TÉCNICO LISBOA

Learning to Grasp Objects in Virtual Environments through Imitation

Alexandre Vieira Filipe

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Dr. Plinio Moreno López
Prof. Alexandre José Malheiro Bernardino

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Dr. Plinio Moreno López
Member of the Committee: Dr. Athanasios Vourvopoulos

January 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Abstract

In spite of the huge progress seen in the field of robotics, robot manipulation still stands as a great challenge for the engineering community.

In this work we will present our 2-phase method to teach grasping tasks, using human demonstrations as training. To record the demonstrations and recreate the tasks a Virtual Environment (VE) was created, where both the human demonstrator, using a glove with sensors, or the method itself can control a virtual hand.

After trained, the method proved capable of performing the demonstrated tasks with some variance in the object starting conditions. Also, taking advantage of our developed setup, tests were made to take conclusions on the presence and absence of haptic feedback, where we verified that its inclusion lead to better demonstrations.

Finally, we will publicly supply our VE, along with all its systems for recording and reproducing tasks and a collection of pre-recorded demonstrations for any researcher to test their own methods and approaches to robot grasping.

Keywords

Robot manipulation, imitation learning, virtual environment, neural network.

Resumo

Mesmo com o atual progresso da área de robótica, a manipulação robótica ainda é um grande desafio para a comunidade, ao ponto de mais comumente se usar uma garra ou mão simplificada para não ter que lidar com a flexiblidade inata da mão humana.

Neste trabalho propomos o uso de demonstrações humanas como guia para a manipulação robótica, um processo chamado aprendizagem por imitação. Para gravar as demonstrações, e de modo a evitar as obstruções inerentes de gravação vídeo, será usado um ambiente virtual. Para interagir com o ambiente virtual será utilizada uma luva com sensores de posição e placas vibratórias na ponta de cada dedo, para simular o contacto com os objetos virtuais.

Após gravadas as demonstrações de uma tarefa, estas serão treinadas numa rede neuronal, que será depois usada para tentar conduzir a mão virtual a realizar a tarefa a partir de um ponto no espaço fornecido, sendo testada no processo a qualidade do nosso método. Também será testado se o uso das placas vibratórias realmente afetou a qualidade das demonstrações.

Também para quem deseje experimentar os seus próprios métodos de aprendizagem de imitação o ambiente virtual e o conjunto de demonstrações gravadas será fornecido gratuitamente.

Palavras Chave

Manipulação robótica, aprendizagem por imitação, ambiente virtual, rede neuronal

Contents

List of Figures

List of Tables

Acronyms

VE	Virtual Environment
NN	Neural Network
AI	Artificial Intelligence

Chapter 1

Introduction

Alongside with the advancements seen in the modern era of technology we see an investment in the field of robotics, where the scientific community aims to build robots capable of assisting the common man on his daily life, from work to recreational activities alike. For such a goal to be possible robots must be capable of doing a wide range of tasks, i.e. to have a performance on par with its human counterpart, or, ideally, surpassing it and its limitations.

In particular we some investment in trying to replicate our finesse using tools and grasping objects, using a claw/gripper or a functional replica of a human hand.

In this thesis we propose to develop a Virtual Environment (VE) where a human subject can use its own real hands to control virtual hands in grasping virtual objects. This will allow to capture the trajectories and the contact points with the objects in a much more precise and robust way than current practices that use cameras, motion capture and instrumented objects. The acquired information will be used to learn grasping skills from humans and transfer these skills to robots with similar kinematics, i.e. humanoid hands robots.

1.1 Motivation

Even with all the advancements recently done on the robotics field, robot manipulation still stands as a huge challenge. Such is due to how many degrees of freedom are involved, to different objects requiring different grasping patterns and the same object requiring different grasping patterns according to the goal of the task. All this makes, even for the simpler tasks, the act of manipulating objects a serious challenge due to the myriad number of possibilities the robot needs to consider for the grasp.

A good approach to this problem is to take the human natural way of grasping objects as a basis for the robot to follow. Human demonstrations can guide the robot to the correct grasping pattern, and make the movement more fluid and natural on the process. Also note that, although the object manipulation of objects is a core activity of our everyday lives that we do instinctively, the study of our subconscious patterns is one that's still ongoing and a very useful one to apply to our particular problem.

With all the above in consideration, we question: how to study and record the human mannerisms and patterns if the hand is such a complicated organ? In this work we suggest the use of a VE that can be interacted with a glove capable of realistic representation of the human hand. This avoids the

common troubles of video recording (figure 1.1), as noise will be minimal and obstructions nonexistent, and also provides a better medium for collecting data, as it's presented in a 3-dimensional environment.

Figure 1.1: The mug is obstructing the position of the 4 non-opposable fingers and its contact points

In general, our motivation is to improve the state of the art of robot manipulation by studying human grasping skills and transferring this knowledge to a robot or virtual hand.

1.2 Objectives

In a broad sense, our objective with this work is to develop a trajectory generation algorithm that successfully executes a grasping action. The algorithm, to do so, will segment the action in reaching and manipulation phases (or, in other words, before and after the object is grasped) to ensure the action does not advance with the object poorly grasped. Both segments are learned by imitation of the action demonstrated by a human. In a more specific sense our work can be divided into the following specific objectives:

- Development of a virtual environment, which will be accessed through a data glove with haptic feedback. The VE must be complete with objects to interact with and means to export the training data there collected, these being the position of the hand, the pose of its fingers and the position of the desired object, along the demonstration. The VE would also allow for the reproduction of the tasks created by the algorithm, to visually assess the quality of the method
- Development or adaptation of a learning algorithm capable of reproducing grasps based on human demonstrations.
- Verify if the inclusion of haptic feedback leads to better grasps

- Provide a public database of demonstrations for the researcher community that does not have the hardware needed to record grasps, or to simply ease the process to test training methods and algorithms.

1.3 Contributions

This work will add to the state of the art with the introduction of our trajectory generation algorithm with a 2-phases system, along with some other systems and network architectures that were developed as part of the iterative process to reach our final model, and conclusions to their effectiveness.

Besides this we have have made a system to incorporate haptic feedback in the demonstrations, and tested if its inclusion lead to better reproductions of tasks.

Finally we have developed a simulator in Unity that includes the used VE and its objects, a system to perform and replicate tasks, with a glove with sensors or with a given input, to detect grasps, to detect force closure and finally with a collection of recorded demonstrations (more than 3000 demonstrations), publicly available for future use to any researcher intending to test their own grasping method or grasping related theories, found in https://github.com/alexamor/Thesis.

1.4 Organization of the Document

This document will start by analysing the state of the art of robot manipulation on Chapter 2, including the methods and trends in current practice. After this a more particular analysis is made to selected pieces of investigation that we believe of special importance to the work to be developed.

Following, in Chapter 3, with the knowledge of the State of the Art, we present our proposal. The detailed explanation of how he realized our proposal and our method, along with the sofware and tools used, can be found in Chapter 4 - Methodology.

Afterwards we have in Chapter 5 - Results, the tests we performed and some intermediate conclusions taken from them. We finalize our work with Chapter 6 - Conclusion, where we present a summary of the conclusions taken from the experiments, and suggest next steps.

Chapter 2

Background

Making use of tools and all the plethora of activities achievable with our hands is one of the most impactful skills of mankind [6]. Thus, it is natural that engineers and researchers have a special interest in making robots capable of mimicking our finesse manipulating objects. The development of techniques to teach robots to efficiently manipulate objects has been a long and continuous process, broad on the different ways the problem has been approached.

In this chapter we aim to explore the current state of robot grasping, in particular the imitation learning approach which uses human motor control as a basis for robot object manipulation, reviewing the state of the art methods and how we can use them as a starting point for our research.

Along the chapter we will describe and analyze the current approaches and methods used in the state of the art of robot grasping, with more detail to the ones more related to our work. In particular, in 2.1 we will analyze the known approaches to teach grasping to robots, in 2.2 the approaches to trajectory planning of the hand to the object, in 2.3 the possibilities for VEs, in 2.4 the currently employed algorithms to run imitation learning methods, in 2.5 the analysis and comparison of our work with some closely related ones and finally in 2.6 we give a brief overview of how our work fits with all the above.

2.1 Approaches to grasping

The literature presents a wide diversity of approaches for robot grasping, categorized by the knowledge the robot has of the object it is trying to grasp.

A survey [7] shows the present-day approaches can be divided in terms of the object by:

- ***Known Object*** - The object to be grasped already exists on a database that the robot can access and retrieve a grasp, usually built offline.
- ***Familiar Object*** - On detection, if the object at hand is not known it looks for similar objects, being that the familiarity can be defined by low-level features, such as color, texture or shape, or in a higher level, looking for objects that serve the same purpose. After finding said familiar object it then tries to transfer the grasp experience.
- ***Unknown Object*** - The robot does not have prior knowledge of the object and needs to develop the grasp model on real time based on its geometry and perceived features.

In terms of how the grasp on itself is constructed, the methods applied are more diverse, ranging from physics-based to human-based, from grasp patterns preprogrammed to developed in real time. In general they fall on the one of the following methodologies:

- ***3-D Mesh Models and Contact-Level Grasping*** - This methodology tries to compute grasp trajectories using physics simulators and the 3-dimensional model of the object and hand. The processing can be done previously or in real-time, allowing to grab all kind of objects, even unknown ones, provided that an approximate geometric description of the object can be obtained reliably.

  Considering the vast number of plausible grasps a heuristic resorting to some kind of metric helps guide the system to compute better quality grasps.

  This line of research is followed by Diankov [8], that proposes to synthesise the grasp based on the object bounding box (see Figure 2.1) and surface normals, and Weisz and Allen [9], which improved the method by classifying the best grasp candidates by perturbing the object in 3 degrees of freedom and how well the grasp handled it. Both showed valid results of taking this approach to object manipulation.

  On the down-side this methodology is quite intense and slow, even by today's standards of computing power, to be efficiently realized in real time and, as Balasubramaniam demonstrated [10], these underperformed in comparison to grasps assisted by humans.

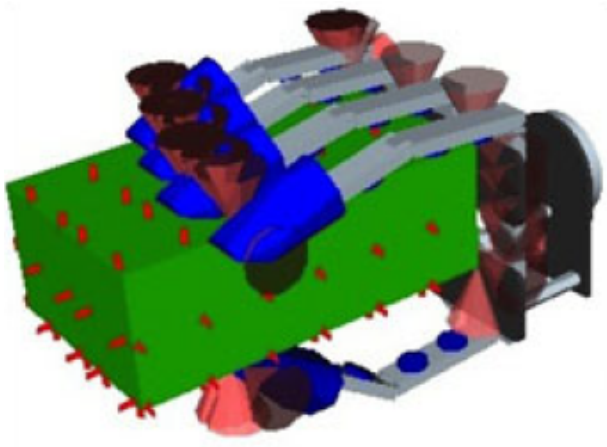


Figure 2.1: Grasp based on normal vectors and bounding box [4]

- ***Learning From Humans*** - Being this the approach taken by this work, it functions by generating grasps by observation or with the aid of humans, performing the same or similar grasps. The more common method of learning from humans is *imitation learning*, the method of using demonstrations of tasks made by humans and using them as a basis, trying to replicate/imitate the task. This can be done by having the the human performing the task himself, or by having the human moving

and pushing the robot members [11] to perform the task. Alternatively we can learn from humans by having human demonstrations to give empirical information about of all the present grasp hypotheses which we use more [12], or by using knowledge of motor capabilities and neuroscience to learn aspects of our grasp methods, as learning how we have limits to the degrees of freedom of our joints [13].

Imitation learning in particular can be applied to supervised learning methods, where an Artificial Intelligence (AI) algorithm uses human demonstrations as training data, obtained by video [14] or positional sensors [15], for example, to conduct a robot to grasp an object in the most similar away as the examples given, or used to accelerate unsupervised methods [2] by using the demonstrations to help initiate the method.

Being imitation learning the adopted approach, and being it broad in its methods, its important to deconstruct it to better understand what it encompasses. Briefly, the subtypes of imitation learning [16] are:

- *Behavioural Cloning* - The demonstration is segmented in iterations and each iteration its treated as independent and identically distributed, having all the same weight. The algorithm will use these as a guide to what to do next on the current iteration. It is a simple and efficient method, but can fail if the task needs long-term planning or if the task has a risk of going off the trained space. This is our work adopted approach.
- *Direct Policy Learning* - Somewhat similar to Behavioural Cloning, Direct Policy Learning is an iterative approach where the AI performs an attempt at the task and the human corrects its or re-demonstrates, doing this until the policy is considered good. Although its a more robust method that Behavioural Cloning, it needs a person throughout all the process.
- *Inverse Reinforcement Learning* - In this approach the AI instead of trying to learn the policy the demonstrator is following it tries to find the reward function, or in simpler terms, the value of each action, and try to predict from the reward function what the policy is, iteratively. It has 2 types, model-given and model-free. In model-given the algorithm has access to the demonstrator's policy and at each iteration compares the deduced policy with it. Works only if the reward function is linear. The model-free approach deals with a complex reward function and does not have the demonstrator's policy, using a simulator to compare the deduced policy with what is being demonstrated. In general Inverse Reinforcement Learning is a more complex approach suitable to more complex tasks.

