Study of machine learning methods for optimization and reliability improvements of high power linacs

Mathieu Debongnie

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE GRENOBLE ALPES

Spécialité : **Physique Subatomique et Astroparticules**

Arrêté ministériel : 25 mai 2016

Présentée par

Mathieu DEBONGNIE

Thèse dirigée par **Jean-Marie De Conto, Professeur, Laboratoire de Physique Subatomique et de Cosmologie**, et codirigée par **Frédéric Bouly, Chargé de recherche, Laboratoire de Physique Subatomique et de Cosmologie**

préparée au sein du **Laboratoire de Physique Subatomique et de Cosmologie**
dans **l'École Doctorale de Physique**

Etude de la modélisation des accélérateurs de particules par des méthodes de « Machine Learning » pour optimiser et fiabiliser l'opération d'un linac de forte puissance

Study of Machine Learning methods for optimization and reliability improvements of high power linacs

Thèse soutenue publiquement le **25 mars 2021**,
devant le jury composé de :

Monsieur Georges QUENOT
Directeur de recherche, LIG, Président
Monsieur David ROUSSEAU
Directeur de recherche, IJCLab, Rapporteur
Monsieur Nicolas DELERUE
Chargé de recherche, IJCLab, Rapporteur
Madame Annick BILLEBAUD
Directrice de recherche, LPSC, Membre
Monsieur Dirk VANDEPLASSCHE
Physicien, SCK•CEN, Membre
Monsieur Tomas JUNQUERA
Ingénieur de recherche, ACS, Membre

Remerciements

Tout d'abord, je tiens à remercier Frédéric Bouly et Jean-Marie De Conto pour m'avoir aidé à monter ce projet de thèse et m'avoir encadré tout du long. Un grand merci également pour tous les échanges enrichissants et la confiance qu'ils m'ont accordé.

Je tiens également à remercier ACS pour avoir accepté de s'occuper de la dimension financière de cette thèse. En particulier, un grand merci à Jean-Claude, Tomas, Daniel, Florian et bien sûr Olga.

Je remercie chaleureusement David Rousseau et Nicolas Delerue pour avoir accepté le rôle de rapporteurs. Un grand merci également à Annick billebaud, Georges Quénot, Dirk Vandeplassche et Tomas Junquera d'avoir accepté de prendre part au jury.

Un grand merci à Maud Baylac pour son accueil dans le service des accélérateurs du LPSC mais aussi pour avoir été d'une grande aide pendant la période de rédaction. Merci également à tous les membres du service pour tous les échanges agréables que l'on a eu.

Merci au SCK CEN de m'avoir permis d'accéder à l'injecteur de MYRRHA pour faire des manips. En particulier, merci à Dirk, Angélique, François, Jorik et Frank pour avoir rendu agréable mes passages sur la machine.

Merci à Didier Uriot et Nicolas Chauvin du CEA Saclay de m'avoir permis d'accéder à l'injecteur de IPHI pour faire des manips.

Je remercie également les doctorants du LPSC avec qui on a pu passer des moments très sympathiques au labo mais également en dehors. En particulier, merci Julien (merci pour la compagnie !), Julien l'autre, Lauraton, Toto (en vrai, il était post-doc mais restons tolérant !), Seb, Flora, Rola, Alex, Dilia... et bien d'autres. Et avant que je n'oublie, longue vie aux oranges !!!

Ce serait une grossière erreur de ma part si je ne remerciais pas aussi mes colocs. Merci Max, Lulu, Lau, Clémentin, Camille et (le petit dernier) Franck pour avoir été des colocs fantastiques. Sans eux, ces trois dernières années auraient été beaucoup moins agréables.

Et bien sûr, je remercie ma belle-mère Jana pour m'avoir poussé à me bouger le cul dans une période difficile pour moi. Sans ce coup de pouce, cette page de remerciements n'aurait jamais vu le jour. Je lui en serai éternellement reconnaissant.

Et maintenant, il est temps pour quelques hommages quelque peu obscurs... Je remercie R. Eric Reuss pour m'avoir permis de terroriser tant de colons ! Merci Didi,

Titi, Vivi et Mimi pour la running joke ! Je suis également reconnaissant aux anneaux de satourne pour toute la gomme avalée ensemble ainsi qu'à Adeline pour la piqure de rappel. Je n'oublie pas non plus les nombreux livreurs à bicyclette que je remercie de ne pas m'avoir laissé mourir de faim pendant la rédaction ! Et finalement, un grand merci aux meetupers pour avoir fait semblant d'essayer !

Contents

List of Figures

List of Tables

List of acronyms

AD	Antiproton Decelerator	p.11
ADS	Accelerator Driven System	p.1
AE	AutoEncoder	p.59
AI	Artificial Intelligence	p.56
CCD	Charge-Coupled Device	p.113
CCDTL	Cell-Coupled Drift Tube Linac	p.23
CERN	European Organization for Nuclear Research	p.9
CiADS	Chinese Accelerator Driven System	p.11
CNGS	CERN ν to Gran Sasso	p.11
CNN	Convolutional Neural Network	p.59
CTF3	CLIC Test Facility	p.11
CW	Continuous Wave	p.4
DC	Direct Current	p.5
DTL	Drift Tube Linac	p.21
ECR	Electron Cyclotron Resonance	p.16
EPICS	Experimental Physics and Industrial Control System	p.100
ESS	European Spallation Source	p.11
FEL	Free Electron Laser	p.77
GA	Genetic Algorithm	p.56
GAN	Generative Adversarial Network	p.59
GPU	Graphical Processing Unit	p.57
HEBT	High Energy Beam Transport line	p.16
HIPPI	High-Intensity Pulsed Power Injectors	p.23
IE	Injection Efficiency	p.73
IPAC	International Particle Accelerator Conference	p.73
IPHI	Injecteur de Protons à Haute Intensité	p.2
ISOL	Isotope Separation On-Line	p.87
ISOLDE	Isotope Separation On-Line DEvice	p.11
LANSCE	Los Alamos Neutron Science CEnter	p.70
LBE	Lead-Bismuth Eutectic	p.31
LEBT	Low Energy Beam Transport line	p.1
LEIR	Low Energy Ion Ring	p.11
LHC	Large Hadron Collider	p.9
linac	LINear ACcelerator	p.1
LT	LifeTime	p.73

MAE	Mean Absolute Error	p.77
MEBT	Medium Energy Beam Transfer line	p.15
MINERVA	MYRRHA Isotopes productioN coupling the linEar AcceleRator to the Versatile proton target fAcility	p.32
ML	Machine Learning	p.1
MLP	MultiLayer Perceptron	p.59
MOGA	MultiObjective Genetic Algorithm	p.70
MOPSO	MultiObjective Particle Swarm Optimization	p.69
MSE	Mean Squared Error	p.62
MSRE	Mean Squared Relative Error	p.119
MYRRHA	Multi-purpose hYbrid Research Reactor for High-tech Applications	p.1
NC	Normal Conducting	p.15
NSGA	Nondominated Sorting Genetic Algorithm	p.72
n-ToF	Neutrons Time of Flight	p.11
PIP-II	Proton Improvement Plan-II	p.75
PMT	Photo-Multiplier Tube	p.74
PPX	Residual Horizontal Oscillations in the Stored Beam	p.73
PS	Proton Synchrotron	p.10
PSB	Proton Synchrotron Booster	p.10
PSI	Paul Scherrer Institute	p.11
PSO	Particle Swarm Optimization	p.2
PXIE	PIP-II Injector Experiment	p.75
ReLU	Rectified Linear Unit	p.137
RFQ	Radio-Frequency Quadrupole	p.1
SC	SuperConducting	p.15
SCK CEN	StudieCentrum voor Kernergie Centre d'Etude de l'énergie Nucléaire	p.1
SCL	Side-Coupled Linac	p.21
SD	Standard Deviation	p.77
SILHI	Source d'Ions Légers à Haute Intensité	p.112
SNS	Spallation Neutron Source	p.11
SPL	Superconducting Proton Linac	p.11
SPS	Super Proton Synchrotron	p.10
SRF	Superconducting Radio-Frequency	p.4
tanh	hyperbolic tangent	p.137
TD	Transport D	p.70
TE	Transverse Electric	p.21

Physical constants

Speed of light in vacuum	c	$= 299\ 792\ 458$ m.s^{-1}
Electron mass	m_e	$= 9.109\ 383\ 701 \times 10^{-31}$ kg
Proton mass	m_p	$= 1.672\ 621\ 923 \times 10^{-27}$ kg
Vacuum permittivity	ϵ_0	$= 8.854\ 187\ 817 \times 10^{-12}$ A^2.s^4.kg.m^{-3}
Vacuum permeability	μ_0	$= 4\pi \times 10^{-7}$ kg.m.A^{-2}.s^{-2}
Elementary charge	e	$= 1.602\ 176\ 634 \times 10^{-19}$ A.s
Electronvolt	eV	$= 1.602\ 176\ 634 \times 10^{-19}$ kg.m^2.s^{-2}

Introduction

Nowadays, there is a general trend towards an increase in the performance of particle accelerators. To satisfy the requirements, whether it is of academic or industrial origin, the future accelerators have to meet higher mean power, reliability and stability. This is especially true for ADS (Accelerator Driven System) projects that aim to drive a nuclear reactor with a particle accelerator. These projects require the construction of proton accelerators with a mean power of the order of a few megawatts and with extremely high reliability to efficiently incinerate nuclear waste. This is the case for the MYRRHA (Multi-purpose hYbrid Research Reactor for High-tech Applications) project led by the Belgian research center on nuclear energy (SCK CEN). In this project, the aim is to use an accelerator that would provide a 4 mA continuous proton beam at 600 MeV to drive a sub-critical nuclear reactor of about 100 MW_{th}. Such a beam would have a mean power of 2.4 MW. To achieve this goal, the design and construction of a superconducting linear accelerator (linac) are currently underway. This accelerator will have to be extremely reliable because it will have to be operable over 3-months cycles with less than 10 beam trips longer than 3 seconds. Indeed, too long and too frequent beam trips can generate thermal stress on the reactor structure and thus can damage it. However, such a reliability requirement represents a level never achieved before. Hence, the design of the accelerator has to be robust, flexible and modular enough to be able to compensate for the failure of some of its components with minimal interruption of the beam.

One of the key points to achieve this goal is to ensure a good configuration of the injector composed of a Low Energy Beam Transport line (LEBT) and a Radio-Frequency Quadrupole (RFQ). The role of the LEBT is to guide and focus the beam from the ion source to the RFQ, the first accelerating element. This role is notably crucial to obtain good quality beams that can be transported along the linac while minimizing the beam losses that can force the shutdown of the machine if they exceed the tolerance level on beam losses (typically 1 W/m for high power proton accelerators). However, the beam dynamics in an injector are complex due to the space charge effects and their compensation. Besides, it is necessary to be able to switch quickly the configuration of the injector over a wide range of configurations (low current and duty cycle for the startup up to the nominal beam current and duty cycle).

In this context, this thesis goal is to explore the possibility of using Machine Learning (ML) methods to develop a numerical model able to reproduce accurately the experimental behavior of an injector and especially a LEBT. This is done to develop a fast and reliable system control for the accelerator to help meet the reliability require-

ments. In particular, this manuscript describes the research undertaken since 2018 at the Laboratoire de Physique Subatomique et de Cosmologie in Grenoble over the training of artificial neural networks under supervised learning to model the MYRRHA and IPHI (Injecteur de Protons à Haute Intensité, CEA Saclay) injectors.

The manuscript is organized as follows:

- the first chapter aims to introduce the context and the global problem that motivated this thesis work. As such, it contains an overview of the particle accelerator technologies, the description of the MYRRHA project and the introduction of the basics of beam dynamics.

- The second chapter aims at introducing the principle of neural networks, ML as well as an optimization algorithm named Particle Swarm Optimization (PSO). Also, a short review of the literature is presented about the use of PSO and neural networks in the field of particle accelerators.

- The third chapter is dedicated to the description of the commissioning and tuning studies performed as part of this thesis on the MYRRHA and IPHI injectors. Moreover, a test of the PSO algorithm on the MYRRHA LEBT is also discussed.

- The fourth chapter is a description of the experimental and simulated datasets gathered to train neural networks to model the MYRRHA and IPHI injectors as well as to evaluate their performances in these tasks. Besides, it contains a description of the process used to determine some reasonable hyperparameters for the networks and the supervised learning algorithm.

- The final chapter is dedicated to the discussion of the training and evaluation of neural networks. The aim of the trained models is to reproduce the general behavior of real injectors. This means that, given a configuration of the LEBT, the network has to predict the beam properties at the exit of the injector. This is done so that optimization algorithms such as PSO can be applied to trained models as surrogates for real machines to speed up the optimization process.

Chapter 1

High power hadron linacs: performances, reliability and ADS principle

Contents

There is presently a growing demand for high-power hadron accelerators to better support various fields of science like particle physics, nuclear physics or neutron-based physics. These applications typically require beams with very high mean power ("MegaWatt Class") which goes significantly beyond the present capability of most existing facilities: the required beam energy is in the MeV to GeV range while the required peak beam current varies from 1 to hundreds of mA.

Also, reliability is a major challenge within the perspective of improving the performances and sustainability of these MegaWatt class accelerators. Indeed, the availability requirements are becoming increasingly challenging to optimize the operational costs of such accelerators. These requirements are even more stringent for an ADS. As an example, for the MYRRHA ADS demonstrator, the actual availability limit sets a maximum of 10 beam interruptions (longer than 3 seconds) over a 3-month operating cycle.

So, to minimize the overall power consumption and thus decrease the operating costs, the use of Superconducting RF (SRF) accelerating cavities is becoming mandatory. This is especially true for machines that require CW (continuous wave) operation or a high duty cycle. The architecture of most of these new machines is therefore based on a SRF linac.

Experience showed that the failure rate of high power linacs is dominated by the reliability of the serial RF units and also by the injector where the beam is produced, bunched and pre-accelerated. It is therefore necessary to design linacs with redundant elements able to survive a certain number of failures without any significant beam interruptions. But it is also of great relevance to improve the tuning methods of the injectors to avoid halo formation and beam losses in the high power sections of accelerators. Besides, operation with different modes (variable duty cycle and peak current) is often required from the commissioning period to user operations and these modes are determined by the tuning of the injector. Hence, improving the tuning process of injectors will also increase the beam availability to reduce the time to switch from one operation mode to another.

In this chapter, the main criteria to evaluate high power accelerator performances will be reviewed. The reliability requirements of machines currently operated or being developed will be commented on. In particular, the requirements for ADS operation and the MYRRHA project will be introduced. The general structure of high-power linacs will be reminded as well as the basics of charged particle beam dynamics for linacs injector. Finally, the resultant problems that motivated this thesis will be exposed.

1.1 Principle of particle accelerators: Introduction and a brief overview

Electrostatic acceleration

In 1897, the electron is discovered by J. J. Thompson. He experimentally showed that a particle beam is emitted when exciting a low pressure gas in a tube with electrodes at its extremities. He also showed that this beam could be deflected with the use of a magnetic field i.e. that the beam consists of charged particles under the influence of the Lorentz force:

$$\frac{d\vec{p}}{dt} = q\left(\vec{E} + \vec{v} \wedge \vec{B}\right), \tag{1.1}$$

where:

- $\vec{v}$ is the particle speed (m/s),
- $\vec{p} = m\vec{v}$ is the momentum of the particle (kg.m/s),
- m is the particle rest mass (kg),
- q is the particle charge (C),
- $\vec{E}$ is the electric field (V/m) and
- $\vec{B}$ is the magnetic field (T).

The total energy E_{tot} of a relativistic particle with regard to its momentum is:

$$E_{tot}^2 = p^2c^2 + m^2c^4, \tag{1.2}$$

where c is the speed of light in vacuum in m/s.

By projecting the last equation in the direction of the momentum, one can show that the energy gain of a particle is due to the influence of the electric field only:

$$\vec{p} \cdot \frac{d\vec{p}}{dt} = \frac{1}{2}\frac{dp^2}{dt} = \frac{1}{2c^2}\frac{dE_{tot}^2}{dt} = qm\vec{v} \cdot \left(\vec{E} + \vec{v} \wedge \vec{B}\right) \tag{1.3}$$

$$\Rightarrow \frac{dE_{tot}}{dt} = q\vec{E} \cdot \vec{v}. \tag{1.4}$$

Hence, an electric field parallel to the particle speed is the best way to accelerate the particle.

The electrostatic accelerators are based on this consideration: several electrodes with increasing voltages are placed along the accelerator to create a constant electric field and accelerate particles. The Cockroft-Walton, the Van de Graaff (cf. Figure 1.1) and the Tandems are Direct Current (DC) accelerators using this structure. The

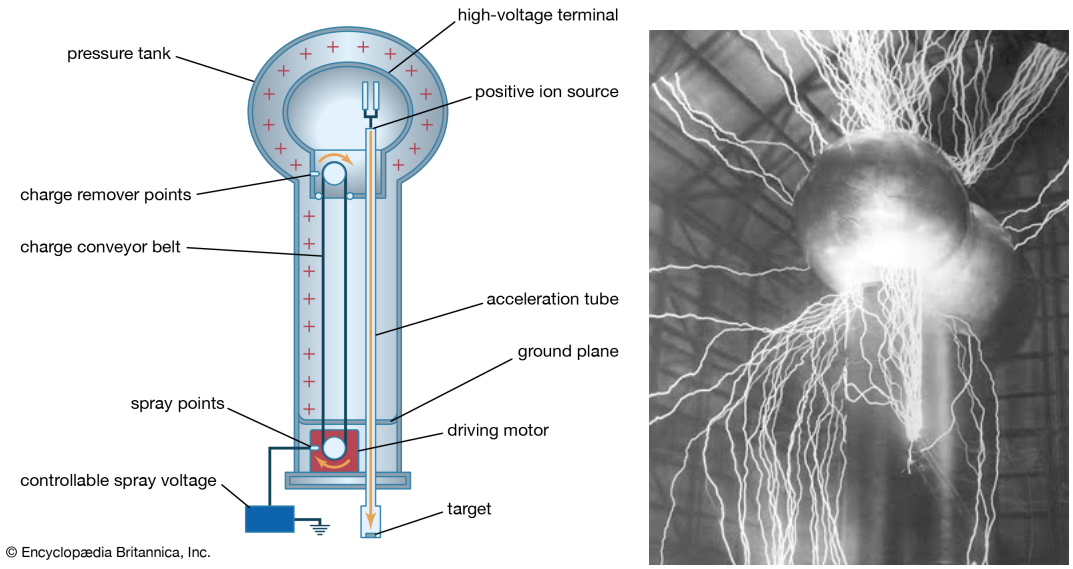

Figure 1.1: Diagram of a Van de Graaff accelerator and a working implementation [1].

energy gain of the particles is directly proportional to the voltage between electrodes which is limited by electrical breakdowns appearing at best at tens of megavolts.

Nowadays, these accelerator types are limited to low energy ion acceleration or used as injectors for RF accelerators using electromagnetic structures. Note that the beam obtained with these DC machines is continuous.

Electromagnetic structure acceleration

In 1924, G. Ising was the first to propose that it would be better to accelerate particles through a succession of modest accelerations instead of a single accelerating gap. The design would therefore consist of several gaps along the path of the particles applying pulsed voltages generated by a spark chamber to obtain acceleration. This implies that the particles are bunched and their arrival in each gap is synchronized with the phase of the applied voltage to encounter an accelerating field.

In 1928, Wideröe successfully implements this new idea by the application of an alternative voltage of 25 kV at 1 Mhz to drift tubes separated by grounded drift tubes (cf. Figure 1.2 a) [2]. With this setup, he managed to accelerate electrons to 50 keV. However, this implementation is strongly limited on the maximum reachable energy because in order to maintain the correct synchronicity between the phase in each gap and the particles with increasing energy either the drift tube lengths should be increased or the frequency of the alternating voltage should be increased. But at a higher frequency, the setup starts to act as a large capacitor which is subject to high power losses. With the drift tubes seen as a capacitor, the intensity of the displacement current required by the tubes can be written as:

$$I = \omega C V, \tag{1.5}$$

with V the voltage amplitude (V), C the capacitance of the accelerating gaps (F) and ω the angular frequency (rad/s).

At high frequency, the dimensions of the electrodes and supporting stems would function like resonant antennas with high power losses. The electromagnetic radiation is also very high ($P = VI = \omega CV^2$). To limit this power, it is possible to surround the space between two tubes leading to the introduction of better adapted RF accelerating structure (cavities). This setup can be considered as a "mono-gap" cavity (cf. Figure 1.2 b). These cavities can be paired and if the phase is identical in two contiguous cavities ("0 mode" or "2π mode") the resulting current in the separating wall is zero. This allows building resonant cavities that consist of "drift tubes" separated by gaps with alternating current at identical phases (cf. Figure 1.2 c). Such a structure was developed by Alvarez in 1945 and allowed to build an accelerator able to accelerate protons up to 32 MeV.

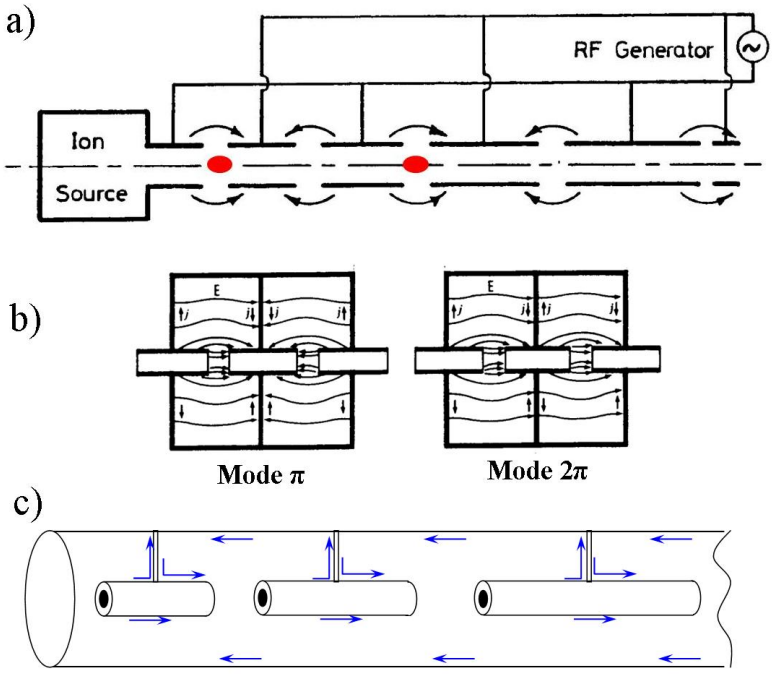

Figure 1.2: a) Principle of a Wideröe drift tube RF linac. When particles enter an accelerating gap they encounter an electric field in the correct direction for acceleration (dependent on their charge). To ensure that the field in each gap has the right phase, the time of flight inside the tubes has to be equal to the half period of the alternating voltage. Hence, as the particles velocity increases, the length of the tubes also increases. b) "Mono-gap cavities" in π and 2π modes. c) Principle of the Alvarez accelerating structure. The arrows show the displacement of charges in the structure at a given instant.

This accelerating principle is utilized in most current accelerator designs as it is the most efficient method to reach very high energies. Electromagnetic accelerators are either linear or circular and can be divided in three main categories:

- Linear accelerator

This type of accelerator is based on the drift tube principle (at least for the first few meters of acceleration) and is able to provide high intensity beams at a high duty cycle. Particles are accelerated along a straight path hence the accelerator is mainly limited by its size. Linacs are often used as the first accelerating structure before the injection of the beam into a synchrotron. The case of high-power linacs is discussed in more detail in Section 1.3.

- Cyclotron

This type of accelerator uses the effect of a constant magnetic field orthogonal to the particle speed which curves the trajectory of the beam [3]. First introduced and demonstrated in 1930 by E. O. Lawrence and M. S. Livingstone, the principle used two semi-circular hollow electrodes separated by a constant gap placed in the magnetic field induced by an electromagnet (cf. Figure 1.3). The particles injected in the center are accelerated each time they pass through the gap and with each acceleration the radius of curvature increases. Hence the trajectory of the particles is akin to a spiral until they are extracted. The extraction is performed by deviating the particles from their circular orbit by applying a voltage to a plate near a window. This process is prone to generate losses. In contrast with linacs, cyclotrons have the advantage to be compact and are used in many industries. Notably, they are utilized as sources for proton therapy in medical treatments. They can generate continuous beams up to a few mA. However, they are limited by the synchronicity issues to reach high energies.

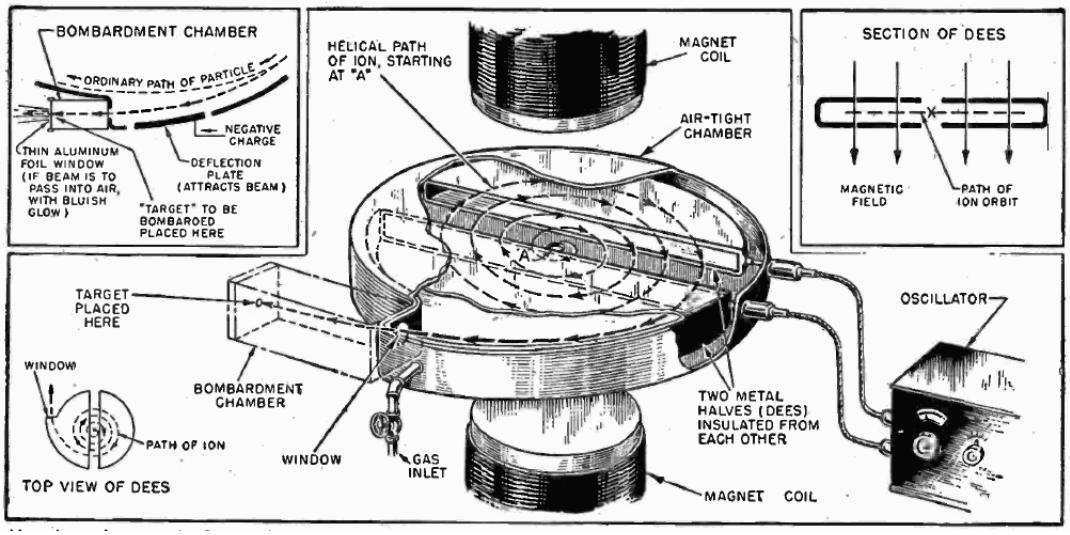

Figure 1.3: Diagram of a cyclotron. [4]

- Synchrotron

This type of accelerator uses a circular design where the particles are kept (or

accumulated) along a circular path thanks to a magnetic field that increases with the beam energy. Only a single accelerating component is required to accelerate the beam as many times as needed. Hence synchrotrons are the accelerator type able to reach the highest energies with intense beams. Their size is related to the beam energy, the particle type and the strength of the magnetic dipole used to bend the trajectory of the particles ($B\rho = p/q$). They are often used as storage rings for various applications. In particular, they can be used for light sources such as Soleil near Paris [5] and particle physics research such as the Large Hadron Collider (LHC) at CERN (European Organization for Nuclear Research) [6].

1.2 Accelerators performances

1.2.1 High mean power: energy and beam current

For many different fields of fundamental or applied physics, the growing demand for ever more efficient hadron accelerators can be illustrated by the Livingston plot (cf. Figure 1.4) [7, 8]. Indeed, the first criterion for evaluating the performance of an accelerator is beam energy. In Figure 1.4, one can see that through the past century the energy provided by hadron accelerators has (almost) continually increased up to the LHC.

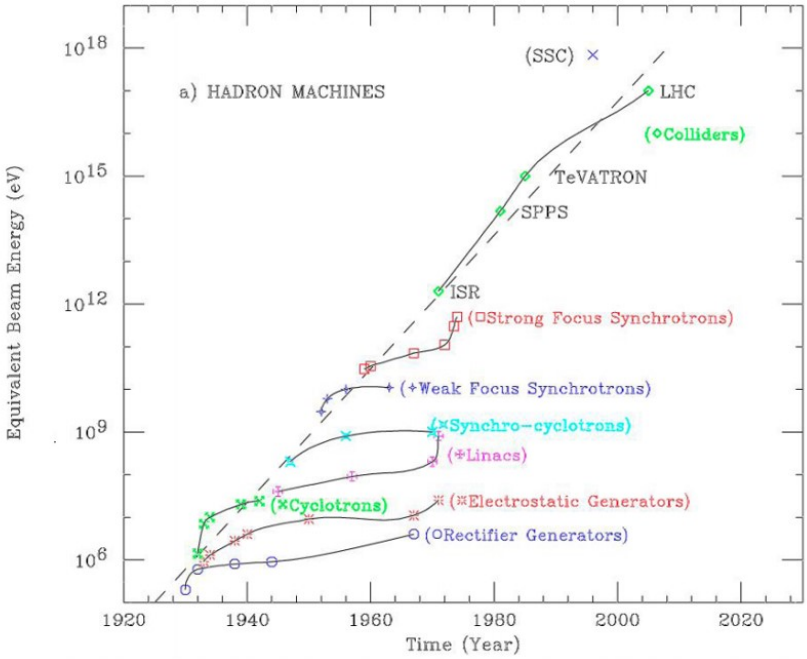

Figure 1.4: The Livingston diagram. [7, 8]

A schematic view of the LHC and the CERN accelerator complex is shown in Figure 1.5 [9]. It is important to note that, to accelerate a hadron beam up to 7 TeV in the LHC ring, several accelerators are used successively. At the beginning of the chain, a linac (LINAC 4) accelerates a H^- beam up to 160 MeV (with a peak current of ∼30 mA, a pulse length between 400 μs to 600 μs, and a 1 Hz repetition rate) [10]. Then the beam is injected in the BOOSTER or Proton Synchrotron Booster (PSB) which is composed of four superimposed rings with a radius of 25 meters. A stripping foil at the PSB injection point will strip the electrons off the hydrogen anions, thus creating a proton beam. This beam is split up vertically into four different beams by pulsed magnets which successively deflect parts of the incoming beam to different angles. Protons are then accumulated in the four rings and the energy is ramped up to 1.4 GeV. These proton bunches are then similarly recombined at the exit of the PSB and further transferred down to the Proton Synchrotron (PS) to be accelerated up to 45 GeV. Then the beam is sent to the Super Proton Synchrotron (SPS) and

accelerated to 450 GeV before being injected into the LHC. At the end of this complex injector chain, two proton beams are accelerated up to 6.5 TeV and stored. The beam current stored in the LHC ring is 0.58 mA [11] which corresponds to a stored energy of ∼340 MJ per beam. When a beam has to be dumped an "instantaneous" beam power of ∼3.8 TW has to be absorbed.

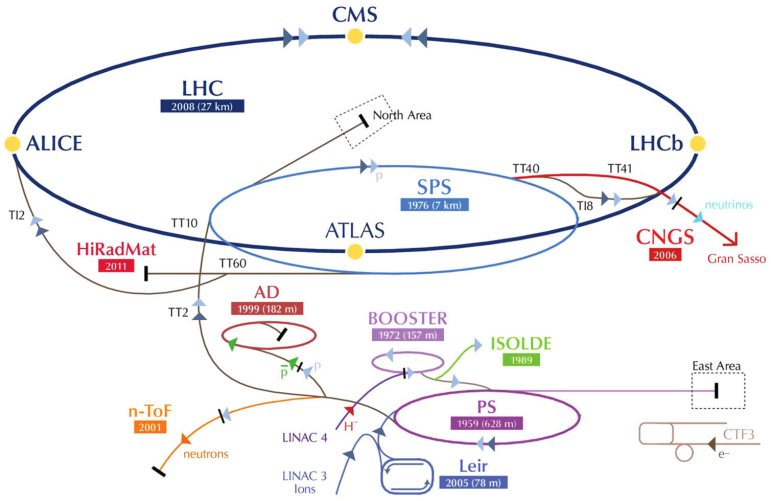

Figure 1.5: CERN accelerator complex with Large Hadron Collider (LHC), Super Proton Synchrotron (SPS), Proton Synchrotron (PS), Antiproton Decelerator (AD), Low Energy Ion Ring (LEIR), Linear Accelerators (LINAC), CLIC Test Facility (CTF3), CERN ν to Gran Sasso (CNGS), Isotopes Separation on Line (ISOLDE) and neutrons Time of Flight (n-ToF). [9]

Nevertheless, more than high instantaneous beam power, there is a strong demand to produce highly intense particles flux with a high duty cycle. This translates into an increase in the average intensity of the accelerated beams and consequently the mean beam power.

Figure 1.6 [12, 13] presents a (non-exhaustive) overview of hadron accelerators around the world. At present, there are two accelerators capable of delivering a beam with a mean power greater than the MegaWatt: the cyclotron at Paul Scherrer Institute (PSI) and the superconducting linac from SNS[1]. It should also be noted that all future multi-megawatt accelerators, whether planned (CiADS[2], MYRRHA[3], SPL[4], SNS upgrade) or under construction (ESS[5]), are now based on superconducting linacs. Indeed, for hadron beams close to GeV, intrinsic limits to cyclotron operation have been reached [14]. At this energy range, the orbit separation is less and less marked

[1]Spallation Neutron Source, at Oak Ridge (United States) driven by a superconducting H^- linac that provides a ∼1.4 MW beam (1 GeV, 25 mA and a duty cycle of 6 %).

[2]Chinese ADS

[3]cf. Section 1.4.2

[4]The Superconducting Proton Linac (SPL) is a multi-GeV, multi-MW linear proton accelerator project at CERN to potentially replace the first part of the LHC injection chain

[5]European Spallation Source

and it becomes increasingly difficult to extract a "mono-energetic" beam. The extraction system also induces beam losses that affect beam availability and require specific maintenance operations. In addition, due to strong space charge effects, the beam dynamics and induced resonances are limiting the increase of the beam current beyond 2 mA. In contrast, linacs are not limited in energy except by their length (and thus construction cost) and experience shows that they can provide beams with peak current on the order of a hundred milliamperes.

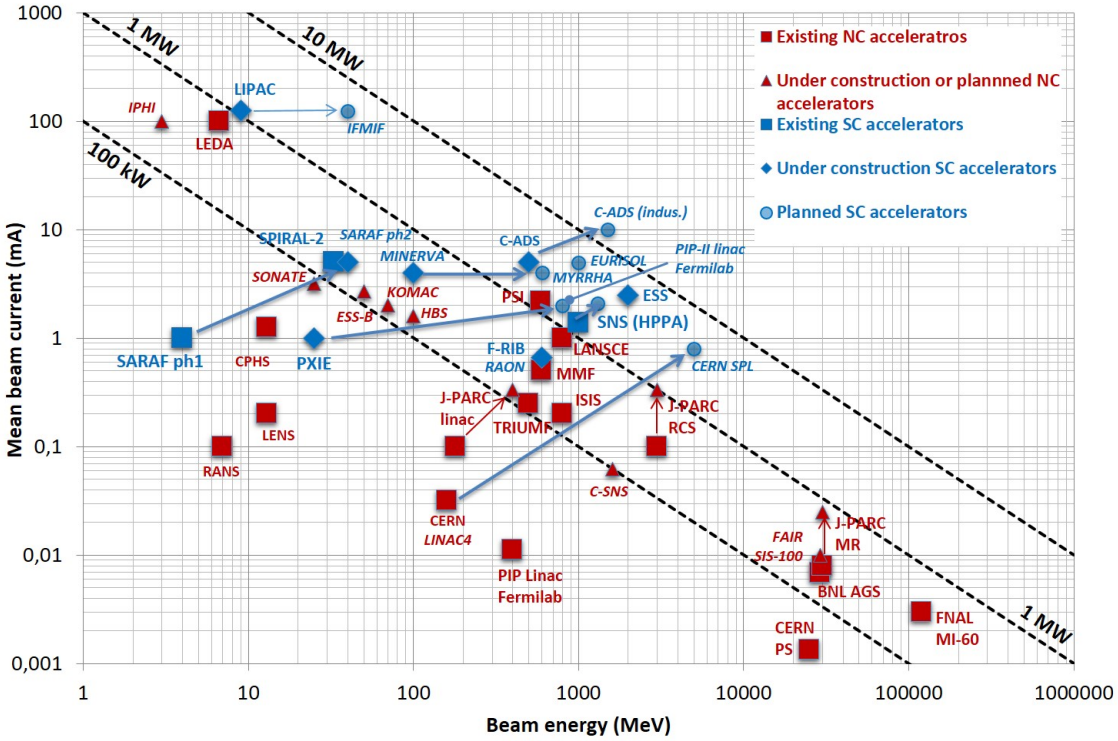

Figure 1.6: A (non exhaustive) panorama of high-power hadron accelerators [12], with updates in [13].

In addition, the repetition frequency of a linac is not limited by the rise time of the magnets as is the case for synchrotrons. It can therefore reach very high values and even go up to CW operation. For example, several high-power linacs adopt as basic repetition frequency that of the main power supplies (50 Hz or 60 Hz). Moreover, the fact that the beam passes only once in each section of a linac limits the effect on the beam of magnetic field errors i.e. beam resonances that constitute the main limitation of synchrotrons in achieving high beam currents have only a minor impact on linear accelerators.

For all these reasons, the ADS (CiADS, MYRRHA) machines are based on the use of a superconducting linac because they require proton beams with a mean power in the MW range (cf. Section 1.4.1). These machines also introduce another performance criterion: they need to operate with an extremely high level of reliability.

1.2.2 Reliability

Reliability is a major issue within the perspective to improve the long-term performance of megawatt accelerators. In order to optimize operating efficiency and cost, the requirements on beam availability are becoming increasingly high. One can take the example of the ESS project, where reliability objectives were defined upstream of the machine's construction, according to user needs. Figure 1.7 shows the acceptable number of beam interruptions according to their duration, for ESS it leads to an overall beam availability of 90 % [15].

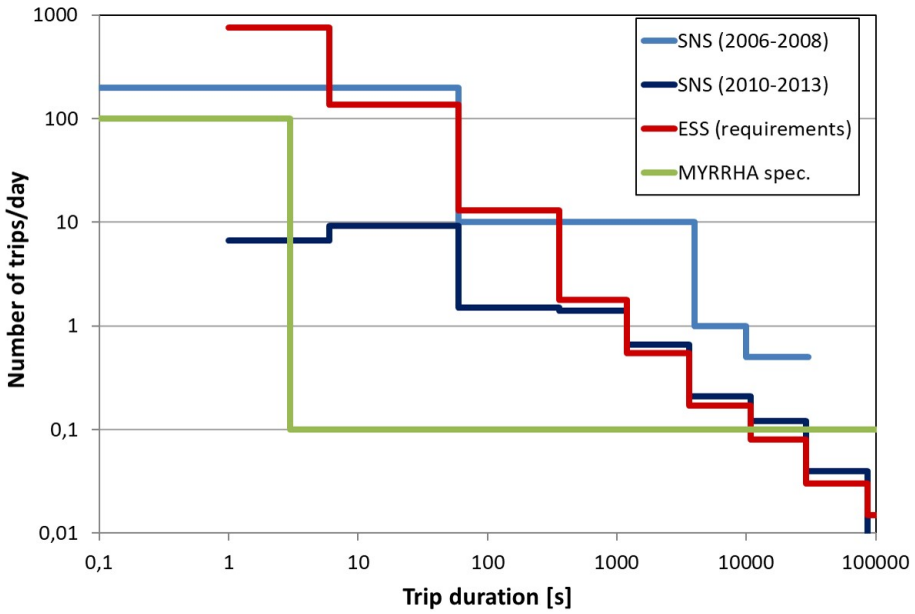

Figure 1.7: Beam trips frequency according to their duration [13, 16]: recorded at SNS during the commissioning period (2006-2008) and then during the 2010-2013 operation period [15], ESS goal [15] and MYRRHA reliability goal. Note that in this graph each "step" actually represents the number of trips/day over the interval delimited by the width of the step.

As a comparison with the statistics on the number of beam trips recorded at SNS over several years of operation, ESS requirements for short stops (up to 6 minutes) appear to be achievable. However, the targets for longer outages (more than 20 minutes) are more restrictive than those obtained during the SNS operation although the reliability of this linac seems to have slightly improved over the last few years [17].

These constraints are even more severe in the ADS case. Indeed, besides the very high level of beam power, the main - and very innovative - challenge for these machines is the extreme level of reliability required since the number of beam interruptions must be limited to very low values: one or two orders of magnitude compared to existing machines.

This is motivated by the fact that beam interruptions lasting more than a few

seconds could, if frequently repeated, induce high thermal stresses on the highly irradiated materials of the window, the target, the fuel cladding or more generally the reactor structures. Such beam interruptions, if systematically associated with a reactor shutdown, could also considerably reduce the availability of the installation, since the reactor restart procedures could last more than 20 hours [18].

In the case of the MYRRHA project(cf. Section 1.4.2), the current limit for the number of such beam interruptions has been set at 10 per 3-month operating cycle, with only interruptions of more than 3 seconds being counted. This specification is mainly motivated by the operation feedback from the PHENIX reactor [18].

In any case, the limit number of allowed beam interruptions will be significantly lower than the number of failures recorded on comparable accelerators in operation today, such as the SNS. This important difference is illustrated in Figure 1.7 and shows that reliability is indeed the main challenge for the MYRRHA accelerator. This figure also shows that reliability is improved during the commissioning and the operation of such high-power accelerators. Still one should also note that it took about 10 years of operation at SNS to reach the design power [17].

So, in order to reach the level of reliability required for the operation of an ADS, it is essential to address this issue from the beginning of the accelerator design. Consequently, it is necessary to anticipate and innovate on the tuning methods to be able to operate these machines in different modes (CW or pulsed) to provide high quality beams to avoid any beam losses that could affect the availability of an ADS. Indeed, for high power linacs, the typical beam loss tolerance is about 1 W/m which means that, for a 1 MW beam, the losses per meter should not exceed 10^{-6}.

1.3 High power linacs structure

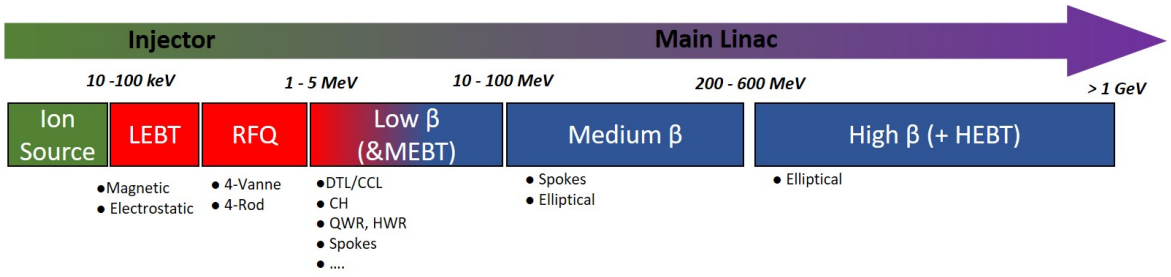

Figure 1.8: Generic topology for high power linacs. [19]

The schematic structure of high power hadron linacs is given in Figure 1.8. The beam is produced in an ion source (generally an Electron Cyclotron Resonance plasma source). It is then guided through a LEBT. In the LEBT the focusing elements can be electrostatic but nowadays they typically are magnetic. For instance, in ESS and MYRRHA (cf. Section 3.1) the beam focusing in the LEBTs is achieved by means of solenoids. Such magnetic elements enable to keep a compact line and to take full advantage of the space charge compensation effect (cf. Section 1.5.3). Then the DC beam is injected in the RFQ which bunches and pre-accelerates the beam (up to a few MeV) with a very high level of transmission (generally ∼95 %).

Then, at the end of the injector, the beam energy is boosted up 10 to 20 MeV with normal conducting RF resonant structures. Each of the coupled cells of these low β cavities has to have different lengths to follow precisely the increase in beam velocity. At higher energy instead, the accelerating structures can be made out of sequences of identical cells thus reducing the construction costs thanks to a higher standardization and to the use of longer vacuum structures. This geometrical transition on the structure of the cavities is generally made with the transition from the use of normal conducting (NC) cavities to Superconducting (SC) cavities. Such a choice is made to optimize the accelerator efficiency that can be seen as minimizing the overall power consumption with regard to the effective power delivered to the beam (cf. Section 1.3.3). So the transition energy or the necessity to use SC structures will depend on many parameters such as the cost, the peak current and in particular the duty cycle.

At transition, the beam is generally guided from the NC part to the SC main linac through a Medium Energy Beam Transfer line (MEBT). The SC linac is composed of several identical lattice meshes. The SC cavities are operated at 4 K or 2 K and they are grouped in cryomodules. In between the cryomodules, magnetic elements (generally quadrupoles) are placed to enable the beam transverse focusing. Several families (2 or 3) of cavities (with their respective range of "β" where they are efficient) are needed to accelerate the hadron beam in the GeV range. At the end of the linac,

the beam is injected into another accelerator or sent to the experimental area via High Energy Beam Transfer lines (HEBTs).

The aim of the following sections is to introduce some of the key elements used in a linac as well as to give an insight into the decision-making behind its design.

First, the working principle of an Electron Cyclotron Resonance (ECR) ion source is briefly introduced in Section 1.3.1.

Then, the design and operation of the first accelerating element, a radio-frequency quadrupole, is introduced.

Finally, the next two sections give some insight into a key choice when designing a linac: the transition energy from normal conducting cavities (cf. Section 1.3.3) to superconducting cavities (cf. Section 1.3.4).

1.3.1 Source of charged particles

The first element of a particle accelerator is always a charged particle source. In the case of a charged nuclei accelerator, the ECR sources have become the standard.

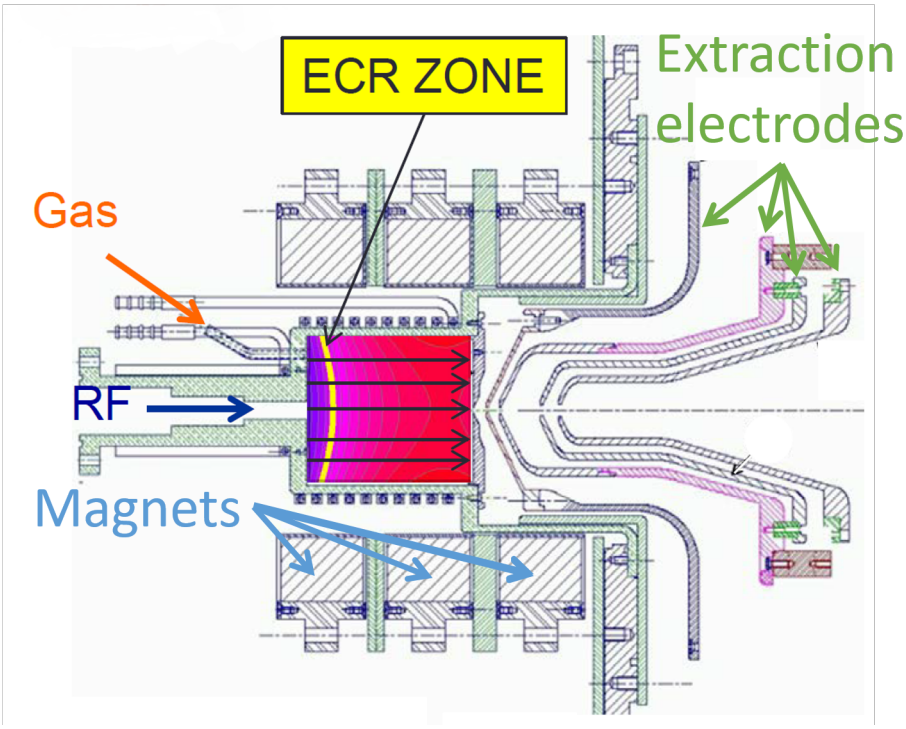

Figure 1.9: Diagram of an ECR source. [20]

An ECR source makes use of the electron cyclotron resonance to ionize a plasma. As shown in Figure 1.9, the source consists of a cylindrical metallic cavity under an intense magnetic field. The field can be set such that its intensity is minimal in the center of the cavity and maximal on the edges. This configuration allows confining of the charged particles of the plasma. The plasma is generated and maintained by

injecting an electromagnetic wave (microwave) whose angular frequency ω is equal to the angular frequency of the circular motion of the electrons ω_{ce} in the magnetic field:

$$\omega = \omega_{ce} = \frac{eB}{m_e} \tag{1.6}$$

where e and m_e are respectively the electric charge and mass of an electron and B is the magnitude of the magnetic field. This frequency is called the electron cyclotron resonance frequency hence the name of this type of ion source. By matching the microwave frequency to this resonance frequency, the electrons are efficiently accelerated and reach average energy of 1 keV up to 10 keV. At this energy, a collision between an electron and an atom will cause the ionization of the atom. Hence, new electrons are made available to be accelerated to maintain a constant rate of ionization. The plasma chamber is set at a high voltage (a few tens of kV) and the ions are extracted through a hole located on the source axis and a series of extraction electrodes.

The popularity of this type of ion source comes from the absence of filament in the chamber. This means that the source does not require maintenance and thus can operate continuously over a long period of time. In addition, ECR sources can produce continuous multi-charged ion beams with high intensities.

1.3.2 Radio-Frequency Quadrupole

At the source output, the beam is continuous and it has to be bunched to then be properly accelerated by the RF resonant structures of the linac. Nowadays, in high power hadron linacs, this process is achieved utilizing a RFQ.

The RFQ concept was invented, in the late 1960s in USSR, by I. M. Kapchinsky et V. A. Teplyakov [21], and started as an experimental device. In the mid-1970- the idea was embraced and improved by a team at Los Alamos National Laboratory in the United States. The first prototype was built and commissioned in 1980 [22]. Since that demonstration, the use of RFQ has become a standard for all high-power linacs.

As exposed in [23], the reason why RFQs became so popular is that they concurrently fulfill three different and crucial functions to ensure an efficient linac operation:

- the focus of the beam particles with an electric quadrupole field (particularly valuable at low energy where space charge forces are strong (cf. Section 1.5.3) and conventional magnetic quadrupoles are less effective),
- the adiabatic bunching of the beam i.e. the continuous beam produced by the source is segmented into bunches with minimal beam loss at the basic RF frequency (this bunching is required for acceleration in the subsequent accelerating cavities) and
- the acceleration of the beam from the source extraction energy up to the minimum required energy for injection into the following part of the linac.

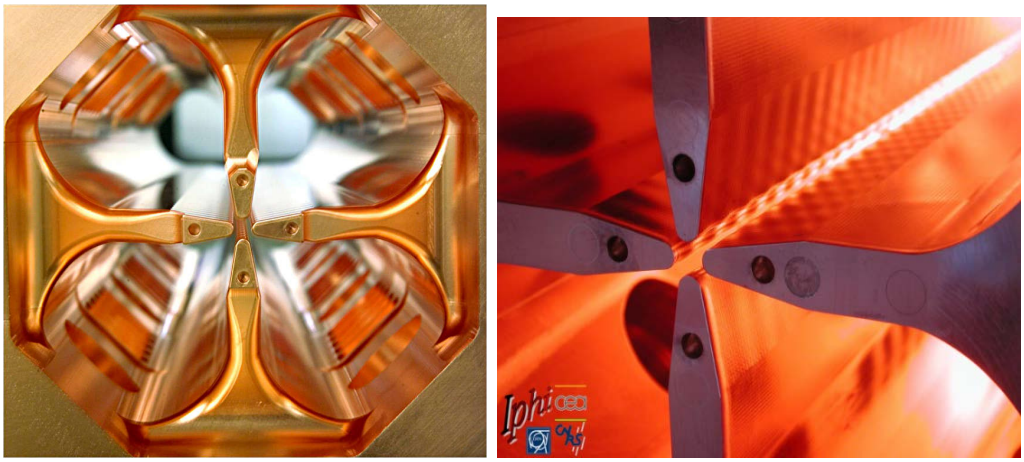

Figure 1.10: The IPHI RFQ, a "4-vane" RFQ. [24]

A RFQ is a closed RF resonator, in which four electrodes (called "vanes" in the example shown in Figure 1.10) are positioned and connected to a cylindrical tank. The resulting RF resonant mode generates a RF voltage between the vane tips that ensures the beam focusing (cf. Figure 1.11). So, a particle traveling through the channel formed by the four vanes will encounter a quadrupolar electric field whose polarity changes over time at the period of the RF i.e. the particle will see the polarity of the quadrupole reverse every half period of the RF. This means that it will cross an alternating gradient focusing channel whose periodicity corresponds to the distance traveled by the particle during half of a RF period that is $\beta\lambda/2$, with β the particle velocity and λ the wavelength of the RF resonating wave. This ensures the transverse focusing of the beam when crossing the RFQ.

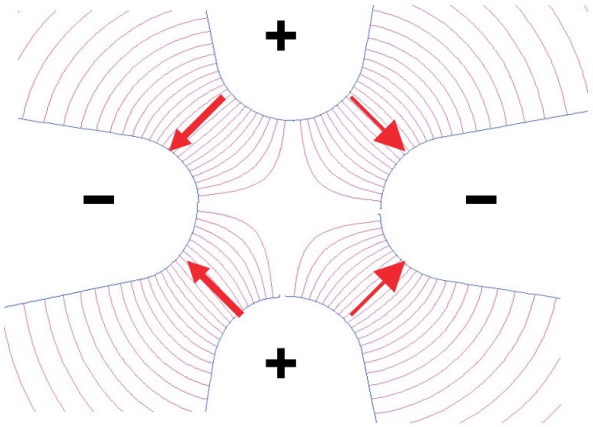

Figure 1.11: Voltages and electric fields across RFQ vanes. [23]

The longitudinal focusing required for bunching and acceleration is provided by a small longitudinal modulation of the vane tips. A sinusoidal profile whose period is

$\beta\lambda$ (cf. Figure 1.12) is machined on the tip of the vanes. To obtain a longitudinal field component, peaks and valleys of the modulation correspond on opposite vanes, whereas on adjacent (at 90°) vanes peaks correspond to valleys and vice versa. With this structure, the resulting electric field vectors can be decomposed into a transverse component (perpendicular to the direction of the beam) and a small longitudinal component (parallel to the beam direction). The transverse component is constant along the length and represents the focusing field. The longitudinal component instead changes sign (direction) every $\beta\lambda/2$ hence a particle traveling with velocity β will see an accelerating field in each cell.

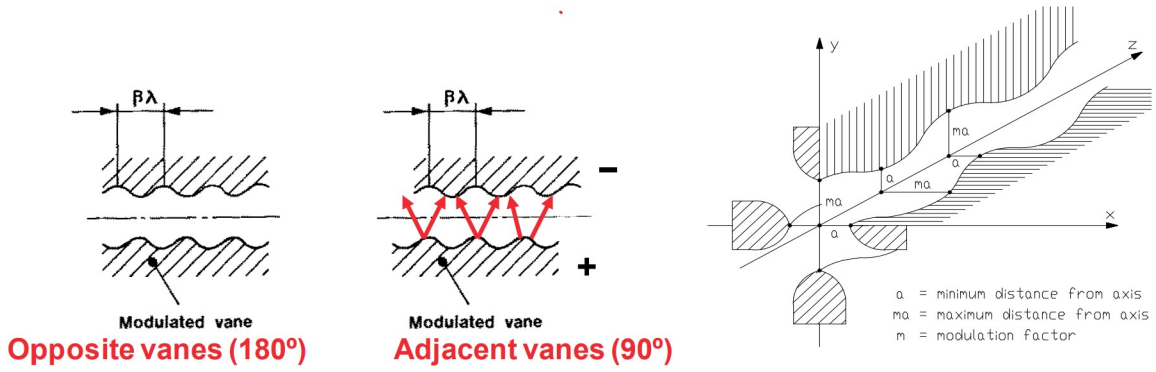

Figure 1.12: Field polarity on vanes and modulation parameters. [23]

There actually exist two types of RFQ. The "4-vane" that was just introduced and the "4-rod". The main advantages of a 4-vane RFQ are that it has a relatively even RF power density and can be easily cooled. In return, it may have a large radial size at frequencies higher than 200 MHz and the construction, tuning and operation are relatively complicated and expensive due to very tight mechanical tolerances. The 4-rod consists of four electrodes that are either circular rods with a modulated diameter or small rectangular bars with a modulated profile on one side, as described by Figure 1.13. They are connected to an array of quarter wavelength parallel plate transmission lines generating a voltage difference between the two plates. Opposite pairs of electrodes are connected to the two plates of a line, resulting in a quadrupolar voltage being generated between the rods. The rods are then inserted into a tank to obtain the RF resonator.

The main advantages of a 4-rod RFQ are the absence (or reduced impact) of dipole modes (this reduces the sensitivity to mechanical errors and simplifies the tuning) and the reduced transverse dimensions compared to the 4-vane RFQ (this makes construction and repair easier). These advantages are particularly evident for low-frequency RFQs (up to about 100 MHz). For higher frequencies ($\geq$ 200 MHz), the transverse dimensions of the 4-rod RFQ become very small and the current and power densities reach high values in some parts of the resonator, in particular at the critical connection between the rods and the supports. As a consequence, cooling is complex (especially

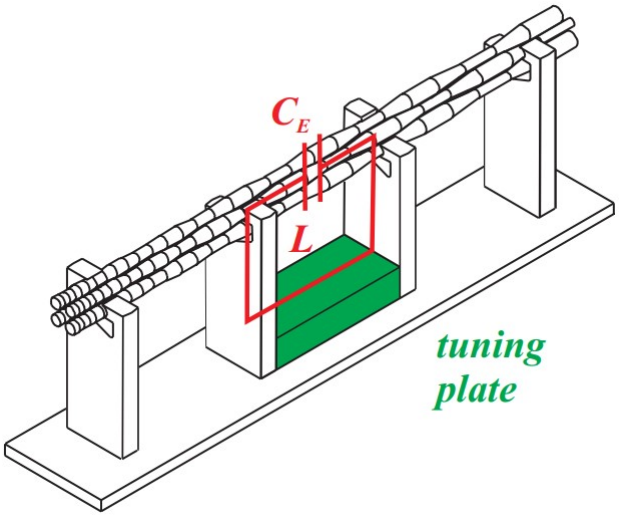

Figure 1.13: Schematic of a 4-rods RFQ. [25]

for RFQs operating at a high duty cycle) with the risk of excessive deformations of the rods and reduced beam transmission.

Finally, at frequencies higher than ∼250 MHz, the 4-vane structure is certainly the best choice for high duty cycles or CW operation which is the case of IPHI (cf. Figure 1.10 and Section 3.2.1) that operates at 352 MHz. But at lower frequencies, the 4-rod structure is more attractive and this structure was chosen for the MYRRHA RFQ (cf. Figure 1.14 and Section 3.1.1) that operates at 176 MHz.

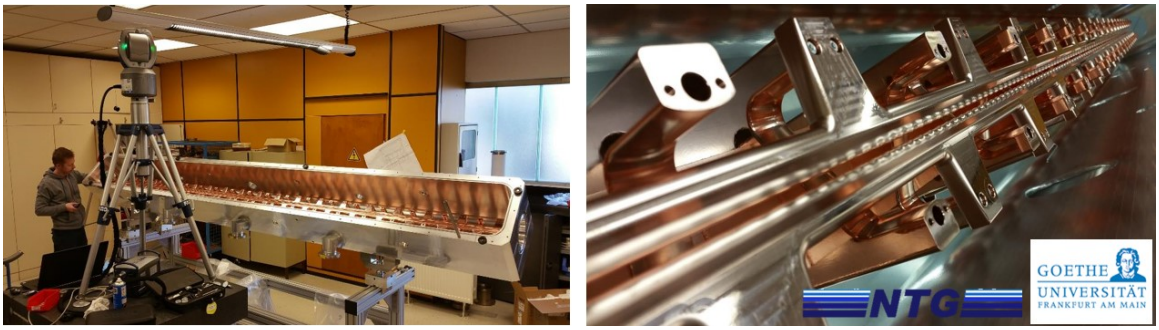

Figure 1.14: The MYRRHA RFQ (Courtesy of A. Bechtold, NTG).

1.3.3 Normal conducting cavities

At the exit of the RFQ, the beam energy is about a few MeV. Usually, the RFQ is followed by NC structures[6] that enable the acceleration of the beam up to tens or

[6]Typically made of copper or copper plated on their inner surfaces.

hundreds of MeV. These structures represent the last section of a high-power linac injector (cf. Figure 1.8). For more details on these structures, the reader can refer to [26] or [27]. Indeed, the aim of this section is not to give a detailed overview of the operation principle of these low β accelerating cavities nor to present an exhaustive list of all existing types of NC structures. It is rather to introduce key considerations when designing the NC sections of high-power linacs.

As introduced in Section 1.1, accelerating cavities are RF resonators with coupled cell cavities. In a linac, each of these is unique since the length of each cell has to be adapted to the particle velocity to ensure the synchronization of the beam with the oscillating electric field.

Usually, these resonant structures are either operating either in "2π-mode" also referred to as "0-mode" (as the Drift Tube Linac (DTL)) or in "π-mode" (as PIM cavities) as shown oinFigure 1.15. 2π-mode means that the electric fields in each accelerating gap of the cavity are oscillating in phase (the accelerating fields always point in the same direction) at the resonant frequency. Whereas, π-mode means that the field in two adjacent gaps oscillates in phase opposition (the fields in adjacent cells always point in opposite directions). The π-mode is generally used in SC cavities for acceleration at higher energies.

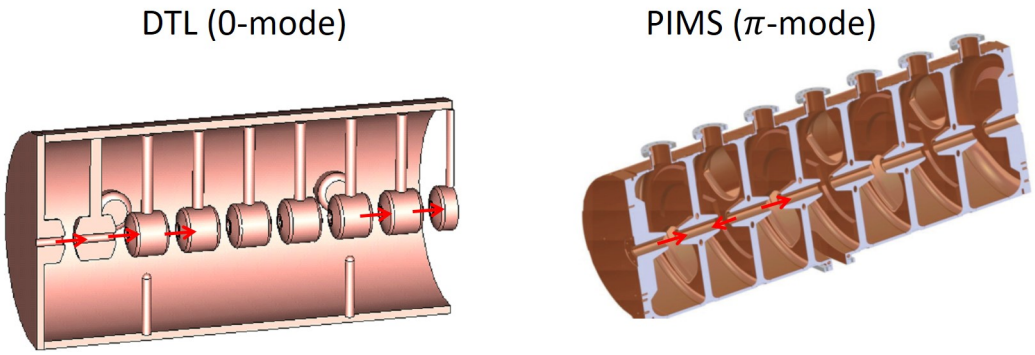

Figure 1.15: Example of a DTL structure (left) and PIMS cavity (right). The red arrows represents the polarity of the electric field along the beam axis.

Therefore to ensure the synchronism, the distance d between two accelerating gaps has to be:

- $d = \beta\lambda$ for the 2π-mode,
- $d = \beta\lambda/2$ for the π-mode.

It actually exists a large number of NC accelerating structure families. Some other resonators can also operate in $\pi/2$-mode, such as the Side-Coupled Linac (SCL). There is also a particular category of linac structures that are based on a Transverse Electric (TE) mode instead of the usual Transverse Magnetic mode. The TE modes are called H modes in the German literature and thus these structures are usually referred to as

“H-mode” structures. Two main types of such structures can be distinguished: "IH-structures" and "CH-structures" (cf. Figure 1.16). IH and CH are very compact in terms of transverse dimensions. This is an advantage at low frequencies but makes their construction difficult if high frequencies are required.

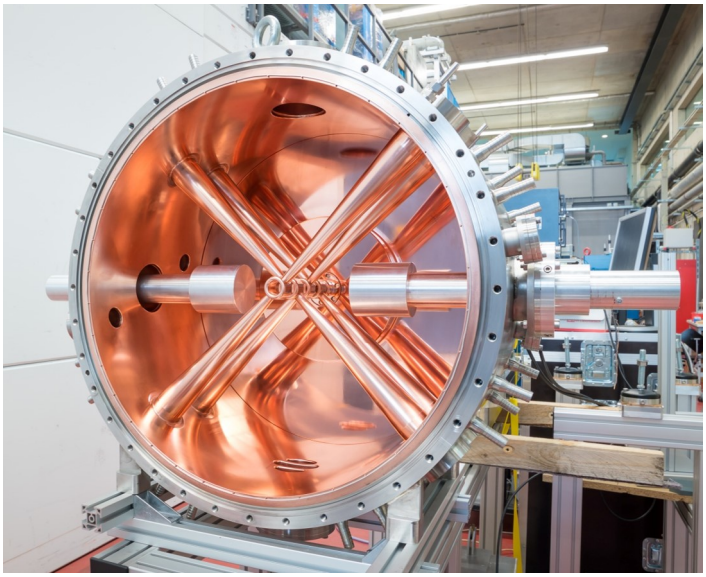

Figure 1.16: A CH-cavity. [28]

One can understand that among this large number of possible solutions choosing the most appropriate accelerating structure for a given project is very complex. The comparison between two potential solutions is based on many parameters (frequency, particle types, energy, beam current, cost, level of confidence in the technology, ...). One of the most important figures of merit used to select an accelerating structure is the shunt impedance Z which represents the efficiency of a cavity to convert RF power into a voltage across a gap. It can be defined as:

$$Z = \frac{V_0^2}{P_{cav}} \tag{1.7}$$

with V_0 the peak RF voltage in a gap and P_{cav} the RF power dissipated as heat in the cavity walls. One can also define the effective shunt impedance seen by a particle crossing the gap at velocity βc:

$$ZT^2 = \frac{(V_0 T)^2}{P_{cav}} \tag{1.8}$$

where T is the transit time factor of the particle crossing the gap. T is a function of the particle velocity β and can be defined as the ratio of the voltage seen by the particle during the crossing divided by the maximum voltage available. If the structure has many gaps, the shunt impedance is generally given per unit length (in $M\Omega/m$).

RF power is expensive and the goal is to maximize the shunt impedance which depends on the mode used for acceleration, the frequency and the geometry of the structure. Other considerations come into play in the overall optimization however the shunt impedance remains one of the essential references for the structure designer. However, comparing structures in terms of shunt impedance is no easy task because the shunt impedance depends on the chosen frequency as well as on several design parameters related to the different projects. A study made in 2008 by the EU-funded Joint Research Activity HIPPI (High-Intensity Pulsed Power Injectors) derived the shunt impedance curves presented in Figure 1.17. It compares eight different designs being studied in different European laboratories [29].

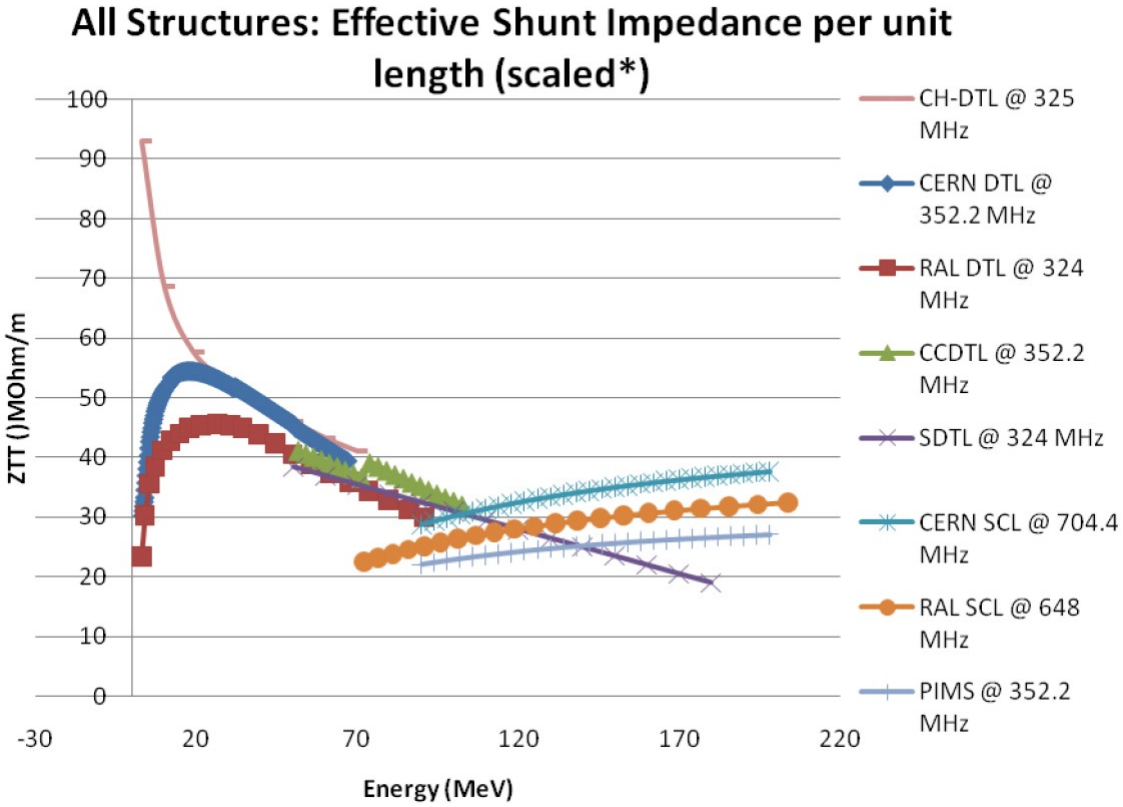

Figure 1.17: Comparison of the effective shunt impedance per unit length for different accelerating structures. [29]

In this study, the shunt impedances are scaled according to design and compared using experimental results of pulsed machines with duty cycles ranging from 5 % to 10 %. One can note that for all structures the shunt impedance has a dependence on beam energy due to the different distribution of RF currents and losses in cells of different lengths. Whereas 2π-mode structures (the DTL and the CCDTL[7]) present

[7]Cell-Coupled Drift Tube Linac

a maximum shunt impedance around 20 MeV to 30 MeV that then decreases quite rapidly with energy. In contrast, π-mode structures have a shunt impedance that slightly increases with energy but starts at lower values. Finally, one can note that a transition point from normal conducting to superconducting cavities "naturally" appears around 100 MeV.

1.3.4 Superconducting cavities

At the end of the injector section, the beam energy is sufficient to start using cavities with fixed gap lengths. These cavities are efficient over a given energy range and as they become less efficient another type of cavity can be used. Obviously, this technical solution enables to decrease the construction cost of the accelerator as it simplifies the serial production. This geometrical transition on the cavities structure generally comes with the question of transiting from NC cavities to SC cavities.

As already commented with Figure 1.6, SRF linacs are becoming more and more mandatory to reach and exceed the MegaWatt frontier. Using superconducting materials enables to reduce the power dissipated in the cavity walls and most of the power delivered by the RF power supply is transmitted to accelerate the beam. Put this way the advantages of superconducting accelerating structures are obvious. Moreover, SC cavities have large beam apertures that lead to lower beam loss (in return the particles in the beam halo could be transported in the superconducting section and then lost in the following beam transport line).

