

# EFFECTIVE NEURAL NETWORKS FOR LEARNING FROM DEMONSTRATION

A Thesis

by

Arash Mehrabi

Submitted to the  
Graduate School of Sciences and Engineering  
in Partial Fulfillment of the Requirements for  
the Degree of

Master of Science

in the  
Department of Artificial Intelligence

Özyeğin University  
September 2024

Copyright © 2024 by Arash Mehrabi

# EFFECTIVE NEURAL NETWORKS FOR LEARNING FROM DEMONSTRATION

Approved by:

---

Prof. Dr. Erhan Öztop, Advisor  
Dept. of Artificial Intelligence and Data  
Engineering  
*Özyeğin University*

---

Assoc. Prof. Dr. Emre Uğur  
Dept. of Computer Engineering  
*Boğaziçi University*

---

Assoc. Prof. Dr. Özkan Bebek  
Dept. of Mechanical Engineering  
*Özyeğin University*

Date Approved: August 12, 2024



*To my parents, my sister, and Zahra*

# ABSTRACT

Learning from demonstration (LfD) is a powerful technique for teaching robots new behaviors by mimicking human demonstrations. One widely used approach in LfD is Behavior Cloning (BC) with human-in-the-loop control. In this method, data collected from humans demonstrations is utilized to create a non-linear controller by learning a mapping from states to desired actions. In this study, we propose a novel BC system that its structure mirrors the structure of an error-based feedback controller. This design choice is based on the premise that embedding such a structure within the learning model can endow our system with a prior bias, resulting in an inherent advantage over controller-agnostic BC systems. This thesis details the components of this novel BC model and demonstrates its application on a two degrees-of-freedom robotic system across various tasks as a proof of concept. To assess the system’s effectiveness, we conducted systematic experiments comparing it to a controller-agnostic BC system. The results reveal that our proposed model significantly outperforms the baseline, suggesting it is a promising candidate for LfD tasks where the demonstration data can be assumed to be generated by a feedback controller, especially in resource-scarce conditions.

## ÖZETÇE

Gösterimden öğrenme (Learning from demonstration, LfD), insan gösterilerini taklit ederek robotlara yeni davranışlar öğretmek için güçlü bir tekniktir. LfD’de yaygın olarak kullanılan yaklaşımlardan biri, döngü içinde insan kontrollü Davranış Klonlama (Behavior Cloning, BC) yöntemidir. Bu yöntemde, insan gösterilerinden toplanan veriler, durumlardan istenen eylemlere bir eşleme öğrenerek doğrusal olmayan bir kontrolör oluşturmak için kullanılır. Bu çalışmada, yapısı hata tabanlı bir geri besleme kontrolörünün yapısını yansıtan yeni bir BC sistemi öneriyoruz. Bu tasarım tercihi, öğrenme modelinin içine böyle bir yapının yerleştirilmesinin sistemimize önceden bir önyargı kazandırabileceği ve bunun da denetleyiciden bağımsız BC sistemlerine göre doğal bir avantaj sağlayacağı varsayımına dayanmaktadır.

Bu tez, bu yeni BC modelinin bileşenlerini detaylandırmakta ve kavram kanıtı olarak çeşitli görevlerde iki serbestlik dereceli bir robotik sistem üzerindeki uygulamasını göstermektedir. Sistemin etkinliğini değerlendirmek için, denetleyiciden bağımsız bir BC sistemiyle karşılaştıran sistematik deneyler gerçekleştirdik. Sonuçlar, önerilen modelimizin temel modelden önemli ölçüde daha iyi performans gösterdiğini ortaya koymakta ve özellikle kaynak kısıtlılığı koşullarında, gösterim verilerinin bir geri besleme denetleyicisi tarafından üretildiğinin varsayılabilceği LfD görevleri için umut verici bir aday olduğunu göstermektedir.

## ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Prof. Erhan Oztop, for his guidance and support throughout my master’s studies. His leadership, vast knowledge, and patience have been invaluable on this journey.

I would also like to thank the rest of my thesis committee: Dr. Emre Ugur and Dr. Ozkan Bebek, for their insightful feedback on this thesis and their support in our previous collaborations.

I would like to extend my sincere thanks to my colleagues at the Artificial Intelligence Lab at Özyeğin University, including Bedirhan Çelebi, Emir Arditi, Negin Amirshirzad, Mehmet Arda Eren, Hanne Say, Hamid Majidi, and Amin D. Alamdari, among others. Their friendship and mentorship have enriched my experience throughout this journey. I’m particularly grateful to Bedirhan Çelebi who read the whole manuscript and provided detailed feedback.

My heartfelt thanks go to my parents, Akram and Siamak, and my sister, Nazli. Their unconditional love, support, and guidance have been the foundation of my achievements.

Finally, I would like to express my gratitude to my wife, Zahra. Over the past three years, her unwavering companionship, love, and support have been crucial to my success. I couldn’t have imagined a better companion than her.

# TABLE OF CONTENTS

|                                                   |          |
|---------------------------------------------------|----------|
| DEDICATION . . . . .                              | iii      |
| ABSTRACT . . . . .                                | iv       |
| ÖZETÇE . . . . .                                  | v        |
| ACKNOWLEDGEMENTS . . . . .                        | vi       |
| LIST OF TABLES . . . . .                          | x        |
| LIST OF FIGURES . . . . .                         | xi       |
| <b>I INTRODUCTION . . . . .</b>                   | <b>1</b> |
| 1.1 Thesis Contribution . . . . .                 | 3        |
| 1.2 Thesis Outline . . . . .                      | 3        |
| <b>II BACKGROUND AND RELATED WORK . . . . .</b>   | <b>4</b> |
| 2.1 Problem Definition . . . . .                  | 4        |
| 2.1.1 Demonstrations . . . . .                    | 4        |
| 2.1.2 Policy Learning . . . . .                   | 4        |
| 2.1.3 Objective Function . . . . .                | 4        |
| 2.1.4 Optimization . . . . .                      | 5        |
| 2.2 Artificial Neural Networks . . . . .          | 5        |
| 2.2.1 Introduction . . . . .                      | 5        |
| 2.2.2 Neurons . . . . .                           | 6        |
| 2.3 Training Artificial Neural Networks . . . . . | 7        |
| 2.3.1 Forward Propagation . . . . .               | 7        |
| 2.3.2 Error Calculation . . . . .                 | 8        |
| 2.3.3 Backpropagation . . . . .                   | 8        |
| 2.3.4 Optimization . . . . .                      | 9        |
| 2.4 Feedback Controller . . . . .                 | 10       |
| 2.4.1 Components of a PD Controller . . . . .     | 10       |

|            |                                                                      |           |
|------------|----------------------------------------------------------------------|-----------|
| 2.4.2      | Application in Data Collection for Training . . . . .                | 11        |
| 2.5        | Literature Review . . . . .                                          | 11        |
| <b>III</b> | <b>LEARNING A FEEDBACK CONTROLLER WITH STATIC TARGETS . . . . .</b>  | <b>15</b> |
| 3.1        | Model . . . . .                                                      | 15        |
| 3.2        | Loss Function . . . . .                                              | 16        |
| 3.3        | Implementation Details . . . . .                                     | 17        |
| 3.4        | Experimental Setup . . . . .                                         | 20        |
| 3.4.1      | Simulation Environment . . . . .                                     | 20        |
| 3.4.2      | Dataset . . . . .                                                    | 20        |
| 3.4.3      | Performance Metrics . . . . .                                        | 20        |
| 3.4.3.1    | Robot Endpoint-Target Distance . . . . .                             | 20        |
| 3.4.3.2    | Torque Prediction Error . . . . .                                    | 21        |
| 3.5        | Results . . . . .                                                    | 22        |
| 3.5.1      | Offline Learning Performance . . . . .                               | 23        |
| 3.5.2      | Control Performance: Interpolation . . . . .                         | 24        |
| 3.5.3      | Control Performance: Extrapolation . . . . .                         | 26        |
| 3.5.4      | End-to-End Learning . . . . .                                        | 29        |
| 3.5.5      | Dynamic Targets . . . . .                                            | 31        |
| 3.6        | Discussion . . . . .                                                 | 33        |
| <b>IV</b>  | <b>LEARNING A FEEDBACK CONTROLLER WITH DYNAMIC TARGETS . . . . .</b> | <b>35</b> |
| 4.1        | Data . . . . .                                                       | 35        |
| 4.2        | Model and Loss Function . . . . .                                    | 38        |
| 4.3        | Performance Metrics . . . . .                                        | 40        |
| 4.3.1      | Tracking Error . . . . .                                             | 40        |
| 4.4        | Results . . . . .                                                    | 40        |
| 4.4.1      | Control Performance: Interpolation . . . . .                         | 41        |
| 4.4.2      | Control Performance: Extrapolation . . . . .                         | 44        |

|          |                                                    |           |
|----------|----------------------------------------------------|-----------|
| 4.5      | Learning Latent Representation of Target . . . . . | 48        |
| 4.6      | Discussion . . . . .                               | 48        |
| <b>V</b> | <b>CONCLUSIONS . . . . .</b>                       | <b>55</b> |
|          | <b>REFERENCES . . . . .</b>                        | <b>60</b> |



## LIST OF TABLES

|   |                                                                                                                                                                                                                                                                                                                                           |    |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1 | Architectural details of the models used, categorized by complexity level                                                                                                                                                                                                                                                                 | 19 |
| 2 | Offline learning performances on test data. Mean and standard deviation of Torque Prediction Errors are shown for the proposed model (DNFC), the baseline and the latent loss ablated version of DNFC . .                                                                                                                                 | 24 |
| 3 | Control Performance of the proposed model (DNFC), the baseline and the latent loss ablated version of DNFC: Mean and standard deviation of Endpoint-Target Distance and Torque Prediction Errors on the Interpolation and Extrapolation test sets . . . . .                                                                               | 31 |
| 4 | Image based control performance of the proposed model (DNFC): Mean and standard deviation of Endpoint-Target Distance and Torque Prediction Errors . . . . .                                                                                                                                                                              | 31 |
| 5 | Comparison of average Tracking Error between DNFC models and baseline across different complexity levels. . . . .                                                                                                                                                                                                                         | 32 |
| 6 | Control Performance of the proposed model (DNFC), the baseline, the latent loss ablated version of DNFC, and the ground truth controller trained & tested with dynamic targets: Mean and standard error of the mean (SEM) of Tracking Error, Torque Prediction Error, and Endpoint-Target Distance on the Interpolation test set. . . . . | 53 |
| 7 | Control Performance of the proposed model (DNFC), the baseline, the latent loss ablated version of DNFC, and the ground truth controller trained & tested with dynamic targets: Mean and standard error of the mean (SEM) of Tracking Error, Torque Prediction Error, and Endpoint-Target Distance on the Extrapolation test set. . . . . | 54 |

## LIST OF FIGURES

|    |                                                                                                                                                                                                                                                                                                       |    |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1  | The proposed Deep Neural Feedback Controller (DNFC) Architecture is illustrated. . . . .                                                                                                                                                                                                              | 16 |
| 2  | Distribution of the train and test sets' target points alongside the demonstration of the robot arm. . . . .                                                                                                                                                                                          | 21 |
| 3  | Mean of the endpoint-target distance on interpolation test sets with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model). . . . .                                 | 26 |
| 4  | Mean of the torque prediction error metric on interpolation test sets with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model). . . . .                           | 27 |
| 5  | Mean of the endpoint-target distance metric on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model). . . . .                       | 28 |
| 6  | Mean of the torque prediction error metric on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model). . . . .                        | 28 |
| 7  | Typical trajectories followed by the ground truth controller (purple), proposed DNFC model (blue) and the baseline model (orange) are shown. Each panel shows reaching performances of the models for a single target marked with (x). Each model is given 50 time steps to reach the target. . . . . | 30 |
| 8  | The end effector trajectories traced by each model when trying to reach the target moving in a circular path. Note that DNFC and the baseline model have been trained using only static targets. . . . .                                                                                              | 33 |
| 9  | Ten sample goal trajectories used in the training set. The start point of each trajectory is illustrated with a diamond, and the end point is marked with an X. . . . .                                                                                                                               | 36 |
| 10 | Histogram of the distances the targets traveled during each episode in the training dataset. The mean distance is roughly 7.5 cm. . . . .                                                                                                                                                             | 37 |
| 11 | Illustration of the radial and angular ranges for the interpolation and extrapolation test sets. . . . .                                                                                                                                                                                              | 38 |

|    |                                                                                                                                                                                                                                                                                                                                                 |    |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 12 | The effects of parameter $\beta$ on the latent loss ( $\mathcal{L}_u$ ). The term $e^{\beta(t/T-1)}$ is multiplied by the $l^2$ -norm of the discrepancy between the latent representation of the target and the robot state at timestep $t$ . As $\beta$ increases, $\mathcal{L}_s$ attains smaller values at the beginning of the episodes. . | 39 |
| 13 | Mean of the tracking error metric on the interpolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.) . . . . .                                      | 42 |
| 14 | Mean of the torque prediction error metric on the interpolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.) . . . . .                             | 43 |
| 15 | Mean of the endpoint-target distance on the interpolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.) . . . . .                                   | 44 |
| 16 | Mean of the tracking error metric on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.) . . . . .                                      | 45 |
| 17 | Mean of the endpoint-target distance on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.) . . . . .                                   | 46 |
| 18 | Mean of the torque prediction error metric on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.) . . . . .                             | 47 |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |    |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 19 | Five sample trajectories from the interpolation test set taken by the DNFC model (Blue), the baseline (Orange), and the ground truth controller (Purple) are shown, with the target trajectory in Gray. The start point of the target trajectory is illustrated by a diamond, and the end point by an X. The start point of the robot end effector is indicated by a black dot. The left and right columns correspond to the same target trajectories; however, the DNFC and baseline models were trained with different initializations. Both models have the same complexity level (1846 learnable parameters). . . . . | 49 |
| 20 | Five sample trajectories from the extrapolation test set taken by the DNFC model (Blue), the baseline (Orange), and the ground truth controller (Purple) are shown, with the target trajectory in Gray. The start point of the target trajectory is illustrated by a diamond, and the end point by an X. The start point of the robot end effector is indicated by a black dot. The left and right columns correspond to the same target trajectories; however, the DNFC and baseline models were trained with different initializations. Both models have the same complexity level (1846 learnable parameters). . . . . | 50 |
| 21 | Visualization of the latent representations of the target ( $\hat{s}$ ) and the robot state ( $s$ ) across the entire duration of three distinct episodes, sampled from the interpolation set. The robot state includes the angle ( $q$ ) and velocity ( $vel$ ) of two joints. Each row represents a different episode. In the left column, the latent representation of target and robot state are depicted throughout the entire episode, while the right column shows their differences ( $diff$ ) across each dimension. . . . .                                                                                     | 51 |

# CHAPTER I

## INTRODUCTION

Learning to map the relationship between world states and actions is fundamental to numerous robotics applications. This relationship, known as a policy, allows a robot to choose actions based on its current world state. Manually creating policies can be extremely difficult, prompting the use of machine learning techniques to aid in their development. This thesis focuses on contributing to a specific method of policy learning called Learning from Demonstration (LfD) [1, 2, 3, 4].

As briefly explained above, developing new robot tasks using traditional programming methods demands considerable coding expertise and significant time investment. Traditional approaches also necessitate that users explicitly define the sequence of actions or movements a robot must perform to complete a task. Motion planning methods [5, 6] aim to alleviate some of the challenges associated with traditional programming by removing the need to define every low-level action; however, it still requires users to specify higher-level actions, such as goal locations. These specifications are not adaptable to changes in the environment and often need redefinition or reprogramming when conditions change.

In learning from demonstration (LfD), policies are derived from examples or demonstrations provided by a teacher [7, 8]. These examples are sequences of state-action pairs captured during the demonstration of the desired behavior. LfD algorithms use this dataset to formulate a policy that replicates the demonstrated behavior. Unlike other methods, such as Reinforcement Learning [9], which develop policies from data gathered through exploration, LfD relies on imitating pre-recorded examples. Consequently, an LfD-derived policy is limited to the states and actions

observed during the demonstrations.

A critical challenge for LfD, particularly in complex and long-duration tasks, is covariate shift problem. [10] This phenomenon occurs when the distribution of the encountered data deviates from the training data distribution. When this happens, the learned model’s performance can degrade significantly. In the worst case, the robot may enter states far outside its training experience, leading to unrecoverable errors and potentially unsafe behavior. Akbulut et al. [11] proposed a potential mitigation strategy for this issue by advocating for the learning of an explicit correction policy that is activated when the anticipated transition state between primitives is not achieved.

Various types of learning—supervised, reinforcement, and unsupervised—have been employed to tackle a wide range of challenges in robot learning. Selecting the appropriate type is complex and depends on the specific requirements and constraints of the problem at hand. Learning from Demonstration can be considered a form of supervised learning because it involves acquiring new skills from external teachers through provided demonstrations. LfD is particularly advantageous when ideal behavior cannot be scripted, as in traditional robot programming, nor easily defined by optimizing a known reward function, as in reinforcement learning, but can instead be demonstrated.

Behavior Cloning (BC) is a proficient method for training robot controllers. It works by establishing a mapping from a comprehensive set of valid robot states to their corresponding actions [12]. This technique leverages observational learning, where the robot learns to mimic actions demonstrated in various states, effectively translating observed behavior into actionable control strategies [13]. While BC is occasionally used synonymously with LfD, the latter encompasses a broader concept, referring to the learning of movement primitives based on demonstrated data in a general sense. This can range from sophisticated learning techniques to simple approaches

like open-loop playback of pre-recorded trajectories [14, 15].

