LEARNING OF TASKS WITH ROBOT PROGRAMMING BY DEMONSTRATION

A Thesis Submitted to
the Graduate School of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Mechanical Engineering

by
Serdar Hakan ARGÜZ

June 2022
İZMİR

ACKNOWLEDGMENTS

First things first. It is made very clear to me that my achievements depend not only on my effort and will, but also on the gracefulness of the internal and external resistances I so playfully struggle against. Knowing this, I do not dare to start by doing anything but showing proper reverence first to all the factors that allowed me to conduct this piece of work, for the fear of God is the beginning of wisdom.

Secondly, in the midst of despair and a sense of homelessness, just like how Tarzan was raised by the gorillas of nature, I was saved by the giants of culture. It is only thanks to them that I have a place to stand, and a way to be in this world. Even the very institutions I inhabit and the systems that sustain me are their creations. Insufficient as this work is, and inadequate as I am, let this be an attempt to pay an endless debt of gratitude to all those that came before me. Nietzsche once said, "Of all that is written, I love only what a person hath written with his blood." I hope I took no half measures in writing this thesis.

On a more personal level, I would like to express my deepest gratitude to my wonderful supervisors Dr. Kerem Altun and Prof. Şeniz Ertuğrul for their endless support. To the degree that the goal of the academy is not to simply deliver diplomas, but to cultivate a certain mode of personality; a master's thesis is not only a research project but also an initiation ritual. I shall thank my supervisors for not only being amazing scientists, but also proper mentors by showing me the way and embodying all the virtues I aspire to have.

Also, many thanks to my other committee members, Dr. Can Dede, Prof. Serhan Özdemir, and Dr. Lütfi Mutlu, for their deeply appreciated guidance. They are not only the examiners but also the creators of this work in some sense, for they have educated and shaped me throughout the years. This thesis is made possible by not only their teachings but also by the diligence and rigor they engraved on me.

Finally, my family and friends are so dear to my heart. No amount of words would be enough to reciprocate the love and care they showed for me. They instinctually knew when to alleviate some of my burden, and when to entrust me with an even nobler aim. We struggled uphill with many of them, building bonds that remain unshaken in the face of adversity. It is always a delight to have their companionship.

ABSTRACT

LEARNING OF TASKS WITH ROBOT PROGRAMMING BY DEMONSTRATION

Increasingly more unstructured environments of today's industry challenge the robots to have the capability to dynamically adapt to variations in the part sizes and positions. Traditional programming methods fall short of answering such needs. Programming by demonstration is an approach that allows the robots to learn tasks from human demonstrations. Improvements in the generalization of individual tasks that compose the complex assembly operations are an indispensable need for a more extensive adoption of PbD in the industry. This thesis aims to improve the generalization of the peg-in-hole task against variations in the hole positions. It uses the change in the hole position a metric for the novelty of the task and tests the success rate at increasing distances. The relationship between the novelty of the task and its success is examined for two different learning strategies. In the first strategy, only the positional characteristics of the task are learned, whereas both positional and force characteristics are learned in the latter. It is found that the success rate of the task decreases in both cases as the distance increases. However, the hybrid position/force learning strategy outperforms the purely positional one at all distances. As a result, this strategy is experimentally shown to be a valid approach to improve the generalization of the peg-in-hole task for changing hole positions. Incorporation of this strategy with existing frameworks and orientation generalization methods is suggested as future work.

Keywords: Programming by Demonstration, Peg-in-Hole, Generalization, Human-Robot Interaction

ÖZET

İŞLERİN ROBOTLARA GÖSTEREREK PROGRAMLAMA İLE ÖĞRETİLMESİ

Günümüz endüstrisinin hızla değişen iş alanları robotların farklı parça boyutları ve pozisyonlarına adapte olabilmesini gerekli kılıyor. Ancak geleneksel programlama yöntemleri bu ihtiyaca cevap vermeye muktedir değiller. Göstererek öğretme (Programming by Demonstration) robotlara işlerin doğrudan programlama olmadan öğretildiği alternatif bir yaklaşım olarak bu ihtiyaca cevap verebilir. Bu yaklaşımın endüstride daha yaygın kullanımı için montaj işlerinin genelleştirilmesi büyük önem taşıyor. Bu tezin amacı en yaygın montaj işlerinden biri olan çubuk-delik (peg-in-hole) işinin değişen delik pozisyonlarına göre genelleştirilmesidir. Bu hedefe ulaşmak amacıyla iki farklı öğrenme stratejisi giderek artan uzaklıktaki delik pozisyonlarında uygulanmış ve performansları karşılaştırılmıştır. İlk stratejide işin yalnızca hareket karakteristikleri, ikincisinde ise ayrıca kuvvet karakteristikleri de öğrenilmiştir. Bulgulara göre kuvvet karakteristiğinin de öğrenildiği ikinci strateji tüm uzaklık seviyelerinde üstün performans vermektedir. Ayrıca, uzaklık arttıkça bu stratejinin başarısı diğerine nazaran daha az düşmektedir. Sonuç olarak, öne sürülen stratejinin uygunluğu deneysel olarak test edilmiş ve değişen delik pozisyonlarında iş başarısını arttırdığı gösterilmiştir. Gelecek çalışmalar olarak işin delik pozisyonuna ilaveten delik oryantasyonu için de genelleştirilmesi, ve bu stratejilerin literaturdeki uygulamalarla beraber kullanılması önerilmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. Programming of Robots

Robots are machines that can perform a variety of tasks. To the degree that each task is associated with a distinct motion pattern, performing them requires the robot to be able to execute a diverse repertoire of motions. Among all the possible ways it can move, the instruction on which one to choose should be provided to the robot externally. Programming of a robot can be defined as the process with which the knowledge of the motion required to perform a task is communicated to the robot. Consequently, the very aspect that characterizes robots as multi-purpose machines also reveals them as fundamentally programmable devices.

Industrial robots are traditionally programmed by hard-coding the waypoints the robot has to pass through to perform the desired task (Ambhore 2020 , Zhu and Hu 2018). By doing so, the task can be executed repeatedly with great efficiency and precision. This premise, however, only holds for structured environments in which there are no significant changes in the sizes and positions of the manipulated parts. In unstructured environments, on the other hand, which is increasingly the case in the industry, the traditional method becomes tedious and time-consuming as it requires the reprogramming of the robot for each changing condition.

One might propose that improvements in the speed and adaptability of the programming can be a remedy to this problem. Designing more functional interfaces, educating the programmers on efficiency, or even employing agile work strategies can all be considered examples of such efforts. However, these changes belong to the very same line of progress that also generates the constant flux of novelty in the workspaces. Improvements to the traditional programming cannot outrun the rate at which the work is complexified. Answering the demands of the future calls for a fundamentally new take on the programming of robots.

1.2. Programming by Demonstration

Programming by Demonstration (PbD) is a Human-Robot Interaction (HRI) approach to the programming of robots. In this approach, the robot is not explicitly programmed but only informed by the demonstrations of the task performed by a human teacher. The skill and knowledge implicit in those demonstrations are transferred from the teacher to the robot. It is in this way that it is brought one step closer to embodying the

dynamic and autonomous adaptability of that of a human.

Demonstrations of a task allow the robot to not merely to memorize the particular executions of the but also pick up on the characteristics of it. In this way, the robot learns a general model of the task from the demonstrations. This learning indicates that PbD, also called Learning from Demonstration (LfD) (Schaal et al. 1997, Argall et al. 2009, Billard and Grollman 2013, Calinon 2018), is also a Machine Learning (ML) technique for the programming of industrial robots (Hussein et al. 2017, Fang et al. 2019). In terms of the ML terminology, the human demonstrator acts as a supervisor and provides the demonstration data. Moreover, the robot attempts to imitate the teacher according to the data it was provided with.

There are many advantages of PbD over the traditional method that are are widely discussed in the literature (Billard et al. 2008, Billard and Grollman 2013, Calinon 2018, Fang et al. 2019, Ambhore 2020). First and foremost, it can encode human-like movements that would otherwise be challenging to explicitly program. This is the aspect of PbD that allows the transfer of human skills to the robot. As an example to one such skill, the way in which a human expert maneveours a work piece during assembly can be given. The second advantage shows itself in the reduction of programming time and effort. It is often the case that the demonstration of a task takes much less time than its explicit programming. As a result, the time it takes to adapt to changing tasks is greatly reduced. Even more importantly for the aim and scope of this thesis, the robot learns the task instead of merely memorizing it. This gives the robot the capability to generalize the task for changing conditions (Ti et al. 2022, Kramberger et al. 2017, Gao et al. 2019).

1.3. Programming by Demonstration in Assembly Operations

The advantages of PbD are of great benefit to many areas of robotics, one of which is the field of robotic assembly. The goal of performing assembly tasks autonomously for parts with potentially different sizes and positions is an aim that cannot be actualized by the traditionally programmed robots currently residing in the assembly lines (Krüger, Lien, and Verl 2009). The PbD approach, however, shows much more premise in doing so (Kyrarini et al. 2019, Duque, Prieto, and Hoyos 2019, Yang et al. 2014). To promote a more widespread adoption of PbD in robotic assembly, the research in the field focuses on two prominent themes (Zhu 2020, Zhu and Hu 2018).

1.3.1. Frameworks for Assembly Operations

The first theme concerns the development of comprehensive frameworks that address the learning of the entire assembly operations (Ding et al. 2020, Wang et al. 2014,

Tang, Lin, and Tomizuka 2015). These frameworks start the learning process by first decomposing a demonstrated assembly operation into a sequence of tasks. Learning methods are applied to not only the learn the individual tasks but also the whole operation in terms of its relation to those tasks. Upon encounter with a novel assembly operation, an appropriate sequence of these tasks is generated by the framework. This approach resembles the hierarchical learning models in which each task is modeled in terms of its constituent components (Hu and Kuchenbecker 2019). The most common component of an assembly operation is widely accepted to be the peg-in-hole task (Zhu 2020, Zhu and Hu 2018, Ding et al. 2020, Yang et al. 2014). Other common examples include slide-in-the-groove, bolt screwing, and pick-and-place operations. Consequently, frameworks for assembly operations primarily concern themselves with the learning and execution of peg-in-hole tasks.

As one of the recent examples of such frameworks, Ding et al. (2020) use a vision capture system to acquire the demonstrations of an assembly operation involving many peg-in-hole tasks. The relationship between the assembled parts is learned using a visio-spatial skill learning (VSL) algorithm. This allows the framework to design a sequence of tasks that would assemble the parts to reach the correct final configuration, independent from the configuration at which the parts are initially presented. The different types of peg-in-hole objects and how they are showed to the robot can be seen in Fig. 1.1.

Planning the assembly operation in terms of separate peg-in-hole tasks allows the generalization of the operation for changing part configurations. Once the current positions of the parts are visually identified and a sequence of peg-in-hole tasks is planned, the framework generates trajectories that would move the parts from their current configuration to the desired one. However, the execution of these trajectories is not guaranteed to insert the peg into the hole successfully. As a result, the task is not fully generalized, but only a limited success rate is achieved. For example, Ding et al. (2020) generalize the task for different part configurations, as shown in Fig. 1.2, and report a success rate of 83.3% for the individual peg-in-hole tasks.

Duque et al. (2019) also employ a similar framework. Hand movements of a demonstrator performing a toy car assembly are acquired via a vision system. During the generalization of the operation, the framework first uses image recognition to identify the configurations of the parts, and then generates trajectories to perform peg-in-hole tasks using them. A picture of those parts and the completed toy car assembly can be seen in Fig. 1.3. Similar to the previous study, the execution of these trajectories yield success rates changing between 60% to 90%.

The failures in the execution of the peg-in-hole tasks can be attributed to many reasons. Among them, there are two issues that are caused by the use of vision systems (Zhu and Hu 2018). Current frameworks rely on vision systems to estimate the poses of the parts (Ding et al. 2020, Abu-Dakka et al. 2014). However, this cannot be done with a

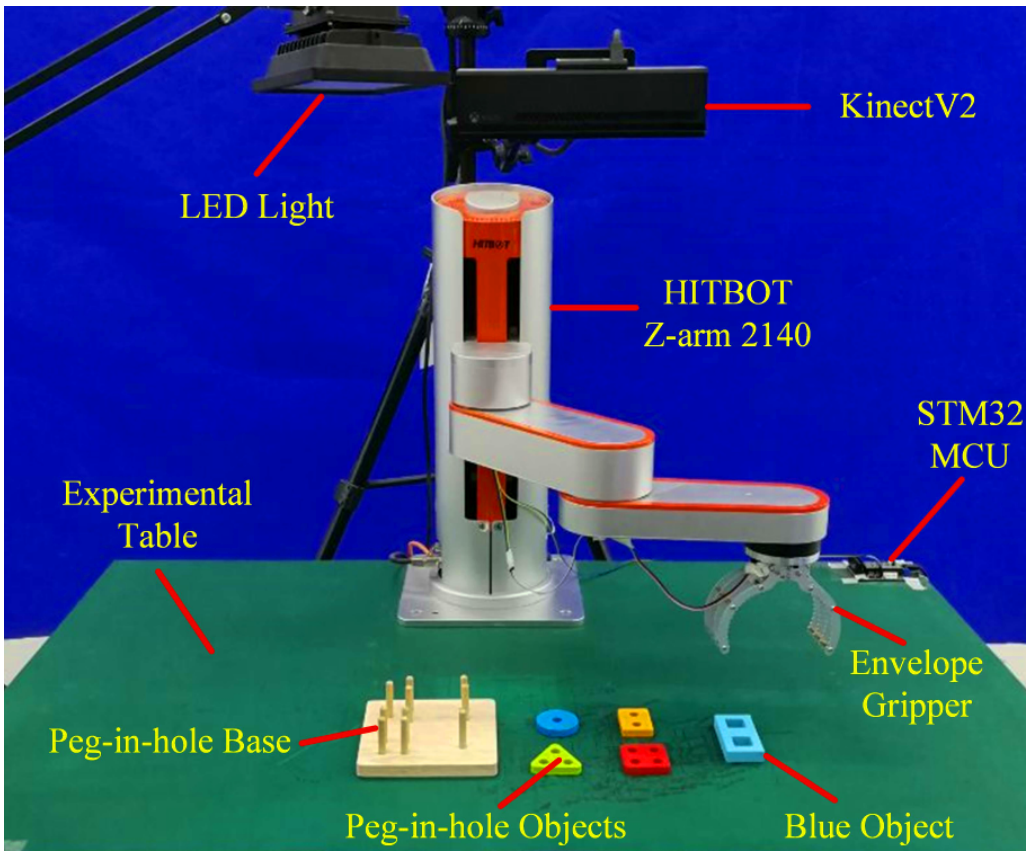

Figure 1.1. The experimental setup used to demonstrate an assembly operation involving multiple peg-in-hole tasks. (Source: Ding et al. 2020)

high degree of accuracy due to the limitations of those systems. If the estimation error is larger than what can be accommodated by the hole clearance, then the robot attempts to push the peg towards a place under which there are no holes. Under these circumstances, the task tends to be not successful. The second problem arises during the execution of the task. While the peg is being inserted, the gripper and the peg cover the hole itself and block the camera's vision. This makes it harder and harder to keep an accurate track of the hole position using the vision system. In these cases, the robot's situation is analogous to a person trying to find a keyhole in the dark.

There is also a second category of failures that are explained by the increased novelty of the situation. Even though PbD frameworks attempt to provide generalization capability under new circumstances, they are not unlimited in their powers to do so. For example, Duque et al. (2019) report four times as many failures in cases with a higher positional variation. Such situations occur when the parts are further away from their positions encountered during the demonstrations. As the discrepancy between the positions of the parts in the phases of demonstrations and execution increases, the generalization performance of the task degrades drastically.

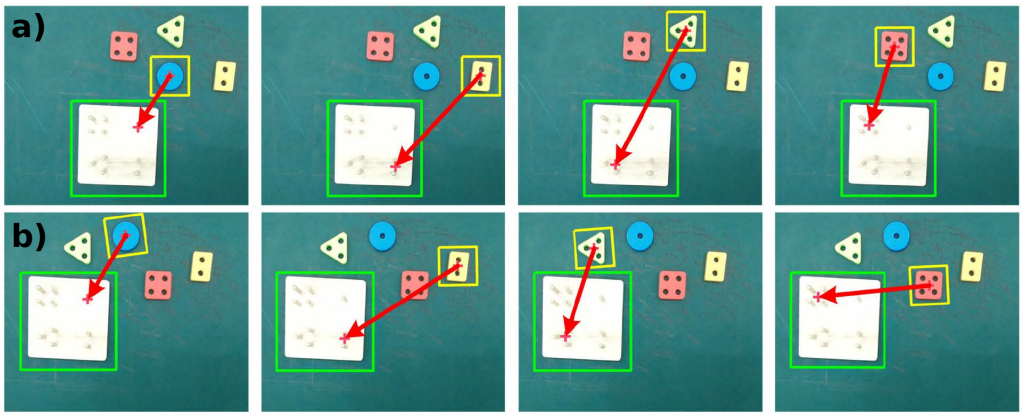

Figure 1.2. a) Configuration of the parts during the demonstrations.
b) Generalizing the task for a novel configuration. (Source: Ding et al. 2020)

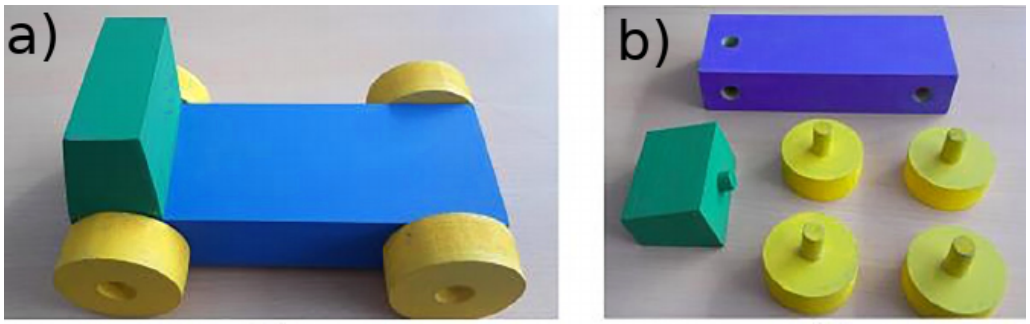

Figure 1.3. a) Toy car, fully assembled b) Parts of the toy car.
(Source: Duque, Prieto, and Hoyos 2019)

1.3.2. Generalization of the Peg-in-Hole Task

Strategies that attempt to address the discussed issues constitute the second main topic of interest in the field. Even though the offered strategies vary significantly in their particulars, a common thread can still be traced among them. This commonality is the incorporation of force and torque characteristics of the task in addition to its kinematics (Skubic and Volz 1998, Nemec et al. 2013, Kramberger et al. 2017, Ti et al. 2022, Tang, Lin, and Tomizuka 2015). In the analogous situation of trying to find a keyhole in the dark, humans rely on force feedback. The field suggests that robots can also do so.

