Learning to Grasp Objects in Virtual Environments through Imitation

Alexandre Vieira Filipe
alexandre.filipe@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

January 2021

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. We propose using the human manipulation of objects as a basis for robot manipulation, applied through imitation learning. Considering that obtaining the data of the human manipulation from video can add noise due to obstructions or simply due to the complexity of a fully dexterous hand, we present the virtual environment as a better medium to collect data and record the samples to be used for robotic imitation. The virtual environment will be interactable with a glove with position sensors capable of fully capture the human hand movements in all its intricacy, and vibrating plaques on the fingertips to simulate the touch of a virtual object (haptic feedback). For the work we consider two experimental conditions: Presence vs. absence of haptic feedback while executing grasps. With the samples recorded, the subconscious human mannerisms will be applied and tested on a virtual hand using the output of neural networks, in particular, a recurrent neural network responsible for the reaching portion of a task and another responsible for the remainder/manipulation portion of the task, being the output followed switched from the reaching recurrent neural network to the manipulation recurrent neural network when the object to interact is considered grasped. Both recurrent neural networks will be trained with the recorded demonstrations, each with their respective portion, enabling the robot to grasp objects successfully in most attempts. It will also be tested if the use of haptic feedback actually leads to better demonstrations. We also aim to provide freely our dataset, to enable researchers without the hardware needed to record demonstrations to test their own training algorithms.
Keywords: Robot manipulation, imitation learning, virtual environment, neural network

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 is 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 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. Contributions

This work will add to the state of the art with the introduction of our trajectory generation algorithm with a 2-phases system, the addition of haptic feedback to the trajectory generation algorithm, the testing in difference in quality of demonstrations trained with and without haptic feedback present, and by offering a public database of demonstrations to aid researchers of this field of study.

2. Background

As the ability to grasp and manipulate objects, making use of tools and all the plethora of activities achievable with our hands is one of the most impactful skills of mankind [8], it's was expected that the engineering community would gain 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.

2.1. Approaches to Grasping

The literature presents a wide diversity of approaches for robot grasping, differentiating themselves by the knowledge of the object the robot is trying to grasp [2], that is, if the object is known, familiar or similar to a known own, or completely unknown. In our work we will deal with known/familiar objects.

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 [2]:3-D mesh models and contact-level grasping, learning from trial and error and learning from humans.

With 3-D mesh models and contact-level grasping a model of the object is created and we try to deduce how it would physically make sense to grasp the object, with learn from trial and error we have a AI repeating attempts to grasp the object without former knowledge, learning with each failure, and with learning from humans we attempt to port the knowledge of grasp of humans to the robots.

In this work we will follow the approach of learning from humans, in particular the particular approach of behavioural cloning, where a human demonstrates a task and the robot tries to replicate it, as it has shown promising results in previous works [9] [7] [5].

2.2. 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 [9] [5] [6]. 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 [9]. 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 [5] [6], 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 (figure 1), 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 [4], 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).

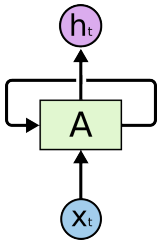


Figure 1: Diagram of a recurrent NN with X the input and h the output [4]

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, 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 Memory), that has an added system to deal with memory 2, 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 are included.

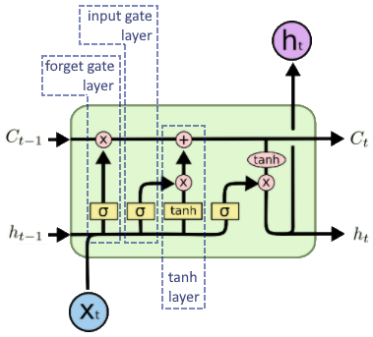


Figure 2: Diagram of an LSTM with X the input, h the output and C the cell state [4]

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 [7].

3. Implementation

In this work we will have a VE where we will record demonstrations, using a glove with sensors, and use this demonstrations as training for an AI to try to perform autonomously these same tasks in the VE. We will now explain how this was implemented.

