Representation Learning for In-hand Object Manipulation

Gonçalo Chambel
goncalo.chambel@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

September 2021

Abstract

More and more often, humans rely on robots to replace them in performing simple tasks. However, the ability of robots to adapt and perform more complex tasks is still reduced, specially when it comes to dexterous tasks. The goal of this thesis is to develop a model that is capable of using in-hand manipulation to control a set of objects using Deep RL methods, based on the tactile feedback obtained from a simulated robotic hand. This is a complex problem for Deep RL algorithms, due to the large number of variables to account for. To bypass this, this work intends to explore the possibility of using a latent representation of the state, in order to test if the updated model learns faster and/or better to perform a set of predefined tasks in a simulated environment. This representation is obtained from another model and should be comprised of only the most important features of the original state. Moreover, to further reduce the number of variables, this work evaluates the performance difference in using different action spaces and synergies to control the robotic hand. The main result of this work is that by using the latent representation of the input, the controller is able to generalize the manipulation to objects that have unique shapes. Regarding the different action spaces, this work also demonstrates that by using synergies, with a dimension of less than half than the original dimension, the controller is able to achieve the same level of performance.

Keywords: In-hand manipulation; Deep RL; Representation Learning; Robotic hand.

1. Introduction

Robots of all kinds have been present in our lives for the past years. They have become an essential part of our modern lives, whether to replace us when performing dangerous tasks, such as disarming bombs, or to make our lives easier when performing simpler tasks, such as moving objects from place to place. But even though robots are becoming an important part of our daily lives, most of them still perform tasks that are simple for us, although not for a robot. An example of such applications is the filed of dexterous manipulation. While vision-based methods have achieved good results in this field, they can lead to hard to solve problems such as not being able to infer important properties of the manipulated object (e.g. its weight or friction). Additionally, visual information can be hampered by possible occlusions. There is another reason as to why developing an in-hand manipulation capable robot is a difficult task, which is related to the high number of dimensions of the state and actions to consider. The goal with this work is to have a model that can manipulate any object and efficiently learn to perform any given task. As indicated before, using vision-based methods may not be the suitable approach, so instead, it is proposed a model that will mostly rely on tactile information. Additionally, this work intends to explore the possibility of using a condensed representation of the input state in order to provide more useful information that will aid in the manipulation of the objects.

This paper is structured as follows. In Section 2 we briefly introduce the most important Reinforcement Learning (RL) concepts and algorithms used in this work. Section 3 gives a system overview, describes the proposed goals and how they are going to be achieved in more detail. Section 4 describes the main results obtained throughout the experiments conducted. Lastly, Section 5 presents the main conclusions to draw from this work.

2. Background

2.1. RL

The purpose of a RL method is to learn a function that determines what action to take in a given state such that the total expected future reward is maximized. Such a function is called an optimal policy. Achieving the optimal policy requires exploration and experimentation in a given environment.

In more detail, most RL problems can be defined as a Markov Decision Process (MDP) defined by the tuple $(S, A, P, r, \rho_0, \gamma, T)$ where S is a set of states, A is a set of actions, $P : S \times A \times S \rightarrow \mathbb{R}_{\geq 0}$ is the transition probability distribution (that models the changes of states), $r : S \times A$ is the reward function, $\rho_0 : S \rightarrow \mathbb{R}_{\geq 0}$ is the initial state distribution, $\gamma \in [0, 1]$ and T is the horizon. The factor γ is the decaying factor which determines how much an updating step influences the current value of the weights of the networks. The horizon T defines how many steps into the "future" the agents has to take into account.

Most of the RL algorithms optimize a stochastic policy $\pi_\theta : S \times A \rightarrow \mathbb{R}_{\geq 0}$. Let $\eta(\pi)$ denote its expected discounted reward for horizon T, t:

$$\eta(\pi) = \mathbb{E}_\tau[\sum_{t=0}^{T} \gamma^t (r(s_t, a_t)], \tag{1}$$

where $\tau = (s_0, a_0, ...)$ denotes the whole trajectory, $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi(a_t|s_t)$ and $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. Within RL problems, there is the distinction between continuous and discrete space environments. There are different algorithms for these scenarios and the problem at hand is defined as a continuous space environment. The work in [7] provides an extensive review of the state of the art of Deep RL methods used in continuous control. Proximal Policy Optimization (PPO) [13] is a popular Deep RL algorithm which was used throughout this work. It simultaneously optimizes a stochastic policy as well as an approximator to the value function. PPO interleaves the collection of new episodes with policy optimization. After a batch of new transitions is collected, optimization is performed with minibatch stochastic gradient descent to maximize the objective [2]:

$$L_{PPO} = \mathbf{E}\text{min}\Bigg(\frac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)}\hat{A}_t,$$
$$\text{clip}\Big(\frac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\Big)\hat{A}_t\Bigg),$$

where $\frac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)}$ is the ratio of the probability of taking the given action under the current policy π to the probability of taking the same action under the old behavioral policy that was used to generate the data. Here ϵ is a hyperparameter that controls the amount of clipping and $\hat{A}_t$ is the advantage function. This loss encourages the policy to take actions that are better than average (have positive advantage) while clipping discourages bigger changes to the policy by limiting how much can be gained by changing the policy on a particular data point.

