UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

Design and implementation of a motor control system for robotics powered by artificial neural networks

Yago González Gómez

2021

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

Design and implementation of a motor control system for robotics powered by artificial neural networks

Autor

Yago González Gómez

Tutor

Álvaro Gutiérrez Martín

2021

Abstract

Modern machine learning (ML) based on artificial neural networks (ANNs) is one of the research disciplines that has grown the fastest in the history of science. However, even though the results obtained keep getting better, most of the progress that is being made is not providing any significant insights on the cognitive process behind artificial neural networks nor how to eventually achieve artificial general intelligence (AGI).

In order to gain insights about the performance of three mainstream ANN-powered ML techniques on biologically plausible challenges, an experimental setup has been designed for this BEng Thesis around the inverse kinematics (IK) problem. IK is often applied in robotics and computer graphics to define how should an articulated entity position itself to reach a certain target with an actuator, and it is similar to how us humans control our bodies at a low level.

The three methods analysed here have been supervised learning, black-box optimisation and reinforcement learning. After defining a set of robust benchmarks that can be shared between them, they have all been applied to perform IK on a simulated robot arm with different degrees of freedom.

The results showcase the advantages of each particular method and highlight the flexibility that reinforcement learning offers over the alternatives, allowing for a better generalisation capacity at the expense of a higher computational cost.

Keywords: machine learning, artificial neural networks, deep learning, motor controller, inverse kinematics, robotics, supervised learning, black-box optimisation, reinforcement learning, bio-inspired learning

Resumen

El aprendizaje automático (ML) moderno basado en redes neuronales artificiales (ANN) es una de las disciplinas de investigación que más rápido han crecido en la historia de la Ciencia. Sin embargo, y a pesar de que los resultados obtenidos mejoran cada vez más, la mayor parte del progreso que se está logrando no está proporcionando ningún conocimiento significativo sobre el proceso cognitivo que hay tras las redes neuronales artificiales, ni de cómo podemos conseguir inteligencia artificial general (AGI).

Con el objetivo de aprender más sobre el rendimiento de tres técnicas de ML populares basadas en ANN en retos biológicamente plausibles, una configuración experimental se ha diseñado para este TFG en torno al problema de cinemática inversa (IK). La cinemática inversa se aplica frecuentemente en robótica y en gráficos por ordenador para definir cómo debe posicionarse una entidad articulada para alcanzar un cierto objetivo con un actuador, y es similar a cómo controlamos los humanos nuestros cuerpos a bajo nivel.

Los tres métodos analizados aquí han sido aprendizaje supervisado, optimización de cajas negras y aprendizaje por refuerzo. Tras definir un conjunto de criterios robustos que pudiesen ser compartidos entre ellos, los tres han sido aplicados para realizar cinemática inversa en un brazo robótico simulado con diferentes grados de libertad.

Los resultados muestran las ventajas de cada método particular y resaltan la flexibilidad que el aprendizaje por refuerzo ofrece sobre las alternativas, lo que permite obtener mejor capacidad de generalización a cambio de un mayor coste computacional.

Palabras clave: aprendizaje automático, redes neuronales artificiales, aprendizaje profundo, controlador motor, cinemática inversa, robótica, aprendizaje supervisado, optimización de caja negra, aprendizaje por refuerzo, aprendizaje bioinspirado

Acknowledgements

The author acknowledges that this BEng Thesis has been developed using resources from the Laboratory of Robotics and Control (Robolabo) at ETSIT-UPM.

Special thanks to Prof. Gutiérrez, one of the most dedicated and professional people that I have had the pleasure to meet. Building this BEng Thesis would not have been possible without his guidance, support, and overall inspiration to strive for excellence.

But most importantly, thanks to Mónica and Felisa. Mom, grandma, this is for you.

Contents

List of Figures

List of Tables

List of Acronyms

AD	Automatic Differentiation.
AGI	Artificial General Intelligence.
AI	Artificial Intelligence.
ANN	Artificial Neural Network.
API	Application Programming Interface.
CBGTC	Cortico-Basal Ganglia-Thalamo-Cortical.
DoF	Degree of Freedom.
HDF	Hierarchical Data Format.
IK	Inverse Kinematics.
MDP	Markov Decision Process.
ML	Machine Learning.
MLP	Multi-Layer Perceptron.
PPO	Proximal Policy Optimisation.
PRNG	Pseudo-Random Number Generator.
RBS	Reset-Based Simulation.
RL	Reinforcement Learning.
ROS	Robot Operating System.
SPA	Semantic Pointer Architecture.
UML	Unified Modeling Language.
URDF	Unified Robot Description Format.
XML	Extensible Markup Language.

List of Algorithms

Chapter 1

Introduction

Creating artificial machinery that is superior to the human being is arguably at the core of most modern engineering disciplines. Even before the concept of engineering was born, the Ancient Greek already fantasised with the idea of automatons inspired by the human body: an example is Talos, the mythological *man of bronze* that was responsible for protecting Europa by striding around the island of Creta, where she was located [1].

Despite the outstanding progress that we have achieved throughout the past millennia, we are still far from even being able to artificially reproduce the behaviour of a complete human, let alone having something better. We have certainly been able to build faster, stronger and overall more powerful machines, but there is a part that is significantly lagging behind: cognition.

Our Greek ancestors did not know what we do now about neurobiology nor anatomy, but we still lack of a complete picture of how the brain works its magic. As a result, Computer Science and Artificial Intelligence (AI) resort to using partial or alternative models that capture a set of its functionality.

Our growing computing capacity has led to the recent boom in AI, which does not seem to have an end in the coming years. Better hardware, better software and better mathematical tools are allowing research in the field to progress at an unprecedented pace. But unfortunately, that does not make us any closer to the original goal of solving cognition. We still do not know how do thoughts take place in our minds, and we even struggle to explain the behaviours that emerge in the nowadays widespread artificial neural networks. This, combined with other issues related to the methodology applied, has started to raise concerns about the scientific rigour of this "progress": from research integrity malpractice [2] to even being compared with alchemy [3], amongst others [4].

This BEng Thesis is focused on making a lower-level analysis of some modern machine learning techniques in order to better understand their inner workings and extrapolating them to human learning. It has been done within the context of solving a biologically plausible task applied to robotic control, and with the greater goal of finding out which of these machine learning methods are more compatible with our biologic processes. Having this information could help in future development of more complex AI systems that mimic human behaviour.

1.1 Document structure

In this introductory chapter, an overview of neural networks (Section 1.2) and the Semantic Pointer Architecture (1.3) is given before defining the challenge addressed by the motor controllers in this BEng Thesis (1.4) and going through the state of the art on the matter (1.5).

On Chapter 2, details are given on the experimental setup that is used throughout the whole project, and Chapter 3 covers in depth the three different types of experiments performed: supervised learning (3.1), black-box optimisation (3.2) and reinforcement learning (3.3). Each of them include the methodology followed by a discussion of the results obtained.

Finally, Chapter 4 summarises the analysis drawing the key conclusions of the BEng Thesis (4.1) and the potential future lines of work (4.2).

1.2 Neural networks

1.2.1 Neurons

Neurons are the main cells in the nervous system, and they are therefore at the core of the human cognitive capabilities. Even though their classification and functional description are still under active research, the typical model of a neuron consists of the parts in Figure 1.1. This "standard" neuron receives electric stimuli received by the dendrites, performs some non-linear processing through the soma and the axon, and delivers different electric stimuli on the synaptic terminals, which are connected to other neurons. This highly simplified model works under the following assumptions, which do not necessarily apply to all types of neurons:

- Information only flows in one direction (from dendrites to synaptic terminals).
- Dendrites only receive electric impulses and apply them a certain gain. This means that they do not apply any non-linearities, although there are studies that show the opposite [5].
- The overall neuron's response to the input stimuli does not have to increase with higher intensities in the dendrites: some of the responses might (thus being excitatory) and others might do the opposite (i.e. inhibitory ones).

The stimuli is done with voltage spikes, which vary in frequency depending on the intensity of the associated magnitude. This interesting behaviour makes neurons intrinsically dynamic, meaning that the neuron's evolution over time is the factor that actually conveys the information (as opposed to its immediate state).

The computational power of a single neuron is relatively limited, but the high-density aggregation of millions of neurons can lead to very intricate interconnectivity from which complex behaviour can emerge.

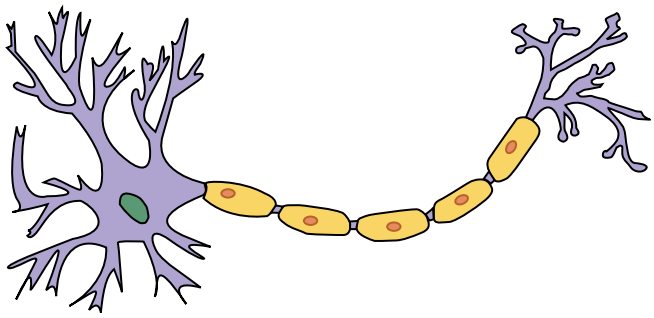


Figure 1.1: Diagram of a biological neuron. The branches on the left represent the **dendrites**, which end in the cell's **soma**. The branches on the right represent the **synaptic terminals**, which are connected to the soma by a long "cable" called the **axon**, Source: Wikimedia Commons (*Dhp1080*, SVG adaptation by *Actam*). CC BY-SA 3.0.

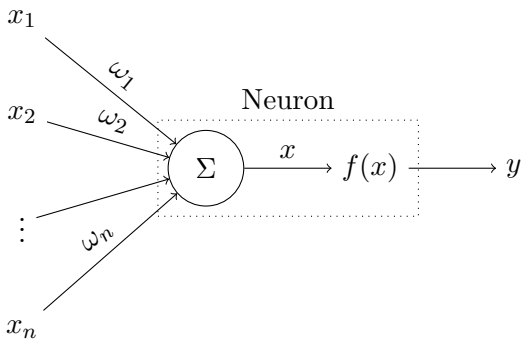


Figure 1.2: Diagram of a neuron as implemented in an artificial neural network.

1.2.2 Artificial neural networks

Artificial neural networks are processing systems that try to mimic the biological neural networks present in the brain of animals like humans. They use an even more simplified model of the neuron (see Figure 1.2), in which the weighted sum of all inputs is then passed to an **activation function** that returns the neuron's response. This activation function can be customised depending the problem at hand, but it is often a non-linear one like a sigmoid ($f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$) or a rectified linear unit ($f(x) = \max(0, x)$).

Neurons are grouped in layers, which share the same inputs and produce an output with the concatenation of all of the neuron's outputs. Layers can be stacked, connecting the output of each neuron with the inputs of all the neurons in the next layer, creating what is called a **multi-layer perceptron** (MLP) as depicted in Figure 1.3. This basically emulates the high connection density of biological neural networks, although vanilla MLPs do not take into account features like skip connections (neurons directly connected to layers that are not the immediately next one) or recurrent connections (neurons with a feedback loop between their output and their input).

The arrangement, size and amount of layers determines the **architecture** of a neural network. A full model is defined by its architecture and the **parameters** of

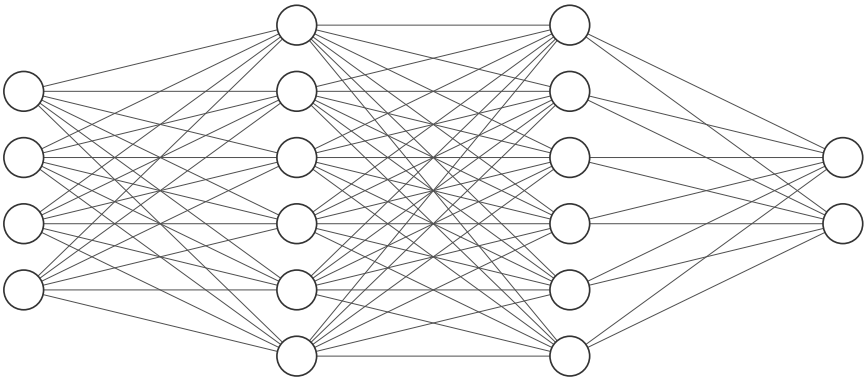


Figure 1.3: Network graph for a 3-layer perceptron, with 3, 6, 6 and 2 neurons in each layer, respectively.

all of its neurons, namely the weights and any variable that influences the behaviour of each neuron's activation function. MLPs are the simplest architectures (yet very widespread), although there are many others.