Clearly, the main reason to use SRF accelerators is to minimize the overall power consumption and thus decrease operating costs. However, some characteristics specific to superconducting systems have to be considered. A glaring example is the efficiency of the cryogenic plant required to maintain SC cavities (made of niobium) at 2 K or 4 K. Hence, one has to keep in mind that this statement depends on the operating RF or beam duty cycle. Following a simplified approach [12] one can define the total power consumption P_{tot} as:

$$P_{tot} = dc\ (P_{cav} + P_b) + P_{cryo} \tag{1.9}$$

where dc is the duty cycle of the machine, P_b the power delivered to the beam and P_{cryo} the power required by the cryogenic plant. This relation suggests that for a given duty cycle, one can ideally find the optimal energy to transition from NC to SC structures to minimize the overall power consumption of a given linac. Still, for machines with a very low duty cycle, SC structures will not be an advantage as it requires a large cryogenic installation running constantly to keep the system at cryogenic temperature.

To illustrate this one can look at the operation efficiency of one cavity that can be expressed as [19]:

$$\eta_{cav} = \eta_{RF} \frac{P_b}{P_b + \frac{P_{cav}+P_{cryostat}}{\eta_{cryo}}} \tag{1.10}$$

where η_{RF} is the RF power supply efficiency that can be estimated at $\sim$60 %, $P_{cryostat}$ is the inherent static losses of a cryomodule ($\sim$10 W) and η_{cryo} is the cryogenic efficiency

that can be estimated using the Carnot cycle and cryogenic plants efficiencies:

$$\eta_{cryo} = \eta_{Carnot}\ \eta_{plant} = \frac{2\text{ K}}{300\text{ K} - 2\text{ K}} 0.25 \sim 0.17\ \%. \tag{1.11}$$

For a NC cavity the equation of course becomes:

$$\eta_{cav} = \eta_{RF} \frac{P_b}{P_b + P_{cav}} \tag{1.12}$$

The last term P_{cav} corresponds to the power consumption of the cavity i.e. the power dissipated in the walls of the cavity as heat:

$$P_{cav} = dc\ \frac{V_{acc}^2}{(r/Q)Q_0} \tag{1.13}$$

where r/Q is known as the geometrical shunt impedance of a cavity because it quantifies the ability of the cavity to focus the electric field on axis:

$$r/Q = \frac{V_{acc}^2}{\omega_0 W}. \tag{1.14}$$

The quality factor of a cavity is Q_0. It quantifies the losses in the cavity wall with respect to the stored energy in the resonating RF wave:

$$Q_0 = \omega_0 \frac{W}{P_{cav}} \equiv \begin{cases} 10^9 \text{ for SC cavities} \\ 10^4 \text{ for NC cavities.} \end{cases} \tag{1.15}$$

Finally, the power delivered to the beam can be estimated using:

$$P_b = dc\ V_{acc} I_b. \tag{1.16}$$

With these relationships, one can compare the efficiencies of a NC and a SC cavities for a given accelerating field. As an example, let us consider the acceleration of a CW ($dc = 1$) 10 mA proton beam at $\beta = 0.37$ ($\sim$70 MeV) using an accelerating field of 5 MV/m (assuming $V_{acc} = E_{acc} L_{acc}$ where E_{acc} and L_{acc} are respectively the accelerating field and length). Using the specifications of the MYRRHA Spoke cavity ($r/Q = 220\ \Omega$, the geometrical factor G = 110 Ω, $L_{Acc} = 0.32$ m and the surface resistance $R_s \sim 30\ \text{n}\Omega$) [30], one can derive:

$$Q_0 = \frac{G}{Rs} \sim 3.5 \times 10^9. \tag{1.17}$$

Substituting Q_0 in Equation 1.13, P_{cav} is estimated to $\sim$ 3.3 W while P_b is estimated at 16 kW with Equation 1.16. With these values, the efficiency of the SC cavity is estimated at $\sim$40 % using Equation 1.10. For a NC cavity (copper):

$$Rs \sim 6\ m\Omega \rightarrow Q_0 \sim 2 \times 10^4. \tag{1.18}$$

Hence, P_{cav} can be estimated at $\sim$ 581 kW and thus the NC cavity efficiency is $\sim$2 %. Besides, the evacuation of 581 kW is quite unrealistic.

In contrast, at $dc = 0.01$, $P_b = 160$ W while P_{cav} becomes 0.033 W for the SC cavity and 5.8 kW for the NC cavity. Hence, the SC cavity efficiency drops below the NC cavity efficiency to about 1.5 %. This shows that a transition is desirable between the two technologies.

Finding the best compromise between NC and SC technologies to optimize cost and efficiency is not a straightforward issue. It depends not only on energy and duty cycle but also on other design parameters such as repetition frequency, peak beam current and pulse length. The result is that while superconductivity is certainly the most attractive technology for linacs at high energy and high duty cycle, at low energy and low duty cycle normal-conducting structures remain more attractive.

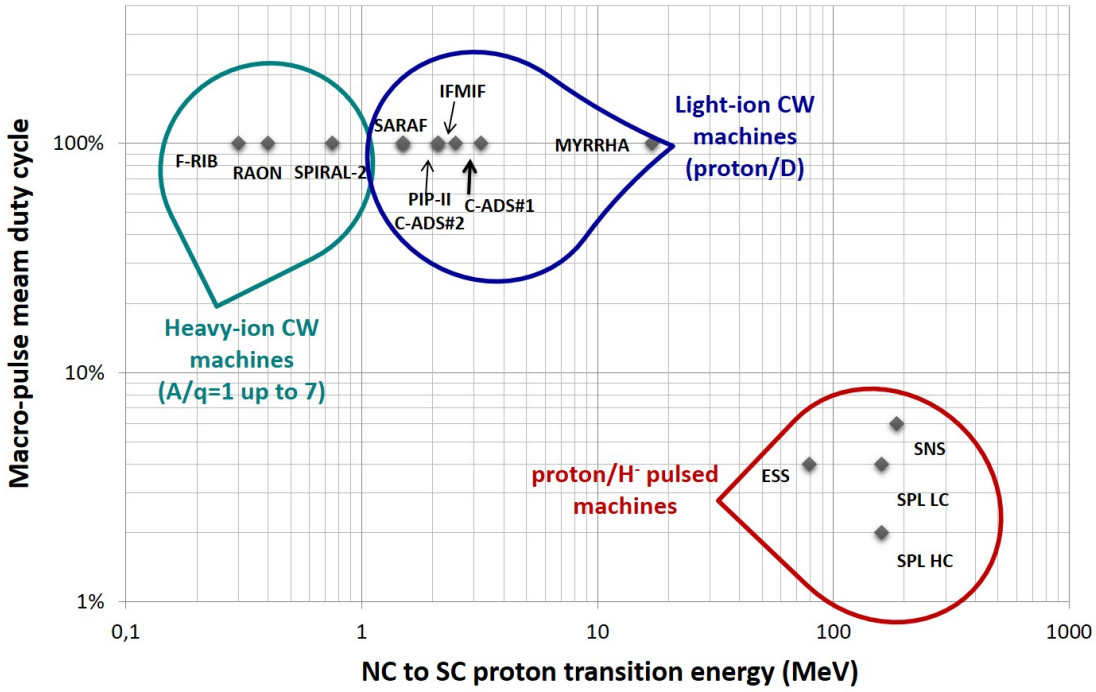

Figure 1.18: Beam duty cycle as function of NC/SC transition energy for different ion linacs. [12, 19]

These considerations are illustrated in Figure 1.18 [12, 19] that plots the actual NC to SC transition energy for the main high power ion SRF linacs (existing or planned). The two main families are clearly apparent. The first one gathers pulsed proton machines (SNS, ESS and SPL) with duty cycles in the range of a few percents and transition energies of the order of 100 MeV to 200 MeV. Note that here the energy transition corresponds to the "natural" energy transition highlighted in Figure 1.17 comparing the acceleration efficiency as regard to the cavities shunt impedance. The second one groups the CW machines (ADS case) for which the rule of thumbs becomes transiting to SRF at the lowest reasonably achievable energy.

After choosing the transition energy, the next step consists of choosing the right types of SC cavities. Similar to NC structures, there exist different types of SC resonators. As shown in Figure 1.19, for particles with a velocity greater than $\beta \sim 0.1$, four families of SC cavities can be identified.

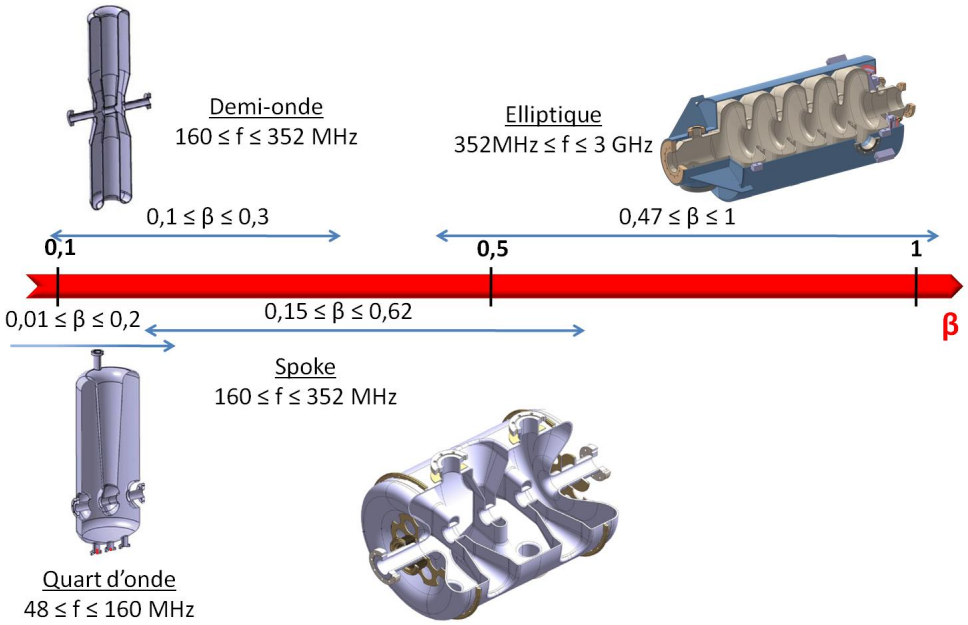

Figure 1.19: The four main SC cavity families and their energy range of application. [31]

At the beginning of acceleration up to $\beta \sim 0.15$, quarter wave cavities (cavities with coaxial line structure equal to $\lambda/4$) are preferred. These cavities are compact, modular and their manufacturing methods are well known which makes them relatively inexpensive. They allow reaching relatively high gradients ("at low β") [32]. However, their non-symmetrical structure with respect to the beam axis makes them sensitive to the phenomenon known as "steering" [33].

Half-wave resonators are symmetrical structures and thus are not very sensitive to this effect. Furthermore, the surface electric fields are lower than quarter wave cavities. In return, their production and resonant frequency adjustments are quite complicated because of their high mechanical rigidity. This last point makes them less accessible than spoke cavities.

Spoke resonators first appeared in the early 1990s [34]. With their multi-gaps structure and their operating frequency that can reach ∼800 MHz, it is possible to consider the acceleration of particles beyond $\beta = 0.5$. They are compact and much more mechanically stable than elliptical cavities. Moreover, recent results obtained in vertical cryostat tests tend to show that they can be very efficient over a wide energy range ($0.15 \leq \beta \leq 0.65$) [35, 36]. This makes spoke resonators very attractive even though their fabrication can be quite complex and expensive due to their size and shape.

Above β =0.5, elliptical cavities remain the most efficient technology for accelerating high-energy particles despite their weak rigidity which makes them more sensitive to mechanical vibrations. Due to their mechanical structure, it is also much easier to adjust their resonant frequency. Their shape allows them to limit parasitic effects such as multipacting and minimize surface field values. Overall, for very high gradient acceleration, this remains the most efficient superconducting technology. This is especially true at high β [31].

Generally, two or three families of cavities are required to accelerate a hadron beam up to the GeV range. As an example for the MYRRHA SC linac (cf. Section 1.4.2), single spoke cavities (352.2 MHz) will be used to accelerate the beam from 17 MeV to ∼100 MeV, followed by double-spoke cavities (352.2 MHz) to accelerate the beam up to ∼172 MeV and finally 5-cell elliptical cavities (704.4 MHz) will be used to reach 600 MeV. For ESS, the energy transition occurs at ∼80 MeV and one family of double-spoke cavities (352.2 MHz) and two families of elliptical cavities (704.4 MHz) will be used to accelerate the proton beam up to 2 GeV [37].

1.4 Accelerator Driven System and the MYRRHA project

The use of nuclear energy for electricity strongly grew since the beginning of the 1960s. Today, it is a major source of energy, supplying about 14 % of the world's electricity. At present, the European Union relies on nuclear fission for approximately 25% of its electricity supply, leading to an approximate annual production of 2500 tonnes of used fuel, which contains about 1 % of nuclear wastes considered as highly radiotoxic (plutonium, minor actinides and long-lived fission) [38]. Within this context, Accelerator Driven Systems are nowadays widely considered as promising devices for nuclear waste incineration, as well as useful schemes for Thorium-based energy production [39].

1.4.1 Principle

Generally, an ADS is composed of three main parts (cf. Figure 1.20):

- a sub-critical reactor in which some fuel bars are replaced by the nuclear waste to be incinerated,
- a high-intensity proton accelerator,
- a target that generates a flux of neutrons through the process of spallation when hit by protons.

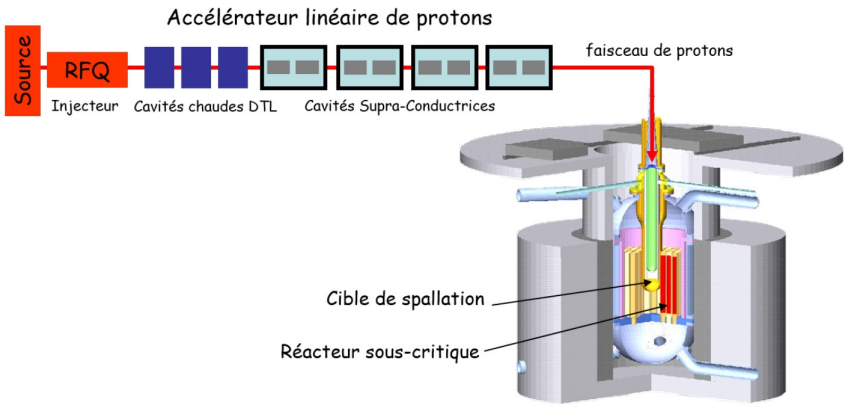

Figure 1.20: ADS Schematic diagram. [40]

The nuclear spallation principle is characterized by the interaction of high-energy protons (of at least several hundred MeV) with a thick target that leads to high neutron emissions. The proton energy being higher than the Coulomb barrier of the target nuclei causes the ejection of nucleons and the excitation of the target nuclei through intra-nuclear elastic scattering. Then, the excited nuclei decay either by fission or by new nucleon emissions. The excited nuclei can also interact with other

nuclei of the target, thus causing an inter-nuclear cascade reaction. These reactions also cause nucleon emissions. Overall, all these interactions are largely dominated by neutron emissions. Moreover, the production of neutrons gets more efficient with heavier nuclei and higher incident proton energy (cf. Figure 1.21).

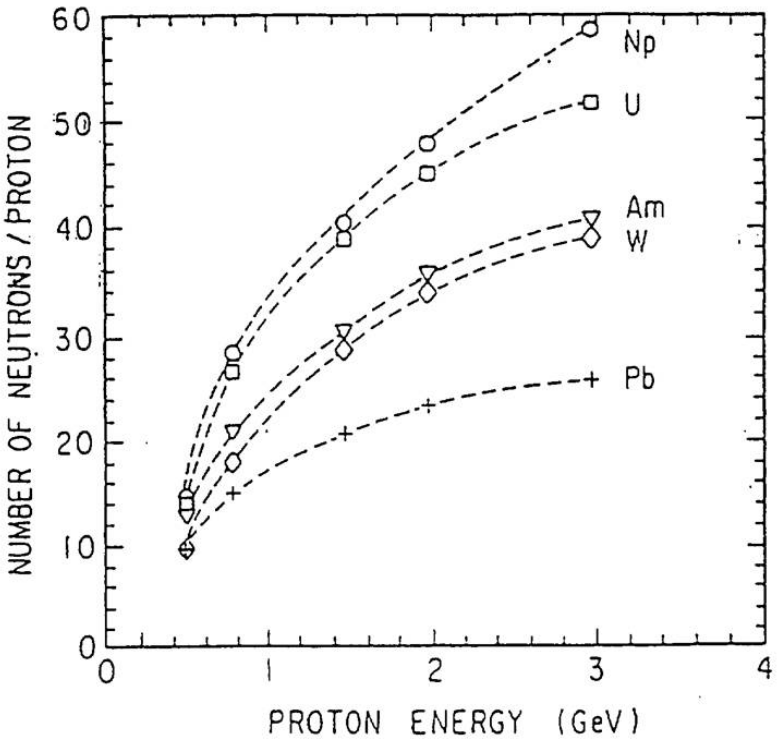

Figure 1.21: Number of neutrons per incident proton generated through the spallation process with respect to the incident proton energy. [41].

Hence, to maintain the reaction in the sub-critical reactor, the accelerator has to provide a continuous proton beam. As previously exposed in Section 1.2.2, this accelerator will have the particularity of tolerating only very few beam interruptions longer than a few seconds. Indeed, interruptions that are too long and too frequent could induce thermal stress and consequently induce early fatigue on the reactor structures. The accelerator reliability (and its control system) emerges as the major technological challenge to be applicable to hybrid reactors. The task is made all the more difficult by the need for a high-power proton beam.

In a first approximation, one can estimate the beam intensity and energy required to operate a sub-critical reactor designed for given thermal power P_{th}:

$$P_{th}(MW) = E_f(MeV) I_{beam}(A) \frac{\xi_{spall}}{\nu} \frac{\varphi^* k_{eff}}{1 - k_{eff}}. \tag{1.19}$$

In this equation, P_{th} depends on:

- E_f, the energy generated per fission ($\sim$ 200 MeV),
- ν, the number of neutrons emitted per fission (an average value of $\sim$ 2.5 can be considered),
- k_{eff}, the effective neutron multiplication factor in the reactor core (less than 1 since the ADS is sub-critical),
- φ^*, the source importance (considered $\sim$ 1.5), it characterizes the efficiency of the external neutrons and thus the coupling quality,
- ξ_{spall}, the spallation target neutron yield per incident proton (which depends on the proton beam energy),
- I, the current of the incident proton beam on the spallation target.

In the case of the MYRRHA ADS demonstrator, a maximum thermal power operation of 100 MW_{th} is targeted with $k_{eff} \sim 0.95$. The proton beam energy will be 600 MeV ($\xi_{spall} \sim 15$ on the Lead-Bismuth eutectic target). Consequently, according to equation 1.19 the minimum beam current required is $\sim$ 3 mA. The accelerator will have to provide a continuous wave (CW) of proton bunches corresponding to a $\sim$ 2 MW beam power. And finally in the case of an industrial application, for a reactor power of 3 GW_{th}, a proton beam power of 25 MW (1 GeV, 25 mA)[8] will be needed. This motivated the use of superconducting linac for the ADS demonstrator developments.

1.4.2 The MYRRHA project

MYRRHA is one of the first large scale ADS projects. Essentially, the project consists of demonstrating the feasibility of driving a nuclear fission reactor using a particle accelerator to enable the incineration of nuclear waste. This project initiated by the SCK CEN (Belgium) was developed within a European collaboration with strong support from Euratom.

As shown in Figure 1.22, a linac is used to produce a CW proton beam (4 mA at 600 MeV) which is fed into a Lead-Bismuth Eutectic (LBE) cooled subcritical reactor to produce the neutron flux required to maintain the fission reaction. The design of the reactor allows the use of a large variety of fuel as well as up to 30 % of long-lived minor actinides such as americium, neptunium and curium. Hence, the possibility to substitute part of the fuel with minor actinides follows the main motivation of the MYRRHA project to demonstrate the feasibility of nuclear waste incineration. The use of a LBE as a coolant comes from its thermo-mechanical properties (low melting point, high boiling point and high heat conductivity) but also from its nuclear properties [42]. Indeed, for MYRRHA, the LBE coolant is also used as a spallation target for the proton beam to produce the neutrons necessary to enable the nuclear reaction.

[8] $\xi_{spall} \sim 30$ for a 1 GeV incident proton beam on target.

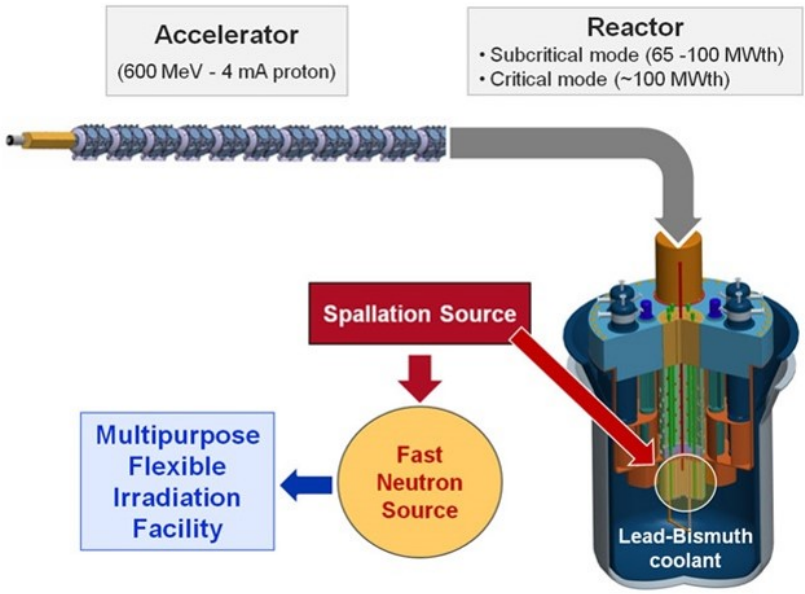

Figure 1.22: The MYRRHA project.

The construction of MYRRHA is divided into three phases:

- Phase 1 also referred to as MINERVA (MYRRHA Isotopes productioN coupling the linEar AcceleRator to the Versatile proton target fAcility) was launched in 2018 by the Belgian government and is scheduled for completion in 2026. It consists of the design and construction of the first linac section up to 100 MeV to confirm the operational reliability of the design required to later drive the reactor with the 600 MeV proton beam. The beam will also be used for experiments such as for the production of radioisotopes and nuclear fusion oriented research. Also, this phase includes the research and development for the linac extension and reactor pre-licensing.
- Phase 2 is scheduled for completion in 2033. It consists of the extension of the linac from 100 MeV to 600 MeV.
- Phase 3 consists of the construction of the reactor and the commissioning of the completed ADS which is scheduled for 2036.

For its operation, the linac of MYRRHA has to produce a high-power proton beam (2.4 MW) that meets the safety requirements relevant to the operation of a nuclear reactor. Indeed, the goal of the MYRRHA project is to show that it is possible to drive a nuclear reactor safely using a particle accelerator. Hence, the linac has to achieve the specifications required to sustain the nuclear reaction while meeting the safety rules for the reactor operation. The most important requirement is reliability because the nuclear reactor requires a stable operation for safety reasons (cf. Section 1.2.2). Based on the PHENIX specifications, an ADS should allow a maximum of 10 beam trips longer than 3 s per 3 months operating cycle (cf. Figure 1.7). This typically

corresponds to an improvement of 1 or 2 order of magnitude over current proton accelerator requirements. The beam requirements are summarized in Table 1.1.

Criterion	Specification
Proton energy	600 MeV
Peak beam current	0.1 to 4 mA
Repetition rate	1 to 250 Hz
Beam duty cycle	2×10^{-4} to 1
Beam power stability	$< \pm 2$ % over 100 ms
Beam footprint on reactor window	Circular ⌀ 85 mm
Beam footprint stability	$< \pm 10$ % over 1 s
# of allowed beam trips longer than 3 s	10 per 3-month operation period
# of allowed beam trips longer than 0.1 s	100 per day
# of allowed beam trips shorter than 0.1 s	unlimited

Table 1.1: MYRRHA main proton beam specifications. [43]

Therefore, the design and construction of the linac are major technological challenges. To meet the reliability requirement, the design follows a philosophy slightly different than other particle accelerators. Instead of trying to extract the highest performances possible of every component at all, the MYRRHA design implements a fault-tolerant strategy based on redundancy. This strategy is the core of the philosophy followed during the design of the linac as such many redundancies are planned to be able to compensate for failures (cf. Figure 1.23).

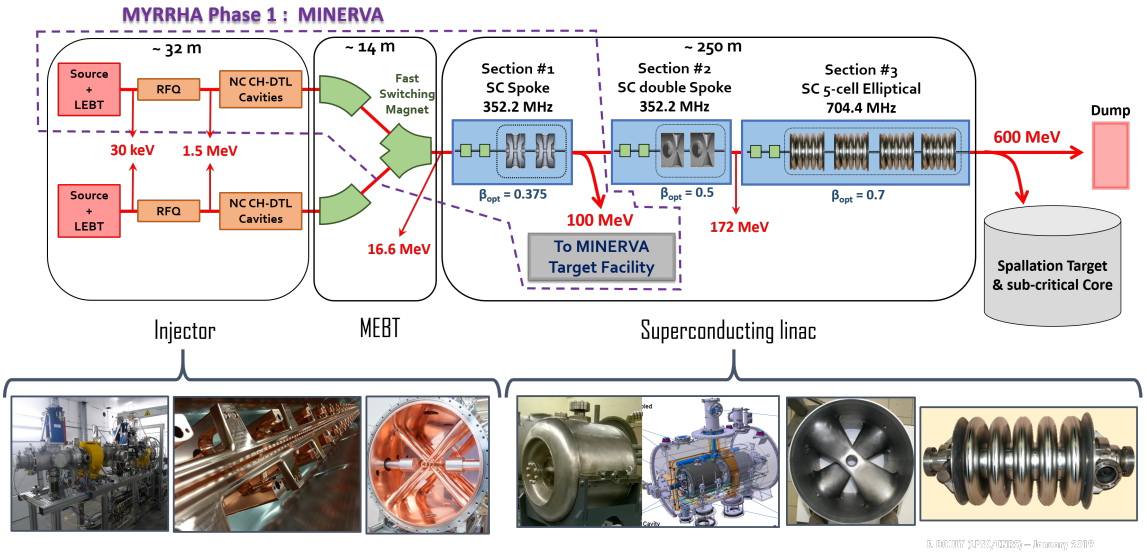

Figure 1.23: Diagram of MYRRHA. [44]

At low energy (from the 30 keV proton source up to ~17 MeV), the plan is to use parallel redundancy. Instead of a classical design with a single injector, there

will be two injectors in parallel. One will provide the 4 mA proton beam to the superconducting linac while the other will be kept on hot standby and ready to be switched with the first one in case of need. A fast switching magnet will be used to switch from one injector to the other and have the lowest impact possible on the overall operation of the accelerator.

At high energy (from ∼17 MeV up to 600 MeV), the plan is to use serial redundancy. In a classical design, a particle accelerator uses accelerating cavities at their highest performance. This allows to minimize the number of cavities and the length of the accelerator hence the construction and operation costs are also minimized. However, the fault-tolerant strategy for the MYRRHA SC linac is based on a design[9] where the cavities are operated with a margin of ∼30 % on the accelerating field i.e. during nominal operation, the nominal energy is reached using cavities at a lower set-point than their maximum one. Hence, if a cavity shuts down, its absence can be compensated by increasing the set-points of the neighboring cavities without perturbing the rest of the linac operation. Several beam dynamics studies have already been performed [45, 46] ensuring the theoretical feasibility of such a fault-tolerant strategy. Moreover, the strategy has been experimentally tested to a certain extent in the SNS [47]. Still, several conditions must be implemented in the linac design to be able to apply such failure compensation strategies:

- each accelerating cavity needs to be independently controlled in amplitude and phase,
- the beam dynamics need to be tolerant enough to allow the presence of inactive cavities (or focusing elements) and accommodate the subsequent retuning of corrective cavities (or focusing elements) without losing the nominal beam properties and
- the beam needs to be rigid enough to tolerate such a retuning.

The current scenario for fault-recovery procedures in the MYRRHA linac is based on the use of a local compensation method i.e. only adjacent elements are used to compensate for the failure. It is defined as follows [43] and should not take longer than 3 seconds.

1. In its nominal configuration, the linac is operating with every element at their nominal parameters: with a 30 % margin for the cavities and 10 % for the quadrupole power supplies.
2. A fault is detected during operation e.g. abnormal beam loss measurements. The beam is immediately and automatically stopped (by the Machine Protection System) at the exit of the LEBT in the operating injector using the equipped chopper.

[9] Three families of superconducting cavities are used in successive sections: 60 single spoke cavities at 352.2 MHz with a β of ∼0.37 from ∼17 MeV up to ∼101 MeV, 18 dual spoke cavities at 352.2 MHz with a β of ∼0.5 from ∼101 MeV up to ∼172 MeV and finally 72 elliptical cavities at 704.4 MHz with a β of ∼0.7 from ∼172 MeV up to ∼600 MeV.

3. The fault is analyzed to find its origin. If successfully diagnosed and compensable, the fault-recovery procedure is initiated. The suggested basic preliminary criterion to meet for the failure to be compensable is:

   (a) for a cavity failure: the 4 nearest neighboring cavities are usable to compensate the cavity loss i.e. the neighboring cavities are not already used to compensate another failure; there can be a maximum of 2 consecutive failed cavities for the spoke cavities and 4 consecutive failed cavities for the elliptical cavities,

   (b) for a quadrupole failure: the 4 nearest quadrupole doublets are usable to compensate the failing quadrupole doublet i.e. the neighboring doublets are not already used to compensate for another failure; there can be a maximum of 1 consecutive failed doublet.

4. The recovery then proceeds as follows (using a cavity failure as an example):

   (a) the failed cavity RF loop is immediately disabled and the cavity quickly detuned typically by $\sim$100 bandwidths to avoid the beam loading effect when the beam is started again,

   (b) in parallel, suitable new set-point values (voltage and phase) are extracted from the Control System database; these values should have been determined beforehand either from past beam experience or from a predictive calculation using a dedicated beam dynamics simulation code;

   (c) the new set-points are applied to the neighboring cavities.

5. As soon as the retuned cavities reach a steady-state, the beam is restarted at first with very short pulses to verify that the transport has been correctly recovered up to the target. The duty cycle is then ramped up to the nominal beam operation.

6. Once the nominal beam has been recovered, maintenance is started if possible i.e. if the faulty element is located in a zone reachable while the machine is operating. If the repair is successful, a return to the nominal configuration can be considered.

Due to the 3 second time limit, the utilization of an algorithm is mandatory. The plan for MYRRHA is to use a computer farm to automatically determine in advance a compensated configuration for every possible single component failure from the current state of the machine. The configurations would then be stored in a database so that when a single component actually fails it is possible to easily get the compensated configuration for this failure and apply it to the machine. If the failed element can not be determined or if no suitable compensated configuration can be found then the failure would be considered as unrecoverable under the 3 second limit and the system would be safely shut down for maintenance. This approach has to be chosen because the time required to determine a compensated configuration is usually longer than three

seconds due to the complexity of beam dynamics simulations. Hence, it is currently impossible to "immediately" compute a compensated configuration in reaction to a failure.

1.5 Charged particle beam dynamics - Basic concepts

This section is a reminder of the basic principles of beam dynamics. In particular, the theory required to understand the beam transport and magnetic focusing in a LEBT is introduced in Section 1.5. In addition, Section 1.5.2 is dedicated to the introduction of the main parameters that enable to characterize the beam properties and the concept of space charge.

1.5.1 Beam transport

1.5.1.1 Equations of motion and magnetic rigidity

Let us consider a charged particle of rest mass m_0, charge q and velocity $\vec{v}$ moving in an electromagnetic field $(\vec{E}, \vec{B})$. The Lorentz force $\vec{F}$ acts on the particle according to the following relationship:

$$\vec{F} = q(\vec{E} + \vec{v} \wedge \vec{B}). \tag{1.20}$$

By applying Newton's second law of motion $\vec{F} = \frac{d\vec{p}}{dt}$, one obtains:

$$\frac{d\vec{p}}{dt} = q(\vec{E} + \vec{v} \wedge \vec{B}). \tag{1.21}$$

Let us introduce a fictional reference particle whose trajectory can be arbitrarily defined to be the ideal trajectory followed by the center of the beam. This curvilinear trajectory is then referred to as the *reference trajectory* with symbol s and its radius of curvature as $\rho(s)$. However, here only the transverse coordinates to the reference trajectory, x and y, are of interest. With these definitions, a natural coordinate system arises as $(\vec{x}, \vec{y}, \vec{s})$ (cf. Figure 1.24).

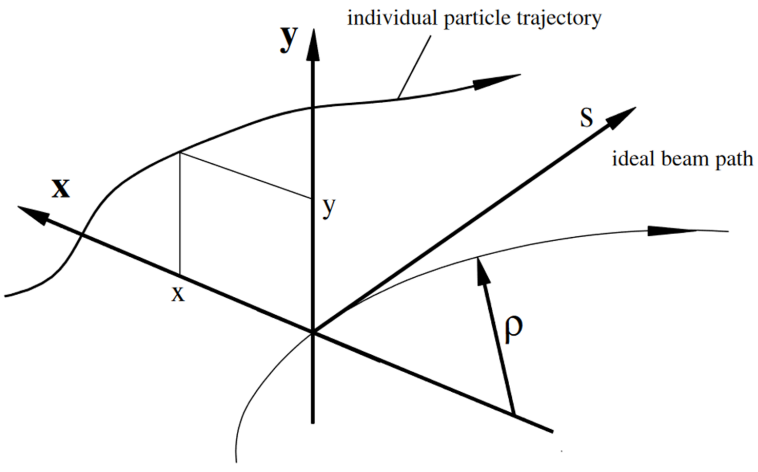

Figure 1.24: Frenet-Serret coordinate system. [48]

Assuming that the field is a constant transverse magnetic field $\vec{B} = b_0\vec{y}$. Equation 1.21 then becomes:

$$\frac{d\vec{p}}{dt} = q\vec{v} \wedge \vec{B}. \tag{1.22}$$

In this coordinate system, the position $\vec{r}$ can be written as:

$$\vec{r} = \rho(t)\vec{x} \tag{1.23}$$

where $\rho(t)$ is the radius of curvature of the trajectory. The velocity $\vec{v}$ and the acceleration $\vec{a}$ can then be derived:

$$\vec{v} = -\dot{s}\vec{s} \tag{1.24}$$

and

$$\vec{a} = -\frac{\dot{s}^2}{\rho(t)}\vec{x} - \ddot{s}\vec{s}. \tag{1.25}$$

where $\dot{s} = ds/dt$ and $\ddot{s} = d^2s/dt^2$

Rewritting Equation 1.22 as:

$$\vec{a} = \frac{q}{m}\vec{v} \wedge \vec{B} \tag{1.26}$$

where $m = \gamma m_0$ the particle mass and noting the particle trajectory is locally circular with the acceleration perpendicular to the velocity, one can obtain:

$$\frac{\dot{s}^2}{\rho(t)} = -\frac{q}{m}|\vec{v}| \cdot |\vec{B}|. \tag{1.27}$$

And thus:

$$\frac{p}{q} = -B_0\rho \tag{1.28}$$

where p is the momentum of the particle.

The *magnetic rigidity* $B\rho$ can then be defined as follows:

$$B\rho = |-B_0\rho| = \frac{|p|}{|q|} = \frac{\beta\gamma m_0 c}{|q|} \tag{1.29}$$

where β and γ are the Lorentz factors and c the speed of light.

As the momentum is conserved, this quantity characterizes the fact that a particle with higher momentum has a higher resistance to deflection by a magnetic field.

1.5.1.2 Transfer matrices

Let us now derive the equations of motion of any particle of the beam with respect to the reference trajectory i.e. using the $(\vec{x}, \vec{y}, \vec{s})$ coordinate system. Assuming a transverse magnetic field ($B\vec{u}_s = 0$), the transverse dynamics are obtained using Newton's second law:

$$m\ddot{x} = -qv_sB_y \tag{1.30}$$

and

$$m\ddot{y} = qv_s B_x. \tag{1.31}$$

The position $\vec{r}$ and the velocity $\vec{v}$ are then written as:

$$\vec{r} = \rho\vec{x} + x\vec{x} + y\vec{y} \tag{1.32}$$

and

$$\vec{v} = \dot{\rho}\vec{x} + \rho\dot{\vec{x}} + \dot{x}\vec{x} + x\dot{\vec{x}} + \dot{y}\vec{y} + y\dot{\vec{y}}. \tag{1.33}$$

Assuming that $\rho = cst$ for a given s and that the gyration is in the $(\vec{x}, \vec{s})$ plane (this means that $\dot{\vec{y}} = \vec{0}$. The temporal derivative of $\vec{x}$ with respect to the rotation around $\vec{y}$ is:

$$\dot{\vec{x}} = \vec{\Omega}_{\frac{R}{R_0}} \wedge \vec{x} = \frac{\dot{s}}{\rho}\vec{y} \wedge \vec{x} = -\frac{\dot{s}}{\rho}\vec{s}. \tag{1.34}$$

Similarly:

$$\dot{\vec{s}} = \frac{\dot{s}}{\rho}\vec{x}. \tag{1.35}$$

From these relationship, one can deduce the velocity and acceleration vectors:

$$\vec{v} = \dot{x}\vec{x} + \dot{y}\vec{y} - \dot{s}\left(1 + \frac{x}{\rho}\right)\vec{s} \tag{1.36}$$

and

$$\vec{a} = \left[\ddot{x} - \frac{\dot{s}^2}{\rho}\left(1 + \frac{x}{\rho}\right)\right]\vec{x} + \ddot{y}\vec{y} - \left[\frac{2\dot{x}\dot{s}}{\rho} + \ddot{s}\left(1 + \frac{x}{\rho}\right)\right]\vec{s}. \tag{1.37}$$

Thus, Equation 1.30 becomes:

$$m\left[\ddot{x} - \frac{\dot{s}^2}{\rho}\left(1 + \frac{x}{\rho}\right)\right] = q\dot{s}\left(1 + \frac{x}{\rho}\right)B_y. \tag{1.38}$$

Introducing the notation $u^{'} = \frac{du}{ds}$:

$$\dot{x} = \dot{s}x^{'} \tag{1.39}$$

and

$$\ddot{x} = x^{''}\dot{s}^2 + x^{'}\ddot{s}, \tag{1.40}$$

Equation 1.38 can then be rewritten:

$$x^{''} + x^{'}\frac{\ddot{s}}{\dot{s}^2} - \frac{1}{\rho}\left(1 + \frac{x}{\rho}\right) = \frac{q}{m\dot{s}}\left(1 + \frac{x}{\rho}\right)B_y. \tag{1.41}$$

By only keeping the linear terms, one obtain:

$$x^{''} - \frac{1}{\rho}\left(1 + \frac{x}{\rho}\right) = \frac{q|v|}{|p|\dot{s}}\left(1 + \frac{x}{\rho}\right)B_y. \tag{1.42}$$

Moreover, using the paraxial approximation ($v_{x,y} << v_s$), one gets:

$$\vec{v} = -\dot{s}\left(1 + \frac{x}{\rho}\right)\vec{s} \tag{1.43}$$

and

$$\frac{|v|}{\dot{s}} = \left(1 + \frac{x}{\rho}\right). \tag{1.44}$$

And thus:

$$x'' - \frac{1}{\rho}\left(1 + \frac{x}{\rho}\right) = \frac{q}{|p|}\left(1 + \frac{2x}{\rho}\right)B_y. \tag{1.45}$$

Assuming that the $y = 0$ plane is an antisymetric plane for the magnetic field, the field can be expressed as its decomposition into multiple poles:

$$B_y(x) = B_0 + \frac{dB_y}{dx}x + \frac{1}{2}\frac{d^2B_y}{dx^2}x^2. \tag{1.46}$$

By limiting the development to the first order, one obtains:

$$B_y = B_0 + Gx \tag{1.47}$$

where G is the field gradient. Finally, with only the linear terms and Equation 1.28, one obtains:

$$x'' + \left(\frac{1}{\rho^2} + k\right)x = 0 \tag{1.48}$$

where $k = \frac{G}{B\rho}$. With a similar development using Equation 1.31, a second equation is obtained:

$$y'' - ky = 0. \tag{1.49}$$

The dynamics in the transverse plane is then described by the following system of equation:

$$\begin{cases} x'' + \left(\frac{1}{\rho^2} + k\right)x = 0 \\ y'' - ky = 0 \end{cases}. \tag{1.50}$$

This means that the dynamics in x and y are uncoupled and follow the same relationship:

$$u'' + Ku = 0 \tag{1.51}$$

where

$$K = \begin{cases} \left(\frac{1}{\rho^2} + k\right) & \text{for } u = x \\ -k & \text{for } u = y \end{cases}. \tag{1.52}$$

The principal solutions of this differential equation are:

$$\text{for } K > 0 : \begin{cases} C(s) = \cos\sqrt{K}s \\ S(s) = \frac{1}{\sqrt{K}}\sin\sqrt{K}s \end{cases} \tag{1.53}$$

and

$$\text{for } K<0: \begin{cases} C(s)=\cosh\sqrt{|K|}s \\ S(s)=\frac{1}{\sqrt{|K|}}\sinh\sqrt{|K|}s \end{cases} . \tag{1.54}$$

These linearly independent solutions satisfy the following initial conditions:

$$\begin{cases} C(0)=1,\, C'(0)=\frac{\mathrm{d}C}{\mathrm{d}s}=0, \\ S(0)=0,\, S'(0)=\frac{\mathrm{d}S}{\mathrm{d}s}=0. \end{cases} \tag{1.55}$$

Any arbitrary solution $u(s)$ can be expressed as a linear combination of these two principal solutions. This result can be written using the following notation:

$$\begin{bmatrix} u \\ u' \end{bmatrix}_s = \begin{bmatrix} C(s) & S(s) \\ C'(s) & S'(s) \end{bmatrix} \begin{bmatrix} u \\ u' \end{bmatrix}_0 . \tag{1.56}$$

Hence, with a first-order approximation, one can express the effect of a magnetic field on the trajectory of a particle using matrix formalism. Such representation is referred to as êxtittransfer matrix.

1.5.1.3 Examples of transfer matrices

In a particle accelerator, magnetic fields are used to guide the charged particles along the reference trajectory. The reference trajectory is defined geometrically by straight sections and bending magnets only. Straight sections are simple spaces without any field called drift spaces. Dipole magnets are placed where the trajectory has to be curved. Solenoids, quadrupoles and higher-order magnets do not influence the reference trajectory but provide the focusing strength necessary to keep all particles close to it. There exists a transfer matrix for each of these elements to model their effects on the beam. Here some of these matrices are given under the approximation that k is constant, that the fields start and end abruptly at the boundaries of the element[10] and ignoring perturbations. The derivation of these matrices is given in [48].

Drift space

A drift space consists in a vacuum chamber without magnetic field. The transfer matrix of a drift space of length L is:

$$M_u = \begin{bmatrix} 1 & L \\ 0 & 1 \end{bmatrix} . \tag{1.57}$$

Magnetic quadrupole

Quadrupoles are very common focusing elements in particle accelerators. The structure of their field simultaneously induces a focusing of the beam in one plane and

[10] This approximation is referred to as the "hard edge" approximation.

a defocusing of the beam in the perpendicular plane. Hence, they are usually installed in doublets with alternating focusing planes to obtain an overall focusing effect in both planes. The transfer matrices for a quadrupole focusing in the x plane are:

$$M_x = \begin{bmatrix} \cos\phi & \frac{1}{\sqrt{K}}\sin\phi \\ -\sqrt{K}\sin\phi & \cos\phi \end{bmatrix} \tag{1.58}$$

and

$$M_y = \begin{bmatrix} \cosh\phi & \frac{1}{\sqrt{K}}\sinh\phi \\ -\sqrt{K}\sinh\phi & \cosh\phi \end{bmatrix} \tag{1.59}$$

where $K = \frac{G}{B\rho}$, $\phi = \sqrt{K}L$ and L is the length of the quadrupole.

Magnetic dipole

A dipole is used to curve the trajectory of the beam. The transfer matrices for a magnet of arc length L and a bending angle $\theta = \sqrt{K}L$ in the x plane are:

$$M_x = \begin{bmatrix} \cos\theta & \frac{1}{\sqrt{K}}\sin\theta \\ -\sqrt{K}\sin\theta & \cos\theta \end{bmatrix} \tag{1.60}$$

and

$$M_y = \begin{bmatrix} 1 & L \\ 0 & 1 \end{bmatrix}. \tag{1.61}$$

These matrices show that a dipole acts as a drift space in the non-deflecting plane and has a focusing effect in the deflecting plane.

Solenoid

At low energies such as in a LEBT, solenoids can be used to focus the beam. It consists of the induction of a magnetic field along the reference trajectory by applying a current in a coil placed around the beam path. With such a configuration, the dynamics of the transverse planes are coupled. The corresponding transfer matrix is:

$$M_s = \begin{bmatrix} C^2 & \frac{2}{K}CS & CS & \frac{2}{K}S^2 \\ -\frac{K}{2}CS & C^2 & -\frac{K}{2}S^2 & CS \\ -CS & -\frac{2}{K}S^2 & C^2 & \frac{2}{K}CS \\ \frac{K}{2}S^2 & -CS & -\frac{K}{2}CS & C^2 \end{bmatrix} \tag{1.62}$$

where $C = \cos\psi$ and $S = \sin\psi$ with $\psi = \frac{Bl_m}{2B\rho} = \frac{Kl_m}{2}$. Hence, the effect of a solenoid on a beam is the combination of a focus and a rotation.

Transfer matrix of a system

Realistically, a particle accelerator uses a succession of a number of these elements to maintain the particles of the beam close to the reference trajectory. The transfer matrix formalism enables one to derive a global transfer matrix for the accelerator by

combining the matrices for each of its elements. For a system with n elements, the global transfer matrix is given by:

$$M_{n\leftarrow 1} = M_{n\leftarrow(n-1)} \cdot M_{(n-1)\leftarrow(n-2)} \cdots M_{2\leftarrow 1}. \tag{1.63}$$

1.5.2 Emittance and beam parameters

1.5.2.1 Emittance concept and Twiss parameters

Equation 1.51 describes the motion of a particle in a single guiding element using the parameter K supposed to be independent of s. This leads to describe the motion of the particle passing through a succession of guiding elements by multiplying their transfer matrices (cf. Equation 1.63). However, to describe the evolution of the trajectory of a particle along the complete transport line, it is necessary to take into account that K actually depends on s.

Let us resume from Equation 1.51:

$$u^{''} + K(s)u = 0. \tag{1.64}$$

Using the variation of parameters method to solve this equation, one obtains for u:

$$u(s) = \sqrt{U}\sqrt{\beta(s)}\cos\psi(s) - \Psi \tag{1.65}$$

where U and Ψ are constants and $\beta(s)$ and $\psi(s)$ are functions to determine. By differentiation of $u(s)$ and with Equations 1.65 and 1.64, one obtains:

$$\beta^{''}(s)2K(s) - \frac{\beta^{'2}(s)}{2\beta(s)} - 2\psi^{'2}(s)\beta(s) = 0 \tag{1.66}$$

and:

$$\psi^{''}(s)\beta(s) + \beta^{'}(s)\psi^{'}(s) = 0. \tag{1.67}$$

The $\beta(s)$ function is normalized by setting $\beta(s)\phi^{'}(s) = 1$ and $\psi(0) = 0$. Then, $\psi(s)$ can be written:

$$\psi(s) = \int_0^s \frac{d\sigma}{\beta(\sigma)} \tag{1.68}$$

and is referred to as the *phase advance* of the particle.

By setting $\alpha(s) = -\beta^{'}(s)/2$, Equation 1.66 becomes:

$$u^{'}(s) = -\sqrt{\frac{U}{\beta(s)}}\left[\alpha(s)\cos\psi(s) - \Psi - \sin\psi(s) - \Psi\right]. \tag{1.69}$$

With Equations 1.65 and 1.69 and setting $\gamma(s) = \frac{1+\alpha^2(s)}{\beta(s)}$, one obtains:

$$U = \gamma_{uu'}(s)u^2(s) + 2\alpha_{uu'}(s)u(s)u^{'}(s) + \beta_{uu'}(s)u^{'2}(s) \tag{1.70}$$

which is a constant of the motion referred to as the Courant-Snyder invariant. This relationship defines an ellipse in the phase space $(u, u^{'})$. The area of this ellipse is πU and is a constant. The parameters $\alpha_{uu'}$, $\beta_{uu'}$ and $\gamma_{uu'}$ are referred to as the *Twiss parameters*. As these parameters depend on s, they depend on the elements that constitute the transfer line. The relationship between the ellipse and the Twiss parameters is illustrated in Figure 1.25.

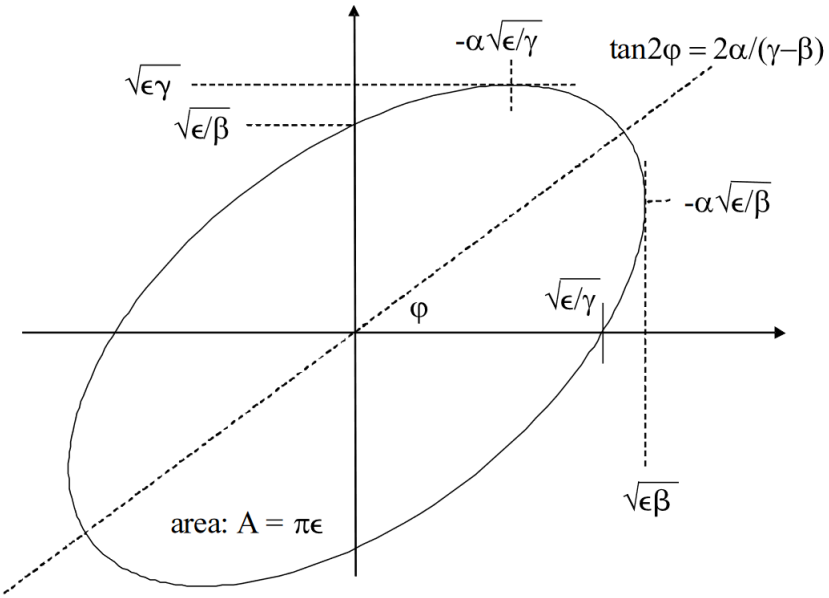

Figure 1.25: Phase space ellipse.

Hence, all particles travel along their individual ellipse in the phase space $(u, u^{'})$ as they are transported through the transfer line. If one chooses the Twiss parameters of the particles on the largest phase ellipse within a particular beam, all other particles within that ellipse will stay within that ellipse. And thus, the collective behavior of a beam formed by many particles can be described by the dynamics of a single particle. To that end, the *emittance* $\epsilon_{uu'}$ is defined as the area of the ellipse that contains a certain fraction of the particles of the beam:

$$\epsilon_{uu'} = \gamma_{uu'} u^2 + 2\alpha_{uu'} uu^{'} + \beta_{uu'} u^{'2}. \tag{1.71}$$

The area that encompasses all the particles in the phase space is:

$$A_{uu'} = \int\int du du^{'}. \tag{1.72}$$

And thus, the total emittance is defined as:

$$\epsilon_{uu'} = \frac{A_{uu'}}{\pi} \tag{1.73}$$

With this definition, the emittance remains constant for a beam of constant energy but its value changes in case of acceleration. To be able to compare the beam quality

in the transverse phase spaces across different energies, one has to use the *normalized emittance* $\epsilon_{uu',norm}$ which is invariant under acceleration. It is defined by:

$$\epsilon_{uu',norm} = \beta_z \gamma \epsilon_{uu'} \tag{1.74}$$

where γ is the Lorentz factor and β_z is the relativistic velocity in the beam propagation direction.

1.5.2.2 Statistical emittance

From the beam representation in the phase space, a *beam matrix* can be defined as:

$$\Sigma \equiv \begin{bmatrix} < u^2 > & < uu^{'} > \\ < uu^{'} > & < u^{'2} > \end{bmatrix} = \begin{bmatrix} \sigma_u^2 & \sigma_{uu'} \\ \sigma_{uu'} & \sigma_{u'}^2 \end{bmatrix} = \begin{bmatrix} \beta & -\alpha \\ -\alpha & \beta \end{bmatrix}. \tag{1.75}$$

Hence, each of the Twiss parameters can be defined using the standard deviations σ_u and $\sigma_{u'}$ and the covariance $\sigma_{uu'}$. The emittance is then defined as:

$$\epsilon = \sqrt{\det\Sigma} = \sqrt{\sigma_u^2 \sigma_{u'}^2 - \sigma_{uu'}^4}. \tag{1.76}$$

Thus, the Twiss parameters become:

$$\alpha = -\frac{\sigma_{uu'}^2}{\epsilon}, \tag{1.77}$$

$$\beta = \frac{\sigma_u^2}{\epsilon} \tag{1.78}$$

and

$$\gamma = \frac{\sigma_{u'}^2}{\epsilon}. \tag{1.79}$$

1.5.2.3 Twiss parameters transport

According to Liouville's theorem, all particles enclosed by an envelope ellipse will stay within that ellipse. Hence, the transformation of the Twiss parameters along the beam line may be derived from the transfer matrices. Starting at $s = 0$, a particle on the ellipse follows:

$$\gamma_0 x_0^2 + 2\alpha_0 x_0 x_0^{'} + \beta_0 x_0^{'2} = \epsilon = \gamma x^2 + 2\alpha x x^{'} + \beta x^{'2}. \tag{1.80}$$

Any particle trajectory starting at $s = 0$ transforms to $s \neq 0$ by:

$$\begin{bmatrix} x \\ x^{'} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_0^{'} \end{bmatrix} = \begin{bmatrix} C(s) & S(s) \\ C^{'}(s) & S^{'}(s) \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_0^{'} \end{bmatrix} \tag{1.81}$$

and thus, by inverting the matrix:

$$\begin{bmatrix} x_0 \\ x_0^{'} \end{bmatrix} = \frac{1}{CS^{'} - C^{'}S} \cdot \begin{bmatrix} S^{'}(s) & -S(s) \\ -C^{'}(s) & C(s) \end{bmatrix} \cdot \begin{bmatrix} x \\ x^{'} \end{bmatrix} = \begin{bmatrix} S^{'}x - Sx^{'} \\ -C^{'}x + Cx^{'} \end{bmatrix} \tag{1.82}$$

because $\det M = 1$. From this equation, one obtains:

$$x_0^2 = S'^2 x^2 - 2SS' xx' + S^2 x'^2, \tag{1.83}$$

$$x_0'^2 = C'^2 x^2 - 2CC' xx' + C^2 x'^2 \tag{1.84}$$

and

$$x_0 x_0' = -C'S' x^2 + (CS' + C'S) xx' - CS x'^2. \tag{1.85}$$

Thus:

$$\begin{aligned} \gamma x^2 + 2\alpha xx' + \beta x'^2 = &\left(S'^2\gamma_0 - 2C'S'\alpha_0 + C'^2\beta_0\right) x^2 && (1.86)\\ &+ 2\left(-SS'\gamma_0 + (CS' + C'S)\alpha_0 + CC'\beta_0\right) xx' && (1.87)\\ &+ \left(S^2\gamma_0 - 2CS\alpha_0 + C^2\beta_0\right) x'^2 && (1.88)\\ = &\ \epsilon && (1.89) \end{aligned}$$

Finally, identifying term by term, one can deduce the transformation of the beam parameters in matrix formulation:

$$\begin{bmatrix} \beta \\ \alpha \\ \gamma \end{bmatrix} = \begin{bmatrix} C^2 & -2CS & S^2 \\ -CC' & CS' + C'S & -SS' \\ C'^2 & -2C'S' & S'^2 \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \alpha_0 \\ \gamma_0 \end{bmatrix}. \tag{1.90}$$

1.5.3 Space charge and compensation

The space charge and its compensation can influence the beam dynamics. This is especially true for beams at low energy such as in a LEBT. The following sections aim to briefly introduce both concepts. For a detailed study of these phenomena, the reader is referred to F. Gérardin's Ph.D. thesis [49].

First, the origin of the space charge effect and its dependency on the beam energy is given in Section 1.5.3.1.

Then, the compensation of the space charge effect through the interaction between the beam and the gas in the chamber is introduced in Section 1.5.3.2.

1.5.3.1 Space charge

Let us consider two particles with identical electrical charge q separated by a distance r. At rest, the Coulomb force is exerted on each particle. This force can be calculated using:

$$\vec{F}_{qq}^{E} = \frac{q^2}{4\pi\epsilon_0}\frac{\vec{r}}{r^3} \tag{1.91}$$

where ϵ_0 is the vacuum permittivity. As both particles have the same charge, the Coulomb force will push them apart from each other.

If the particles are moving in the same direction with velocity $v = \beta c$, they will generate two parallel currents that induce a magnetic field B. This field will exert a force $\vec{F}_{qq}^{M}$ expressed by:

$$\vec{F}_{qq}^{M} = q\vec{v} \times \vec{B}. \tag{1.92}$$

The effect of this force attracts the particles towards each other.

In a particle accelerator, a beam contains many particles with identical charges that propagate in the same direction. In this beam, each particle is subjected to the Coulomb force generated by every other particle. Therefore, the resulting force is the sum of the repulsive electrical forces and the attractive magnetic forces.

Let us consider a continuous beam with cylindrical symmetry propagating along the z axis with constant velocity $v = \beta c$. The charge density in the beam is independent of z:

$$\rho(x, y, z) = \rho\left(\sqrt{x^2 + y^2}\right) = \rho(r). \tag{1.93}$$

- Electric field contribution
  The symmetry of the system imposes that only the radial component of the electric field E_r is non-zero. Using the Gauss's law on a cylinder of length L and radius r, one can show that the radial electric field is:

$$E_r(r) = \frac{1}{\epsilon_0 r} \int_0^r \rho(u) u du. \tag{1.94}$$

- Magnetic field contribution
  Here, the symmetry of the system imposes that only the azimuthal component of the magnetic field B_θ is non-zero. Using the Ampère-Maxwell equation in free space, one shows that the azimuthal magnetic field is:

$$B_\theta(r) = \frac{\mu_0}{r} \int_0^r J(u) u du \tag{1.95}$$

  where $J(r)$ is the current density. Because all the particles of the beam move with the same longitudinal speed $v_z = \beta_z c \vec{u_z}$, the current density is:

$$J(r) = \rho(r) \beta_z c. \tag{1.96}$$

  Hence, the magnetic field is given by:

$$B_\theta(r) = \frac{\mu_0 \beta_z c}{r} \int_0^r \rho(u) u du. \tag{1.97}$$

  With Equation 1.94, one can see that:

$$B_\theta(r) = \frac{\beta_z}{c} E_r(r). \tag{1.98}$$

Hence, the space charge force $\vec{F} = q(\vec{E} + \vec{v} \wedge \vec{B})$ seen by a particle of the beam is radial:

$$F_r = qE_r(1 - \beta^2). \tag{1.99}$$

where $\beta \simeq \beta_z (>> \beta_x + \beta_y)$. From this relationship, it is apparent that there is a competition between a repulsive force qE_r and an attractive force $-q\beta^2 E_r$. The ratio between both contributions $\tau = 1/\beta^2$ depends only on the velocity of the beam and is shown in Figure 1.26. This clearly shows that the space charge force is always repulsive except when $\beta = 1$ where both contributions balance each other perfectly. The intensity of the repulsion is especially high as the velocity is low. This means that in an accelerator the space charge force is maximal in the LEBT.

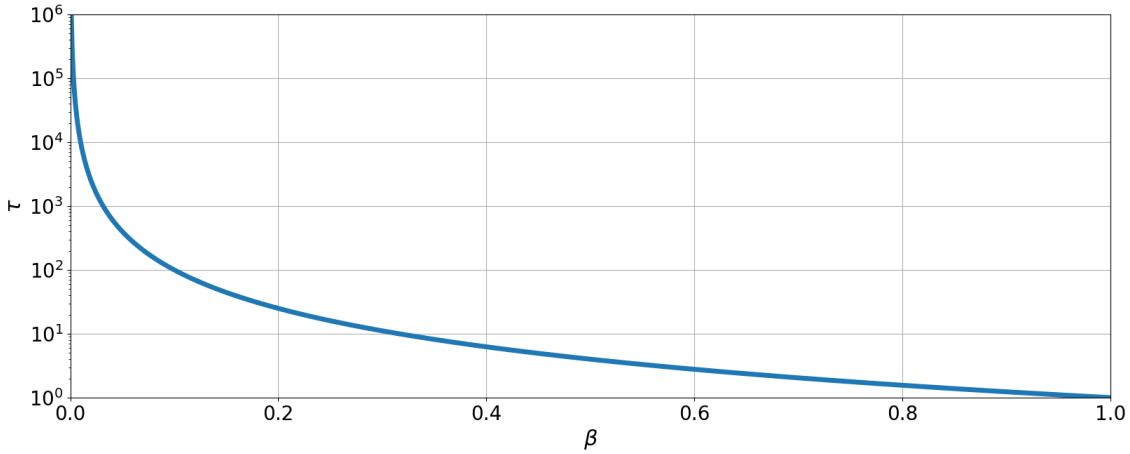

Figure 1.26: Ratio of the contributions to the space charge force with respect to the particle velocity.

Assuming that the density of the beam is uniform:

$$\rho(r) = \begin{cases} \rho_0 \text{ for } r \leq R_b \\ 0 \text{ for } r > R_b \end{cases} \tag{1.100}$$

where R_b is the radius of the beam. The beam current can be obtain by integrating Equation 1.96:

$$I = \beta c \int_0^{2\pi} d\theta \int_0^{R_b} \rho(u) u du. \tag{1.101}$$

From which the charge density ρ_0 can be deduced:

$$\rho_0 = \frac{I}{2\pi\epsilon_0 \beta c R_b^2}. \tag{1.102}$$

With this relationship, Equation 1.94 becomes:

$$E_r(r) = \begin{cases} \frac{I}{2\pi\epsilon_0 \beta c R_b^2} r \text{ for } r \leq R_b \\ \frac{I}{2\pi\epsilon_0 \beta c r} \text{ for } r > R_b. \end{cases} \tag{1.103}$$

Thus with Equation 1.99, the space charge force is:

$$F_r(r) = \begin{cases} \frac{qI}{2\pi\epsilon_0\beta c R_b^2}(1-\beta^2)r \text{ for } r \leq R_b \\ \frac{qI}{2\pi\epsilon_0\beta c r}(1-\beta^2) \text{ for } r > R_b. \end{cases} \tag{1.104}$$

In this case, the space charge force:

- is zero at the center of the beam ($r = 0$),
- is linear inside the beam ($r \leq R_b$),
- decrease as $1/r$ outside of the beam ($r > R_b$) and
- increase linearly with the beam current I.

1.5.3.2 Space charge compensation

There is always some amount of gas remaining in a LEBT despite the pump system. The contributors to this residual gas are the ion source and the desorption from the walls of the chamber. The beam interacts with this gas by ionizing its content. The charged particles produced through the inelastic scattering will present two types of behavior based on the sign of their charges. The particles with the same charge as the beam will get pushed out towards the walls while the particles with opposite charges will be confined and accumulate around the beam axis (cf. Figure 1.27). These behaviors are caused by the electrostatic potential induced by the charged particles of the beam. Indeed, a voltage is established between the beam axis and the walls of the chamber. This voltage is referred to as the space charge potential well.

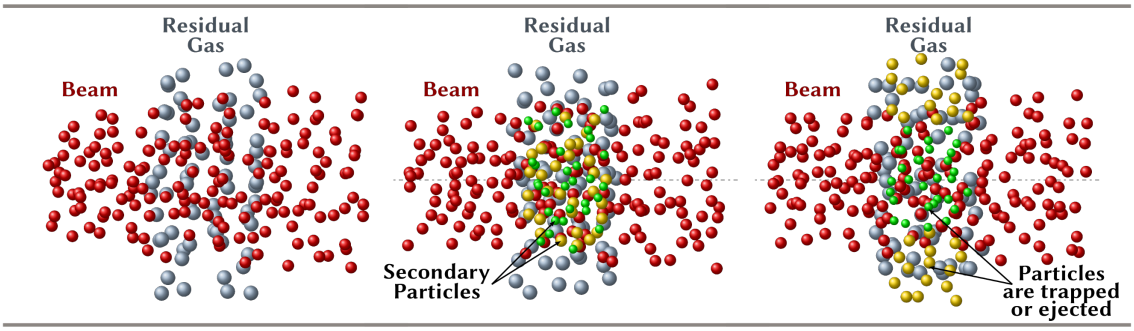

Figure 1.27: WARP [50] simulation of the effect of the space charge potential well on the ionized particles of the residual gas. [51]

Due to the accumulation of particles with opposite charges around the beam axis, the space charge potential well is progressively neutralized. This last phenomenon is referred to as space charge compensation. For a 100 mA uniform H^+ beam at 100 keV with a radius of 10 mm, Figure 1.28 shows different degrees of neutralization assuming that the distribution of the trapped particles is identical to the distribution of the beam.

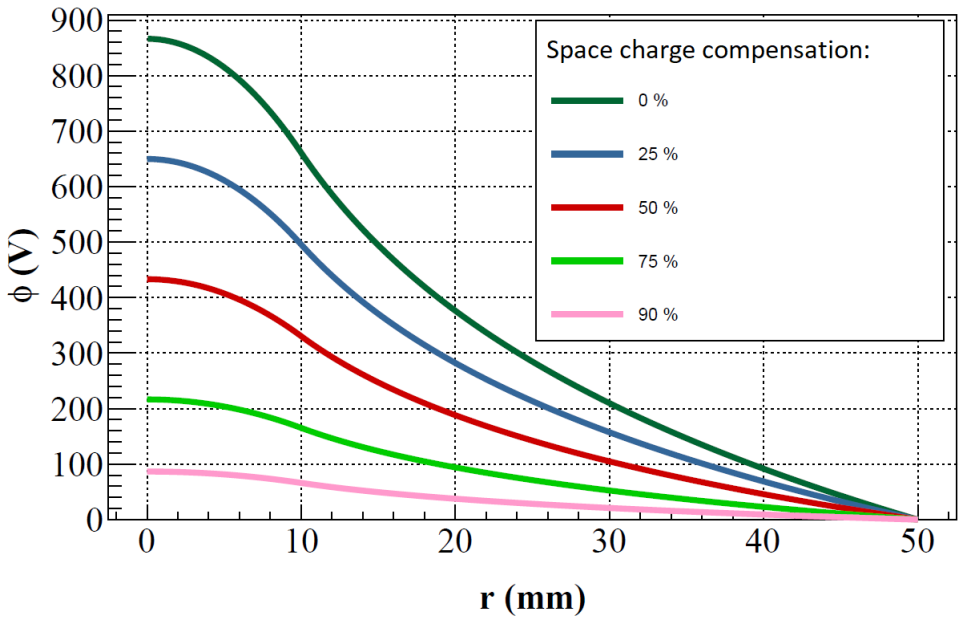

Figure 1.28: Space charge potential well at different degrees of compensation. [49]

Another quantity used to describe the space charge compensation is the space charge compensation time T_{SCC}. It is defined as the time it takes to fully compensate for the space charge of a continuous beam. This quantity can be estimated using:

$$T_{SCC} = \frac{1}{\sigma_i(E) v_f n_g} \tag{1.105}$$

where σ_i is the total ionization cross section of the gas with respect to the kinetic energy of the beam E, v_f is the velocity of the beam and n_g is the gas density. It corresponds to the time between the injection of the beam and the compensation of its space charge through its interaction with the gas. This means that the space charge compensation presents transient effects.

Therefore, the space charge potential well depends on the following parameters:

- the beam current I,
- the beam velocity β,
- the beam radius R_b,
- the residual gas density n_g and
- the space charge compensation time T_{SCC}.

1.6 Summary

In this chapter, an overview of the field of particle accelerators has been given from their discovery up to the design of high power superconducting linacs without forgetting about the development of cyclotron and synchrotrons. This lead to the discussion on the performances of accelerators. The quest for higher beam energy for research in particle physics such as for CERN but also for higher mean power for many applied physics projects such as for SNS, PSI and of course MYRRHA. With the latter, the need for higher reliability has been introduced. Indeed, MYRRHA being a demonstrator project of an ADS has a very strict reliability requirement: a maximum of 10 beam trips per 3-month operation cycle. This requirement exceeds the ability of current particle accelerators and requires adopting a philosophy focused on fault tolerance during the design phase.

To illustrate some parts of the accelerator design, the structure of high power linacs as well as some key elements have been introduced. First, an ECR ion source provides a continuous high intensity beam of protons for days with very low maintenance requirements. Then, a RFQ is used as the first accelerating and bunching element of the injector. After that, the beam is further accelerated up to the target energy by RF cavities. The decision behind the transition energy from NC cavities to SC cavities has also been introduced. Indeed, this is one of the key decisions to make during the design it will condition the fault-tolerant strategies and the operating price of the completed accelerator.

Next, the concept of an ADS and in particular the MYRRHA project has been introduced. The goal of an ADS is to use an accelerator to provide neutrons through the spallation process inside a nuclear reactor. This allows designing sub-critical nuclear cores dependant on an external neutron source to maintain fissile reactions. With such designs, it is possible to use a large variety of fuel as well as minor actinides for their incineration. To demonstrate this, the MYRRHA project plans on using a 2.4 MW CW proton beam (4 mA, 600 MeV) to control a 100 MW_{th} reactor. To meet the reliability requirement on the beam imposed by the operation of a reactor, the design of MYRRHA includes a fault-tolerant strategy based on redundancies. In particular, the design includes two identical injectors for parallel redundancy at low energy. When a failure occurs in the working injector, the second injector in hot standby would take over to provide the beam to the main SC linac. For the SC linac, the design assumes a nominal operation with a power margin on the different elements. Hence, when a failure occurs, the neighboring cavities, cryomodules or quadrupole can take over for the failing one. One of the difficulties for such a strategy to be applicable is to know a compensated configuration for every recoverable failure. This requires building a database with either previously experienced compensated failures or predicted configurations based on simulations.

