

**GPU** Graphical Processing Unit. iv, 38, 40, 109, 187, 191, 201, 203

**GUI** Graphical User Interface. 31

**HA** Hardware Acceleration. iv, xvii, 5–7, 38, 39, 65, 66, 109, 187, 190–192, 197, 201, 202

**IP** Interaction Point. 2, 13, 14

**IR** Insertion Region. 13, 14, 63, 104

**LHC** Large Hadron Collider. iv, 1–3, 7, 12, 18, 19, 29, 31, 33, 35–38, 63, 65

**LHCb** Large Hadron Collider beauty. 13

**LOF** Local Outlier Factor. 64

**LS2** Long Shutdown 2. v, 5, 67, 68, 75, 102, 104, 195, 201, 202

**LSS** Long Straight Section. 12, 13

**MAD-X** Methodical Accelerator Design. 3, 12

**MB** Model-Based. 54, 205, 206

**MC** Monte Carlo. 74

**MF** Model-Free. 54, 125, 175, 206

**MICADO** MINimisation des Carrés des Distortions d’Orbite. 32

**ML** Machine Learning. 4, 6, 7, 39, 63, 65, 85

**NAF** Normalised Advantage Functions. 58, 59

**NAF2** Normalised Advantage Functions with Double-Q Learning. 110

**NFQ** Neural Fitted Q Iteration. 55, 56

**OFC** Orbit Feedback Controller. xvi, 2, 3, 5, 18, 29–32, 34–38, 104, 171, 172, 176, 177, 179, 186, 190, 195, 198, 201, 202

