LEARNING ROBOT MANIPULATION TASKS VIA OBSERVATION

by

MICHAIL THEOFANIDIS

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington, in the context of the Ph.D. program,
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2019

To my family and friends for always being by my side.

ACKNOWLEDGEMENTS

Firstly I would like to express my sincerest gratitude to my supervisor Prof. Fillia Makedon for believing in me and for her continuous and unconditional support in my research. Her intuition and guidance have been invaluable towards completing my Ph.D.

In addition, I would like to thank my colleagues and labmates at the Heracleia Lab, UTA for the numerous fruitful conversations and collaborations we had over the years. They all played a special role in the successful completion of this Thesis. Special thanks go to my dear friends Alexandros Lioulemes, Joe Cloud and Maher Abujelala for their constant support and mentoring. Their knowledge has been an indispensable factor towards improving my research skills and becoming a better scientist. Moreover, I would like to thank the committee members of this thesis and professors at UT Arlington for their valuable research insights during my Ph.D.

Lastly, I would like to thank my parents Stavros and Olga, as well as my two siblings Faidon and Sofia for being the greatest source of emotional support and encouragement I could ever ask for.

November 26, 2019

ABSTRACT

LEARNING ROBOT MANIPULATION TASKS VIA OBSERVATION

MICHAIL THEOFANIDIS, Ph.D.

The University of Texas at Arlington, 2019

Supervising Professor: Fillia Makedon

The coexistence of humans and robots has been the aspiration of many scientific endeavors in the past century. Most anthropomorphic or industrial robots are highly articulated and complex machines, which are designed to carry out tasks that often involve the manipulation of physical objects. Traditionally, robots learn how to perform such tasks with the aid of a human programmer or operator. In this regard, the human acts as a teacher who provides a demonstration of a task. From the data of the demonstration, the robot must learn a state-action mapping that accomplishes the task. This state-action mapping is often addressed in the literature as a *policy* [1]. A common strategy for the acquisition of robot motor policies for a task is often achieved with Learning from Demonstration (LfD) algorithms [1].

Initial attempts to create LfD methods relied purely on supervised learning algorithms [2], while most modern paradigms rely on Reinforcement Learning (RL). This phenomenon indicates a shift from supervised learning to goal-oriented algorithms [3]. The development of the Dynamic Movement Primitive (DMP) framework [4] was an essential contribution to this trend, as it provides an abstraction layer between the dimensions of state, action, and environment by computing a policy with distinct

meta-parameters that affect the behavior of the robot [5]. One of the advantages of the DMP framework is its ability to learn motor policies by transforming motion trajectories (high-dimensional space) to specific motion features (low-dimensional latent space) via regression.

The DMP framework learns policies that lie in the trajectory level. However, humans and other animals are capable of learning new behaviors simply by observation. Robots need to achieve the same performance even if there is a substantial domain shift in the environment, embodiment, and perspective between the robot and the teacher [6]. Most modern large deep neural network models can enable complex motor skill representation across different embodiments. As such, we propose a method to learn end-to-end visuomotor policies for robotic arms from demonstrations. The method computes state-action mappings in a supervised learning manner from raw images and motor commands. At the core of the system, a Convolutional Neural Network (CNN) extracts image features and produces motion features. The motion features encode and reproduce motor commands according to the Dynamic Movement Primitives (DMP) framework.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

Introduction

As technology becomes more important in manufacturing and domestic applications, creating easily re-programmable machines has become increasingly necessary. During the past century, many advances in the field of Human-Computer Interaction (HCI) enabled users to perform complex operations with computers, without any knowledge of their internal operation. Compared to computers, robots are not as easily programmed because they can move in the real world. This phenomenon makes the creation of user-friendly robotic applications a challenging task. In the following section, we briefly discuss the evolution of robot programming and provide an overview of modern imitation learning paradigms.

1.1 Evolution of Robot Programming

1.1.1 Early Robot Programming

Robot programming traces back to the 1980s with manufacturing robotics [11], as a response to the rapidly increasing number of industrial robots [12]. Early robot programming relied on manual teleoperation control through various interface devices. Initial research focused on the design and creation of safe and easy-to-use interfaces that enabled robotic arms to perform simple manual tasks. Common technologies for robot operation include control devices, 2D and 3D Graphical User Interfaces (GUIs), Virtual Reality (VR) interfaces [13], mechanical trackers and exoskeletons [14]. Figure 1.1 provides an illustration of early robot programming concepts. One of the key problems that researchers tried to address is the creation of interfaces for operators

that have no prior knowledge of robotics. This endeavor entailed the decoupling of the kinematic complexity of the robot from it's programming [8]. Thus, human-robot interfaces should have a wide range of communicative modalities that allow a human operator to control multiple Degrees of Freedom (DoFs) while visualizing the work-space of the robot and tracking the robot's surroundings.

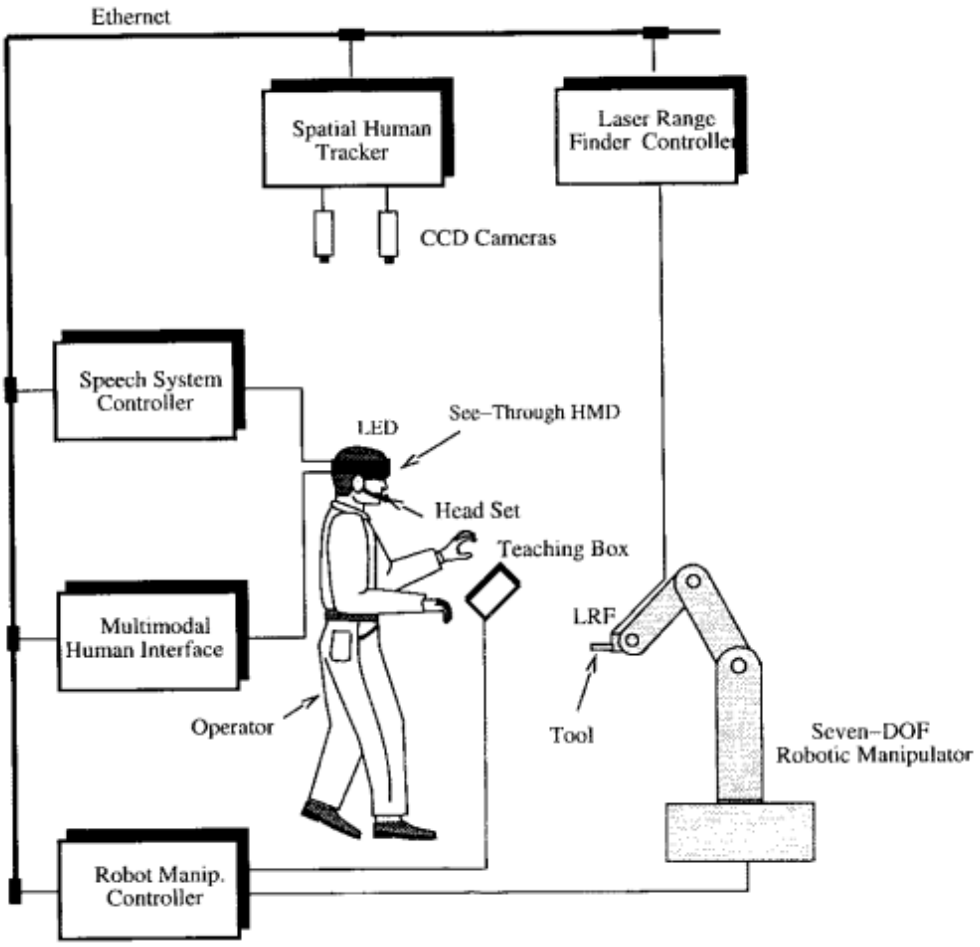


Figure 1.1. Diagram depicting multi-modal teleoperation architecture for robotic arms used in manufacturing scenarios [7].

By utilizing one or multiple of the previously mentioned technologies, human operators were able to demonstrate how a robot should perform a task. During

the demonstration, the system captures sensorimotor information. As sensorimotor information, we define variables such as the position of the robot and the location of the object [2]. This information is then stored in a database and used by the robot to replicate the task. By performing multiple demonstrations, robot operators could created a battery of different point-to-point movements that could be played back to perform different tasks.

1.1.2 From Programming to Imitation

As robot programming became a topic of interest in the field of HRI, researchers developed the idea that robots should learn how to perform a task by human guidance. This idea was indirectly influenced by the human tendency to learn practical skills from other humans by imitation. Eventually, as HRI became more human-centric and bio-inspired, the term robot programming was replaced with learning from imitation [11]. This shift raised questions about how to predict, encode and recognize motion [2]. Moreover, just replicating demonstrations is not adequate, because playing back a trajectory does not guarantee the completion of a task in a different environment or by a dissimilar robot. To address how to transfer motor-skills across different robotic agents and novel environments, researchers believed that from the demonstration data we must find which features generalize the knowledge of the task.

Motion features are often addressed in the literature as motion primitives [15]. Motor primitives can be used as building blocks to create new trajectories and encode demonstrations. This trend prompted the creation of many imitation frameworks that enable robots to learn motor skills. Figure 1.2 provides a graph that describes a generalized pattern of imitation learning frameworks [11]. Initially, a demonstration of the motor skill must be presented by a teacher. Then the system must extract and pass the features of the motor skill to the controller. The controller reproduces new

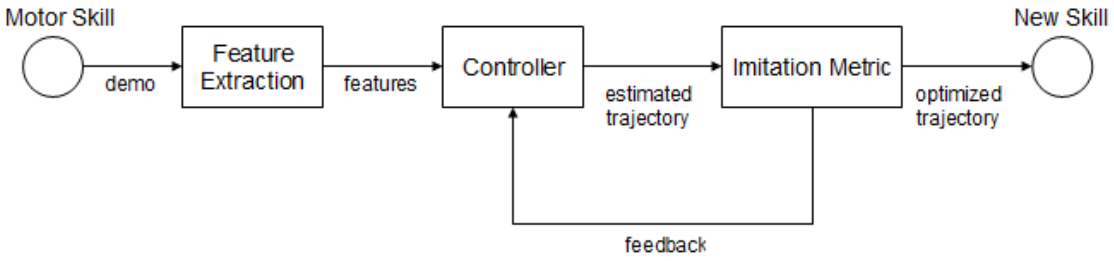


Figure 1.2. Generalized pattern of imitation learning frameworks showing how acquired skills are turned into learned skills.

motor trajectories by recombining these features according to an imitation evaluation metric [2]. Once this optimization process is complete, the robotic agent is able to reproduce a new motor trajectory that performs the task.

Another way to analyze Figure 1.2 is to assume that the controller creates a mapping between the state of the robot environment and the actions of the robot. This mapping is often addressed as a *policy* in the respective literature [1]. From this perspective, imitation methods create policies from examples of state-action mappings, which are given in the form of task demonstrations. During a demonstration, information must be obtained about the state of the robot, the state of the environment, and the action that the robot must perform. From this data, the controller creates a policy that dictates the actions of the robot with respect to the state of the environment.

1.2 A Taxonomy for Imitation Learning methods

Ever since the establishment of imitation learning as a topic of research in the field of HRI, researchers have been trying to answer the questions *what to imitate*, *how to imitate* and *who to imitate* [2]. The first question asks what features should be selected to represent motor skills. The second seeks to answer which strategy

should be implemented by the controller; the last question regards the collection of the data from the demonstration. Encoding, generalizing, and reproducing motor skills remains an open problem in imitation learning. Moreover, the mathematical formulation of the state and action dimensions varies according to the mechanical structure of the robot and task that the robot must learn. Note that the feature selection method, data collection process, and policy formulation algorithm are closely associated. Thus, a suitable metric to categorize imitation methods is according to feature selection, policy derivation, and data collection.

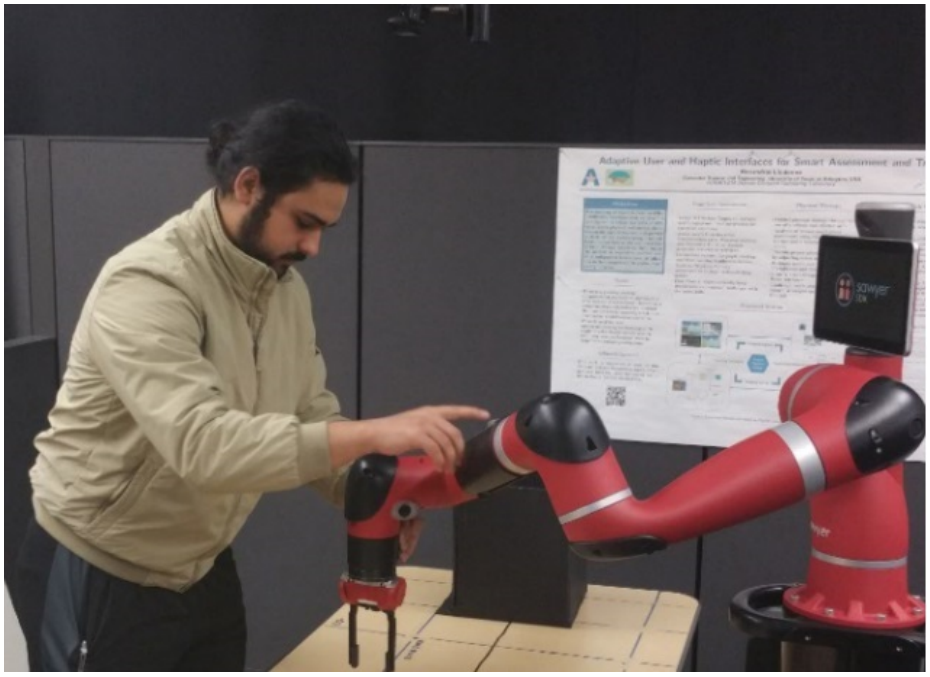


Figure 1.3. A human operator performs a demonstration with kinesthetic teaching.

1.2.1 Data collection

Despite their numerous differentiation, all imitation learning methods begin with the data collection phase, which aims to create a dataset for learning a motor

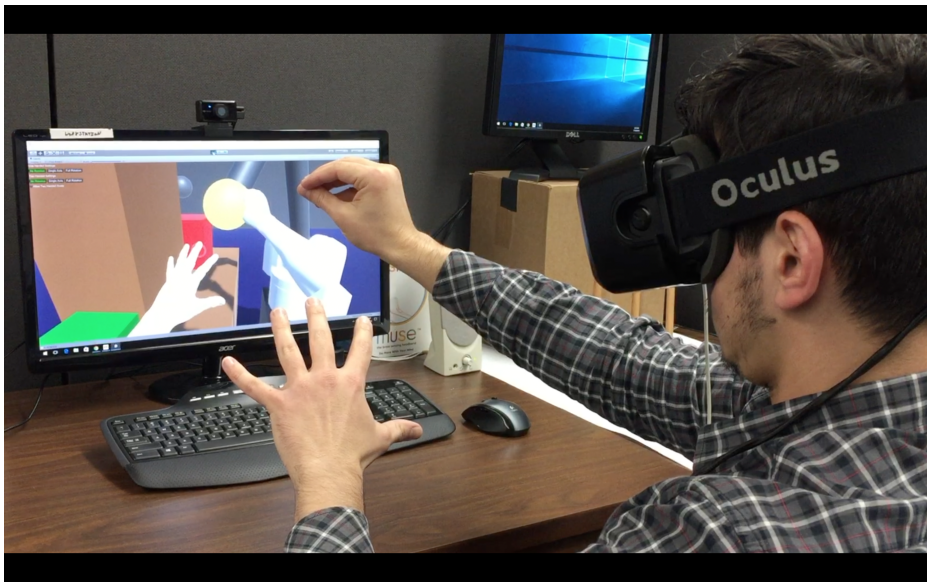


Figure 1.4. A human operator teaches a robot in VR simulation. Figure reproduced from [8].

skill. This dataset may contain more than one demonstration per motor skill, as some imitation algorithms require multiple examples to build a generalized and adaptable model of the motor skill. The purpose of this dataset is to extract state-action pairs and features to create a policy. One key issue that regards the creation of this dataset is if the state-action dimension of the teacher and the learner is identical. Note that researchers refer to this problem as the problem of *correspondence*. This criterion can be used to classify robot programming techniques into two categories [1]. The first category requires datasets were the learner can use the state-action pair of the teacher directly. Creating such a dataset is possible with teleoperation or kinesthetic teaching, as seen in figure 1.3. On the other hand, in the second category, the teacher's state-action mapping is indirectly recorded, such as the case of a robot learning to perform tasks through videos [16].

1.2.2 Feature selection

Imitation learning datasets contain information about trajectories that describe how to execute a motor skill. Initial attempts of feature selection in robot trajectories came from the realization that storing time and position vectors for every task is ineffective in terms of storage. To overcome this problem, a third or fifth-order polynomial with parabolic blends can be used to represent a robot trajectory. Effectively, this reduces the amount of stored data since the entire trajectory can be reproduced just by storing the parameters of the polynomial. In this case, the parameters of the polynomial represent the features that encapsulate motion. However, motion primitives can describe higher-level concepts. It is possible to assume an entire point-to-point movement as a primitive by labeling it with a symbolic relation such as "in contact", "close to" or "grasp-object" [2]. Note that in the first example, the feature extraction process took place at the *trajectory level* and in the second example, it occurred at the *symbolic level* [2]. Imitation learning algorithms at the symbolic level often utilize entire point-to-point movements as features, while learning approaches at the trajectory level deal with continuous signals.

1.2.3 Policy derivation

Recent literature [1] suggests that both AI and engineering approaches can be used for policy derivation. Symbolic approaches that deal with sequences of predefined motions, often rely on traditional graph-based approaches to recombine different sets of point-to-point motions to construct different behaviors. Deriving policies at the trajectory level is complicated, since algorithms of this category deal with continuous spaces. Methods that derive policies in a continuous space often utilize machine learning, mathematical models or a combination of both. In this regard, classical

control theory approaches estimates policies with the use of mathematical models. However, the problem of such models is their lack of generalization. Initial attempts to create generalized motor skill policies involved supervised learning algorithms, such as Gaussian Mixture Models (GMMs) [2]. As a general rule, most trajectory level learning methods perform regression with supervised learning machine learning algorithms to create generalized state-action mappings from state-action pairs that emerge from the demonstration dataset. However, imitation learning algorithms of this category can adapt and evolve with Reinforcement Learning. Algorithms that learn motor policies in continuous spaces by self-exploration are considered a different subcategory with its own separate literature [17].

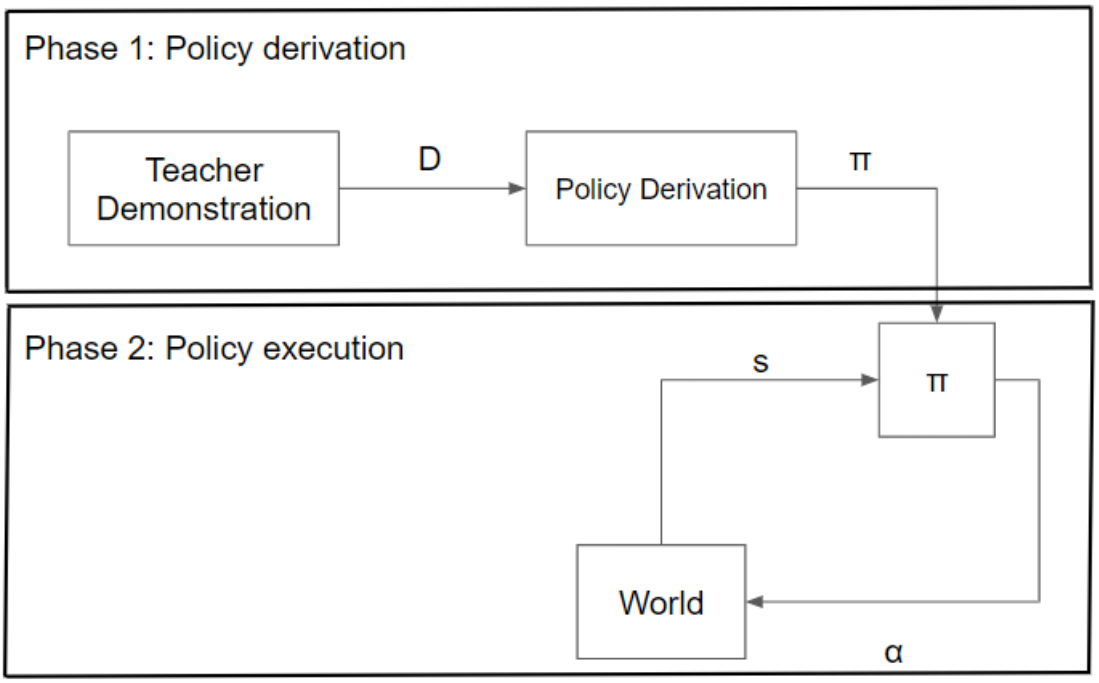


Figure 1.5. Learning from Demonstration framework depicting the multi-phase system in which a policy is derived and later executed. Figure reproduced from [1].

1.3 Learning from Demonstration

In the previous sections, we provided an overview and taxonomy of imitation learning algorithms. In this section, we will discuss a special case of imitation learning algorithms, which are addressed in the respective literature as Learning from Demonstration (LfD) algorithms [1] [2]. Figure 1.5 illustrates the LfD framework. LfD algorithms operate in two phases [1]; the policy derivation phase and the policy execution stage. In the demonstration stage, a teacher performs a demonstration of a task. While the demonstration is conducted, the LfD system records demonstration data D that associate with the state and actions of the teacher. The learner then forms a policy that accomplishes the demonstrated task. In the policy execution phase, the robot applies the policy to perform the task. While the learner interacts with the work, it has to pick actions α when it is at state s with respect to the world.

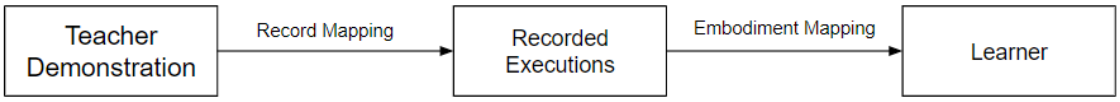


Figure 1.6. State-action mappings between teacher and learner showing the sequence of mappings between the original demonstration and the learner reproducing the movements [1].

1.4 Classification of LfD methods

LfD methods require datasets of state-action pairs to learn a policy. As expressed in Figure 1.5 the acquisition of such a dataset is possible with the assistance of a teacher. However, the learner must be able to use the state-action pair that the system recorded from the teacher. Finding a mapping between the state-action

dimension of the learner and the state-action dimension teacher is important because it enables the learner to learn a policy from the demonstrations of the teacher. This mapping is often addressed as the *correspondence* problem [1], [2]. As figure 1.6 illustrates, the correspondence problem can be defined by two independent sub-mappings, the *record mapping* and the *embodiment mappings*. The record mapping transfers the data that were collected during the policy derivation phase and creates a dataset of recorded executions, while the embodiment mapping adjusts the dimensions of the stored state-action pairs into the state-action dimensions of the learner.

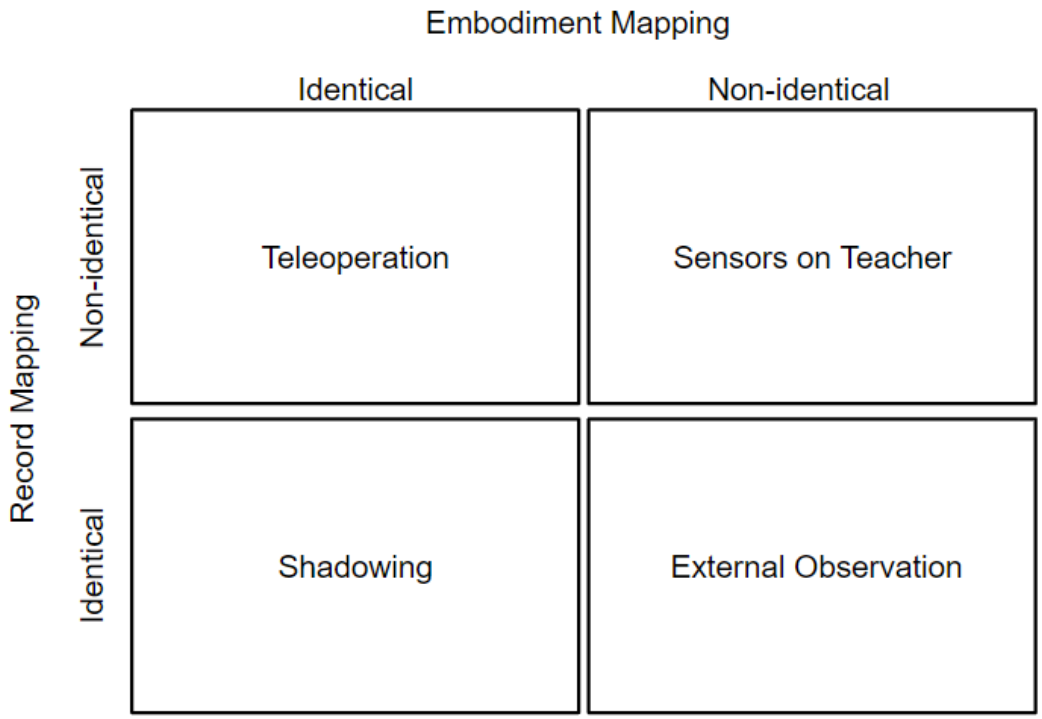


Figure 1.7. Categories of LfD methods [1].