2.2. Representation Learning

Representation Learning is a concept where the objective is to extract useful, unlabeled information from the input data. The goal is to train a model that will learn what are the most important features of a given input, to later use that information. Usually, this representation is in a latent space, *i.e.* it is represented with a lower dimension that the original one. There are several ways one could achieve a representation of the input but in this thesis it was only used one: the Autoencoder (AE).

The ability of an AE to condense information and to only map the most important features of the input allows for models to be trained in a different way. More specifically, it can be of use for manipulation-like problems where the input has a high dimension since it allows to reduce that dimension.

2.3. Related Work

One of the most recent works in this field is [2], in which the authors developed a model capable of manipulating a cube to a specified pose, using vision. The model was trained with PPO and it was first conducted in a simulated environment and then transposed to the real environment. A key point in this work that allowed for the model to have a good performance on the real environment, were the randomizations (e.g. physical properties of the object, added noise) introduced in the training period.

Other works [15], [5], [12] focus on using tactile data. The objective in [15] was to move several cylindrical objects from one point to another using a two fingered hand. Y. Chebotar *et al.* [5] also add to this aspect in the sense that the model is able to adapt to the changing environment. A. Rajeswaran *et al.* [12] focused on having a robotic hand that is capable of performing every-day tasks such as opening a door, manipulating a tool, etc. The results showed that this new method was up to 30 times faster than learning from scratch, in some tasks.

Representation Learning became a topic of interest since it provides a way to characterize data in a more efficient way [3]. There are multiple approaches to tackle this problem and using an AE is a common approach. Z. Zhu *et al.* [17] use this method to compute a 3D shape of an object from multiple 2D images of that same object. A more manipulation-oriented work was presented in [14]. By developing a RL algorithm that leverages representations learned from an AE, they were able to model a 5 Degrees of Freedom (DOF) robot to manipulate a pole to a given orientation, while using visual data. M. C. Gemici *et al.* [9] propose a learning algorithm that takes as input tactile sensory signals of a robot obtained while performing actions on the objects (which were food in this cause), to then extract useful features from these signals and map them to physical properties of the objects. Z. Wang [16] uses Representation Learning to infer important object properties, such as its pose, from tactile data. The learned representation is used to train a control policy which is then transferred to a real robot.

2.3.1 Literature Analysis

In conclusion, in-hand manipulation has been a topic of interest for the past few years and there are many possibilities for future directions. Although the works presented in the above sections are successful in what was desired for each work, most of them face some difficulties/limitations. For example O. M. Andrychowicz *et al.* [2] relied on intensive simulation, data and model augmentation to perform a single task and since it uses vision, the model won't most likely adapt to objects of the same dimension, but with different weights or friction coefficients. A. Rajeswaran *et al.* [12] relied on human demonstrations to speed up the training process and to achieve the desired behaviour, thus needing extra hardware. Y. Chebotar *et al.* [5] were able to adapt to the changing environment and to test the model on a real robot, but the tactile sensors used are not feasible for a real-life application since they are too big (16 cm x 16 cm).

3. Methodology

3.1. Architecture Overview

As mentioned before, an important objective is to develop a model capable of in-hand manipulation tasks, in a simulated environment. The first approach is to use the aforementioned Deep RL methods in their standard form, to train a model capable of learning a specific task. This model would take as input a state and will output a certain action. This action will control the movement of the joints in a humanoid hand. Although this approach by itself can lead to good results, it also leads to a few problems, which may influential the learning process and overall performance of the model. Those problems are:

- the high number of DOF
- nonexistent generalization for different objects and for different objects

The problem of the high dimensional action space can be solved by reducing the number of actions the model can take, while maintaining the ability to manipulate objects (using synergies). Regarding the remaining problems, their solution is not so trivial. To solve them, an new approach is proposed. The main concept of this new approach is to have the Deep RL methods learn over a reduced feature set that only contains the relevant information for manipulating an object. The overall architecture for this new approach is presented in Figure 1.