## 1.1 *Thesis Contribution*

In this study, we seek to advance the field of LfD by introducing a novel neural network architecture known as the Deep Neural Feedback Controller (DNFC). This architecture is designed to mirror the topology of a feedback error-based controller. The rationale behind this design is that incorporating a structural bias into the network can enhance the efficiency of learning from demonstration data, particularly when the data originates from a feedback control mechanism. This assumption is well-founded for LfD data derived from human demonstrations, where the generation of movements typically involves sensory feedback mechanisms. [16]

Overall, our study provides a novel behavior cloning model capable of automatically forming a latent space representation that facilitates the learning of nonlinear feedback controllers. This model demonstrates superior performance compared to controller-agnostic baselines on both interpolation and extrapolation tasks, considering both static and dynamic target scenarios.

## 1.2 *Thesis Outline*

This thesis is organized as follows. Chapter 2 provides the necessary background information to understand the subsequent chapters. It also reviews previous works in the literature related to LfD. Chapter 3 introduces our proposed model (DNFC) and presents its results on a two degrees-of-freedom *reaching* task, where targets remain static during the execution of episodes. These results are systematically compared with a controller-agnostic baseline and our proposed model’s ablated version. In Chapter 4, we present the results on a more complex *tracking* task, where goals move linearly during the execution of episodes. Finally, Chapter 5 concludes with the findings from our experiments and suggests future directions for this research.

## CHAPTER II

### BACKGROUND AND RELATED WORK

In this section, we formally define the problem of Learning from Demonstration (LfD), or imitation learning, using mathematical notations as follows:

#### ***2.1 Problem Definition***

Given a set of demonstrations provided by an expert, where each demonstration consists of a sequence of state-action pairs, the goal is to learn a policy that can replicate the expert's behavior.

##### **2.1.1 Demonstrations**

Let  $\mathcal{D} = \{(\tau_i)\}_{i=1}^N$  be a set of  $N$  demonstrations, where each demonstration  $\tau_i$  is a trajectory defined as  $\tau_i = \{(s_{i,t}, a_{i,t})\}_{t=1}^{T_i}$ . Here,  $s_{i,t} \in \mathcal{S}$  denotes the state and  $a_{i,t} \in \mathcal{A}$  denotes the action at time step  $t$ , and  $T_i$  is the length of the  $i$ -th trajectory.

##### **2.1.2 Policy Learning**

The objective is to learn a policy  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  such that the policy, when deployed in the environment, behaves similarly to the expert demonstrated behavior. The learned policy  $\pi$  should ideally minimize some form of error or discrepancy between the actions chosen by the policy and those demonstrated by the expert.

##### **2.1.3 Objective Function**

To quantify the similarity between the expert's behavior and the learned policy, an objective function  $J$  can be defined. Typically, this is a measure of the discrepancy between the actions taken by the learned policy  $\pi$  and the actions demonstrated in

the trajectories  $\mathcal{D}$ . For example, a common choice is the cumulative loss function over all demonstrations:

$$J(\pi) = \sum_{i=1}^N \sum_{t=1}^{T_i} \mathcal{L}(\pi(s_{i,t}), a_{i,t}) \quad (1)$$

where  $\mathcal{L}$  is a loss function that measures the deviation between the action  $\pi(s_{i,t})$  produced by the policy and the expert’s action  $a_{i,t}$ .

#### 2.1.4 Optimization

The policy  $\pi$  is learned by solving the optimization problem:

$$\pi^* = \arg \min_{\pi} J(\pi) \quad (2)$$

where  $\pi^*$  is the policy that minimizes the objective function  $J$  over the space of possible policies.

In summary, the problem of learning from demonstration in robotics involves extracting a policy from a set of expert demonstrations such that the policy closely approximates the expert’s behavior in the given environment. Next, we briefly introduce artificial neural networks as they are commonly used in modern LfD papers.

## 2.2 Artificial Neural Networks

To find a policy that minimizes the objective function mentioned in equation 1, many approaches have been tried out in the literature. However, currently the most effective one is using Artificial Neural Networks (ANNs). In this section, we briefly explain the fundamental principals of ANNs.

### 2.2.1 Introduction

Artificial Neural Networks (ANNs) are computational models inspired by the human brain’s neural architecture [17, 18]. They consist of interconnected layers of nodes or neurons, where each connection has an associated weight. These networks can learn

and make decisions by adjusting these weights based on input data, enabling them to recognize patterns and make predictions. Deep Learning, a subset of machine learning, leverages ANNs with many hidden layers, known as deep neural networks, to perform complex tasks. By utilizing vast amounts of data and computational power, deep learning algorithms have achieved remarkable success in areas such as image [19] and speech recognition [20], natural language processing [21, 22], and autonomous systems [23]. The capability of deep learning to automatically extract and learn intricate features from raw data distinguishes it from traditional machine learning techniques, making it a pivotal technology in the advancement of artificial intelligence.

### 2.2.2 Neurons

In Artificial Neural Networks (ANNs), neurons are the fundamental processing units that mimic the behavior of biological neurons. Each neuron receives one or more input signals, which can be represented as numerical values. These inputs are typically multiplied by corresponding weights, which are parameters that determine the strength and significance of each input. The weighted inputs are then summed together to produce a single value.

This summed value is passed through an activation function, which introduces non-linearity into the model. The activation function determines whether a neuron should be activated or not based on the input it receives. Common activation functions include the sigmoid function, hyperbolic tangent ( $\tanh$ ), and rectified linear unit (ReLU). The output of the activation function is the neuron's output, which can then be passed as input to the neurons in the subsequent layer of the network.

Mathematically, if we denote the input signals as  $x_1, x_2, \dots, x_n$ , the corresponding weights as  $w_1, w_2, \dots, w_n$ , and the bias term as  $b$ , the neuron's output  $y$  can be expressed as:

$$y = f \left( \sum_{i=1}^n w_i x_i + b \right) \quad (3)$$

Here,  $f$  represents the activation function. This process of taking inputs, applying weights, summing them, adding a bias, and passing through an activation function allows neurons to transform inputs into meaningful outputs, facilitating the learning and decision-making processes within the ANN.

## 2.3 *Training Artificial Neural Networks*

Training an artificial neural network (ANN) involves several key steps: forward propagation, error calculation, backpropagation, and optimization. Each step plays a crucial role in enabling the network to learn from data and improve its performance over time. Here's a detailed explanation of each step:

### 2.3.1 **Forward Propagation**

Forward propagation is the process through which the input data passes through the neural network to produce an output. This involves generating weights and biases, which are parameters of the network that are adjusted during training. The steps are as follows:

- **Input Layer:** The input features are fed into the input layer of the network.
- **Hidden Layers:** The input data is then passed through one or more hidden layers. Each neuron in a hidden layer performs a weighted sum of the inputs from the previous layer, adds a bias, and applies an activation function (e.g., ReLU, Sigmoid) to introduce non-linearity, as denoted in equation 3.
- **Output Layer:** The transformed data reaches the output layer, which produces the final predictions of the network.

Mathematically, for each neuron  $j$  in layer  $l$ , the output  $a_j^l$  can be expressed as:

$$a_j^l = f \left( \sum_i w_{ij}^{l-1} a_i^{l-1} + b_j^l \right) \quad (4)$$

where  $w_{ij}^{l-1}$  are the weights,  $a_i^{l-1}$  are the activations from the previous layer,  $b_j^l$  are the biases, and  $f$  is the activation function.

### 2.3.2 Error Calculation

Error calculation measures how well the network's predictions match the actual target values. This is done using a loss function, which quantifies the difference between the predicted and actual values. Common loss functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks.

For example, in a regression task, which we frequently encounter in this thesis, the Mean Squared Error (MSE) is calculated as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (5)$$

where  $y_i$  are the actual target values,  $\hat{y}_i$  are the predicted values, and  $n$  is the number of samples.

### 2.3.3 Backpropagation

Backpropagation is the process of passing the error back through the network to update the weights and biases. This is done in three steps:

1. **Compute Gradients:** Calculate the gradient of the loss function with respect to each weight and bias in the network. This involves applying the chain rule of calculus to propagate the error from the output layer back through each hidden layer to the input layer.
2. **Gradient Descent:** Use the gradients to update the weights and biases in the direction that reduces the loss. The magnitude of the update is controlled by the learning rate, a hyperparameter that needs to be set before training.

Mathematically, the weight update for a weight  $w$  can be expressed as:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w} \quad (6)$$

where  $\eta$  is the learning rate and  $\frac{\partial L}{\partial w}$  is the gradient of the loss function  $L$  with respect to the weight  $w$ .

### 2.3.4 Optimization

Optimization involves adjusting the weights and biases to minimize the loss function. The most common optimization algorithm is Gradient Descent, which iteratively updates the parameters to find the minimum of the loss function. Variants of Gradient Descent, such as Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent, and advanced optimizers like Adam, RMSprop, and Adagrad, introduce different strategies for more efficient and effective training.

- **Stochastic Gradient Descent (SGD):** Updates the weights for each training example individually, which can lead to faster convergence but more noisy updates.
- **Mini-batch Gradient Descent:** Updates the weights based on a small batch of training examples, balancing between the efficiency of batch gradient descent and the noise reduction of SGD.
- **Adam Optimizer:** Combines the advantages of both RMSprop and SGD with momentum, adjusting the learning rate based on the first and second moments of the gradients.

Using these techniques, the neural network iteratively improves its performance on the training data, gradually learning to make better predictions.

In summary, training an artificial neural network involves forward propagation to generate predictions, error calculation to measure performance, backpropagation to

distribute the error, and optimization to update the weights and biases, all working together to enable the network to learn from data.

## 2.4 *Feedback Controller*

A Proportional-Derivative (PD) controller [24] is a fundamental control mechanism commonly employed in robotics and automation to regulate the behavior of dynamic systems. The PD controller combines two control strategies: proportional control and derivative control, to compute the control input necessary to achieve desired system performance.

### 2.4.1 Components of a PD Controller

1. **Proportional Control (P):** This component generates a control signal proportional to the error between the desired setpoint and the current state of the system. The proportional term helps to reduce the overall error magnitude. Mathematically, it is represented as:

$$P(t) = K_p \cdot e(t)$$

where  $K_p$  is the proportional gain and  $e(t)$  is the error at time  $t$ .

2. **Derivative Control (D):** This component produces a control signal proportional to the rate of change of the error. The derivative term helps to predict future errors and apply corrective actions accordingly, thus improving the system's stability and response time. It is expressed as:

$$D(t) = K_d \cdot \frac{de(t)}{dt}$$

where  $K_d$  is the derivative gain and  $\frac{de(t)}{dt}$  is the derivative of the error with respect to time.

The overall control input  $u(t)$  applied to the system is the sum of these two components:

$$u(t) = P(t) + D(t) = K_p \cdot e(t) + K_d \cdot \frac{de(t)}{dt}$$

#### 2.4.2 Application in Data Collection for Training

In this research, a feedback PD controller was utilized to collect data necessary for training our model.

The system parameters, including mass, length, and inertia of the robotic arm segments, were defined, and the proportional and derivative gains ( $K_p$  and  $K_d$ ) were set. Inverse kinematics was used to determine the joint angles required to reach the target position by solving geometric equations. A trajectory generation function using polynomial interpolation was employed to smoothly transition the robotic arm from its current state to the target state. At each time step, the PD controller computed the control input (torque) based on the current and desired joint angles and their rates of change, with the resulting state transitions and control inputs being recorded and stored for subsequent model training.

The implementation of the PD controller facilitated the generation of accurate and consistent trajectories, providing a rich dataset for model training. The controlled behavior of the robotic arm ensured that the collected data was representative of the system’s dynamics, thus enhancing the robustness and generalization capability of the trained model.

### 2.5 Literature Review

The field of learning from demonstration (LfD) has expanded significantly in recent years. Early LfD research primarily focused on kinesthetic teaching, where a human physically guides a robot through a task to provide demonstrations [25, 26, 27]. While intuitive, this approach is limited to tasks that can be easily performed manually. Teleoperation addressed this limitation by enabling demonstrations through remotely

controlled interfaces [28, 29, 30]. More recently, Learning by Observation has gained traction, where robots learn from observing human demonstrations through visual perception [31, 32, 33]. This approach broadens the range of learnable tasks and eliminates the need for physical interaction with the robot.

Various representations have been explored to encode demonstrations and facilitate skill learning. Dynamic Movement Primitives (DMPs) provide a compact and flexible representation for encoding trajectories, enabling robots to adapt to variations in task conditions [34, 35]. Gaussian Mixture Models (GMMs) offer a probabilistic framework for representing complex demonstrations, capturing uncertainties and variations in the data [36, 37]. Hidden Markov Models (HMMs) have been utilized to model sequential demonstrations, capturing the underlying structure and transitions between different phases of a task [38].

The advent of deep learning has further revolutionized LfD by enabling the handling of high-dimensional sensory inputs and complex control policies. Deep neural networks, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have been used to learn end-to-end policies directly from raw sensory data [39, 40, 41, 42, 43].

LfD has been successfully applied to a wide range of robotic tasks, including manipulation, assembly, and locomotion. In manipulation, LfD has enabled robots to learn grasping skills [44], object handover [45], and pick and place [46]. In assembly, LfD has been employed to teach robots complex assembly sequences [47] and to enable them to collaborate with humans [48, 49]. In locomotion, LfD has facilitated the learning of walking gaits [50, 51] and navigation strategies [52, 53, 54].

Behavioral Cloning (BC), a subcategory of Learning from Demonstration (LfD), is a fundamental approach in robot learning. It involves teaching a robot to perform tasks by directly mimicking the actions of a human demonstrator. The robot learns to map perceived states (such as sensor readings or images) to specific actions (such as

motor commands). This concept isn’t new. In the 1980s, researchers like Pomerleau [13] pioneered autonomous driving by training neural networks to imitate human drivers.

Recent research strives to enhance BC performance in various aspects. For instance, [55] proposed DAgger, an iterative algorithm that enhances BC by incorporating states encountered during policy execution into the training dataset. This approach was effectively used by Seita et al. [56] for a sequential fabric smoothing task. By continuously updating the dataset with real-world experiences, DAgger allows the robot to learn from its mistakes and improve over time. However, it’s worth noting that DAgger requires ongoing expert supervision to provide feedback and corrections, which can be resource-intensive.

Furthermore, Nair et al. [57] demonstrated that robots can learn to manipulate ropes by analyzing visual data from demonstrations in a self-supervised manner. This approach involves the robot learning an inverse dynamics model, essentially understanding how to achieve desired rope configurations from visual input. This work highlights the potential of self-supervised learning in expanding the scope of tasks that robots can learn through demonstration, particularly in complex manipulation scenarios.

In a study closely related to our work, [58] demonstrated that neural network controllers often struggle to stabilize a system, even when they are trained to a high degree of accuracy. The authors propose two novel architectures,  $\lambda$ -*QRnet* and *u-QRnet*, to improve the stability and reliability of these controllers. These architectures are based on *QRnet* [59], which incorporates an *LQR* controller. The addition of the *LQR* component enhances the stability of the controller.

In contrast to prior studies, we concentrate on encoding a structural bias within the learning architecture. This strategy enhances performance while simultaneously reducing computational demands. To our knowledge, this is the first study to propose

a deep learning architecture specifically designed to synthesize a nonlinear feedback controller through learning from demonstration.



## CHAPTER III

# LEARNING A FEEDBACK CONTROLLER WITH STATIC TARGETS

### 3.1 *Model*

We introduce a Behavior Cloning (BC) model designed to map a specified task context or target objective into control commands for a robot, based on the robot’s current state. The primary outputs of this model are the torques for robots equipped with revolute joints. The task context may be represented by an image frame from a camera or a vector describing the context or target. As we are developing a feedback controller, it is essential that the robot’s state be provided as an input to our proposed architecture.

The two input channels to the model are processed as follows (refer to Figure 1). Initially, an encoder module, which is a deep neural network, receives a target representation, such as an image or a feature vector. This module maps the input to a latent space representation that matches the dimensionality of the robot’s state, denoted as  $\hat{s}$ . Following this, the actual robot state, represented as  $s$ , is subtracted from  $\hat{s}$ , resulting in what we term the *latent error*. This error is then fed into the output module, a non-linear multi-layer perceptron (MLP), which is trained to convert this error into the appropriate torques to adjust the robot’s motion.

In this chapter, we systematically evaluate the capability of our model by employing Cartesian input to define the context for the target-reaching task. Furthermore, we demonstrate that our model is capable of processing image inputs and surpasses the performance of the standard Behavior Cloning (BC) baseline model under identical conditions. Detailed descriptions of the functioning of the proposed model are

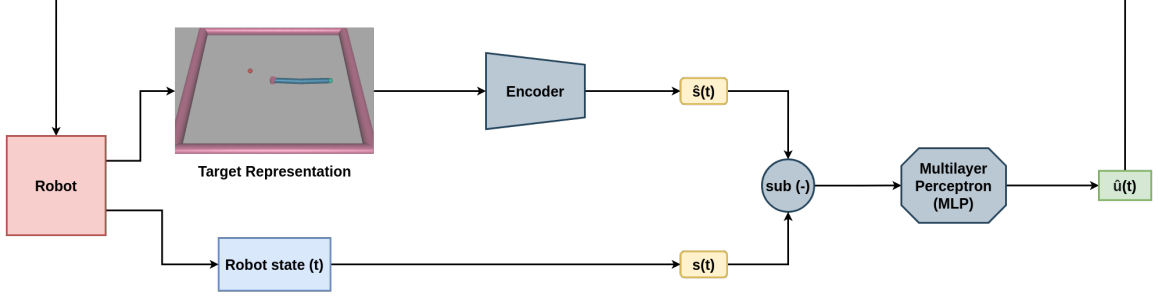


**Figure 1:** The proposed Deep Neural Feedback Controller (DNFC) Architecture is illustrated.

provided in the subsequent subsections.