Moreover, if the state-action pairs in the recorded execution are identical to the states and actions of the teacher during the demonstration, we consider the recording

mapping as *identical* or as *nonidentical* if additional transformations are required. Similarly, if the state-action pairs in the recorded execution dataset match the state and action dimensions of the learner, the embodiment mapping is *identical* and *non-identical* otherwise. This phenomenon defines a metric that categorizes LfD methods [1] [1]. When the recording mapping is nonidentical, the demonstration dataset contains encoded information of the teacher demonstration. Also, when the embodiment mapping in nonidentical, the learner must first decode the information which is stored in the recorded execution. This implies that LfD methods that require nonidentical mappings to solve the problem of correspondence have higher complexity. Figure 1.7 presents the four categories of LfD methods. On the horizontal axis LfD methods are categorized according to the type of embodiment mapping, while on the vertical axis, they are classified according to the type of record mapping. As previously stated, both mappings can be either identical or nonidentical. The simplest category of LfD algorithms is *Teleoperation*, followed by *Sensors of Teachers*, *Shadowing* and finally *External Observation*, which is the most complicated case of LfD algorithms.

1.4.1 Teleoperation

Teleoperation LfD algorithms require a human operator to directly program a robot to perform a task. During the demonstration of the task, the system records data with the aid of the robot's internal sensors. In this category, the robot directly records the states and actions that perform the task. Teleoperation is the simplest method of LfD because the state-action pairs of the demonstration are fully observable and accessible by the robot. Therefore, the recorded data does not require any further transformation to be used by the robot that learns the policy. Traditionally, teleoperation approaches are ideal when a human teaches a robot how to learn a simple task. This is the case for example when a robot has to learn a repeatable pick and

place task. Applications such as those are common in industrial plants, where factory workers have to teach manufacturing robots how to move objects in an automation pipeline. Kinesthetic teaching and teaching a robot how to perform a task with the aid of a joy-stick belong in this category of LfD algorithms.

1.4.2 Sensors on Teacher

LfD algorithms of this class use sensor information which directly records the state and action of the teacher into a dataset. This means that no additional transformations are required during the record mapping. However, that is not the case during the embodiment mapping, as the recorded data must be transformed to be used by the learner. This category of LfD algorithms is usually employed when the morphology of the teacher and learner are vastly different. For example, when a human teacher places sensors on their body when executing a demonstration of the task and a robot it called to learn the task with the recorded data [15]. A similar scenario that would require the employment of this LfD method is when a master robot teaches a slave robot how to perform a task and the two robots have different DoFs.

1.4.3 Shadowing

In Shadowing LfD methods, during the record mapping the state-action pairs of the teacher's execution are not directly recorded. They are first encoded before transferring to the learner in the embodiment mapping. The data from the teacher's demonstration, which are stored into the recorded execution dataset, are an encodement or a "shadow" of the true state-action pair of the teacher. However, since the embodiment mapping is identical, the encoded information in the dataset can be directly used by the learner. As such, the learner learns from the "shadow" of the

teacher and not by the teacher directly. When compared to teleoperation LfD methods, shadow LfD methods are algorithmically complex. An example of shadow LfD method can be seen in [9], where a humanoid robot learns how to perform specific arm gestures by observing a human teacher through a camera. Figure 1.8 describes how the LfD system of [9] operates. The pipeline begins with a human that performs a demonstration of a gesture. A camera records the demonstration and provides a stream of images to a neural network, which classifies the images into a specific set of distinct gestures. Lastly, the gestures dataset assigns a specific robot pose for each gesture class.

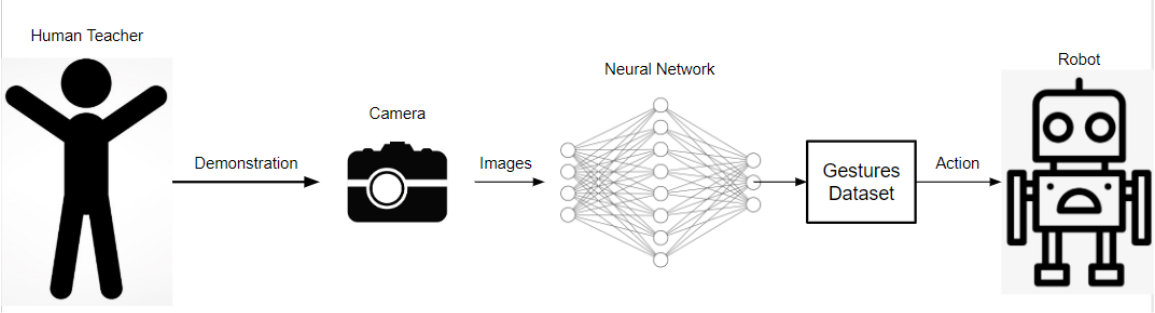


Figure 1.8. Example of a Shadowing LfD method depicting a human performing a demonstration in front of a camera which is later encoded by a neural network into robot gestures [9].

1.4.4 External Observation

LfD methods that rely on external observations assume that both the robot and the recording method can partially observe the true state-action dimensions of the teacher. Since the teacher's information is inferred through various transformations, this category of LfD methods is characterized by the highest complexity and uncertainty. Traditionally, in the case of external observation the sensor which is used to

record the demonstration of the teacher is vision-based, which means in this category the learning component of the LfD system computes visuomotor policies [18], [19]. Examples of this category is a robotic arm that learns pole balancing via stereo-vision and human demonstration [20], a robot that learns to play air hockey [21], and a robotic arm that learns pushing/reaching task policies by utilizing a Deep Convolutional Neural Network [19].

CHAPTER 2

Dynamic Movement Primitives

Researchers in many scientific areas such as physics, robotics, neuroscience and biology have attempted to model complex behaviors by designing nonlinear dynamic systems. The Dynamics Movement Primitives (DMP) framework models the behavior of goal-oriented attractor and rhythmic systems by utilizing simple regression algorithms. The essence of the DMP framework is that a simple dynamical system described by a set of linear differential equations can be encoded and reproduced by learning the forcing term which drives the system. The model can learn both point-to-point and cycle trajectories. DMPs are time independent and are capable of describing the motions of complex mechanical systems with multiple Degrees of Freedom (DoF). Learning the open parameters of the system is straightforward and computationally efficient, which makes it suitable for real-world applications.

2.1 Model Development

A brief overview of the DMP framework will be presented here as described from the authors in [22], [4]. DMPs are a series of different differential equations encompassing discrete and rhythmic movements.

$$\tau \dot{u} = a_z(\beta_z(g - x) - u) + f \tag{2.1}$$

$$\tau \dot{x} = u \tag{2.2}$$

Equations 2.1 and 2.2 are a generalization of the damped-spring model. The variables x and u are the position and velocity of the robot's joints, respectively. τ is the final time T that represents the duration of the demonstration, x_0 and g is the initial position and target position, respectively. Note that equation 2.1 is stable when $(\dot{u}, x) = (0, g)$. The terms a_z and β_z are control gains which render the system critically damped at $\beta_z = a_z/4$ and f acts as the forcing term that drives the system. Equations 2.1 and 2.2 will be addressed as the *transformation system* of the model. The *forcing term* is further defined as:

$$f(s) = (g - x_0)\frac{\sum_i^N w_i \psi_i(s) s}{\sum_i^N \psi_i(s)} \tag{2.3}$$

Where $\psi_i(s) = exp(-h_i(s-c_i)^2)$ are Gaussian basis functions, with width h_i and centers c_i, while w_i are adjustable weights that changes the shape of the Gaussians over the phase s and N denotes the number of Gaussian functions. The forcing term is dependent on s, which is derived from the equation:

$$\tau \dot{s} = -\alpha s \tag{2.4}$$

In equation 2.4, s represents a phase variable that fades over time as it starts with a value of s=1 at time t=0 and becomes zero s=0 at t=τ. In the literature, equation 2.4 is referred to as the *canonical system*, because it determines how the forcing term drives the system over time. The computation of the phase variable s is made possible by setting α in equation 2.4 so that s becomes zero at the final time τ. Figure 2.1 illustrates how the canonical system behaves.

The forcing term of equation 2.3 and the canonical system of equation 2.4 are designed to model the behavior of point-to-point attractor systems containing a distinct beginning x_0 and goal g. The DMP model can describe systems with rhythmic

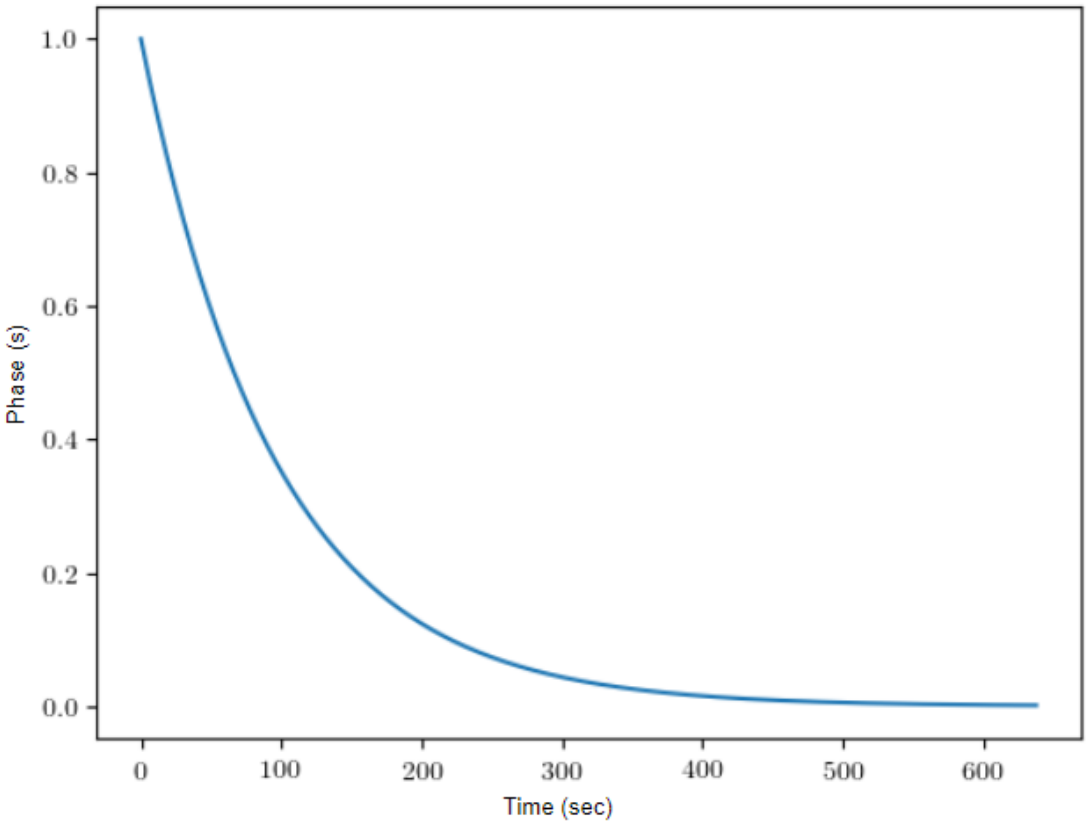


Figure 2.1. Example of the decay of the phase variable s for duration τ=656 sec and α=25.

behavior that follow cycle trajectories by introducing terms to the canonical system and the forcing term that describe periodicity. To achieve this, the authors of [4] suggest changing the canonical system of equation 2.4 to a phase oscillator as shown in equation 2.5, where $\phi \in [0, 2\pi]$ is the phase angle of the oscillator.

$$\tau\dot{\phi} = 1 \tag{2.5}$$

Furthermore, the forcing term of equation 2.3 is changed to equation 2.6 to include the new phase term ϕ, amplitude ρ and instead of the exponential term of equation 2.3, in equation 2.6 the term ψ_i changed to $\psi_i(\phi) = exp(h_i(cos((\phi - c_i) -$

1)), which is a von Mises basis function that act as Gaussian-like functions that are periodic. The system can be initialized with $\rho = 1$, $\tau = 1$ and $\phi = 0$.

$$f(\phi, r) = \frac{\sum_i^N w_i \psi_i(\phi)}{\sum_i^N \psi_i(\phi)} r \tag{2.6}$$

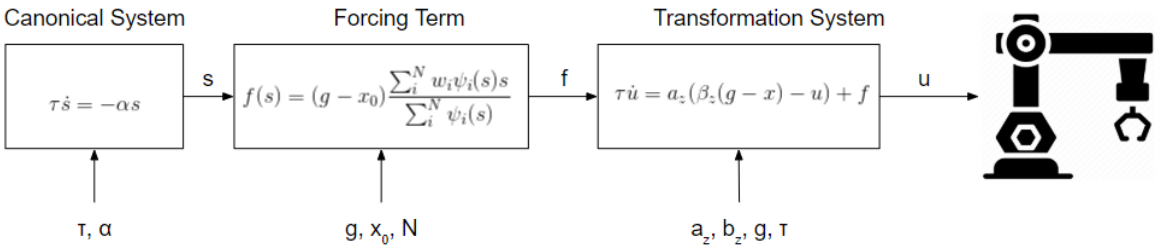


Figure 2.2. Graphical representation of the DMP framework for point-to-point systems .

2.2 The Dynamic Movement Primitives Framework

Figure 2.2 illustrates how the DMP framework operates for the cases of discrete attractor systems. Initially, the canonical system is estimated by using the given duration of the task and the variable α from equation 2.4. The canonical system estimates the variable s, which drives the forcing term f. To compute the forcing term, the weights of the Gaussians w_i, the goal and starting location of the system g, x_0 and the number of Gaussian functions N all must be defined. Finally, the forcing term is transformed into a control command u by employing the transformation system of equation 2.1, which requires the goal g, the duration of the task τ and the control gains a_z, b_z. Control commands can either be a position, velocity, or acceleration command. If the system utilizes equation 2.5 for the canonical system and equation 2.6 for the

forcing term, then the system is capable of encoding rhythmic systems that can perform circular trajectories.

The duration of the task τ, the goal g, and the starting location x_0 depend on the task and can be trivially extracted from the data. The control gains a_z and b_z are selected so that the transformation system is stable. The authors of [4] prove that the transformation system can be expressed as a bounded-input, bounded-output (BIBO) problem which is stable when $a_z = 25$ and $b_z = a_z/4$, while the parameter of the canonical system α can be set at $\alpha = a_z/3$. The parameters of the Gaussian functions of the forcing term in equation 2.3 can be set to $N = 20$, while the center of the Gaussian functions c_i and their height h_i are selected so that the Gaussians are equally spaced in $x(t) = exp(-\alpha(t/\tau))$. The Gaussians weights w_i are the only parameters in the pipeline of Figure 2.2 which are not defined and instead must be learned. This mechanism indicates that by learning the weights w_i, the system can learn motor policies by considering the weights as the features that encode and reproduce motion.

2.3 Learning Movement Primitives

To learn a DMP motor policy, first a teacher must provide a demonstration of a task. From the demonstration, the position $x(t)_demo$, velocity $u(t)_demo$, acceleration $\dot{u}(t)_demo$ and the duration $t = [0, ..., T]$ of the performed trajectory must be recorded. The phase variable s can be computed offline according to equation 2.4 by setting α so that s starts with a value of one at $t = 0$ and becomes zero at the final time $t = T$. To continue with the computation of the motor policy and compute the weights w_i, we then estimate the forcing term of the demonstration f_{target} from equation 2.7,

since the goal g and start x_0 can be found at $x(t=0)$ and $x(t=T)$ and the control gains of the transformation system can be estimated offline.

$$f_{target} = \tau \dot{u_{demo}} - a_z(\beta_z(g - x_{demo}) - u_{demo}) \tag{2.7}$$

The system learns the weights of the Gaussians by performing regression using the Locally Weighted Regression (LWR) algorithm as suggested in [4]. Learning the weights is a function approximation problem where the parameters of f are computed by minimizing the cost function of equation 2.8 such that the representation weights w_i are as close as possible to f_{target}.

$$J_i = \sum_{t=0}^{T} \psi_i(t)(f_{target} - w_i \xi(t))^2 \tag{2.8}$$

where $\xi(t) = x(t)(g - x_0)$ for discrete systems and the solution of equation 2.8 is given from equations 2.10.

$$w_i = \frac{s^T \Gamma_i f_{target}}{s^T \Gamma_i s} \tag{2.9}$$

$$\Gamma_i = \begin{pmatrix} \psi(1) & 0 & 0 \\ 0 & \psi(2) & 0 \\ ... & & \\ 0 & 0 & \psi(T) \end{pmatrix} \tag{2.10}$$

2.4 Dynamic Movement Primitives for Multiple Degrees of Freedom

In the previous chapter we discussed how the DMP framework can learn motor policies for single DoF systems. Figure 2.3 provides a paradigm that describes how the DMP framework can be extended for systems with multiple DoFs that perform

point-to-point trajectories. Each DoF of the system is synchronized according to a single canonical system as equation 2.4 suggests and computes N number of forcing terms according to equation 2.3, where N is this case is the number of DoFs that characterizes the system. Each forcing term is then transformed to a low-level control command u.

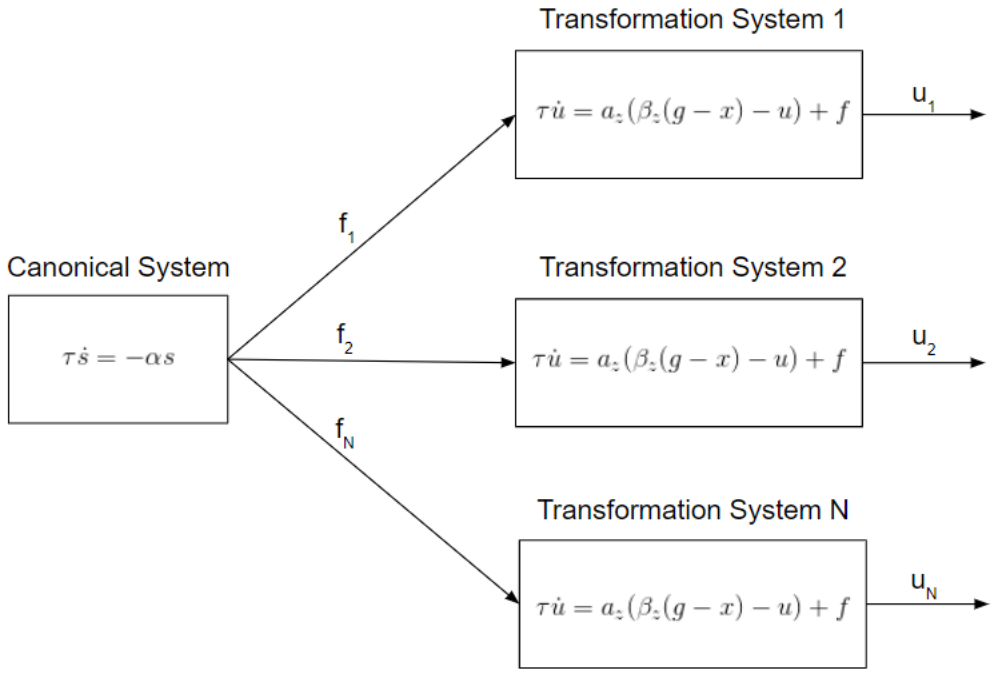


Figure 2.3. Extension of the Dynamics Movement Primitives framework for a system with multiple degrees of freedom where each degree of freedom is represented by a different transformation system.

2.5 Adaptation with Reinforcement Learning

The vast majority of modern imitation learning paradigms rely on self exploration instead of supervised learning. This phenomenon indicates a shift to goal-oriented algorithms for motor policy adaption and generalization. The development

Algorithm 1 Policy learning by weighted exploration.

Initialization: Policy with parameters θ_0 While θ_k has not converged: Create sample rollouts using:
$\alpha = (\theta_k + \epsilon_t)\psi_i(s), \epsilon \sim N(0, \sigma^2)$
Estimate the importance of each rollout:
$Q(s, \alpha, t) = \sum_{t=0}^{T} r(s_t, \alpha_t, t)$
Discard the rollouts with low reward.
Update using the following rule:

$$\theta_{k+1} = \theta_k + \frac{\sum_{t=0}^{T} \epsilon_t Q(s, \alpha, t)}{\sum_{t=0}^{T} Q(s, \alpha, t)}$$

of the DMP framework was an essential contribution to this trend, as it provides an abstraction layer between the dimensions of state, action, and environment by providing policies with distinct meta-parameters that affect the behavior of the system. This formalization enabled the creation of RL policy gradient search algorithms that consider the meta-parameters of the policy as the action space and the joint or Cartesian configuration of the robot as the state space [4], [5].

The fusion of the DMP framework and the RL paradigm has proven to successfully learn complicated goal-oriented tasks such as robot grasping [23] and the *Ball-in-a-Cup* task [24]. However, a single DMP cannot generalize over every possible trajectory that resembles a different task. To adapt motor policies learned from DMPs, various policy search reinforcement learning algorithms have been proposed [25],[26],[24]. From the respective literature, we utilize the Policy learning by Weighting Exploration with the Returns (PoWER) algorithm as described in [5]. A pseudocode of the implemented RL algorithm can be seen in the section Algorithm 1.The general goal of the algorithm is to find a rollout, which in our case is a joint trajectory, that maximizes the reward. Different rollouts can be generated by using equation 2.1 with different forcing terms according to equation 2.3. Note that the forcing term is

the learned policy from the DMP that constitutes the motor policy that generates the entire trajectory. As denoted by equation 2.3 a motor policy is affected by w_i, h_i and c_i. Although all three of these parameters can be modified to generate a new policy. In the current algorithm, we only modify w_i to generate new policies. As such, the algorithm initially generates multiple actions as the sum of the current weights w_i, plus a random exploration rate ϵ_t. The exploration rate is sampled from a normal distribution with mean zero and a standard deviation equal to the standard deviation of the respective weighted Gaussian that is learned from the DMP, which is seen as a separate probability distribution. This decision inserts bias into the system, because this means that the learned Gaussians that contribute the most to the learned policy will probably still contribute the most in the modified policy. Finally, the algorithm updates the parameters of the policy by using the rollouts that received the highest Q values. The number of samples, the percentage of the accepted rollouts, and the selected reward function are defined by the user. Note that learning and further optimization with RL may occur in Cartesian space [2],[27] or in joint space [28], or both [22]. The learning space of the framework is critical as it directly affects the formulation of the reward function [24],[3].

2.6 Learning Motor Skills with Dynamic Movement Primitives

In this section, we provide an example to illustrate how the framework works as expressed in [29]. To learn a motor policy from a demonstration, we record *x(t)* and by knowing the duration $t = T$ we can derive $\dot{x(t)}$ and it's derivative. Figure 2.4 is an example of the data collection process, where a human teacher tries to teach a robot how to grab a 3D-printed block. An example of the Cartesian trajectory that was collected can be seen in Figure 2.4. Furthermore, the system learns by imitation by employing the LWR algorithm [30] to produce the motor policy. Note that the

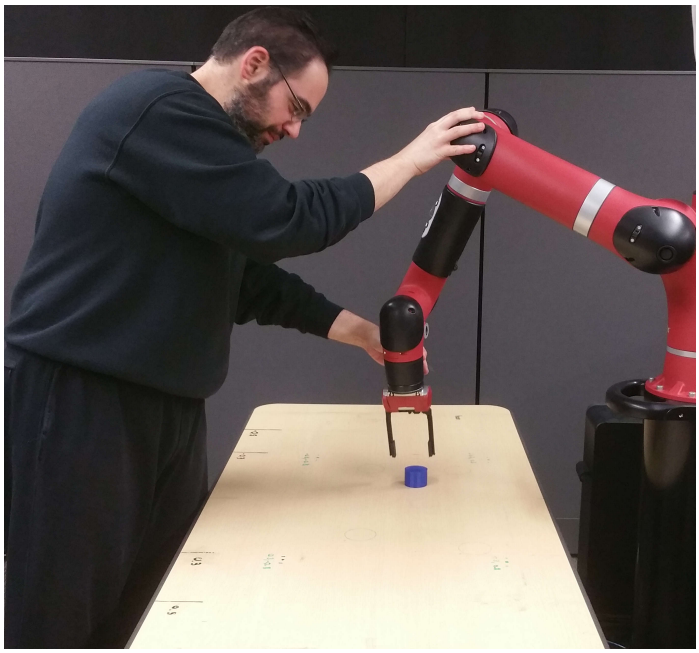


Figure 2.4. Human teacher demonstrating the task via kinesthetic teaching, the data is recorded into a database.