In terms of objects we can have the object modeled, and through a database find the grasps for this object, or one familiar to it or, alternatively, have the system informed of which which demonstrations the system will use, not needing the model of the object.

Due to the complexity of the human hand some techniques have been suggested to ease the computation load in this approach, from GPU acceleration [17] to reducing the degrees of freedom of the hand [18] according to the synergies between the non-opposable fingers.

- ***Learning Through Trial and Error*** - An unsupervised learning method that, by retrying the grasp several times, hopes to learn to grasp the object. Usually applied by means of reinforcement leaning and reward functions, it starts with no knowledge on how to grasp the presented object, having the sole information of the value of the reward function after some action. With each subsequent attempt the algorithm adjust its parameters, making the next attempts more similar to the previous best ones and avoiding the patterns used on the failed ones.

  This approach also has its flaws, making robots performing unnatural movements and, above all, having an extremely slow training, ranging from tens to hundreds of hours needed to train a single grasp [2].

2.2 Trajectory planning

Having the currently employed methods of grasping in mind, we now part on a more particular problem, the planning and definition of the trajectory of the hand to the object.

All the currently used methods share a thing in common, they discretize the trajectory in actions, that is to say, in key positions, reducing the trajectory to a sequence of actions and the links between them. Employing this format, the methods differentiate themselves in two groups, methods that need to preplan the whole trajectory and methods capable of performing the trajectory in real-time, calculating the next action based on its current one.

From the preplanned methods, or motion planning methods, we highlight the Rapid Random Tree [19] that creates a rapidly growing tree of actions with its root on the starting position. The nodes/actions are traveled randomly, guided by a heuristic function, terminating when the final position is found. Although these methods make extra calculations, as actions not needed for the final trajectory are also calculated, these are more focused on the avoidance of obstacles and the Rapid Random Tree in particular still presents itself as a relatively fast method.

From the methods that only calculate the trajectory in real-time, or with motion controllers, we can highlight the more particular model predictive control methods, as PLATO [20], that train a policy that will define the decision making of the robot. This method showed promising results, having the drawback of needing pretaining.

In our work we will use a real-time motion controller, but we will use a more common approach to trajectory planning, where we will treat the full task a sequence of stages/iterations, that are basically captures of the demonstration/reproduction at a defined time frame, and have at the current iteration the

method outputting the following iteration using an algorithm that has been trained with demonstrations, in our case, a recurrent Neural Network (NN). The information the algorithm will use at the current iteration can be simply the information present in the current iteration or, alternatively, the algorithm can use also the information present on the previous iterations, to give a temporal perspective to the demonstration/reproduction (more on this later).

2.3 Virtual Environments

The currently used VEs vary in scale based on the purpose they are meant to achieve, being the only constant the need of a reliable collision detection and physics simulation. For that a wide range of software has been developed, from dedicated physics simulators in conjunction with 3D rendering software (as seen in [2]) to game engines such as Unity3D or Unreal Engine that include both.

The Virtual Environments can be fully created from scratch, using the engine object primitives, such as cubes or spheres, or made with 3D modelling software, or by having its objects exported from public available premade packages (found in, for example, the Unity Asset Store) that include more elaborated objects, or even exporting the full VE, such as Xiaofeng Gao Virtual Kitchen [21], which is a free VE designed specifically for robot learning, that includes a variety of objects to perform common manipulation tasks.

To host the VE some engine or software must be in place to do so. There is quite an array of available engines, free or otherwise, that have already been used to perform grasping demonstrations. From these we can highlight the 3 most common.

- ***Mujoco HAPTIX*** - Created and developed by the Movement Control Laboratory of the University of Washington, Mujoco is a paid physics engine developed in C/C++, that includes their own simulator, HAPTIX, made mainly for robotics simulations. It excels in the realistic reproduction of real-life physics, capable of accurately replicate contact-rich interactions between objects. Even with its pay-wall it found itself as a well known name in the robotics sector. In grasping works in particular, in can be found in some [2], having them taking advantage of the already developed plugins for Virtual Reality equipment/tools, and physics engine that accurately replicates the contact forces present in the tasks.

- ***Unreal Engine 4*** - Being from one of the longest running Game Engines, Unreal Engine 4, or UE4 for simplicity, is a well-known, freely available software developed by Epic Games. To encourage new developers to use the engine it is made to be user friendly. To employ its physics it uses NVidia's PhysX physics engine. As a game engine the physics aren't made to be realistic to the detail, prioritizing performance. It has been used in some grasping works [22], taking advantage

of the relative easiness of developing 3D environments and, being Virtual Reality the current tech trend in videogames, it possesses the tools to easily integrate the peripherals.

- ***Unity*** - Unity is the currently most used game engine, in particular by small independent developers. Created by Unity Technologies, it was made to appeal to a broader audience of developers, being freely available and user friendly. Similar to Unreal Engine 4, it uses NVidia's PhysX engine, and again, prioritizing performance over being incredibly accurate. It has been applied in quite a number of grasping works [1] [3] [23], probably due to the same reasons as Unreal Engine 4. Also, being a very popular tool among new developers it's quite easy to find support online.

2.4 Algorithms

To enable an AI to understand demonstrations and try to replicate tasks based on them, an algorithm or system must be in place.

A common method is to use a NN, an algorithm based on training from examples. Of course NNs are a diverse method, and a number of different types of NNs have already been used to train manipulation tasks [1] [3] [24]. The simpler form is to use a linear NN that predicts, from a stage of the task, where the hand will go next, as seen in [1]. This carries the issues innate of non recurrent NNs, and to circumvent them the authors have as input of the NN not only the current stage/step, but also the previous 4, to place the stage in time.

Alternatively, some other works had success with recurrent NNs [3] [24], as these are prepared to deal with temporal tasks. In particular, the most commonly used type of recurrent NN used in this kind of works is the LSTM, short for Long Short Term Memory, as the use of memory cells makes it ideal to reconstruct temporal tasks.

What distinguishes recurrent NNs from their linear counterparts and makes them more suitable to temporal tasks is the presence of a loop 2.2, making the NN output dependent of the previous iterations. Due to this they have been a trend in the fields of speech, language, image captioning [5], among others, where the predictions need more that the information of the current item (as an example, to predict the next word in a sentence we need not only the current word, but also some previous words of the sentence, to make some sense of the sentence).

Although simple recurrent NNs can solve sequence/time dependent problems they only function well for short term memory, that is, for problems that the do not need to remember information for to long, as unused information tends to be deprecated due to their memory functions (the memory functions usually have a scaling function, for example, a tanh function, and if they do not have a system in place to decide which information is important to keep, old information might be reduced to almost null).

To solve this a subtype of recurrent NNs was introduced, the LSTMs (short for Long Short Term

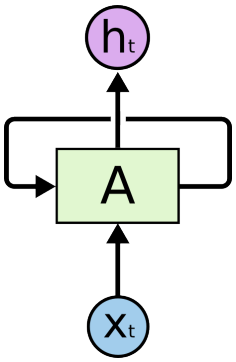


Figure 2.2: Diagram of a recurrent NN with X the input and h the output [5]

Memory), that has an added system to deal with memory (figure 2.3), allowing the LSTM to calculate and deduce which old information it will remember and forget, solving the long term memory issue. The main segment of the memory system is the cell state, which carries the relevant memory between iterations. To maintain the cell state, or memory, other layers were added. First the forget gate layer deduces of which of the previous cell state is important to memorize and which we can forget. It does so by calculating a number between 0 and 1 for each value of the cell state, being 0 to completely forget and 1 to completely keep, and multiplying the previous cell state by it. Next we have 2 layers to add new information to the cell state, the input gate layer which chooses which values to update, and a tanh layer that provides the new information scaled. After this memory management the LSTM proceeds to deduce, from the updated cell state and the current input, what the output will be, saving the output and cell state for the next iteration.

Another method work worthy of mention is the Markovian Decision Process, a method that, based on heuristics, from a current step has a number of unequal probability next steps. Demonstrations can be applied as training to give more weight to certain next steps. Although a more difficult to apply method, it has also shown to be successful [2].

2.5 Related Work

In this section we're going to discuss particular pieces of investigation that we believe to be of special importance to our work. These can be supportive, demonstrating the effectiveness of our proposed approach, or serve as contrast to our work, allowing us to show how we can improve the current state of robot grasping. In general, all the following works used imitation learning at its basis and some level of human motion capture, similar to our proposed work.

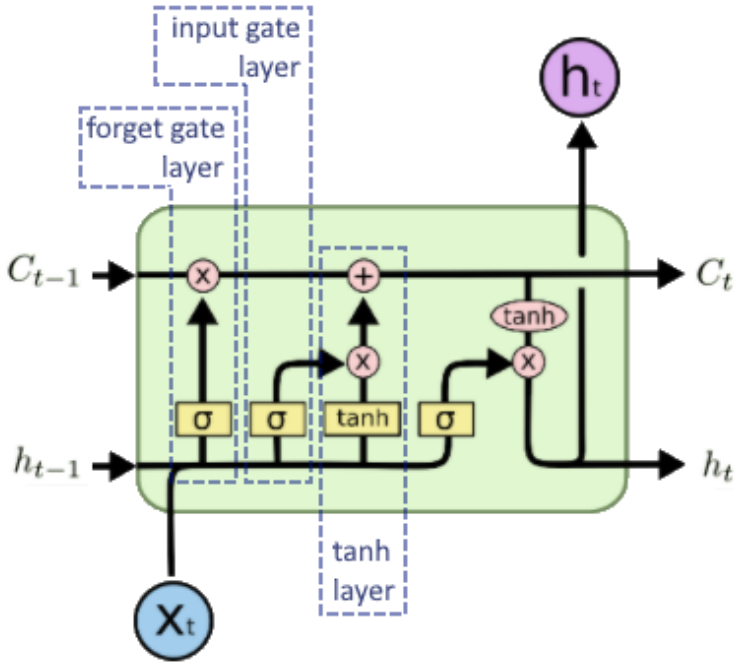


Figure 2.3: Diagram of an LSTM with X the input, h the output and C the cell state [5]

2.5.1 Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation [1]

Developed by Zhang and McCarthy, this project intends to train a robot to grasp different objects by imitation learning, being the human samples recorded by means of teleoperation (the human subject controlling the robot in real time remotely). The teleoperation is accomplished using a HTC Vive headset and controller, using as a visual interface a 3D mapping of the scenery captured by two cameras present on the robot head (see Figure 2.4). The 3D mapping was mostly needed to mimic the head movement, as the robot couldn't easily move its head and the difference between its movement and the user movement lead to nausea and motion sickness. The robot hand is a simple two finger claw, that closes and opens at the press of a controller button and moves according to the controller movement.

In this example the training data is processed by a neural network that receives the data of the 5 most recent steps, including in the data of each step a RGB image, a depth image, captured by the stereo camera placed on the head, and the position of the effectors on the hand. The neural network outputs the angular and linear velocity of the robot hand, as well as the boolean indicating the opening/closing of the claw, of the next step.

After training and testing the grasping method on a diverse number of objects and tasks, including the placement of fruit in a bowl, inserting a block on a shape sorting toy, aligning a hammer with a nail, mounting the wheels on a toy plane, grabbing cloth, etc, it was concluded that the method showed good

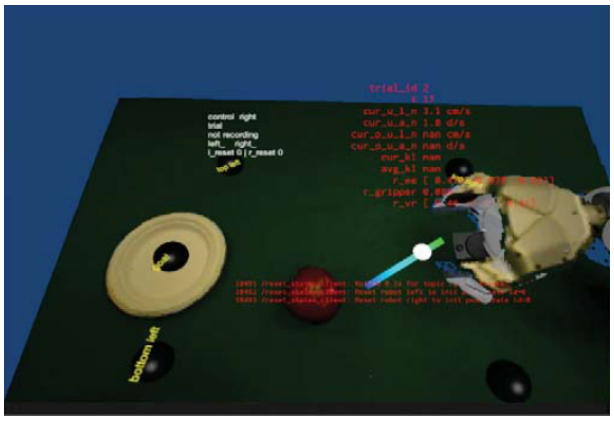


Figure 2.4: Image perceived by the user performing the teleoperation [1]

results, managing to perform even the harder tasks (the worst task, consisting of grasping and placing two toys in succession, displayed a success rate of 80%) in an average of 15 minutes worth of training.

Relating to our work this piece of investigation proves the effectiveness and efficiency of learning from demonstrations. Also, in terms of improvement, in our work we can perform imitation learning with a completely dexterous hand and, as the paper itself proposes, use haptic feedback for a more realistic grasping in the VE.

2.5.2 Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations [2]

In this work Rajeswaran and Kumar propose to train a virtual robot with reinforcement learning to perform elaborate manipulation tasks (moving a ball to a goal, moving a pen to a goal with a specified orientation, hammering a nail and opening a door closed by a handle, as shown in Figure 2.5) with a five fingered hand on a VE. The method can be bootstrapped by a human performing the same tasks on the same VE using a glove with positional sensors. The VE utilizes the MuJoCo [25] physics simulator and the user interacts with it via a HTC Vive headset and a CyberGlove III, a glove with 18 to 22 positional sensors. The work also aims to compare the performances of a system that uses human samples as its starting point to one that does not.

