FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Realistic Digital Twin of a Dexterous Robotic Manipulator

Tiago Pereira Correia

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Gil Manuel Magalhães de Andrade Gonçalves

July 31, 2023

Reconhecimentos

The following people collaborated with the supervising team in carrying out this work.: Prof. Vítor Hugo Pinto and Francisco Ribeiro.

This work was carried out with the support of SYSTEC and in the scope of Project *Digitalização da Arte Humana (Cibertoque)*, with reference POCI-01-0247-FEDER-072627, co-funded by FEDER, through COMPETE 2020.

Tiago Correia

Resumo

As tecnologias associadas aos processos automatizados têm vindo a crescer continuamente nos últimos anos, e a maior parte deste crescimento deve-se aos benefícios que traz para a indústria. Mais especificamente, na indústria automóvel, é quase inegável a vantagem de automatizar alguns processos que normalmente envolvem trabalho humano, como o polimento, tarefas de inserção de cabos e deteção de defeitos de pintura. Através de sistemas ciber-físicos colaborativos é possível aumentar a produção, mantendo a flexibilidade humana.

Em particular, a utilização extensiva destes sistemas extremamente complexos, como o caso de um manipulador robótico capaz de imitar a destreza humana, pode pôr em perigo o bem-estar destes componentes, provocando elevadas perdas. Com isto, uma nova linha de desenvolvimento tem vindo a ganhar muita atenção nos últimos anos, chamado *Digital twin* ou gémeo digital. Esta tecnologia permite ao utilizador monitorizar, analisar e controlar digitalmente, em tempo real, o estado do sistema, permitindo uma manutenção pró-ativa e a capacidade de prever a necessidade de uma manutenção. Além disso, pode ser utilizada para melhorar a comunicação e a colaboração numa unidade fabril. Todavia a fidelidade destas funcionalidades torna-se extremamente dependente do realismo apresentado pela simulação.

Considerando o exposto, o projeto em questão foca-se na elaboração e adaptação de um gémeo digital realista de um manipulador robótico, de modo a alcançar uma elevada semelhança para com o sistema físico. Este processo implica uma calibração do ambiente simulado. De outra forma, esta calibração pode ser interpretada como um problema de otimização na qual o principal objetivo é encontrar os parâmetros físicos presentes no simulador, de forma a diminuir a discrepância entre ambientes. Foi também desenvolvida um módulo capaz de obter e processar toda a data necessária do ambiente real e simulado para a otimização. Diversos algoritmos de otimização e diferentes abordagens para cada um desses algoritmos foram desenvolvidos visando encontrar o melhor e mais eficiente.

Keywords: Gêmeo Digital, Otimização, Simulação Realista, Modelação

Abstract

Technologies associated with automated processes have been growing abruptly over the past few years, and most of this growth is due to the benefits it brings to the industry. More specifically, in the automotive industries, to automate some processes that usually involve human labour, like polishing, insertion tasks and detection of paint defects, the necessity of cyber-physicals collaborative systems in order to increase production output keeping human flexibility is almost undeniable.

In particular, the extensive use of these extremely complex systems, as in the case of a dexterous robotic manipulator, may endanger the welfare of those components, prompting immense losses. With this, a new term has gained a lot of attention in the past few years, called Digital Twin. This technology enables the user to monitor, track and control digitally, in real-time, the state of the system, allowing for proactive maintenance and the ability to predict when maintenance is required, as well as the capability to implement training algorithms in complex environments. Nevertheless, the fidelity of these features is highly dependent on the realism presented by the simulation counterpart.

In brief, this work focused on the development and adaptation of a Digital Twin with the purpose of identifying the simulated counterpart that best replicates real-world behaviour. This process implies the calibration of the simulation environment, which in turn may be interpreted as an optimisation problem where the objective is to reduce the discrepancy between the real and simulated environment. A pipeline responsible for data acquisition and processing for both simulated and real systems was developed. Subsequently, several algorithms for optimisation and different model approaches were considered and compared in this study with the aim of finding the best one.

Keywords: Digital Twin, Optimisation, Realistic Simulation, Modeling

Agradecimentos

Em primeiro lugar quero agradecer ao meu orientador Professor Gil Gonçalves por possibilitar-me esta oportunidade. Inclusive, gostaria de demonstrar a minha sincera gratidão para com o Professor Vítor Pinto. Desde a sua amizade nos momentos de descontração até à sua disponibilidade e apoio nos períodos mais difíceis, contribuindo para o meu crescimento no decorrer deste processo. Da mesma forma, quero deixar uma nota de agradecimento ao Professor Francisco Ribeiro pela sua disponibilidade e inestimável generosidade demonstradas ao longo deste projeto, mesmo não possuindo qualquer responsabilidade para tal.

Expresso o meu apreço aos meus estimados amigos que me acompanharam, não só ao longo desta jornada, bem como em inúmeras outras. O vosso apoio e animação foi absolutamente precioso e estou verdadeiramente grato pelos diversos momentos que partilhámos neste trajeto. Sem as vossas terríveis influências toda esta caminhada teria sido extremamente penosa.

Gostaria de estender os meus sinceros agradecimentos a uma pessoa especial cuja presença teve um impacto significativo na minha vida e neste projeto. O seu encorajamento, paciência e crença inabalável em mim foram uma fonte de inspiração ao longo deste processo.

Por último, mas mais importante, demonstro a gratidão para com a minha família, em especial à minha mãe, ao meu pai e à minha irmã. O vosso sacrifício e apoio incondicional foi o principal motivo que possibilitou todo este percurso. Agradeço a perseverança e compreensão que demonstraram nos momentos de maior sufoco.

Tiago Correia

"It's not the load that breaks you down,
it's the way you carry it."

Lou Holtz

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AI	Artificial intelligence
API	Application Programming Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
DoF	Degrees of Freedom
DLO	Deformable Linear Object
DT	Digital Twin
FEM	Finite Element Method
GP	Gaussian Process
HRC	Human-Robot Collaboration
I4.0	Industry 4.0
ML	Machine Learning
MSE	Mean Squared Error
MuJoCo	Multi-Joint dynamics with Contact
PPO	Proximal Policy Optimization
PSO	Particle Swarm Algorithm
RL	Reinforcement learning
ROS	Robotic Operating System
TP	Teleoperation teaching
UR5	Universal Robots 5
URDF	Universal Robot Description Format
XML	Extensible Markup Language

Chapter 1

Introduction

The fourth industrial revolution, also known as Industry 4.0 (I4.0), is adjusting the standards of the current industrial production. Technologies associated with automated processes have been growing abruptly over the past few years, and most of this growth is due to the benefits it brings to the industry. Reducing human labour and operations costs, and increasing the production output are a few of the advantages that contribute to the expansion of these technologies. Businesses, for these reasons, are trying to adapt their traditional manufacturing system to a more intelligent one with better efficiency. However, this expansion makes the manufacturing process increasingly complex, resulting in the use of even more complex hardware and software.

1.1 Context

A particular technology utilised, by the industry, to automate the manufacturing process is collaborative robots. A collaborative robot is generally human-like in terms of movement and poses, as well as the usefulness of the hand and arm. More specifically, robotic arms have been improving manufacturing efficiency as well as replacing workers in harsh environments, monotonous tasks or heavy procedures. Additionally, the integration of a robotic hand increases the use case scenarios of the previously referred robotic arm, highlighting pick and place and grasping use cases. Consequently, it can be established that these two combined technologies are an important part of the modern manufacturing system, and, in correlation to the expansion of Industry 4.0, we may also verify that this system is expanding throughout the industry.

Digital Twin (DT) is another term that's been flourishing over the past decade. Similarly, in the context of I4.0, this technology, as well as simulation technologies, have been growing as a result of the essential role they play in the development of the previously referred automated processes. Virtualisation is one of the key design principles in I4.0, and it involves the use of software in order to create a virtual copy of the physical system.

Virtual models, as is the case of a DT, may be applied to validate and safely test manufacturing system strategies before taking them into practice [1]. Additionally, it monitors, tracks and controls digitally, in real-time, the state of the system. It also provides the possibility to update the digital model taking into account the changes that occurred in the physical system. In the same way, the danger of harming high-value components in the development of complex manufacturing systems, as previously referred, may be decreased by using a digital twin in a realistic simulator. Ultimately, another application of a digital twin is to optimise the performance of the corresponding physical system. For all of these reasons, an increased effort from the industry has been taking place to make it a viable solution.

1.2 Motivation

This work is integrated within a project that primarily seeks to facilitate the transformation of manual processes into automated ones. More specifically, in automotive industries, there are still several processes that are executed manually, highlighting the processes related to quality inspection and the processes related to wiring fitting.

In order to automate these procedures, a robotic system can be used to allow the replication of human labour with a high degree of detail and dependence on tactile perception. The system composition of this robotic system is composed of a robotic arm manipulator coupled with a dexterous robotic hand with tactile sensors on each finger. Ultimately, the main goal of this system is to perform procedures that usually involve human hand and arm movements, like polishing, insertion tasks and detection of paint defects using the tactile sensors.

Additionally, there is a need to make this system efficient enough in order to turn it into a viable solution and justify the large investments in those components. Robotic arms have been in the industry for a long time and are also widely studied in research and industrial environment, and for that reason, there is no obstacle to the use of this system. However, the use of robotic hands with a high degree of freedom has complex control problems and, for that reason, is not used in the industry as much as robotic arms. This complexity also arises as a result of the synergies present in the human hand. Moreover, the process of monitoring these complex technologies is essential, allowing the user to track, in real-time, the current state of the system.

1.3 Objectives

Given all the previous considerations, it is easy to understand that there is a need to have software capable of monitoring and tracking these systems. Especially with highly intricate components like a dexterous robotic hand, virtual representations of the real system play a pivotal role. This can be achieved using realistic digital twins.

However, despite having access to the simulated representation of the real component provides useful insights, it is equally crucial to ensure that the simulator itself is properly calibrated. This process consists of the attainment of parameters, settings and the overall behaviour of the simulation that allows a similar resemblance between this virtual system and the real one.

Achieving this previous objective allows us to possess a more accurate representation within the simulation environment, thereby facilitating the implementation of various control techniques without the risk of damaging the integrity of real components. This, in turn, aligns with the goal of the overall project on which this work is inserted. However, there are numerous more utilities for this technology, as mentioned before.

1.4 Document Structure

Besides the introduction, this dissertation contains 5 more chapters. In Chapter 2, there is an introduction regarding the several definitions of a digital twin, as well as its applications and the different levels of integration of this technology. Additionally, there is an overview concerning the term “machine learning”. The physical component used in this work is also referred to. Regarding Chapter 3, different technologies and methodologies used for the calibration and model adaptation for the digital twin are addressed. Chapter 4 presents all the steps that were taken during the implementation. The results and their analysis were made in Chapter 5. Lastly, Chapter 6 provides the conclusions derived from the outcomes of this work, along with the identification of several potential pathways for improvement and future investigation.

Chapter 2

Background

This chapter has the purpose of providing an understanding of the concepts and fundamental principles discussed further below. By the end of this chapter, you should have a good grasp of the key concepts and be well-prepared to dive deeper into the topic at hand.

2.1 Digital Twin

The concept of DT has undergone changes over the years, but with the advent of Industry 4.0, it has become more encompassing. To trace the evolution of DT in recent years, a literature search was conducted using the Scopus API to search for the term “Digital Twin” [1]. This search shows the abrupt growth of this term in scientific articles, especially in recent years.

Several definitions of this term have appeared, the first made by Michael Grieves in 2002 at the University of Michigan as a “digital equivalent to a physical product”. After that, numerous interpretations have appeared, one of them by NASA [2], introduced as “an integrated multiphysics, multiscale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history... to mirror the life of its corresponding flying twin.”. Extensive research about the different definitions of a DT was done by Y. Liu *et al.* [3].

A Digital twin is, in many aspects, similar to a simulator, however, this last runs separately from the real-time process. In other words, a DT may be interpreted as a simulation whose states are updated in real-time via sensor data [4].

In a DT, data flows in two directions: from the physical object or system to the digital twin and from the digital twin to the physical object or system. The data flowing from the physical system to the digital twin is often referred to as “sensor data”, as it is collected by sensors that are placed on or near the physical object or system. This sensor data is used to update the digital twin with the current state of the physical system.

2.1.1 Applications

DTs have a wide range of applications, including in manufacturing, smart cities, retail, and healthcare. In the specific case of manufacturing, it can be used to optimise the design and operation of factories and production lines, improving efficiency and reducing downtime [5]. In recent years, DT has also been widely studied and researched in the construction industry in order to modernise and embrace the advancement of new technologies [6]. Another countless number of applications is presented in [7], [8].

2.1.2 Digital Twin for Human-Robot Collaboration

Human effort is being replaced through automation [9] as a consequence of the cost of human work. However, there is a need for flexible and adaptive labour, and for this reason, creating a balance between the flexibility of manual and efficient automatic manufacturing processes have its advantage. This can be achieved by collaborative robots [10].

Integrating a DT into a Human-Robot Collaboration (HRC) system allows the physical system to have its data from the DTs, which in turn makes it possible to monitor and analyse, in real-time, the operation condition of the physical system. The needs and the usefulness of DTs in the design, development, and operating HRC system as well as a demonstrator of an HRC digital twin on an industrial case [11].

On the other hand, as the level of collaboration between human labour and robots increases, also the system tends to be more complex, which can be verified since the First Industrial Revolution. With this, the use of a digital twin enables a better understanding of the current state of the robot from the human counterpart.

An interesting example was given by X. Wang *et al.* [12] referring to the importance of a digital twin for Collaborative Human-Robot Construction Work, along with the implementation concerning this system. Accessing pertinent information, preventing human workers from potential hazards in the construction workplace, reduction of human effort in demanding tasks, and the availability of supervision are some advantages of the HRC system, along with a digital twin using a Virtual Reality(AR) given by the authors.

2.1.3 Integration Level

There are several methods to implement a digital representation of an existing physical object. This implementation differs in the data integration between these two components, some of them manual and some automated in real-time [5].

- **Digital model**

  A digital model is a digital representation of an existing physical object without any type of automated data exchange between the two. The digital data of the physical object can still be taken into account in the development of this model, however, the exchange of data is done manually, represented in figure 2.1. In other words, the state of the virtual model has no direct impact on the physical model and the other way around.

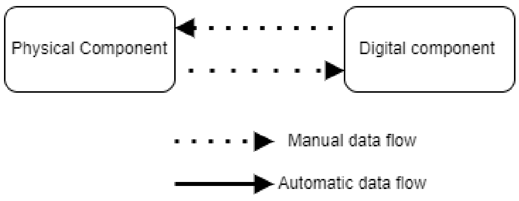


Figure 2.1: Data Stream in a Digital Model

- **Digital Shadow**

  A digital shadow is, in many ways, very similar to a digital model. However, the data between the physical object and the digital object can be automated, as may be seen in figure 2.2. This means that a change in the state of the physical object has a direct impact on the state of the digital object and not the other way around.

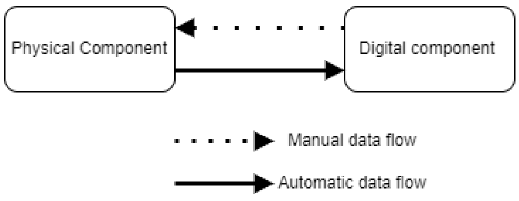


Figure 2.2: Data Stream in a Digital Shadow

- **Digital Twin**

  A DT, despite being similar to a digital shadow, automates the data flow also between the digital and the physical object, which is demonstrated on 2.3. In other words, a change in the state of the digital object has a direct impact on the state of the physical object and the other way around.

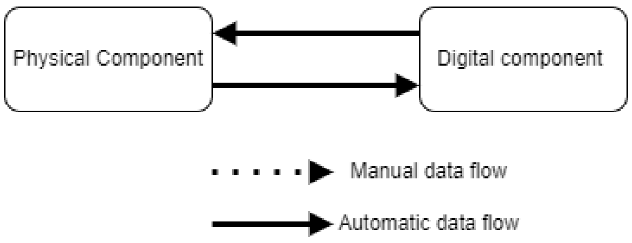


Figure 2.3: Data Stream in a Digital Twin

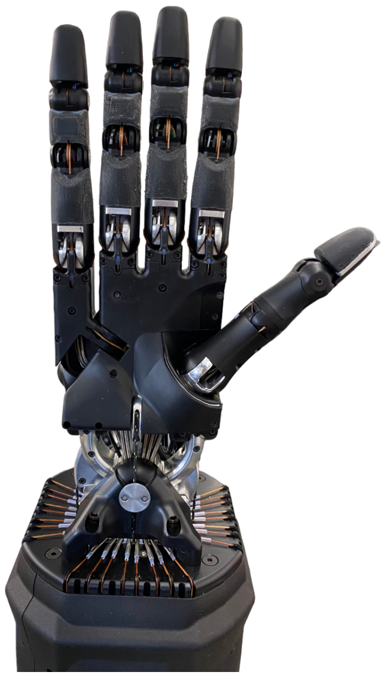


Figure 2.4: Real illustration of Shadow Dexterous Hand.

2.2 Physical Component

There are two components in the overall perspective of the project that this work is inserted on. The Universal Robots 5 (UR5) is a robotic manipulator designed and manufactured by Universal Robots [13]. However, the fact that this technology is already largely used in the industry and it possesses a considerably low complexity makes this component cause any obstacle in its use.

Copulated to the previous component, there is a dexterous robotic hand called Shadow Dexterous Hand [14], developed by Shadow Robot. This hand is the main focus of this work due to its complexity as well as its difficult realistic representation in a simulated environment.

2.2.1 Shadow Dexterous Hand

The robotic hand used in this project is the Shadow Dexterous Hand, shown in Figure 2.4 with 20 Degrees of Freedom (DoF) and a further 4 under-actuated movements for a total of 24 joints. It is an anthropomorphic robotic hand and comprises five fingers, one of which is a thumb.

The dexterous hand comprises a total of 26 Hall effect sensors with a resolution of 0.2 degrees, which are specifically designed to detect and measure magnetic field rotations along the axis of each joint. This sensor is responsible for providing all the information from the sensors to the communication bus. To this effect, EtherCAT (Ethernet for Control Automation Technology) bus is used for communication between the robotic system and the main machine responsible for the control loop of the real hand. In this particular case is used a NUC (Next Unit of Computing) for this objective.