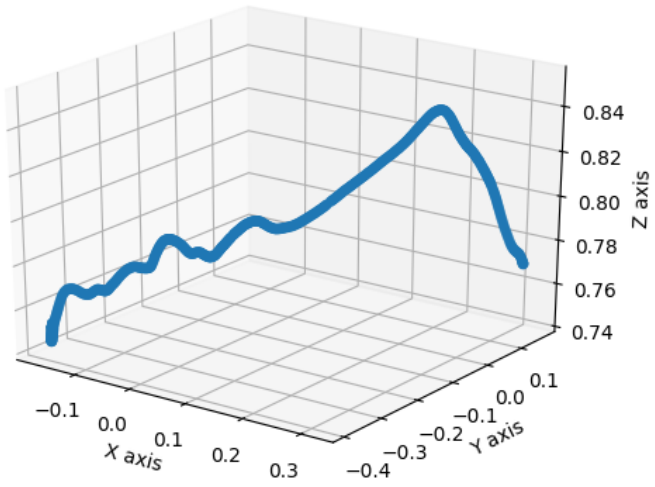


Figure 2.5. A representation of the data from the trajectory shown in figure 2.4.

system learns in joint space and not in Cartesian space, which means that the system abides by the principles expressed in Figure 2.3. The system learns the weights w_i of

the forcing term in equation 2.3 by performing regression with the LWR algorithm [4]. The trajectory which is learned by the system can be seen in Figure 2.6.

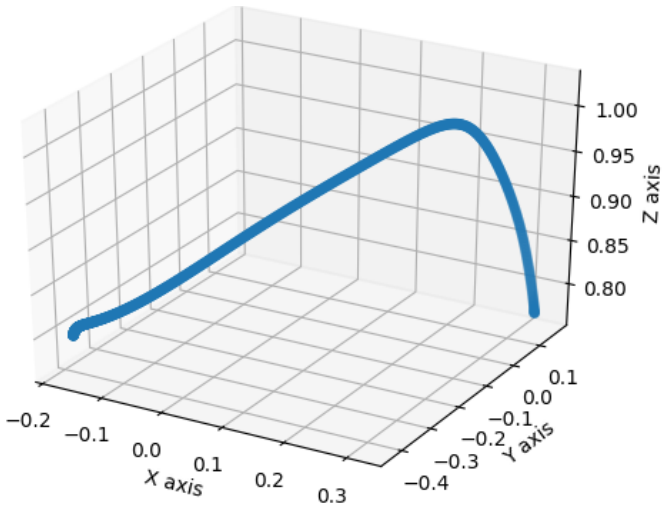


Figure 2.6. The smoothing introduced in the learned trajectory with data from Figure 2.4.

We can further adapt the motor policy to generate new motor skills by utilizing the PoWER algorithm 1. Before performing regression the forcing term can be represented by a battery of Gaussian functions which are equally spread on the s plane. As the leftmost graph of Figure 2.7 suggests. After regression, the functions are spread disproportionately in the s plane, as each weight w_i has changed, and with it each Gaussian has a changed shape as the rightmost graph of Figure 2.7 shows. Figure 2.8 shows in red line the joint trajectory of Figure 2.6, which is a result of of the initial Gaussian policy of Figure 2.7. To adapt the policy according to the PoWER algorithm we have to provide a new goal. The new goal was given in the form of a vector $g = [-2.7, 3.4, 0.6, -0.3, 1.8, -2.7]$. As we can see in Figure 2.6 the black lines, which represent the adapted trajectories, reach the new goal, because the PoWER algorithm to changed the weights w_i. The new Gaussian weights can

be seen in the right section of Figure 2.7. From comparing the two figures we can conclude that the policy adapted and it produced a new forcing term that drives the trajectories to the new desired goal.

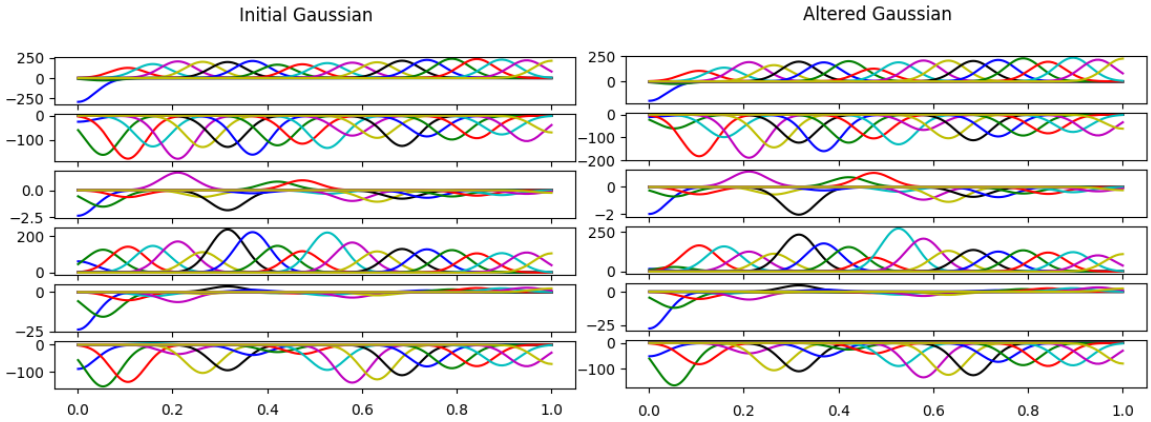


Figure 2.7. The left figure depicts the initial Gaussian representation of the motor policy which is learned from 2.4. The right figure shows how the Gaussian policies changed with Reinforcement Learning.

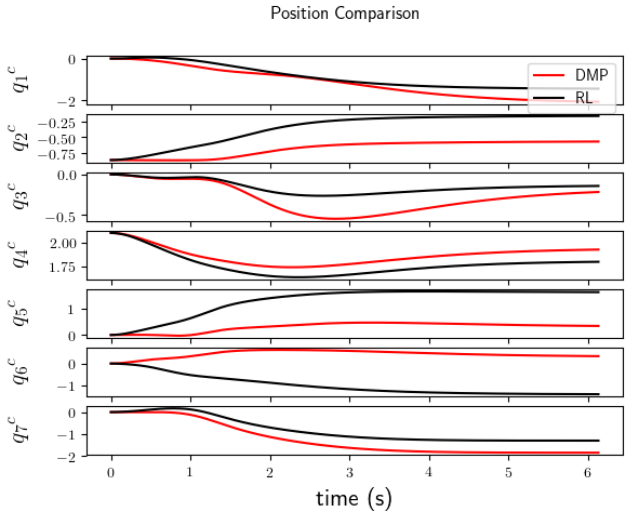


Figure 2.8. Comparing the position trajectory of the initial Gaussian policy in red line with the altered Gaussian policy in black line from Figure 2.7.

CHAPTER 3

A Motion and Force Analysis System for Human Upper-limb Detection

This chapter describes a novel system that can demonstrate the potential to track and estimate the torques that affect the human arm of an individual that performs rehabilitation exercises with the use of the Microsoft Kinect v2. The system focuses on eliminating the jerky motions captured by the Kinect with the incorporation of robotic mechanics methodologies. In order to achieve these results, the system takes full advantage of the dynamic and kinematic formulas that describe the motion of rigid bodies. Simulation experiments are depicted to demonstrate the results of the system.

The system can be used to monitor the physical state of the upper limbs of a human. To further elaborate, with the data collected from the Kinect, the system provides a precise estimation of the motion parameters (position, velocity, and acceleration) and torques that affect the shoulder and elbow of the human. The system emphasizes the use of the Microsoft Kinect v2 as opposed to other systems that require several different sensors such as embedded accelerometers, EMGs or even wearable exoskeleton arms [31] and mechanical manipulators [32]. We will discuss how the system incorporates a variety of key methodologies that originate from the field of robotics mechanics and kinematics in particular.

A considerable amount of research has been conducted in the fields of computer vision, human-computer and human-robot interaction to track the human body. Whereas for entertainment or rehabilitation applications [33], [34], each of the above fields utilizes different technologies and diverse methodologies to track the state of

the human body [35]. In this thesis, we combined a variety of these techniques to track and calculate the motion parameters and torques that affect the shoulder and elbow of an individual who performs rehabilitation exercises. Specifically, research in the area of human exoskeletons shows that the human arm can be mathematically represented as a kinematic chain [36]. [37] of seven DoF, much like a mechanical manipulator. By making this assumption, we can express the relationship of the human joint's rotation and translation in relevance to a world frame by using the Denavit-Hartenberg (DH) parameters [38]. Moreover, we can derive the forward and inverse kinematic equations of the human arm to obtain a relationship between the position and orientation of the end effector (wrist) with the rotation of the joints. These set of equations are useful as they facilitate the calculation of the rotational position, velocity, and acceleration of the joints over a specific trajectory. This makes the estimation of the torques that affect the human arm tractable with the use of the Recursive Newton Euler (RNE) method, which is described in more detail in [39]. We will elaborate more on these methods in the next section of this chapter.

In contrast with other studies, we consider marker-less and low cost solutions for obtaining the necessary positions for our calculations [40]. Our system takes advantage of the Kinect v2 skeleton tracking algorithm to track the position of the wrist. Unfortunately, due to the probabilistic nature of the Kinect skeleton tracker, the positions are collected with certain inaccuracies under specific circumstances [41]. For this reason, the system eliminates jerky data obtained from the Kinect through a polynomial fitting process in the joint space that is derived from the inverse kinematic equations of our human kinematic model. Thus, the proposed system utilizes techniques in computer vision and robot mechanics to solve the human arm tracking problem.

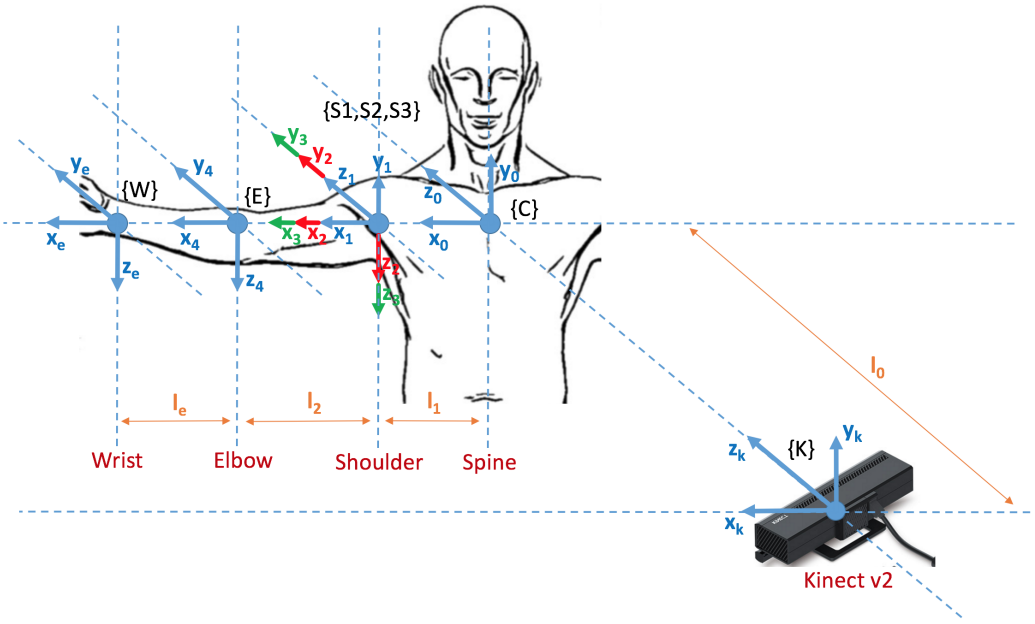


Figure 3.1. 4 DoF Kinematic Model of the Human Arm depicting the coordinate frames for each joint and sensor.

3.1 Human Arm Kinematic model

In this section, we provide a thorough analysis of the bio-mechanic model that the proposed system utilizes to track the human arm. As mentioned above, the human arm can be represented as a kinematic chain, much like a robotic arm. Since the system focuses in the behavior of the shoulder and elbow, we designed a 4 Degree of Freedom (DoF) kinematic chain to express the shoulder glenohumeral rotation and the elbow flexion. Figure 3.1, provides a graphical illustration of the developed human arm model.

A summary of the frames description can be seen in Table 3.1. The kinematic chain begins from frame {K}, which acts as the world frame of the model. Notice that frame {K} denotes where the Kinect is stationed, meaning that when we derive the forward and inverse kinematic equations of the model, all the frame positions are

Table 3.1. Kinematic model frames

Frame	Location	Description
K	Kinect {K}	World frame
0	Chest {C}	Human Chest Position
1	Shoulder {S1}	Abduction: -90 $\leq \theta_1 \leq$ 90
2	Shoulder {S2}	Flexion: -130 $\leq \theta_2 \leq$ 45
3	Shoulder {S3}	Pronation: -90 $\leq \theta_3 \leq$ 90
4	Elbow {E}	Flexion: 0 $\leq \theta_1 \leq$ -150
e	Wrist {W}	Position

related to the Kinect directly. Next, frame {C} represents the chest of the human who is positioned l_0 meters along the z_k axis of the Kinect. As expected, frames {S1},{S2} and {S3} describe the glenohumeral rotation of the shoulder. Frame {S1} and {S2} rotate around the axis z_1 and z_2 as shown in Figure 3.1. Note, that to avoid the formation of an Euler gimbal lock in the shoulder, which would make the solution of the inverse kinematics extremely complicated, frame S3 rotates around the axis x_3 [38]. Frame {E} follows by representing the flexion of the elbow around z_4 axis. Finally, the position of the end effector or wrist {W} is located along the axis x_4 of the elbow frame.

3.1.1 Forward Kinematics of the Human Arm Kinematic model

Based on the modified DH table 3.1 we can determine the rotation and translation of frame $i-1$ to i according to the following matrix:

$$
{}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & \alpha_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i \alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}
$$

$$ {}^{k}_{e}T = {}^{k}_{0}T *^{0}_{1}T *^{1}_{2}T *^{2}_{3}T *^{3}_{4}T *^{4}_{e}T \tag{3.2} $$

Note, that cθ stands for the cosine of θ and sθ stands for the sine of θ. Based on the multiplication that are shown above the general transformation from the Kinect frame to the human's wrist is:

$$ {}^{k}_{e}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & l_e r_{11} + l_2 c_1 c_2 + l_1 \\ r_{21} & r_{22} & r_{23} & l_e r_{21} + l_2 s_1 c_2 \\ r_{31} & r_{32} & r_{33} & l_e r_{31} + l_2 s_2 + l_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3} $$

$$ \begin{cases} r_{11} = c_1 c_2 c_4 - c_1 s_2 c_3 s_4 + s_1 s_3 s_4 \\ r_{12} = -c_1 c_2 s_4 - c_1 s_2 c_3 c_4 + s_1 s_3 c_4 \\ r_{13} = c_1 s_2 s_3 + s_1 c_3 \\ r_{21} = s_1 c_2 c_4 - s_2 s_2 c_3 s_4 - c_1 s_3 s_4 \\ r_{22} = -s_1 c_2 s_4 - s_1 s_2 c_3 c_4 - c_1 s_3 c_4 \\ r_{23} = s_1 s_2 s_3 - c_1 c_3 \\ r_{31} = s_2 c_4 + c_2 c_3 s_4 \\ r_{32} = -s_2 s_4 + c_2 c_3 c_4 \\ r_{33} = -c_2 s_3 \end{cases} \tag{3.4} $$

3.1.2 Inverse Kinematics of the Human Arm Kinematic model

For the derivation of the inverse kinematic equations we consider the positions of the wrist and elbow as the known variables and the joint angles as unknown variables. Traditionally, in robotics the orientation and position of the end effector

are the only known parameters. However, in this particular case, instead of using the wrist's orientation, we use the position of the elbow to find the joint angles as it can be directly obtained from the Kinect.

$$ {}^{k}_{e}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x_e \\ r_{21} & r_{22} & r_{23} & y_e \\ r_{31} & r_{32} & r_{33} & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5} $$

$$ {}^{k}_{4}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x_4 \\ r_{21} & r_{22} & r_{23} & y_4 \\ r_{31} & r_{32} & r_{33} & z_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6} $$

Position of Elbow:

$$ \begin{cases} x_4 = l_2 * c_1 * c_2 + l_1 \\ y_4 = l_2 * s_1 * c_2 \\ z_4 = l_2 * s_2 + l_0 \end{cases} \tag{3.7} $$

Position of Wrist (End Effector):

$$ \begin{cases} x_e = l_3 * (c_1 * c_2 * c_4 - c_1 * s_2 * c_3 * s_4 + \\ s_1 * s_3 * s_4) + l_2 * c_1 * c_2 + l_1 \\ y_e = l_3 * (s_1 * c_2 * c_4 - s_1 * s_2 * c_3 * s_4 - \\ c_1 * s_3 * s_4) + l_2 * s_1 * c_2 \\ z_e = l_3 * (s_2 * c_4 + c_2 * c_3 * s_4) + l_2 * s_2 + l_0 \end{cases} \tag{3.8} $$

Angles Derivation $(\theta_1, \theta_2, \theta_3, \theta_4)$:

$$\begin{cases} s_2 = \frac{z_4 - l_0}{l_2} \\ c_2 = \pm\sqrt{(1 - {s_2}^2)} \\ \theta_2 = atan2(s_2, c_2) \end{cases} \tag{3.9}$$

$$\begin{cases} s_1 = \frac{y_4}{l_2 * c_2} \\ c_1 = \pm\sqrt{(1 - s_1)} \\ \theta_1 = atan2(s_1, c_1) \end{cases} \tag{3.10}$$

$$\begin{cases} c_4 = \frac{l_2^2 + l_3^2 + \sqrt{((x_e - x_1)^2 + (y_e - y_1)^2 + (z_e - z_1)^2)}}{2 * l_2 * l_3} \\ s_4 = \pm\sqrt{(1 - c_4^2)} \\ \theta_4 = atan2(s_4, c_4) \end{cases} \tag{3.11}$$

$$\begin{cases} c_3 = \frac{z_e - z_4}{l_3} \\ s_3 = \pm\sqrt{(1 - c_3^2)} \\ \theta_3 = atan2(s_3, c_3) \end{cases} \tag{3.12}$$

3.1.3 Human Arm Dynamic Model

In robotics dynamics, the RNE method is often used to solve the inverse dynamics problem. Specifically, it is used to provide an estimate of the torques that produce the robot's joints given the angles, velocities, and accelerations of the joints [38][42]. However, the RNE method can be applied to every kinematic chain, as long as the motion parameters and the mass of the links are known. In our case, since we are measuring the torques that affect the human arm, we estimated the mass of the human upper limbs by taking into consideration the relative relation between

the human weight and the weight of the human arm. [10] Figure 3.2 depicts the analogies of anthropomorphic data that were taken into consideration in the scope of this study.

The RNE method is divided into two steps as seen below. In the first step (Outward iteration), the method calculates the relative angular and linear motion parameters from the starting joint to the end effector. The second step (Inwards iteration) iterates backwards and provides an estimation of the moments and torques that affect the joints.

3.1.3.1 Outward iteration

$0 \rightarrow 3$

ω : relative angular velocity of joints

$\dot{\theta}$: joint angular velocity

$\ddot{\theta}$: joint angular acceleration

$\hat{Z}$: the axis of rotation

P : matrix indicating the direction of the center of mass

m : mass of link

c : center of mass

F : linear Force applied in the center of mass

N : moments

$$
\begin{cases}
^{i+1}\omega_{i+1} = {}_{i}^{i+1}R * {}^{i}\omega_i + \dot{\theta}_{i+1} * {}^{i+1}\hat{Z}_{i+1} \\
^{i+1}\dot{\omega}_{i+1} = {}_{i}^{i+1}R * {}^{i}\dot{\omega}_i + {}_{i}^{i+1}R^i * \omega_i \times \dot{\theta}_{i+1} * {}^{i+1}\hat{Z}_{i+1} + \\
\ddot{\theta}_{i+1} * {}^{i+1}\hat{Z}_{i+1} \\
^{i+1}\dot{v}_{i+1} = {}_{i}^{i+1}R(^{i}\dot{\omega} \times {}^{i}]P_{i+1} + {}^{i}\omega_i \times ({}^{i}\omega_i \times {}^{i}]P_{i+1}) + {}^{i}\dot{v}_i) \\
^{i+1}\dot{v}_{C_{i+1}} = {}^{i+1}\dot{\omega}_{i+1} \times {}^{i+1}P_{C_{i+1}} + {}^{i+1}\omega_{i+1} \times ({}^{i+1}\omega_{i+1} \times \\
^{i+1}]P_{C_{i+1}}) + {}^{i+1}\dot{v}_{i+1} \\
^{i+1}F_{i+1} = m_{i+1} * {}^{i+1}\dot{v}_{C_{i+1}} \\
^{i+1}N_{i+1} = {}^{C_{i+1}}I_{i+1} * {}^{i+1}\dot{\omega}_{i+1} + {}^{i+1}\omega_{i+1} \times {}^{C_{i+1}}I_{i+1} * \\
^{i+1}]\omega_{i+1}
\end{cases}
\tag{3.13}
$$

3.1.3.2 Inward iteration

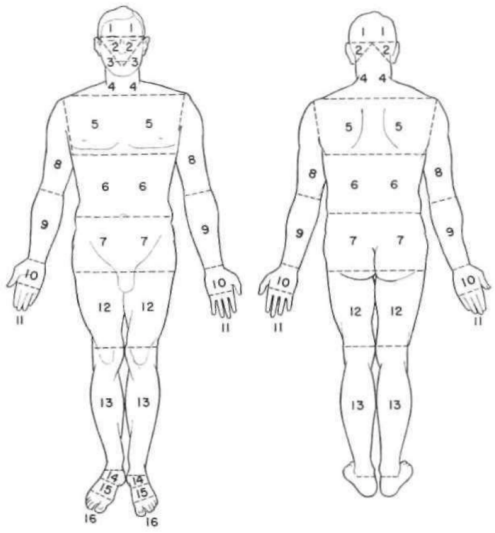
f : Force Propagation

τ : Torque applied to the joints

n : Accumulative torque applied to the joints

$4 \rightarrow 1$

$$
\begin{cases}
^{i}f_i = {}_{i+1}^{i}R * {}^{i+1}f_{i+1} + {}^{i}F_i \\
^{i}n_i = {}^{i}N_i + {}_{i+1}^{i}R * {}^{i+1}n_{i+1} + {}^{i}P_{C_i} \times {}^{i}F_i + {}^{i}P_{i+1} \times \\
{}_{i+1}^{i}R * {}^{i+1}f_{i+1} \\
\tau_i = {}^{i}n_i^T * {}^{i}\hat{Z}_i
\end{cases}
\tag{3.14}
$$



Segment	8 Subjects	
Cranium (1) and upper jaw (2)	71.64	
Lower jaw (3) and neck (4	38.32	
Head and nech (1 + 2 + 3 + 4)		109.96
Chest (5)	186.10	
Abdomen (6)	137.47	
Pelvis (7)	182.95	
Whole trunk (5 + 6 + 7)		506.52
Upper arm (8)	28.04	
Forearm (9)	14.90	
Palm and trumb (10)	5.20	
The four fingers (11)	1.95	
The whole hand (10 + 11)	7.15	
Both upper extremities		100.19
Thigh (12)	81.63	
Shank (13)	43.56	
Base of foot (14)	13.77	
Middle foot (15)	13.77	
The five toes (16)	2.70	
The whole toes (14 + 15 + 16)	16.47	
Both lower extremities		283.33
Total body		**1,000.00**

Figure 3.2. Anthropomorphic data for the human body segments captured by [10]. The table values are the mass distributions for different body parts.

