EXPLORATIONS ON INVERSE REINFORCEMENT LEARNING FOR THE ANALYSIS OF MOTOR CONTROL AND COGNITIVE DECISION MAKING MECHANISMS OF THE BRAIN

A Thesis

by

Emir Arditi

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 Computer Science

Özyeğin University
December 2021

EXPLORATIONS ON INVERSE REINFORCEMENT LEARNING FOR THE ANALYSIS OF MOTOR CONTROL AND COGNITIVE DECISION MAKING MECHANISMS OF THE BRAIN

Approved by:

Prof. Dr. Erhan Öztop, Advisor
Department of Computer Science
Özyeğin University

Asst. Prof. Dr. Reyhan Aydoğan
Department of Computer Science
Özyeğin University

Assoc. Prof. Dr. Emre Uğur
Department of Computer Science
Boğaziçi University

Date Approved: 29 December 2021

To my family, my friends and my future Liko

ABSTRACT

Reinforcement Learning is a framework for generating optimal policies given a task and a reward/punishment structure. Likewise, Inverse Reinforcement Learning, as the name suggests, is used for recovering the reasoning behind an optimal policy based on demonstrations from an expert. We set out to explore whether recent Reinforcement Learning and Inverse Reinforcement Learning methods can serve as a computational tool for investigating optimality principles of motor control and cognitive decision-making mechanisms of the brain. For this purpose, we have targeted several different tasks involved with different parts of the sensorimotor learning mechanism of the brain. We aim to recover the optimality principles employed by the brain for various control and decision-making tasks. If this is achieved, we can analyze, understand, mimic and improve demonstrated behavior with less bias, which we hope is a step forward in understanding the process of learning in both human-based and artificial systems.

For the scope of this thesis, we have evaluated two tasks. The first task was investigating the applicability of perceptual development for Reinforcement Learning. For this task, we have proposed a perceptual development based learning regime for a Reinforcement Learning agent, and the results obtained suggest that a suitable perceptual development regime may improve the learning progress and yield better-performing agents. The second task was to predict reward function parameters of a provided trajectory in a standing up under perturbation scenario. For this task, we have proposed two different Inverse Reinforcement Learning approaches. Our results indicate that we were able to infer valid reward parameters on synthetic data.

ÖZETÇE

Pekiştirmeli öğrenme, farklı ortamlarda, verilen ödül ceza yapısına göre en uygun politikaları bulma sistemidir. Benzer şekilde, Tersine Pekiştirmeli Öğrenme de, adından anlaşılabileceği gibi, bir uzmandan alınan en uygun politikanın arkasındaki sebepleri bulmak için kullanılır. Bu araştırmada, güncel Pekiştirmeli Öğrenme ve Tersine Pekiştirmeli Öğrenme metotlarının, beynin motor kontrol ve bilişsel karar alma mekanizmalarının arkasındaki eniyileme prensiplerini modelleyen araçlar olarak kullanılabilmesini keşfetmeyi amaçlıyoruz. Bu amaç için, beynin farklı duyusal motor özelliklerini hedefleyen farklı görevleri hedefledik. Niyetimiz, beyin tarafından farklı alanlar için oluşturulan en iyileme kriterlerini keşfedebilmek. Bu başarılabildiği takdirde, varolan veya yeni bir metot ile, insan davranışlarını daha düşük bir yanlılık ile analiz edebilir, anlayabilir ve taklit edebiliriz.

Bu tezin kapsamı doğrultusunda, iki tane görevi hedefledik. İlk görev, algısal gelişimin Pekiştirmeli Öğrenme'ye uygulanabilirliğinin araştırılmasıdır. Bu görev için, bir Pekiştirmeli Öğrenme ajanı, kendi önerdiğimiz bir algısal gelişim tabanlı gelişimsel rejim ile eğittik. Sonuçlarımız, uygun bir algısal gelişim rejiminin, Pekiştirmeli Öğrenme'nin öğrenme ilerlemesini geliştirebileceğini ve daha iyi ajanlar üretebileceğini önerdi. İkinci görev ise, Tersine Pekiştirmeli Öğrenme ile, uzmanların ödül fonksiyonu parametrelerini keşfetmekti. Bunun için, iki tane farklı Tersine Pekiştirmeli Öğrenme mekanizması oluşturduk ve sonuçlarımız geçerli ödül fonksiyonu parametreleri keşfettiğimizi önermektedir.