Finally, the basic principles of beam dynamics have been introduced. In particular, the equations of motion of a particle in a magnetic field have been derived to introduce the concept of magnetic rigidity. The latter expresses the fact that a particle with higher momentum will have a higher resistance to deflections by a magnetic field.

Then, the transfer matrix formalism has been shown to present the first-order effects on the beam of some components of the beam optics. The concept of the beam emittance has then be presented to link the dynamics of a single charged particle to the characterization of a whole beam. Finally, the space charge and its compensation have been introduced. These last effects have a strong influence on the dynamics of a low energy beam such as in the LEBT that guides the beam from the ion source to the RFQ.

Keeping these concepts in mind, the next section is dedicated to the introduction of the objectives of this thesis and the structure of the manuscript.

1.7 Thesis objectives

During the operation of any accelerator, it is important to minimize beam losses. Indeed, the particles lost along the beamline will hit the walls of the accelerator and cause damages. These damages can accumulate over time and cause severe issues preventing the operation of the accelerator until its reparation. This is especially concerning for accelerators with higher mean power and duty cycle such as the MYRRHA accelerator.

One of the ways to minimize such losses is to ensure the quality of the beam to be accelerated. This is especially important for the MYRRHA accelerator because the compensated configurations planned for the fault-tolerant strategy require the accelerator to work in many different configurations. The configuration of the LEBT is crucial for the beam because of its role to guide, clean and focus the beam from the ion source to its injection into the RFQ. However, the optimization of the LEBT configuration is quite a difficult task. Indeed, the properties of the beam at the exit of the ion source are usually not well known, the low energy beam is subject to a strong space charge effect and the alignment of the different elements has to be compensated. In addition, it is necessary to be able to quickly switch the configuration of the injector over a wide range of configurations (low current and duty cycle for the startup up to the nominal beam current and duty cycle).

In this context, the objective of this thesis is to explore novel methods for the optimization and modelization of particle accelerators. In particular, the main purpose of this work is to study the possibility of using Machine Learning methods to develop a numerical model able to reproduce accurately the experimental behavior of an injector and especially a LEBT. To do so, neural networks based models have been trained to reproduce the behavior of two machines: the MYRRHA LEBT and the IPHI injectors. The networks have been trained using both simulated and experimental datasets to achieve the highest fidelity possible to their respective real machines.

As such, the principles of ML, neural networks and Particle Swarm Optimization are introduced in Chapter 2.

Then, a description of the MYRRHA LEBT and IPHI as well as of the experimental work performed on those is given in Chapter 3. The aim is to illustrate:

- the behavior of these machines,
- the online test of a PSO algorithm and
- how the experimental datasets have been constituted.

Chapter 4 is dedicated to a more formal description of the function to model and the datasets that were used.

Finally, the results of the training of the neural network models are discussed in Chapter 5.

Chapter 2

Artificial Intelligence, Machine Learning and Particle Accelerators

Contents

On the one hand, the developments in Artificial Intelligence (AI) have lead the particle accelerator community to develop an interest in a few algorithms. As such, there have been multiple attempts to optimize different aspects related to particle accelerator design or operation. Notably, meta-heuristic algorithms such as Genetic Algorithms (GAs) have been tested either on accelerator simulations or directly on an operating accelerator. Many experiments have shown promising results such as the increase of beam lifetime in storage rings.

On the other hand, the recent advances in Machine Learning have lead many to try to implement such methods in their own fields and many new applications have been developed. In particular, neural network developments provide opportunities to create new modelization tools which are faster than current simulation codes.

This chapter is dedicated to introducing these concepts. Section 2.1 describes the principle, the structure and the training of *feedforward* neural networks and introduces a metaheuristic algorithm: Particle Swarm Optimization. Section 2.2 presents various attempts at applying AI and ML methods to particle accelerators. Section 2.3 describes the objectives of the thesis in the context of AI and ML methods.

2.1 Principle and theory

This section is dedicated to the description of the tools used in this work: Machine Learning with neural networks and Particle Swarm Optimization.

First, the principle behind neural networks and ML is explained in Section 2.1.1.

Next, the way a neuron works is introduced and the structure of neural networks is described in Section 2.1.1.1.

Then, a description of supervised learning and its main components (backpropagation, stochastic gradient descent as well as training, validation and test datasets) is given in Section 2.1.1.2.

Afterward, a brief introduction to reinforcement learning is done in Section 2.1.1.3.

Finally, in Section 2.1.2, an optimization algorithm is introduced: PSO.

2.1.1 Machine Learning

Neural networks appeared in the '60s. Since 2009, deep artificial neural networks have become the most influential tool for machine learning mostly because of the implementation of neural networks on Graphical Processing Units (GPUs). Indeed, with GPUs, it becomes possible to train bigger neural networks on a larger database.

The development of deep neural networks is a very active field of research and numerous new concepts and tools have emerged. Notably, this field has seen the recent introduction of architectures [52], regularization methods [53], activation functions [54] and optimization methods [55].

Also, deep neural networks have become essential for many applications such as speech recognition [56, 57], image processing [58–60], medical diagnosis [61] or weather forecasting [62].

2.1.1.1 Artificial Neural Networks

Artificial Neuron

An artificial neuron is loosely based on real biological neurons. In a brain, stimuli are received by a neuron. They take the shape of neurotransmitters attaching to receptors on the wall of the cell which causes a change in the electric potential. If the resulting potential is higher than an excitation threshold, the neuron is activated and will then release neurotransmitters to stimulate other neurons. In a similar manner an artificial neuron receives a vector of real numbers as its input $\mathbf{x} = (x_0, x_1, ..., x_j)$ to represent the stimuli. A weighted sum is performed on the input with a weight vector of real numbers $W = (w_0, w_1, ..., w_j)$ to represent the relative importance of each stimulus. Then a bias b is applied to the weighted input to represent the excitation threshold and gives the final excitation z:

$$z = \sum_j w_j x_j + b. \tag{2.1}$$

Finally an activation function $f(z)$ is applied to the final excitation and its result a is the output of the artificial neuron (cf. Figure 2.1). Hence the effect of a neuron on a set of input can be written:

$$f(z) = f(W \cdot X + b) = a. \tag{2.2}$$

Therefore, the output of an artificial neuron (called *activation*) for a given input is determined by the choice of the activation function and by the values of the weights and the bias.

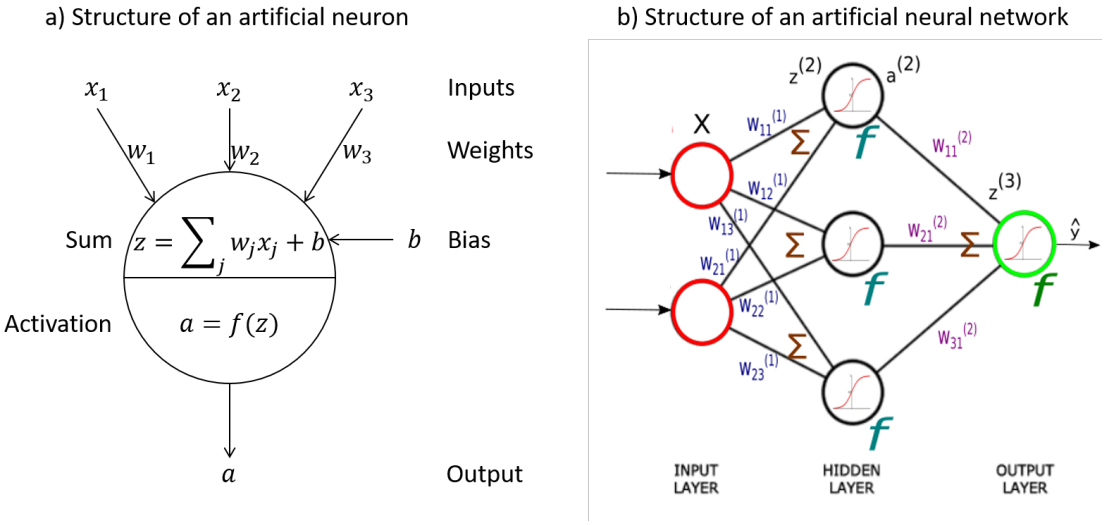

Figure 2.1: Left: Diagram of an artificial neuron. Right: Diagram of a neural network.

Artificial Neural Network

By itself, a neuron can model simple relations but a network of neurons can model a wide variety of continuous functions. A typical network is built by arranging neurons by layers and linking the output of the neurons of one layer to the input of the neurons in the next layer. The first layer is called the input layer. The last layer is called the output layer. Any layers in between are called hidden layers. The structure of a three-layer neural network is shown in Figure 2.1.

The design of the input and output layers is usually straightforward. The input layer consists of as many neurons (called *input neurons*) as there are inputs to the function that we want to approximate. Its sole role is to transmit the input to the first hidden layer.

The output layer consists of as many neurons as there are outputs to the function that we want to approximate.

However, the design of the hidden layers does not follow such simple rules. The number of hidden layers and the number of neurons in each hidden layer are called the *hyperparameters* of the network. These are typically determined by a trade-off against the time required to train the network.

In essence, a neural network represents a function whose shape is determined by a high number of parameters (the weights and biases of every neuron).

Note that the networks described here, where the output from one layer is used as the input to the next layer, are called *feedforward* neural networks. This means that there are no loops as it would make it hard to compute i.e. the input of a neuron never depends on its own output.

There exists a type of neural network that allows such loops called *recurrent neural networks*. The idea in these models is to have neurons that activate for a short duration before becoming quiet again. That signal can excite other neurons that may activate a little while later also for a short duration. That causes still more neurons to activate and so a cascade of neuron activation is obtained. In this case, there is no problem having loops as the output of a neuron only affects its input after some time has passed.

There are four main types of feedforward networks:

1. Multilayer perceptrons (MLP),
2. Autoencoders (AE),
3. Convolutional networks (CNN),
4. Generative adversarial networks (GAN).

MLPs [63] are the standard of feedforward networks. The network shown in Figure 2.1 is an example of a MLP. They are universal function approximators as shown by Cybenko's theorem [64] provided that there is a sufficient number of hidden neurons. Hence they are suited to create mathematical models by *regression* and can be used for *classification* (a type of regression where the output is a categorical variable). The typical choices for the activation function are the sigmoid function:

$$f(z) = \sigma(z) \equiv \frac{1}{1 + e^{-z}}, \tag{2.3}$$

the hyperbolic tangent:

$$f(z) = tanh(z) \equiv \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.4}$$

and the rectified linear unit:

$$f(z) = max(0, z). \tag{2.5}$$

An AE [63] is a network that learns to copy its inputs to its outputs hence the number of output neurons is equal to the number of input neurons. Between the input and output layers, there is a hidden layer with fewer neurons than the input and output layers to force the network to "encode" the most important information then "decode" to reproduce the input (cf. Figure 2.2). This type of network is suited for dimensionality reduction.

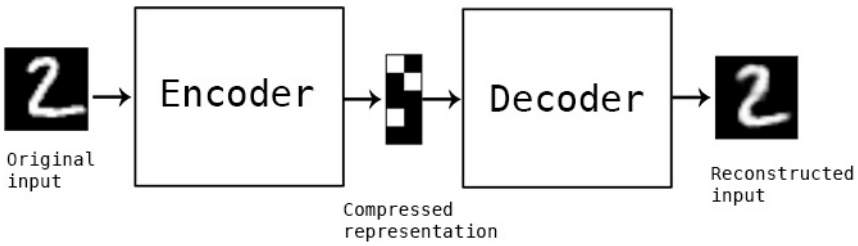

Figure 2.2: Principle of an autoencoeder. [65]

CNNs have been introduced for image processing [66, 67]. They emulate how a visual cortex works and are very good to model spatial relations over multiple dimensions. They typically consist of a series of convolution layers alternating with pooling layers and finally a MLP (cf. Figure 2.3).

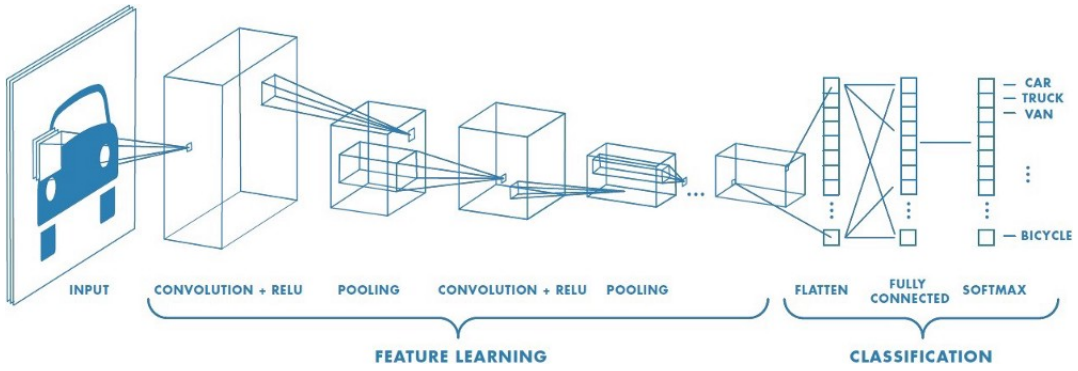

Figure 2.3: Diagram of a convolutional neural network [68].

A GAN uses two networks trained against each other in a zero-sum game [52]. Given a training set, the first network G is used to generate new data with the same statistics as the training set i.e. if trained on photographs the network will generate new photographs that look at least superficially authentic to a human observer (cf. Figure 2.4). The second network D is trained to discriminate between real data from the training set and data generated by the first network. As the training progresses the network G becomes better and better at fooling the network D.

Figure 2.4: Faces generated using styleGAN. The people in the middle do not actually exists. The faces are generated by mixing the features of the faces in source B with the coarse features of the faces in source A. [69]

Among all the possible network architecture, MLPs trained in regression mode are the most suited for the task of modeling a physical phenomenon with no time dependence. The training itself is performed under the supervised learning paradigm described next.

2.1.1.2 Supervised Learning

Principle

The following description of supervised learning is based on [70].

In supervised learning, the aim is to learn a mapping from an input space X to an output space Y. This assumes that there is a function h such that:

$$h(x \in X) = y \in Y. \tag{2.6}$$

Then the goal is to find an estimate $\hat{h}$ of this function using a set of N provided examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ called *training set*. With this representation, $\mathbf{x}_i$ is the i^{th} observation with $\mathbf{x}_i = \{x_i^1, x_i^2, ..., x_i^d\} \in X$ where d is the dimension of the input space. The d values in x_i are called features or attributes. For example, the features of weather model could be the outside temperature, the humidity, the atmospheric pressure and

so on. The $y_i \in Y$ are the *labels* corresponding to their respective x_i. Typically a label can take two types of values.

In a classification setting, the label is one category belonging to a finite set of possible categories. For instance, the categorical labels to describe the weather could be $Y = \{sunny, rainy, ...\}$. If there are only two categories then it is referred to as a *binary classification* setting whereas when there are more than two categories then it is a *multi-class classification* setting.

In the regression setting, y_i is a real number. To keep the weather model example, the label could be the amount of rain in millimeters.

In both settings, the aim is to utilize the estimate $\hat{h}$ found using the training set for prediction $\hat{h}(x_i) = \hat{y}_i$ on observations outside the training set (this is called *generalization*).

Feedforward neural networks utilize a supervised learning method called *backpropagation* for training. To quantify how well the estimate $\hat{h}$ is performing its task, a *cost function* C is then introduced:

$$C(w,b) \equiv \frac{1}{2N} \sum_i \parallel y_i - \hat{h}(x_i) \parallel^2 = \frac{1}{2N} \sum_i \parallel y_i - \hat{y}_i \parallel^2, \tag{2.7}$$

where w represents the collection of all the weights in the network, b all the biases and N is the number of examples in the data set. This function is known as the *mean squared error* (MSE) and is used because it is non-negative and becomes small ($C(w,b) \approx 0$) when the prediction $\hat{y}_i$ is close to the label y_i for all training inputs. [71]

Backpropagation principle

The following description of backpropagation is based on [71].

Backpropagation is an algorithm that computes the gradient of the error made by a network with respect to the weights and biases of the network for a single input-output example. It does so efficiently which allows the use of gradient methods for training and minimizing the cost function (cf. Equation 2.7).

To simplify the notation needed to understand backpropagation, it is useful to describe the weights, biases and activations of a network using matrix-based indices. Let us call w^l_{jk} the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. In a similar way, let us use b^l_j for the bias of the j^{th} neuron in the l^{th} layer and a^l_j for the activation of the j^{th} neuron in the l^{th} layer. With these notation, the activation a^l_j is related to the activations in the $(l-1)^{th}$ layer by the Equation 2.2, so one can write:

$$a^l_j = f(\sum_k w^l_{jk} a^{l-1}_k + b^l_j), \tag{2.8}$$

where the sum is over all neurons k in the $(l-1)^{th}$ layer. With this notation it is possible to define the weight matrix w^l for each layer l where the value in the j^{th} row and k^{th} column is the weight w^l_{jk}. In a similar way, let us define the bias vector b^l and

the activation vector a^l. Finally, let us introduce the vectorization of the activation function:

$$f(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} f(x_1) \\ f(x_2) \end{bmatrix}, \tag{2.9}$$

that is, the vectorized activation function is applied elementwise on the vector. With these notations, Equation 2.8 can be rewritten as:

$$a^l = f(w^l a^{l-1} + b^l) = f(z^l), \tag{2.10}$$

where $z^l \equiv w^l a^{l-1} + b^l$ is called the *weighted input* to the neurons in layer l.

Now the goal of backpropagation is to compute the partial derivatives $\partial C/\partial w$ and $\partial C/\partial b$ of the cost function C with respect to any weight w or bias b in the network. To do that, backpropagation consists in first deriving the partial derivatives $\partial C_i/\partial w$ and $\partial C_i/\partial b$ for a single training example. Then the partial derivatives $\partial C/\partial w$ and $\partial C/\partial b$ are obtained by averaging over training examples:

$$\frac{\partial C}{\partial w} = \frac{1}{N}\sum_i \frac{\partial C_i}{\partial w} \text{ and } \frac{\partial C}{\partial b} = \frac{1}{N}\sum_i \frac{\partial C_i}{\partial b}. \tag{2.11}$$

This is possible because the cost function for a single training example is $C_i = \frac{1}{2} \parallel y_i - a^L \parallel^2$ where L is the number of layers in the network and $a^L = a^L(x_i) = \hat{y}_i$ is the vector of activations output from the network when x_i is the input.

To get the partial derivatives backpropagation is based on four equations.

- The error in the output layer

$$\delta^L = \nabla_a C \odot f'(z^L) \tag{2.12}$$

  where δ^L is the vector whose components are the error $\delta^L_j \equiv \partial C/\partial z^l_j$ of neuron j in layer l, $\nabla_a C$ is defined to be a vector whose components are the partial derivatives $\partial C/\partial a^L_j$, $\odot$ is the elementwise product of two vectors and $f^{'}$ is the derivative of the activation function. In the case of MSE (cf. Equation 2.7), $\nabla_a C = a^L - y$ and so Equation 2.12 can be written:

$$\delta^L = (a^L - y) \odot f'(z^L). \tag{2.13}$$

- The error in the layer l in terms of the error in the layer $l+1$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(z^l), \tag{2.14}$$

  where $(w^{l+1})^T$ is the transpose of the weight matrix w^{l+1} for the $(l+1)^{th}$ layer.

- The rate of change of the cost for any bias in the network

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j. \tag{2.15}$$

- The rate of change of the cost for any weight in the network

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j. \tag{2.16}$$

With these four equations, the backpropagation algorithm can be explicitly written as:

1. **Input** x

   Set the corresponding activation a^1 for the input layer.

2. **Feedforward**

   For each $l = 2, 3, ..., L$ compute the activations a^l and weighted inputs z^l using Equation 2.10.

3. **Output error** δ^L

   Compute the vector δ^L using Equation 2.12.

4. **Backpropagate the error**

   For each $l = L-1, L-2, ..., 2$ compute the vectors δ^l with Equation 2.14.

5. **Output**

   Compute the gradient of the cost function using Equations 2.15 and 2.16.

With this algorithm, it is possible to compute the gradient of the cost function for a single example. However, training a network consists of modifying the weights and biases to minimize the cost function. This minimization can be performed utilizing a variety of optimization methods such as stochastic gradient descent.

Gradient descent methods

Gradient descent methods are iterative methods utilized to minimize a differentiable cost function. It works by using the gradient of the cost function to move in the opposite direction.

Suppose that C is a function of m variables, $v_1, v_2, ..., v_m$. Then a change ΔC in C produced by a small change $\Delta v = (\Delta v_1, \Delta v_2, ..., \Delta v_m)^T$ is:

$$\Delta C \approx \nabla C \cdot \Delta v. \tag{2.17}$$

With the following choice:

$$\Delta v = -\lambda \nabla C, \tag{2.18}$$

where λ is a small, positive parameter (called the *learning rate*). Then Equation 2.17 becomes:

$$\Delta C \approx -\lambda \nabla C \cdot \nabla C = -\lambda \parallel \nabla C \parallel^2 . \tag{2.19}$$

As $\| \nabla C \|^2 \geq 0$, this guarantees that $\Delta C \leq 0$. Therefore, this gives a way to follow the gradient to a minimum by repeatedly applying the update rule:

$$v \to v' = v - \lambda \nabla C. \tag{2.20}$$

To train a neural network, the idea is to apply the update rule of gradient descent to find the weights and biases which minimizes the cost function given by Equation 2.7. Hence, the update rule can be rewritten for weights as:

$$w^l_{jk} \to w'^l_{jk} = w^l_{jk} - \lambda \frac{\partial C}{\partial w^l_{jk}} \tag{2.21}$$

and for the biases as:

$$b^l \to b'^l = b^l - \lambda \frac{\partial C}{\partial b^l}. \tag{2.22}$$

With this, it is reasonable to hope that a network can be trained successfully. However, even if the backpropagation algorithm is efficient, computing the gradient of the cost function can take quite some time. Indeed, the idea is to compute the gradients ∇C_i separately for each training example then average them, $\nabla C_i = \frac{1}{N} \sum_i \nabla C_i$. Therefore, when the number of training inputs is very large the computation of the gradient can take a long time and so does the training. To increase the learning speed, a variation of gradient descent called *stochastic gradient descent* can be used. In this case, the idea is to estimate the gradient ∇C by computing ∇C_i for a small sample (called a *mini-batch*) of randomly chosen training inputs.

Formally, let us consider a mini-batch constituted by randomly picking m examples in the training data set: $X_1, X_2, ..., X_m$. Provided that the sample size m is large enough, the expectation is that the estimate of the gradient over the mini-batch is roughly equal to the gradient computed on the whole training set:

$$\frac{\sum_{j=1}^{m} \nabla C_j}{m} \approx \frac{\sum_i \nabla C_i}{N} = \nabla C. \tag{2.23}$$

Then, the update rule for the weights can be rewritten as:

$$w^l_{jk} \to w'^l_{jk} = w^l_{jk} - \frac{\lambda}{m} \sum_j \frac{\partial C_j}{\partial w^l_{jk}}, \tag{2.24}$$

and for the biases as:

$$b^l \to b'^l = b^l - \frac{\lambda}{m} \sum_j \frac{\partial C_j}{\partial b^l}, \tag{2.25}$$

where the sum is over all the training examples X_j in the current mini-batch. The training is performed by repeatedly picking new mini-batches to train. Once all training examples have been used in a mini-batch an *epoch* of training has been completed and the training can be continued by starting a new training epoch.

Validation and test of the network

An issue that can be encountered while training a neural network is called *overfitting*. Overfitting is a modeling error that occurs when a model is forced to fit almost perfectly a limited set of data points at the price of the underlying behavior. For instance, when trying to fit 3 data points on a parabola with a polynomial of degree 10, there is a high likelihood that the resulting polynomial will fit very closely the data points but will have large oscillations between the points. When applied to data outside of the sample, the polynomial will make large errors. In all cases, it is important to test a model against data that is outside of the data used to develop it.

During the training of a neural network under supervised learning, it is customary to have validation and test steps to control that the network is properly generalizing instead of overfitting. These steps are performed using two sets (called *validation dataset* and *test dataset* respectively) which have the same statistics as the training dataset but were not used during the training. The validation set is used to regularly evaluate the network performances. When the cost function decreases on the training set and the validation set, this means that the training is going well. However, if the cost function decreases on the training set but increases on the validation set then the network is probably overfitting and the training should be stopped. The performances on the validation set can also be used to compare the performances between different networks i.e. the network with the lowest cost function on the validation set is selected as the best. Since this approach can itself lead to some overfitting to the validation set, the test set should be used as a final performance evaluation of the selected model.

2.1.1.3 Reinforcement Learning

There exist many algorithms that implement the concept underlying reinforcement learning but their descriptions go well beyond the scope of this work. Here, the aim is to give the reader an intuitive understanding of reinforcement learning, that is, what are the requirements and results that can be expected from this endeavor so that it can be discussed in the context of particle accelerator control (cf. Section 2.3). For a comprehensive introduction, the reader is referred to the book by Sutton and Barto [72].

In reinforcement learning, the aim is for an *agent* to learn what to do in an *environment* (how to map situations to actions) to maximize a numerical reward signal. Contrary to supervised learning, reinforcement learning does not require a set of labeled examples. In this paradigm, the environment is typically considered as a large Markov decision process (fully or, as in most cases, partially observable) whose exact mathematical model is not known and exact methods are not feasible. Learning occurs by allowing the agent to interact with the environment following a balance between *exploration* (trying new behaviours) and *exploitation* (following already known behaviors).

More formally, let us define a set of environment and agent states S and a set of actions A that the agent can perform. The probability of transition (at time t) from

state $s \in S$ to state $s' \in S$ under action $a \in A$ is given by:

$$P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a). \tag{2.26}$$

An immediate reward is associated to this transition $R_a(s, s')$.

A reinforcement learning agent interacts with its environment in discrete time steps. At each time t, the agent receives the current state s_t and reward r_t. Then, the agent chooses an action a_t to perform which is subsequently sent to the environment. Under the effect of the action, the environment changes from state s_t to a new state s_{t+1} and the reward r_{t+1} is determined for the transition (s_t, a_t, s_{t+1}). The goal of the agent is to learn a *policy* $\pi : A \times S \to [0, 1], \pi(a, s) = Pr(a_t = a | s_t = s)$ which maximizes the expected cumulative reward.

To maximize the cumulative reward, the agent may have to take actions that are unfavorable in the short-term but favorable in the long-term. This makes a reinforcement learning agent particularly well suited to problems with long-term versus short-term reward trade-offs such as playing Go [73], checkers [74] or backgammon [75], controlling robots [76, 77] and many other applications often reaching similar or better performances than humans [78].

The use of reinforcement learning in large environments is possible in the following situations:

- when a model of the environment is known, but an analytic solution is not available,
- when a simulation model of the environment is given or
- when the only way to collect information about the environment is to interact with it.

2.1.2 Particle Swarm Optimization

Optimization is a process in which the best element (concerning some criterion) is selected from some set of available candidates. Optimization problems arise in all quantitative disciplines and the development of solution methods has been of interest for centuries. [79]

PSO was first introduced by Kennedy and Eberhart in 1995 [80] as an optimization algorithm for nonlinear functions. The fundamentals of PSO are "nature-inspired" as they are based upon the behavior of animals hunting for food, such as a school of fish or a flock of birds.

Let us consider a simple 2D coordinate system wherein a *population* (or *swarm*) of *particles*[1] are given random starting coordinates (x, y) an velocities $\vec{v} = v_x + v_y$. At these coordinates, the value of a function is computed $f(x, y) = s$ and assigned as the *performance* or *score* of the particle at that position, this means that particles represent potential solutions to the optimization problem for the function $f(x, y)$. For every

[1] Note that here particles have nothing to do with real physical particles.

iteration, the positions and velocities of the particles are updated according to three contributions: an inertia-like term based on their previous velocities, an individual bias term based on their own "best memory" p_i and a collective bias term based on the "best memory" of the swarm g. Then, the value of the function is computed for the new particle positions and the "memories" are updated. The hope after several iterations is that most particles have converged to the same position and that this position is the global minimum of the function.

More formally, let us consider, that in a search space of dimension j, a particle i can be assigned a position $\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^j)$ and a velocity $\vec{v}_i = (v_i^1, v_i^2, ..., v_i^j)$ and that at any position in the search space, a function $f(\mathbf{x})$ can be evaluated. For a swarm with n particles, the algorithm principle is based on the following steps:

1. **Initialize the swarm**

   For each particle $i = 1, 2, ..., n$:

   - randomly pick a position $\mathbf{x}_i$ and a velocity $\vec{v}_i$
   - assign the current position as the best known position: $\mathbf{p}_i \leftarrow \mathbf{x}_i$
   - find the swarm's best known position: if $f(\mathbf{x}_i) < f(\mathbf{g})$ then $\mathbf{g} \leftarrow \mathbf{p}_i$

2. **Update the particles velocities**

   For each particle $i = 1, 2, ..., n$:

   - pick random numbers: r_p, $r_g \sim U(0, 1)$
   - update the particle's velocity: $\vec{v}_i \leftarrow \omega\vec{v}_i + \psi_p r_p(\mathbf{p}_i - \mathbf{x}_i) + \psi_g r_g(\mathbf{g} - \mathbf{x}_i)$

3. **Update the particles position**

   For each particle $i = 1, 2, ..., n$:

   - update the particle's position: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \lambda\vec{v}_i$

4. **Update the particles best known position**

   For each particle $i = 1, 2, ..., n$:

   - update the particle's best known position: if $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ then $\mathbf{p}_i \leftarrow \mathbf{x}_i$
   - update the swarm's best known position: if $f(\mathbf{p}_i) < f(\mathbf{g})$ then $\mathbf{g} \leftarrow \mathbf{p}_i$

5. **Loop back to step 2 until a termination criterion is met**

where, ω, ψ_p and ψ_g are parameters set by the user to tune the behavior and the efficacy of the PSO method [81–83]. The parameter λ represents the learning rate, that is the proportion at which the velocity affects the movement of the particle. The termination criterion can be the number of iterations performed or that a solution with a suitable function evaluation has been found.

Multi-Objective Particle Swarm Optimization

The algorithm described above is for the optimization of a single objective but there exist variants able to optimize multiple objectives simultaneously (often referred to as *MOPSO*) [84, 85]. In this case, the algorithm is modified to use the concept of Pareto dominance to determine and keep in an archive all the non-dominated solutions encountered during the optimization process to approximate the Pareto optimum of the problem i.e. the algorithm determines all the best trade-offs between the objectives.

More formally, let us consider an optimization problem with k objective functions $\mathbf{f} = (f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_k(\mathbf{x}))$. Typically, there does not exist a solution that minimizes all objectives simultaneously. Therefore, the goal is to find Pareto optimal solutions, that is solutions that cannot be improved in any of the objectives without degrading at least one of the other objectives. With this approach, a solution $\mathbf{x}_1$ is said to (Pareto) dominate another solution $\mathbf{x}_2$ if:

1. for all $i = 1, 2, ..., k$: $f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2)$ and
2. for at least one $j \in \{1, 2, ..., k\}$: $f_j(\mathbf{x}_1) < f_j(\mathbf{x}_2)$.

Then a solution $\mathbf{x}^*$ is called Pareto optimal if it is not dominated by any other solution. The set of Pareto optimal solutions is called the *Pareto front*.

To accommodate this concept with PSO, a MOPSO algorithm maintains a list of the solutions on the Pareto front. The swarm's best known position is then ambiguous, that is all solutions on the Pareto front could be selected to be the swarm's best solution. In practice, at every iteration, each particle will be assigned its own *leader* which is a solution selected from the Pareto front. The way this selection is made has a strong influence on the efficacy of the algorithm [86], notably to avoid a premature convergence of the swarm to a local Pareto front and to promote the diversity of solutions in the Pareto front.

In this work, the leaders were selected following a probability distribution proportional to the inverse of the local density of the Pareto front. This means that Pareto optimal solutions were preferentially selected if their neighborhood was less crowded hence promoting the explorations of parts of the Pareto front that are not well known yet.

2.2 State of the Art

In the previous section, the principles of Machine Learning with neural networks and PSO and its multi-objective variant have been introduced. Although they have different applications, both have attracted attention from the particle accelerator community. Although still in the early adoption phase, both methods have been applied successfully several times before. This section is dedicated to the description of a few examples of such applications in the field of particle accelerators.

2.2.1 PSO examples

Multi-objective particle swarm and genetic algorithm for the optimization of the LANSCE linac operation

The following work is reported by Pang and Rybarcyk in the journal Physics Research A in 2013 [87].

At the Los Alamos Neutron Science Center (LANSCE) [88], 750-keV H^+ and H^- beams are first transported through separate LEBTs, then to a common LEBT called transport D (TD) which leads both beams to the drift tube linac (DTL) consisting of four RF tanks at 201.25 MHz. The DTL accelerates the beams to 100 MeV to a beam transport section that directs the H^+ beam to an isotope production facility and the H^- beam to a coupled-cavity linac (CCL) to be accelerated up to 800 MeV. Due to the excessive beam tails that come with the partially bunched beam and strong space charge effects at the entrance of the DTL, considerable efforts are put to try and minimize the loss in the machine and optimize the beam quality by tuning in a high dimensional parameter space. Therefore, they used both a multi-objective genetic algorithm (MOGA) and MOPSO to optimize the machine setpoints for the H^+ beam as simulated by a GPU-accelerated version of the beam dynamics simulator PARMILA [89]. They tried two different optimizations described thereafter.

- Beam matching into the DTL

  Here, the objective is to improve the beam matching into the DTL to minimize the transverse emittance growth and reduce the beam halo formation. To do so, they simulated a beam from the first of the two buncher cavities through twelve quadrupole focusing magnets in the LEBT to the end of the third quadrupole in the DTL tank 1. For the optimization, they used a figure of merit known as the mismatching factor given by Equation 2.27. It is calculated based on the Twiss parameters across one period of the focusing lattice, i.e. at the first and the third quadrupoles in tank 1.

$$mmf = \sqrt{\frac{1}{2}(r + \sqrt{r^2 - 4})} - 1 \tag{2.27}$$

with $r = \beta_1\gamma_2 + \beta_2\gamma_1 - 2\alpha_1\alpha_2$ where α_i, β_i and γ_i are the Twiss parameters calculated at the quadrupoles ($i = 1$ for the first quadrupole and $i = 2$ for the third quadrupole).

The goal of the matching process is to minimize the beam losses and the average of the horizontal and vertical mismatch factors by tuning the gradients of the last four quadrupoles in the TD LEBT (right before the DTL). They showed that both algorithms reached a minimal beam loss very fast but that their implementation of MOPSO performed systematically better than their MOGA to minimize the mismatch factor. Figure 2.5 shows the resultant transverse RMS size of the beam using the final solution of MOPSO compared with those calculated using one of the operational settings taken in 2012. They showed that the beam is much better matched and the mismatch factor is reduced by almost five times.

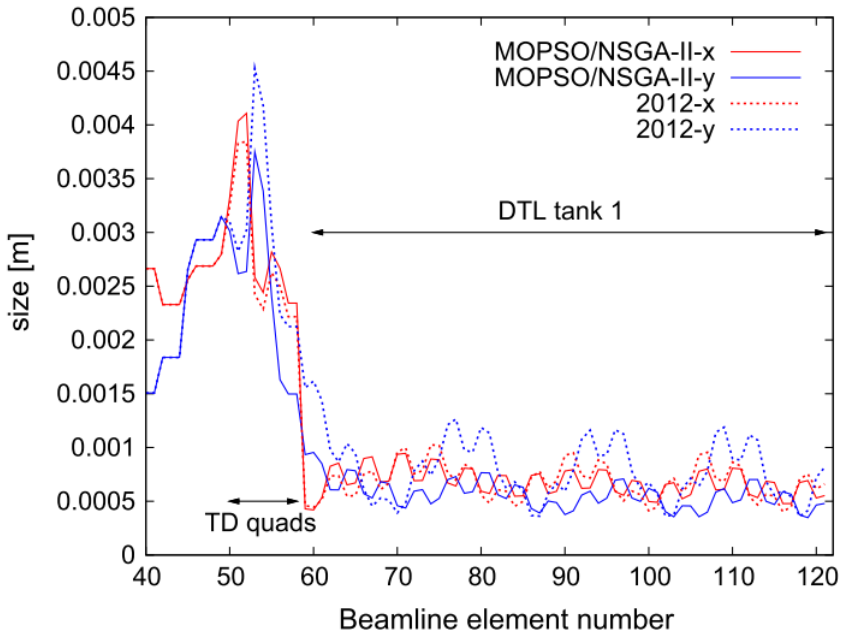

Figure 2.5: Transverse beam size along the accelerator with the optimized configuration found with the optimization algorithms (solid lines) and with the 2012 linac setting. The mismatch factor is 0.74 for the 2012 linac setting and is 0.15 for the MOPSO final setting. The aperture of the beam line in the LEBT region is 2.54 cm and 0.75 cm in tank 1 of the DTL. [87]

- DTL and buncher tuning

  Here, the objective is to adjust the RF field phase and amplitude setpoints for the two bunchers in the LEBT and the four DTL tanks to get the best beam quality at the end of the DTL and also to minimize the total radiation produced by beam loss along the linac.

In this machine, beam quality is mostly dominated by longitudinal beam dynamics. So, the first goal for the algorithms was set to minimize the total emittances of a beam in the longitudinal direction (calculated using 95 % of the beam instead of the r.m.s. emittance to better represent the beam halo and tail). The second goal is to minimize the beam losses after tank 2 to minimize the radiation. The third goal is to get a phase width under 0.1 rad with respect to 201.25 MHz to allow the beam to pass through the CCL. The algorithms had 11 parameters to optimize: the phase and amplitude setpoints of two bunchers and four DTL tanks except for the phase setpoint of DTL tank 1 which serves as a reference point for all the other phase setpoints. They were under two constraints: the transmission rate at the end of the DTL had to be above 75 % and the average final energy of the beam had to be within the range of 100 ± .02 MeV.

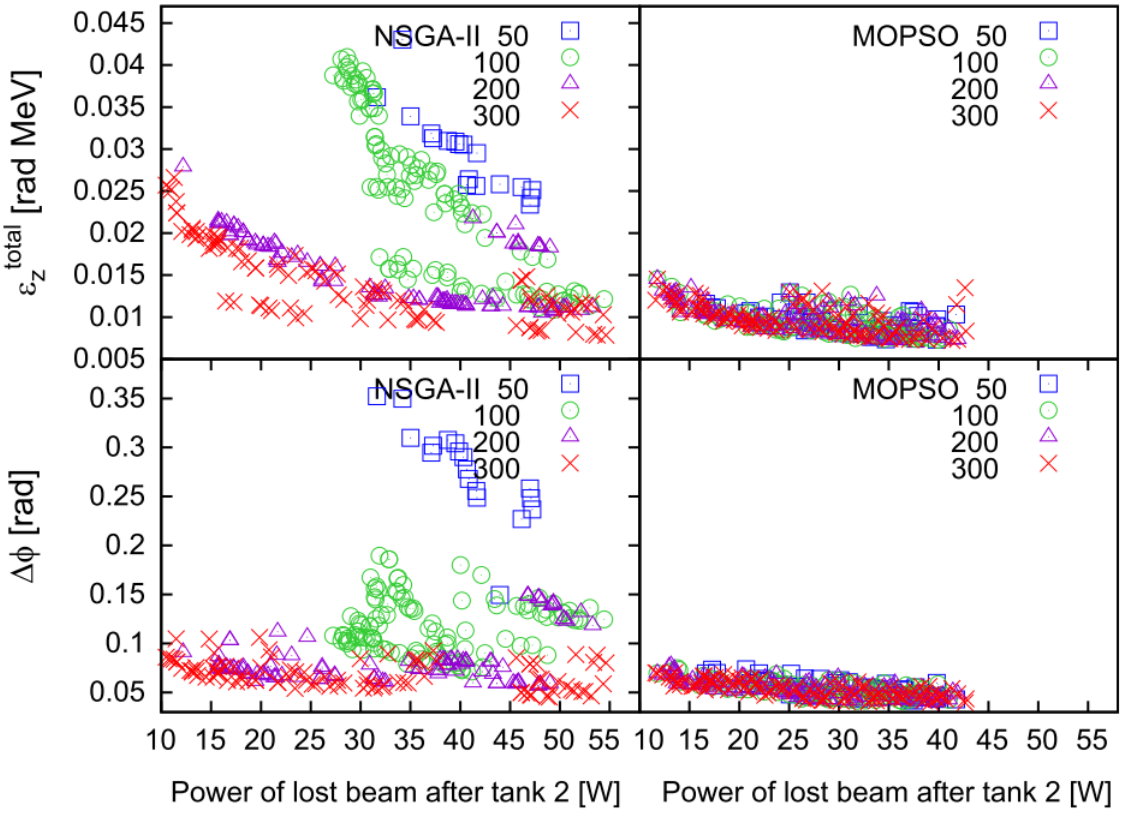

Figure 2.6: The 2D projections of the estimated 3D Pareto front in the objective space obtained by the NSGA-II (Nondominated Sorting Genetic Algorithm) and MOPSO at different iterations (50, 100, 200 and 300). [87]

The 2D projections of the Pareto front at different iterations for both algorithms are shown in Figure 2.6. They showed that both methods produced similar Pareto fronts but MOPSO converges faster. The longitudinal phase space of one of the final solutions indicates a phase width of 0.06 rad, well below the required 0.1 rad. Also, the final transmission rates range from 78 % up to 90 % and

beam losses after tank 3 are well below 0.25% which is lower than their latest measured value of 0.75 %. They concluded that MOPSO provided a variety of interesting solutions but that experimental validation was required to confirm their feasibility.

Online multi-objective optimization at Diamond Light Source

The following work is reported by Apollonio, Fielder, Martin, Bartolini, Rogers and Henderson in the 2018 proceedings of the 9th International Particle Accelerator Conference (IPAC) [90].

Diamond [91] is a synchrotron light source at wavelengths ranging from X-rays to far infrared. It uses an electron beam of up to 300 mA accelerated at an energy of 3 GeV around a 561.6 m circumference storage ring. Initially, the electron beam is formed by a 90 keV electron gun then accelerated to 100 MeV by a linac before entering a booster synchrotron to increase its energy up to the 3 GeV before injection into the storage ring. The performance of the machine is usually measured in terms of injection efficiency (IE), lifetime (LT) or residual peak to peak horizontal oscillations in the stored beam (PPX).

The problem lies in the fact that optimizing the machine requires to privilege LT against IE or PPX against IE. To improve on single-objective optimization techniques or 1D and 2D parameter scans, Appolonio *et al.* have implemented several multi-objective optimization algorithms: a Genetic Algorithm (MOGA) [92, 93], a Simulated Annealing (MOSA) [92, 94] and a Particle Swarm Optimizer (MOPSO) [87, 90]. To illustrate the effectiveness of the algorithms, they performed the following tests.

- Injection Efficiency against Residual Horizontal Oscillations in the Stored Beam (IE, PPX)

  Here the goal is to maximize IE and to minimize PPX. They used four parameters: the two last horizontal steering magnets in the booster transfer line and the amplitude and the timing of a pinger magnet used to combat the residual kicks. For the test, they initially degraded the performance of the machine with a low IE (15 to 20 %) and a large PPX (1200 to 1500 μm). The three algorithms were then put to test with initial settings to equalize their execution time (about 1 hour each). The results are summarized in Figure 2.7.

  They showed that all three algorithms were able to restore the system from an initially bad configuration to the typical operating mode (IE of 82 % and PPX of 1280 μm). However, MOSA provided a low variety of solutions compared to MOGA and MOPSO seems to perform best with a large variety of solutions and a more dominating front.

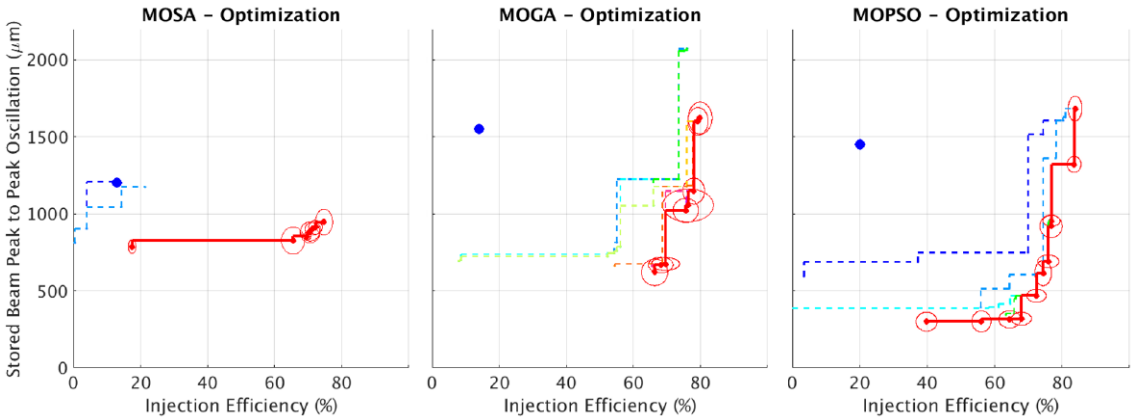

Figure 2.7: Front evolution for the (IE,PPX) optimization with three different algorithms: MOSA (left), MOGA (center) and MOPSO (right). The initial setting (blue dot) evolves through intermediate fronts (dashed lines) to the final front (red line). [90]

- Injection Efficiency and Lifetime (IE, LT)

  Here the goal is to maximize both IE and LT by acting on the sextupoles (divided into six families) of the storage ring. As the LT direct measurement needs long settling times, they used a proxy variable based on the beam loss rate monitored by a photo-multiplier tube (PMT). They defined a LT proxy variable using a measure of the PMT rates as a function of the beam current I_b at the start of an optimization run as:

$$LT_{proxy} = \frac{PMT_{rate}^{meas}(I_b)}{PMT_{rate}^{calib}(I_b)} \frac{\sigma_y^{calib}}{\sigma_y^{meas}}. \tag{2.28}$$

  For a given machine, Equation 2.28 shows that LT_{proxy} should be equal to one for all current, with deviations expected when sextupoles are altered. A degradation of the LT would then appear as $LT_{proxy} > 1$. Then, the goal becomes to maximize IE and minimize LT_{proxy}.

  Here too, they degraded the nominal configuration with (IE, LT_{proxy} = (83 %,1) which corresponds to LT = 13.2 hrs at 260 mA. They altered the harmonic sextupole strength by 5 % to obtain an initial setting with (IE, LT_{proxy} = (43 %,1.8) or a LT = 6.6 hrs. They performed a first MOPSO run with 3 generations and 25 individuals, picked a solution on the front as a new starting point for a second MOPSO run terminating with the final front shown in Figure 2.8. They showed that a spoiled machine can be restored to its standard performance and believe that they may be operating the machine at its best as they could not find a better configuration so far.

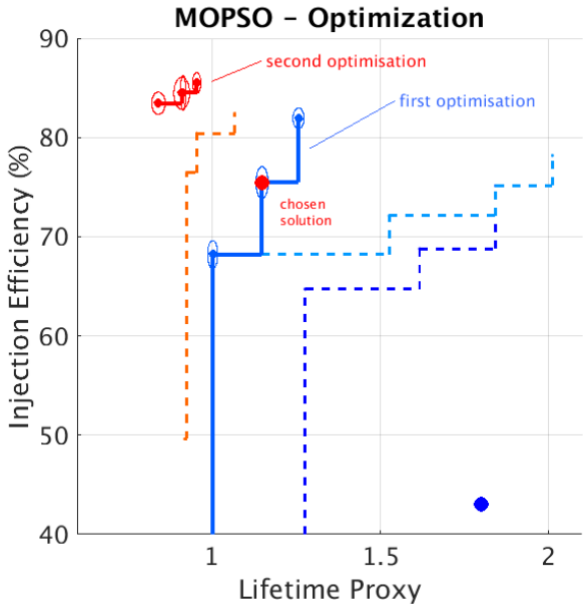

Figure 2.8: (IE,LT_{proxy}) two-step optimization with MOPSO. [90]

Conclusions

With these works, it appears clear that multi-objective optimization methods are tools that should be available to operators. Indeed, manually scanning configuration spaces with high dimensionality to find suitable setpoints is time consuming and the resulting setpoints may not be optimal. The MO methods and in particular MOPSO have proved to be useful, both off line optimizer using beam dynamics simulations and on line when directly plugged on the control system of the accelerator, to find the Pareto front of complex parameters such as the lifetime of a beam in a storage ring, the injection efficiency in a DTL.

2.2.2 Neural Networks examples

Model of the PXIE RFQ cooling system and resonant frequency response

The following work is reported by Edelen *et al.* in the proceedings of IPAC 2016 [95].

As part of the Proton Improvement Plan-II (PIP-II) [96] Injector Experiment (PXIE) accelerator, a four-vane RFQ accelerates a 30 keV, 1 mA to 10 mA H^- ion beam to 2.1 MeV. It is designed to operate at a frequency of 162.5 MHz at any duty cycle (including CW mode). The resonant frequency is controlled solely by a water-cooling system. As large differences in duty factor are expected, the RF heating is also expected to vary significantly resulting in variable detuning of the RFQ. The RF

amplifiers have enough power to maintain the field when the cavity is out of tune by up to 3 kHz. However, the measured, uncontrolled change in resonant frequency after a roughly 5°C reduction in the cold supply water temperature can go up to 50 kHz. To compensate for these detunings, a model predictive control is expected to be used. The controller would use measurements from the water system and the RF system to plan future sequences of cooling settings. Edelen *et al.* trained a neural network to serve as the model for this controller.

The inputs of the model were the temperature of the water entering each cooling sub-circuit and returning from the RFQ, the two flow control valve read-backs, the ambient temperature and humidity and a measure proportional to the power entering the cavity. Due to the time dependency of the problem, 30 minutes of previous system data were provided with a decaying sample interval. The output of the model was the predicted resonant frequency of the RFQ.

They used a simple feedforward network architecture with two hidden layers of 25 and 7 neurons respectively. The activation function of the hidden neurons was:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1, \tag{2.29}$$

and the output neuron used a linear activation function.

The training used cross-validation. The testing data consisted of a 2D scan over vane valve settings and RF field amplitudes under a higher constant wall valve setting than was seen during training (99 % open for the test data while the highest value seen during training was 75% open).

Figure 2.9 shows the best performing network they obtained with a mean absolute prediction error of 346 Hz on the test set, 98 Hz on the validation set and 116 Hz across all data (training, validation and test data).

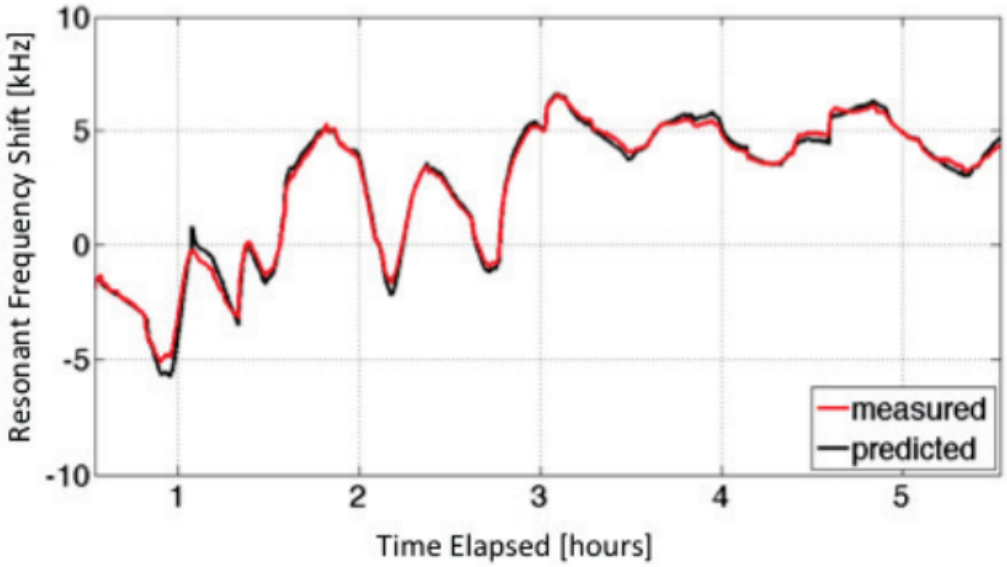

Figure 2.9: Measured and predicted resonant frequency values for the PXIE RFQ as a function of time. [95]

Rapid switching between beam parameters in a free electron laser

Another work reported by Edelen *et al.* in the proceedings of the 38th International Free Electron Laser Conference in 2017 concerns the rapid switching between configurations of a free-electron laser to obtain different beam parameters [97].

Free electron laser (FEL) facilities must accommodate requests for a variety of electron beam parameters to supply their users with the appropriate photon beam for any given experiment. Even with skilled operators tuning the machine, the time required to switch from one setting to another reduces the amount of useful experimental time. In this work, the authors explored the possibility to use neural networks to decrease the switching time. They used simulations of a compact THz FEL based on the Twente/Eindhoven University FEL (cf. Figure 2.10).

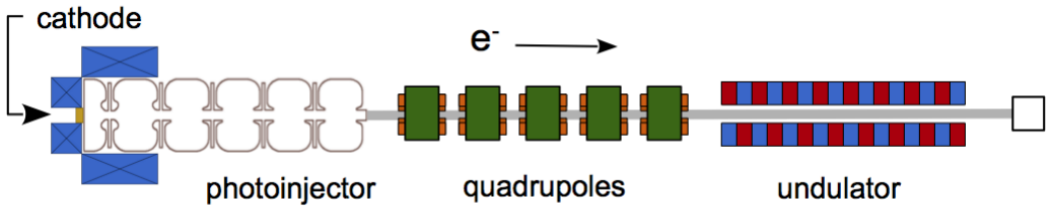

Figure 2.10: Layout of the FEL accelerator. [97]

They first trained a neural network model using supervised learning to create a surrogate for the full simulations that reproduces the relevant behavior of the FEL and can execute quickly to facilitate the training of a neural network controller trained using reinforcement learning. The model's inputs were: the RF power, the RF phase, the solenoid strength, and the quadrupole settings. The outputs were: the Twiss parameters, beam energy, emittance and transmission at the entrance of the undulator. The model consisted of four hidden layers with 50, 50, 30 and 30 hidden neurons respectively using a hyperbolic tangent activation. The performances of the trained model are given in Table 2.1 and a representative plot from the validation set is shown in Figure 2.11. These show that the trained model is reasonably accurate.

Model performance

Parameter	Train MAE	Train STD	Val. MAE	Val. STD
α_x [rad]	0.018	0.042	0.067	0.091
α_y [rad]	0.022	0.037	0.070	0.079
β_x [m/rad]	0.004	0.009	0.008	0.012
β_y [m/rad]	0.005	0.011	0.012	0.017

Table 2.1: Model performance in terms of Mean Absolute Error (MAE) and Standard Deviation (SD) between the simulated and predicted Twiss parameters. [97]

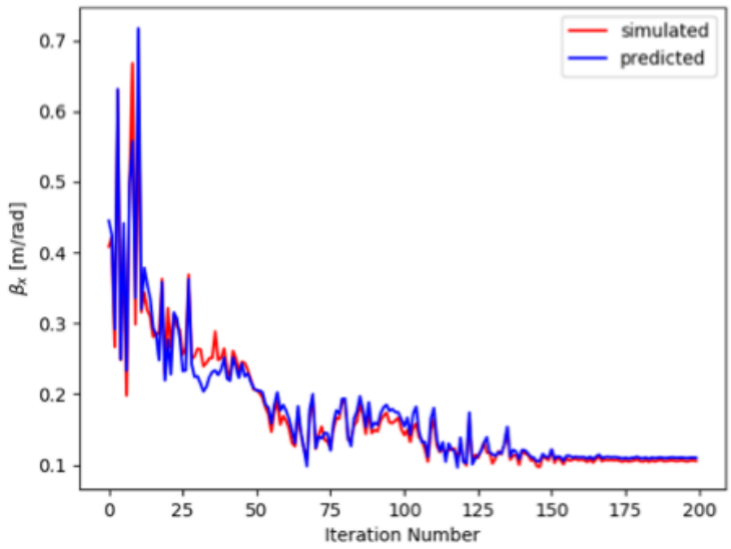

Figure 2.11: Model predictions and simulated values on the validation data set for a 5.7 MeV beam. [97]

The neural network controller was built as an inverse of the network model i.e. the inputs and outputs of the controller are respectively the outputs and the inputs of the model. This means that the controller inputs can be understood as the desired beam properties at the entrance of the undulator while its outputs represent a suggested configuration to obtain those beam properties. Note that in this initial study, the controller outputs corresponding to the RF power, phase and solenoid strength were ignored. The change in beam energy was controlled manually. The controller was first trained using the training set of the model under supervised learning then by interacting with the model under reinforcement learning.

To test the trained controller, they plugged it directly into the physics-based simulation. Given random requested energy values within 3-6 MeV and the following desired Twiss parameters: $\alpha_{x,y} = 0$ rad and $\beta_{x,y} = 0.106$ m/rad, the controller had to reach these in a single iteration. Table 2.2 shows that for a given energy, the controller manages to reach the desired beam size to within about 10 % and the beam will be close to a waist, requiring minimal further tuning to reach the target values.

Controller performance

Parameter	Train MAE	Train STD	Val. MAE	Val. STD
α_x [rad]	0.012	0.075	0.046	0.063
α_y [rad]	0.013	0.079	0.045	0.064
β_x [m/rad]	0.008	0.004	0.006	0.023
β_y [m/rad]	0.014	0.011	0.011	0.069

Table 2.2: Ability of the controller to reach $\alpha_{x,y} = 0$ rad and $\beta_{x,y} = 0.106$ m/rad for 3-6 MeV beams in one iteration.

Conclusions

Edelen and her team have pioneered the use of neural networks to build predictive models and controllers for particle accelerators. They showed that with careful training, it is possible to reproduce complex behaviors of accelerator components such as a RFQ resonant frequency under various duty cycles and to model and control the beam dynamics in a Free Electron Laser.

2.3 Summary

In Chapter 1, the fact that new particle accelerator projects have higher and higher requirements in terms of power and reliability has been introduced. This is especially true for ADS such as the MYRRHA project for which the control systems have to be improved with new tools and new strategies to meet the reliability requirements. Indeed, the Fast Fault-recovery strategy envisioned requires changing the machine setpoints in less than three seconds which is not doable with the current systems. In addition, the tuning of a machine usually requires scanning many parameters for the operator to find suitable configurations with no guarantees that there are no better configurations (cf. Chapter 3).

One limiting factor is the intrinsic limit of beam dynamics simulations to model the behavior of the real machine at low beam energy. Hence, optimizing parameters on a simulation does not guarantee that the corresponding real configuration will provide the expected beam properties. Therefore, it is clear that there is a demand for the development of new tools to:

1. optimize the configuration of a particle accelerator and
2. model with high fidelity and quick execution the behavior of a particle accelerator.

In chapter 2, MOPSO and machine learning with neural networks have been introduced and a few examples of applications on particle accelerators have been presented.

On the one hand, MOPSO seems to be a powerful multi-objective non-linear optimizer able to handle efficiently optimization problems based on beam dynamics simulations but also on real machines. Its ability to provide a Pareto front from which the operator can pick suitable configurations can be extremely valuable to confirm that previously used configurations were optimal (or at least close to being) and to find new options for the machine setpoints. However, to operate, MOPSO requires to sample the configuration space in ways that could be unexpected by the operator. This causes no issues when used on simulations but, when used on a real machine, some unexpected configurations could potentially damage the machine. In addition, the natural noise on measurements can push the swarm in wrong direction.

On the other hand, neural networks have been shown to be able to model with reasonably good fidelity the complex behavior of a RFQ based on real measurements and of a free electron laser based on simulation. These results are very encouraging to utilize machine learning and neural networks for the modelization of high-power proton linacs. However, the difficulty comes from obtaining the required data for the training because simulations do not reproduce accurately the beam dynamics and the real machine is not available for exhaustive sampling.

Considering all of this, the current work aims to explore the ability of neural networks to model proton beam dynamics in a particle accelerator. Such models would help accelerator projects such as MYRRHA to reach their objectives. Indeed, MYRRHA has to operate in CW mode which sets a very low limit for the average

beam power losses to limit the damage to the machine. To reach this requirement, it is necessary to ensure a high beam quality which is largely determined by the injector where the low energy beam dynamics is strongly affected by non-linear effects: the space charge effect and its compensation.

Hence, the next chapter is dedicated to the experimental study of the IPHI and MYRRHA injectors (cf. Sections 3.2 and 3.1). The idea is to try to obtain a model trained on simulated data and/or experimental data representative of the beam dynamics of these injectors. The target performance for the trained model is to reproduce the behavior of the real machine with a fidelity high enough to enable the utilization of MOPSO or the training of a neural network controller.

Chapter 3

Tuning and commissioning of the MYRRHA and IPHI injectors

Contents

Typical proton injector designs include a Low Energy Beam Transport (LEBT) section. Its function is to ensure reliable transport of the Direct Current (DC) proton beam from the source to a Radio Frequency Quadrupole (RFQ) and to condition the beam to ensure its proper acceleration and minimize beam losses further down the accelerator. A centered matched converging beam has to be provided at the RFQ input with reasonable transverse emittances (ideally lower than the design values) and proper Twiss parameters.

In addition, the LEBT should enable to clean the proton beam from other species also produced by the ion source such as H_2^+ and H_3^+. If these molecular ions are injected into the RFQ, they may create important parasitic losses. Therefore, it is important to intercept a maximum of these species in the LEBT.

Finally, the LEBT is responsible to control the beam current and beam time structure injected into the RFQ. Indeed, depending on the objectives, several different regimes of beam current and duty cycle can be desired. For example, during the commissioning of the accelerator, the LEBT should output a low beam current with a low duty cycle to avoid damages down the line but, during nominal operation, the LEBT will be required to output at full capacity.

This shows that tuning a LEBT is a crucial process for the correct operation of a proton accelerator. However, this is far from a trivial task. As its name indicates, a LEBT is responsible for the transport of a low energy beam that is subject to space charge effects that complicate the beam dynamics and hence the tuning of the machine. This is especially true for machines with a relatively high beam current such as MYRRHA with a 4 mA proton beam and IPHI (Injecteur de Protons à Haute Intensité at CEA Saclay) with an 80 mA proton beam. For these, the time required for the commissioning or to change the operating regime by hand typically takes a few hours up to a few days with no guarantees that the final tuning is optimal.

Using beam dynamics simulators such as TraceWin (general purpose beam dynamics simulator developed by CEA) [98] and Toutatis (specialized beam dynamics simulator for the RFQ developed by CEA) [99] may help to understand the general behavior of the machine and find a starting point to search for a suitable configuration. However, they do not reproduce accurately the beam dynamics due to the approximations used to reduce the computation time required for a run.

In this context, it is clear that improving existing tuning methods or developing new ones has the potential to reduce the time required for the commissioning and subsequent changes in operating regimes. This would then increase the availability of the machine and allow to do more science.

In this chapter, the behaviors of the LEBT of MYRRHA and IPHI are experimentally studied. This is done to:

- illustrate the cumbersomeness of the tuning process of a LEBT,
- gather experimental data to create a training data set and

- to study the difference between the simulated models of the lines and their respective real counterparts.

In Section 3.1, the commissioning of the MYRRHA LEBT at Louvain-la-Neuve is described. First, the different elements equipping the line are described. Then, the commissioning itself is illustrated. Next, the test of a Particle Swarm Optimization algorithm as online optimization is discussed. Finally, emittances measurements made to calibrate the simulated model are described.

In Section 3.2, the characterization of IPHI is shown. As for the MYRRHA LEBT, a technical description of IPHI is first given before the results are discussed.

In Section 3.3, the simulated models of the MYRRHA LEBT and IPHI are compared to their respective real machines. In particular, the efforts made to get the models as close as possible to the experimental measurements are described.

3.1 MYRRHA

As an ADS demonstrator, the accelerator of the MYRRHA project has to meet very strict reliability and stability requirements (cf. Section 1.2.2). Hence, its design includes many redundancies and safety margins. But even with these, a poor proton beam injection into the first accelerating element, the RFQ, will generate beam losses further down the accelerator. This will trigger the machine protection system which will interrupt the beam to avoid damaging the machine. Therefore, it is crucial to ensure that the LEBT is correctly configured.

The tuning of a LEBT can be difficult because the space charge effect is stronger at low energy and induces non-linear effects on the beam transport. Indeed, it is known that the space charge and its compensation are difficult to model and not reproduced in classic beam dynamics codes such as TraceWin. In addition, the properties of the beam at the exit of the proton source are not well known and can be difficult to measure as will be discussed in this chapter. For both reasons, models of a LEBT usually do not reproduce accurately the behavior of a real machine. Hence, a machine learning based model trained on experimental data would be a useful tool to improve the injector tuning (to be faster and more precise).

Initially, the LEBT of MYRRHA has been built and commissioned at Laboratoire de Physique Subatomique et de Cosmologie (LPSC, Grenoble) from 2014 to 2017. Then in 2018, the LEBT has been moved to Centre de Ressources du Cyclotron (CRC, Louvain-la-Neuve) in preparation for the coupling of the RFQ and the first CH cavities. Following the move, the second commissioning has been performed from March to June 2019. The experimental data required to train a neural network to model the LEBT of MYRRHA were collected during this second commissioning. In addition, the beam properties have been measured in the middle of the LEBT to be used to determine the input parameters of the TraceWin model of the LEBT (cf. Section 3.3.1).

This section is dedicated to the description of the LEBT, the gathering of experimental data, the results of a PSO test realized as part of its second commissioning and the emittance measurements in the middle of the LEBT. The section is organized as follows.

The description of the different elements equipping the LEBT is given in Section 3.1.1.

The objectives of the second commissioning are described in Section 3.1.2.

Then, the description of the commissioning and the LEBT characterization is given in Section 3.1.3.

Next, the results of the PSO test are summarized in Section 3.1.4.

And finally, the emittance measurements using the Allison scanners are illustrated and their precision is discussed in Section 3.1.5.

3.1.1 LEBT description and purpose

The LEBT represents the three first meters of the MYRRHA accelerator (cf. Figure 3.1) [100]. Its first purpose is to ensure the reliable transport of the DC 30 keV

proton beam from the source to the RFQ and to condition the beam for a proper acceleration that minimize beam losses in the following accelerating devices. The beam injected into the RFQ should be centered, converging, have a reasonable emittance (ideally lower or equal to $\epsilon_{RMS,norm.,proton} = 0.2$ mm.mrad which is the design value) and have the following Twiss parameters:

- $\alpha = 0.88$ and
- $\beta = 0.04 \frac{\text{mm}}{\text{mrad}}$.

In addition, the LEBT should allow the cleaning of the proton beam from unwanted species. Indeed, the ECR ion source produces protons from the ionization of a dihydrogen gas. Within this process, other species than protons (such as H_2^+ and H_3^+) are also produced and extracted from the source. When injected into the RFQ, these species may induce parasitic losses. Hence, they should be intercepted as much as possible in the LEBT.

Finally, the LEBT should allow the creation of the time structure of the beam necessary to commission the linac, to monitor the reactor sub-criticality, to operate the MINERVA linac for ISOL (Isotope Separation On-Line) experiments, and to operate the accelerator in its nominal regime. [101]

Indeed, in the linac commissioning phase, the beam power will be ramped from a few watts up to full power. To do so, the peak beam current will be increased from a few hundreds of μA up to 4 mA. Concurrently, the duty cycle will be ramped from about 0.02 % up to 100 %. This is done by starting with a 200 μs long pulsed beam at 1 Hz then increasing the repetition rate up to 250 Hz to reach CW operation. [101, 102]

Then depending on which machine is operated (the MYRRHA ADS or the MINERVA experiments), several beam time structures have to be planned.

As shown in Figure 3.1, the LEBT is equipped with the following functional elements.

- An ECR ion source

  The ion source of MYRRHA (cf. Figure 3.2) is designed and built by Pantechnik to deliver a stable proton beam current of up to 20 mA at 30 keV. Its operation is responsible to ensure a stable operation (without electrical breakdowns) with a good beam quality (without non-linear effects nor beam "distortions"). This is done by tuning the following parameters:

  - the hydrogen gas pressure in the source,
  - the RF power injected in the source and
  - the voltage of the first extraction electrode, V_{puller}.

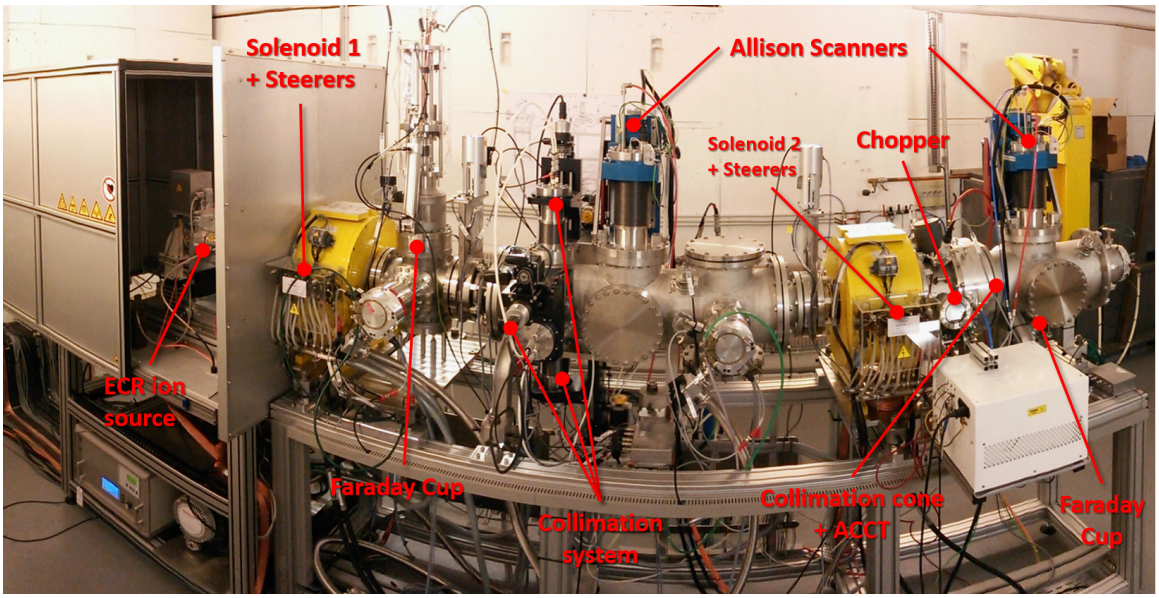

Figure 3.1: Picture of the ion source and the LEBT with its legend during its first commissioning at LPSC. [100]

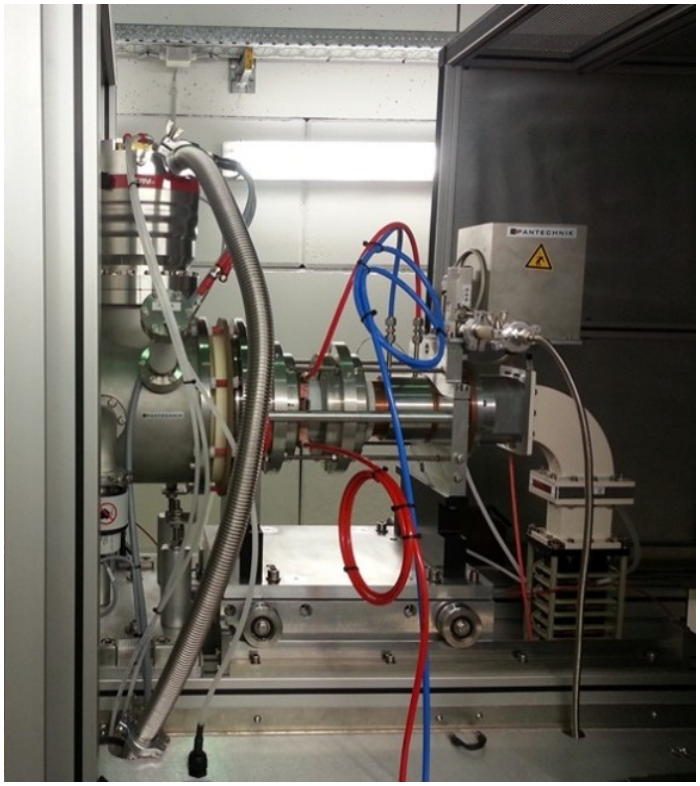

Figure 3.2: Picture of the Pantechnik ECR ion source. [103]

- Two solenoids

  By applying a current to the solenoids, a magnetic field is produced along the beam path. This field is used to focus and guide the beam while controlling the transverse beam characteristics.