To perform the tasks, a reinforcement learning algorithm was implemented. The algorithm used is a Markov Decision Tree, a decision tree that at each action presents a number of possible following actions. A reward function classifies each of the possible actions, allowing the chosen following action to be picked randomly, having the actions with a better classification a bigger probability to be chosen. Although this algorithm could eventually achieve the proposed tasks without any human demonstrations, these are added as pre-training and to speed up the learning, utilizing the randomly initialized method as a baseline for the quantification of the advantages/disadvantages of using human demonstrations as

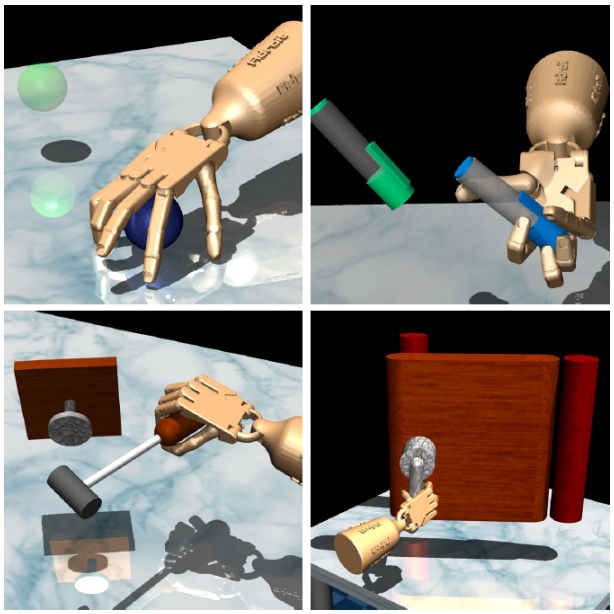


Figure 2.5: Examples of tasks trained and performed in [2]

examples.

After performing the tests the results were evident, as the use of imitation learning led to a speedup of the training process, up to 30 times faster, to achieve the same results as its unsupervised counterpart. It was also noted that the presence of human interactions led to more natural and fluid movements of the hand and fingers.

Now with our work in consideration, we can utilize this piece of investigation as proof of the quality human demonstrations add. Again, this work mentions the usage of haptic feedback could lead to better performance, which we mean to test in our work.

2.5.3 Learning real manipulation tasks from virtual demonstrations using LSTM [3]

With robots capable of helping disabled and elderly people in mind, Rahmatizadeh proposes the training of the same robots to be performed by its users. For such training to be plausible for the target audience the author emphasizes that the process needs to be easily performed and relatively quick.

This is achieved by using a Xbox controller or keyboard and mouse as the robot claw controller, ensuring that the training data can be obtained, even if the user struggles with is its motor capabilities. The data is again collected on a virtual environment, hosted in Unity in this case, and through a recurrent neural network (An LSTM, to be precise) applied to a real-life robot (An example of a task can be seen in Figure 2.6). A recurrent neural network differs from a linear neural network as it saves data along the processing, being better suited for sequential tasks.

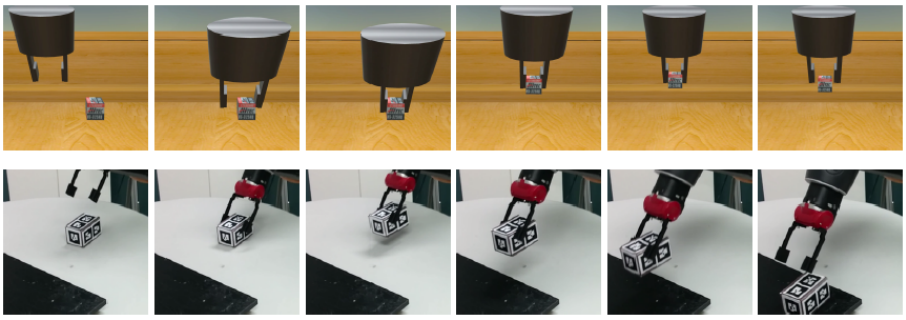


Figure 2.6: Example of a task performed in [3] in the VE and real life

Using a feedback controller for reference and other types of neural networks the author came to the conclusion that the imitation learning processes clearly outperformed the feedback controller, and in particular that their recurrent neural network performed better than the further tested neural networks.

We can utilize this work as proof that, even with simple inputs inserted with a console controller, imitation learning still shows favorable results. We can easily improve on this by using a complete hand instead of a claw/gripper.

2.6 Overview

With all the previously analyzed works in consideration, we can see that there are gaps to fill and improvements to be made on the state of the art of robot manipulation, which we aim to do with with this work. In the table 2.1 we can observe an overview of the presence or absence of some of the aspects previously mentioned.

Work	Imitation Learning	Use of Humanoid Hand	Haptic Feedback	Virtual Environment
Zhang and McCarty	✓			✓
Rajeswaran and Kumar	✓	✓		✓
Rahmatizadeh	✓			✓
Our proposed work	✓	✓	✓	✓

Table 2.1: Comparation between analyzed pieces of work and the work to be developed in this investigation

In general, we aim to improve the state of the art by introducing our two-stage model for grasping, that uses a humanoid hand, by including and testing the presence of haptic feedback in the demonstrations, and by providing an array of demonstrations with a simple VE to apply them, publicly available for any researcher to try and test their own grasping methods.

Chapter 3

Proposal

With a clear perspective on our objectives and the state of the art of the area of work, we now share our proposal for this work.

The general idea is to have a human, using a glove with sensors, to perform tasks in a VE. These demonstrations will be saved and used as training for the LSTMs. The LSTMs, after trained, will be given a starting position and output recurrently the next position, creating a sequence of positions that aims to recreate the demonstrated task. While the LSTMs are outputing the positions the VE will have the virtual hand following them, to allow to visually assess what the LSTMs are outputing (figure 3.1).

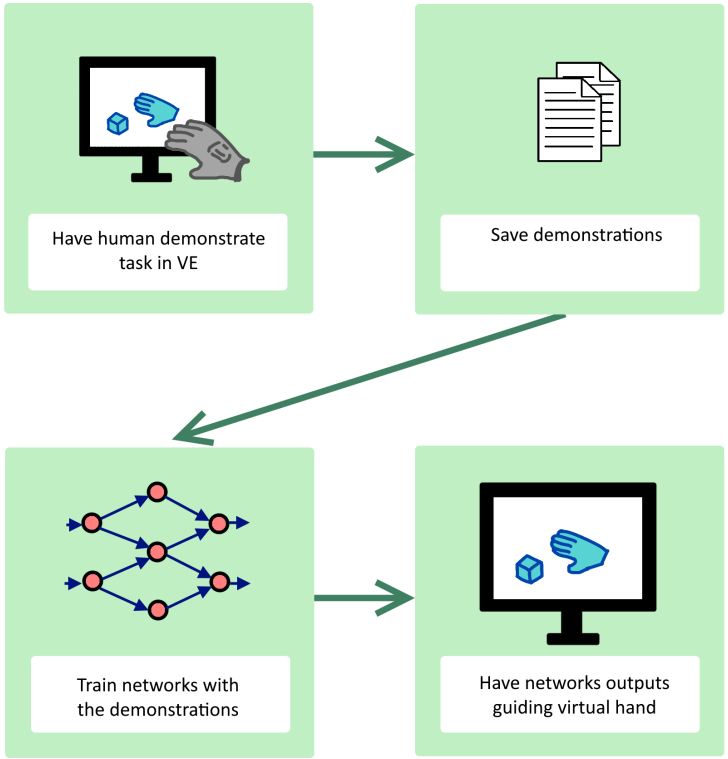


Figure 3.1: A simple diagram of the proposed process

Some particulars of the aspects involved can be found in the subsections below.

3.1 VE

The VE will consist of a simple table, with objects to interact with. Some objects will have other objects they can be interacted with while grasped, for example, having a square slot to fit a gasped cube in. To interact with the VE a glove with sensors will be used, having a virtual hand mimicking the gloved hand movement (figure 3.2). Also, to take advantage of the ease of adaptation of the human mind, besides a common human hand the VE will also include replicas of real robot hands to control.

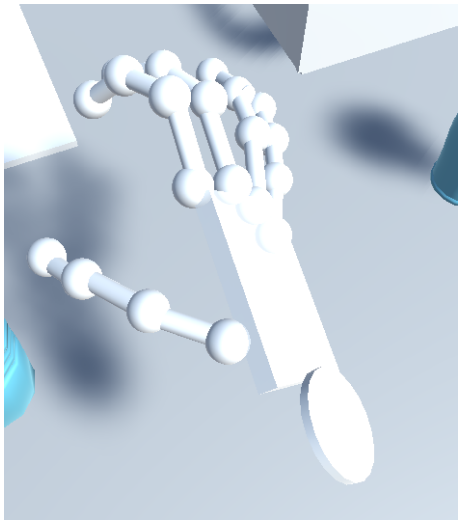


Figure 3.2: The used virtual hand

3.2 Data

The data to be captured in the demonstrations will be a discrete sequence of iterations, captured at a defined rate. Each iteration will include the information of the hand and object to interact positions and rotations, as well as the information of the position of each finger joint. The rotation will be represented by a quaternion, the position by its 3-dimensional value. Both the recorded rotation and position are the relative rotation and position of the hand to the object. The finger joints bending are represented by an integer that represents how much the joint is bent (figure 3.3). We will use the relative positions and rotations to allow some dynamism and shorten the data size.

Also to note, to insert some variation, the objects spawn at a random plausible position at the start of each demonstration and reproduction. Some tests were also performed were the objects had some variation in its rotation.

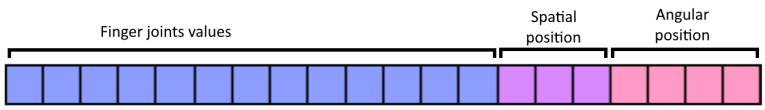


Figure 3.3: Structure of the array that represents each iteration

Although the VE includes more, for the final test 6 tasks, involving 6 graspable objects (a hammer, a can, a bottle, a mug, a knife and a parallelepiped) and some objects to interact with (boxes and bases to drop objects, for example) (figure 4.2), were recorded. In total, for each trained task 200 demonstrations were used, 180 for actual training and 20 as plausible starting positions for testing.

3.3 Algorithm

The algorithm we are going to use is an LSTM, a type of recurrent NN, using the mean squared error function to measure loss and having the initial values set randomly. To be precise we are going to use 2 LSTMs, one for the reaching phase of the task (reaching is considered everything until the object is considered grasped, being the object considered grasped when the thumb and at least one other finger is touching the object) and the other for the manipulation phase of the task (manipulation is considered everything after the object is considered grasped, including the finger adjustment, hand movement, interaction with a second object, dropping the object, etc, until the end of the task). A detailed explanation of what lead to the segmentation of the task can be found in the following chapter.

Both the LSTMs will be trained with the recorded demonstrations, each with their respective phase.

After the training they will be prepared to, from a given starting point, try to predict the next iteration and, doing so recurrently, create a sequence of iterations that will, hopefully, create a reproduction of the task.

While the method is outputting the iterations these are being followed by the virtual hand in the VE, to give a visual perspective of what the method is predicting.

In summary, the process starts by giving a starting point to the reaching LSTM, from which it will predict the following iterations. The moment the guided virtual hand considers the object grasped it sends a flag to the LSTMs program to inform that from that point forward the manipulation LSTM will predict the following iterations.

3.4 Experiments

To test the quality of our method we will give a set of random starting positions for each task for the method try to reproduce the task from. A success rate (being a success every time the method managed

to perform the task) will be recorded for each task, to indicate how successfully our method performs.

Other tests, for example, to test how well our method deals with variance, and the impact of the inclusion of haptic feedback, will also be made.

Our project had its tests run in the VE, but, as we will suggest, it could be applied to a real life robot. To be able to do so some systems would have to be in place, including a system to detect the present object and its pose and other to convert the object and hand pose to the scale the VE works with. With this systems in place, the robot could communicate with the algorithm, sending its current state and having the algorithm responding with the next one iteratively, and doing so create a movement in real time that will, hopefully, allow the robot to perform the task.

Chapter 4

Methodology

In this chapter we will present the details of the construction of our work. We will start by first presenting the used tools and software. After this we will explain the inner workings and build decisions for both the VE aspect and the training aspect of our work.

4.1 Equipment

To be able to interact with the VE some peripherals will be needed, in particular a glove with sensors so the user can interact with the environment. It will also be needed software to host the VE and run the NN.

4.1.1 Glove

The used glove was the VMG35-Haptic (4.1(a)), made by Virtual Motion Labs.

The glove possesses 3 sensors in each finger, two to measure the bending of the finger (one in the knuckle that joins the finger and the hand and another on the joint between the first 2 phalanges) and a third to measure the lateral movement of the finger (with the exception of the middle finger that does not possess the lateral movement sensor). There is also a sensor positioned on the palm of the hand that measures how open the palm is (as represented in 4.1(b)).

To position the glove in the 3-dimensional space the glove also possesses two gyroscopes, one on the hand and another on the wrist, that measure the 3 angular degrees of freedom of each. These are used to place the virtual hand on its correct angular position and, using the wrist rotation to assume where the arm is, allow some lateral and vertical movement.

Finally, to mimic the touch of an object, it uses 5 vibrating plates, one on the tip pf each finger.