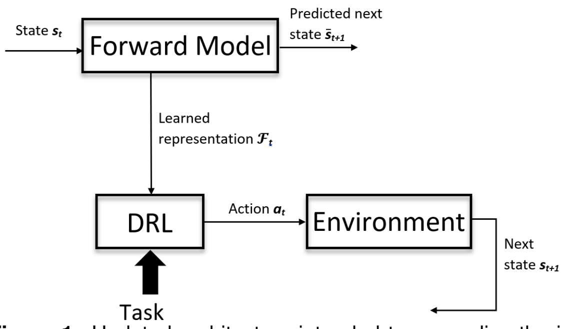

State s_t
Forward Model
Predicted next state $\bar{s}_{t+1}$
Learned representation $\mathcal{F}_t$
DRL
Action a_t
Environment
Next state s_{t+1}
Task

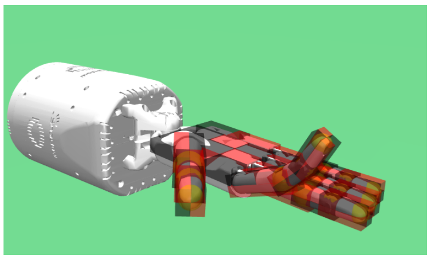
Figure 1: Updated architecture intended to generalize the inputs given to the model.

The difference between the proposed architecture and the common procedure with Deep RL methods, is that now the input is first passed through another model, the Forward Model, which will produce two outputs. The most important one here is the learned representation $\mathcal{F}_t$, which will comprise only the most important features of the state that are relevant for the task at hand. This Forward Model will have an AE-like architecture and it should be trained with a rich dataset (multiple objects and different tasks). Its input will be the state and the output will be a prediction of the next state given a certain action. The training set for this model should comprise data obtained while manipulating different objects and with different tasks. This way, the Forward Model will be able to represent the state, regardless of the type of object or task. Moreover, the Forward Model will be able to "filter" the information contained in the state vector and feed that pre-processed information to the Deep RL model. This new input will allow for:

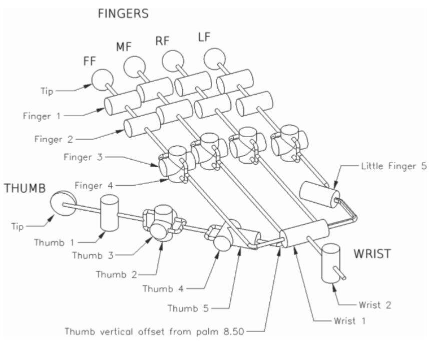
- a faster training process
- robustness to different objects and to different tasks

3.2. Simulator

Deep Learning methods usually need a large amount of both data and time to learn a model so a good approach is to train a certain model in a simulator first, where it is possible to train faster and there is no risk to damage a physical robot. There are many options when it comes to simulating robotic environments and some articles, such as [8] and [11]. Both showed that MuJoCo has one of hightes ratings. A key factor in favor of MuJoCo, is that OpenAI created an interface called OpenAI Gym [4] that allows to easily implement RL methods in their environments and disposes of premade models to be used in the MuJoCo engine. In particular, the OpenAI Gym framework includes a simulated model of the Shadow Dexterous Hand [1] shown in Figure 2(a) which is a viable model to use to train the models proposed. For this reason, the MuJoCo engine was chosen.



(a) Simulated model of the Shadow Hand hand



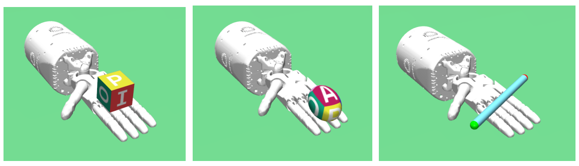
(b) Actuators present in the Shadow Hand. Image taken from [6].

Figure 2: On the left, the simulated model of the Shadow Hand, provided by the OpenAI Gym framework; on the left, the location of all the actuator in the Shadow Dexterous Hand.

As it can be seen from Figure 2(b), the hand has 24 DOF with 4 being passive while 20 are actuators. Another key feature of the simulated model of the Shadow Hand developed by the OpenAI Gym interface is that the hand has 92 built-in tactile sensors which makes this model very suitable for the proposed goals. These simulated tactile sensors provide readings of pressure applied to the sensor and will be used to provide feedback to the Deep RL algorithm. It is also important to define how a certain action will control the model. The action is a vector of values where each component will control a particular actuator from the ones shown in Figure 2(b). An action can be defined as such, $A = \{a_0, a_1, \dots, a_n\}$ where $n+1$ is the dimension of the action vector, which by default is the total number of actuators (20) and where $a_i \in [-1, 1]$.

Another important factor when choosing the right simulator is what kind of data it is possible to extract from the simulator. Apart from the touch sensors, there are other information that are going to be used to train the model, which are available within the simulator. These are: touch sensors values; object's pose (position and orientation), and linear and angular velocity; joint amplitudes and velocities

The touch sensor values are the ones previously presented in Figure 2(a) where each sensor detects a measure of pressure. The object position is measured in meters, the orientation is measured in radians and the velocities are measured in meters per iteration (linear) and radians per iteration (angular). Lastly, the information about the state of the hand is also included, namely the amplitude of each joint, in radians, and the angular velocity of each joint, again in radians per iteration. With this information, one is able to fully define the state of the environment and it is with this information that the model will try to learn the optimal policy. Finally, the OpenAI Gym framework also made available 3 different objects, presented in Figure 3 which will be important when testing the ability of the model to adapt to new objects. These objects will be useful when testing the trained model given different objects and to also gather more diverse data for the Forward Model.