In spite of the high level of abstraction and simplification made in ANNs versus biological neural networks, they have proven to give fascinating results in a lot of advanced tasks, such as classifying images [6], playing complex games [7] and natural language processing/generation [8].

1.3 The Semantic Pointer Architecture

Eliasmith introduced the concept of the Semantic Pointer Architecture (SPA) [9] as a model to explain how the brain performs high-level cognitive functions that can later be mapped to low-level information, in a similar way to how embeddings work in machine learning.

This means that a semantic pointer has some synthesised information (the **shallow semantics**) of a concept that is more complex (the **deep semantics**). Having such a differentiation allows for faster processing when the use of shallow semantics is sufficient, but of course implies that there is a loss of information in the semantic pointer.

In the SPA, concepts are represented with high-dimensional vectors, and they support two complementary operations called **binding** and **unbinding**. Conceptually speaking, binding two vectors $(\vec{v}, \vec{w})$ creates a third one $(\vec{u})$ that is different to the original ones, but that still preserves information about both of them. Mathematically, it can be defined as the circular convolution of the two original vectors:

$$\vec{u} = \vec{v} \circledast \vec{w} = (u_1, u_2, \ldots, u_i) \tag{1.1}$$

$$u_i = \sum_{j=1}^{D} v_j w_{(i-j) \bmod D}, \quad D = |\vec{v}| = |\vec{w}| \tag{1.2}$$

This operator has the interesting particularity that it satisfies the following

property:

$$\vec{u} \circledast \vec{w}^{+} = \vec{v} + \overrightarrow{\text{noise}} \approx \vec{v}, \quad \vec{w}^{+} = (w_1, w_D, w_{D-1}, \ldots, w_2)^{\intercal} \tag{1.3}$$

which is (an approximation of) the inverse of the binding process, i.e. the definition of the unbinding process.

Having these two operators allow aggregating information and doing computations with it. For instance, if we denote each word as a vector representing that concept in a D-dimensional vector, we can represent the sentence "He likes avocados" as a single proposition P, and then easily query its subject:

$$P = \overrightarrow{\text{he}} \circledast \overrightarrow{\text{subject}} + \overrightarrow{\text{likes}} \circledast \overrightarrow{\text{action}} + \overrightarrow{\text{avocados}} \circledast \overrightarrow{\text{object}} \tag{1.4}$$

$$P \circledast \overrightarrow{\text{subject}} = \overrightarrow{\text{he}} + \overrightarrow{\text{noise}} \tag{1.5}$$

The SPA was the foundation for the SPA Unified Network (Spaun) [10], an astonishingly competent model capable of learning how to perform multiple tasks just by receiving visual inputs over a time series. Spaun made use of more complex artificial neurons and a highly customised neural architecture that was much more strongly inspired by biologically plausible neural networks.

These results are quite aligned with the motivation this BEng Thesis: giving a biologically inspired approach to neural network models can lead to the creation very effective models that are capable of performing hard tasks. Furthermore, the binding operator in the SPA performs a very similar operation to the one done to fully interconnect two layers in a MLP, which might be a hint that with the right setup and proper time dynamics, mainstream neural networks are not that far away from advanced models like Spaun.

1.4 The inverse kinematics problem

1.4.1 Overview

The problem that the motor controllers developed in this BEng Thesis pretend to solve is a very fundamental yet complex need of robotic systems: identifying how to map points in the three-dimensional Cartesian space to the motor range of a specific robot, or in other words, how must a robot position itself to reach a certain point in space. Such calculation of the joint positions for a specific point in space is known as **inverse kinematics** (IK).

Using a Cartesian space for high-level computations allows for much more intuitive systems that can also be reused across different robots, regardless of their morphology or hardware specifications. This makes it a very appealing coordinate system in a myriad of robotic applications, but adds the need of a middle system that transforms these 3D **Cartesian coordinates** to the n-dimensional joint positions that are compatible with the robot, also known as **generalised coordinates**. The motor controller is the responsible for this translation that, in most real cases, is not trivial at all.

Even though robotics is the most frequent application of IK, it can also be used in other areas like computer graphics [11], where the methodology and mathematics used are practically identical.

1.4.2 Mathematical definition

The inverse kinematics situation can be mathematically formalised as the mapping between the set of possible Cartesian coordinates C and the generalised coordinates $G \subseteq \mathbb{R}^n$, where n is the number of joints in the robot in question. Even though a tuple of multiple Cartesian coordinates could be associated with multiple parts of a robot, we will consider positioning a single end actuator for the sake of simplicity: $C \subseteq \mathbb{R}^3$.

As it is exemplified in Section 3.1.4.2, this mapping is often one-to-many, meaning that for a specific Cartesian coordinate there are multiple possible generalised coordinates, all equally valid. This is the opposite of a surjective function, meaning that the inverse of this mapping could be modelled by a function of such properties.

It is also possible to include in the IK map the orientation of the end actuator—thus extending the dimensionality of C—but the controllers in this BEng Thesis are limited to the mapping of positions, as the extra component is not especially relevant for the global goal of the project.

1.4.3 Role in Biology

Inverse kinematics is not limited to robotics: it is actually a biologically plausible need that naturally emerges for us humans. For instance, whenever we want to reach an object with a hand we do not explicitly think how should our articulations be placed to get there; we use visual feedback instead altogether with other signals to identify the target in our surroundings, and then send the corresponding stimuli to our muscles. The process it is very similar to the decompression of semantic pointers as described by the SPA, where spatial coordinates are semantic pointers that are linked to higher-dimensional information (the motor stimuli).

From a cognitive point of view it is an interesting challenge because it is a first layer that could then be used as foundations for more complex motor behaviour, and also because it is probably one of the first things that children learn after birth. Having a good understanding of how IK works both in humans and in machines can therefore help us grasp new details about learning.

1.5 Previous work

1.5.1 Analytical and numerical solutions

For rather simple robotic systems it is feasible to derive analytical equations that can be used to calculate the inverse kinematics, which is very advantageous because it provides stable and reliable values for the mappings. However, that approach becomes unmanageable as soon as more degrees of freedom are added and multiple solutions appear for the same sets of Cartesian points. There is software that can help generating the equations of more complex systems [12], but it is still limited by the maximum number of degrees of freedom that it can handle.

Another alternative is to use numerical or heuristic approaches, which find the potential solutions iteratively by using differential equations that model the system

or approximations thereof. Depending on the underlying algorithm, they can be quite computationally expensive to compute.

All these methods are by default ignorant of the rest of the limitations of the environment, which makes them prone to causing collisions with other elements in the robot's surroundings. That is why many applications require adding constraints, which a difficult and cumbersome process that often has to be manually integrated with the solving system. But most importantly, all these approaches rely on mathematical models that are not really relevant for the emergence of cognition.

1.5.2 Modelling with machine learning

The main difference between the aforementioned methods that are nowadays used to solve inverse kinematics and the ones applied in this BEng Thesis is the underlying technology. The controllers developed in this project are based on artificial neural networks (ANNs), instead of the traditional mathematical tools.

Other works have also tried to approach the IK problem with ANNs, but they used oversimplified versions of a robotic arm [13] or lack transparency on the neural architecture and general methodology used [14].

Chapter 2

The simulation environment

As explained in detail in Chapter 3, all the motor control systems developed in this BEng Thesis are strictly based on machine learning techniques. This implies that actual learning experiences have to be fed to the systems for them to capture the environment's dynamics.

To achieve this, an open-source simulation framework has been built on top of the Bullet physics engine [1]. This framework constitutes a performant and cost-effective solution, which is especially critical in this type of scenarios that would otherwise require very expensive hardware to reproduce in a real environment. Furthermore, it allows for much better reproducibility, as discussed later in this chapter.

2.1 Physical environment

2.1.1 PyBullet

PyBullet [15] is a Python module that acts as bindings for the Bullet physics engine[2]. PyBullet is arguably the most popular open-source alternative to other propietary simulation frameworks like MuJoCo [16] or CoppeliaSim [17].

Even though MuJoCo is probably more widespread in the machine learning research community, each of its personal licenses are bound to a specific device, making them unusable across different hosts. This is important because it makes it hard running simulations in cloud services (which might be hosted in different machines), and hinders the reproducibility of the results.

PyBullet was therefore chosen in order to keep this BEng Thesis as reproducible as possible, and to overall contribute to the model of completely open research. This philosophy has also been reflected by favouring free software[3] over other alternatives for the whole BEng Thesis.

The internals of PyBullet work over a client-server model, in which a physics server does the computations and then returns the status for commands sent by a client. The server is automatically spawned by PyBullet, and it can also perform a

[1] `https://github.com/YagoGG/beng-thesis`

[2] `https://github.com/bulletphysics/bullet3`

[3] It is worth highlighting that "free" here does not make reference to the costs of the software, but to the licensing and general philosophy behind it, in accordance to what the GNU project defines as the *four essential freedoms*: `https://www.gnu.org/philosophy/free-sw.en.html`.

real time visualisation of the environment from a graphical user interface powered by OpenGL.

The fact that all interactions are made through the client make the simulator's code very centred around the `BulletClient` objects, which handle the connection between both parts of the system. As such, it is the basic building block for all the simulations in this BEng Thesis, as it is explained in more detail in Section 2.1.4.

2.1.2 Action-observation environments

This BEng Thesis's environments consist of a scenario where an agent is placed. As summarised in Figure 2.1, agents can gather information describing the environment's **state** (s) in what is called an **observation** (o), and can interact with the environment by taking an **action** (a). The information conveyed by observations might or might not fully describe the full state of the environment, leaving room for some entropy that could potentially be out of the agent's reach. The simulation environments also introduce an additional variable called the **reward** (r), which is a benchmark of the agent's performance on the task at hand.

Time is discretised in timesteps, so that the environment's **state** is frozen at each frame until the next step is executed. The variables for a specific timestep t are denoted with a subscript (e.g. the state at a time t would be written as s_t), and the sequence of steps until the task is accomplished or failed is often called in ML literature an **episode**.

$$\left.\begin{array}{ll} \text{States:} & s_t \in S \subseteq \mathbb{R}^{d_s} \\ \text{Observations:} & o_t \in O \subseteq \mathbb{R}^{d_o} \\ \text{Actions:} & a_t \in A \subseteq \mathbb{R}^{d_a} \\ \text{Rewards:} & r_t \in \mathbb{R} \end{array}\right\} \; \forall\, t,\; d_s \geq d_o \tag{2.1}$$

Algorithm 2.1 describes the flow followed by any implementation of these environments for each episode.

Algorithm 2.1 Episode of a simulation environment

```
t = 0
Start the environment from an initial state
while state ≠ "terminal" do
    for agent = 1, 2, ..., N do
        Fetch the agent's reward r_t
        Take an observation o_t
        Compute an action a_t
        Execute the action in the environment
    end for
    Update the environment's state, based on the actions taken
    t ← t + 1
end while
```

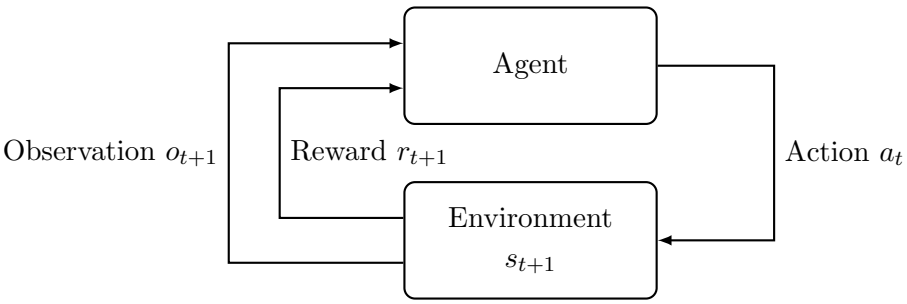


Figure 2.1: Diagram of a simulation environment.

This design choice follows practices that are very well-established in the reinforcement learning research community, and has been chosen for its flexibility and fitness for the problem at hand.