4.1.2 Unity

The VE is hosted in Unity, a well known game engine. Although it is not a tool made for the purpose of robot training, it has shown to be a reliable tool to do so, due to the incorporated physics engine, the ease to create 3D environments and react to its objects' collisions/interactions, and a big community, which provides a wide array of free assets, including modeled 3D objects and premade scripts/tutorials.

(a) VMG35-Haptic Glove

(b) Sensors on the glove (light blue - sensors, dark blue - sensor reading directions)

Figure 4.1

Although the program was originally programmed in the 2018 version, it was later updated to use the 2019 version, simply to facilitate the video capture of the recordings (as the latter included premade tools for this).

4.1.3 Tensorflow

Tensorflow is a free Python based software to create and host NNs, and being one of the most well regarded tool for this kind of work, it was the one used in this work.

We used the latest Tensorflow version at the time, which was 2.1.0.

4.2 Implementation

Having been presented the used tools, we will now delve in the particulars of the construction of our work, both in terms of the VE and the method used to train. To note that, although some portions of the build were made with the used tools in mind, we believe the same approaches and methods can be done using similar tools or software.

4.2.1 Virtual Environment

To perform the demonstrations, and observe the reproductions, a simple VE was created. To avoid filling the scene with distractions, the scene wasn´t made to simulate a real place like a kitchen or a workshop,

consisting simply of a long table with the objects needed for the tasks placed along it (Figure 4.2). The avatar, which only had the hand visible, being the hand itself slightly translucent to permit some vision through it, had the camera placed on its head at eye level, slightly tilted down, as to look at the table.

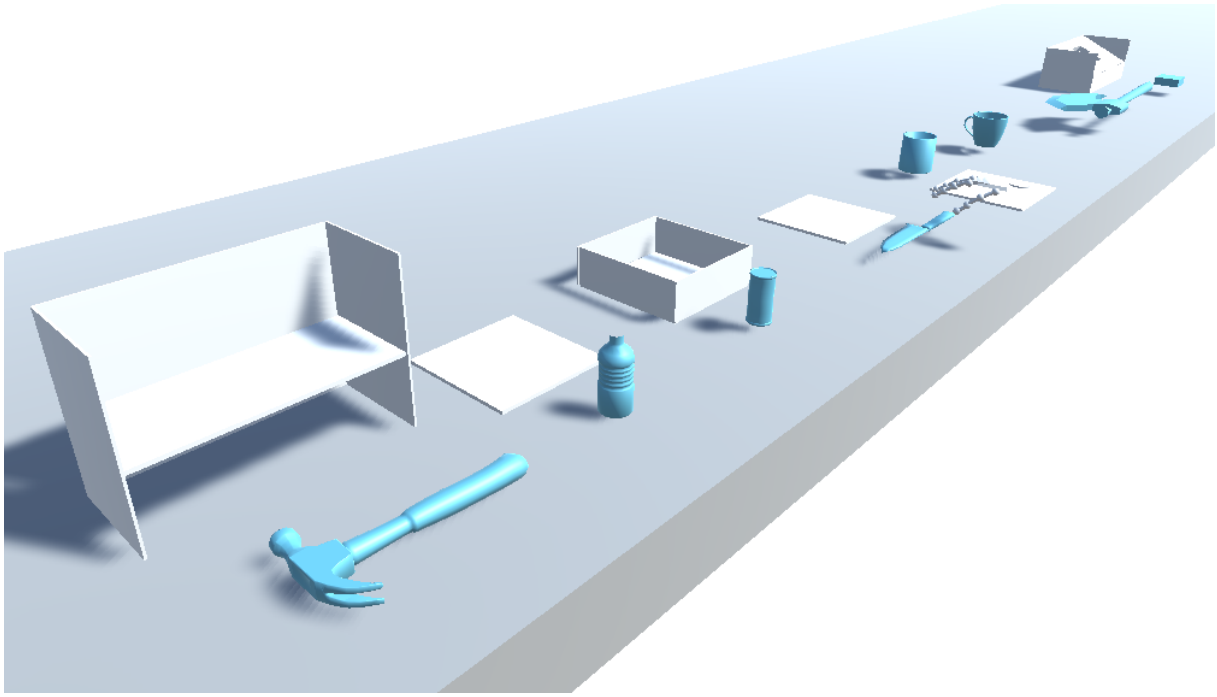


Figure 4.2: The VE table and objects

The virtual hand is controlled by a sensor glove, capable of capturing the hand and finger movements. Unfortunately the glove we used didn't have a sensor for measuring bigger lateral movements, only 2 gyroscopes to record the angular position. Some small movement could be deduced from the values of the gyroscopes, but broad movements were difficult and unreliable. Such problem was solved with a simple script that enabled the movement of the avatar with the WASD+QE keys to move it in all 3 directions, moving the hand with it, enabling the bigger translations of the hand. The angular sensors of the glove on the hand and wrist, as mentioned before, could be used to assert some lateral movement of the hand, leading to a more precise auxiliary to the keyboard imposed movement. A example of how the VE was controlled in real life can be seen in Figure 4.3.

Initially the work was meant to include a Virtual Reality Headset, but due to unforeseen circumstances the final version of the work did not include one. Nonetheless, some previous works didn't use a headset and lead to favorable results, some of them mentioned on the State of the Art, and, as such, we believe that this setback doesn't fault the quality of our work.

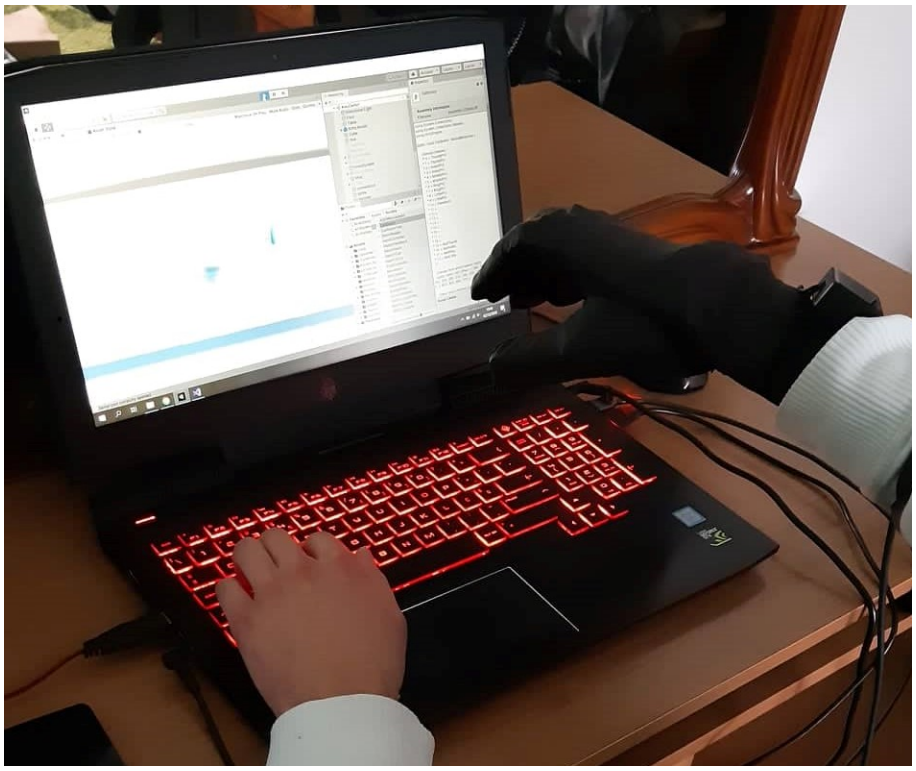


Figure 4.3: Example of the VE control

4.2.1.A Glove capture

Before starting on how the glove is recorded, it's important to understand how the glove is captured. The glove, when connected, routinely sends data packets to the computer about the values read in its sensors. The one we used in particular included in the data packets a integer for each of the finger sensors and some floats to represent the angular position read on the gyroscopes. The sensor integers are proportional to the corresponding joint bending, so, as to realistically capture the position, a calibration function, recorded outside of the demonstration, the values of each sensor for a closed fist and an extended hand, representing the minimal and max value of each finger bend. These values were then used to convert the real-time read value to the correspondent joint angle in degrees. These were then applied to the object that corresponded to that joint, rotating to the given value. Some data packets were also sent to the glove to activate the vibrating plate on the tip of each finger.

It's important to note that gloves of similar function most likely work the same way, and the method used can be performed with any other glove, but to keep the explanation simple we might include details of our used glove architecture (as seen previously), from which the parallel work of other gloves can be deduced by comparison.

Also to note that the lateral movement of the fingers was observed to be minimal, and very rarely independent (all the fingers separated and joined simultaneously). To take advantage of this, the lateral

movement was applied as the mean of the values read from all non opposable fingers, reducing the data size for the NN.

4.2.1.B Object interaction

Having the glove fully captured it was next needed to enable objects to be grabbed. This operation took some iterations, from which only the most important stages will be presented and justified.

Initially it was pondered to implement a system to mimic the realistic grasp, performed by calculating what real-life forces actuated,including the pushing forces and also the contact forces on the point of contact, spread out through an area to represent the area of contact, and then applying said forces to the object on the point they appeared. Such forces needed to be calculated because, even though our VE engine includes a Physics engine capable of imposing such forces, it does not automatically process contact forces (we used Unity, but other engines, Unreal Engine for example, possess the same downside), as the Physics engine was made with performance in mind, not realism (nonetheless, for the relative small scale we're working on such feat of realism is possible, just not planned for). Alternatively it's possible to rely on the VE engine colliders interactions, but pinching objects is usually buggy/jittery. Such implementation would be based on the work of Markus Holl [26].

Unfortunately, after several approaches to this method, we weren't able to achieve this natural mode of grasp. Nonetheless, keeping the same objective in mind we returned to the basic of doing a technique commonly used in this field, that is to create a condition that indicates the object is grasped, and, when such condition is verified, the object is fixed to the hand, this way avoiding having to deal with the forces calculations.

After some deliberation and testing the final and current version checks if the thumb and other finger is touching the object. If such is the case the object is considered grabbed and the object is fixed to the hand, and the corresponding touching fingers blocked on their position. The following fingers to touch the object also get their position frozen the moment they touch the object, allowing the fingers to construct the grasp by naturally landing on the object.

Finally, to allow the object to be dropped, the sensor values of the fingers when they get frozen is recorded and when the cumulative value of the frozen finger sensors surpasses a threshold (attained by opening the hand), the object is dropped and the fingers freed. A timeline of a demonstrated task can be seen in Figure 4.4, along with the number of fingers touching. While a finger is touching its haptic feedback plaque is vibrating.

4.2.1.C Demonstration Recording

When a demonstration starts the scene is set and, to guarantee diversity on the demonstrations, the object to interact with spawns at a random place inside a plausible area. We consider a plausible area

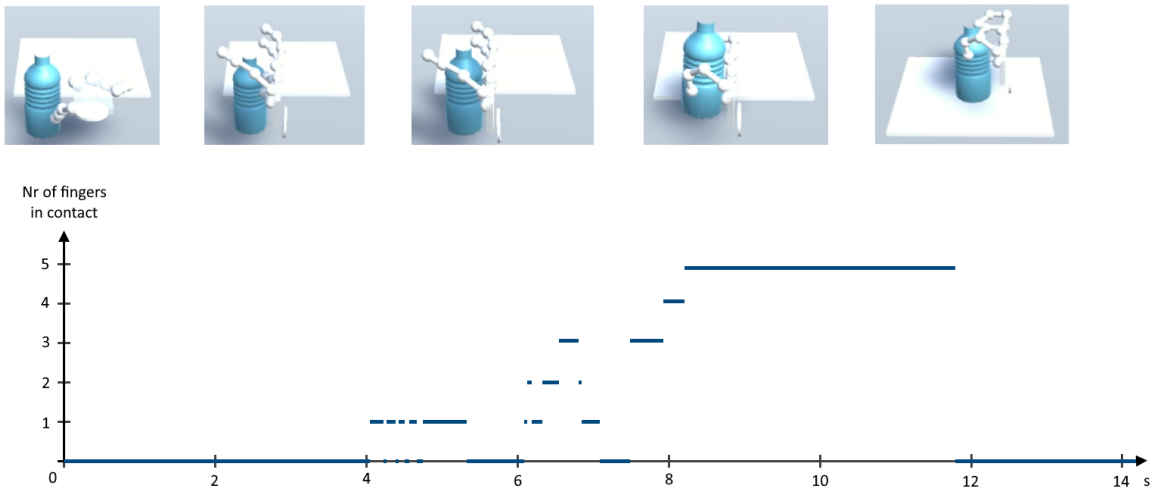


Figure 4.4: Timeline of an example of a demonstration, showing the ongoing task and the current number of fingers touching the object

one that inserts some variance but doesn't spawn the object far enough that the task to be made is so different the training method would have a hard time considering it the same task. The form and size chosen for this area is a square with a side of about 25cm (this value is a approximation obtained trough comparison of the size of the virtual objects and examples of their real life counterparts sizes). To better perceive this area Figure 4.5 has a visual representation of it.