3.2 Motion and Force Analysis System Overview

A block diagram of the proposed system can be seen in Figure 3.3. The diagram summarizes the majority of the processes that compose the overall system. As an input to the system, the user must capture the person who is performing the exercise with the Kinect, according to the configuration that Figure 3.1 suggests. Once the trajectory of the subject's arm has been captured, the Kinect passes the Cartesian positions of the chest, shoulder, elbow, and wrist frames to the first unit of the system that reconstruct raw model. The raw model provides an abstract illustration of the physical configuration that was captured by the Kinect.

The system then applies a median filter to the raw data to eliminate any abnormal behavior from the skeleton tracking algorithm of the Kinect. This module produces a smooth trajectory that is used by the system's Inverse Kinematics Solver (IK Solver) to provide an estimation of the angles of the human arm joints. Note that this unit utilized the inverse kinematics equations that were described in the

previous section. At this point, the system has produced the first estimation in joint space. In the next iteration, the system must make sure that all data in joint space are characterized by a polynomial profile. This happens because the motion of all rigid bodies, such as our model, must be expressed with a polynomial function that can produce a second, third or even forth derivative (jerk)[38].

Once the polynomial fitting process is completed, the system recreates the kinematic model according to the forward kinematic equations and calculates the velocities and accelerations of the human arm joints. Lastly, the system provides an estimation of the torques that affect the subject's arm with the RNE method and presents all of the results in the graphical user interface as denoted in Figure 3.3.

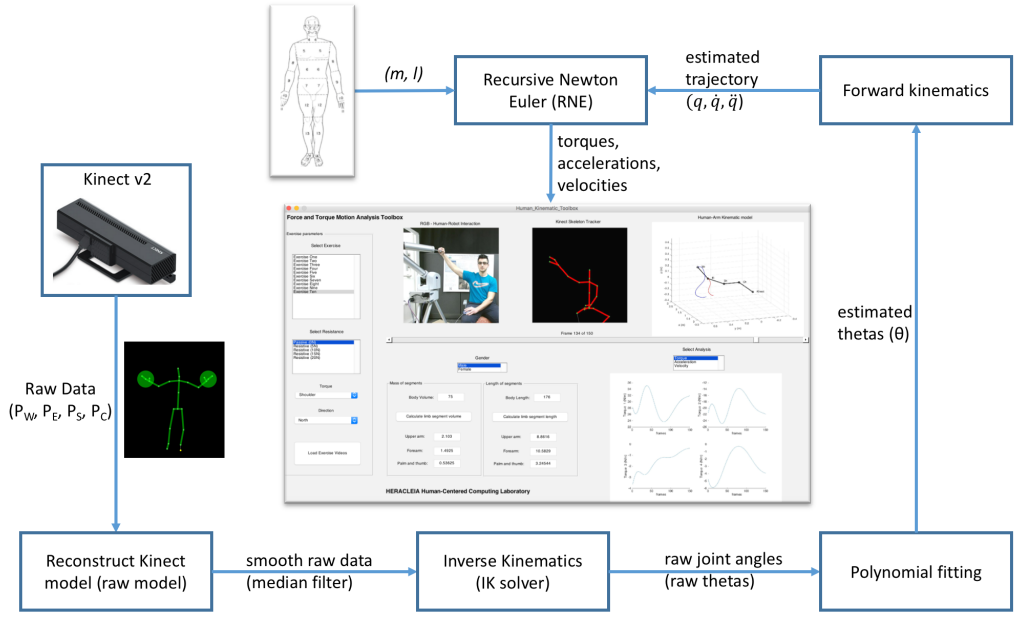


Figure 3.3. End-to-end pipeline for capturing human key-points with Kinect v2 through processing of data and finally application of RNE to calculate torques that affect the human arm.

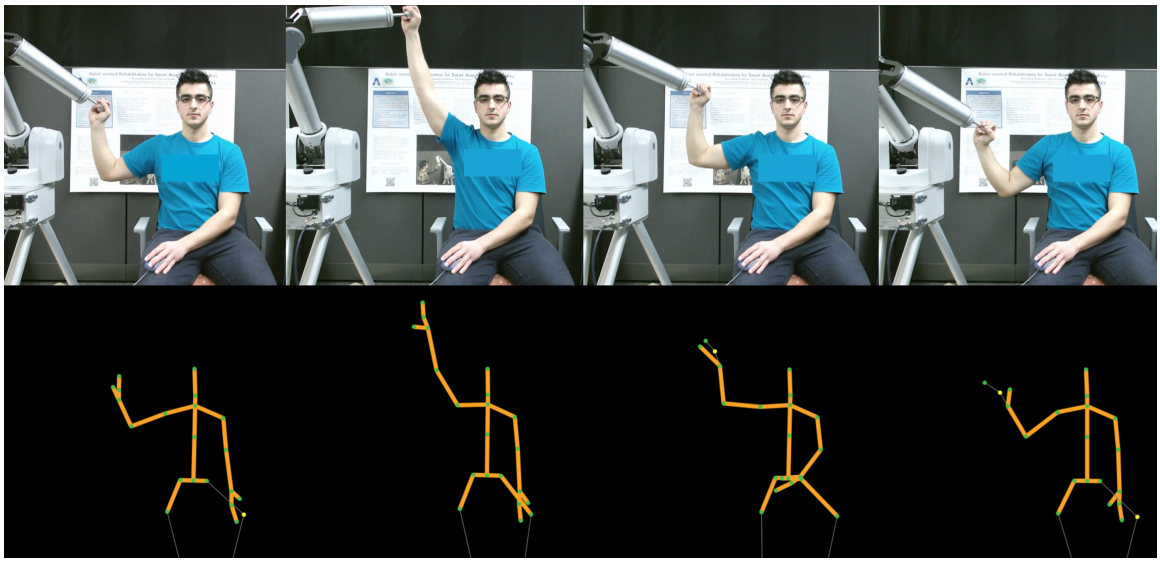


Figure 3.4. The above images present four frames for a particular exercise, and jerky motions from the Kinect is inevitable. Note, that the Kinect doesn't estimate precisely the position of the left arm, because of the same depth information with the rest of the body.

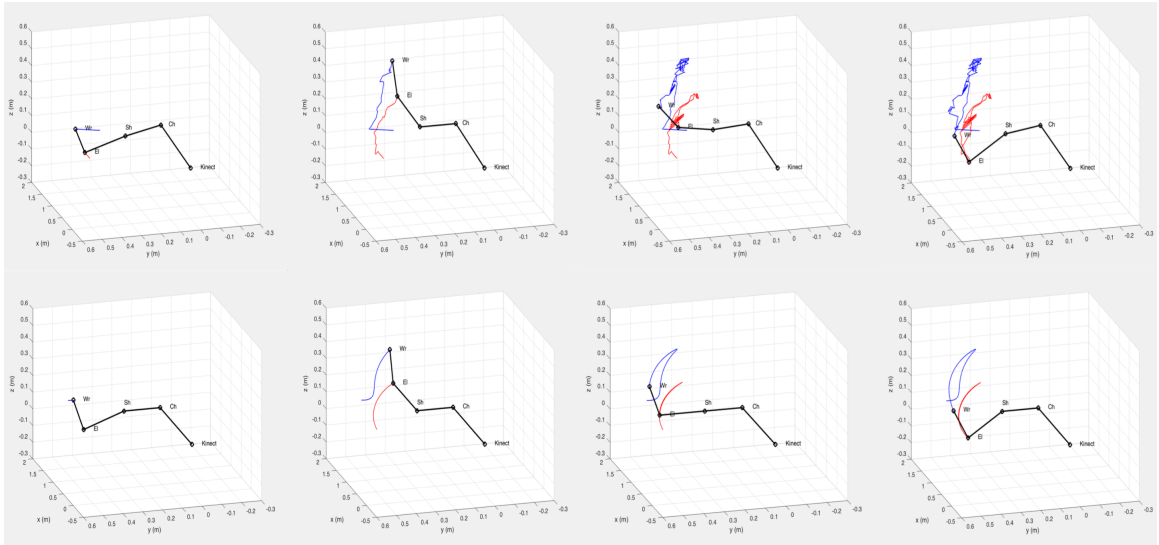


Figure 3.5. The top four diagrams show the evolution of the exercise motion for the raw Kinect model. The bottom four diagrams present the corrected motion based on the polynomial fitting motion estimation.

3.3 Experimental Assumptions

Before we continue to the experimental results section, we must mention certain assumptions that led to the system's development. The following hypothesis are centered around the idea that we try to estimate the kinematic and dynamic properties of the human arm with techniques that have been applied to mechanical manipulators. Our assumptions can be summarized as follows:

- First of all, since our study focuses on tracking the human shoulder and elbow, we disregarded the degrees of freedom that the human wrist provides. Traditionally, the human arm is represented with 7 DoFs (3 at the wrist, 1 at the elbow and 3 at the shoulder). However our kinematic model is limited to only 4 DoFs. The inclusion of a 7 DoF model in our system would be extremely challenging, due to the Kinect's inability to provide an accurate estimation of the wrist's rotation.
- Second, we did not include in our calculations any external forces that are exclusive to the human arm. The reason behind this decision is the absent of a wrist frame, which would suggest the incorrect propagation of the external force in the kinematic chain by the RNE unit of the system. Additionally, our model doesn't consider any relevant friction between the joint links as it is difficult to simulate the effects of muscle fatigue.
- Last but not least, the assumption that the human arm can be abstracted as a rigid body kinematic chain is incorrect, because on of the key principles of rigid body dynamics is that the modeled body does not succumb to deformation. This is clearly an incorrect statement in our case. Despite this fact, this assumption is widely used in the related work which was mentioned in section 2.

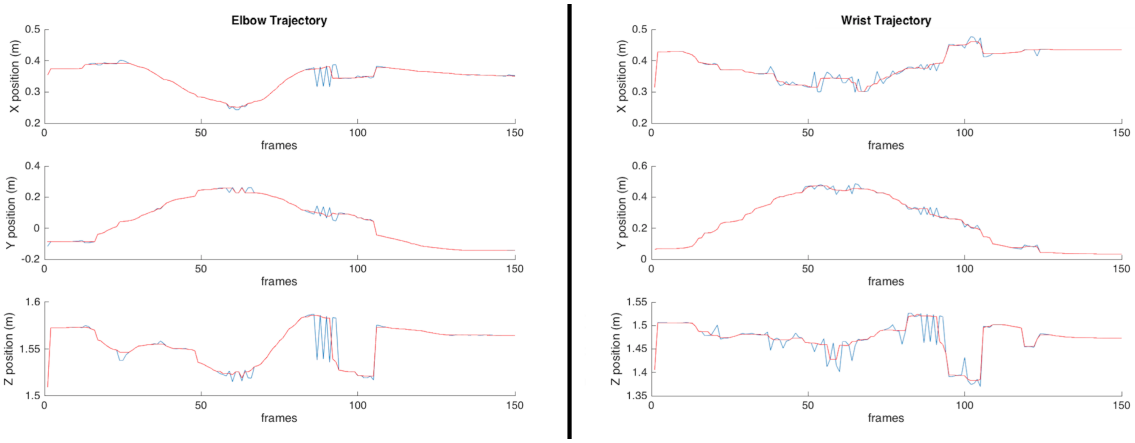


Figure 3.6. Captured Elbow and Wrist positions by the Kinect,blue line. Filtered Elbow data, red line. Note that the proposed method eliminates oscillations present in the original signal.

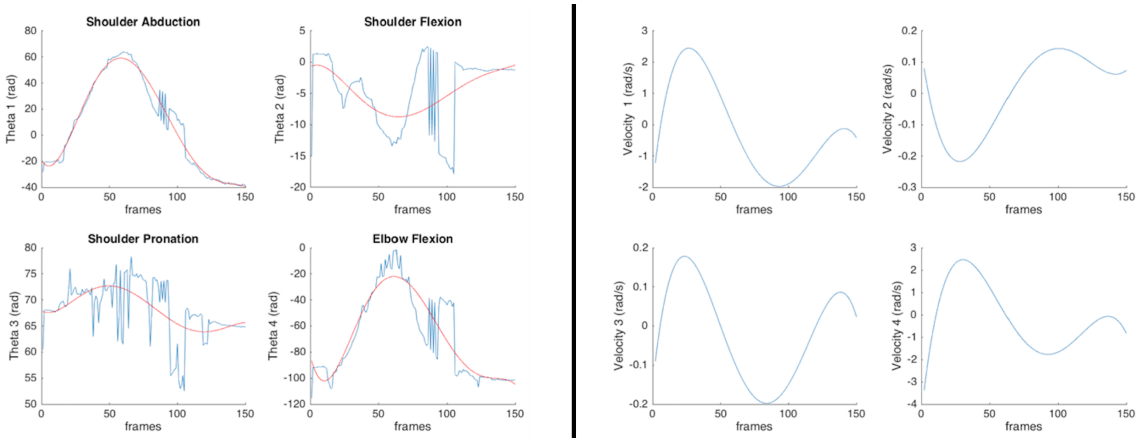


Figure 3.7. The image of the left side depicts the polynomial fitting of the estimated joint angles from the IK solver. The fitted solutions are shown in red color, while the raw angles are in blue. The right image shows the velocities of the fitted joint angles.

3.4 Experimental Results

The team conducted various exercises using the Barret WAM arm robotic manipulator in order to extensively validate the behavior of the system. The robotic arm was used in the experiment as an ad-hoc simulator of various rehabilitation exercises.

Figure 3.4 shows the physical configuration of the experiment and the captured Kinect skeleton tracker, while the results of the experiment are presented in Figure 3.5. The results indicate that the system greatly improved the initial estimation of the Kinect. Specifically, in Figure 3.5 the upper four sub figures show the gradual evolution of the captured Kinect trajectories of the human wrist and elbow in blue and red colors. It can be clearly seen from the rapid fluctuations of the trajectories that the Kinect does not regard the physical properties of the human arm as these trajectories doesn't correlate with the motion of the real human arm. However, the four bottom sub pictures of Figure 3.5, that demonstrate the final estimation of the system in equivalent time frames provide a more accurate description of the actual physical trajectory that the human arm followed in the exercise.

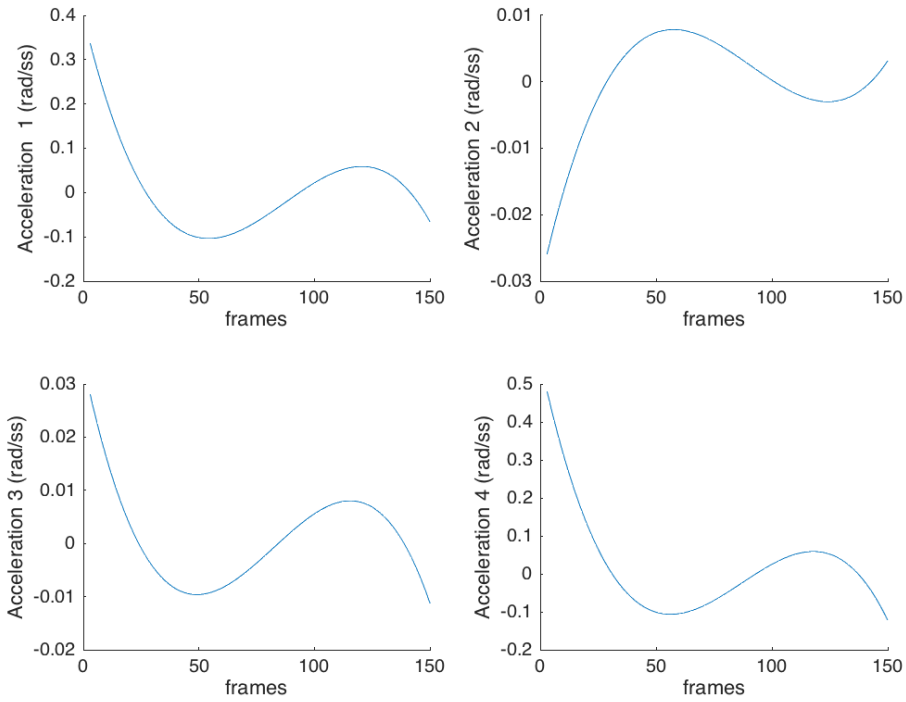


Figure 3.8. Angular accelerations for the four joints.

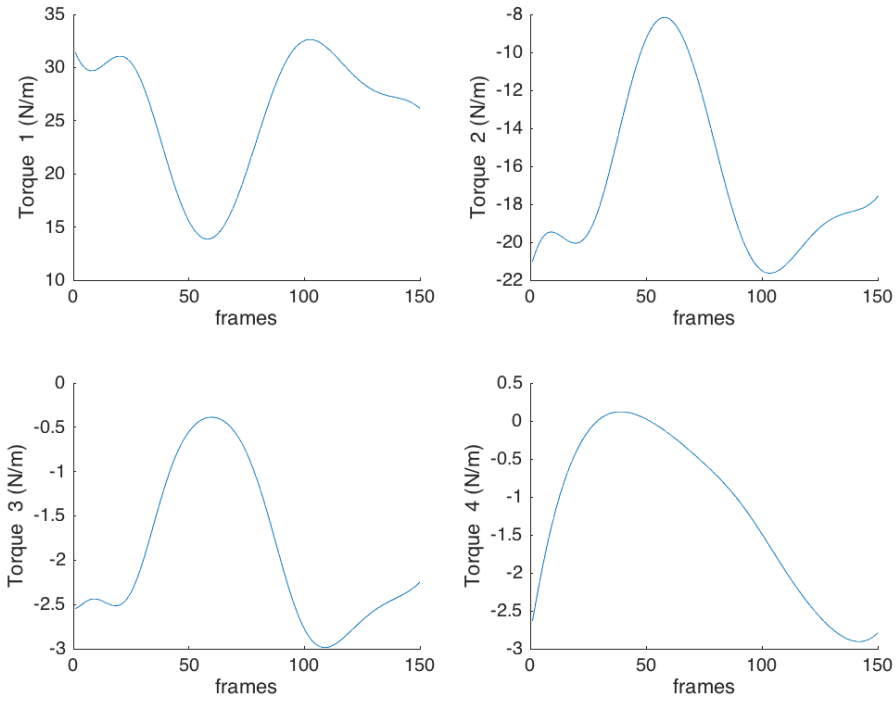


Figure 3.9. Estimated Torques (N/m) for the four joints.

It is our omission not to refer to the predominant reason that causes these inaccurate readings of the Kinect v2. Traditionally, if in this particular instance the human arm was performing a free motion without any wearable attachment to the wrist, the Kinect should be more accurate and without fluctuations. But in this setup, since the participant is grabbing the end-effector of the robotic arm, the Kinect's depth sensor regards the robotic arm as a natural extension of the human's arm and thus provides false readings in certain frames.

Up to this point, the user of the system can perceive how the system obtained the raw data from the Kinect and produced an accurate estimation of the real motion. We will further enrich the experimental results section, by providing additional details for the intermediate steps of the system, which are detrimental for the final derivation

of the torque calculations. Starting from the manipulation of the raw Kinect data, the system applies a median filter to eliminate any irregular variation in collected data. Figure 3.6 describe how the median filter is applied in the current experimental set up. Obviously, the median filter removed certain values in the captured elbow and wrist trajectory that would otherwise make the derivation of the inverse kinematics solution unsolvable. After the trajectory filtering is done, the system performs the polynomial fitting process in the joint space obtained from the solution of the inverse kinematic equations and the filtered trajectories.The left side of Figure 3.7 shows analytically polynomial fitting process of the raw angles obtained from the solution of the inverse kinematics equations and the filtered trajectories. Lastly, the systems derives the velocities, accelerations and torques through the derivation of the joint space trajectories. The results are shown in the right side of Figure 3.7, Figure‘3.8 and Figure 3.9.

CHAPTER 4

VARM: Using Virtual Reality to Program Robotic Manipulators

In this chapter we present an application that demonstrates how Human-Robot Interaction (HRI) may benefit from Virtual Reality (VR) and non-pervasive technologies, such as the Leap Motion Controller, to present a safe, visual and interactive way to program robotic arms [8]. By interacting with the virtual robot, the user can define the task of the real robot. The system acts as a teleoperation interface system to program industrial robotic arms. The system demonstrates the potential to create a programmable interface that enables users with no prior knowledge of robotics to safely program mechanical manipulators with the use of VR and the Leap Motion Controller. The system takes full advantage of the Leap Motion to navigate the virtual work-space of the robot that was created through the kinematic properties of the real robot. The implementation of the application was deemed possible by interfacing the Unity Engine with the four Degrees of Freedom (DoF) Barrett WAM robotic arm. Preliminary experimental results show the ability of the system to engage and train appropriately the user in robot programming.

4.1 Programming Robotic Manipulators with Virtual Reality VARM

HRI is a multidisciplinary field of science that strives to establish safe, intuitive and robust communication methods between robots and humans. In this chapter, we present an application that demonstrates how HRI may benefit from VR and non-pervasive technologies, such as the Leap Motion Controller, to present a safe, visual and interactive way to program robotic arms. The application's goal is to provide

an immersive and easy-to-use interface for users that wish to program a mechanical manipulator to perform simple or complicated manual tasks, such as to pick and place an object from a particular location to another. Specifically, the application utilizes the Unity game engine to provide a VR environment of the robot's work-space in combination with the Oculus Rift headset for visualization. The Leap Motion Controller was mounted on top the Oculus Rift device to capture the hands of the user and enable gesture recognition to traverse the virtual environment and interact with the virtual robot. By interacting with the virtual robot, the user can define the task of the real robot. Once the task has been defined, the task is sent to the real robot, which in our case is the four DoF Barrett Whole Arm Manipulator (WAM). Figure 4.1 illustrates the set up.

Whereas for industrial or medical applications, the field of HRI utilizes different technologies and diverse methodologies to program robotic arms. The variety of the techniques depends on the level of autonomy of the robot and the task it must accomplish. These techniques range from direct physical interaction with the robot [43] to computer aided graphical interfaces (GUI) [44],[13]. Note that a common challenge that HRI tries to combat is to provide an interface that decouples the complexity of the kinematics of the robot from it's programming [43],[44],[13] as the visualization of the dexterous work-space of the robot is difficult. Additionally, in applications that regard the teleoperation of one or multiple slave robots it has been shown that the emerging technology of Leap Motion [43],[16] can provide more interactive ways to define a robot's trajectory and define a robot's motion through gesture recognition [45], [46],[47].

In our work, we combined a variety of the previously mentioned techniques that have been used to program industrial robotic arms. Specifically, the presented application emphasis the use of Leap motion in combination with VR to provide

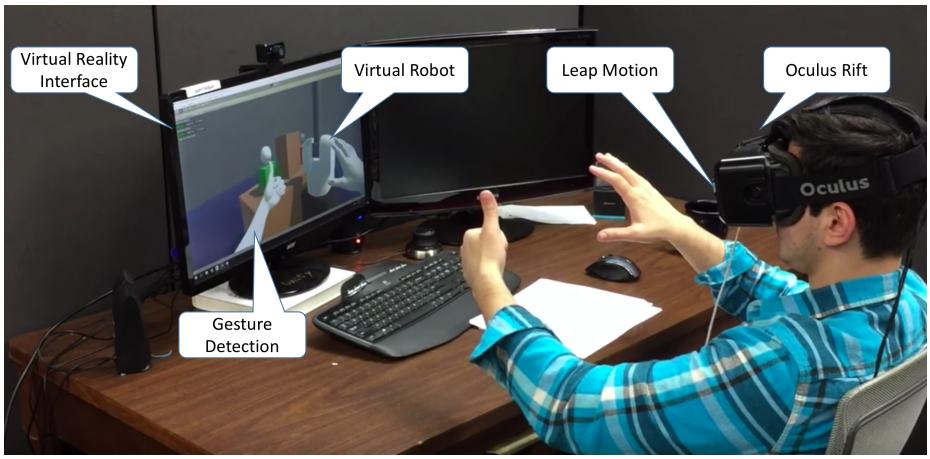


Figure 4.1. The interface of the proposed method with the various components highlighted.

an immersive and interactive experience while the user programs the robot. By directly interacting with the robot in VR the user can define a task without any prior knowledge of the kinematics or the dynamics. In addition, gesture recognition enhances the experience by relating the gestures of the user into particular commands that control the navigation of the user in the VR and the motion of the virtual robot. As seen from related work, gestures captured from the Leap device can be correlated to relative gripper postures in the real robot [46]. An equivalent methodology has been applied in the presented application, since the task performed by the real robot is based on the interaction between the user and the virtual robot. To elaborate more, the user defines the trajectory of the real robot by performing a grabbing gesture on the end effector of the virtual robot to move it through the desired trajectory. The virtual robot then follows the grabbing gestures of the user, while the system records the performed movement of the virtual robot. Once the trajectory is defined, the captured motion is sent to the real robot, which mimics the behavior of the virtual robot. A set up of the system can be seen in Figure 4.1

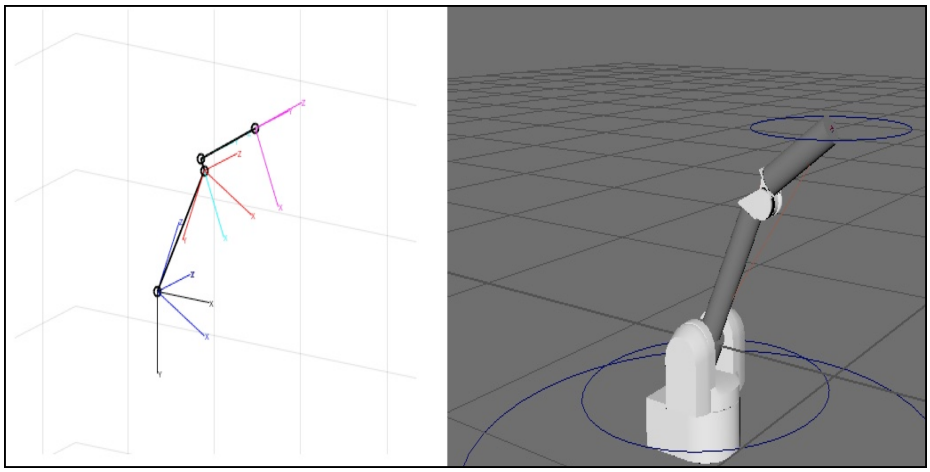


Figure 4.2. Barrett WAM Arm kinematic model and the Virtual Robot in Maya showing the link configurations.