2.1.3 OpenAI Gym

OpenAI Gym [18] is a toolkit that was originally focused on the development of reinforcement learning algorithms. Its core interface, named by its creators the *unified environment interface*, has now become the *de facto* way to implement simulated environments for reinforcement learning research. The main advantage of OpenAI's environments is that they expose a consistent application programming interface (API), allowing for out-of-the-box intercompatibility with other software.

The contract behind these environments —which are modelled just as described in Subsection 2.1.2— is given by the `gym.Env` abstract class, and it can be synthesised as follows:

- **Methods**
  - `reset()`: restores the environment to an initial state, and returns the corresponding observation.
  - `step(action)`: runs one timestep of the environment's dynamics after the agent has taken $a_t =$ `action`. It should return a tuple with o_{t+1}, r_{t+1}, whether the environment reached a terminal state, and a `dict` with additional information.
  - `render(mode)`: creates a render of the environment, which can be used to debug its state or as a component of the observations. The `mode` indicates whether the returned output should be adapted for humans, or if it should be an RGB/text representation.
  - `close()`: destroys the environment and performs any necessary clean-ups.
  - `seed(seed)`: sets the seed for any pseudo-random number generator that might be used in stochastic processes within the environment.
- **Attributes**
  - `action_space`: an object describing the set of possible actions that can be taken by the agent (A).

- `observation_space`: same, for the observations of the environment (O).
- `reward_range`: tuple describing the minimum and maximum possible values of the reward.

2.1.4 Architecture

As the foundations for the environments used in this BEng Thesis, a set of base classes have been created to make the overall codebase more extensible, all of which are located in the `brain/environments/` directory:

- `World`: this class represents a simulated scenario. It configures the basic parameters of the simulation like the gravity and the duration of each timestep.
- `Scene:` a `World` containing a solid plane in $Z = 0$ that acts as the ground.
- `Body`: an abstraction of an object (also known as body) in a PyBullet simulation. It has a unique identifier given by the Bullet server, and a set of associated attributes like a visual shape, a collision box, a position and an orientation.
- `BaseBulletEnv`: it is responsible for instantiating PyBullet's client-server connection and setting up the rendering parameters (viewport size, camera rotation, etc.). It is an actual child of the `gym.Env` class, so this class and all of its children conform to the unified environment interface. It is an abstract class, thus requiring to implement certain methods to create a usable environment (see Section 2.3).

2.2 Franka Emika Panda™ 2 robot arm

Developing a robotic motor control system that is capable of solving challenging problems is no easy task at all, especially when aiming at a generic methodology that can be applied to different robot morphologies. To make the task more manageable, this BEng Thesis has been focused on the control of robotic arms.

These highly versatile devices mimic the functionalities of a human arm by having multiple segments or *links* united by joints, in the same way that a human arm has bones connected by articulations. These joints are often actionable, allowing to move parts of the arm in different axis and directions; and the state of the arm can be described by the positions of its joints. These are commonly known as a **degrees of freedom**.

In order to assess the performance of the controllers developed in this project, the Franka Emika Panda™ 2 robot arm has been chosen as the device of study. This robot has 7 degrees of freedom, which have been named as described in Figure 2.2 and allow reaching a workspace like the one depicted in Figure 2.3. The specific model used for the experiments also features a two-finger gripper (see Figure 2.4), which provides two additional degrees of freedom (one per finger) to the overall system. All the joints with their action ranges are listed in Table 2.1.

Franka Emika provides the assets required to simulate the arm in the main robotics platform, including the files that indicate the geometry, action ranges and other

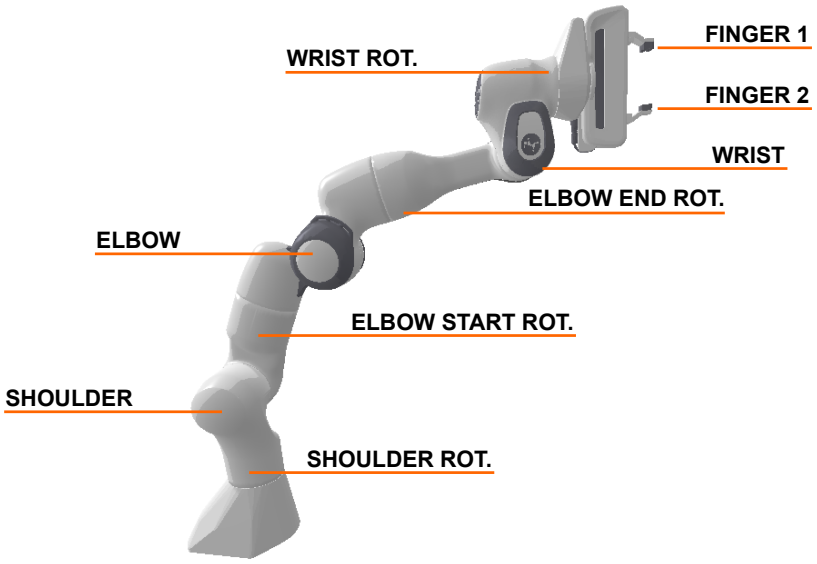


Figure 2.2: Render of the simulated Franka Emika Panda™ 2 robot with the name of all of its 9 degrees of freedom.

parameters of the arm. The original files are provided in the Xacro format, which is a variation of the Extensible Markup Language (XML) that allows for macros that expand certain expressions to larger XML blocks. Xacro[4] is widely used across the robotics community as it is part of the Robot Operating System (ROS), a popular framework for robotic software. By using a tool built into ROS, the Xacro definitions can be transformed into the Unified Robot Description Format (URDF)[5], which is also based in XML and is partially supported by PyBullet.

The `brain.robots.Panda` class handles the creation of the robot's entity in the simulator, defines the initial position of all of the robot's joints, and provides a myriad of auxiliary methods to ease the control of the robot.

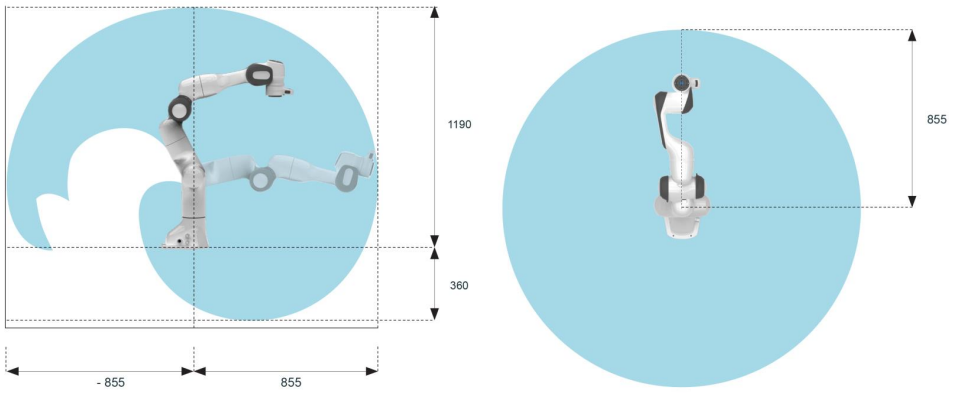
2.3 Available environments

To simulate and study the motor tasks in this BEng Thesis with the Panda™ 2 robot arm, two environments have been developed by leveraging the architecture outlined in Subsection 2.1.4.

The first one, `PandaEnv`, is an environment in which there is only a ground plane and a Panda™ 2 robot arm fixed to the Cartesian coordinate origin.

- **Initial state:** it is given by the arm's resting position as defined in the `brain.robots.Panda` class, therefore being always the same for all episodes.

[4] `http://wiki.ros.org/xacro`
[5] `http://wiki.ros.org/urdf`



(a) Lateral view. (b) Zenithal view.

Figure 2.3: Illustrations of the Franka Emika Panda™ 2 robot arm's workspace. All dimensions are in mm. Source: Franka Emika.

Figure 2.4: Franka Emika Hand, the gripper used as the end effector in the simulated robot arm. Source: WiredWorkers.

- **Observation space:** a flattened vector with the current positions of each actionable joint, as well as the torque being currently applied:

$$\left.\begin{aligned}\theta_t &= (\theta_{\text{joint1}-t}, \theta_{\text{joint2}-t}, \ldots)\\ \tau_t &= (\tau_{\text{joint1}-t}, \tau_{\text{joint2}-t}, \ldots)\end{aligned}\right\} \ \forall \text{ joint in actionable joints} \tag{2.2}$$

$$o_t = (\theta_t, \tau_t) \tag{2.3}$$

- **Action space:** the action range of the 9 actionable joints (see Table 2.1).
- **Termination criterion:** the episode is considered as finished only if the arm crashes with the floor or with itself (i.e. collisions between parts of the arm).

The second one, `PandaTargetEnv`, is an extension of the previous environment that also includes the concept of a *target*. Targets are locations that act as goals for the motor controller. They are represented by a translucent red sphere and they do not have a collision box, which means they can be pierced through without interfering with the arm's movement.

		Position		Velocity
Joint	**Type**	**Minimum**	**Maximum**	**Max. magnitude**
Shoulder rot.	Revolute	-2.9671 rad	2.9671 rad	2.175 rad/s
Shoulder	Revolute	-1.8326 rad	1.8326 rad	2.175 rad/s
Elbow start rot.	Revolute	-2.9671 rad	2.9671rad	2.175 rad/s
Elbow	Revolute	-3.1416 rad	0 rad	2.175 rad/s
Elbow end rot.	Revolute	-2.9671 rad	2.9671 rad	2.610 rad/s
Wrist	Revolute	-0.0873 rad	3.8223 rad	2.610 rad/s
Wrist rot.	Revolute	-2.9671 rad	2.9671 rad	2.610 rad/s
Finger 1	Prismatic	0 mm	40 mm	20 mm/s
Finger 2	Prismatic	0 mm	40 mm	20 mm/s

Table 2.1: List of actionable joints in the simulated Franka Emika Panda™ 2 robot, with their type and movement constraints as defined in the simulation's URDF file. *Revolute* joints allow rotation along an axis, whereas *prismatic* joints provide linear sliding movement.

`PandaTargetEnv` environments behave just like `PandaEnv` ones, with the exception that targets are randomly sampled by default from a space reachable for the arm:

$$T = (x_{\text{target}}, x_{\text{target}}, z_{\text{target}}) \begin{cases} -0.5 \leq x_{\text{target}} < 0.5 \\ -0.5 \leq y_{\text{target}} < 0.5 \\ 0 \leq z_{\text{target}} < 0.5 \end{cases} \tag{2.4}$$

And that the observation space includes not only the contents described in Equation 2.3, but also the target's location in Cartesian coordinates:

$$o_t = (\theta_t, \tau_t, x_{\text{target}}, y_{\text{target}}, z_{\text{target}}) \tag{2.5}$$

It is important to note that `PandaEnv` (and therefore `PandaTargetEnv`) allows controlling the Panda™ arm with two kinds of settings either the **target position** (the angle where each joint should be), or the **target velocity** (how fast should each joint move). Even though both are relatively similar concepts, one option might be better than the other when building the intelligent motor controllers. The right choice depends on the controller's inner workings, and whether it is aware of the state of the environment over time.

This is because position control can potentially be used without the need of simulation in the domain of time, whereas velocity control is intrinsically dependent on the temporal axis. Reaching a static target location can therefore be done with a single command in position control, whereas velocity control has to progressively adapt its outputs until the target is reached (leading to an equilibrium with null velocity in all joints).