(a) Cube (b) Egg (c) Pen

Figure 3: Objects to be tested on the different models.

3.3. Action Dimension

An important factor when implementing a RL model is to define what and how many actions the agent can take. For the case of the ShadowHand, the algorithm would have to choose 20 different values every time instant, making the process of learning the influence of each actuator, relatively slow. One approach to solve this problem, or at least to facilitate the algorithm when searching for the best policy, is to reduce the number of active actuators. This would effectively make the model learn faster, but it would also mean that the hand would lose DOF, thus losing dexterity when manipulating. So, instead of completely removing a few actuators from the hand, intra-synergies were added. These synergies will allow for some joints to move according to others, much like when someone bends a finger, they bend all three joints at the same time. With these synergies, it is possible to reduce the number of actions needed for the model, with minimal cost to the dexterity of the hand.

These synergies were obtained by trial-and-error, while testing for the ability of the hand to move an object. Several changes were made to the base DOF and they are summarized in Table 1. This table indicates which actuators were used,

Table 1: Updates made to the DOF of the Shadow Hand. In this table, ”F” stands for ”Finger, ”T” for ”Thumb” and ”W” for ”Wrist”. The actuators that are referred in Table 1 are the ones presented in Figure 2(b).

	Actuators											
	F1	F2	F3	F4	Little F5	T1	T2	T3	T4	T5	W1	W2
Used	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	No	No
Parent joint	F2	F4	None		F4 (of the little finger)	T2	None		None	None		
α	1	1	N.A.		0.5	0.7	N.A.		N.A	N.A		

and if they had a parent joint. By saying that a certain joint has a parent joint, it is meant that the joint itself does not move independently, but instead, moves according to its parent joint. The relation between the parent joint and the joint is given by

$$\theta_2 = \alpha \cdot \theta_1, \tag{2}$$

where θ_2 is the joint and θ_1 is the parent joint. α is just a multiplying factor that is also presented in Table 1.

3.4. Controller

The Controller is one of the most important parts of this work, since it will be responsible for actually manipulating the object. This Controller should take into consideration the state of the object and act on it, in order to achieve the desired result. Deep RL methods will be used to train this model, since they are useful when a model has a specific task to fulfil and the action-state space is too large to fully explore it. In order to further define the Controller, it is necessary to identify the input, the output, and the tasks, *i.e.* the state, the actions and the reward functions respectively. The state and actions were already defined before but a small change to the actions was made. Instead of using control, *i.e.* $\theta_t = \beta \cdot t$ where θ_t represent the amplitudes of each joint, a_t is the current action and β is a scaling factor, it was used a relative control where $\Delta\theta = \beta \cdot a_t$ where $\Delta\theta = \theta_t - \theta_{t-1}$.

It is also necessary to define a set of reward functions that will aid the Controller in learning specific tasks. Three main tasks were considered:

- ”random” babbling of the object
- rotate the object a certain amount, in a given axis
- lift the object of the palm of the hand

The last two tasks are more goal-oriented and the first one is a more exploration-oriented task. The first task has the objective to explore the object, while constantly manipulating it, *i.e.* explore different states. This is important in order to collect data for training the Forward Model. The second and third tasks have the purpose of evaluating the controller in dexterous manipulation movements. These were the tasks that were implemented and tested since they cover three different objectives, with three different types of in-hand manipulation. In the next subsections, the reward function for each task will be presented.

3.4.1 Task 1: Task-free object manipulation

The motivation behind this task is related to how babies would learn to manipulate an object. Babies often pick up an object without any particular reason and play with it with no particular objective. The objective of this task is not directly related to evaluating the performance of the Controller, but instead to gather data for the Forward Model. As explained before, it is important the the Forward Model is trained with rich and diverse manipulation data, so if the Controller is performing a task where the objective is to explore as much of the object as possible, the data collected will be as diverse as possible. In a certain way, exploring the object, can be seen as ”inspecting” every side of the object. With vision, this would translate to viewing every side of the object. A way to mimic vision would be to keep track of all the orientations that the object was already in. So, by making use of the simulated environment (which allows the direct access to the object orientation), it is possible to motivate the algorithm to achieve orientations that have yet to be achieved. The reward function for this will evaluate, at every time instant, if the current orientation was already visited or not. If yes, a reward of +1 is given. Moreover, there is a set of key orientations that will award a bigger amount of reward to the model, if said orientation is achieved. The main reward for the model comes from the use of a dictionary that contains most possible orientations. This dictionary does not contain every single possible orientation, as that would be an infinite number. Instead, each orientation has 4 values (since they are expressed in quaternions) and each value was discretized into 11 bins. Additionally, the algorithm is penalized with a reward of -10 every time the object is dropped. With the reward function presented, one is able to manipulate the object by directly exploring different orientations.