### 3.2 Loss Function

To develop a neural architecture that functions as a non-linear feedback controller, the training of our model's weights utilizes a loss function that extends beyond the simple discrepancy between the predicted torques and the actual torques from the demonstration data. We also include an auxiliary latent loss term designed to encourage the latent representation,  $\hat{s}(t)$ , to become the target for the system state  $s(t)$ . This addition is grounded in the principle that the system state should gradually converge to the target state through the intervention of the learned feedback controller. We will now outline the individual loss components that comprise the comprehensive loss function used for training. The primary component of the loss function in our model is based on the error in torque prediction, expressed by the following equation:

$$\mathcal{L}_u = \frac{1}{2} \sum_{t=0}^{T-1} \|u(t) - \hat{u}(t)\|^2 \quad (7)$$

Here,  $\hat{u}(t)$  and  $u(t)$  denote the predicted and actual joint torques, respectively, at each discrete time step  $t$ . The term  $T$  designates the final time step of the robotic action, which is set at 50 for the purposes of this chapter. In the context of the 2-DOF manipulator employed in this study, both  $\hat{u}(t)$  and  $u(t)$  are two-dimensional vectors.

In our model, we also aim for the robot state vector  $s(t)$  to closely mirror the

latent representation,  $\hat{s}(t)$ , especially as the time  $t$  nears the end time  $T$ . To achieve this objective, we introduce a *latent* loss function that incorporates an exponential weighting factor, defined as follows:

$$\mathcal{L}_s = \sum_{t=0}^{T-1} \alpha e^{\beta(t/T-1)} \|s(t) - \hat{s}(t)\|^2 \quad (8)$$

In this equation,  $\alpha$  and  $\beta$  are constants. The design of the weighting factor ensures that the influence of the latent loss is minimal at the beginning of the sequence (when  $t$  is close to 0) and increases significantly as  $t$  approaches  $T$ , thereby emphasizing the accuracy of the latent state representation particularly towards the end of the episode.

The total loss used in training our model is calculated as the sum of the individual losses, as outlined in Equation 7 for the control discrepancy and Equation 8 for the latent representation alignment:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_u + \mathcal{L}_s \quad (9)$$

This approach ensures that the importance of aligning the system state  $s(t)$  with the latent representation  $\hat{s}(t)$  increases as the episode progresses, reflecting the growing criticality of reaching the target state. In typical target tracking scenarios,  $\hat{s}(t)$  may change dynamically as the target moves. However, in our current experiments, the target remains stationary throughout the episode, making the input to the encoder and consequently the latent representation  $\hat{s}(t)$  constant. Thus, for the purposes of this chapter, we could simplify our notation by dropping the time dependence from  $\hat{s}(t)$  and refer to it as  $\hat{s}$ . However, we retain the time-dependent notation to maintain generality in the model description.

### 3.3 Implementation Details

Define  $C_k$  and  $L_k$  as a convolutional layer ( $C$ ) and a linear layer ( $L$ ), each containing  $k$  filters or units, respectively. For instance, the notation  $C_{32} - L_{128}$  represents a

structure where a convolutional layer equipped with 32 filters precedes a linear layer that comprises 128 units.

When using Cartesian coordinates of the target points as the target representation, we employ a multilayer perceptron (MLP) consisting of three layers for the encoder. Each layer is a linear layer that includes a bias term. The activation function between the linear layers is the ReLU (Rectified Linear Unit) [60], while the output layer does not incorporate any activation function. For instance, in a model that contains 570 learnable parameters, the architecture of the encoder MLP is described as  $L_2 - L_{10} - L_{10} - L_4$ . Here, the first and the last linear layers serve as the input and output layers, respectively. It is important to note that the function  $s(t)$  must always be four-dimensional since the state space is 4-dimensional.

When employing images as the target representation, we utilize a compact convolutional neural network (CNN) for the encoder. The structure of this CNN encoder is denoted as  $C_8 - C_{32} - C_{32} - C_{32} - L_{128} - L_4$ . The ReLU activation function is applied between each layer to introduce non-linearity. This implementation of image target representations showcases the versatility of our method, demonstrating its capacity to effectively generalize and function with targets depicted both as images and Cartesian coordinates.

The MLP positioned after subtracting the latent representations, specifically,  $\hat{s}(t) - s(t)$ , comprises three layers mirroring the characteristics of the encoder MLP. This MLP is tasked with generating the desired torque values for each joint  $i$  at each timestep  $t$ , represented as  $\hat{u}_i(t)$ . These values are determined based on the computed error at each respective timestep. For instance, in a model with 570 learnable parameters, the architecture of this MLP is structured as  $L_4 - L_{16} - L_{16} - L_2$ . Here, the final layer,  $L_2$ , acts as the output layer and is designed to match the dimensionality of the action space, which is 2-dimensional in this environment. For the activation function of the output layer, the hyperbolic tangent is employed in both our proposed

**Table 1:** Architectural details of the models used, categorized by complexity level

| DNFC / Baseline                 | Encoder                                                            | MLP                             |
|---------------------------------|--------------------------------------------------------------------|---------------------------------|
| Low Complexity (270 params.)    | $L_2 - L_6 - L_6 - L_4$                                            | $L_4 - L_{10} - L_{10} - L_2$   |
|                                 | Baseline: $L_6 - L_6 - L_6 - L_8 - L_8 - L_5 - L_2$                |                                 |
| Medium Complexity (570 params.) | $L_2 - L_{10} - L_{10} - L_4$                                      | $L_4 - L_{16} - L_{16} - L_2$   |
|                                 | Baseline: $L_6 - L_{10} - L_{10} - L_{11} - L_{11} - L_{10} - L_2$ |                                 |
| High Complexity (1846 params.)  | $L_2 - L_{20} - L_{20} - L_4$                                      | $L_4 - L_{32} - L_{32} - L_2$   |
|                                 | Baseline: $L_6 - L_{10} - L_{24} - L_{24} - L_{24} - L_{10} - L_2$ |                                 |
| DNFC - Image                    | $C_8 - C_{32} - C_{32} - C_{32} - L_{128} - L_4$                   | $L_4 - L_{128} - L_{128} - L_2$ |

model and the baseline model. This choice is due to the torque values in the Reacher environment being constrained within the  $[-1, 1]$  range.

As detailed in section 3.2, to ensure that  $s(t)$  converges towards  $\hat{s}(t)$  by the end of the episodes, a supplementary loss function is implemented, which is outlined in Equation 8. From empirical research, the coefficients  $\alpha$  and  $\beta$  have been set to values of 5 and 10, respectively. These values have been selected to optimize the performance of the loss function in guiding the convergence effectively.

The baseline model, when using Cartesian coordinates as target representations, employs a basic MLP consisting of 6 layers. The input for this model includes the Cartesian coordinate of the target, which is concatenated with the robot’s state at timestep  $t$ , forming a 6-dimensional vector. The depth of the baseline’s layers corresponds to that of the deepest stream of the DNFC model. For example, in a model that possesses approximately 570 learnable parameters, the MLP’s architecture is defined as  $L_6 - L_{10} - L_{10} - L_{11} - L_{11} - L_{10} - L_2$ . Similar to the DNFC model, the ReLU activation function is utilized between the layers to ensure non-linearity and effective learning dynamics.

All the models discussed in this study were trained utilizing the Adam optimizer [61], set at a learning rate of  $3 \times 10^{-4}$ . Each model was trained for 1200 epochs. Detailed specifications regarding the architecture of each model are systematically outlined in Table 1.

### **3.4 Experimental Setup**

#### **3.4.1 Simulation Environment**

All experiments in this study were conducted within the Reacher environment from Gymnasium [62], which utilizes the MuJoCo physics engine [63]. The Reacher setup features a two-jointed robotic arm, with each segment of the arm measuring 10 cm, cumulatively giving the arm a total length of 20 cm. The primary objective in this environment is to maneuver the robot’s end effector to a specified target point. Notably, the Reacher environment does not impose joint limits, allowing the robotic arm to reach any target point within a circle with a radius of 20 cm.

#### **3.4.2 Dataset**

The training dataset comprises 466 trajectories, with the target points systematically selected from a grid organized within the polar coordinate system, as depicted in Figure 2. The generation of these trajectories involved the use of inverse kinematics and a PD controller, adapted from [64]. The radial distances of the target points extend from 1 cm to 15 cm, increasing in 1 cm increments. Furthermore, the polar angles were selected at discrete intervals ranging from 15 to 165 degrees, with each step measuring 5 degrees. This structured approach ensures a comprehensive coverage of possible target positions within the specified operational space of the robot arm.

#### **3.4.3 Performance Metrics**

To assess the effectiveness of both our model and the baseline model on the test datasets, we employed two specific metrics, described as follows:

##### *3.4.3.1 Robot Endpoint-Target Distance*

This metric calculates the Euclidean distance between the endpoint of the Reacher and the target point at the conclusion of each episode. Let  $(x', y')$  represent the position of the Reacher’s endpoint at an episode’s end, and  $(x, y)$  denote the target

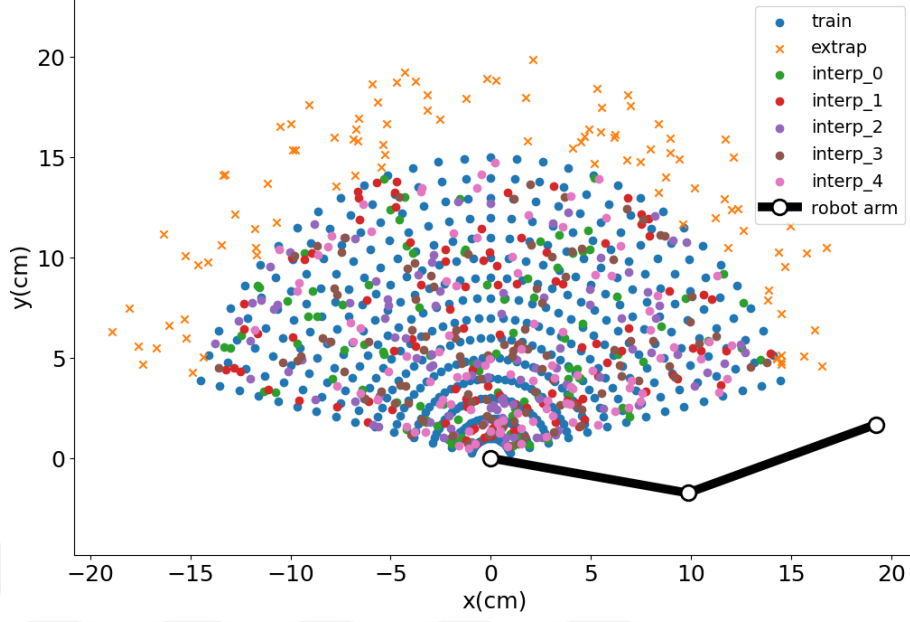


**Figure 2:** Distribution of the train and test sets’ target points alongside the demonstration of the robot arm.

point’s position. The Endpoint-Target Distance ( $d$ ) is defined by the equation:

$$d = \sqrt{(x' - x)^2 + (y' - y)^2} \quad (10)$$

In these experiments, an episode is terminated when the Endpoint-Target Distance falls below 1 cm. If this condition is not met, the episode continues up to a maximum of 50 steps, as established in our experimental protocol.

#### 3.4.3.2 Torque Prediction Error

To evaluate the accuracy with which the proposed model and the baseline predict the joint torques given the robot state, we use the Torque Prediction Error as a metric. This error measures the difference between the predicted torque values and the ground truth torque values, i.e., the torques calculated by the PD controller, needed to reach the target point. From a machine learning perspective, understanding the magnitude of this error is crucial as it directly impacts the performance of the online feedback control system.

Let  $T$  represent the total number of timesteps in a trajectory,  $u_i(t)$  the torque value calculated by the controller for the  $i$ -th joint at timestep  $t$ , and  $\hat{u}_i(t)$  the predicted torque value by the models for the  $i$ -th joint at the same timestep. The total average error over the entire trajectory is then defined by the following equation:

$$E = \frac{1}{2T} \sum_{t=0}^{T-1} \sum_{i=1}^2 |u_i(t) - \hat{u}_i(t)| \quad (11)$$

This equation,  $E$ , captures the average error in the predicted joint torques across the entire trajectory.

### 3.5 Results

To ensure statistically robust results, each model was trained five times, with each training session employing a different initialization. This approach allowed us to evaluate the performance of both the proposed model and the baseline under varying conditions. Additionally, to demonstrate the impact of the ‘latent loss’ component, which is outlined in Equation 8, ablation experiments were conducted. In these experiments, the latent loss component was omitted from the total loss calculation of our model, providing insights into its contribution to the model’s enhanced performance.

Given that both the proposed model and the baseline are designed to function as controllers, the ultimate test of their utility is their performance in real-time control scenarios. This involves online testing where the models control the robot directly. However, to first assess how well the models learn from the data presented to them, we conducted offline testing. This phase is crucial as it offers preliminary insights into the models’ learning capabilities and performance variations without the complexities of real-world control.

Following the offline evaluation, we then explored the *control performance* of both the proposed model and the baseline. This was done by assessing their ability to generate reaching movements towards both interpolation and extrapolation targets.

This phase of testing is vital for understanding how well the models can handle new, unseen scenarios. Importantly, during these assessments, we carefully controlled the complexity of both the proposed model and the baseline to ensure a fair and precise evaluation of their performances.

### 3.5.1 Offline Learning Performance

In this phase of the experiment, both the baseline and the proposed model were tasked with predicting torques based on provided state and target information, without directly controlling the robot. The actual control of the robot was handled by the ground truth controller, which was responsible for generating the original demonstration data. This setup ensures that the models’ predictions can be evaluated independently of their influence on the robot’s movements.

The models were trained using data derived from reaching executions toward the target points illustrated in blue dots in Figure 2, totaling 466 data points. To assess their predictive accuracy, we obtained their torque prediction errors on trajectories realized while reaching for 500 randomly selected targets (represented by colored dots, excluding the yellow ones in the figure). These target points actually form the interpolation test set, which will be investigated in details later.

To account for the inherent stochasticity in the training process, each model underwent five separate training sessions with different initializations. The results from these sessions were aggregated to provide a robust analysis of the models’ performance, which is detailed in Table 2. This approach not only underscores the repeatability of the training but also enhances the reliability of the conclusions drawn regarding each model’s offline learning capabilities.

As demonstrated by the data in Table 2, both the proposed model and the baseline exhibit similar performance levels, with our model slightly outperforming the baseline at the first two lower complexity levels. However, at the highest complexity level, our

**Table 2:** Offline learning performances on test data. Mean and standard deviation of Torque Prediction Errors are shown for the proposed model (DNFC), the baseline and the latent loss ablated version of DNFC

| Method                     | Offline Torque Prediction Error |
|----------------------------|---------------------------------|
| <b>Approx. 270 Params</b>  |                                 |
| DNFC                       | 0.0174 ( $\pm 0.0031$ )         |
| Baseline                   | 0.0196 ( $\pm 0.0042$ )         |
| DNFC <sub>ablation</sub>   | 0.0174 ( $\pm 0.0033$ )         |
| <b>Approx. 570 Params</b>  |                                 |
| DNFC                       | 0.0108 ( $\pm 0.0016$ )         |
| Baseline                   | 0.0127 ( $\pm 0.0020$ )         |
| DNFC <sub>ablation</sub>   | 0.0135 ( $\pm 0.0013$ )         |
| <b>Approx. 1846 Params</b> |                                 |
| DNFC                       | 0.0069 ( $\pm 0.0006$ )         |
| Baseline                   | 0.0061 ( $\pm 0.0005$ )         |
| DNFC <sub>ablation</sub>   | 0.0075 ( $\pm 0.0010$ )         |

model slightly underperforms compared to the baseline. This performance pattern suggests that both models are proficient at learning the dataset in the offline setting to a comparable extent.

Additionally, the version of our model with the ‘latent loss’ component ablated exhibited a higher error rate than the original DNFC model. This finding highlights the beneficial impact of the proposed latent loss on offline learning, confirming its value in enhancing model accuracy.

The next critical phase of our study, detailed in the following subsections, addresses the essential question of how the control performance between the DNFC and the baseline differs. This will provide deeper insights into the practical effectiveness of the models when applied in real-world scenarios.

### 3.5.2 Control Performance: Interpolation

In the interpolation test setting, we evaluated the control performance of our models using five separate test sets, each containing 100 target points. This resulted in results

structured as  $(5 \times 5 \times 100)$  tensors. In this data structure:

- The first dimension reflects the five different models, each trained with a unique random seed to ensure diversity in initialization.
- The second dimension represents the five distinct test sets to which each model was subjected.
- The third dimension consists of 100 individual episodes within each test set.

For comprehensive statistical analysis, we first averaged the results across the 100 episodes of each test set to derive an average performance metric for each set. Following this, we calculated and reported the mean, standard deviation (STD), and standard error of the mean (SEM) across the resulting  $(5 \times 5)$  matrix. This process effectively condensed the data to 25 distinct samples for analysis ( $N = 25$ ). These statistics provide a robust measure of performance consistency and variability, offering a clear view of how each model performs under interpolation conditions. This approach not only helps in understanding the generalizability of the models within the trained range but also ensures that the performance metrics are reliable and statistically validated.

In the interpolation test sets, the evaluation of the endpoint-target distance metric showed that our method (DNFC) consistently achieved lower mean distances compared to the baseline across all three complexity levels, as illustrated in Figure 3. A Welch’s independent samples t-test confirmed that the differences in means are statistically significant ( $p < 1e - 04$ ) for all model complexity levels, highlighting a substantial distinction in performance. Similarly, for the torque prediction error metric, our method demonstrated superior performance with statistically significant difference ( $p < 1e - 04$ ) across all complexity levels, as depicted in Figure 4. The performance distinctions are further evidenced by the green bars in both Figure 3 and Figure 4, which represent the ‘latent loss’ ablated version of our model. These

results indicate that the proposed latent loss significantly contributes to the enhanced performance of our model.