The first category of strategies addresses the lack of accuracy in position estimation. Force feedback can be incorporated to the vision systems to obtain a better measurement of the hole position. For example, Jasim et al. (2014) propose a framework that can find the position of a hole using end-effector force measurements. However, strategies of this type do not fully address the problem of generalization for varying hole positions. This is because a successful insertion into the hole is still not guaranteed even with the full knowledge of the hole position.

The peg-in-hole task can fail due to a misalignment of the peg and hole during the insertion. In such cases, the peg experiences a high magnitude of reaction forces and ultimately gets stuck in the hole. This issue becomes increasingly obtrusive for tighter clearances on the hole. As a result, a second category of strategies is still needed to improve the insertion performance of the peg for a variety of hole clearances. In a recent example of such research, Gao et al. (2019), employ an admittance control scheme using the orientation and the angular velocity of the end-effector. The study tests the performance of a variety of controllers for different hole sizes. Even with the knowledge of the exact hole positions, reported success rates range between 76% to 88%, where lower success rates corresponding to tighter clearances.

1.4. Problem Statement

Traditional programming of robots fails to provide autonomous and dynamic adaptation to changing environments. Programming by Demonstration shows itself as a promising alternative to take up the task. However, the use of PbD in a realistic setting requires enhancements in its capacity to generalize tasks, the most important of which is the peg-in-hole task. Current frameworks of PbD accomplish varying success rates and report diminishing performance for increased novelty and tighter clearances (Gao et al. 2019, Duque, Prieto, and Hoyos 2019). Strategies to improve the success rate of the peg-in-hole tasks for changing hole positions and sizes constitute a pressing need for the field.

The most notable way of addressing these issues involves the extension of the PbD framework to the learning of the force characteristics of the task, in addition to its kinematics. Such approaches are already well-developed in the field and are generally applied for in-contact tasks such as polishing, painting, or collaborative transportation (Rozo, Jiménez, and Torras 2011, Rozo, Calinon, and Caldwell 2014, Rozo et al. 2015, Silvério et al. 2018). Tang et al. (2016) and Gao et al. (2019) both apply the idea to the peg-in-hole task by using admittance controllers. Development and implementation of such strategies to increase the generalization performance is a significant area of ongoing research.

1.5. Motivation & Scope

This thesis is motivated by the developments in the field toward the use of PbD in industrial settings. More specifically, it concerns itself with the performance of assembly operations using a PbD approach. In particular, it restricts its scope to the learning and execution of the individual peg-in-hole tasks and not an overarching assembly framework.

The initial aim of this thesis is to explicate the relationship between the success

rate of a peg-in-hole task and the novelty of the situation caused by a variation in the hole position. Doing so provides a benchmark against which the other strategies can be tested. Through this effort, it further aims to devise and test PbD strategies that improve the generalization performance of the peg-in-hole task for changing hole positions. Towards this goal, there are three research questions to be examined. These are the following:

1. How does the success rate of a peg-in-hole task change as the distance of the hole increases to its demonstrated position? The answer to this question not only explicates the relationship but also establishes a benchmark.

2. What is the generalization performance of a hybrid position/force learning strategy? This second question tests the performance of an alternative approach to further examine conditions for successful generalization.

3. How do the tested strategies compare to each other at different levels of novelty? This question attempts to answer not only if one approach is more performant than the other but also examines their relative advantages in dealing with novelty.

1.6. Outline

The work presented in this thesis is organized into five chapters. Chapter 1 presents an introduction to the field of PbD and guides the reader to the aim of the thesis. In chapter 2, the necessary theoretical background for the PbD process is discussed in relation to its four steps: demonstration, representation, implementation, and refinement. These steps serve as organizational units throughout the thesis to provide an overarching structure. Chapter 3 discusses the methodology by explaining the particular implementations of these steps in this study. The results of these efforts are presented in chapter 4. Finally, the last chapter interprets the findings of the study and outlines the future work that can be undertaken.

The following chapter discusses the theory of PbD with an integrated literature survey. Programming of a robot by demonstration can be understood as a four step process. The chapter explains the design choices, background information, and the necessary theory for each of these steps. Additionally, relevant examples from the different realizations of these steps are provided.

Chapter 3 discusses the methodology used in this thesis. The four-step process discussed in chapter 2 is used as an outline to not only organize the information of this chapter but also to relate theory to practice. The equipment, methods, and algorithms used for data acquisition, learning, control, and experimental evaluation are all presented in this chapter.

Chapter 4 presents the results of the study. Findings are presented in a structure

aligned with the previous outline. Furthermore, discussions are included to answer the research questions posed in the previous section.

The final chapter concludes the thesis by delivering the final remarks. It summarizes the work and reflects on the degree to which the aim of the thesis is accomplished. Suggestions for further work are also outlined in this chapter.

CHAPTER 2

THEORY

Programming by Demonstration, also called Learning by Demonstration, can be investigated under the broad field of machine learning (Hussein et al. 2017, Fang et al. 2019, Hussein et al. 2017). The term learning in this context has a particular meaning that can best be understood by contrasting it with the concept of mimicking. An agent that copies the actions of yet another agent is said to be mimicking that agent. For example, a robot can mimick a human demonstrator by attempting to perform a task the same way it was demonstrated. However, it does so without having an explicit model for the task it. As a result, the robot cannot appropriately perform it under novel circumstances beyond what has been demonstrated so far. It merely memorizes a particular execution of the task but does not learn it. On the other hand, learning happens if the robot gains the capability to accomplish the task under novel circumstances. This allows the execution of the task even in unstructured environments. This capability is called the generalization of a task.

The term supervised learning is used to describe a type of learning where an agent learns a task with the aid of an external teacher (Hussein et al. 2017). The particular aid can take many forms, such as providing labeled data, corrections, or feedback to the agent. Imitation learning then can be taken as a particular case where the aid is not simply providing labeled data but demonstrating a task to the agent. Broadly speaking, Programming by Demonstration can be thought as a subcategory of imitation learning simply due to the involvement of a human demonstrator. The subtle distinction between PbD and imitation learning in general is that imitation learning can be applied to all types of agents, such as software agents and biological agents, whereas PbD concerns itself explicitly with robots. Hence, the term "programming" is used to refer to the programming of robots.

2.1. Overview

The process of programming a robot by demonstration can be carried out in four distinct steps. Each one of these steps involves design choices about how the learning is accomplished (Hussein et al. 2017, Billard et al. 2008). The process starts with the enacting of demonstrations by a human teacher. In this step, the characteristics of the task are captured through data acquisition. The second step involves processing this data, also called the representation step. Here, using one or several learning algorithms, the data is reduced to form a model of the task. The model informs the robot about what to do, but it does not provide the capability to execute it. This is where the third step,

implementation, comes in. At this step, the robot is employed with one or several types of controllers to execute the task without human intervention. These three steps alone allow the robot to learn a task from scratch and start executing it. Additionally, some literature includes a fourth step, refinement of the model (Hussein et al. 2017, Billard et al. 2008). This last step allows the robot to hone its newly learned skill and makes it more capable of performing it.

The steps of the PbD process are used as organizational blocks throughout the thesis. This chapter includes a section for each step. In the demonstration section, the questions of how and what to demonstrate are covered. The representation section discusses the established learning methods and presents the theoretical formulation for the one used in this thesis. Controllers that are commonly employed in the literature are covered in the implementation section. Finally, the representation step discusses how the learning and execution can be enhanced.

2.2. Demonstration

The demonstration step is the starting point of the PbD process. It is the step at which the robot observes the demonstration of the task. In this context, the phrase "observation" refers to acquiring task-relevant variables from the demonstration using particular means. Consequently, the questions of "How to demonstrate?" and "What to demonstrate?" must be addressed at the demonstration step (Billard and Grollman 2013). Answers to these questions constitute the two main design choices at this step. The first question is answered by the modality of the demonstrations, whereas the latter is responded to by selecting variables relevant to the task. The following subsections are dedicated to an investigation of these aspects of the demonstration step.

2.2.1. Modality of the Demonstration

Modality of a demonstration refers to the particular way in which the demonstrator translates its knowledge of the task to the robot. For example, the demonstrator can visually show the task and its data can be captured via a vision system. Alternatively, the demonstrator can manipualate the robot itself to perform the task via kinesthetic teaching. These two options can be given as examples to the two broad categories of demonstration modalities: indirect and direct (Fang et al. 2019, Kumar et al. 2016). Table 2.1 shows the most common indirect and direct modalities. Indirect modalities involve the robot as an external observer, whereas direct modalities allow the robot to experience the task.

In the indirect demonstration modality, the robot is not physically involved with the execution of the task. Instead, it takes an observational role. Using one or several types of

Table 2.1. Commonly used direct and indirect demonstration modalities.

Indirect Demonstration	**Direct Demonstration**
Vision Systems	Kinesthetic Teaching
Sensors-on-Teacher	Teleoperation

sensors, it observes the task externally as it is performed by a human teacher. The use of vision systems and the sensors-on-teacher are considered the two most common methods under this category. The Fig. 2.1 and Fig. 2.2 show examples for both cases, respectively. One of the relative advantages of this modality is that it allows capturing of complex physical maneuvers of a human expert (Billard and Grollman 2013, Billard et al. 2008). For example, motion capture sensors on a human might be used to demonstrate tasks that would otherwise be hard to perform directly on the robot. This capability to capture human-like movements shows itself fully as an advantage in the field of humanoid robotics. However, on the other hand, this modality also creates challenges in transferring the task knowledge to the robot. This common problem is referred to as the correspondence problem in the literature (Nehaniv, Dautenhahn, et al. 2002). In general, the correspondence problem happens when the demonstrated states cannot be actualized in terms of the states of the robot. One such example is a case where the kinematic chains of the teacher and the robot differ considerably. It may be challenging, and sometimes even impossible, for the robot to recreate the same motion as it was demonstrated (Billard et al. 2008).

Figure 2.1. Demonstration of an assembly operation captured via vision based hand recognition. (Source: Duque, Prieto, and Hoyos 2019)

As an example to the use of indirect demonstrations modalities, Duque et al. (2019) employs a vision based hand recognition system. The robot observes the demonstrations for an assembly of parts in the manner depicted in Fig. 2.1 The other common indirect demonstration method is the use of sensor-on-teacher. This modality is particularly useful for humanoid robots. For instance, Koenemann et al. (2014) employ this modality with a humanoid robot in order to imitate a human teacher, as can be seen in Fig. 2.2.

In the modality of direct demonstration, the teacher performs the task through the robot. That is, the robot is directly manipulated by the teacher to perform the task. In this case, the robot does not observe the task but experiences it, so to speak. As it does so,

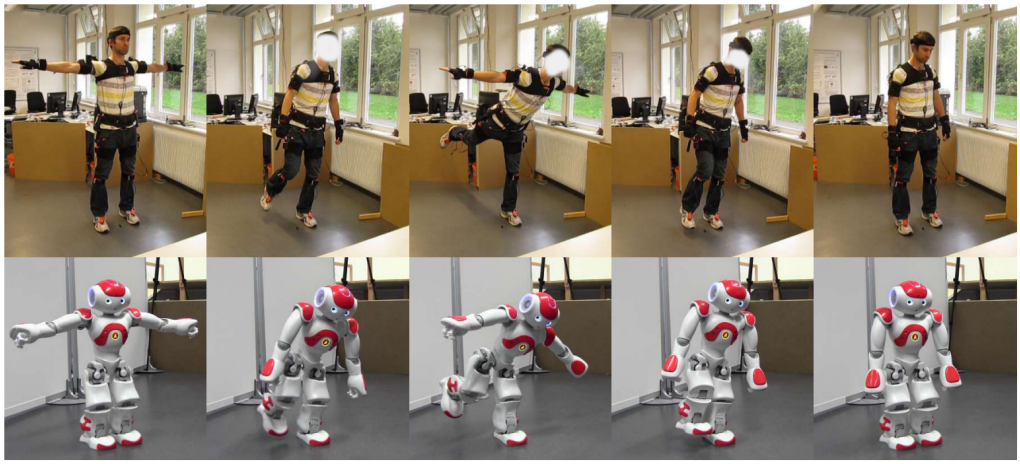

Figure 2.2. Demonstration of whole-body motion using sensors-on-teacher. (Source: Koenemann, Burget, and Bennewitz 2014)

task-relevant variables are recorded via the sensors on the robot itself instead of sensors on the teacher. There are two main modalities under this category (Fang et al. 2019, Argall et al. 2009). The first option is the teleoperation of the robot through a haptic device. Here, the task is performed remotely as an operator controls the robot. The second option is to move the robot manually using kinesthetic teaching. Many collaborative robots, such as the Universal Robots manipulators, come with free-drive capabilities that allow such feat. Using such features, a robot can freely be moved around by manipulating its links and joints. Fig. 2.3 and Fig. 2.4 show examples of these two modalities, respectively. For both examples, direct demonstrations display an essential advantage over the indirect approach in dealing with the problem of correspondence. Since the demonstration itself is recorded in terms of the robot's movement as it experiences it, it becomes not a problem to recreate it. However, the main drawback of this modality shows itself in capturing dynamic, complex, and human-like movements. For example, if the desired task is to teach dance moves to a humanoid robot, using a sensor-on-teacher modality is much more feasible than making the robot dance via teleoperation.

The choice of the demonstration modality consititutes one of the most important design choices on the implementation of PbD process. In addition to the general advantages and disadvantages of these modalities, the type of application and its characteristics can also become a deciding factor. For example, Kramberger et al. (2014) compare the performance of different demonstration modalities for the peg-in-hole task, and conclude the kinesthetic teaching to be the most appropriate. Zhu et al. (2020) provides the table 2.2 that can be used for further considerations.

Figure 2.3. Demonstrations using kinesthetic teaching modality.
(Source: Rozo et al. 2015)

2.2.2. Variables to Demonstrate

The second design decision about the demonstration step is the selection of variables to record. Whether it observes or experiences it, the robot cannot discern what variables are relevant to the task. One can imagine a practically unlimited number of variables that do not contribute to capturing the essence of the task. Consequently, the selection of what variables to record is provided apriori by the human programmer (Billard and Grollman 2013). Only the variables that are necessary to create a model of the task are selected as a part of the demonstration. In general, these variables represent the state (or configuration) of the robot and its immediate environment. However, the modeling of the robot that gives rise to these variables is not unique. For example, the programmer can prefer to use variables that do not strictly correspond to physical quantities. Two broad categories of approaches to selecting variables are called the low-level and high-level approaches in the literature (Calinon 2018, Billard and Grollman 2013, Ambhore 2020). These two approaches can be distinguished by how they define the robot's state and its environment.

In the low-level approach, the task is expressed in terms of the trajectories of some selected variables. In general, these variables are taken to be either the joint-space or task-space coordinates of the robot. Doing so allows the robot to learn the kinematic characteristics of the task. This is often referred to as kinematic learning or position learning of a task.

It is worth pointing out that there are many tasks whose appropriate learning cannot be done at a purely positional level. These are tasks that require the learning of force characteristics in addition to positional ones (Koropouli, Lee, and Hirche 2011, Rozo, Jiménez, and Torras 2011). Tasks such as surface polishing, painting, and many assembly operations can be given as examples of this category. In such cases, the selection of variables to record is extended to cover the contact characteristics of the task. The additional variables are often the wrench information coming from a force-torque sensor

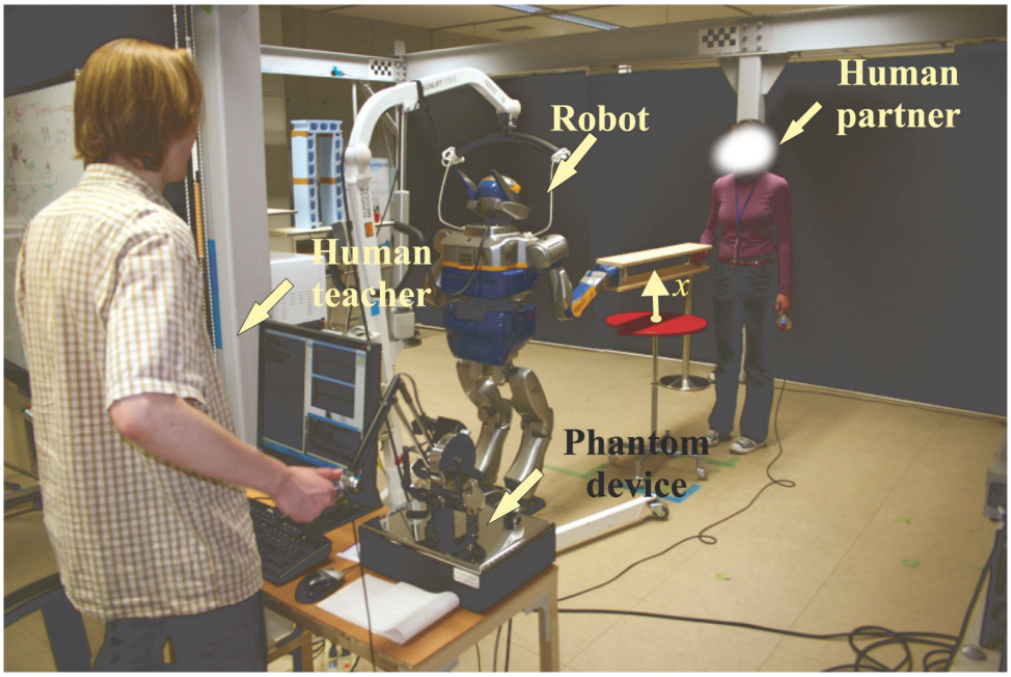

Figure 2.4. Demonstrations performed using haptic teleoperation.
(Source: Rozo et al. 2015)

(Silvério et al. 2018, Steinmetz, Montebelli, and Kyrki 2015). Learning a task in this manner can be referred to as the hybrid position/force learning from demonstrations (Rozo, Calinon, and Caldwell 2014, Steinmetz, Montebelli, and Kyrki 2015, Conkey and Hermans 2019).