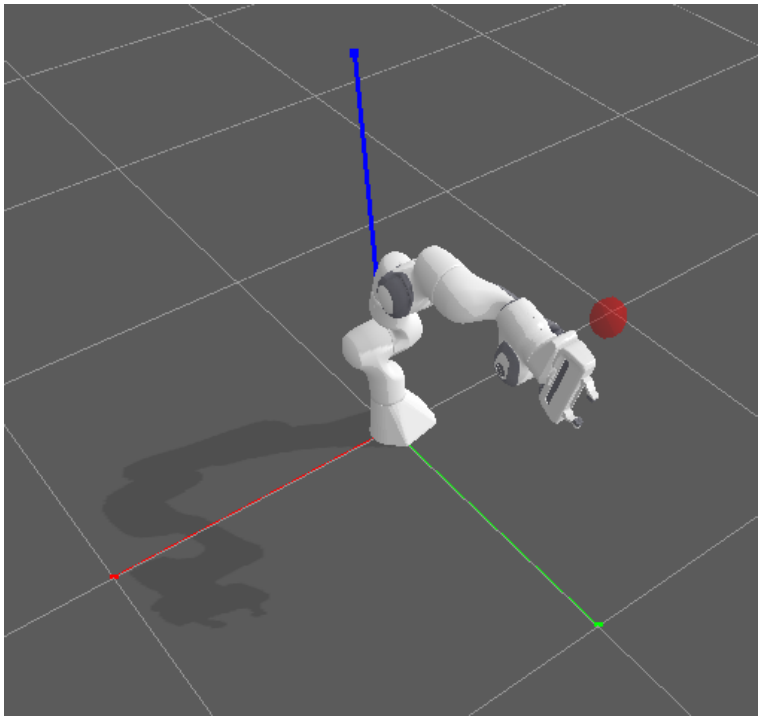


Figure 2.5: Example of a `PandaTargetEnv`. The red sphere represents the randomly chosen location for the target in the episode displayed.

2.4 Reproducibility

Randomness can create reproducibility issues, because running the same experiment in two different machines or points in time can lead to inconsistent results. `PandaTargetEnv` is the only environment that has a random component, as the targets are randomly selected by default from a constrained space.

To avoid this, the locations of the targets are generated using the Mersenne Twister [19] general-purpose pseudo-random number generator (PRNG). The values produced depend on a initial seed value that is fixed across experiments, making the sequence of randomly generated values equal across different executions.

All the seeds used in this environment are explicitly set and documented in the project, allowing for third-party researchers to identically reproduce the samples used for this BEng Thesis, as well as to generate new ones in a deterministic manner.

During the development of this Thesis it has also been noted that PyBullet itself can introduce some noise when performing dynamic simulations, especially when collisions take place. However, the corresponding noise is relatively insignificant and completely deterministic across runs, so it should not constitute an issue for reproducibility.

Lastly, project dependencies are as well a frequent source of issues when running third-party code. To prevent this, the Thesis's code makes use of Poetry [20], a dependency management tool for Python that creates a lock file with a record of all the installed dependencies, their versions, and a hash of their original binaries. This ensures that other people installing the project will be able to have a virtually

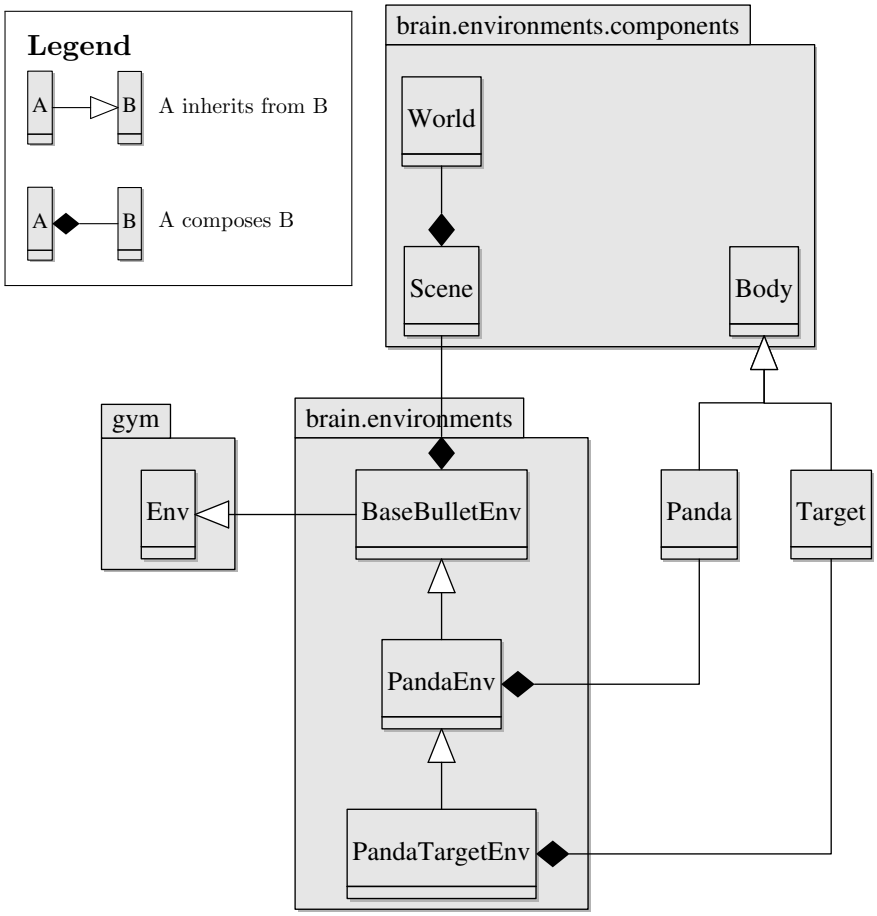


Figure 2.6: UML diagram of the Thesis's environment framework.

identical development environment as the author's.

2.5 Time-invariant environments

Although the simulation environments used in this BEng Thesis are intrinsically dynamic, there might be some experiments in which the time component can be dismissed. This means that the environment state that is relevant for the analysis can be obtained by taking a single action, and then evaluating it to produce one single final output.

Such a situation can happen for instance when studying the performance of an inverse kinematics controller that uses position control, as the benchmark of interest (the final position of the arm's actuator) can be computed directly just with the target position of the joints.

PyBullet offers for these situations the possibility of obtaining the final state of the environment without the need of computing the middle states, significantly decreasing the computation power required to perform the simulation and therefore reducing the execution time. This technique —which the author has named Reset-Based Simulation (RBS)— has been able to make certain simulations up to 10 times faster, and can be used directly over the already existing environments described in

Section 2.3.

The motor control systems that perform inverse dynamics need to know the evolution of the environment over time, and therefore are not compatible with RBS.

Chapter 3

Experiments

The versatility of artificial neural networks has given birth to plenty of new approaches to machine learning problems, each with different features and a particular adequateness. Three different ones have been selected for this BEng Thesis: supervised learning, black box optimisation, and reinforcement learning.

An experimental set-up has been created for each one of them, with the shared goal of reaching randomly located targets in a three-dimensional space. Having a common challenge and similar characteristics allow easily comparing the experiments by using three benchmarks:

- **Resource efficiency:** the amount of computing resources that are required to train the motor controller, and how performant is the model once trained.
- **Accuracy:** the average error tolerance of the resulting model, i.e. the difference between the position of the target T and the arm's position after following the controller's commands $\hat{T}$. This has been quantified with the mean squared error, which is defined as follows for a set of $(T, \hat{T})$ pairs with cardinality n:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(T_i - \hat{T}_i)^2 \tag{3.1}$$

- **Morphology generalisation:** the maximum number of degrees of freedom that can be handled by the model.

To assess the morphology benchmark and to keep the development of the motor controllers' incremental, only a subset of the 11 degrees of freedom available in the simulated robot arm have been enabled throughout these experiments. This was achieved in a progressive fashion by enabling a single degree of freedom while keeping the remaining ones locked in their initial position, and then adding more DoFs until the performance was noticeably diminished.

3.1 Supervised learning approach

3.1.1 Context

Supervised learning is a machine learning technique that allows generalising the mapping between a series of inputs and outputs that are already matched. It is

commonly used nowadays in problems such as regression, image classification and time series forecasting, amongst others.

From a formal perspective, it is assumed that the dataset under study has its outputs or labels correlated with the corresponding inputs by an unknown function f, such that $f : X \to Y$, where $X \subseteq \mathbb{R}^m$ and $Y \subseteq \mathbb{R}^n$, $m, n \geq 1$ are the input and output spaces, respectively. The goal of the supervised learning system is to obtain another function $\hat{f} : X \to Y$ that behaves as close as possible to the original f. This concept of "closeness" is defined by what is commonly known as a **loss function** ℓ:

$$\left.\begin{matrix} f(x) = y \\ \hat{f}(x) = \hat{y} \end{matrix}\right\} \ell(y, \hat{y}) \tag{3.2}$$

$$x \in X,\ y \in Y \tag{3.3}$$

where ℓ frequently maps to a real number and $\hat{y}$ is called the "predicted value" for an input x.

Supervised learning algorithms attempt to find $\hat{f}$ by using samples of the mapping made by the actual f. These samples constitute what is often called a **dataset**:

$$D = \{(x, y), \quad y = f(x)\ \forall\ x \in X_{\text{dataset}} \subseteq X\} \tag{3.4}$$

3.1.2 Regression with artificial neural networks

There are many approaches to find $\hat{f}$ within the context of supervised learning, with some significant differences depending on whether Y is a continuous space (making the problem a regression one) or a discrete space (making the problem a classification one). Here we will focus on regression.

One of the most basic solutions is linear regression, but there are more complex alternatives like Bayesian regression or the nowadays widespread deep neural networks. This experiment is focused on the latter because of their power, robustness and ease of implementation.

To achieve this type of regression, the whole neural network's transfer function acts like $\hat{f}$, and the network's parameters are adjusted to minimise the loss function. This is performed by taking a set of samples of the dataset (called a *batch*) and evaluating the network's average performance using ℓ. The network's parameters are then tuned using the gradient of this averaged loss. Unsurprisingly, the choice of ℓ and the definition of how should the network's parameters evolve with the gradient are two important factors that can heavily impact the performance of the resulting model.

3.1.3 Definition

A certain resemblance can be found between the supervised learning scenario and the one that is tried to be solved with inverse kinematics: if we assume that each Cartesian coordinate maps to a single robotic generalised coordinate, the robot's behaviour could be given by a function like f. In that case, the Cartesian coordinate space would be equivalent to X, and the generalised coordinate space would be equivalent to Y.

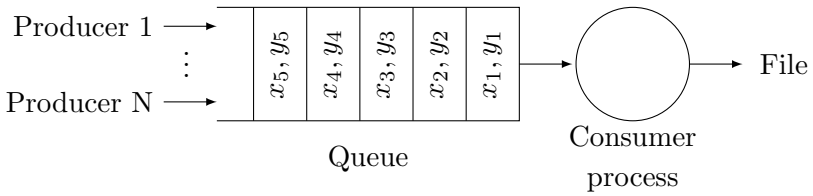


Figure 3.1: Producer-consumer model followed by the dataset generator.

Even though this premise does not usually hold for complex robot arms with several degrees of freedom like the Panda™ 2, a proposal is made in 3.1.4.2 on how to address this issue, allowing to get unique values when a single Cartesian coordinate has more than one generalised solution.

3.1.4 Methodology

3.1.4.1 Dataset generation

There is an initial dissonance when applying supervised learning to the inverse kinematics problem, created by the lack of the initial dataset that is critical for supervised learning. However, there is a particularity in the IK problem, and it is the fact that the function f^{-1} can be evaluated easily: it is the equivalent of transforming generalised coordinates to Cartesian coordinates, or in other words, choosing a position for the arm's joints and measuring where the actuator is placed. This is commonly known as **forward kinematics**.

It is therefore very simple to create a dataset D like the one depicted in Equation 3.4, by transforming it as follows:

$$D = \{(x, y), \quad x = f^{-1}(y) \ \forall \ y \in Y_{\text{dataset}} \subseteq Y\} \tag{3.5}$$

where Y_{dataset} can be randomly sampled from Y.

As it always happens with supervised learning, it is important that the dataset has enough samples so that it captures sufficient information about the system's kinematics: the larger $|Y_{\text{dataset}}|$ is, the better generalisation capabilities the model will have.

A parallelised simulation tool has been designed to create these datasets, allowing to generate a big amount of samples in little time. This tool spawns multiple workers, each one with its own process and seed, and starts creating (x, y) tuples with pseudo randomly selected values of y. It implements a producer-consumer model: the tuples get pushed to a queue that is shared by all the processes, and they are collected by a separate process that is responsible for storing them in the disk using the HDF5® [21] data format (see Figure 3.1). The main advantage of such a design is that multiple simulations can be executed at the same time, which makes an extremely noticeable difference in hosts with multi-core CPUs. This, combined with the fact that it allows using RBS as explained in Section 2.5, yields a high sample production throughput.