Now referring to the recording itself, a flag indicates if the following demonstration is to be recorded. If so, a new text file is opened and the values of the demonstration will be written on the file, being the data written the current values observed in the demonstration. The current values are written at a set interval of time (in our case at every 0,2 seconds). The data recorded consists of the scaled values indicating the bending of each sensor and the relative spatial and angular position between object and hand (as only these 2 objects interacted this passes as a better way to save their individual positions, reducing the NN entries by 7). In a more mathematical manner, the spacial position saved in the demonstrations is

$$(x, y, z)_{saved} = (x, y, z)_{object} - (x, y, z)_{hand}, \tag{4.1}$$

with $x, yandz$ the 3-dimensional spatial coordinates of the corresponding objects.

To tackle the problem that is relative angles quaternions were used, being that the relative position was obtained by multiplying the quaternions of the component (which in quaternions translates to applying an angle after the other), and deciphered by the inverse operation, multiplying by the inverse. Again,

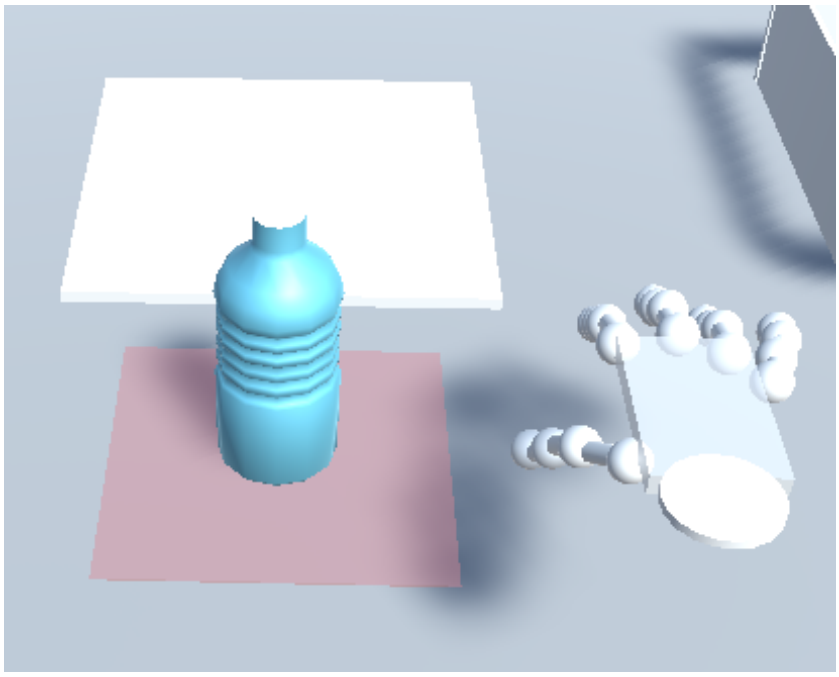


Figure 4.5: The object at the start of a demonstration spawns with its center somewhere inside the red area

mathematically, the saved quaternion is obtained by

$$(w, x, y, z)_{saved} = (w, x, y, z)_{hand} * (w, x, y, z)_{object}, \tag{4.2}$$

and the hand angle is obtained from it by

$$(w, x, y, z)_{hand} = (w, x, y, z)_{saved} * Inv(w, x, y, z)_{object}. \tag{4.3}$$

To note that in quaternions multiplication is not commutative.

The saved file, simple as it was, was also subject to an iterative process. Initially a single file recorded the complete demonstration. Training the NN we observed that some of the reproductions nearly grabbed the object and, before the grasp condition was achieved, the hand followed for the manipulation part (at the time, lifting or rotating the object), without the object in hand. On the successful grasps the hand couldn't move, as the relative position to the object was fixed due to the grasp, which indicated a need for a different referential.

To solve this the demonstrations/reproductions were segmented in 2 parts, each saved on a separate file, initially the reaching part and, after a grasp had been established, the manipulation part. The manipulation part, to avoid the previous problem of the static hand, used instead the relative position of the object at the moment it was considered grasped, allowing the hand to move freely relative to those coordinates.

On the most recent iteration, to allow interaction between objects, the manipulation segment used as a referential for the hand the position of the object to be interacted with (for example, a box for the object to be dropped on).

4.2.1.D Hands

To grasp the object 3 hand models were created/inserted to the VE, one simplistic which we call the Skeletal Hand, and two others that are virtual replicas of the real life robot hands of Vizzy and ICub, two of the robots present in the ISR (Instituto Superior de Robótica). Although the final tests only used the Skeletal Hand, demonstrations were recorded and saved in the database for public use.

To map the glove read values to the virtual hands we used a similar system to all 3 hand models. In terms of rotation and position we simply deduced them from the values read and place the hand model with the deduced values. To map the finger movements we had simply scaled the value read from the sensor and applied it to the respective joint, considering that the value read is a integer from 0 to 1000, by

$$\angle_{joint} = \frac{sensorValue - sensorValueMin}{sensorValueMax - sensorValueMin} * (\angle_{jointmax} - \angle_{jointmin}) + \angle_{jointmin}, \tag{4.4}$$

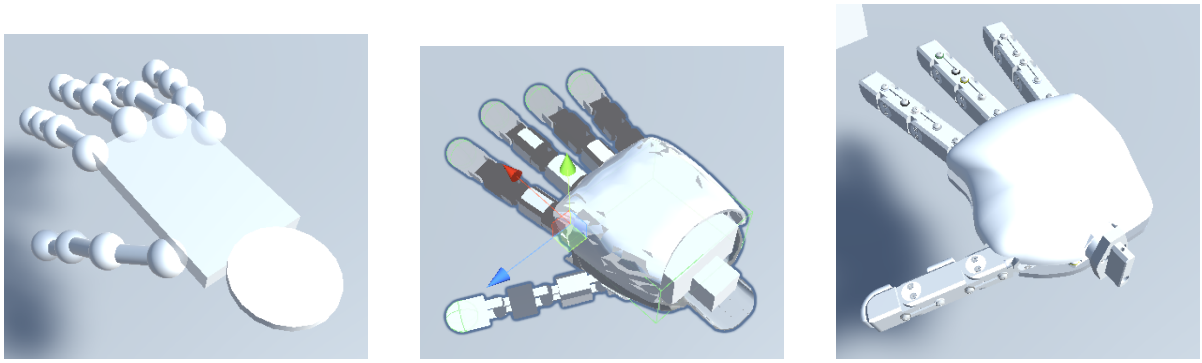
With the $\angle_{jointmin}$ and $\angle_{jointmax}$ the maximum and minimum angle the corresponding joint can move, $sensorValueMin$ and $sensorValueMax$ the maximum and minimum value the glove sensor can read, the $sensorValue$ the value read from the glove, and $\angle_{joint}$ the angle applied to the joint.

- *Skeletal Hand* - a simple hand made to mimic the anatomical proportions of the human hand. The joints are represented by small spheres and the connections between joints by small tubes (figure 4.6(a)).

- *ICub Hand* - Being a commonly used robot in robotics research, the ICub is a robot made to mimic a human child and includes a complete 5 finger dexterous hand. Although it does not possess independent lateral finger movement (all lateral movement is controlled by a single motor), as mentioned before, we're treating the lateral movement as a single value, so this kinematic limitation does not affect the demonstrations (figure 4.6(b)). Also, iCub's thumb has 4 phalanges, instead of 3 like a human. We deal with the extra phalange by giving to the corresponding joint the same value as the following joint.

- *Vizzy Hand* - also included is a functional model of the hand of Vizzy, a Vislab developed robot that possesses a 4 fingered hand. The hand does possess some kinematic limitations, as the hand only has 3 motors, one for the thumb, another for the index, and a final for the remaining two

fingers, that control the complete bending of its corresponding finger(s). To solve this problem we use the mean value of the sensors corresponding to each motor and apply the mean value to all encompassed joints. The little finger is ignored, as the hand contains one less finger, using for the middle+ring fingers motor the average of these two fingers. As the Vizzy hand does not allow lateral finger movement, the value measured of it is discarded (figure 4.6(c)).



(a) Skeletal Hand **(b)** ICub Hand **(c)** Vizzy Hand

Figure 4.6: Hand Models

A brief overview of the hands number of fingers and the number of degrees of freedom (the value does not include the hand spacial and rotational position, only the fingers degrees of freedom), or motors in the case of the robotic hands, can be seen in table 4.1.

	Fingers	Degrees of freedom/motors
Skeletal Hand	5	13
ICub Hand	5	12
Vizzy Hand	4	3

Table 4.1: The virtual hands number of fingers and degrees of freedom

4.2.1.E VE Structure

Having a detailed explanation of some particular aspects of the VE we will now give a brief overview of the VE. The explanation will be kept simple and without details particular to the used engine, Unity, as the same process is believed to be possible to replicate in other engines.

In terms of objects they are placed along a table. Some of the present objects 3d models were obtained in Unity Asset Store, while others where made from scratch. All the interactable objects are tagged as such and have a script that randomizes its position at the start of the runtime. The script has some public variables to define the variance in randomization, if the rotation is also randomized and how much.

In terms of the hand(s) its object actually consist of a bigger body, containing arms and head, but to keep the image simple only the hand is visible. The arms, although invisible, serve to, using the values of the gyroscopes in the hand and wrist, to deduce the spatial position of the hand (as the wrist gyroscope implies the forearm rotation and, being this one dependent of the arm, a approximate position of the hand is possible to be deduced. The head is used simply to anchor the position of the camera. A script is present in this body to, as mentioned before, to mode with the ASDW+QE keys to move the body sideways, front/back and up/down, moving the hand and camera along with it.

The body, besides the movement script, also includes the main script of the VE, which includes all the code that makes the hand movement possible, using the glove or guided through other means, and defines what sort of operation is being done and which object it involves. As public variables it contains booleans to indicate if we're recording a demonstration, having the hand following values of a text file, or being guided by real time values outputed by a NN (the guiding of the hand through text file was not used in the final version of the project, being only part of the iterative process to find ways to communicate with the NN. Nonetheless it was keep as used as a debug or testing tool). For a reasons of organization, most of the code dealing with the communication with the NNs is found on a separate script, having the main script simply invoking its functions. Another variable is used to indicate the used hand. Such is possible by hand all 3 models of hands present at the end of the forearm, and having the variable indicate which models to deactivate and which to activate. Some other public variables are present, including the port used for the glove, some flags used to activate side functions that will be later used (obtain force closure, for example) and reference of the objects to interact with and objects present in the body and hands, to enable their movement, but as some of these are more specific to our work and some alternatives are possible, these variables and processes will not be explained in more detail. Besides the main script a calibrator script is also present on the body, that when enabled indicates the user to open and close the hand, records the values to calibrate the position and opening of the hand, if re-calibration is needed.

Besides the body script another script is vital to the VE, and is present in the tip of each fingertip object. This script detects the collisions (entries and exits) and indicates the main script of them, for purposes of defining which fingers are touching the object, and to activate the respective vibrating plaque, in case haptic feedback is enabled.

4.2.1.F Database

The complete project of this work, and a collection of manipulation tasks demonstrations, is available to everyone in https://github.com/alexamor/Thesis.

The repository contains the Unity project where the work was done, in particular, the VE consisting of the table, objects, hands and scripts. Although originally developed in Unity 2018 it was latter updated

to v2019.4 to ease the recording of demonstrations.

Some objects that weren't used in the final recordings are also present, as they were used in initial parts of the project.

Besides the VE the used demonstrations are also present which include:

- *Demonstrations with Haptic Feedback* - 200 demonstrations of each created task, with haptic feedback enabled. The created tasks are: grab a hammer and place it on a shelf, grab a bottle and place it on a base, grab a can, rotate it and drop it on a box, grab a knife and place ot on a base, grab a mug and bring it to a base and grab a square parallelepiped and place it in a square slot. These were the demonstrations used to test the success-rate of our method, with and without rotation added, in Chapter 6.

- *Demonstrations without Haptic Feedback* - 200 demonstrations of each created task, but with haptic feedback disabled. These were recorded to compare the quality of demonstrations with and without haptic feedback. These demonstrations were used for the comparison between the presence and absence of haptic feedback in Chapter 6.

- *Demonstrations with Robot Hands* - 100 demonstrations for each possible task, using both the Vizzy and iCub hand. Not all the tasks were recorded with the robot hands, as some were impossible due to the kinetics' limitations and proportions of the robot hands. These demonstrations were not used for any test, being their main purpose to complement the database.

- *Miscellaneous Demonstrations* - Some miscellaneous demonstrations used for other tests, or part of the trial and error process, that were not present for the final results, were also kept. These include the recording of demonstrations with a starting random rotation (more on this in Chapter 6) and different batches of demonstrations with haptic feedback.

A brief overview of the demonstrations present can be seen in table 4.2. Miscellaneous demonstrations are not present in the table.

	Hammer	Bottle	Can	Knife	Mug	Cube
With Haptic Feedback	200	200	200	200	200	200
Without Haptic Feedback	200	200	200	200	200	200
ICub Hand	100	100	100	100	-	-
Vizzy Hand	100	-	-	100	-	-
Average Duration (s)	10,5	10,7	10,3	8,2	11,3	11,7

Table 4.2: Number of demonstrations per task and per mode and average duration of each

Instructions on how to use the project and its VE can be found in the readme file present on the GitHub repository.

4.2.2 Neural Network

To be able to replicate an action based on provided examples some form of machine learning is needed. Various methods are able to perform such task, but, considering the data size of our demonstration (to be able to completely capture the human hand we will use 20 variables - 4 for the quaternion angle, 3 for the position, and 13 for the finger joints, and this using some shortcuts to eliminate redundancy) a NN presents itself as the best and most commonly used method.