ACKNOWLEDGEMENTS

First of all, I would like to start by expressing my gratitude towards my advisor Prof. Dr. Erhan Öztop for his continuous support during my M.Sc. His leadership, motivation, immense knowledge, and especially patience allowed me to push myself towards my goals.

Besides my advisor, I would like to thank the rest of my thesis committee: Asst. Prof. Dr. Reyhan Aydoğan and Assoc. Prof. Dr. Emre Uğur, for their encouragement not only regarding my thesis, but also throughout this M.Sc. study.

I would like to thank my fellows in Artificial Intelligence Lab (AILab) at Özyeğin University: Cihan Eran, Umut Çakan, Onur Keskin, Negin Amirshirzad, Begüm Sunal; and in Video, Vision and Graphics Lab (VVGL) at Özyeğin University: Barış Özcan and Furkan Kınlı. I couldn't have imagined having better colleagues, mentors, and friends alongside with me during this journey.

I would also like to thank my mother Rakel, my father Leon, my grandmother Şirin, and my brother Emre for their unconditional support through every journey I take. I am aware that none of this would have been possible without any of them.

Finally to İlkim, who, for the last 6 years, has never left me alone in any journey I took. Without her endless support and patience, I would not have had the courage to embark on, or conclude this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Figuring out the learning and decision making mechanisms of the brain have always been a widely investigated task throughout history. It is a task studied in multiple different disciplines including philosophy [1], psychology [2], medicine [3] and more recently, computer science [4]. Various propositions were made to uncover these mechanisms, some were accepted and some were denied over time. Despite all these advancements, the mechanisms behind these processes are still widely unclear.

One theory standing out over the last decade is Reinforcement Learning. The history of Reinforcement Learning consists of two different branches, and both of these branches are investigated independently. Afterward, these branches were combined and formed the modern Reinforcement Learning principles used today. The first branch concerned learning by trial and error and was initially proposed in the form of the psychology of animal learning. This branch led to some of the earliest work in AI. The second branch concerns the problem of optimal control, and the solution methodologies mainly consist of value functions and dynamic programming. This branch primarily did not involve the process of learning. After some point, these two branches started to combine to form the modern Reinforcement Learning principles still being used today.

Another essential concept that focuses on discovering how humans learn or act is Inverse Reinforcement Learning. In Reinforcement Learning, the objective is to imitate **how** humans learn, whereas, in Inverse Reinforcement Learning, the objective is to uncover **why** an expert performs an action. Even though the research on Inverse Reinforcement Learning does not have as much multi-disciplinary history as Reinforcement Learning; modern Inverse Reinforcement Learning models

form a multi-disciplinary field including neuroscience, mathematics, and cognitive science.

1.1 Motivation

Using both of these concepts, in this thesis, we propose novel frameworks to explore how humans learn and how the cognitive decision-making mechanisms of the brain work. The first part of this thesis proposes a Perceptual Development Imitation regime for Reinforcement Learning to investigate the effect of an infant-based development scenario. The infant-based development is modeled as both a developmental enhancement and observational noise.

The second part of this thesis proposes two Inverse Reinforcement Learning frameworks to discover the reward function parameter of subjects performing a squat-to-stand action. The first framework is a brute-force-based approach, whereas the second one initializes Conditional Neural Movement Primitives and Deep Neural Networks.

1.2 Thesis Outline

This thesis is organized as follows. Chapter 2 gives background information regarding the concepts in the scope of the following chapters. Chapter 3 describes previous works in the literature of Developmental Learning, Imitation Learning, and Inverse Reinforcement Learning. Afterward, in Chapter 4, our work regarding Perceptional Development Imitation is presented. Later, Chapter 5 describes the methods behind our proposed IRL framework and later presents the results of our experiments. Finally, in Chapter 6, final thoughts and the future directions of these works are presented.

CHAPTER II

BACKGROUND INFORMATION

2.1 Introduction

Artificial Neural Networks(ANN) are commonly used tools in modern RL. To understand the underlying principles behind modern RL, one must first understand the principles behind ANNs. In this chapter, first, the concept of neural networks will be presented. Later, RL and IRL concepts will be introduced and the usage of ANNs in these concepts will be explained.