**OFSU** Orbit Feedback Service Unit. 5, 29, 31, 177, 186, 190

**OT** Orbit Trigger. 182, 185

**PCA** Principal Component Analysis. 63

**PER** Prioritised Experience Replay. 64

**PG** Stochastic Policy Gradient. 57–59, 173

**PI** Proportional-Integral. v, xvi, 32, 34, 36, 69, 107, 108, 126, 159, 160, 172, 175

**PId** Pseudo-Identity. 33, 190

**PI<sub>inv</sub>** Pseudo-Inverse. 32–37, 172, 185–187, 189–191, 193, 195, 197

**PPO** Proximal Policy Optimisation. 55, 60, 109, 110, 114, 173, 175

**QFB** Tune Feedback. v, 2, 5, 6, 30, 31, 37, 68, 69, 102, 104, 106, 111, 114, 122, 175, 185, 195, 202

**QPI** Tune Pseudo-Inverse. 185

**QPI** Tune Pseudo-Inverse. 107, 175

**QRM** Tune Response Matrix. 106, 107, 175

**ReLU** Rectified Linear Unit. 86, 88, 111

**RF** Radio-Frequency. 12, 13, 178

**RL** Reinforcement Learning. v, 4–7, 52–57, 64, 65, 105–107, 110, 122, 125, 171, 175

**RM** Response Matrix. 3, 32–37, 104, 105, 171, 172, 186, 187, 190, 197, 201

**RMS** Root Mean Square. 32, 37

**RTA** Real-Time Action. 196

**SAC** Soft Actor-Critic. 62, 109, 110, 120

**SARSA** State-Action-Reward-State-Action. 53, 55, 56

**SIS** Software Interlock System. 37

**SL** Supervised Learning. 4, 40, 85

**SNR** Signal-to-Noise Ratio. 21

**SSS** Short Straight Section. 13

**SVD** Singular Value Decomposition. 7, 32, 33, 35, 37, 39, 106, 186, 190, 194, 201

**TCP** Transmission Control Protocol. 30, 178

**TD** Temporal Difference. 53, 55

**TD3** Twin-delayed Deep Deterministic Policy Gradient. 62, 109, 110, 116

**TRPO** Trust Region Policy Optimisation. 59

**UAT** Universal Approximation Theorem. 44

**UDP** User Datagram Protocol. 30, 36, 37, 178, 183

**UL** Unsupervised Learning. 64

**VAE** Variational Auto-Encoder. 50

**WMA** Weighted Moving Average. x–xii, 78, 80, 84, 99–101, 103

# Bibliography

- [1] R J Steinhagen. *LHC Beam Stability and Feedback Control-Orbit and Energy*. PhD thesis, RWTH Aachen U., September 2007.
- [2] European Organization for Nuclear Research, O Bruning, and European Council for Nuclear Research. *LHC Design Report, Volume I: The LHC Main Ring*. CERN, 2004.
- [3] Taking a closer look at LHC - magnetic dipoles. [https://www.lhc-closer.es/taking\\_a\\_closer\\_look\\_at\\_lhc/0.magnetic\\_dipoles](https://www.lhc-closer.es/taking_a_closer_look_at_lhc/0.magnetic_dipoles). Accessed: 2021-5-26.
- [4] S Baird. *Accelerators for pedestrians*. CERN, February 2007.
- [5] Claude Bovet, J P Papis, Hermann Schmickler, and L Vos. LHC BPM design. Technical report, CERN, December 1997.
- [6] Peter Forck, D Liakin, and P Kowina. Beam position monitors. In D Brandt, editor, *CERN Accelerator School: Beam Diagnostics*, pages 187–228. CERN, 2009.
- [7] Marek Gasior and Rhodri Jones. The principle and first results of betatron tune measurement by direct diode detection. Technical report, CERN, 2005.
- [8] L Grech, G Valentino, D Alves, M Gasior, S Jackson, R Jones, T Levens, and J Wenninger. An alternative processing algorithm for the tune measurement system in the LHC. In *Proceedings of the 9th International Beam Instrumentation Conference (IBIC 2020)*, 2020.

- [9] Leander Grech, Gianluca Valentino, and Diogo Alves. A machine learning approach for the tune estimation in the LHC. *Information*, 12(5):197, April 2021.
- [10] R S Sutton and A G Barto. *Reinforcement Learning - An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, 2018.
- [11] OpenAI. Part 2: Kinds of RL algorithms — spinning up documentation. [https://spinningup.openai.com/en/latest/spinningup/rl\\_intro2.html](https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html), November 2018. Accessed: 2021-5-27.
- [12] Gianluca Valentino, Roderik Bruce, Stefano Redaelli, Roberto Rossi, Panagiotis Theodoropoulos, and Sonja Jaster-Merz. Anomaly detection for beam loss maps in the large hadron collider. *IOP Conf. Series: Journal of Physics: Conf. Series*, 874(2017):01, 2017.
- [13] CERN. The large hadron collider. <https://home.cern/topics/large-hadron-collider>. Accessed: 2019-4-4.
- [14] Eleftherios Skordis, Francesco Cerutti, Vasilis Vlachoudis, Pablo Ortega, Roderik Bruce, Stefano Redaelli, Alessio Mereghetti, Pascal Hermes, Alfredo Ferrari, and Anton Lechner. Impact of beam losses in the LHC collimation regions. Technical Report CERN-ACC-2015-271, CERN, 2015.
- [15] Y Kadi, V Kain, B Goddard, and R Schmidt. Attenuation and emittance growth of 450 GeV and 7 TeV proton beams in Low-Z absorber elements. In *Accelerators and Storage Rings*, pages 581–583, 2004.
- [16] B Muratori and Werner Herr. Concept of luminosity. In D Brandt, editor, *CAS - CERN Accelerator School: Intermediate Course on Accelerator Physics*, pages 361–378. CERN, January 2006.
- [17] Jorg Wenninger. Orbit feedback at LHC. Civil Engineering Induced Vibration on the LHC, May 2013.
- [18] MAD - methodical accelerator design. <http://mad.web.cern.ch/mad/>. Accessed: 2019-6-26.