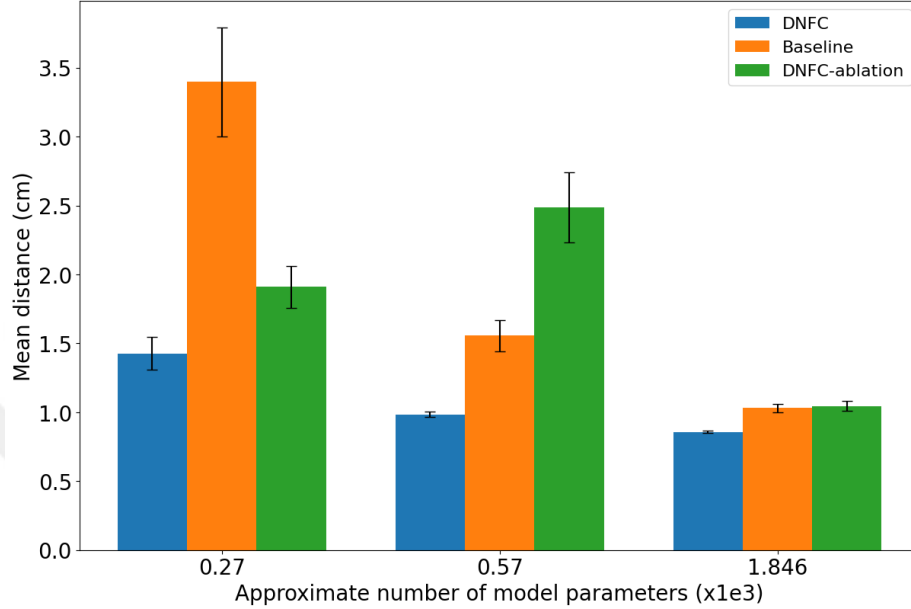


**Figure 3:** Mean of the endpoint-target distance on interpolation test sets with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model).

### 3.5.3 Control Performance: Extrapolation

Extrapolation presents a significant challenge for supervised learning systems, particularly for Behavior Cloning (BC), as it involves evaluating system performance on data that it has not previously encountered. This challenge becomes more pronounced when the data lies outside the convex hull of the training data. In response to this issue, the extrapolation data was sampled from points outside the convex hull of the training data, as indicated by the orange points in Figure 2.

For the extrapolation test setting, the statistical measures of mean, standard deviation (STD), and standard error of the mean (SEM) were calculated in a manner akin to the interpolation setting. However, each test set in this setting comprised only 20 episodes, as opposed to 100 in the interpolation tests. Consequently, the

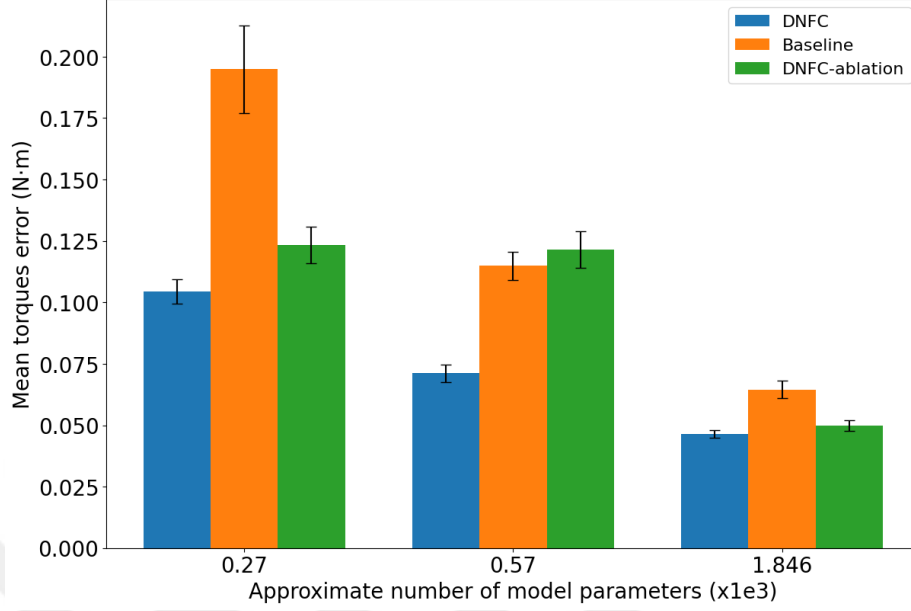


**Figure 4:** Mean of the torque prediction error metric on interpolation test sets with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model).

total number of samples ( $N$ ) for this section remains 25.

The means and standard errors of the mean (SEMs) of the model performances are visually presented in Figure 5 and Figure 6 for the endpoint-target distance and torque prediction error metrics, respectively. As illustrated in these figures, our method surpasses the baseline in both metrics under the extrapolation setting as well. Regarding the endpoint-target distance, the difference in means is statistically significant ( $p < 0.05$ ) across all three cases. In terms of torque prediction error, except for the models with approximately 570 parameters, the differences in means are statistically significant ( $p < 0.001$ ).

Additionally, Figures 5 and Figure 6 include the performance of the ‘latent loss’ ablated versions of our model (represented by green bars). These comparisons distinctly demonstrate that the proposed latent loss substantially enhances the performance of our model, affirming its significant contribution to the observed superiority in model efficacy.

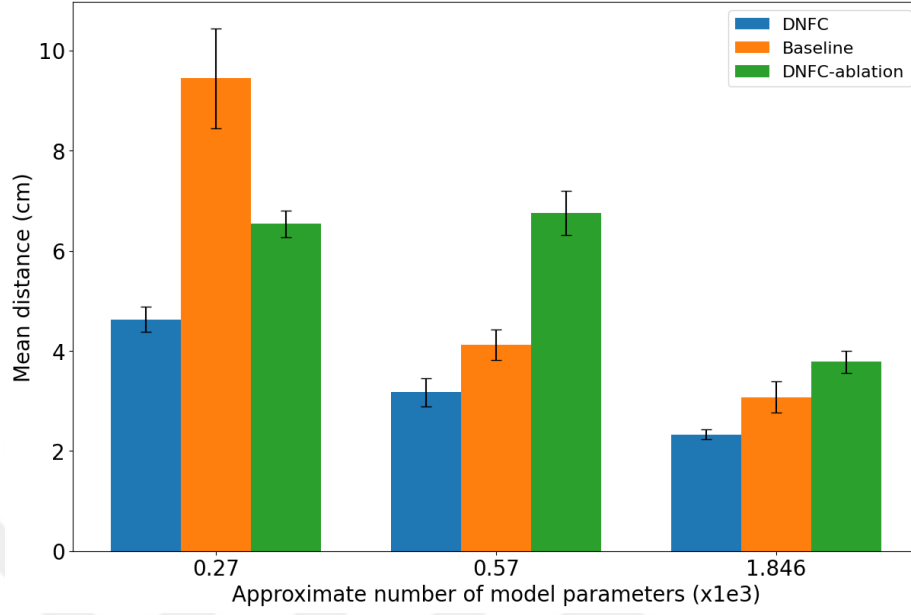


**Figure 5:** Mean of the endpoint-target distance metric on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model).

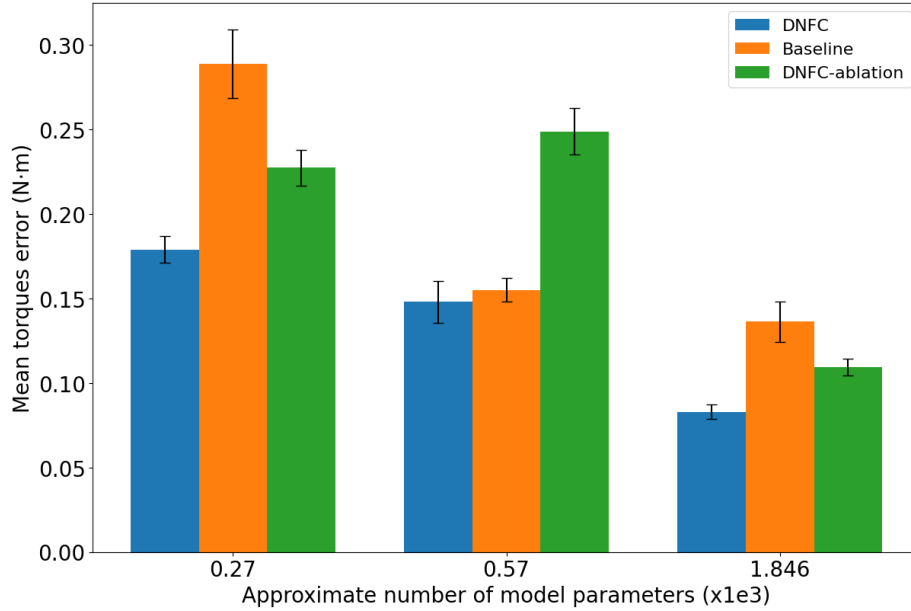


**Figure 6:** Mean of the torque prediction error metric on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model).

Overall, the results clearly indicate that at lower complexity levels, the performance differential between DNFC and the baseline is considerably more pronounced. Additionally, in the online control tasks, the statistical significance of the performance differences is remarkably stronger than in the offline dataset. This suggests that the superiority of our DNFC model is not merely due to its enhanced ability to learn from data but is primarily derived from the error-feedback-like structural bias inherent in its design.

Furthermore, the ablation experiments, which involved disabling the novel loss component proposed in Equation 9, reveal that this loss, in conjunction with the structural bias, plays a crucial role in developing a high-performance controller. These findings underscore the effectiveness of the integrated approach in improving the functionality of the model in control tasks.

The mean and standard deviation (STD) values for both the interpolation and extrapolation settings are reported in Table 3. Additionally, typical trajectories followed by the robot when controlled by different models are visually depicted in Figure 7. These trajectories represent the robot’s performance for a set of target points selected from both the interpolation and extrapolation datasets. This visual representation allows for a direct comparison of how each model handles trajectory planning and control in varying complexity scenarios.

#### **3.5.4 End-to-End Learning**

Previously, targets were defined by their Cartesian coordinates, simplifying the task for the system, which only needed to approximate the transformation from Cartesian to intrinsic coordinates—a straightforward task given the analytical potential for conversion to the joint space of a 2-DOF robot. To push the boundaries of our model’s capabilities, we experimented with a more challenging input type: bitmap images used as target representations. This change necessitated the use of convolutional layers in

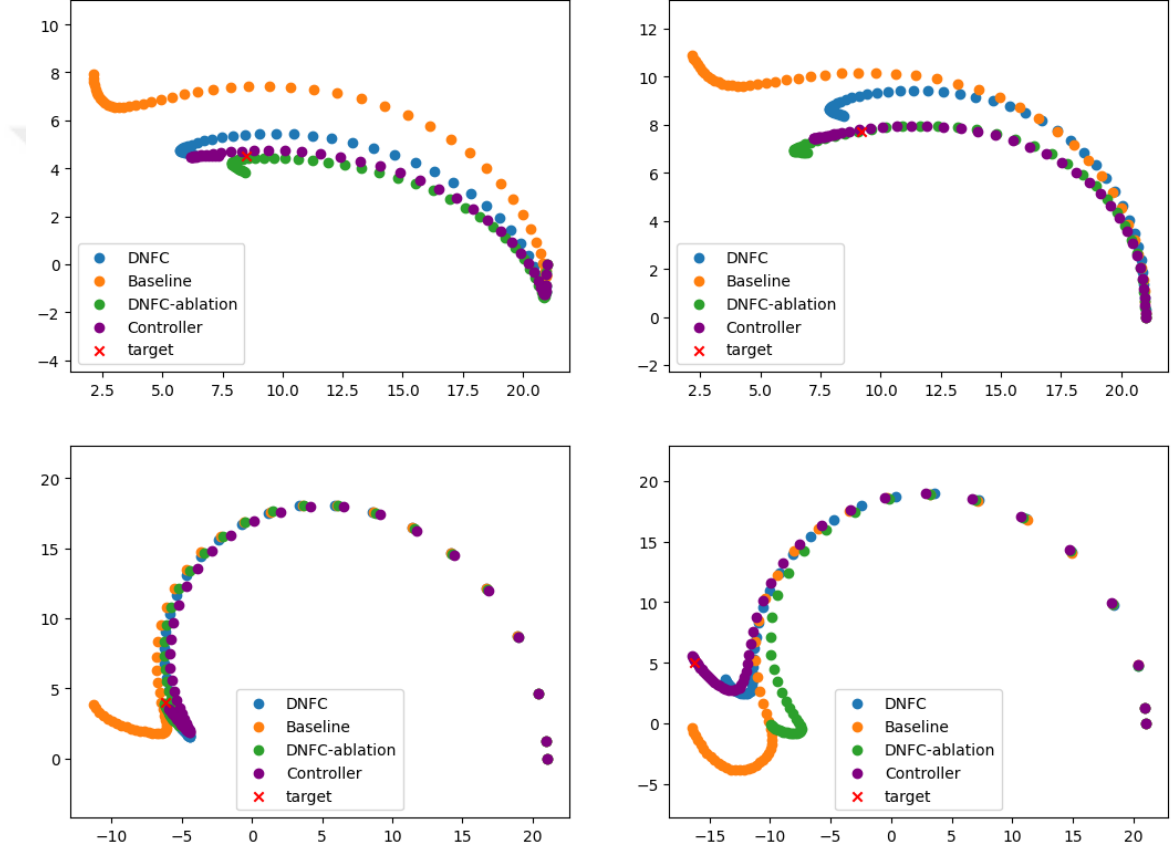


**Figure 7:** Typical trajectories followed by the ground truth controller (purple), proposed DNFC model (blue) and the baseline model (orange) are shown. Each panel shows reaching performances of the models for a single target marked with (x). Each model is given 50 time steps to reach the target.

**Table 3:** Control Performance of the proposed model (DNFC), the baseline and the latent loss ablated version of DNFC: Mean and standard deviation of Endpoint-Target Distance and Torque Prediction Errors on the Interpolation and Extrapolation test sets

| Method                     | Interpolation         |                         | Extrapolation         |                         |
|----------------------------|-----------------------|-------------------------|-----------------------|-------------------------|
|                            | Endpoint-Target Dist. | Torque Prediction Error | Endpoint-Target Dist. | Torque Prediction Error |
| <b>Approx. 270 Params</b>  |                       |                         |                       |                         |
| DNFC                       | 1.43 ( $\pm 0.58$ )   | 0.104 ( $\pm 0.025$ )   | 4.63 ( $\pm 1.25$ )   | 0.179 ( $\pm 0.04$ )    |
| Baseline                   | 3.40 ( $\pm 1.99$ )   | 0.195 ( $\pm 0.089$ )   | 9.45 ( $\pm 4.98$ )   | 0.289 ( $\pm 0.102$ )   |
| DNFC <sub>ablation</sub>   | 1.91 ( $\pm 0.77$ )   | 0.123 ( $\pm 0.038$ )   | 6.54 ( $\pm 1.32$ )   | 0.227 ( $\pm 0.053$ )   |
| <b>Approx. 570 Params</b>  |                       |                         |                       |                         |
| DNFC                       | 0.98 ( $\pm 0.09$ )   | 0.071 ( $\pm 0.017$ )   | 3.17 ( $\pm 1.43$ )   | 0.148 ( $\pm 0.062$ )   |
| Baseline                   | 1.55 ( $\pm 0.57$ )   | 0.115 ( $\pm 0.028$ )   | 4.12 ( $\pm 1.55$ )   | 0.155 ( $\pm 0.035$ )   |
| DNFC <sub>ablation</sub>   | 2.49 ( $\pm 1.27$ )   | 0.121 ( $\pm 0.038$ )   | 6.75 ( $\pm 2.22$ )   | 0.249 ( $\pm 0.069$ )   |
| <b>Approx. 1846 Params</b> |                       |                         |                       |                         |
| DNFC                       | 0.86 ( $\pm 0.04$ )   | 0.046 ( $\pm 0.008$ )   | 2.33 ( $\pm 0.48$ )   | 0.083 ( $\pm 0.022$ )   |
| Baseline                   | 1.03 ( $\pm 0.15$ )   | 0.064 ( $\pm 0.017$ )   | 3.08 ( $\pm 1.58$ )   | 0.136 ( $\pm 0.060$ )   |
| DNFC <sub>ablation</sub>   | 1.04 ( $\pm 0.17$ )   | 0.050 ( $\pm 0.011$ )   | 3.78 ( $\pm 1.13$ )   | 0.109 ( $\pm 0.025$ )   |

the encoder to handle these less-informative inputs effectively.

This part of the study was replicated twice to validate the concept. The same datasets previously described for interpolation and extrapolation assessments were employed here, though the total sample size was adjusted to  $N = 2 \times 5 = 10$  for this specific test setting. Outcomes of this experimental setup are documented in Table 4. Results indicate a decline in the system’s extrapolation performance when utilizing image-based inputs, although interpolation results remain robust.

**Table 4:** Image based control performance of the proposed model (DNFC): Mean and standard deviation of Endpoint-Target Distance and Torque Prediction Errors

|                   | Endpoint-Target Dist. | Torque Prediction Error |
|-------------------|-----------------------|-------------------------|
| Interpolation Set | 2.04 ( $\pm 1.34$ )   | 0.182 ( $\pm 0.138$ )   |
| Extrapolation Set | 11.39 ( $\pm 1.78$ )  | 0.427 ( $\pm 0.007$ )   |

### 3.5.5 Dynamic Targets

To evaluate the adaptability of our model, DNFC, which was initially trained using static targets, we introduced dynamic targets in the testing phase. Specifically, during each episode, targets move along a linear path at a constant velocity from one randomly selected point, A, to another, B. This scenario tests the model’s ability to cope

with changing target locations, which mimics more realistic and variable operational environments.

To quantify the model’s performance in this dynamic setting, we employed the Tracking Error metric. This metric calculates the average Euclidean distance between the robot’s end effector and the moving target’s position throughout an episode.

During the dynamic target experiments, 500 pairs of points (A, B) were randomly selected from the robot’s workspace to create movement scenarios. These points dictated the linear paths that the targets traversed at a constant speed, setting the stage for testing the tracking capabilities of our models. The data below, presented in a table, aggregates the average Tracking Error for both DNFC and the baseline models across different complexity levels:

| Model Complexity | DNFC Tracking Error (cm) | Baseline Tracking Error (cm) |
|------------------|--------------------------|------------------------------|
| Low              | $2.45 \pm 0.55$          | $3.58 \pm 1.56$              |
| Middle           | $2.11 \pm 0.37$          | $2.67 \pm 0.31$              |
| High             | $1.56 \pm 0.12$          | $2.33 \pm 0.25$              |

**Table 5:** Comparison of average Tracking Error between DNFC models and baseline across different complexity levels.

These results indicate that the DNFC models consistently outperform the baseline in a statistically significant manner across all complexity levels. Please note that in this section, the models were tasked with tracking moving targets while they were trained only via data obtained from reaching executions toward static goals.