2.2 Artificial Neural Networks

The idea of creating an Artificial Neural Network similar to the biological brain was first proposed by McCulloh and Pitts in the early 1940s [5]. In this work, they modeled the biological brain as a logical calculus and proposed a threshold logic. Afterwards, Kelley proposed backpropagation [6], an algorithm which computes the gradient in weight space of a feed forward NN w.r.t. a loss function in order to update the weights of the network. Most of the modern ANNs still use this algorithm for weight updates.

2.2.1 ANN Structure

ANNs can also be stated as function approximators. A simple ANN consists of an input layer, output layer and H amount of hidden layers. When $H > 1$, the networks are referred as Deep Neural Networks(DNN). In traditional DNNs, every neuron in every layer is connected to every neuron in the next layer. Increasing the depth and height of networks allow networks to model an learn a more complex space, whilst inducing harder convergence. An example representation of a simple ANN can be found in Figure 1.

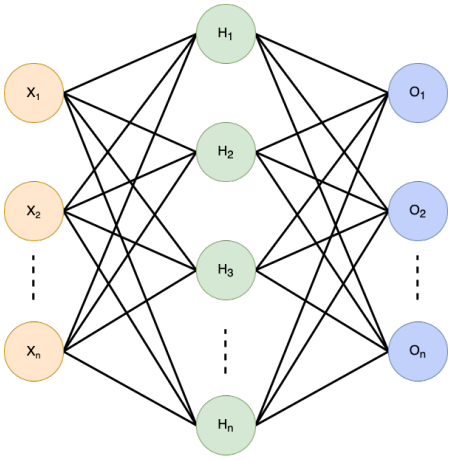

Figure 1: The structure of a Simple ANN

2.2.2 Neurons

Neuron is the smallest component of an ANN. They represent basic mathematical functions. An arbitrary number of inputs are provided to the neuron and the neuron outputs arbitrary amount of values to arbitrary amount of neurons. In a simple ANN, the feed-forward mechanism usually follows neurons passing information to next layers. The outputted value of each neuron is multiplied by a unique weight between every neuron. Simple weight representation can be found in Figure 1. The output of a single neuron can be formulated as follows:

$$h_{(j)} = \sum_{i=1}^{n} (\omega_{ji} * x_i) + b_j \tag{1}$$

where h_j, n, ω_{ji}, x_i, b represents the output of layer j, # of neurons in layer j, weight between the current neuron and the This structure allows NNs to learn linearly solvable tasks, but fails to perform in non-linear tasks. To resolve this problem, activation functions were proposed.

2.2.3 Activation Functions

Activation functions are used to introduce non-linearity to an ANN. This non-linearity is introduced to allow ANNs to model complex representations. There are several popular activation functions used in the modern literature. These functions can be found in Figure 2.

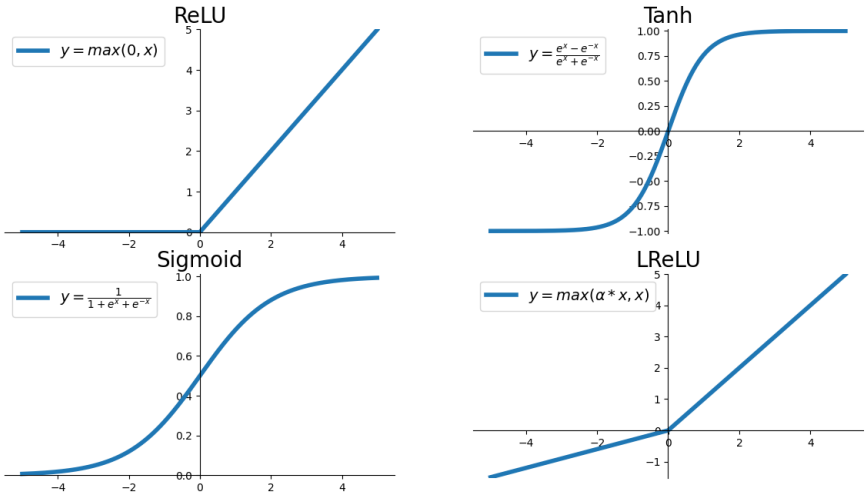

Figure 2: Representations of popular activation functions

Each of these activation functions have different strengths, weaknesses and purposes. For example, sigmoid is a popular function for the output layer of regression based models due to its constrained nature. When we include these activation functions to the feed-forward model, the equation described above converts to:

$$h_{(j)} = \sigma(\sum_{i=1}^{n} (\omega_{ji} * x_i) + b_j) \tag{2}$$

2.2.4 Gradient Descent

Gradient descent is an iterative differentiation algorithm to minimize the error calculated by the loss function[7]. In each iteration, weights are updated in steps determined by the learning rate according to the inverse of the gradients calculated. There are several ways to conduct gradient descent. The first is called batch gradient descent. In batch gradient descent, the errors are calculated based on the error on the complete data set and the weights are updated accordingly. Secondly, in step gradient descent, the network is updated based on the errors of each individual data in the data set. Another common approach is to use stochastic gradient [8] descent in which the errors are calculated on mini-batches of the

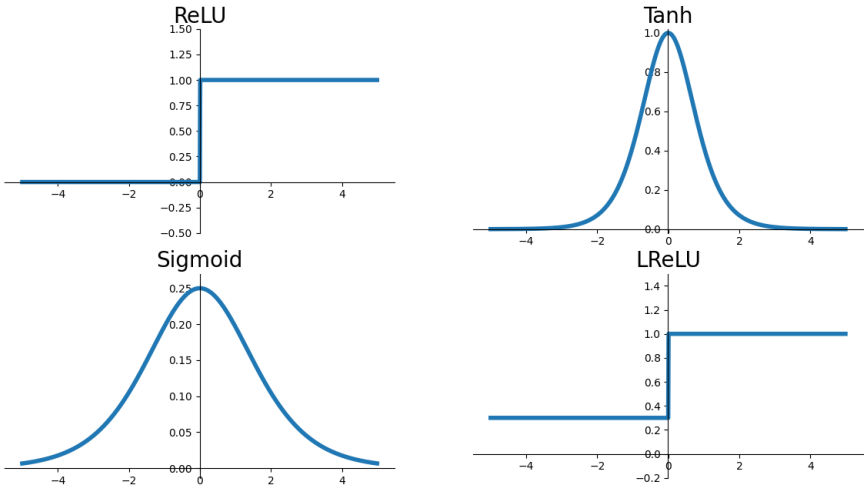

Figure 3: Representations of derivatives of popular activation functions

data. This is the most commonly used gradient descent approach in the modern literature. The backpropagation chain is constructed based on the connectivity of the neurons. While calculating the derivative of the error, the derivative of activation functions should be used. Representations of the derivatives of activation functions can be found in Figure 3. 3

2.3 Reinforcement Learning

In general, RL is a method to find how to behave optimally in environments that can be modeled within the Markov Decision Process (MDP) framework [9]. An MDP is defined by a tuple ($MDP \sim \langle S, A, P, R \rangle$) where S, A represents the state and action spaces; P represents the state transition probabilities $P(s|a)$, and finally R represents the reward function/probability distribution for actions taken.

The most commonly used RL systems can be grouped as either State/Action Value-Based or Policy Approximation based methods in current literature.

2.3.1 State and State/Action Value-Based RL

The main goal behind State and State/Action Value-Based RL is to calculate the *correct* value when a state and/or an action are given. The value is modeled as

$V(s)$ and is defined as the maximum possible reward that can be achieved from state s to the terminal state. This can be summarized with the following equation:

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] \tag{3}$$

where $V, \pi, E_\pi, R_t, , s$ represents the value function, policy, expected value according to the currency policy, reward function, discount factor and the state respectively.

In setups where the environment dynamics are known, and the state space is not too complex, algorithms like *value iteration* [10] or *policy iteration* [11] can be employed to find the optimal value function V^*. The implementation details of value function is provided on Algorithm 1.

Algorithm 1 Value Iteration

Initialize array v arbitrarily(e.g. $v(s) = 0$ for all s)
θ = threshold
while $\delta < \theta$ **do**
  $\delta = 0$
  for each $s \in S$ **do**
    $temp = V(s)$
    $v(s) = max_a \sum_{s'} p(s^{'}|s,a)[r(s,a,s^{'}) + \gamma v(s^{'})]$
  end for
end while
Output a deterministic policy π such that
$\pi(s) = argmax_a \sum_{s'} p(s^{'}|s,a)[r(s,a,s^{'}) + \gamma v(s^{'})]$

Another widely used function in RL is the Q-function $Q(s,a)$. The most crucial difference between $Q(s,a)$ and $V(s)$ is that $Q(s,a)$ not only depends on the state but also the action taken. This allows the system to discriminate the effectiveness of an action taken. The most common algorithm group for modeling $V(s)$ or $Q(s,a)$ are called Tabular Solution algorithms. Tabular algorithms include Dynamic Programming, Monte Carlo Methods and Temporal Difference Learning(TD). In literature, especially where the model is unknown, Temporal Difference Learning is the most commonly used algorithm group.