- Two pairs of steerers

  By applying a current to a steerer, a magnetic field is produced perpendicular to the beam path. This field induces a deflection of the beam path used to steer the beam and thus enables the compensation of misalignments present in the LEBT. One steerer provides control in one direction. Hence pairs are required to control X-Y steering. Each pair is installed in the solenoids which induces a coupling between the effects of the solenoids and steerers.

- A collimation system

  The collimation system (cf. Figure 3.3) consists of four movable metallic fingers with rounded-in tips. Its role is to intercept part of the beam to clean it from the particles with unwanted properties (diverging too much or too far from the beam center), to intercept the unwanted ion species produces by the ECR ion source such as H_2^+ and H_3^+ and to control the beam current injected into the RFQ. The position of the fingers can be adjusted to control the intercepted fraction of the beam.

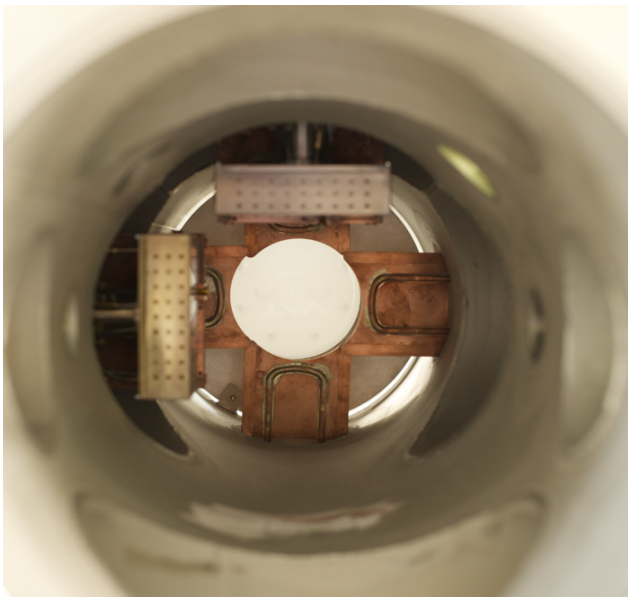

Figure 3.3: Picture of the Allison scanners (two boxes in the foreground) and the collimation system (four metallic fingers with rounded in tips in the background). [103]

- A chopper

  The chopper (cf. Figure 3.4) is used to control the time structure of the beam injected into the RFQ. It works by applying a voltage between the two electrodes to produce a static electric field to deflect the beam into the plate around the collimation cone. This plate is actively cooled to prevent heating when the beam is deflected by the chopper.

- A collimation cone

  The collimation cone (cf. Figure 3.4) is cooled with water and is the last protection system before the beam injection into the RFQ. Particles in the beam that are spread out too much will impact the collimation cone instead of entering the RFQ. This effectively limits the size and direction of the beam injected into the RFQ.

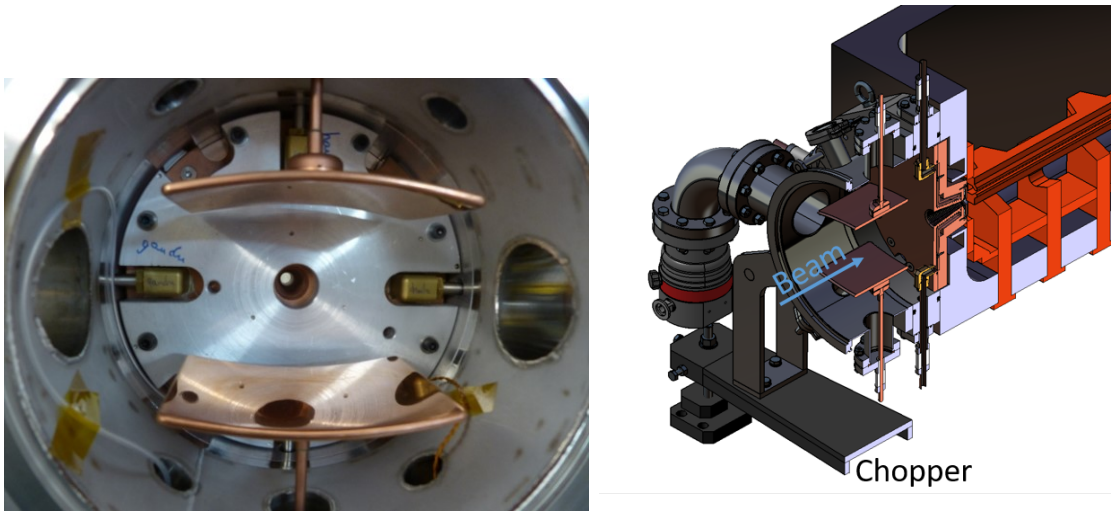

Figure 3.4: Picture and schematics of the chopper and the collimation cone. [103]

In addition, the LEBT is equipped with diagnostic tools to measure beam properties at several points of the LEBT.

- Two Faraday cups

  A Faraday cup is a copper cup that can be moved to intercept the beam to measure its current. The LEBT is equipped with two such cups. The first is installed just after the first solenoid to measure the beam current injected into the LEBT. While the second was installed after the collimation cone to measure the current that would be injected into the RFQ. Since then, it has been removed to couple the RFQ. The Faraday cups have an estimated measurement error of about 1 % [104].

- Two Allison scanners

  An Allison scanner (cf. Figure 3.3) consists of a slit followed by an electrostatic deflector then a second slit and finally a Faraday cup (cf. Figure 3.5). The first slit selects a slice of the beam. The combination of the voltage applied on the electrostatic deflector and the second slit selects a momentum. Then, the Faraday cup measures the current of the selected beam slice with the selected momentum. Hence, the emittance of the beam can be reconstructed by scanning the position of the first slit and the voltage of the electrostatic deflector. The Allison scanners can be installed either in the X or Y directions and in the middle of the LEBT or after the collimation cone (when the RFQ is not coupled).

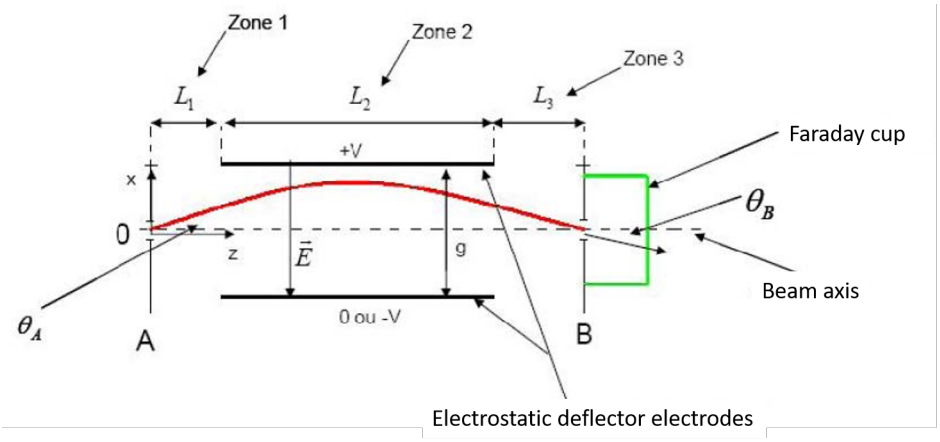

Figure 3.5: Diagram of an Allison scanner.

- An ACCT

  The active ac-transformer is used to measure transient beam current and beam pulses. It consists of the measurement of the voltage induced in a secondary circuit by the magnetic field generated by the beam (cf. Figure 3.6).

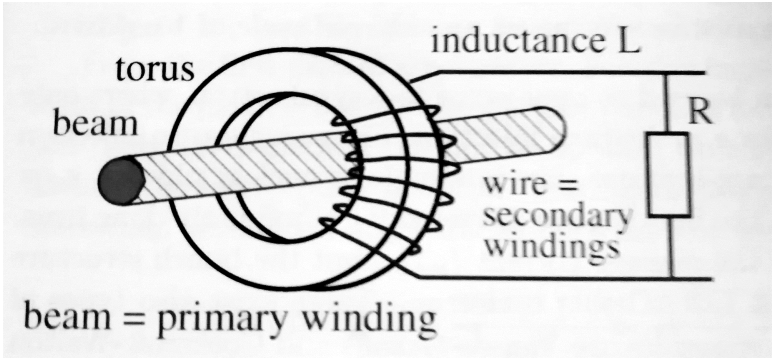

Figure 3.6: Diagram of an ACCT. [105]

- Three pressure gauges

  The pressure gauges measure the pressure of the residual gas in the chamber as well as the injected gas.

3.1.2 Objectives

The main objective of the commissioning is to optimize the beam transmission through the LEBT while ensuring that the beam has the right characteristics to be injected into the RFQ:

- reasonable emittance i.e. lower or equal to $\epsilon_{RMS,norm,proton} = 0.2$ mm.mrad (the design value),
- $\alpha = 0.88$ and
- $\beta = 0.04 \ \frac{\text{mm}}{\text{mrad}}$.

These tasks are performed by scanning the configuration space (cf. Section 3.1.3) to determine the nominal configuration which provides the expected beam properties and by testing the functionality of the equipment of the LEBT. In our case, three additional objectives were defined as follows:

1. Gather data to constitute a dataset for neural network training

   Gathering data simply means that the measurements performed during the configuration scans are kept in an archive in addition to being used to determine the nominal configuration.

2. Test a PSO implementation on the machine

   The PSO test is used to verify its ability to find a nominal configuration and to compare its performance to the performance of a human operator.

3. Calibrate the input parameters for the TraceWin model

   The experimental data are used to determine the input parameters that describe the beam at the LEBT entrance for the TraceWin model. This is necessary to try to get the numerical model as representative of the real machine as possible.

3.1.3 LEBT characterization: commissioning and data gathering

The search of the nominal configuration consists of regularly scanning the main controls of the LEBT: both solenoids, the steerers, the collimator and the amount of gas injected in the LEBT. The goal is to find a configuration that outputs about 4.2 mA at the LEBT exit (the desired beam current to be injected into the RFQ).

Ion source operation

The ion source was tuned following the setpoints determined during its first commissioning at LPSC [100]. Hence, the source body was set to 30 keV and the puller to 22.5 keV.

The injection of hydrogen in the source body was kept as low as possible while ensuring stable operation to decrease the likelihood of an electrical breakdown. The injected RF power was constantly tweaked either by hand or with the help of an automated control module to maintain the ion source output at around 8 mA.

First solenoids scan

First, the beam current has been measured at the exit of the LEBT while both solenoids were scanned from 50 A up to 110 A with a step size of 2 A with a fully opened collimator, no steering and no injection of gas. Hence for a solenoids scan, 31 steps are performed on each solenoid corresponding to 961 measurements for the whole scan this takes about 30 minutes to perform. The result is shown in Figure 3.7.

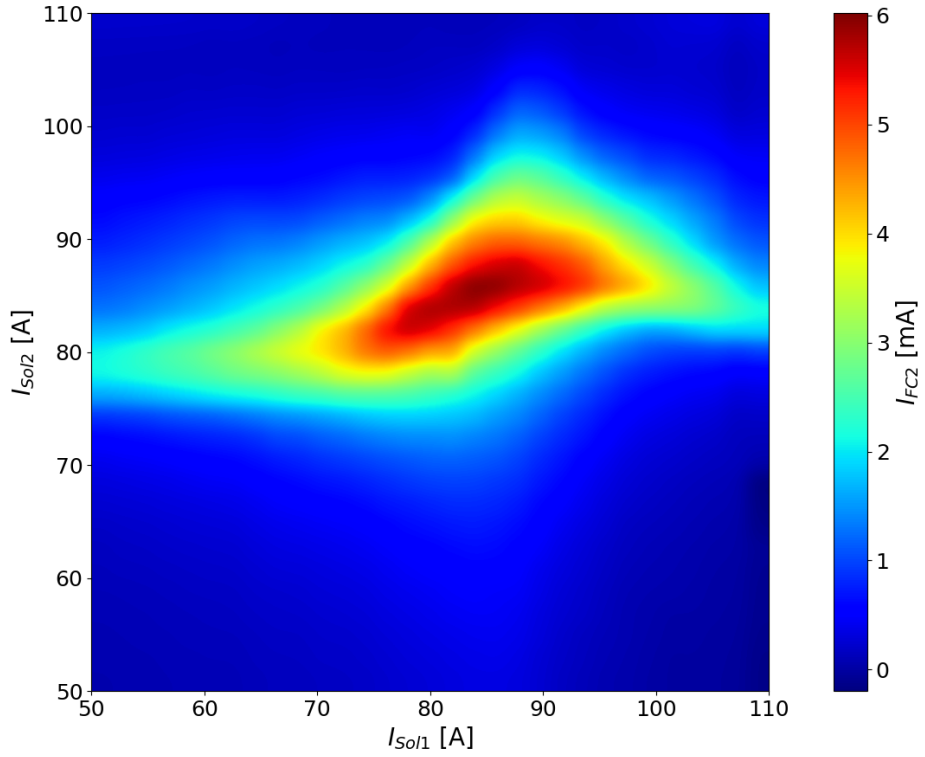

Figure 3.7: Scan over the solenoids of the beam current measured at the exit of the LEBT with a fully opened collimator, no steering and no gas injection.

setpoint	Unit	Lower bound	Upper bound	Step size
I_1^{sol}	A	50	110	2
I_2^{sol}	A	50	110	2
I_{2H}^{st}	A	0		-
I_{2V}^{st}	A	0		-
r_{col}	mm	55 (fully open)		-
Ar injection	-	No		-

Table 3.1: Parameters of the first solenoids scan.

Second solenoids scan

According to the previous commissioning, it was chosen to inject argon gas in the vacuum chamber to compensate for the space-charge effect in the LEBT [100]. Hence, for all the following experiments, argon gas has been injected in the chamber in order to maintain a measured pressure around $p_{meas} = 2 \times 10^{-5}$ mbar.

First, a new solenoids scan has been performed with gas injection. Figure 3.8 shows that the injection of argon gas into the chamber does increase the range of solenoids setpoints that provide a good transmission. In addition, it shows that the design setpoints for the solenoids ($I_1^{sol} = 65.6$ A and $I_2^{sol} = 77.9$ A) is a good candidate for the nominal configuration of the LEBT.

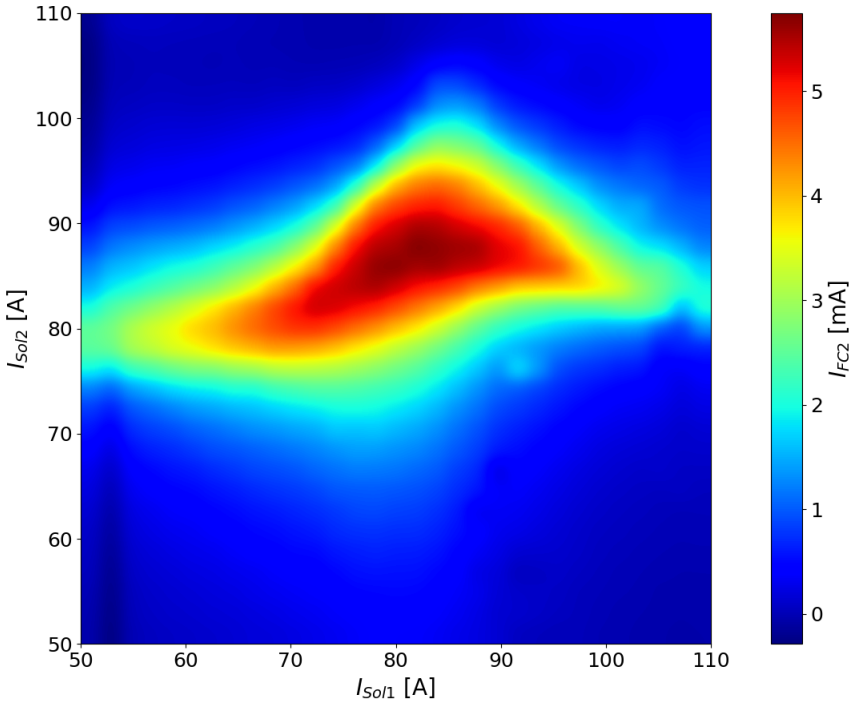

Figure 3.8: Scan over the solenoids of the beam current measured at the exit of the LEBT with Argon gas injection and without steering.

setpoint	Unit	Lower bound	Upper bound	Step size
I_1^{sol}	A	50	110	2
I_2^{sol}	A	50	110	2
I_{2H}^{st}	A	0		-
I_{2V}^{st}	A	0		-
r_{col}	mm	55 (fully open)		-
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)		-

Table 3.2: Parameters of the second solenoids scan.

Steerers scan

Then, the solenoids are set to their designed setpoints and the steerers are scanned to compensate for the unavoidable misalignments of the LEBT. In practice, the vertical and horizontal steerers installed in the second solenoid were scanned from -3 A up to 3 A with a step size of 0.5 A. This means that there were 13 steps per steerer for a total of 169 measured configurations. Although the number of measurements per steerers scan is lower than for a solenoid scan, it took about one hour to perform due to the low response time of the steerers power supply to a change in setpoints.

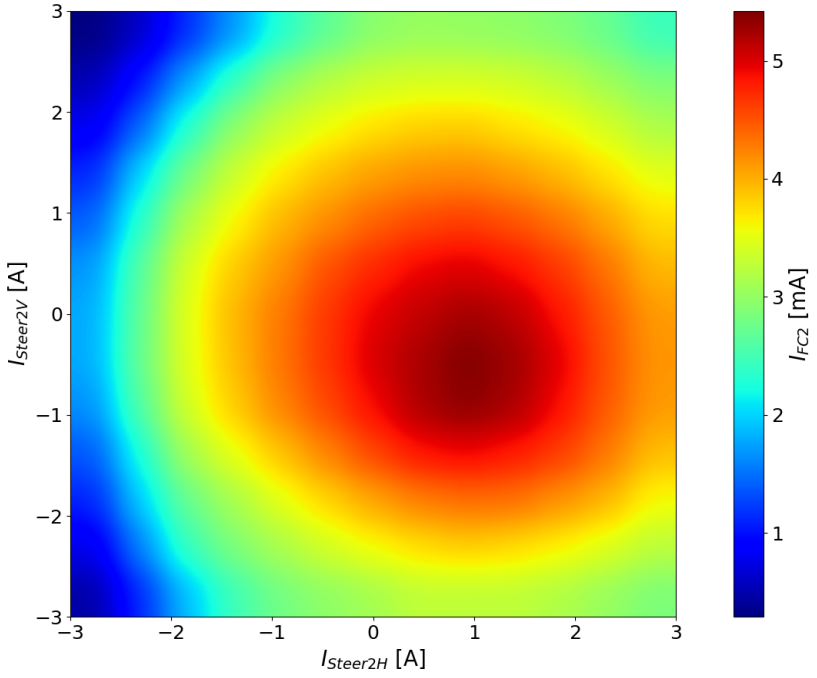

Figure 3.9: Scan of the steerers in the second solenoid with $I_1^{sol} = 65.6$ A, $I_2^{sol} = 77.9$ A.

The steerers scan aims to maximize the transmission (hence the beam current at the exit of the LEBT). The results shown on Figure 3.9 indicates that the highest transmission is achieved around $I_{2V}^{st} = -0.5$ A and $I_{2V}^{st} = 1$ A. This corresponds to relatively low steering of the beam which means that the LEBT is reasonably well

aligned. After some manual tuning around these values, the highest transmission was reached for $I_{2V}^{st} = -0.5$ A and $I_{2V}^{st} = 0.75$ A.

setpoint	Unit	Lower bound	Upper bound	Step size
I_1^{sol}	A	65.6		-
I_2^{sol}	A	77.9		-
I_{2H}^{st}	A	-3	3	0.5
I_{2V}^{st}	A	-3	3	0.5
r_{col}	mm	55 (fully open)		-
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)		-

Table 3.3: Parameters of the steerers scan.

Third solenoids scan

Next, a solenoids scan is once again performed with the optimized steerers setpoints ($I_{2V}^{st} = -0.5$ A and $I_{2H}^{st} = 0.75$ A) to verify the suitability of the designed solenoids setpoints as the nominal setpoints for further operations. The scan is shown in Figure 3.10. It shows an even better transmission than the second solenoids scan because of the optimized steering. Also with a measured beam current at the LEBT exit of about 5.25 mA, the design setpoints for the solenoids is indeed a good setting for the nominal configuration.

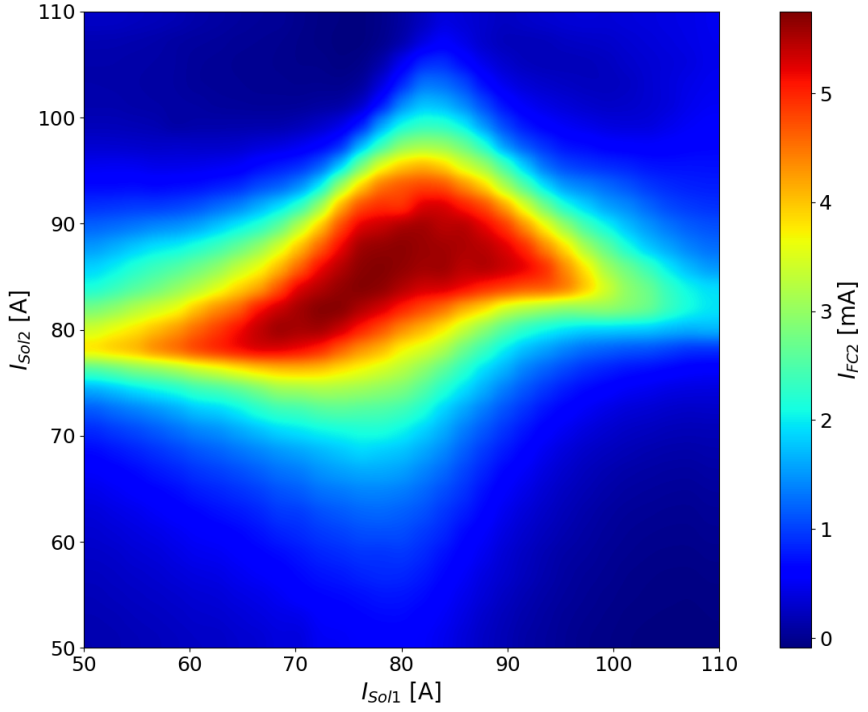

Figure 3.10: Scan over the solenoids of the beam current measured at the exit of the LEBT with Argon gas injection and steering.

setpoint	Unit	Lower bound	Upper bound	Step size
I_1^{sol}	A	50	110	2
I_2^{sol}	A	50	110	2
I_{2H}^{st}	A	0.75		-
I_{2V}^{st}	A	-0.5		-
r_{col}	mm	55 (fully open)		-
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)		-

Table 3.4: Parameters of the third solenoids scan.

Collimator scan

The next step of the commissioning is to determine the collimator opening to obtain a beam current of about 4.2 mA at the LEBT exit with the nominal configuration for the solenoids ($I_1^{sol} = 65.6$ A, $I_2^{sol} = 77.9$ A) and the steerers ($I_{2V}^{st} = -0.5$ A and $I_{2H}^{st} = 0.75$ A). A total of 7 measurements were made for this scan (cf. Figure 3.11). Following these measurements, a suitable collimator opening was determined to be $r_{col} = 14$ mm.

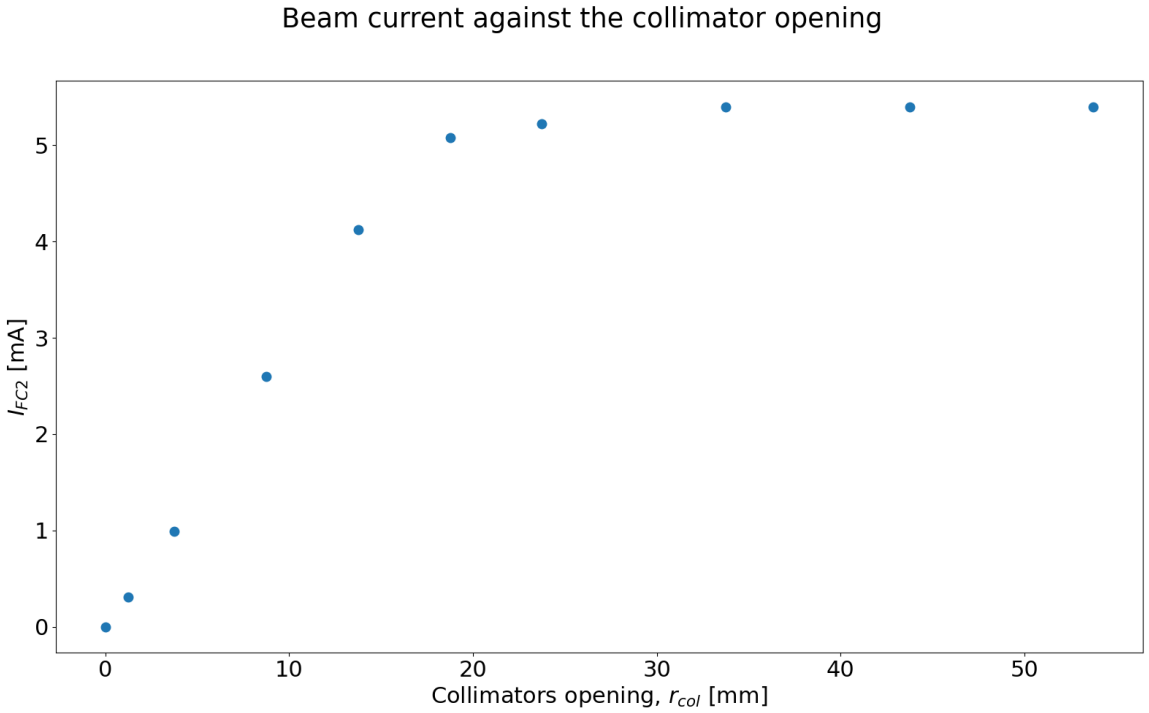

Figure 3.11: Evolution of the beam current measured with the second Faraday cup at the exit of the LEBT with regards to the collimator opening.

setpoint	Unit	Lower bound	Upper bound	Step size
I_1^{sol}	A	65.6		-
I_2^{sol}	A	77.9		-
I_{2H}^{st}	A	0.75		-
I_{2V}^{st}	A	-0.5		-
r_{col}	mm	0	55	variable
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)		-

Table 3.5: Parameters of the collimator scan.

Injection of argon gas

Finally, the injection of argon gas was scanned to confirm the choice to maintain the pressure around 2×10^{-5} mbar during the commissioning. Indeed, the amount of Ar gas injection is the result of a compromise. On the one hand, the more Ar gas is injected the faster the compensation is established i.e. the shorter the length of the path traveled by the beam before the space-charge compensation is effective. On the other hand, increasing the amount of injected gas increases the likelihood of an electrical breakdown in the ion source or the chopper. In addition, injecting too much gas would degrade the RFQ vacuum after its coupling to the LEBT.

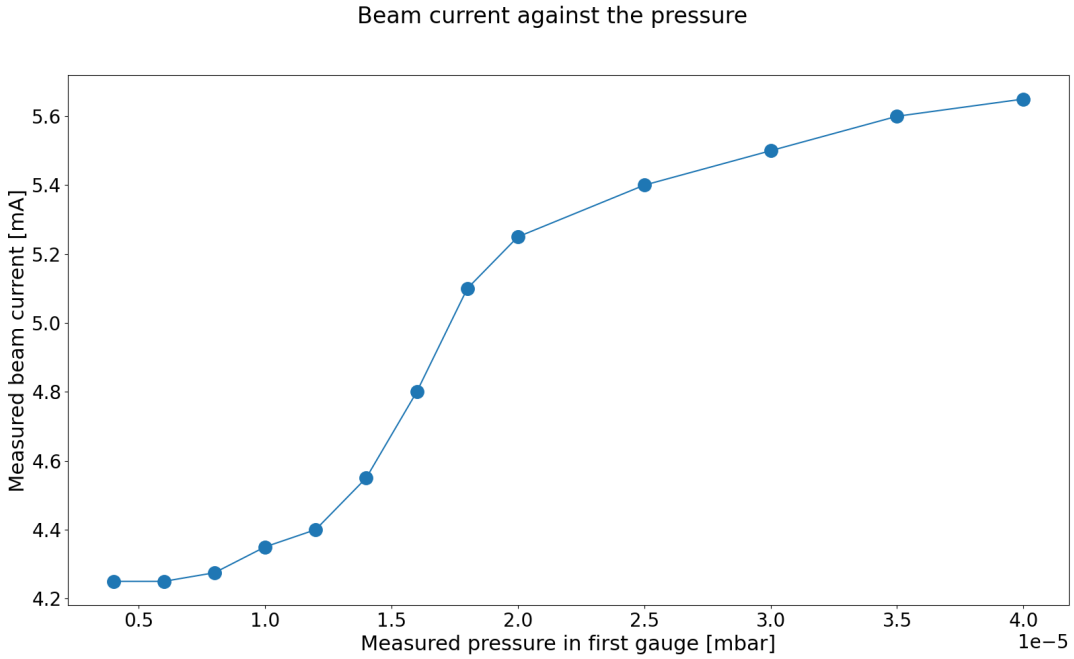

Figure 3.12: Study of the measured beam current at the LEBT exit against the injection of Argon gas. The line is a guide for the eye. The measurement error is within the symbol size.

Here, the gas injection has been studied for a beam current of 8 mA at the ion source exit and with a LEBT configuration of $I_1^{sol} = 65.6$ A, $I_2^{sol} = 77.9$ A, $I_{2H}^{st} = 0.75$ A,

$I^{st}_{2V} = -0.5$ A and $r_{col} = 60$ mm. The current measured in the first Faraday cup was $I^{b}_{FC1} = 7.4$ mA. The pressure measured in the first gauge is used as the indicator of the Ar injection.

Figure 3.12 shows the evolution of the beam current measured in the Faraday cup at the exit of the LEBT. It shows that the beam current measured in the Faraday increases as the space charge compensation increases due to the injection of Ar gas. Most of this increases is established when the measured pressure reaches $p_{meas} = 2 \times 10^{-5}$ mbar. Hence, maintaining this pressure was suitable during the commissioning as supported by the space charge compensation study performed for the commissioning at LPSC [100].

LEBT characterization and data gathering

After the determination of a nominal configuration for the LEBT, multiple solenoids and steerers scans have been performed. This was done to characterize the behavior of the LEBT over a wide range of configurations to constitute an experimental data set that represents the LEBT. This data set was required to train a neural network based model to reproduce the behavior of the LEBT. Overall, two families of scans have been performed:

- Scans of the solenoids and the collimator with optimized steerers

  With the steerers configuration optimized in the previous step, both solenoids and the collimator are scanned to find a nominal configuration (with a measured beam current of about 4.2 mA at the exit of the LEBT) and to gather data about the behavior of the LEBT. For a given collimator opening, both solenoids were scanned from 50 A up to 110 A with a step size of 2 A. This means that there were 31 steps per solenoid for a total of 961 measured configurations per scan. The tested collimator extensions were: 1.25, 2.5, 3.75, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 30, 35, 40, 45, 50 and 55 mm. In total, 19 solenoids scans were performed corresponding to 18 259 measured configurations.

setpoint	Unit	Lower bound	Upper bound	Step size
I^{sol}_1	A	50	110	2
I^{sol}_2	A	50	110	2
I^{st}_{2H}	A	0.75		-
I^{st}_{2V}	A	-0.5		-
r_{col}	mm	1.25, 2.5, 3.75, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 30, 35, 40, 45, 50, 55		-
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)		-

Table 3.6: Parameters of the solenoids scans for the LEBT characterization.

- Scans of the steerers and the collimator with the nominal solenoids

This final step was performed to gather data about the effect of the steerers installed in the second solenoid with various collimator openings. Although not strictly necessary for the commissioning, the scans performed during this step are of interest to constitute the dataset required for neural networks training. In practice, the horizontal steerer installed in the second solenoid was scanned from -0.5 A up to 2.5 A with a step size of 0.25 A while the vertical steerer was scanned from -2 A up to 1 A with a step size of 0.25 A. The choice for the boundaries was made to have the maximum transmission close to the center of the ranges. This means that there were 13 steps per steerer for a total of 169 measured configurations per scan. The tested collimator extensions were: 5, 10, 15, 20 and 25 mm. In total, 6 steerers scans were performed corresponding to 1 014 measured configurations.

setpoint	Unit	Lower bound	Upper bound	Step size
I_1^{sol}	A	65.6		-
I_2^{sol}	A	77.9		-
I_{2H}^{st}	A	-0.5	2.5	0.25
I_{2V}^{st}	A	-2	1	0.25
r_{col}	mm	5, 10, 15, 20, 25		-
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)		-

Table 3.7: Parameters of the steerers scan.

3.1.4 PSO test

As seen in Section 3.1.3, the manual determination of a nominal configuration of a LEBT is a lengthy process. It requires performing many scans over the various configuration settings of the solenoids, the steerers, the collimator opening and the gas injection. Overall, the time invested in this process is a time where the machine is not available. Hence, to increase the availability of a machine, it would be useful to have a faster method to tune it. As introduced in Section 2.2, the use of PSO is potentially an alternative to manual tuning that can optimize many settings against multiple objectives in an efficient way.

Hence, during the recommissioning, a few hours were dedicated to test a PSO algorithm on the LEBT. The algorithm used a Python implementation and an EPICS[1]-Python interface to directly control the LEBT settings that are left free to optimize. The algorithm was constrained so that it would not try to test settings outside of the boundaries tested during the recommissioning (cf. Table 3.8).

For the PSO test, the cost functions to optimize were:

$$O_1(I^b_{LEBT,out}) = (I^b_{LEBT,out} - I^{b*}_{LEBT,out})^2 = (I^b_{LEBT,out} - 4.2)^2, \tag{3.1}$$

[1]Experimental Physics and Industrial Control System

and

$$O_2(I_1^{sol}, I_2^{sol}, I_{2H}^{st}, I_{2V}^{st}, r_{col}) = (I_1^{sol} - 65.6)^2 + (I_2^{sol} - 77.9)^2 + (I_{2H}^{st})^2 + (I_{2V}^{st})^2 - x_{col} \quad (3.2)$$

where $x_{col} = 55 - r_{col}$. In Equation 3.2, x_{col} is used instead of r_{col} because the EPICS setpoint for the collimator corresponds to the collimator extension into the chamber i.e. when the setpoint is equal to 0 mm the collimator is fully opened and when it is equal to 55 mm the beam is completely intercepted.

The first objectives allows to define a target current in the second Faraday cup $I_{FC_2}^b = I_{LEBT,out}^b$. The second one favors configurations closer to the design configuration ($I_1^{sol} = 65.6$ A, $I_2^{sol} = 77.9$ A, $I_{2H}^{st} = 0$ A and $I_{2V}^{st} = 0$ A) and smaller collimator opening as to intercept a maximum of the halo. The PSO algorithm is configured to minimize both cost functions simultaneously using an ϵ-dominant approach. The result of this approach is a list of the best trade-offs achieved on the machine between the two cost functions (cf. Figure 3.13).

Setpoint	Unit	Lower bound	Upper bound
I_1^{sol}	A	50	110
I_2^{sol}	A	50	110
I_{2H}^{st}	A	-0.5	2.5
I_{2V}^{st}	A	-2	1
x_{col}	mm	0 (fully open)	55
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)	

Table 3.8: Parameters of the of the online PSO optimization on the MYRRHA LEBT.

The configuration closest to 4.2 mA (candidate solution with O_1 closest to 0) suggested by the algorithm is $I_1^{sol} = 66.1$ A, $I_2^{sol} = 78.1$ A, $I_{2V}^{st} = 0.06$ A, $I_{2H}^{st} = -0.56$ A and $r_{col} = 14.85$ mm. To get this result, the algorithm ran for about 1.5 hours. This relatively long execution time is caused by the reaction time of the steerers power supplies (around 10 s to apply a change in setpoint). Nevertheless, this optimization approach showed promising results. It should be tested in more detail to optimize the beam transmission through the RFQ.

3.1.5 Allison scanners: emittance measurements and measurement error

The Allison scanners were used in the middle of the LEBT to measure the beam characteristics after the first solenoid in both planes (cf. Figure 3.16). Two sets of measurements have been performed for different settings of the first solenoid and different collimator openings. The first set was measured to find a waist of the beam (cf. Table 3.9) The Twiss parameters estimated on this second set of measurements are shown in Figure 3.14. From these, it appears that a waist is obtained for $I_1^{sol} = 77.5$ A. The emittance measurement with this setpoint is shown in Figure 3.15.

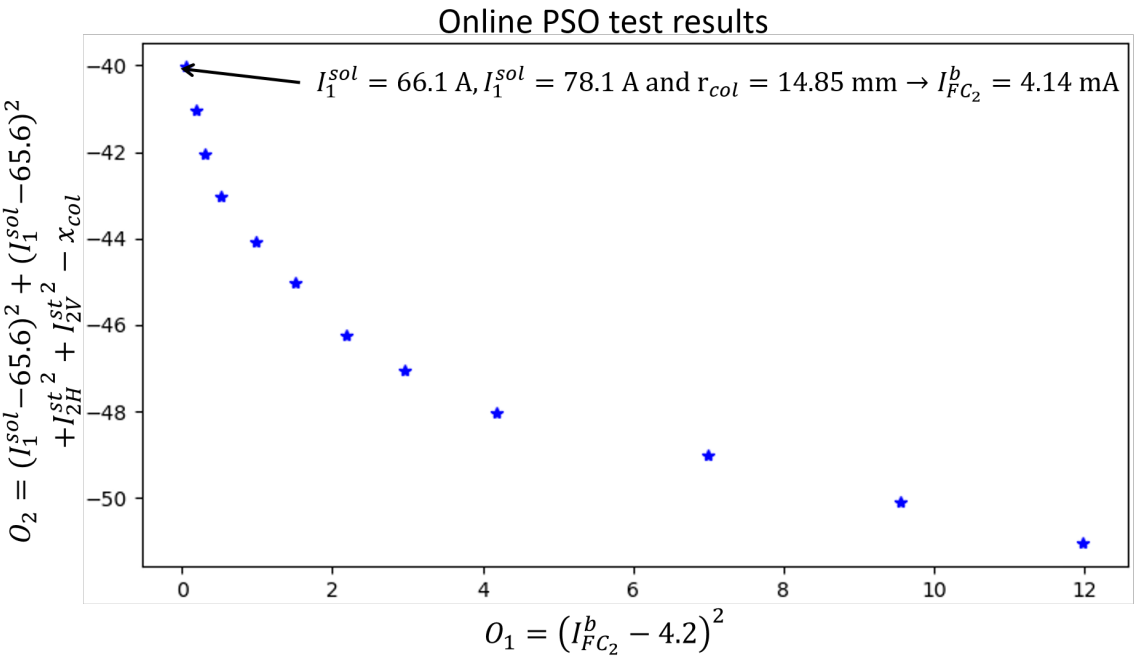

Figure 3.13: Pareto front after the PSO execution on the LEBT. Each star represents a configuration of the LEBT.

Figure 3.14: Evolution of α_y and β_y in the middle of the LEBT with respect to the first solenoid setpoint I_1^{sol}.

The second set was measured to gather data to try to get the simulated model on TraceWin to be as close as possible to the real machine (cf. Table 3.10 and Figure 3.17). This calibration is described in Section 3.3.1.

Setpoint	Unit	Value
I_1^{sol}	A	0, 10, 20, 30, 40, 50, 60, 70, 72.5, 75, 77.5, 80, 82.5, 85 90, 100, 110, 120
r_{col}	mm	55 (fully open)
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)

Table 3.9: Parameters of the second set of emittance measurements.

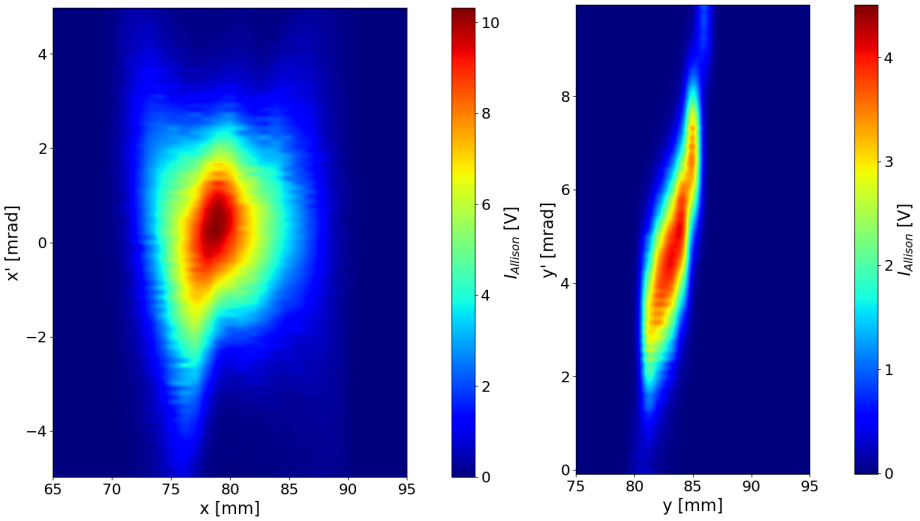

Figure 3.15: Allison scanners measurements with $I_1^{sol} = 77.5$ A in the middle of the LEBT. Left: horizontal emittance. Right: vertical emittance.

Setpoint	Unit	Value
I_1^{sol}	A	60, 63, 65.6, 69, 72
r_{col}	mm	55 (fully open)
Ar injection	-	Yes ($p_{meas} = 2 \times 10^{-5}$ mbar)

Table 3.10: Parameters of the first set of emittance measurements.

As this set is used to calibrate the TraceWin model, it is interesting to look into the measurement error. The measurement error of such an apparatus is typically considered to be between 10 % and 20 % [104]. However, the estimation of the error is difficult as it usually depends on the beam characteristics. Thereafter, a non-exhaustive analysis of the source for errors is made to infer its influence on the estimation of the beam emittance.

Estimation of the measurement error

The following development is based on the work of R. Duperrier [106].

The principle of an Allison scanner (cf. Figure 3.5) is:

1. to filter part of the beam using a fixed slit A to select a position,

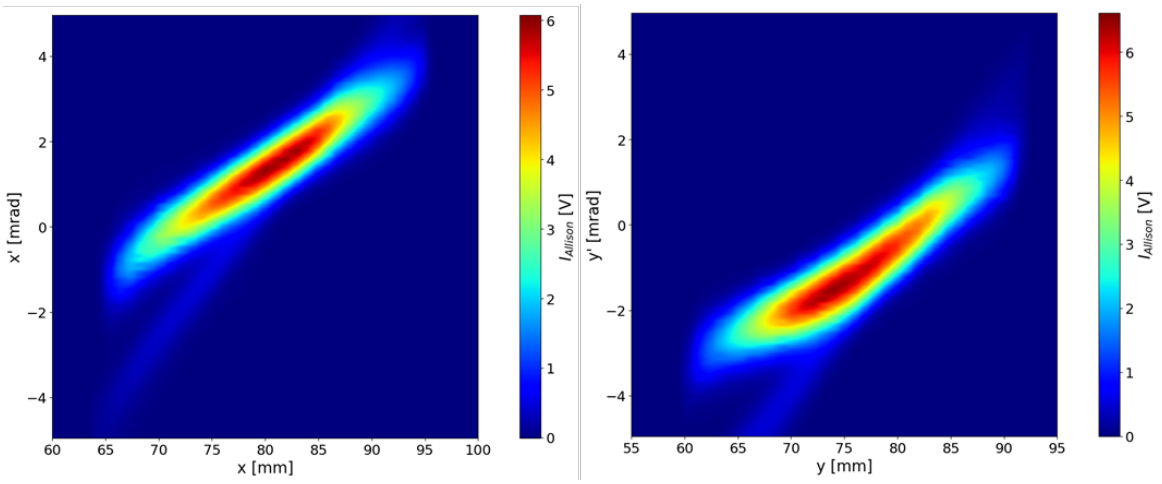

Figure 3.16: Allison scanners measurements in both planes (horizontal on the left and vertical on the right) with $I_2^{sol} = 65.6$ A.

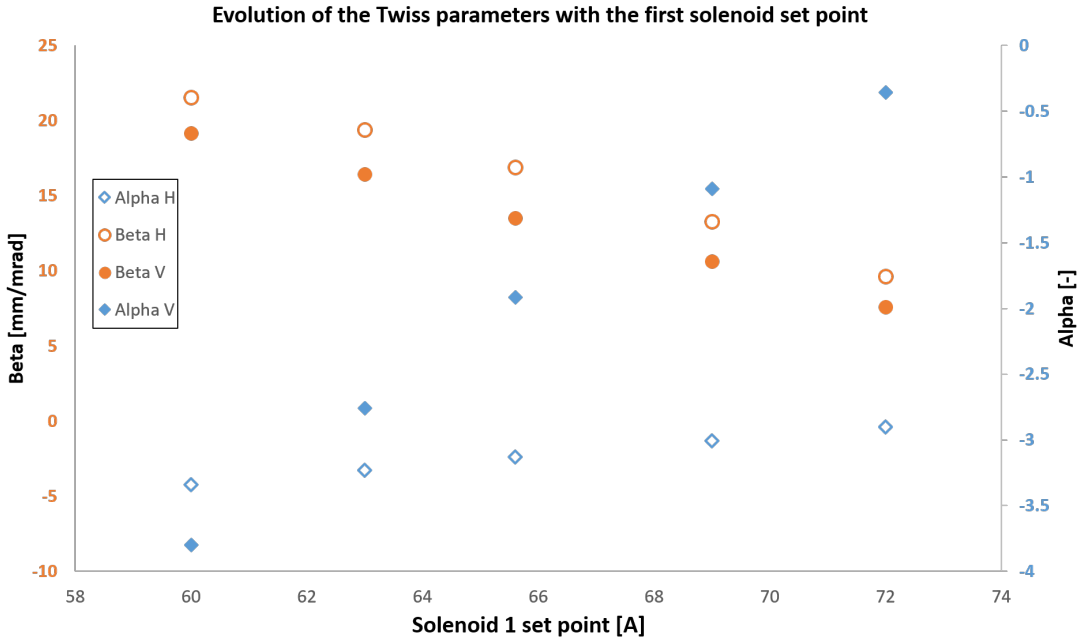

Figure 3.17: Evolution of α_u and β_u in the middle of the LEBT with respect to the first solenoid setpoint I_1^{sol}.

2. to bend the selected part of the beam using an electric field between two electrodes,
3. to filter again the selected particles using a second fixed slit B to select an angle and
4. to measure the current of the remaining particles using a Faraday cup.

The selected angle is scanned by varying the voltage between the electrodes (the

intensity of the electric field and hence the bending of the particle trajectories). Once the voltage ramp is complete for a position, the Allison scanner is moved by a few tenths of a millimeter and the voltage ramp is performed again. This process is repeated over the entire beam passage area.

The system shown in Figure 3.5 can be broken down into three areas: a drift space of length L_1, an electrostatic deflector with a gap g, a voltage ΔV and a length L_2 and, finally, a second drift space of length L_3. To estimate the measurement error in an Allison scanner, let us derive the dynamics of a particle in this system.

Let us consider a particle of mass m and charge q that enters the first slit of an Allison scanner with the coordinate x_0 and transverse speed $\dot{x}_0$. The dynamics of the particle can expressed with the following three relationships:

$$\begin{cases} \ddot{x} = a \\ \dot{x} = \int_0^t \ddot{x}dt = at + \dot{x}_0 \\ x = \int_0^t \dot{x}dt = \frac{at^2}{2} + \dot{x}_0 t + x_0 \end{cases} \tag{3.3}$$

Let us introduce $x^{'}$, the angle of the particle given by:

$$x^{'} = \frac{\dot{x}}{\dot{z}}. \tag{3.4}$$

- Drift space

  The particle is not subject to any forces ($a = 0$) so the system becomes:

$$\begin{cases} \ddot{x} = 0 \\ x^{'} = x_0^{'} \\ x = x_0^{'}\Delta z + x_0. \end{cases} \tag{3.5}$$

- Electrostatic deflector

  The particle path is influenced by the electric field E. This effect can be approximated by a kick ($a = \frac{F}{m} = \frac{qE}{m}$) and the system becomes:

$$\Delta x^{'} = \frac{\Delta \dot{x}}{\dot{z}} = \int_0^t \frac{\ddot{x}}{\dot{z}}dt = \int_0^t \frac{qE}{\beta cm}dt. \tag{3.6}$$

  This integral can be solved using a change of variable:

$$t = \frac{\Delta z}{\dot{z}} \Rightarrow dt = \frac{dz}{\dot{z}}. \tag{3.7}$$

  So that the integral becomes:

$$\Delta x^{'} = \int_{z_0}^{z} \frac{qE}{\beta^2 c^2 m}dz = -q\frac{\Delta V L_z}{\beta^2 c^2 mq} \tag{3.8}$$

where $\Delta z = L_z$ is the length of the electrodes and the electric field has been substituted by the voltage ΔV over the gap between the electrodes g with the relationship $E = -\frac{\Delta V}{g}$.

By introducing the kinetic energy of the particle U and knowing that it is equal to the ion source extraction voltage ($U = \frac{mv^2}{2q} = \frac{\beta^2 c^2 m}{2q}$ in the non relativistic approximation), Equation 3.8 becomes:

$$\Delta x^{'} = -\frac{\Delta V L_z}{2gU}. \tag{3.9}$$

And thus, with the kick approximation, the following deflection is applied in the middle of the electrodes:

$$x^{'} = x_0^{'} + \Delta x^{'}. \tag{3.10}$$

The dynamics of the particle in the Allison scanner can then be calculated by expressed as a succession of drifts and a kick (cf. Figure 3.5). For a particle with initial position x_0 and angle $x_0^{'}$:

1. the particle travels through a drift space of length $L_1 + \frac{L_2}{2}$

$$\begin{cases} x_1 = x_0 + x_0^{'}(L_1 + \frac{L_2}{2}) \\ x_1^{'} = x_0^{'} \end{cases}, \tag{3.11}$$

2. in the middle of the electrodes, the particle is subjected to a kick

$$\begin{cases} x_2 = x_1 \\ x_2^{'} = x_1^{'} + \Delta x^{'} \end{cases} \tag{3.12}$$

3. then the particle travels through a drift space of length $\frac{L_2}{2} + L_3$

$$\begin{cases} x_3 = x_2 + x_2^{'}(\frac{L_2}{2} + L_3) \\ x_3^{'} = x_2^{'} \end{cases}. \tag{3.13}$$

By substituting Equation 3.11 in Equation 3.12 then Equation 3.12 in Equation 3.13:

$$\begin{cases} x_3 = x_0 + x_0^{'}(L_1 + L_2 + L_3) + \Delta x^{'}(\frac{L_2}{2} + L_3) \\ x_3^{'} = x_0^{'} + \Delta x^{'} \end{cases} \tag{3.14}$$

where $\Delta x^{'} = -\frac{\Delta V L_2}{2Ug}$ (Equation 3.9).

In an ideal case, for infinitely thin slits, a particle would be transmitted through the Allison scanner and would contribute to the current measured in the Faraday cup if $x_3 = x_0$ hence the first relationship in Equation 3.14 becomes:

$$0 = x_0^{'}(L_1 + L_2 + L_3) + \Delta x^{'}(\frac{L_2}{2} + L_3) \Rightarrow x_0^{'} = -\frac{\Delta V L_2}{2Ug}\left(\frac{\frac{L_2}{2} + L_3}{L_1 + L_2 + L_3}\right). \tag{3.15}$$

This means that only particles with the initial angle given in the latter relationship are transmitted. Let us call this angle $x'_{0,ideal}$.

Now, the measurement error can be separated into multiple sources:

- Slits opening

  For slits with a width δ, a particle is transmitted if it enters at A with $x \in [-\frac{\delta}{2}, \frac{\delta}{2}]$ and exits at B with $x \in [-\frac{\delta}{2}, \frac{\delta}{2}]$. In the worst case, $x_0 = \pm\frac{\delta}{2}$ at A and $x_0 = \mp\frac{\delta}{2}$, the first relationship of Equation 3.14 becomes:

  $$\mp \frac{\delta}{2} = \pm\frac{\delta}{2} + x'_0(L_1 + L_2 + L_3) + \Delta x'(\frac{L_2}{2} + L_3) \tag{3.16}$$

  and thus the transmitted angle becomes:

  $$x'_0 = x'_{0,ideal} \pm \frac{\delta}{L_1 + L_2 + L_3}. \tag{3.17}$$

  A numerical application with $\delta = 0.1$ mm, $L_1 = L_3 = 4.5$ mm et $L_2 = 60$ mm gives an error of ± 1.4 mrad.

- Shifted slits

  For misaligned slits with a shift of α, a particle is transmitted if $x_3 = x_0 \pm \alpha$. Similarly to before, the first relationship of Equation 3.14 then gives:

  $$x'_0 = x'_{0,ideal} \pm \frac{\alpha}{L_1 + L_2 + L_3}. \tag{3.18}$$

  If the uncertainty on the relative alignment of the slits is about 0.1 mm, the error is once again of ± 1.4 mrad. This is a systematical error and its effect is an apparent shift in the angle of the beam. Hence, it does not affect the estimation of the beam emittance nor the position shift.

- Tilted slits

  For misaligned slits with a tilt of η, a particle is transmitted if $x_3 = x_0 + y_0 \sin(\eta)$ which means that there is a coupling between planes. Similarly to before, the first relationship of Equation 3.14 then gives:

  $$x'_0 = x'_{0,ideal} \pm \frac{y_0 \sin(\eta)}{L_1 + L_2 + L_3}. \tag{3.19}$$

  The value of y_0 can not exceed half of the length of the slits in the scanner referential. So with a tilt of about 5.5 mrad, a slit length $L_{slits} = 80$ mm and its uncertainty $\delta_{L_{slits}} = 0.1$ mm, the error is about 3.2 mrad. Note that this is a pessimistic estimation as it supposes that the whole slit length is useful for the measurements i.e. that the size of the beam is around the slit length. If the half-width of the beam (≈ 15 mm) is used instead, the error is about 1.1 mrad.

- Tilted Allison scanner with respect to the reference plane

  For a misaligned Allison scanner with a tilt of θ, the beam is rotated with respect to the scanner. For a beam with a transverse geometrical emittance $\epsilon = \epsilon_x = \epsilon_y$ represented by the matrix σ:

$$\sigma = \frac{\epsilon}{\pi} \begin{pmatrix} \beta_x & -\alpha_x & 0 & 0 \\ -\alpha_x & \gamma_x & 0 & 0 \\ 0 & 0 & \beta_y & -\alpha_y \\ 0 & 0 & -\alpha_y & \gamma_y \end{pmatrix}. \tag{3.20}$$

  By applying the rotation matrix to this beam matrix, the apparent emittance $\epsilon_{x,meas}$ measured by the scanner is equal to:

$$\epsilon_{x,meas} = \epsilon[\cos^4(\theta) + \sin^4(\theta) + \sin^2(\theta)\cos^2(\theta)(\beta_x\gamma_x + \beta_y\gamma_y - 2\alpha_x\alpha_y)]. \tag{3.21}$$

  This means that the error is a function of the beam that is measured. With the first solenoid in its nominal configuration ($I_1^{sol} = 65.6$ A) and at the position of the emittance meter, the beam dependant factor is equal to about 2.05. By substituting this value in Equation 3.21 with a tilt angle of about 5.5 mrad, the emittance is overestimated by a factor of about 1.5×10^{-6}.

With the evaluation of these sources of error, the overall resolution of the Allison scanners can be evaluated:

- Position and angle errors

  The position uncertainty is essentially determined by the opening δ of slit A and the precision of the driving motor of the scanner $\delta_m \approx 50$ μm (precision of the motor step when the scanner is moved). Hence:

$$\Delta x = \delta + \delta_m \approx 0.15 \text{ mm}. \tag{3.22}$$

  For the angle, the derivative of Equation 3.15 is used to estimate the absolute uncertainty with a Taylor development:

$$\Delta x_0^{'} = \sum \left| \frac{\partial x_0^{'}}{\partial u_i} \right| \Delta u_i \tag{3.23}$$

  with u_i a variable of Equation 3.15. By using Equation 3.17, the uncertainty becomes:

$$\Delta x_0^{'} = \frac{\delta}{L_1 + L_2 + L_3} \approx 1.4 \text{ mrad}. \tag{3.24}$$

- Error for each moment of the distribution

  The estimation of the emittance use the evaluation of the second order moments of the beam in phase space:

$$\sigma_x^2 = \langle (x - \langle x \rangle)^2 \rangle, \tag{3.25}$$

$$\sigma_{x'}^2 = \langle (x' - \langle x' \rangle)^2 \rangle \tag{3.26}$$

  and

$$\sigma_{xx'}^2 = \langle (x' - \langle x' \rangle)(x - \langle x \rangle) \rangle. \tag{3.27}$$

  By taking into account the uncertainty for each parameters, Δx and $\Delta x'$, the uncertainty on the position moment is:

$$\Delta\sigma_x^2 = \pm 4 \Delta x^2 \tag{3.28}$$

  and on the angle moment is:

$$\Delta\sigma_{x'}^2 = \pm 4 \Delta x'^2. \tag{3.29}$$

  For the last term, the uncertainty is:

$$\Delta\sigma_{xx'}^2 = \pm 4 \Delta x \Delta x'. \tag{3.30}$$

- Evaluation of the error on the emittance

  With the second order moments, the emittance is given by:

$$\epsilon_x = \left[\langle (x - \langle x \rangle)^2 \rangle \langle (x' - \langle x' \rangle)^2 \rangle - \langle (x' - \langle x' \rangle)(x - \langle x \rangle) \rangle^2 \right]^{\frac{1}{2}}. \tag{3.31}$$

  Using a Taylor development and the estimated error for each moment, the uncertainty on the emittance is given by:

$$\Delta\epsilon_x = 2 \left[\beta_x \Delta x'^2 + \gamma_x \Delta x^2 + \Delta x \Delta x' \right] \approx 2 \left[2.1\beta + 0.225\gamma + 0.22 \right]. \tag{3.32}$$

  In the middle of the LEBT with the first solenoid in its nominal configuration $I_1^{sol} = 65.6$ A, the estimated value is about 15 mm/mrad for β and about 0.37 mrad/mm for γ (cf. Table 3.12). Hence, the uncertainty on the emittance is about ± 63.6 mm.mrad.

The development to obtain Equation 3.32 shows that the measurement error on the emittance depends on the geometry of the Allison scanners (slit opening, tilt, ...) but mostly on the beam parameters that are measured. In cases such as when $I_1^{sol} = 65.6$ A, the measurement error on the emittance may exceed 100 % because the divergence measured at a given position of the scanner is of the order of the measurement error on the angle (cf. Equation 3.24).

For the calibration of the TraceWin model, using the measured emittance directly did not prove to be conclusive (cf. Section 3.3.1.1). However, according to the Equations 3.28 and 3.29, the measurement errors on the size $\Delta\sigma_x = 0.3$ mm and the angle spread $\sigma_{x'} = 2.8$ mrad of the beam are much lower. Hence, the input beam parameters were optimized to match the size and spread of the beam at the position of the Allison scanners (cf. Section 3.3.1.1).

3.2 IPHI

The end goal of a LEBT is to ensure a reliable transport of a DC proton beam from the source to the RFQ and to condition the beam to ensure its proper acceleration and minimize beam losses in the following accelerating devices. In the case of MYRRHA, the RFQ was not yet installed when the experiments were performed. As such, IPHI, a second machine with an already operational RFQ, has been studied.

IPHI (Injecteur de Protons à Haute Intensité [107]) is a prototype of a low energy section for next-generation accelerators with high proton beam current: 100 mA accelerated to 3 MeV. IPHI consists of a proton source and its LEBT at 95 keV then a RFQ able to accelerate the beam up to 3 MeV followed by a diagnostic beamline to characterize the beam at the RFQ exit. IPHI is developed in a collaboration between CEA, CNRS and CERN and is installed at the CEA Saclay site.

The high beam intensity and the RFQ make IPHI a prime candidate to extend the exploration of the abilities of the neural network to model the complex physics in proton accelerators. Hence, a week of data gathering has been performed on-site to train a neural network to model the transmission and beam current at the exit of the LEBT. During this week, a record has been established of the highest power achieved in an injector.

This section is dedicated to the description of IPHI and the gathering of experimental data. The section is organized as follows.

First, the technical description of IPHI is given in Section 3.2.1.

Then, the objective of the experiments is discussed in Section 3.2.2.

In Section 3.2.3, the methodology followed to characterize IPHI is described and illustrated with an example of a solenoids scan.

3.2.1 Technical description

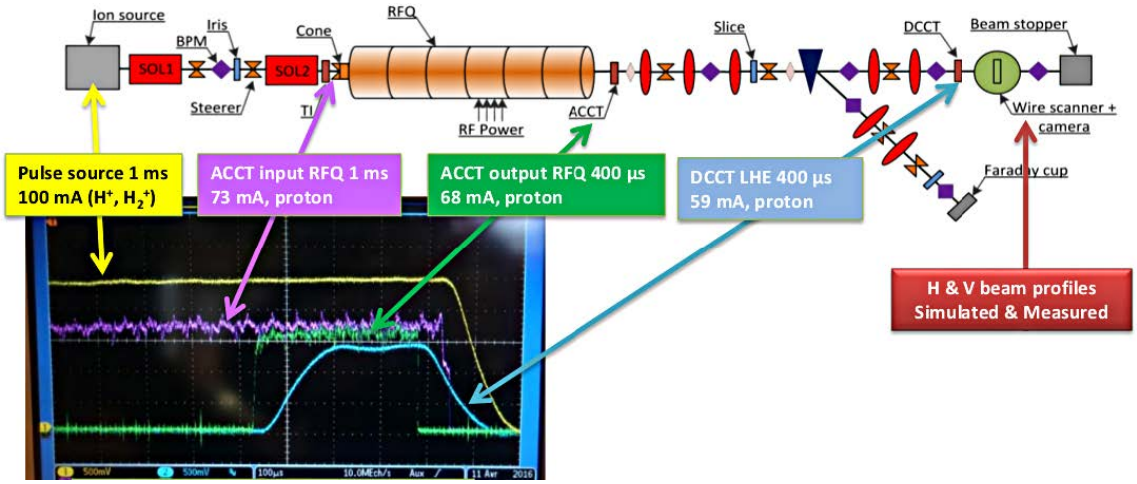

Figure 3.18: Diagram of IPHI with the first beam accelerated in April 2016. [24]

As shown in Figure 3.18, the first part of IPHI is equipped with the following functional elements.

- An ECR ion source
  SILHI (Source d'Ions Léger à Haute Intensité, cf. Figure 3.19) is a 2.45 GHz Electron Cyclotron Resonance ion source designed to produce an intense beam of 100 mA of H^+. It uses a 5-electrode extraction system at 95 keV. SILHI can be operated in pulsed and DC mode.

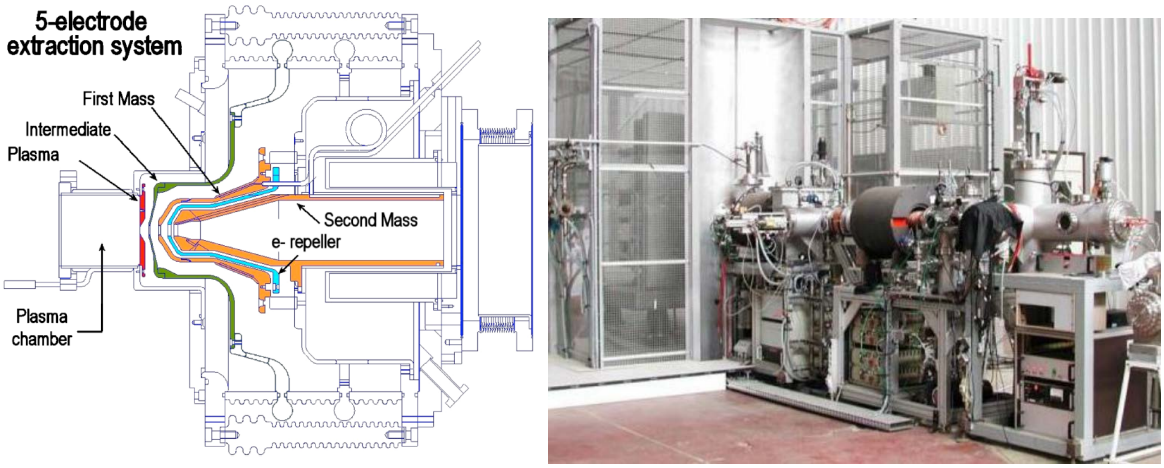

Figure 3.19: Left: diagram of the IPHI ion source: SILHI. [24] Right: picture of the IPHI LEBT with the cage around SILHI in the back. [51]

- Two solenoids
  They are used to focus and guide the beam towards the RFQ.
- A fixed collimator
  It is installed just after the first solenoid to intercept part of the beam halo.
- Two pairs of steerers
  The first pair is installed just after the first solenoid and the second pair is installed just before the second solenoid.
- A collimation system (iris)
  The collimation system consists of a metallic iris. Its role is to intercept part of the beam to clean it from the particles with unwanted properties (diverging too much or too far from the beam center) and control the beam current injected into the RFQ. The opening of the iris can be adjusted to control how much of the beam is intercepted.
- A collimation cone
  The collimation cone is the last protection system before the beam injection into the RFQ. Particles in the beam that have unwanted properties impact the collimation cone instead of entering the RFQ.
- A RFQ
  The RFQ [108] (see Figure 3.20) is a 6 meters long accelerating cavity designed to accept a CW beam of 100 mA of H^+ at 95 keV. The proton beam is accelerated to 3 MeV and bunched at 352 MHz in the RFQ.

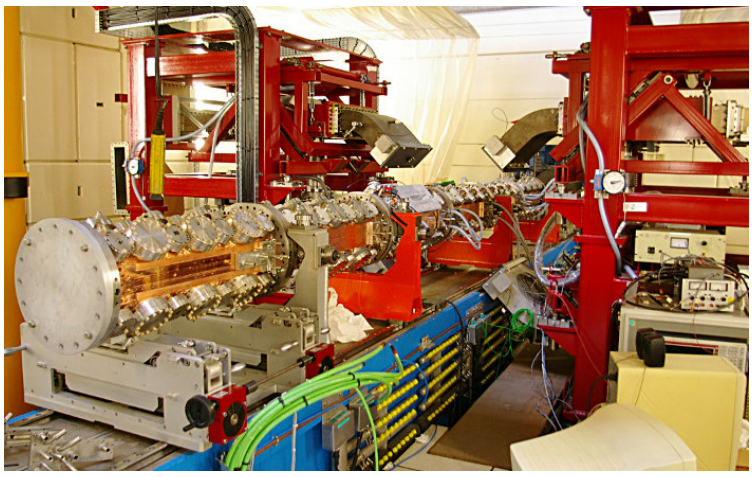

Figure 3.20: Picture of the IPHI RFQ. [24]

In addition, the first part of IPHI is equipped with diagnostic tools to measure beam properties.

- A Charge-Coupled Device (CCD) camera
  The camera is used to see the beam at the source output.

- A Faraday cup
  The Faraday cup is used to measure the beam current after the first solenoid. It is also used to intercept the beam to prevent it from going further into the LEBT.

- Two ACCTs
  The active ac-transformers are used to measure the beam current at the entrance and the exit of the RFQ. This allows for determining the transmission of the beam through the RFQ.

The RFQ is followed by the Medium Energy Beam Transport (MEBT) line. In the first section after the RFQ, three quadrupoles are used to control the beam optics. Then the beam is either allowed to go into the straight section consisting of two quadrupoles, various beam diagnostics and a 300 kW beam dump. Or it is deflected into a second line that consists of two quadrupoles and a low power beam stopper (several kW).

3.2.2 Objectives

The main objective of the experimental campaign on IPHI was to gather data to build a data set that represents the beam dynamics of the machine. This data set would then be used to train a neural network to model the behavior of IPHI. The interest in working on IPHI stems from the fact that its LEBT design is similar to the MYRRHA LEBT design, that its RFQ was already coupled to the LEBT and that the proton beam current is high. Hence, IPHI could be seen as both a confirmation that neural networks based models of proton injectors are realistic and an advance on the projected development of the MYRRHA injector.

As for MYRRHA, the characterization of IPHI consists of scanning the configuration space of its LEBT. However, here the beam properties of interest are the beam current at the RFQ exit and the transmission through the RFQ. Indeed, these quantities are the figures of merit for a particle injector equipped with a RFQ.