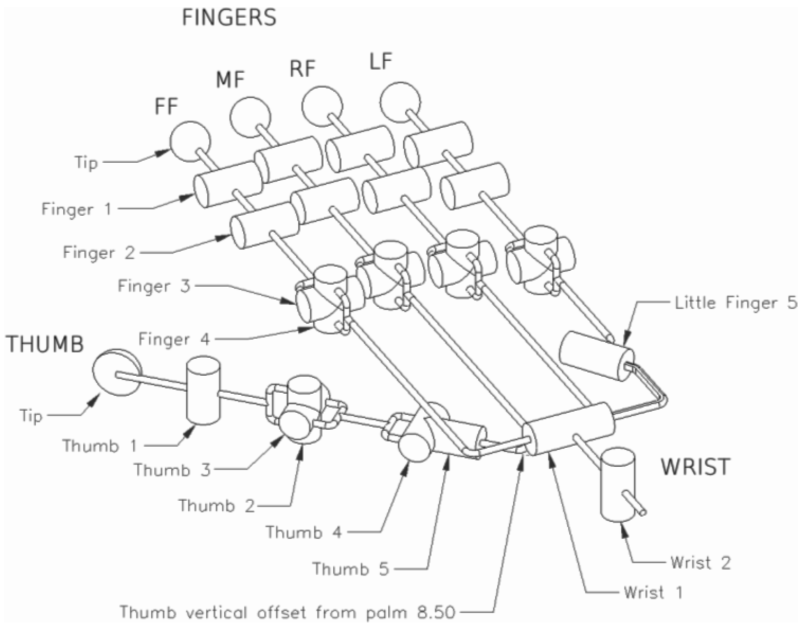


Figure 2.5: Shadow Dexterous Hand Kinematic Diagram. [14]

The four fingers are named according to the UK convention: First, Middle, Ring and Little. It is from this nomenclature that derives the designation of each finger on Figure 2.5. Both these fingers have 3 DoF and the exact same size with only a slight positional variation.

Taking a closer look at one of those individual fingers, we may see four finger joints: The distal, middle, proximal and the abduction joint. This last is co-planar with the proximal joint. These joints are numbered as Figure 2.6 shows. Starting with the distal, which corresponds to the number 1, till the end of the finger. The nomenclature used to identify each of the joints in each finger follow this same rule, only adding the prefix "FJ". This prefix denotes the element "F", standing for "finger", and the class "J", corresponding to the class "joint". For instance, the "FJ1" joint represents the distal joint of one of these 4 fingers, as shown in that same Figure. It is also possible to observe the positive rotation identified by the arrow's direction.

It is also important to refer that both the "FJ1" and "FJ2" joints are coupled together such that: $FJ1\_angle <= FJ2\_angle$. This means that any flexion of joint "FJ1" beyond the angle of joint "FJ2" forces joint "FJ2" to flex so that the constraint is maintained.

Beyond these four fingers, that is also one thumb. This one has 5 DoF, and all of the joints actuators are independent of each other. The nomenclature follows the rules of previous fingers, but with some adjustments. In this case, the joints are numbered as Figure 2.7 shows and is used the prefix "THJ". Also, in this case, it is possible to observe the positive rotation by identifying the direction of the arrows.

On a separate note, this particular hand has received some attention from the scientific community in the last few years. In the development phase, we can take advantage of its integration with Robotic Operating System (ROS). It is also compatible with BioTac Sensors [16], produced

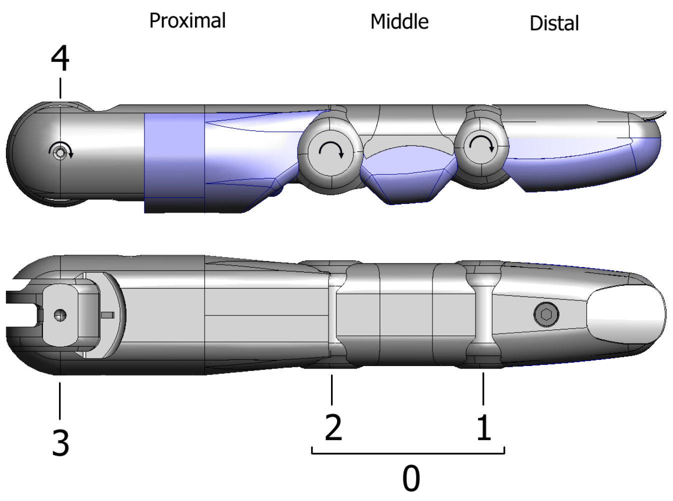


Figure 2.6: Finger representation and nomenclature [15].

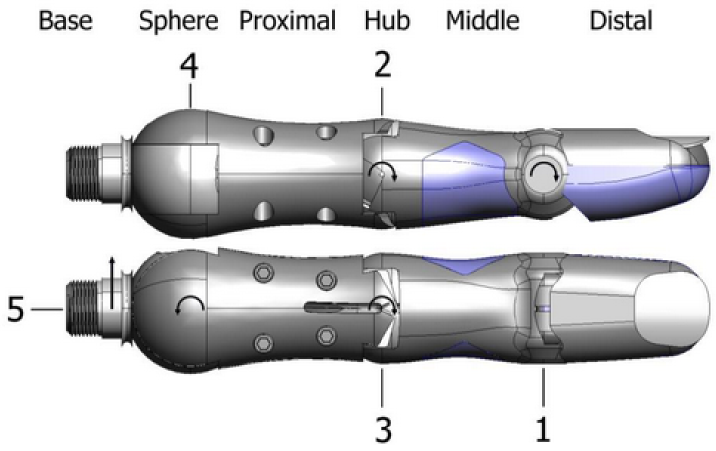


Figure 2.7: Thumb representation and nomenclature [15].

by SynTouch. These sensors are capable of detecting, similar to human fingertips, a diverse number of sensory information, such as temperature, forces, etc. It is also compatible with the most recent metaAI fingertips sensor called DIGIT [17], which provides a high-resolution human-like tactile reading.

2.3 Machine learning

Currently, in the manufacturing industry, robotic systems are typically programmed to execute a specific task previously settled. This limits manufacturing automation in different industries, and for this reason, the development of Artificial Intelligence (AI) in system control has grown. With this, a new term, which is itself a branch of AI, has flourished in the past few years, called "Machine Learning" (ML) [18]. ML is a form of data analysis that involves the use of automated algorithms to create analytical models. In other words, this concept focuses on the idea that systems can learn from data, identify certain patterns and "learn" from it.

There are several methods or approaches to teach or program the robot in certain scenarios, as can be seen in [19]:

- **Direct Programming:**

  Despite not being effectively a "teaching" method, it is a low-level approach where the environment is precisely setted-up, and there can be observed a hard-coded implementation of robot movement. It has a low computational complexity. However, it may be extremely time-consuming and inflexible in unpredictable environments;

- **Imitation learning:**

  This approach, unlike the previous one, is a type of "teaching" method and a type of learning based on demonstration. It may be classified as:

  → Kinaesthetic teaching: It focuses on recording and analysing the process of manually moving the robot [20].

  → Teleoperation teaching (TP): This method is based on the concept of controlling the robot using another device. Such as, in the case of a hand manipulator robot, gloves with a high-resolution human-like tactile reading [21], or even by processing, in real-time, an image of a human hand pose obtained by a camera demonstrated in [22].

  → Observational learning: It has a very similar approach as TP, however, this one uses the motion capture of its own system.

- **Reinforcement learning (RL):**

  RL consists in the idea of learning from trial and error with a reward/punishment system [19]. Despite being more complex than other methods and computationally heavy, it allows the robot to learn from a task then not even humans are able to demonstrate.

This method will be more detailed in chapter 3.6.

Chapter 3

Literature Review

3.1 Simulation

After understanding the importance of a DT, it is necessary to analyse the most adequate software in order to find the one that better resembles the real world. This allows the achievement of the most accurate virtual representation of the physical robot. A physics engine capable of enabling data collection movement from the physical world into a realistic simulation is necessary to accomplish the desired outcome. In order to recreate the insertion task of non-rigid cables, there's also a need for this engine to be capable of supporting the simulation of deformable objects, which is further explored in 3.3.1.

It is known that while each physics engine has its advantages and disadvantages, no engine is superior in every parameter to the others, with each engine performing best under specific conditions and in particular parameters. Despite the growing complexity of modern robots, numerous simulators are capable of simulating these systems with sufficient performance to make them viable solutions.

Erez *et al.* [23] compared the performance of several physics engines, focusing on the numerical challenges typical for robotics as opposed to multi-body dynamics. MuJoCo (Multi-Joint dynamics with Contact) outperforms other alternative simulators, in the case of Bullet [24], Open Dynamics Engine (ODE) [25], Havok [26] and PhysX [27], in terms of critical factors such as speed and accuracy.

Francisco Ribeiro *et al.* [28] also conducted a performance comparison between MuJoCo and Gazebo, two widely used simulation software. Gazebo offers four different physics engines, namely Dynamic Animation and Robotics Toolkit (DART) [29], Simbody [30] and also ODE and Bullet. It explored the capability of those simulators regarding the usability for realistic simulation of a dexterous robotic hand and manipulator system for highly detailed tasks. In this study, it was concluded that MuJoCo, when compared to Gazebo, presents a more promising result. Another notable contribution of his work was the development of a method to integrate the ROS framework with MuJoCo seamlessly. This integration effectively addresses one of the primary drawbacks of the MuJoCo simulator.

3.2 MuJoCo

The proposed solution uses MuJoCo [31] physics engine as a simulation base. The primary objective of this software is to enhance performance by offering advanced features for modelling and simulating intricate dynamics. Notably, it has the capability to simulate objects with deformable properties, such as ropes and cables or even high complexity particle systems [32], enabling a more comprehensive and accurate representation of complex systems. This feature stands as a notable advantage when compared to other physics engines.

The library provides a C Application Programming Interface (API) for interaction with the virtual environment, and the acquisition of data from sensors in the simulation depends on this same API. The models of this engine, in the particular case of this work, the model of a multi-fingered robotic hand, are represented in an Extensible Markup Language (XML) type format file.

However, there are some downsides to using MuJoCo: the incapability of native ROS support, as said before, and the lack of inverse kinetics and path planning.

3.2.1 MuJoCo model

The MuJoCo modeling XML File (MJCF) is an XML-based file format utilised in MuJoCo simulators to describe the dynamics, kinetics, and properties of rigid bodies. It offers a wider range of features in comparison to the widely used Universal Robot Description Format (URDF), including the incorporation of set parameters that directly influence the physics of the simulation. This corresponds to the intended functionality. It can be considered a hybrid between both a modelling format and a programming language. This unique combination allows for a flexible and expressive representation of the model, bridging the gap between the descriptive nature of a modelling format and the computational power and versatility of a programming language [33].

However, it is important to notice that there are several entities called "model" in Mujoco. After defining the model in the MJCF file, the software has the capability to generate multiple instances of the same model in various media formats, such as files or memory, as well as different levels of description, including high-level or low-level representations [34]. All the combinations are shown in table 3.1.

Table 3.1: MuJoCo model combinations

	High level	**Low Level**
File	MJCF file (XML)	MJB (binary)
Memory	mjCModel (C++ class)	mjModel (C struct)

This flexibility allows for versatile and adaptable utilisation of the model in different contexts and applications.

After a change has been made in the MJCF file, for the change to take effect in the simulator, there is a need to load the model with *mj_loadXML* function. This specific function parses the MJCF file, compiles it, and returns a low-level model if there was not an error occurring during the process.

3.2.2 Physical Parameters

After comprehending the significance of simulation in CPS systems, it becomes essential to grasp the influence of physical realism on the simulation [35].

In order to obtain the simulation most similar to reality, it is necessary to establish the best physical parameters for the simulator. These parameters determine how objects interact with each other and their environment and correspond to constants, such as *inertia*, *friction loss*, *stiffness*, *damping*, and *range* of each component, among others. In the case of MuJoCo, these parameters are defined in an XML file with MJCF format, which was explained in 3.2.1.

To achieve a simulated robotic hand movement that closely resembles a real hand, it is imperative to calibrate the simulation environment. This objective can be accomplished by determining the optimal parameter values that enable a more accurate representation of reality. It is crucial to notice that all the real movements from the real hand and arm suffer from physical randomisation occurring in real-time. In addition to the aforementioned considerations, it is important to acknowledge that the presence of sensor deviations and measurement noise inherent in all sensors further constrains the process. This calibration can be intended as an optimisation problem in which the objective is to decrease the error between real and simulated trajectories.

However, despite the extensive research on optimisation algorithms, the optimisation of these specific parameters does not have a pervasive state of the art. OpenAI developed a calibration method in this article [36]. In this method, the setup consisted of 16 PhaseSpace tracking cameras and 3 Basler RGB cameras. A 3D tracking system to accurately determine the positions of the fingertips was developed. This system served multiple purposes, including conducting calibration procedures and serving as a reference for RGB image-based tracking. Specifically, the PhaseSpace Impulse X2E tracking system was utilised, incorporating active LED markers that emit distinct ID codes through blinking and linear detector arrays within the cameras to detect and identify the marker positions.

Due to the high cost and complexity associated with the aforementioned setup, an alternative approach was employed and analysed in this study. It was thought the implementation of an optimisation method based on the readings from the actual sensors present in hand, which despite being less effective, is extremely cheap and easier to reply with a limited or null budget.

Associated with that, the extensive use of the real hand provokes some changes in the hardware, as in the case of tension exerted on each joint, as well as the greasing between components, causing changes in the dynamic of the movement in the robotic hand. Having this in mind, making several calibrations throughout the use of the hand becomes imminent to keep the simulation resemblance to the real hand, and recurring all the time to the setup previously referred to, becomes unfeasible.

3.3 Deformable Objects

In recent years, there has been a proliferation of robotics applications that involve manipulating deformable objects in various fields [37], such as the manufacturing industry. A deformation occurs when an external force is applied to an object, causing it to alter its shape properties. Once the external force is ceased, the deformation can be plastic, elastic, or last-plastic, depending on the object's response.

As mentioned before, the simulation of these objects was one of the considerations when choosing the simulation software. There are several approaches to manipulate deformable objects in the simulation, being those mainly concentrated on Deformable Linear Objects (DLOs), such as the example of ropes, cables, beams, etc. The capability to simulate DLOs, as seen in Figure 3.1, was one of the reasons for the usage of MuJoCo and is based on composite objects along with solver optimisation to speed up the simulation and not overload the system.

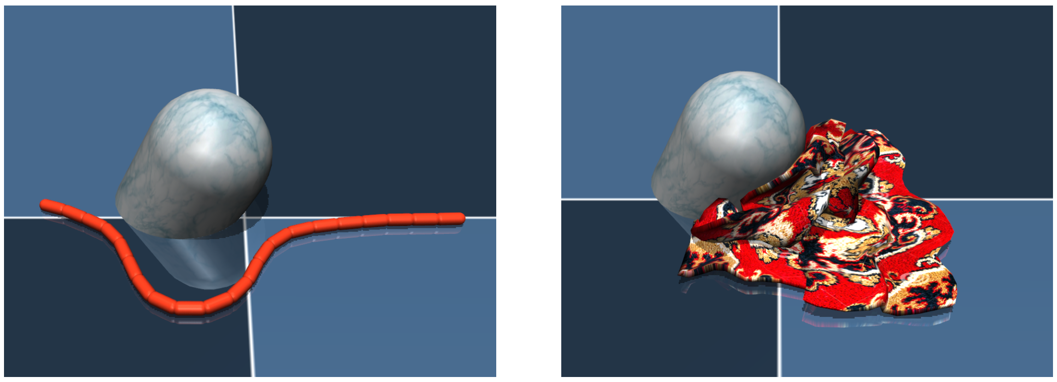

(a) Representation of a rope in MuJoCo (b) Representation of a cloth in MuJoCo

Figure 3.1: Examples of DLOs on MuJoCo.

In MuJoCo, composite objects were introduced with the objective of characterising a collection of existing elements [38]. The user, defining a composite element, can posteriorly specify the attributes, the elements as well as the number of elements in order to create a particle system, as in the case of a rope or cloth. It is also possible to define the dimension of the grid. For instance, the rope visualised in 3.1a corresponds to an assembly of elements in 1D. However, in the case of the cloth 3.1b, the elements are spread out in a 2D grid. This feature facilitated the use of DLOs in MuJoCo.

3.3.1 Manipulation of Deformable Objects

Manipulation tasks of DLOs include insertion of a cable through a series of holes, which is used in the case of this project, but also untangling a rope, manipulating a tube into the desired shape, and, most commonly, tying knots [37].

W. Wang *et al.* [39] approached the problem of inserting a rope and a string through a series of holes. To achieve the desired result, which was to drive the tip of the rope through the hole, numerous vector fields were set in the centre of each hole. An approximate Jacobian was used to estimate and control the string's motion. A vector field based on magnetic fields was also constructed, a digital twin system implementation distance for the insertion task.

Another method was presented by E. Yoshida *et al.* (2015) [40] for planning motions of a flexible object based on precise simulation using the Finite Element Method (FEM). This method uses an optimisation-based motion planning called Covariant Hamiltonian Optimisation and Motion Planning (CHOMP) [41] [42] that can integrate objective functions of minimum energy manipulation. Together with a collision detection system, a Model-based FEM may be applied to validate the planned trajectory. Comparing the force applied to an object for optimised and non-optimised trajectories, it was concluded that the latter was successful without excessive force.

3.4 Data Model

The communication between a digital twin and its physical counterpart can be divided into two categories: Data transmission from the physical twin to the digital twin, which includes sensor data and status data, and data transmission from a digital twin to the physical component, which includes remote control commands and software updates [43].

N. Kousi *et al.* [44] demonstrates an implementation of a digital twin system, providing an infrastructure for integrating all the hardware components involved in the assembly and synthesising all the data coming from the shop floor under a unified common environment. Furthermore, it demonstrates techniques to transform sensor data into usable information and, eventually, knowledge.

3.4.1 Data acquisition

Data acquisition represents one crucial part of the implementation of a DT. This data or information, as mentioned before, is bidirectional.

The data flowing from the DT to the physical object or system is frequently used to control or influence the physical system. On the other hand, the physical-to-virtual connection enables the process by which data collected from the physical system is used to update the states maintained in the virtual representation [45]. This sensor data is typically collected from various sensors installed in the physical system.

3.4.2 Data Processing

Bearing in mind the large number of multi-source real-time data from sensors during the robot operation, it is necessary to establish what and which type of data is exchanged between the physical e digital in order to make a more viable system.

A framework of a digital twin system based on target recognition and location grasp robot was developed by the authors of [46]. With this, a data processing interface was implemented to clean, iterate and process all data coming from the different sensors to provide data support for the simulation of the twin system.

3.5 Optimisation

In recent years, several advancements where made in the field of optimisation, mostly because, in itself, optimisation is an immensely researched topic. Several articles referencing a large number of algorithms that may be used in an enormous number of scenarios may be encountered in [47]–[49].

The field of optimisation has witnessed a growing number of algorithms with diverse methods and approaches. Researchers and practitioners continue to develop new techniques and adapt existing ones to address specific optimisation challenges. This expanding landscape of algorithms offers a rich toolbox for solving complex problems and continues to drive advancements in optimisation theory and practice.

Considering the preceding factors, it is crucial to first understand and examine the problem at hand, carefully analysing the objective and how that objective can be achieved. Only after this assessment proceeds the selection and the adjustments of the most fitting optimisation algorithm specifically designed to address the complexities inherent in the problem at hand. This approach proves advantageous as it allows for the research of methods that are compatible with the end problem, eliminating the need to invest time in algorithms that are not applicable to the specific problem at hand.

However, there is a necessity to have a general view of methods used in optimisation so that the very best algorithm may be chosen.

Considering the general form of a single-objective, non-linear optimisation problem:

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x}), \tag{3.1}$$

where Ω represents the feasible region, in other words, the upper and lower bounds which define the search space and $\mathbf{x}$ the variables that are modified to obtain the optimum. The $f(\mathbf{x})$ represents the objective function.

Depending on the access we have to the objective function, several algorithms can be used. On the one hand, if there is no access to the mathematical expression representing the objective function, it must be used a non-gradient dependent algorithm.