The robot's state can also be represented at a level higher than the trajectories of some selected variables. For example, one approach might be to describe the motion of a robot not intrinsically in terms of the angles of its motors, but using symbolic states such as "moving forward", "standing", "walking", and "holding an object" (Billard et al. 2008, Calinon 2018). The evolution of the states then can be explained in terms of appropriate high-level actions. However, the robot is unaware of what it means to walk or stand by default. The programmer has to provide apriori definitions for the high-level states and actions in terms of the lower-level ones. Once done so, the demonstration can be recorded not simply as trajectories of low-level variables but as a set of high-level state-action pairs. In general, high-level representation allows one to work with potentially high degree-of-freedom robots (i,e. humanoid robots) and teach them complex tasks. However, it also requires implementing a large number of apriori states and actions as a drawback (Billard et al. 2008).

Hierarchical learning of tasks can also be considered as employing the high-level approach (Hussein et al. 2017). For example, assembly frameworks discussed in the introduction section decompose the operations into a sequence of individual tasks (Ding et al. 2020, Zhu 2020). In this context, these tasks can be considered high-level actions.

Table 2.2. A comparison of demonstration modalities.
(Source: Zhu 2020)

	Vision Systems	**Sensors-on-Teacher**	**Kinesthetic Teaching**	**Teleoperation**
Non-expert teaching	✓	✓	✓	✓
Supervised teaching	✓		✓	✓
Remote teaching	✓	✓		✓
Single demonstration	✓	?	?	?

These individual actions can either be demonstrated separately (Daniel, Neumann, and Peters 2012), or they can be extracted from a demonstration of the complete task via decomposition (Zhu 2020, Asfour et al. 2008, Tang, Lin, and Tomizuka 2015). Since the demonstration becomes encoded in terms of a combination or sequence of the high-level tasks, it becomes appropriate to understand it as a high-level demonstration approach.

2.3. Representation

The demonstration step only provides the data associated with the individual performances of the task. This data, as it is, can be used to replicate the demonstrations themselves. However, doing so would only be considered mimicking the task and not learning it. To successfully learn the task, the robot has to represent the data to form a model of the task. This is often done by employing data reduction and feature extraction methods onto the data (Billard et al. 2008, Billard and Grollman 2013, Calinon 2018). The use of such methods constitutes the second step of the PbD process.

Depending on how the demonstrations were encoded in the first place, the representation process can be done in one of two ways: high-level and low-level. If the demonstrations were to be encoded at a high level, then the goal is to create a policy that represents the task. The policy informs the robot of the appropriate actions it can take under novel states it encounters. Here, both the actions and states are high-level symbolic definitions, as they are already defined apriori in the demonstration step. Even though there exist many different algorithms for high-level learning, they generally involve graph-based techniques that encode and extend the demonstrations as directed graphs between states, changing via actions.

On the other hand, if the demonstrations were recorded at a low level as trajectories of some selected variables; then the representation step takes the form of a feature extraction problem. Learning algorithms used for such cases are either statistical or dynamical system based methods. The three most commonly used methods are: Dynamic Movement Primitives (DMP), Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM) (Billard et al. 2008). DMP falls under the category of the dynamical system based methods,

whereas the latter two are primarily statistical methods.

Even though it might be instructive to discuss the details of all the methods that are used for low-level and high-level learning, it goes far beyond the scope of this thesis when considering the sheer number of available methods in the literature. This study takes a low-level representation approach and employs DMP to learn from the demonstrations. There are two primary reasons behind this decision: (i) High-level representation is often preferred when working with high degree-of-freedom robots, such as humanoid robots. It seems to be the consensus in the field to prefer low-level representation for industrial manipulators, as it is the case with this thesis. (ii) There are three commonly used learning methods for the low-level approach. Among them, DMP shows itself as the one with the most premise in generalization and robustness against perturbations (Ambhore 2020, Billard et al. 2008, Ijspeert et al. 2013). Since this thesis aims to improve peg-in-hole learning in a way that allows robots to perform it successfully under novel circumstances, DMP is the most natural choice for the learning method.

2.3.1. Dynamic Movement Primitives

The concept of a movement primitive is a bio-inspired idea that is used to explain the construction of complex movements from simpler ones. It is known that the stimulation of certain synaptic pathways in animals causes particular motion patterns in the muscles. The number of such pathways is fundamentally limited due to the physical constraints of their body. Nevertheless, they can seemingly perform an unlimited number of different movements. Giszter et al. suggested that these complex movements can be explained by the use of a limited number of movement primitives in conjunction (S. F. Giszter, Ferdinando A Mussa-Ivaldi, and Emilio Bizzi 1993, S. Giszter, F. Mussa-Ivaldi, and Bizzi 1993).

The concept of a movement primitive can also be applied to robots. Mathematically speaking, this is done by using the primitives to serve as a basis for some function space of possible trajectories. For example, a primitive can be defined as a map in the form $P_i : \mathbb{R} \longrightarrow \mathbb{R}^n$, where P_i denotes the i-th primitive, and $\mathbb{R}^n$ charts the configuration space of the robot (Williamson 1996, Edsinger 2001). A primitive of this form is referred to as a motor primitive as it generally corresponds to a joint-space trajectory (Kober and Peters 2009). Here, the term motor primitive serves as a bridging analogy that connects the motor skills of biological organisms with the actuators of robots. Once the motor primitives for basic movement patterns are learned, a linear combination of them can be used to create complex composite movements.

Dynamic Movement Primitives (DMP) is a method that combines the idea of movement primitives with dynamical systems (Ijspeert, Nakanishi, and Schaal 2002, Ijspeert et al. 2013). It is called a movement primitive method because it involves the

use of a linear combination of trajectories to create complex functions. However, the linear combination of these primitives does not simply correspond to the final joint-space trajectory (Schaal et al. 1997). Instead, it is used as a driving function in a set of non-linear differential equations. These differential equations define an attractor by expressing the dynamics of a mass-spring-damper system. Hence, the use of the term "dynamic" in the method's name.

DMP can be understood as a trajectory generation method that generates a simple trajectory in the form $y : \mathbb{R} \longrightarrow \mathbb{R}^n$ using a dynamical system given by a system of ODEs. It does so in a way that provides the following properties:

- The trajectory is guaranteed to converge to the desired goal without any steady-state error.
- The velocity with which the trajectory is to be covered can be changed parametrically.
- The trajectory generation can approximate any arbitrary curve that ends up at the desired goal.
- Given demonstrated trajectories, the method can learn from them in a way that mimics not simply their waypoints but also their geometrical properties.

The last requirement separates the DMP method from other similar trajectory generation techniques. If not for this feature, one could always find waypoints through some space using their choice of a search algorithm and create a simple spline trajectory that passes through these points with the desired velocity profile. The DMP flourishes in the cases where the task involves humanly maneuvers that rellies on geometric invariance. For example, Fig. 2.5 shows how DMP can generalize the motion required to write a letter for different sizes by preserving its geometric properties.

In dynamical systems theory, the term attractor refers to a sub-manifold of the state space (either a point, a curve, etc.) that the system naturally evolves to and stability stays at. The simplest example of an attractor is the equilibrium of a mass-spring-damper system. This is because an unforced mass-spring-damper system converges to its equilibrium position no matter where it starts. The governing equation of a mass-spring-damper system with the equilibrium position at $y = g$ can be organized in the following way:

$$\ddot{y} = \alpha_y \left(\beta_y \left(y - g \right) - \dot{y} \right) \tag{2.1}$$

Here, α_y and β_y are left to our choice. Preferably, a critically damped system is selected, which can be achieved by $\alpha_y = 4\beta_y$. The DMP method's perspective is to conceptualize this second-order differential equation as a trajectory generator. This equation can already be used to generate trajectories that move towards the goal from

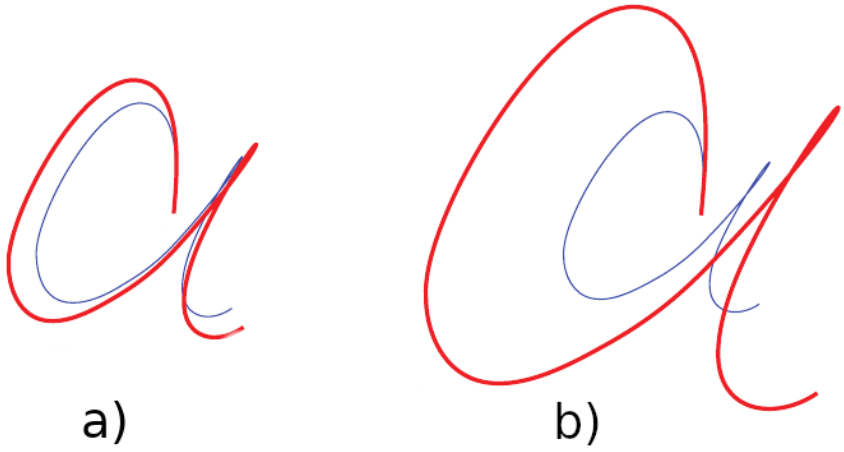

Figure 2.5. Geometric invariance properties of DMP method.
a) The demonstrated curve for the letter "a".
b) The curve generated for a new end point, the shape of the letter is preserved. (Source: Ijspeert et al. 2013)

an arbitrary initial condition. However, using it in this way would also fully specify the velocity with which to move along the curve. To provide a parametric curve velocity, a new variable z is defined as a time constant multiply of the original curve velocity $\dot{y}$. By changing the time constant, the same path generated by the second-order system can be traced at different velocities.

$$\tau \dot{y} = z \tag{2.2}$$

This new variable z can be used to rewrite (2.1). We first make the substitutions: $\dot{y} = \frac{z}{\tau}$, $\ddot{y} = \frac{\dot{z}}{\tau}$. Then, multiplying the whole equation by τ^2 gives:

$$\tau \dot{z} = \alpha_z \left(\beta_z \left(y - g\right) - z\right) \tag{2.3}$$

The new consants relate with the old ones in the following way: $\alpha_z = \tau \alpha_y$ and $\beta_z = \tau \beta_y$. The condition to make the system critically damped is still given by $\alpha_z = 4\beta_z$. At this point, the previous second-order differential equation is converted into two first-order differential equations. The difference is that we can now set some time constant τ to alter the velocity of the response. The formulation so far satisfies the first two aims of the DMP method: convergence to the goal and a parametric velocity. The next aim of the method is to be able to generate any arbitrary curve. However, given the constants α_z, β_z, and τ, the current formulation can only generate an unforced critically damped mass-spring-damper system response between the initial condition and the goal. We gain the ability to deviate from this unforced response by adding a forcing term to (2.3).

$$\tau \dot{z} = \alpha_z \left(\beta_z \left(y - g\right) - z\right) + f \tag{2.4}$$

With the addition of the forcing term, the convergence to the goal is not guaranteed unless we specify the forcing term further. Hence, we select the forcing term f as a function that approaches zero as the trajectory moves towards the goal. As a result, the system behaves as if it is an unforced system as it gets closer to the goal. This idea can be formulated by expressing f not as a function of time but of some phase variable x. This new phase variable is a quantity that acts as a measure of how much of the trajectory has been completed so far. Consequently, we desire it to start at one and be guaranteed to approach zero in time. Any such function can be selected as the phase variable; the most convenient of which is an exponential decay function. We define x as an exponential decay function of time, given by the expression $x = e^{\frac{-\alpha_x t}{\tau}}$. We omit the time variable simply by expressing x in a differential equation as follows:

$$\tau \dot{x} = -\alpha_x x \tag{2.5}$$

The equations presented so far defines a system of three first-order differential equations. The last missing piece of information is the formula for the forcing term f. This term is where the concept of the movement primitive appears. As discussed earlier, the idea is to define the primitives as a basis so that any desired function can be approximated via their linear combination. In function approximation, the most commonly used bases are sinusoidals and exponentials. They are most valuable when we want to specify the trajectory based on its frequency content or increasing/decreasing characteristic. Neither of them is particularly desired in the case of trajectory generation for a robot. To fully express the intricate geometric features of the trajectory, the use of impulse functions would suffice. However, one would need an infinite number of impulse functions to approximate the trajectory. This discussion shows that we are looking for functions that are centered at some point, in some sense, just like the impulse functions, but have some width so that we would not need an infinite number of them. The DMP method proposes the use of Gaussian kernels of the form (Ijspeert, Nakanishi, and Schaal 2002, Ijspeert et al. 2013):

$$\psi_i(x) = exp(-h_i(x - c_i)^2) \tag{2.6}$$

where c_i are the centers of the Gaussian kernels and h_i are coefficients describing their width. The forcing term is then defined as the normalized linear combination of these basis functions, with some additional multipliers.

$$f(x) = \frac{\sum \psi_i w_i}{\sum \psi_i} x(g - y) \tag{2.7}$$

Here, the multiplier x guarantees that the forcing term approaches 0 as the trajectory

approaches to goal, that is, as x itself approaches 0. The term $(g - y)$ is used for scaling the trajectory for varying goals and initial conditions, and it provides the geometric invariance properties (Ijspeert, Nakanishi, and Schaal 2002, Ijspeert 2003, Ijspeert et al. 2013).

The final aim of the DMP method is to learn what kind of trajectory to generate from a given set of demonstrations. This is done by simply learning the weights w_i from the demonstrations. To solve for w_i, we first express the equation (2.4) in terms of $y, \dot{y}, \ddot{y}$ by substituting $z = \tau\dot{y}$ and $\dot{z} = \tau\ddot{y}$.

$$\tau^2\ddot{y} - \alpha_z\left(\beta_z(g - y) - \tau\dot{y}\right) = f \tag{2.8}$$

A demonstration is given by the time series y_{demo} and its numerical derivatives $\dot{y}_{demo}, \ddot{y}_{demo}$. The required driving force that would follow the demonstration can be solved from the previous equation as follows:

$$f_{target} = \tau^2\ddot{y}_{demo} - \alpha_z\left(\beta_z(g - y_{demo}) - \tau\dot{y}_{demo}\right) \tag{2.9}$$

We select the weights w_i such that the driving force $f(x)$ calculated using (6) approximates the f_{target}. This is a function approximation problem whose solution can be found as follows using locally-weighted regression (Ijspeert et al. 2013).

$$w_i = \frac{\mathbf{s^T \Gamma_i f_{target}}}{\mathbf{s^T \Gamma_i s}} \tag{2.10}$$

$$\mathbf{\Gamma_i} = \begin{bmatrix} \psi_i(x_1) & & & 0 \\ & \psi_i(x_2) & & \\ & & \dots & \\ 0 & & & \psi_i(x_N) \end{bmatrix} \quad \mathbf{s} = \begin{bmatrix} x_i \\ x_2 \\ \dots \\ x_N \end{bmatrix} (g - y_0) \quad \mathbf{f_{target}} = \begin{bmatrix} f_1 \\ f_2 \\ \dots \\ f_N \end{bmatrix} \tag{2.11}$$

$x_1, x_2, ..., x_N$ are the values of the phase variable x at the instances corresponding to samples of the demonstration data. Similarly, f_i is given by the f_{target} computed at the i-th index of the demonstrations $\ddot{y}_{demo}$, $\dot{y}_{demo}$ and y_{demo}.

Once the weights are learned from the demonstrations, the system of three differential equations given by the (2.2), (2.3), and (2.5) can be solved. The trajectory's desired goal and initial condition are provided to the solver by the selection g and y_0, respectively. The system generates a trajectory that not only converges to the goal but does so in a way that mimics the geometry of the demonstrations. To the degree that a task depends on the geometric features of the movement required to execute it, the DMP method allows the generation of trajectories that perform the task even for novel initial conditions and goals. This is how DMP allows the generalization of a task.

Two implementation details have to be discussed for the appropriate use of DMP on a multi-degree-of-freedom robotic system. In its current form provided by the formulation above, it can only be applied to learn from a single demonstration and can generate a trajectory for a single variable. More often than not, this variable is a low-level variable corresponding to either a joint-space or task-space coordinate of the robot. The demonstration, therefore, is a time series holding the values of this variable encountered while the task was performed. The DMP method can be expanded to learn from multiple demonstrations and be used for multi-degree-of-freedom robots.

The DMP formulation presented above can be independently applied to individual degrees of freedom. A system of differential equations for each degree of freedom, without the need of any coupling, can be formulated to generate trajectories for the selected variables (Ijspeert et al. 2013). While doing so, the only constraint is to solve the differential equations at the same time steps and for the same period. In that way, the generated trajectories for individual degrees of freedom are synched in time. Since the system of differential equations generate a smooth curve, the resulting trajectory is also a smooth trajectory in the joint space. It can be easily followed as long as the desired poses are reachable within the workspace with the available joint position and velocity limits.