4.2 System Components

In this section, we provide a thorough analysis of the modeling process of the virtual robot and how the application enables the interaction between the user and the robot through gesture recognition. As mentioned above, the user will be able to use his hands as a tool to navigate and interact with the robot, which means that the work-space of the virtual robot must accurately represent the work-space of the real robot.

4.2.1 Forward Kinematics of the 4-DoF Barrett WAM

Figure 4.2 illustrates the kinematic model that was used to construct the virtual model of the robot. The frame placement of the kinematic chain is defined according to the DH parameters, which are provided in the Barrett WAM user manual.

Table 4.1. DH Table for 4-DOF Barrett WAM

i	α_i	a_i	d_i	θ_i
1	-90	0	0	θ_1
2	90	0	0	θ_2
3	-90	0.045	0.55	θ_3
4	90	-0.045	0	θ_4
e	0	0	0.35	0

Based on the DH table described above the homogeneous coordinate matrix of the frames can be derived according to the following matrix :

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1} $$

Thus, the forward kinematics of the robot are derived by the following equation:

$$ {}^{k}_{e}T = {}^{k}_{0}T *^{0}_{1}T *^{1}_{2}T *^{2}_{3}T *^{3}_{4}T *^{4}_{e}T \tag{4.2} $$

Which determines that the resulting homogeneous transformation from the base frame of the robot to the robot's end effector frame:

$$ {}^{k}_{e}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x_e \\ r_{21} & r_{22} & r_{23} & y_e \\ r_{31} & r_{32} & r_{33} & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3} $$

$$
\begin{cases}
r_{11} = c_1c_2c_3c_4 - s_1s_3c_4 - c_1s_2s_4 \\
r_{12} = -c_1c_2s_3 - s_1c_3 \\
r_{13} = c_1c_2c_3s_4 - s_1s_3s_4 + c_1s_2c_4 \\
r_{21} = s_1c_2c_3c_4 + c_1s_3c_4 - s_1s_2s_4 \\
r_{22} = -s_1c_2s_3 + c_1c_3 \\
r_{23} = s_1c_2c_3s_4 + c_1s_3s_4 + s_1s_2c_3 \\
r_{31} = -s_2c_3c_4 - c_2s_4 \\
r_{32} = s_2s_3 \\
r_{33} = -s_2c_3s_4 + c_2c_4 \\
x_e = l_2r_{13} + z_4r_{11} + z_3(c_1c_2c_3 - s_1s_3) + l_1(c_1c_2) \\
y_e = l_2r_{23} + z_4r_{21} + z_3(s_1c_2c_3 + c_1s_3) + l_1(s_1s_2) \\
z_e = l_2r_{33} + z_4r_{31} + z_3(-s_2c_3) + l_1(c_2)
\end{cases}
\tag{4.4}
$$

4.2.2 Inverse Kinematics of the 4-DoF Barrett WAM

In contrast to the forward kinematics problem, the goal of the inverse kinematics problem is to find a set of joint configurations given a particular end-effector position and orientation. The difficulty of the inverse kinematics problem arises from the fact that it depends on the physical configuration of the robot. Predominantly, non-redundant robotic arms can be solved analytically, while more complicated redundant manipulators require more advanced mathematical solutions such as artificial intelligent, pseudo-inverse or transpose Jacobian solutions. However, these methodologies require algorithms with high time complexity costs, when compared with the analytical solutions [48], [49].

The 4 DoF Barrett WAM is a kinematic redundant manipulator, which means that an analytical solution is impossible to exist. However, we solved the inverse kinematics analytically by setting the redundant third joint of the robot as a free parameter and thus effectively converting the redundant kinematic chain of the Barrett WAM robot to a non-redundant one. Note, that although this simplification allows to decouple the kinematic problem and solve it analytically, the solution of the inverse kinematics now exists for any particular x, y, z coordinate within the Barrett WAM work-space configuration, but only one fixed orientation that is defined by the value of the third joint angle θ_3.

For the purposes of the VR application, it was decided that the value of θ_3 should be zero. This alters the final position vector as follows:

$$\begin{cases} x_e = c_1(l_2c_2s_4 + l_2s_2c_4 + z_4c_2c_4 - z_4s_2s_4 + z_3c_2c_4 + l_1s_2) \\ y_e = s_1(l_2c_2s_4 + l_2s_2c_4 + z_4c_2c_4 - z_4s_2s_4 + z_3c_2c_4 + l_1s_2) \\ z_e = -l_2s_2s_4 + l_2c_2c_4 - z_4s_2c_4 - z_4c_2s_4 - z_3s_2 + l_1c_2 \end{cases} \tag{4.5}$$

By taking the sum of the squares of the position vectors, the solution of θ_4 can be given from the equation:

$$A\tan\frac{\theta_4}{2}^2 + B\tan\frac{\theta_4}{2} + C = 0 \tag{4.6}$$

where,

$$\begin{cases} A = \frac{x_e^2+y_e^2+z_e^2-z_3^2-z_4^2-l_1^2-l_2^2+2(l_2l_1+z_4z_3)}{4} \\ B = -(l_2z_3 - l_1z_4) \\ C = \frac{x_e^2+y_e^2+z_e^2-z_3^2-z_4^2-l_1^2-l_2^2-2(l_2l_1+z_4z_3)}{4} \end{cases}$$

As a result, θ_4 has two solutions (an elbow up and elbow down solution) which are:

$$\theta_4 = 2 \arctan \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{4.7}$$

Moreover, to derive the solution of θ_2 we take the sum of squared the x and y position vectors. Note, that as with θ_4, θ_2 also has two solutions (shoulder up, shoulder down):

$$\theta_2 = 2 \arctan \frac{\pm M\sqrt{x_e^2 + y_e^2} - z_e L}{\pm L\sqrt{x_e^2 + y_e^2} + z_e M} \tag{4.8}$$

where,

$$\begin{cases} M = (l_2 c_4 - z_4 s_4 - l_1) \\ L = (l_2 s_4 + z_4 c_4 + z_3) \end{cases}$$

Last but not least, the solution of θ_1 can be found from the x and y coordinates of the robot's end effector.

$$\theta_1 = \arctan \frac{y_e}{x_e} \tag{4.9}$$

From the partial analytical solution described above it is important to mention that θ_4 has two independent solutions, θ_1 has an independent solution that can result in two different joint configurations and θ_2 can be computed by two different solutions that depend on θ_4. This implies that there is a total of four unique kinematic configurations for every Cartesian point of the robot's end effector. This phenomenon is typical for planar manipulators as the same elbow up/down position of the robot can be expressed with different joint configurations.

4.2.3 Gesture Detection

For accurate estimation of static and dynamic gestures along with finger 3D positions, we selected the Leap Motion as the acquisition device [50]. According to the specifications provided by the manufacturer, it reaches up to 200 frames per second processing speed, with a high accuracy of 0.2mm. It has a field of view of 150 degrees with $0.25m^3$ interactive 3D space. It is powered over USB and works more efficiently with USB 3.0 port.

By leveraging the flexibility and prowess of the state-of-art SDK provided by the manufacturer, the depth signals recorded by the device are transformed into quantifiable entities such as fingers, hands, gestures, and positions. The Leap Motion provides coordinates in millimeters with respect to its frame of reference as shown in Figure 4.2. Also a class called *Frames* in the SDK represents a set of hands as well as fingers which are tracked in the captured frame. From the Frame object, one can access the hand object and get the fingers position as well as other details, palm velocity/orientation, etc. Several dynamic gesture detection utilities are provided by the developers, specifically the *pinch detector* enabled us to detect and track the activity of holding a ball and moving it around in the virtual environment and *click detector* for detecting the click gesture.

According to the designed protocol, the left hand is designated for moving around in the virtual environment, while the right hand is used for holding a ball, which controls the robot's end-effector position. For detecting the gestures to move around in the virtual world, we are considering the *isExtended()* member function of the the *pointable* class of the left hand, which is assigned to each finger. If the user wants to move to the right, *isExtended()* is called for each finger *pointable* and if the

index finger returns true, then the *move right* activity is activated. Similarly, if the user wants to move to the left, the *isExtended()* of only the thumb should be true.

For holding the object, the pinch detects utility was used. The reason for using this gesture is that it provides a natural sense of holding an object. The pinch detector activates if the user makes a pinch gesture around 3cm to 10cm away from the object and the distance between the index finger and thumb is less than 1cm.

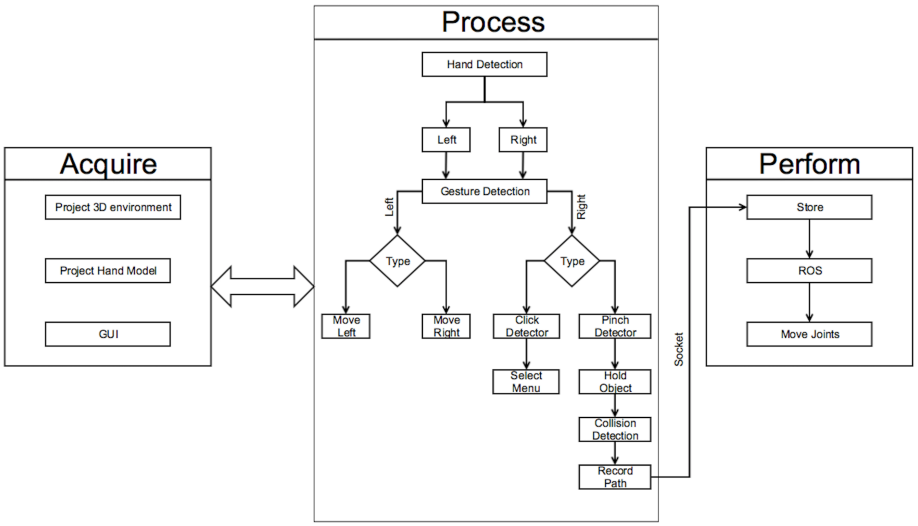


Figure 4.3. VARM System Architecture depicting the three stage pipeline: acquire, process, and perform. Note that the process performed relies on which hand is used.

4.3 VARM System Architecture

The proposed system can be segmented into three parts as Figure 4.3 depicts:

1. *Acquire*: The acquisition phase involves projecting the 3D environment developed in Unity to the Oculus Rift. The default settings of the Leap Motion project only the 3D skeleton. To make it look more realistic a 3D hand model prefab developed by the manufacturer was attached to the Leap Motion *user*

object. The transformation of the 3D hand model changes according to the position and orientation of each joint in the hand, which is captured by the Leap Motion Controller. The avatar's point of view was defined as a transformation that is modified according to the user's head movement, which is captured by the Oculus Rift.

2. *Process*: Once the GUI and the 3D environment is projected, the next phase involves the detection of hand gestures. The Hand detection module is responsible for getting the 3D joint coordinates for both hands. Further processing is the conducted based on whereas the left or right hand is detected. If the user performs a pinch gesture the pinch detection activates given that the conditions detailed in the previous Section are satisfied. The inverse kinematics module of the system is triggered as soon as the ball moves according to the grabbing gesture. The forward kinematics provide visualization of the robot's joint configuration by changing the angular values of the virtual robot, which is projected to the Oculus. When the ball starts moving the 3D coordinates of the ball are recorded given that the collision detection module returns a false indication. This is possible by making the obstacles as *triggers* in Unity, so that it sends a signal to the system indicating a collision. This prevents the path recording module to capture incoming coordinates that exist inside an obstacle. The goal position is a translucent sphere which has a *trigger* attached to it and as soon as the ball hits the goal sphere the system understands that the user has reached the goal. The application will stop and the 3D coordinates are then sent to the actual robot.
3. *Perform*: The stored trajectory of the robot's end effector is passed to the robot by using socket communication to connect to ROS, which then transmits

the necessary control signals to the robot's motors according to the inverse kinematics.

A video example of the proposed VR teleoperation system is illustrated at this link:

https://www.youtube.com/watch?v=h5JdnUqQf9A.

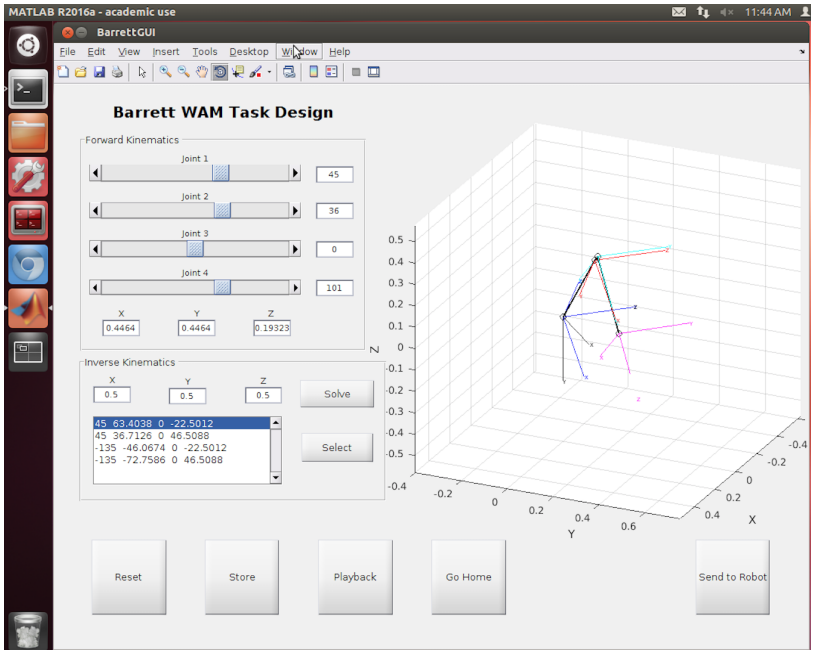


Figure 4.4. 2D Graphical User Interface to Control the Barrett WAM. The Interface is used as a baseline comparison to measure the effectiveness of the proposed method.

4.4 Experimental Hypothesis and Case Study

To evaluate the usability and effectiveness of the proposed interface system, we created a hypothesis that compares our system with two different approaches that have already been established in the related bibliography. The comparison is meant to test the system in terms of safety and usability. To test the hypothesis we performed an experimental case study that provides measurements to support or debunk the

hypothesis. Our goal will be to prove whereas that the VARM can provide a safe and intuitive interface to program robotic arms when compared to an application that utilizes a 2D GUI (Figure 4.4) and an application that enables the user to teach the robot through direct physical interaction.

Figure 4.5. Physical configuration of the experiment depicting the boxes used and the relative position of the robotic arm. The VR setup can be found in the upper left hand corner of the image.

At this point, we have to mention the experimental configuration that was used to test the hypothesis and provide details about the test protocol that was followed throughout the process. The experiment involved a robotic challenge that would help determine the strength or weakness of the hypothesis. Figure 4.5 illustrates the scenario of the challenge. To elaborate more, the participants of the experiment were asked to use the three provided interface applications (Learning from Demonstration, 2D GUI, Virtual Environment) to make the robot move from the green box of Figure 4.5 to the red one without colliding with any objects in the real environment. Figure 4.6 depicts an overview of the three different interface systems.

The hypothesis was tested by 11 different participants of both genders, with little to no background in the field of robotics, from the age of 20 to 25. The development team of VARM provided a quick overview of the experimental process and explained the challenge that every participant must accomplish with all three inter-

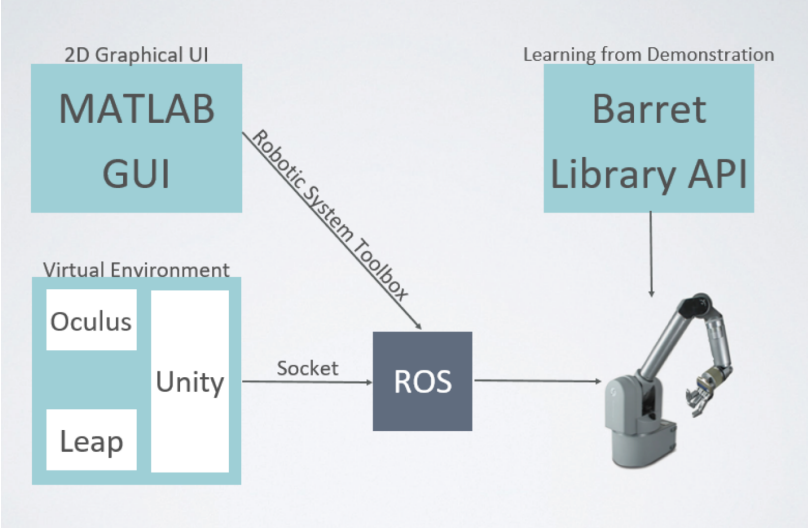


Figure 4.6. Overview of the three different interface systems where both MATLAB control code and VR interact with the Barrett WAM via ROS.

faces. Furthermore, each participant was exposed to a tutorial run by a member of the team and was given a trial run to get a better understanding of every different system before they had a test run. Note that during the test run the team measured how much time it took each user to accomplish the task and how many times the robot collided with an obstacle.

4.5 Experimental Results

Results showed that the participants took less time to complete the task of the experiment by directly interacting with the real robot, when compared to the VARM and 2D GUI approaches. The VARM interface stood second in this comparison, while the 2D GUI was last. Figure 4.7 illustrates these results. Moreover, Figure 4.8 showcases that the same pattern was monitored from the recordings of the obstacle collisions. To be more specific the average time to perform the task took 24 seconds by direct interaction, 92 seconds when using the 2D GUI and 42 seconds with the VARM. The average number of collisions when performing the task directly was 0,

6 when using the 2D GUI and 3 when using the VARM. Also, regardless of the measurements that were collected, it is clear that the safest approach is the 2D GUI, the most dangerous one is the direct interaction with the real robot and the VARM stands somewhere in between.

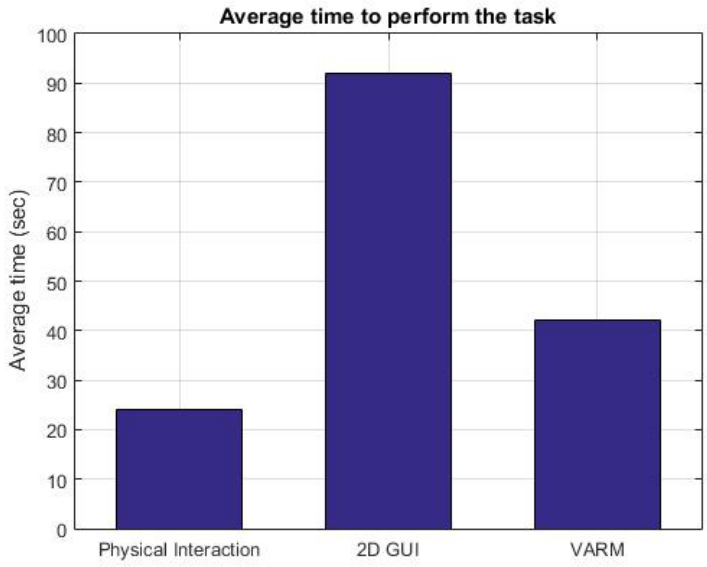


Figure 4.7. Average time to perform a task between the thee interface systems of Figure 4.6. Note that due to the cumbersome interface of the 2D GUI, it required over double the time as VARM to perform the task.

As a final notice, from the numerical data that were collected, it was shown that the VARM interface stood in between the other two approaches in terms of both task design effectiveness and safety. That is to be expected since VARM tries to emulate a real interaction between a robot and a human, but it is lackluster it terms of freedom and realism, since no matter how realistic a virtual environment is, it still alienates the user from the real world.

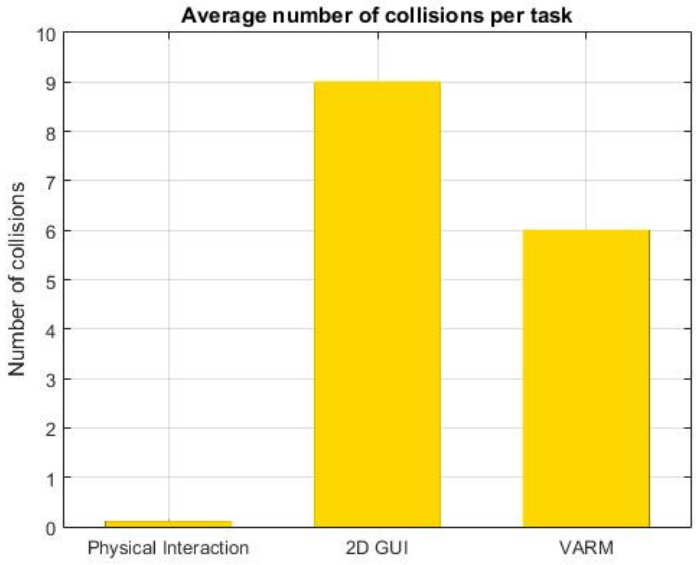


Figure 4.8. Number of collisions per task when comparing the three interface systems of Figure 4.6. Physical interactions avoid collisions as the robot can be directly manipulated in the original environment .

4.6 Summary

Given the high tracking and gesture detection accuracy provided by the Leap Motion Controller, the proposed interface enables a user to program a robotic arm in an intuitive manner. The experimental results proved that the users felt that interacting with a robot is more natural with a gesture based VR environment than with a teaching box or a graphical interface. The system gives the freedom to make errors and collide with objects in the virtual world which could prove costly. Thus, a collision avoidance system could be implemented in the VR that would prevent the real robot from hitting a real object. Moreover, the interaction could be improved by enabling the user to interact directly with any part of the robotic arm, not just the end effector. Lastly, it is important to highlight that the system requires an accurate replication of real world in the virtual world. However, this could be eliminated

by integrating a depth sensor and projecting a point cloud representation of the robot's work-space to the Oculus, or by having multiple stereo cameras that provide continuous visual feedback back to the user in the VR environment.

CHAPTER 5

Combining Forward and Inverse Models with Reinforcement Learning for Motor Policy Adaptation

In this chapter, we describe a two-phase system for robot motor skill learning. In the first phase, a pair of pre-trained forward and inverse models [51] perform a series of actions based on the demonstration of a desired motor skill. In the second phase, the system learns an initial state-action policy by using the Dynamic Movement Primitives (DMP) framework [4]. The initial policy is further optimized with Reinforcement Learning (RL) according to a task-dependent reward function. The system is tested on the Degrees of Freedom (DoFs) Sawyer Robotic Arm. The system learns goal-dependant motor skills by employing Learning from Demonstration (LfD), with a human teacher providing a demonstration which is used to acquire an initial motor policy by utilizing the DMP framework [4],[15].

The adaptation is possible by incorporating goal-driven policy search RL methods [5],[52]. However, the adaptation capabilities of a single motor policy still remains an open problem. In this chapter, we plan to incorporate forward and inverse models in the learning loop to improve environmental adaptation[51]. The purpose of the models is to provide an initial suboptimal motor policy, which can be learned by a DMP and further optimized with RL.