On the other hand, when we have the complete representation of the objective function, it is possible to choose whether utilise a gradient or a non-gradient algorithm. However, it is more commonly preferred to use gradient-dependent algorithms due to their ability to deliver better results for the same amount of time. These algorithms converge quickly and accurately on convex functions [47].

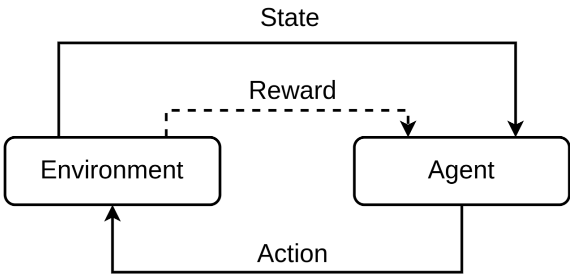


Figure 3.2: Simplified representation of the basic environment and agent iteration found in an RL algorithm.

In conclusion, an all-around understanding of the problem at hand, and careful selection of an appropriate optimisation strategy having in mind the wide range of available algorithms, provides the necessary mechanism to tackle optimisation problems effectively.

3.6 Reinforcement Learning

Reinforcement learning in robotics aims to enable robots to learn, adapt, improve, and perform tasks with dynamic constraints through discovery and spontaneous learning processes. Using sensors and other data sources, as mentioned in 3.4, a digital twin can accurately model the behaviour and performance of the physical system in real-time. Expressing it differently, RL can be applied to digital twins with the final goal of optimising the performance and efficiency of the real system.

As illustrated in Figure 3.2, in RL algorithms, a specific agent interacts with its environment and, via trial and error, explores the different actions while learning from the rewards received in each step. The main objective is to find the optimal strategy or policy that maximises the obtained reward. However, the way that this reward is obtained and the policies used differ from algorithm to algorithm.

Particularly using the dexterous shadow hand, OpenAI *et al.* [36] developed a reinforcement learning algorithm that is capable of performing a vision-based object reorientation. In this work, the policy was trained using a Proximal Policy Optimization (PPO). On the same note, a PPO requires training of two different networks: a policy network; responsible for mapping the observations into actions. And a value network; responsible for predicting future rewards from the present state. All the simulation was made using MuJoCo, and the hand model used was based on OpenAI Gym robotics environment [50]. Another similar algorithm for RL with the same setup as OpenAI also developed the previous one in a different work [51]. This time the objective was not only to reorient an object but also to resolve a Rubik's Cube. These two scientific works are references to the use of RL in the shadow dexterous robotic hand.

In this same work [50], several environments with different goals using the shadow dexterous hand and other models were presented. A particularly simple and interesting one called "Reach" is better explained in [52]. The main objective of this reinforcement learning environment is to make the thumb touch each of the other four fingers individually.

On a different note, a robotic arm associated with a digital twin utilising reinforcement learning was implemented in [53]. This article demonstrates the potential of reinforcement learning in developing a digital twin. However, the main contribution comes from the connectivity between a digital twin's physical and virtual components. A framework for training and heuristics to tackle virtual-physical connections was also developed and may, in turn, manifest the range of applications of a digital twin.

3.7 Teleoperation

Regarding a digital twin, robot teleoperation allows a human operator to remotely control and interact with a physical system through its digital replica. This technology may be advantageous in several situations and, for that reason, has been researched for education, underwater, nuclear and even in healthcare situations.

One convenience regarding teleoperation is the ability to operate robotics systems in hazardous or difficult-to-access environments or when the operator can't be near the physical system.

Another advantage of teleoperation in a digital twin is the ability to optimise and improve the performance of a physical system. Using the digital replica to simulate different scenarios and test different control strategies, operators can identify and implement the most effective solutions for improving efficiency and reducing downtime.

A framework to efficiently transfer human poses to collaborative robots using a human digital twin and a collaborative robot digital twin was proposed in [22]. A 2D human pose was taken as input, and as a result, a real collaborative robot was controlled using a PLC program in the implemented robot digital twin. With this, all the functionalities and uses of teleoperation were displayed, as well as an interesting imitation learning-based framework regarding implementing a teleoperated system.

Chapter 4

Implementation

4.1 Algorithm fundamentals

The development of realistic digital twins, as the name suggests, implies the really approximate resemblance between the simulation and the real systems. With this in mind, a calibration of the simulation model becomes imperative, as said before. This calibration can be interpreted as an optimisation problem in which the objective function corresponds to the error encountered between these two systems.

However, to implement this optimisation algorithm, it is preliminarily to design a strategy to obtain all the necessary data from both the real and simulated hand, process it and later on compare it.

In Figure 4.1, it is possible to visualise the main steps involved in order to obtain the best possible simulation model.

Firstly, there is a need to obtain all the joint data from the real hand while executing a specific trajectory. This is accomplished by the several sensors presented in the hand, as explained in 2.2.1. The data is then published into a topic called *joint_states* in which, during the execution of the trajectory, is saved and later on processed in order to be used for comparison. Besides that, a similar process must be done throughout the simulation in every iteration.

After obtaining all the necessary data, it is compared both trajectories. In this work, it was used the Mean Squared Error (MSE) for this effect. The MSE is a popular metric used for comparing data in various fields, such as statistics and ML. It measures the average squared difference between actual values and simulated ones, and the main expression may be seen in 4.1:

$$MSE = \frac{\sum\limits_{y=i}^{n} (Y - \hat{Y}_i)^2}{n} \tag{4.1}$$

Where in this particular case, the Y corresponds to the observed values from the real hand trajectory and $\hat{Y}_i$ to the observed values from iteration number i obtained from the simulation. The n corresponds to the total number of observations.

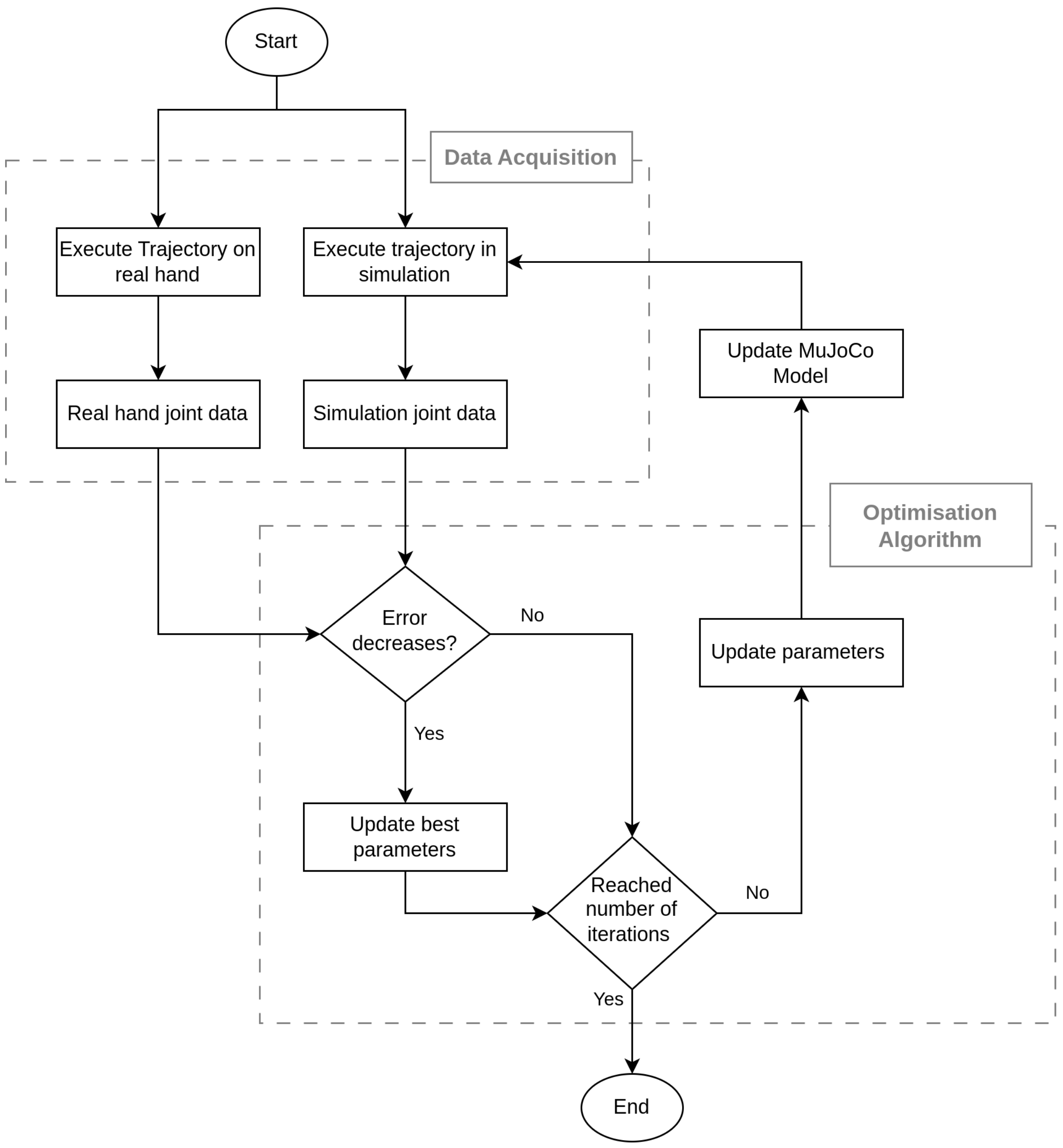


Figure 4.1: Simplified Flowchart of the Optimisation Algorithm

The MSE was chosen because of the extensive use of this method in occasions similar to the one encountered in this problem as well as because of the simplicity and straightforward metric.

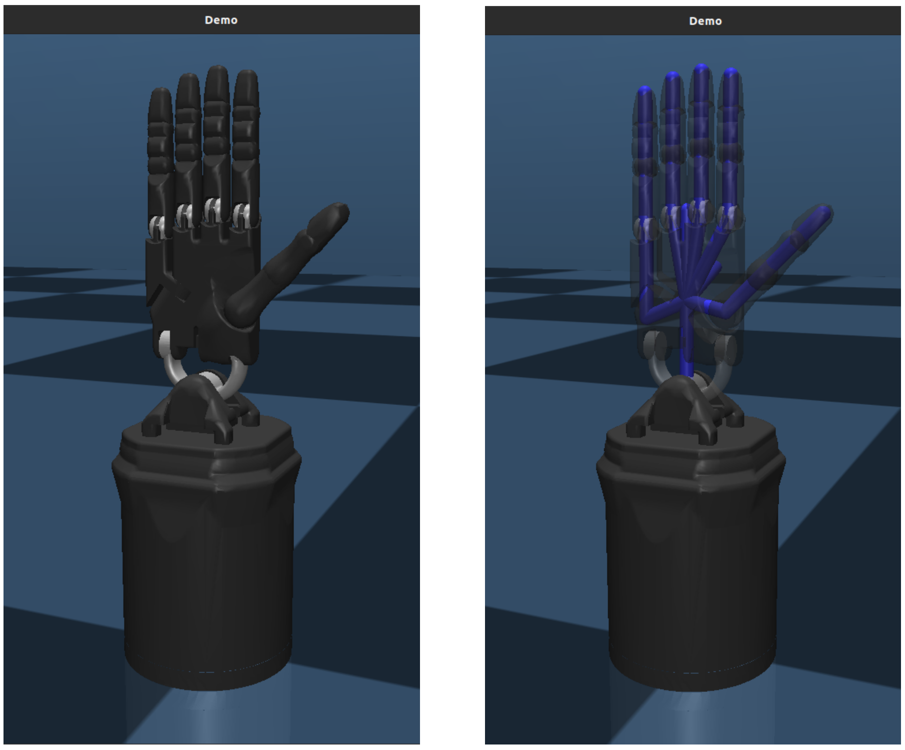
After the comparison is performed, the algorithm checks whether the error has decreased or increased. If the error has decreased, the model is saved, and the global optimum is updated. Follows the optimisation algorithm that depending on the method, sets different values for the parameters presented in the MuJoCo model and updates it. Once all the iterations are completed, the algorithm concludes and updates the best global values of the model.

This entire process is further elaborated and explained in subsequent sections.

4.2 MuJoCo Hand Model

As previously mentioned, the hand model in the MuJoCo simulation is formulated in an MJCF format and is also in this file that the parameters are located.

The model representation in MuJoCo may be seen in Figure 4.2. The Image 4.2a shows the simulation with the visual appearance of the real hand. Alongside, it is possible to visualise a transparent representation of the hand which in turn allows the visualisation of the tendons. This permits us to understand how the simulated hand was designed 4.2b.



(a) Simulation hand model (b) Simulation hand model with visibility of the tendons

Figure 4.2: Simulation hand model in MuJoCo.

In the next code fragment, we can find an example of a definition of a model in MuJoCo. It is important to note that it is composed of several components which define. Starting with the units used in simulation, radians in this specific case, and the coordinate system, defining the specified relation between an object and its parent. It also defines several visual properties in the simulation. In short, this file contains descriptions of the simulated objects, their connections, and other properties required for the simulation.

Yet, this code excerpt is not a simple example. This is the model that defines the visual and physical properties of the simulated dexterous hand:

```
 1  <mujoco>
 2      <compiler angle="radian" coordinate="local" />
 3      <visual>
 4          <map fogstart="3" fogend="5" force="0.1" znear="0.1" />
 5          <quality shadowsize="2048" offsamples="8" />
 6          <global offwidth="800" offheight="800" />
 7      </visual>
 8      <option cone="elliptic" impratio="300" timestep="0.002" />
 9      <size njmax="500" nconmax="200" />
10      <default class="main">
11          <mesh scale="0.001 0.001 0.001" />
12          <joint limited="true" damping="0.05" armature="0.005" margin="
                0.0" frictionloss="0.1" />
13          <default class="collision">
14              <geom group="3" />
15          </default>
16          <default class="visual">
17              <geom contype="0" conaffinity="0" group="0" />
18          </default>
19  ...
20  </mujoco>
```

Among these specifications, one particular aspect holds utmost importance for attaining a simulation that faithfully replicates real-world behaviour: the *joint* component, which can be found in line 12. Also, in this line, we can observe the default values of each joint in the hand. Calibrating the joint components involves adjusting the parameters of the joints to match the physical characteristics of the real system.

What is the meaning of each component that may be specified in *joint*:

- *stiffness*: The stiffness parameter determines the level of resistance exhibited by a joint when subjected to deformation or displacement. For instance, a higher stiffness value indicates a joint that is less susceptible to flexing or bending.

- *springref* (Spring reference): The spring reference parameter denotes the reference position of the spring linked to the joint. It plays a significant role in determining the behaviour of the joint when it deviates from its resting position.

- *frictionloss*: Friction loss, as the name suggests, represents the amount of energy dissipation due to friction during joint movements.
- *damping*: The damping has an impact on how quickly a joint returns to rest following motion. It basically symbolises the resistance to the motion brought on by internal or external forces.
- *range*: The range identifies the joint's range of motion. It details the joint's minimum and maximum angles.

Not all parameters need frequent updates or calibration. For instance, the *mass* parameter can be determined in advance by weighing each component. During regular usage, the mass does not undergo significant changes and, as a result, does not require extensive calibration.

There are other MJCF files that define different models, such as the configurations of the actuators and parameters that define the environment where the hand stands. Nevertheless, these parameters are not the focus of the study, considering the high difficulty of verifying and analysing their influence.

On another note, it is important to refer that in the simulation, by default, both the middle, first, ring and little finger are considered to have the same physical parameters. This indicates that all the joints within a specific category share the same default parameters. Initially, this approach is reasonable because, as explained earlier in 2.2.1, these fingers are very similar to each other, with slight positional variations.

In the next code excerpt, presented the MJCF file containing the default settings from the thumb as well as from the other 4 fingers:

```
<default class="finger">
    <joint axis="1 0 0" range="0 1.5708"/>
    <default class="FJ4">
        <joint range="-0.349066 0.349066"/>
    </default>
    <default class="FJ3">
        <joint/>
    </default>
    <default class="FJ2">
        <joint/>
    </default>
    <default class="FJ1">
        <!-- <joint stiffness="0.002" springref="-100.0"/> -->
        <joint/>
    </default>
</default>
<default class="thumb">
    <joint frictionloss="0.001" damping="0.05" />
    <default class="THJ5">
    </default>
    <default class="THJ4">
    </default>
    <default class="THJ3">
    </default>
    <default class="THJ2">
    </default>
    <default class="THJ1">
    </default>
</default>
```

As it is possible to see, most of the parameters in each finger are the default values previously referred to, with only minor changes, mainly in the thumb. For this reason, trying to optimise the default values without differentiating all the joints present in each finger would result in a poor result, especially when attributes from one joint may influence the motion of another joint in the same finger. For instance, adjusting the *springref* in the FJ1 joint may provoke a different dynamic in the motion of joint FJ2. This topic and the nomenclature were explored in 2.2.1.

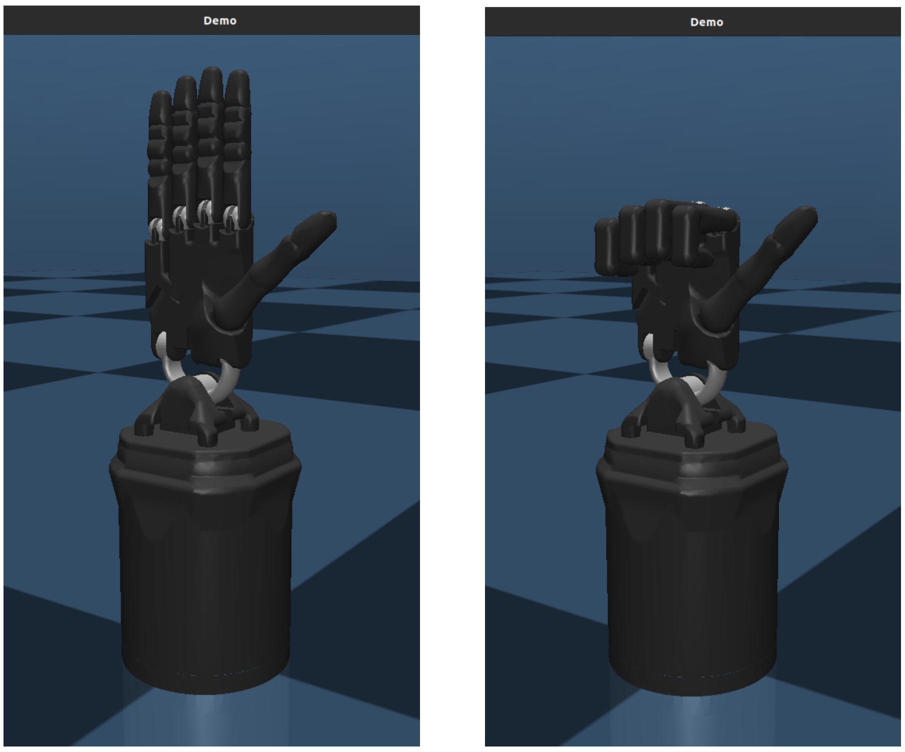
4.2.1 Model approach

Two distinct approaches were devised for the optimisation process, initially targeting the four fingers. The first approach involved a reduced number of parameters and a narrower search space. Conversely, the second approach encompassed all the previously mentioned parameters for each joint and entailed a broader search space. Following the completion of these two procedures, another calibration method will be employed specifically for the thumb. The selection of this strategy will be based on the results obtained from the preceding approaches.