With a large enough number of samples, the resulting datasets are well balanced and they fully cover the arm's action space, as displayed in figures 3.2 and 3.3.

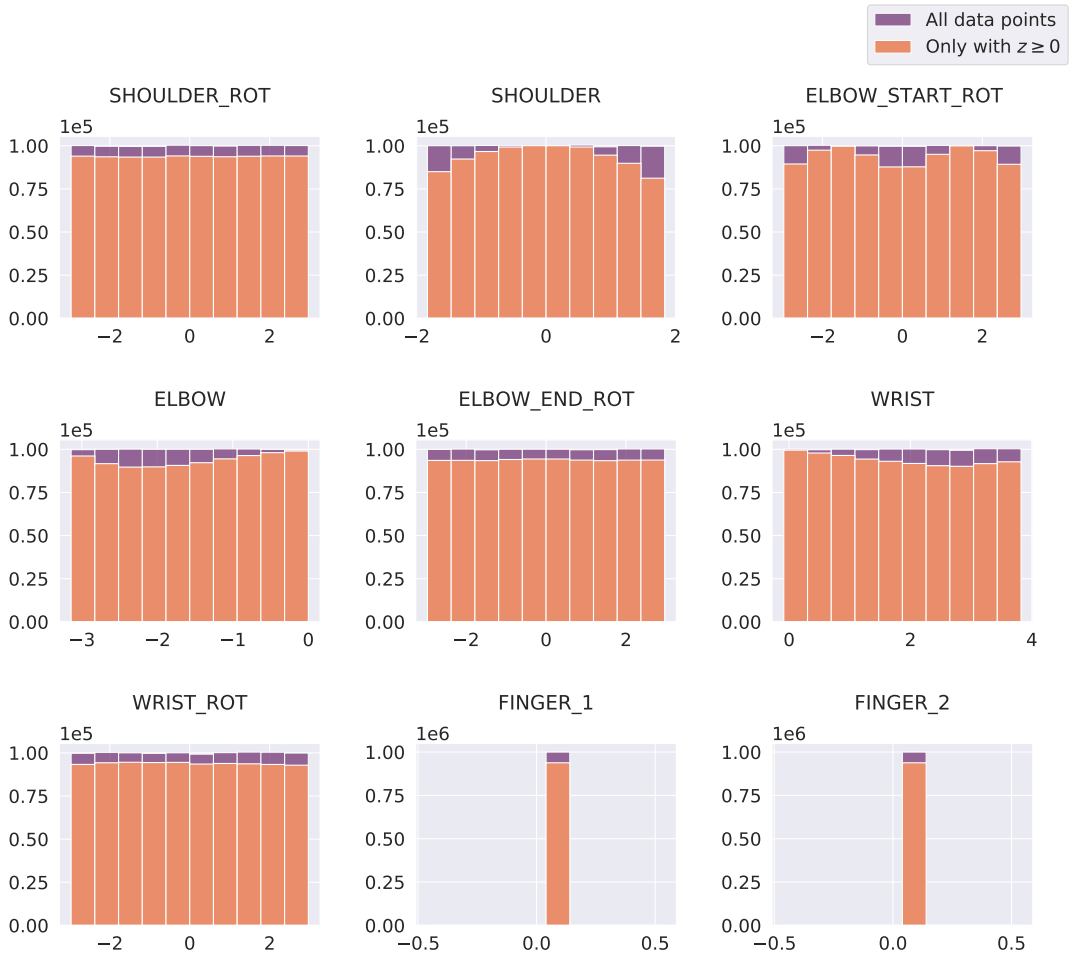


Figure 3.2: Histogram of the joint positions sampled in a randomly generated dataset of 10^6 samples and the gripper's fingers locked. The original dataset (in purple) contained samples where the gripper's position was under the X-Y plane because of PyBullet ignoring physics simulations (a side effect of RBS). The orange bars represent the dataset after discarding those entries, which constitute approximately 6% of the total. As expected, all the DoFs enabled have a uniform distribution in the original dataset, and they remain relatively balanced in the pruned one.

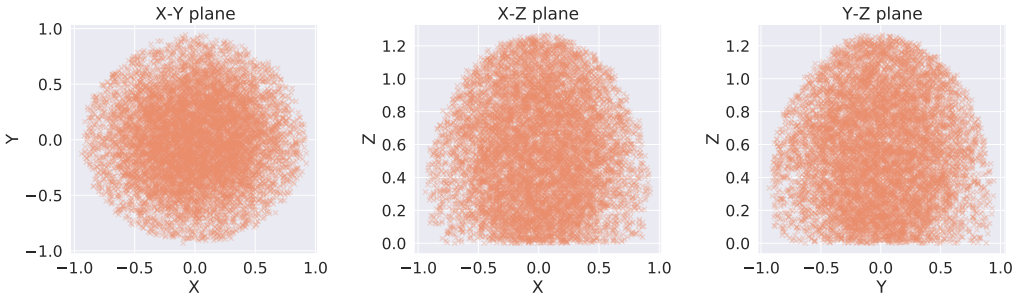


Figure 3.3: Orthogonal projections of the gripper positions in a randomly generated dataset of 10^6 samples with all the degrees of freedom but the fingers unlocked. Subsampled to 10^4 data points for enhanced visibility.

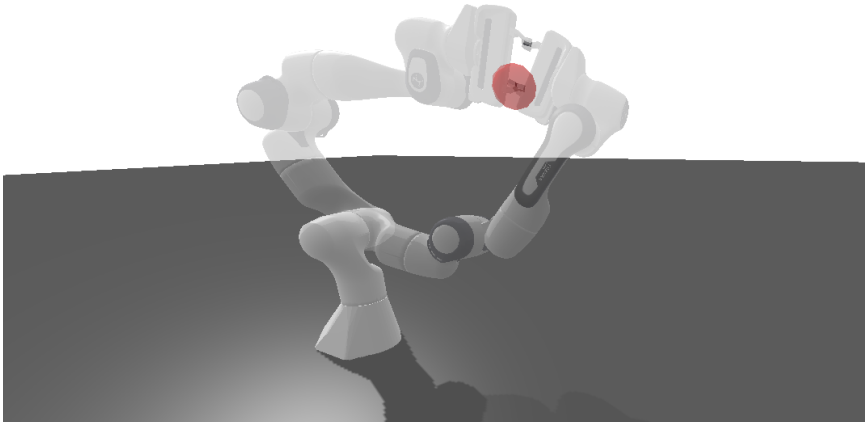


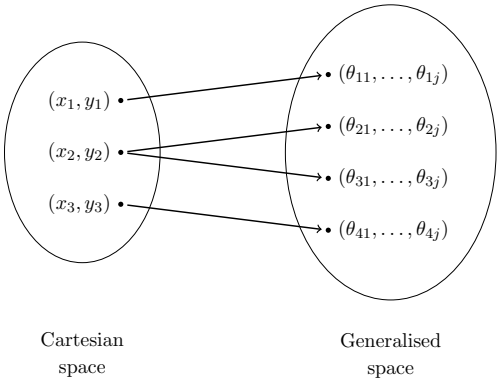
Figure 3.4: Example of the arm's gripper reaching the same location (red ball) with two different sets of joint positions.

3.1.4.2 Achieving univocity

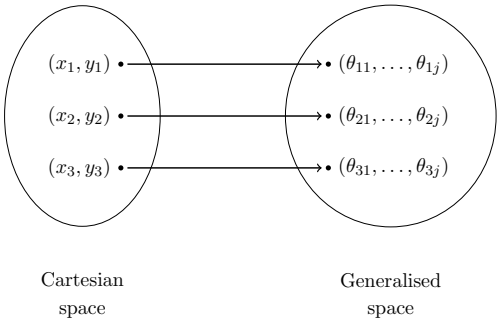
Preliminary results of this experiment created a model that barely moved the arm. In fact, it produced almost the same output for all inputs, which was keeping the arm in close alignment with the Z (vertical) axis. However, upon reducing the enabled degrees of freedom to e.g. `SHOULDER` and `ELBOW`, the model behaved correctly.

The issue was not a matter of the model's generalisation capabilities or the architecture's size, but something more fundamental: when many degrees of freedom were enabled, it was possible to reach the same point in space with multiple sets of generalised coordinates (like the example in Figure 3.4). This breaks the initial premise of supervised learning that we are performing, because that behaviour no longer fits the definition of a mathematical function.

The author's hypothesis for these results is that data points of incoherent solutions—like the example before—were producing gradients that cancelled each other when the neural network performed the backpropagation to update its weights.



(a) Original.



(b) After picking the least energy-consuming option for each domain value.

Figure 3.5: Mapping diagram of the inverse kinematics before and after considering energy limitation.

This theory is reinforced by the fact that limiting the arm's movement to non-overlapping degrees of freedom removes the problem entirely.

To address this conflict without limiting the degrees of freedom, an algorithm has been designed to guarantee that each Cartesian-space coordinate maps to a single point in the generalised space. This technique works by taking the option that requires the least energy to execute, as depicted in Figure 3.5. Energy is calculated in each joint by taking the work W done by some torque τ:

$$W = \int \tau(t) \cdot \mathrm{d}\theta \tag{3.6}$$

$$\overset{\theta=\omega t}{=} \int \tau(t) \cdot \omega(t) \cdot \mathrm{d}t \tag{3.7}$$

Since time is discretised in the simulations and the torque can only be measured within timesteps, both torque $\tau(t)$ and angular velocity $\omega(t)$ are assumed to be constant for each timestep Δt:

$$W \overset{\Delta t \approx 0}{\approx} \sum_{N} \tau(N \cdot \Delta t) \cdot \omega(N \cdot \Delta t) \cdot \Delta t \tag{3.8}$$

$$= \sum \tau[t] \cdot \omega[t] \cdot \Delta t \tag{3.9}$$

By knowing the energy required to reach a certain set of joint positions from the resting position, it is now possible to choose the most efficient generalised coordinates to get to a specific Cartesian point. This has been named the **efficient dataset**.

We can express the process in a more mathematically formal way. Let $k : G \to C$ be the forward kinematics function, where $G \subset \mathbb{R}^j$ is the generalised coordinate space (with $j = 11$ for the simulated Panda™), and $C \subset \mathbb{R}^3$ is the 3D space reachable by the arm's gripper. By using the original definition of the dataset given in Equation 3.5, we can define a new dataset D_{energy} that also includes the energy required to reach each data point:

$$D_{\text{energy}} = \{(c, \theta, W_\theta), \quad c = k(\theta) \ \forall \ \theta \in G_{\text{dataset}} \subseteq G\} \tag{3.10}$$

Where W_θ is the work required to reach the joint positions θ from the arm's resting state as per Equation 3.9. In the efficient dataset, out of all the n possible generalised coordinates $\theta_1, \ldots, \theta_n$ that lead to each cartesian point c, it is only preserved each tuple that satisfies:

$$(c, \theta) : W_\theta = \min(W_{\theta_1}, \ldots, W_{\theta_n}) \tag{3.11}$$

Recall that G_{dataset} in Equation 3.10 is a finite set of random samples of G, which is a side effect of not being able to work with the latter in the actual implementation due to its infiniteness. This means that neither D nor D_{energy} include all possible $\theta_1, \ldots, \theta_n$ for a certain c.

This can lead to inconsistent behaviour, since two very close points could have radically different values of θ in the efficient dataset. To solve it, the implementation groups all entries in D_{energy} that are very near each other in the Cartesian space, and considers them as the same point c. The full procedure is described in Algorithm 3.1, and a visual example is showcased in Figure 3.6. As depicted in Figure 3.7, the result matches the expectations: a dataset that is quite uniform in the use of energy throughout all of its points.