Furthermore, the DMP method can also be used to learn from multiple demonstrations. It is often the case that each demonstration involves particularities encountered only during that demonstration and are not intrinsic to the task. These might be random variations in the teacher's performance or in the environment. The use of multiple demonstrations solves this problem by, in some sense averaging across the multiple demonstrations (Hersch et al. 2006). There are a couple of standard approaches to incorporating multiple demonstrations into the DMP formulation. The first approach is a statistical one that combines DMP with the GMM. GMM can be used to combine the multiple demonstrations into one equivalent demonstration. In some sense, it calculates the invariant aspects that are mostly shared across all demonstrations (Ding et al. 2020). Then, this single demonstration can be learned simply by following the formulation above. Perhaps a simpler approach is to directly average the demonstrations without the need of using an additional method. The value f_{target} computed for the learning of the weights can be taken as the average across all the demonstrations. As a result, the particularities of the demonstrations disappear to a large extent, and the learned weights are informed by all the demonstrations. However, to find the average f_{target}, one has to sum the trajectories $\ddot{y}_{demo}$, $\dot{y}_{demo}$ and y_{demo} for different demonstrations. Since the demonstrations are often recorded as time series, at some sample rate, different demonstrations do not necessarily have the same time span nor the same number of indices. Hence, they cannot be added directly. As a workaround for that problem, demonstrations are often stretched to be aligned with each other. One of the most commonly used algorithms in this regard is the Dynamic Time Warping (DTW) algorithm (Ding et al. 2020). The DTW algorithm has a couple of useful

mathematical properties that prevent the generation of unnecessary artifacts or frequency content as the demonstrations are stretched. These properties become especially useful if the demonstrations have largely varying time spans and must be significantly processed. For more straightforward cases, however, basic stretching methods also suffice. As an example, simply aligning the ends of each demonstration and inserting extra indices as needed is a simple alternative. The values at the intermediate indices can be found using any smooth interpolation method (cubic, spline, etc.). This also prevents the generation of non-smooth artifacts as the demonstrations are stretched.

2.4. Implementation

The model generated in the learning step gives the robot the means with which it can perform the task all by itself. This capability, however, takes a different form depending on the approach that is taken at the learning step. For example, the model generated using a high-level approach is a policy that provides the appropriate action for each state the robot can encounter. To perform the task, the necessary actuator commands associated with these high-level actions, or means of generating them dynamically, have to be provided to the robot (Billard et al. 2008). In low-level learning, this is directly the case. The robot directly follows the generated joint space or task space trajectories by employing potentially several types of controllers (Billard et al. 2008).

As discussed in the previous section, the low-level learning approach represents the task using the trajectories of the task-relevant variables. As a result, it generates desired trajectories for them to be followed. For example, if the DMP method were to be employed in the task space, it would generate a task space trajectory to be followed. If it is employed in the joint space, which is often the case, it generates a joint space trajectory to be followed. Both of these options require the implementation of a position controller on the robot so that these trajectories can be followed.

The position level variables may not be the only variables recorded during a demonstration. That is, the answer to what constitutes a demonstration for the task can be extended to include additional variables than the positional ones. As discussed previously, in-contact tasks can appropriately be learned only by incorporating not only the kinematics of the task but also its force characteristics. Painting, polishing, drawing, and peg-in-hole tasks can be given as examples of such in-contact tasks. By extension of the previous remark that the variables that are used in the learning step require controllers to be implemented, the implementation of the in-contact tasks requires a force-based controller. In the literature, admittance/impedance control and hybrid position/force control schemes are commonly used (Gao et al. 2019, Rozo, Jiménez, and Torras 2011, Rozo, Calinon, and Caldwell 2014, Rozo et al. 2015, Silvério et al. 2018, Steinmetz, Montebelli, and Kyrki 2015).

2.5. Refinement

Refinement of a learned model is an additional step counted among the phases of PbD by some literature (Hussein et al. 2017, Billard et al. 2008). The concept of refinement refers to a process with which the model's performance can be improved for some desired metric. There are two main advantages to employing such a process. The first is an increase in the task performance, quantified in terms of the desired metric. The robot becomes increasingly more capable of performing the task as the model is refined. The second advantage is a non-obvious one concerning a reduction in the number of initial demonstrations. Continuous refinement of a model can be used to start executing the task with a minimal number of demonstrations.

Since all demonstrations are performed by human teachers, they are often sub-optimal in one or many of the desired metrics such as the execution time or overall energy use. Continually executing a task with these inefficiencies causes a considerable cost in an industrial setting. The refinement process provides a solution to this problem. The PbD process can be combined with optimization techniques to generate or alter the models to optimize them for some desired metric. Alternatively, it can be combined with other learning methods such as Reinforcement Learning (RL) to continually search for alterations of the model that improve some desired goal.

The refinement process can also cut down the number of initial demonstrations. It is generally the case that demonstrations include particularities that are not essential to the task. For example, the human expert might do an unnecessary maneuver in one of the demonstrations. This poses a challenge to the learning process as the robot cannot distinguish what is the task and what is a particularity. One of the ways to overcome this problem is to provide a large number of demonstrations. However, doing so contradicts the aim of PbD as it attempts to be faster and more feasible than cumbersome traditional programming. An alternative approach is to start with a few demonstrations, then continuously refine the model as the task is performed. This can be accomplished in one of several ways, depicted in Fig. 2.6. The most straightforward method is to ask for additional human feedback. The robot can stop and ask for further supervision from the human expert if it encounters a novel case it cannot solve. Alternatively, the robot can try to explore possible solutions and find a suitable one using other learning methods such as Reinforcement Learning (Billard et al. 2008).

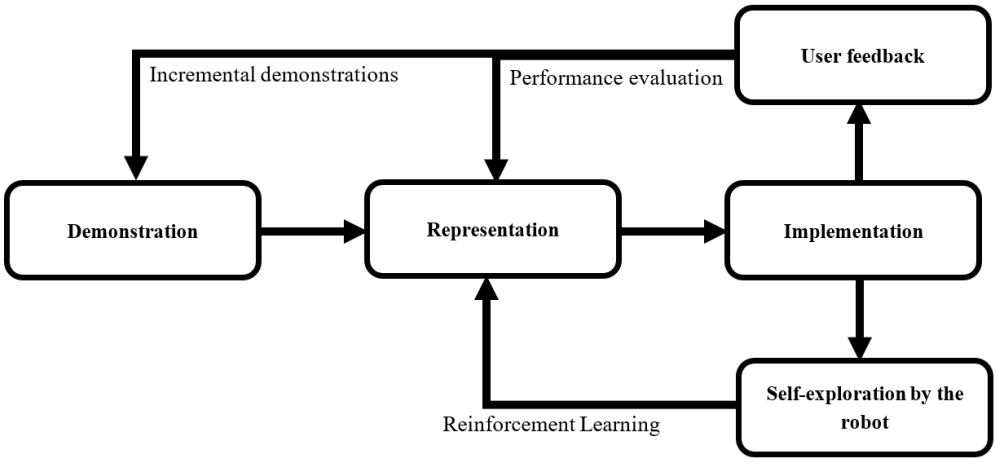

Figure 2.6. A diagram of the refinement process.

CHAPTER 3

METHODOLOGY

3.1. Overview

This section outlines the specific ways in which the methodology of the PbD is used in this study. It does so by organizing the information it provides along the lines of the four steps of the PbD process. To start with, the equipment used for the experimental evaluation is presented in this overview section. Also, an overall discussion on the approach taken to answer the research questions is presented. The following section, demonstration, discusses the modality and the means of data acquisition employed in this study. The third section discusses the particular use of DMP to learn the desired position and force trajectories from demonstrations. Finally, the last section discusses the implementations of these trajectories and the criteria for the successful execution of the task. In accordance with the aim of the thesis to enhance the generalization performance fundamentally through a change of the learning approach rather than continuous improvement, no refinement step is included in this study.

3.1.1. Equipment

Learning a task from demonstration is a process that fundamentally relies on real-life data. Consequently, it cannot be accurately implemented and tested in a simulation environment. This becomes especially true in the hybrid position/force learning case where the contact forces between the objects are a primary data source. Accurate modeling of the contact for a peg-in-hole task is an enormous task in and of itself. Instead, this study takes an experimental approach and uses PbD on a UR5 manipulator.

Universal Robots is one of the most well-known robot manufacturer companies. They specialize in collaborative robots (cobots) designed and built to work alongside human operators. This requires the robots to be equipped with safety features such as protective stops upon encountering unexpected torques. Also, they come with a free-drive capability that allows the robot arms to be manipulated by hand. There are four main models manufactured by the UR company: UR3, UR5, UR10, and UR16. The numbers indicate the maximum payload these manipulators can carry. In addition to the models, there is also a serie distinction between them. There are two series of manipulators. The older series manipulators (CB3 series) are named UR3-16, whereas the new e-Series manipulators get an "e" after their name, i.e., UR3e. This study uses a CB3 series UR5 manipulator whose picture is provided in Fig. 3.1 below.

Figure 3.1. The UR5 manipulator used in this study.

Robotiq is a company that specializes in the production of robotic tools. They design and manufacture grippers, force-torque sensors, and alike, compatible with many brands of industrial manipulators. One of the primary collaborators of the Robotiq is the Universal Robots. Many of the Robotiq products come pre-configured to be mounted on UR manipulators. Also, there are software extensions that allow the control of this equipment from the UR programming interface. In this study, a 2F-85 Robotiq gripper is used as the end-effector. 2F stands for "2 finger" whereas 85 indicates the width of the gripper. Additionally, a FT300 force-torque sensor is mounted between the flange and the gripper. The use of such a force-torque sensor is of great importance for the implementation of hybrid learning schemes. A picture of the gripper and the force-torque sensor mounted onto the UR5 manipulator is provided in Fig. 3.2 below.

3.1.2. Approach

This study has two important aims. The first aim is to provide insight into the relationship between the success rate of a peg-in-hole task and the novelty of the position at which it is attempted to be executed. Once this relationship is explicated, it can be used as a benchmark to test alternative strategies to improve the generalization performance. Consequently, the second aim of this study is to devise, implement and test a PbD strategy that improves the generalization performance.

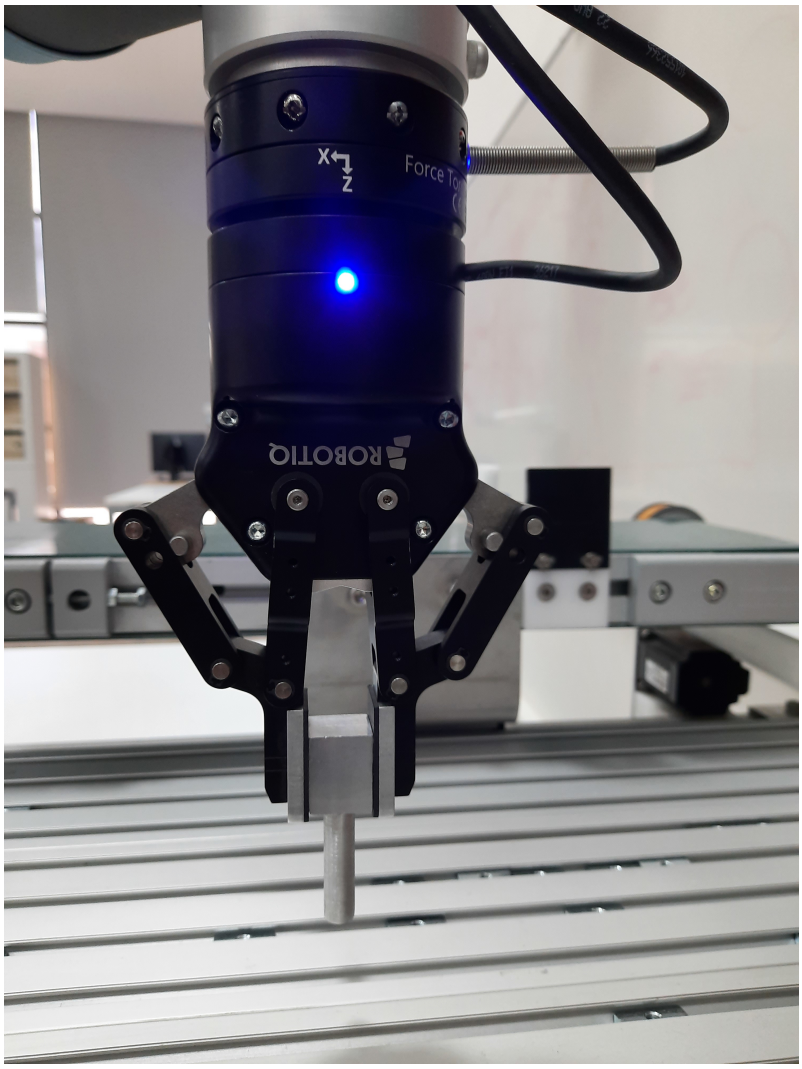

Figure 3.2. Robotiq 2F-85 gripper and FT300 force torque sensor, mounted on a UR5 manipulator.

In accordance with this thesis's aims, three research questions must be tackled. These questions are already provided in the first chapter of this thesis. In this chapter, the approach taken to answer these questions will be discussed.

The first research question examines the generalization performance of a peg-in-hole task learned from demonstrations. In particular, it asks about the change in success rate when the task is executed at different locations. The first part of the thesis uses the following approach to address this question.

- A peg-in-hole task is demonstrated using a UR5 manipulator. Modality of kinesthetic teaching is employed. The hole is kept at a fixed location during these demonstrations. This location is the original location at which the task is initially learned.
- DMP method is applied to perform positional learning from these demonstrations. This is done so in a two-step process. First, multiple demonstrations are learned in

the task space, generating an equivalent trajectory. Then, the joint-space coordinates of this equivalent demonstration are learned using DMP again.

- Trajectories for forty novel hole positions with increasing distances to the original hole position are generated.
- These trajectories are then sent to the UR5 via the ROS interface to be executed. The successful and unsuccessful executions of the task are experimentally evaluated at each hole position.
- This provides a performance metric for the generalization performance of the peg-in-hole task executed at increasingly distant locations.

The second question this thesis examines is the generalization performance for a hybrid position/force learning strategy. This question is also addressed experimentally.

- UR5 manipulator is equipped with a Robotiq FT300 force-torque sensor. The end-effector force-torque values experienced during the demonstrations are also incorporated into the demonstration data as if they are additional variables.
- DMP method is applied in the same manner as the first part to learn the positional characteristics of the task. Additionally, it is also applied to the recorded force data to learn the force characteristics.
- In addition to the trajectories for 40 novel hole positions, desired force profiles to be followed during these trajectories are generated.
- An admittance control scheme is implemented on the UR5 manipulator using the values from the FT300 sensor as force feedback.
- The generated position and force trajectories are executed, and the performance at each hole location is noted.
- This evaluation provides a success rate of the hybrid position/force learning strategy at each hole location.

Finally, the last question of the thesis asks is how do these strategies compare to each other. After the success rates for both strategies are evaluated experimentally, they are compared in the results section. Further discussions on this comparison are also provided in the conclusions chapter.

3.1.3. Outline

This thesis attempts to enhance the position-level generalization of a peg-in-hole task by testing different learning strategies. To do so, it compares two learning approaches for the peg-in-hole task: purely positional learning and hybrid position/force learning. Positional learning involves learning only the configuration of the robot observed during the demonstrations of the peg-in-hole task, whereas hybrid learning makes use of the recorded end-effector forces too. Correspondingly, their PbD steps require different alterations. The implementations of the PbD steps for both strategies are discussed throughout this chapter in detail. Additionally, a brief summary of these distinctions at each step is provided in the following paragraph.

In the demonstration step, the positional learning approach makes use of only the joint positions. The end-effector forces are also acquired using the force-torque sensor at the last joint for the hybrid learning case. Similarly, positional learning only learns the recorded joint positions, and it uses DMP to generate target joint space trajectories. In the hybrid learning case, the end-effector forces are also learned using DMP. A target force profile to be followed is generated, in addition to the joint space trajectories. These two learning approaches also require different controllers for their implementation steps. In the first case, the joint trajectories can directly be sent to the robot via the actionlib interface exposed by the ROS driver. However, the latter case requires an admittance control based force compensator to be implemented on top of the built-in position controller. The following sections discuss the steps of the PbD process and the experimental evaluation of the performances of both strategies.

3.2. Demonstrations

This section describes the methodology used in this thesis concerning the demonstration step of the PbD process. As discussed in the previous section, the demonstration step is characterized by its answers to the questions of how and what to measure as the demonstration of a task. The task, in this case, is a peg-in-hole task that includes a hole of 12 mm diameter and a peg of 10.2 mm diameter. A picture of the peg and the hole can be seen in Fig. 3.3. The task was performed by mounting the hole at a specific position and manually manipulating the UR5 to insert the peg into the hole. As the demonstration data, the joint variables ($\theta_{1\text{-}6}$) and the end-effector forces as they are measured by a force-torque sensor are acquired.

The first question the demonstration step answers is "how" the demonstrations are observed or experienced by the robot. The answer to this question is provided by the modality of the teaching. In this particular thesis, kinesthetic teaching is preferred

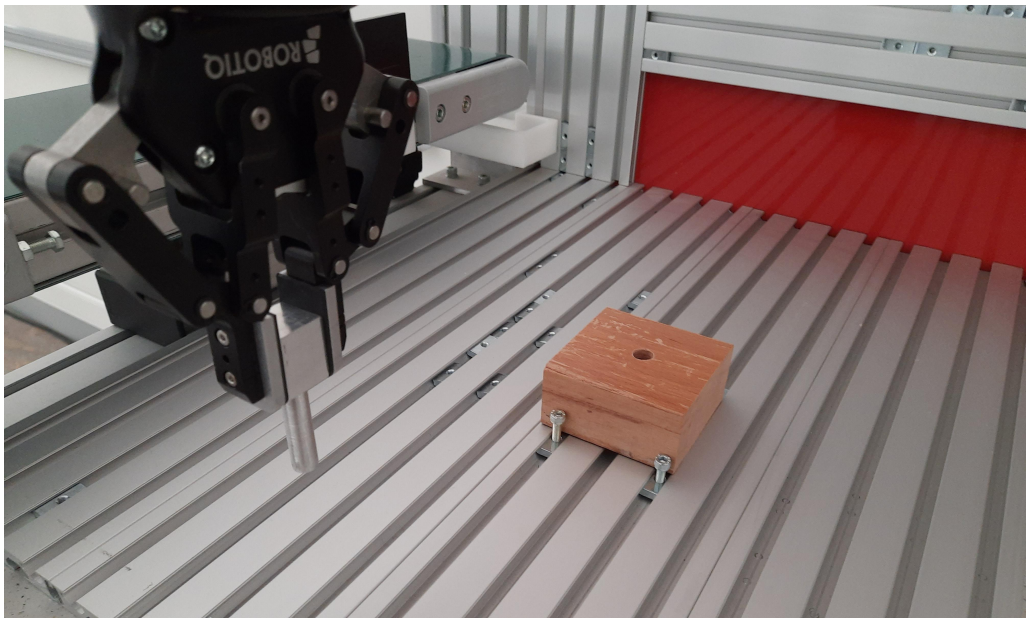

Figure 3.3. A picture of the peg and the hole.

due to its advantages in teaching force characteristics for peg-in-hole tasks (Kramberger 2014). To be used in conjunction with kinesthetic teaching, the UR5 manipulator features a free-drive mode in which its joints can be freely moved by hand. The demonstrations are performed by a human teacher using this functionality. While doing so, the robot is manipulated carefully without contacting its end-effector to prevent any artificial force readings.