- **First approach:** This approach has the main objective of testing the previously developed pipeline and understanding the effect of a smaller search space associated with a shorter number of parameters.

For this effect, it starts by not considering the *range* parameter. The decision to exclude this parameter was based on observations made during the analysis of real hand motion, where it was noted that the limits of joint movement only exhibited slight deviations from the default value of 1.57 radians. Besides these deviations not being particularly significant, also the presence of two different values for the range attribute added complexity to the algorithm. Consequently, a simplified version of the approach was chosen to streamline the optimisation process.

In this strategy, a trajectory was created that, starting from an open hand position, closes the middle, first, ring and little finger and then stretch them again. This trajectory can be seen in 4.3. In this figure, we may see the stretch hand, the initial position, on 4.3a. It is followed by the closed hand 4.3b, which corresponds to the intermediate position. The sequence concludes with the hand returning to the stretched position, as illustrated in figure 4.3a.



(a) Initial and Final position (b) Intermediate position

Figure 4.3: Illustration of intermediate steps in trajectory planning for calibration in the simulated environment.

All trajectories are defined in a YAML format file, where each trajectory is assigned a unique name and includes the respective goal for that joint and the time duration until the objective is reached.

Furthermore, it is important to note that in this particular trajectory, the FJ4 joint is not involved, and for this reason, no additional parameters or changes have been included.

Also, noting the fact that the FJ3 joint has a really similar motion in both simulations and reality, which can be observed in 4.4, associated with a search for a simplified and fast solution, it was decided not to consider changes in this joint for this particular approach. This joint presents a joint MSE of 0.001194, which is particularly small compared to the other two joints.

In addition, after analysing the motion of the simulated and real hand visually, it was observed that the main difference in the discrepancy between those two movements occurred especially in joint FJ1, which directly affects the FJ2 joint too. In figure 4.5a is displayed a hand configuration from a specific trajectory. Also, it is possible to observe that the FJ1 joint is fully stretched. On the other hand, Figure 4.5b displays the hand configuration of the simulated hand in that same trajectory. However, in this specific instance, the FJ1 joint is already beginning to fold, thereby influencing the movement of the FJ2 joint as well. These images highlight the primary discrepancies between the two environments, leading to the initial belief that these two joints played a more significant role, when compared to the others, in causing these differences.

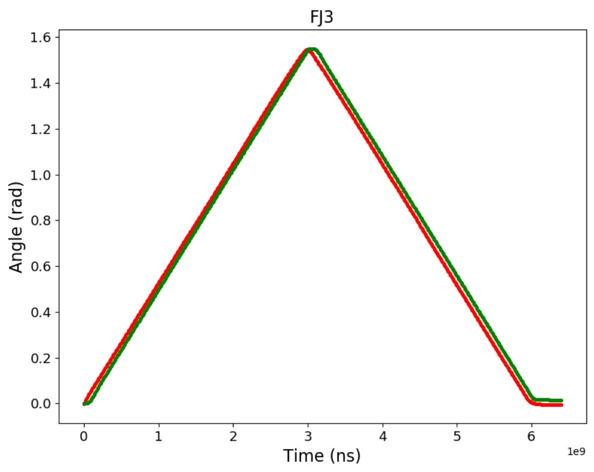
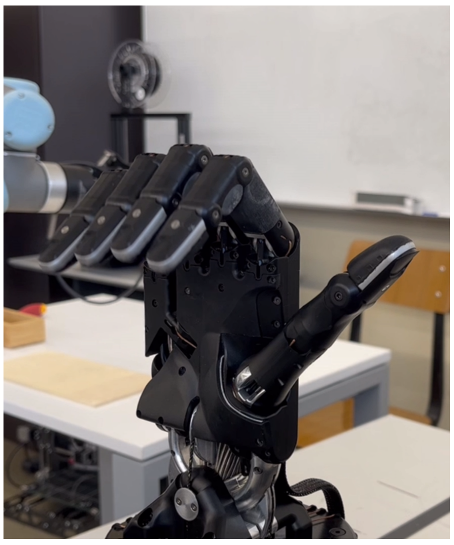


Figure 4.4: Graphical representation of FJ3 joint angle evolution before any optimisation process(in green is represented the simulated trajectory, and in red is the real hand trajectory).


(a) Real representation of the hand while closing

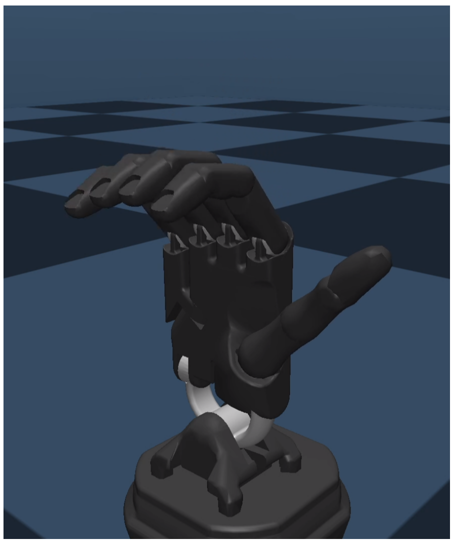

(b) Simulated representation of the hand while closing before calibration

Figure 4.5: Representation of a closing hand trajectory.

The parameter's default values may be intended as the starting point for the optimisation algorithm. The more appropriate they are, the best chance is for the optimisation process to return promising results. It is crucial that their initial values hold significance in the context of simulation. The values for the *frictionloss* and *damping* are already represented in the model. Nevertheless, the parameters *stiffness* and *springref* are not already defined, making it crucial to determine suitable initial values for these parameters before commencing the optimisation process. This ensures that the search space of the optimisation aligns with the intended behaviour and characteristics of the system. After conducting some research, it was discovered that the typical default values for these parameters can be observed in the comment provided in line 28 of the next code fragment.

In the following code fragment, it is also shown the changes made in the class *finger*. It was created a new class for all four fingers mentioned above, changing the default "FJ" prefix of all the fingers to "FFJ", "MFJ", "RFJ" and "LFJ", described in table 4.1, following an adaptation from the nomenclature explained in 2.2.1.

Table 4.1: Joint nomenclature for each finger.

Finger	**prefix**	**Joint**		
		FJ1	FJ2	FJ3
First	F	FFJ1	FFJ2	FFJ3
Middle	M	MFJ1	FFJ1	FFJ1
Ring	R	RFJ1	RFJ2	RFJ3
Little	L	LFJ1	LFJ3	LFJ3

```
19 <default class="finger">
20     <joint axis="1 0 0" range="0 1.5708" />
21     <default class="FFJ4">
22         <joint range="-0.349066 0.349066" />
23     </default>
24     <default class="FFJ3">
25         <joint frictionloss="0.001" damping="0.05" />
26     </default>
27     <default class="FFJ2">
28         <joint stiffness="0.002" springref="-100.0" frictionloss="0.001"
               damping="0.05" />
29     </default>
30     <default class="FFJ1">
31         <joint stiffness="0.002" springref="-100.0" frictionloss="0.001"
               damping="0.05" />
```

```
    </default>
    ...
```

- **Second approach:** This approach has the main goal of obtaining the best possible solution and verifying if augmenting the search space brings several consequences affecting the final result. It starts by taking into perspective the first one but increases the number of parameters being optimised, pursuing the optimal solution. The *range* parameter, not taken into account in the last method, is used in this one.

  Furthermore, in this approach, the FJ3 joint was considered. As previously mentioned, it has been observed that modifying the parameters of certain joints can impact the motion of other joints. All the parameters considered in joints FJ1 and FJ2 were added to this joint with the same default values.

  Important to add that the trajectory used in this method was the same one used in the first one. And for this reason, the FJ4 joint on every finger was not taken into account because it is not involved in this particular trajectory.

  In the next code fragment, we can see the modifications to the hand model:

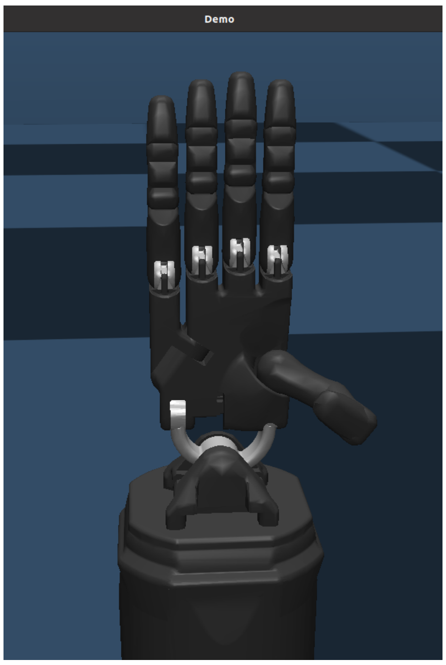
```
<default class="finger">
    <joint axis="1 0 0" range="0 1.5708" />
    <default class="FFJ4">
        <joint range="-0.349066 0.349066" />
    </default>
    <default class="FFJ3">
        <joint stiffness="0.002" springref="-100.0" frictionloss="0.001"
             damping="0.05" range="0 1.5708"/>
    </default>
    <default class="FFJ2">
        <joint stiffness="0.002" springref="-100.0" frictionloss="0.001"
             damping="0.05" range="0 1.5708"/>
    </default>
    <default class="FFJ1">
        <joint stiffness="0.002" springref="-100.0" frictionloss="0.001"
             damping="0.05" range="0 1.5708"/>
    </default>
...
```

- **Thumb approach:** After understanding which one of the previous approaches presented better results, the idea consists in using that approach to calibrate the thumb. This finger works separately from the other ones, and the actuators are totally independent of the rest, so for this reason, the optimisation may also be done independently. Furthermore, unlike the other fingers (excluding the little finger), the thumb possesses a total of five joints, thereby increasing the overall number of parameters requiring optimisation. Additionally, it is noteworthy that the thumb, unlike the little finger, consists of five independent actuators, introducing a distinctive feature not present in the little finger.

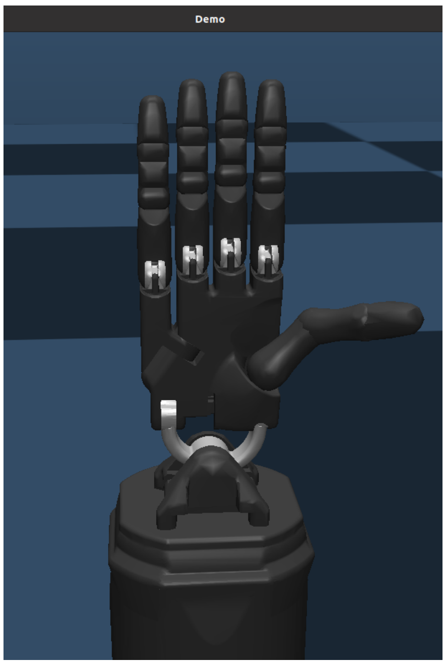
In accordance with the previously illustrated nomenclature, the thumb adheres to the same conventions, employing the designation "THJ" in conjunction with the respective joint number.

The thumb, by default, has a quite similar representation of the real motion in the simulation. It was also noted that adding the *stiffness*, *springref*, and *range* would result in poorer results, mainly because of a higher number of parameters to optimise, which directly affects the complexity of the problem. As the number of parameters increases, the search space tends to grow exponentially, and the possibility of overfitting rises. Having in mind these two observations, it was only optimised the *damping*, *frictionloss* parameters.

A new trajectory specifically designed for this finger was generated. Figure 4.6 presents the intermediate steps of this trajectory. It begins from a stretched position, as previously depicted in Figure 4.3a. Then, it progresses through subsequent stages represented by Figures 4.6a and 4.6b, ultimately concluding in the same initial position. This trajectory was created having in mind the different actuator ranges in each joint on the thumb, meaning that the trajectory goes through all the scope regarding the range limits. Nevertheless, due to its larger motion time in this trajectory, it is expected that the optimisation process for the thumb may require slightly more time compared to the other fingers.



(a) First Intermediate position



(b) Second Intermediate position

Figure 4.6: Illustration of intermediate steps in trajectory planning for thumb calibration in the simulated environment.

4.3 Data Acquisition and Processing Pipeline

In order to create a highly realistic digital twin resembling its real-world counterpart, it becomes imperative to fine-tune specific physical parameters within the underlying physics engine. However, prior to implementing this algorithm, the development of a robust pipeline is essential. This pipeline should be capable of extracting data from both the real hand and the simulated hand, processing and analysing the collected data, and ultimately facilitating a comprehensive comparison between the two datasets.

Figure 4.7 presents a comprehensive class diagram, offering a detailed illustration of the data acquisition and processing algorithm, along with its connection to the main optimisation algorithm. It elucidates the relationships and interdependencies among the various classes involved in the strategy. By visualising the structure and interactions within the algorithm, the class diagram offers a formal representation that facilitates a deeper understanding of the data flow and integration points between the data acquisition, processing, and optimisation components.

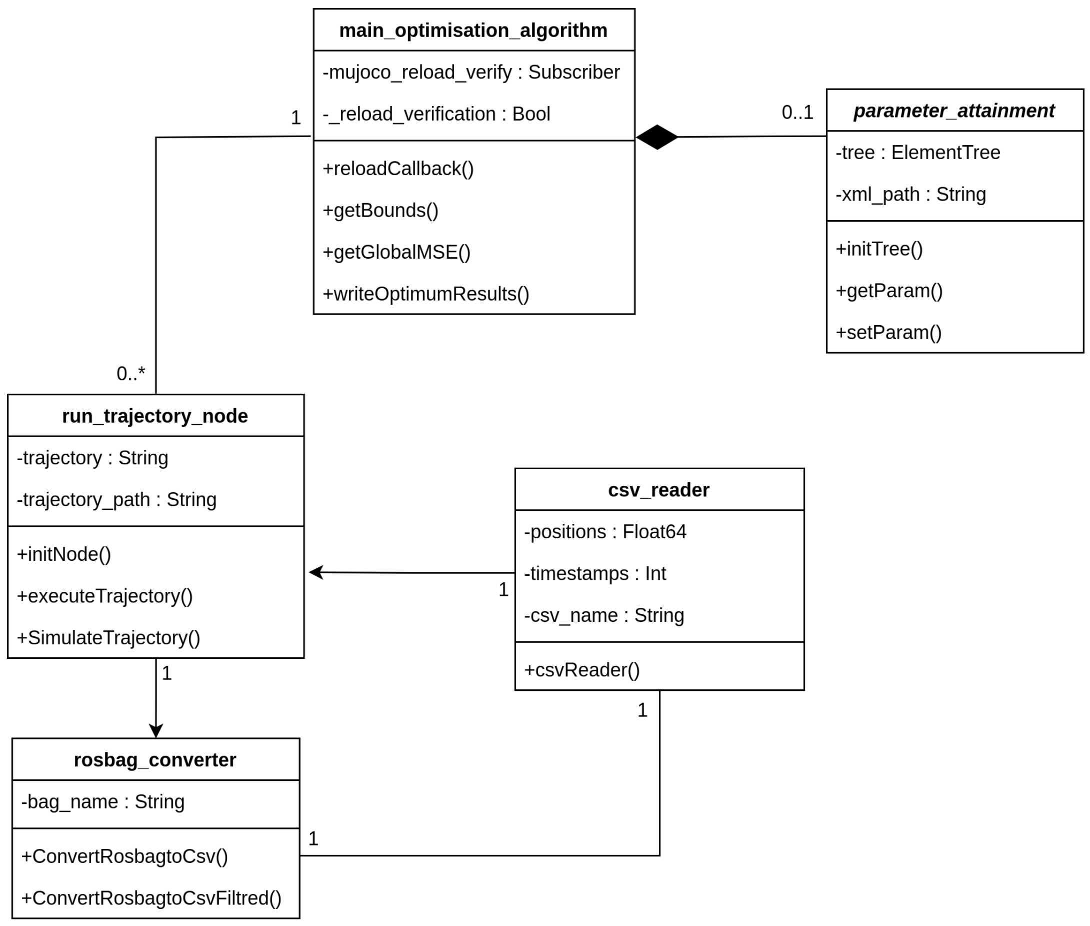


Figure 4.7: Acquisition and processing data pipeline class diagram

Rosbag Converter module (rosbag_converter): In order to compare the angles of the real and simulated robotic hand, it is logical to understand that there is a need to access the data from both the real hand and the simulated hand.

Starting with the real, while executing the predefined trajectory of the hand, a topic called *joint_states* was recorded in a rosbag. This topic contains information about the joint names, the angle of each joint (position) in radians, the velocity and the effort. This last, effort, can be intended as a force applied in each joint. Beyond this information, we can also acquire details about the timestamp, which will be crucial later.

The same must be done in every simulation execution during the optimisation. However, the rosbag does not always start at the same exact moment nor stop because of different system loads and delays at the beginning of each trajectory. Moreover, to facilitate a comparison between each trajectory generated in the simulation and the actual hand movement, it is essential to synchronise their starting points. To address this challenge, a dedicated module needs to be developed. This module will effectively filter the data obtained from the rosbag and convert it into a CSV (Comma-Separated Values) format. The CSV file can then be accessed and utilised for a comprehensive comparison with the real hand trajectory.

In this module, it was first implemented a simple program, using pandas Python module [54], that converts the rosbag to a CSV with only the necessary information, such as the position of each joint to be later used on the main algorithm, the velocity, so that it could be known when the trajectory has started, and the timestamp of every reading. This allows us to have a global view of the trajectory and confirm that during the execution of different trajectories, a different number of readings were recorded, proofing the necessity to filter the data accurately.

Having that in mind, based on the previous one, a program was implemented that, in addition to converting the rosbag to a CSV, also filters the data. The commencement of the trajectory can be ascertained by examining the velocity of the joints, whereby, if a substantial leap is observed, it serves as clear evidence for the initiation of the movement. After that, it defines the hand movement pre-establishing the number of iterations, and ends up removing all unnecessary data since the beginning of the rosbag's record and the start of the hand trajectory, along with the end of the trajectory and the shutdown of the rosbag's record.

Additionally, this module facilitates the preservation and storage of backup data from the trajectories executed in the simulated hand.

CSV interpreter module (csv_reader): Having all the necessary data organised and filtered in a CSV file, a module was created to interpret this file and return only the absolutely essential information. In brief, this program takes into input the filtered CSV file and returns the angle of each joint and the instant that occurred each reading. It is relevant to note that this process must be executed in every simulation and, for that reason, must be computationally light so as not to unnecessarily extend the execution time of each iteration in the main optimisation program.

Parameter Attainment module (Parameter_attainment): Considering the preceding points, determining optimal parameters requires the capacity to access and modify the parameters specified in the MuJoCo model file. To fulfil this requirement, the xml.etree.ElementTree library [55] in Python was employed. This library offers a user-friendly and efficient API for generating and interpreting XML data.

However, it is crucial to remember that some parameters, such as the *range*, have multiple values in the same attribute of a specific element, which hinders the use of this module. To overcome this problem, two separate functions were created to read and process the attribute values depending on the number of numeric numbers present in the string of that same attribute. Despite that, it was also developed a way to read and update the attributes of a particular defined element or all the elements simultaneously from the XML file and save them on an array, depending on the pretended result and input. In the case of this work, each element from the file may be intended as a certain joint from a certain finger.