NN on itself is a diverse method, from which two main types emerge: feedforward NN, that from a set of entry parameters return an output, commonly used in classification problems, and recurrent NN, that are similar to feedforward NN but also possess a loop that allows information from previous iterations to affect the current outcome, making the NN have a sort of funtional memory.

With this in consideration it is now important to understand what we want to achieve with the NN. The training consists of a sequence of iterations that describe the execution of a task and from the NN we want that, through some information of the current state/iteration, we can obtain the next state/iteration. This implies that the NN entry is the information of the current state and the output is the next state.

With this objective in mind some alternatives occur. Using the simpler NN, the feedforward NN, its possible to have as entry the data of the current state and output the next. Such was tested and the results were very poor, never being able to conduct the virtual hand to perform a simple grasp task. This is due to the fact that our demonstration is dependent in time, making that the current state isn't always enough information to correctly predict the next one, easily observable with the example in 4.7. Another good example is a task that requires the glove to go to a point and return. Without some sort of temporal information a state in the middle wouldn't be able to deduce if the hand should move forward or backward. Of course this was also due to more complex aspects of temporal demonstrations, but these are simple examples that confirm the need of temporal data. With temporal data 2 alternatives are still possible with feedforward NN. First, the inclusion of the velocity of the hand at each iteration would give some temporal perspective to an iteration. Although this could solve the most evident problems of temporal demonstrations, it wouldn't really place the demonstration in a stage on time, would just add a little temporal information to the current state, and, as such, could still lead to similar problems as previously mentioned. It would also lead to a bigger data size of the entry/output set of a NN, leading to the need of more middle layers nodes and complicating the prediction, and so it should be avoided if possible.

Another alternative, and one as demonstrated by T. Zhang [1] to be a plausible method, is to instead of using as entry only the current state, to use the last x iterations (in Zhang's case, 5) to predict the next iteration. Although this would indeed put some temporal perspective to the task, and was proved to be a successful method to perform manipulation operations, the same is not certain to be the case for our work, as we are working with a complete human hand (Zhang uses a claw that is represented by its

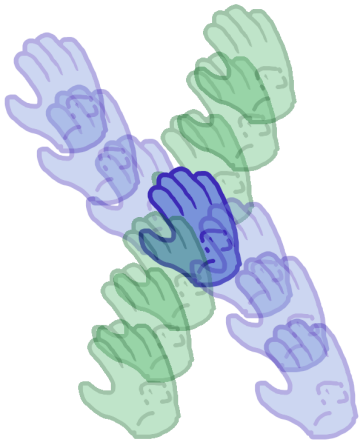


Figure 4.7: Example of a common issue with non temporal methods. We train a method with the blue and green demonstrations. Without any sort of temporal reference(velocity or previous step) the hand at the center can't be certain which is the path to follow

position and boolean to represent if the claw is open/closed), leading to a NN with 100 parameters at its entry for our case. As such, this method was also avoided.

After some testing and deliberation the NN to be employed was a recurrent NN. This one, although more elaborated, is prepared to deal with temporal demonstrations as it uses information of the previous iterations. This was also proved to be an effective method for manipulation training by some previous successful works [3] [24]. The particular type of recurrent NN that we ended up using was an LSTM (Long Short Term Memory), a type of recurrent NN that has short term (i.e. memory of immediately before instances) and long term memory (i.e. some memory of the instances further back from the current). This one was chosen as the one to be so as it's the one best adapted for this kind of tasks, which is evident by being the same choice of recurrent NN employed by the previously mentioned works.

In conclusion, the NN that was employed is an LSTM that has as entry the current state and outputs the following one (having each state a total of 20 parameters), that, by being recurrently called, creates a sequence of states that represent a reproduction of the task, if successful.

4.2.2.A LSTM Structure

As previously mentioned the demonstrations were segmented into two phases. To reflect this two LSTMs are also used, one being trained for the reaching phase and another for the manipulation phase. Although used for separate segments the LSTMs share the same structure, differentiating in the trained data and the information used as a starting condition (more on this later).

The LSTMs contain just 3 layers, the input layer with 20 nodes (the data size of an iteration), an output layer of the same size (as the output will be the iteration to use next as an input to the LSTM), and an intermediate layer of size 3280 (8*size of input*size of output + 4*size of input, rule of thumb of

number of intermediate nodes recommended for NNs with multiple inputs) (figure 4.8). The optimizer is *adam* using the mean squared error to calculate the loss, i.e. the estimation or prediction error. This optimizer and error function were chosen as they are one of the recommended for NNs with multiple inputs. Other loss functions were tested but proved to be less reliable, being slower or, in most cases, having difficulty stabilizing.

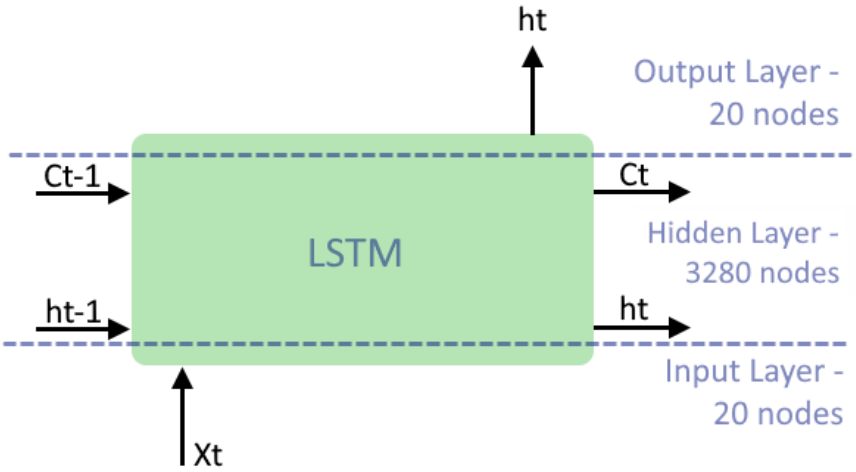


Figure 4.8: Diagram of the used LSTM with its layers and sizes with X the input, h the output, C the cell state, and t and t-1 to represent current and previous iterations

To initialize the LSTM the C0 (initial cell state) and h0 (initial previous iteration) are defined by a rectified unit activation function (creates a vector of the max value or zero, whichever is bigger, for every value).

As we are working with an LSTM, a NN that has short-term memory, the input is actually multidimensional, that is, it processes, in our case, 10 iterations at a time, from which it will try to predict the 11th one. We chose 10 as the short-term memory range as it represents 2 seconds of demonstration, and this seemed a sensible value and proved to carry enough information for a reproduction to be made. It was also tested a range of 5 iterations (1 second of demonstration) but the reproductions had a harder time performing the tasks (most of the reproductions ended up switching between two positions or stabilizing in space, as in 1 second of demonstration there wasn't enough movement, so a good prediction of the following movement was to keep still).

In table 4.3 we have a brief overview of the final LSTM parameters.

Referring to the training the data is first treated by scaling the demonstrations values to values that can be processed by the LSTMs, that is, values from 0 to 1. To do so, a scale was chosen based on the values present on the demonstrations. For example, the fingers' joints were read as integers from 0 to 1000, and the angles, being a quaternion, would always be found between -1 and 1. For the

Size of Input	Size of Output	Size of Middle Layer	Nr of Layers	Activation Function	Optimizer	Loss Function
20*10	20	3280	3	relu	adam	mean squared error

Table 4.3: Parameters of the used LSTMs

spatial position, being it relative, a script was made to indicate the max and minimal value present in the demonstrations. Having a scale defined, the demonstrations values could be converted to values between 0 and 1 through simple math.

Afterwards the demonstrations were segmented in groups of 10 successive iterations and the 11th following so the LSTMs could process them. The full collection of groups taken from all the recorded demonstrations were inputed as train for the LSTMs for an undefined number of epochs (an epoch represents the training of the full set of training data). To ascertain when to stop training, the loss was read at the end of each epoch, and when the loss stabilized the training process ended. To have a perception of the training time, the training of both networks needed for each task takes about 24h to 36h to complete on the used computer (to compare, the computer specs are: CPU: Intel i7-7700HQ 2.80GHz, GPU: NVidia GeForce GTX 1050, RAM: 16GB, even though the program only uses a little more than 1GB of RAM. CUDA drivers and SDK were present to allow the network to take advantage of GPU resources).

Having the LSTMs trained for a task they are now ready to perform reproductions. To do so the reaching LSTM needs 10 values to start recurrently predicting the following ones. This starting point can be chosen by hand, but to test we performed a similar procedure to the one commonly seen in simpler NNs tests, in which we reserve a portion of the training data as validation data, and then use the first 10 iterations of these demonstrations as the starting point for the reproduction. This was done to ensure the reproductions start at a plausible starting position.

While an LSTM is outputing recurrently the next iterations it is simultaneously sending these values, unscaled to the real values using the inverse of the scale previously defined, to the VE, that has the hand following the received values. After receiving a value the VE sends back the values that it has actually placed the hand in. This will be the same values that it has received, with the exception of when the hand pushed the object in the current iteration. The LSTM will receive the real position, scale it, add to the stack, and predict the next one. The iteration outputs are limited to sending one every fifth of a second, to make the hand move at the speed it was recorded to (figure 4.9).

To make the Python based program that runs the LSTMs communicate with the VE, in our case a C# based software, in real time, offline ports were used. To do so an unoccupied port was used (56000), and both the Python program and the VE have a sender and receiver using this port, from which they send back and forth 21 floats (20 representing an iteration and a final one used as a boolean flag, to be

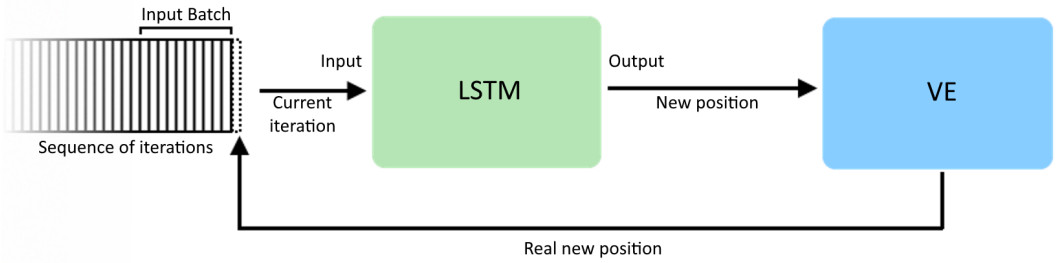


Figure 4.9: Diagram of the task reproduction process

explained next).

At a certain moment/iteration the object will be considered grabbed and the VE will send a boolean flag indicating such. The Python program, receiving this flag, will exchange the reach LSTM for the manipulation LSTM, as we are now on the second phase, being this one that will output the next predictions until the end of the reproduction.

The change to the manipulation LSTM comes with an obstacle, as it receives as a start position a single iteration, and, unlike the reach LSTM, we cannot define a random start, as the demonstration is already ongoing. So we need to define how we will represent the start of the manipulation LSTM, and train it accordingly.

First, and most obvious, we used as a start the previous 9 iterations belonging to the end of the corresponding reach phase. This showed moderate results, but in some tasks, the hammer for example, the results were very bad, with 15% success rate in that example. We believe such happened due to the discontinuity present (the moment the object is grabbed the reference point changes from the object to the object it will interact with) and leads to the LSTM having trouble dealing with it (we believe the hammer was a more flagrant case as it is a more complex task in the number of changes of direction, first the hand gravitates towards the object, then lifts it in the opposite direction, aligns with the shelf, and finally moves forward).

Considering that the discontinuity was the problem we moved to an alternative, the 9 iterations before the object were grabbed, as seen by the manipulation LSTM, were just replicas of the first iteration of the manipulation phase. This led to better results on the hammer, but worse results in tasks that had the objective further away. What was observed is that the hand, after grabbing the object, either stood almost still, or moved slower and dropped the object at the front of the objective. We believe that this was due to the fact that it had a static start, being trained to standing still while in the demonstration the

hand was never still.

Finally, and the one used in the final version of the project, was a mix of the previous 2, using the previous phase's last iterations but avoiding the discontinuity. This was done by registering the values referring to the relative position (the last 7 values of a iteration) of the first iteration of the manipulation phase and the last iteration of the corresponding reach phase. The 9 last values of the reach phase were subtracted by the values present on the last reach iteration and added the values present in the first iteration of the reach phase, effectively changing the reach values to near to what they were supposed to be if it was using the manipulation object as the reference (figure 4.10). This changed the overall quality of the reproductions, achieving better or the same results in all the tasks, compared with either of the two previous methods.