The task that is being demonstrated is a peg-in-hole type of task. During these demonstrations, a hole of 12 mm diameter is maintained at a fixed position in the robot's workspace. This position will be referred to as the original hole position throughout this document. A peg of 10.2 mm diameter that is firmly gripped by the robot is inserted into the hole by manually manipulating the robot. A sufficient number of such demonstrations are recorded to negate the idiosyncratic inefficiencies of any particular demonstration. The initial conditions for each demonstration are altered without any apparent pattern while keeping the goal at the same fixed position. Fig. 3.4 shows how the demonstrations are performed.

The second design choice concerning the demonstration step is the selection of variables to record. In this thesis, two alternative sets of variables are recorded, corresponding to two strategies that are tested out for their performance on the peg-in-hole task. Firstly, the joint variables of the UR5 manipulator, as the robot experiences them during the demonstrations, are recorded. Learning of the task using only those variables constitutes purely positional learning. Additionally, the end-effector forces, as they are measured by the force-torque sensor at the last joint, are recorded. The use of this additional set of variables in conjunction with the joint variables allows hybrid position and force learning.

Figure 3.4. Kinesthetic teaching of the peg-in-hole task.

3.2.1. UR ROS Driver

The joint variables and the end-effector forces are recorded into a workstation connected to the same network as the robot. By default, the UR5 manipulator uses a data format called RTDE (Real-Time Data Exchange Format), using which these values can manually be accessed. More conveniently, ROS industrial project has developed a UR ROS driver that reads the RTDE messages through the network and publishes their values as ROS messages. These values then can be read and recorded on a computer in the same local area network as the robot. The driver can provide sampling frequencies up to 125 Hz, which is the rate at which the underlying RTDE messages are sent.

The connection with the robot for data acquisition can be made by connecting it to a local area network with a computer. This is done by connecting the control box of the UR5 and the PC to a switch using ethernet cables. In the next step, appropriate static IPs should be assigned for both the PC and the robot. On the computer, this is simply done by assigning a manual IP v4 address in the network settings. Similarly, the robot features a network setting tab in its graphical programming interface. This tab can be accessed through the teach pendant. Once the connection is established, the robot and the computer should be able to ping each other. Then, the UR ROS driver can be launched from the

computer using the `ur5_bringup.launch` file that comes with the driver package. This starts a series of packages whose details are discussed in the implementation section. For the purpose of data acquisition, it suffices to know that the driver publishes the joint states to a ROS topic at a rate of 125 Hz. In this study, a subscriber node is used to monitor these values as they are published and record them to a CSV file. A rate of 100 Hz is preferred for data acquisition and trajectory generation, as it provides time steps that are easier to reason about.

3.2.2. Robotiq FT Sensor

The force-torque readings experienced by the FT300 sensor are not published through the UR ROS driver by default. There are additional steps that have to be taken to be able to read those values on the computer. The sensor uses the RS-485 standard for serial communication with the robot. The manufacturer distributes the sensor with an RS-485 to USB converter. In regular use, the USB output of the sensor is connected to the USB port on the robot's control box. The programming interface of UR manipulators, PolyScope, allows the installment of software extensions, which are called URCaps. Once the URCap for the Robotiq FT300 sensor is installed, and the connection is established, the force-torque sensor can be accessed through the programming interface of the robot.

To establish a connection with an external computer, the USB connection should be made between the sensor and the computer, not the control box of the robot. This causes the sensor to be not accessible through the default means using the teach pendant. However, it allows its values to be read into the computer. Before it starts continually streaming data, the sensor waits for the appropriate MODBUS command. For the successful connection, IP settings for the target computer should also be set in the MODBUS settings under the installation tab. Similar to the UR ROS driver, ROS industrial project features a series of Robotiq packages that allows the control of Robotiq hardware through ROS. The most recent version of the package is primarily developed for the ROS kinetic version. However, it also works for melodic in this case. Once the `robotiq_ft_sensor` package is built, the `rq_sensor` file can be run to establish the connection with the sensor. This executable reads the force-torque sensor values through the network and publishes them to a topic. In this study, the same node that subscribes to the joint states topic is also used to subscribe to this topic. The force-torque values are recorded alongside the joint positions at a rate of 100 Hz.

3.3. Representation

Once the demonstration step is completed, the resulting data is to be represented using a learning method. In this thesis, the DMP is applied in two different ways to the recorded data. In both cases, the algorithm is implemented in MATLAB, and the generated trajectories are saved into CSV files to be later sent to the robot. The formulae of the DMP method are already discussed in the theory section. This section discusses the particular ways in which that formulation is applied to the demonstration data.

The learning using the DMP method is carried out in two steps in this study. DMP method is known for its capability to encode the spatial characteristics of the motion associated with the execution of a task. However, this also provides a challenge as it learns the particular maneuvers included in the demonstrations that are not essential to the task. As a workaround for this problem, some studies first employ GMM to generate an equivalent trajectory (Ding et al. 2020). This thesis takes a similar approach but uses DMP from start to finish. The process can be summarized in the following three stages:

- First, the demonstration data is expressed in the task space and learned using DMP. Here, a task space representation in terms of the end effector XYZ coordinates and XYZ Euler angles is used. A trajectory between the average initial and goal values of each of these variables is generated as an equivalent trajectory for the demonstrations.
- In the second stage of the learning, the equivalent trajectory is brought back to the joint space using the inverse kinematics of the robot and learned as a single demonstration using the DMP method again.
- In the last step, the learned model is used to generate trajectories will be changed later for forty novel hole positions.

3.3.1. Initial Learning Phase

The initial learning phase first converts the joint space trajectories to task space ones. Here, a task space representation using the XYZ coordinates of the end effector position and XYZ Euler angles of its orientation is adopted. The force values encountered at each instant are also expressed in the ground frame for learning Cartesian force characteristics. Once the DMP learns the weights, it can be used to generate trajectories between any desired initial and goal point. In this step, the initial and goal points are taken to be the average initial and goal points observed in the demonstrations. This generates a single equivalent trajectory that is informed by all of the demonstrations.

An examplary code snippet for the algorithm used is provided below. Let d^{th} demonstration is represented by a matrix:

$$D_d = \left[T \vdots \theta_1 \vdots \theta_2 \vdots \theta_3 \vdots \theta_4 \vdots \theta_5 \vdots \theta_6 \vdots F_x \vdots F_y \vdots F_z \right]$$

Each entry in this matrix is a column vector including the variable's value recorded at that time instant T. An arbitrary number of demonstrations can be loaded into a cell array and be accessed using $D\{d\}$. The algorithm's first part starts by expressing the demonstration data in the task space.

```
% Express the demonstration data in the task space
for d = 1 : num_demonstrations
    for r = 1 : num_rows
    % Time column is the same
    D_task{d}(r,1) = D{d}(r,1);

    % Convert the joint variables to task space variables
    D_task{d}(r,2:7) = UR5_ForwardKinematics( D{d}(r,2:7));

    % Express the end effector forces in the ground frame
    C06 = computed using the joint variables at the row r.
    F6 = [ D{d}(r, 8); D{d}(r, 9); D{d}(r, 10) ];
    F0 = C06 * F6;

    D_task{d}(r, 8) = F0(1);
    D_task{d}(r, 9) = F0(2);
    D_task{d}(r, 10) = F0(3);

    end
end
```

Next, the average initial and end values for the demonstrations can be computed. We do so by adding the initial and final values for each variable separately, and then dividing the sum by the number of demonstrations.

```
% Average demonstration time
t_end = 0;
for d = 1:num_demonstrations
    t_end = t_end + D_task{d}(end,1);
end
t_end = t_end / num_demonstrations;

% Average initial and goal values
Y0_d = linspace(0,0,num_variables);
G_d = linspace(0,0,num_variables);

for col = 1:num_variables
    for dem = 1:num_demonstrations
        Y0_d(col) = Y0_d(col) + D_task{dem}(1,col+1);
        G_d(col) = G_d(col) + D_task{dem}(end,col+1);
    end

    % Now we take the average across many demonstrations
    Y0_d(col) = Y0_d(col) / num_demonstrations;
    G_d(col) = G_d(col) / num_demonstrations;

end
```

The DMP algorithm can now be applied to each variable separately according to the formulation presented in the second chapter. In the formulation above, there are nine variables. The first six provide a task space representation for the robot's configuration. Here, the ground frame components of the end-effector position vector are taken to be the XYZ coordinates, and the end-effector orientation is expressed in terms of XYZ Euler angles. These six variables are used for the initial positional learning phase in the task space. The last three variables are the ground frame components of the forces experienced by the end-effector. This information is incorporated into the learning for the hybrid learning case.

```
% Initialize a time array to be used for the solution
T_output = 1:0.01:t_end;

for col = 1:num_variables
    %  Compute the target driving force to be approximated
    Fd = 0;

    for d = 1:num_demonstrations
        T  = D_task{d}(:,1);
        Y  = D_task{d}(:,col+1);
        Yd = numerical derivative of Y;
        Ydd = numerical derivative of Yd;
        Fdi = tau^2*Ydd - alpha_z*(beta_z*(G_d(col) - Y) - tau*Yd);

        % If the demonstrations have different number of indices,
        % they can be stretched using a simple stretching method
        Fdi = imresize(Fdi, size(T_output), 'cubic');
        Fd = Fd + Fdi;

    end

    % The average target driving force across many demonstrations
    Fd = Fd ./ num_demonstrations;

    % The weights can be found as:
    S = x(T_output).*(G_d(col) - Y0_d(col));
    P = @(i) diag( psi(i, x(T_output) ) );
    W = zeros(N); % weights

    for i = 1:N
        W(i) = (S' * P(i) * Fd) / (S' * P(i) * S);
    end

end
```

After this step, the learned weights can be used to generate a trajectory. In the initial learning phase, we provide the average initial and goal values directly to the solver for the system of differential equations. This generates a single smooth trajectory that is equivalent to the multiple demonstrations.

```
for col = 1:num_variables
    % Solution
    g = G_d(col);
    y0 = Y0_d(col);
    f = @(x) forcing_func(x, psi, N, W, g, y0);

    R0 = [y0; 0; 1];
    Rd = @(t,R)1/tau*[R(2);
                alpha_z*beta_z*(g-R(1)) - alpha_z*R(2) + f(R(3));
                -alpha_x*R(3)];

    [t,R] = ode45(Rd, T_output, R0);
    R1 = R(:,1);
    OUTPUT = [OUTPUT , R1];
end
```

The output trajectory is encoded in the same matrix format as the demonstrations. This allows the capability to learn the generated trajectory as a single demonstration using a similar formulation to what is presented above.

The algorithm can be exemplified by showing the generated trajectory for some training demonstrations. Fig. 3.5 shows several functions that represent the trajectories of some selected variable. These functions are not actual demonstration values but are created manually for exemplary purposes. The second part of the figure shows the equivalent trajectory generated using the DMP implementation presented above.

3.3.2. Main Learning Phase

This learning phase uses the equivalent trajectory generated in the previous phase as a single demonstration and learns the weights using the DMP method. This second phase differs from the first one in the following ways:

- The task space coordinates $(X, Y, Z, \theta_x, \theta_y, \theta_z)$ are converted back to joint space using the inverse kinematics of the robot. Performing the final learning and trajectory generation in terms of the joint variables allows a more straightforward implementation step.
- Instead of multiple demonstrations, the equivalent trajectory is used as a single demonstration.
- Most notably, goal points for the trajectories are altered for novel hole positions.

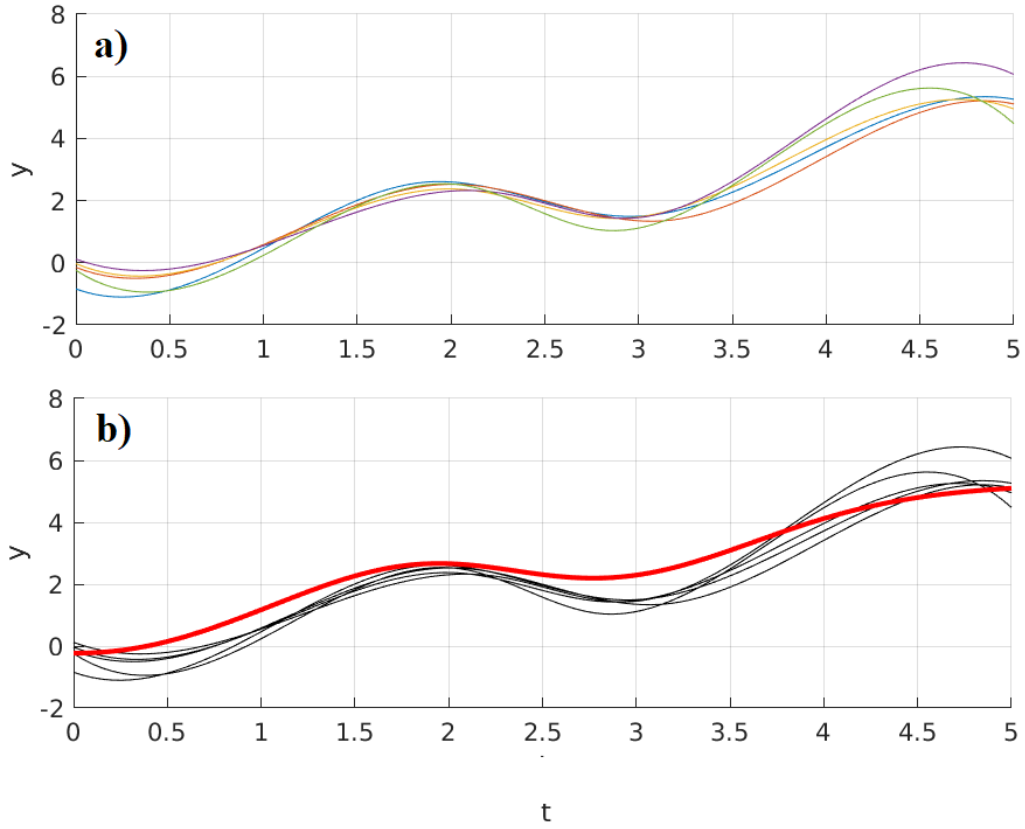

Figure 3.5. a) Multiple demonstrations for some variable.
b) Equivalent trajectory generated using DMP.

To test the performance of the task under novel conditions, the DMP method is used to generate trajectories for 40 new hole locations. These hole locations are organized along with co-centric circles around the original hole position with increasing radii. The radii of the circles are 30 mm, 60 mm, 90 mm, 120 mm, and 150 mm. Each circle features eight holes, equally spaced with 45 degrees between them. Fig. 3.6 shows these hole placements on the XY plane. The Z coordinate of these points is taken to be the average Z coordinate obtained in the demonstrations. This information provides the depth of penetration into the hole, as shown in the demonstrations. Together with the Z coordinate, the hole placements define 40 points in the task space of the robot.

The desired end-effector orientation to achieve at these locations is selected to be looking directly downwards. Inverse kinematics of the robot is used to find the necessary joint variables to obtain these desired poses. The resulting joint variable values are the corresponding goals for each degree of freedom. The target joint values are provided as the goal points for each hole position to the DMP method. Consequently, 40 novel joint-space trajectories are generated. These trajectories are saved into CSV files to be read by a ROS node in the implementation process.

In addition to the joint variables, the Cartesian forces are also learned using the DMP method. The initial learning step generates an equivalent force profile expressed

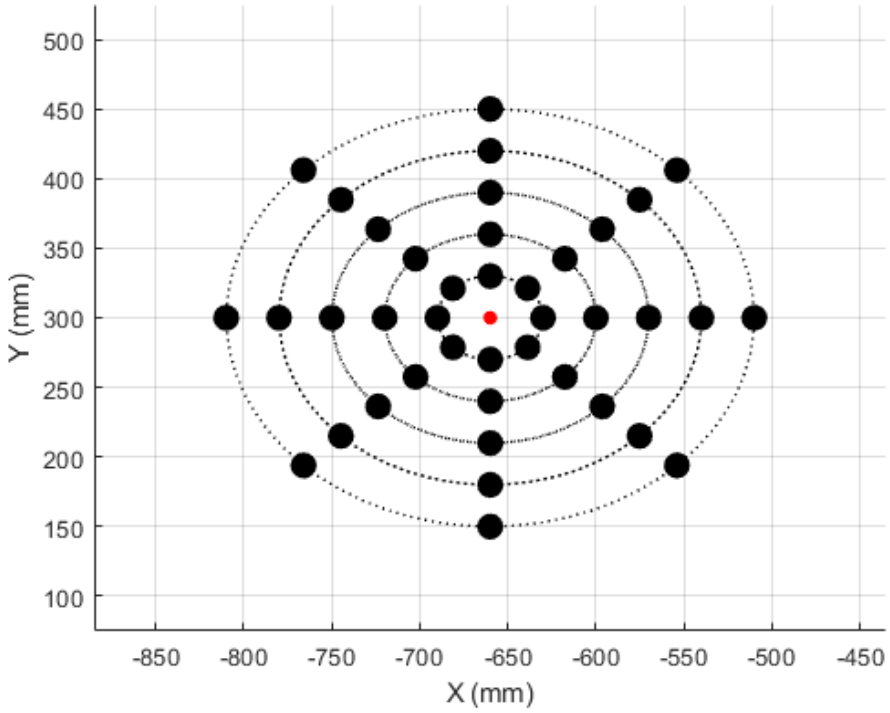

Figure 3.6. Hole positions in the task space (shown in the XY plane).

in terms of the components of the end-effector forces in the ground frame. In the DMP implementation, the components of these force trajectories are learned just like any other degree of freedom. In the trajectory generation step, however, they do not require to be scaled or changed for different hole positions. That is, the target end-effector force is not to be changed to perform the same peg-in-hole task at a different location. As a result, the average end values for each component of the force, as observed from the demonstrations, are used as the goals for the DMP system. The resulting force profiles are also saved as CSV files to be later followed by an admittance controller.