Algorithm 3.1 Generation of an efficient dataset

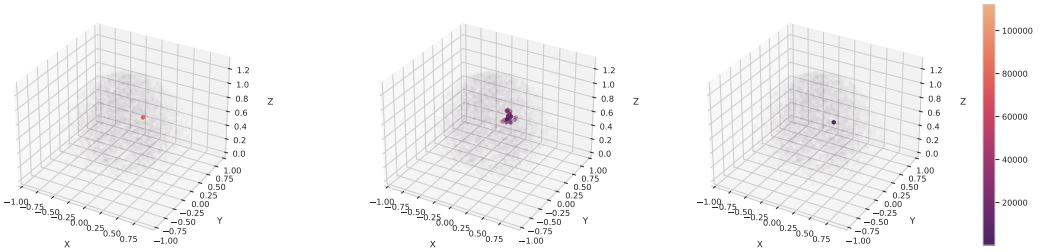
1: Take a dataset D_{energy}
2: $D_{\text{efficient}} = \varnothing$
3: **while** $D_{\text{energy}} \neq \varnothing$ **do**
4: Draw a random sample (c, θ, W_θ) from D_{energy}
5: Take all the entries in D_{energy} such that $|c_i - c| < \varepsilon$
6: Add to $D_{\text{efficient}}$ the entry with the minimum W_θ
7: Remove all the entries selected in this iteration from D_{energy}
8: **end while**

3.1.4.3 Network architecture

The neural network architecture used for this experiment was a multi-layer perceptron with 3 inputs (the Cartesian coordinates of the target), two hidden layers of 256 and 128 neurons respectively, and a final layer with as many neurons as degrees of freedom were enabled, as depicted in Figure 3.8.

The activation function each neuron in the input and output layers is a linear one ($f(x) = Ax + B$), whereas the hidden layers use the sigmoid function ($f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$). The output layer is then connected to the motor controller, which denormalises the output values and sends them to the actor's actuators.

Even though multiple other architectures were studied in this experiment, they did not show a significant performance enhancement. The architecture proposed here offers a good balance between network capacity and adequate use of computation resources.

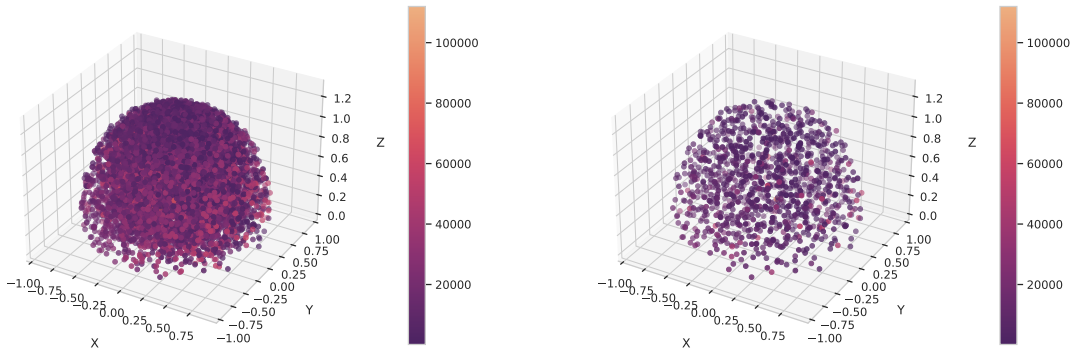


(a) Random point from the original dataset.

(b) Cluster of all points in the original dataset within a radius of $\varepsilon = 0.1$ of the randomly selected point.

(c) Point in the cluster with the least energy required.

Figure 3.6: Steps taken in a example pruning batch of the many that transform D_{energy} into $D_{\text{efficient}}$. All original points in D_{energy} are displayed with low opacity in the background as a reference.



(a) Points in an original D_{energy} with 10,000 samples.

(b) The same dataset after the optimisation process ($D_{\text{efficient}}$).

Figure 3.7: Distribution of the data points in the Cartesian space before and after optimisation for an example dataset.

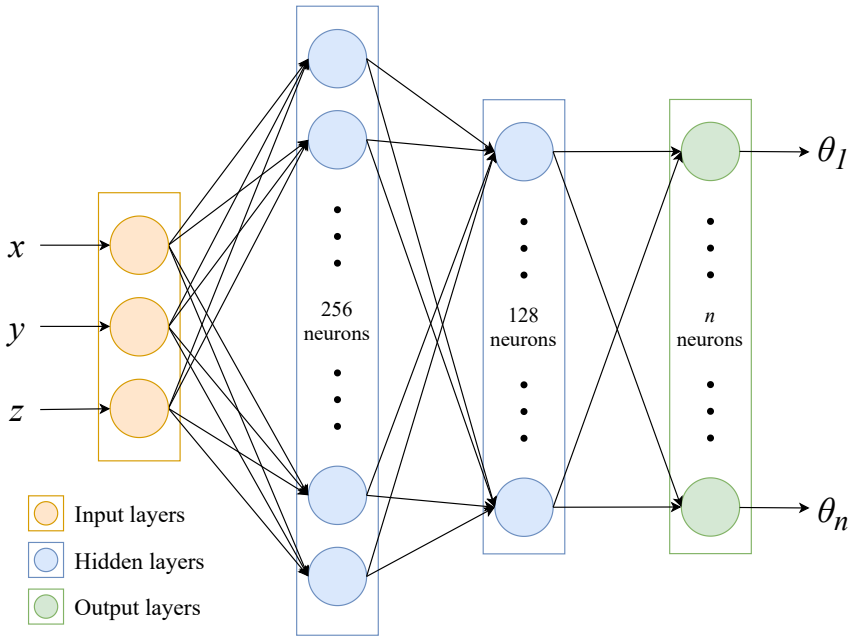


Figure 3.8

3.1.5 Training and results

After implementing the architecture in PyTorch [22], an initial training was made with a plain dataset D. The training was executed with batches of 10^5 samples using datasets of 10^6 entries with different degrees of freedom enabled. The Adam optimiser [23] was used with an initial learning rate of 0.01, and it was progressively reduced an order of magnitude every time the loss stagnated (i.e. changed less than 10^{-4} for 1000 epochs). Training for 10k epochs took about 2 minutes per model, and it was more than enough to reach convergence as showcased in Figure 3.9.

As it was mentioned in Section 3.1.4.2, the results with this dataset were very good when the number of degrees of freedom did not overlap in the Cartesian space, but they became much worse when adding extra DoFs. This is also visible in Figure 3.9 and when running the models in inference: the one capable of using `SHOULDER` and `ELBOW` still works well, but as soon as the rotation component is added the errors start being quite noticeable. These errors were not reduced by increasing the ANN's architecture size, which suggested that the problem was indeed in the dataset.

The same experiment was then repeated using this time the efficient dataset, resulting in the training curves in Figure 3.10. Unfortunately, the models trained with this dataset did not perform better: in fact, even the simplest cases (e.g. only `SHOULDER` enabled) performed worse than with the normal dataset D. Since the algorithm that generates the efficient dataset does not alter the original samples but drops some of them, it might be caused by too much information getting lost along the process. It is however interesting to note that reducing the radius ε (which therefore increases the size of the resulting $D_{\text{efficient}}$) did not improve the results but quite the opposite.

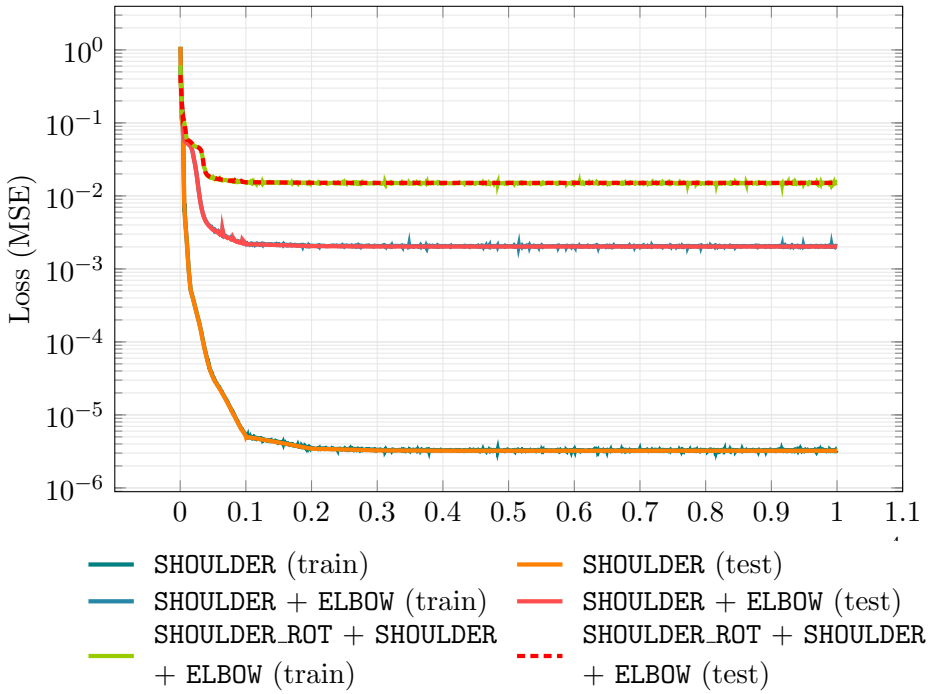


Figure 3.9: Losses for the train and test splits of regular datasets D with different degrees of freedom enabled.

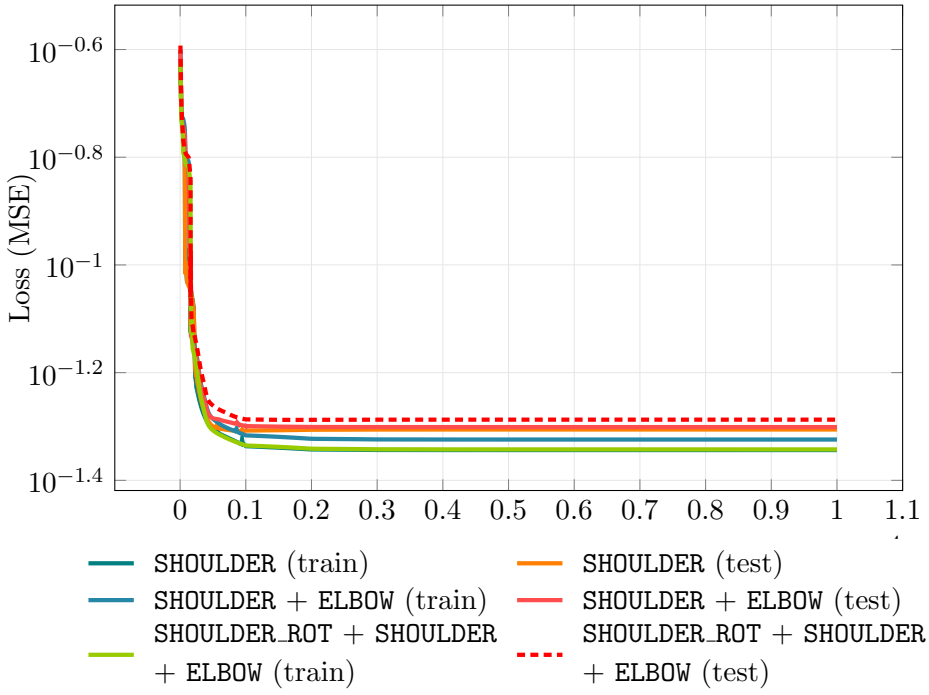


Figure 3.10: Losses for the train and test splits of efficient datasets $D_{\text{efficient}}$ with different degrees of freedom enabled.

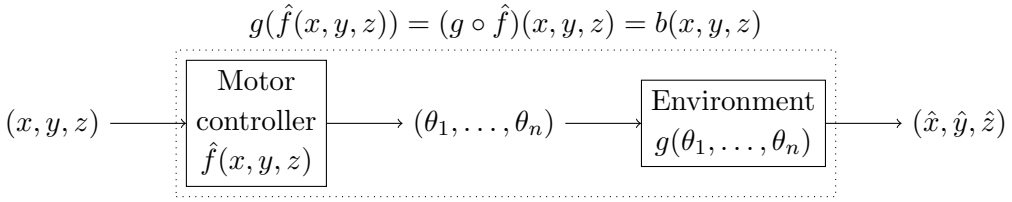


Figure 3.11: Model of the inverse kinematics problem as a chain of functions.

3.2 Black box approach

3.2.1 Definition

If we model the whole inverse kinematics system as a set of chained functions like in Figure 3.11, we see that the problem we are trying to solve can also be seen as finding a function $\hat{f}$ such that $(g \circ \hat{f})(x,y,z) = (x,y,z)$. Or, more simply put, an $f = g^{-1}$, where g is a **black box function**: we only know its output for a given input (by simulating it), but we do not have an analytical expression for it. Solutions like the ones presented in Section 1.5.1 attempt to find it by using differential equations that are very complex, but as it was explained in Section 3.1.2, artificial neural networks can also be used to approximate the function f in a way that the resulting estimate function ($\hat{f}$) is defined by the network's architecture and the parameters obtained during training.