For a practical demonstration, we visualized trajectories for a circular path tracking test. The results, depicted in Figure 8, show notable differences between the controllers. While the ground truth controller does not perfectly track the trajectory, particularly towards the end, DNFC demonstrates significant radial tracking ability. However, it does not maintain a perfect circular path. In stark contrast, the baseline model displays considerable difficulty, failing to generate a radial trajectory and directing the end effector away from the intended path.

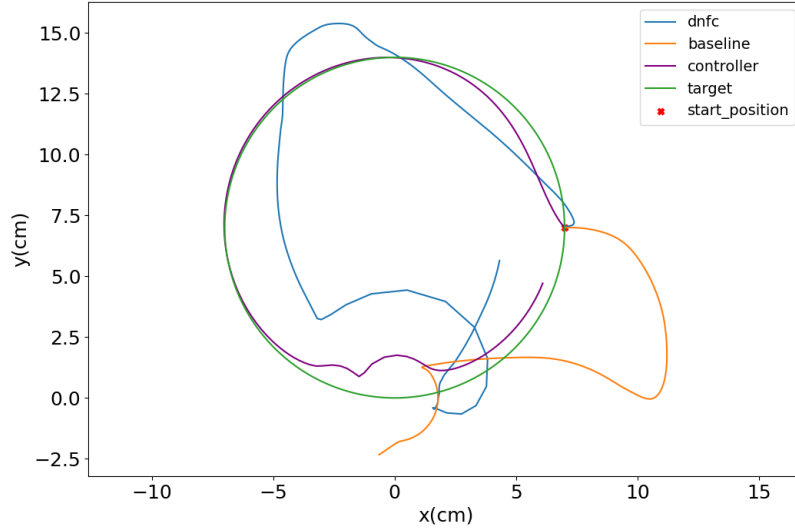


**Figure 8:** The end effector trajectories traced by each model when trying to reach the target moving in a circular path. Note that DNFC and the baseline model have been trained using only static targets.

### 3.6 Discussion

This research develops a novel deep neural network framework named the Deep Neural Feedback Controller Network (DNFC), designed to function as a non-linear feedback controller by learning from demonstrated robot trajectories. The effectiveness of DNFC is demonstrated through rigorous experiments and comparisons using a robotic reaching task. As a learning from demonstration (LfD) system, DNFC uniquely automates the construction of latent state representations which serve as objectives for the robot’s state, adapting to various implementations of the target task. When compared to straightforward behavior cloning (BC) using neural networks with equivalent parameter complexities, DNFC exhibits significantly enhanced performance. This improvement underscores the advantageous structural bias inherent in DNFC. Moreover, DNFC’s performance advantage becomes more pronounced at lower parameter complexities, suggesting its utility as a resource-efficient BC system capable of autonomously identifying task objectives. Additionally, results from the ablation

study confirm that the proposed latent loss plays a critical role in enhancing DNFC's performance.



## CHAPTER IV

# LEARNING A FEEDBACK CONTROLLER WITH DYNAMIC TARGETS

A more challenging task for a feedback controller is tracking dynamic targets, where targets move during each episode. In this chapter, we systematically investigate the effectiveness of our proposed model for such tasks. In the previous chapter, we concluded that DNFC is highly effective in resource-scarce conditions. Therefore, in this chapter, we examine its performance under these conditions with dynamic targets.

### 4.1 *Data*

Since it is often challenging to gather data for robots in the real world, it is crucial to develop models that can learn effectively with a limited amount of data. Therefore, in this chapter, we limit the number of data points in our dataset to simulate real-world conditions.

In this chapter, targets move linearly during the execution of each episode as follows: First, two random points within the polar grid described in Section 3.4.2, are selected. One point is designated as the start of the target’s trajectory, and the other as the end point.

The target then begins at the start point and linearly moves toward the end point at a constant speed of 0.1 cm per step. If the target reaches the end point before the maximum number of steps allowed in each episode (200 in this setup), it waits at the end point until the episode concludes. A few sample goal trajectories that are used in the training set are illustrated in Figure 9.

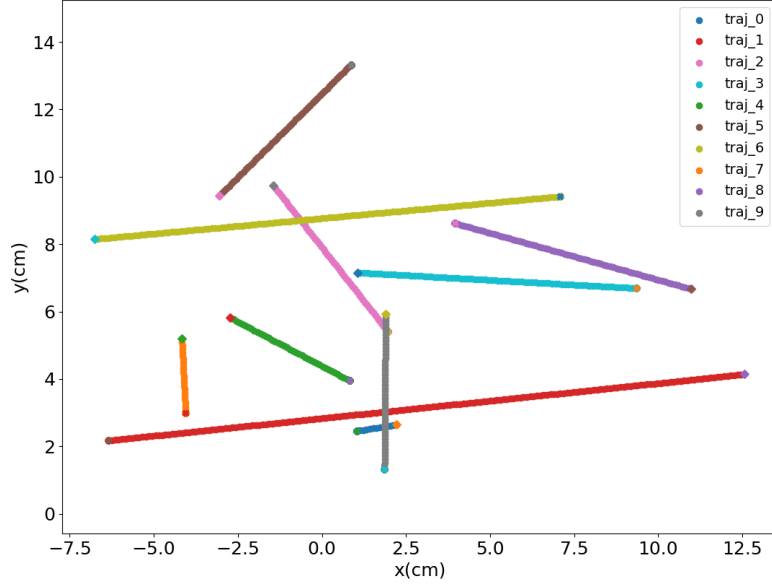


**Figure 9:** Ten sample goal trajectories used in the training set. The start point of each trajectory is illustrated with a diamond, and the end point is marked with an X.

We intentionally limited the number of data points in our training set to simulate real-world conditions where obtaining data is challenging. In this setup, we selected 256 goal trajectories. Each episode consists of 200 steps, resulting in a total of  $256 \times 200$  data points in our training set. As in the previous chapter, a PD controller is tasked with reaching the current position of the goal at each step, allowing it to track the goal throughout the entire episode. The data collected by the PD controller is then used as the training dataset. The robot end-effector does not start its movement exactly from the target's start point. Instead, a uniform random noise within the range of  $[-5, +5]$  cm is added to both the x and y coordinates of the target's start point to determine the initial position of the end-effector.

Each data point in the dataset includes the timestep, the values of joint angles and joint velocities at that timestep, the current Cartesian position of the goal, and the corresponding torques calculated by the PD controller for that timestep. Since

the robot has two joints and the target operates on a 2D plane, each data point is 9-dimensional. Specifically, 4 dimensions correspond to the state space (2 joint angles and 2 joint velocities), while the action space is 2-dimensional. The torques are constrained to be within the  $[-1, +1]$  interval due to environmental limits.

Since there are 200 steps in each episode and the goal can travel 0.1 cm per step, the maximum distance a goal can travel during an episode is 20 cm =  $200 \times 0.1$ . However, because the start and end points of each episode are chosen randomly, goals typically do not travel this maximum distance. The distribution of the distances traveled by the targets in each episode in our dataset is illustrated in Figure 10. As shown in the figure, the mean distance is approximately 7.5 cm, indicating that the goal generally becomes idle at the end point after roughly 75 steps. Consequently, when the goal reaches the end point and then the robot catches it, it generates very small torque values to stay there. We will later discuss why this is important for our proposed model and how it contributes to its superior performance.

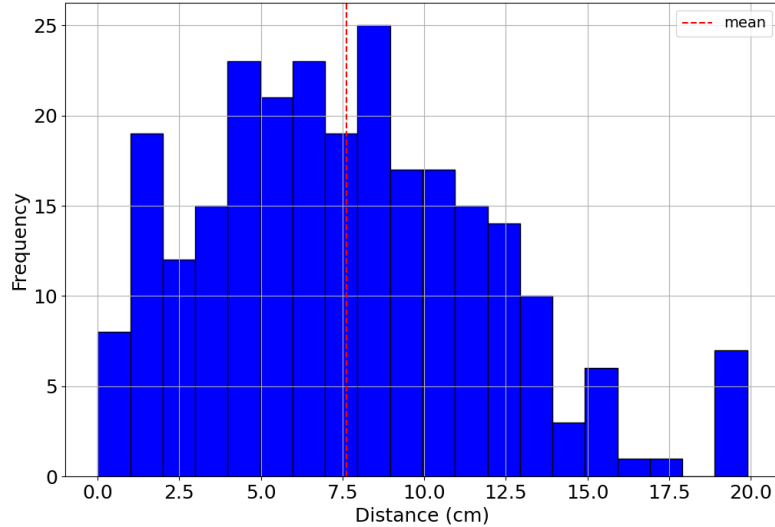


**Figure 10:** Histogram of the distances the targets traveled during each episode in the training dataset. The mean distance is roughly 7.5 cm.

In this chapter as well, we tested the models on two distinct test sets: interpolation

and extrapolation. In the interpolation test set, the targets move within the same radial and angular range of the training set; however, in the extrapolation test set, the radial range extends beyond the radial range of the training set. This distinction is illustrated in Figure 11.

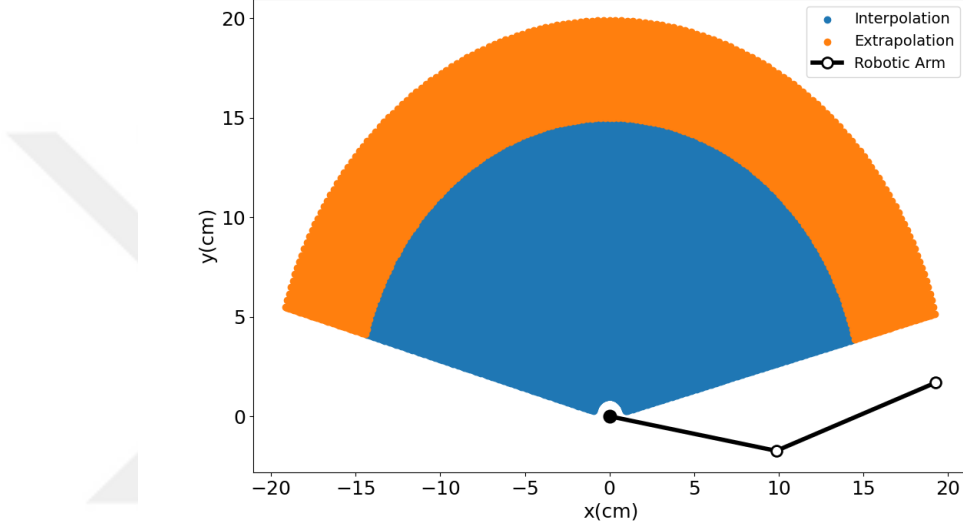


**Figure 11:** Illustration of the radial and angular ranges for the interpolation and extrapolation test sets.

## 4.2 Model and Loss Function

The model architecture is the same as the one described in Section 3.1. However, the loss function has changed. The *torque* loss described in Equation 7 remains the same in this chapter. A minor change has been made to the *latent* loss. The new latent loss is formulated as follows:

$$\mathcal{L}_s = \sum_{t=0}^{T-1} e^{\beta(t/T-1)} \|s(t) - \hat{s}(t)\|^2 \quad (12)$$

where  $t$  is the timestep,  $T$  denotes the maximum number of timesteps within each episode, and  $s(t)$  and  $\hat{s}(t)$  denote the state of the robot and the latent representation

of the target at timestep  $t$ , respectively. The only difference between this equation and Equation 8 is the absence of the parameter  $\alpha$ .

The  $\alpha$  parameter is added as a coefficient to  $\mathcal{L}_s$  when summing it with  $\mathcal{L}_u$  to obtain the total loss, resulting in the following equation:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_u + \alpha \mathcal{L}_s \quad (13)$$

The reason behind these minor changes is to improve the interpretability of the equations. In Equation 13,  $\alpha$  determines the extent to which the model prioritizes generating latent representations of targets that mirror the robot state, especially at the end of the episode. The effect of  $\beta$  is illustrated in Figure 12. As shown in the figure, as the value of  $\beta$  increases,  $\mathcal{L}_s$  attains smaller values at the beginning of episodes, meaning the final loss is mainly determined by  $\mathcal{L}_u$ . This means the model focuses more on generating torques that are similar to the controller’s outputs rather than producing latent representations similar to the robot state.

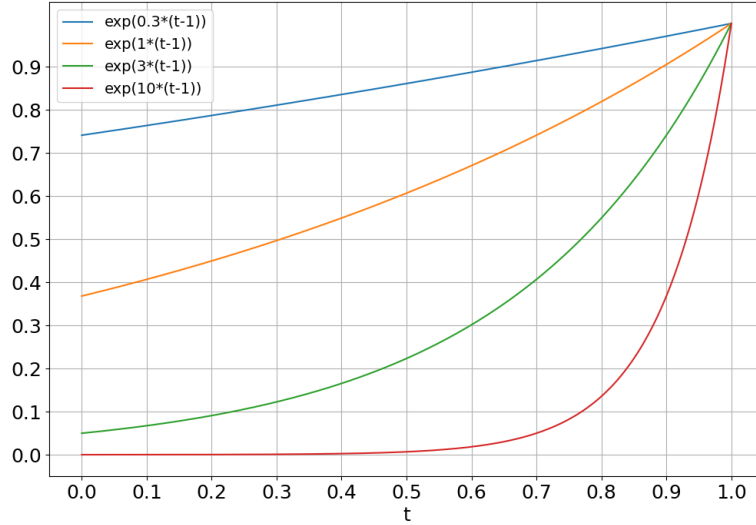


**Figure 12:** The effects of parameter  $\beta$  on the latent loss ( $\mathcal{L}_u$ ). The term  $e^{\beta(t/T-1)}$  is multiplied by the  $l^2$ -norm of the discrepancy between the latent representation of the target and the robot state at timestep  $t$ . As  $\beta$  increases,  $\mathcal{L}_s$  attains smaller values at the beginning of the episodes.

### 4.3 *Performance Metrics*

A new performance metric is introduced due to the nature of the problem addressed in this chapter. Since the targets are moving during the episodes in this chapter, we not only desire that the robot maintains a small distance to the final position of the goal, but also tracks the goal throughout the entire episode.

#### 4.3.1 Tracking Error

This metric calculates the Euclidean distance between the position of the target and the robot’s end-effector at each timestep  $t$ , then averages it over the entire episode. Ultimately, the metric indicates the average distance between the robot’s end-effector and the goal throughout the episode. If the position of the Reacher’s end-effector at timestep  $t$  of an episode is denoted by  $(x'(t), y'(t))$  and the target point’s position at the same timestep is denoted by  $(x(t), y(t))$ , then the Tracking Error metric (TE) is computed as:

$$\text{TE} = \frac{1}{T} \sum_{t=0}^{T-1} \sqrt{(x'(t) - x(t))^2 + (y'(t) - y(t))^2} \quad (14)$$

where  $T$  denotes the total number of timesteps that exist in each episode (200 in this chapter.)

Besides this new metric, the Robot Endpoint-Target Distance and Torque Prediction Error metrics introduced in section 3.4.3 are also used in this chapter as well.

### 4.4 *Results*

To ensure statistically robust results, each model was trained three times, with each training session using a different initialization. This approach allowed us to evaluate the performance of both the proposed model and the baseline under varying conditions. Additionally, as in the previous chapter, ablation experiments were conducted to demonstrate the impact of the latent loss on the performance of the proposed

model.

We tested our models on 128 target trajectories, with the start and end points selected randomly. Therefore, the size of the test set is half of the training set, providing a fair assessment of the models' behavior. For each interpolation set and extrapolation set, 128 different trajectories with varying properties were selected that we will mention their details in their corresponding section.

For comprehensive statistical analysis, similar to section 3.5, we first averaged the results across the 128 episodes of the test set to derive an average performance metric over the entire test set. Following this, we calculated and reported the mean, standard deviation (STD), and standard error of the mean (SEM) across the resulting 3-dimensional vector, where 3 corresponds to the number of training sessions each model underwent.

It is important to note that in this section, all models were trained for only 200 epochs, and the following results were produced under these conditions. The reason for limiting the number of training epochs was to create resource-scarce conditions to evaluate the models and measure their performance in such scenarios.

#### **4.4.1 Control Performance: Interpolation**

In the interpolation test set, the movement of the targets throughout the entire episodes remained within the polar grid described in section 3.4.2, i.e., within the convex hull of the training dataset. This task is easier for the models since they are being tested on a dataset that falls within the same radial and angular range as the training set.

The results of the tracking error are depicted in Figure 13. From the figure, it is evident that the DNFC model outperforms the baseline across all levels of model complexity. Furthermore, all models achieve lower errors as their complexity increases; however, the baseline benefits more significantly from this increase. Another notable

point is the large gap in performance between our model and the baseline at lower complexity levels, such as when the models have 270 and 570 learnable parameters. This indicates that our model is a strong candidate when there are constraints on model size, as it not only demonstrates acceptable behavior but also surpasses the baseline by a significant margin.