Temporal difference can be defined as the difference between the value of the

following state and the current state. The first TD-based-RL algorithm was proposed by Sutton [4] and by using Monte Carlo techniques, it aimed to extract the policy using the flow of events instead of a model-based approach. Later, Q-Learning [12] and SARSA [13] were proposed. These algorithms proposed to model the connection between the state space and the action space using a table, while the main goal is to do repetitive trials on the environment to update the Q values on the table and reach an optimal policy. The most important difference between Q-Learning and SARSA is that Q-Learning is an off-policy algorithm while SARSA is an on-policy algorithm. While the Q-value updates are proposed, on-policy algorithms stay faithful to the current action selection method and the system's current state, while off-policy algorithms conduct updates using the highest Q value for the current state.

The most considerable limitation of the Table-Based TD-Learning is the restrictions imposed by the usage of a table. These restrictions expect the state space to be discrete and require the values of all state-action pairs to be computed. In an attempt to resolve these restrictions, Mnih et al. [14] proposed Deep Q-Learning(DQN) to replace the table with a neural network that conducts function approximation. DQN is also crucial milestone in RL since it is the first converging NN-Based RL method. Since neural networks are data and repetition driven, to converge the system, Mnih used a technique entitled Experience Replay. During experience replay, instead of conducting the updates real-time, after each action, state s, action a, next state s' and reward r is stored in a list, and after each t steps, the neural network is updated using K random elements from the experiences stored.

According to Mnih, the limitation of DQN is that since Q-values are rapidly changing, the updates become unstable. To resolve this, Mnih et al. proposed DQN with Target Networks[15] which introduced the addition of a *target network* to the DQN setup. The main network is updated after each T steps in this structure, but the Q values are taken from the target network. After each C

Algorithm 2 DQN with Target Networks

```
Generate Memory D with N capacity
With random weights θ, generate action-value function Q
Q⁻ = Q, with Q⁻, generate action-value function Q̂
for episode = 1 to M do
  for t = 1 to T do
    with ϵ probability, pick a random action and assing a_t
    else, a_t = argmax_a Q(s_t, a; θ)
    Take action a_t, observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) on D
    s_t = s_{t+1}
    Sample random minibatch of transitions from D
    if s_{t+1} is the terminal state then
      y_t = r_t
    else
      y_t = r_t + γ max_{a'} Q̂(s_t, a_t; θ⁻)
    end if
    Perform gradient descent over (y_t − Q(s_t, a_t, θ))² on network parameters θ
    Every C step, θ⁻ = θ
  end for
end for
```

steps, the weights of the target network are equated with the main network. The detailed algorithm can be found on Algorithm 2.

TD-Based Deep Reinforcement Learning methods are groundbreaking in terms of the applicability of RL to real-life tasks. The primary restriction of these algorithms was that they were only applicable to discrete state spaces.

2.3.2 Policy-Based Reinforcement Learning Methods

The most common technique for Policy-Based RL is Policy Gradient(PG) methods. As the name suggests, instead of computing the value, these approaches try to directly model and learn the policy itself. The policy is commonly denoted as $\pi_\theta(a|s)$. This modeling allows policy gradient-based methods to work on environments with continuous state space.

One of the oldest examples of PG methods is REINFORCE [16]. REINFORCE updates the policy parameters based on estimated returns from Monte-Carlo methods using episode data. The concept behind this algorithm is validated because the expected sample gradient from the Jacobian matrix is equal to the actual

gradient.

Another popular PG-Based RL method is called Actor-Critic(AC) algorithms. The most significant difference between AC algorithms and REINFORCE is that AC algorithms model the value space alongside the policy space. The reason behind modeling the value space is to better approximate policy parameters. Depending on the algorithm used, the critic network is responsible for updating the w value for the action-value function $Q_w(a|s)$ or the state-value function $V_w(s)$ whilst according the the value samples of the critic network, actor updates the θ parameter of the policy $\pi_\theta(a|s)$.