In recent years, many imitation frameworks have been applied to solve the problem of deriving policies that generalize the behavior of a robot over different environmental states. Initial attempts to create learning from demonstration methods purely relied on supervised learning algorithms [2], while the vast majority of

modern imitation paradigms rely on RL [3]. This phenomenon indicates a shift to self exploration algorithms. The development of the DMP framework was an essential contribution to this trend, as it provides an abstraction layer between the dimensions of state, action, and the environment by providing a policy with distinct meta-parameters. This formalization enabled the creation of RL policy search algorithms that consider the meta-parameters of the policy as the action space and the joint or Cartesian configuration of the robot as the state space [25],[52]. Moreover, another discrepancy between the proposed methods is that the learning occurs on the Cartesian space [2],[27] or the joint space [28], or both [22]. The learning space of the framework is important because it directly affects the formulation of the reward function [3],[24].

The fusion of the DMP framework with RL has proven to successfully learn complicated goal-oriented tasks such as robot grasping [23] and the *Ball-in-a-Cup* task [24]. However, a policy cannot generalize over every possible trajectory that resembles a different task. Experimentation has shown that tasks which are composed of a plethora of different sequences of kinematic configurations requires multiple demonstrations and policies to be learned. A scenario which best illustrates this case is a robot that learns to play tennis by demonstration [53]. The authors of [53] propose the construction of a library of motor primitives (MoMP), which associates different environmental stimuli with the equivalent meta-parameters of the learned motor policies. To learn the initial suboptimal motor policy, they combine the meta-parameters in action space with the use of a gating network. Another way to address the issue of motor primitive generalization and combination, the authors of [22] also propose the creation of a motion library that combines motor primitives with traditional domain planning artificial intelligence methods. In our work, instead of using a gating network, we propose the incorporation of a forward model and an inverse model

that estimates the initial suboptimal mapping, without having to perform multiple demonstrations and learn multiple DMPs.

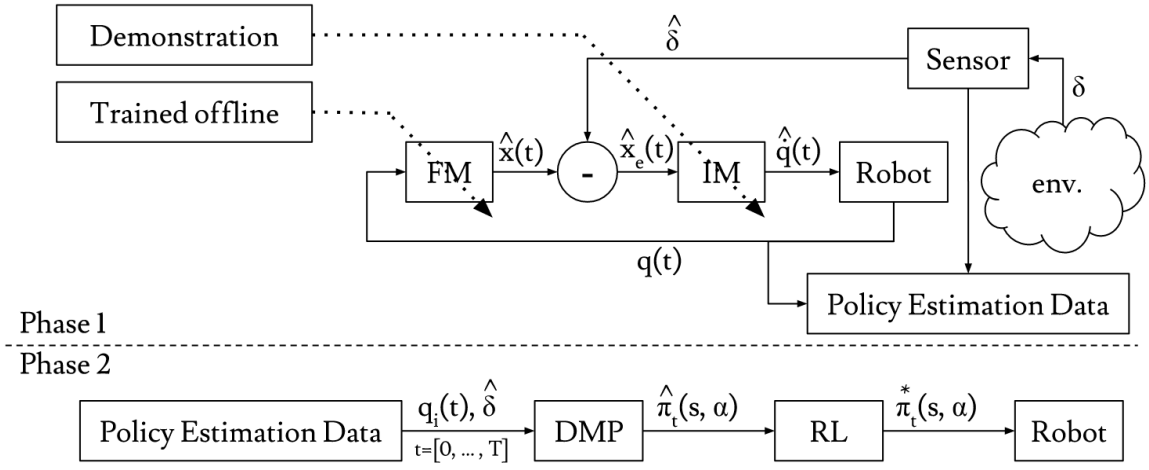


Figure 5.1. Architecture of system that combines forward and inverse models with reinforcement to learn motor policies.

5.1 Forward and Inverse Model System Architecture

In this section we will provide an overview of the proposed system architecture. The system can be seen as a two phase system where phase one performs policy estimation and phase two does policy optimization. Figure 5.1 illustrates the architecture of the system. In phase one, the pair of pre-trained forward and inverse models estimate an initial policy. The Gaussian representation of this suboptimal policy is then learned according to the presented DMP framework and it is improved with RL.

5.1.1 Phase One: Policy Estimation

Phase one starts with the system receiving a goal $\hat{\delta}$ from the external sensor. The system computes a distance vector that describes the relationship of the robot

with respect to $\hat{\delta}$ by combining the output of the forward model with the stimuli of the sensor. The relationship vector, denoted as $\hat{x}_e$ is fed to the inverse model which provides a motor command to the robot to drive it closer to the goal. As the robot tries to reach the goal, the state of the robot and the goal is recorded in a dataset for phase two.

In the scope of this thesis, we consider the robot's joint positions $q(t)$ as the robot's state and the position of the graspable object as the goal $\hat{\delta}$. Given this scenario, the forward model solves the forward kinematics of the robot, while the inverse model receives an estimation of the Cartesian distance vector from the robot's end-effector to the goal and provides a joint velocity command. The velocity command is then applied to the joints of the robot to minimize the distance vector $\hat{x}_e(t)$.

Since both models represent non-linear mappings, we trained two different multilayer perceptron neural networks with distinct hyperparameters, which were trained in a supervised manner. The forward model was trained offline by utilizing the forward kinematics equation of the respective robot [51], [54]. The architecture of the forward model depends purely on the geometrical characteristics of the robot [54], while the composition of the second network relies on the complexity of the demonstration [51]. Thus, the inverse model was trained offline with data collected from the demonstration of the task. The forward model was incorporated to make the system as model agnostic as possible.

The demonstration was performed with kinesthetic teaching and it depicted how the robot can reach the goal δ. During the demonstration, we recorded the location of the object $\hat{\delta}$, the estimation of the position of the robot's end-effector $\hat{x}$, the state of the robot $q(t)$ and the duration of the demonstration. To enhance the accuracy of the inverse model, we augmented the dataset of the demonstration with simple third

and fifth order polynomial splines of various parabolic blends [55]. Effectively, this enables the inverse model to learn different paths from the start location of the robot until the robot successfully reaches the goal location.

The architecture of the forward and inverse model is heavily based on the mechanical articulation of the robot platform, which is part of the system. The forward model estimates the forward kinematic equation of robots with multilayer feed-forward neural networks. In the current experiments we utilize the 7-DoF Sawyer Robotic Arm. We formalize the kinematic problem as a supervised problem and we propose an MLP architecture to solve the problem [54].Figure 5.2 illustrates the kinematic model of the Sawyer Robotic Arm. The model was constructed by reverse engineering the geometrical properties of the physical robot. According to the homogeneous transformation of the joint frames from Figure 5.2, the DH Table 5.1 of the model was formulated. Note though, that in the table we do include the elevation of the robot above the world frame, which is estimated to be 0.3160 meters. Based on the Denavit–Hartenberg (DH) table, the homogeneous coordinate matrix of the frames can be derived according to matrix (5.1).

Table 5.1. DH Table for the 7DOF Sawyer Robotic Arm

i	α_i	a_i	d_i	θ_i
1	-90°	0.0810	0	θ_1
2	90°	0	0.1910	θ_2
3	-90°	0	0.3990	θ_3
4	90°	0	-0.1683	θ_4
5	-90°	0	0.3965	θ_5
6	90°	0	0.1360	θ_6
7	0	0	0.1785	θ_7

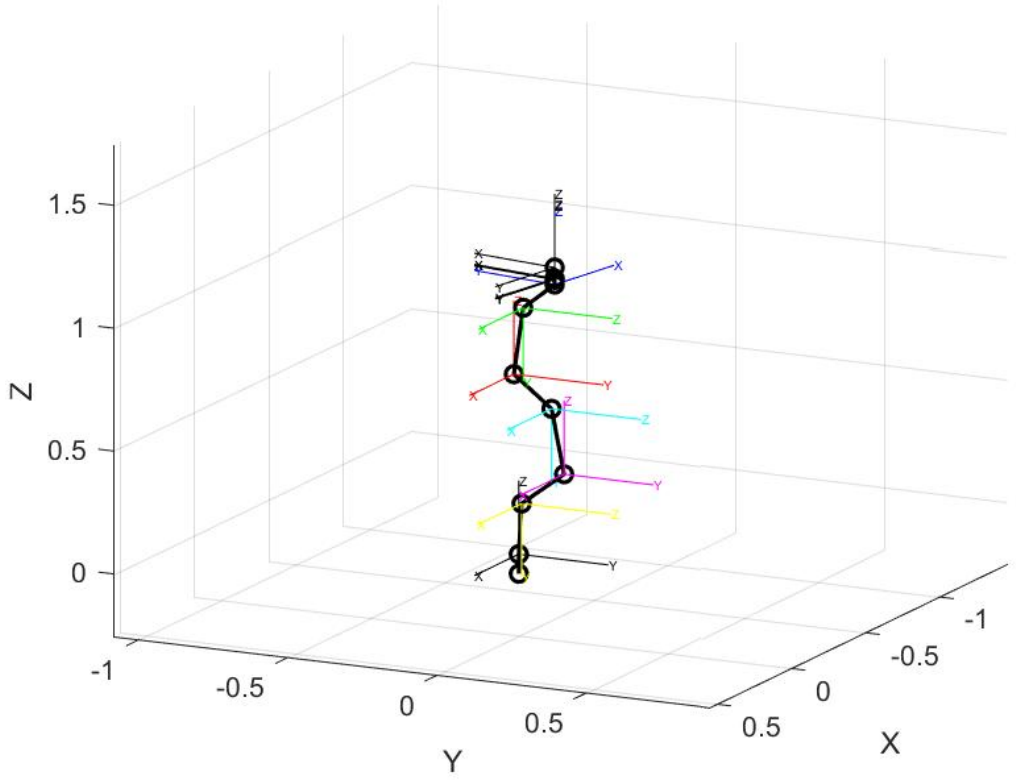


Figure 5.2. Kinematic Model of the Sawyer Robot.

$$ {}^{i-1}_{i}T = \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1} $$

$$ {}^{1}_{7}T = {}^{1}_{2}T *^{2}_{3}T *^{3}_{4}T *^{4}_{5}T *^{5}_{6}T *^{6}_{7}T \tag{5.2} $$

$$\begin{cases} -175^\circ \leq \theta_1 \leq 175^\circ \\ -175^\circ \leq \theta_2 \leq 175^\circ \\ -175^\circ \leq \theta_3 \leq 175^\circ \\ -170^\circ \leq \theta_4 \leq 170^\circ \\ -170^\circ \leq \theta_5 \leq 170^\circ \\ -170^\circ \leq \theta_6 \leq 170^\circ \\ -180^\circ \leq \theta_7 \leq 180^\circ \end{cases} \tag{5.3}$$

To solve the forward kinematics problem of equation 5.1, we employ the multi-layered feed-forward neural network of Figure 5.3. The input layer of the network represents a vector of joint angle values (θ_1,θ_2,θ_3,θ_4,θ_5,θ_6,θ_7), while the output of the network stands for the Cartesian coordinates of the robot's end effector. Both the input and output units contain linear units for normalization purposes.

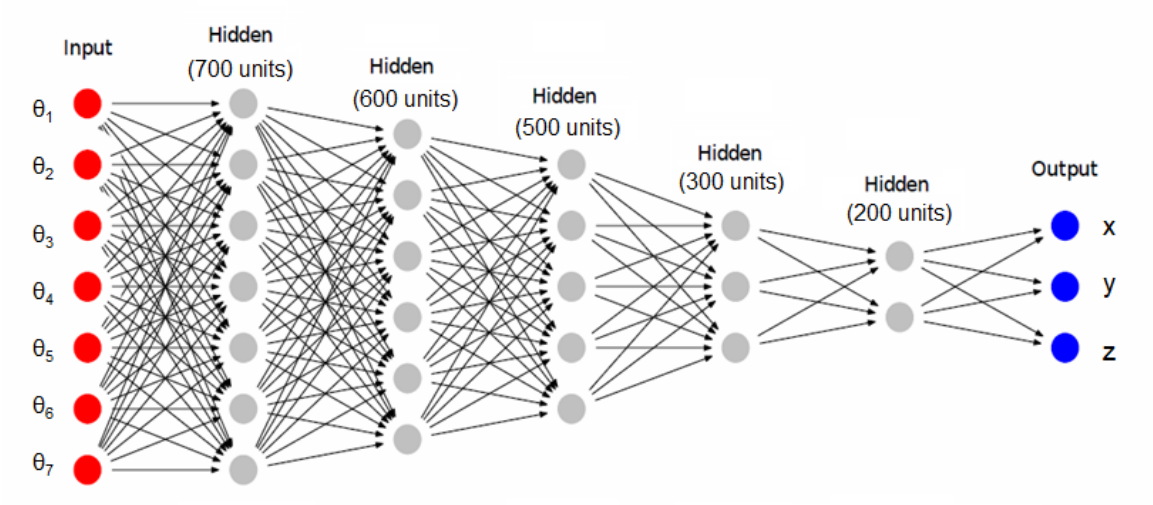


Figure 5.3. Architecture of the Forward Model.

The network was trained using the back-propagation algorithm with the mean squared error of the output units as a metric. During the back-propagation process, we used ADAM optimizer. To produce the training dataset of the network, 4 million random kinematic configurations of joint angles with their equivalent Cartesian positions were utilized. During the creation of the dataset, we made sure that the joint angle values uniformly cover the ranges of equation 5.3. Because of the size of the dataset, the network was trained with a batch size of 100 units and 30 epochs. Also, 10% of the dataset was used for cross validation and 10% was used for testing purposes.

After the training is complete, the networks achieved 99.997% validation accuracy. To demonstrate the effectiveness of the network, in this section we will compare the network estimations with the output of the forward equation as computed by equation 5.1 for the same input joint trajectory samples. Figure 5.4 shows the sample trajectory in joint space.

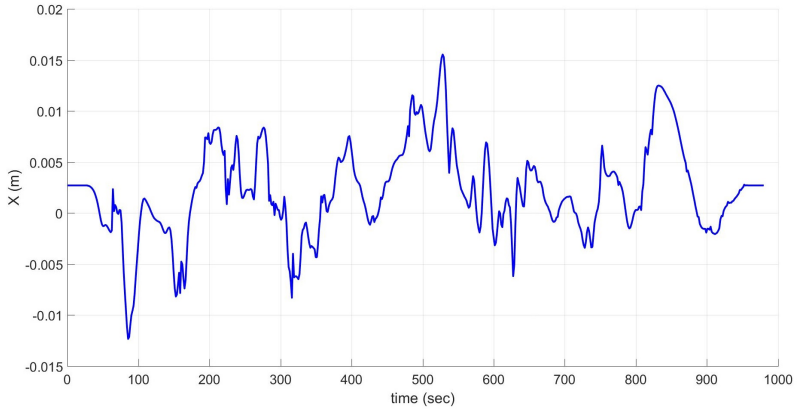


Figure 5.4. Error between the forward kinematic equation and the network in the x dimension.

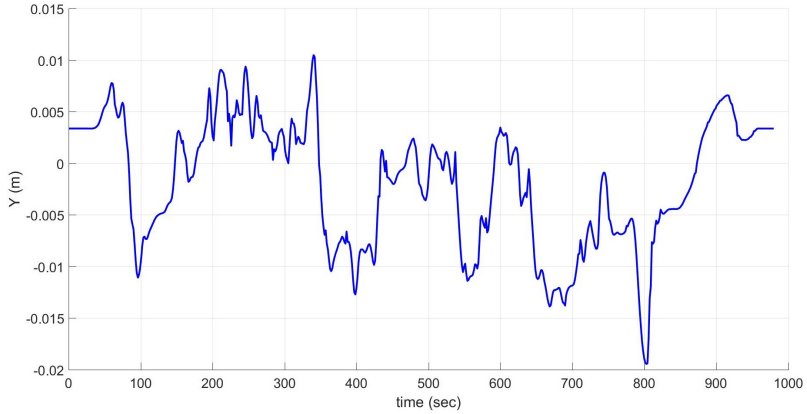


Figure 5.5. Error between the forward kinematic equation and the network in the y dimension.

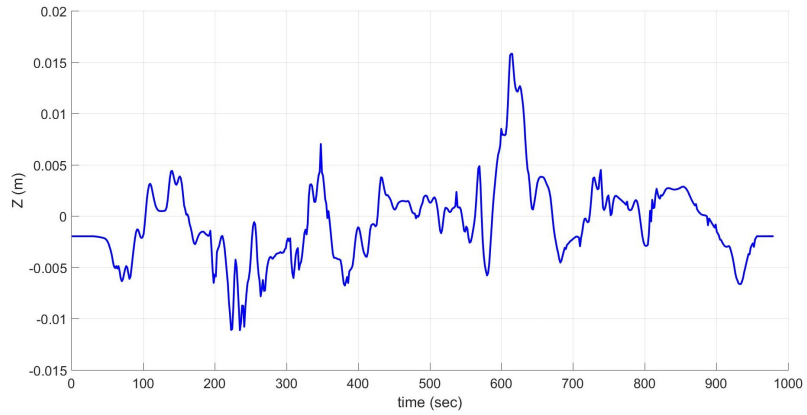


Figure 5.6. Error between the forward kinematic equation and the network in the z dimension.

The difference between the estimations of the forward kinematic equations and the proposed network is shown in figures 5.4, 5.5 and 5.6, where every figure represents one of the Cartesian dimensions of the robot's end effector. Note that the scale in the vertical axis is in meters.

Figure 5.7 illustrates the augmented dataset of the inverse model. The green trajectory depicts the demonstration data in Cartesian space, while the other trajectories were generated with a variety of different splines. In total 125 trajectories were used to train the inverse model. The architecture of the inverse model is a multi-

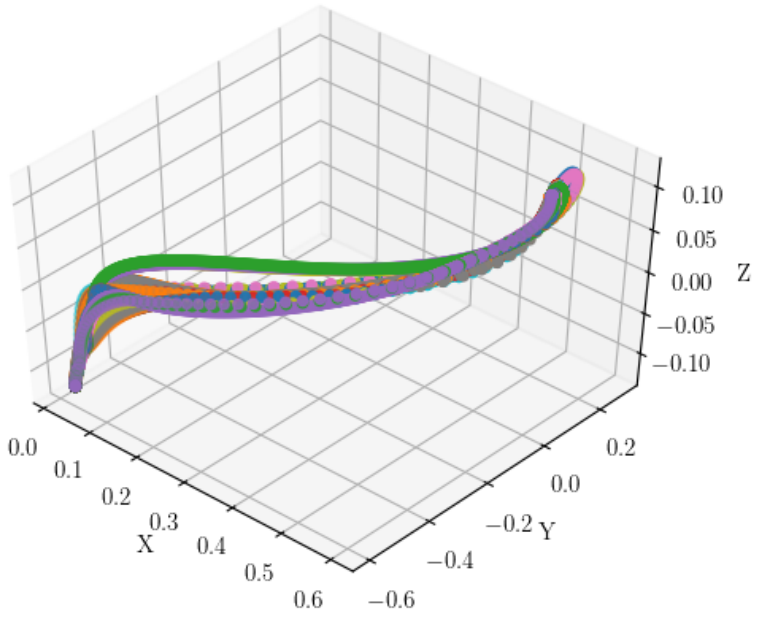


Figure 5.7. Inverse Model Training Data.

layered feed-forward neural network with three inputs and as many outputs as the DoFs of the robot. The input is the Cartesian vector $\hat{x}_e(t) = x_e(t) - \hat{\delta}$ and the output is the equivalent joint velocity vector. The network contained three layers with 300, 200 and 100 sigmoid units and it was trained using the back-propagation algorithm with the mean squared error loss function. During the back-propagation process we used the Adam optimizer.

5.1.2 Phase Two: Policy optimization

To enter phase two, a stopping condition must be met. This stopping condition is determined by a function that measures the error between the robot's end-effector and the goal position. Phase two starts by learning the motor policy using the trajectory data that was computed from the forward and inverse models.

To estimate the transformation system, the forcing terms and the canonical system of the DMP component of the system. τ is the final time T that represents how long it took the inverse model to drive the robot's end-effector to the target, x_0 and g are the initial position and target position of the robot's joint. The terms a_z and β_z are control gains selected to make the system critically damped and f acts as the control input that drives the system towards the goal.

By using the Locally Weighted Regression (LWR) algorithm we learn the parameters of the motor policy as expressed in [4]. This motor policy constitutes a suboptimal policy, which is then passed to the RL module to be further optimized to reach the goal. According to the DMP framework, the learned suboptimal policy $\hat{\pi}_t(s, \alpha)$ can be expressed as a battery of Gaussians that expand in the s domain. For the purposes of this study, the shape of the Gaussians can only be altered if the parameters w_i are changed. As such, the RL module utilizes the Policy learning by Weighting Exploration with the Returns (PoWER) algorithm to find the optimal set of parameters w_i that generate the optimal policy $\pi_t^*(s, \alpha)$ which in turn drives the robot to the goal $\hat{\delta}$, as described in Chapter 2. Note, that if the learned DMP can generalize its motor policy to reach the goal, the RL is not expected drastically modify the Gaussians of the policy. On the other hand, if the initial policy $\hat{\pi}_t(s, \alpha)$ can not generalize its action space to reach the goal, the RL module must alter the weights of the Gaussians accordingly.

5.2 Experimental Section

In this section, we present our hypothesis which quantifies how much the forward and inverse model improves environmental adaptation. Based on this hypothesis, we provide an experimental protocol that evaluates the effectiveness of the proposed system. We continue by providing some of the technical details of the experimen-

tal set-up and the implementation details of the second phase, such as the reward function. We conclude this section by presenting the results of the experiments.

5.2.1 Hypothesis

Since the purpose of the models is to provide an initial guess of the optimal policy, the evaluation metric has to measure how close the initially estimated policy is to the optimal one. If the RL module requires a single episode to optimize the estimated policy, it means that the estimated policy is the optimal one. Thus, if we have two policies $\pi_t(\hat{s}, \alpha)_1$ and $\pi_t(\hat{s}, \alpha)_2$ and they require e_1 and e_2 episodes to converge to the optimal policy, if $e_1 > e_2$ for any given goal g, then we can assume that $\pi_t(\hat{s}, \alpha)_2$ is more adaptable than $\pi_t(\hat{s}, \alpha)_1$.

5.2.2 Experimental Protocol

The experimental protocol includes two different sets of grasping scenarios with the Sawyer Robotic Arm. In the first scenario, the robot attempts to grab a small cube from a table, while in the second one the robot has to learn how to grab a cylindrical cup from the table. In both scenarios the robot's initial position in joint space is $q(t = 0) = [0°, -50°, 0°, 120°, 0°, 0°, 0°]$. For each scenario a total of five different experiments were conducted by placing the graspable object in five different goal locations δ. The Cartesian location of the goals according to the world frame of the robot were:

$$\hat{\delta}_1 = [0.237, -0.466, -0.04]$$

$$\hat{\delta}_2 = [-0.217, -0.474, -0.04]$$

$$\hat{\delta}_3 = [1.110, -0.551, -0.04]$$

$$\hat{\delta}_4 = [-0.179, -0.678, -0.04]$$

$$\hat{\delta}_5 = [0.217, -0.680, -0.04]$$

Note that for the cube grabbing experiments the final orientation of the end effector was set to $[180°, 0°, 90°]$, while to grasp the cylindrical cup, the orientation was set to $[-135°, -90°, 135°]$.

Before the experimental procedure, we performed a demonstration with kinesthetic teaching, depicting how the robot can grab a small cube from a goal location $\hat{\delta}_{Experiment} = [0, 035, -0, 573, -0, 04]$. After we collected the data from the demonstration, we trained the forward and inverse model as suggested in the previous section. In every experiment, the models generate trajectory data to compute the initial policy $\pi_t(\hat{s}, \alpha)$ that is later optimized by the RL module in phase two. To evaluate whereas the models produce an adaptable policy $\pi_t(\hat{s}, \alpha)_{mdl}$, for every experiment we also learn a policy $\pi_t(\hat{s}, \alpha)_{demo}$, which is computed without the help of the models. By comparing the number of episodes that were required for the two policies to converge to the optimal one for all δ_i we can reason according to our hypothesis whereas the models increase the adaptability of the system or not.

5.2.3 Implementation Details

At this point we ought to mention that the motor policies are composed of 20 Gaussian functions. The RL module creates 10 different samples in each episode and discards half of the least successful ones. The success of a rollout is correlated with a user defined reward function. The reward function is defined in equation 5.4 and is inspired by [27]. The reward function contains two branches, which are meant to evaluate a roll-out at it's final state ($t = T$) and throughout it's duration ($t \neq T$).