Control Trajectory Node module (run_trajectory_node): In this module, we can find the node responsible for the execution of the defined trajectory in the simulator and also in the real robotic hand. It is liable for the initialisation of that same node, the trajectory selection and the rosbag's recording during the simulation of that same trajectory.

In addition to what has been mentioned, to execute a specific trajectory, a method that takes the pretended trajectory's name and previously initialised node as input was developed. This program, in brief, sends the order to execute a particular trajectory to the node mentioned before.

In parallel with this previous method, another program focuses on starting the record of the *joint_states* topic before the start of the trajectory's execution. It also ensures that the record is ceased after the end of that trajectory.

This module establishes direct interaction with the Rosbag Converter module. The output derived from the aforementioned method, consisting of a Rosbag containing positional information and timestamps, serves as the raw data input for subsequent processing within the Rosbag Converter module.

Similarly, the CSV Interpreter module interfaces directly with the Rosbag Converter module, creating an additional dependency on the present module. This is due to the fact that each filtered CSV file returned by the Rosbag Converter module is utilised as input for the CSV Interpreter module. As a result, the main optimisation program gains access to each of the joint angle values, enabling comprehensive analysis and subsequent optimisation.

Main algorithm (main_optimisation_algorithm): Lastly, this module encompasses the main optimisation algorithm, leveraging the functionalities provided by all the previously mentioned programs. The objective is to find the set of parameters that best match the real-world hand movement data.

Due to the high complexity of the system, it would be very computationally heavy to try to find the best possible fit for the hand model. However, by significantly augmenting the publication rates of the *joint_states* topic in the simulation, more data points will be generated, thereby reducing the errors associated with comparing unsynchronised points in time. In essence, the augmented publication rate in the simulation ensures a higher density of points along the trajectory, creating a more accurate representation when compared to the relatively lower publication rate of real-world data.

For that reason, it was developed several helper functions used for the optimisation process. One of these functions includes finding the closest points in an array so that comparing the real movement and the simulated one can produce the best possible results.

Furthermore, it incorporates functionalities for computing the MSE of individual joints, individual fingers, or the overall median MSE across all hand joints. This will subsequently serve as the cost function within the optimisation algorithm. It is important to note that the value of the MSE may have a slight variation between simulations with the exact same parameters. This is due to the error that is generated from the discrepancy between the sample rate in the simulated and real trajectory, which was previously explained. The difference in the sample rates of each component causes a maximum error equal to half of the period of the component with a higher sample rate, in this case, the simulated trajectory. In this specific case, the sample rate of the trajectory angles is 250 Hz, provoking a maximum discrepancy of 0.002 seconds. This variance, however, is very small and does not affect the overall outcome of this algorithm.

Additionally, this implementation incorporates methods to establish predetermined bounds within a specified percentage of the default value of each parameter and to generate random parameters within those previously set limits.

To ensure the preservation and subsequent analysis of all algorithmic data, a method was implemented to write the results of each iteration from the optimisation algorithm onto a file. This file provides a comprehensive overview of the evolution of the error throughout the optimisation process, offering a global perspective.

The main program starts by initialising variables and setting up some components of ROS (Robot Operating System). It reads real hand movement data from a CSV file and prepares it for comparison with simulated data. Bounds for the parameters are defined based on the provided numeric parameters. The main optimisation loop proceeds by iteratively comparing the positions of the simulated hand with those of the real hand. The MSE is computed within each iteration to measure the disparity between these positions. If it has been verified that the error has decreased, the best parameters and the minimum error are globally updated.

Ensuring that both the simulated and real hand follow precisely the same trajectory, as defined in the YAML file, is of utmost importance. This is a crucial process to allow comparison and analysis between the simulated and real hand trajectories.

In addition, once all the parameters have been updated in the MuJoCo model during each iteration, it becomes essential to reload this same model. A straightforward explanation of how the model operates and how it is possible to reload it was made in 3.2.1. Nevertheless, to enable this functionality, a ROS publisher and subscriber were also implemented within this module as well as in the main ROS program responsible for the processes concerning the MuJoCo model.

These components establish communication through a new topic named *mujoco_reload_verify*, which indicates whether reloading is feasible and confirms whether the reloading process has occurred.

It is worth highlighting that this algorithm was not specifically developed for a particular type of optimisation. Consequently, its general implementation enables its usage in a vast range of simulation-based optimisation scenarios.

4.4 Optimisation Algorithm

With the aim of calibrating the simulation and, in parallel, showcasing the effectiveness of the pipeline previously developed, there is a necessity to create and implement a compatible optimisation method.

It is essential to analyse the problem at hand and verify what type of optimisation method should be used in this particular case. On the one hand, there is no mathematical expression that can be accessed that represents the hand movement in each joint regarding all the parameters that define the physics of the simulation in the MuJoCo model file. On the other hand, considering the high complexity of the system, attempting to find the best possible fitting expression for the hand model would impose a significant computational burden.

This method must allow the implementation of a black-box optimisation algorithm, which in turn, is the type of optimisation required for this specific scenario.

Having in mind the expression defining the general optimisation problem, presented in Equation 3.1, a black-box optimisation may be interpreted as the analysis of the function $f(\mathbf{x})$ where the constraints defining Ω are unknown or non-existent [56]. In turn, the optimisation and evaluation of functions with this type of mathematical representation can be more computationally expensive compared to problems with a known objective function.

In this specific scenario, the evaluation of the objective function requires the execution of simulations due to the involvement of an unknown function representing the model of the hand. Consequently, this situation leads to an increase in the execution time of the algorithm when compared to other problems regarding known objective functions. This increase is mostly due to time increments caused by each simulation iteration.

4.4.1 Random Search Optimisation

The first method used for optimisation was a Random Search algorithm. A profound explanation of this method may be encountered in [57]. This method may be used for optimising gradient-free objective functions. It can be adapted to effectively address black-box optimisation problems, which in turn satisfy the specifications of the problem at hand. A random search algorithm refers to an algorithm that incorporates elements of randomness or probability in its search process [58].

In this particular case, the algorithm implemented involves random changes to the parameters and saves the best solution encountered up to the current iteration. Despite not being the most efficient and even the best overall algorithm for this situation, this method presents some advantages compared to other gradient-free methods, being those mainly:

- *Simplicity*. Compared to other optimisation methods, it is considerably easy to implement and does not depend on hyperparameters, unlike other gradient-free methods.
- *Parallelization*. Keeping in mind that each iteration is independent of others enables the execution of multiple instances simultaneously, exploring, in parallel, different parts of the search space. As a result, the overall process can be accelerated.

4.4.2 Particle Swarm Optimization

Subsequently, a Particle Swarm Optimization (PSO) [59] algorithm was developed for the problem. This algorithm is a population-based stochastic optimisation technique established on the concept of swarm.

The main idea of a PSO algorithm is based on two related studies: The evolutionary algorithm uses a swarm technique to simultaneously search a large region in the search space of the objective function. The artificial life, in other words, the behaviour of social animals.

It is grounded on a population of particles travelling through the search space to discover the optimal solution for the given problem. It starts by defining the number of particles present in the population alongside the initial position of each particle. Those particles move on the search space at a determined velocity on every iteration, consequently changing their position. Each particle gradually converges towards the closest best global position during all the desired iterations.

On the other hand, this algorithm can be described in an alternative way [60].

Assuming that the swarm size is N, each particle position is defined as $X_i = (x_{i1}, x_{i2}, ..., x_{iD})$, in a D-dimensional space, the velocity $V_i = (v_{i1}, v_{i2}, ..., v_{iD})$, the individual optimal position (i.e. the optimal solution found by a particular particle) is $P_i = (p_{i1}, p_{i2}, ..., p_{iD})$ and the optimal global position (i.e. the optimal solution found by any of the particles in the population) is $P_g = (p_{g1}, p_{g2}, ..., p_{gD})$.

The following main formulas characterise the canonical PSO algorithm:

$$v_{i,t+1} = \omega * v_{i,t} + c_1 * rand_1 * (p_{i,t} - x_{i,t}) + c_2 * rand_2 * (p_{g,t} - x_{i,t}) \tag{4.2}$$

$$x_{i,t+1} = x_{i,t} + v_{i,t+1} \tag{4.3}$$

In 4.2 and 4.3, it is possible to visualise the expression responsible for defining the velocity on which each particle moves. In this expression, ω represents the inertia weight. The inertia determines the relation between the current particle velocity and the previous. On another note, c_1 and c_2 represent the cognitive and social parameters, respectively. The first one represents the tendency that the particle has to move towards its best local position. It roughly defines its self-awareness. The second one represents the tendency that the particle has to move towards the best global position. It establishes the influence of the swarm. The $rand_1$ and $rand_2$ represent random values between 0 and 1, which in turn adds randomness in order to match the real approach case.

A deeper explanation of this method may also be encountered in this article [60].

This algorithm has certain drawbacks, such as the presence of hyperparameters and the sensitivity to its values. Associated with that, there is a lack of guaranteed global optimum and high computation intensity. In other words, it may require a considerable number of iterations to evaluate the search space correctly, and despite that does not guarantee the best global optimum because the initial positions heavily influence it.

However, its implementation is relatively straightforward despite being slightly more complex than a random search algorithm. Besides that, it also better explores the search space when the hyperparameters are tuned. It is also more efficient in the sense that it uses past best global solutions to find the optimal one, not residing only in randomisations.

4.4.3 Bayesian Search Optimisation

Ultimately, it was developed, and the Bayesian Search Optimisation algorithm. This method is widely employed in the optimisation of hyperparameters for machine learning algorithms but could be viewed as the optimisation of an unknown black-box function [61]. Notably, this aligns with the objective of the present work.

This algorithm serves as a robust strategy for identifying the extrema of objective functions that involve costly evaluations or display significant noise. It is recognised as a technique with an efficient approach in terms of the number of iterations required to converge to a solution [62]. It integrates principles of Bayesian statistics and optimisation in order to find the most promising areas within the search space, balancing the algorithm between exploration and exploitation.

A more profound explanation may be encountered in [62], [63].

The algorithm used in this method was implemented using the scikit-optimize library (skopt) [64]. The method uses Gaussian Processes (GPs) for Bayesian optimisation. In turn, GPs are versatile and robust probabilistic models that find common applications in machine learning for tasks such as regression and classification. This approach assumes that the function values follow a multivariate Gaussian distribution, with the covariance of the function values determined by a GP kernel based on the parameters. In other words, the idea is to get an approximation of the objective function using a Gaussian process.

However, it also presents some disadvantages, such as the computational cost for a high number of iterations and for a large search space. In particular, the probabilistic models associated with this algorithm can become highly complex, mainly when dealing with a large number of iterations. As a result, they can require a significant amount of time to execute successfully.

Additionally, it is important to note that this method is highly conditional on hyperparameters and the initial values employed to construct the probabilistic model. Consequently, the performance of the algorithm becomes directly dependent on these factors, potentially introducing additional complexity or even transforming it into another optimisation problem itself.

Chapter 5

Results and discussion

Several approaches for the MuJoCo model were considered, as explained in 4.2.1, as well as several methods for optimisation, with the main objective of concluding which one is overall the best.

As said earlier, one of the main objectives is to make this calibration obtain the best possible results in a short interval of time. In all of the optimisation algorithms implemented, it was defined a certain maximum number of iterations as a stop condition. However, depending on the algorithm, the execution time may vary widely for the same number of iterations.

To prevent the optimisation process from running for an excessively long time and in order to create a considerably fair comparison between methods, it was decided to set a run-time reference of 4 hours for the optimisation process. It is important to refer that the 4-hour mark is not the stop condition but only a reference when defining the maximum number of iterations. For instance, the prediction of execution time for the Bayesian Search method is always dependent on the Central Processing Unit (CPU) load, which may alter over time, as well as the number of parameters and the length of the search space. For this reason, it is not practicable to predict with total certainty the expected execution time, though it is possible to have a general idea. On another note, this also enables a better comprehension of the cost associated with each method.

Table 5.1: Default parameter values in each joint for the first, middle, ring and little finger.

	FJ1	**FJ2**	**FJ3**
Damping	0.05	0.05	0.05
Friction loss	0.001	0.001	0.001
Stiffness	0.002	0.002	0.002
Spring reference	-100.0	-100.0	-100.0
Range	0 - 1.5708	0 - 1.5708	0 - 1.5708

It is important to add that the initial MSE present in the simulation hand for the default parameters in the MuJoCo model was 0.056890. Table 5.1 presents those default parameter values for the first, middle, ring and little finger.

First Model Approach: For each model approach, a specific search space was defined based on a predetermined percentage deviation from the default values. For this first strategy, it was used a ±40% deviation. This keeps the search space slightly small, allowing for fast optimisation and a better understanding of the sensitivity of the parameters to changes.

The first optimisation method used for this model approach was the Random Search algorithm because of the simple and straightforward implementation. It was settled 1000 as the maximum number of iterations.

In Figure 5.1 is possible to see the error evolution throughout all the iterations. The evolution of the optimisation process shows a rapid improvement during the initial hundred iterations. This method depends on the generation of random values, and for this reason, it is complicated to predict the error evolution and even to guarantee the existence of a satisfactory global error.

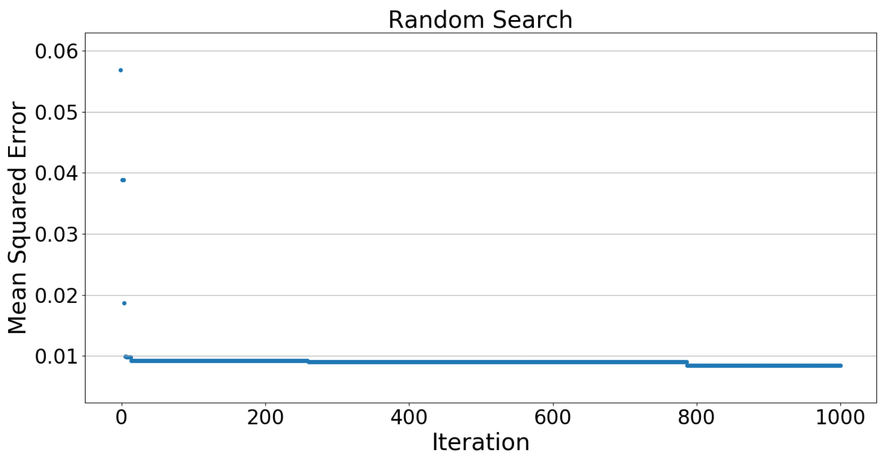


Figure 5.1: Graphical representation of the best MSE evolution using the Random Search algorithm on the first approach.

Figure 5.2 shows the evolution of the error with a lower error scale allowing for a better observation after 100 iterations. Due to fast conversion in the early stages of the process, it becomes difficult to analyse these values after a certain number of iterations. When the error is updated, it undergoes significant jumps, but these jumps are relatively infrequent.

After completing all the iterations, the algorithm achieved a considerable improvement, yielding the best result for the MSE corresponding to 0.008471. This corresponds to a reduction of 85.11% compared to the initial error.

Secondly, it was developed a PSO algorithm. This algorithm has to take as input several hyperparameters. Starting with the number of particles in the swarm that was set to be 10, randomly

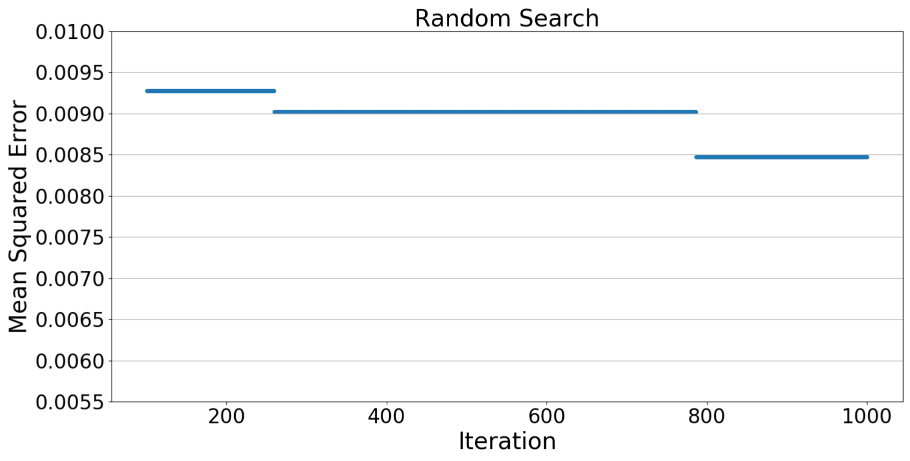


Figure 5.2: Graphical representation of the best MSE evolution using the Random Search algorithm on the first approach after 100 iterations.

generated within the deviation specified earlier. Each particle, in the end, made a total of 100 iterations corresponding to 1000 overall iterations throughout the algorithm. As explained in 4.4.2, *c1*, *c2* and *w* must also be settled. It was attributed to those constant values of 0.5, 0.3 and 0.9, respectively. These were defined through trial and error for a few iterations, verifying which one had the more promising results.

In Figure 5.3 is presented the MSE evolution throughout all the iterations. Similar to the Random Search, the evolution of the optimisation process shows a rapid improvement during the initial hundred iterations and subsequent convergence in the later stages.

Figure 5.4 represents the evolution of the error with a lower error scale allowing for a better observation. In contrast to the Random Search method, the current method exhibits more frequent updates of the best solution, with smaller differences in error throughout the number of iterations.

In the end, the algorithm had a decrease of 85.89% compared to the MSE obtained before optimisation. The final MSE value corresponded to 0.008024.

Lastly, it was developed a Bayesian Search Optimisation algorithm. This algorithm also has some hyperparameters as input. The maximum number of iterations was set to 500 due to the increasing computational cost observed during the execution of the algorithm, especially in later steps. Although the number of iterations was considerably reduced, the Bayesian Search method took a longer time to complete the optimisation process when compared to the other methods. It required a total of 4 hours and 4 minutes, surpassing the 4-hour mark.

In Figure 5.5, it is possible to notice the MSE evolution. Similar to the other two algorithms, it is noted a quick reduction in early iterations, stabilising thereafter.

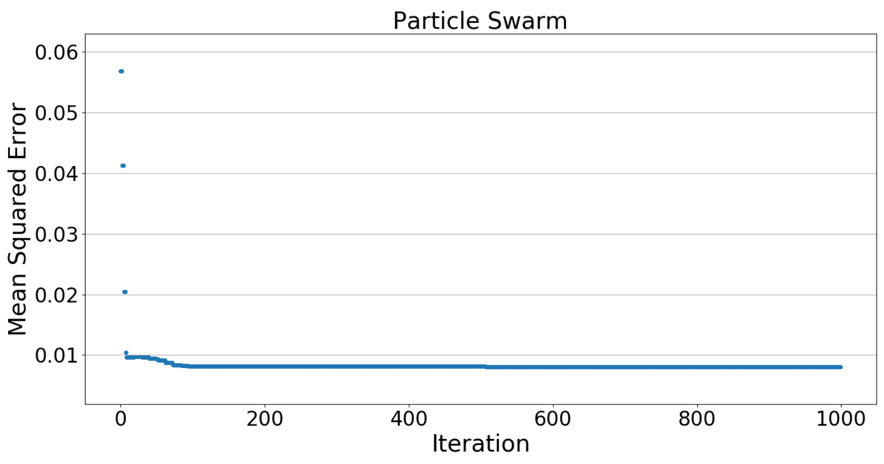


Figure 5.3: Graphical representation of the best MSE evolution using the PSO algorithm on the first approach.