3.4. Implementation

This section discusses the controllers using which the generated trajectories are followed. First, the ROS Control stack is overviewed so that the interfaces provided by the UR ROS driver can be understood. Afterward, sending trajectories to the robot using the Joint Trajectory Action interface is discussed. Finally, the implementation of a hybrid position/force controller using the available position control is presented.

Joint trajectories are sent to the robot via the UR ROS driver. The driver, under the hood, uses the ROS Control stack to provide a standard interface to the robot's resources. ROS control stack is a collection of tools and packages that provide the capability to interact with the robots through ROS. The overall structure in which this stack is used is presented in Fig. 3.7.

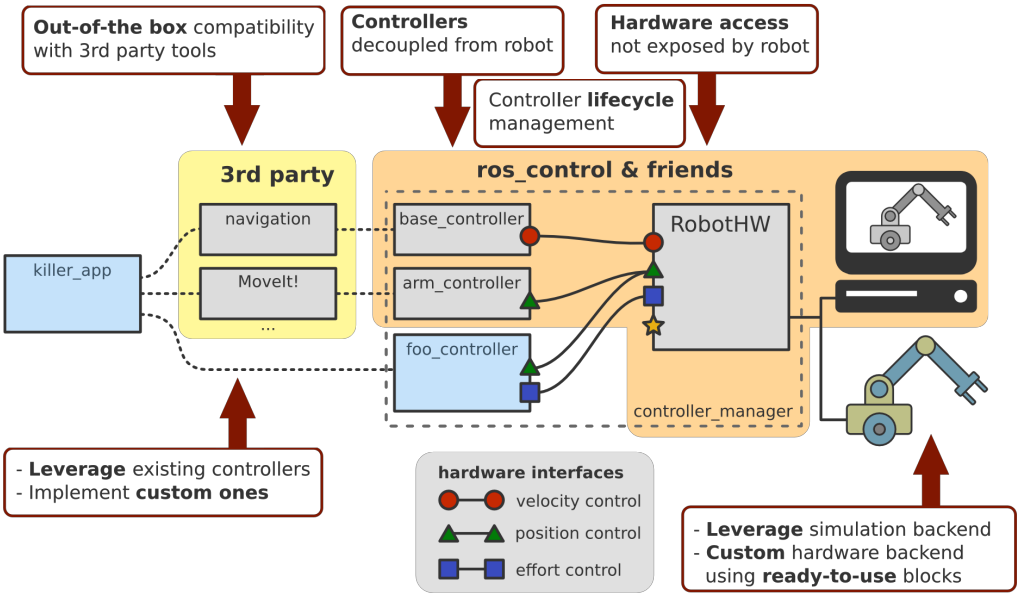

Figure 3.7. A diagram explaning the overall structure of ROS Control.

On the far right of the diagram, there is the actual robot or its simulation. Programatically, the robot is expressed as a collection of tools and resources that can be manipulated. In the `RobotHW` module, these resources are abstracted out so that the controllers can interact with the resources in a generic way. In the UR ROS driver, this is done by interfacing with the RTDE messages that are already sent and received by the robot through the network.

Once the robot hardware is abstracted, controllers that interact with these resources can be written. This layer of potentially several controllers can be seen in Fig. 3.7 on the left of the `RobotHW` module. ROS control provides a collection of controllers with a standard API, so the different robots can be accessed using the same means. In this particular implementation, a joint state controller is instantiated by the UR ROS driver. Using this controller, commands for joint positions can be sent either through a topic or action interface. It is worth pointing out that these ROS controllers are not the actual controllers that control the physical motion of the robot. The robot comes with built-in torque, velocity, and position controllers that operate at a much higher rate. Through the RTDE interface, the robot accepts waypoint commands to its internal controller. The UR ROS driver, in this case, merely abstracts this capability and provides the joint state controller around it. Even though the joint state controller operates in a closed loop by manually checking the current joint state, doing so is almost obsolete as it merely sends control input commands to the existing built-in controllers on the robot.

The controller layer potentially hosts several controllers connected to the same hardware. These controllers are started and potentially switched using a controller manager class, provided again by the ROS control stack. At this layer, if the robot hardware allows, several controllers can be written (impedance, velocity, etc.) and can be switched smoothly

in the runtime using the controller manager. In the case of the UR ROS driver, the only controller that is provided is a joint state controller. This is partly because the UR5 manipulator only allows waypoint commands to be sent.

The joint state controller can be interfaced in one of two ways. The first method would be to directly publish to the joint state command topic. However, using the topics to send movement commands is considered a highly problematic practice. This is primarily because the topic interface does not track the status of the previous command after its execution starts. More importantly, it does not provide a way to intercept the previous command and smoothly transition to the next one. As the second option, an action interface is preferred when working with motion commands. The joint state controller provides an action server called `Joint Trajectory Action` ; which is a standard action type that comes with the ROS Control stack. Requests to this action server can be made by sending a `JointTrajectoryGoal`, which is yet another standard message type that comes with the `control_msgs` package under the ROS Control stack.

As its members, the `JointTrajectoryGoal` message includes a vector of waypoints of the type `JointTrajectoryPoint`. Each waypoint holds four pieces of data: joint positions, velocities, accelerations, and the time at which the waypoint is to be reached. The controller generates a smooth trajectory between the waypoints, satisfying the provided position, velocity, and acceleration values, and sends it to the robot to be executed. By default, the velocities and accelerations can be left as zero, and the corresponding values will be calculated by the controller only by using the provided position and time information using cubic spline interpolation. If the velocities at these waypoints are also provided, a fifth order spline is used instead.

There are two potential uses of the joint state controller via the Joint Trajectory action interface. The first option is to generate a single trajectory goal using all the known waypoints and send it in one step. This way, the robot is programmed offline, and it executes the trajectory it is provided with in the runtime. This method can be used to perform simple positional control, as is the case with the first learning strategy in this thesis. However, it lacks the capability to dynamically respond to changes happening during runtime.

The second way of using the joint state controller is by sending the waypoints one by one in a real-time loop. Here, the loop frequency can be controlled using a `ROS::Rate()` object. In general, modern computers have no problem doing computations and sending waypoints through the network at a rate of around 100 Hz. That is, such a rate can easily be achieved and maintained securely on modern hardware. In more critical applications, real time version of the Linux Kernel can be used to provide even more consistent execution times. As the new waypoints or trajectories are sent through the action interface, the action server processes them according to the preemption policy. The preemption policy of the action server states that once a new command is received, any future waypoints

that are coming from a previous command are discarded. As the waypoints are discarded dynamically, spline trajectories between the old and the new waypoints are generated automatically. Here, the action server guarantees smoothness of the trajectory and the satisfaction of the desired velocity (if provided) at the waypoints. The points in this smooth trajectory are sent to the robot through the robot hardware interface by the controller, and is followed by the built-in controllers of the robot operating at possibly much higher rates.

This approach outlines a strategy using which position commands can be sent to the robot dynamically. As a workaround, any desired velocity can be sent as a velocity condition at some desired waypoint. In this case, the next target position can be calculated simply by incrementing the current one with the velocity times the sampling time. Doing so allows controlling the velocity of the joints indirectly through an otherwise position-based controller.

The hybrid learning approach requires a force controller to be actualized. Such a controller can take many forms, especially if the robot can be controlled by directly providing joint torques. However, UR5 does not allow joint torque commands. Hence, the controller should be built around using the joint state controller. In this case, a second-order admittance control is employed with a changing equilibrium position as a control scheme, as shown in the diagram 3.8. The output force is measured by a force-torque sensor at the end-effector and resolved in the ground frame using the orientation of the end effector at that particular time. This value is compared with a desired force value coming from the previously generated CSV file. The error in the force is then used to solve the dynamics of a second-order virtual mass-spring-damper system discretely to generate an extra displacement in the task space. This extra displacement acts as the output of a second-order Cartesian admittance control. That is, the robot moves as if the environment behaves as a virtual mass-spring-damper system until it experiences the desired force. This same scheme can be implemented either for a desired force to be exerted by the end-effector, or experienced by it. The difference between these cases is only gain of -1 in front of the system. Also, the parameters of the virtual mass-spring-damper system can be selected via trial and error in that particular environment. For the peg-in-hole task that is the interest of this thesis, the environment can be assumed to be highly stiff and with little to no damping. The extra Cartesian displacement that is found as a result of this calculation is added to the current task space coordinate of the robot. The resulting pose is solved using inverse kinematics, and the corresponding joint values are sent to the joint state controller as the next waypoint. Additionally, the task space velocities resulting from the desired position level trajectories and the output of the mass-spring-damper system are combined and converted to the joint space using the jacobian matrix. The joint space velocities are provided as velocity conditions to the corresponding waypoints so that the trajectory action interface can generate smooth spline fits between them.

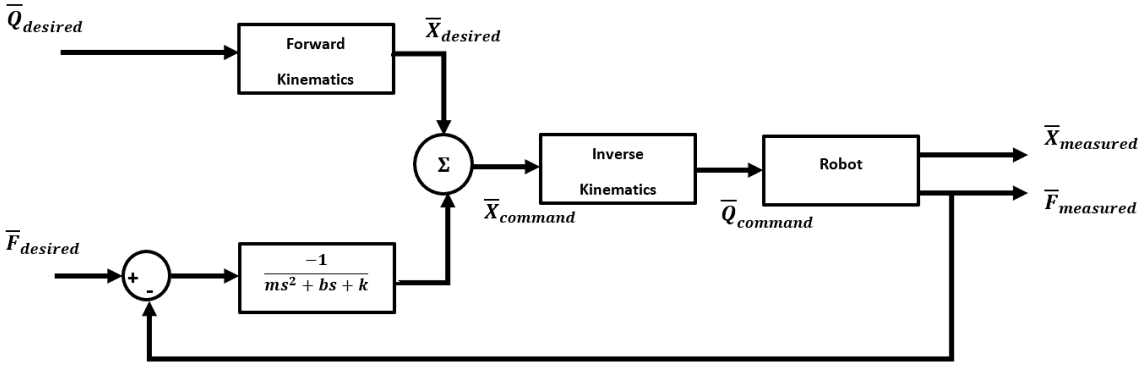

Figure 3.8. The control system diagram for the admittance control of the robot.

CHAPTER 4

RESULTS

4.1. Demonstrations

Demonstrations are performed in the manner that is discussed in the methodology section. The joint variables encountered during these demonstrations can be seen in Fig. 4.1.

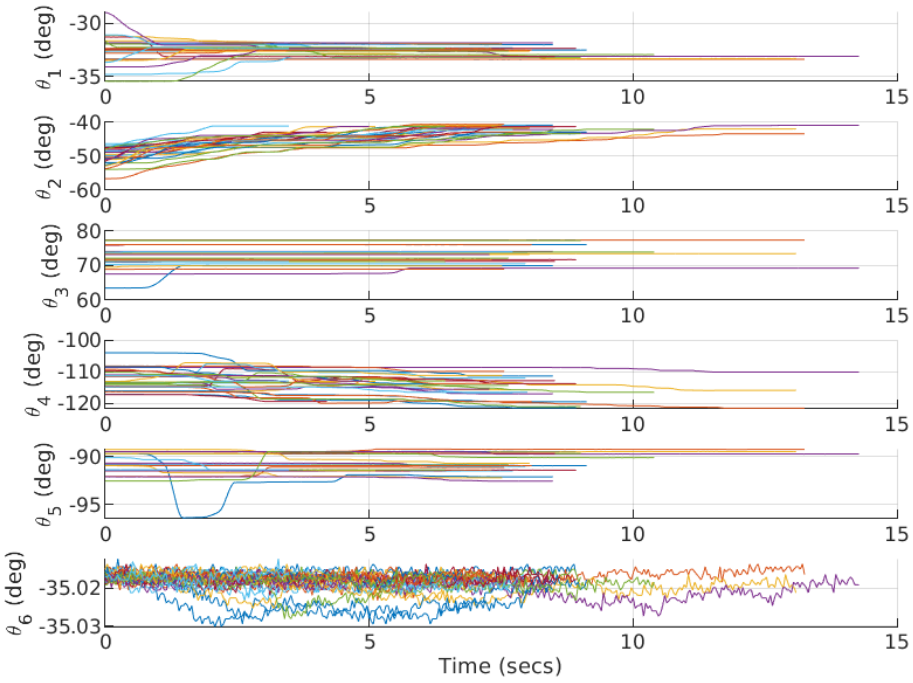

Figure 4.1. Demonstration data, joint space coordinates of the robot.

Two remarks can be made about this plot. The first is that the joint variables for different demonstrations start at different values, indicating the different initial conditions. Secondly, they all converge to a particular value for each degree of freedom. These values are the joint space coordinates corresponding to the robot configuration when the peg is in the hole. Fig. 4.2 shows the XYZ coordinates of the same trajectories in the task space. The points long the shown curves are the end-effector positions at those instances, computed using the forward kinematics of the robot. To keep the plot simple, the information regarding the orientation of the end effector along these trajectories is omitted. This plot shows more clearly that the several demonstrations start at different initial conditions and end up approximately at the same place where the hole is located.

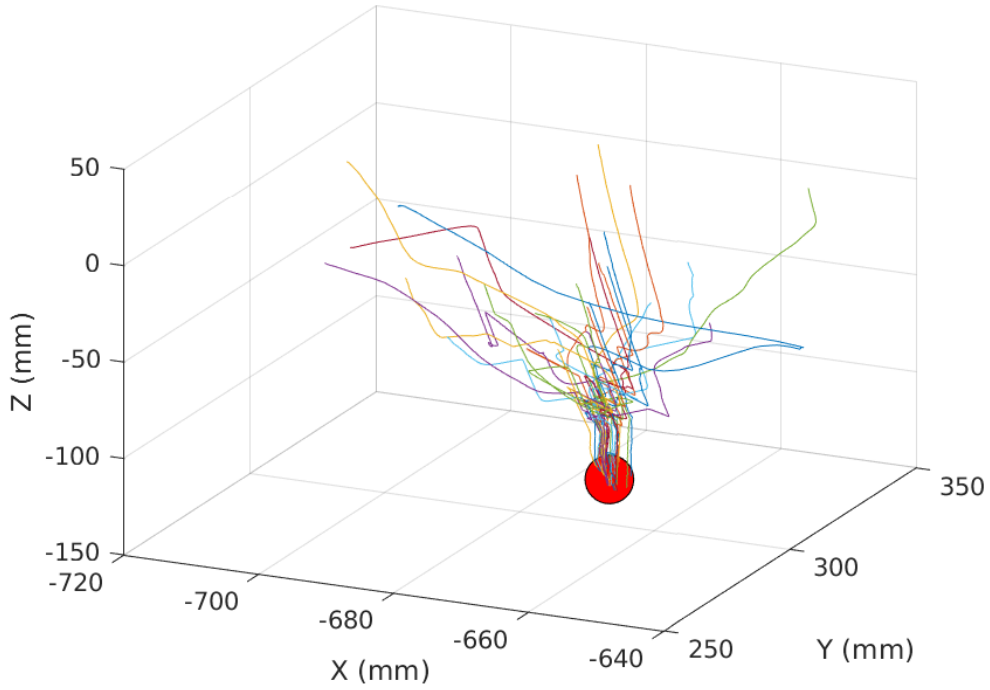

Figure 4.2. Demonstration data, shown in the Cartesian task space. Computed using the forward kinematics.

Additionally, end-effector forces that are encountered during these demonstrations are also recorded. Fig. 4.3 shows the ground frame components of these end-effector forces. The first remark about this data is that the magnitudes of the forces are almost insignificant, ranging around 0.2 N. Secondly, the end-effector experiences noisy forces during the initial phase of the motion. There are two factors that can explain this phenomenon:

- Human demonstrator might be unintentionally touching the flange of the robot and inducing artificial force readings
- The end-effector might experience inertial forces as it accelerates towards the peg

Once the insertion phase starts, the magnitudes of each of the force components diminish significantly. This indicates that a successful peg-in-hole application is one in which the end-effector performs the insertion without experiencing high reaction forces.

4.2. Representation

DMP method is applied to the demonstration data in the manner that is discussed in the theory and methodology sections. In the initial learning phase, the task space representations of these demonstrations are learned to generate an equivalent trajectory. Fig. 4.4 shows this curve in red color. This plot shows a clear example of how the DMP

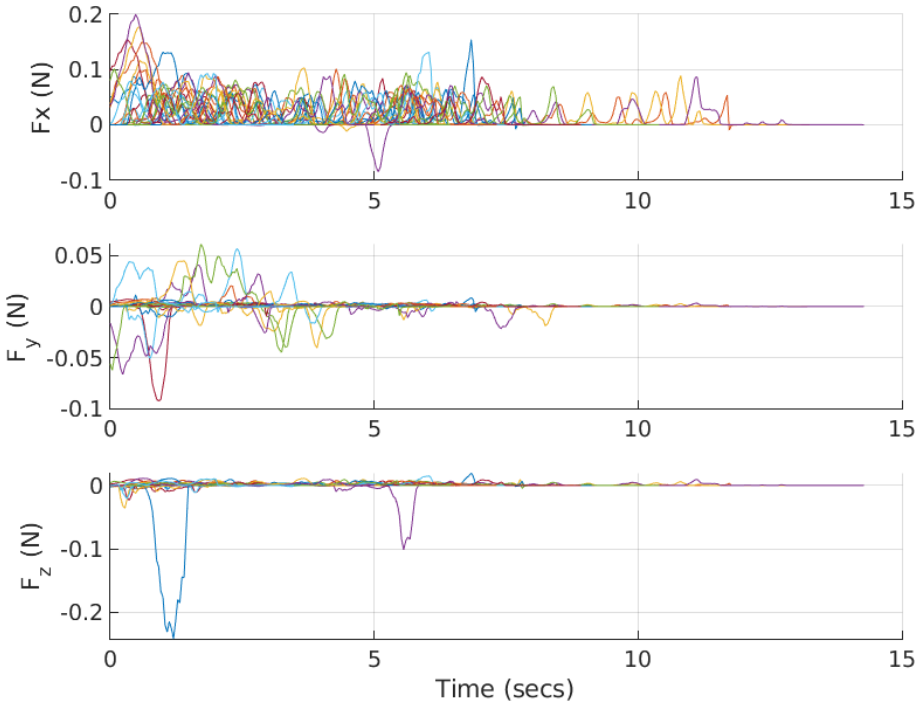

Figure 4.3. Ground frame components of the forces experienced by the end-effector.

method learns the geometric characteristics of the task. The curve's upper part displays a maneuver towards the hole position. On the other hand, the lower part picks up on the demonstrated insertion motion and generates a straight line moving almost directly downwards.