However, training in mainstream ANN programming frameworks is based on automatic differentiation (AD) [24] to perform backpropagation. This technique works by generating a computation graph with the definitions of all functions involved in the training process and their derivatives, which allows calculating the training gradients very easily without the precision loss and subsequent error propagation that is often associated with numerical differentiation.

AD offers many advantages and is compatible with most of machine learning applications, but it has the inconvenient that black box functions are not supported in the computation graph (as nor their analytical definitions nor their derivatives are available). The previous experiment tried to overcome this issue by sampling the function and using a dataset to capture its mappings, but as we have seen, the results still have room for improvement.

An alternative approach is to replace the AD engine with an alternative algorithm that adjusts the parameters that define $\hat{f}$. Random parameter search can be an option, but as soon as the number of parameters start growing, it becomes a totally impracticable path. Nonetheless, there are other strategies for black box optimisation problems that can be applied here, such as particle swarm optimisation [25] or natural evolution strategies [26]. Even though these techniques are not as widespread in the machine learning community as those based in AD, they can provide better training speeds on multi-core machines because they are compatible with parallelisation [27].

3.2.2 Methodology

3.2.2.1 Parameter optimisation

The same neural architecture as in Figure 3.8 was used in this experiment, but in this case the network's 34,178 parameters were externally adjusted using other Python modules and then re-loaded into the model. The parameter adjustment part was implemented using with two different third-party modules, each of them using a different optimisation technique:

- **SciPy [28]:** this popular scientific computing package has an optimisation submodule that supports a wide array of algorithms. The one used here (`scipy.optimize.differential_evolution` uses differential evolution [29]).
- `blackbox` **[30]:** an optimisation module that also supports parallelisation.

3.2.3 Training and results

The training for this experiment follows the steps described in Algorithm 3.2, where the optimiser$(L, \omega_1, \ldots, \omega_n)$ function is given by SciPy/`blackbox` and it progressively adjusts the ω parameters so that L is minimised. Unfortunately, after approximately 30h of execution in a 20-core machine, the results with both optimisation modules were very poor: there was not a noticeable difference between the model's control logic and a totally random one.

Algorithm 3.2 Adjustment of the inverse kinematics neural network with black box optimisation

Initialise the neural network with some arbitrary parameters $\omega_1, \ldots, \omega_n$
for episode $\leftarrow$ 1 to 2000 **do**
    $L_{\text{episode}} \leftarrow 0$
    for batch $\leftarrow$ 1 to 10 **do**
        Generate a random target point in the Cartesian space (x, y, z)
        Compute $(\theta_1, \ldots, \theta_n) = \hat{f}(x, y, z)$ with the neural network
        Get $(\hat{x}, \hat{y}, \hat{z})$ by simulating the robot arm with those joint positions
        if arm crashed **then**
            $L_{\text{episode}} \leftarrow L_{\text{episode}}+$ crash penalty
        else
            $L_{\text{episode}} \leftarrow L_{\text{episode}} + |(x, y, z) - (\hat{x}, \hat{y}, \hat{z})|$
        end if
    end for
    $\omega_1, \ldots, \omega_n \leftarrow$ optimiser$(L, \omega_1, \ldots, \omega_n)$
end for

It is reasonable to expect that, given more time and computational resources, the neural network would eventually converge to a model that fulfils the motor controller's requirements: after all, it is more than 30k parameters that have to be adjusted in a

very wide search space. However, considering the relatively small size of the problem, such a need of computational resources suggests that this is probably not the optimal approach.

3.3 Reinforcement learning approach

3.3.1 Definition

Just like the simulations in this BEng Thesis, reinforcement learning (RL) is based on the concept of an environment that can be observed and acted upon by an agent. RL's goal is to define which actions are the best to maximise the reward obtained by the agent when the environment has a certain state, in what is commonly known as a **policy**).

Finding good policies is quite a hard task, and it is a topic under very active development in the RL research community. Out of the many different algorithms to elaborate the policies, this experiment is focused in one called Proximal Policy Optimisation (PPO) [32]. PPO is compatible with continuous action spaces and it has shown a good performance in other robotic manipulation tasks [33, 34], making it a very appealing candidate for the motor control system in this BEng Thesis.

Just like most of RL optimisation algorithms, PPO models the environment as a Markov Decision Process (MDP). In this kind of processes the time is discretised, so the system's state changes in steps. The evolution of the state over time is stochastic and it follows the Markov property, which means that the probabilities of each possible transition depend exclusively on the current state and the last action taken. As a result, it is not necessary to know the environment's history to determine the probabilities of each potential transition. It is up to the agent to decide what action to take, but the outcome of that specific action has some uncertainty because of the stochastic nature of process. It is also worth highlighting that, because of the temporal awareness of MDPs, the problem solved with the RL approach is not strictly inverse kinematics but inverse dynamics, since the output of the motor controller are the velocities of each joint (instead of the static target joint positions).

PPO works by exploring the environment from time to time. Exploration is achieved by taking random actions (instead of the one that would predictably yield the highest reward), in the hopes that it discovers a new course of action that eventually gives an even better reward. This is an interesting aspect of PPO because that means that there is not a distinction between "learning" and "predicting", unlike with other ML techniques that have training and inference as two clearly separate stages.

Consequently, the policy is not deterministic, and therefore has a probability distribution that we denote as:

$$a_t \sim \pi_\theta(\cdot|s_t) \tag{3.12}$$

where s_t is the environment's state and a_t the action to be taken, both at a time t. The parameters that govern the policy (θ) are determined by a neural network, as it will be explained later on in this section.

The environment's initial state s_0 is sample from a distribution of all possible start states (ρ_0), which in the case of this experiment is given by the `PandaTargetEnv`

conditions outlined in Section 2.3. The sequence of actions taken and subsequent states within an episode is called a **trajectory**, and is denoted as:

$$\tau = (s_0, a_0, s_1, a_1, \ldots), \quad s_0 \sim \rho_0(\cdot) \tag{3.13}$$

We can therefore express the **probability of a specific trajectory** of T steps taking place as:

$$P(\tau|\pi) = \rho_0(s_0) \cdot \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \cdot \pi(a_t|s_t) \tag{3.14}$$

And if we define $R(\tau)$ as the **reward** produced by a trajectory τ, we can denote the goal of the RL system as finding the optimal policy π^* such that:

$$\pi^* = \arg\max_{\pi} \underset{\tau\sim\pi}{E} [R(\tau)] \tag{3.15}$$

In the case of PPO, like the Soft Actor-Critic [35] and other algorithms based in the actor-critic model, the agent has two different estimators, both parametrised by neural networks:

- The **critic**, which is responsible for estimating the expected reward that is going to be obtained. It can be seen as an assessment of how good or bad the current state is for future reward.

- The **actor**, which determines the probability distribution of the next move.

3.3.1.1 Biological parallelism in humans

Actor-critic models are not an exclusively synthetic artefact, purpose-built for RL problems. The human brain, which we can probably consider the ultimate idol for artificial general intelligence tool, also follows a similar principle in the basal ganglia.

Even though there is still a limited knowledge regarding the functionality of this part of the brain, is has been associated with the control of psychomotor behaviour [36] in a similar way to what the motor controllers in this BEng Thesis attempt to behave.

Furthermore, the cortico-basal ganglia-thalamo-cortical loop (also known as the CBGTC loop) is a set of neural circuitry that connects the basal ganglia to other parts of the brain. Some of these circuits are considered to work independently from each other, producing an equilibrium similar to the one achieved by actor-critic RL models.

Lastly, there is a significant difference in RL systems like PPO, caused by the fact that they do not distinguish between training and inference: it is much closer to the learning process humans follow. We interact with the environment as we try to solve some problem, and throughout repetition and the attempt of different (sometimes random) approaches we end up identifying valuable information about the dynamics of our surroundings.

3.3.2 Methodology

These experiments makes use of the PyTorch implementation of PPO from Stable Baselines 3 [37]. It is parallelised, allowing to run multiple simulations in different CPUs for a much higher training throughput.

The generality of reinforcement learning allows for a great flexibility when designing environment: there are plenty of orthogonal changes that can be made to the experiments, all of them with a significant potential impact in the results. In this series of trials, a focus has been put mainly in the action space (i.e. available degrees of freedom) and the reward function.

The first trials were made with the goal of lifting the gripper as much as possible, to verify that the RL agent was being properly stimulated by the reward. The reward function also included a penalty for the use of energy, so that the agent tries to avoid oscillatory behaviour and useless movements overall:

$$r_t = z_{\text{gripper}} - \frac{\text{clip}(\log 10(W_t), 0, W_{\max})}{W_{\max}} \cdot k \tag{3.16}$$

where W_t is the energy consumption during the t-th timestep as defined in in Equation 3.9, $W_{\max}$ the maximum energy consumed under normal operation conditions, k is a constant, and the "clip" function is defined as:

$$\text{clip}(x, \min, \max) = \begin{cases} \min, & \text{if } x < \min \\ x, & \text{if } \min \leq x \leq \max \\ \max, & \text{if } \max < x \end{cases} \tag{3.17}$$

For trials that involved reaching random targets, two different reward functions were used: one of them with energy penalty and another one without it:

$$\begin{aligned} r_t = 1 - |(x_{\text{target}}, y_{\text{target}}, z_{\text{target}}) - (x_{\text{gripper}}, y_{\text{gripper}}, z_{\text{gripper}})| \\ - \frac{\text{clip}(\log 10(W_t), 0, W_{\max})}{W_{\max}} \cdot k \end{aligned} \tag{3.18}$$

$$r_t = 1 - |(x_{\text{target}}, y_{\text{target}}, z_{\text{target}}) - (x_{\text{gripper}}, y_{\text{gripper}}, z_{\text{gripper}})| \tag{3.19}$$

All the trials described here make use of Adam [23] optimiser and the hyperparameters listed in Table 3.1. Additionally, it is important to highlight that `PandaTargetEnv`'s random target generation was constrained in certain cases in which the degrees of freedom were limited. This was done to keep all the targets within reach, e.g. in the $Y = 0$ plane when no horizontal rotation was possible.

3.3.2.1 Network architecture

The experimental setup has two different MLPs of $128 \times 256 \times 256$ neurons each for the respective parametrisations of the actor and the critic.

Hyperparameter	Value
Learning rate	0.0003
Batch size	64
Discount factor (γ)	0.95
GAE (λ)	0.95
Clip range (ϵ)	0.2
Value coefficient (c_1)	0.5
Entropy coefficient (c_2)	0.0
Max. gradient clipping	0.5
Initial log. standard deviation	0.0

Table 3.1: Hyperparameters for the PPO models used in the RL experiment.

Trial ID	Degrees of freedom	Reward eq.	Comments
62	All	3.16	Very good. Does not depend on the observations (all runs are the same).
73	`SHOULDER`	3.18	Good. There is a significant oscillation once targets are reached.
74	`SHOULDER`	3.19	Good. The oscillation is slightly minor, but more violent.
75	`SHOULDER`, `WRIST`, `ELBOW`	3.19	Moderately adequate. The arm identifies correctly where the targets are, but struggles to reach them in some cases.

Table 3.2: Description of the most significant trials in this experiment.