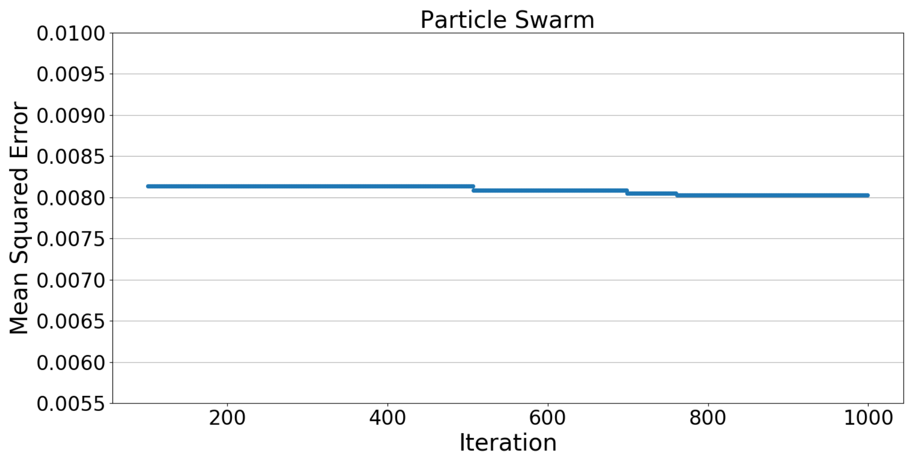


Figure 5.4: Graphical representation of the best MSE evolution using the PSO algorithm on the first approach after 100 iterations (10 iterations for each particle).

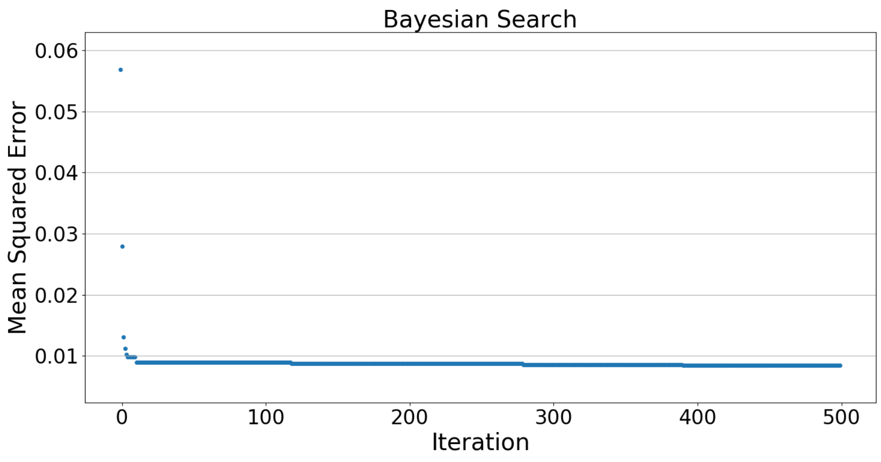


Figure 5.5: Graphical representation of the best MSE evolution using the Bayesian Search algorithm on the first approach.

Figure 5.6 illustrates the error evolution, presenting a lower scale that enables a more detailed observation of the error variations after 100 iterations. In contrast to the Random Search method, the current method exhibits a higher frequency of updates in the best value, accompanied by a smaller difference in error. The error tends to decrease throughout the optimisation process consistently, producing the belief that if the number of iterations of the algorithm were extended would present even better results.

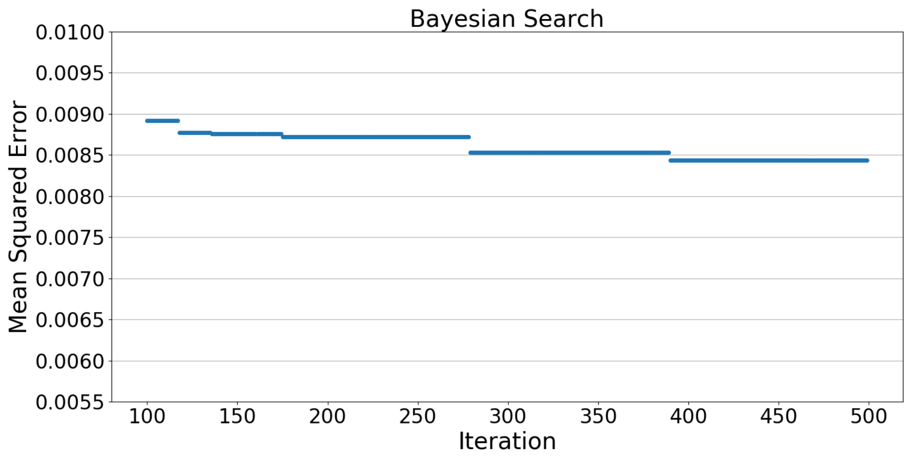


Figure 5.6: Graphical representation of the best MSE evolution using the Bayesian Search algorithm on the first approach after 100 iterations.

The final MSE achieved was 0.008416, for a global reduction of 85.21% when compared to the initial parameters.

When comparing all the previous optimisation methods, the best technique implemented so far was the PSO optimiser. Table 5.2 details all the results from each algorithm.

Table 5.2: Optimisation results obtained from the first approach.

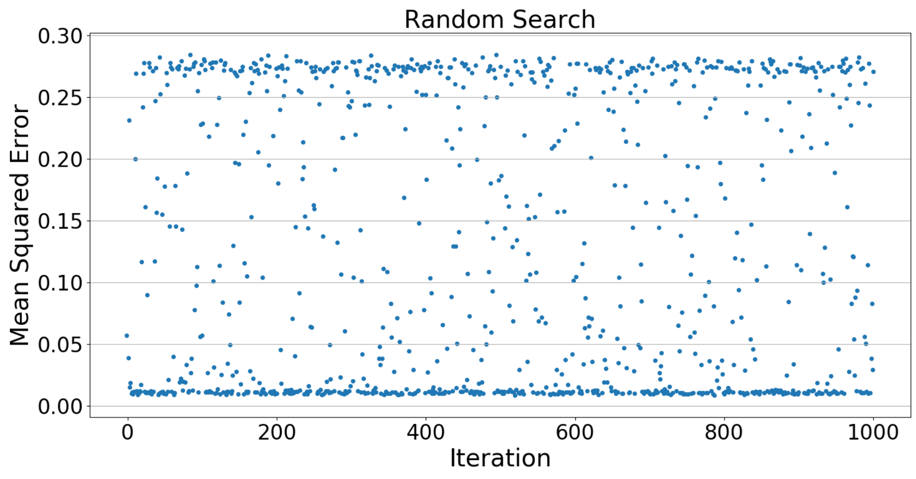
Method	**Reduction (%)**	**MSE**
Random Search	85.11	0.008471
PSO	85.89	0.008024
Bayesian Search	85.21	0.008416

However, all the methods presented a very similar result, with minor differences. This may be caused by a small search space associated with a considerably small number of iterations. In Figure 5.7, it is possible to verify the limitation provoked by the small search space. Especially on the Random Search, represented in 5.7a, the upper and lower boundaries are clearly defined and visually represented. Similarly, behaviour may be observed in 5.7b and 5.7c. However, in these last two, it is mostly verified the lower boundaries due to dependency on previously generated results or Gaussian methods based on the prior best results, allowing a conversion for the optimal solution over time. This analysis provides valuable insights regarding the delimitation of boundaries in the upcoming approach and may also explain the extremely similar results between methods.

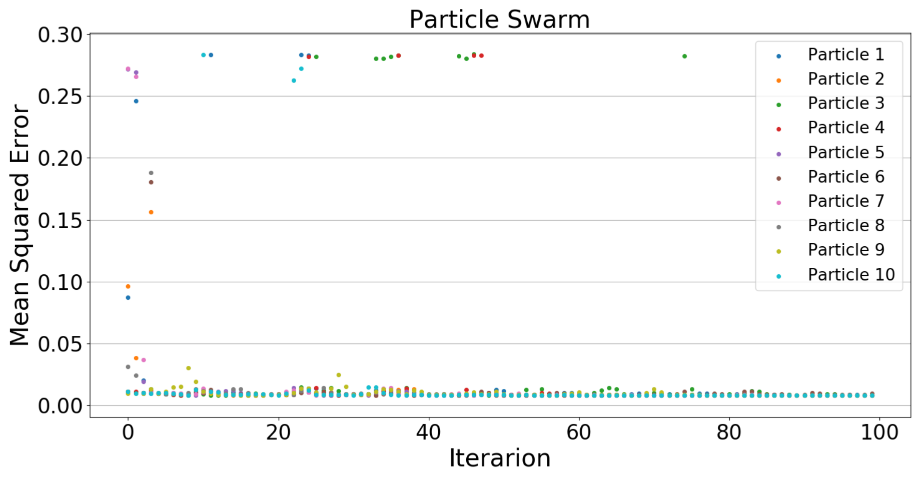
It is worth mentioning that the Bayesian Search algorithm had only half the number of iterations compared to the Particle Swarm optimisation method. The disparity in iteration counts primarily originated from the computational load imposed by the Bayesian Search algorithm, leading to slower execution, particularly during the advanced stages of the optimisation process. Also, it is noteworthy that with a more robust CPU, the performance of this method would be notably enhanced, leading to a faster and more efficient method and possibly a more satisfactory result.

In Figures 5.8 and 5.9, it is possible to see the difference in radians between the trajectory of both *FJ1* and *FJ2* before and after optimisation (in green represents the simulated trajectory and in red the real hand trajectory). This helps to demonstrate the global effect of the optimisation in the simulation model. In these figures is used a model obtained by the PSO due to the better results out of the three.

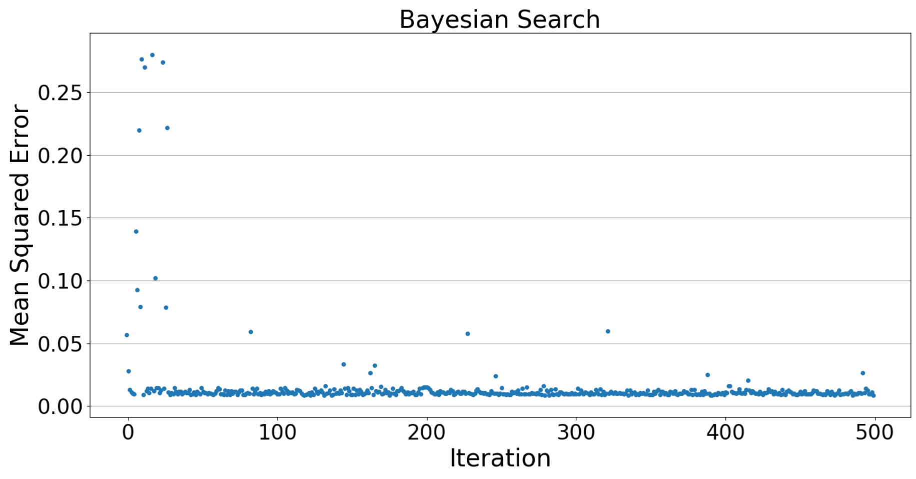
Second Model Approach: For this second approach, it was used a $\pm 70\%$ deviation for all the parameters except the *range*, which in turn was considered a $\pm 5\%$ deviation from the default 1.5708 radians. This decision was made based on the observation that the range values significantly influence the behaviour of each joint during motion. Additionally, it was noticed that in the real trajectory, the maximum and minimum angle values did not deviate significantly from the default values represented in *range*. If the deviation of this parameter were not reduced, it would have increased the possibility to occur an overfitting problem, consequently leading to the production of poorer results. Both the number of parameters and the search space have experienced an



(a) Random Search evolution

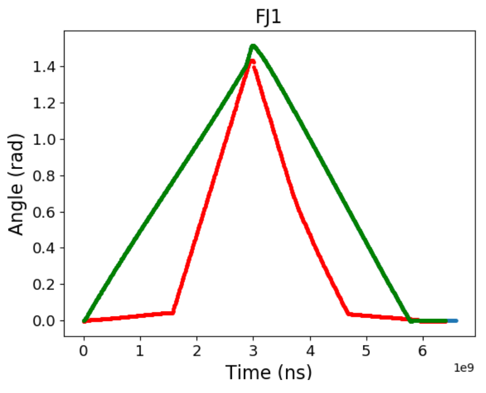


(b) PSO evolution

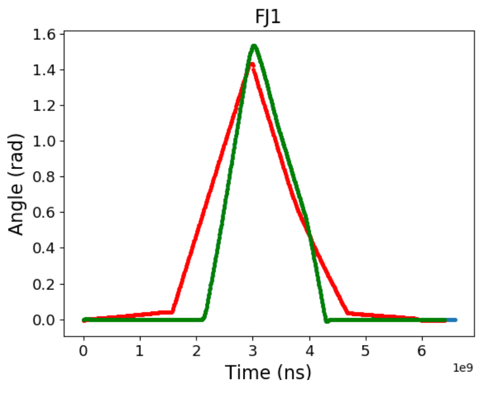


(c) Bayesian Search evolution

Figure 5.7: Graphical representation of MSE of each iteration for the first approach.

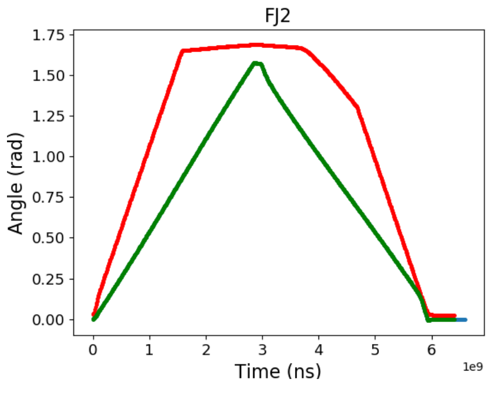


(a) Joint FJ1 trajectory before optimisation using the first approach.

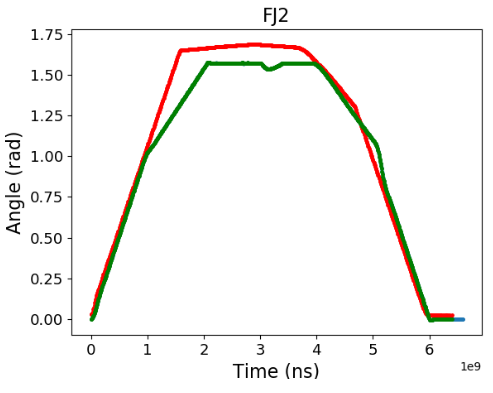


(b) Joint FJ1 trajectory after optimisation using the first approach.

Figure 5.8: Graphical representation of FJ1 joint angle evolution throughout the optimisation process for the first approach (in green is represented the simulated trajectory, and in red is the real hand trajectory).



(a) Joint FJ2 trajectory before optimisation using the first approach.



(b) Joint FJ2 trajectory after optimisation using the first approach.

Figure 5.9: Graphical representation of FJ2 joint angle evolution throughout the optimisation process for the first approach (in green is represented the simulated trajectory, and in red is the real hand trajectory).

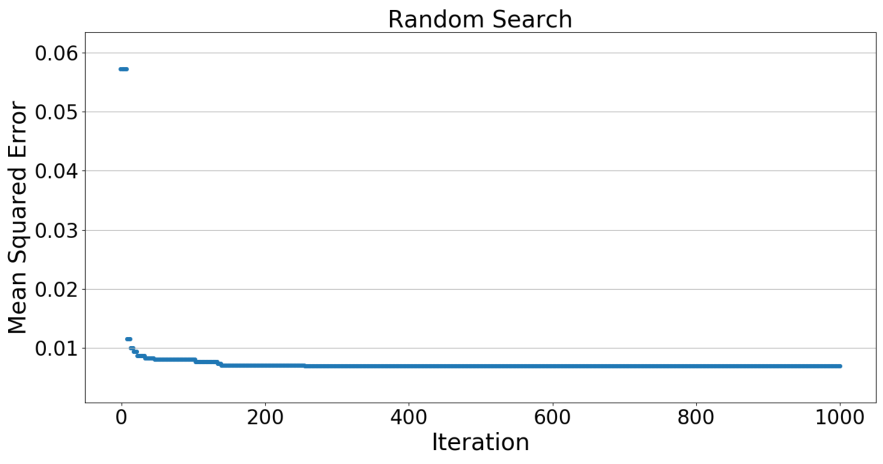


Figure 5.10: Graphical representation of the best MSE evolution using the Random Search algorithm on the second approach.

increase in size.

Figure 5.10 shows the error evolution throughout all the iterations using a Random Search algorithm with a maximum of 1000 iterations. The evolution of the optimisation process shows a rapid improvement during the initial hundred iterations, exhibiting similar behaviour to the approach mentioned earlier.

Figure 5.11 illustrates the error evolution with a reduced scale, facilitating a detailed observation of the results after 100 iterations.

Despite depending on mostly randomisations and having a bigger search space when compared to the previous method, it still has a considerably nice response and is able to improve the previous error obtained from the same method in the first approach by 17.75%.

After completing the entire iteration process, the algorithm demonstrated a significant improvement, resulting in an MSE of 0.006967. This represents a reduction of 87.75% compared to the initial error obtained using the default values.

Next was developed the PSO algorithm. The hyperparameters for this method, the *c1*, *c2* and *w* constants, were the same as the previous method. Also, the number of iterations and number o particles were kept the same. This allows an understanding of how this technique answers for an increase in the search space and the addition of another parameter.

In Figure 5.12 is presented the error evolution throughout all the iterations. The obtained response was similar to the preceding methods and approaches.

Figure 5.13 illustrates the evolution of the error with a reduced scale, facilitating a more detailed observation after 100 iterations, which in turn corresponds to 10 iterations for each particle. In this stage, it is possible to observe the constant convergence of the best solution over time. The MSE constantly decreases, placing the idea that if each particle had a few more iterations, a better result could have been found.

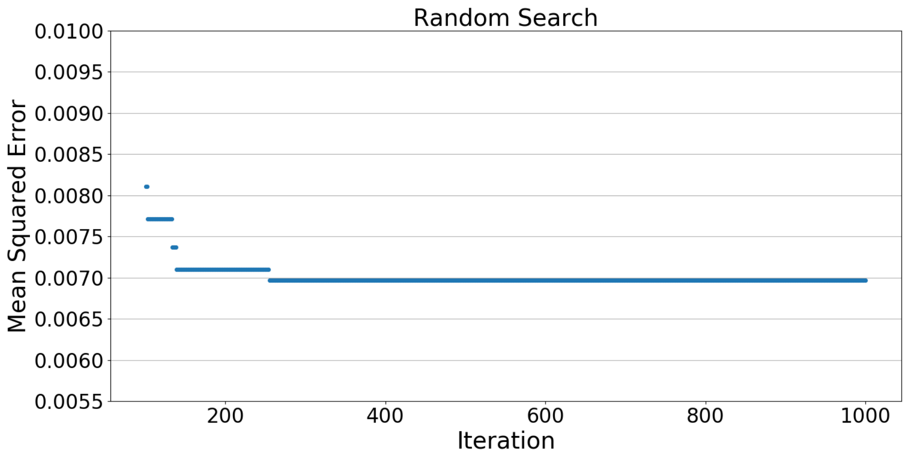


Figure 5.11: Graphical representation of the best MSE evolution using the Random Search algorithm on the second approach after 100 iterations.