In the second phase, this equivalent curve is converted back to joint space to be learned again in the second phase of the representation process. The resulting joint space trajectories are shown in Fig. 4.5. Similarly, each force component is reduced to an equivalent trajectory between their average initial and final values. Fig 4.6 shows the equivalent force profiles obtained from the demonstrations.

These nine trajectories (six joint variables and three force components) are fed into the DMP method as demonstrations to be learned. As a result, the weights used in the trajectory generation step are acquired. The novel hole positions that are discussed in the methodology section are expressed in the joint space using the inverse kinematics of the robot. Here, the target orientation at the hole position is taken to be pointing directly downwards. Acquired joint values are used as goals for each degree of freedom and provided as goals to the DMP method. Fig. 4.7 shows the end-effector positions of generated trajectories in the task space for each hole position.

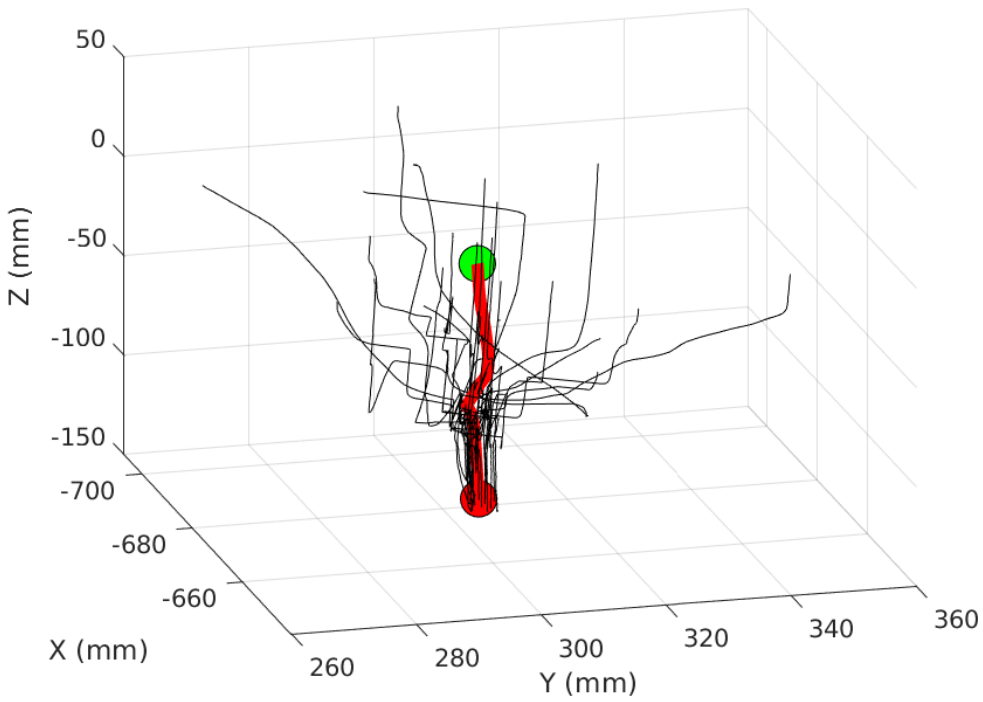

Figure 4.4. Demonstrated trajectories in task space (black), the equivalent trajectory generated using DMP (red).

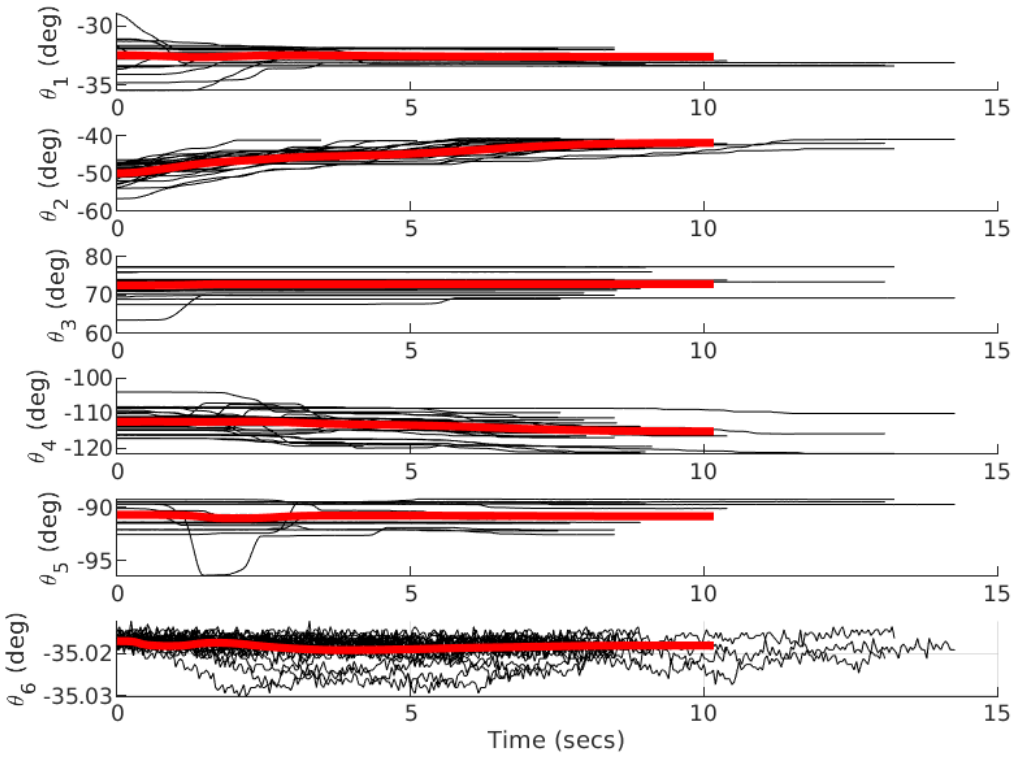

Figure 4.5. Demonstrated trajectories in joint space (black), the equivalent trajectory generated using DMP (red).

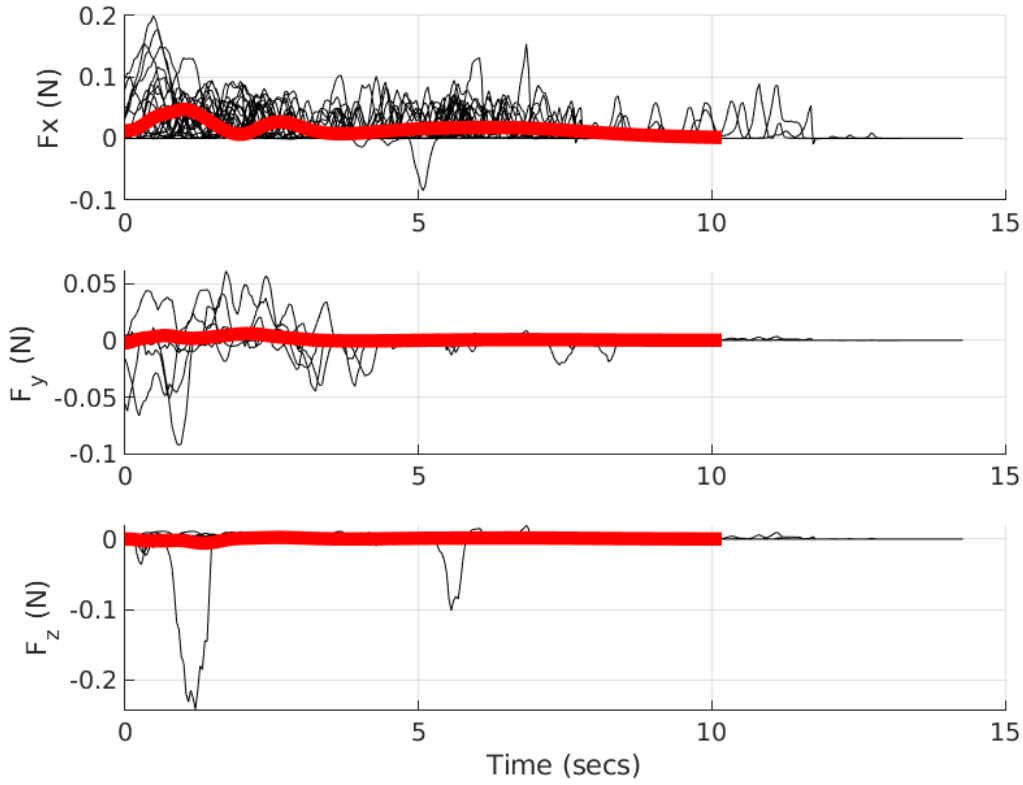

Figure 4.6. Demonstrated force profiles (black), the equivalent force profiles generated using DMP (red).

For the force profiles, the learning is performed using the ground frame components of the end-effector force. This is because the hybrid position/force controller discussed in the methodology section uses a Cartesian admittance control approach for force compensation. Average end-values of the demonstrations are used as the goals for each component, Consequently, the resulting force trajectories are the same as those shown in Fig. 4.6.

4.3. Implementation

The experiments are then carried out in two separate parts. The first part attempts to answer the first research question by explicating the relationship between the success and the novelty of the task. As shown in Fig. 4.7, generated trajectories become increasingly warped as the goal position gets further away from the original position. This effect shows a clear challenge towards the generalization of the task for novel hole positions. The change of success rate for the different hole positions is tested by sending the generated joint trajectories to the robot, using the joint state controller discussed in the methodology chapter. If the peg is successfully inserted at a hole position, then that case is accepted as a success. Consequently, the hole positions at which the peg could not be inserted into the

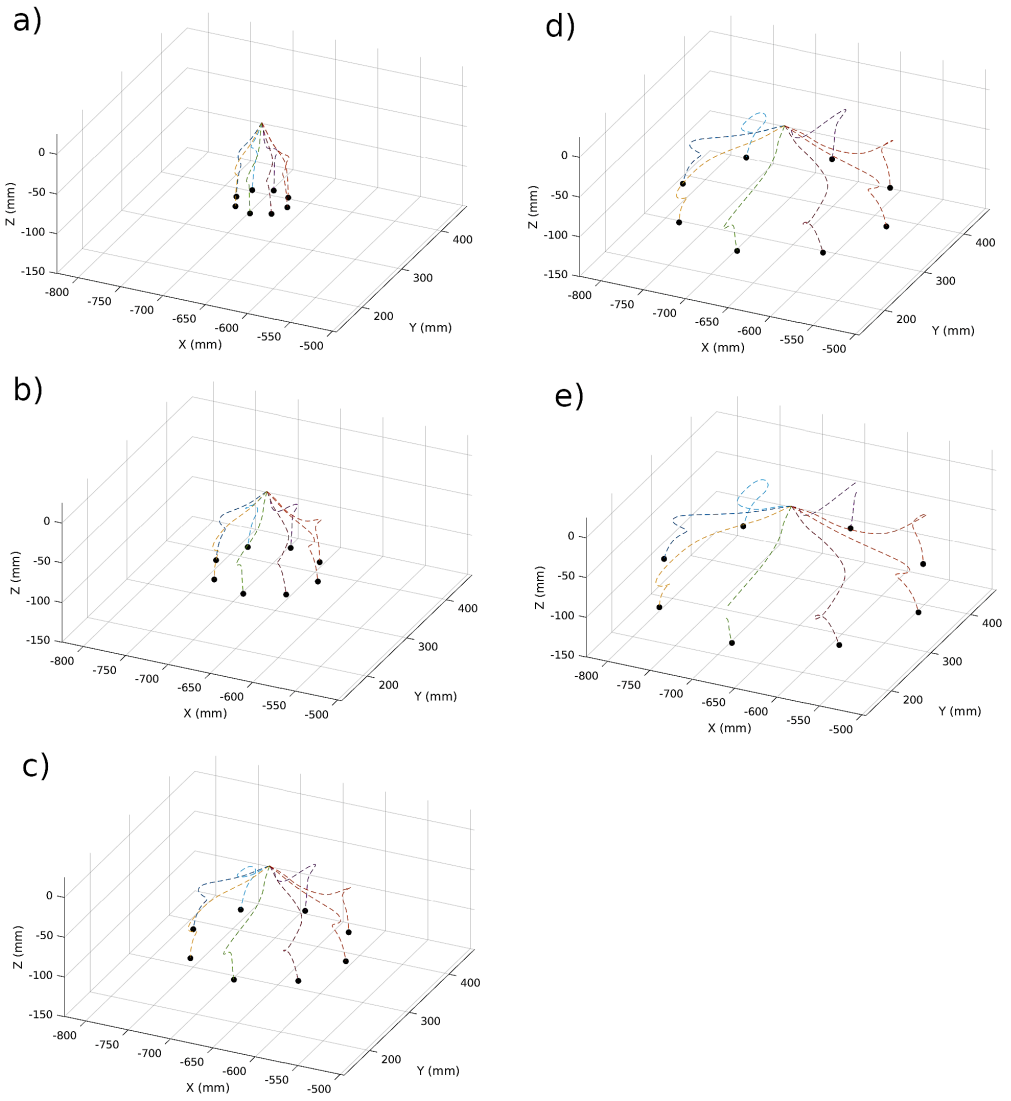

Figure 4.7. Generated trajectories shown in the task space, for the hole positions at radius levels a) 30 mm, b) 60 mm, c) 90 mm, d) 120 mm, e)150 mm.

hole are taken to be the failure cases. Fig. 4.8 shows the success and failure of the task at different hole positions. In line with the expected results, the task shows a clear pattern of increasing failure as the distance to the original hole position increases.

It is also worth pointing out that the success or failure at a hole position remains unaffected between different executions. That is, if a hole position is a failure case, it always is a failure case. This is purposefully designed so by the selection of the hole clearances. The difference between the hole and peg diameters is considerably higher than the repeatability of the UR5 manipulator, even at the outer regions of its workspace. As a result, the effects repeatability and the resolution of the robot on the success/failure of the tasks are mostly eliminated by experiment design. This allows the isolation of only the effects caused by the intrinsic strengths or weaknesses of the PbD approach tested.

The second research question this thesis attempts to answer is the generalization

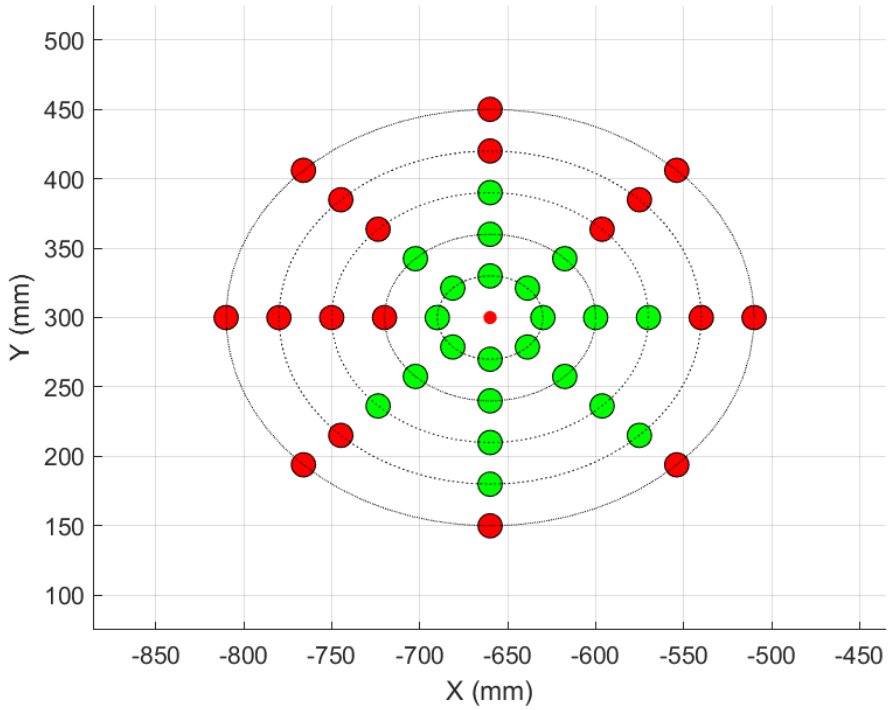

Figure 4.8. Hole positions at which the task was succesfull (green), or failure (red) (positional learning).

performance of a hybrid position/force learning strategy. Consequently, the second part of the experiment performs the task at the same hole locations but using a hybrid strategy this time. In addition to the generated joint trajectories, desired force profile is also sent to the hybrid position/force controller. The controller provides the robot the capability to move in a certain direction until it experiences the desired force. This allows the robot to perform small motions according to the learned force profile as it approaches the hole. From another perspective, the robot moves in the opposite direction if it experiences a discrepancy between the measured and the desired forces. This effect is practically the same as impedance control during the later phases of the insertion motion, where the desired force profile approaches zero. The robot escapes from the sides of the hole as it experiences lateral forces and aligns itself at the center. The success and failures of the task obtained using this strategy are shown in Fig. 4.9.