3.2.3 Methodology

The characterization of IPHI was done by scanning the configuration space of its LEBT. Two types of scans were planned to be performed with different collimator openings: solenoids scans and steerers scans. For both types of scans, the beam current was measured at the RFQ entrance and exit.

Solenoids scan

Both solenoids were scanned by changing regularly the current applied to them. The first solenoid was scanned from 50 A up to 120 A with a step size of 2 A. The second solenoid was scanned from 145 A up to 185 A with a step size of 2 A. Both vertical steerers setpoints were set to 0.3 A while the first horizontal steerer was set to 0 A and the second to -0.1 A. This configuration of the steerers is the nominal

configuration. With these settings, each solenoids scan counts 756 points. Thirteen of these scans have been performed corresponding to thirteen different iris openings from 10 mm up to 130 mm (fully open) with a step size of 10 mm. Hence a total of 9 828 configurations have been measured.

Setpoint	unit	Lower bound	Upper bound	Step size
I_1^{sol}	A	50	120	2
I_2^{sol}	A	145	185	2
I_{1H}^{st}	A	0		-
I_{1V}^{st}	A	0.3		-
I_{2H}^{st}	A	-0.1		-
I_{2V}^{st}	A	0.3		-
r_{col}	mm	10	130	10

Table 3.11: Summary of the solenoids scans performed on IPHI.

As an illustration, the solenoids scan performed with a nominal steerers configuration ($I_{1H}^{st} = 0$ A, $I_{1V}^{st} = 0.3$ A, $I_{2H}^{st} = -0.1$ A and $I_{2V}^{st} = 0.3$ A) and the iris fully opened ($r_{col} = 130$ mm) is shown on Figure 3.21 and 3.22 for the measurements of the beam current at the exit of the LEBT and the RFQ transmission respectively.

The highest beam current measured at the exit of the RFQ was about 60.6 mA with $I_1^{sol} = 102$ A and $I_2^{sol} = 163$ A with a RFQ transmission of about 93.5 %. On the other hand, the highest RFQ transmission measured was around 95 % with $I_1^{sol} = 98$ A and $I_1^{sol} = 163$ A for a beam current at the exit of the RFQ of about 60.4 mA. The 5 % beam current loss is expected to come from species other than H^+ populating the beam. The fact that the highest measured beam current and the highest RFQ transmission do not coincide highlights the fact that the beam has to enter the RFQ with fairly specific properties (incident angle and divergence).

Steerers scan

The planned steerers scan followed a similar logic to the solenoids scan. The idea was to scan regularly the second pair of steerers. The second vertical steerer was scanned from -0.6 A up to 2 A with a step size of 0.2 A while the second horizontal steerer was scanned from -1 A up to 0.8 A with a step size of 0.2 A. The first solenoid was set to 101.25 A and the second solenoid to 162.9 A. The first vertical steerer was set to 0.3 A and the first horizontal steerer to 0 A. This corresponds to the nominal configuration except for the second pair of steerers. Unfortunately, when scanning the steerers, the RFQ kept on failing due to electrical breakdowns making the steerers scans impossible to perform. Indeed, scanning the second steerers caused the beam to enter the RFQ with a wrong angle and impact the RFQ rods. This lead to the repeated failure of the RFQ.

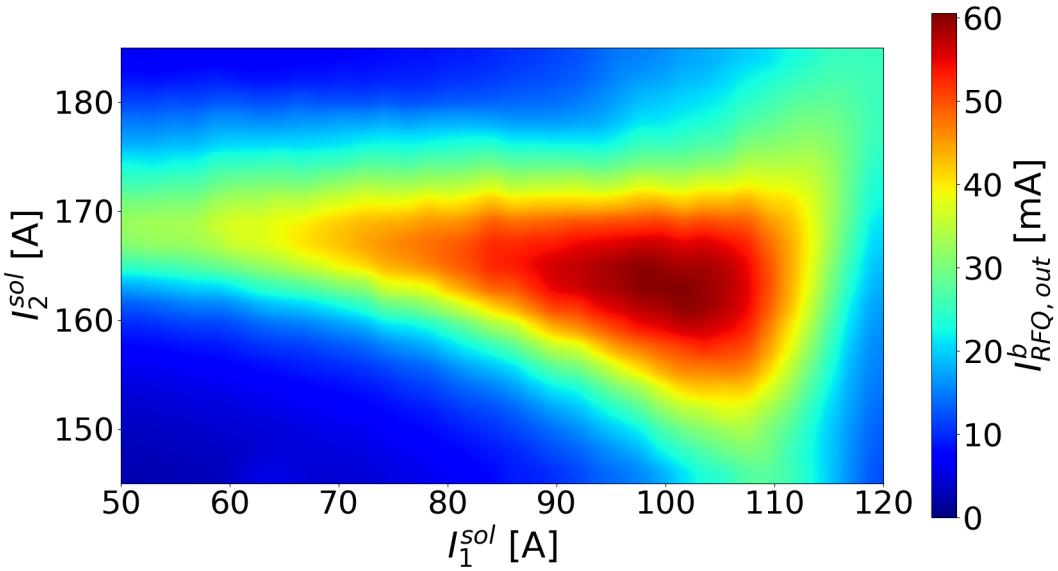

Figure 3.21: Solenoids scan with $r_{col} = 130$ mm. Beam current at the RFQ exit.

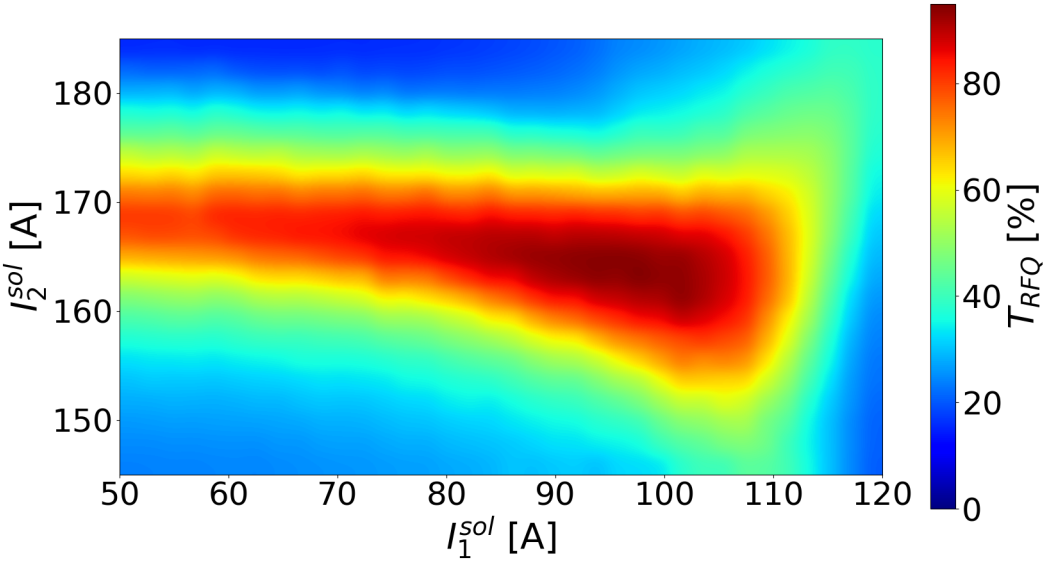

Figure 3.22: Solenoids scan with $r_{col} = 130$ mm. RFQ transmission.

3.3 TraceWin models

One of the main concerns over training a neural network to model a particle accelerator is data availability. Indeed, training a network usually requires large data sets over a wide range of examples to accurately represents the target task. However, it is expensive to operate a real particle accelerator and the range of configurations that can be tested is limited to prevent damage to the machine. Hence, the number of experimental data that can be obtained and used for training is fairly limited ("only" ten thousand and twenty thousand measurements on IPHI and MYRRHA respectively). Therefore, even though simulations do not reproduce perfectly the beam dynamics in a LEBT, they can be seen as an additional source of training data.

In this section, the TraceWin models of IPHI and MYRRHA are compared to experimental data and discussed. The section is organized as follows.

In Section 3.3.1, the model of MYRRHA is focused on and some efforts are made to improve its fidelity to the real machine.

In Section 3.3.2, the model of IPHI is compared to the data from the real machine.

3.3.1 MYRRHA model

During the recommissioning of the MYRRHA LEBT, its TraceWin model was calibrated to have simulations provide results as realistic as possible. For this, both Allison scanners were installed between the solenoids (cf. Figure 3.23) to measure the emittance in the horizontal and vertical direction at the same point. The procedure to calibrate the model was as follows.

1. Experiment: solenoids and steerers scan to find a nominal configuration of the LEBT (cf. Section 3.1.3).

2. Experiment: emittance measurements with Allison scanners and beam center measurements with wire scanner to obtain the beam characteristics to match with the simulation (cf. Section 3.1.5).

3. Simulation: using the TraceWin model, adjustment of the beam initial parameters (Twiss parameters α and β and emittance ϵ) and degree of space charge compensation to fit the measured emittance (cf. Section 3.3.1.1).

4. Simulation: using the TraceWin model, adjustment of the calibration of the steerers field maps to fit the measured beam center deflections (cf. Section 3.3.1.2).

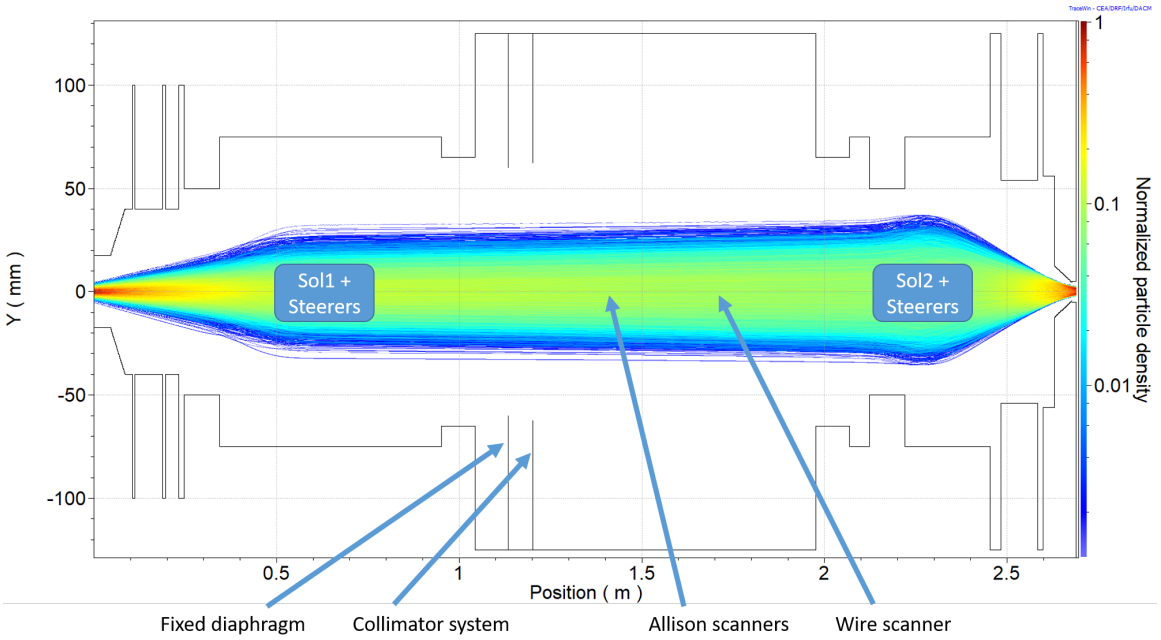

Figure 3.23: Setup of the MYRRHA LEBT for the calibration of its TraceWin model (vertical plane).

3.3.1.1 Calibration on the emittance measurements

This calibration was based on the horizontal and vertical emittance measurements for five different configurations of the first solenoid around its nominal setpoint, without steering and with Ar gas injection: $I_1^{sol} = 60, 63, 65.6, 69$ and 72 A. For each of these measurements, the Twiss parameters α and β, as well as the emittance ϵ, have been estimated after the elimination of the non H^+ trace in the measurements (cf. Table 3.12).

I_1^{sol} [A]	Orientation (H/V)	α [/]	β [mm/πmrad]	ϵ_{norm} [πmm.mrad]
60	H	-4.2	21.6	0.072
	V	-3.8	19.1	0.076
63	H	-3.3	19.4	0.066
	V	-2.8	16.4	0.072
65.6	H	-2.4	16.9	0.064
	V	-1.9	13.5	0.068
69	H	-1.3	13.3	0.06
	V	-1.1	10.6	0.064
72	H	-0.37	9.7	0.06
	V	-0.36	7.6	0.07

Table 3.12: Results of the analysis of the emittance measurement for the calibration of the TraceWin model of the MYRRHA LEBT.

Then, using the LEBT TraceWin model, five sets of beam initial parameters have

been adjusted to fit the emittance and Twiss parameters evaluated on the corresponding five solenoid configurations (cf. Table 3.13). This fit was performed using the "adjust_beam" function of TraceWin. This built-in function optimizes the input parameters of the beam to match a diagnostic down the line (the Allison scanner measurements in the present case). To be as realistic as possible, this optimization has been performed with TraceWin configured for multi-particle simulations (as opposed to simpler but less realistic beam envelope simulation).

I_1^{sol} [A]	Orientation (H/V)	α [/]	β [mm/πmrad]	ϵ_{norm} [πmm.mrad]
60	H	-9.8	1.7	0.055
	V	-7.9	1.5	0.034
63	H	-6.6	1.0	0.04
	V	-8.3	1.1	0.047
65.6	H	-10.4	1.2	0.034
	V	-8.9	1.1	0.025
69	H	-6.4	0.54	0.042
	V	-5.7	0.63	0.022
72	H	-4.7	0.31	0.023
	V	-7.4	0.54	0.026

Table 3.13: Results of the adjustment of the beam initial parameters using the TraceWin model of the MYRRHA LEBT.

Finally, each set of beam initial parameters has been compared against the five solenoid configurations to determine which provides the overall fit. The criterion used to select the best set of beam initial parameters was determined to be the Mean Squared Relative Error (MSRE) between the $\sigma_{u,sim}$ and $\sigma_{u',sim}$ of the simulation and the $\sigma_{u,meas}$ and $\sigma_{u',meas}$ estimated from the measurements with:

$$MSRE = \frac{\sum (\frac{Y_i - Y_i^*}{Y_i})^2}{N} \tag{3.33}$$

where Y is substituted with the parameters estimated on the simulation, Y^* is substituted with the parameters estimated on the measurements and N is equal to five as there are five sets of beam initial parameters. The value of σ_u and $\sigma_{u'}$ are given by:

$$\sigma_u = \sqrt{\beta_u \epsilon_u} \tag{3.34}$$

and

$$\sigma_{u'} = \sqrt{\frac{(1+\alpha_u^2)\epsilon_u}{\beta_u}}. \tag{3.35}$$

As such the set of beam initial parameters with the lowest MSRE was determined to be the set derived for the measurement with $I_1^{sol} = 69$ A (cf. Table 3.14). From then on, this set has been used for every TraceWin simulation.

I_1^{sol} [A]	Orientation (H/V)	MSRE on σ_u [mm^2]	MSRE on $\sigma_{u'}$ [mrad^2]
60	H	0.07	0.4
	V	0.04	0.3
63	H	0.004	0.3
	V	0.01	0.2
65.6	H	0.007	0.1
	V	0.001	0.1
69	H	0.003	0.1
	V	0.0005	0.09
72	H	0.0002	0.2
	V	0.002	0.2

Table 3.14: Mean Squared Relative Errors on σ and $\sigma^{'}$ for the beam initial parameters candidates.

3.3.1.2 Calibration on the beam center position measurements

This calibration was performed to determine the steerers field maps intensity factor. It used wired scanner measurements of the position of the beam center for different settings of the first pair of steerers, with I_1^{sol} = 65.6 A and with Ar gas injection. The intensity factor was tuned until the simulated beam deflection was close to the measured beam deflections, Δx and Δy (cf. Table 3.15 and Figure 3.24). Both the measured and simulated deflections show the coupling between the x and y planes caused by the superposition of the magnetic field of the solenoid.

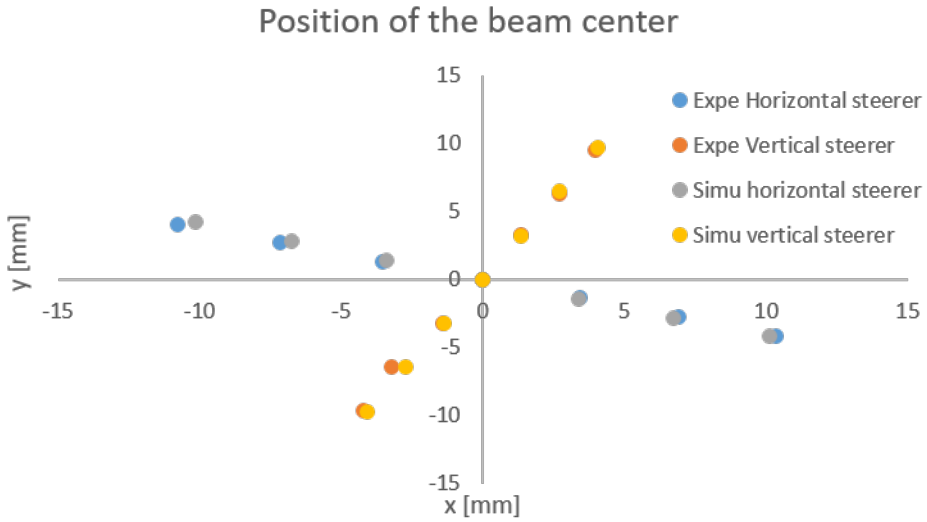

Figure 3.24: Position of the beam center as measured with the wire scanner and simulated after calibration of the steerers.

Change in setpoint	Measured deflection		Simulated deflection	
	Δx	Δy	Δx	Δy
A	mm	mm	mm	mm
$\Delta I_{1H}^{st} = 1$	3.61	-1.25	3.37	-1.41
$\Delta I_{1V}^{st} = 1$	1.37	3.2	1.36	3.23

Table 3.15: Average measured and simulated deflection of the beam center for an increment of 1 A in the first pair of steerers.

3.3.1.3 Comparison of the real and simulated LEBT

After the calibration of the initial parameters of the beam and the field maps of the steerers, the TraceWin model of the LEBT reproduces with reasonable agreement on some of the beam properties. However, when trying to reproduce the beam current measured at the exit of the LEBT during a solenoids scan, the simulation provides some of the LEBT behavior but with clear deviations (cf. Figure 3.25).

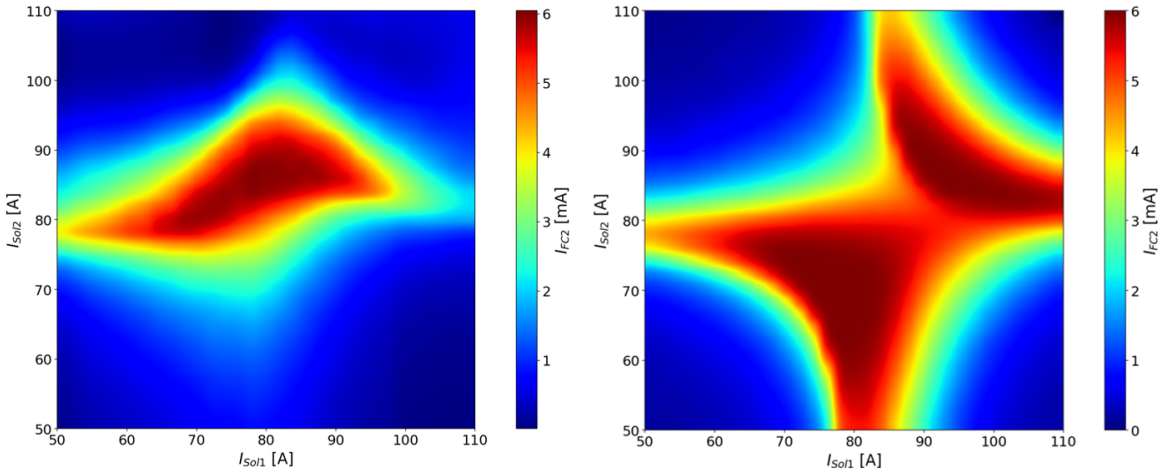

Figure 3.25: Beam current measured at the exit of the LEBT during a solenoids scan. Left: experimental scan. Right: simulated scan.

These discrepancies are expected to be caused by two main contributions: the alignment, the space charge effect and its compensation. Indeed, the alignment is "perfect" in the model but there exists unavoidable misalignment in the real machine. Even if those have been estimated to be quite reasonable (cf. Section 3.1.3), they are likely to be partly responsible for the differences between the TraceWin model simulations and the real machine.

In addition, the space charge effect and its compensation are known to be difficult to model. This many-body problem can be solved with realistic results using particle-in-cell simulation codes such as WARP (cf. Figure 1.27) [50] but such simulations are usually very long which makes it impractical. Therefore, TraceWin uses reasonable approximations to maintain a shorter computation time at the cost of realism. For

example, TraceWin assumes that the space charge compensation is uniform in the LEBT but it is not exactly right. Indeed, in TraceWin the space charge compensation is interpreted as a decrease in the apparent beam current that depends on the longitudinal position. However, the space charge compensation varies depending on both longitudinal and transverse directions due to the interactions between the beam and its environment (cf. Figure 3.26) [49].

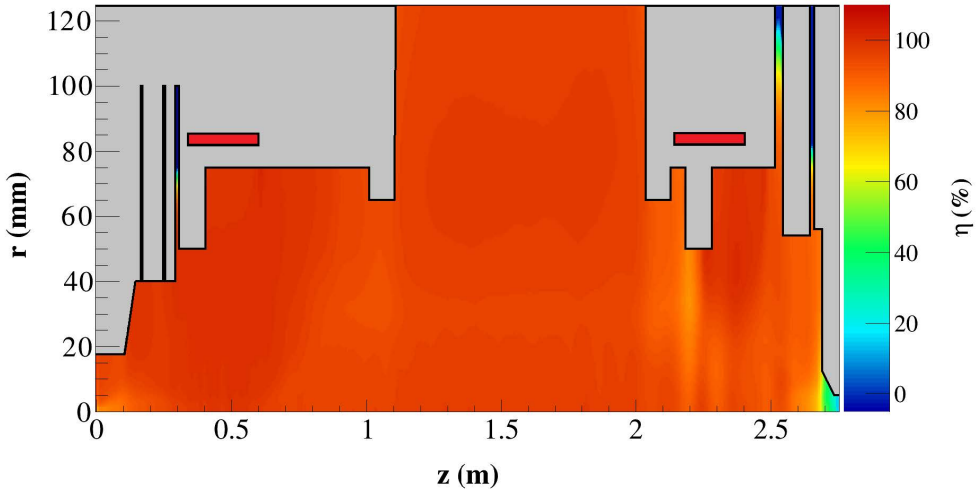

Figure 3.26: Space charge compensation map simulated with WARP. [49]

Let us also note that the calibration performed here is based on determining the beam input parameters that best match the data obtained using the Allison scanners. However, these measurements are subject to significant errors (cf. Section 3.3.1.1) so even if the simulation matches perfectly the beam characteristics coming from an experimental estimation there may be significant differences from the actual beam properties.

3.3.2 IPHI model

During the commissioning of SILHI and the IPHI LEBT in 2013, some care has been put to calibrate the TraceWin model of the LEBT [51]. The procedure to calibrate the model was as follows.

1. Experiment: optimization of the beam transmitted through the cone to find an optimal configuration of the solenoids.

2. Experiment: emittance measurement to obtain the beam characteristics to match with the simulation.

3. Simulation: using the TraceWin model, adjustment of the beam initial parameters (Twiss parameters α and β and emittance ϵ) and degree of space charge compensation to fit the measured emittance.

4. Simulation: using the adjusted TraceWin model, determination of optimal solenoid values for RFQ injection.

5. Experiment: validation of the previous result by measuring the emittance.

6. Numerical validation: SOLMAXP/WARP simulations.

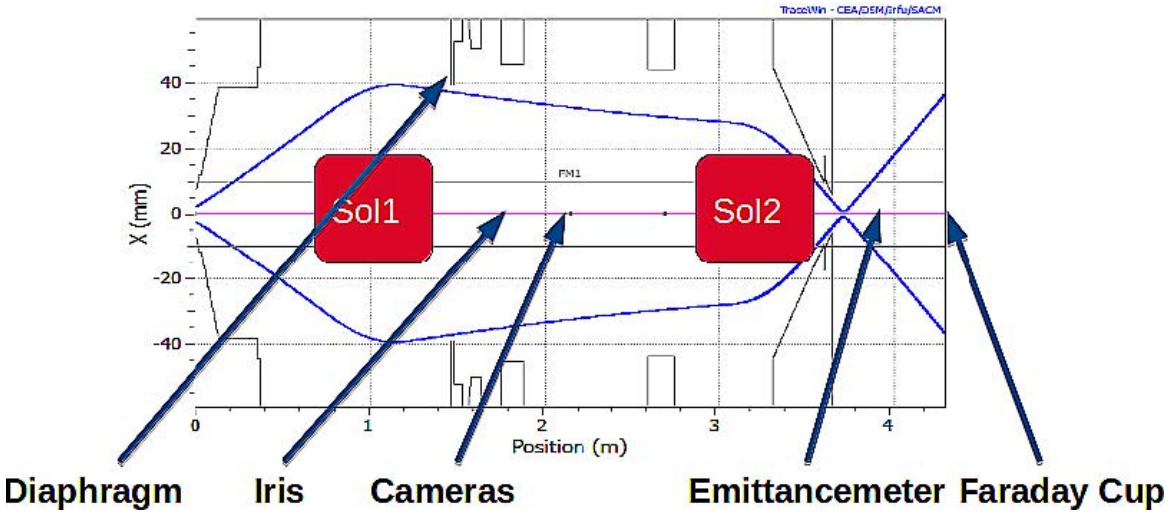

Figure 3.27: Setup of the IPHI LEBT during its commissioning phase in 2013. [51]

After this calibration process, the simulation with the adjusted TraceWin model was in reasonable agreement with the experimental measurements (cf. Figure 3.28 and 3.29). Some discrepancies are still present and are expected to be caused by misalignments in the real machine.

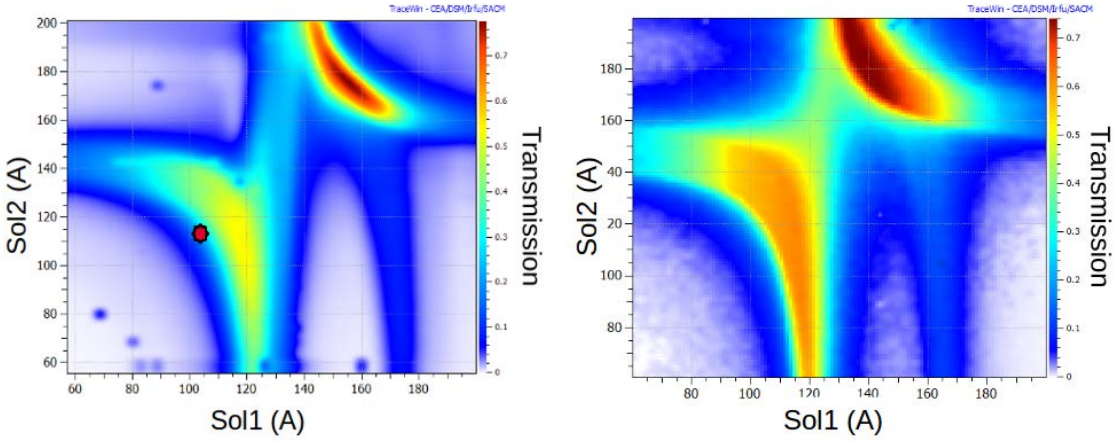

Figure 3.28: Solenoids scan of the IPHI LEBT with a beam current of about 40 mA. Left: experimental scan. Right: Simulated scan. The red dot represents the nominal configuration of the LEBT. [51]

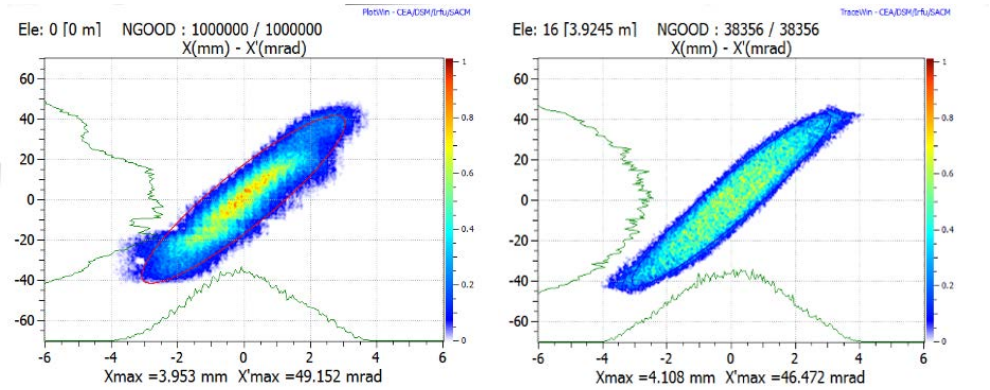

Figure 3.29: Emittance measurements at the exit of the IPHI LEBT with a beam current of about 30 mA. Left: experimental emittance. Right: Simulated emittance.

When looking at the RFQ, the simulated scans of the beam current output and its transmission also show similar behaviors to their respective experimental scans (cf. Figures 3.30 and 3.31 respectively). However, the simulation shows that the setpoint for the second solenoid to have a good transmission should be significantly lower than what was observed during the experiment. This is likely due to the approximations used in TraceWin to account for the space charge effect and its compensation.

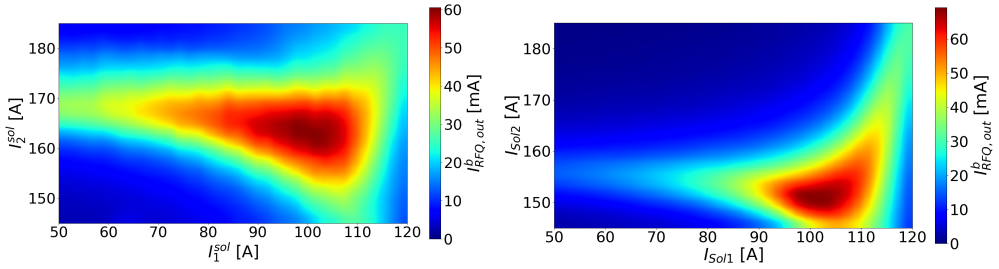

Figure 3.30: Beam current measured at the exit of the RFQ during a solenoids scan of IPHI. Left: experimental scan. Right: Simulated scan.

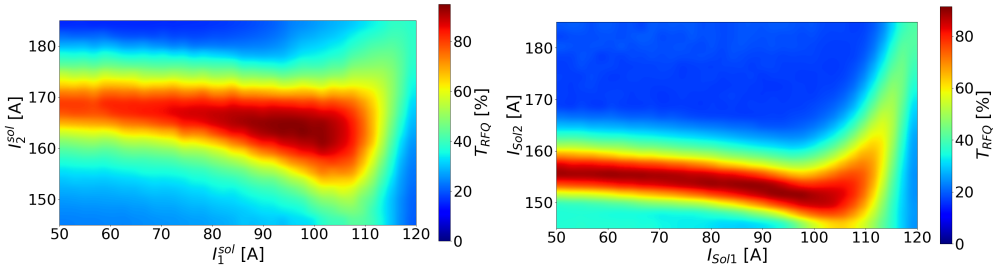

Figure 3.31: RFQ transmission during a solenoids scan of IPHI. Left: experimental scan. Right: Simulated scan.

3.4 Summary

In this chapter, the experimental work performed on two particle injectors, MYRRHA and IPHI, and the calibration of the corresponding TraceWin models have been described.

In the case of MYRRHA, a complete recommissioning of the LEBT has been performed. The ion source and the chopper were successfully restarted and operated. The steerers, the solenoids and the collimator have been used to measure several scans of the beam current at the exit of the LEBT. The injection of Ar gas has been studied to operate in a space charge compensated regime. Online automatic optimization of the LEBT configuration has been performed successfully using a PSO algorithm interfaced with the EPICS controls. Also, the beam emittance and the position of the beam center have been measured in the middle of the LEBT using the Allison scanners and a wire scanner, respectively. These measurements have then been used to calibrate the TraceWin model of the LEBT.

For IPHI, the beam current at the exit of the RFQ and the RFQ transmission have been characterized through several scans of the solenoids with different iris openings. In addition, a record of the beam power has been established. Also, the calibration of the TraceWin model has been discussed.

In both cases, the TraceWin models provides reasonable results but discrepancies with the experimental measurements are still present. They are expected to be caused by the misalignment in the LEBT, the difficulty to simulate the effect of space charge and its compensation. The use of these models is still valid but a configuration determined using those still has to be manually tuned on the machine to obtain the desired results. Taking into account the computation time required, it shows that the development of a neural network based model that would reproduce the experimental results with minimal computation time would provide a valuable tool for the operator.

In Section 2.1.1, it has been introduced that the training of a neural network requires multiple data sets that represent the target behavior to be modeled. In Chapter 4, the constitution of such data sets using experimental data and simulated data of the MYRRHA LEBT and IPHI is discussed as well as the determination of suitable network structures and hyperparameters. Then in Chapter 5, the training neural networks based models of these two machines using these data sets are discussed.

Chapter 4

Model development based on Artificial Neural Networks

Contents

Neural networks can become powerful tools that can even exceed human abilities on specific tasks. However, developing a network for any task can prove to be challenging. Indeed, there exist multiple typical architectures which can be tuned to create a suitable network (cf. Section 2.1.1) but there does not exist clear rules about what choices to make. This is especially true for the hyperparameters (number of hidden layers, number of hidden neurons per layers, learning rate and batch-size). Usually, the developer has to go through a try-and-error process to train many networks with different parameters then select the best performer.

This Chapter is dedicated to the description of the available data sets and the selection process of the hyperparameters that were used to train neural networks to model the MYRRHA LEBT and IPHI.

Firstly, the structure of the data sets is described in Section 4.1.

Then, Section 4.2 is dedicated to the design of the network.

Next, the architecture of the network and its activation functions are discussed in Section 4.2.2.1.

Afterward, convergence studies over the number of hidden layers and the number of hidden neurons per layer against the ability to train the network are shown in Section 4.2.2.3.

And finally, convergence studies on the learning rate and the batch-size are discussed in Section 4.2.2.4.

4.1 Datasets

Datasets are the core of machine learning and especially supervised learning. Indeed, as stated in Section 2.1.1, the goal of supervised learning is to learn the function that maps outputs to their corresponding inputs. This means that the datasets have to represent as best as possible the task at hand. Therefore, this section describes the dataset structure used in the present work.

4.1.1 Description of the function to model

In the present work, neural network training is used to model the behavior of a proton beam in a real accelerator. Such behavior depends on multiple parameters: the beam current and energy, the setpoints of the accelerator components and the residual pressure in the vacuum chamber. The behavior can be characterized using the observables of the proton beam (the physical quantities of the beam that can be measured such as the beam current or Twiss parameters). On a real machine, these observables are determined by the diagnostic tools equipped at various points. Considering this, the goal of the network training is to model the function that maps the state of the machine and the beam characteristic at its entrance to the beam characteristics at the end of the machine.

The real machines studied in this work are the IPHI accelerator (which consists of a LEBT and a RFQ) and the LEBT of the MYRRHA injector (their technical descriptions are given in Section 3.2 and Section 3.1 respectively). Both LEBTs have similar components:

- two solenoids to focus the beam and guide it towards the RFQ input,
- two pairs of steerers to correct the beam alignment and
- a collimator to intercept part of the beam to clean the beam from its halo and control the beam current at the end of the LEBTs.

This means that the configuration of both LEBTs can be described using seven setpoints:

- the current applied in the first and second solenoids, I_1^{sol} and I_2^{sol} respectively,
- the current applied in both pairs of steerers, I_{1V}^{st}, I_{1H}^{st}, I_{2V}^{st} and I_{2H}^{st} where the number indicates in which solenoid the steerer is installed and the letter indicates the direction of the steering (V for vertical and H for horizontal) and
- the collimator opening, r_{col}.

In addition, the LEBT of MYRRHA is equipped with three pressure gauges p_1, p_2 and p_3 that complete the information about the state of the machine. In both cases, the source is considered to provide a constant stable beam ($I_{in}^b = cst$ and $E_{in}^b = cst$) and in the case of IPHI, the RFQ setpoints are considered as constants.

For IPHI, the beam characteristics of interest are:

- the beam current at the end of the RFQ, I^b_{out} and
- the transmission of the RFQ, $t_{RFQ} = I^b_{out}/I^b_{in}$.

Indeed, to minimize the damages in the RFQ caused by impinging particles, beam transmission is a crucial quantity that should be maximized at any beam current.

For MYRRHA, as the RFQ was not installed yet at the time of the data gathering campaign, only the beam current at the end of the LEBT, I^b_{out}, can be easily measured. As such the mapping functions that neural networks are trained to model can be written as:

$$f(s) = f(I^{sol}_1, I^{sol}_2, I^{st}_{1V}, I^{st}_{1H}, I^{st}_{2V}, I^{st}_{2H}, r_{col}) = (I^b_{out}, t_{RFQ}) \tag{4.1}$$

for IPHI and:

$$f(s) = f(I^{sol}_1, I^{sol}_2, I^{st}_{1V}, I^{st}_{1H}, I^{st}_{2V}, I^{st}_{2H}, r_{col}, p_1, p_2, p_3) = I^b_{out} \tag{4.2}$$

for MYRRHA.

The datasets are therefore build to follow these descriptions. This means that the datasets for IPHI contain nine real numbers per example with seven used to describe the state of the machine and two to describe the beam properties corresponding to that state. In the case of MYRRHA, the datasets consist of eleven real numbers per example with ten to describe the state of the LEBT and one for the beam properties.

4.1.2 Dataset for IPHI

One dataset was used to train networks. This dataset was obtained from experimental measurements of the beam current at the exit of the RFQ for several configurations of the LEBT.

4.1.2.1 Experimental dataset

The experimental dataset counts 9828 examples measured directly on the IPHI accelerator with the help of N. Chauvin and D. Uriot. Each example consists of the five entries described below.

1. I^{sol}_1 - Double, The current setpoint for the first solenoid [A]
2. I^{sol}_2 - Double, The current setpoint for the second solenoid [A]
3. r_{col} - Double, The opening radius of the collimator [m]
4. t_{RFQ} - Double, The transmission of the RFQ [%]
5. $I^b_{RFQ,out}$ - Double, The proton beam current at the exit of the RFQ [mA]

The attempts at scanning the steerers were not successful. Indeed, the RFQ operation became unstable due to the beam injection at a wrong angle.

For each example, the first three entries describe a configuration of the LEBT and the last two entries give the corresponding beam current at the output of the RFQ as well as the RFQ transmission. The configuration were selected following a regular scan strategy: for a set collimator opening, the first solenoid setpoint was scanned from 50 A up to 120 A with 2 A steps and the second solenoid setpoint was scanned from 145 A up to 185 A with 2 A steps. So each solenoids scan counts 976 measurements. In total 13 scans were performed for openings ranging from 10 mm to 130 mm with 10 mm steps.

The dataset description including the ranges is given in Table 4.1.

Name	Unit	Type	Lower bound	Upper bound	Step size
I_1^{sol}	A	Double	50	120	2
I_2^{sol}	A	Double	145	185	2
r_{col}	mm	Double	10	130	10
t_{RFQ}	%	Double	12	97	-
$I^b_{RFQ,out}$	mA	Double	0.9	60.6	-

Table 4.1: Structure and ranges of the experimental dataset for IPHI.

Here the standard partition of the dataset was performed while taking care to ensure that all 13 scans are equally represented in each dataset. This means that each scan is randomly separated as follows:

- 60 % to constitute the training dataset,
- 20 % to constitute the validation dataset and
- 20 % to constitute the test dataset.

4.1.3 Datasets for MYRRHA

Three datasets were used to train networks. Two were obtained from TraceWin simulations the last one from experimental measurements. As for IPHI (cf. Section 4.1.2, the end goal is to model the real machine as accurately as possible. However, the quantity of experimental data available for training is very limited due to the availability of the machine and the time it requires to perform measurements (cf. Chapter 3). Hence, although simulated data do not reproduce exactly the real machine behavior, it can be used to supplement the training data sets (cf. Chapter 5).

4.1.3.1 Simulated dataset

Two datasets were computed using a TraceWin model of the MYRRHA LEBT. Both datasets count about 100 000 examples but the selection methods were different. The configurations computed for the first dataset were selected randomly while the configuration computed for the second set followed a regular scanning method.

Beforehand, some efforts were put to calibrate the model of the LEBT using experimental measurements of the emittance and Twiss parameters measured with the Allison scanners (cf. Section 3.3.1.1).

Random selection dataset

This dataset counts 100 537 examples computed using the calibrated model of the MYRRHA LEBT. Each example consists of the eight entries described below.

1. I_1^{sol} - Double, The current setpoint for the first solenoid [A]
2. I_2^{sol} - Double, The current setpoint for the second solenoid [A]
3. I_{1H}^{st} - Double, The current setpoint for the horizontal steerer in the first solenoid [A]
4. I_{1V}^{st} - Double, The current setpoint for the vertical steerer in the first solenoid [A]
5. I_{2H}^{st} - Double, The current setpoint for the horizontal steerer in the second solenoid [A]
6. I_{2V}^{st} - Double, The current setpoint for the vertical steerer in the second solenoid [A]
7. r_{col} - Double, The opening radius of the collimator [mm]
8. $I_{LEBT,out}^{b}$ - Double, The proton beam current at the exit of the LEBT [mA]

For each example, the first seven entries describe a configuration of the LEBT and the last entry gives the corresponding beam current at the exit of the LEBT. The ranges used for the random selection were determined so that only a minimum of simulation would lead to zero transmission. The dataset description including the ranges is given in Table 4.2.

Name	Unit	Type	Lower bound	Upper bound
I_1^{sol}	A	Double	50	110
I_2^{sol}	A	Double	50	110
I_{1H}^{st}	A	Double	-3	3
I_{1V}^{st}	A	Double	-3	3
I_{2H}^{st}	A	Double	-3	3
I_{2V}^{st}	A	Double	-3	3
r_{col}	mm	Double	0	60
$I_{LEBT,out}^{b}$	mA	Double	0	6

Table 4.2: Structure and ranges of the simulated random dataset for MYRRHA.

The whole data set is randomly separated according to the following standard partition:

- 60 % to constitute the training dataset,
- 20 % to constitute the validation dataset and
- 20 % to constitute the test dataset.

Scan dataset

This dataset counts 117 242 examples computed using the calibrated model of the MYRRHA LEBT. Each example consists of the eight entries described below.

1. I_1^{sol} - Double, The current setpoint for the first solenoid [A]
2. I_2^{sol} - Double, The current setpoint for the second solenoid [A]
3. I_{1H}^{st} - Double, The current setpoint for the horizontal steerer in the first solenoid [A]
4. I_{1V}^{st} - Double, The current setpoint for the vertical steerer in the first solenoid [A]
5. I_{2H}^{st} - Double, The current setpoint for the horizontal steerer in the second solenoid [A]
6. I_{2V}^{st} - Double, The current setpoint for the vertical steerer in the second solenoid [A]
7. r_{col} - Double, The opening radius of the collimator [mm]
8. $I_{LEBT,out}^{b}$ - Double, The proton beam current at the exit of the LEBT [mA]

For each example, the first seven entries describe a configuration of the LEBT and the last entry gives the corresponding beam current at the exit of the LEBT. One half of the configurations were selected following a regular scan strategy on the solenoids: for a set collimator opening, both solenoids were scanned from 50 A up to 110 A with 2 A steps. The other half of the configuration was selected following a regular scan strategy on the steerers in the second solenoid: for a set collimator opening, both steerers installed in the second solenoid were scanned from -3 A up to 3 A with 0.2 A steps. In total, 122 scans were computed with collimator openings ranging from 0 mm to 60 mm with 1 mm steps.

The dataset description including the ranges is given in Table 4.3.

The whole data set is randomly separated according to the following standard partition:

- 60 % to constitute the training dataset,
- 20 % to constitute the validation dataset and
- 20 % to constitute the test dataset.

Name	Unit	Type	Lower bound	Upper bound	Step size
I_1^{sol}	A	Double	50	110	2
I_2^{sol}	A	Double	50	110	2
I_{1H}^{st}	A	Double	-3	3	0.2
I_{1V}^{st}	A	Double	-3	3	0.2
I_{2H}^{st}	A	Double	-3	3	0.2
I_{2V}^{st}	A	Double	-3	3	0.2
r_{col}	mm	Double	0	60	1
$I_{LEBT,out}^{b}$	mA	Double	0	6	-

Table 4.3: Structure and ranges of the simulated scan dataset for MYRRHA.

4.1.3.2 Experimental dataset

This dataset counts 19 273 examples measured directly on the MYRRHA LEBT with the help of F. Davin. Each example consists of the eight entries described below.

1. I_1^{sol} - Double, The current setpoint for the first solenoid [A]
2. I_2^{sol} - Double, The current setpoint for the second solenoid [A]
3. I_{1H}^{st} - Double, The current setpoint for the horizontal steerer in the first solenoid [A]
4. I_{1V}^{st} - Double, The current setpoint for the vertical steerer in the first solenoid [A]
5. I_{2H}^{st} - Double, The current setpoint for the horizontal steerer in the second solenoid [A]
6. I_{2V}^{st} - Double, The current setpoint for the vertical steerer in the second solenoid [A]
7. r_{col} - Double, The opening radius of the collimator [mm]
8. p_1 - Double, The pressure inside the chamber measured by the first pressure gauge [mbar]
9. p_2 - Double, The pressure inside the chamber measured by the second pressure gauge [mbar]
10. p_3 - Double, The pressure inside the chamber measured by the third pressure gauge [mbar]
11. $I_{LEBT,out}^{b}$ - Double, The proton beam current at the exit of the LEBT [mA]

For each example, the first seven entries describe a configuration of the LEBT, the three pressure measurements provide insight into the state of the LEBT and the last entry gives the corresponding beam current at the exit of the LEBT.

Out of the whole dataset, 18 259 examples were selected following a regular scan strategy on the solenoids: for a set collimator opening, both solenoids were scanned from 50 A up to 110 A with 2 A steps. In total 19 solenoids scans were included in the dataset with the following collimator openings: from 1.25 mm up to 5 mm with 1.25 mm steps, from 7.5 mm up to 25 mm with 2.5 mm steps and from 30 mm up to 55 mm (fully open) with 5 mm steps.

The other 1 014 examples were selected following a regular scan strategy on the steerer in the second solenoid: for a set collimator opening, both steerers were scanned from -3 A up to 3 A with 0.5 A steps. In total 6 steerers scans were included in the dataset with the following collimator openings: from 5 mm up to 20 mm with 5 mm steps and 55 mm (fully open).

The number of steerers examples is very limited in contrast with the number of solenoids examples. This is because the power supplies of the steerers have a slow response to change in setpoints. Indeed, each configuration took about 20 seconds to be measured against about 2 seconds per configuration in a solenoids scan. This means that a solenoids scan with 961 points was executed in about half an hour while it took about an hour for a steerers scan with 169 points. Hence, the measurement of solenoids scan was prioritized due to the limited time available on the machine.

The dataset description including the ranges is given in Table 4.4.

Name	Unit	Type	Lower bound	Upper bound	Step size
I_1^{sol}	A	Double	50	110	2
I_2^{sol}	A	Double	50	110	2
I_{1H}^{st}	A	Double	-3	3	0.5
I_{1V}^{st}	A	Double	-3	3	0.5
I_{2H}^{st}	A	Double	-3	3	0.5
I_{2V}^{st}	A	Double	-3	3	0.5
r_{col}	mm	Double	0	55	variable
p_1	mbar	Double	$1.5\ 10^{-5}$	$2.3\ 10^{-5}$	-
p_2	mbar	Double	$1.1\ 10^{-5}$	$1.8\ 10^{-5}$	-
p_3	mbar	Double	$8.1\ 10^{-6}$	$1.2\ 10^{-5}$	-
$I_{LEBT,out}^{b}$	mA	Double	0	6.1	-

Table 4.4: Structure and ranges of the experimental scan dataset for MYRRHA.

As for the experimental dataset of IPHI (cf. Section 4.1.2.1), the standard partition of the dataset was performed while taking care to ensure that all 19 solenoids scans and all 6 steerers scans are equally represented in each dataset. This means that each scan is randomly separated as follows:

- 60 % to constitute the training dataset,
- 20 % to constitute the validation dataset and
- 20 % to constitute the test dataset.

4.2 Neural Network structure and hyperparameters determination

The TensorFlow API (v1.13) was used with Python (v3.6) to implement the neural networks and the supervised learning training. TensorFlow is a free and open-source machine learning platform offering a variety of tools to train, monitor and use neural networks [109].

Here, the processes to optimize and design the final neural network selected to model injectors are described.

At first, the choice of network architecture is discussed in Section 4.2.1.

Then, the selection of the activation functions is described for the hidden layers in Section 4.2.2.1 and the output layer in Section 4.2.2.2.

Next, the network performance is studied against the number of hidden layers and the number of neurons per layer in Section 4.2.2.3.

Finally, the learning rate and batch-size are determined against the efficiency of the training in Section 4.2.2.4.

4.2.1 Architecture discussion

In this document, the goal is to create a model of the beam dynamics in a particle accelerator. To this end, neural networks are good candidates as they can be used as universal approximators. However, the choice of the network architecture (cf. Section 2.1.1.1) can sometimes be difficult as there exists many possibilities. Fortunately, each architecture has strengths and drawbacks that can guide the user to make a suitable choice for its dataset and the targeted application.

For regression tasks such as the one discussed in this work, the goal is to predict an output based on an input. This means that generative networks such as GANs (Generative Adversarial Networks) are not suited as their goal is to generate realistic but fake data.

In addition, the available dataset consists of real numbers that describe the configuration of the machine as inputs and the beam properties as outputs. This means that convolutional networks are not suited for the task either as they are designed to take image-like inputs.

Finally, the datasets are either simulated or experimental but represent settled states of the machine i.e. the dataset does not contain any time evolution. This means that time dependent networks such as recurrent networks can also be discarded.

These considerations leave two types of networks to discuss: multi-layer perceptrons (MLPs) and autoencoders (AEs). Actually, between these two options, MLPs are the clear choice for regression tasks. Indeed, they are built to model a function that maps continuous inputs to continuous outputs. But it is interesting to think about AEs as they can be complementary to MLPs and help their training. As a brief reminder, AEs are trained so that their outputs reproduce their inputs with high fidelity with the constraint that the number of neurons in one of the hidden layers is

lower than the number of their inputs. This effectively creates a bottleneck and forces the network to condense the information contained in the inputs. In other words, half of the AE is trained to downsize the dimensionality of the inputs with the lowest loss of information possible. This process (called *feature extraction*) can be useful when the number of inputs is very high or if the user suspects that the inputs are not the best representation of the problem at hand. In such cases, an AE can initially be trained on the dataset. Then, its first layers up to the bottleneck are used as the input for the training of a MLP. However, in this work, this option has not be tested as the inputs are representative (cf. Sections 4.1.2 and 4.1.3) and in low number (up to ten).

In conclusion, for this work, it seems that using MLPs is the choice to make. This choice is also supported by similar works reported by Edelen [95, 97].

4.2.2 Activation function and hyperparameters

For regression tasks, there are three typical activation functions: sigmoid, hyperbolic tangent (tanh) and rectified linear unit (ReLU) (cf. Section 2.1.1.1). The sigmoid and tanh activation functions provide essentially the same properties but the use of the tanh is usually preferred over the sigmoid function [110]. In addition to the selection of the activation function, it is also required to determine the hyperparameters of the networks and the supervised learning method: number of hidden layers, number of neurons per layer, learning rate and batch-size. Taking all these variables into account, it is clear that there is an infinite number of possible combinations and that the user could spend a very long time tuning a network. In practice, several combinations should be tested to select the best one among those.

The comparison of two networks with different combinations of hyperparameters is not straightforward. Even more, when training two networks with identical complexities on the same datasets can lead to very different final performances. This is due to the intrinsic random dependencies of the network:

- the weights are randomly initialized hence the training can begin from a more or less favorable start,
- the examples are randomly selected from the training set which can bias the learning and
- the examples are randomly partitioned into training, validation and test set which can also lead to biased training.

Therefore, to compare networks, a technique called *cross-validation* is commonly used [111]. This method consists of taking k different samplings of the training and validation dataset to train k networks with identical complexities (cf. Figure 4.1). The networks are then evaluated on the test dataset and their averaged performance (evaluated with the Mean Squared Error between the network predictions and the target values) is used as a less biased evaluation of the network complexity. This evaluation can then be used for comparison between different network complexities.

The results from cross-validation studies performed in this work are reported here.

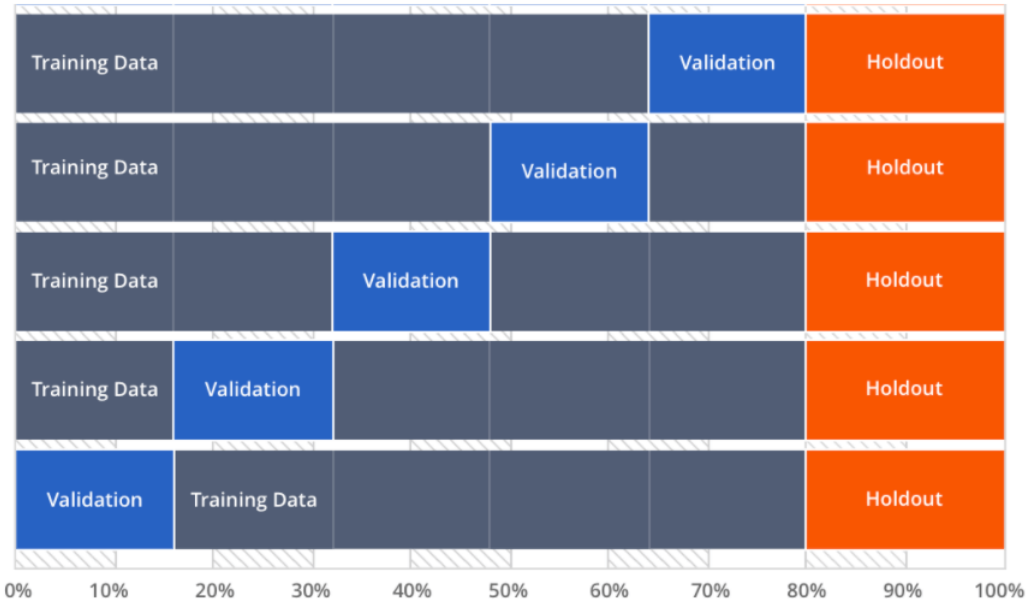

Figure 4.1: Dataset partitions for cross validation with $k = 5$. [112]

In Section 4.2.2.1, the tanh and ReLU activation functions have been tested for the hidden layers.

In Section 4.2.2.2, the sigmoid and ReLU activation function have been tested for the output layer.

In Section 4.2.2.3, the number of hidden layers and the number of units per layer are determined.

And finally, in Section 4.2.2.4, the learning rate and the batch-size are determined.

4.2.2.1 Hidden layers activation function

To compare the tanh and ReLU activation functions for the hidden layers, two networks have been build to be trained on the random simulated dataset of MYRRHA (cf. Section 4.1.3.1). Both networks have been build with the following structure: one input layer with 7 neurons (corresponding to I_1^{sol}, I_2^{sol}, I_{1H}^{st}, I_{1V}^{st}, I_{2H}^{st}, I_{2V}^{st} and r_{col}) three hidden layers with 96 neurons each and one output layer with 1 neuron (corresponding to $I_{LEBT,out}^{b}$). One of the networks was built with the tanh activation function for the hidden layers while the other used the ReLU activation function. The output neuron used the sigmoid activation function. The training was performed with a learning rate of 0.1 and a batch-size of 128.

The inputs were scaled using the following formula:

$$x_{scaled} = \frac{x - \frac{x_{max}+x_{min}}{2}}{\frac{x_{max}-x_{min}}{2}} = \frac{2x - x_{max} - x_{min}}{x_{max} - x_{min}}, \tag{4.3}$$

where x_{min} and x_{max} are respectively the minimum and maximum value in the training dataset for the variable x. This means that $x_{scaled} \in [-1, 1]$ is centered around zero

which helps with training [110].[1]

The output was also scaled to fit the $[0, 1]$ range of the sigmoid function used by the output neuron:

$$I^b_{LEBT,out,scaled} = \frac{I^b_{LEBT,out}}{10}. \tag{4.4}$$

The final Mean Squared Error (MSE, cf. Equation 2.7) of the models on the test dataset were equal to $\text{MSE}(I^b_{LEBT,out,scaled}) = 3.3310^{-3}$ mA^2 for the tanh network and to $\text{MSE}(I^b_{LEBT,out,scaled}) = 1.810^{-3}$ mA^2 for the ReLU network. Based on the MSEs, it seems that ReLU activations give better performances than tanh activations. Hence, the ReLU activation has been favored in this work.

4.2.2.2 Output layer activation function

The comparison of four activation functions (tanh, sigmoid, linear and ReLU) for the output layer has been performed in this work. Three networks have been build to be trained on the random simulated dataset of the MYRRHA LEBT (cf. Section 4.1.3.1). The network was built with the following structure: one input layer with 7 neurons (corresponding to $I^{sol}_{1,scaled}$, $I^{sol}_{2,scaled}$, $I^{st}_{1H,scaled}$, $I^{st}_{1V,scaled}$, $I^{st}_{2H,scaled}$, $I^{st}_{2V,scaled}$ and $r_{col,scaled}$ (cf. Equation 4.3) three hidden layers with 96 neurons each and one output layer with 1 neuron. For all the networks the hidden layers used the ReLU activation function. One network used the tanh activation for its output neuron, one used the sigmoid activation, one used the linear activation and the final one used the ReLu activation. The training was performed with a batch-size of 128 and a learning rate of 0.1 for the tanh, sigmoid and ReLU activations and of 10^{-7} for the linear activation.

The inputs were scaled following Equation 4.3.

Depending on the activation function of the output neuron, the output was scaled following Equation 4.4 for the sigmoid and ReLU output neuron and

$$I^b_{LEBT,out,scaled} = \frac{I^b_{LEBT,out} - 5}{5}, \tag{4.5}$$

for the tanh output neuron to fit the [-1, 1] range of the tanh function. In the case of the ReLU and linear output neuron, the range is $[0, \infty[$ and $]-\infty, \infty[$ respectively hence a test has also been performed without scaling the output.

During this study, the training of the ReLU and linear output networks with the non-scaled output $I^b_{LEBT,out}$ was unsuccessful. None of the networks managed to learn anything representative of the LEBT behavior despite numerous attempts. Indeed, during their training, the MSE output was either a negative value or a NaN flag which prevented the Stochastic Gradient Descent method to work properly. As such, the output was systematically scaled for the rest of this work.

Even with scaling the output, the training of the linear output network was unsuccessful and was therefore abandoned. Similarly, the ReLU and tanh output networks

[1] In order to convince ourselves, training without scaling was attempted but to no avail.

failed to learn anything else than to output 0 and -1, respectively, despite numerous attempts. Fortunately, the training of sigmoid output networks was successful (the trained network was able to reproduce the general behavior of the LEBT). Hence, the sigmoid output neuron was selected for this work.

4.2.2.3 Network hyperparameters determination

Determining the number of hidden layers and the number of neurons per layer requires testing a wide variety of network structures. In practice, this process is performed according to one of the following procedures: manual search, grid search (exhaustive search through a subset of the hyperparameter space) and random search.

In this work, hyperparameter tuning was performed following a grid search procedure using the random simulated dataset of MYRRHA (cf. Section 4.1.3.1). For each set of hyperparameters, four networks were trained according to a cross-validation approach with $k = 4$ and for 100 epochs. The grid search tested 15 different structures with 32, 64, 96, 128 and 192 neurons per hidden layer and 2, 3 and 4 hidden layers. The input layer consists in 7 neurons (corresponding to I_1^{sol}, I_2^{sol}, I_{1H}^{st}, I_{1V}^{st}, I_{2H}^{st}, I_{2V}^{st} and r_{col}, cf. Equation 4.3). The hidden layers used ReLU neurons and the output layer used a sigmoid neuron (corresponding to $I_{LEBT,scaled}^{b}$, cf. Equation 4.4). The training was performed with a learning rate of 0.1 and a batch-size of 128.

The final average performance of each structure is given in Table 4.5. According to this study, the network with 4 layers and 96 neurons per layer performed best with an average MSE of 0.0017 and should be selected. However, training the networks with 4 layers was fairly difficult and required to restart the training multiple times before actually obtaining some learning. This is especially made apparent by the performance of the network with 4 layers and 192 neurons per layer with a final average MSE of about 0.02. From these considerations, the network with 3 layers and 96 neurons per layer has been selected for this study as it offers good performances (low average MSEs) and is easier to train (less failed training attempts and shorter training time).

	32 neurons	64 neurons	96 neurons	128 neurons	192 neurons
2 layers	0.44	0.77	0.42	0.38	0.62
3 layers	0.63	0.19	0.18	0.37	0.24
4 layers	0.22	0.43	0.17	0.18	2.02

Table 4.5: Average MSE in mA^2 of the networks depending on the number of hidden layers and the number of neurons per layer.

4.2.2.4 Supervised learning hyperparameters determination

The supervised learning algorithm with stochastic gradient descent takes two hyperparameters: the learning rate and the batch-size. Both should be optimized using a cross-validation approach to ensure that the required training time to reach good

performance is reduced. In this work, the learning rate and the batch-size have been optimized successively.

Learning rate optimization

Six learning rates have been tested in this work: 0.01, 0.05, 0.08, 0.1, 0.5 and 1. Those were tested according to a cross-validation approach with $k = 4$ on the random simulated dataset of MYRRHA (cf. Section 4.1.3.1) with a training of 100 epochs. The structure of the network used for this study was: one input layer consists in 7 neurons (corresponding to $I^{sol}_{1,scaled}$, $I^{sol}_{2,scaled}$, $I^{st}_{1H,scaled}$, $I^{st}_{1V,scaled}$, $I^{st}_{2H,scaled}$, $I^{st}_{2V,scaled}$ and $r_{col,scaled}$, cf. Equation 4.3), three hidden layers with 96 ReLU neurons each and one output layer with a sigmoid neuron (corresponding to $I^{b}_{LEBT,scaled}$, cf. Equation 4.4). The batch-size was set to 128.

The final average performances for each learning rate are given in Figure 4.2. The training achieves the lowest MSEs for a learning rate between 0.1 and 0.05 with the best one at 0.1. Hence, an initial learning rate of 0.1 was selected in this work. To fine tune the network, the learning rate was also scheduled to be divided by five times when the training did not improve the performances for twenty consecutive epochs with a minimum of 0.0001.

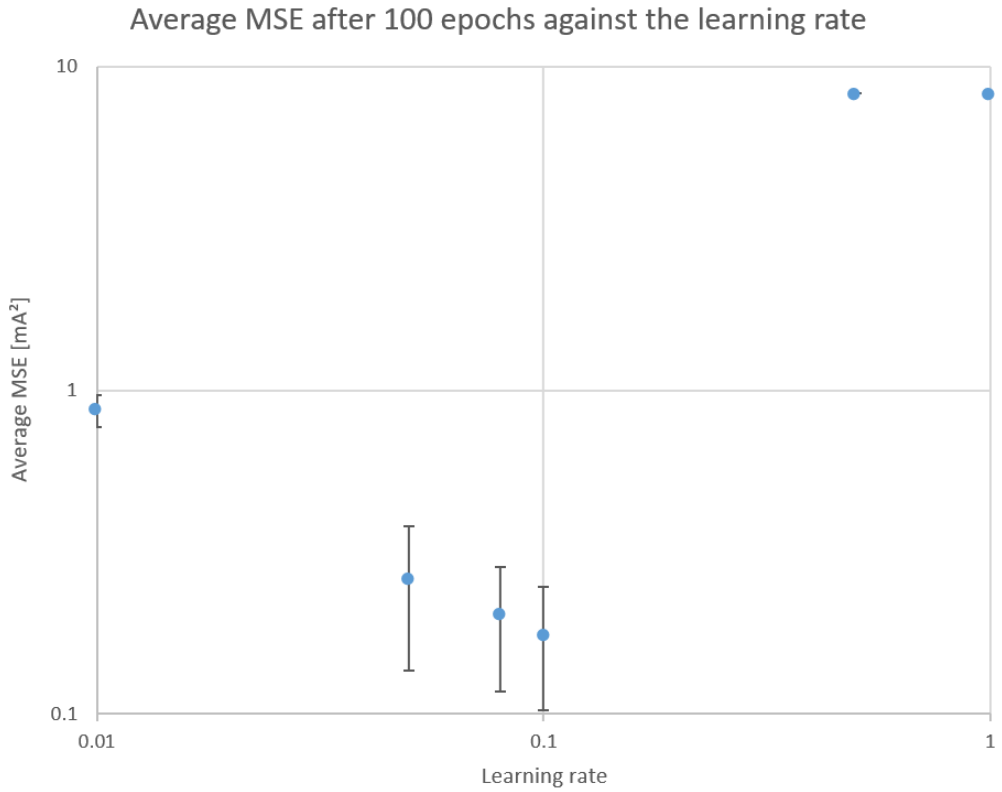

Figure 4.2: Average MSE for the prediction of trained networks on the MYRRHA random simulated dataset against the learning rate.

Batch-size optimization

Six batch-sizes have been tested: 32, 64, 96, 128, 160 and 192. They were tested following the same approach as for the learning rate. The learning rate was set to 0.1.

The final average MSE for each batch-size is given in Figure 4.3. The best results were obtained with batch-sizes of 64 and 96 with a slightly better performance with 64 examples which has been selected in this work.

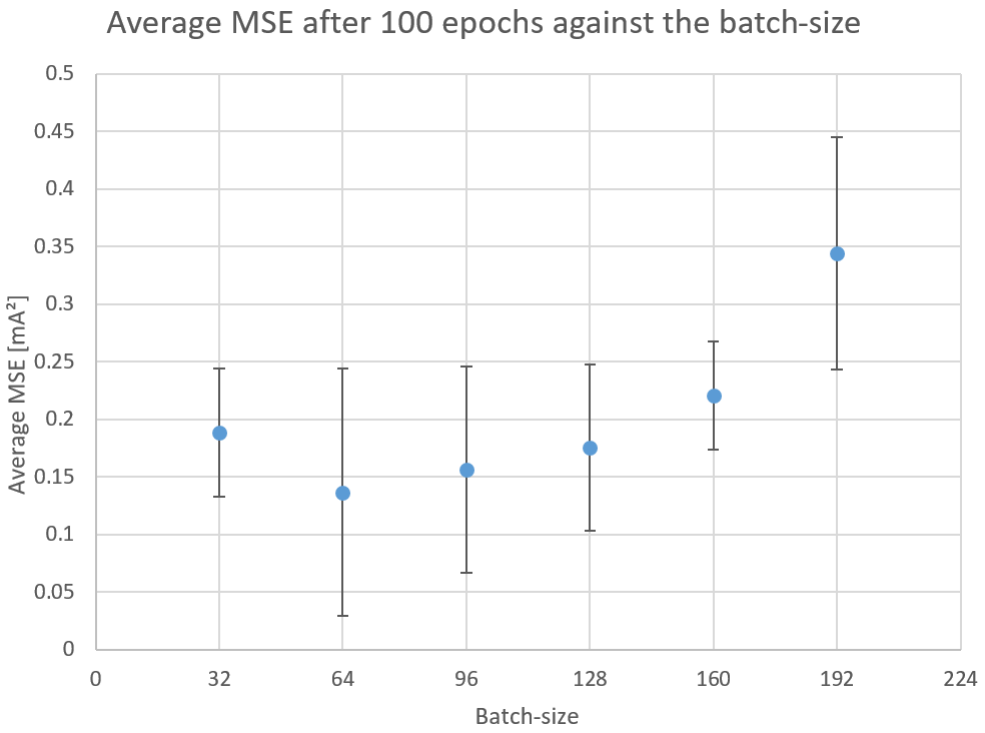

Figure 4.3: Average MSE for the prediction of trained networks on the MYRRHA random simulated dataset against the batch-size.

4.3 Summary

In this chapter, the function to model has been described for both the IPHI and MYRRHA machines. In both cases, the function takes the state of the machine as input and provides the relevant beam properties at the exit of the machine (the exit of the MYRRHA LEBT or the exit of the IPHI RFQ).

Then, the datasets available to train the neural networks were described. For IPHI, one simulated dataset and one experimental dataset were available to train networks. For MYRRHA, two simulated datasets with different sampling strategies and one experimental dataset were available.

Finally, the process followed to determine the structure and the hyperparameters of the neural networks as well as the hyperparameters of the supervised learning methods were described. In this work, MLPs are favored for their universal approximator properties. The choice for the various hyperparameters was settled following a cross-validation approach with $k = 4$ over 100 epochs. For the neural network hyperparameters, the final performances favored a structure with three hidden layers with 96 ReLU neurons each and one output layer with sigmoid neuron(s). For the supervised learning methods, the hyperparameters were set to an initial learning rate of 0.1 and a batch-size of 64.

Chapter 5

NN training and results

Contents

As a reminder, the goal of this thesis is to explore the possibility to train neural networks to model an injector as well as test new tools for the optimization of an injector. In particular, the aim is to develop a suite of tools that would facilitate the operation of the MYRRHA injector. As such, the idea is to train neural networks with the experimental and simulated dataset of the MYRRHA LEBT to model the beam current at the exit of the LEBT with respect to its configuration. Also, in preparation for the installation of the MYRRHA RFQ[1], neural networks are also trained with the experimental dataset of the IPHI injector to model the beam current and the beam transmission at the exit of the RFQ. Then, the PSO algorithm is tested on the resulting models.

In this chapter, multiple training strategies are discussed with respect to the performances of the resulting networks. In addition, the application of the PSO algorithm on the best performing models is described.

[1]The RFQ was not available at the time of the collection of experimental data but is now operational.

5.1 MYRRHA model

The training of a neural network to model the MYRRHA LEBT aims at obtaining a regression model that reproduces the behavior of the LEBT. As the RFQ was not available during the gathering of the datasets, it was chosen to train the model to predict the beam current that can be measured at the exit of the LEBT (the beam current to be injected into the RFQ) for a wide range of setpoints on the solenoids, the steerers and the collimator. The training is based on the experimental data collected during the commissioning of the LEBT at Louvain-la-Neuve and data simulated using the TraceWin model of the LEBT (cf. Section 4.1.3).