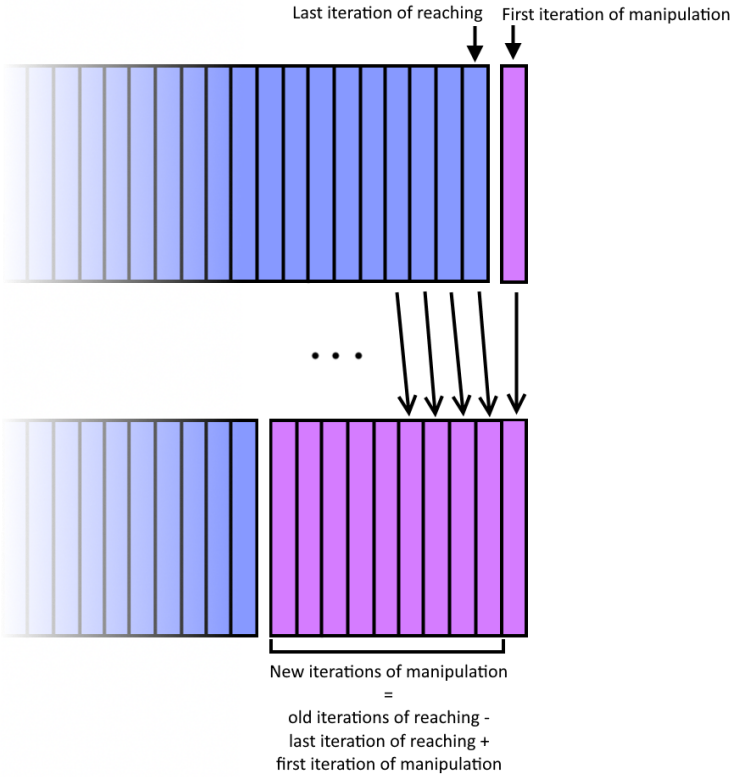


Figure 4.10: Visual representation of the change to manipulation iterations, with blue the reaching iterations and purple the manipulation iterations

Chapter 5

Results

To verify the quality of our work a collection of tests was made. Although they came with varied purposes, from testing the overall quality of the method to testing the differences of including or not haptic feedback, most of them followed the same process.

First, from 200 demonstrations of a task, 180 of these demonstrations were used to train the NNs and the remaining 20 for testing purposes to use later. It was also tested the use of less demonstrations (100), but as the results were shown to be way less consistent, 200 demonstrations was the value used for training. For comparison with other works 200 demonstrations rounds to about 40 minutes of demonstrations.

Afterwards, having the NNs completely trained, of the 20 testing demonstrations one is chosen and the first 10 iterations of it are used as a plausible start for the task. The NN will then try to predict the next iteration, and, doing so recurrently, create a sequence of iterations that will, hopefully, recreate the task at hand. The successfulness of the reproduction is verified on the VE, which has the virtual hand following the values outputed by the NNs. More details on how the the NNs work or how the communication with the VE is achieved can be found in Chapter 4.

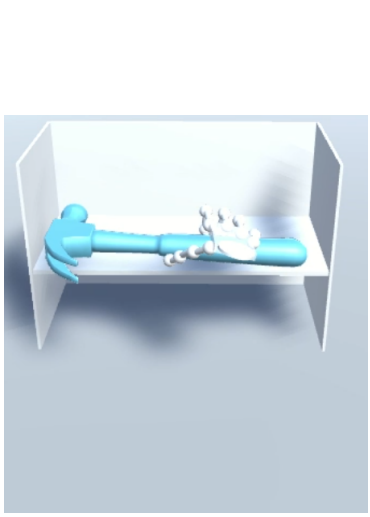
The success of a reproduction is considered so when the goal of the task is achieved (the hammer is placed on the shelf, the can is rotated and placed in the box, etc.). After using the 20 starting positions the result consists on the percentage of successful reproductions. It should be noted that during the tests the object still spawns at a random position, to make both NNs having to deal with not directly trained starting positions.

5.1 Quality of Reproductions

The first experiment to be made is to simply test the method by trying to reproduce a number of tasks. The tested tasks were:

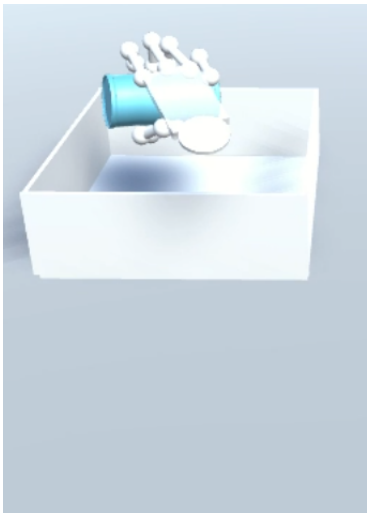
- Grab an hammer and place it on a shelf (Figure 5.1(a))
- Grab a bottle and place it on a base (Figure 5.1(b))
- Grab a can, rotate it sideways, and drop it on a box (Figure 5.1(c))

- Grab a knife and place it on a base (Figure 5.2(a))
- Grab a mug and bring it to a base (Figure 5.2(b))
- Grab a cube and place it on a square slot (Figure 5.2(c))



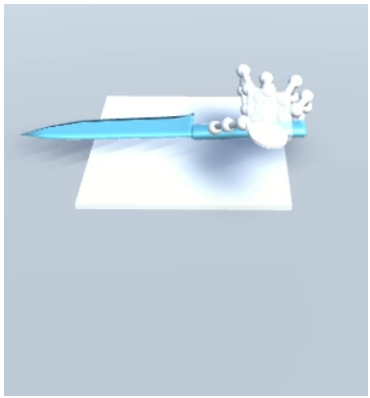
(a) Hammer task

(b) Bottle task

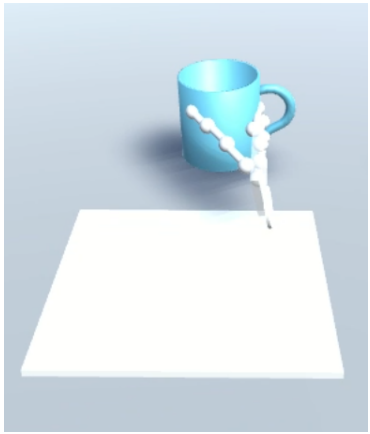


(c) Can task

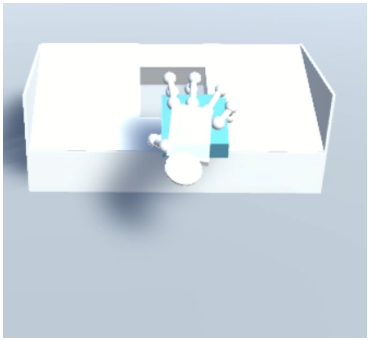
Figure 5.1: Performed tasks 1



(a) Knife task



(b) Mug task



(c) Cube task

Figure 5.2: Performed tasks 2

The success rate of the reproductions can be found in the graph 5.3.

In general results were good, considering the common values for this kind of work, showing the plausibility of our method.

In particular we have the cube task on the low side of the results. This was somewhat expected as

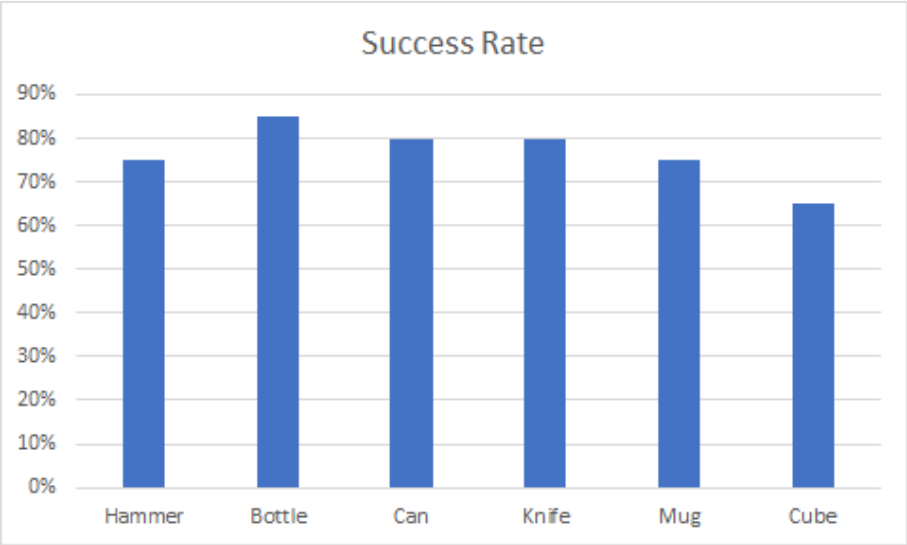


Figure 5.3: Success Rate of the different reproduced tasks

from all the tasks it's the one with less error tolerance, as the task is only considered successful if the cube is placed inside a slot only slightly bigger than the cube itself (for example, some of the failures ended with the cube resting slightly on the side or below the slot).

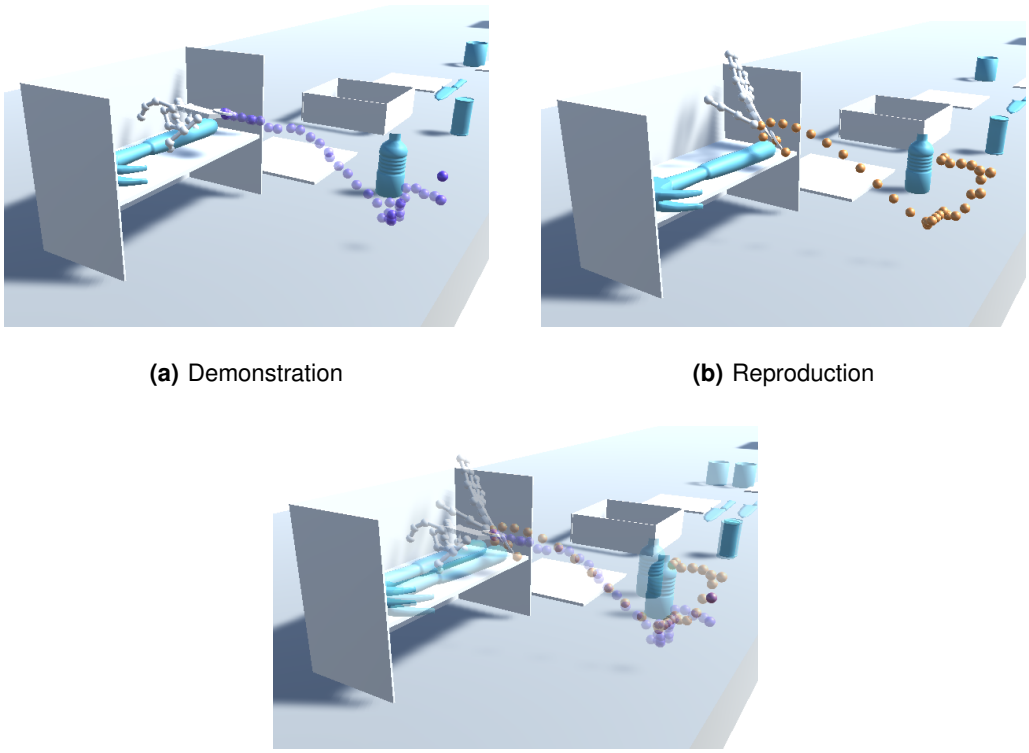
We also found that the reproductions had a smoother movement and moved in arches, possibly due to the LSTMs trying to fit the movement in a mathematical function manner 5.4.

5.2 Rotation Tolerance

The demonstrations and reproductions were done with random starting positions so we're aware that our method has position tolerance, but the case that the object starts slightly rotated, or the reading of the object rotation is not well measured and presents a slight deviation, should also be considered.

Initially, to test for the rotation variation, we made a new batch of demonstrations that made the object not only appear in a random position, but also with a random rotation (up to $\pm 10º$ and up to $\pm 20º$, two sets were recorded).

After testing the method trained with these new demonstrations (having the tests also rotating the objects at the start) the results, right from the start, showed a very low success rate, rounding to about 15%, for both the $\pm 10º$ and $\pm 20º$ sets. From observation of the failures we could see that at the start of the reproductions the hand rotated excessively to fit to the object, leading to not being able to grab the object with its excessive rotation, or grabbing it from an uncomfortable position, not trained for, and not being able to correctly continue with the task. We believe that this was due to the LSTMs normal behaviour of trying to linearize the movement and also to the fact that we didn't simply just made the task more varied, but actually added a new phase (of rotating the hand before starting the task), which



(a) Demonstration

(b) Reproduction

(c) Both

Figure 5.4: Comparison for a task with a similar object positions between a task demonstrated by a human and one obtained with the method

led to an increase of difficulty for the task. Another aspect that might justify these bad results is that with the added variance the 200 demonstrations were no longer enough to encompass the full range of possibilities, needing more to fully capture the new combinations of rotations and positions.

To contour this problem we considered to, instead of training with the rotated object, to simply use the method previously trained with the object with no starting rotation variance, and test nonetheless with the object spawning with random rotations, leaving the method to try to deal with the shift in rotation. Also to note that the glove, when connected at the start of each demonstration, also starts with a slight variation of its angular position, so, as we're using the relative position, the training with the object starting with a static rotation naturally includes a slight variance in its angular position.

The success rate of the reproductions, for $\pm 10º$ and $\pm 20º$, and the previous default $0º$ values, to help compare, can be found in the graph 5.5.