3.4.2 Task 2: Rotating the object

Opposed to the previous task, this one has a well defined objective. The goal is to rotate the object a certain amount, around a given axis. Note that the goal orientation is different from the start orientation in a single axis. In this experiment the object starts in the same orientation every time and the goal orientation is chosen by:

$$\theta_g = \text{random } x, x \in |x - \theta_c| > \frac{\pi}{6} \tag{3}$$

where θ_g is the goal orientation and θ_c is the current orientation. By forcing the goal orientation to be at least $\pm$ 30º than the current orientation, it is possible to eliminate cases where the goal orientation is achieved ”by accident”.

Regarding the reward function, a mixed between sparse and dense reward functions was adopted. Similarly to the reward function presented above, the method is both rewarded for achieving some key orientations and for achieving the goal orientation. The difference here is that these key orientations are computed so as to serve as intermediary orientations between the current and goal orientations. They are obtained by computing the orientations that would lead to the goal orientation in the most straightforward way. Since we are only dealing with rotation in one axis these rotations are simple to compute. Although the orientations are originally defined in quaternions, in order to compare the current orientation with the key orientation, they are converted from quaternions to euler angles.

3.4.3 Task 3: Lifting an object

This final task intends to explore the ability of the model to achieve precise grasping. The goal is to lift an object off the palm of the hand, ideally using the tip of the fingers. The reward function for this task is relatively simple and its given by:

$$r = is\_on\_palm() + (h_s - h_i) * 10, \tag{4}$$

where h_s is the starting height of the object, h_i is the current height of the object and the function $is\_on\_palm()$ will reward $\pm$ 1 depending if the object is on the palm of the hand or not (-1 for being in contact with the palm and +1 otherwise). The parameter h_s is defined as a constant, bigger than h_i could ever be. This way the algorithm will be constantly trying to minimize this difference. Additionally, the multiplying factor of 10 is applied so that the ”height” factor of the reward function has the same impact as the ”palm” factor. It is possible to detect if the object is in the palm of the hand by verifying the status of the touch sensors located in the palm. If any of them has a value different than 0 (meaning that some pressure is being applied) then it is considered that the object is on the palm.

3.5. Forward Model

The Forward Model is the main contribution of this work. Its goal is to provide the Controller with valuable information to be used when manipulating an object. This information should be obtained by training the Forward Model with the manipulation dataset, to extract the most useful features from the state vector. In the case of this work, this model will follow a specific architecture, similar to an AE, since it will have an encoder and decoder network. The encoder part will be responsible for encoding the state vector, into a latent representation, which in turn will be decoded to the output vector (the next state). The difference from the Forward Model to the traditional AE, is that the output vector will not be equal to the input vector. This way, it is possible to train the Forward Model to learn to predict the next state (given an action), and to compute a representation of the current state, at the same time. The architecture used is depicted in Figure 4.

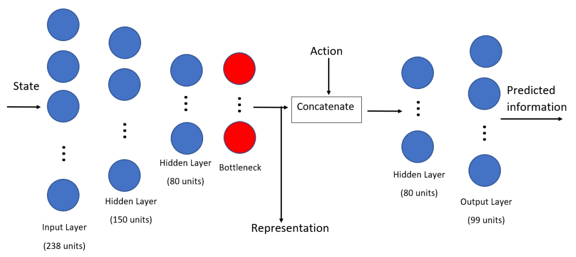


Figure 4: Detailed architecture, with number of layers and units per layer of the Forward Model used.

There are five main elements in Figure 4: two input signals (the state and action) and three output signals (the latent representation, the predicted touch values and the predicted object pose for the next time instant). In this case, the action vector is only taken into account for the decoder network. An important aspect of every model, is to have an adequate loss function. This function is what will dictate how the model will learn to perform a certain task, by evaluating the difference between the predicted output and the true output. Due to the distinct nature of this problem, a custom loss was defined:

$$L = L_{pos} + \beta * L_{rot} + L_{sensors}. \tag{5}$$

In (5), L_{pos} is related to the difference between the predicted and true object position, L_{rot} is related difference in predicted and true object orientation and the same for $L_{sensors}$, considering the true and predicted values of the touch sensors. The factor β acts as a scalling term, in order to give more importance to certain losses. In this case, since the prediction of orientation is of substantial importance, β was set to 10. The loss L_{pos} is relatively simple and it is given by the Mean Squared Error metric. Regarding L_{rot}, the loss is given by the norm of the Mean Absolute Error (MAE) between the true and predicted orientations. However, since we are dealing with quaternions, and the quaternion q originates the same orientation as the quaternion $-q$, a small modification was made.