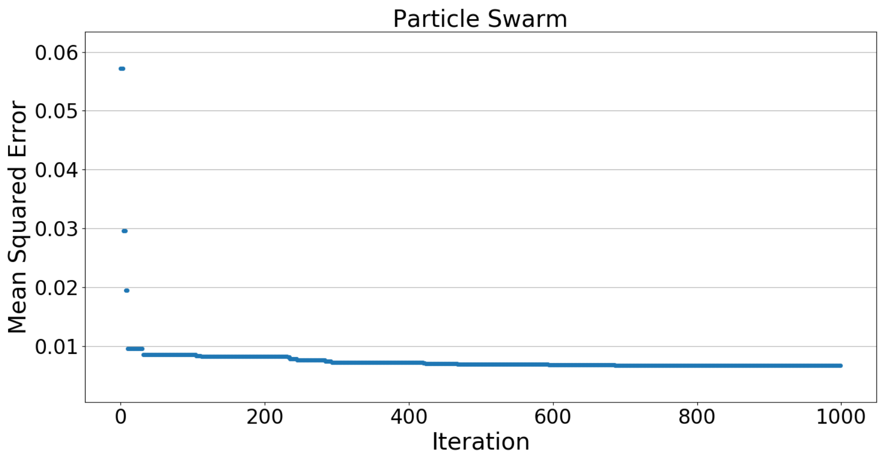


Figure 5.12: Graphical representation of the best MSE evolution using the PSO algorithm on the second approach.

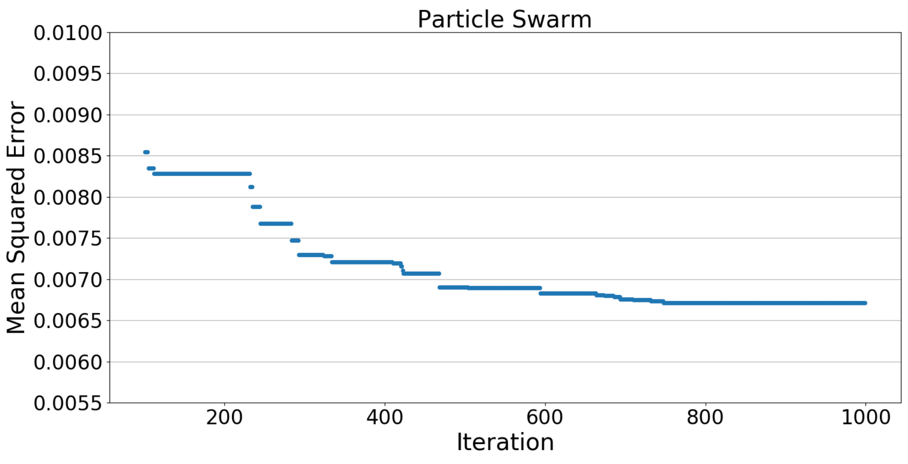


Figure 5.13: Graphical representation of the best MSE evolution using the PSO algorithm on the second approach after 100 iterations (10 iterations for each particle).

In the end, this method presented an improvement of 16.35% compared with the previous approach. When considering the default model, the global refinement of the model resulted in a decrease in the error of 88.20%. It has a final MSE of 0.006712.

Finally, the Bayesian Search algorithm was implemented for this approach. However, due to the increased number of parameters, the algorithm experienced a slight decrease in speed during the later iterations. On this occasion, this technique required a total of 4 hours and 17 minutes, experiencing a slight increase compared to the first approach.

Demonstrated in Image 5.14 is the evolution of the MSE throughout the optimisation process.

Figure 5.15 graphically represents the evolution of the MSE after 100 iterations, and similarly to the other methods, it verifies a slow convergence for the best result found.

The final MSE obtained in the optimisation process was 0.006277. This represents a significantly decreased of 88.97% in comparison to the initial parameters and 25.41% compared to the first approach. Despite the increase in time execution, for the same number of iterations, this method displayed the most significant improvement in percentage terms between approaches.

When analysing all the results, illustrated in 5.3, it was verified that until now, the Bayesian Search using the second approach was the method with the lowest MSE. This indicates that the model resulting from this approach offers a superior approximation of the simulation trajectory to the actual one.

The optimal parameters obtained by this method are displayed in Table 5.4. These, consequently, correspond to the best parameters found during these two approaches for the hand model. However, it is noteworthy to mention that in certain joints, some of the parameters correspond to the actual upper or lower limits of the previously defined search space characterised by a deviation of $\pm70\%$ in relation to the default values. This prompts the idea that the search space could be even more extended.

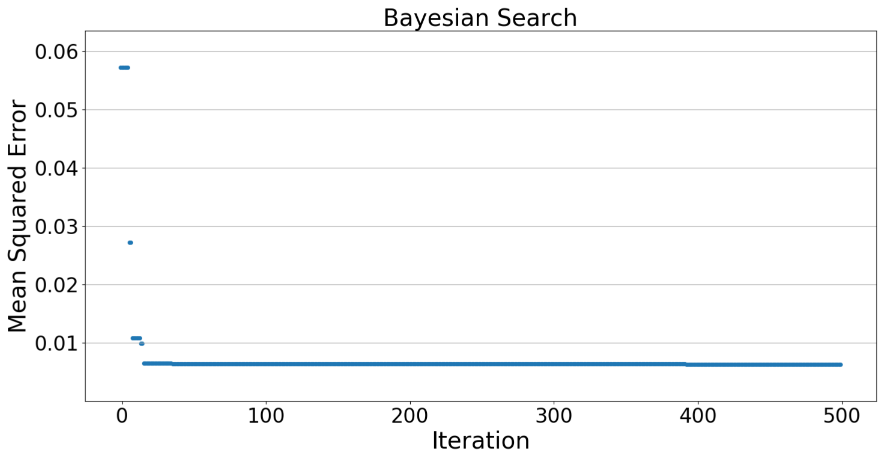


Figure 5.14: Graphical representation of the best MSE evolution using the Bayesian Search algorithm on the second approach.

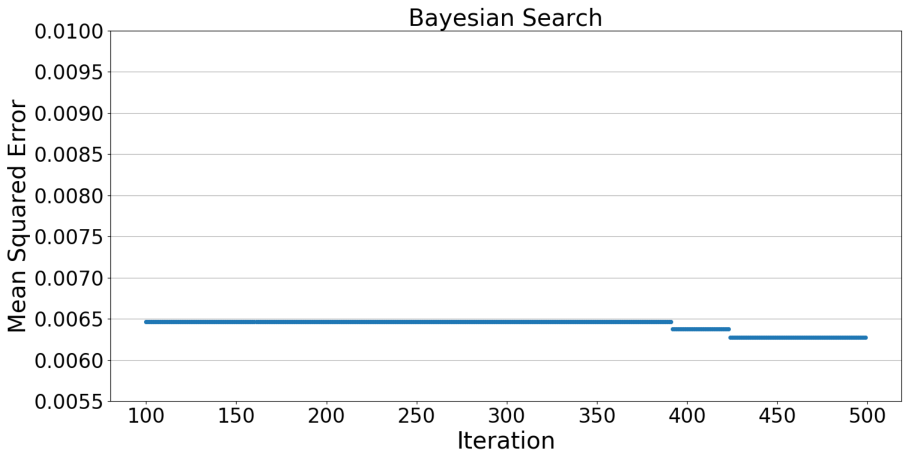


Figure 5.15: Graphical representation of the best MSE evolution using the Bayesian Search algorithm on the second approach after 100 iterations.

Table 5.3: Optimisation results obtained from the second approach.

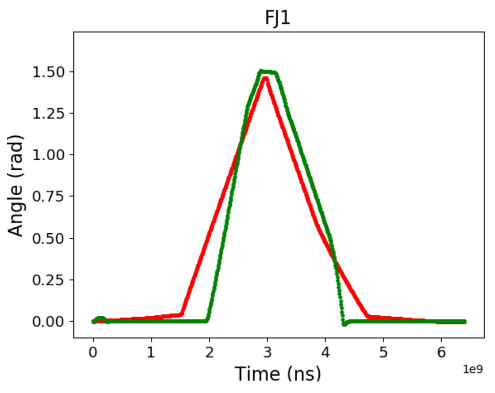
Method	**Reduction** (%)	**Improvement** (%) (Relation to first approach)	**MSE**
Random Search	87.75	17.75	0.006967
PSO	88.20	16.35	0.006712
Bayesian Search	88.97	25.41	0.006277

Also, it was verified some incongruities in relation to the FJ2 joint. As illustrated earlier in Image 5.9a, the maximum range for this joint was close to 1.65 radians, yet the maximum range during the optimisation process had tendencies to reduce instead of increase. After a careful analysis, it was verified that in order to reduce the error in the FJ2 joint, the maximum range for this joint needed to increase slightly. Nevertheless, considering the interconnection between this joint and the FJ1, when trying to increase the range limit in the FJ2 joint, a slight delay would be imposed on the motion of the FJ1 joint, consequently augmenting the error. Due to this barrier, for the search space specified, the algorithm found an optimal balance between the error entered in the FJ1 and FJ2 joint.

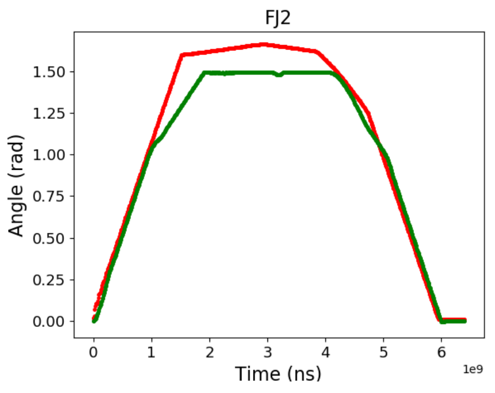
In Figure 5.16 is displayed a graphical representation of the joint angles throughout the trajectory execution after optimisation. For comparison, the motion using the default values were illustrated before in 5.8a, 5.9a and 4.4 for the FJ1, FJ2 and FJ3 joints, respectively.

Table 5.4: Parameter values in the MuJoCo model after optimisation using the second approach.

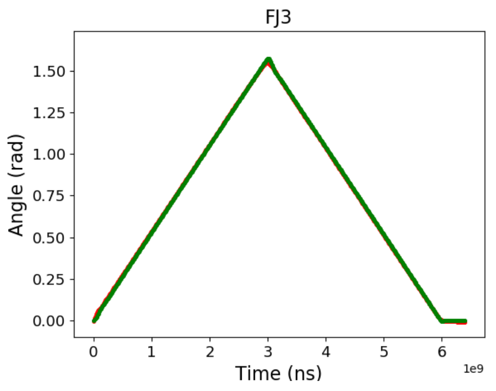
	FJ1	**FJ2**	**FJ3**
Damping	0.015	0.015	0.015
Friction loss	0.0003	0.0003	0.0003
Stiffness	0.001072	0.0006	0.0006
Spring reference	-66.263255	-30.00	-30.00
Range	0.0 - 1.495199	0.0 - 1.492260	0.0 - 1.60934



(a) Joint FJ1 trajectory after optimisation using the second approach.



(b) Joint FJ2 trajectory after optimisation using the second approach.



(c) Joint FJ3 trajectory after optimisation using the second approach.

Figure 5.16: Graphical representation of the joints FJ1, FJ2 and FJ3 angles after the optimisation process using the second approach (in green is represented the simulated trajectory, and in red is the real hand trajectory).

Thumb Calibration: After meticulous consideration of various approaches and thorough analysis of optimisation algorithms, it was concluded that among the three options, the Bayesian Search yielded the most favourable outcomes overall. This decision was based on considering factors such as the large search space and the substantial number of parameters to optimise.

Taking into consideration the previous conclusions, it was verified that the interval that defined the search space could be even more extended and because of that, it was defined as $\pm 200\%$, the deviation from the default parameters.

On another note, the thumb in the simulation already had a very similar motion to the real hand, which can be verified in Figure 5.18. The initial median MSE for the default values of this finger was 0.001514. The default values for *damping* and *frictionloss* were the ones used until now for the other fingers and represented in the previous excerpt of the MJCF file.

Figure 5.17 represents the graphical evolution of the MSE from the best parameters throughout each iteration.

Table 5.5: Parameters values in the MuJoCo model after thumb calibration.

	THJ1	**THJ2**	**THJ3**	**THJ4**	**THJ5**
Damping	-0.05	-0.05	0.05	-0.05	-0.05
Friction loss	0.003	-0.001	0.001	0.00016	0.00196

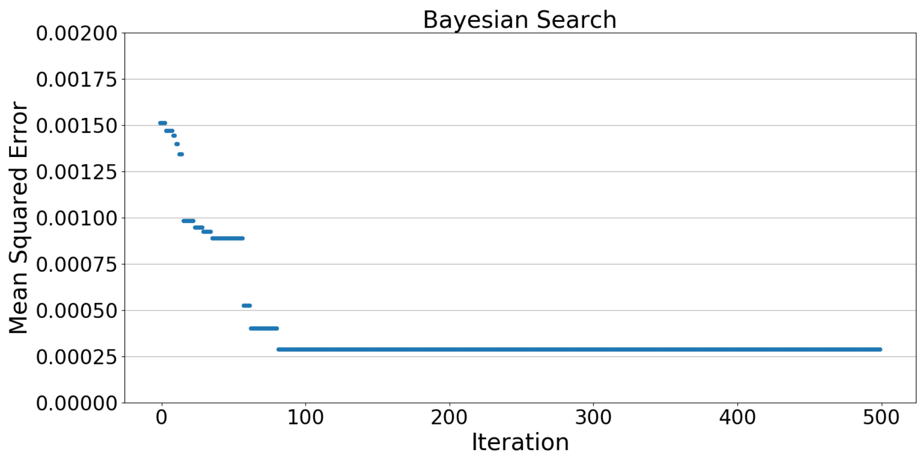
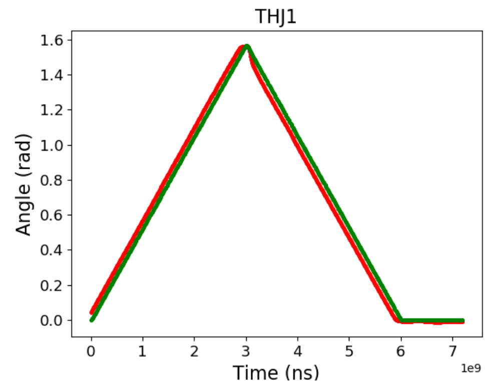


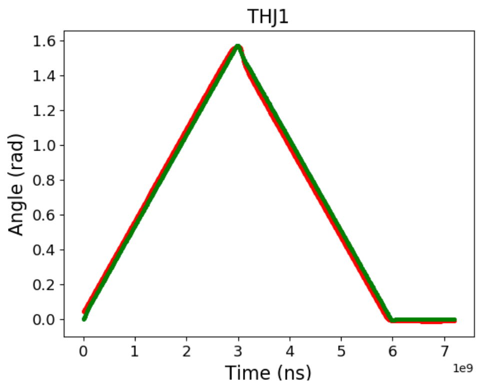
Figure 5.17: Graphical representation of the best MSE evolution using the Bayesian Search algorithm for the thumb.

The behaviour verified in this optimisation process was, in part, similar to the others, even though there were some differences in the parameters and search space. For instance, one slight difference verified in this method compared to the others was the slower conversion in the first 100 iterations, which the considerably larger search space can explain.

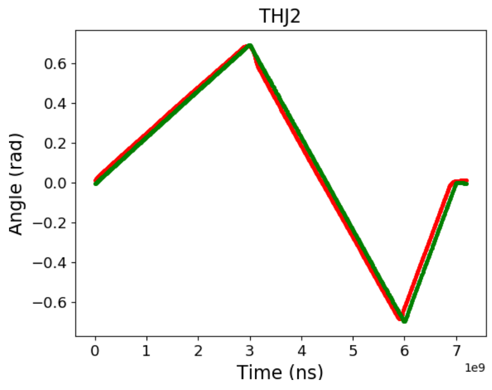
The final value of each parameter after optimisation may be visualised in 5.5. On the same note, the final MSE obtained with this implementation was 0.000289, corresponding to a reduction of 80% compared to the initial MSE verified in this finger. Figure 5.18 represents the angle values during the execution of the trajectory of each of the joints present in the thumb. In green is displayed the simulated motion, and in red is the real one.



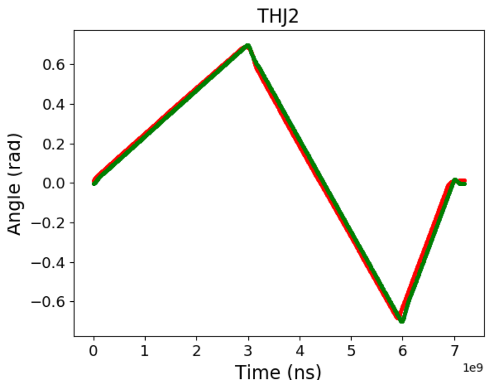
(a) Joint FJ1 trajectory before thumb calibration.



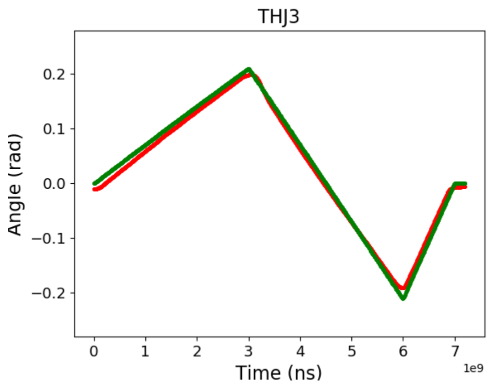
(b) Joint FJ1 trajectory after thumb calibration.



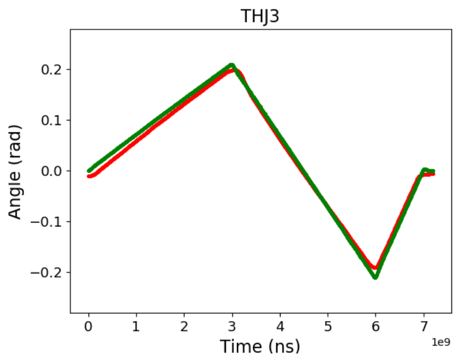
(c) Joint FJ2 trajectory before thumb calibration.



(d) Joint FJ2 trajectory after thumb calibration.

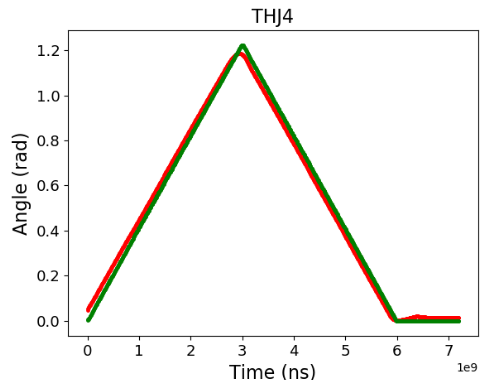


(e) Joint FJ3 trajectory before thumb calibration.