Ideally, one would only use experimental data to train a model to reproduce the behavior of the real machine. However, because of how the experimental data is gathered (cf. Section 3.1), the experimental dataset contains "only" about twenty thousand examples. This number may seem low compared to the hundreds of thousands or millions of examples that are available in some training datasets [113, 114] but the actual number of training data required to model a LEBT is unknown. Indeed, the heuristic is that there should be enough data to be representative of the behavior to model. Hence, we started by training a model only using the experimental dataset. Then, a second model was trained using the simulated dataset (with random sampling) that contains about five times more examples than the experimental dataset. Finally, the model trained on the simulated dataset has been retrained using the experimental dataset.

This section is dedicated to the description and discussion of the training and the performances of these models.

The network trained using experimental data is discussed in Section 5.1.1.

Then, the network trained using the simulated data is described in Section 5.1.2.

Finally, Section 5.1.3 shows the training of the network retraining.

5.1.1 Network training using experimental data

The first attempts to train a model on the experimental dataset ($\sim$ 20 000 measured configurations) described in Section 4.1.3.2 were performed using the network structure and hyperparameters determined through the cross-validation process described in Section 4.2. In this section, this set of hyperparameters is referred to as the initial hyperparameters. As a reminder, the initial neural network structure is given in Table 5.1. This network was trained with a batch-size of 64 examples and an initial learning rate of 0.1. The inputs and the output are reminded in Table 5.2.

Layer type	Input	Hidden	Hidden	Hidden	Output
# of neurons	10	96	96	96	1
Activation function	-	ReLU	ReLU	ReLU	sigmoid

Table 5.1: Initial neural network structure for the training on the MYRRHA experimental dataset.

	Name	Symbol
Inputs	First solenoid setpoint	I_1^{sol}
	Second solenoid setpoint	I_2^{sol}
	First horizontal steerer setpoint	I_{1H}^{st}
	First vertical steerer setpoint	I_{1V}^{st}
	Second horizontal steerer setpoint	I_{2H}^{st}
	Second vertical steerer setpoint	I_{2V}^{st}
	Collimator position	r_{col}
	Pressure in first gauge	p_1
	Pressure in second gauge	p_2
	Pressure in third gauge	p_3
Output	Beam current	$I_{LEBT,out}^{b}$

Table 5.2: Inputs and output of the experimental dataset of the MYRRHA LEBT.

A typical training curve of this initial network is shown in Figure 5.1. This curve indicates that the network learns quickly for about ∼10 epochs then slows down and finally progresses very slowly after ∼50 epochs. This behavior is essentially expected however the training stagnates at a point where the model reproduces the general behavior (cf. Figure 5.2) but with a higher MSE than expected from the cross-validation process. Indeed, the MSE reached at the end of the training is ∼4.45 mA^2 while the MSE reached during the cross-validation process was ∼0.2 mA^2 (cf. Table 4.5).

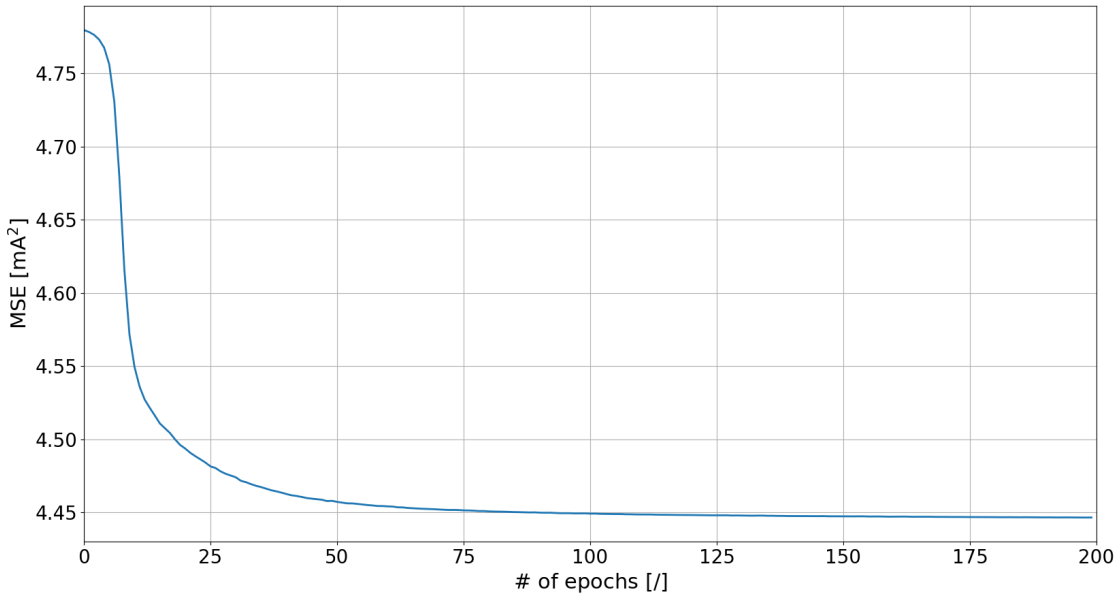

Figure 5.1: Typical training curve for the initial neural network structure on the MYRRHA experimental dataset.

The difference between the obtained and expected performances likely comes from the difference between the experimental dataset (on which the network has been trained) and the simulated dataset (on which the cross-validation has been performed).

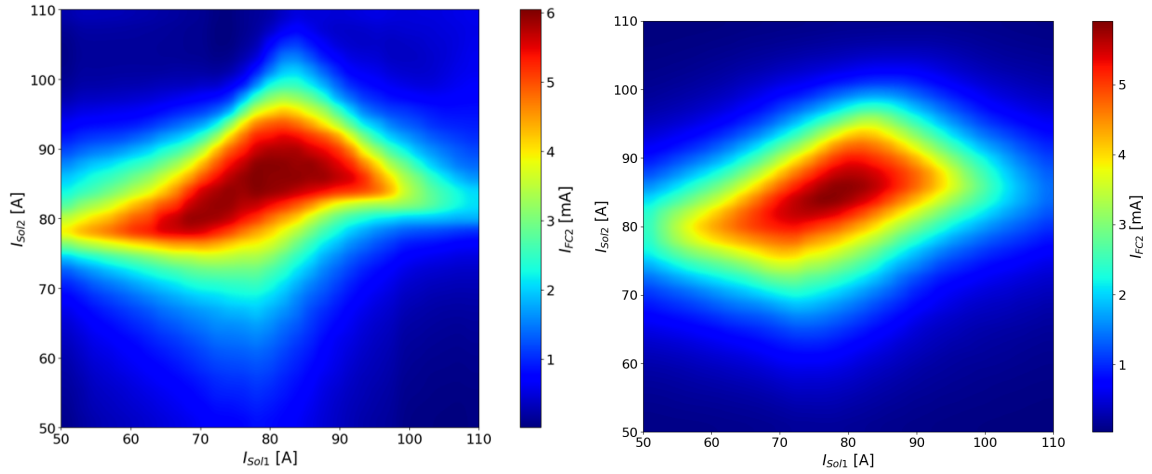

Figure 5.2: Beam current at the exit of the LEBT during a solenoids scan. Left: experimental scan. Right: scan reproduced with the trained initial network.

Indeed, the simulated dataset counts ∼5 times more data than the experimental dataset. In addition, the simulated dataset is randomly distributed in the configuration space of the LEBT while the experimental dataset is organized along regular scans of the machine.

Nevertheless, with a trial and error process on the hyperparameters, a network has been trained with much better performance on the experimental dataset. The main differences with the initial hyperparameters are the decrease in neurons per hidden layer from 96 down to 64 (cf. Table 5.3) and the increase of the batchsize from 64 up to 128.

Layer type	Input	Hidden	Hidden	Hidden	Output
# of neurons	10	64	64	64	1
Activation function	-	ReLU	ReLU	ReLU	sigmoid

Table 5.3: Best performing neural network structure for the training on the MYRRHA experimental dataset.

The training curve of the best performing network is shown in Figure 5.3. One can easily see that this new network performs much better than the initial one. According to the MSE, the training improves the network by 1 order of magnitude over around 30 epochs. Then, there is a first slow down with a gain of 1 order of magnitude over around 200 epochs. Then a second slow down with a gain of a factor 2 over around 300 epochs. At epoch #600, the function used to estimate the error of the network was changed in an attempt to speed up the learning process. As a result, the MSE of the network reaches its lowest at ∼650 epochs with a value of $\sim 2.4 \times 10^{-3}$ mA^2. However, the inspection of the data revealed that even if the MSE got lower the actual performances of the network became worse and the error estimation was reverted back for around 40 epochs. Indeed, the comparison between the real data and the

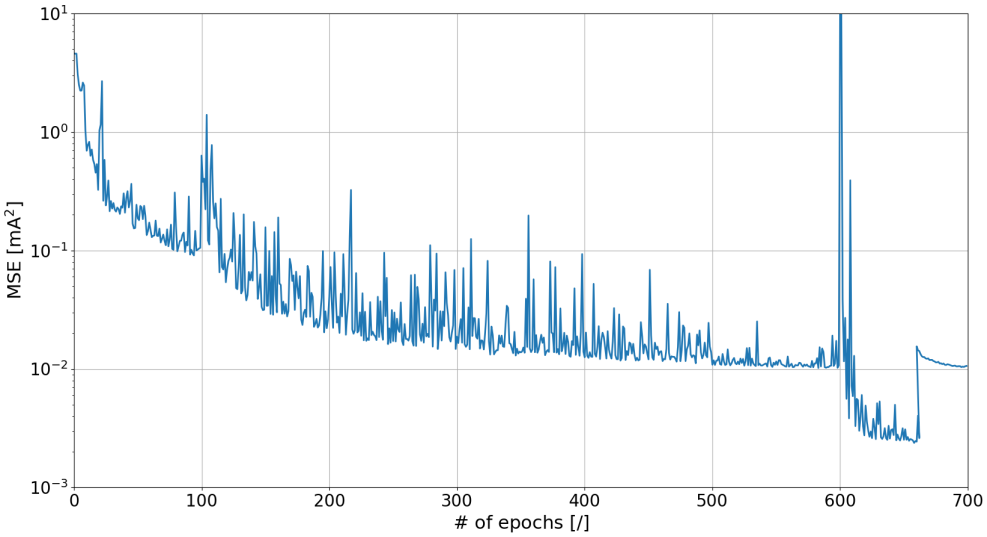

Figure 5.3: Training curve for the best performing neural network on the MYRRHA LEBT experimental dataset.

predictions of the model at epoch #650 shows that the network has overfitted part of the data. As an illustration, Figure 5.4 shows that the experimental data with the collimator fully opened (top row) is reasonably reproduced by the model while the data with the collimator mostly closed (bottom row) is not reproduced at all.

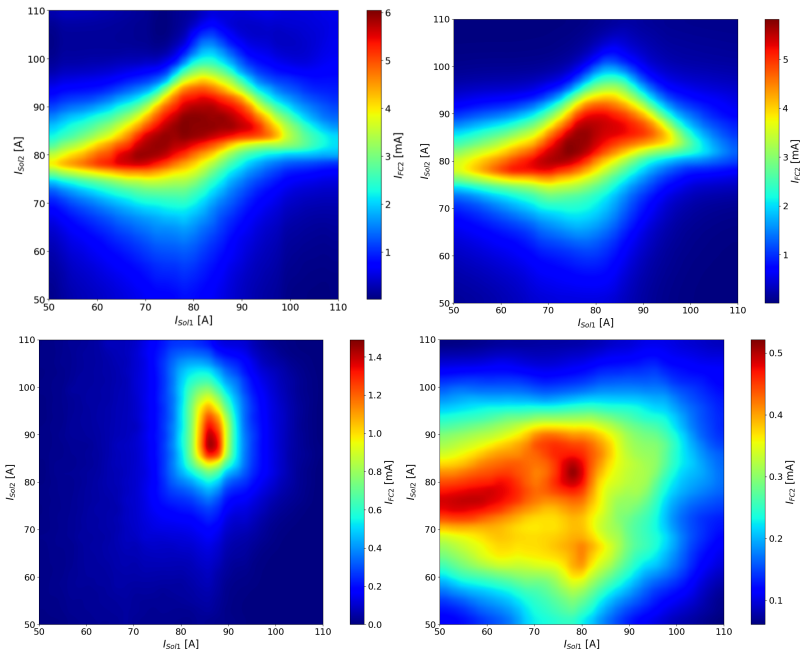

Figure 5.4: Measured beam current at the exit of the LEBT during a solenoids scan with r_{col} = 55 mm (top) and r_{col} = 3.75 mm (bottom). Left: experimental scans. Right: scans reproduced with the best performing network at 650 epochs.

When investigating the predictions of the model at the end of the training (at epoch #700), the network can reproduce every experimental scan (cf. Figure 5.5). With these considerations, the apparent loss in performances over the last few dozens of epochs i.e. the MSE increases from $\sim 2.4 \times 10^{-3}$ mA^2 up to $\sim 10^{-2}$ mA^2 is actually desirable as it indicates that the network is generalizing. This shows that the MSE is not suitable as the sole performance indicator and thus that the predictions of the model should be manually inspected to rule out major kinks.

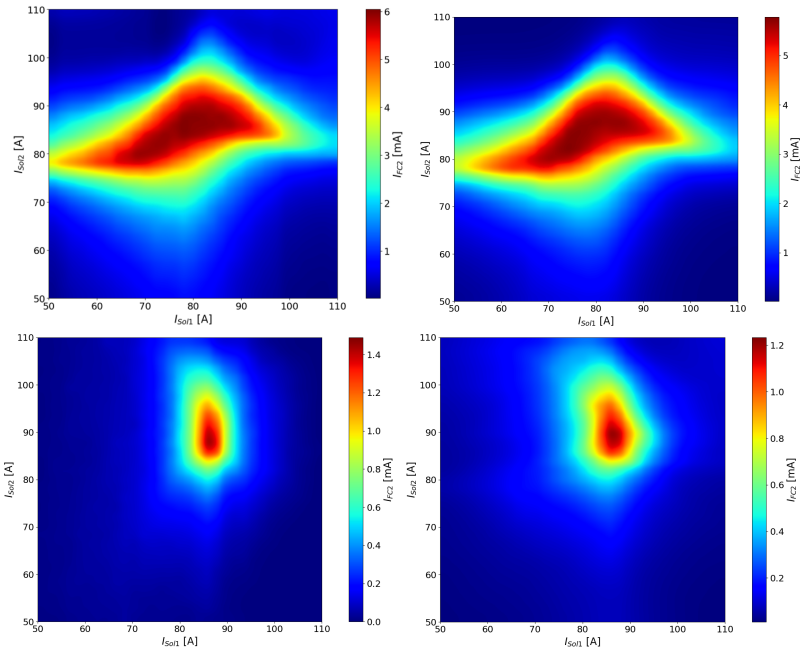

Figure 5.5: Beam current at the exit of the LEBT during a solenoids scan with $r_{col} =$ 55 mm (top) and $r_{col} = 3.75$ mm (bottom). Left: experimental scan. Right: scan reproduced with the best performing network at 700 epochs.

As such, the absolute error in the model predictions is also a good indicator. The difference between the experimental and the predicted scans is shown in Figure 5.6. Overall, the model can reproduce the measurements within ± 0.5 mA. This confirms that the interpolation made by the network within the scans[2] is reasonably good.

However, the interpolation with respect to the collimator opening presents the following problems (cf. Figure 5.7):

- The first problem is the overshoot present close to the fully open configurations i.e. close to $r_{col} = 55$ mm. This is caused by the fact that a network is a continuous function. Indeed, here the output of the network is 0 outside of the training range (below $r_{col} = 2.5$ mm and above $r_{col} = 55$ mm). This is adequate for the collimator position $r_{col} < 0$ mm because no beam is transmitted through the LEBT. However, on the other side, the absence of training data between $r_{col} = 45$ mm and $r_{col} = 55$ mm leaves the network free to overshoot as a form

[2]As a reminder, the training data is selected so that it contains 60 % of each scans.

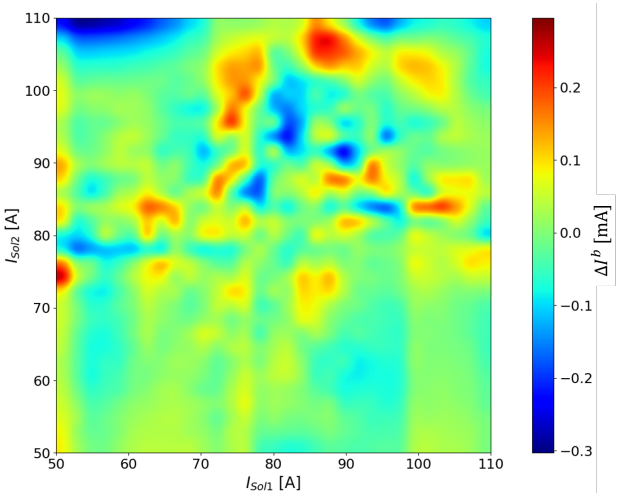

Figure 5.6: Difference expressed in mA between measurements and predictions for the beam current measured at the exit of the LEBT during a solenoids scan with $r_{col} = 55$ mm.

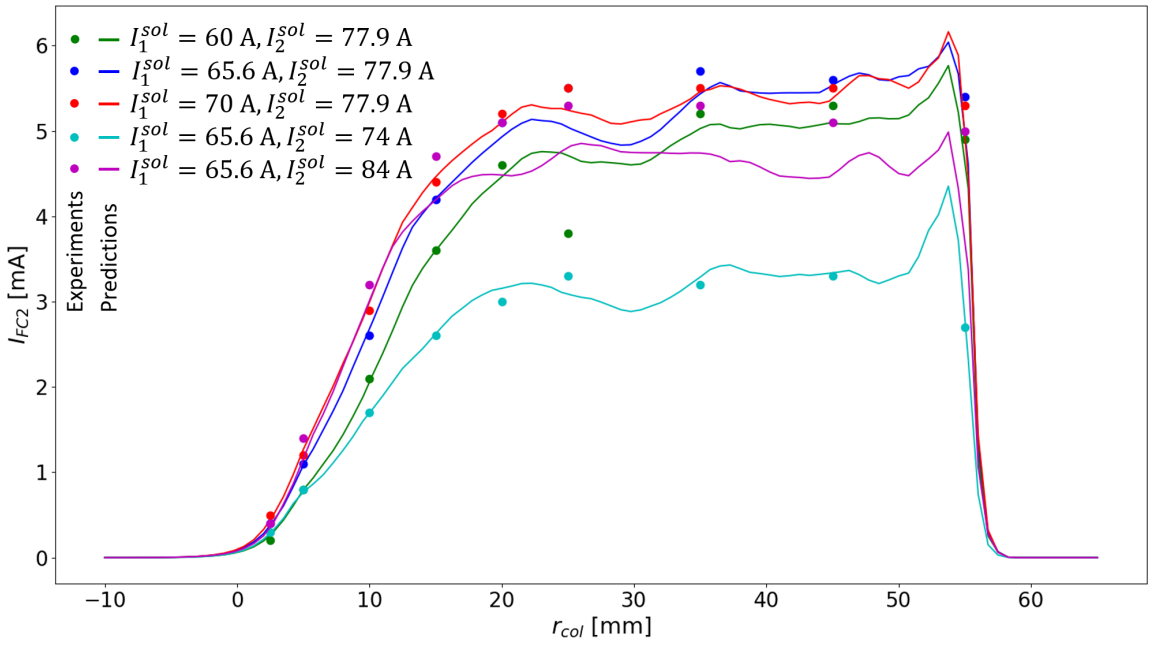

Figure 5.7: Comparison of measurements (dots) and predictions (lines) for five configurations close to the design configuration ($I_1^{sol} = 65.6$ A and $I_2^{sol} = 77.9$ A) with respect to the collimator opening. The error in the experimental measurements is within the size of the symbol.

of overfitting to the last experimental data point and the output of the network out of the training range.

- The second problem is the fluctuations that can be observed between $r_{col} = 25$ mm

and $r_{col} = 55$ mm. Indeed, in this region, the curves should be essentially flat as the collimator does not intercept the beam. Once again, this is likely caused by the absence of training data between two sampled collimator positions.

- The third problem is the underestimation of the purple curve over the plateau. This is likely due to the network not fitting correctly some of the gradients present in the training data. This will be discussed in more depth in Section 5.1.2.

The first and second problems can likely be addressed by increasing the amount of training data. However, collecting experimental data is difficult because of the time it takes and the limited availability of the machine. As such, one can resort to two other solutions: using data augmentation or simulated data.

Data augmentation

Data augmentation is a practice in Machine Learning that consists of artificially increasing the amount of training data based on non-biasing operations[3] on the initial dataset. The difficulty of this technique stands in the qualifier "non-biasing". Indeed, it is not always straightforward if a specific operation will introduce a bias in the training data. Let us discuss a few approaches to use data augmentation in the context of the MYRRHA LEBT.

On the one hand, one can use this method by prolonging the training dataset for $r_{col} > 55$ mm. Indeed, although it does not have physical meaning (the collimator cannot be more open than fully) it allows increasing the training range. Hence, the boundary that causes the overshoot close to $r_{col} = 55$ mm would be displaced to a collimator position that is irrelevant for our usage. This approach is obviously non-biasing as it adds data outside of the range of interest.

On the other hand, one can assume the fluctuations observed between the sampled collimator positions can also benefit from data augmentation. The simplest approach would be to linearly interpolate between two scans which would allow the generation of an arbitrary number of additional training data. However, this approach suffers from the fact that the beam dynamics in the LEBT are non-linear and thus the interpolation can generate data that would bias the network towards a linear model. This should be avoided as it would defeat the purpose of training a non-linear model.

Of course, one can argue that a non-biasing alternative for the latter approach is to linearly interpolate between two points only if they do not change in value between two scans i.e. two configurations for which the change in collimator position has no effect. However, this alternative presents an issue that comes from the continuous nature of a network. Indeed, when trained on a given example, the change in the network parameters will cause the output to change for the given example as well as for neighbouring examples. Hence, it is likely that increasing the data only where linear interpolation is reasonable will cause the network to be biased towards the linear model.

[3] A simple example is mirroring pictures to double a dataset to train a CNN. Indeed, a CCN should be able to correctly label a cat whether it faces one direction or the other.

Simulated data

In contrast with data augmentation, using simulated data is very straightforward. Using simulation codes of the beam dynamics, it is possible to generate a large quantity of data (limited only by the available time and computation resources). If the code reproduces perfectly the real machine then the simulated data could be directly added to the experimental data. However, in most cases, there are discrepancies between simulated models and experiments. This is also true for the simulated model of the MYRRHA LEBT (cf. Section 3.3.1.3). Hence, the simulated data should not be merged directly with the experimental data.

In this work, the use of simulated data was preferred to data augmentation. Indeed, even with the discrepancies, simulated data contains at least part of the dynamics that we want to model. In addition, a dataset that consists of only simulated data can be generated with an arbitrary number of examples. This means that one can see if more data benefits the training. Thus, Section 5.1.2 is dedicated to the training of a network on a dataset with only simulated data.

5.1.2 Network training using simulated data

A model was trained using the simulated dataset with random sampling (cf. Section 4.1.3.1). This dataset contains 100 000 configurations simulated with the TraceWin model of the MYRRHA LEBT. The structure of the network and the hyperparameters of the training are the ones determined using cross-validation (cf. Section 4.2) i.e. the structure described in Table 5.1, an initial learning rate of 0.1 and a batchsize of 64.

A typical training curve is shown in Figure 5.8. At first, the training quickly improves the network as the MSE decreases by 1 order of magnitude over ∼15 epochs. Then, the MSE decreases by a second order of magnitude over ∼40 epochs. Finally, the MSE is further divided by 3 over the following ∼550 epochs. The training was stopped after 600 epochs as the decrease in MSE was becoming very slow. Note that in the case of experimental data, the final MSE was equal to $\sim 1 \times 10^{-2}$ mA^2 while here the final MSE was equal to $\sim 3 \times 10^{-2}$ mA^{-2}. One can note that despite the larger dataset (∼5 times more data than the experimental dataset) and the absence of measurement errors the MSEs of the predictions of the trained networks have very close values.

Similar to the network trained on the experimental dataset, the predictions of the model have to be inspected to verify its performance. To do so, the second simulated dataset was used. Indeed, the first simulated dataset was constituted from randomly sampled configurations while the second simulated dataset was constituted from simulated scans of the LEBT model. Hence, it is possible to compare a simulated scan to its predictions in a similar way to the case of experimental data.

Such a comparison is given in Figure 5.9. At first glance, the general behavior is reproduced which is satisfactory. However, there are some clear differences between the simulated scan and the predicted scan. In particular, the network predictions do not reproduce correctly the effect of the collimator on the beam (for $I_1^{sol} < 65$ A). This becomes even more apparent when looking at the difference between the simulated and

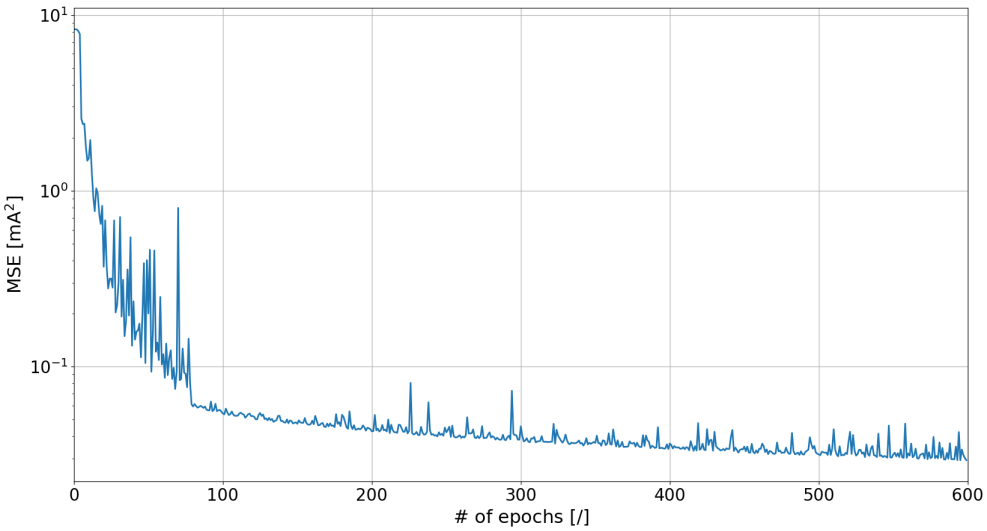

Figure 5.8: Typical training curve on the MYRRHA LEBT simulated dataset.

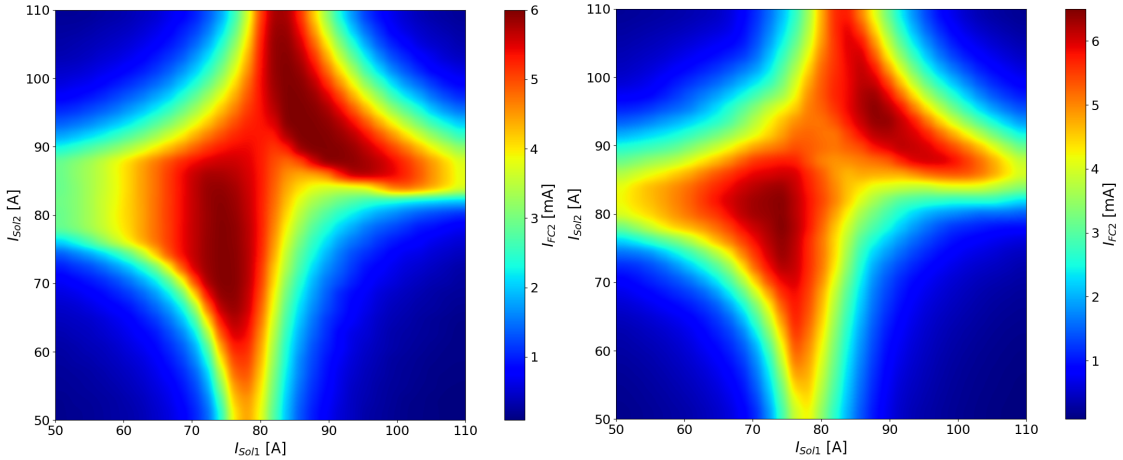

Figure 5.9: Beam current at the exit of the LEBT during a solenoids scan with $r_{col} = 35$ mm. Left: simulated scan. Right: scan reproduced with the best performing network at 600 epochs

predicted scan (cf. Figure 5.10). Here, it appears clearly that the network predictions are very good almost everywhere except for the regions where the beam current changes abruptly. In these regions, the error on the predictions can reach as high as 1.3 mA which is about twice the highest errors in the case of the experimental data.

This behavior can be observed over the whole training range as highlighted by the comparison between the simulated and predicted beam current with respect to the collimator position (cf. Figure 5.11). Compared to the case of the experimental data, there are fewer fluctuations on the plateau and the overshoot seems to have mostly disappeared which are two things that were expected with the increase in the size of the dataset. However, for the experimental dataset, the predictions mostly followed

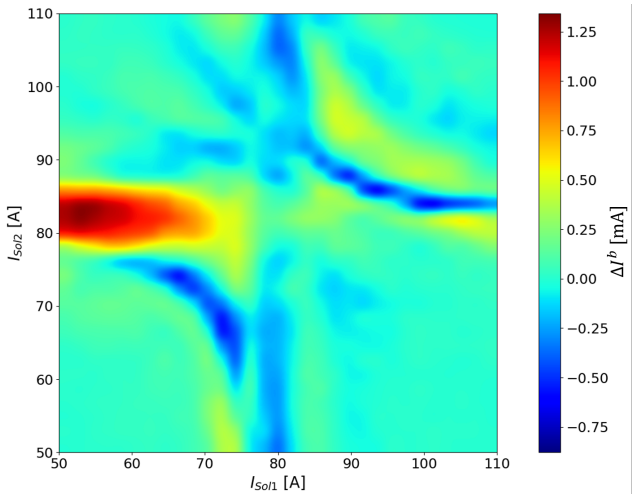

Figure 5.10: Difference between the simulated and predicted beam current at the exit of the LEBT during a solenoids scan with $r_{col} = 35$ mm.

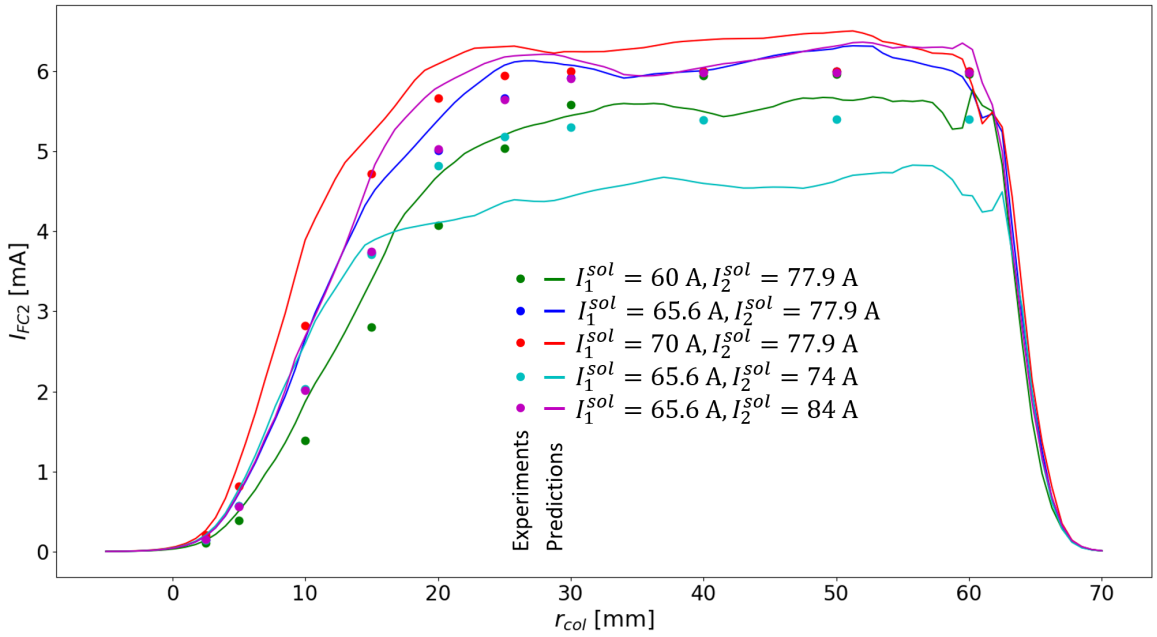

Figure 5.11: Comparison of simulations (dots) and predictions (lines) for five configurations close to the design configuration ($I_1^{sol} = 65.6$ A and $I_2^{sol} = 77.9$ A) with respect to the collimator opening.

the experimental measurements while for the simulated dataset the predictions are systematically overestimation or underestimation.

This effect is systematic and is caused by the presence of abrupt changes of the beam current with respect to the configuration in the simulated dataset that are not as present in the experimental dataset. In other words, the predictions of the trained model do not change sharply enough with respect to the configuration when

the simulated beam current changes abruptly between two nearby configurations. To illustrate this, the simulated and predicted beam current are put in comparison in Figure 5.12 for three nearby setpoints on the second solenoid with respect to the first solenoid setpoint when the collimator is fully open. It shows that the predictions are quite good for $I_1^{sol} < 60$ A when $I_2^{sol} = 76$ A and 80 A but not for the transition between these two setpoints.

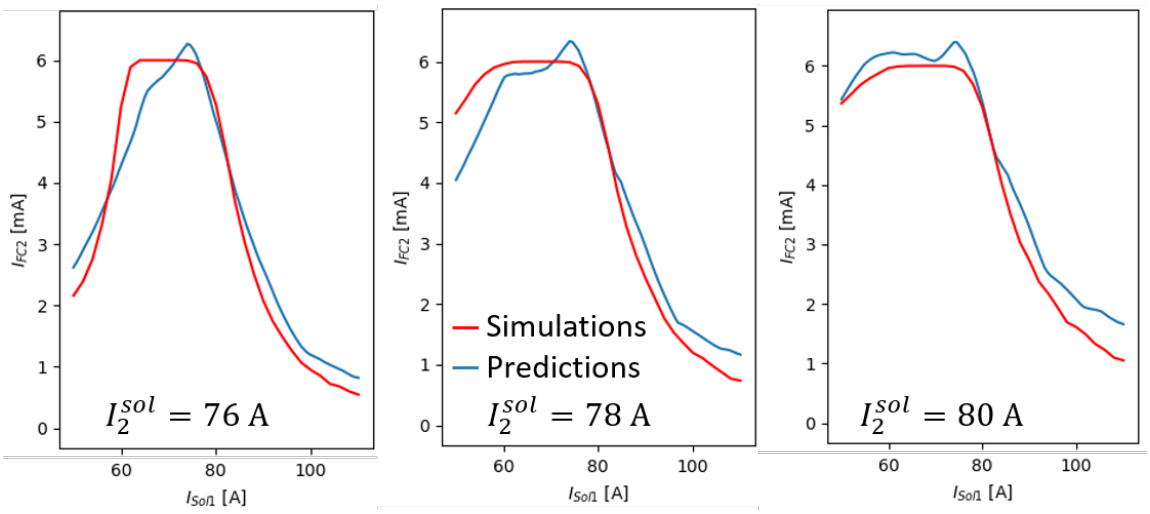

Figure 5.12: Evolution of the simulated (red line) and predicted (blue line) beam current at the exit of the LEBT with respect to I_1^{sol} with the collimator fully open and $I_2^{sol} = 76$, 78 and 80 A.

There are two potential ways to mitigate this issue: adapting the loss function used to train networks or increasing the size of the networks.

Indeed, changing the loss function to incorporate a term that penalizes the difference between the gradients predicted by the trained model and the gradients present in the dataset can potentially help reduce the issue.

Also, with a larger number of neurons available, a network should be able to reproduce more details. However, the training of neural networks with 256 neurons per hidden layer showed no real improvements in our case. The other option is to increase the number of hidden layers but the attempts at training a network with 4 hidden layers failed with the algorithm used in this thesis (cf. Appendix A.1.3). The best results obtained with a network with 4 hidden layers was a MSE on the predictions of $\sim$4 mA^2 which is comparable to the results described for the training of the initial neural network structure on the experimental dataset (cf. Figure 5.2).

Overall, this issue is not that concerning considering that the main reason to train networks on the simulated dataset is not to model the simulation code perfectly. Indeed, the simulated dataset was used as an alternative to artificially increase data for the experimental dataset. This choice was based on the fact that the initial conditions used in the TraceWin model are based on experimental measurements. Of course,

there is no certainty that these conditions perfectly reproduce the real beam characteristics but it does reproduce the trends. Hence, it was deemed more favorable than using a simple linear model. The remaining question is if a network trained on the simulated dataset will keep the advantages from the higher number of training data after being retrained using the experimental dataset. This aspect is discussed in the next section.

5.1.3 Network training using simulated then experimental data

Here, the model obtained after training a network for 600 epochs on the randomly sampled dataset obtained with the TraceWin model (cf. Section 5.1.2) was trained for a second time on the experimental dataset. The training curve is shown in Figure 5.13. At epoch #600, the MSE shows an increase of about one order of magnitude because the model was then trained and tested on the experimental dataset. Then the MSE steadily decreases with the progression of the training. The retraining was stopped after 900 epochs when the MSE dropped down to $\sim 2.3 \times 10^{-2}$ mA^2.

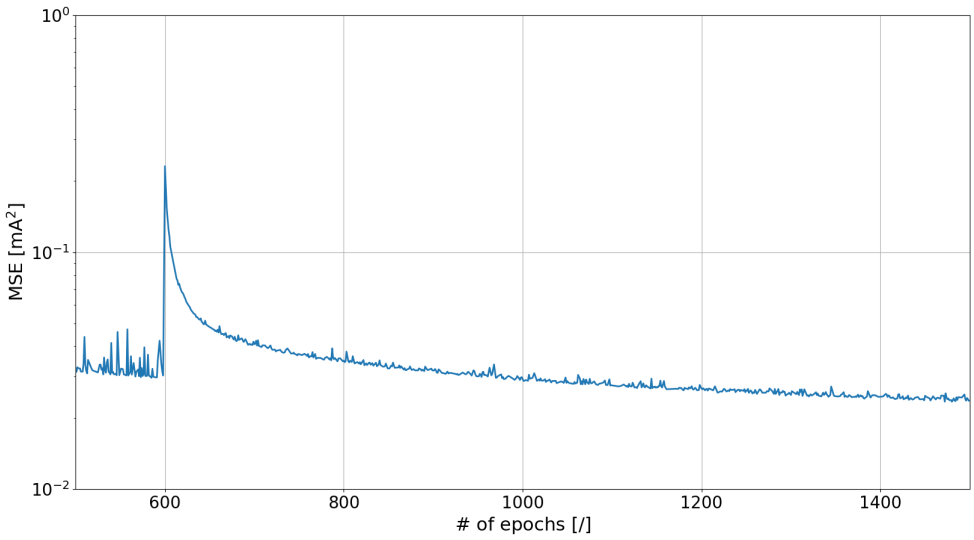

Figure 5.13: Training curve of the model trained first on the MYRRHA simulated dataset then on the experimental dataset.

The inspection of the predictions shows that the predictions of the retrained model reproduce the experimental data (cf. Figure 5.14). From this comparison, the results from the retrained network are quite similar to the predictions of the network trained only on experimental data. In addition, the absolute error made by the predictions of the retrained model is within ±0.5 mA (cf. Figure 5.15) as for the pure experimental network.

The main difference between the retrained model and the pure experimental model is shown in Figure 5.16. While the predictions of the pure experimental model present many fluctuations of the beam current with respect to the collimator position, the

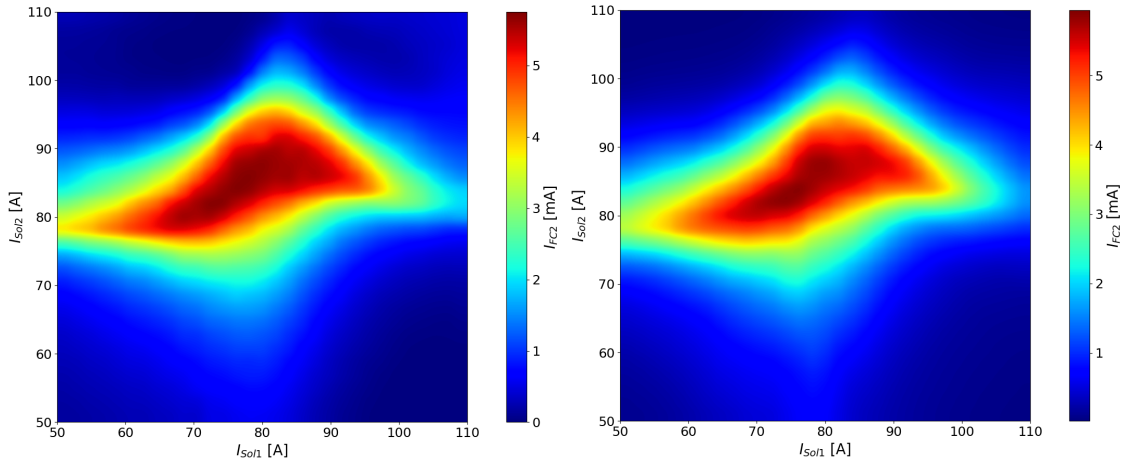

Figure 5.14: Beam current at the exit of the LEBT during a solenoids scan with $r_{col} = 55$ mm. Left: experimental scan. Right: scan reproduced with the retrained network at epoch #1500.

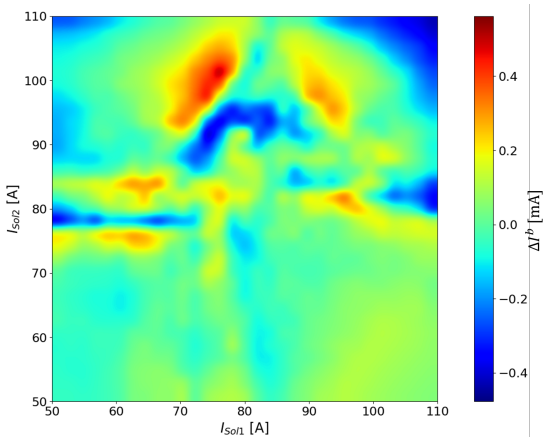

Figure 5.15: Difference in mA between the simulated and predicted beam current at the exit of the LEBT during a solenoids scan with $r_{col} = 35$ mm.

predictions of the retrained model are much more stable. Moreover, the problem of the overshoot has also been resolved for the retrained model. This means that even if the simulated dataset does not reproduce perfectly the real machine, it can be used to significantly improve the quality of the final network model. However, the retrained is still not perfect. The beam current predicted appears to be systematically underestimated for $r_{col} > 20$ mm except for the turquoise curve ($I_1^{sol} = 65.6$ A and $I_2^{sol} = 74$ A). Actually, according to Figure 5.15, modeling the gradients present in the training dataset is still not done correctly.

Fortunately, the range over which the model is most accurate corresponds to the configuration space we are most interested in. Indeed, the LEBT is operated to output a beam current of ∼4.2 mA. This means that the LEBT will usually be operated with $r_{col} < 20$ mm. However, the model should be more exhaustively compared to the real

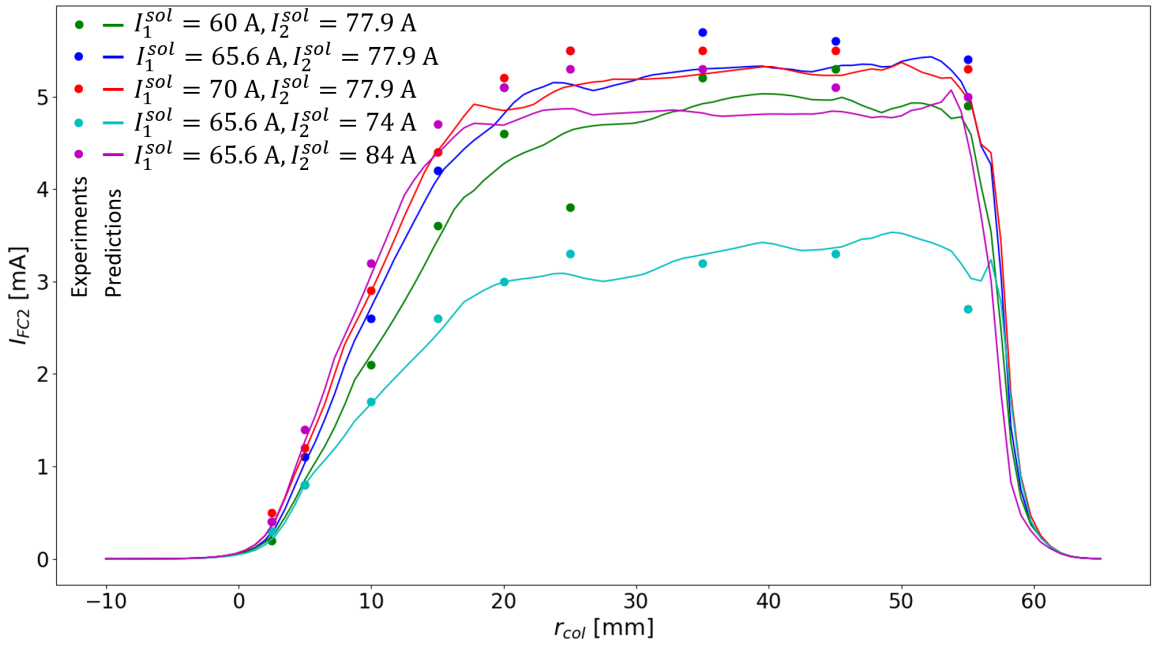

Figure 5.16: Comparison of simulations (dots) and predictions (lines) for five configurations close to the design configuration (I_1^{sol} = 65.6 A and I_2^{sol} = 77.9 A) with respect to the collimator opening.

machine before being put to practical use.

As a reminder, the role of the LEBT is to guide and focus the beam from the ion source to the RFQ. In this sense, the quality of the LEBT configuration is evaluated from two quantities: the beam current at the RFQ output and the beam current transmission through the RFQ. Hence, a network has to be trained to model these quantities with regard to the LEBT configurations to be of practical use. In the case of MYRRHA, the RFQ was not available when the experimental dataset was constituted. Hence, in preparation for the MYRRHA injector with the LEBT and RFQ, networks were trained to model the IPHI injector that consists of a LEBT and a RFQ. The results of these trainings are discussed in Section 5.2.1.

But before that, Section 5.1.4 is dedicated to the comparison between a test of the PSO algorithm using the model trained here and the online PSO test performed on the MYRRHA LEBT (cf. Section 3.1.4).

5.1.4 Application of PSO on the trained model

The model trained on the MYRRHA datasets can be used as a surrogate to the real machine. Here, a PSO algorithm was used to provide candidate configurations to obtain a target beam current at the exit of the LEBT. This test is performed to illustrate a strategy to configure the LEBT:

1. Apply the PSO algorithm to the trained model to find a candidate configuration of the LEBT that would provide a given set of beam parameters.

2. Apply the candidate configuration on the LEBT.

3. Apply the PSO algorithm to the LEBT to refine its configuration.

With this strategy, the process to find an optimized configuration can be greatly shortened.

Similarly to the online test of the PSO algorithm (cf. Section 3.1.4), the algorithm was applied to the trained model to optimize the configuration of the LEBT. As a reminder, the algorithm optimized the two following cost functions:

$$O_1(I^b_{LEBT,out}) = (I^b_{LEBT,out} - I^{b*}_{LEBT,out})^2 = (I^b_{LEBT,out} - 2.5)^2, \tag{5.1}$$

and

$$O_2(I^{sol}_1, I^{sol}_2, I^{st}_{2H}, I^{st}_{2V}, r_{col}) = (I^{sol}_1 - 65.6)^2 + (I^{sol}_2 - 77.9)^2 + (I^{st}_{2H})^2 + (I^{st}_{2V})^2 - x_{col} \tag{5.2}$$

where $x_{col} = 55 - r_{col}$. In Equation 5.2, x_{col} is used instead of r_{col} because the setpoint used in the control system for the collimator corresponds to the collimator extension into the chamber i.e. when the setpoint is equal to 0 mm the collimator is fully opened and when it is equal to 55 mm the beam is completely intercepted.

The first objectives allows to define a target current in the second Faraday cup $I^b_{FC_2} = I^b_{LEBT,out}$. The second one favors configurations closer to the design configuration ($I^{sol}_1 = 65.6$ A, $I^{sol}_2 = 77.9$ A, $I^{st}_{2H} = 0$ A and $I^{st}_{2V} = 0$ A) and smaller collimator opening as to intercept a maximum of the halo.

As for the online test, the algorithm was configured to run for 10 iterations with a population of size 25 and a target beam current of 4.2 mA. With these parameters, the algorithm was able to suggest 3 potential configurations of the LEBT in the vicinity of the design configuration that would provide around 4.2 mA (cf. Figure 5.17). As a reminder, the online test of the PSO algorithm suggested a configuration with $I^{sol}_1 = 66.1$ A, $I^{sol}_2 = 78.1$ A and $r_{col} = 14.85$ mm.

Other target beam currents were tested with similar results. Also, increasing the number of iteration increases the number of candidate solutions with similar characteristics as well as the execution time of the algorithm.

The whole optimization process took about 10.5 seconds with about 2.5 seconds used for the evaluation of the candidate solution with the trained model. So about 8 seconds were actually used by the PSO algorithm making it the slowest part of the process contrary to the online test where the slowest part was the evaluation of the candidate solution. This means that the PSO algorithm should be optimized to decrease the overall execution time. Also, here the candidate solutions were evaluated in series with an evaluation time of less than 0.01 second per. This process can be parallelized to reduce the time spent on the evaluation part of the process. Indeed, the average execution time was measured over 100 executions[4] of the model for a given

[4]The test was performed on a laptop with an Intel i7-7820HQ CPU @ 2.9 GHz.

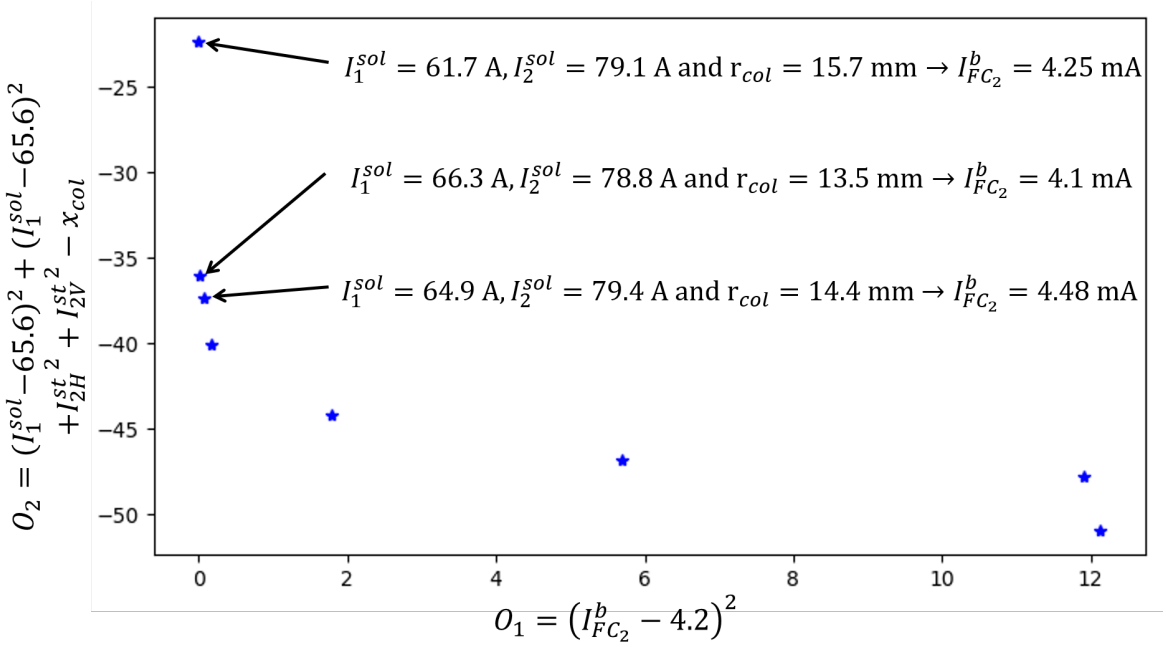

Figure 5.17: Result of the PSO applied on the network model retrained on the datasets of MYRRHA with $I_{LEBT,out}^{b*} = 2.5$ mA. Each star represents a candidate configuration of the LEBT.

number of configurations (cf. Figure 5.18). This shows that the execution time is essentially constant up to about 30 configurations then ramps up to a linear scaling with the number of configurations. This behavior indicates that at first the execution time is dominated by the memory overheads. Hence, below a few hundred configurations it is better to evaluate those in parallel.

This test shows that the trained model can be useful as a surrogate to the real machine. Of course, the predictions of the model are not perfectly accurate hence it is still necessary to apply the candidate solutions determined with the PSO algorithm to the real machine to test and tweak those. Nevertheless, the execution time of the trained model is very low which means that the application of the PSO algorithm on the model is a few hundred times faster than its application on the real machine and still has room for improvements. The optimization of the PSO algorithm would likely allow finding candidate configurations in less than 1 second.

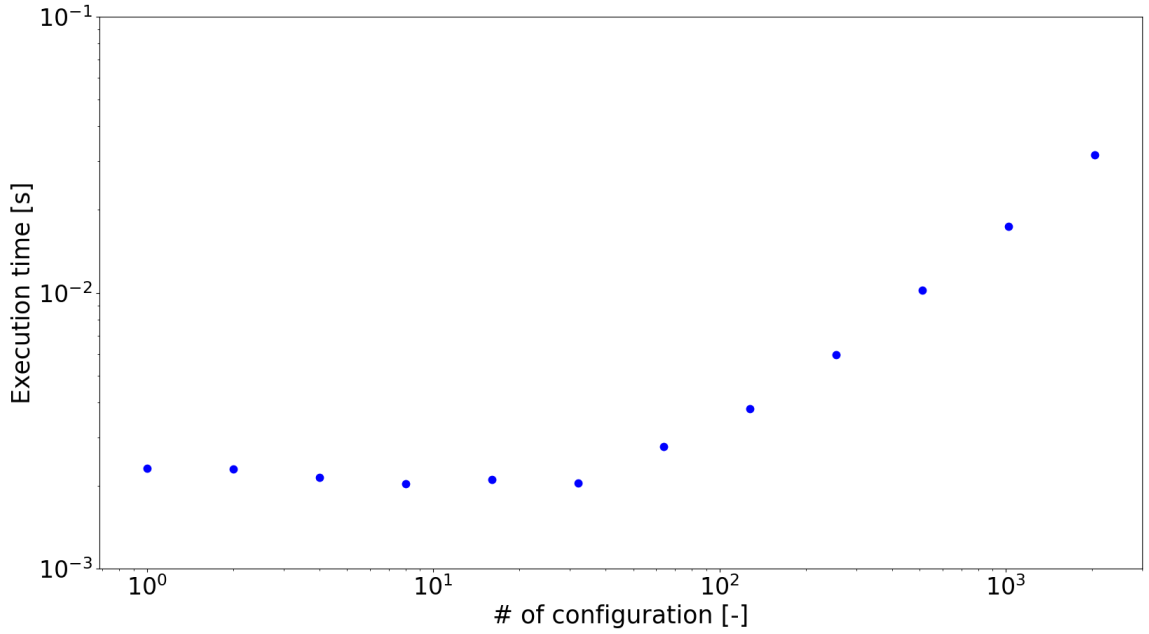

Figure 5.18: Average execution time of the trained model with respect to the number of configurations.

5.2 IPHI model

Here, the training of a neural network aimed to model the IPHI injector. The model is trained as a regression model that should reproduce the behavior of a LEBT and a RFQ. In particular, the model has to predict the beam current measured at the exit of the RFQ and the beam transmission through the RFQ given the configurations of the LEBT. The training is based on the experimental data collected during a series of experiments on IPHI (cf. Section 4.1.2.1).

Section 5.2.1 is thus dedicated to the description of the training of this model and the discussions of the results that were obtained.

5.2.1 Network training using the experimental dataset of IPHI

A model was trained using the experimental dataset of the IPHI injector (cf. Section 4.1.2). The structure of the network was adapted from the ones determined using cross-validation on the simulated dataset of the MYRRHA LEBT (cf. Section 4.2). The adaptation consisted of doubling the size of the output layer and reducing the size of the input layer to match the structure of the training dataset reminded in Table 5.5. The structure of the network is given in Table 5.4. This network was trained with a batch-size of 64 examples and an initial learning rate of 0.1.

Layer type	Input	Hidden	Hidden	Hidden	Output
# of neurons	3	96	96	96	2
Activation function	-	ReLU	ReLU	ReLU	sigmoid

Table 5.4: Initial neural network structure for the training on the IPHI experimental dataset.

	Name	Symbol
Inputs	First solenoid setpoint	I_1^{sol}
	Second solenoid setpoint	I_2^{sol}
	Iris position	r_{col}
Outputs	Beam current	$I^b_{RFQ,out}$
	Transmission of the RFQ	t_{RFQ}

Table 5.5: Inputs and output of the experimental dataset of the IPHI LEBT.

A typical training curve is shown in Figure 5.19. The MSE for the prediction of the RFQ transmission (orange line) decreases by an order of magnitude over the first ∼35 epochs then it is divided by 4 times over the next ∼150 epochs. After that, the MSE stagnates around 3 %^{2}. For the prediction of the beam current, the MSE decreases by an order of magnitude over ∼60 epochs. Then, it takes around 400 epochs to divide the MSE by a factor of 4. At epoch #1000, the MSE was down to ∼0.6 mA^2. From these values, it appears that the MSE decreases ∼4 times faster for the prediction of the beam current than for the prediction of the transmission.

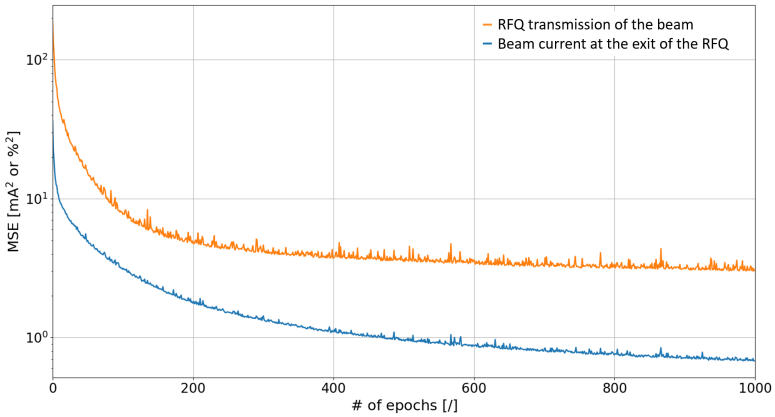

Figure 5.19: Training curves (blue for the beam current and orange for the transmission) of the model trained on the IPHI experimental dataset.

The inspection of the predictions shows that the trained model can simultaneously reproduce both quantities (cf. Figure 5.20) with relatively low errors (cf. Figure 5.21).

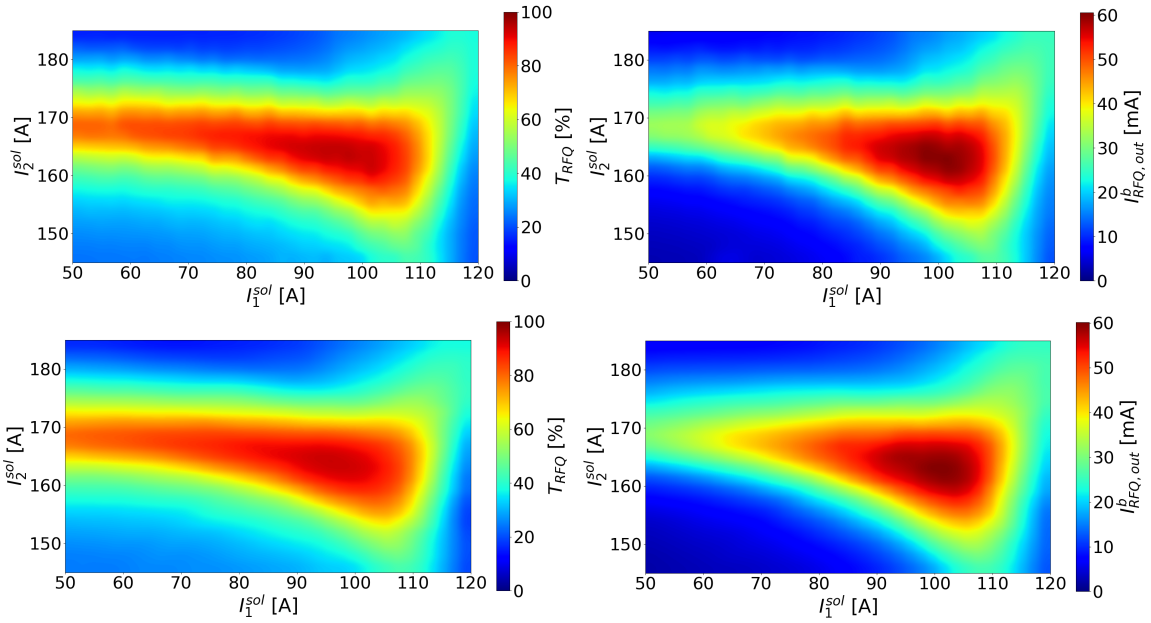

Figure 5.20: Comparison between the experimental data (top) and the model predictions (bottom) for the RFQ transmission (left) and the beam current measured at the exit of the RFQ (right) during a solenoids scan with $r_{col} = 130$ mm.

5.2.2 Application of PSO on the trained model

As for the model trained on the MYRRHA datasets, the model trained on the IPHI dataset can be used as a surrogate to the real machine. In particular, it is possible to apply an optimization algorithm to the trained model to determine a configuration

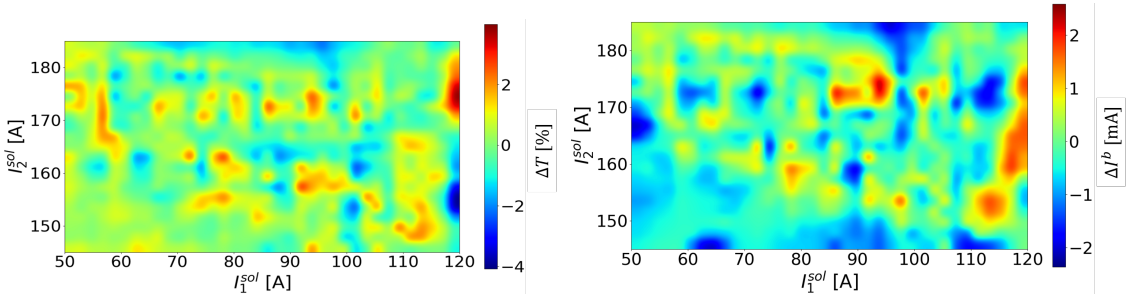

Figure 5.21: Difference between the experimental data and the model predictions for the RFQ transmission (left) and the beam current measured at the exit of the RFQ (right) during a solenoids scan with $r_{col} = 130$ mm.

suitable to obtain a given set of beam characteristics at the exit of the RFQ. To illustrate this, a PSO algorithm was applied to the trained model to optimize the configuration of the LEBT with the two following objectives.

- Maximize the transmission through the RFQ
  To avoid damaging the RFQ during its operation, the transmission should always be as high as possible. For the algorithm this objective is interpreted as minimizing the beam losses through the RFQ:

$$\min O_1 = \min(1 - t_{RFQ}(I_1^{sol}, I_2^{sol}, r_{col})). \tag{5.3}$$

- Obtain a target beam current at the exit of the RFQ
  This objective is quite straightforward and is interpreted by the algorithm as minimizing the square of the difference between the target beam current and the actual beam current:

$$\min O_2 = \min(I_{RFQ,out}^{b*} - I_{RFQ,out}^{b}(I_1^{sol}, I_2^{sol}, r_{col}))^2. \tag{5.4}$$

The algorithm was configured to run for 10 iterations with a population of size 15 and a target beam current of 25 mA. With these choices, the algorithm was able to suggest 3 potential configurations of the LEBT that would provide a beam current between 24 and 27 mA with transmission higher than 96 % (cf. Figure 5.22). Several other target beam currents were tested with similar results. Also, increasing the size of the population or the number of iterations did not really improve the results of the optimization but increased its execution time and the number of suggested configurations.

While this optimization took about 9 seconds the computation of one iteration took within 0.07 and 0.1 s. So, about 8 seconds of the optimization process is spent in the PSO code rather than in the evaluation of the candidate configurations. Hence, this corroborates the fact found in Section 5.1.4 that with a trained model as a surrogate to the real machine, the PSO algorithm becomes the slowest element and should be optimized for faster execution times.

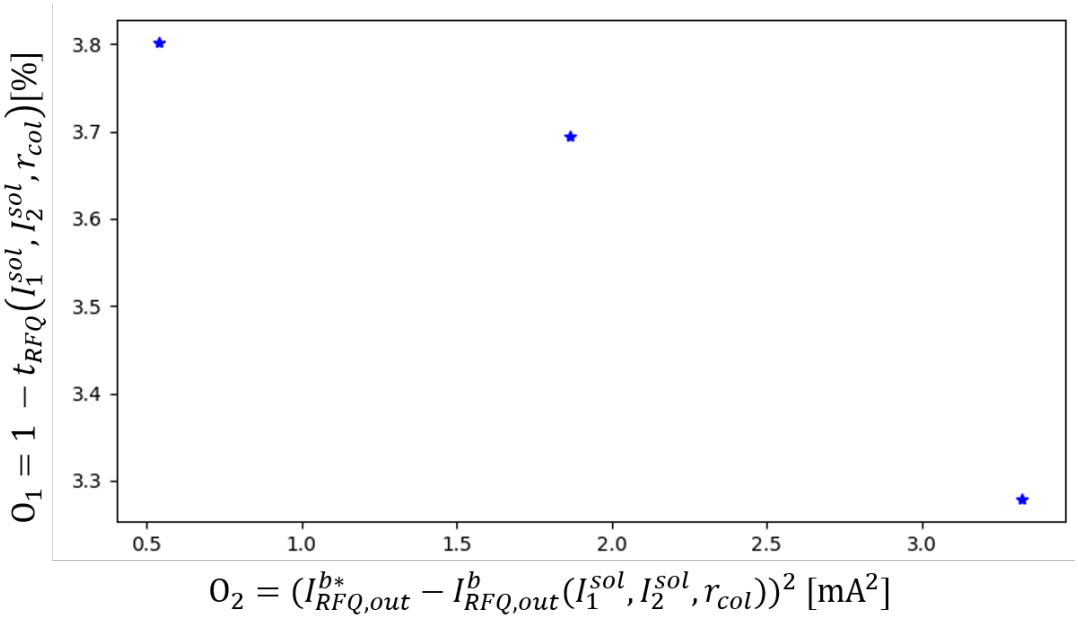

Figure 5.22: Result of the PSO applied on the network model trained on the experimental dataset of IPHI with $I_{RFQ,out}^{b*} = 25$ mA. Each star represents a candidate configuration of the LEBT.

Overall, this test is successful. Indeed, it shows that from a limited number of experimental measurements [5] it is possible to train a model that can be used to suggest candidate configurations for a wide range of target beam currents. Of course, the model does not reproduce the real machine perfectly so the suggested configurations should be adjusted to actually get the targeted beam properties. But, the suggested configurations are a good starting point. In addition, this test demonstrates that the PSO algorithm is suitable to suggest candidate configurations for an injector with both a LEBT and a RFQ. Ideally, this algorithm should be tested on the real machine as for the online test performed on the LEBT of MYRRHA (cf. Section 3.1.4).

[5] About 10 000 in this case

5.3 Summary

In this chapter, the training of several networks was discussed.

First, the training of a network on the experimental dataset of the MYRRHA LEBT showed that a network could be trained to reproduce the behavior of the real machine. Indeed, the model was able to predict the beam current measured at the exit of the LEBT over the configurations that were measured during the commissioning of the LEBT. However, a few problems were raised: an overshoot was observed for configurations with the collimator almost fully open, large fluctuations were also observed for positions of the collimator that were not sampled in the experimental dataset.

Then, the training of a network on the randomly sampled simulated dataset of the MYRRHA LEBT showed that the predictions systematically over- or underestimated the simulated beam current. It leads to discuss the fact that the network was not reproducing correctly some gradients present in the simulated dataset. However, this training showed that the increase in the size of the dataset allowed to reduce the fluctuations and the overshoot presented by the model trained on the experimental dataset.

The next training discussed was actually retraining of the model previously trained on the simulated dataset using the experimental dataset. Despite the difference between the TraceWin model of the LEBT and the real LEBT, the predictions of the retrained network were improved compared to the network trained only on the experimental dataset. In particular, the predictions of the retrained model showed fewer fluctuations for the collimator positions not sampled in the experimental dataset. The overshoot was also strongly mitigated.

Finally, the training of a network on the experimental dataset for the IPHI injector has been discussed. In this case, the network had to predict two quantities instead of a single one: the transmission of the RFQ and the beam current measured at the exit of the RFQ. Here too, the network was shown to predict with relatively good accuracy the experimental behavior of the injector.

In addition, the PSO algorithm was tested on both models. In the case of the model trained on the MYRRHA datasets, the algorithm had to suggest candidate configurations for the LEBT to obtain a target beam current at the exit of the LEBT in the vicinity of the design configuration. For the model trained on the IPHI dataset, it also had to suggest candidate configurations for the LEBT but here it was to obtain a target beam current at the exit of the RFQ while maximizing the transmission. In both cases, the trained models were used as surrogates for their respective real machines and the PSO algorithm performed quite well. With only 10 iterations, a few candidate configurations were suggested by the algorithm with an execution time of about 10 seconds in total.

Conclusions and perspectives

As exposed in Chapter 1, many applications are requesting ever more efficient particle accelerators. This means that there is pressure to increase the mean power up to megawatts. With this comes the need to improve the reliability and thus the beam availability. This is particularly true for ADS projects such as MYRRHA. Indeed, it is crucial to meet a level of reliability never achieved before on high-power accelerators. To meet this requirement, it is important to keep the beam losses along the accelerator lower than 1 W/m. It is therefore necessary to ensure the quality of the beam to be accelerated by optimizing the configuration of the injector. This manuscript presents the research work undertaken to explore the possibility of developing new tools based on neural networks to help in this endeavor. In particular for the tuning of SC linacs injectors.