$$L_{rot} = \min(\mathsf{MAE}(q - \hat{q}), \mathsf{MAE}(q + \hat{q})) \tag{6}$$

where q is the true quaternion and $\hat{q}$ is the predicted quaternion. Lastly, regarding $L_{sensors}$, considering the type of data we have (where the data points for each entry are mostly zeros), standard loss functions cannot be applied. In order to bypass this problem, the following loss function was implemented:

$$L_{sensors} = \begin{cases} \beta 2 \cdot (\hat{y} - y), & \text{if } (y > 0 \wedge \hat{y} < 1e^{-3}) \\ 0, & \text{else} \end{cases} + \begin{cases} \beta 10 \cdot \hat{y}, & \text{if } (y == 0 \wedge \hat{y} > 1e^{-1}) \\ 0, & \text{else} \end{cases}, \tag{7}$$

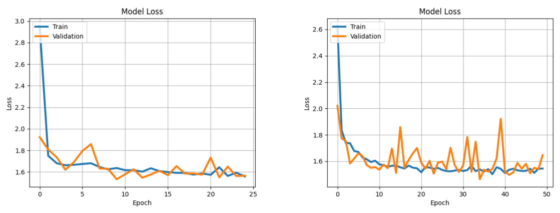
where y are the true touch sensors values, $\hat{y}$ are the predicted ones, and β is the same as the one in (5). This function computes a vector (with the dimension equal to y and $\hat{y}$) where each value indicates the error for the correspondent sensor. If the true sensor is active but the predicted one is not, then that value is given by the first term of the loss in (7). On the other hand, if the true sensor is not active but the predicted one is, then the value is given by the second term of the equation.

4. Results & discussion

So far, all the methodologies that were used in order to achieve the proposed goals have been described. It was presented a method for evaluating the ability of the Controller to perform certain tasks, and a method for evaluating if by using a Forward Model, the performance of the Controller will increase or not. In this, section the main results from each experiment are going to be discussed.

4.1. Train and test Forward Model

The Forward Model was designed to help the Controller in making more informed decisions regarding the manipulation of a set of objects. The objective is to have the Forward Model filter the raw observation obtained from the environment and output a new input to the Controller, that will only consist of relevant information for manipulating all objects. All the layers are fully connected, with the ReLU activation function, except for the bottleneck layer, which used the *tanh* activation function. The data to train the Forward Model was obtained by collecting information on every iteration throughout training of the Controller, using Task 1. In order to obtain the best Forward Model, different models were trained and tested in the test dataset where two different parameters were changed: the batch size and the dimension of the latent representation (number of units of the bottleneck layer). An example of two different models is presented in Figure 5.



(a) Loss for Forward Model with a representation of 5 units.
(b) Loss for Forward Model with a representation of 40 units.

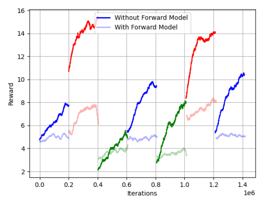
Figure 5: Loss for two different models, trained with a batchsize of 256.

There are two main observations from the results presented in Figure 5. The first is that the loss decreases with training as would be expected, indicating the model is learning. The second observation is that there seems to be no difference, in terms of loss, in using a representation of 5 units versus using one of 40 units. Theoretically, the representation with 40 units would be able to more accurately predict the information since it has more "space" to code the information into, but this was not verified. A possible reason is that the model was not able to learn how to predict the values regarding the touch sensors. Upon closer observations of the results it was found that the loss for the sensors was mostly constant during training, and the only thing the model learned was to correctly predict the orientation and position, with an average error of 17º and 15 millimeters, respectively. This suggests that the model is not capable of consistently predicting the touch values, which could have hindered the learning process for the remaining losses, thus explaining the similar results between all models.

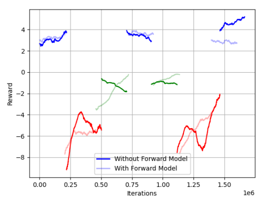
Nonetheless, from this point on, a specific model needed to be chosen, so it was opted to use the model that was trained with batch size 256 and had 40 units for the representation. This was the chosen model, since as stated before, a bigger representation dimension is often related to more accurate representations.

4.2. Controller versus Forward Model

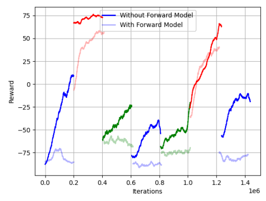
The goal with this experiment is to evaluate the difference in performance between using a model that learns without the state representation, and one that does. To evaluate the performance of both, the models were trained with multiple objects at the same time. Instead of just using a single object during training, the object is changed mid training, forcing the model to adapt to the new object. The training will start off with one object, then change to another and so on, going through the objects depicted in Figure 3. The order in which each object appear is fixed and is as follows: Cube → Egg → Pen → Cube → Pen → Egg → Cube. With this order, it is possible to test every possible transition between two different objects. For this experiment, a different model was trained for each object with 200000 iterations. Each task was trained 4 different times, each time with a different seed, to minimize outliers. First we will analyse the training process for each task, and after the test results.