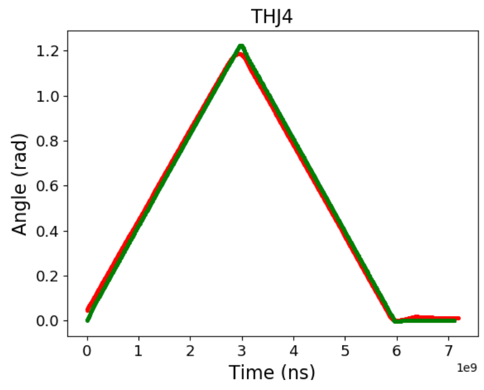


(f) Joint FJ3 trajectory after thumb calibration.

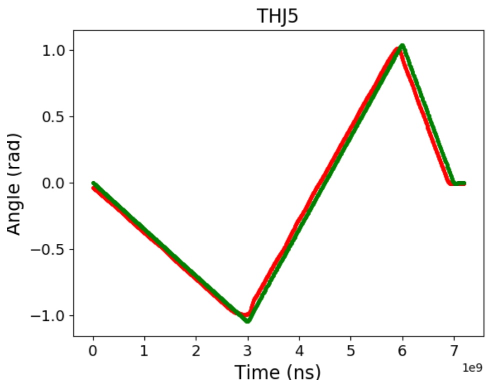
Figure 5.18: Graphical representation of the joints FJ1, FJ2, FJ3, FJ4, and FJ5 angles before (on the left side) and after (on the right side) the optimisation process used on the thumb calibration (in green, the simulated trajectory; in red, the real hand trajectory).



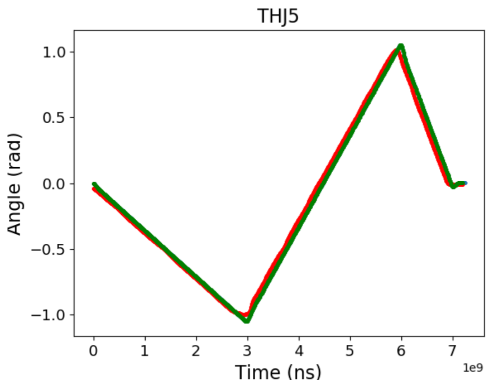
(g) Joint FJ4 trajectory before thumb calibration.



(h) Joint FJ4 trajectory after thumb calibration.



(i) Joint FJ5 trajectory before thumb calibration.



(j) Joint FJ5 trajectory after thumb calibration.

Figure 5.18: Graphical representation of the joints FJ1, FJ2, FJ3, FJ4, and FJ5 angles before (on the left side) and after (on the right side) the optimisation process used on the thumb calibration (in green, the simulated trajectory; in red, the real hand trajectory).

Chapter 6

Conclusion

Having in mind the high complexity of systems present in nowadays industry, having a realistic digital copy of those systems may have several advantages. This dissertation consists of the development and adaptation of a realistic DT for a dexterous robotic manipulator using MuJoCo physics engine.

To achieve a more realistic simulation, it is essential to perform a calibration process on this software. This calibration may be intended as an optimisation problem where the objective is to decrease the error between the trajectories executed by the simulated and real component. After the implementation of a pipeline capable of extracting the data and processing it, several optimisation algorithms were developed. Being those a Random Search, a Particle Swarm Optimization and a Bayesian Search algorithm. The different approaches and, principally, the developed pipeline responsible for the main acquisition and processing of data can be found in `https://github.com/TCorreia14/Shadow-simulation-calibration`.

The achieved results allow the conclusion that depending on the length of the search space and the hyperparameters used in each algorithm, different approaches present the best result. It was also determined that the correct delimitation of the search space plays a very important part in the algorithm's success.

On a separate note, in relation to this dissertation, a publication was submitted containing all the explanations about the implementation of the main pipeline used for the obtainment of data from the simulation and real robotic manipulator named: "Realistic model parameter optimization: Shadow robot dexterous hand use-case". Additionally, would also refer the co-authorship in the scientific paper on "Modeling and Realistic Simulation of a Dexterous Robotic Hand: SVH Hand use-case", which mainly mentions the adequacy of MuJoCo for realistic simulations.

In terms of future work, several topics may be explored and refined. Starting with the improvement of the previously explored methods. Having now a better understanding of each parameter reference value for the simulation model, defining a customised search space depending on the type of parameter would produce a potentially better result. Associated with that, finding better values for other parameters present in the model, such as the diagonal inertia, the mass of each component body, and the full inertia matrix, among others. Another particularly interesting topic to explore would be a development of an RL algorithm capable of, in a different and more engaging way, demonstrating all the results obtained regarding the simulation realism. In addition, a significant scientific contribution would be the adaptation of the model to incorporate a branched rope, allowing a simulation of insertion tasks, which is one of the main objectives of the project this work is inserted on.

References

[1] F. Pires, A. Cachada, J. Barbosa, A. P. Moreira, and P. Leitão, “Digital twin in industry 4.0: Technologies, applications and challenges”, 2019.

[2] E. Glaessgen and D. Stargel, “The digital twin paradigm for future nasa and u.s. air force vehicles”, Apr. 2012, ISBN: 978-1-60086-937-2. DOI: 10.2514/6.2012-1818. [Online]. Available: https://ntrs.nasa.gov/citations/20120008178.

[3] Y. Liu, S. Ong, and A. Nee, “State-of-the-art survey on digital twin implementations”, *Advances in Manufacturing*, vol. 10, no. 1, 2022, Cited by: 13. DOI: 10.1007/s40436-021-00375-w. [Online]. Available: https://link.springer.com/article/10.1007/s40436-021-00375-w.

[4] G. Shao, S. Jain, C. Laroque, L. H. Lee, P. Lendermann, and O. Rose, “Digital twin for smart manufacturing: The simulation aspect”, in *2019 Winter Simulation Conference (WSC)*, 2019, pp. 2085–2098. DOI: 10.1109/WSC40007.2019.9004659.

[5] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, “Digital Twin in manufacturing: A categorical literature review and classification”, *IFAC-PapersOnLine*, vol. 51, pp. 1016–1022, 11 Jan. 2018, ISSN: 24058963. DOI: 10.1016/J.IFACOL.2018.08.474.

[6] J. Opoku, S. Perera, R. Osei-Kyei, and M. Rashidi, “Digital twin application in the construction industry: A literature review”, *Journal of Building Engineering*, vol. 40, Aug. 2021, ISSN: 23527102. DOI: 10.1016/j.jobe.2021.102726.

[7] T. Uhlemann, C. Lehmann, and R. Steinhilper, “The digital twin: Realizing the cyber-physical production system for industry 4.0”, *Procedia CIRP*, vol. 61, pp. 335–340, 2017.

[8] I. Verner, D. Cuperman, A. Fang, M. Reitman, T. Romm, and G. Balikin, “Robot online learning through digital twin experiments: A weightlifting project”, in Jan. 2018, pp. 307–314, ISBN: 978-3-319-64351-9. DOI: 10.1007/978-3-319-64352-6_29.

[9] D. Romero, T. Wuest, J. Stahre, and D. Gorecky, “Social factory architecture: Social networking services and production scenarios through the social internet of things, services and people for the social operator 4.0”, in *Advances in Production Management Systems*, 2017.

[10] R. Meziane, M. J.-D. Otis, and H. Ezzaidi, “Human-robot collaboration while sharing production activities in dynamic environment: Spader system”, *Robotics and Computer-Integrated Manufacturing*, vol. 48, pp. 243–253, 2017, ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2017.04.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584515301447.

[11] A. A. Malik and A. Brem, “Digital twins for collaborative robots: A case study in human-robot interaction”, *Robotics and Computer-Integrated Manufacturing*, vol. 68, Apr. 2021, ISSN: 07365845. DOI: 10.1016/j.rcim.2020.102092.

[12] X. Wang, C.-J. Liang, C. C. Menassa, and V. R. Kamat, “Interactive and immersive process-level digital twin for collaborative human–robot construction work”, *Journal of Computing in Civil Engineering*, vol. 35, 6 Nov. 2021, ISSN: 0887-3801. DOI: 10.1061/(asce)cp.1943-5487.0000988.

[13] *Universal robots*. [Online]. Available: https://www.universal-robots.com/products/ur5-robot/ (visited on 02/05/2023).

[14] *Shadow robot*. [Online]. Available: https://www.shadowrobot.com/dexterous-hand-series/ (visited on 01/04/2023).

[15] *Dexterous hand documentation*. [Online]. Available: https://shadow-robot-company-dexterous-hand.readthedocs-hosted.com/en/latest/index.html (visited on 01/04/2023).

[16] *Syntouch*. [Online]. Available: https://syntouchinc.com/robotics/ (visited on 01/04/2023).

[17] M. Lambeta, P.-W. Chou, S. Tian, *et al.*, “DIGIT: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation”, *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 3838–3845, Jul. 2020. DOI: 10.1109/lra.2020.2977257. [Online]. Available: https://ieeexplore.ieee.org/document/9018215.

[18] R. Sutton, “Learning to predict by the method of temporal differences”, in *Machine Learning*. Aug. 1988, vol. 3, pp. 9–44. DOI: 10.1007/BF00115009.

[19] P. Kormushev, S. Calinon, and D. G. Caldwell, “Reinforcement learning in robotics: Applications and real-world challenges”, *Robotics*, vol. 2, no. 3, pp. 122–148, 2013, ISSN: 2218-6581. DOI: 10.3390/robotics2030122. [Online]. Available: https://www.mdpi.com/2218-6581/2/3/122.

[20] P. Kormushev, S. Calinon, and D. G. Caldwell, “Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input”, *Advanced Robotics*, vol. 25, no. 5, pp. 581–603, 2011. DOI: 10.1163/016918611X558261. [Online]. Available: https://doi.org/10.1163/016918611X558261.

[21] A. Suresh, D. Gaba, S. Bhambri, and D. Laha, “Intelligent multi-fingered dexterous hand using virtual reality (vr) and robot operating system (ros)”, in Jan. 2019, pp. 459–474, ISBN: 978-3-319-78451-9. DOI: 10.1007/978-3-319-78452-6_37.

[22] H. Lee, S. D. Kim, and M. A. U. A. Amin, "Control framework for collaborative robot using imitation learning-based teleoperation from human digital twin to robot digital twin", *Mechatronics*, vol. 85, Aug. 2022, ISSN: 09574158. DOI: 10.1016/j.mechatronics.2022.102833.

[23] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx", *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, pp. 4397–4404, June Jun. 2015, ISSN: 10504729. DOI: 10.1109/ICRA.2015.7139807.

[24] E. Coumans, "Bullet physics simulation", in *ACM SIGGRAPH 2015 Courses*, ser. SIGGRAPH '15, Los Angeles, California: Association for Computing Machinery, 2015, ISBN: 9781450336345. DOI: 10.1145/2776880.2792704. [Online]. Available: https://doi.org/10.1145/2776880.2792704.

[25] E. Drumwright, J. Hsu, N. Koenig, and D. Shell, "Extending open dynamics engine for robotics simulation", in *Simulation, Modeling, and Programming for Autonomous Robots*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 38–50, ISBN: 978-3-642-17319-6.

[26] *Havok physics engine*. [Online]. Available: https://www.havok.com/.

[27] *Physx physics engine*. [Online]. Available: https://developer.nvidia.com/gameworks-physx-overview.

[28] F. M. Ribeiro, T. Correia, J. Lima, G. Gonçalves, and V. H. Pinto, "Modeling and realistic simulation of a dexterous robotic hand: Svh hand use-case", in *2023 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2023, pp. 132–138. DOI: 10.1109/ICARSC58346.2023.10129643.

[29] J. Lee, M. X. Grey, S. Ha, *et al.*, "Dart: Dynamic animation and robotics toolkit", *Journal of Open Source Software*, vol. 3, no. 22, p. 500, 2018. DOI: 10.21105/joss.00500. [Online]. Available: https://doi.org/10.21105/joss.00500.

[30] M. A. Sherman, A. Seth, and S. L. Delp, "Simbody: Multibody dynamics for biomedical research", *Procedia IUTAM*, vol. 2, pp. 241–261, 2011, IUTAM Symposium on Human Body Dynamics, ISSN: 2210-9838. DOI: https://doi.org/10.1016/j.piutam.2011.04.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210983811000241.

[31] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control", *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, ISSN: 21530858. DOI: 10.1109/IROS.2012.6386109.

[32] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications", *IEEE Access*, vol. 9, pp. 51 416–51 431, 2021, ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3068769.

[33] *Modeling - mujoco documentation*. [Online]. Available: https://mujoco.readthedocs.io/en/stable/modeling.html (visited on 03/11/2023).

[34] *Overview - mujoco documentation*. [Online]. Available: https://mujoco.readthedocs.io/en/stable/overview.html#instance (visited on 03/07/2023).

[35] B. Acosta, W. Yang, and M. Posa, "Validating robotics simulators on real-world impacts", *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6471–6478, 2022. DOI: 10.1109/LRA.2022.3174367.

[36] OpenAI, M. Andrychowicz, B. Baker, *et al.*, *Learning dexterous in-hand manipulation*, 2019. arXiv: 1808.00177.

[37] J. Sanchez, J. A. Corrales, B. C. Bouzgarrou, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey", *International Journal of Robotics Research*, vol. 37, pp. 688–716, 7 Jun. 2018, ISSN: 17413176. DOI: 10.1177/0278364918779698.

[38] *Modeling: Composite objects*. [Online]. Available: https://mujoco.readthedocs.io/en/stable/modeling.html#curdf (visited on 02/22/2023).

[39] W. Wang, D. Berenson, and D. Balkcom, "An online method for tight-tolerance insertion tasks for string and rope", *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2015-June, pp. 2488–2495, June Jun. 2015, ISSN: 10504729. DOI: 10.1109/ICRA.2015.7139532.

[40] E. Yoshida, K. Ayusawa, I. G. Ramirez-Alpizar, K. Harada, C. Duriez, and A. Kheddar, "Simulation-based optimal motion planning for deformable object", in *2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 2015, pp. 1–6. DOI: 10.1109/ARSO.2015.7428219.

[41] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning", in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 489–494. DOI: 10.1109/ROBOT.2009.5152817.

[42] M. Zucker, N. Ratliff, A. D. Dragan, *et al.*, "Chomp: Covariant hamiltonian optimization for motion planning", *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013. DOI: 10.1177/0278364913488805.

[43] F. Tao, B. Xiao, Q. Qi, J. Cheng, and P. Ji, "Digital twin modeling", *Journal of Manufacturing Systems*, vol. 64, pp. 372–389, Jul. 2022, ISSN: 02786125. DOI: 10.1016/j.jmsy.2022.06.015.

[44] N. Kousi, C. Gkournelos, S. Aivaliotis, C. Giannoulis, G. Michalos, and S. Makris, "Digital twin for adaptation of robots' behavior in flexible robotic assembly lines", vol. 28, Elsevier B.V., 2019, pp. 121–126. DOI: 10.1016/j.promfg.2018.12.020.

[45] E. VanDerHorn and S. Mahadevan, "Digital twin: Generalization, characterization and implementation", *Decision Support Systems*, vol. 145, Jun. 2021, ISSN: 01679236. DOI: 10.1016/j.dss.2021.113524.

[46] C. Chen, Q. Liu, P. Lou, W. Deng, and J. Hu, "Digital twin system of object location and grasp robot", in *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, 2020, pp. 65–68. DOI: 10.1109/ICMCCE51767.2020.00021.

[47] A. J. Lockett, "Review of optimization methods", in *General-Purpose Optimization Through Information Maximization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 17–41, ISBN: 978-3-662-62007-6. DOI: 10.1007/978-3-662-62007-6_2. [Online]. Available: https://doi.org/10.1007/978-3-662-62007-6_2.

[48] N. V. Sahinidis, "Optimization under uncertainty: State-of-the-art and opportunities", *Computers & Chemical Engineering*, vol. 28, no. 6, pp. 971–983, 2004, FOCAPO 2003 Special issue, ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2003.09.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098135403002369.

[49] G. Venter, "Review of optimization techniques", in Dec. 2010, ISBN: 9780470686652. DOI: 10.1002/9780470686652.eae495.

[50] M. Plappert, M. Andrychowicz, A. Ray, *et al.*, *Multi-goal reinforcement learning: Challenging robotics environments and request for research*, 2018. eprint: arXiv:1802.09464.

[51] OpenAI, I. Akkaya, M. Andrychowicz, *et al.*, *Solving rubik's cube with a robot hand*, 2019. arXiv: 1910.07113.

[52] *Gymnasium-robotics documentation: Reach*. [Online]. Available: https://robotics.farama.org/envs/shadow_dexterous_hand/reach/ (visited on 06/21/2023).

[53] M. Matulis and C. Harvey, "A robot arm digital twin utilising reinforcement learning", *Computers and Graphics (Pergamon)*, vol. 95, pp. 106–114, Apr. 2021, ISSN: 00978493. DOI: 10.1016/j.cag.2021.01.011.

[54] Pandas Development Team, *Pandas*, 2020. [Online]. Available: https://pandas.pydata.org.

[55] Python Software Foundation, *xml.etree.ElementTree*, 2006. [Online]. Available: https://docs.python.org/3/library/xml.etree.elementtree.html.

[56] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. Le Digabel, "Two decades of blackbox optimization applications", *EURO Journal on Computational Optimization*, vol. 9, p. 100 011, 2021, ISSN: 2192-4406. DOI: https://doi.org/10.1016/j.ejco.2021.100011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2192440621001386.

[57] "An overview of simulation optimization via random search", in *Simulation*, ser. Handbooks in Operations Research and Management Science, S. G. Henderson and B. L. Nelson, Eds., vol. 13, Elsevier, 2006, pp. 617–631. DOI: https://doi.org/10.1016/S0927-0507(06)13020-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050706130200.

[58] Z. Zabinsky, "Random search algorithms", in Jun. 2010, ISBN: 9780470400531. DOI: 10.1002/9780470400531.eorms0704.

[59] J. Kennedy and R. Eberhart, “Particle swarm optimization”, in *Proceedings of ICNN’95 - International Conference on Neural Networks*, vol. 4, 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[60] D. Wang, D. Tan, and L. Liu, “Particle swarm optimization algorithm: An overview”, *Soft Computing*, vol. 22, pp. 387–408, 2018. DOI: 10.1007/s00500-016-2474-6. [Online]. Available: https://doi.org/10.1007/s00500-016-2474-6.

[61] J. Mockus, V. Tiesis, and A. Zilinskas, “The application of bayesian methods for seeking the extremum”, in Sep. 2014, vol. 2, pp. 117–129, ISBN: 0-444-85171-2.

[62] E. Brochu, V. M. Cora, and N. de Freitas, *A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning*, 2010. arXiv: 1012.2599.

[63] J. Snoek, H. Larochelle, and R. P. Adams, *Practical bayesian optimization of machine learning algorithms*, 2012. arXiv: 1206.2944.

[64] scikit-optimize Development Team, *scikit-optimize (skopt)*, 2021. [Online]. Available: https://github.com/scikit-optimize/scikit-optimize.