3.1. 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 3). 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.

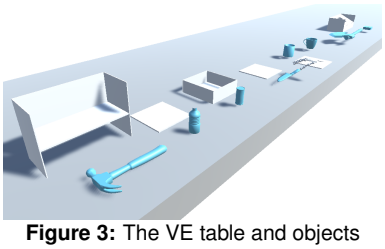


Figure 3: The VE table and objects

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.

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.

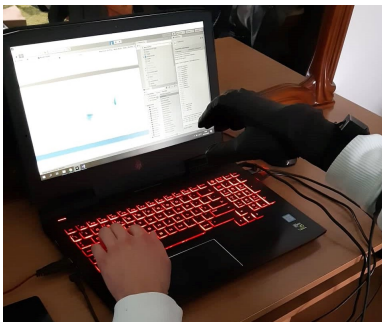


Figure 4: Example of the VE control

3.1.1 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.

3.1.2 Object interaction

Having the glove fully captured it was next needed to enable objects to be grabbed. This operation took some iterations but the final version had simply the condition of object grabbed, that was verified if the thumb and at least one other finger was touching the object, that when verified glued the

object to the hand. While grabbed the remaining fingers were glued to the object the instant they touched it, to replicate the adjustment of the fingers to the object. When the fingers were glued to the object their sensor values were saved and if it was verified that the sum of all touching fingers opened more than a threshold, the object would be dropped.

3.2. 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 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 5 has a visual representation of it.

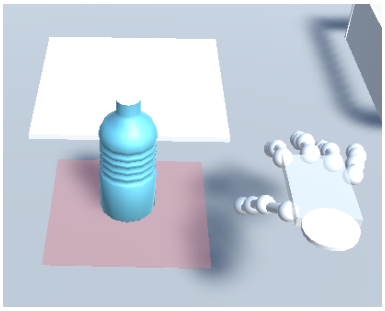


Figure 5: The object at the start of a demonstration spawns with its center somewhere inside the red area

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). 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.

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).

3.2.1 Hands

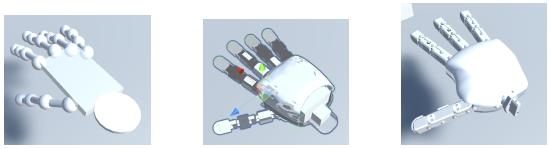
To grasp the object 3 hand models were created/inserted to the VE, one simplistic which we call the Skeletal Hand (Figure 6(a)), and two others that are virtual replicas of the real life robot hands of Vizzy (Figure 6(c)) and ICub (Figure 6(b)), two of the robots present in the ISR (Instituto Superior 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 (for example, the value read from the sensor goes from 0 to 1000, and the respective joint can move from 0º to 90º, so if the sensor reads the value 500 we would rotate the joint to 45º).

The kinematic limitations of the robot hands were kept and dealt with by using the average of the read sensors (For example, if a single motor controls two joints we apply the average of the two joints to both).

3.3. 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,



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

Figure 6: Hand Models

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.

3.3.1 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 and hands. 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. These were the demonstrations used to test the success-rate of our method, with and without rotation added, in the Experiments.

- *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 the Experiments.

- *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 Experiments) and different batches of demonstrations with haptic feedback.

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

3.4. 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 input is the

information of the current state and the output is the next state.

As previously mentioned we are going to use an LSTM, a type of recurrent NN, as these are more suited for sequential tasks, that has an input and output of the size of a iteration, and that by being recurrently called, creates a sequence of states that represent a reproduction of the task, if successful.

3.4.1 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). 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.

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).

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 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.

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 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.

To define the initial states for the manipulation LSTM some alternatives were done along the work, but the final version defined it by saving the value of the first manipulation iteration and the last reach iteration and having for the initial state the last 9 reach iterations, subtracted by the last reach iteration and summed by the first manipulation iteration, and the first manipulation iteration, making a continuous sequence that is near to what would be read if the iterations previous to the manipulation portion, using the manipulation referential, were saved.