(a) Average reward during training for Task 1.



(b) Average reward during training for Task 2.



(c) Average reward during training for Task 3.

Figure 6: Reward during training for all three tasks for the two different models. Blue lines for the Cube, red for the Egg and green for the Pen.

The results after training for each task are shown in Figure 6. It is important to note that each line in Figure 6 represents a different model, and different colors represent different objects. Additionally, each model's starting point, is the previous model's ending point. So this can be seen as one big individual model that was divided into 7 different models (one for each object used in the aforementioned order). Each of these lines will also be referred to as training periods.

The first observation is that both models were not able to properly learn to perform Task 2. The small increase in reward for the different training periods indicates minimal learning. The second conclusion is that the models using the raw Controller tend to have a higher average reward. This is mostly visible in Tasks 1 and 3 (Figures 6(a) and 6(c) respectively). However, this does not necessarily imply that the Forward Model failed to fulfil its purpose. It is normal that the raw Controller outperforms the models using the Forward Model, because in this case the Controller is only training with one object at a time, while the Forward Model was designed to be object-agnostic.

In the previous observations, the models using the Forward Model were being compared to a model specialized in learning a particular task for a given object. This is an unfair comparison, since the purpose of the Forward Model is not to be better than the raw Controller in these cases. The main goal of the Forward Model is to provide more stable object manipulation, when using previously unseen objects. In order to test this, each trained model was tested with each object, using Task 1, using a specific key metric. This particular key metric is defined as the number of different orientations that are achieved by the models. The results are presented in Table 2. For each object, two models were tested: the model using the Forward Model (represented by "FM + C") and the model using the raw Controller (represented by "C")

From the overall results, there are two main observations. The first, which was also observed in the Figures 6(a), 6(b) and 6(c), is that the models using the raw Controller are able to achieve a higher average reward for almost every combination of model-object than the models using the Forward Model. However, it is also possible to observe that the models using the Forward Model consistently achieve a lower standard deviation. This leads to the conclusion that although the average performance is lower, the reproducibility and reliability of the model is better when using the Forward Model. Another important observation concerns the models with which the Pen is tested. It is possible to confirm that for almost every model, the models where the Forward Model was used have a higher reward than the ones where it was not used. This increase in performance is once again related to the ability of the model using the Forward Model to be more versatile. This is an important result since the Pen is a different object than the Cube or the Egg, which means that if other objects with unique shapes were tested, the models using the Forward Model would most likely perform better. However, this is not true for the cases the model being tested was also trained with the Pen (lines 3 and 5) but the reason for this was explained before. The results for Tasks 2 and 3 are not shown here since they do not add anything to what was previously concluded.

To conclude this experiment, a last table was obtained. Table 3 is the summary of the results above

Table 2: Results for the different models tested with all objects for Task 1: task-free object manipulation. The colored cells represent the cases where the object tested is the same as the object that the model was trained with.

Models \ Object	Cube		Egg		Pen	
	FM + C	**C**	**FM + C**	**C**	**FM + C**	**C**
1 - Cube	5.43±2.72	7.49±3.75	9.22±3.11	13.47±3.52	3.71±1.81	3.23±1.61
2 - Egg	4.48±2.58	6.35±3.96	10.93±3.96	16.34±4.43	3.49±1.73	3.21±1.54
3 - Pen	4.98±2.71	5.46±2.73	9.12±3.18	9.88±3.28	4.01±2.27	5.09±2.79
4 - Cube	5.30±2.75	9.17±3.80	9.33±3.10	13.34±4.04	3.78±1.94	3.74±1.76
5 - Pen	5.14±2.74	6.64±3.01	9.19±3.47	11.19±3.56	4.03±2.19	8.40±4.91
6 - Egg	4.59±2.50	4.49±2.74	10.53±3.98	14.79±4.69	3.55±1.84	2.98±2.37
7 - Cube	5.50±2.76	10.45±4.78	8.51±2.64	14.82±4.22	3.87±2.03	5.11±3.54

(including the results from Tasks 2 and 3) where each value is the average reward for a given Task, using each of the objects. For instance, the first value is the average of the rewards of all the models using the Forward Model, that were tested with the Cube and using Task 1. It is once again clear that although the models using the Forward Model have a lower reward among all tasks, the standard deviation has the opposite effect.