The results were what was to be predicted, the method lowers its success rate as more variance is added to it. The can task presents itself as an exception, having a increase in success rate at $\pm 20º$, probably due to the innate randomness of the testing process. From comparison we can also see that the hammer and cube task are less tolerant to the rotation variance, which is most likely due to the fact

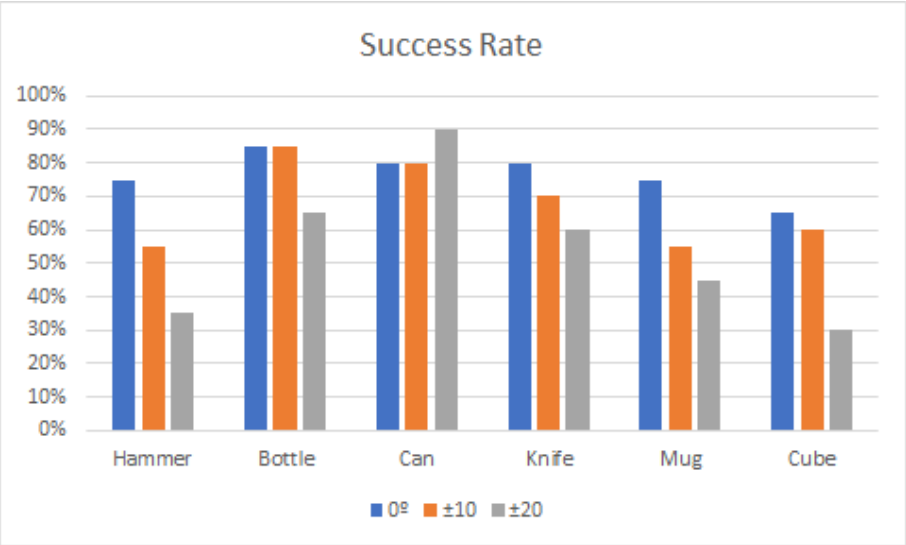


Figure 5.5: Success Rate of the tasks with different initial rotations

that these tasks are dependent on the rotation of the object, unlike the others (the cube and the hammer might not fit in their target base if placed tilted). This leads to another point, that some objects, as the bottle and the can, were tested of their rotation tolerance but, being symmetric in nature, their rotation is irrelevant and don't make the task different. Nonetheless the tests to these objects prove tolerance to reading errors.

Also to note, in comparison with the previous method of training with rotation variance demonstrations, the reproductions in these tests were more fluid and smooth, having the hand rotating linearly while reaching the object or base.

In conclusion, the method proved to be capable of dealing with some variance of rotation, albeit at a lower success rate.

5.3 Haptic Feedback Inclusion

As a final test, considering that for all the recorded demonstrations we had the gloves using haptic feedback, it would be of interest to test if the inclusion of the haptic feedback actually leads to better demonstrations, as it is in the recommended next steps of a number of works in this area.

To test this first we did the most obvious, record again the demonstrations but without the use of the haptic feedback. For this the all the previous tasks were re-recorded, used for training of the LSTMs, and finally, tested. The results of these tests can be seen in the graph 5.6.

As we can observe the exclusion of haptic feedback lead to slightly worse reproductions. Some of the tasks deviated more from the average, as the knife which managed to perform better without haptic feedback and the mug which performed especially poorly. Besides the natural randomness present in

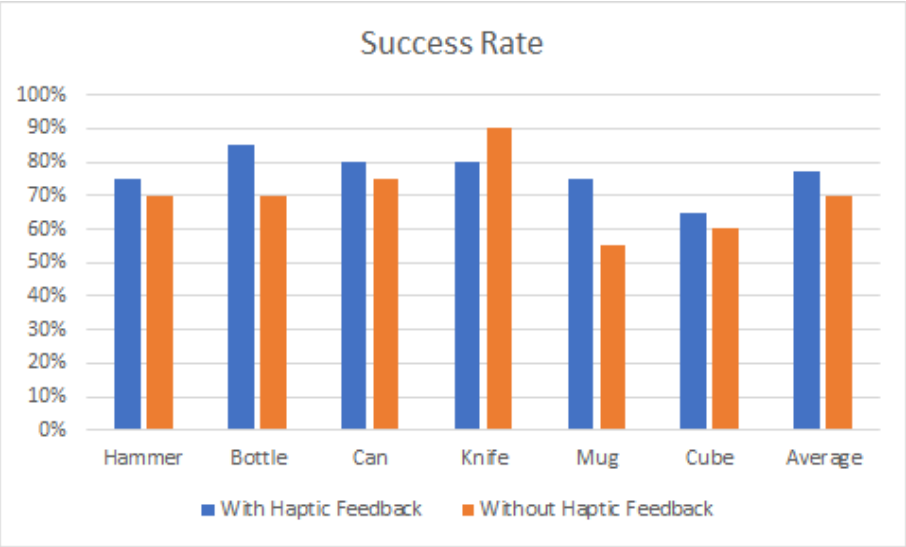


Figure 5.6: Success Rate of the tasks with and without haptic feedback

the recording of demonstrations and the testing of reproductions, some justifications for the pecularity of results of this tasks are that the knife, having a small handle and as all the grasps made are precision grasps (the object is grasped by the tip of the fingers) is is easier to rely only on visual aid to perform this task, and the mug, as it obstructs the view of the fingers, the haptic feedback might have a bigger weight.

Having the results of the presence of haptic feedback we also took advantage of these results to test the quality of a commonly used metric to evaluate the quality of grasps, the force closure.

Force closure is a boolean property of grasps that verifies if the grasp is good enough to sustain any perturbation and, as such, can be used to ascertain the quality of a grasp [27].

To verify if a grasp is a force closure we used the theorem that affirms if the convex hull of the torques includes the center, then the grasp presents force closure. To verify this in Unity we grab the object and register each point of contact and the normal of this point of contact of the object to calculate the torque of each (the torque is calculated as the external product of the inverse normal, representing the contact force, and radius, that is a vector that goes from the center of the object to the contact point). Afterwards, to simulate friction, a cone of friction is created and, having it centered on each normal and obtaining from its surface 4 equidistant friction normals, use all these normals to calculate torques. After registering all the torques of a grasp on a text file we use QHull, a convex hull software that receives vectors and outputs information of the convex hull created by them, including the vertexes of the convex hull. By inserting all the torques of a grasp and the center (0,0,0,0,0,0) we can verify if the center is contained by its vertexes, because if the center is not a vertex it implies the center is inside the polygon that is the convex hull and, as such, the inputed grasp is a force closure.

To test the quality of the haptic feedback using the force closure property we grabbed objects on the

VE and saved the torques to later verify if the property is present. 100 grasps of varied objects were saved with the haptic feedback turned on and another 100 grasps on the same objects were saved with the haptic feedback turned off. The percentage of grasps considered force closures can be seen in the Table 5.1. The table also includes the previous average success rate as a means of comparison.

	With Haptic Feedback	Without Haptic Feedback
Success Rate	77%	70%
Force Closure	77%	78%

Table 5.1: Percentage of successes by trying to reproduce tasks and percentage of force closure grasps measured

As we can observe, according to the force closure results, the presence of haptic feedback has no weight in the quality of the grasps, although according to our previous results haptic feedback lead to better reproductions. To note that even though the force closure tests only affect the grasp, ignoring if a task is being done or if the object is simply being grasped, the haptic feedback only aids on the grasp portion of the tasks and, as such, the fact that in the force closure tests no tasks were done should have no bearing on the results.

From this we can conclude that although force closure can indicate some value of the quality of a grasp, it does not directly implicates the general quality of a grasp, nor of the task the grasp is presented in. One justification for this is that the force closure does not measure finger penetration, as observed by M.Chessa [23], as the contact point is assumed as the point of entry of the finger, leading to an overlook of an aspect that might influence the quality of a grasp.

In conclusion, as commonly suggested in previous works we can now conclude that haptic feedback does actually influence the quality of demonstrations, albeit slightly.

Chapter 6

Conclusion

In this work we developed a VE consisting of a table with a number of objects to interact with, for the purpose of demonstrating simple tasks, using a glove with sensors to interact with it. These demonstrations were meant to be used as a guide to teach robots to perform the same grasping tasks as we humans, through a approach often called imitation learning, a common approach to machine learning that uses the human example as a basis to teach elaborate tasks to robots. We also took the opportunity to test the common notion that the use of haptic feedback leads to better demonstrations and, by consequence, better reproductions.

To export the knowledge present on our demonstrations we used NNs, in particular LSTMs, a memory based NN, that received as training the recorded demonstrations. As a novel approach we segmented the demonstration in 2 phases, before and after the object is considered grasped, to ensure we have a firm grasp before moving on with the reproduction.

After testing some reproductions, starting with varied starting positions, we could confirm that our method is capable of reproducing trained manipulation tasks for a complete dexterous hand, even when some untrained variance is introduced.

Also, as previously mentioned, the quality improvement created by the use of haptic feedback was tested, and it was observed that its inclusion made the reproduction just slightly better.

Finally, for anyone who wants to test their own grasping methods the VE and recorded demonstrations are freely available at https://github.com/alexamor/Thesis.

6.1 Next Steps

First of, considering we have a system capable of guiding a virtual hand to perform grasping task and, as some demonstrations were even recorded using real life robot hand replicas (for the iCub and Vizzy robots), we think it would be of interest to create a method to transfer this model to real life robots.

Following the same train of thoughts that the better immersion of the added haptic feedback lead to better demonstrations, the same could be possible by again improving immersion using a headset. Also, the inclusion of a headset could lead to more information of study, for example the aid presented by our gaze of the objects.

Bibliography

[1] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 5628–5635.

[2] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Robotics Proceedings - Robotics: Science and Systems XV*, 2017.

[3] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni, "Learning real manipulation tasks from virtual demonstrations using lstm," *arXiv preprint arXiv:1603.03833*, 2016.

[4] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34, July 2008.

[5] C. Olah, "Understanding lstm networks," 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] D. A. Rosenbaum, *Human Motor Control*. Pennsylvania State University, University Park, PA: Academic Press, 2009.

[7] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, April 2014.

[8] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2010, aAI3448143.

[9] J. Weisz and P. K. Allen, "Pose error robust grasping from contact wrench space metrics," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 557–562.

[10] R. Balasubramanian, L. Xu, P. D. Brook, J. R. Smith, and Y. Matsuoka, "Physical human interactive guidance: Identifying grasping principles from human-planned grasps," *IEEE Transactions on Robotics*, vol. 28, no. 4, pp. 899–910, Aug 2012.

[11] J. Romero, H. Kjellström, and D. Kragic, "Modeling and evaluation of human-to-robot mapping of grasps," in *2009 International Conference on Advanced Robotics*, 07 2009, pp. 1 – 6.

[12] R. Detry, E. Baseski, M. Popovic, Y. Touati, N. Kruger, O. Kroemer, J. Peters, and J. Piater, "Learning object-specific grasp affordance densities," 01 2009.

[13] M. Ciocarlie and P. Allen, “Hand posture subspaces for dexterous robotic grasping,” *I. J. Robotic Res.*, vol. 28, pp. 851–867, 06 2009.

[14] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, “Imitation from observation: Learning to imitate behaviors from raw video via context translation,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 1118–1125.

[15] N. Singh and E. Todorov, “Imitation learning for reaching and grasping in virtual environments,” in *International Conference on Development and Learning 2006*, 2006.

[16] Z. Lorincz, “A brief overview of imitation learning,” *SmartLAb AI*, 09 2019. [Online]. Available: https://medium.com/@SmartLabAI/a-brief-overview-of-imitation-learning-8a8a75c44a9c

[17] H. Tian, C. Wang, D. Manocha, and X. Zhang, “Realtime hand-object interaction using learned grasp space for virtual environments,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 8, pp. 2623–2635, Aug 2019.

[18] M. T. Ciocarlie and P. K. Allen, “Hand posture subspaces for dexterous robotic grasping,” *The International Journal of Robotics Research*, vol. 28, no. 7, pp. 851–867, 2009.

[19] C.-b. Moon and W. Chung, “Practical probabilistic trajectory planning scheme based on the rapidly-exploring random trees for two-wheeled mobile robots,” *International Journal of Precision Engineering and Manufacturing*, vol. 17, pp. 591–596, 05 2016.

[20] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, “Plato: Policy learning using adaptive trajectory optimization,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 3342–3349.

[21] X. Gao, R. Gong, T. Shu, X. Xie, S. Wang, and S.-C. Zhu, “Vrkitchen: an interactive 3d virtual environment for task-oriented learning,” in *ICML 2019 Workshop RL4RealLife Program Chairs*, 2019.

[22] S. Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, J. A. Castro-Vargas, S. Orts-Escolano, and J. Garcia-Rodriguez, “A visually realistic grasping system for object manipulation and interaction in virtual reality environments,” *Computers Graphics*, vol. 83, p. 77–86, Oct 2019. [Online]. Available: http://dx.doi.org/10.1016/j.cag.2019.07.003

[23] M. Chessa, G. Maiello, L. K. Klein, V. C. Paulun, and F. Solari, “Grasping objects in immersive virtual reality,” in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2019, pp. 1749–1754.

[24] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine, “Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3758–3765.

[25] V. Kumar and E. Todorov, “Mujoco haptix: A virtual reality system for hand manipulation,” in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Nov 2015, pp. 657–663.

[26] M. Höll, M. Oberweger, C. Arth, and V. Lepetit, “Efficient physics-based implementation for realistic hand-object interaction in virtual reality,” in *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2018, pp. 175–182.

[27] A. Bicchi, “On the force-closure property of robotic grasping,” *IFAC Proceedings Volumes*, vol. 27, no. 14, pp. 213 – 218, 1994, fourth IFAC Symposium on Robot Control, Capri, Italy, September 19-21, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667017473174