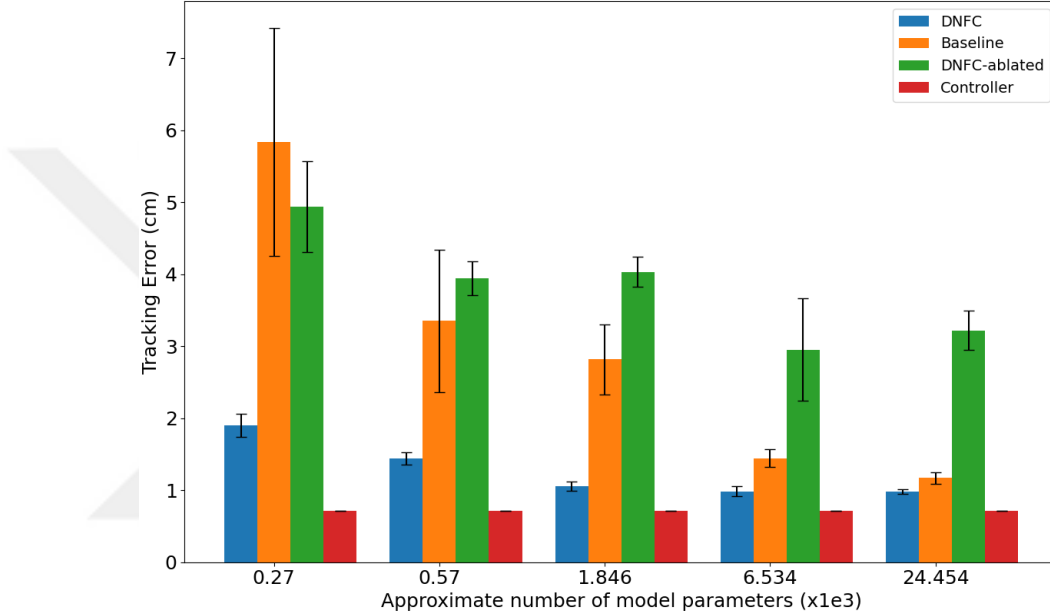


**Figure 13:** Mean of the tracking error metric on the interpolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.)

We observe a similar behavior when considering the torque prediction error metric, as shown in Figure 14. Notably, here the latent loss ablated version of our model outperforms the baseline almost at all complexity levels. This indicates that the structure of our model, even without the aid of latent loss, enables it to better mimic the ground truth controller.

Finally, the performance of the models with respect to the robot endpoint-target distance is illustrated in Figure 15. Once again, we observe that our proposed model outperforms the baseline almost at all model complexity levels. However, we also notice the poor performance of the ablated version of DNFC which fails to achieve

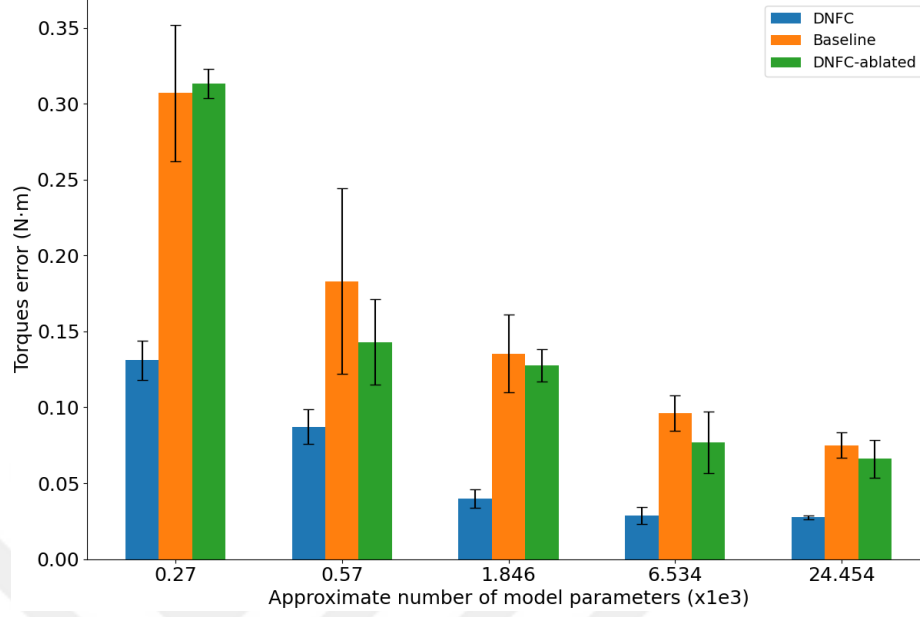


**Figure 14:** Mean of the torque prediction error metric on the interpolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.)

lower distances even in high complexity levels of the model. Furthermore, none of the models could approximate the distance achieved by the ground truth controller, even at high model complexity levels.

One way to understand the significant performance gap between DNFC and its latent loss ablated version is by considering the instances where the model reached the target endpoint and remained there until the episode concluded. While the model generated no or very small torques, the latent loss compelled it to produce latent representations of targets that mimic the robot’s state at the end of the episode. This process implicitly aids the model in mapping the Cartesian coordinates of the target to its intrinsic coordinates. Without this feature, the latent loss ablated version of DNFC performed poorly, even compared to the baseline. It appears that the feedback-like structure, without the assistance of latent loss, hinders the model’s learning process.

The mean and standard error of the mean (SEM) values for our proposed model (DNFC), the baseline, the latent loss ablated version of DNFC, and the ground truth

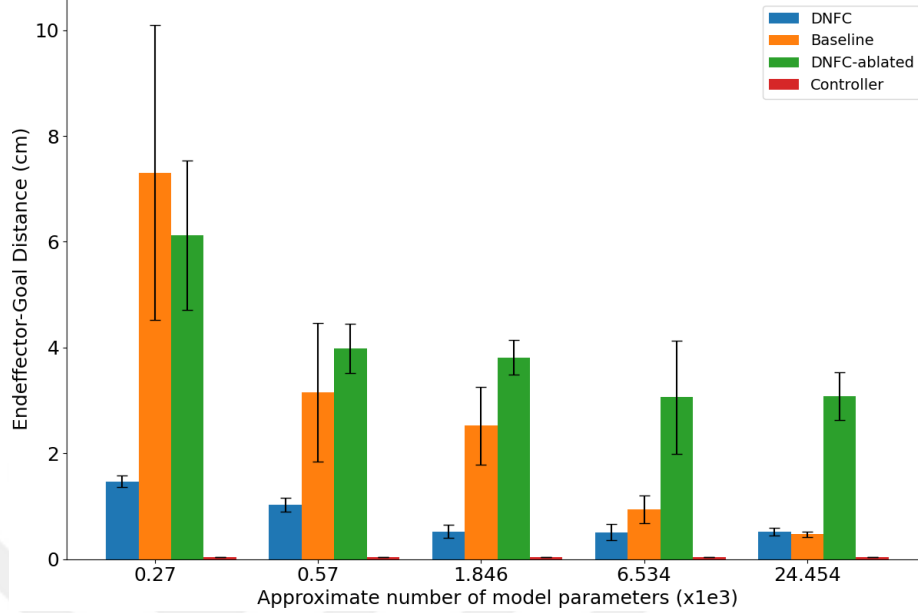


**Figure 15:** Mean of the endpoint-target distance on the interpolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.)

controller can be found in Table 6.

#### 4.4.2 Control Performance: Extrapolation

Extrapolation is a major challenge for supervised learning systems, especially for Behavior Cloning (BC), as it requires assessing the system’s performance on previously unseen data. This difficulty is amplified when the data falls outside the convex hull of the training set. To address this issue, we tested models on target trajectories that start and end at points beyond the radial range from which the training data were sampled, specifically  $15\text{cm} < r < 20\text{cm}$ . Given that these new trajectories are often longer, we increased the maximum number of steps allowed in each episode from 200 to 300. Consequently, there are two new features in this test set: radial distance and trajectory length.

The comparison between the performance of models regarding the Tracking Error (TE) metric is illustrated in Figure 16. As shown in the figure, our proposed model

(DNFC) outperforms the baseline at all model complexity levels. The difference in performance is particularly pronounced at the lowest model complexity level (270 learnable parameters). Furthermore, at this level, the latent loss ablated version of DNFC outperforms the baseline, suggesting the effectiveness of the structure itself when resources are scarce. As the complexity of the models increases, the performance gap between the DNFC and the baseline decreases, becoming very close at the highest complexity level. However, the latent loss ablated version of DNFC does not benefit significantly from the increased model complexity. This shows that it probably needs more epochs of training for becoming an effective controller and its feedback controller-like structure without enough training prevents it from efficient learning.

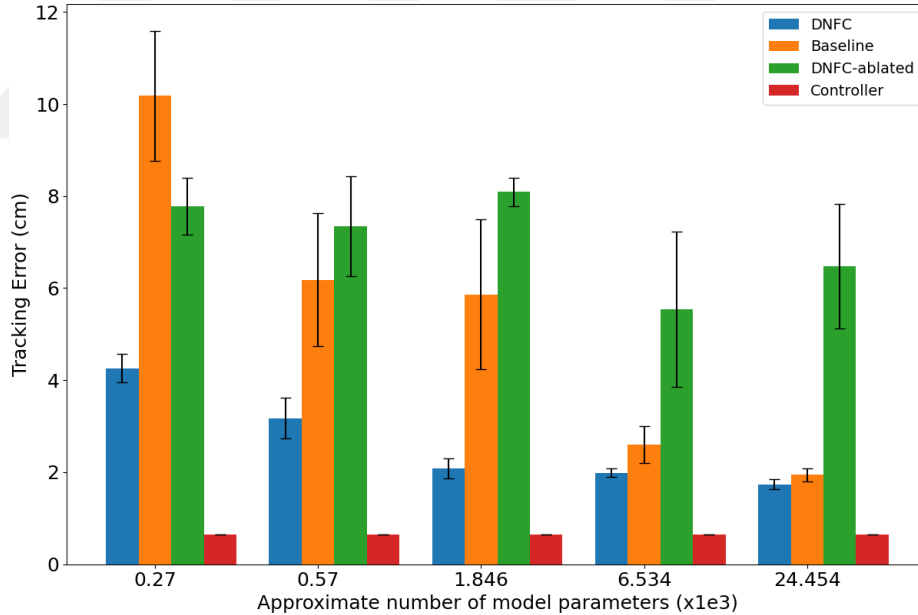


**Figure 16:** Mean of the tracking error metric on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.)

Regarding the Endeffector-Target Distance metric, the comparison of the models' performances is depicted in Figure 17. Besides the familiar trend of DNFC outperforming the baseline at almost all model complexity levels, it is noteworthy that none

of the models reached to the same distance that the ground truth controller achieved. We observed a similar trend in the interpolation set results; however, the gap is now more pronounced, suggesting that the problem tackled in this section is more challenging. Once again, we observe that the performance gap between DNFC and the baseline is amplified at lower model complexity levels, highlighting the strength of our proposed model in resource-scarce conditions. Finally, the poor performance of the latent loss ablated version of DNFC demonstrates the importance of the proposed latent loss in achieving superior performance with our model.

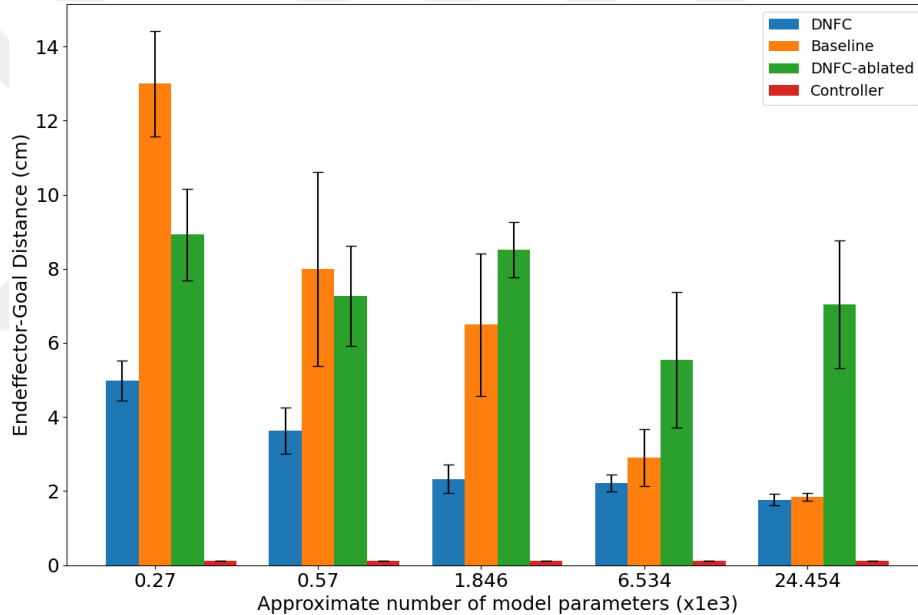


**Figure 17:** Mean of the endpoint-target distance on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.)

We observe that both DNFC and its latent loss ablated version perform relatively better when considering the Troque Prediction Error metric, as shown in Figure 18. The performance of the latent loss ablated version of DNFC is very close to the baseline, suggesting the feedback controller-like structure of DNFC, indeed helps the model to become an efficient feedback controller. Since this metric measures how each model mimics the ground truth controller in various states, it is meaningless to

report values corresponding to the performance of the controller itself, as they should be equal to zero.

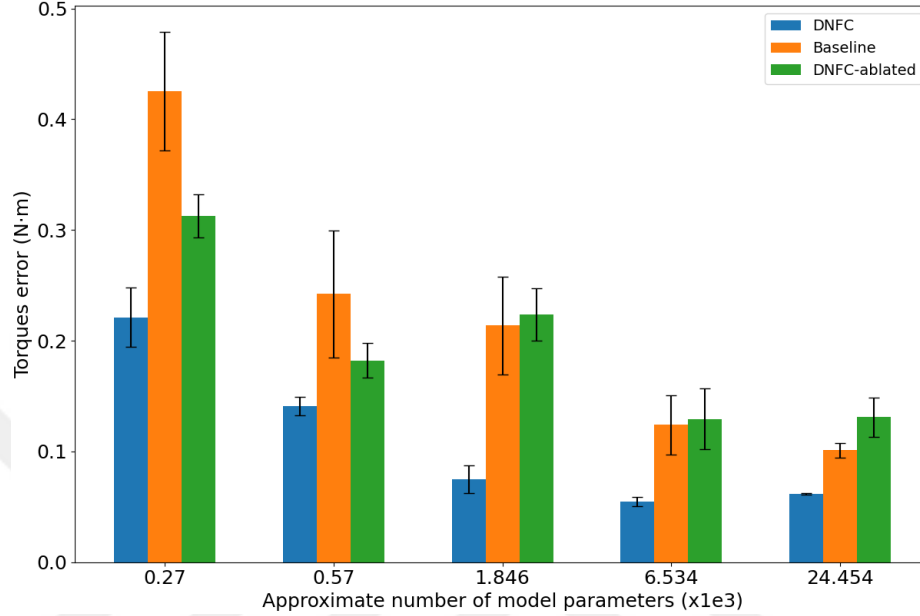


**Figure 18:** Mean of the torque prediction error metric on the extrapolation test set with SEM bars for different number of model parameters, i.e., complexity levels (Blue: DNFC, the proposed model; Orange: baseline; Green: latent loss ablated version of the proposed model; Red: Controller, the ground truth.)

Finally, the mean and standard error of the mean (SEM) values for our proposed model (DNFC), the baseline, the latent loss ablated version of DNFC, and the ground truth controller are presented in Table 7. Sample trajectories taken by our proposed model (DNFC), the baseline, and the ground truth controller on the interpolation and extrapolation test sets are depicted in Figures 19 and 20, respectively. In these figures, the left and right columns correspond to the same target trajectories; however, the same model exhibit different behaviors due to different training initializations. Otherwise, all other characteristics of the environment and models are the same, and both DNFC and the baseline have the same level of model complexity, with approximately 1846 learnable parameters in both models. As is evident from the figures, the baseline often exhibits unreasonable movements, such as going in the opposite direction of the target trajectory, while DNFC is more stable in this regard.

Furthermore, it appears that the baseline sometimes fails to accurately *track* the target, instead passing through positions the target has passed and then overshooting. Eventually, the performance of both models worsens on the extrapolation test set, which is understandable given its nature.

#### 4.5 *Learning Latent Representation of Target*

As discussed in previous sections, we anticipate that incorporating the latent loss component into our training pipeline will facilitate the learning of an approximate Cartesian-to-intrinsic coordinates mapping within the Encoder module. Consequently, we expect the latent representation of the target, denoted by  $\hat{s}$ , to serve as a reference for the robot’s state, denoted by  $s$ , at each time step  $t$ . As illustrated in Figure 21, our expectations are confirmed, as  $s$  closely follows the trajectory of  $\hat{s}$ . In the figure, the left plots display the actual values of  $s$  and  $\hat{s}$ , while the right plots depict their difference,  $\hat{s} - s$ , across each dimension. For instance,  $s_1$ , representing the angular position of the first joint ( $q_1$ ), frequently converges to  $\hat{s}_1$ , resulting in a near-zero difference, particularly at the conclusion of episodes. Although  $s_2$ , corresponding to the angular position of the second joint ( $q_2$ ), follows the trend of  $\hat{s}_2$ , it converges to  $\hat{s}_2 + 2$  instead of  $\hat{s}_2$ . Further research is needed to understand the underlying cause of this behavior; however, it is clear that the Encoder learns a pattern. The mean and STD of the difference vector across the entire interpolation test set for each dimension is  $-0.23 \pm 0.30$ ,  $-2.21 \pm 0.45$ ,  $0.05 \pm 0.81$ , and  $-0.27 \pm 1.14$ , respectively. Nevertheless, the results suggest that the Encoder module has successfully identified a reasonable mapping pattern from the Cartesian coordinates of the target to the robot’s intrinsic state values (i.e., the angular position and velocity values of the two joints).