The experimental work performed to tune two proton beam injectors, MYRRHA and IPHI, was described in Chapter 3. During one of the experiments on the MYRRHA injector, the PSO algorithm was successfully tested as an online optimization algorithm. Also, the experimental datasets constituted for both machines were used to evaluate the fidelity of the reference simulation using their respective TraceWin models. In both cases, the models provided reasonable results but with apparent discrepancies with the experimental data.

The constitution of the datasets to train neural networks to model the MYRRHA and IPHI injectors was presented in Chapter 4. In addition, an initial network structure and the hyperparameters for the training are determined using a cross-validation approach.

In the final chapter, the training of neural networks has been discussed for both injectors.

In the case of the MYRRA LEBT, networks were trained to predict the beam current that can be measured at the exit of the LEBT given its configuration. The best performing network was first trained on the randomly sampled simulated data then retrained on the experimental dataset. It showed that the small size of the experimental dataset could be compensated with simulations even though the TraceWin model presents some discrepancies with the real machine.

For IPHI, networks were trained to predict the beam current that can be measured at the exit of the RFQ and the transmission of the RFQ given the configuration of

the LEBT. The best performing network was shown to reproduce both quantities simultaneously

In addition, the best trained models for both injectors were used as surrogates for their respective machines with a PSO algorithm. With the trained model of the MYRRHA injector, the algorithm was able to suggest candidate configurations in the vicinity of the design configurations for a variety of target beam currents. With the trained model of IPHI, the algorithm was able to suggest potential configurations that matched a variety of beam currents while maximizing the transmission. In both cases, the optimization process was very quick with the slowest part being the PSO algorithm itself while the evaluation of the potential configurations took about 10 % of the total execution time. At that point, these performances could still be improved by parallelizing the trained model evaluations and optimizing the PSO algorithm.

Perspectives

Overall, the use of neural networks to help with the tuning of an injector is a promising option. It provides a nonlinear model of the experimental behavior of the machine with a very low execution time. Of course, the results presented in this manuscript are just the first steps towards many more developments. Here are some examples of perspectives for the development of tools based on neural networks.

Improving the fidelity of the trained models

The final model of the MYRRHA injector showed a systematic underestimation of the beam current for $r_{col} > 20$ mm. To improve the predictions of the trained model other training strategies could be tested. For example, the number of hidden layers in the neural network can be increased with the following strategy (cf. Figure A):

1. Train a first network with n_1 hidden layers (usually 2 or 3).
2. Build a second network with $n_2 > n_1$ hidden layers where the first n_1 layers use the weight and biases determined in the previous step.
3. Train the second network while keeping the first n_1 layers frozen (ignore the weights and biases of these layers during the training).

In a sense, this approach corresponds to training a network with another network as input. Hopefully, with this method, the additional layers are trained to add nuances to the initial network. This could prove to be useful to model the more abrupt variations of the beam current present in some regions of the configuration space. In theory, networks with more layers should have a higher capacity to model a system.

Predicting injectors configurations with a network

On another note, the PSO algorithm was successfully used both online with the real machine and with the trained models without prior knowledge of the problem. Although this makes the algorithm very practical, it is still necessary to test many configurations to find the final candidates. Training a network with the reinforced

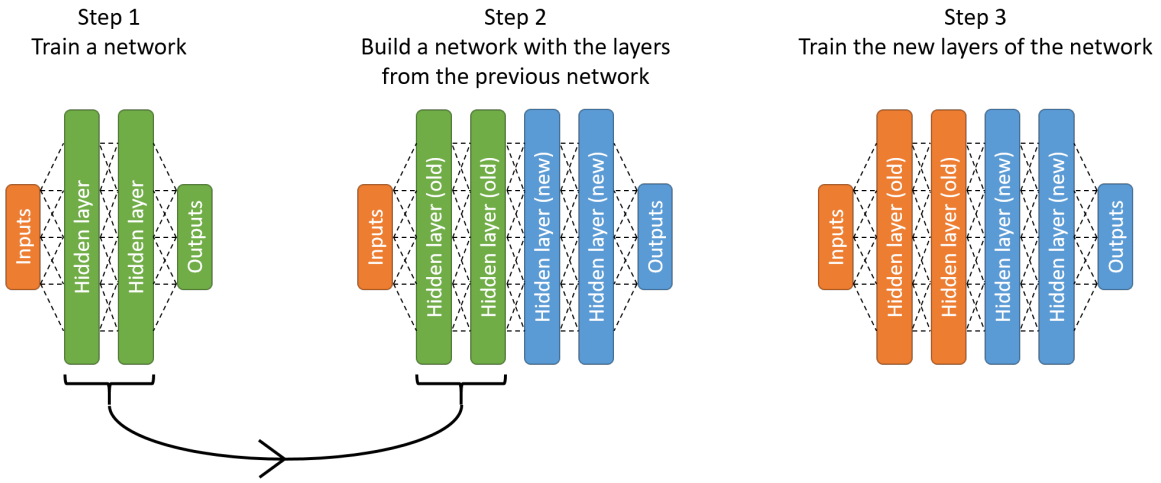

Figure A: Strategy to increase the number of hidden layers in a neural network.

learning paradigm using the trained model as the learning environment could potentially lead to the creation of an algorithm much more adapted to the optimization of an injector. The idea is to train a predictor network whose task would be to suggest a suitable configuration of the injector to obtain given beam properties. For such training, the predictor network has to interact many times with the injector to learn which configuration provides which beam properties. Hence, the use of the trained model as a surrogate to the real machine would speed up significantly the training of a predictor network. Indeed, the predictor can interact with the trained model hundreds of times per second while it can interact with the injector only once every few seconds. Furthermore, with the trained model, the training of the predictor can be parallelized.

Essentially the following two main types of predictor can be considered:

- One-shot predictors
  In this case, the predictor is tasked to suggest a candidate configuration in a single iteration. This configuration would then be applied and refined through other systems on the machine.

- Multi-shot predictors
  Here, the task of the predictor is to suggest a first candidate configuration to be tested on the machine. Then, using the obtained beam properties, the predictor would suggest an improved candidate configuration to be tested on the machine. This process can be repeated until the desired beam properties are obtained until the maximum number of iterations is reached or until any other relevant stopping criterion is met.

The multi-shot predictor concept can be pushed even further. If plugged into the control systems of a machine and allowed to run continuously, a multi-shot predictor could be used as an automatic controller for an injector i.e. a "smart" control loop.

Future experimental work on the MYRRHA injector

The RFQ has been coupled to the LEBT in the middle of 2020 and is now operational [115]. Hence, following the results on the IPHI injector, a new experimental dataset of the MYRRHA injector should be constituted to train a network to predict the beam properties at the exit of the RFQ given the LEBT configuration. This would then allow the application of the PSO algorithm on the trained model to determine a candidate configuration to obtain desired beam properties. This configuration would then be applied to the machine for further online optimization by the algorithm.

Modelization of the MYRRHA RFQ with a network

Another potential application is to train a network to model the transport of the distribution of a beam through the MYRRHA RFQ. Indeed, to simulate correctly the dynamics of the beam in a RFQ such as with the Toutatis simulation code [116], it is necessary to track every particle of the beam. This type of computation takes a fair amount of time. Hence, a network could be trained to predict the distribution of the beam at the exit of the RFQ given the beam distribution at the entrance of the RFQ and the configuration of the machine. This would lead to a significant speedup for start-to-end simulations which could facilitate the computation of fault compensated configurations.

Fault compensation algorithm for the MYRRHA SC linac

While this thesis is focused on the tuning of the injector, the fault-tolerant strategy planned for the SC linac described in Section 1.4.2 may also benefit from the use of neural networks. As a reminder, the idea is that if a cavity of the SC linac fails then the neighboring cavities would be retuned to compensate for the failure (cf. Figure B). The current solution is that the compensated configurations have to be computed in advance with a dedicated algorithm then fine tuned using the model of the linac [115]. This process may benefit from a network trained to model the SC linac. One possibility is to train a network to model the behavior of a cavity so it can be used to simulate the compensated configuration. Another one is to try to use a PSO algorithm to optimize the convergence criteria of the dedicated algorithm.

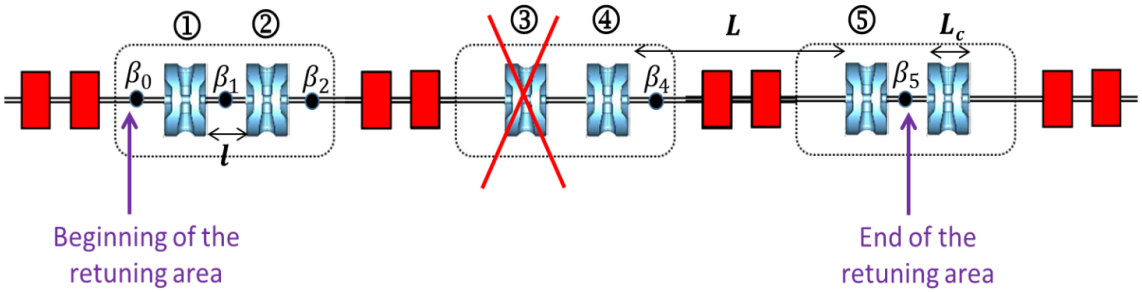

Figure B: Principle of the fault compensation in the case of a SC cavity failure. [115]

To conclude, the perspectives described here aim at the development of a suite of

new tools to help configure the MYRRHA injector and SC linac. However, these are not the only potentially useful applications of machine learning in the field of particle accelerators. Indeed, although improvements are still needed before the technology becomes fully mature, neural networks and machine learning are powerful and adaptable contraptions that can be used to develop ad hoc tools where needed. The accelerator community is aware of this potential as has been demonstrated in Section 2.2 as well as by this thesis work to improve the tuning of an injector (decrease the time required to obtain a beam of good quality). Eventually, these new developments will improve the reliability of high-power linacs.

Bibliography

[1] Massachusetts Institute of Technology. *Energy, Environment, Cancer*. URL: https://libraries.mit.edu/mithistory/impact/energy-environment-and-cancer/.

[2] Rolf Wideröe. "Über ein neues Prinzip zur Herstellung hoher Spannungen". In: *Arbeiten aus dem Elektrotechnischen Institut der Technischen Hochschule Aachen: Band III: 1928*. Ed. by W. Rogowski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1929, pp. 157–176. ISBN: 978-3-662-40440-9. DOI: 10.1007/978-3-662-40440-9_14. URL: https://doi.org/10.1007/978-3-662-40440-9_14.

[3] M. Seidel. *Cyclotrons, in CAS - CERN Accelerator School: Introduction to Accelerator Physics: Constanta, Romania 25 September*. 2018.

[4] Unknown author. "Pioneer Atom Splitter". In: *Radio-Craft* 18 (9 June 1947).

[5] *The Soleil website*. URL: https://www.synchrotron-soleil.fr/en.

[6] *The LHC webpage*. URL: https://home.cern/science/accelerators/large-hadron-collider.

[7] G. Hoffstaetter. *Livingston Plot for hadron Accelerators*. Cornell University.

[8] T. Satogata, C. Hernalsteens, and R. Gamage. *Introductions, Relativity, E&M, Accelerator Overview*. URL: https://toddsatogata.net/2017-USPAS/Lectures/2017-01-16-Uspas-Lecture1.pdf.

[9] J. W. G. Thomason et al. "Proton driver scenarios at CERN and Rutherford Appleton Laboratory". In: *Phys. Rev. ST Accel. Beams* 16 (5 May 2013), p. 054801. DOI: 10.1103/PhysRevSTAB.16.054801. URL: https://link.aps.org/doi/10.1103/PhysRevSTAB.16.054801.

[10] Giulia Bellodi et al. "Linac4 Commissioning Status and Challenges to Nominal Operation". In: *HB2018, Daejeon, Korea, paper MOA1PL03, pp. 14–19*. 2018. DOI: 10.18429/JACoW-HB2018-MOA1PL03.

[11] Lucio Rossi, Oliver Brüning, et al. "Progress with the High Luminosity LHC Project at CERN". In: *10th Int. Partile Accelerator Conf.(IPAC'19), Melbourne, Australia, 19-24 May 2019*. JACOW Publishing, Geneva, Switzerland. 2019, pp. 17–22. DOI: 10.18429/JACoW-IPAC2019-MOYPLM3.

[12] J.-L. Biarrotte. "High power proton/deuteron accelerators". In: *Proc. 16th Int. Conf. on RF superconductivity, Paris, France*. 2013.

[13] F. Bouly. “MYRRHA Superconducting linac and Fault Tolerance Concept”. In: *30th International Linear Accelerator Conference, Virtual conference*. 2020. URL: http://linac2020.org/.

[14] M. Seidel. “Cyclotrons for high-intensity beams, in CAS - CERN Accelerator School: High Power Hadron Machines: Bilbao, Spain 24 May”. In: CERN, Aug. 2013. DOI: 10.5170/CERN-2013-001. URL: https://cds.cern.ch/record/1312630.

[15] E. Bargalló et al. “ESS Availability and Reliability Approach”. In: *Proc. 6th International Particle Accelerator Conference (IPAC'15), Richmond, VA, USA, May 3-8, 2015*, pp. 1033–1035. DOI: 10.18429/JACoW-IPAC2015-MOPTY045. URL: http://jacow.org/ipac2015/papers/mopty045.pdf.

[16] D. Vandeplassche and L. Medeiros-Romao. “Accelerator Driven Systems”. In: *Proceedings of IPAC2012, New Orleans, Louisiana, USA*. May 2012. URL: https://accelconf.web.cern.ch/IPAC2012/papers/moyap01.pdf.

[17] J. Galambos. “Operations Experience of SNS at 1.4MW and Upgrade Plans for Doubling the Beam Power”. In: *Proc. 10th International Particle Accelerator Conference (IPAC'19), Melbourne, Australia, 19-24 May 2019*, pp. 4380–4384. DOI: 10.18429/JACoW-IPAC2019-FRXPLM1. URL: http://jacow.org/ipac2019/papers/frxplm1.pdf.

[18] G. Rimpault et al. “The Issue of Accelerator Beam Trips for Efficient ADS Operation”. In: *Nuclear Technology* 184.2 (Mar. 2017), pp. 249–260. DOI: 10.13182/NT12-75. URL: https://hal.archives-ouvertes.fr/hal-02865131.

[19] F. Bouly. “Proton linacs as ADS drivers”. In: *EUCARD2: Status of accelerator driven systems research and technology development, Meyrin, CERN, Switzerland*. 2017.

[20] Thuillier. T. *JUAS - Particle Sources*. URL: https://indico.cern.ch/event/356897/contributions/1769000/attachments/709946/974547/particle_source_lecture_2015.pdf.

[21] I. M. Kapchinskii and V. A. Teplyakov. “LINEAR ION ACCELERATOR WITH SPATIALLY HOMOGENEOUS STRONG FOCUSING.” In: *Instrum. Exp. Tech. (USSR) (Engl. Transl.) No. 2, 322-6(Mar-Apr 1970)*. (1970).

[22] K. R. Crandall, R. H. Stokes, and T. P. Wangler. “RF quadrupole beam dynamics design studies”. In: *Proc. LINAC'79, Montauk, NY, USA. Sep. 1979*. Sept. 1979. URL: https://accelconf.web.cern.ch/l79/papers/s4-1.pdf.

[23] M. Vretenar. “The radio-frequency quadrupole”. In: (Mar. 2013). DOI: 10.5170/CERN-2013-001.207. URL: https://cds.cern.ch/record/1536736.

[24] N. Chauvin. *IPHI, a high intensity proton accelerator for neutron production*. 2019.

[25] Ph. Fischer and A. Schempp. “Tuning of a 4-rod CW-Mode RFQ Accelerator”. In: *Proceedings of EPAC 2006, Edinburgh, Scotland*. June 2006. URL: https://accelconf.web.cern.ch/e06/PAPERS/TUPLS040.PDF.

[26] M. Vretenar. "Linear accelerators". In: (Mar. 2013). DOI: 10.5170/CERN-2013-001.207. URL: https://cds.cern.ch/record/1536736.

[27] Thomas P Wangler. *RF Linear accelerators*. John Wiley & Sons, 2008.

[28] N.F. Petry et al. "Test of a High Power Room Temperature CH DTL Cavity". In: *Proc. of International Particle Accelerator Conference (IPAC'17), Copenhagen, Denmark,* May 2017, pp. 2237–2239. ISBN: 978-3-95450-182-3. DOI: JACoW-IPAC2017-TUPVA069. URL: http://jacow.org/ipac2017/papers/tupva069.pdf.

[29] C. Plostinar. "Comparative assessment of HIPPI normal conducting structures, CARE-Report-2008-071-HIPPI". In: *CARE-Report-2008-071-HIPPI* (2008). URL: http://purl.org/net/epubs/manifestation/3705/CARE-Report-08-071-HIPPI.pdf.

[30] David Longuevergne et al. "Performances of the Two First Single Spoke Prototypes for the MYRRHA Project". In: *28th International Linear Accelerator Conference*. 2017, THPLR030. DOI: 10.18429/JACoW-LINAC2016-THPLR030.

[31] F. Bouly. "Etude d'un module accélérateur supraconducteur et de ses systèmes de régulation pour le projet MYRRHA". PhD thesis. Université Paris Sud - Paris XI, Nov. 2011. URL: https://tel.archives-ouvertes.fr/tel-00660392.

[32] D Longuevergne et al. "Troubleshooting and performances of Type-B SPIRAL2 series cryomodule". In: *Proc. of the 27th Linear Accelerator Conference, LINAC14, Geneva, Switzerland*. 2014.

[33] A. Facco and V. Zviagintsev. "Study on beam steering in intermediate-β superconducting quarter wave resonators". In: *Particle Accelerator Conference, 2001. PAC 2001. Proceedings of the 2001.* Vol. 2. IEEE. 2001, pp. 1095–1097.

[34] J.R. Delayen et al. "Recent developments in the application of RF superconductivity to high-brightness and high-gradient ion beam accelerators". In: *Proceedings of the Fifth Workshop on RF Superconductivity, DESY, Hamburg, Germany*. 1991.

[35] J.R. Delayen. "Applications of Spoke Cavities". In: *Proc. LINAC*. 2010.

[36] D. Longuevergne et al. "Performances of the Two First Single Spoke Prototypes for the MYRRHA Project". In: *Proc. of Linear Accelerator Conference (LINAC'16), East Lansing, MI, USA*. May 2017, pp. 916–919. ISBN: 978-3-95450-169-4. DOI: JACoW-LINAC2016-THPLR030. URL: http://jacow.org/linac2016/papers/thplr030.pdf.

[37] C. Darve et al. "ESS Superconducting RF Collaboration". In: *Proc. of International Particle Accelerator Conference (IPAC'17),Copenhagen, Denmark*. May 2017, pp. 1068–1070. DOI: JACoW-IPAC2017-MOPVA090. URL: http://jacow.org/ipac2017/papers/mopva090.pdf.

[38] *Nuclear Energy Today, 2nd ed.* OECD/NEA, 2012. URL: https://www.oecd-nea.org/jcms/pl_14560/nuclear-energy-today-second-edition.

[39] OECD/NEA. *Accelerator-driven systems (ads) and fast reactors (fr) in advanced nuclear fuel cycles: a comparative study*. 2002. URL: https://www.oecd-nea.org/jcms/pl_33830/accelerator-driven-systems-ads-and-fast-reactors-fr-in-advanced-nuclear-fuel-cycles?details=true.

[40] Thibault Chevret. "Mesure de la réactivité de réacteurs sous-critiques pilotés par accélérateur par l'analyse d'expériences d'interruptions de faisceau programmées". Thèse de doctorat. PhD thesis. 2016. URL: http://www.theses.fr/2016CAEN2025.

[41] G.J. Van Tuyle et al. *Accelerator-driven sub-critical target concept for transmutation of nuclear wastes*. Tech. rep. Brookhaven National Lab., Upton, NY (United States), 1991.

[42] *The MYRRHA website*. URL: https://myrrha.be/.

[43] F. Bouly et al. *MYRRHA start-to-end design studies*. 2020.

[44] Frédéric Bouly et al. "Superconducting LINAC Design Upgrade in View of the 100 MeV MYRRHA Phase I". In: *10th International Particle Accelerator Conference*. 2019, MOPTS003. DOI: 10.18429/JACoW-IPAC2019-MOPTS003.

[45] Jean-Luc Biarrotte and Didier Uriot. "Dynamic compensation of an rf cavity failure in a superconducting linac". In: *Phys. Rev. ST Accel. Beams* 11 (7 July 2008), p. 072803. DOI: 10.1103/PhysRevSTAB.11.072803. URL: https://link.aps.org/doi/10.1103/PhysRevSTAB.11.072803.

[46] Frédéric Bouly, Jean-Luc Biarrotte, and Didier Uriot. "Fault Tolerance and Consequences in the MYRRHA Superconducting Linac". In: *27th International Linear Accelerator Conference*. 2014, MOPP103.

[47] J. Galambos et al. "Operational Experience of a Superconducting Cavity Fault Recovery System at the SNS". In: *5th International Workshop on the Utilisation and Reliability of High Power Proton Accelerators*. 2007.

[48] H. Wiedemann. *Particle Accelerator Physics*. Springer, 2015.

[49] F. Gérardin. *Etude de la compensation de charge d'espace dans les lignes basse énergie des accélérateurs d'ions légers de haute intensité*. 2018.

[50] D. Grote et al. *WARP*. URL: http://warp.lbl.gov/home.

[51] N. Chauvin et al. *LEBT lines: simulations and experimental results*. 2014.

[52] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[53] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[54] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2015. arXiv: 1511.07289 [cs.LG].

[55] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[56] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[57] W. Xiong et al. "Toward Human Parity in Conversational Speech Recognition". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.12 (2017), pp. 2410–2423.

[58] S. Song and J. Xiao. "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 808–816.

[59] Chen-Yu Lee and Simon Osindero. *Recursive Recurrent Nets with Attention Modeling for OCR in the Wild*. 2016. arXiv: 1603.03101 [cs.CV].

[60] W. Yang et al. "End-to-End Learning of Deformable Mixture of Parts and Deep Convolutional Neural Networks for Human Pose Estimation". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 3073–3082.

[61] Holger R. Roth et al. *Detection of Sclerotic Spine Metastases via Random Aggregation of Deep Convolutional Neural Network Classifications*. 2014. arXiv: 1407.5976 [cs.CV].

[62] A. G. Salman, B. Kanigoro, and Y. Heryadi. "Weather forecasting using deep learning techniques". In: *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. 2015, pp. 281–285.

[63] D. E. Rumelhart and J. L. McClelland. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[64] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* 2 (1989), pp. 303–314.

[65] F. Chollet. *Building Autoencoders in Keras*. URL: https://blog.keras.io/building-autoencoders-in-keras.html.

[66] Yann LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems 2*. Ed. by D. S. Touretzky. Morgan-Kaufmann, 1990, pp. 396–404. URL: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf.

[67] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[68] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way*. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[69] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[70] K. Janod. *La représentation des documents par réseaux de neurones pour la compréhension de documents parlés*. 2017.

[71] M. Nielsen. *Neural Networks and Deep Learning*. URL: http://neuralnetworksanddeeplearning.com/.

[72] R. S. Sutton and A. G. Peters. *Reinforcement Learning: An Introduction*. 2020.

[73] Deepmind. *AlphaGo*. URL: https://deepmind.com/research/case-studies/alphago-the-story-so-far.

[74] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229.

[75] Gerald Tesauro. "Programming backgammon using self-teaching neural nets". In: *Artificial Intelligence* 134.1 (2002), pp. 181 –199. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00110-2. URL: http://www.sciencedirect.com/science/article/pii/S0004370201001102.

[76] J. Peters and S. Schaal. "Policy Gradient Methods for Robotics". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 2219–2225.

[77] M. P. Deisenroth, G. Neumann, and J. Peters. *A Survey on Policy Search for Robotics*. 2013.

[78] V. Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533.

[79] Ding-Zhu Du, Panos M. Pardalos, and Weili Wu. "History of optimization". In: *Encyclopedia of Optimization*. Ed. by Christodoulos A. Floudas and Panos M. Pardalos. Boston, MA: Springer US, 2009, pp. 1538–1542. ISBN: 978-0-387-74759-0. DOI: 10.1007/978-0-387-74759-0_268. URL: https://doi.org/10.1007/978-0-387-74759-0_268.

[80] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942–1948 vol.4.

[81] Maurice Clerc. "Stagnation Analysis in Particle Swarm Optimisation or What Happens When Nothing Happens". 17 pages. Dec. 2006. URL: https://hal.archives-ouvertes.fr/hal-00122031.

[82] Andrea Serani et al. "On the use of synchronous and asynchronous single-objective deterministic particle swarm optimization in ship design problems". In: June 2014.

[83] Ioan Cristian Trelea. “The particle swarm optimization algorithm: convergence analysis and parameter selection”. In: *Information Processing Letters* 85.6 (2003), pp. 317 –325. ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(02)00447-7. URL: http://www.sciencedirect.com/science/article/pii/S0020019002004477.

[84] C. A. Coello Coello and M. S. Lechuga. “MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization”. In: *Proceedings of the Evolutionary Computation on 2002. CEC ’02. Proceedings of the 2002 Congress - Volume 02*. CEC ’02. USA: IEEE Computer Society, 2002, 1051–1056. ISBN: 0780372824.

[85] K. E. Parsopoulos and M. N. Vrahatis. “Particle Swarm Optimization Method in Multiobjective Problems”. In: *Proceedings of the 2002 ACM Symposium on Applied Computing*. SAC ’02. Association for Computing Machinery, 2002, 603–607. ISBN: 1581134452. DOI: 10.1145/508791.508907. URL: https://doi.org/10.1145/508791.508907.

[86] Margarita Reyes-Sierra and Carlos Coello. “Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art”. In: *International Journal of Computational Intelligence Research* 2 (Jan. 2006), pp. 287–308. DOI: 10.5019/j.ijcir.2006.68.

[87] X. Pang and L.J. Rybarcyk. “Multi-objective particle swarm and genetic algorithm for the optimization of the LANSCE linac operation”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 741 (2014), pp. 124 –129. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2013.12.042. URL: http://www.sciencedirect.com/science/article/pii/S0168900213017464.

[88] Los Alamos National Laboratory. *Los Alamos Neutron Science Center*. URL: https://lansce.lanl.gov/.

[89] Los Alamos National Laboratory. *Parmila*. URL: https://laacg.lanl.gov/laacg/services/download_PMI.phtml.

[90] M. Apollonio et al. “Online Multi Objective Optimisation at Diamond Light Source”. In: *Proc. 9th International Particle Accelerator Conference (IPAC’18), Vancouver, BC, Canada, April 29-May 4, 2018* (Vancouver, BC, Canada). International Particle Accelerator Conference 9. https://doi.org/10.18429/JACoW-IPAC2018-WEPAF054. Geneva, Switzerland: JACoW Publishing, June 2018, pp. 1944–1947. ISBN: 978-3-95450-184-7. DOI: doi:10.18429/JACoW-IPAC2018-WEPAF054. URL: http://jacow.org/ipac2018/papers/wepaf054.pdf.

[91] *The Diamond Light Source*. URL: https://www.diamond.ac.uk/Home.html.

[92] Ian Martin et al. “An Online Multi-Objective Optimisation Package”. In: *8th International Particle Accelerator Conference*. 2017, THPAB153. DOI: 10.18429/JACoW-IPAC2017-THPAB153.

[93] K. Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197.

[94] A. SUPPAPITNARM et al. "A SIMULATED ANNEALING ALGORITHM FOR MULTIOBJECTIVE OPTIMIZATION". In: *Engineering Optimization* 33.1 (2000), pp. 59–85. DOI: 10.1080/03052150008940911. eprint: https://doi.org/10.1080/03052150008940911. URL: https://doi.org/10.1080/03052150008940911.

[95] A. L. Edelen et al. *Neural Network Model Of The PXIE RFQ Cooling System and Resonant Frequency Response*. 2016. arXiv: 1612.07237 [physics.acc-ph].

[96] Fermilab. *Proton Improvement Plan-II (PIP-II)*. URL: https://pip2.fnal.gov/.

[97] Auralee Edelen et al. "Using A Neural Network Control Policy For Rapid Switching Between Beam Parameters in an FEL". In: *38th International Free-Electron Laser Conference*. 2018, WEP031. DOI: 10.18429/JACoW-FEL2017-WEP031.

[98] D. Uriot. *TraceWin rapid explanation*. URL: http://irfu.cea.fr/en/Phocea/Page/index.php?id=780.

[99] R. Dupperier. *Toutatis rapid explanation*. URL: http://irfu.cea.fr/en/Phocea/Page/index.php?id=781.

[100] F. Bouly, N. Chauvin, and M. Debongnie. *MYRRHA Low Energy Beam Transport Line Commissioning and Space-charge experiments*. 2020.

[101] D. Uriot et al. *MYRTE Task 2.6 deliverable: Start to End Model of the 100 MeV MYRRHA linac*. 2019.

[102] J. L. Biarotte et al. "Beam operation aspects for the MYRRHA linear accelerator". In: *International Workshop on Technology and Components of Accelerator-driven Systems*. 2013, pp. 135–147.

[103] F. Bouly et al. *The MYRRHA LEBT, Commissioning and Space Charge Compensation Experiments*. 2018.

[104] *Internal communications with B. Cheymol*.

[105] P. Forck. *Lecture Notes on Beam Instrumentation and Diagnostics*. 2008.

[106] R. Duperrier. *Quelques formules pour évaluer les performances d'un émittancemètre de type ALLISON*. 2007.

[107] P.-Y. Beauvais and B. Pottin. *IPHI: Injecteur de Protons à Haute Intensité*. URL: http://irfu.cea.fr/Phocea/Vie_des_labos/Ast/ast_technique.php?id_ast=793.

[108] Olivier Piquet, Michel Desmons, and Alain France. "RF Tuning of the IPHI RFQ". In: *5th International Particle Accelerator Conference*. July 2014, TH-PME055. DOI: 10.18429/JACoW-IPAC2014-THPME055.

[109] Google. *TensorFlow*. URL: https://www.tensorflow.org/.

[110] Yann A. LeCun et al. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. URL: https://doi.org/10.1007/978-3-642-35289-8_3.

[111] J. Brownlee. *A Gentle Introduction to k-fold Cross-Validation*. URL: https://machinelearningmastery.com/k-fold-cross-validation/.

[112] Data Robot. *Cross-Validation*. URL: https://www.datarobot.com/wiki/cross-validation/.

[113] C. Adam-Bourdarios et al. "The Higgs Machine Learning Challenge". In: *Journal of Physics Conference Series*. Vol. 664. Journal of Physics Conference Series. Dec. 2015, p. 072015. DOI: 10.1088/1742-6596/664/7/072015.

[114] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575.

[115] Frédéric Bouly. "MYRRHA Superconducting linac & Fault Tolerance Concept". In: *30th International Linear Accelerator Conference 2020 - Virtual Conference*. Liverpool, United Kingdom, Sept. 2020. URL: http://hal.in2p3.fr/in2p3-03062748.

[116] R. Duperrier et al. "TOUTATIS, the CEA Saclay RFQ code". In: *eConf* C000821 (2000). Ed. by A. W. Chao, THB19. arXiv: physics/0008121.

[117] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: JMLR Workshop and Conference Proceedings, May 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[118] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

Appendix A

Codes developed and used during the thesis with commentaries

This appendix contains the commented codes that were developed and used during the thesis.

The codes used for the machine learning based applications are given in Section A.1.

Then, the codes used to perform Particle Swarm Optimizations are given in Section A.2.

A.1 Machine learning

Here is a compilation of the codes and files used to train a neural network model with supervised learning. The codes are separated as follows.

- Model builder (Python file)
  This file called *modelBuilder.py* provides the functions used to create an object that contains a Keras model and several helper functions (see Appendix A.1.1).

- Model parameters (data file)
  This file called *modelParameters.dat* contains the description of the parameters of the model and of the training to perform with the model (see Appendix A.1.2).

- Training (Python file)
  This file called *training.py* contains the functions to train a Keras model (see Appendix A.1.3).

- Data pipeline (Python file)
  This file called *importData.py* contains the functions to control the data pipeline used to train a Keras model (see Appendix A.1.4).

A.1.1 modelBuilder.py

The following code is used to create an object that contains a Keras model implemented using TensorFlow v1.15 and several helper functions to save and load models and archive their parameters.

This implementation allows to build or load any dense neural networks with or without branching. The structure of the network and the activation function of each layer are defined in an external file called *modelParameters.dat* (an example of such a file is given in Section A.1.2).

The following four activation functions can be used.

- The sigmoid
$$f(x) = \frac{1}{e^x + e^{-x}}$$
The weights of a sigmoid layer are initialized using a Glorot normal initialization. [117]

- The hyperbolic tangent (*tanh*)
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
The weights of a *tanh* layer are initialized using a Glorot normal initialization. [117]

- The Rectified Linear Unit (*ReLU*
$$f(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ else} \end{cases}$$
The weights of a *ReLU* layer are initialized using a He normal initialization. [118]

- The linear function
$$f(x) = x$$
The weights of a linear layer are initialized using a He normal initialization. [118]

The optimization function for the weights and biases is the Stochastic Gradient Descent.

```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf
import os
from shutil import copyfile
import glob

class Model:

    # Instantiate a model with the values provided in a external file

    def __init__(self, modelDescriptionFileName):

        self.modelDescriptionFileName = modelDescriptionFileName
        self.modelDirName = None
        self.modelBootstrap = -1

        self.nCycle = 5
        self.nEpoch = 10
        self.batchSize = None
        self.useL2Cost = None
        self.learningRate = None
        self.optimizer = tf.keras.optimizers.SGD

        self.stateNames = []
        self.stateFeatures = []
        self.hiddenUnits = []
        self.actionNames = []
        self.actionFeatures = []
        self.actionUnits = []

        self.initializer()

        # Checks if the model already exists:
        # If it does --> Load the last version of the model to continue training
        # else --> Create the model

        if (not os.path.isdir(self.modelDirName)):

            os.mkdir(self.modelDirName)
            copyfile(self.modelDescriptionFileName,
                     self.modelDirName + '\\' + self.modelDescriptionFileName)

        modelName = 'model*.h5'
        modelFiles = glob.glob(self.modelDirName + '\\' + modelName)
        if (len(modelFiles) > 0):
            modelVersions = [
                int(modelFile[(len(self.modelDirName)+6):].split('.')[0])
                for modelFile in modelFiles
                ]
            if (len(modelVersions) > 1):
                modelVersions.sort()

            loadVersion = modelVersions[self.modelBootstrap]

            print('Attempting to load from model{:03d}.h5!'.format(loadVersion))

            self.model = tf.keras.models.load_model(
```

```
61                self.modelDirName + '\\model{:03d}.h5'.format(loadVersion)
62                )
63
64            print('Loading succesful!')
65
66        else:
67
68            print('Compiling new model!')
69
70            self.model = self.MyModel()
71
72            self.model.compile(
73                optimizer=self.optimizer(lr=self.learningRate),
74                loss='mean_squared_error',#'huber_loss',
75                metrics=['mean_squared_error']
76            )
77
78            print('New model compiled succesfuly!')
79
80        self.updateLog()
81
82
83    # Records the parameters of the model in a human readable way
84
85    def updateLog(self):
86
87        with open(self.modelDirName + '\\modelSummary.log', 'w+') as file:
88
89            file.write('Model Description file :' +
90                       self.modelDescriptionFileName + '\n')
91            file.write('Model directory : ' + self.modelDirName + '\n')
92            file.write('Batch size : ' + str(self.batchSize) + '\n')
93            file.write('Learning rate :' + str(self.learningRate) + '\n')
94
95            line = ''
96            for stateName in self.stateNames:
97                line += stateName + '\t'
98            file.write('Inputs : ' + line + '\n')
99
100           file.write('NN structure\n')
101           line = ''
102           for layerDepth in range(len(self.hiddenUnits)):
103
104               for actionHead in range(len(self.hiddenUnits[layerDepth])):
105
106                   line += self.hiddenUnits[layerDepth][actionHead][0] + \
107                           '(' + self.hiddenUnits[layerDepth][actionHead][1] + \
108                           ')' + '\t'
109
110               file.write(line.rstrip() + '\n')
111               line = ''
112
113           file.write('Outputs structure\n')
114           line = ''
115           line2 = ''
116           for actionHead in range(len(self.actionFeatures)):
117
118               for action in range(len(self.actionFeatures[actionHead])):
119
120                   line += self.actionFeatures[actionHead][action] + ' + '
```

```
                    line2 += self.actionUnits[actionHead][action][0] + \
                             '(' + self.actionUnits[actionHead][action][1] + \
                             ')' + ' + '

                line = line[:-3] + '\t'
                line2 = line2[:-3] + '\t'

            file.write(line.rstrip() + '\n')
            file.write(line2.rstrip() + '\n')
            line = ''



    # Helper function to initialize the parameters of the model

    def initializer(self):

        def setModelDir(modelDirName):
            self.modelDirName = modelDirName

        def setModelBootstrap(modelBootstrap):
            self.modelBootstrap = int(modelBootstrap)

        def setNCycle(nCycle):
            self.nCycle = int(nCycle)

        def setNEpoch(nEpoch):
            self.nEpoch = int(nEpoch)

        def setBatchSize(batchSizeStr):
            self.batchSize = int(batchSizeStr)

        def setUseL2Cost(useL2CostStr):
            self.useL2Cost = bool(useL2CostStr)

        def setLearningRate(learningRateStr):
            self.learningRate = float(learningRateStr)

        def setStateFeatures(stateFeaturesStr):
            self.stateNames = [
                name.strip()
                for name in stateFeaturesStr.split(',')
                ]
            self.stateFeatures = [
                tf.feature_column.numeric_column(key=feature, shape=(1,))
                for feature in self.stateNames
                ]

        def setHiddenUnits(hiddenUnitsStr):
            self.hiddenUnits = [[[
                x.strip()
                for x in column.split('-')
                ]
                for column in layer.split(';')
                ]
                for layer in hiddenUnitsStr.split(',')
                ]

        def setActionFeatures(actionFeaturesStr):
            self.actionFeatures = [[
```

```
                action.strip()
                for action in head.split(';')
                ]
                for head in actionFeaturesStr.split(',')
                ]

            for actionHead in self.actionFeatures:
                for action in actionHead:
                    self.actionNames.append(action)

        def setActionUnit(actionUnitsStr):
            self.actionUnits = [[[
                x.strip()
                for x in action.split('-')
                ]
                for action in head.split(';')
                ]
                for head in actionUnitsStr.split(',')
                ]

        setter = {
            'modelDir': setModelDir,
            'modelBootstrap': setModelBootstrap,
            'nCycle': setNCycle,
            'nEpoch': setNEpoch,
            'batchSize': setBatchSize,
            'useL2Cost': setUseL2Cost,
            'learningRate': setLearningRate,
            'stateFeatures': setStateFeatures,
            'hiddenUnits': setHiddenUnits,
            'actionFeatures': setActionFeatures,
            'actionUnits': setActionUnit
        }

        with open(self.modelDescriptionFileName, 'r') as file:
            for line in file:
                line = [
                    x.rstrip()
                    for x in line.split(':')
                    ]
                for x in line:
                    print(x)
                setter[line[0]](line[1])



    # Model construction function
    # Inputs: parameters of the model
    # Output: a corresponding Keras model

    def MyModel(self):

        initializerSwitcher = {
            'sigmoid': tf.variance_scaling_initializer(
                scale=1.0,
                mode='fan_avg',
                distribution='truncated_normal'
                ),
            'tanh': tf.variance_scaling_initializer(
                scale=1.0,
```

```
                mode='fan_avg',
                distribution='truncated_normal'
                ),
            'relu': tf.variance_scaling_initializer(
                scale=2.0,
                mode='fan_in',
                distribution='truncated_normal'
                ),
            'lin': tf.variance_scaling_initializer(
                scale=2.0,
                mode='fan_in',
                distribution='truncated_normal'
                )
            }

        # Instantiate NN body with branching if necessary
        net = [[tf.keras.layers.Dense(
                        int(self.hiddenUnits[layerDepth][actionHead][1]),
                        activation=self.hiddenUnits[layerDepth][actionHead][0],
                        kernel_initializer=initializerSwitcher[
                            self.hiddenUnits[layerDepth][actionHead][0]
                            ]

                    )
                    for actionHead in range(len(self.hiddenUnits[layerDepth]))
                    ]
                    for layerDepth in range(len(self.hiddenUnits))
                    ]

        outputsLayer = [[tf.keras.layers.Dense(
                            int(self.actionUnits[actionHead][action][1]),
                            kernel_initializer=initializerSwitcher[
                                self.actionUnits[actionHead][action][0]
                                ],
                            name=self.actionFeatures[actionHead][action]
                            )
                            if (self.actionUnits[actionHead][action][0] == 'lin')
                            else
                            tf.keras.layers.Dense(
                            int(self.actionUnits[actionHead][action][1]),
                            activation=self.actionUnits[actionHead][action][0],
                            kernel_initializer=initializerSwitcher[
                                self.actionUnits[actionHead][action][0]
                                ],
                            name=self.actionFeatures[actionHead][action]
                            )
                        for action in range(len(self.actionFeatures[actionHead]))
                        ]
                        for actionHead in range(len(self.hiddenUnits[-1]))
                        ]

        inputs = [
            tf.keras.layers.Input(shape=(1,), name=name)
            for name in self.stateNames
            ]

        inputLayer = tf.keras.layers.concatenate(inputs)

        x = [inputLayer]

```

```
        for layerDepth in range(len(net)):

            for actionHead in range(len(net[layerDepth]) - 1, -1, -1):

                if (len(x) > actionHead):

                    x[actionHead] = net[layerDepth][actionHead](x[actionHead])

                else:

                    x.append(net[layerDepth][actionHead](x[0]))

        outputs = []

        for actionHead in range(len(outputsLayer)):

            for action in outputsLayer[actionHead]:

                outputs.append(action(x[actionHead]))

        return tf.keras.Model(inputs=inputs, outputs=outputs)
```

A.1.2 modelParameters.dat

This file contains the parameters of a Keras model. It is read by the *modelBuilder.py* and *training.py* files to create and train a network.

- modelDir
  Set the name of the directory in which to put all the files following the creation and training of a model.

- batchSize
  Set the size of a batch of data for the model training.

- nCycle
  Set the number of cycles to perform the training. Each cycle corresponds to *nEpoch* epochs.

- nEpoch
  Set the number of epochs to perform per cycle.

- useL2Cost
  Set if L2 normalization is used during training.

- learningRate
  Set the initial learning rate for the training.

- stateFeatures
  Set the input(s) of the model. They correspond to the feature names defined in the data pipeline as set in the *importData.py* file (see Section A.1.4).

- hiddenUnits
  Set the hidden layers of the model.

  A layer is described by its activation function(s) and its respective number of hidden neurons. With a single activation function, the syntax is: function name - number of neurons. With multiple activation functions (this case corresponds to a branching model) the syntax is: function name - number of neurons; function name - number of neurons; ...

  Layers are separated with a comma e.g. relu-96,relu-96 corresponds to 2 $ReLU$ layers with 96 neurons each.

- actionFeatures
  Set the output(s) of the model. They correspond to the feature names defined in the data pipeline as set in the *importData.py* file (see Section A.1.4).

- actionUnits
  Set the output layer of the model.

  The syntax is the same as for the hidden layers.

```
modelDir:3Xrelu96_sig_log_cv2
batchSize:64
nCycle:10
nEpoch:10
useL2Cost:False
learningRate:0.1
stateFeatures:Sol1,Sol2,Steer1V,Steer1H,Steer2V,Steer2H,SlitHminus,SlitHplus,
                SlitVminus,SlitVplus,Pres1,Pres2,Pres3
hiddenUnits:relu-96,relu-96,relu-96
actionFeatures:Current
actionUnits:sigmoid-1
```

A.1.3 training.py

This file is called to train a model. It reads the *modelParameters.dat* file to get the parameters of the model and the training schedule to perform on the model. It calls *modelBuilder.py* to create or load the model and *importData.py* to get the data pipelines for the training.

The training is separated into cycles of multiple epochs. After each cycle, the model is saved and evaluated on the test dataset. In addition, its predictions over the complete dataset are saved for manual investigation.

```python
import importData_log as importData
import modelBuilder
import tensorflow as tf
import numpy as np
import os
from collatePredReal import collatePredictionsAndExperiment

# For debugging purposes
# tf.enable_eager_execution()

# Name of the file containing the parameters of the model to work
# with and the parameters for the training
modelDescriptionFile = 'modelParameters.dat'


# Main function handling the model and its training
def main(argv):

    assert len(argv) == 1

    # Instantiating of the model through modelBuilder.py
    modelHolder = modelBuilder.Model(modelDescriptionFile)
    model = modelHolder.model


    # Declaration of the data pipelines through importData.py
    # Training data pipeline
    trainData = importData.train_input_fn(modelHolder.batchSize)

    # Validation data pipelines
    # When training on simu
    valData = importData.valid_input_fn(128)
    # When retraining on expe
    # valData = importData.valid_input_fn(107)

    # Test data pipeline
    testData = importData.test_input_fn(modelHolder.batchSize)

    # Scan data pipeline for the manual verification of the model
    scanData = importData.scan_input_fn(modelHolder.batchSize)


    # Callbacks declaration
    callbacks = [

        # Callback to monitor the training using TensorBoard
        tf.keras.callbacks.TensorBoard(
            log_dir=modelHolder.modelDirName + '\\logs',
            histogram_freq=1,
            batch_size=modelHolder.batchSize,
            write_grads=True
            ),

        # Callback to reduce the learning rate if the training
        # is stagnating
        tf.keras.callbacks.ReduceLROnPlateau(
            monitor='loss',
            factor=0.2,
            patience=8,
            min_lr=0.0001,
```

```
            verbose=1,
            mode='min'
            )
    ]

    # Declaration of the training parameters
    # Number of cycle to perform (the model is saved every cycle and
    # its performances on the scan dataset is evaluated)
    nCycle = modelHolder.nCycle
    # Number of epoch to perform (each epoch the model is trained once over every
    # examples in the training dataset)
    nEpoch = modelHolder.nEpoch

    # Main training loop
    for cycle in range(nCycle):

        # Load the number of epochs already performed if any
        if os.path.isfile(modelHolder.modelDirName + '\\trainingStatus.dat'):

            with open(modelHolder.modelDirName + '\\trainingStatus.dat', 'r') as f:

                initEpoch = int(f.readline())

        else:

            initEpoch = 0

        # Set the final epoch number (the number of epoch actually performed
        #  is finalEpoch - initEpoch = nEpoch)
        finalEpoch = nEpoch + initEpoch

        # Training command when training on experimental data
        # model.fit(trainData, epochs=finalEpoch, initial_epoch=initEpoch,
        #     callbacks=callbacks, verbose=1,
        #     validation_data=valData, validation_steps=36)

        # Training command when training on simulation data
        model.fit(trainData, epochs=finalEpoch, initial_epoch=initEpoch,
            callbacks=callbacks, verbose=1,
            validation_data=valData, validation_steps=158)

        # Evaluation of the model performance on the test dataset
        model.evaluate(testData)
        print('\n')

        # Log the final epoch reach in this cycle
        with open(modelHolder.modelDirName + '\\trainingStatus.dat', 'w') as f:

            f.write(str(finalEpoch))

        # Save the model
        model.save(
            modelHolder.modelDirName + '\\model{:03d}.h5'.format(
                int(initEpoch / nEpoch)
                )
            )


        # Evaluation and archiving of the model performance
        # on the scan dataset for manual exploration
```

```
        predictions = np.asarray(model.predict(scanData))
        np.savetxt(
            modelHolder.modelDirName + '\\predictionsScan{:03d}.txt'.format(
                int(initEpoch / nEpoch)
                ),
                predictions
            )

    # Helper function that prepare the eploration of the results on the scan dataset
    collatePredictionsAndExperiment(modelHolder.modelDirName)


if __name__ == '__main__':

    tf.logging.set_verbosity(tf.logging.INFO)
    tf.app.run(main)
```

A.1.4 importData.py

This file provides the data pipelines required to train, evaluate and test a neural network. The datasets in the *TFRecord* format are parsed and scaled into single examples then collated into batches of data. This file defines the names of the model inputs and outputs as used in the *modelParameters.dat* file (see Section A.1.2).

```python
import tensorflow as tf
import collections
import numpy as np

# Declaration of the working folder
PATH = 'C:\\Thesis\\pythonScript_thesis\\projet_IPHI\\IPHI_manip\\NNs\\'

# Declaration of the dataset paths
def getPath(trainFileNames=[
        'data\\simuRepl_0.tfrecord', 'data\\simuRepl_1.tfrecord',
        'data\\simuRepl_2.tfrecord', 'data\\simuRepl_3.tfrecord'
        ],
    testFileNames=['data\\simuRepl_4.tfrecord'],
    scanFileNames=['data\\mapSol.tfrecord','data\\mapSt.tfrecord']):

    # Id of the validation data in trainFileNames
    # (used to switch between the data subsets for cross validation)
    dataId = 0


    validPath = []
    validPath.append(trainFileNames.pop(dataId))
    trainPath = trainFileNames
    testPath = testFileNames
    scanPath = scanFileNames

    return trainPath, testPath, validPath, scanPath

trainPath, testPath, validPath, scanPath = getPath()

# Definitions of the feature names in the data sets
featuresNames = [
    'Sol1',
    'Sol2',
    'Steer1V',
    'Steer1H',
    'Steer2V',
    'Steer2H',
    'SlitHminus',
    'SlitHplus',
    'SlitVminus',
    'SlitVplus',
    'Pres1',
    'Pres2',
    'Pres3'
]

# Definition of the label names in the data sets
predNames = [
    'Current'
]

# Scaling function for the features
def scaleFeature(featureDict):

    # Sol1 is [50, 110]
    featureDict['Sol1'] = (featureDict['Sol1'] - 80.) / 30.
    # Sol2 is [50, 110]
    featureDict['Sol2'] = (featureDict['Sol2'] - 80.) / 30.
    # Steer1V is [-3, 3]
```

```
    featureDict['Steer1V'] = featureDict['Steer1V'] / 3.
    # Steer1H is [-3, 3]
    featureDict['Steer1H'] = featureDict['Steer1H'] / 3.
    # Steer2V is [-3, 3]
    featureDict['Steer2V'] = featureDict['Steer2V'] / 3.
    # Steer2H is [-3, 3]
    featureDict['Steer2H'] = featureDict['Steer2H'] / 3.
    #SlitHminus is [0, 60]
    featureDict['SlitHminus'] = (featureDict['SlitHminus'] - 30.) / 30.
    #SlitHplus is [0, 60]
    featureDict['SlitHplus'] = (featureDict['SlitHplus'] - 30.) / 30.
    #SlitVminus is [0, 60]
    featureDict['SlitVminus'] = (featureDict['SlitVminus'] - 30.) / 30.
    #SlitVplus is [0, 60]
    featureDict['SlitHplus'] = (featureDict['SlitHplus'] - 30.) / 30.
    # Pres1 is [0, 5e-5]
    featureDict['Pres1'] = (featureDict['Pres1'] - 2.5e-5) / 5.0e-5
    # Pres2 is [0, 5e-5]
    featureDict['Pres2'] = (featureDict['Pres2'] - 2.5e-5) / 5.0e-5
    # Pres3 is [0, 5e-5]
    featureDict['Pres3'] = (featureDict['Pres3'] - 2.5e-5) / 5.0e-5

    return featureDict

# Scaling function for the labels
def scalePreds(predDict):

    # Current is [0, 10]
    predDict['Current'] = (predDict['Current']) / 10.

    # predDict['Current'] = tf.math.log(predDict['Current'])/tf.math.log(10.)

    return predDict


# Parsing function for training of an example from the data sets
def _parse_function_train(example_proto):

    # Separate a full example in its feature elements and its label elements
    featuresSet = {
        'Inputs': tf.FixedLenFeature([13], dtype=tf.float32),
        'Outputs': tf.FixedLenFeature([1], dtype=tf.float32)
    }
    parsedFeatures = tf.parse_single_example(example_proto, featuresSet)

    # Fill a dictionary with the features
    myFeatures = {}
    for idx, names in enumerate(featuresNames):
        myFeatures[names] = parsedFeatures['Inputs'][idx]

    # Scale the features
    myFeatures = scaleFeature(myFeatures)

    # Fill a dictionary with the labels
    myPreds = {}
    for idx, names in enumerate(predNames):
        myPreds[names] = parsedFeatures['Outputs'][idx]

    # Scale the labels
    myPreds = scalePreds(myPreds)
```

```
121
122     return myFeatures, myPreds
123
124 # Parsing function for predictions of an example from the data sets
125 def _parse_function_predict(example_proto):
126
127     # Separate a full example in its feature elements and its label elements
128     featuresSet = {
129         'Inputs': tf.FixedLenFeature([13], dtype=tf.float32),
130         'Outputs': tf.FixedLenFeature([1], dtype=tf.float32)
131     }
132     parsedFeatures = tf.parse_single_example(example_proto, featuresSet)
133
134     # Fill a dictionary with the features
135     myFeatures = {}
136     for idx, names in enumerate(featuresNames):
137         myFeatures[names] = parsedFeatures['Inputs'][idx]
138
139     # Scale the features
140     myFeatures = scaleFeature(myFeatures)
141
142     return myFeatures
143
144 # Function that create the training data pipeline
145 def train_input_fn(batchSize):
146
147     # Load the dataset
148     dataset = tf.data.TFRecordDataset(trainPath)
149
150     # Shuffle the dataset
151     dataset = dataset.shuffle(20000)
152
153     # Parse the dataset
154     dataset = dataset.map(_parse_function_train)
155
156     # Separate the dataset into batches with a size equal to batchSize
157     dataset = dataset.batch(batchSize).prefetch(2)
158
159     return dataset
160
161 # Function that create the test data pipeline
162 def test_input_fn(batchSize):
163
164     # Load the dataset
165     dataset = tf.data.TFRecordDataset(testPath)
166
167     # Parse the dataset
168     dataset = dataset.map(_parse_function_train)
169
170     # Separate the dataset into batches with a size equal to batchSize
171     dataset = dataset.batch(batchSize)
172
173     return dataset
174
175 # Function that create the validation data pipeline
176 def valid_input_fn(batchSize):
177
178     # Load the dataset
179     dataset = tf.data.TFRecordDataset(validPath)
180
```

```
    # Parse the dataset
    dataset = dataset.map(_parse_function_train)

    # Separate the dataset into batches with a size equal to batchSize
    dataset = dataset.batch(batchSize)

    return dataset

# Function that create the scan data pipeline
def scan_input_fn(batchSize):

    # Load the dataset
    dataset = tf.data.TFRecordDataset(scanPath)

    # Parse the dataset
    dataset = dataset.map(_parse_function_predict)

    # Separate the dataset into batches with a size equal to batchSize
    dataset = dataset.batch(batchSize)

    return dataset
```

A.2 Particle Swarm Optimization

Here is a compilation of the codes and files used to perform a Particle Swarm Optimization. The codes are separated as follows.

- Particle Swarm Optimization (python file)
  This file called *MOPSO.py* contains the objects and functions used for PSO (see Appendix A.2.1). It allows for both single and multi objective optimization.

- Interface to the MYRRHA LEBT (python file)
  This file called *PSO2LEBT.py* contains the objects and functions used to apply the code in *MOPSO.py* to the MYRRHA LEBT (see Appendix A.2.2.

- Interface to the neural network model of the MYRRHA LEBT (python file)
  This file called *PSO2model.py* contains the objects and functions used to apply the code in *MOPSO.py* to neural network models of the MYRRHA LEBT (see Appendix A.2.3.

A.2.1 MOPSO.py

The following code defines two objects and their functions.

- A *Particle* object
  This object essentially contains:
  - a position (a candidate solution of the optimization problem),
  - a velocity (the rule to update the position between iterations),
  - the function to update its position and velocity,
  - the memory of the best position previously visited and
  - various helper functions.

  The initial position is randomly chosen using a uniform distribution within the boundaries of the search space:

$$x_j^0 \sim U(x_{j,min}, x_{j,max})$$

  where the j subscript corresponds to the j^{th} parameters and the min and max subscript respectively corresponds to the lower and higher boundaries of the parameters.

  The initial velocity is randomly chosen using a scaled uniform distribution:

$$v_j^0 \sim \frac{U(-1,1) * (x_{j,max} - x_{j,min})}{4}.$$

- A *Population* object
  This object contains:
  - the *Particles*,
  - the search space,
  - the objective(s),
  - the link to the function used to evaluate the particle position,
  - the memory of the global best position previously visited,
  - the rules to perform the single or multi objective optimization and
  - various helper functions.

  Note that the search space, the objective(s) and the evaluation function are defined in another file that serves as an interface between the PSO code and the actual optimization problem (see Appendix A.2.2 for the MYRRHA LEBT interface and Appendix A.2.3 for the neural network model of the MYRRHA LEBT interface).

```python
import numpy as np
import multiprocessing as mp
import matplotlib.pyplot as plt
import sys, os, time

# Define the weight used in the particle velocities update
socialWeight = 2.05
personalWeight = 2.05
inertialWeight = 0.729

# Definition of the optimization functions
def npMin(arr):
    return arr.min()

def npArgMin(arr):
    return arr.argmin()

def npLess(value, arr):
    return np.less(value, arr)

def npLessEqual(value, arr):
    return np.less_equal(value, arr)

def npMax(arr):
    return arr.max()

def npArgMax(arr):
    return arr.argmax()

def npGreater(value, arr):
    return np.greater(value, arr)

def npGreaterEqual(value, arr):
    return np.greater_equal(value, arr)

# Dictionary to switch between minimizing and maximizing functions
optTypeDict = {
    'Min': [npMin, npArgMin, npLess, npLessEqual],
    'Max': [npMax, npArgMax, npGreater, npGreaterEqual]
    }

# Definition of the Particle object
class Particle:

    # Instantiate a particle from archive if any or from scratch
    def __init__ (self, id, pop, bootstrap):

        self.id = id
        bounded = []
        scale = []
        offset = []

        for i in range(len(pop.searchSpace)):

            bounded.append(True)
            scale.append(pop.searchSpace[i].range[1] - pop.searchSpace[i].range[0])
            offset.append(pop.searchSpace[i].range[0])

            if (scale[-1] == offset[-1]):

```

```
                scale[-1] = 20.
                offset[-1] = -10.
                bounded[-1] = False

        self.bounded = bounded

        # Randomly set initial values for position and velocity
        scale = np.asarray(scale)
        offset = np.asarray(offset)
        pos = offset + np.random.random_sample(len(scale)) * scale
        vel = (-1 + 2 * np.random.random_sample(len(scale))) * scale / 4.
        bestPos = pos

        # Check for position, velocity or best known position bootstraping
        # and resolve if any
        if np.any(bootstrap[2] or bootstrap[3] or bootstrap[5]):

            if (id < len(next(iter(pop.restoredData['particles'].values())))):

                for i in np.where(pop.hasPVBackup)[0]:

                    if bootstrap[2]:

                        data = pop.restoredData \
                            ['particles'][pop.searchSpace[i].name][id]

                        if bounded[i]:

                            if (data < pop.searchSpace[i].range[0]):

                                data = pop.searchSpace[i].range[0]

                            elif (data > pop.searchSpace[i].range[1]):

                                data = pop.searchSpace[i].range[1]

                        pos[i] = data

                    if bootstrap[3]:

                        data = pop.restoredData \
                            ['particles'][pop.searchSpace[i].name + '_v'][id]

                        if not (data > scale[i]):

                            vel[i] = data

                    if bootstrap[5]:

                        data = pop.restoredData \
                            ['particles'][pop.searchSpace[i].name + '_best'][id]

                        if bounded[i]:

                            if (data < pop.searchSpace[i].range[0]):

                                data = pop.searchSpace[i].range[0]

                            elif (data > pop.searchSpace[i].range[1]):