The term w_1 (0 to 1 inclusive) weights the importance of reaching state g at time $t = T$.

$$r(t) = \begin{cases} w_1 \exp\left(-||g - x||^2\right) & \text{if } t = T \\ \frac{(1-w_1)}{T} \exp\left(-||g - x||^2\right) & \text{if } t \neq T \end{cases} \tag{5.4}$$

For the system to collect $\hat{\delta}$ we use a manually-calibrated downward-facing webcam that performs color detection to estimate $\hat{\delta}$ of the location δ with respect to the world frame of the robot. Lastly, a threshold is used to stop the robot once it is within 1cm of the goal.

Experiment	**1**	**2**	**3**	**4**	**5**	**Total**
Models	18	17	14	6	6	61
Demonstration	29	12	19	13	9	82

Table 5.2. Results from 'cube' scenario

Experiment	**1**	**2**	**3**	**4**	**5**	**Total**
Models	14	18	17	8	8	65
Demonstration	17	6	15	12	13	63

Table 5.3. Results from 'cup' scenario

5.2.4 Experimental Results

The final results of the experiments can be seen in table 5.2 for the first scenario and in table 5.3 for the second scenario. The first line in each table indicates the number of episodes that were necessary to convert policy $\pi_t(\hat{s}, \alpha)_{mdl}$ to the optimal

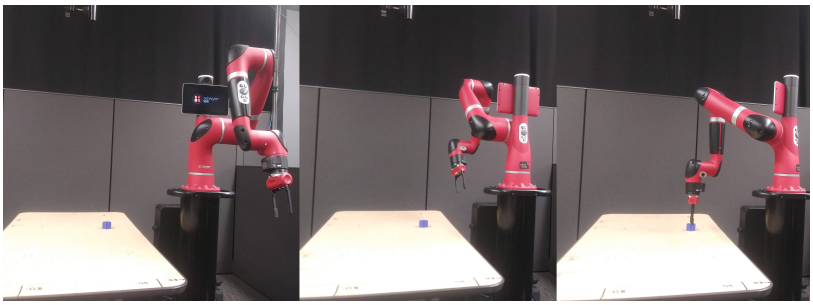


Figure 5.8. Scenario 1: The robot performs the second experiments and attempts to grab the cube at $\hat{\delta}_2 = [-0.217, -0.474, -0.04]$..

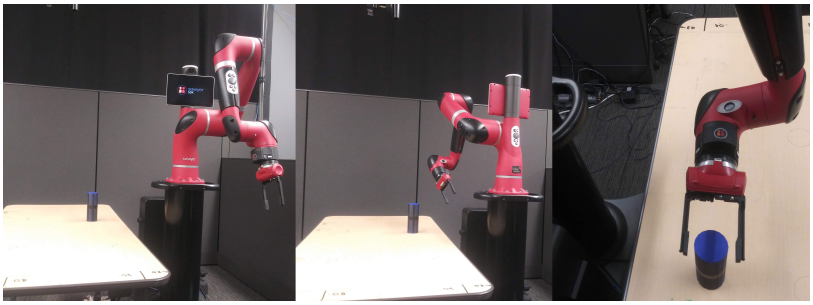


Figure 5.9. Scenario 2: The robot performs the second experiments and attempts to grab the cylinder at $\hat{\delta}_2 = [-0.217, -0.474, -0.04]$..

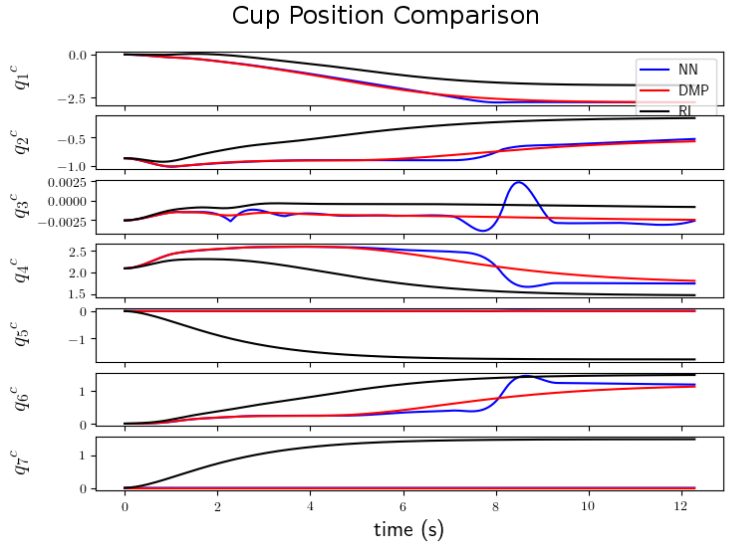


Figure 5.10. Calculated trajectories for scenario 2, experiment 2.

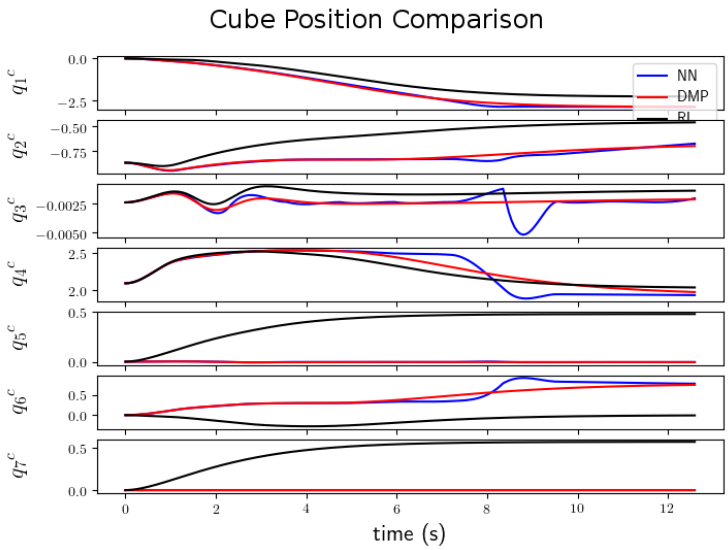


Figure 5.11. Calculated trajectories for scenario 1, experiment 2.

one, while the second line shows the number of episodes for policy $\pi_t(\hat{s}, \alpha)_{demo}$. The columns of the tables represent how many episodes were necessary for the policies to adapt in each experiment and what is the total number of episodes per scenario.

Figures 5.8 and 5.9 illustrate how the system performed in the first and second scenario during the second experiment. Specifically, they show the starting pose of the robot, the final pose of the trajectory that was generated by the inverse model, and the final pose that was performed by the optimal policy. In figures 5.10 and 5.11 the blue line represents the trajectory that was computed by the inverse model that drove the robot to a position near the goal. The red line represents the trajectory of the initial policy and the black line is the optimal policy that enables the robot to reach the goal location.

Since an optimal policy converges in a single episode, with a seven DoF robot and five experiments, there are a total of 45 episodes required. According to our experimental hypothesis and from the results of table 5.2, we deduce that since $\pi_t(\hat{s}, \alpha)_{mdl}$

is closer to the optimal policy in almost every experiment, the models increase the adaptability of the system. However, from the results of table 5.3 we note that the adaptation of both policies is almost identical in the cup grasping scenario, despite the fact that $\pi_t(\hat{s}, \alpha)_{mdl}$ adapted more quickly in experiments one, four and five. The main reason for this phenomenon is that the inverse model was trained on data that involved the cube and not the cup grabbing scenario.

CHAPTER 6

Learning Visuomotor Policies by Combining Movement Primitives and Convolutional Neural Networks

One common characteristic of all animals and humans is their ability to sense, learn and perform actions. Researchers in the field of robotics are aware of this phenomenon and they are trying to incorporate these behaviors in robotic systems. In the first chapter of his book, Craig [12] regards robotics as an interdisciplinary field of science of locomotion, mechanical control, artificial intelligence, and sensor analysis. Most robotic systems are designed by combining multiple components that perform each of these operations separately. One of the groundbreaking effects of the Dynamic Movement Primitive (DMP) frameworks is its ability to learn motor skills. In this chapter, we propose a system that introduces sensory capabilities in the learning loop of the DMP framework.

6.1 Method to learn end-to-end visuomotor policies

In this section, we present a method to learn end-to-end visuomotor policies for robotic arms from demonstrations. The method computes state-action mappings in a supervised learning manner from raw images and motor commands. At the core of the system, a Convolutional Neural Network (CNN) extracts image features and produces motion features. The motion features encode and reproduce motor commands according to the DMP framework. To evaluate the effectiveness of the proposed learning method, we conduct experiments with a PR2 robot in a simulation

environment. The purpose of these experiments is to evaluate the system's ability to control a robot to perform tasks.

In recent years, robotics researchers put substantial effort into building autonomous agents by leveraging the abilities of learning into the classical sense-reason-act loop [56]. Early research in the field of Human-Robot-Interaction (HRI) introduced the notion that robots should learn to perform tasks through human guidance and not programming. This idea is influenced by the human tendency to learn practical skills by observing and imitating other humans [11]. This trend established the area of imitation learning, which lead to the development of Learning from Demonstration (LfD) approaches [1].

LfD algorithms operate in two stages [1]; the demonstration and the execution stage. In the demonstration stage, the LfD method records data that associate with the world state and the action of the demonstrator. The learner then creates a state-action mapping which accomplishes the demonstrated task. This mapping is often referred to in LfD literature as a *policy*. In the execution stage, the robot applies the policy to perform the task in a similar setting. LfD methods can be categorized according to how the data is collected, what features are selected, and how the policy is derived [11], [1].

In recent years, many LfD frameworks have been applied to solve the problem of deriving policies that generalize the behavior of a robot over different state-action mappings. The development of the DMP framework [4] was an essential contribution to this trend, as it provides an abstraction layer that connects the dimensions of state, action, and environment by computing a motor policy with distinct meta-parameters that affect the behavior of the system. However, the experiments reported in [28], [53], [22] have shown that tasks which are composed of a plethora of different sequences of kinematic configurations require multiple DMP policies to be learned.

A typical test case for multiple DMPs is a robot learning tennis by demonstration [28], [53]. Muelling et al. proposes to solve the issue of motor primitives generalization by combining the meta-parameters of multiple DMP policies through regression with a gating network.

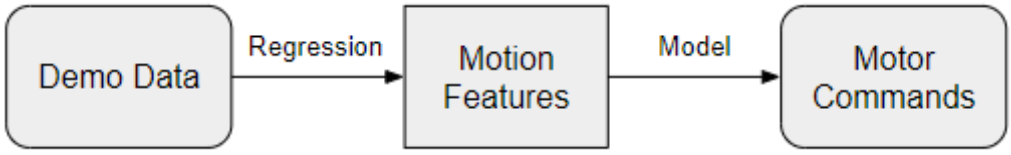


Figure 6.1. LfD with the DMPs.

Figure 6.2. LfD with CNN.

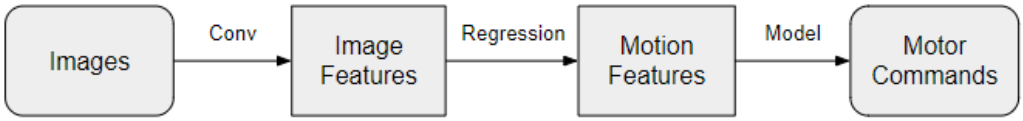


Figure 6.3. Proposed LfD method.

With the development of modern CNNs, researchers began to develop systems that train on perception and control data to learn end-to-end policies from raw images to low-level motor commands [18], [19], such as joint torques. LfD systems that incorporate CNNs have the ability to learn policies from datasets with videos of

demonstrations and their respective joint motor commands [57]. Since CNNs have the capacity to extrapolate visual information on a higher abstract dimension with respect to the presented data [58], LfD methods which combine CNNs can learn to perform tasks from different embodiment [59]. Such is the case when the authors of [6] learn tasks from both human and robot demonstrations to infer a policy of a new task from just one new human demonstration.

Recently, [60], [61] present methodologies that make use of both CNNs and DMPs. Specifically, the authors of [60] augment the learning capabilities of the DMP framework by creating a deep encoder-decoder network that maps raw images to the parameters of a DMP policy. Since the meta-parameters of a DMP policy lie in different dimensions, a custom cost function is used to train the network. Similarly, [61] proposes a two-step learning method that combines CNNs and DMPs. In the first step, a CNN learns to map images to task parameters, while in the second step a fully connected neural network converts the learned task parameters and a clock signal to DMP parameters.

Overall, the DMP framework has the capacity to transform motion trajectories, which is a high-dimension space, to specific motion features, which is a low-dimension latent space, via regression. The motion features are then converted into motor commands by employing a dynamic model as suggested in figure 6.1. On the other hand, CNN LfD methods utilize convolutions to extract image features from images and then output motor commands via regression. As such, both DMP and CNN frameworks have the capacity to extract low-dimensional features of motion and images respectively and map them to motor commands. In this chapter, we propose an alternative LfD method to compute visuomotor policies by combining characteristics of both DMPs and CNNs. Figure 6.3 illustrates our proposed method that utilizes convolutions to extract image features, maps them with regression to DMP motion

features and then produces motor commands with dynamic models. The novelty of the proposed method is that regression is not performed between a high and low dimension space, but between two low-dimension latent spaces, which often yields more successful results [62].

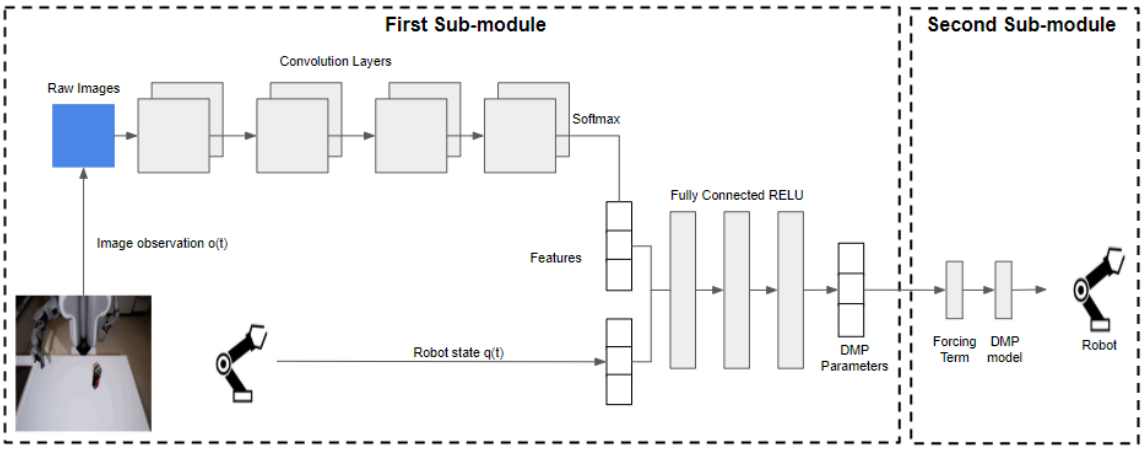


Figure 6.4. The architecture of the proposed learning method depicting the end-to-end system.

6.2 Architecture of the Proposed Learning Method

In this section, we will provide an analysis of the proposed learning method. Figure 6.3 suggests that the proposed method has two sub-modules that operate in order. The first one is composed of a CNN that maps raw images and the state of the robot to DMP parameters, while the second sub-module converts the DMP parameters to motor commands. An analytical illustration of the system can be seen in Figure 6.4.

6.2.1 Data Pre-processing

For training, the learning method requires a dataset D containing multiple demonstrations of each training task. Every demonstration lasts a specific amount of time which is denoted as the time vector $t = [0, .., t_i, .., t_f]$, where $t_f = \tau$ is the duration of the demonstration and i symbolizes the number of data points per demonstration. Furthermore, we collect observations o_i, which are RGB images of the demonstration at time t_i, and we record the robot state vector of joint angle positions q_i at time t_i. The DMP parameters for each task are computed using q and t. Specifically, a fixed number of interval recordings i and Gaussians in equation 2.3 are set. The number of Gaussians, control gains a_z, b_z, and the variable α are set to constants according to [4], enabling us to derive phase variable s from equation 2.4. The starting position x_0 and goal g is assumed to be the initial $q(t = 0)$ and $q(t = T)$ terminal state of the robot for each demonstration. To compute the vector weights w_i we used the LWR algorithm [4] as previously in chapter 2. Thus, only the goal g, starting pose x_0, phase s, and weights w_i are changing in each demonstration. This means that these variables are the parameters of the DMP motor policy that encode and generalize the joint trajectories of the demonstration.

6.2.2 First Sub-module Architecture

The network's architecture is inspired by [18], [19], and [6] in terms of functionality and composition. We proceed to training the first sub-module after appending the DMP parameters to D and normalizing the dataset. The observations o_i are used as input and DMP parameters as output. The inputted RGB images are segmented into 3 channels and their dimensions reduced to 125x125. Next, they are fed into 4 convolution layers with 16 filter of size 4 and stride of 4. The output of the last

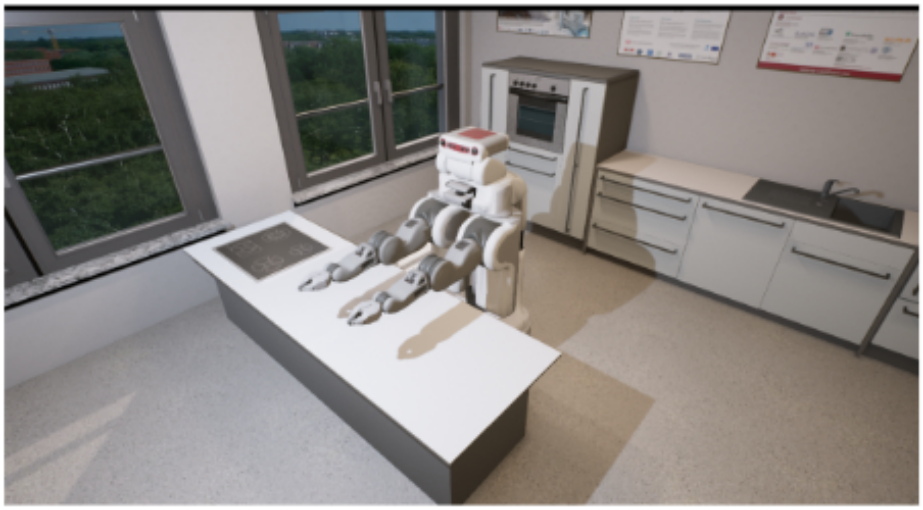


Figure 6.5. Simulation Environment in Unreal Engine.

Figure 6.6. Robot reaches an object from the side.l Engine.

Figure 6.7. Robot reaches an object from the top.

convolution layer is transformed into spatial features with the use of the soft-argmax operator [18], [19]. The spatial image features are then fused with the state of the robot q_i, which is represented by a vector of joint positions. Note that during the concatenation of the features and the state, we add a bias term to make the image feature dimensions less dominant [18], [19], [58]. The fused dimension is then passed

to 3 fully connected layers with 200 units and ReLU activation units. The output of the network represents the DMP parameters, which are used to derive the new forcing term that drives the robot according to equation 2.1 of the transformation system in Chapter 2.

6.3 Experimental Section

For the experiments, we have made use of a game engine-based simulation. In this section, we first introduce the simulation and describe the experimental framework. Next, we present test cases to challenge the robustness of the learning method. We continue by elaborating on the data collection method and explain the metrics of evaluation. Finally, we present and discuss the experimental results.

6.3.1 Experimental Framework

Figure 6.4 provides an illustration of the simulation environment. The environment contains a simulation of a PR2 robot in a virtual kitchen setting. The simulation was developed with Unreal Engine due to its photo-realistic renderings and powerful physics engine. All object in the simulation, including the robot, can be controlled via ROS (using a ROS bridge). The proposed method was implemented with TensorFlow.

6.3.2 Test cases

The training data is acquired with simulated tasks depicting various reaching motions using the robot's right arm. The left arm remains still throughout the experiment. Each demonstration begins with the same starting arm pose while the target objects are placed in novel positions and orientations. The duration of the demonstration, τ, remains constant. This means we can extract the DMP parameters of each demonstration and reduce the number of parameters needed to be learned by the

system as the goal g and weight vector w in equation of the forcing term are sufficient for reproducing the movement. The size of the DMP parameter vector (shown in figure 6.4) is based on the resolution of the DMP forcing term (i.e. the number of Gaussians basis functions) and the degrees of freedom of the manipulator (which is seven in our case with the PR2). The simplification in representation enables training with mean squared error as opposed to custom cost functions as proposed in [60].

Additional variance is introduced within the dataset by introducing different reaching paths from the initial position of the robot to the object. As Figures 6.6 and 6.7 suggest, the robot is able to reach the object from the side or from the top. To be exact, if the observed object is cylindrical in shape, the robot is ordered to approach the object sideways, while if the object is cube-like, the robot approaches to object from the top. The end-to-end network recognizes this automatically and is thus able to infer and select a primitive accordingly.

The trajectory of the demonstration is generated by finding the inverse kinematics solution of a position near the object, and then interpolates polynomial joint space trajectories leading from the starting joint position. When the joint trajectories are calculated, they are passed to the position controller of the simulated robot and as the robot executes the trajectory, a camera within the simulation records the observations o_i.

To establish a metric of evaluation, we decided to train an additional CNN. The second neural network has a similar architecture as the one in Figure 6.4, with the exception that it learns to produce joint velocity commands dq_i in every interval based on the observation o_i and robot state q_i. The proposed learning method encapsulates the network that outputs DMP parameters and the second network which outputs velocities. After both networks are trained with data from the same demonstrations, we compare how the robot behaves while being controlled by each network in a set of

test cases. In these cases, new objects are introduced and are placed in a set of novel location with respect to the torso of the robot. For every object and every location, we proceed by comparing how each network commands the robot to reach the object.

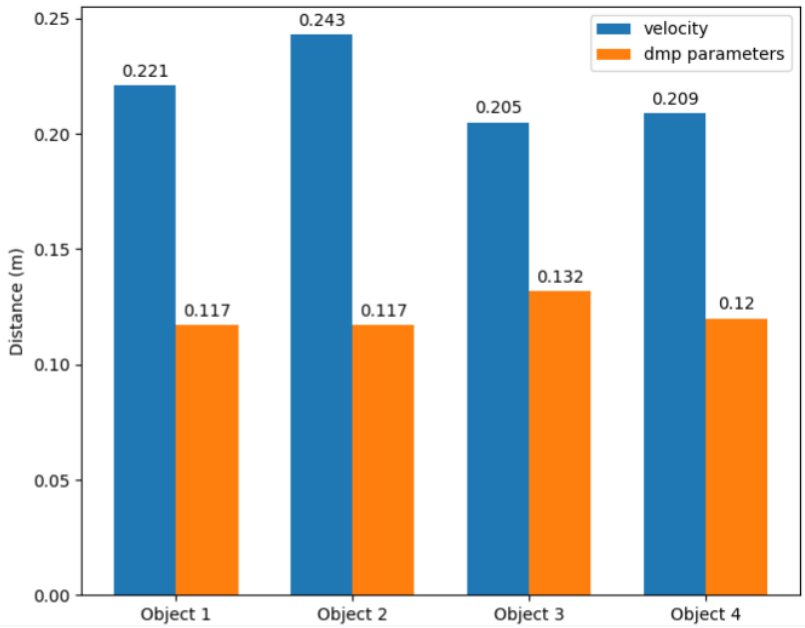


Figure 6.8. Error between end effector and object..

6.3.3 Implementation Details

As seen from Figures 6.6 and 6.7 the robot's right arm always start from $q(t = 0) = [-90°, -70°, 0°, 0°, -70°, 0°, 0°]$. The forcing term of the DMPs is characterized by 20 Gaussian functions. The dataset contains demonstrations of 27 different objects, which are placed in 22 random locations. Each demonstration lasted 4 seconds divided into 100 intervals. During testing, 4 different new objects were placed in the following 10 locations δ for a total of 40 cases. As aforementioned, for each test case we compare how the network that learns DMPs and the network that learns velocities, control

the robot. The Cartesian positions (x-y-z) and orientations (yaw angle in radians) of all the δ with respect to the torso frame of the robot are as follows:

$$\delta_1 = [0.488, 0.009, -0.33, -0.0105]$$

$$\delta_2 = [0.682, -0.029, -0.33, 0.0963]$$

$$\delta_3 = [0.608, 0.07, -0.33, -0.232]$$

$$\delta_4 = [0.615, -0.255, -0.33, -1.522]$$

$$\delta_5 = [0.514, 0.212, -0.33, -0.6333]$$

$$\delta_6 = [0.478, 0.147, -0.33, -0.8206]$$

$$\delta_7 = [0.627, -0.123, -0.33, 0.9057]$$

$$\delta_8 = [0.521, -0.285, -0.33, 0.3928]$$

$$\delta_9 = [0.571, -0.038, -0.33, 0.8010]$$

$$\delta_{10} = [0.627, -0.183, -0.33, 0.9661]$$

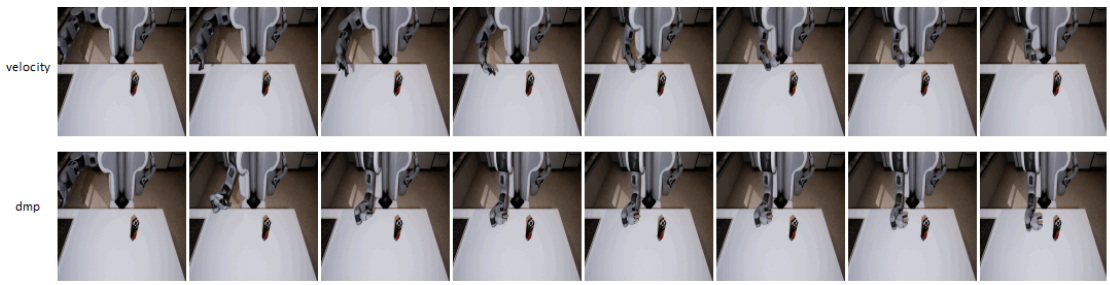


Figure 6.9. Test case for object 1 and δ_4.