Over time, studies regarding applying Actor-Critic with neural networks are conducted. An example of these approaches is Deep Deterministic Policy Gradients (DDPG) [17]. DDPG is a model-free, off-policy algorithm that combines DQN and deterministic policy gradients. Unlike DQN, DDPG can be used on continuous state spaces using the actor-critic scheme. Also, DDPG conducts soft updates on the target network rather than hard updates. During soft update, instead of directly equating the target network, a stepwise update is conducted using a τ parameter. The formula for this update is as follows:

$$\theta^{'} = \tau\theta + (1 - \tau)\theta^{'} \tag{4}$$

Soft update aims to tackle the instability on the update gradients caused by directly equating θ and $\theta^{'}$.

Another commonly used Policy-Gradient Based RL method is Proximal Policy Optimizations (PPO) [18]. PPO aims to solve environments with multivariate and continuous state and action spaces. The algorithm for PPO is provided on Algorithm 3.

2.4 Inverse Reinforcement Learning

The purpose of Inverse Reinforcement Learning(IRL) is to extract the reward function $R(s)$ from observations drawn from a policy, $\pi(s)$. IRL can be used in various fields, including autonomous driving and robot navigation.

Algorithm 3 PPO Algorithm

K = repetition count
for iteration = 1,2... **do**
  Conduct the old policy π_θ for T steps.
  Using the estimations from critic network and samples from policy π_θ calculate Advantage Estimations $\hat{A}_1 ... \hat{A}_T$
  Perform gradient descent over $(y_t - Q(s_t, a_t, \theta))^2$ on network parameters θ
  for repetition = 1 **to** K **do**
    Update the loss function of the actor using θ
    Update the critic network according to the newly formed π_θ
  end for
  $\theta_{old} \leftarrow \theta$
end for

Discovering the reward function of a human or a setup is a widely tackled subject in literature. One of the earliest works in this subject dates back to 1964 [19]. In this work, Kalman forms a linear control system in a 1D space and aims to find the return values that explain the system best. In another study, Boyd successfully solves an IRL system on a 2D space using the characteristics of linear matrix inequality [20]. Later, Ng and Russell put forth the concept of reward uncertainty which potentially harms the IRL system, and proposed solutions based on MDPs [21]. Afterward, Abbeel and Ng proposed Apprenticeship Learning and revealed that IRL systems could generate a policy alongside a reward function and surpass the expert's performance based on the generated solution space of the policy.

2.4.1 The Difference Between IRL and Behavior Cloning

In a setup where transition model $P_{sa}(s_{t+1}|s_t, a_t)$ and samples $[(s_0, a_0), (s_1, a_1)...]$ from a policy of an expert π^* is given, the difference between IRL and Behavioral Cloning(BC), is clearly observable. For this case, while IRL focuses on extracting the reward function R, BC uses supervised learning to directly extract the policy of the expert π^*.

During BC, approaches like neural networks, decision trees, support vector machines, and other ML algorithms are used to reveal the mappings from the state space to action space using gradients. In IRL, to find the policy, methods

like Apprenticeship Learning [22] are used alongside the extracted reward function R.

2.4.2 Formulation of IRL

The main principle behind the problem of IRL is to discover the reward function which explains the subject behavior the best. The formulation of this problem is as follows:

$$E[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi^*] \geq E[\sum_{t=0}^{\infty} \gamma^t R^*(s_t)|\pi] \forall \pi \tag{5}$$

This equation states that the reward expectation of the optimal policy in the possible state space must be greater than or equal to the reward expectation in the possible state space of the given policy examples. This inequality is formed because IRL assumes that the provided policy is always optimal.

Even though this is a convex feasibility problem, there are possible constraints. R=0 is a solution, and this causes reward uncertainty. Reward uncertainty makes it harder to find a closed-form solution. Another constraint is that IRL systems usually only have access to a set of samples from the expert, and that is why it is not possible to sample on top of examples outside these samples(for the left side of the equation). It is also unclear whether the expert acts with an optimal policy based on the provided environment.