4. 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.

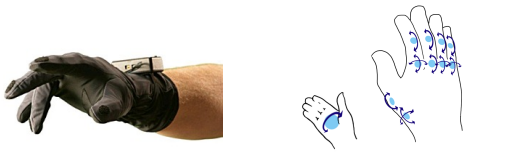
The used glove was the VMG35-Haptic (figure 7(a)), made by Virtual Motion Labs. This glove possesses 2 gyroscopes, one on the hand and another on the wrist, to indicate the angular position of the hand, sensors that measure the bend of finger joints and palm (figure 7(b)) and vibrating plaques on the tip of each finger to permit haptic feedback.

The VE will be hosted in Unity v2019 and the LSTMs will be made using Tensorflow.

5. Experiments

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.



(a) VMG35-Haptic Glove

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

Figure 7

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.

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 8(a)), grab a bottle and place it on a base (Figure 8(b)), grab a can, rotate it sideways, and drop it on a box (Figure 8(c)), grab a knife and place it on a base (Figure 9(a)), grab a mug and bring it to a base (Figure 9(b)) and grab a cube and place it on a square slot (Figure 9(c)).

The success rate of the reproductions can be found in the graph 10.

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 from all the tasks it's the one with less error tolerance, as the task is only considered successful

(a) Hammer task **(b)** Bottle task **(c)** Can task

Figure 8: Performed tasks 1

(a) Knife task **(b)** Mug task **(c)** Cube task

Figure 9: Performed tasks 2

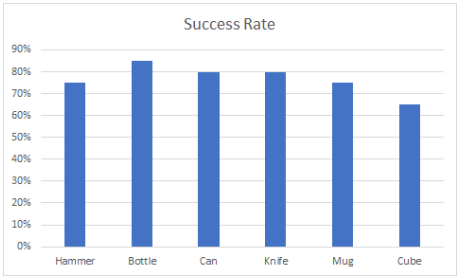


Figure 10: Success Rate of the different reproduced tasks

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).

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 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 11.

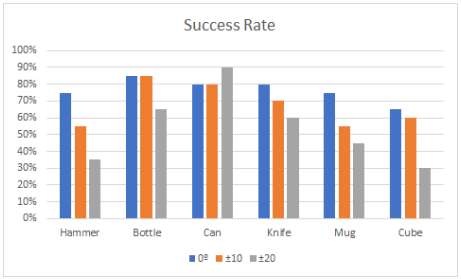


Figure 11: Success Rate of the tasks with different initial rotations

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 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 12.

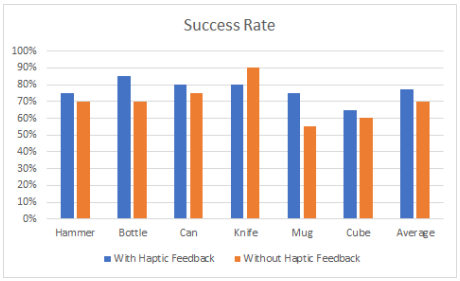


Figure 12: Success Rate of the tasks with and without haptic feedback

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 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 [1].

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 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 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 [3], 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.

6. Conclusions

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

We think it would be of interest to create a method to transfer this model to a real life robot, as some demonstrations were even recorded using a robot hand replica.

Other more simple improvements that could be added or tested are to have better immersion, using a Virtual Reality Headset for example, and create more elaborate tasks, using malleable objects, for example.

References

[1] A. Bicchi. On the force-closure property of robotic grasping. *IFAC Proceedings Volumes*, 27(14):213 – 218, 1994. Fourth IFAC Symposium on Robot Control, Capri, Italy, September 19-21, 1994.

[2] J. Bohg, A. Morales, T. Asfour, and D. Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, April 2014.

[3] 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)*, pages 1749–1754, 2019.

[4] C. Olah. Understanding lstm networks, 2015.

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

[6] 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)*, pages 3758–3765, 2018.

[7] 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.

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

[9] 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)*, pages 5628–5635, May 2018.