5. Conclusions

There were two main goals for this thesis. The first was related to developing a model, the Controller, capable of using tactile feedback to manipulate a set of objects, based on a set of different tasks. The second goal was to understand if using a representation of the state instead of the actual state would lead to an increase in performance and/or a faster learning/adaptation when training with never before seen objects.

Regarding the first objective, the main conclusion is that the Controller was able to learn the proposed tasks, to some degree, except Task 2. Regarding Task 1, the Controller learned but the maximum number of orientations achieved in a single episode were roughly half of the total possible orientations, indicating that the model could have performed better. With Task 3, it is safe to assume that the model learned correctly, but with Task 2, the conclusion is the opposite, the model did not learn properly.

Regarding the second goal, a main experiment (Section 4.2) was conducted. First we conclude that the model using the Controller is able to perform better (reward-wise) than the model using the Forward Model. This can be explained by the fact that the Controller specializes in performing a particular task, given a certain object, *i.e.* it is overfitting, while the Forward model specializes in having a more general control that is applicable for all objects. Despite having a lower average reward, the models using the Forward Model have, on average, a smaller standard deviation. This leads to the conclusion that the Forward Model is able to provide a more reproducible and reliable control. Another important conclusion is regarding the specific case where the Pen is used. The fact that the performance of the Forward Model, when using the Pen is, most of the times, is equal or even greater that the performance with the Controller, leads to the fact that the Forward Model could provide a better result when using object with unique shapes. Moreover, it leads to conclude that the Forward Model was indeed able to generalize to different objects to some degree, achieving the proposed objective of having a more general controller to different objects.

It is important to note that using Deep RL methods to solve a given problem, although providing state of the art results in various fields, are not a very reliable method. The work in [10] concludes that a small variance in the hyperparameters or in the architecture of the networks can lead to significantly different results. Moreover, the hand/object interaction is a non-trivial relation, considering the high dimension of the state vector and actions. However, it was shown that is it possible to manipulate a set of object, based on tactile feedback, using Deep RL methods.

5.1. Future work

Regarding future work, an important improvement is to design a better Forward Model. As it was demonstrated, this Forward Model had some troubles with predicting the next state, specially regarding the touch sensors information. This may have lead to undesired results and a possible solution would be to further adapt the loss function or the architecture of the model itself. Moreover, PPO is an Actor-Critc algorithm, meaning that it trains 2 networks simultaneously: one to choose the best action (actor) and another to evaluate the expected reward from that action (critic). A possible improvement could be to integrate the Forward Model with the Deep RL algorithm itself, more specifically, with the critic network. This would provide the Deep RL algorithm a better approximation to future states.

Lastly, another future improvement could be re-

Table 3: Average rewards for all the models for the different tasks and objects.

	Cube		Egg		Pen	
	FM + C	**C**	**FM + C**	**C**	**FM + C**	**C**
Task 1	5.06±0.4	7.15±2.08	9.55±0.86	13.4±2.23	3.78±0.21	4.54±1.92
Task 2	0.16±0.03	0.19±0.05	0.01±0.01	0.02±0.02	0.00±0.00	0.00±0.00
Task 3	0.11±0.06	0.23±0.18	0.59±0.2	0.64±0.24	0.23±0.05	0.24±0.07

lated to the actual setup of the experiments, *i.e.* the objects and tasks used. Regarding the objects, the Forward Model could have been even more general if other objects were added or even if the weight of the current object was changed, which was not tested in this thesis. Regarding the tasks, the reward functions could be improved, specially regarding tasks 1 and 2, and even more specifically to Task 2.

References

[1] *ShadowRobot Dexterous Hand*, 2005.

[2] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[3] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Y. Chebotar, O. Kroemer, and J. Peters. Learning robot tactile sensing for object manipulation. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3368–3375. IEEE, 2014.

[6] G. Cotugno, K. Althoefer, and T. Nanayakkara. The role of the thumb: Study of finger motion in grasping and reachability space in human and robotic hands. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(7):1061–1070, 2016.

[7] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[8] T. Erez, Y. Tassa, and E. Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015.

[9] M. C. Gemici and A. Saxena. Learning haptic representation for manipulating deformable food objects. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 638–645. IEEE, 2014.

[10] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[11] S. Ivaldi, V. Padois, and F. Nori. Tools for dynamics simulation of robots: a survey based on user feedback. *arXiv preprint arXiv:1402.7050*, 2014.

[12] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[14] H. Van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3928–3934. IEEE, 2016.

[15] H. Van Hoof, T. Hermans, G. Neumann, and J. Peters. Learning robot in-hand manipulation with tactile features. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 121–127. IEEE, 2015.

[16] Z. Wang. Representation learning for tactile manipulation. 2018.

[17] Z. Zhu, X. Wang, S. Bai, C. Yao, and X. Bai. Deep learning representation using autoencoder for 3d shape retrieval. *Neurocomputing*, 204:41–50, 2016.