#### 4.6 *Discussion*

In this chapter we trained our proposed model (DNFC), its latent loss ablated version, and the baseline on targets that move linearly throughout each episode. We

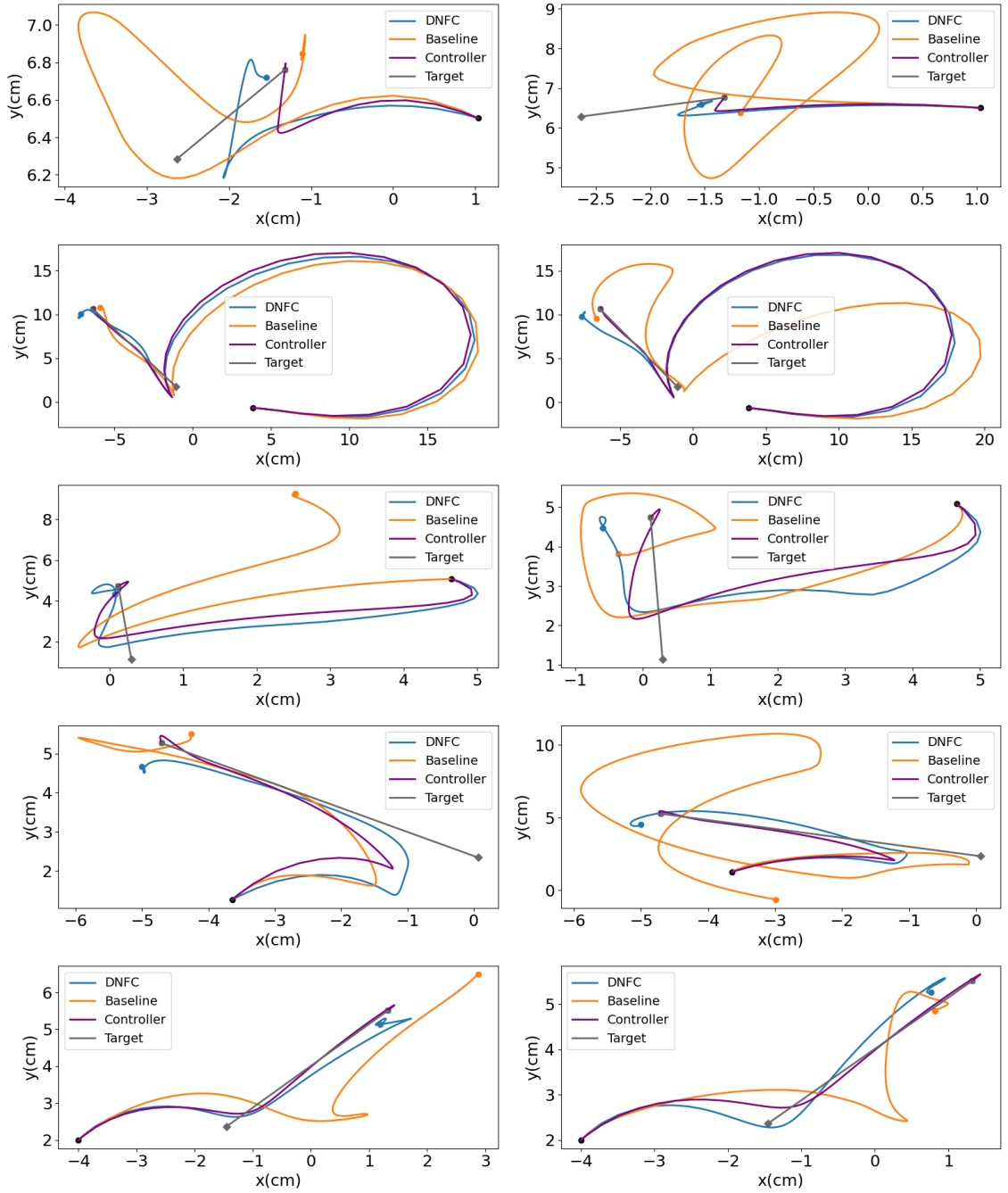


**Figure 19:** Five sample trajectories from the interpolation test set taken by the DNFC model (Blue), the baseline (Orange), and the ground truth controller (Purple) are shown, with the target trajectory in Gray. The start point of the target trajectory is illustrated by a diamond, and the end point by an X. The start point of the robot end effector is indicated by a black dot. The left and right columns correspond to the same target trajectories; however, the DNFC and baseline models were trained with different initializations. Both models have the same complexity level (1846 learnable parameters).

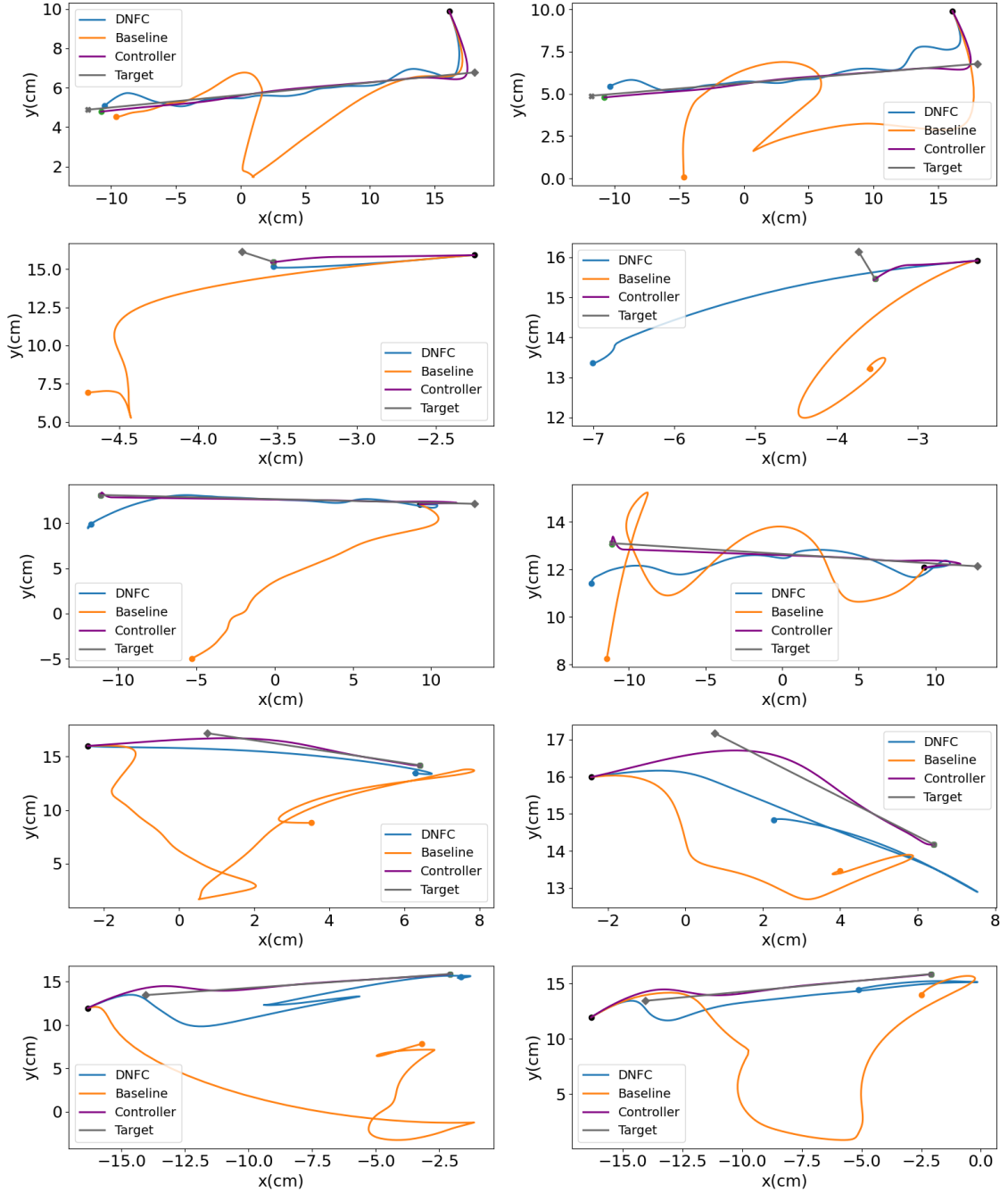


**Figure 20:** Five sample trajectories from the extrapolation test set taken by the DNFC model (Blue), the baseline (Orange), and the ground truth controller (Purple) are shown, with the target trajectory in Gray. The start point of the target trajectory is illustrated by a diamond, and the end point by an X. The start point of the robot end effector is indicated by a black dot. The left and right columns correspond to the same target trajectories; however, the DNFC and baseline models were trained with different initializations. Both models have the same complexity level (1846 learnable parameters).

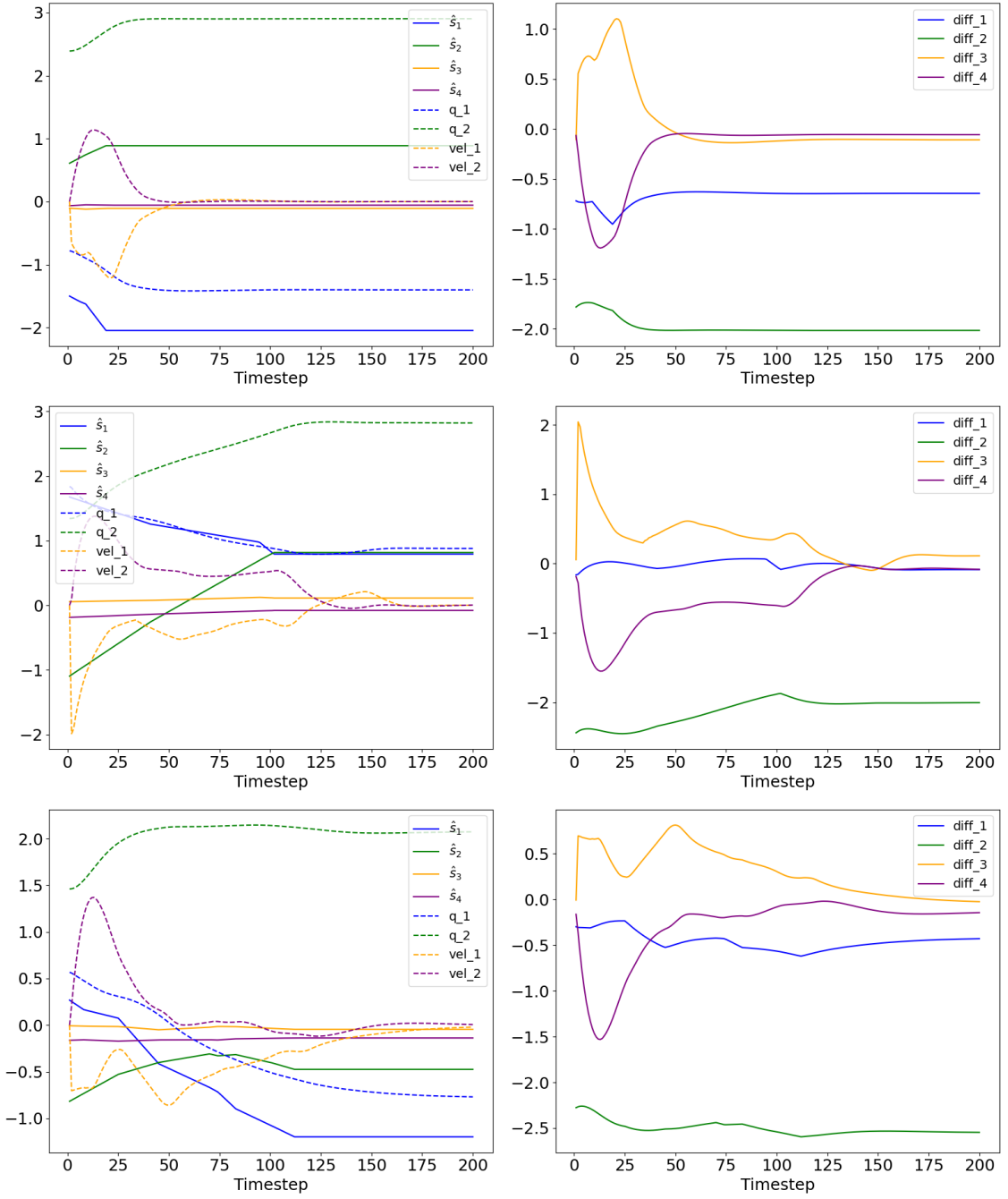


**Figure 21:** Visualization of the latent representations of the target ( $\hat{s}$ ) and the robot state ( $s$ ) across the entire duration of three distinct episodes, sampled from the interpolation set. The robot state includes the angle ( $q$ ) and velocity ( $vel$ ) of two joints. Each row represents a different episode. In the left column, the latent representation of target and robot state are depicted throughout the entire episode, while the right column shows their differences ( $diff$ ) across each dimension.

aimed to learn a controller that can *track* linear movements of targets during each episode. We tested the models on interpolation and extrapolation test sets and observed that our proposed model (DNFC) is able to outperform the baseline in almost all model complexity levels. Furthermore, the performance gap between DNFC and the baseline is more pronounced in low complexity models suggesting that our proposed model is a better candidate in resource-scarce conditions. Finally, investigating the learned latent representations of targets clearly demonstrates that the Encoder module has learned an approximate Cartesian-to-intrinsic mapping; however, in some dimensions, e.g., the second dimension which corresponds to the angular position of the second joint,  $q_2$  converges to  $\hat{s}_2 + 2$  instead of converging to  $\hat{s}_2$  which needs further investigation.

**Table 6:** Control Performance of the proposed model (DNFC), the baseline, the latent loss ablated version of DNFC, and the ground truth controller trained & tested with dynamic targets: Mean and standard error of the mean (SEM) of Tracking Error, Torque Prediction Error, and Endpoint-Target Distance on the Interpolation test set.

| Method                          | Tracking Error      | Torque Prediction Error | Endpoint-Target Dist. |
|---------------------------------|---------------------|-------------------------|-----------------------|
| <b>Approx. 270 Parameters</b>   |                     |                         |                       |
| DNFC                            | 1.90 ( $\pm 0.16$ ) | 0.131 ( $\pm 0.013$ )   | 1.47 ( $\pm 0.11$ )   |
| Baseline                        | 5.84 ( $\pm 1.58$ ) | 0.307 ( $\pm 0.045$ )   | 7.31 ( $\pm 2.79$ )   |
| DNFC <sub>ablation</sub>        | 4.94 ( $\pm 0.63$ ) | 0.313 ( $\pm 0.010$ )   | 6.12 ( $\pm 1.41$ )   |
| Controller                      | 0.71 ( $\pm 0.00$ ) | —                       | 0.04 ( $\pm 0.00$ )   |
| <b>Approx. 570 Parameters</b>   |                     |                         |                       |
| DNFC                            | 1.44 ( $\pm 0.09$ ) | 0.087 ( $\pm 0.011$ )   | 1.02 ( $\pm 0.13$ )   |
| Baseline                        | 3.35 ( $\pm 0.99$ ) | 0.183 ( $\pm 0.061$ )   | 3.15 ( $\pm 1.31$ )   |
| DNFC <sub>ablation</sub>        | 3.94 ( $\pm 0.23$ ) | 0.143 ( $\pm 0.028$ )   | 3.99 ( $\pm 0.46$ )   |
| Controller                      | 0.71 ( $\pm 0.00$ ) | —                       | 0.04 ( $\pm 0.00$ )   |
| <b>Approx. 1846 Parameters</b>  |                     |                         |                       |
| DNFC                            | 1.05 ( $\pm 0.07$ ) | 0.040 ( $\pm 0.006$ )   | 0.52 ( $\pm 0.12$ )   |
| Baseline                        | 2.81 ( $\pm 0.49$ ) | 0.135 ( $\pm 0.026$ )   | 2.52 ( $\pm 0.73$ )   |
| DNFC <sub>ablation</sub>        | 4.03 ( $\pm 0.21$ ) | 0.128 ( $\pm 0.011$ )   | 3.81 ( $\pm 0.33$ )   |
| Controller                      | 0.71 ( $\pm 0.00$ ) | —                       | 0.04 ( $\pm 0.00$ )   |
| <b>Approx. 6534 Parameters</b>  |                     |                         |                       |
| DNFC                            | 0.98 ( $\pm 0.07$ ) | 0.029 ( $\pm 0.006$ )   | 0.51 ( $\pm 0.15$ )   |
| Baseline                        | 1.44 ( $\pm 0.12$ ) | 0.096 ( $\pm 0.012$ )   | 0.94 ( $\pm 0.27$ )   |
| DNFC <sub>ablation</sub>        | 2.95 ( $\pm 0.71$ ) | 0.077 ( $\pm 0.020$ )   | 3.06 ( $\pm 1.06$ )   |
| Controller                      | 0.71 ( $\pm 0.00$ ) | —                       | 0.04 ( $\pm 0.00$ )   |
| <b>Approx. 24454 Parameters</b> |                     |                         |                       |
| DNFC                            | 0.98 ( $\pm 0.03$ ) | 0.027 ( $\pm 0.001$ )   | 0.52 ( $\pm 0.08$ )   |
| Baseline                        | 1.17 ( $\pm 0.07$ ) | 0.075 ( $\pm 0.008$ )   | 0.47 ( $\pm 0.05$ )   |
| DNFC <sub>ablation</sub>        | 3.22 ( $\pm 0.28$ ) | 0.066 ( $\pm 0.012$ )   | 3.07 ( $\pm 0.45$ )   |
| Controller                      | 0.71 ( $\pm 0.00$ ) | —                       | 0.04 ( $\pm 0.00$ )   |

**Table 7:** Control Performance of the proposed model (DNFC), the baseline, the latent loss ablated version of DNFC, and the ground truth controller trained & tested with dynamic targets: Mean and standard error of the mean (SEM) of Tracking Error, Torque Prediction Error, and Endpoint-Target Distance on the Extrapolation test set.