- [19] R J Steinhagen. Tune and chromaticity diagnostics. In D Brandt, editor, *CERN Accelerator School, Dourdan France, 28 May - 6 June 2008*, CAS, pages 317–359. CERN, 2009.
- [20] R J Steinhagen. Feedbacks on tune and chromaticity. In *8th European Workshop on Beam Diagnostics and Instrumentation for Particle Accelerators (DIPAC)*, pages 43–47, 2007.
- [21] T Persson and R Tomás. Improved control of the betatron coupling in the large hadron collider. *Phys. Rev. ST Accel. Beams*, 17(5):051004, May 2014.
- [22] A L Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, July 1959.
- [23] G W Hill. On the part of the motion of the lunar perigee which is a function of the mean motions of the sun and moon. *Acta Math.*, 8:1–36, 1886.
- [24] D W Kerst and R Serber. Electronic orbits in the induction accelerator. *Phys. Rev.*, 60(1):53–58, July 1941.
- [25] R Q Twiss and N H Frank. Orbital stability in a proton synchrotron. *Rev. Sci. Instrum.*, 20(1):1–17, January 1949.
- [26] Laurent Deniau, Hans Grote, Ghislain Roy, and Frank Schmidt. *The MAD-X Program - User's Reference Manual*. CERN, September 2020.
- [27] LHC report: full house for the LHC. <https://home.cern/news/news/accelerators/lhc-report-full-house-lhc>. Accessed: 2021-8-20.
- [28] M Gasior, R Jones, T Lefevre, H Schmickler, and K Wittenburg. Introduction to Beam Instrumentation and Diagnostics. In *CERN Accelerator School: Advanced Accelerator Physics Course*, December 2014.
- [29] Alfonso Benot Morell. Commissioning of the prototype stripline BPM system for the CLIC drive beam. International Conference on Accelerator Optimization (oPAC), 2015.



- [30] M Gasior and J L Gonzalez. Improving FFT frequency measurement resolution by parabolic and gaussian interpolation. Technical Report AB-Note-2004-021 BDI, CERN, February 2004.
- [31] Sofia Kostoglou, Gianluigi Arduini, Leandro Intelisano, Yannis Papaphilippou, and Guido Sterbini. Origin of the 50 hz harmonics in the transverse beam spectrum of the large hadron collider. February 2020.
- [32] Raul Murillo-Garcia, Quentin King, and Marc Magrans De Abril. Control of fast-pulsed power converters at CERN using a function generator controller. Technical Report CERN-ACC-2015-0128, CERN, October 2015.
- [33] CERN. ROOT a data analysis framework — ROOT a data analysis framework. <https://root.cern.ch/>. Accessed: 2018-8-15.
- [34] L K Jensen, J Wenninger, M Andersen, L Ponce, S Jackson, K Fuchsberger, and R J Steinhagen. Software architecture for the LHC Beam-Based feedback system at CERN. Technical Report CERN-ACC-2013-0257, CERN, November 2013.
- [35] CERN Beams Department. FESA – Front-End software architecture — be-dep-co.web.cern.ch. <https://be-dep-co.web.cern.ch/content/fesa>. Accessed: 2019-4-4.
- [36] Michel Arruat, Leandro Fernandez, Stephen Jackson, Frank Locci, Jean-Luc Nougaret, Maciej Peryt, Anastasiya Radeva, Maciej Sobczak, and Marc Vanden Eynden. Front-end software architecture. In *ICALEPCS07*, volume 7, pages 310–312, 2007.
- [37] J Wenninger and R Steinhagen. LHC orbit feedback control requirements. Technical report, CERN AB-OP, March 2007.
- [38] Jorg Wenninger. Personal communication, June 2018.
- [39] W Herr. Algorithms and procedures used in the orbit correction package COCU (closed orbit correction utilities). Technical Report SL-95-007, CERN, May 1995.

- [40] J DeFranza and D Gagliardi. *Introduction to linear algebra with applications*. Higher Education. Mc Graw Hill, 2009.
- [41] Jorg Wenninger. Personal communication, April 2019.
- [42] Ian Buck. The evolution of GPUs for general purpose computing, 2010.
- [43] Nick Congleton. AMD vs. nvidia GPUs: Who's winning the 2018 graphics war? – make tech easier. <https://www.maketecheasier.com/amd-vs-nvidia-who-is-king-of-gpus/>, October 2018. Accessed: 2019-1-28.
- [44] Tim Dettmers. Which GPU(s) to get for deep learning. <http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/>, November 2018. Accessed: 2019-1-28.
- [45] CUDA (compute unified device architecture) definition. <https://techterms.com/definition/cuda>, July 2015. Accessed: 2018-12-3.
- [46] OpenCL - the open standard for parallel programming of heterogeneous systems. <https://www.khronos.org/opencl/>, July 2013. Accessed: 2018-12-3.
- [47] Techlab. <https://techlab.web.cern.ch/>. Accessed: 2018-12-3.
- [48] NVIDIA. *NVIDIA Quadro vs. GeForce GPUs - Features and Benefits*. NVIDIA, 2003.
- [49] Mark Harris and View All Posts by. 12 things you should know about the tesla accelerated computing platform. <https://devblogs.nvidia.com/12-things-tesla-accelerated-computing-platform/>, November 2014. Accessed: 2018-12-3.
- [50] James G Malcolm, Pavan Yalamanchili, Chris McClanahan, Vishwanath Venugopalakrishnan, John Melonakos, and others. ArrayFire: a GPU acceleration platform. *Proceedings of SPIE - The International Society for Optical Engineering*, May 2012.