```

```
                                data = pop.searchSpace[i].range[1]

                        bestPos[i] = data

        # Assign particle position and velocity
        self.vel = vel
        self.pos = pos
        self.bestPos = pos

        # Set initial scorre and best score depending on if minimizing or maximizing
        score = [
            float('inf') if goal.optType == 'Min'
            else -float('inf') for goal in pop.objectives
            ]
        bestScore = [
            float('inf') if goal.optType == 'Min'
            else -float('inf') for goal in pop.objectives
            ]

        # Check for score of best score boostraping and resolve if any
        if bootstrap[4]:

            if (id < len(next(iter(pop.restoredData['particles'].values())))):

                for i in np.where(pop.hasSBackup)[0]:

                    score[i] = pop.restoredData \
                        ['particles'][pop.objectives[i].goalName + \
                        '(' + pop.objectives[i].diag + ')'][id]

                    if bootstrap[5]:

                        bestScore[i] = pop.restoredData \
                            ['particles'][pop.objectives[i].goalName + \
                            '(' + pop.objectives[i].diag + ')_best'][id]

        # Assign particle score and best score
        self.score = np.asarray(score)
        self.bestScore = np.asarray(bestScore)

        self.dimRange = np.asarray([dim.range for dim in pop.searchSpace])
        self.pop = pop

    # Alias function for the evaluation of the particle
    def eval(self):
        return self.pop.evalFn(self)

    # Score updating function
    def updateBest(self):

        self.bestPos = self.pos
        self.bestScore = self.score

    # Position and velocity updating function
    def updatePosition(self, leaderPos):
        rand = np.random.random_sample(2)

        self.vel = inertialWeight * (
                            self.vel +
                            personalWeight * rand[0] * (self.bestPos - self.pos) +
```

```
                            socialWeight * rand[1] * (leaderPos - self.pos)
                            )
        self.pos = self.pos + self.vel

        for i in range(len(self.pos)):

            if self.bounded[i]:

                if (self.pos[i] < self.dimRange[i, 0]):

                    self.pos[i] = self.dimRange[i, 0]
                    self.vel[i] = - self.vel[i] / (inertialWeight * (
                        socialWeight + personalWeight
                        ))

                elif (self.pos[i] > self.dimRange[i, 1]):

                    self.pos[i] = self.dimRange[i, 1]
                    self.vel[i] = - self.vel[i] / (inertialWeight * (
                        socialWeight + personalWeight
                        ))

    # Archiving function
    def write(self):

        print('Particle #' + str(self.id) + ' current position: ', self.pos)
        print('Particle #' + str(self.id) + ' current velocity: ', self.vel)
        print('Particle #' + str(self.id) + ' personal best: ', self.bestScore)


# Definition of the Population object
class Population:

    # Initialization of the population
    def __init__ (self, searchSpace, objectives, evalFn, nPart = -1, epsilon = [],
                  bootstrap = [False, False, False, False, False, False],
                  backupTarget = 'Best', backupType = 'Last',
                  backupName='PSO_history', nProc = 0):

        print('Starting population initialization')

        # Determination of the number of particles
        if (nPart == -1):

            self.nPart = 5 * len(searchSpace)
        else:

            self.nPart = nPart

        self.nProc = nProc
        self.searchSpace = searchSpace
        self.objectives = objectives
        self.evalFn = evalFn
        self.particles = []
        self.backupName = backupName

        # Load the archive for bootstrapping if any
        if np.any(bootstrap):

            print('Restoring archives:')
```

```
241
242             self.restoredData = self.restoreBackup(bootstrap)
243
244             if bootstrap[0]:
245
246                 print('gBest')
247                 print('# of keys:', len(self.restoredData['gBest'].keys()))
248                 print('# of records:',
249                     len(next(iter(self.restoredData['gBest'].values()))))
250
251             if bootstrap[2]:
252
253                 print('particles')
254                 print('# of keys:', len(self.restoredData['particles'].keys()))
255                 print('# of records:',
256                     len(next(iter(self.restoredData['particles'].values()))))
257
258
259             self.hasPVBackup = []
260             self.hasSBackup = []
261             self.hasGPVBackup = []
262             self.hasGSBackup = []
263
264             if np.any(bootstrap[2:-2] or bootstrap[0]):
265
266                 for i in range(len(self.searchSpace)):
267
268                     self.hasPVBackup.append(False)
269                     self.hasGPVBackup.append(False)
270
271                     if np.any(bootstrap[2:-2]) and \
272                         self.searchSpace[i].name in self.restoredData['particles']:
273
274                         self.hasPVBackup[-1] = True
275
276                     if bootstrap[0] and \
277                         self.searchSpace[i].name in self.restoredData['gBest']:
278
279                         self.hasGPVBackup[-1] = True
280
281                 print('Recoverable position keys:')
282
283                 if bootstrap[0]:
284
285                     print('gBest:', self.hasGPVBackup)
286
287                 if np.any(bootstrap[2:-2]):
288
289                     print('particles:', self.hasPVBackup)
290
291
292             else:
293
294                 self.hasPVBackup = np.full((len(self.searchSpace),), False)
295                 self.hasGPVBackup = np.full((len(self.searchSpace),), False)
296
297             if np.any(bootstrap[-2] or np.all(bootstrap[:2])):
298
299                 for i in range(len(self.objectives)):
300
```

```
                    self.hasSBackup.append(False)
                    self.hasGSBackup.append(False)

                    if bootstrap[-2] and \
                        self.objectives[i].goalName + \
                        '(' + self.objectives[i].diag + ')' \
                        in self.restoredData['particles']:

                        self.hasSBackup[-1] = True

                    if np.all(bootstrap[:2]) and \
                        self.objectives[i].goalName + \
                            '(' + self.objectives[i].diag + ')' \
                            in self.restoredData['gBest']:

                        self.hasGSBackup[-1] = True

                print('Recoverable score keys:')

                if bootstrap[0]:

                    print('gBest:', self.hasGSBackup)

                if np.any(bootstrap[2:-2]):

                    print('particles:', self.hasSBackup)

            else:

                self.hasSBackup = np.full((len(self.objectives),), False)
                self.hasGSBackup = np.full((len(self.objectives),), False)

        self.epsilon = np.asarray(epsilon)

        # Instantiate the particles
        for i in range(self.nPart):

            self.particles.append(Particle(i, self, bootstrap))

        self.backupTarget = backupTarget
        self.backupType = backupType

        # Discriminate between single objective and multi objective optimization
        if (len(self.epsilon) == 0):

            print('SO population')

            self.gBestPos = self.particles[0].bestPos
            self.gBestScore = self.particles[0].bestScore

            if bootstrap[0]:

                for i in np.where(self.hasGPVBackup)[0]:

                    self.gBestPos[i] = self.restoredData['gBest'] \
                        [self.searchSpace[i].name][0]

                if bootstrap[1]:

                    for i in np.where(self.hasGSBackup)[0]:
```

```
361
362                     self.gBestScore = np.asarray(
363                         self.restoredData['gBest'] \
364                         [self.objectives[i].goalName + \
365                         '(' + self.objectives[i].diag + ')'] \
366                         [0]
367                         )
368
369         else:
370
371             print('MO population')
372
373             if bootstrap[0] and np.any(self.hasGPVBackup):
374
375                 pos = np.empty((0, len(self.searchSpace)))
376                 scores = np.empty((0, len(self.epsilon)))
377
378                 for i in range(len(next(iter(self.restoredData['gBest'].values())))):
379
380                     id = np.random.choice(len(self.particles))
381                     pos = np.append(pos,
382                         self.particles[id].bestPos.reshape((1, -1)),
383                         axis = 0
384                         )
385                     scores = np.append(scores,
386                         self.particles[id].score.reshape((1, -1)),
387                         axis = 0
388                         )
389
390                     for j in np.where(self.hasGPVBackup)[0]:
391
392                         pos[i,j] = self.restoredData['gBest']\
393                             [self.searchSpace[j].name][i]
394
395                     for j in np.where(self.hasGSBackup)[0]:
396
397                         scores[i,j] = self.restoredData['gBest']\
398                             [self.objectives[j].goalName + \
399                             '(' + self.objectives[j].diag + ')'][i]
400
401                 self.gBestPos = pos
402
403                 if np.any(self.hasGSBackup):
404
405                     minDist = np.linalg.norm(
406                         scores - (scores / self.epsilon).astype(int) * self.epsilon,
407                         axis = 1
408                         ).reshape((-1,1))
409                     self.gBestScore = np.append(
410                         np.append((scores / self.epsilon).astype(int),
411                         scores,
412                         axis = 1
413                         ), minDist, axis = 1)
414
415             else:
416
417                 self.gBestPos = np.asarray([self.particles[0].bestPos])
418                 self.gBestScore = np.asarray([[
419                     float('inf') for i in range(2 * len(epsilon) + 1)
420                     ]])
```

```
421
422             self.leaders = [particle.bestPos for particle in self.particles]
423
424     # Define the single objective particle update
425     def moveSO(self):
426
427         for particle in self.particles:
428
429             particle.updatePosition(self.gBestPos)
430
431     # Define the multi objective particle update
432     def moveMO(self):
433
434         for particleId in range(len(self.particles)):
435
436             self.particles[particleId].updatePosition(self.leaders[particleId])
437
438     # Alias for the particle evaluation
439     def eval(self, particle):
440
441         return particle.eval()
442
443     # Helper function that switches between single and multi objective optimizations
444     def optimize(self, nIter):
445
446         if (len(self.epsilon) == 0):
447
448             print('Starting Single-Objective Optimization')
449             self.optimizeSO(nIter)
450
451         elif (len(self.epsilon) > 0):
452
453             print('Starting Multi-Objective Optimization')
454             self.optimizeMO(nIter)
455
456     # Optimization function with a single objective
457     def optimizeSO(self, nIter):
458
459         print('Initial population:')
460         self.write()
461
462         history = []
463         x = []
464
465         fig = plt.gcf()
466         fig.show()
467         fig.canvas.draw()
468
469         if self.nProc > 1:
470
471             pool = mp.Pool(self.nProc)
472
473         for i in range(nIter):
474
475             t = time.time()
476
477             if self.nProc < 2:
478
479                 scores = [particle.eval() for particle in self.particles]
480
```

```
481                 for particle in self.particles:
482 
483                     particle.score = scores[particle.id]
484 
485             else:
486 
487                 scores = pool.map(self.eval,
488                     [particle  for particle in self.particles]
489                     )
490 
491                 for particle in self.particles:
492 
493                     particle.score = scores[particle.id]
494 
495             print('Last iteration duration:', time.time() - t)
496             scores = np.asarray(scores)
497 
498             for j in range(len(self.particles)):
499 
500                 print('Scores =', np.asarray([self.particles[j].bestScore,
501                     scores[j]])
502                     )
503 
504                 if(optTypeDict[self.objectives[0].optType][0](
505                     np.asarray([self.particles[j].bestScore, scores[j]])
506                     ) == scores [j]):
507 
508                     self.particles[j].updateBest()
509 
510             bestParticle = optTypeDict[self.objectives[0].optType][1](scores)
511 
512             if (optTypeDict[self.objectives[0].optType][0](
513                 np.asarray([self.gBestScore, scores[bestParticle]])
514                 ) == scores [bestParticle]):
515 
516                 self.gBestScore = scores[bestParticle]
517                 self.gBestPos = self.particles[bestParticle].pos
518 
519             if (self.gBestScore < self.objectives[0].tol):
520 
521                 print('Iter = ', i)
522                 self.write()
523                 break
524 
525             self.moveSO()
526 
527 
528 
529             x.append(i)
530             history.append(self.gBestScore)
531             plt.plot(x, history, 'b-')
532             plt.xlim([0,i])
533             fig.canvas.draw()
534 
535             print('Iter = ', i)
536             self.write()
537             print('gBestPos = ', self.gBestPos)
538             print('gBestScore = ', self.gBestScore)
539 
540         plt.show()
```

```
541
542     # Optimization function with multiple objectives
543     def optimizeMO(self, nIter):
544
545         print('Initial population:')
546         self.write()
547
548         fig = plt.gcf()
549         fig.show()
550         fig.canvas.draw()
551         iters = []
552         history = []
553
554         if self.nProc > 1:
555
556             pool = mp.Pool(self.nProc)
557
558         for i in range(nIter):
559
560             t = time.time()
561
562             if self.nProc < 2:
563
564                 scores = [particle.eval() for particle in self.particles]
565
566                 for particle in self.particles:
567
568                     particle.score = scores[particle.id]
569
570             else:
571
572                 scores = pool.map(
573                     self.eval, [particle  for particle in self.particles]
574                     )
575
576                 for particle in self.particles:
577
578                     particle.score = scores[particle.id]
579
580             print('Last iteration duration:', time.time() - t)
581
582             epsiCoord = (scores / self.epsilon).astype(int)
583 ## Needs modification to handle correctly a maximization...
584             epsiCornerDist = np.linalg.norm(
585                 scores - (scores / self.epsilon).astype(int) * self.epsilon, axis = 1
586                 )
587             hasLeader = np.full((len(self.particles), ), False)
588
589             for particleId in range(len(self.particles)):
590
591                 if np.any(np.isinf(scores[particleId])):
592
593                     break
594
595                 isChallenger = np.full((self.gBestScore.shape[0], ), True)
596                 isNonDominated = np.full((self.gBestScore.shape[0], ), False)
597
598                 for goalId in range(len(self.epsilon)):
599
600                     isChallenger = np.logical_and(isChallenger,
```

```
                        optTypeDict[self.objectives[goalId].optType][3](
                            epsiCoord[particleId, goalId], self.gBestScore[:, goalId]
                            ))
                    isNonDominated = np.logical_or(isNonDominated,
                        optTypeDict[self.objectives[goalId].optType][2](
                            epsiCoord[particleId, goalId], self.gBestScore[:, goalId]
                            ))

                if np.any(np.logical_or(isChallenger, isNonDominated)):

                    isUndefeated = np.logical_or(
                        np.logical_not(isChallenger),
                        np.logical_and(isChallenger, np.logical_not(isNonDominated)))

                    for championId in np.where(
                        np.logical_and(isChallenger, isUndefeated))[0]:

                        if np.less(epsiCornerDist[particleId],
                            self.gBestScore[championId, -1]):

                            isUndefeated[championId] = False


                    if np.any(np.logical_not(isUndefeated)):

                        self.gBestScore = self.gBestScore[isUndefeated]
                        self.gBestPos = self.gBestPos[isUndefeated]

                    if np.any(np.logical_or(
                        np.logical_not(isUndefeated), np.all(isNonDominated)
                        )):

                        self.gBestScore = np.append(self.gBestScore,
                            np.append(np.append(epsiCoord[particleId],
                            scores[particleId]),
                            epsiCornerDist[particleId]).reshape((1, -1)), axis = 0
                            )
                        self.gBestPos = np.append(self.gBestPos,
                            self.particles[particleId].pos.reshape((1, -1)),
                            axis = 0
                            )
                        self.particles[particleId].updateBest()
                        self.leaders[particleId] = self.particles[particleId].pos
                        hasLeader[particleId] = True

            for particleId in np.where(hasLeader)[0]:

                hasLeader[particleId] = np.any(
                    np.logical_not(
                        np.all(self.gBestPos - self.particles[particleId].pos,1)
                        )
                    )

            localDensity = np.ones((self.gBestScore.shape[0], ), dtype = int)

            if np.any(np.logical_not(hasLeader)):

                for scoreId1 in range(self.gBestScore.shape[0]):

                    for scoreId2 in range(scoreId1 + 1, self.gBestScore.shape[0]):
```

```
660
661                     for goalId in range(len(self.epsilon)):
662
663                         if (
664                             (self.gBestScore[scoreId1, goalId] + 2 >= \
665                                 self.gBestScore[scoreId2, goalId])
666                             and (self.gBestScore[scoreId1, goalId] - 2 <= \
667                                 self.gBestScore[scoreId2, goalId])):
668
669                             localDensity[scoreId1] += 1
670                             localDensity[scoreId2] += 1
671                             break
672
673             invDensity = 1. / localDensity
674             totalDensity = np.sum(invDensity)
675             probDistribution = invDensity / totalDensity
676             leaderIds = np.random.choice(self.gBestScore.shape[0],
677                                          len(self.particles),
678                                          p = probDistribution)
679
680             for particleId in np.where(np.logical_not(hasLeader))[0]:
681
682                 self.leaders[particleId] = self.gBestPos[leaderIds[particleId]]
683
684         self.moveMO()
685         dist = [np.linalg.norm(
686                 score[
687                     len(self.epsilon):2*len(self.epsilon)
688                     ]
689                 ) for score in self.gBestScore[:]
690             ]
691         minDist = np.min(dist)
692         iters.append(i)
693         history.append(minDist)
694         fig.clf()
695
696         if (len(self.epsilon) > 2):
697
698             plt.subplot(121)
699             plt.plot(iters, history, 'b-')
700             plt.xlim([0, i])
701
702             plt.subplot(122)
703             plt.plot(range(len(localDensity)), localDensity, 'r-')
704             fig.canvas.draw()
705
706         elif (len(self.epsilon) == 2):
707
708             plt.subplot(221)
709             plt.plot(iters, history, 'b-')
710             plt.xlim([0, i])
711
712             plt.subplot(222)
713             plt.plot(self.gBestScore[:,2], self.gBestScore[:,3], 'b*')
714
715             plt.subplot(223)
716             plt.plot(range(len(probDistribution)), probDistribution, 'r-')
717             fig.canvas.draw()
718
719         print('Iter = ', i)
```

```
            self.write()
            print('Shortest distance to corner = ', minDist)
            self.archiveMO(i)
            print('Archive size = {}'.format(
                self.gBestScore.shape[0]
                ))
            print('Solution closest to corner:',
                self.gBestPos[np.asarray(dist).argmin()]
                )
            print('Scores closest to corner:',
                self.gBestScore[np.asarray(dist).argmin()] \
                    [len(self.epsilon): 2 * len(self.epsilon)]
                )
            print('Epsilon coordinates closest to corner:',
                self.gBestScore[np.asarray(dist).argmin()][:len(self.epsilon)]
                )

            print('\r\n')

        plt.show()

    # function that archives the Pareto front in multi objective optimization
    def archiveMO(self, i):

        if (self.backupTarget != 'False'):

            if ((self.backupTarget == 'Best') or (self.backupTarget == 'All')):

                with open(self.backupName + '_gBest.dat',
                    'w' if self.backupType == 'Last' else 'a') as archive:

                    archive.write('Iteration #{}\n'.format(i))
                    line = ''

                    for param in self.searchSpace:
                        line += param.name + ','
                    for goal in self.objectives:
                        line += goal.goalName + '(' + goal.diag + '),'

                    line = line[:-1]
                    line += '\n'
                    archive.write(line)

                    np.savetxt(archive, np.append(self.gBestPos,
                        self.gBestScore[:, len(self.epsilon):2 * len(self.epsilon)],
                        axis = 1
                        ),
                        delimiter = ','
                    )

            if ((self.backupTarget == 'Particles') or (self.backupTarget == 'All')):

                with open(self.backupName + '_particles.dat',
                    'w' if self.backupType == 'Last' else 'a') as archive:

                    archive.write('Iteration #{}\n'.format(i))
                    line = ''

                    for param in self.searchSpace:
                        line += param.name + ','
```

```
                    for param in self.searchSpace:
                        line += param.name + '_v,'
                    for goal in self.objectives:
                        line += goal.goalName + '(' + goal.diag + '),'
                    for param in self.searchSpace:
                        line += param.name + '_best,'
                    for goal in self.objectives:
                        line += goal.goalName + '(' + goal.diag + ')_best,'

                    line = line[:-1]
                    line += '\n'
                    archive.write(line)

                    for particle in self.particles:

                        line = ''

                        for coord in particle.pos:

                            line += str(coord) + ','

                        for vel in particle.vel:

                            line += str(vel) + ','

                        for score in particle.score:

                            line += str(score) + ','

                        for coord in particle.bestPos:

                            line += str(coord) + ','


                        for score in particle.bestScore:

                            line += str(score) + ','

                        line = line[:-1]
                        line += '\n'
                        archive.write(line)

    # function that loads back ups
    def restoreBackup(self, target):

        masterDict = {}

        if np.any(target[:2]):

            if os.path.isfile(self.backupName + '_gBest.dat'):

                with open(self.backupName + '_gBest.dat', 'r') as file:

                    file.readline()
                    keys = file.readline().strip().split(',')
                    data = np.transpose(
                        np.asarray([[
                            float(value) for value in line.split(',')
                            ] for line in file])
                        )
```

```
840                     masterDict['gBest'] = {k: v for (k, v) in zip(keys, data)}
841
842             else:
843
844                 print('No gBest backup to restore, \
845                     switching to standard gBest initialization routine.')
846
847         if np.any(target[2:]):
848
849             if os.path.isfile(self.backupName + '_particles.dat'):
850
851                 with open(self.backupName + '_particles.dat', 'r') as file:
852
853                     file.readline()
854                     keys = file.readline().strip().split(',')
855                     data = np.transpose(
856                         np.asarray(
857                                 [[
858                                 float(value) for value in line.split(',')
859                                 ] for line in file]
860                             )
861                         )
862                     masterDict['particles'] = {k: v for (k, v) in zip(keys, data)}
863
864             else:
865
866                 print('No particle backup to restore, \
867                     switching to standard particle initialization routine.')
868
869         return masterDict
870
871     # function that print the population parameters
872     def write(self):
873
874         print('Population summary')
875         print('# of particles: ', self.nPart)
876         print('Social weight: ', socialWeight)
877         print('Personal weight: ', personalWeight)
878         print('Inertial Weight: ', inertialWeight)
879
880
```

A.2.2 PSO2LEBT.py

This file called *PSO2LEBT.py* sets the search space, the objectives and the evaluation function required to perform a PSO on the MYRRHA LEBT. In addition, it maintains the necessary archives.

The search space defines the LEBT parameters to optimize and their boundaries.

- LBE_SOL1
  This parameter is the current to apply to the first solenoid I_1^{sol} in amperes and is limited to the range explored during the commissioning of the machine (between 50 A and 110 A).

- LBE_SOL2
  This parameter is the current to apply to the second solenoid I_2^{sol} in amperes and is limited to the range explored during the commissioning of the machine (between 50 A and 110 A).

- LBE_D4
  This parameter is the current to apply to the horizontal steerer installed in the second solenoid I_{2H}^{st} in amperes and is limited to the range explored during the commissioning of the machine (between -0.5 A and 2.5 A).

- LBE_D4
  This parameter is the current to apply to the vertical steerer installed in the second solenoid I_{2V}^{st} in amperes and is limited to the range explored during the commissioning of the machine (between -2 A and 1 A).

In the example shown below, two objectives are defined.

- A targeted beam current
  The first objective is to obtain a beam current at the exit of the LEBT of about 4.2 mA. Here, the score associated with this objective is the square of the difference between the target beam current and the beam current obtained with the current candidate solution:

$$S_1 = (\overline{I^b_{LEBT,out,meas}} - I^b_{target})^2.$$

- A setpoint constraint
  The second objective imposes a constraint on the set points of the candidate solution. Here, the goal is to minimize the current consumption during operation (to illustrate the concern on minimizing operation costs). The associated score is given by:

$$S_2 = I_1^{sol} + I_2^{sol} + (I_{2H}^{st})^2 + (I_{2V}^{st})^2.$$

```python
import MYRRHA_LEBT_epics
from MOPSOv2 import Population
import numpy as np
import time
from epics import caget

# Instantiate the EPICS controller for the MYRRHA LEBT
con = MYRRHA_LEBT_epics.Controller('log.dat')

# Set the target beam current
tgtCurrent = 4.2

# Link the MOPSO algorithm to the MYRRHA LEBT
def modelLEBT(part):

    # Apply the candidate solution to the MYRRHA LEBT
    con.setMultiplePVs(['LBE_SOL1','LBE_SOL2','LBE_D3','LBE_D4'],part.pos)

    # Create array of measurements
    measure = []
    # Measure the beam current 10 times before averaging
    for _ in range(10):
        tmp = caget('MYRRHA_DAQ:channel2')
        idx = int(len(tmp)/2)
        currMeasured = tmp[idx] * 5
        while(currMeasured > 100.):
            idx += 1
            currMeasured = tmp[idx] * 5
        measure.append(currMeasured)
        time.sleep(0.02)
    avMeasure = np.average(measure)

    # Evaluate the candidate solution
    # Distance to target beam current
    dist2targetCurrent = (avMeasure - tgtCurrent) ** 2

    # Distance to design steerers set points (0)
    dist2minD3 = part.pos[2] ** 2
    dist2minD4 = part.pos[3] ** 2

    # Compile the scores
    scores = [
        dist2targetCurrent,
        part.pos[0] + part.pos[1] + dist2minD3 + dist2minD4
        ]

    # Append the candidate solution and its evaluation to the archive
    with open('PSOhistory.dat','a') as f:
        f.write('Particle {} position and scores :\t'.format(part.id))
        for setpoint in part.pos:
            f.write(str(setpoint) + '\t')
        for score in scores:
            f.write(str(score) + '\t')
        f.write('\n')


    return scores

# definition of the SearchSpace object that contains the parameters to optimize
# and the boundaries of the search
```

```
class SearchSpace:
    def __init__(self, name = '', range = []):
        self.name = name
        self.range = range

# define the parameters to optimize and their boundaries for the MYRRHA LEBT
params = [
    SearchSpace('Sol1', range = [50.0, 110.0]),
    # SearchSpace('Steer1V', range = [-1.0, 1.0]),
    # SearchSpace('Steer1H', range = [-1.0, 1.0]),
    # SearchSpace('Coll', range = [-1.0, 1.0]),
    SearchSpace('Sol2', range = [50.0, 110.0]),
    SearchSpace('Steer2H', range = [-0.5, 2.5]),
    SearchSpace('Steer2V', range = [-2.0, 1.0])
    ]

# Definition of the Goal object that contains the name of the objective,
# the "diagnostic" used to evaluate the objective, the type of optimization
# and the tolerance to reach to meet a stopping criterion
class Goal:
    def __init__(self, name = '', optType = 'Min', tol = 0.):
        self.goalName = name
        self.diag = name
        self.optType = optType
        self.tol = tol

# define the objective for the MYRRHA LEBT
goals = [
    Goal('LBE_Trans'),
    Goal('Config')
    ]

# Set the options for the optimization
bootstrap = [False, False, False, False, False, False]
backupTarget = 'All'
backupType = 'Last'
nProc = 0

epsilons = [0.1, 0.1]

if __name__ == '__main__':
    # Instantiate the population
    pop = Population(params, goals, modelLEBT, epsilon = epsilons,
        bootstrap = bootstrap, backupTarget = backupTarget,
        backupType = backupType, nProc = nProc)

    # Create an archive
    with open('PSOhistory.dat','w') as f:
        for elem in params:
            f.write(elem.name + '\t')
        for goal in goals:
            f.write(goal.goalName + '\t')
        f.write('\n')

    t = time.time()
    # Start the optimization for 10 iterations
    pop.optimize(10)

    # Archive the Pareto front determined during the optimization
    with open('PSOhistory.dat','a') as f:
```

```
        f.write('Population final bests :\n')
        for setpoints, scores in zip(pop.gBestPos, pop.gBestScore):
            for setpoint in setpoints:
                f.write(str(setpoint) + '\t')
            for score in scores:
                f.write(str(score) + '\t')
        f.write('\n')

    print('total execution time = {}'.format(time.time() - t))
```

A.2.3 PSO2model

This file called *PSO2model.py* sets the search space, the objectives and the evaluation function required to perform a PSO on a neural network model of the MYRRHA LEBT. In addition, it maintains the necessary archives.

The search space defines the inputs of the LEBT model to optimize and their boundaries.

- Sol1
  This parameter is the current to apply to the first solenoid I_1^{sol} in amperes and is limited to the range the model has been trained on (between 50 A and 110 A).

- Sol2
  This parameter is the current to apply to the second solenoid I_2^{sol} in amperes and is limited to the range the model has been trained on (between 50 A and 110 A).

- Steer2H
  This parameter is the current to apply to the horizontal steerer installed in the second solenoid I_{2H}^{st} in amperes and is limited to the range the model has been trained on (between -3 A and 3 A).

- Steer2V
  This parameter is the current to apply to the vertical steerer installed in the second solenoid I_{2V}^{st} in amperes and is limited to the range the model has been trained on (between -3 A and 3 A).

- Coll
  This parameter is the opening of the collimator installed between the solenoids r_{col} in millimeters and is limited to the range the model has been trained on (between 3.75 mm and 51.23 mm

In the example shown below, two objectives are defined.

- A targeted beam current
  The first objective is to obtain a beam current at the exit of the LEBT of about 4.2 mA. Here, the score associated with this objective is the square of the difference between the target beam current and the beam current obtained with the current candidate solution:

$$S_1 = (I^b_{LEBT,out,pred} - I^b_{target})^2.$$

- A setpoint constraint
  The second objective imposes a constraint on the set points of the candidate solution. Here, the goal is to find an optimized configuration close to the nominal

configuration (I_1^{sol} = 65.6 A, I_2^{sol} = 77.9 A, I_{2H}^{st} = 0.75 A, I_{2V}^{st} = −0.5 A, r_{col} = 65.6 A). The associated score is given by:

$$S_2 = (I_1^{sol} - 65.6)^2 + (I_2^{sol} - 77.9)^2 + (I_{2H}^{st} - 0.75)^2 + (I_{2V}^{st} - 0.5)^2.$$

```python
from MOPSOv2 import Population
import tensorflow as tf
import numpy as np
import time
from epics import caget

# Loads the model
model = tf.keras.models.load_model('model69.h5')

# Set the target current
tgtCurrent = 4.2

# Scaling function for the model features
def scaleFeature(featureDict):

    # Sol1 is [50, 110]
    featureDict['Sol1'] = (featureDict['Sol1'] - 80.) / 30.
    # Sol2 is [50, 110]
    featureDict['Sol2'] = (featureDict['Sol2'] - 80.) / 30.
    # Steer1V is [-3, 3]
    featureDict['Steer1V'] = featureDict['Steer1V'] / 3.
    # Steer1H is [-3, 3]
    featureDict['Steer1H'] = featureDict['Steer1H'] / 3.
    # Steer2V is [-3, 3]
    featureDict['Steer2V'] = featureDict['Steer2V'] / 3.
    # Steer2H is [-3, 3]
    featureDict['Steer2H'] = featureDict['Steer2H'] / 3.
    #SlitHminus is [0, 60]
    featureDict['SlitHminus'] = (featureDict['SlitHminus'] - 30.) / 30.
    #SlitHplus is [0, 60]
    featureDict['SlitHplus'] = (featureDict['SlitHplus'] - 30.) / 30.
    #SlitVminus is [0, 60]
    featureDict['SlitVminus'] = (featureDict['SlitVminus'] - 30.) / 30.
    #SlitVplus is [0, 60]
    featureDict['SlitHplus'] = (featureDict['SlitHplus'] - 30.) / 30.
    # Pres1 is [0, 5e-5]
    featureDict['Pres1'] = (featureDict['Pres1'] - 2.5e-5) / 5.0e-5
    # Pres2 is [0, 5e-5]
    featureDict['Pres2'] = (featureDict['Pres2'] - 2.5e-5) / 5.0e-5
    # Pres3 is [0, 5e-5]
    featureDict['Pres3'] = (featureDict['Pres3'] - 2.5e-5) / 5.0e-5

    return featureDict

# Links the MOPSO algorithm to the model for the MYRRHA LEBT
def model(part):

    print(['LBE_SOL1', 'LBE_SOL2','LBE_D3','LBE_D4', 'LBE_Coll'],part.pos)

    # Translate the candidate solution to the model input
    config = {
        'Sol1' : np.asarray(part.pos[0]).reshape((-1,)),
        'Sol2' : np.asarray(part.pos[1]).reshape((-1,)),
        'Steer1V'  : np.asarray(0.0).reshape((-1,)),
        'Steer1H' : np.asarray(0.0).reshape((-1,)),
        'Steer2V' : np.asarray(part.pos[3]).reshape((-1,)),
        'Steer2H' : np.asarray(part.pos[2]).reshape((-1,)),
        'SlitHminus'  : np.asarray(part.pos[4] - 3.75).reshape((-1,)),
        'SlitHplus'  : np.asarray(part.pos[4] + 3.75).reshape((-1,)),
        'SlitVminus'  : np.asarray(part.pos[4] + 3.75).reshape((-1,)),
```

```
        'SlitVplus'  : np.asarray(part.pos[4] - 3.75).reshape((-1,)),
        'Pres1' : np.asarray(2e-5).reshape((-1,)),
        'Pres2' : np.asarray(1.6e-5).reshape((-1,)),
        'Pres3' : np.asarray(1e-5).reshape((-1,))
        }
    config = scaleFeature(config)

    # Apply the input to the model
    modelCurrent = model.predict(config) * 10.

    # Evaluate the candidate solution
    # Distance to target beam current
    dist2targetCurrent = (modelCurrent - tgtCurrent) ** 2
    print(modelCurrent)

    # Distance to the nominal configuration
    dist2MinSol1 = (part.pos[0] - 65.6) ** 2
    dist2MinSol2 = (part.pos[1] - 77.9) ** 2
    dist2minD3 = (part.pos[2] - 0.75) ** 2
    dist2minD4 = (part.pos[3] + 0.5 ) ** 2

    # Compile the scores
    scores = [
        dist2targetCurrent[0][0],
        dist2MinSol1 + dist2MinSol2 + dist2minD3 + dist2minD4 - part.pos[4]
        ]
    print('Scores : ', scores)

    # Append the candidate solution and its evaluation to the archive
    with open('PSOhistoryModel2.dat','a') as f:
        f.write('Particle {} position and scores :\t'.format(part.id))
        for setpoint in part.pos:
            f.write(str(setpoint) + '\t')
        for score in scores:
            f.write(str(score) + '\t')
        f.write(str(modelCurrent))
        f.write('\n')

    return scores

# definition of the SearchSpace object that contains the parameters to optimize
# and the boundaries of the search
class SearchSpace:

    def __init__(self, name = '', range = []):

        self.name = name
        self.range = range

# define the parameters to optimize and their boundaries
# for the model of the MYRRHA LEBT
params = [
    SearchSpace('Sol1', range = [50.0, 110.0]),
    # SearchSpace('Steer1V', range = [-1.0, 1.0]),
    # SearchSpace('Steer1H', range = [-1.0, 1.0]),
    SearchSpace('Sol2', range = [50.0, 110.0]),
    SearchSpace('Steer2H', range = [-3.0, 3.0]),
    SearchSpace('Steer2V', range = [-3.0, 3.0]),
    SearchSpace('Coll', range = [3.75, 51.25])
    ]
```

```
121
122 # Definition of the Goal object that contains the name of the objective,
123 # the "diagnostic" used to evaluate the objective, the type of optimization
124 # and the tolerance to reach to meet a stopping criterion
125 class Goal:
126
127     def __init__(self, name = '', optType = 'Min', tol = 0.):
128
129         self.goalName = name
130         self.diag = name
131         self.optType = optType
132         self.tol = tol
133
134 # define the objective for the model of the MYRRHA LEBT
135 goals = [
136     Goal('LBE_Trans'),
137     Goal('Config')
138     ]
139
140 # Set the options for the optimization
141 bootstrap = [False, False, False, False, False, False]
142 backupTarget = 'All'
143 backupType = 'Last'
144 nProc = 0
145
146 # epsilons = []
147 epsilons = [0.1, 1.]
148
149 if __name__ == '__main__':
150     # Instantiate the population
151     pop = Population(params, goals, model, epsilon = epsilons,
152         bootstrap = bootstrap, backupTarget = backupTarget,
153         backupType = backupType, nProc = nProc)
154
155     # Create an archive
156     with open('PSOhistoryModel.dat','w') as f:
157
158         for elem in params:
159             f.write(elem.name + '\t')
160
161         for goal in goals:
162             f.write(goal.goalName + '\t')
163
164         f.write('\n')
165
166     t = time.time()
167     # Start the optimization for 200 iterations
168     pop.optimize(200)
169
170     # Archive the Pareto front determined during the optimization
171     with open('PSOhistoryModel.dat','a') as f:
172
173         f.write('Population final bests :\n')
174         print(pop.gBestScore.shape)
175         print(pop.gBestPos.shape)
176         configs = {
177             'Sol1' : np.asarray(pop.gBestPos[:, 0]),
178             'Sol2' : np.asarray(pop.gBestPos[:, 1]),
179             'Steer1V'  : np.repeat(0.0, pop.gBestPos.shape[0]),
180             'Steer1H' : np.repeat(0.0, pop.gBestPos.shape[0]),
```

```
            'Steer2V' : np.asarray(pop.gBestPos[:, 3]),
            'Steer2H' : np.asarray(pop.gBestPos[:, 2]),
            'SlitHminus'  : np.asarray(pop.gBestPos[:, 4] - 3.75),
            'SlitHplus'  : np.asarray(pop.gBestPos[:, 4] + 3.75),
            'SlitVminus'  : np.asarray(pop.gBestPos[:, 4] + 3.75),
            'SlitVplus'  : np.asarray(pop.gBestPos[:, 4] - 3.75),
            'Pres1' : np.repeat(2e-5, pop.gBestPos.shape[0]),
            'Pres2' : np.repeat(1.6e-5, pop.gBestPos.shape[0]),
            'Pres3' : np.repeat(1e-5, pop.gBestPos.shape[0])
            }
        configs = scaleFeature(configs)
        modelCurrents = np.asarray(model.predict(configs))
        for setpoints, scores, current in zip(
            pop.gBestPos, pop.gBestScore, modelCurrents
            ):
            for setpoint in setpoints:
                f.write(str(setpoint) + '\t')
            for score in scores:
                f.write(str(score) + '\t')
            f.write(str(current[0]))
            f.write('\n')

    print('total execution time = {}'.format(time.time() - t))
```

Appendix B

Résumé des chapitres en français

B.1 Introduction

De nos jours, il y a une tendance générale à l'augmentation des performances pour les projets d'accélérateurs de particules. Afin de satisfaire ces exigences, qu'elles soient d'origine académique ou industrielle, les futurs accélérateurs doivent atteindre des énergies moyennes, des fiabilités et des stabilités plus élévées. En particulier, les projets d'ADS ("Accelerator Driven System"), dont le but est de piloter un réacteur nucléaire à l'aide d'un accélérateur de particules. Ces projets nécessitent de construire des accélérateurs de protons extrêmement fiables dont la puissance atteint quelques mégawatts. Ceci afin de permettre l'incinération efficace de déchets nucléaires. C'est le cas du projet MYRRHA (Multi-purpose hYbrid Research Reactor for High-tech Applications) porté par le Centre de recherche sur l'Energie Nucléaire (SCK • CEN) en Belgique. Dans ce projet, l'objectif est la construction d'un accélérateur capable de fournir un courant continu de protons de 4 mA accélérés à 600 MeV pour piloter un réacteur sous-critique d'environ 100 MW_{th}. Avec ces spécifications, le faisceau atteindra une puissance de 2.4 MW. Afin d'atteindre cet objectif, le design et la construction d'un accélérateur linéaire (linac) supraconducteur est actuellement en cours. En plus de la puissance à atteindre, il est nécéssaire que cet accélérateur soit extrêmement fiable car il devra pouvoir fonctionner lors de cycles de 3 mois avec moins de 10 arrêts du faisceau plus long que 3 secondes. En effet, des arrêts du faisceau trop longs et trop répétés peuvent causer des contraintes thermiques et donc endommager la structure du réacteur. Cependant, ce niveau de fiabilité n'a encore jamais été atteint. C'est pourquoi, le design de l'accélérateur doit être suffisament robuste, flexible et modulaire afin de permettre la compensation d'éventuelles pannes avec des arrêts du faisceau minimaux.

Un des points clefs pour atteindre cet objectif est de garantir un bon réglage de l'injecteur composé d'une LEBT ("Low energy Beam Transport line") et d'un RFQ ("Radio-Frequency Quadrupole"). Le rôle de la LEBT est de guider et focaliser le faisceau depuis la source d'ions jusqu'au RFQ, le premier élément accélérateur. En effet, le rôle de la LEBT notamment est crucial pour obtenir un faisceau de bonne qualité qui pourra être transporté le long du linac tout en minimisant les pertes de

faisceaux qui peuvent causer l'arrêt de la machine lorsqu'elles deviennent trop importantes. Cependant, dans un injecteur, la dynamique du faisceau est complexe et fortement influencée par les effets de charges d'espaces et leur compensation. De plus, il est nécessaire de pouvoir changer rapidement la configuration de l'injecteur dans une large gamme de configurations (bas courant et cycle utile pour le démarrage jusqu'au courant nominal avec un cycle utile de 100 %).

Dans ce contexte, cette thèse explore la possibilité d'utiliser des méthodes d'apprentissage automatique ("Machine Learning") pour le développement d'un modèle numérique capable de reproduire fidèlement le comportement expérimental d'un injecteur et plus spécifiquement d'une LEBT. En particulier, ce manuscrit décrit le travail de recherche entrepris depuis 2018 au Laboratoire de Physique Subatomique et de Cosmologie à Grenoble sur l'entraînement de réseaux de neurones artificiels par apprentissage supervisé avec pour objectif de modéliser l'injecteur de MYRRHA ainsi que celui de IPHI (Injecteur de Protons à Haute Intensité, CEA Saclay).

Le manuscrit est organisé comme suit :

- Le premier chapitre est dédié à un aperçu des technologies d'accélérateurs de particules, à la description du projet MYRRHA et à l'introduction des concepts de base de la dynamique des faisceaux de particules chargées.
- Le second chapitre est une introduction des réseaux de neurones, de l'apprentissage supervisé et d'un algorithme d'optimisation appelé Optimisation par Nuées de Particules (PSO). Une courte revue de la littérature est également donnée sur l'utilisation des réseaux de neurones et de PSO dans le domaine des accélérateurs de particules.
- Le troisième chapitre est consacré à la description des travaux expérimentaux réalisés dans le cadre de cette thèse sur les injecteurs de MYRRHA et IPHI. De plus, un test de PSO sur la LEBT de MYRRHA y est discuté.
- Le quatrième chapitre est une description des bases de données expérimentales et simulées rassemblées afin d'entraîner des réseaux de neurones pour modéliser les injecteurs de MYRRHA et IPHI ainsi que pour évaluer leurs performances sur ces tâches. Il contient également une description du processus utilisé pour déterminer des hyper-paramètres raisonnables pour les réseaux et l'apprentissage supervisé.
- Finalement, l'entraînement et l'évaluation de réseaux de neurones sont discutés dans le cinquième chapitre.

B.2 Linacs de hadrons de haute énergie: performances, fiabilité et principe d'un ADS

Il y a une demande croissante d'accélérateurs de haute puissance afin de mieux supporter divers domaines scientifiques tels que la physique des particules, la physique nucléaire ou les physiques basées sur les neutrons. Ces applications demandent typiquement des faisceaux dont l'énergie moyenne est très élevée ("classe MégaWatt") ce qui va significativement au-delà des capacités de la plupart des installations qui existent actuellement: l'énergie du faisceau requise est dans la gamme du MeV au GeV alors que le courrant crête du faisceau varie de 1 jusqu'à des centaines de mA.

De plus, la fiabilité est un challenge majeur parmi les perspectives d'amélioration des performances et de la soutenabilité de ces accélérateurs de classe MégaWatt. En effet, les contraintes sur la disponibilité du faisceau deviennent de plus en plus difficiles à remplir. Ces contraintes sont encore plus strictes dans le cas d'un réacteur nucléaire piloté par un accélérateur de particules (ADS). Par exemple, pour le projet de démonstration d'un ADS, MYRRHA, l'opération de l'accélérateur doit générer moins de 10 arrêts non-programmés du faisceau pour chaque cycle d'opération de 3 mois.

Ainsi, afin de minimiser la consommation globale d'énergie et donc les coûts d'opération, l'utilisation de cavités accélératrices Radio-Fréquence Supraconductrice (SRF) devient obligatoire. Ceci est particulièrement vrai pour des machines qui nécessitent d'opérer des faisceaux continu (CW) ou à haut cycle utile. L'architecture de la plupart de ces nouvelles machines est donc basée sur des accélérateurs linéaires supraconducteurs (linacs SC).

L'expérience a montré que le taux de pannes des linacs de haute puissance est dominé par la fiabilité des unité RF en séries ainsi que par l'injecteur où le faisceau est produit, mis en paquets et pré-accéléré. Il est donc nécessaire de concevoir des linacs avec des éléments redondants capables de survivre à un certain nombre de pannes sans interruptions significatives du faisceau. Mais il est également très important d'améliorer les méthodes de réglage de l'injecteur afin d'éviter la formation de halos et les pertes de faisceau dans les sections de haute énergie des accélérateurs. De plus, l'opération de l'injecteur dans divers modes (cycle de service et courant faisceau variables) est souvent requise de la mise en marche jusqu'aux opérations par les utilisateurs et ces modes sont déterminés par le réglage de l'injecteur. Dès lors, améliorer le processus de réglage des injecteurs augmente également la disponibilité du faisceau en réduisant le temps requis pour passer d'un mode à l'autre.

Le projet MYRRHA

L'objectif d'un ADS est d'utiliser un accélérateur pour fournir des neutrons à l'aide du processus de spallation à l'intérieur d'un réacteur nucléaire. Cette approche permet de concevoir un réacteur sous-critique dépendant d'une source externe de neutrons pour maintenir la réaction en chaîne. Avec ce type de réacteurs, il est possible d'utiliser une large gamme de combustibles y compris des actinides mineurs permettant ainsi

de les incinérer. Afin d'en démontrer la faisabilité, le projet MYRRHA est basé sur l'utilisation d'un faisceau de protons CW de 2.4 MW (4 mA, 600 MeV) pour contrôler un réacteur de 100 MW_{th}. Afin de satisfaire à la contrainte sur la fiabilité, la conception de l'accélérateur est basée sur l'application d'une stratégie de tolérance aux pannes utilisant des redondances. En particulier, l'accélérateur est conçu avec deux injecteurs identiques afin d'assurer une redondance en parallèle à basse énergie. Lorsqu'une panne survient dans l'injecteur en service, le second injecteur en veille prendrait le relais pour fournir le faisceau au linac SC principal. Pour le linac SC, le design impose une opération nominale avec une marge de sécurité sur la puissance des différents éléments. Par conséquent, lors d'une panne, les cavités, cryomodules ou quadrupoles voisins peuvent prendre le relais pour celui en panne. Une des difficultés liées à cette stratégie est de connaître une configuration compensée pour chaque panne restaurable. Ceci nécessite de constituer une banque de données avec soit des pannes compensées précédemment soit des configurations prédites à l'aide de simulations.

Objectifs de la thèse

Durant l'opération de tout accélérateur, il est important de minimiser les pertes de faisceau. En effet, les particules perdues le long de la l'accélérateur vont heurter les parois de celui-ci et l'endommager. Ces dégâts peuvent s'accumuler avec le temps et causer des problèmes sévères empêchant l'opération de l'accélérateur jusqu'à sa réparation. Ceci est particulièrement problématique pour les accélérateurs de haute puissance moyenne et cycle de service élevé tel que l'accélérateur de MYRRHA.

Une des façons de minimiser ces pertes est de garantir la qualité du faisceau à accélérer. C'est particulièrement important pour l'accélérateur de MYRRHA car la stratégie de tolérance aux fautes impose l'opération de l'accélérateur dans de nombreuses configurations différentes. La configuration de la LEBT est cruciale pour le faisceau car son rôle est de guider, nettoyer et focaliser le faisceau depuis la source de particules jusqu'au RFQ, le premier élément accélérateur. Cependant, l'optimisation de la configuration de la LEBT est compliquée. En effet, les propriétés du faisceau à la sortie de la source sont généralement mal connues, la basse énergie du faisceau le rend susceptible aux effets de charges d'espace et le désalignement des différents éléments doit être compensé. De plus, il est nécessaire de pouvoir changer rapidement la configuration de l'injecteur.

Dans ce contexte, la thèse a pour objectif d'explorer de nouvelles méthodes pour l'optimisation et la modélisation d'accélérateurs de particules. En particulier, la raison principale de ce travail est d'étudier la possibilité d'utiliser des méthodes d'apprentissage automatique pour développer un modèle numérique capable de reproduire le comportement expérimental d'un injecteur. Pour se faire, des modèles basés sur les réseaux de neurones ont été entrainés pour reproduire le comportement de deux machines : la LEBT de MYRRHA et l'injecteur de IPHI. Les réseaux ont été entraînés à l'aide de jeux de données simulés et expérimentaux afin d'obtenir la plus haute fidélité possible à leurs machines respectives.

B.3 Intelligence artificielle, apprentissage automatique et accélérateurs de particules

L'apprentissage automatique est un ensemble de paradigmes dont l'objectif est d'ajuster les paramètres d'un modèle afin qu'il représente au mieux un jeu de données. Dans cette thèse, le modèle en question est un réseau de neurones et le paradigme utilisé pour l'ajuster est l'apprentissage supervisé.

Réseau de neurones

Comme son nom l'indique, un réseau de neurones est composé de plusieurs éléments simples, des neurones artificiels. Le fonctionnement de ces neurones s'inspire du fonctionnement de vrais neurones biologiques. Dans un cerveau, des stimuli sont reçus par chaque neurone. Ils prennent la forme de neurotransmetteurs qui s'attachent aux récepteurs présents sur la paroi de la cellule et causent une modification du potentiel électrique. Si le potentiel dépasse un seuil d'excitation, le neurone est activé et libère des neurotransmetteurs pour stimuler d'autres neurones. De manière similaire, un neurone artificiel reçoit un vecteur de nombres réels en tant qu'entrée $\mathbf{X} = (x_0, x_1, ..., x_j)$ ce qui représente les stimuli. Une somme pondérée est ensuite appliquée sur ces entrées avec un vecteur de nombres réels $\mathbf{W} = (w_0, w_1, ..., w_j)$ afin de représenter l'importance relative de chaque stimulus. Puis, un biais est appliqué sur le résultat de la somme pour représenter le seuil d'excitation. Le résultat de cette dernière opération donne l'excitation finale z du neurone :

$$z = \sum_j w_j x_j + b. \tag{B.1}$$

Finalement, une fonction d'activation $f(z)$ est appliquée à cette excitation finale et son résultat correspond à la sortie du neurone. L'effet du neurone sur un ensemble d'entrées s'écrit :

$$f(z) = f(\mathbf{W} \cdot \mathbf{X} + b) = a. \tag{B.2}$$

La sortie d'un neurone (appelée *activation*) pour une entrée donnée est déterminée par la fonction d'activation choisie et la valeur des poids et du biais.

Un réseau de neurones consiste à arranger plusieurs neurones en plusieurs couches et à utiliser la sortie des neurones d'une couche comme entrées pour les neurones de la couche suivante. La première couche est la *couche d'entrée*, la dernière couche est la *couche de sortie* et toutes les couches intermédiaires sont appelées *couches cachées*.

Le choix de la fonction d'activation, du nombre de couches cachées et du nombre de neurones par couche sont appelés *hyper-paramètres*. Ces derniers conditionnent la capacité du réseau à modéliser une fonction. En effet, plus il y a de couches cachées et de neurones par couche, plus le réseau de neurones est capable d'avoir une sortie détaillée. Cependant, plus le réseau est grand, plus il devient délicat de l'entraîner. Il est donc nécessaire de trouver un compromis pour les hyper-paramètres.

Apprentissage supervisé

Le principe de l'apprentissage supervisé est d'apprendre la fonction h qui lie un espace d'entrée X à un espace de sortie Y :

$$h(x \in X) = y \in Y. \tag{B.3}$$

Pour un réseau de neurones, cela correspond à trouver un ensemble de poids et biais tel que l'erreur entre la prédictions $\hat{y}_i$ du réseau pour une entrée donnée x_i et la sortie attendue y_i est minimisée. L'erreur quadratique moyenne est souvent utilsée et dans le cadre d'un réseau on l'écrit :

$$C(w,b) \equiv \frac{1}{2N} \sum_i \parallel y_i - \hat{h}(x_i) \parallel^2 = \frac{1}{2N} \sum_i \parallel y_i - \hat{y}_i \parallel^2, \tag{B.4}$$

avec w l'ensemble des poids du réseau, b l'ensemble des biais et N le nombre d'exemples dans le jeu de données. Pour des réseaux de neurones, cette fonction est minimisée à l'aide d'une méthode appelée **backpropagation**.

Optimisation par Nuée de Particules

Une optimisation est un processus lors duquel le meilleur élément (par rapport à des critères donnés) est sélectionné parmi un ensemble de candidats disponibles. Des problèmes d'optimisation existent dans toutes les disciplines quantitatives et le développement de méthodes pour les résoudre a été une source d'intérêt depuis plusieurs siècles.

L'Optimisation par Nuée de Particules (PSO) est un algorithme d'optimisation développé pour les fonctions non-linéaires. Le principe fondamental de cet algorithme est inspiré de la nature car basé sur le comportement d'animaux chassant pour se sustenter, comme le fait un banc de poissons ou une nuée d'oiseaux.

Considérons un système de coordonées à deux dimensions $(\vec{u}_x, \vec{u}_y)$ dans lequel une *population* de particules[1] dont les coordonées (x, y) et vitesses $\vec{v} = (v_x, v_y)$ de départ sont attribuées aléatoirement. À ces coordonées, la valeur d'une fonction est estimée $f(x, y) = s$ et attribuée à la particules correspondante comme une évaluation de sa performance. Ceci implique que chaque particule représente une solution candidate au processus d'optimisation de la fonction $f(x, y)$. À chaque itération, la position et la vitesse de chaque particule sont mises à jour selon trois contributions: un terme basé sur la vitesse de la particule de l'itération précédente représentant son inertie, un biais personnel basé sur son "meilleur souvenir" p_i et un biais collectif basé sur le "meilleur souvenir" de la population g. Ensuite, la fonction $f(x, y)$ est estimée pour la nouvelle position de chaque particule et leur "souvenir" est mis à jour. L'espoir étant qu'après plusieurs itérations, la population aie convergé vers une position commune qui correspond au minimum global de la fonction.

[1]Ici, le terme "particule" n'a rien à voir avec des particules réelles.

B.4 Mesures expérimentales : configuration et démarrage de IPHI et MYRRHA

Typiquement, la conception des injecteurs de protons comprend une ligne de transport basse énergie (LEBT). Sa fonction est d'assurer un transport fiable du faisceau DC depuis la source jusqu'à un RFQ et de conditionner le faisceau pour assurer son accélération et minimiser les pertes de faisceau tout le long de l'accélérateur. Un faisceau convergent, centré et apparié, doit être fourni à l'entrée du RFQ avec des émittances transverses raisonnables (idéalement plus basses que les valeurs de conception) et avec les bons paramètres de Twiss.

De plus, la LEBT devrait permettre de nettoyer le faisceau de protons des autres espèces aussi produites par la source d'ions tels que H_2^+ et H_3^+. Si ces ions moléculaires sont injectés dans le RFQ, ils peuvent créer des pertes parasites importantes. Dès lors, il est important d'intercepter un maximum de ces espèces dans la LEBT.

Finalement, la LEBT est responsable du contrôle de l'intensité du faisceau et de sa structure temporelle. En effet, en fonction des objectifs, différents régimes d'intensité du faisceau et de cycle de service peuvent être souhaités. Par exemple, pendant la mise en marche de l'accélérateur, la LEBT devrait fournir une intensité de faisceau basse avec un cycle de service bas alors que durant l'opération nominale de la machine, la LEBT peut avoir à fournir toute sa capacité.

Ceci montre que le réglage d'une LEBT est un processus crucial pour l'opération correcte d'un accélérateur de proton. Cependant, c'est loin d'être une tâche triviale. Comme son nom l'indique, une LEBT est en charge d'un faisceau à basse énergie qui est donc soumis à des effets de charge d'espace qui compliquent la dynamique de la machine et donc son réglage. C'est particulièrement vrai pour des machines avec une intensité relativement haute telle que MYRRHA avec un faisceau de protons de 4 mA et IPHI avec un faisceau de protons de 80 mA. Pour ces machines, le temps requis pour la mise en marche ou pour changer le mode d'opération manuellement est typiquement de quelques heures à quelques jours sans garantie que le réglage final soit optimal.

L'utilisation de codes de simulation de la dynamique faisceau tels que TraceWin (développé par le CEA Saclay) et Toutatis (code développé par le CEA pour simuler un RFQ) peut aider à comprendre le comportement général de la machine et trouver un point de départ pour chercher une configuration convenable. Cependant, ces codes ne reproduisent pas parfaitement la dynamique du faisceau à cause des approximations utilisées pour limiter le temps de calcul nécessaire.

Dans ce contexte, il est clair qu'améliorer les méthodes de réglage ou d'en développer de nouvelles a le potentiel de réduire le temps nécessaire à la mise en marche et aux changements d'un régime d'opération à un autre. La disponibilité de la machine serait donc augmentée ce qui permettrait de faire plus de sciences.

Dans ce chapitre, les travaux expérimentaux faits sur les injecteurs de MYRRHA et IPHI sont décrits. Ces travaux ont notamment permis de rassembler les mesures expérimentales nécessaires pour constituer les jeux de données indispensables à l'entraînement

de réseaux de neurones.

L'injecteur de MYRRHA

Au moment où les mesures ont été éffectuées, l'injecteur de MYRRHA était constitué des éléments suivants.

- Une source d'ions de type Résonance Cyclotron Electronique (ECR) de 20 mA à 30 keV.
- Deux solénoïdes pour assurer la focalisation du faisceau.
- Deux paires de steerers pour compenser les désalignements dans la LEBT.
- Un système de collimation pour intercepter une partie du faisceau.
- Un hachoir électrostatique pour générer la structure temporelle du faisceau.
- Un cône de collimation pour protéger le RFQ des dernières particules indésirables.
- Deux coupes de Faraday pour mesurer l'intensité du faisceau.
- Deux scanners de type Allison pour mesurer les émittances transverses du faisceau.
- Un ACCT pour mesurer l'intensité du faisceau transitoire.
- Trois jauges de pression pour mesurer la pression du gaz résiduel dans la LEBT.

Les mesures expérimentales ont été organisées selon des scans des solénoïdes et des steerers pour des positions données du collimateur. Un scan consiste à faire varier de façon régulière les commandes des éléments et à mesurer l'intensité du faisceau à la sortie de la LEBT. Ces mesures ont été archivées pour créer un jeu de données exérimentales pour l'entraînement de réseaux de neurones.

En plus, un test de l'algorithme PSO directement sur la machine a été fait. L'optimisation a consisté à trouver une configuration de la ligne qui:

1. fournit une intensité du faisceau de 4.2 ma
2. reste proche de la configuration de design (65.6 A et 77.9 A dans les premier et second solénoïdes respectivement.

L'algorithme était configuré pour avoir une population de 25 particules et un nombre maximum d'itérations égal à 10. Après environ 1.5 h, une configuration satisfaisante a été trouvée. La durée de cette optimisation était principalement due au temps de réponse relativement long des alimentations des steerers (environ 10 secondes pour changer la configuration d'un steerer).

L'injecteur de IPHI

Les injecteurs de IPHI et de MYRRHA sont construits selon les mêmes principes. La principale différence est que le RFQ de IPHI était opérationnel lors des campagnes expérimentales. L'injecteur de IPHI comprend les éléments suivants :

- Deux solénoïdes.
- Un collimateur fixe
- Deux paires de steerers.
- un système de collimation.
- Un cône de collimation.
- Un RFQ pour mettre en paquets à 352 MhZ et accélérer le faisceau jusqu'à 3 MeV.
- Une caméra CCD.
- Une coupe de Faraday.
- Deux ACCTs pour mesurer l'intensité du faisceau à l'entrée et à la sortie du RFQ.

Ici aussi, les mesures expérimentales ont été organisées selon des scans des solénoïdes pour des positions données du collimateur. Il a été tenté de faire également des scans des steerers mais les pertes faisceaux engendrées dans le RFQ le faisaient claquer systématiquement, ne permettant donc pas de poursuivre ce type de scans. Les mesures correspondent ici à l'intensité du faisceau à la sortie du RFQ et à la transmission du faisceau dans le RFQ. Ces mesures ont servi à constituer un jeu de données pour l'entraînement de réseaux de neurones.

Comparaisons entre expériences et simulations

Pour les deux injecteurs, les mesures expérimentales ont été utilisées pour valider et calibrer les modèles TraceWin de leurs machines respectives. Dans les deux cas, les simulations reproduisent raisonnablement les expériences mais des différences peuvent être observées. Ces différences proviennent notamment du fait que les propriétés du faisceau à la sortie de la source sont mal connues, ainsi que l'alignement des éléments de l'injecteur et les effets de la charge d'espace.

B.5 Développement de modèles basés sur les réseaux de neurones artificiels

Les réseaux de neurones sont des outils puissants qui peuvent outrepasser les compétences humaines sur des tâches spécifiques. Cependant, quelque soit la tâche attribuée, le développement d'un réseau de neurones peut se réveler un véritable challenge. En effet, il existe plusieurs structures standards de réseaux qui peuvent être ajustées pour obtenir un réseau adapté mais il n'existe pas de règles bien définies sur les choix à faire. C'est particulièrement vrai pour les hyper-paramètres. En général, l'utilisateur doit procéder par essai-erreur pour entraîner un grand nombre de réseaux avec des paramètres différents et sélectionner le meilleur performeur.

Dans cette thèse, le but des entraînements est d'obtenir des modèles qui reproduisent le comportement de machines réelles.

Pour MYRRHA, les variables disponibles correspondent à la configuration de la LEBT (deux solénoïdes, deux paires de steerers et la position d'un collimateur) et aux mesures de pressions dans la LEBT (trois jauges de pression). Pour chaque set de variables, la quantité d'intérêt correspond à l'intensité du faisceau mesurée à la sortie de la LEBT.

Dans le cas de IPHI, les variables correspondent à la configuration de la LEBT (deux solénoïdes, deux paires de steerers et la position d'un collimateur) et les quantités d'intérêt correspondantes sont l'intensité du faisceau mesurée à la sortie du RFQ et la transmission du faisceau à travers le RFQ.

Les jeux de données utilisés pour entraîner les réseaux suivent ces descriptions dans la mesure du possible.

Pour MYRRHA, les trois jeux de données suivants ont été constitués.

- Un jeu de données expérimentales
  Ce jeu de données contient les mesures de l'intensité du faisceau pour environ 20 000 configurations de la LEBT. Ces mesures sont organisées selon des scans réguliers des steerers et des solénoïdes pour des positions données du collimateur. Les bornes de ce jeu de données sont résumées dans la Table 4.4.

- Un jeu de données simulées selon un échantillonage aléatoire avec le modèle TraceWin
  Ce jeu de données contient l'intensité simulée du faisceau pour environ 100 000 configurations de la LEBT. Les bornes de ce jeu de données sont donnés dans la Table 4.2.

- Un jeu de données simulées selon des scans réguliers avec le modèle TraceWin.
  Ce jeu de données contient l'intensité simulée du faisceau pour environ 100 000 configurations de la LEBT. Les bornes de ce jeu de données sont résumées dans la Table 4.3. Notons que ce jeu de données n'a pas été utilisé pour entraîner des neurones mais pour inspecter la qualité des réseaux entraînés sur l'autre jeu de données simulées.

Pour IPHI, un seul jeu de données a été utilisé.

- Un jeu de données expérimentales
  Ce jeu de données contient les mesures de l'intensité du faisceau pour environ 10 000 configurations de la LEBT. Les bornes de ce jeu de données sont résumées dans la Table 4.1.

A l'aide du jeu de données simulées aléatoirement, une structure type de réseaux a été définie selon un processus de validation croisée. La fonction d'activation choisie est la fonction Unité Linéaire Rectifiée (ReLU) $f(x) = max(x, 0)$ pour les couches cachées et la fonction sigmoïde $f(x) = (1 + e^{-x})^{-1}$. La structure du réseau compte 3 couches cachées avec 96 neurones chacune. L'algorithme supervisé utilise des lots de 96 exemples et un taux initial d'apprentissage de 0.1.

B.6 Entraînement de réseaux de neurones et résultats

Les jeux de données décrits dans le chapitre précédent ont été utilisés pour entraîner des réseaux de neurones avec pour but de reproduire le comportement des injecteurs réels. En particulier, l'objectif est de développer un ensemble d'outils permettant de faciliter le réglage de l'injecteur de MYRRHA.

Puisque le RFQ de MYRRHA n'était pas disponible lors de la mise en service de la LEBT, les données d'entraînement représente le fonctionnement de la LEBT seule. Plusieurs réseaux ont été entraînés pour prédire l'intensité du faisceau à la sortie de la LEBT en fonction de sa configuration.

Jeu de données expérimentales

Sur ce jeu de données, la structure initiale déterminée au chapitre précédent n'a pas permis d'obtenir de prédictions précises de l'intensité du faisceau. Heureusement, suite à un procédé d'essai-erreur à partir de cette structure, une structure dont l'entraînement a donné des résultats satisfaisants a pu être déterminée. La différence majeure avec la structure initiale est la réduction du nombre de neurones par couche cachée de 96 à 64.

Une fois entraîné, les prédictions de cette nouvelle structure ont été inspectées. Cette inspection a montré que le réseau prédit correctement le comportement de la LEBT lorsqu'on l'utilise pour reproduire un scan des solénoïdes ou des steerers. Cependant, les prédictions présentent de fortes oscillations entre les positions échantillonnées du collimateur.

Jeu de données simulées aléatoirement

Pour ce jeu de données, la structure initiale a pu être entraînée avec de bons résultats. En effet, l'inspection des predictions a montré que le réseau reproduit bien le comportement général du modèle TraceWin. Cependant, les prédictions ne reproduisent pas bien les changements brusques de l'intensité du faisceau présents dans certaines régions du modèle TraceWin. Malgré ce constat, les prédictions présentent très peu de fluctuations contrairement au réseau entraîné sur les données expérimentales.

Jeu de données simulées aléatoirement puis expérimentales

Afin d'obtenir un modèle performant qui reproduit le comportement de la machine réelle et non pas de la machine simulée, le réseau obtenu après l'entraînement avec les données simulées a été entraîné à nouveau mais cette fois avec les données expérimentales. L'inspection des prédictons après ce second entraînement a montré que le comportement de la machine réelle est à nouveau correctement prédit par le réseau lorsque l'on reproduit les scans de solénoïdes et de steeerers. De plus, les fluctuations présentent entre les positions échantillonnées du collimateur ont été largement réduites.

Application de PSO sur le modèle entraîné

Afin d'illustrer une utilisation du modèle entraîné, une optimisation avec l'algorithme PSO a été effectuée de façon similaire à celle faite directement sur la machine mais cette fois avec le réseau entraîné. Dans les deux cas, l'optimisation fournit des configurations candidates qui sont similaires. La principale différence est que l'application de PSO sur la LEBT réelle a pris environ 1.5 heures alors que son application sur le modèle entraîné n'a duré que 10 secondes. En fait, dans le cas de l'utilisation du modèle entraîné, la partie la plus lente de l'optimisation était en fait l'algorithme PSO lui-même (environ 8 secondes contre 2 secondes pour les 250 évaluations avec le réseau de neurones).

Jeu de données éxpérimentales de l'injecteur IPHI

En préparation de la mise en service du RFQ de MYRRHA, des réseaux de neurones ont été entraînés sur le jeu de données de l'injecteur IPHI (LEBT + RFQ). Ici, la structure initiale de réseau a été entraînée avec succès. L'inspection des prédictions a montré que le réseau est capable de prédire correctement la transmission du RFQ et l'intensité du faisceau simultanément.

Application de PSO sur le modèle entraîné pour IPHI

Une optimisation avec PSO a également été effectuée sur le modèle entraîné de IPHI de manière similaire à l'application de PSO sur le modèle entraîné de la LEBT de MYRRHA. La principale différence est qu'ici l'algorithme devait optimiser la configuration de la LEBT afin d'obtenir une intensité du faisceau cible tout en maximisant la transmission du faisceau à travers la ligne. Comme pour l'application sur le modèle de MYRRHA, le processus complet a pris environ 10 secondes. De plus, trois configurations ont été suggérées par l'algorithme avec une intensité du faisceau proche de la valeur voulue et une transmission de plus de 96 %.

B.7 Conclusions et perspectives

Les projets d'accélérateurs de particules nécessitent d'atteindre des puissances moyennes plus élevées et d'améliorer la fiabilité. Ce constat est particulièrement vrai pour les projets d'ADS tels que MYRRHA. En effet, il est crucial de satisfaire un niveau de fiabilité qui n'a encore jamais été atteint à ce jour au niveau mondial afin de pouvoir opérer un ADS. Afin de satisfaire cette contrainte, il est important de minimiser les pertes du faisceau le long de l'accélérateur. Une façon de le faire est de garantir la qualité du faisceau à accélérer en optimisant la configuration de l'injecteur. Ce manuscrit présente le travail de recherche entrepris pour explorer la possibilté de développer de nouveaux outils basés sur des réseaux de neurones afin de faciliter cet effort.

Afin d'illustrer la réalité de la tâche, la mise en marche et le réglage de deux injecteurs, ceux de IPHI et MYRRHA, sont décrits dans le chapitre 3. Durant l'une des expériences sur l'injecteur de MYRRHA, l'algorithme PSO a été testé en tant que méthode d'optimisation en ligne de la machine. Aussi, les jeux de données expérimentales constitués lors de ces travaux ont servi pour évaluer la représentativité des modèles TraceWin de référence pour leurs machines respectives. Pour les deux injecteurs, les modèles ont fourni des résultats raisonnables mais avec d'apparentes différences par rapport aux données expérimentales.

La constitution des jeux de données pour entraîner des réseaux de neurones dont le but est de modéliser les injecteurs de MYRRHA et IPHI est présentée dans le chapitre 4. De plus, une structure initiale de réseau et les hyper-paramètres pour l'entraînement y sont déterminés à l'aide d'une validation croisée.

Dans le chapitre final, l'entraînement de réseaux de neurones est discuté pour les deux injecteurs.

Dans le cas de la LEBT de MYRRHA, des réseaux ont été entraînés pour prédire l'intensité du faisceau qui peut être mesurée à la sortie de la LEBT en fonction de sa configuration. Le réseau le plus performant a d'abord été entraîné sur le jeu de données simulées aléatoirement puis entraîné une seconde fois sur le jeu de données expérimentales. Cette approche a montré que le petit nombre de donées expérimentales peut être compensé avec des simulations malgré que le modèle TraceWin présente des différences avec la machine réelle.

Dans le cas de IPHI, des réseaux ont été entraînés pour prédire l'intensité du faisceau et sa transmission à travers le RFQ en fonction de la configuration de la LEBT. Le modèle le plus perfomant est capable de reproduire ces deux quantités simultanément.

De plus, les réseaux les plus performants de chaque injecteur ont été utilisés en tant que substituts à leurs machines respectives avec l'algorithme PSO. Avec le modèle de la LEBT de MYRRHA, l'algorithme a été capable de suggérer des solutions candidates proches de la configuration de référence pour une variété d'intensités du faisceau désirées. Avec celui de l'injecteur IPHI, l'algorithme a été capable de suggérer des

configurations candidates qui correspondent à l'intensité du faisceau désirée tout en maximisant la transmission du RFQ. Dans les deux cas, le processus d'optimisation a été très rapide avec l'algorithme PSO lui-même comme plus grande contribution au temps d'exécution. Il a donc été suggéré que ces performances pourraient être améliorées en parallélisant les évaluations avec le modèle et en optimisant l'algorithme PSO.

D'après les résultats obtenus dans le chapitre 5, les modèles entraînés peuvent encore être améliorés. En particulier, le modèle final de la LEBT de MYRRHA a montré une sous-estimation systématique de l'intensité du faisceau pour les grandes ouvertures du collimateur. Afin d'améliorer les prédictions du modèle entraîné, il serait intéressant de tester des stratégies d'entraînement plus élaborées. Par exemple, le nombre de couches cachées dans le réseau de neurones peut être augmenté avec la stratégie suivante :

1. Entraînement d'un premier réseau avec n_1 couches cachées (typiquement 2 ou 3).

2. Construction d'un second réseau avec $n_2 > n_1$ couches cachées où les n_1 premières couches utilisent les poids et biais déterminés à l'étape précédente.

3. Entraînement du second réseau en gardant les n_1 premières couches "gelées" (les poids et biais de ces couches sont ignorés lors de l'entraînement).

Dans un sens, cette approche correspond à entraîner un réseau avec une autre réseau comme entrées. Avec cette méthode, les couches additionnelles devraient être entraînées pour rajouter de la nuance au réseau initial. Cette technique pourrait se révéler utile pour modéliser les variations abruptes de l'intensité du faisceau présentes dans certaines régions de l'espace des configurations.

Une autre application potentiellement intéressante de telles stratégies est d'entraîner un réseau pour modéliser le transport de la distribution d'un faisceau dans un RFQ. En effet, afin de simuler correctement la dynamique du faisceau dans un RFQ, un code de simulation tel que Toutatis (développé par le CEA) nécessite de traquer chaque particule du faisceau. Ce type de simulation est très demandant en ressources de calculs. Dès lors, il serait intéressant d'entraîner un réseau pour prédire la distribution du faisceau à la sortie du RFQ en fonction de la distribution à l'entrée du RFQ et de la configuration de la machine.

D'un autre côté, l'algorithme PSO a été utilisé avec succès en ligne sur la machine réelle et avec les modèles entraînés sans connaissance préalable du problème à optimiser. Même si cela rend cet algorithme très pratique, il est quand même nécessaire de tester un grand nombre de configurations afin de trouver les candidats finaux. Entraîner un réseau en apprentissage renforcé à l'aide du modèle entraîné comme environnement d'apprentissage pourrait mener à la création d'un algorithme bien plus adapté à l'optimisation d'un injecteur. L'idée est d'entraîner un réseau prédicteur dont

la tâche serait de suggérer une configuration adéquate pour obtenir les paramètres désirés pour le faisceau. Pour ce type d'entraînement, le réseau prédicteur doit interagir un grand nombre de fois avec l'injecteur pour apprendre quelles configurations fournissent quelles propriétés du faisceau. Donc, l'utilisation d'un réseau modèle en tant que substitut à la machine réelle réduirait significativement le temps nécessaire pour entraîner un réseau prédicteur. En effet, le prédicteur peut intéragir avec le modèle plusieurs centaines de fois par seconde alors qu'il ne pourrait interagir avec la machine réelle qu'une fois toutes les quelques secondes. De plus, avec un réseau modèle, l'entraînement du prédicteur peut être parallélisé.

Fondamentalement, deux types principaux de prédicteurs peuvent être considérés :

- Prédicteur "one shot"
  Dans ce cas, le prédicteur est tâché de suggérer une configuration candidate en une seule itération. Cette configuration serait ensuite appliquée à la machine réelle et raffinée à l'aide d'autres systèmes.

- Prédicteur "multi shot"
  Ici, la tâche du prédicteur est de suggérer une première configuration candidate à tester sur la machine. Ensuite, en se basant sur les propriétés du faisceau réellement obtenues, le prédicteur devrait suggérer une seconde configuration canditate à tester sur la machine. Ce processus serait répété jusqu'à ce que les propriétés désirées soient obtenues, jusqu'à un nombre maximum d'itérations ou jusqu'à ce que tout autre critère d'arrêt soit rempli.

Le concept de prédicteur "multi shot" peut être poussé plus loin. S'il est branché directement au système de contrôle de la machine et permis d'opérer en continu, un prédicteur "multi shot" pourrait être utilisé comme un système de contrôle automatique pour un injecteur.

A propos du projet MYRRHA, le RFQ a été couplé à la LEBT dans le milieu de l'année 2020 et est opérationnel. Au vu des résultats obtenus sur l'injecteur IPHI, il serait intéressant de constituer un nouveau jeu de données pour l'injecteur de MYRRHA afin d'entraîner un réseau de neurones pour prédire les propriétés du faisceau à la sortie du RFQ en fonction de la configuration de la LEBT. Ceci permettrait d'utiliser l'algorithme PSO sur le modèle entraîné pour déterminer une configuration candidate afin d'obtenir les propriétés du faisceau requises. Cette configuration serait ensuite appliquée à la machine et raffinée en ligne avec l'algorithme PSO.

Toujours à propos du projet MYRRHA, même si cette thèse est focalisée sur le réglage de l'injecteur, la stratégie de tolérance aux pannes prévue pour le linac SC peut également bénéficier de l'utilisation de réseaux de neurones. L'idée est que si un élément tombe en panne, alors les éléments voisins changeraient de réglages afin de compenser la panne. Les configurations compensées doivent être calculées à l'avance à l'aide d'un algorithme dédié puis raffinées en utilisant le modèle TraceWin du linac. Ce processus pourrait bénéficier d'un réseau entraîné pour modéliser le linac SC.

Les perspectives présentées ici visent au développement d'un ensemble de nouveaux outils pour faciliter le réglage de l'injecteur de MYRRHA et du linac SC. Cependant, ce ne sont pas les seules utilisations potentiellement utiles de l'apprentissage automatique dans le domaine des accélérateurs de particules. En effet, les réseaux de neurones et l'apprentissage automatique sont des outils puissants et adaptables qui peuvent être utilisés pour développer des outils ad hoc quand ils sont nécessaires. La communauté des accélérateurs est bien consciente de ce potentiel comme démontré par le grand nombre de sollicitations reçues durant ce travail de thèse. Beaucoup de travaux et de développements sont encore à faire avant que ces méthodes ne soient complètement matures mais elles vont certainement être de plus en plus largement utilisées dans les années à venir.

Summary

Recent particle accelerator projects need to meet higher and higher reliability and stability levels. This is especially true for ADS (Accelerator Driven System) projects that aim to drive a nuclear reactor with a particle accelerator. These require building high-power (a few MW) proton accelerators with extremely high reliability to incinerate nuclear waste without compromising the reactor structure. This is the case for the accelerator of the MYRRHA (Multi-purpose hYbrid Research Reactor for High-tech Applications) that would provide a 4 mA CW (Continuous Wave) proton beam at 600 MeV (this corresponds to a 2.4 beam power of MW). This project, led by the SCK CEN in Belgium, is based on the construction of a superconducting linear accelerator (linac) and aims to go under the limit of 10 unscheduled beam trips longer than 3 seconds for each operation cycle of 3 months. This represents a level of reliability never achieved before in the world.

A key point to achieve this goal is to ensure a good configuration of the injector to minimize the beam losses that can force the shutdown of the machine if they exceed the tolerance level. This thesis explores the possibility of using Machine Learning methods to develop a numerical model able to reproduce accurately the experimental behavior of a injector. Especially, this thesis studies the training of artificial neural networks under supervised learning to model the MYRRHA and IPHI (Injecteur de Protons à Haute Intensité, CEA Saclay) injectors. With this goal in mind, several experimental and simulated datasets have been constituted. In this manuscript, the constitution of these datasets, the training of neural networks and their performances are presented and discussed.

Résumé

Les projets récents d'accélérateurs de particules demandent d'atteindre des niveaux de fiabilité et de stabilité de plus en plus stricts. En particulier, les projets d'ADS (Accelerator Driven System), dont le but est de piloter un réacteur nucléaire à l'aide d'un accélérateur de particules, nécessitent de construire des accélérateurs de protons à haute puissance (quelques MW) et extrêmement fiables afin de permettre l'incinération de déchets nucléaires sans compromettre la structure du réacteur. C'est le cas de l'accélérateur du projet MYRRHA (Multi-purpose hYbrid Research Reactor for High-tech Applications) avec un courant de protons CW (Continuous Wave) de 4 mA accéléré jusqu'à 600 MeV (soit une puissance faisceau de 2.4 MW). Ce projet, porté par le SCK CEN en Belgique, est basé sur la construction d'un accélérateur linéaire (linac) supraconducteur et a pour objectif de générer moins de 10 arrêts non programmés de faisceau plus longs que 3 secondes pour chaque cycle d'opération de 3 mois. Ceci représente un niveau de fiabilité qui n'a encore jamais été atteint au niveau mondial.

Un des points clefs pour atteindre cet objectif est de garantir un bon réglage de l'injecteur afin de minimiser les pertes faisceaux qui peuvent causer l'arrêt de la machine lorsqu'elles deviennent trop importantes. Cette thèse explore la possibilité d'utiliser des méthodes d'apprentissage automatique pour le développement d'un modèle capable de reproduire fidèlement le comportement expérimental d'un injecteur. En particulier, cette thèse étudie l'entraînement de réseaux de neurones artificiels par apprentissage supervisé avec pour objectif de modéliser l'injecteur de MYRRHA ainsi que celui du projet IPHI (Injecteur de Protons à Haute Intensité, CEA Saclay). Dans ce but, plusieurs ensembles de données, expérimentaux et simulés, ont été constitués. Dans ce manuscrit, la constitution des jeux de données, l'entraînement de réseaux de neurones sur ces données et les performances de ces derniers sont présentés et discutés.