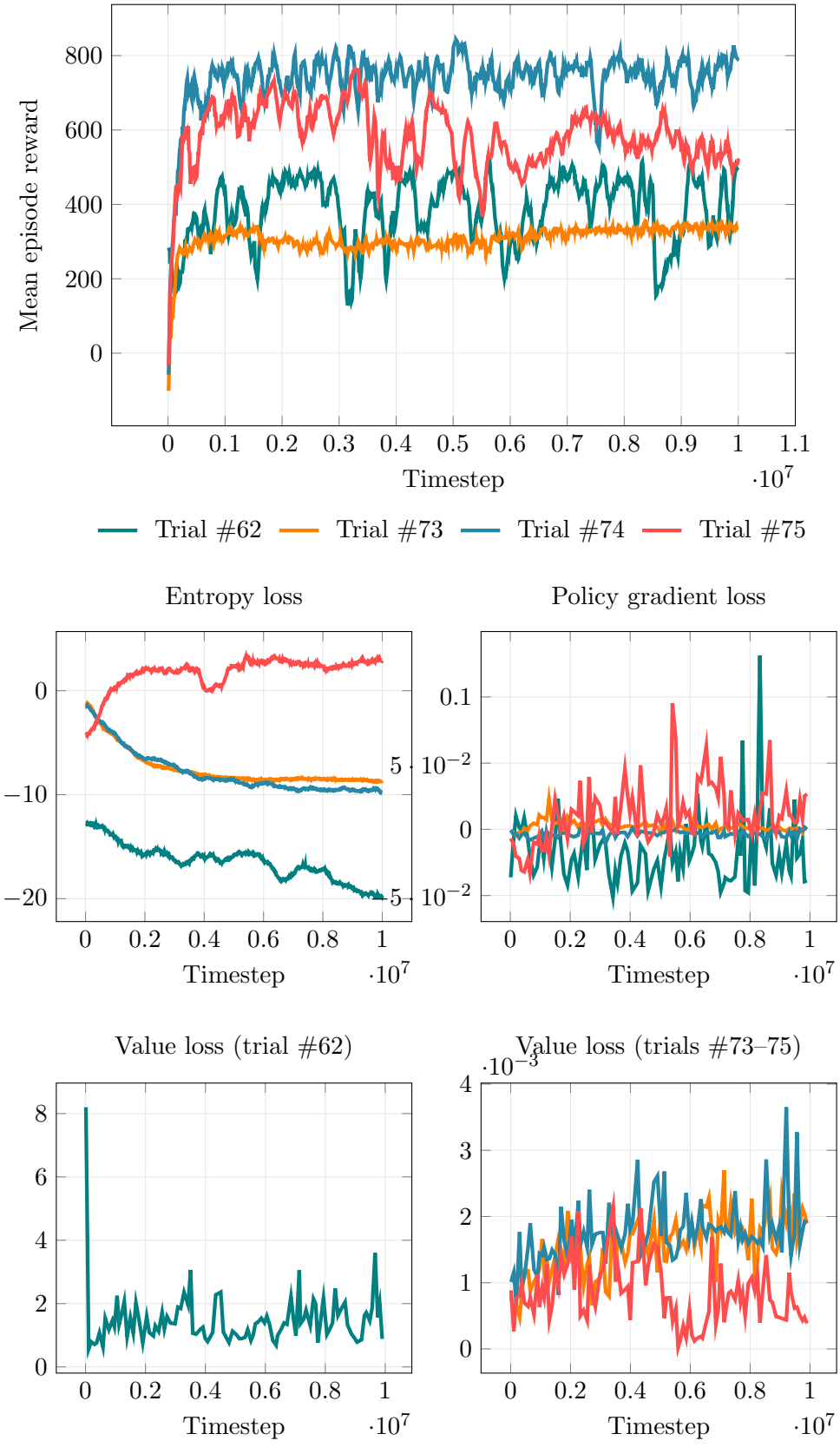
3.3.3 Training and results

As it has already been mentioned, PPO does not differentiate between training and inference: it simply keeps interacting with the environment and learns the dynamics as it goes.

Multiple trials have been run, combining different reward functions and degrees of freedom. Each trial has been executed for 10 million timesteps, allowing the agent to get enough experiences to extract insights from the environment. Some of the most significant trials are detailed in Table 3.2, and the main evolution metrics are plotted in Figure 3.12.

As it could be seen comparing trials with and without energy penalties, it was clear that the energy penalty favoured smoother movements (making a more conservative use of energy), but it sometimes hindered exploration and could therefore impact the correct learning of the environment's dynamics.

The results show that PPO is indeed very powerful, as it is relatively capable of controlling 3 degrees of freedom with a modest architecture. However, it is also a technique that requires a very large amount of computational resources: the number of simulations that have to be executed is so huge that it dwarfs the time spent in the training of the underlying neural networks. This is also a problem when developing



(a)

Figure 3.12: Evolution of the agent's metrics throughout over time.

the models, because the iteration frequency is much lower: it is less feasible to try different hyperparameters and experimental settings.

Chapter 4

Conclusion

4.1 Conclusion

This BEng Thesis has covered three of the main modern machine learning techniques based on artificial neural networks to design various motor controllers.

It has been shown that systems trained end-to-end, like supervised learning ones, are much faster to train and are very robust, but their performance drops drastically as soon as there are overlapping degrees of freedom. Black box optimisation has proven to be extremely resource-consuming, showcasing how valuable are techniques like automatic differentiation for the speedy and efficient training of neural networks. Finally, reinforcement learning has provided the best performance, but also at a very noticeable cost. Methods with a higher sample efficiency would be able to extract more information per sample of the environment, and therefore reduce the resources spent on training.

It has also been discussed how, out of the three methods, reinforcement learning is the most biologically plausible one and the only one that natively considers a time component, although in a highly restrictive manner since it models the environment as an MDP.

4.2 Future lines of work

Considering the insights obtained throughout this project, there are multiple potential lines of work that could be followed for further analysis:

- Studying if alternative clustering methodology could lead to the generation of a functional efficient dataset that would make the supervised learning system compatible with several degrees of freedom.
- Use temporal encoding for inputs and outputs in the supervised learning model, so that both the Cartesian and generalised coordinates can be represented as time series and can therefore be analysed with more biologically inspired neural networks [38].
- Performing the reinforcement learning process with a wider matrix of hyper-parameters, especially with different learning rates and/or learning rates that evolve throughout time (e.g. via scheduling).

- Trying to use reinforcement learning with an increasingly harder set of examples (curriculum learning) [39], or using a generative adversarial network to create examples that optimise learning [40, 33].

Bibliography

[1] Rhodius Apollonius. *The Argonautica*. Project Gutenberg, 1997.

[2] Michael L. Littman. Collusion rings threaten the integrity of computer science research. *Commun. ACM*, 64(6):43–44, May 2021.

[3] Ali Rahimi and Ben Recht. Reflections on random kitchen sinks. `http://www.argmin.net/2017/12/05/kitchen-sinks/`.

[4] D. Sculley, Jasper Snoek, Alex Wiltschko, and Ali Rahimi. Winner's curse? on pace, progress, and empirical rigor, 2018.

[5] A. Polsky, B. W. Mel, and J. Schiller. Computational subunits in thin dendrites of pyramidal cells. *Nature Neuroscience*, 7(6):621–627, Jun 2004.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[7] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Nov 2019.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[9] C Eliasmith. *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford University Press, 2013.

[10] Chris Eliasmith, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, 2012.

[11] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir. Inverse kinematics techniques in computer graphics: A survey. *Computer Graphics Forum*, 37(6):35–58, 2018.

[12] OpenRAVE. IKFast: The Robot Kinematics Compiler. `http://openrave.org/docs/0.8.2/openravepy/ikfast/`.

[13] Adrian-Vasile Duka. Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. *Procedia Technology*, 12:20–27, 2014. The 7th International Conference Interdisciplinarity in Engineering, INTER-ENG 2013, 10-11 October 2013, Petru Maior University of Tirgu Mures, Romania.

[14] Ahmed R. J. Almusawi, L. Canan Dülger, and Sadettin Kapucu. A new artificial neural network approach in solving inverse kinematics of robotic arm (denso vp6242). *Computational Intelligence and Neuroscience*, 2016:5720163, Aug 2016.

[15] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. `https://pybullet.org`, 2016–2021.

[16] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[17] E. Rohmer, S. P. N. Singh, and M. Freese. CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[18] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[19] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30, 1998.

[20] Poetry: Dependency Management for Python. `https://github.com/python-poetry/poetry`, 2018–2021.

[21] The HDF Group. Hierarchical Data Format, version 5. `https://www.hdfgroup.org/HDF5/`, 1997-2021.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga,

Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computing Research Repository*, abs/1412.6980, 2015.

[24] Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(1):5595–5637, January 2017.

[25] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[26] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(27):949–980, 2014.

[27] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[28] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[29] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, December 1997.

[30] Paul Knysh and Yannis Korkolis. Blackbox: A procedure for parallel optimization of expensive black-box functions. *Computing Research Repository*, abs/1605.00998, 2016.

[31] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. `https://spinningup.openai.com/`, 2018.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[33] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. Reinforcement and imitation learning for diverse visuomotor skills. In

Proceedings of Robotics: Science and Systems, Pittsburgh, Pennsylvania, June 2018.

[34] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[35] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018.

[36] André Parent and Lili-Naz Hazrati. Functional anatomy of the basal ganglia. i. the cortico-basal ganglia-thalamo-cortical loop. *Brain Research Reviews*, 20(1):91–127, 1995.

[37] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[38] Stanisław Woźniak, Angeliki Pantazi, Thomas Bohnstingl, and Evangelos Eleftheriou. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nature Machine Intelligence*, 2(6):325–336, Jun 2020.

[39] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[40] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 2817–2826. JMLR.org, 2017.

[41] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *CoRR*, abs/1906.02243, 2019.

Appendix A

Ethical, economical, social and environmental aspects

A.1 Ethical & social impact

The recently acquired capabilities of AI to perform critical tasks like driving has put the focus on the regulation that this kind of technologies should have, and where should we draw the line that defines what is not okay for AI to address.

One of the main ethical concerns is the lack of traceability for the outputs given by most AI systems, which raises questions about whether a certain recommendation is properly founded or if it might be the result of a technical issue. Moreover, the dependency of these systems on experience and pre-existing data can lead to the transmission of biases that were originally in the source. For instance, certain text generators trained on web data have displayed racist or misogynistic behaviours, which are clearly unacceptable.

Besides that, focusing specifically on research, that has also been reflected on the diminishing of the quality of certain research to unprecedented levels. This has also led to some researchers acting in unethical ways, in what could even be considered as committing academic fraud [2].

Even though a big part of these issues are not caused by the technology itself, just like with other technological breakthroughs in the past (e.g. the Internet) we need to make sure as a society to have sufficient regulation and make a responsible use of all that AI has to offer.

A.2 Economical impact

Since this BEng Thesis is heavily research-oriented, the potential economical impact it might have in the short to mid term is quite limited. However, it contributes to the major effort of obtaining better AI systems and move closer to AGI, which could be a significant change in the lifestyle of humanity as a whole.

Having a potentially unlimited source of automated intelligent workers could radically alter the way we conceive economics and our lives overall, but that point is still probably too far away to even speculate about it.

A.3 Environmental impact

Another problem associated with deep learning is the big amount of computing resources that it requires to train and iterate in the design of models. Some members of the AI community have already expressed their concerns [41] regarding the energy consumption associated to all this processing, and the fact that we should eventually move to technologies that are less demanding or to computing resources that are sustainable.

Appendix B

Budget

The development of this BEng Thesis has involved two main components resource-wise:

- **Labour:** qualified personnel with a background in engineering, and more specifically, in machine learning. Two people have been involved:
  - A supervisor with a PhD in Telecommunication Engineering who has been responsible of overseeing the progress of the project and providing guidance on the approach to follow along the way.
  - A BEng student who has been responsible for the design of the experiments, the development of the underlying code platform, the analysis of experimental results and writing this report.
- **Computational power:** a cloud host capable of handling the training of the models. The host kindly provided by Robolabo for this BEng Thesis had an Intel® Core™ i7-8700K CPU with two NVIDIA® GeForce® RTX 2080 Ti GPUs and 32 GB of RAM, that was shared with other 2 researchers and a sporadic additional one.

Using fair per-hour salaries for both people and the costs of a similar host using Google Compute Engine[1] (a2-highgpu-1g), a budget for the whole BEng Thesis has been created (see Table B.1).

[1] `https://cloud.google.com/compute`

Cost of labour			
	Hours	**Cost/hour (€)**	**Total (€)**
Thesis supervisor	40	60	2,400
Social security			792
Engineering student	500	30	15,000
Social security			4950
Cost of materials			
	Hours	**Cost/hour (€)**	**Total (€)**
Cloud computing	600	3.10	1,860
VAT (21%)			391
Total costs			
TOTAL			25,393

Table B.1: Summary of the estimated budget.