The results obtained from these experiments can be visualized in several ways. The first option is to visualize whether the peg-in-hole task was successful or not at the particular hole positions. Fig. 4.8 displays this information for the positional learning case, and Fig. 4.9 displays it for the hybrid position/force learning case. The first critical remark about these graphs is that the task was not successful at the hole positions further away from the original hole position, especially for the positional learning case. Secondly, the hybrid position/force learning strategy yields a higher number of success cases at each radius level. For each of these levels, we can compute a success ratio as the ratio of the success cases and the total number of hole positions at that level. The success ratio, used as

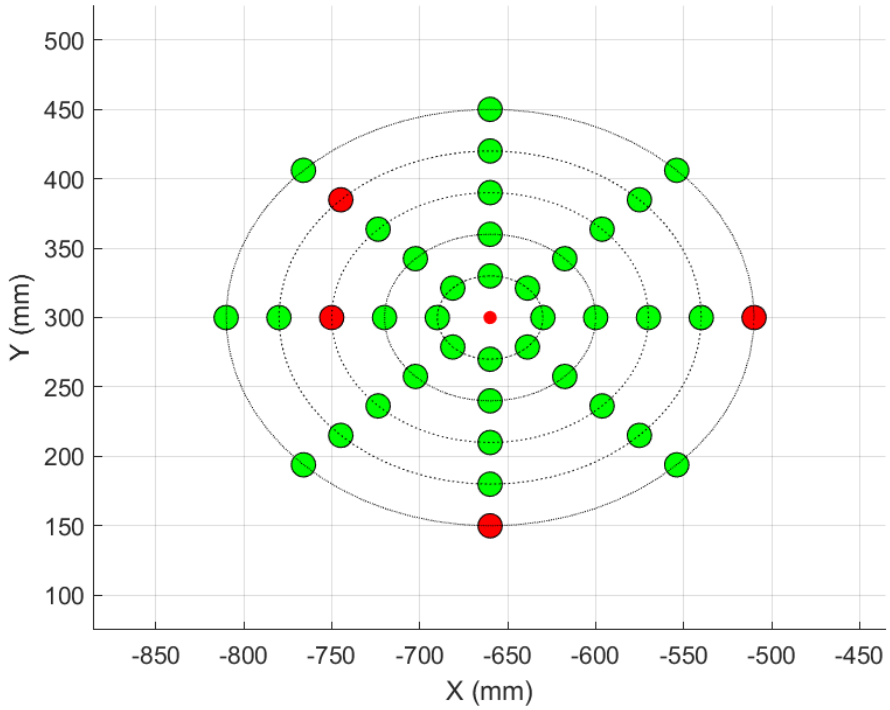

Figure 4.9. Hole positions at which the task was succesfull (green), or failure (red) (hybrid position/force learning).

a metric, provides a way to quantify the performance of the strategies at different distances to the original hole position. This comparison answers the final research question of the thesis as it compares the generalization performance of the two strategies. Fig. 4.10 displays the success ratio of the two strategies at each radius level. As can be seen from the graph, the hybrid position/force learning strategy outperforms the purely positional learning one at all distances, especially at higher ones.

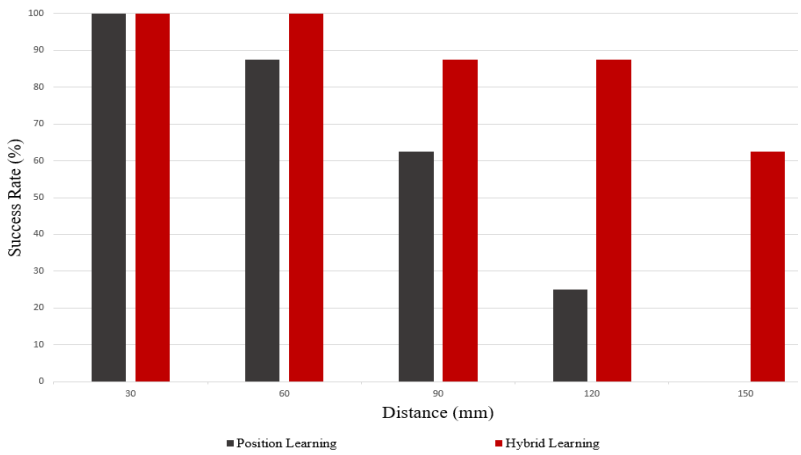

Figure 4.10. Success rate of the strategies at different distances to the original hole position.

CHAPTER 5

CONCLUSIONS

Robots are reprogrammable machines by their very nature. Traditionally, this is done by hard-coding the waypoints they must pass through to perform the desired task. However, this approach fails to adapt to changes in the environment dynamically. The unstructured environments of today's workplaces require a change of approach. PbD is a promising alternative approach that allows the robots to learn tasks not via hard-coding but from demonstrations.

A primary area of application for the PbD is the field of robotic assembly. Research in the field develops frameworks that learn assembly operations in relation to their constituent subtasks, such as the peg-in-hole task. The need to generalize the peg-in-hole task for changing conditions is an area of great interest in the field. This thesis aims to improve the generalization performance of the peg-in-hole task for varied hole positions in particular. It does so by testing out the effect of two strategies on the success rate of the task. The questions tackled with this study provide a way to investigate the relationship between the novelty of the hole position and the success rate of the task.

The first research question examines the change of success ratio for increasingly distant hole positions. To answer this question, demonstrations for a peg-in-hole task are performed while keeping the hole at a fixed location. The executions are then carried out at different hole positions organized along with circles with increasing radii. Fig. 4.8 shows the success and failures of the task at each hole position.

It can be observed from these results that the inner circles include a higher number of success cases than the outer ones. This establishes an experimental validation of the hypothesis that the success rate of the task drops as the distance to the original hole position increases. To the degree that distance to the original hole position is a source of novelty, the relationship can be extended to a general one between success and novelty. A direct comparison of the success rate and the distance can be seen in Fig. 4.10. The decrease is so much so that there are no successful executions in the outermost circle.

There can be made another observation about the results of positional learning. The inner and outer radius levels have 100% success and failure, respectively, and the intermediate levels display diminishing success rates. A reasonable question would be to ask what determines the success or failure at a particular hole location and not at a whole radius level. The hole positions along the 90 mm radius circle have a 62.5% success rate, but what factors determine which hole is a success case? This question can be answered by referring back to the task space plot of the demonstrations, shown in Fig. 4.4. The equivalent trajectory extracted from the demonstrations includes a maneuver to approach

the hole. This motion captures the way in which the teacher performed the motion. If the demonstrations in which the hole is approached from the left side are over-represented in the demonstration data, the generated trajectories also include this feature. The hole positions that are similar in the approach direction to the demonstrations show a pattern of being more successful. This extends the previous finding that the success rate drops as the novelty of the task increases.

The second question this thesis asks is the change of success rate for the hybrid position/force learning approach. This question is answered in a similar way to the first one. Demonstrations for the peg-in-hole task are learned using not only the configuration of the robot but also the forces the end-effector experiences. As a result, the generated trajectories include not only the joint space trajectories but also desired force profiles to be followed. The executions are carried out by sending these trajectories to an admittance controller. The success and failure cases at each hole position are shown in Fig. 4.9.

The results for the hybrid learning case show two critical findings. The first one is an overall increase in task performance. The number of hole positions at which the task is successful is much higher than the purely positional learning case. Secondly, the success ratio still decreases for the increased distance to the hole position. This indicates that, even though there is a performance increase, the task's success is still negatively affected by the novelty.

The third question this thesis asks is a direct comparison between the positional and hybrid learning strategies. Even though both strategies show diminishing success rates for increased novelty, the rate of decrease is different in both strategies. The hybrid learning case not only performs better at each radius level but also has less relative success drop between levels. This indicates that the hybrid learning case shows more promise in dealing with the increased novelty of the task.

This thesis adds novelty to the peg-in-hole task only by changing the position of the hole. There are other ways in which the novelty of the task can be changed. Prominent examples include a variation in the hole sizes and orientations. Explicating the relationship between the success rate and them is a promising candidate for future work.

The further work that can be undertaken includes strategies to provide even more generalization performance to the individual tasks and the assembly operations in general. The ultimate aim is to develop PbD frameworks that can learn and generalize any assembly operation in realistic settings. The work of this thesis only improves the generalization performance for varying hole positions of an individual peg-in-hole task. Strategies of this type can be used in conjunction with the established assembly frameworks to increase overall performance. Furthermore, a similar approach can be taken for not only the hole's position but also its orientation. This study employs a Cartesian admittance control scheme only for the translational task space coordinates of the end-effector. A similar approach can be taken for the orientational ones. Doing so might increase the

generalization performance not only for orientation variations but also for size variations. This is because one of the primary sources of failure in the peg-in-hole tasks with tight clearances is misalignment issues between the peg and the hole. These issues can be tackled by the use of orientation-based admittance control schemes.

REFERENCES

Abu-Dakka, Fares J, Bojan Nemec, Aljaž Kramberger, Anders Glent Buch, Norbert Krüger, and Ales Ude. 2014. "Solving peg-in-hole tasks by human demonstration and exception strategies." *Industrial Robot: An International Journal.*

Ambhore, Sushilkumar. 2020. "A Comprehensive Study on Robot Learning from Demonstration." In *2020 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA),* 291–299. IEEE.

Argall, Brenna D., Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. "A Survey of Robot Learning From Demonstration." *Robotics and autonomous systems* 57 (5): 469–483.

Asfour, Tamim, Pedram Azad, Nikolaus Vahrenkamp, Kristian Regenstein, Alexander Bierbaum, Kai Welke, Joachim Schroeder, and Ruediger Dillmann. 2008. "Toward humanoid manipulation in human-centred environments." *Robotics and Autonomous Systems* 56 (1): 54–65.

Billard, Aude, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. 2008. "Survey: Robot programming by demonstration." *Handbook of robotics* 59 (BOOK_CHAP).

Billard, Aude, and Daniel Grollman. 2013. "Robot learning by demonstration." *Scholarpedia* 8 (12): 3824.

Calinon, Sylvain. 2018. "Learning from demonstration (programming by demonstration)." *Encyclopedia of robotics:* 1–8.

Conkey, Adam, and Tucker Hermans. 2019. "Learning task constraints from demonstration for hybrid force/position control." In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids),* 162–169. IEEE.

Daniel, Christian, Gerhard Neumann, and Jan Peters. 2012. "Learning concurrent motor skills in versatile solution spaces." In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 3591–3597. IEEE.

Ding, Guanwen, Yubin Liu, Xizhe Zang, Xuehe Zhang, Gangfeng Liu, and Jie Zhao. 2020. "A Task-Learning Strategy for Robotic Assembly Tasks from Human Demonstrations." *Sensors* 20 (19): 5505.

Duque, David A, Flavio A Prieto, and Jose G Hoyos. 2019. "Trajectory generation for robotic assembly operations using learning by demonstration." *Robotics and Computer-Integrated Manufacturing* 57:292–302.

Edsinger, Aaron Ladd. 2001. "A gestural language for a humanoid robot." PhD diss., Massachusetts Institute of Technology.

Fang, Bin, Shidong Jia, Di Guo, Muhua Xu, Shuhuan Wen, and Fuchun Sun. 2019. "Survey of imitation learning for robotic manipulation." *International Journal of Intelligent Robotics and Applications* 3 (4): 362–369.

Gao, Xiao, Jie Ling, Xiaohui Xiao, and Miao Li. 2019. "Learning force-relevant skills from human demonstration." *Complexity* 2019.

Giszter, S, FA Mussa-Ivaldi, and E Bizzi. 1993. "Movement primitives in the frog spinal cord." In *Neural Systems: Analysis and Modeling,* 431–446. Springer.

Giszter, Simon F, Ferdinando A Mussa-Ivaldi, and Emilio Bizzi. 1993. "Convergent force fields organized in the frog's spinal cord." *Journal of neuroscience* 13 (2): 467–491.

Hersch, Micha, Florent Guenter, Sylvain Calinon, and Aude G Billard. 2006. "Learning dynamical system modulation for constrained reaching tasks." In *2006 6th IEEE-RAS International Conference on Humanoid Robots,* 444–449. IEEE.

Hu, Siyao, and Katherine J Kuchenbecker. 2019. "Hierarchical task-parameterized learning from demonstration for collaborative object movement." *Applied Bionics and Biomechanics* 2019.

Hussein, Ahmed, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. "Imitation learning: A survey of learning methods." *ACM Computing Surveys (CSUR)* 50 (2): 1–35.

Ijspeert, AJ. 2003. "Learning control policies for movement imitation and movement recognition." In *Neural information processing system,* 15:1547–1554.

Ijspeert, Auke Jan, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. 2013. "Dynamical movement primitives: learning attractor models for motor behaviors." *Neural computation* 25 (2): 328–373.

Ijspeert, Auke, Jun Nakanishi, and Stefan Schaal. 2002. "Learning attractor landscapes for learning motor primitives." *Advances in neural information processing systems* 15.

Kober, Jens, and Jan Peters. 2009. "Learning motor primitives for robotics." In *2009 IEEE International Conference on Robotics and Automation,* 2112–2118. IEEE.

Koenemann, Jonas, Felix Burget, and Maren Bennewitz. 2014. "Real-time imitation of human whole-body motions by humanoids." In *2014 IEEE International Conference on Robotics and Automation (ICRA),* 2806–2812. IEEE.

Koropouli, Vasiliki, Dongheui Lee, and Sandra Hirche. 2011. "Learning interaction control policies by demonstration." In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 344–349. IEEE.

Kramberger, Aljaž. 2014. "A comparison of learning-by-demonstration methods for force-based robot skills." In *2014 23rd International Conference on Robotics in Alpe-Adria-Danube Region (RAAD),* 1–6. IEEE.

Kramberger, Aljaž, Andrej Gams, Bojan Nemec, Dimitrios Chrysostomou, Ole Madsen, and Aleš Ude. 2017. "Generalization of orientation trajectories and force-torque profiles for robotic assembly." *Robotics and autonomous systems* 98:333–346.

Krüger, Jörg, Terje K Lien, and Alexander Verl. 2009. "Cooperation of human and machines in assembly lines." *CIRP annals* 58 (2): 628–646.

Kumar, Vikash, Abhishek Gupta, Emanuel Todorov, and Sergey Levine. 2016. "Learning dexterous manipulation policies from experience and imitation." *arXiv preprint arXiv:1611.05095.*

Kyrarini, Maria, Muhammad Abdul Haseeb, Danijela Ristić-Durrant, and Axel Gräser. 2019. "Robot learning of industrial assembly task via human demonstrations." *Autonomous Robots* 43 (1): 239–257.

Nehaniv, Chrystopher L, Kerstin Dautenhahn, et al. 2002. "The correspondence problem." *Imitation in animals and artifacts* 41.

Nemec, Bojan, Fares J Abu-Dakka, Barry Ridge, Aleš Ude, Jimmy A Jørgensen, Thiusius Rajeeth Savarimuthu, Jerome Jouffroy, Henrik G Petersen, and Nobert Krüger. 2013. "Transfer of assembly operations to new workpiece poses by adaptation to the desired force profile." In *2013 16th International Conference on Advanced Robotics (ICAR),* 1–7. IEEE.

Rozo, Leonel, Danilo Bruno, Sylvain Calinon, and Darwin G Caldwell. 2015. "Learning optimal controllers in human-robot cooperative transportation tasks with position and force constraints." In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS),* 1024–1030. IEEE.

Rozo, Leonel, Sylvain Calinon, and Darwin G Caldwell. 2014. "Learning force and position constraints in human-robot cooperative transportation." In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication,* 619–624. IEEE.

Rozo, Leonel, Pablo Jiménez, and Carme Torras. 2011. “Robot learning from demonstration of force-based tasks with multiple solution trajectories.” In *2011 15th International Conference on Advanced Robotics (ICAR),* 124–129. IEEE.

Schaal, Stefan, et al. 1997. “Learning from demonstration.” *Advances in neural information processing systems:* 1040–1046.

Silvério, Joao, Yanlong Huang, Leonel Rozo, Sylvain Calinon, and Darwin G Caldwell. 2018. “Probabilistic learning of torque controllers from kinematic and force constraints.” In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* 1–8. IEEE.

Skubic, Marjorie, and Richard A Volz. 1998. “Learning force-based assembly skills from human demonstration for execution in unstructured environments.” In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146),* 2:1281–1288. IEEE.

Steinmetz, Franz, Alberto Montebelli, and Ville Kyrki. 2015. “Simultaneous kinesthetic teaching of positional and force requirements for sequential in-contact tasks.” In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids),* 202–209. IEEE.

Tang, Te, Hsien-Chung Lin, and Masayoshi Tomizuka. 2015. “A learning-based framework for robot peg-hole-insertion.” In *Dynamic Systems and Control Conference,* vol. 57250, V002T27A002. American Society of Mechanical Engineers.

Ti, Boyang, Yongsheng Gao, Ming Shi, and Jie Zhao. 2022. “Generalization of orientation trajectories and force–torque profiles for learning human assembly skill.” *Robotics and Computer-Integrated Manufacturing* 76:102325.

Wang, Yue, Rong Xiong, Longbin Shen, Kaixiao Sun, Jiafan Zhang, and Liwei Qi. 2014. “Towards learning from demonstration system for parts assembly: A graph based representation for knowledge.” In *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent,* 174–179. IEEE.

Williamson, Matthew M. 1996. “Postural primitives:: Interactive Behavior for.” In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior,* 4:124. MIT Press.

Yang, Yang, LL Lin, YT Song, Bojan Nemec, Ales Ude, Anders Glent Buch, Norbert Krüger, and Thiusius Rajeeth Savarimuthu. 2014. “Fast programming of peg-in-hole actions by human demonstration.” In *2014 International Conference on Mechatronics and Control (ICMC),* 990–995. IEEE.

Zhu, Zuyuan. 2020. “Robot Learning Assembly Tasks from Human Demonstrations.” PhD diss., University of Essex.

Zhu, Zuyuan, and Huosheng Hu. 2018. “Robot learning from demonstration in robotic assembly: A survey.” *Robotics* 7 (2): 17.