Another common representation in IRL represents the reward function not from the state itself, but features extracted from the state. When this is applied, the initial equation becomes:

$$R(s) = \omega^T \phi(s), \omega \in R^n, \phi : S \rightarrow R^n \tag{6}$$

$$\omega^T \mu(\pi^*) \geq \omega^T \mu(\pi) \forall \pi \tag{7}$$

Where $\mu(\pi)$ is the expected cumulative discounted sum of the feature expectations. The aim is to find an ω value that satisfies this inequality.

To prevent $\omega = 0$ from becoming a solution, a concept called max-margin is proposed. Max margin adds a +1 to the right side of the equation. This also

assists the system with a higher value difference feedback when constructed policy and optimal policy drifts apart. A more complex approach built towards the max-margin is called structured prediction. Structured prediction converts the inequality to the following:

$$\omega^T \mu(\pi^*) \geq \omega^T \mu(\pi^) + m(\pi^*, \pi) \in \pi \tag{8}$$

where $m(\pi^*, \pi)$ represents the number of times the constructed policy and optimal policy acted differently.

As mentioned above, since the expert policy is not guaranteed to be optimal, the current form of max-margin cannot be served as a viable IRL mechanism. In an attempt to solve the aforementioned problem, slack variables, are introduced to the max margin formulation. Slack variables allow the system to generalize on multiple MDPs and is formulated as follows:

$$\min_{\omega} \|\omega_2^2\| + c \sum_i \xi^{(i)} \tag{9}$$

s.t.

$$\omega^T \mu(\pi^{(i)*}) \geq \omega^T \mu(\pi^{(i)}) + m(\pi^{(i)*}, \pi^{(i)}) - \xi^{(i)} \in, \pi^{(i)} \tag{10}$$

Another important foundation of IRL is feature matching. Abbeel states that, for any policy to be as successful as the expert policy, the features must match or at least partially match [22]. This statement can be formulized as follows:

$$\|\mu(\pi) - \mu(\pi^*)\|_1 \leq \epsilon \tag{11}$$

Apprenticeship Learning was successful in finding close-to-optimal policies by implementing the aforementioned additions.

CHAPTER III

LITERATURE REVIEW

3.1 Imitating Human-Like Developmental Scenarios for Reinforcement Learning

Since it is impossible to separate a changing sensorimotor ability and learning from scaffolding provided by the engineer in engineered learning scenarios, we review works that modify the sensorimotor processing or the learning regime together. It is known that infant skill development is accelerated by a range of scaffolding behaviors offered by the caregiver (see, e.g., [23, 24]). In educational sciences, the concept of scaffolding is used as an essential guide for effective teaching (instructional scaffolding [25]). The non-stationarity of the sensorimotor system of the infants, either induced by scaffolding or by perceptual development, cannot be modeled using the standard MDP framework, although some attempts in line with such non-standard considerations exist. For example, Uchibe et al. [26] proposed to create a helpful state via continual supervised learning within an RL framework to address somewhat non-stationary environments. However, the agent was assumed to have access to the entire environment observation. Introducing explicit scaffolding to supervised learning has been proposed by Elman et al. [27] in a study that investigates whether neural networks can solve otherwise unsolvable problems by starting with a small subset of the problem. Scaffolding ideas are also used by Nagai et al. [28] to enhance decision making, where the quality of the observation sharpens over time. With an instructional scaffolding inspiration Zimmer et al. [29], Fachantidis et al. [30], and Celik et al. [31] use teacher-student models where a teacher network improves the training performance of the student network by generating more complex samples for the targeted set over time. Also, Suay et al. [32] investigates the effects of the observation space size in an interactive RL scenario where a human guides the RL system through the process. This

approach is similar to ours in the sense that observation space is modified but different because this approach aims to limit the observation space using a guided feedback mechanism, in which the observation is still given entirely to the system, but the space is less explored. In our approach, the observation space is altered by directly constraining some attributes and revealing them to the RL mechanism over time.

From a broader perspective, the so-called Curriculum Learning (CL) [33] in neural networks can be considered as addressing some components of perceptual development and scaffolding. In usual CL, the examples fed to the neural network are not sampled randomly but selected from a predetermined regime to enhance learning. Several studies show this is effective in DRL. For example, Matiisen et al.[34] initializes two different networks, one for solving the task and the other for simplifying the task as needed, which is reported to yield successful results on Minecraft maze task. Rusu et al. [35] proposed a skill transfer method for DRL, where after a network is acquired, the skill for *Task A*, it is augmented with additional resources to continue training with a new task, *Task B*. This method can be considered a form CL since multiple tasks can be the different complexity level versions of the same task. These approaches either keep a stationary MDP, model the system using transfer learning, or initiate multiple networks, thereby creating a Mixture of Experts[36] scenario.