| Method                          | Tracking Error       | Torque Prediction Error | Endpoint-Target Dist. |
|---------------------------------|----------------------|-------------------------|-----------------------|
| <b>Approx. 270 Parameters</b>   |                      |                         |                       |
| DNFC                            | 4.25 ( $\pm 0.31$ )  | 0.221 ( $\pm 0.027$ )   | 4.97 ( $\pm 0.54$ )   |
| Baseline                        | 10.17 ( $\pm 1.41$ ) | 0.425 ( $\pm 0.053$ )   | 13.00 ( $\pm 1.42$ )  |
| DNFC <sub>ablation</sub>        | 7.78 ( $\pm 0.61$ )  | 0.312 ( $\pm 0.019$ )   | 8.93 ( $\pm 1.24$ )   |
| Controller                      | 0.64 ( $\pm 0.00$ )  | —                       | 0.12 ( $\pm 0.00$ )   |
| <b>Approx. 570 Parameters</b>   |                      |                         |                       |
| DNFC                            | 3.17 ( $\pm 0.45$ )  | 0.141 ( $\pm 0.008$ )   | 3.63 ( $\pm 0.62$ )   |
| Baseline                        | 6.18 ( $\pm 1.44$ )  | 0.242 ( $\pm 0.057$ )   | 7.99 ( $\pm 2.62$ )   |
| DNFC <sub>ablation</sub>        | 7.35 ( $\pm 1.09$ )  | 0.182 ( $\pm 0.015$ )   | 7.26 ( $\pm 1.35$ )   |
| Controller                      | 0.64 ( $\pm 0.00$ )  | —                       | 0.12 ( $\pm 0.00$ )   |
| <b>Approx. 1846 Parameters</b>  |                      |                         |                       |
| DNFC                            | 2.08 ( $\pm 0.22$ )  | 0.075 ( $\pm 0.013$ )   | 2.32 ( $\pm 0.38$ )   |
| Baseline                        | 5.86 ( $\pm 1.62$ )  | 0.214 ( $\pm 0.044$ )   | 6.49 ( $\pm 1.92$ )   |
| DNFC <sub>ablation</sub>        | 8.09 ( $\pm 0.30$ )  | 0.223 ( $\pm 0.024$ )   | 8.51 ( $\pm 0.75$ )   |
| Controller                      | 0.64 ( $\pm 0.00$ )  | —                       | 0.12 ( $\pm 0.00$ )   |
| <b>Approx. 6534 Parameters</b>  |                      |                         |                       |
| DNFC                            | 1.99 ( $\pm 0.10$ )  | 0.055 ( $\pm 0.004$ )   | 2.21 ( $\pm 0.22$ )   |
| Baseline                        | 2.59 ( $\pm 0.40$ )  | 0.124 ( $\pm 0.027$ )   | 2.90 ( $\pm 0.78$ )   |
| DNFC <sub>ablation</sub>        | 5.54 ( $\pm 1.68$ )  | 0.129 ( $\pm 0.027$ )   | 5.55 ( $\pm 1.83$ )   |
| Controller                      | 0.64 ( $\pm 0.00$ )  | —                       | 0.12 ( $\pm 0.00$ )   |
| <b>Approx. 24454 Parameters</b> |                      |                         |                       |
| DNFC                            | 1.74 ( $\pm 0.11$ )  | 0.061 ( $\pm 0.000$ )   | 1.76 ( $\pm 0.16$ )   |
| Baseline                        | 1.94 ( $\pm 0.14$ )  | 0.101 ( $\pm 0.006$ )   | 1.84 ( $\pm 0.10$ )   |
| DNFC <sub>ablation</sub>        | 6.47 ( $\pm 1.35$ )  | 0.131 ( $\pm 0.017$ )   | 7.04 ( $\pm 1.73$ )   |
| Controller                      | 0.64 ( $\pm 0.00$ )  | —                       | 0.12 ( $\pm 0.00$ )   |

## CHAPTER V

### CONCLUSIONS

In this thesis, a deep neural network (DNN) architecture for behavior cloning (BC) problems is introduced. We have named it the Deep Neural Feedback Controller (DNFC) because its architecture mimics that of a feedback controller. The DNFC structure consists of two streams. In one stream, a representation of the target (e.g., its Cartesian coordinates or image) is given to an encoder, which is a deep neural network (DNN) that generates its latent space representation. The current state of the robot is then subtracted from the target’s latent representation, yielding the *latent* error. Finally, another DNN generates torques for the robot’s joints based on this *latent* error.

To ensure that the latent representation of the target serves as a goal for the robot’s state, particularly at the conclusion of each episode, we employed a novel loss function. This function is based on the discrepancy between the target’s latent representation and the robot’s current state, multiplied by a factor that is an exponential function dependent on the timestep. This latent loss, combined with a straightforward torque loss, has been shown to be effective in training a model that functions as a non-linear feedback controller.

To assess the performance of our proposed model (DNFC), we systematically compared it to a controller-agnostic behavior cloning (BC) system, which is a simple DNN. Furthermore, to demonstrate the efficacy of the proposed loss function, we compared the DNFC to its latent loss-ablated version. We evaluated models at various levels of complexity, i.e., different numbers of learnable parameters, ensuring that models with similar complexity levels (i.e., approximately the same number of

learnable parameters) were compared to each other. For all cases, we limited the number of training epochs to the same relatively small number to assess the models under fair and resource-scarce conditions.

We conducted experiments in both static and dynamic target scenarios. In both cases, a PD controller was used to generate trajectories, which then served as ground truth data for training the models. In the static targets scenario, the target remains stationary during the execution of each episode, making it a simple *reaching* task. In the dynamic targets scenario, the target moves linearly during each episode, requiring the models to *track* the goal as it moves.

We found that for both static and dynamic target scenarios:

- Our proposed DNFC model outperforms the baseline at almost all levels of model complexity. This indicates that for a behavior cloning problem, where demonstration data is produced by a system that can be considered a non-linear feedback controller, a model structured to mimic a feedback controller is superior to a controller-agnostic baseline.
- There is a significant performance gap between the DNFC and its latent loss-ablated version, with the DNFC demonstrating superior performance. This highlights the effectiveness of the novel latent loss used in training our proposed model.
- The difference in performance between the DNFC and the baseline is more pronounced at lower levels of model complexity, especially at the lowest level. This suggests that under resource-scarce conditions, the DNFC is a better choice for tackling behavior cloning problems.
- The latent loss-ablated version of DNFC outperforms the baseline at the lowest model complexity level across all tasks and metrics. This suggests that even the

structure alone, without the additional benefit of the latent loss, is advantageous for behavior cloning problems under very limited resource conditions.

- As the model complexity increases, the performance gap between DNFC and the baseline decreases. This suggests that when there are no resource limitations, both models achieve nearly the same performance, with our proposed model performing slightly better.
- The performance of both DNFC and baseline models worsens when tested on the extrapolation set. However, at lower model complexity levels, DNFC demonstrates acceptable performance, while the baseline fails to perform well under these severely limited-resource conditions.

For dynamic target scenarios we found that:

- DNFC learns to track the target during the execution of episodes from the demonstrations of the ground truth controller. However, the baseline often fails to do this; it passes the target trajectory, overshoots, and often fails to recover, ending its movement far from the target’s endpoint.
- The latent representations of DNFC on test sets show that it approximately learns to map Cartesian coordinates to intrinsic coordinates.
- The baseline sometimes demonstrates unreasonable movements, such as moving in the opposite direction of the target trajectory. However, we did not observe such movements from DNFC, suggesting that it is more reliable and stable in this regard.
- While using latent loss in dynamic target scenarios may seem counterintuitive—since there isn’t a single goal state for the encoder module to identify from the Cartesian coordinates of the target—our findings indicate that latent loss significantly enhances the model’s performance. This can be rationalized

by considering that, in many training episodes, the target reaches its end-point; hence, ceases to move before the episode concludes. In these instances, the PD controller successfully reaches the target’s final position and remains there, generating minimal torque until the episode ends. The latent loss thus aids the encoder module in learning an approximate Cartesian-to-intrinsic mapping during these stationary phases by encouraging it to output values similar to the robot’s final state given the Cartesian coordinates of the target.

- It appears that increasing the complexity level of the ablated version of DNFC does not significantly improve its performance. This may be due to the limited number of training epochs conducted in the experiments, indicating that the ablated version of DNFC requires more training epochs to become an effective controller.

For static target scenarios we found that:

- When models are trained only on static target scenarios, DNFC can still demonstrate acceptable performance when tasked with tracking dynamic targets, even when they move along circular trajectories. This is because the model learns the correct transformation of Cartesian to intrinsic coordinates, enabling it to generalize to cases where the target’s position changes during the execution of an episode.
- DNFC has demonstrated that it can work with image representations of targets. Therefore, our proposed model can generalize to work with less informative, non-processed input, such as bitmap images, instead of more informative Cartesian coordinates.

Our proposed model has been trained and tested only in one environment (MuJoCo) and on a few tasks. Future work includes applying the DNFC model to a

broader range of learning from demonstration (LfD) tasks and investigating the end-to-end capability of the proposed model on more complex tasks, including dynamic ones.



## REFERENCES

- [1] S. Schaal, “Learning from demonstration,” *Advances in neural information processing systems*, vol. 9, 1996.
- [2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.
- [3] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, “Survey: Robot programming by demonstration,” *Springer handbook of robotics*, pp. 1371–1394, 2008.
- [4] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, “Recent advances in robot learning from demonstration,” *Annual review of control, robotics, and autonomous systems*, vol. 3, no. 1, pp. 297–330, 2020.
- [5] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, “Chomp: Covariant hamiltonian optimization for motion planning,” *The International journal of robotics research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.
- [6] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding locally optimal, collision-free trajectories with sequential convex optimization,” in *Robotics: science and systems*, vol. 9, pp. 1–10, Berlin, Germany, 2013.
- [7] J. Babič, J. G. Hale, and E. Oztop, “Human sensorimotor learning for humanoid robot skill synthesis,” *Adaptive Behavior*, vol. 19, no. 4, pp. 250–263, 2011.
- [8] L. Peternel, T. Petrič, E. Oztop, and J. Babič, “Teaching robots to cooperate with humans in dynamic manipulation tasks based on multi-modal human-in-the-loop approach,” *Autonomous robots*, vol. 36, pp. 123–136, 2014.
- [9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [10] J. Chang, M. Uehara, D. Sreenivas, R. Kidambi, and W. Sun, “Mitigating covariate shift in imitation learning via offline data with partial coverage,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 965–979, 2021.
- [11] B. Akbulut, T. Girgin, A. Mehrabi, M. Asada, E. Ugur, and E. Oztop, “Bimanual rope manipulation skill synthesis through context dependent correction policy learning from human demonstration,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3904–3910, 2023.
- [12] N. Amirshirzad, M. Asada, and E. Oztop, “Context based echo state networks for robot movement primitives,” in *2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pp. 1077–1082, IEEE, 2023.

- [13] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” in *Advances in Neural Information Processing Systems* (D. Touretzky, ed.), vol. 1, Morgan-Kaufmann, 1988.
- [14] C. G. Atkeson, J. G. Hale, F. Pollick, M. Riley, S. Kotosaka, S. Schaul, T. Shibata, G. Tevatia, A. Ude, S. Vijayakumar, *et al.*, “Using humanoid robots to study human behavior,” *IEEE Intelligent Systems and their applications*, vol. 15, no. 4, pp. 46–56, 2000.
- [15] E. Oztop, L.-H. Lin, M. Kawato, and G. Cheng, “Extensive human training for robot skill synthesis: Validation on a robotic hand,” in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1788–1793, IEEE, 2007.
- [16] J. Mathew and F. Crevecoeur, “Adaptive feedback control in human reaching adaptation to force fields,” *Frontiers in human neuroscience*, vol. 15, p. 742608, 2021.
- [17] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.
- [18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
- [20] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu, “Deep speech 2 : End-to-end speech recognition in english and mandarin,” in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 173–182, PMLR, 20–22 Jun 2016.
- [21] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter,

- C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.
- [22] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” *CoRR*, vol. abs/1810.04805, 2018.
- [23] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, L. de Paula Veronese, T. Oliveira-Santos, and A. F. De Souza, “Self-driving cars: A survey,” *Expert Systems with Applications*, vol. 165, p. 113816, 2021.
- [24] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems (8th Edition) (What’s New in Engineering)*. Pearson, 2018.
- [25] A. G. Billard, S. Calinon, and R. Dillmann, “Learning from humans,” *Springer handbook of robotics*, pp. 1995–2014, 2016.
- [26] S. R. Ahmadzadeh, R. Kaushik, and S. Chernova, “Trajectory learning from demonstration with canal surfaces: A parameter-free approach,” in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 544–549, IEEE, 2016.
- [27] G. J. Maeda, G. Neumann, M. Ewerton, R. Lioutikov, O. Kroemer, and J. Peters, “Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks,” *Autonomous Robots*, vol. 41, pp. 593–612, 2017.
- [28] P. Abbeel, A. Coates, and A. Y. Ng, “Autonomous helicopter aerobatics through apprenticeship learning,” *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.
- [29] R. Peters, C. Campbell, W. Bluethmann, and E. Huber, “Robonaut task learning through teleoperation,” in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 2, pp. 2806–2811 vol.2, 2003.
- [30] I. Havoutis and S. Calinon, “Learning from demonstration for semi-autonomous teleoperation,” *Autonomous Robots*, vol. 43, pp. 713–726, 2019.
- [31] J. Kaiser, S. Melbaum, J. C. V. Tieck, A. Roennau, M. V. Butz, and R. Dillmann, “Learning to reproduce visually similar movements by minimizing event-based prediction error,” in *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*, pp. 260–267, 2018.
- [32] R. Dillmann, “Teaching and learning of robot tasks via observation of human performance,” *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 109–116, 2004.

- [33] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, “End-to-end driving via conditional imitation learning,” in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4693–4700, IEEE, 2018.
- [34] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, “Dynamical movement primitives: Learning attractor models for motor behaviors,” *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.
- [35] A. Pervez and D. Lee, “Learning task-parameterized dynamic movement primitives using mixture of gmms,” *Intelligent Service Robotics*, vol. 11, no. 1, pp. 61–78, 2018.
- [36] S. Calinon, “A tutorial on task-parameterized movement learning and retrieval,” *Intelligent service robotics*, vol. 9, pp. 1–29, 2016.
- [37] S. M. Khansari-Zadeh and A. Billard, “Learning stable nonlinear dynamical systems with gaussian mixture models,” *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.
- [38] G. E. Hovland, P. Sikka, and B. J. McCarragher, “Skill acquisition from human demonstration using a hidden markov model,” in *Proceedings of IEEE international conference on robotics and automation*, vol. 3, pp. 2706–2711, Ieee, 1996.
- [39] A. Pervez, Y. Mao, and D. Lee, “Learning deep movement primitives using convolutional neural networks,” in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 191–197, 2017.
- [40] J. Schulman, J. Ho, C. Lee, and P. Abbeel, “Learning from demonstrations through the use of non-rigid registration,” in *Robotics Research: The 16th International Symposium ISRR*, pp. 339–354, Springer, 2016.
- [41] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, “One-shot visual imitation learning via meta-learning,” in *Conference on robot learning*, pp. 357–368, PMLR, 2017.
- [42] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber, “A system for robotic heart surgery that learns to tie knots using recurrent neural networks,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 543–548, 2006.
- [43] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine, “Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration,” in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3758–3765, IEEE, 2018.
- [44] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection,” *The International journal of robotics research*, vol. 37, no. 4-5, pp. 421–436, 2018.

- [45] K. Strabala, M. K. Lee, A. Dragan, J. Forlizzi, S. S. Srinivasa, M. Cakmak, and V. Micelli, “Toward seamless human-robot handovers,” *Journal of Human-Robot Interaction*, vol. 2, no. 1, pp. 112–132, 2013.
- [46] M. Deniša, A. Gams, A. Ude, and T. Petrič, “Learning compliant movement primitives through demonstration and statistical generalization,” *IEEE/ASME transactions on mechatronics*, vol. 21, no. 5, pp. 2581–2594, 2015.
- [47] S. Chernova and A. L. Thomaz, *Robot learning from human teachers*. Morgan & Claypool Publishers, 2014.
- [48] Z. Zhang, G. Peng, W. Wang, Y. Chen, Y. Jia, and S. Liu, “Prediction-based human-robot collaboration in assembly tasks using a learning from demonstration model,” *Sensors*, vol. 22, no. 11, p. 4279, 2022.
- [49] E. Matheson, R. Minto, E. G. Zampieri, M. Faccio, and G. Rosati, “Human–robot collaboration in manufacturing applications: A review,” *Robotics*, vol. 8, no. 4, p. 100, 2019.
- [50] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, “Learning from demonstration and adaptation of biped locomotion,” *Robotics and autonomous systems*, vol. 47, no. 2-3, pp. 79–91, 2004.
- [51] R. Calandra, N. Gopalan, A. Seyfarth, J. Peters, and M. P. Deisenroth, “Bayesian gait optimization for bipedal locomotion,” in *Learning and Intelligent Optimization: 8th International Conference, Lion 8, Gainesville, FL, USA, February 16-21, 2014. Revised Selected Papers 8*, pp. 274–290, Springer, 2014.
- [52] D. A. Pomerleau, “Efficient training of artificial neural networks for autonomous navigation,” *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.
- [53] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, “Agile autonomous driving using end-to-end deep imitation learning,” *arXiv preprint arXiv:1709.07174*, 2017.
- [54] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, “Learning monocular reactive uav control in cluttered natural environments,” in *2013 IEEE international conference on robotics and automation*, pp. 1765–1772, IEEE, 2013.
- [55] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 627–635, PMLR, 11–13 Apr 2011.
- [56] D. Seita, A. Ganapathi, R. Hoque, M. Hwang, E. Cen, A. K. Tanwani, A. Balakrishna, B. Thananjeyan, J. Ichnowski, N. Jamali, K. Yamane, S. Iba, J. Canny,

- and K. Goldberg, “Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
- [57] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine, “Combining self-supervised learning and imitation for vision-based rope manipulation,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2146–2153, 2017.
  - [58] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, “Neural network optimal feedback control with enhanced closed loop stability,” in *2022 American Control Conference (ACC)*, pp. 2373–2378, IEEE, 2022.
  - [59] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, “Qrnet: Optimal regulator design with lqr-augmented neural networks,” *IEEE Control Systems Letters*, vol. 5, no. 4, pp. 1303–1308, 2020.
  - [60] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.
  - [61] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
  - [62] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, “Gymnasium,” Mar. 2023.
  - [63] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.
  - [64] A. Adamczak, “Mujoco-reacher-control-system.” <https://github.com/AdamAdamczak/Mujoco-Reacher-Control-System>, 2023.