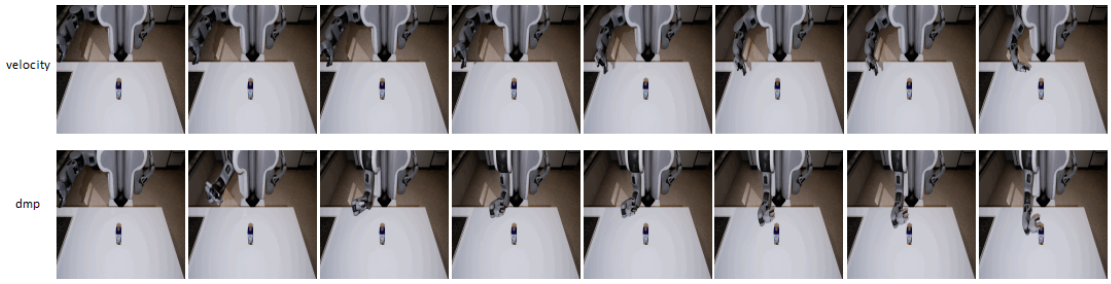


Figure 6.10. Test case for object 2 and δ_2.

Figure 6.11. Test case for object 3 and δ_8.

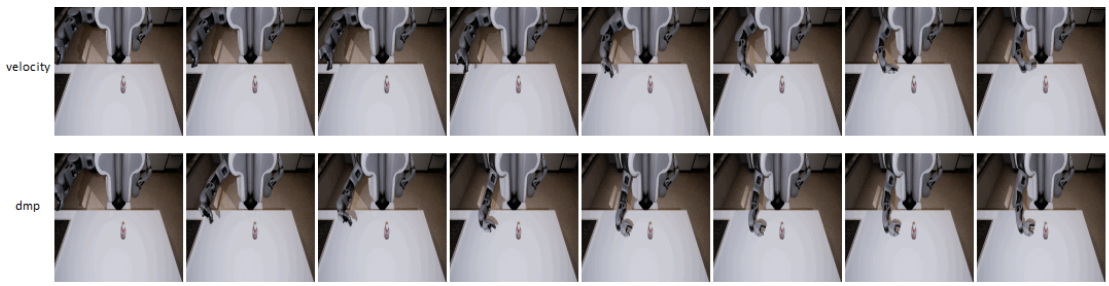


Figure 6.12. test case for object 3 and δ_1.

6.3.4 Experimental Results

The first metric of evaluation is the euclidean distance of the robot's end effector at the terminal state from the center of the object. This metric is selected to establish which system is more successful at guiding the robot near the object. Figure 6.8 presents a comparison between the two methods with regards to the average error

distance in meters from the center of the objects for all 10 locations δ per object. We notice that the network that learns velocities has an average distance error of 20 cm in comparison to the proposed method that has an average error of about 11 cm.

Although Figure 6.8 shows that the proposed method is successful at guiding the robotic arm near the object, we present further comparisons. Figures 6.9 to 6.12 and Figures 6.13 to 6.16 provide additional comparisons between how the robot behaves when controlled by each network. Both Figures illustrate how each network guides the robot when presented with the 4 different object for cases δ_4, δ_2, δ_8, and δ_1 respectively. Figures 6.9 to 6.12 shows sequences of images that capture the behavior of the robot while it is being controlled by each network individually. Figures 6.13 to 6.16 presents the equivalent Cartesian trajectories that the robot's end effector followed. A common trend in all cases is that the robot's arm performed an arch shape trajectory when it is controlled by the network that produces velocities. In contrast, the network that learns DMPs guides the robot thought trajectories which are characterized by over-shots and oscillations. Despite this phenomenon, the proposed method retained a sense of goal and direction in comparison to the velocity network. In Figures 6.13 to 6.16 we highlight that despite the perturbations, the blue trajectory converges to a position near the object's true position, which is denoted as a red dot.

Another noteworthy event is that the proposed methodology retained the form of the primitive movement of the trajectory. This is apparent for example in Figure 6.11 where the robot is trying to reach the object from the top and not sideways, which happens in the rest of the Figures. This can be attributed to the fact that the object of Figure 6.11 is not cylindrical in shape and thus the network chooses to approach it from the top. The exact opposite happens in the rest of the examples where the robot approaches the object sideways because they are of cylindrical shape.

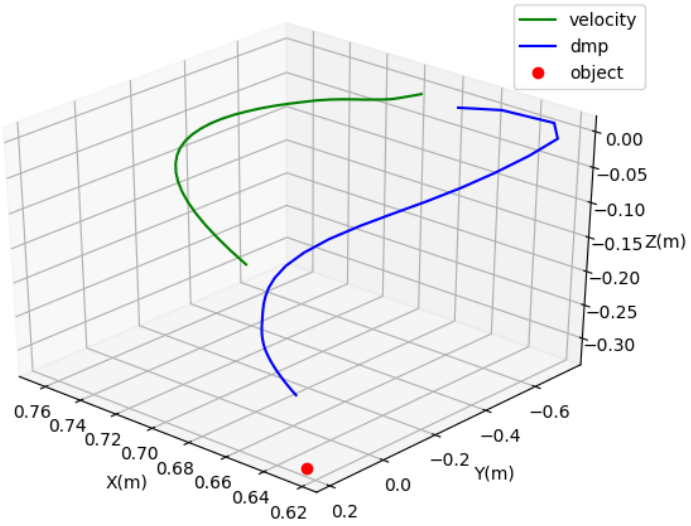


Figure 6.13. Test case for object 1 and δ_4.

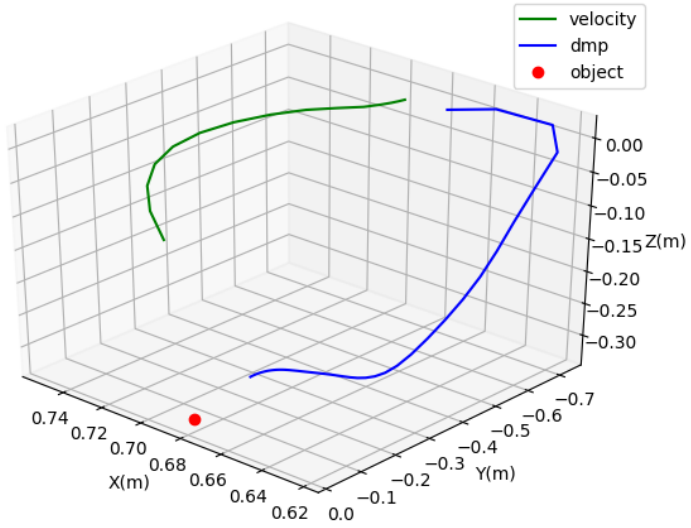


Figure 6.14. Test case for object 2 and δ_2.

We found that the network that learned DMPs created a mapping between images and the parameters of the forcing term of a spring damper system which leads the robot arm to a specific goal. This explains why the robot performed oscillations

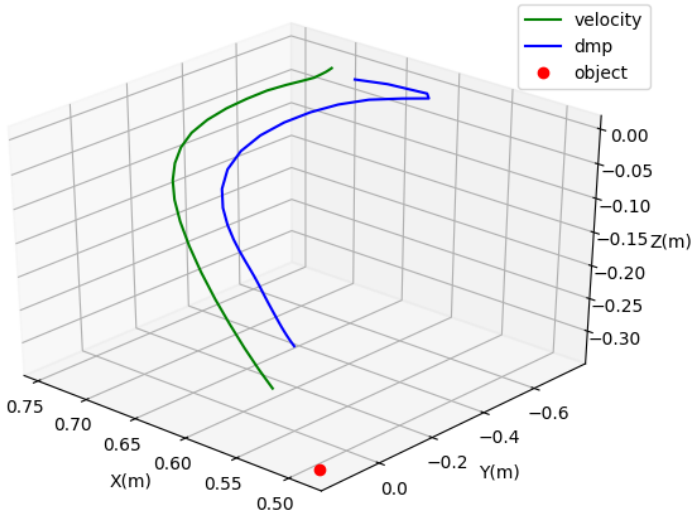


Figure 6.15. Test case for object 3 and δ_8.

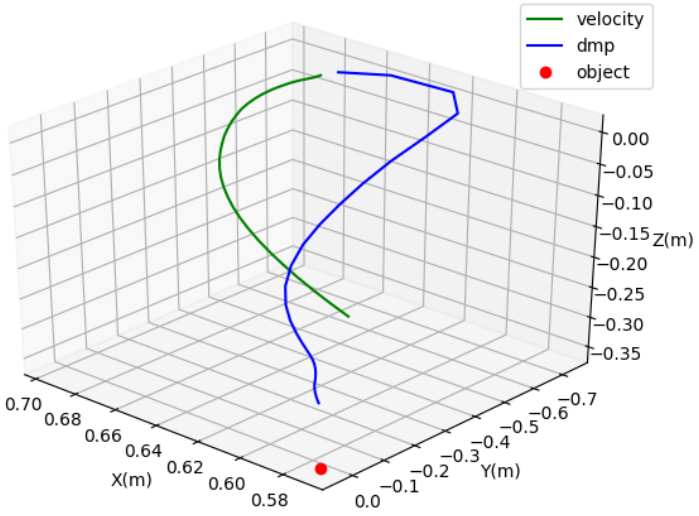


Figure 6.16. Test case for object 3 and δ_1.

and over-shots in the experiments, but retained the form of the learned trajectory and the goal. The generated trajectory is characterized by oscillatory motion due to lack of data and not optimizing the neural network hyper-parameters. Moreover

the velocity network that was used for testing interpolated between the two different primitive motions. This result can be explained by other relevant studies such as [28], [53]. An intuitive explanation is that the geometric interpolation of a top down and a side ways approach to an object will result in an arching trajectory.

Due to a couple of simplifications that were made in the experimental section, the proposed learning method did not require a custom cost function like [60]. It must be mentioned though that deriving a custom cost function for every DMP parameter can be a challenging task, especially when the network architecture becomes more complex. However, optimizing the network's parameters with respect to such a function will result in better overall performance.

CHAPTER 7

Epilogue

7.1 Summary

This thesis explores concepts and ideas from a variety of different scientific fields that associate with robotics, such as robot kinematics and dynamics, computer vision, control theory, and machine learning. Some of the examined techniques are model-based such as robot kinematics and robot dynamics, while others are model-agnostic, such as the Dynamic Movement primitives (DMP) framework and Convolutional Neural Networks (CNNs). Each proposed system utilized a combination of these techniques intending to synthesize semi-autonomous robotic systems for robot manipulation for a variety of different applications.

Throughout this research, we present the potential and advantage of fusing model-based with model-agnostic approaches. The intuition behind this idea is that model-based approaches are very robust and accurate, but can't generalize over different environmental parameters, while model-agnostic techniques achieve the opposite. As such, the combination of model-based approaches, in the form of deterministic mathematical models and formulas, and model-agnostic approaches, which is advocated through the use of machine learning, can have beneficiary effects in an intelligent system.

7.2 Conclusion and Future Work

The DMP framework can enable a robot to act and learn based on a control module and a learning component. However, sensing is a modality that is absent from

the learning loop of the DMP framework. In this thesis, we attempted to augment the learning capabilities of the DMP framework by incorporating DNNs. The research of Chapter 6 proposes a novel learning method to fuse sensing, learning, and motion by learning end to end visuomotor policies by combining CNNs and DMPs.

The proposed method of Chapter 6 performed better than the state of the art approach since it learned to distinguish between different behaviors with regards to visual environmental stimuli. On the other hand, traditional frameworks interpolated between motor commands to produce negative results. Thus, the parameter space of the DMP framework is a latent space that concatenates information about motion. It also proves that it is a suitable dimension to learn motor skills with regression.

In essence, the proposed method learned the parameters of a forcing term, which drives an attractor system that behaves as the mass-spring-damper system. As such, the system learned the non-optimal parameters of a controller that drives the robot from a specific start location to the desired goal location. Computing the optimal parameters of such a system is difficult with machine learning because the DMP framework learns a general policy and not an optimal one. Note, that is is the reason why the motion of the robot in Chapter 6 was oscillatory.

Finally, although the experiment of Chapter 6 was successful, the system was limited because the CNN only learned a couple of the DMP parameters, such as the weights of the forcing term and the goal of each trajectory. A real-life system must be able to learn the phase and the starting position of the system as well. Thus, a possible avenue for future exploration is to train the model of Chapter 6 with a custom cost function, such as the one presented in [60]. Also, by creating a cost function that takes into consideration the derivatives of the DMPs, the system will become more stable and accurate.

7.3 Published Implementations

One of the products of this thesis is the release of various software implementations to analyze and examine the proposed methodologies. In this section, we provide the systems that are publicly available to promote experimentation by the research community:

1. pyrdmp: Implementation of the DMP framework and the Policy learning by Weighting Exploration with the Returns (PoWER) algorithm. Chapter 2 provides the fundamentals of the implementation. Readers can find the implementation on the following link: https://github.com/heracleia/pyrdmp.
2. Learning Visuomotor Policies by Combining Movement Primitives and Convolutional Neural Networks. The system presented in Chapter 6 can be found in the following link: https://github.com/MichailTheofanidis/CNN-DMP-fusion, while the data-set of the reaching attempts by the robot can be found in: https://github.com/MichailTheofanidis/CNN-DMP-fusion-Datasets-Results.
3. Combining Forward and Inverse Models with Reinforcement Learning for Motor Policy Adaptation. The learning method presented in Chapter 5 along with a library to perform kinematic computations for the sawyer robotic arm can be found in: https://github.com/heracleia/sawyer-nn-pyrdmp.

REFERENCES

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[2] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, 2007.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[4] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[5] J. Kober and J. Peters, "Imitation and reinforcement learning," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.

[6] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine, "One-shot imitation from observing humans via domain-adaptive meta-learning," *arXiv preprint arXiv:1802.01557*, 2018.

[7] G. C. Burdea, "Invited review: the synergy between virtual reality and robotics," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 3, pp. 400–410, 1999.

[8] M. Theofanidis, S. I. Sayed, A. Lioulemes, and F. Makedon, "Varm: Using virtual reality to program robotic manipulators," in *Proceedings of the 10th Interna-*

tional Conference on PErvasive Technologies Related to Assistive Environments. ACM, 2017, pp. 215–221.

[9] M. Ogino, H. Toichi, Y. Yoshikawa, and M. Asada, "Interaction rule learning with a human partner based on an imitation faculty with a simple visuo-motor mapping," *Robotics and Autonomous Systems*, vol. 54, no. 5, pp. 414–418, 2006.

[10] R. Drillis, R. Contini, and M. Bluestein, "w," *Artificial limbs*, vol. 8, no. 1, pp. 44–66, 1964.

[11] C. Sylvain, "Robot programming by demonstration: A probabilistic approach," 2009.

[12] J. J. Craig, *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India, 2009.

[13] R. Crespo, R. García, and S. Quiroz, "Virtual reality simulator for robotics learning," in *Interactive Collaborative and Blended Learning (ICBL), 2015 International Conference on*. IEEE, 2015, pp. 61–65.

[14] A. De Santis, B. Siciliano, A. De Luca, and A. Bicchi, "An atlas of physical human–robot interaction," *Mechanism and Machine Theory*, vol. 43, no. 3, pp. 253–270, 2008.

[15] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[16] L. Peppoloni, F. Brizzi, C. A. Avizzano, and E. Ruffaldi, "Immersive ros-integrated framework for robot teleoperation," in *3D User Interfaces (3DUI), 2015 IEEE Symposium on*. IEEE, 2015, pp. 177–178.

[17] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[19] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[20] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97. Citeseer, 1997, pp. 12–20.

[21] D. C. Bentivegna, A. Ude, C. G. Atkeson, and G. Cheng, "Humanoid robot learning and game playing using pc-based vision," in *IEEE/RSJ international conference on intelligent robots and systems*, vol. 3. IEEE, 2002, pp. 2449–2454.

[22] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE, 2009, pp. 763–768.

[23] O. Kroemer, R. Detry, J. Piater, and J. Peters, "Grasping with vision descriptors and motor primitives," in *Informatics in Control, Automation and Robotics*. Springer, 2011, pp. 211–223.

[24] J. Kober and J. Peters, "Learning motor primitives for robotics," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE, 2009, pp. 2112–2118.

[25] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, 2009, pp. 849–856.

[26] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006, pp. 2219–2225.

[27] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with em-based reinforcement learning," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.* IEEE, 2010, pp. 3232–3237.

[28] K. Muelling, J. Kober, and J. Peters, "Learning table tennis with a mixture of motor primitives," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on.* IEEE, 2010, pp. 411–416.

[29] J. Cloud, M. Theofanidis, J. Brady, and F. Makedon, "Importance of effective teaching in robot motor skill learning," in *Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments.* ACM, 2019, pp. 489–492.

[30] S. Schaal, C. G. Atkeson, and S. Vijayakumar, "Scalable techniques from nonparametric statistics for real time robot learning," *Applied Intelligence*, vol. 17, no. 1, pp. 49–60, 2002.

[31] C. J. van Andel, N. Wolterbeek, C. A. M. Doorenbosch, D. H. E. J. . Veeger, and J. Harlaar, "Complete 3D kinematics of upper extremity functional tasks," *Gait and Posture*, vol. 27, no. 1, pp. 120–127, 2008.

[32] E. Magrini, F. Flacco, and A. De Luca, "Control of generalized contact motion and force in physical human-robot interaction," in *2015 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2015, pp. 2298–2304.

[33] M. Abujelala, A. Lioulemes, P. Sassaman, and F. Makedon, "Robot-aided rehabilitation using force analysis," in *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '15. New York, NY, USA: ACM, 2015, pp. 97:1–97:2. [Online]. Available: http://doi.acm.org/10.1145/2769493.2769582

[34] A. Lioulemes, P. Sassaman, S. N. Gieser, V. Karkaletsis, F. Makedon, and V. Metsis, "Self-managed patient-game interaction using the barrett wam arm for motion analysis," in *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '15. New York, NY, USA: ACM, 2015, pp. 34:1–34:8. [Online]. Available: http://doi.acm.org/10.1145/2769493.2769517

[35] H. Zhou and H. Hu, "Human motion tracking for rehabilitation-A survey," *Biomedical Signal Processing and Control*, vol. 3, no. 1, pp. 1–18, 2008.

[36] S. Parasuraman, C. Y. Kee, and A. Oyong, "Human upper limb and arm kinematics for robot based rehabilitation," *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, pp. 845–850, 2009.

[37] S. Parasuraman, A. W. Oyong, and V. Ganapathy, "Development of robot assisted stroke rehabilitation system of human upper limb," in *2009 IEEE International Conference on Automation Science and Engineering*. IEEE, 2009, pp. 256–261.

[38] J. J. Craig, *Introduction to robotics: mechanics and control*. Pearson Prentice Hall Upper Saddle River, 2005, vol. 3.

[39] L. Ferrajoli and A. De Luca, "A modified newton-euler method for dynamic computations in robot fault detection and control," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3359–3364, 2009.

[40] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1297–1304. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2011.5995316

[41] Q. Wang, G. Kurillo, F. Ofli, and R. Bajcsy, "Evaluation of pose tracking accuracy in the first and second generations of microsoft kinect," in *Healthcare Informatics (ICHI), 2015 International Conference on.* IEEE, 2015, pp. 380–389.

[42] M. S. Erden and A. Billard, "End-point impedance measurements at human hand during interactive manual welding with robot," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 126–133, 2014.

[43] S. Devine, K. Rafferty, and S. Ferguson, "Real time robotic arm control using hand gestures with multiple end effectors," in *Control (CONTROL), 2016 UKACC 11th International Conference on.* IEEE, 2016, pp. 1–5.

[44] B. F. Yousef, "A robot for surgery: Design, control and testing," in *Advances in Robotics and Virtual Reality.* Springer, 2012, pp. 33–59.

[45] Y. Pititeeraphab, P. Choitkunnan, N. Thongpance, K. Kullathum, and C. Pintavirooj, "Robot-arm control system using leap motion controller," in *Biomedical Engineering (BME-HUST), International Conference on.* IEEE, 2016, pp. 109–112.

[46] D. Bassily, C. Georgoulas, J. Guettler, T. Linner, and T. Bock, "Intuitive and adaptive robotic arm manipulation using the leap motion controller," in *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of.* VDE, 2014, pp. 1–7.

[47] J. Artal-Sevil and J. Montañés, "Development of a robotic arm and implementation of a control strategy for gesture recognition through leap motion device," in *Technologies Applied to Electronics Teaching (TAEE), 2016.* IEEE, 2016, pp. 1–9.

[48] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, "Humanoid motion planning for dual-arm manipulation and re-grasping tasks," in

Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on. IEEE, 2009, pp. 2464–2470.

[49] G. K. Singh and J. Claassens, "An analytical solution for the inverse kinematics of a redundant 7dof manipulator with link offsets," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.* IEEE, 2010, pp. 2976–2982.

[50] G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with leap motion and kinect devices," in *Image Processing (ICIP), 2014 IEEE International Conference on.* IEEE, 2014, pp. 1565–1569.

[51] M. I. Jordan and D. E. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognitive science*, vol. 16, no. 3, pp. 307–354, 1992.

[52] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Autonomous Robots*, vol. 33, no. 4, pp. 361–379, 2012.

[53] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

[54] M. Theofanidis, S. I. Sayed, J. Cloud, J. Brady, and F. Makedon, "Kinematic estimation with neural networks for robotic manipulators," in *International Conference on Artificial Neural Networks.* Springer, 2018, pp. 795–802.

[55] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[56] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[57] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2786–2793.

[58] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1126–1135.

[59] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-shot visual imitation learning via meta-learning," *arXiv preprint arXiv:1709.04905*, 2017.

[60] A. Gams, A. Ude, J. Morimoto, *et al.*, "Deep encoder-decoder networks for mapping raw images to dynamic movement primitives," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–6.

[61] A. Pervez, Y. Mao, and D. Lee, "Learning deep movement primitives using convolutional neural networks," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 191–197.

[62] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

BIOGRAPHICAL STATEMENT

Michal Theofanidis was born in Athens, Greece in 1989. In 2013 he completed his undergraduate studies at the University of Piraeus, Greece with a B.Sc. from the Department of Digital Systems and he obtained his M.Sc. from the Department of Engineering Mathematics at the University of Bristol, UK. After his graduation he worked as a Research Assistant at the National Center of Scientific Research Demokritos in Greece for a semester.

During his Ph.D. research, he had the opportunity to work for two large National Science Foundation (NSF) grants (US-government funded), the iRehab NSF center project (NSF 1338118) and the iWork project (NSF 1719031). The iRehab center sponsors projects that include personalized rehabilitation therapy for individuals suffering from motor disabilities and cognitive impairments. Moreover, his contribution to the iWork project was to build a smart robot teleoperation system for manipulation, to ensure safe human-robot interaction in intelligent manufacturing or Industry 4.0. Specifically, he designed a robot Learning from Demonstration (LfD) imitation learning framework, which assumes that a robot learns motor skills from a human teacher via kinesthetic teaching.

During the summer semester of 2019, he interned at the Institute of Artificial Intelligence at the University of Bremen in Germany under the supervision of Prof. Michael Beetz. His work there involved learning end-to-end visuomotor policies by combining Convolutional Neural Networks and Dynamic Movement Primitives.