- [51] CUDA zone. <https://developer.nvidia.com/cuda-zone>, July 2017. Accessed: 2018-8-21.
- [52] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.
- [53] Victor Roman. Machine learning project: Predicting boston house prices with regression. <https://towardsdatascience.com/machine-learning-project-predicting-boston-house-prices-with-regression-b4e47> January 2019. Accessed: 2019-4-28.
- [54] Gianluca Valentino. Linear regression lecture, February 2017.
- [55] G Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.
- [56] Anastasis Kratsios. The universal approximation property. *Ann. Math. Artif. Intell.*, January 2021.
- [57] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [58] X Glorot, A Bordes, and Y Bengio. Deep sparse rectifier neural networks. In G Gordon and Dunson, D , Dudík,, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, AISTATS’2011, pages 315–323, 2011.
- [59] Yann LeCun, Patrick Haffner, Bottou Léon, and Yoshua Bengio. Object recognition with Gradient-Based learning. In David A Forsyth, Joseph L Mundy, Vito di Gesù, and Roberto Cipolla, editors, *Shape, Contour and Grouping in Computer Vision*, volume 1681 of *Lecture Notes in Computer Science*, pages 319–345. Springer,, Berlin, Heidelberg, 1999.
- [60] Grace W Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future. *J. Cogn. Neurosci.*, pages 1–15, February 2020.

- [61] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 1D convolutional neural networks and applications: A survey. *Mech. Syst. Signal Process.*, 151:107398, April 2021.
- [62] Chengping Rao and Yang Liu. Three-dimensional convolutional neural network (3D-CNN) for heterogeneous material homogenization. February 2020.
- [63] David Foster. *Generative Deep Learning*. O’Reilly Media, Inc., July 2019.
- [64] Diederik P Kingma and Max Welling. Auto-Encoding variational bayes. December 2013.
- [65] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger, editors, *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’14, pages 2672–2680, Cambridge, MA, USA, December 2014. MIT Press.
- [66] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2107–2116, 2017.
- [67] Sen Wang, Daoyuan Jia, and Xinshuo Weng. Deep reinforcement learning for autonomous driving. November 2018.
- [68] Antonio Loquercio and Davide Scaramuzza. Learning to control drones in natural environments: A survey. Technical report, NCCR-ROBOTICS, 2018.
- [69] Google. Google environmental report. Technical report, Google, December 2016.
- [70] Yuanlong Li, Yonggang Wen, Kyle Guan, and Dacheng Tao. Transforming cooling optimization for green data center via deep reinforcement learning. September 2017.

- [71] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.
- [72] Simon Hirlaender and Niky Bruchon. Model-free and bayesian ensembling model-based deep reinforcement learning for particle accelerator control demonstrated on the FERMI FEL. December 2020.
- [73] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. July 2017.
- [74] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.
- [75] 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. September 2015.
- [76] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, June 2013.
- [77] Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Piotr Trochim, Siqu Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm\_control: Software and tasks for continuous control. June 2020.

- [78] C Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, May 1989.
- [79] Martin Riedmiller. Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning, ECML’05*, pages 317–328, Berlin, Heidelberg, 2005. Springer-Verlag.
- [80] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI’16*, pages 2094–2100, Phoenix, Arizona, 2016. AAAI Press.
- [81] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S A Solla, T K Leen, and K Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.
- [82] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-Learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, June 2016.
- [83] John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust region policy optimization. February 2015.
- [84] J Achiam. Proximal policy optimization — spinning up documentation. <https://spinningup.openai.com/en/latest/algorithms/ppo.html>, 2018. Accessed: 2021-6-16.
- [85] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, January 2014.
- [86] Chris Yoon. Deep deterministic policy gradients explained – towards data science. <https://towardsdatascience.com/>

- deep-deterministic-policy-gradients-explained-2d94655a9b7b, March 2019. Accessed: 2019-7-11.
- [87] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in Actor-Critic methods. February 2018.
- [88] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy maximum entropy deep reinforcement learning with a stochastic actor. January 2018.
- [89] Auralee Edelen, Christopher Mayes, Daniel Bowring, Daniel Ratner, Andreas Adelmann, Rasmus Ischebeck, Jochem Snuverink, Ilya Agapov, Raimund Kammering, Jonathan Edelen, Ivan Bazarov, Gianluca Valentino, and Jorg Weninger. Opportunities in machine learning for particle accelerators. November 2018.
- [90] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. LOF: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000.
- [91] 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. *Phys. Rev. Accel. Beams*, 23(12):124801, December 2020.
- [92] K Matsuoka. Noise injection into inputs in back-propagation learning. *IEEE Trans. Syst. Man Cybern.*, 22(3):436–440, May 1992.
- [93] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. June 2016.
- [94] Leander Grech. rl-qfb git repository.
- [95] J G Ziegler and N B Nichols. Optimum settings for automatic controllers. *J. Dyn. Syst. Meas. Control*, 115(2B):220–222, 1942.
- [96] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

- [97] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines, 2017.
- [98] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. <https://github.com/hill-a/stable-baselines>, 2018.
- [99] Simon Hirländer. MathPhysSim/FERMI\_RL\_Paper: Preprint release, December 2020.
- [100] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for Large-Scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, November 2016. USENIX Association.
- [101] L Grech. RL\_OFC git repository.
- [102] John Schulman. The nuts and bolts of deep RL research. Deep RL Bootcamp Lecture 6, August 2017.
- [103] R. Brun and F. Rademakers. ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth. A*, 389:81–86, 1997.
- [104] Release 5.34/20 - 2014-08-13 — ROOT a data analysis framework. <https://root.cern.ch/content/release-53420>. Accessed: 2019-7-31.
- [105] PassMark software - CPU benchmarks. <https://www.cpubenchmark.net/>. Accessed: 2021-6-19.



- [106] Intel. Introducing the intel phi coprocessor: Architecture for discovery. <https://www.intel.com/content/dam/www/public/us/en/documents/presentation/xeon-phi-architecture-for-discovery-presentation.pdf>.
- [107] TechPowerUp. GPU specs database. <https://www.techpowerup.com/gpu-specs/>. Accessed: 2021-6-22.
- [108] David Levinthal. Performance analysis guide for intel® core™ i7 processor and intel® xeon™ 5500 processors. Technical report, Intel Corporation, 2009.
- [109] Rhodri Jones. BE-BI 2019: The year in review. BI Day 2019, 2019.
- [110] Frederic William Hogue. FESA quality assurance. 1st Developers@CERN Forum, September 2015.
- [111] Diogo Alves, Kajetan Fuchsberger, Stephen Jackson, and Jorg Wenninger. Test-driven software upgrade of the LHC beam-based feedback systems. In *2016 IEEE-NPSS Real Time Conference (RT)*. IEEE, June 2016.
- [112] C Roderick and R Billen. The LSA database to drive the accelerator settings. Technical Report CERN-ATS-2009-100, November 2009.
- [113] Leander Grech, Valentino Gianluca, Diogo Miguel Louro Alves, Stephen Jackson, and Jorg Wenninger. Feasibility of hardware acceleration in the LHC orbit feedback controller. Technical Report (accepted for publication), CERN, April 2019.
- [114] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239):2, March 2014.
- [115] GitLab CI/CD. <https://docs.gitlab.com/ee/ci/>. Accessed: 2021-8-31.
- [116] Anaïs Schaeffer. LS2 report: A new schedule. <https://home.cern/news/news/accelerators/ls2-report-new-schedule>, June 2020. Accessed: 2021-8-23.

[117] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. June 2017.