3.2 Inverse Reinforcement Learning

IRL can be adopted for Imitation Learning (IM) or understanding the principles behind an observed behavior. For the former case, the focus is on finding a suitable reward function that would effectively elicit an action policy that will lead to a 'useful' imitation, often a policy that yields actions similar to the demonstration. This is best done by directly searching/learning a policy to generate a behavior similar to the observed one. It is generally accepted that such approaches [37, 38, 39] yield better imitation capability compared to the standard behavior cloning

[40, 41], i.e., directly learning the state → action mapping from the observation.

In the latter case, the focus tends to be more on finding an explicit and often human interpretable reward function that would allow imitation and explain the observed action and thus provide insights into the observed behavior. For finding an explicit reward function, a straightforward approach is to employ an RL solver inside a parameter search loop to form candidate policies, which are then used to assess the similarity of the observed behavior to the ones dictated by the candidate policies [42, 43, 44]. Once the reward function is found, it is possible to obtain a policy to imitate the observed action by RL. However, beyond this, an accurate reward function estimation means that the optimality principle employed by the agent may be uncovered. This can be a powerful tool when the demonstrating agent is a biological system, e.g., a human, as it facilitates how the brain mechanisms of action generation and problem-solving work. Although this holds a great promise, it should be taken cautiously since IRL is an ill-posed problem.

To overcome the ill-posed nature of IRL, the approaches mentioned above impose constraints on the solution space by requiring certain assumptions to hold. For RL-based IRL methods (such as [42, 43, 44]), another strong assumption is that the inner RL solver always converges to a single optimal policy concerning the current candidate reward function. This assumption holds in discrete-state, discrete-action toy problems where dynamic programming (DP) can be adopted as the RL solver. However, this is not the case in many realistic settings, for example, when a continuous-state and continuous-action RL solver (such as DDPG or PPO) is needed, limiting the validity of the assumption.

In particular, it is not clear how to construct an IRL method by using a direct policy learner in the reward parameter search loop. Further, it is of interest to know the applicability of such IRL algorithms for analyzing human sensorimotor data. In the literature, IRL methods that do not require an RL solver, which can be applied to continuous-state, continuous-action tasks, are scarce and come with additional complexities and assumptions limiting their applicability to real-world

problems. For example, the method of [45] is a local method and requires the computation of the derivative of the transition dynamics of the system, and the method of [46] requires the system to be modeled as a Linear Markov Decision Process. Due to the constraints earlier, the use of IRL for analyzing experimental data from complex biological systems is limited. The ones that exist often revert to discretization or other simplifications. For example, [47] studied the behavior of C. elegans worm by adopting the latter method; but, they had to represent the state space of the worm as a 2D mesh of temperature and temperature gradients.

CHAPTER IV

IMITATING PERCEPTUAL DEVELOPMENT VIA RL

This chapter covers the details of the research entitled Imitating Perceptual Development via RL. This research aims to investigate the effect of perceptual development emulation to increase the effectiveness of RL systems. The chapter first describes the environments and methods applied. Later, results are provided, and finally, conclusions and discussions are driven based on the results.

4.1 Environment Setup

The environment selected for this research is a modified version of the famous game Pong. Pong is a simple game composed of two paddles and a single ball. The players can take *Up*, *Down*, or *Stay* actions to control their paddle. In a usual game of Pong, the opponent's strategy is position invariant. To obtain richer gameplay, we modified the original game and programmed the opponent to act differently according to its position. If the opponent is at the lower half of the game area, it tries to align its paddle center position with the ball vertical position (see Figure 4). On the contrary, if the opponent is at the upper half of the frame, it tries to keep the ball aligned with the center of the upper half of its paddle. Thus, complex policies can be extracted by utilizing the opponent's position while training a learner for this environment.

4.2 Reinforcement Learning Setup

Proximal Policy Optimization (PPO) algorithm [18] is used as the deep reinforcement learning method to learn a winning policy against the opponent described above. PPO is a policy gradient-based on-policy RL method that works well with continuous state and continuous action-based problems. Although this algorithm is mainly developed for continuous action tasks, it is also useful for environments

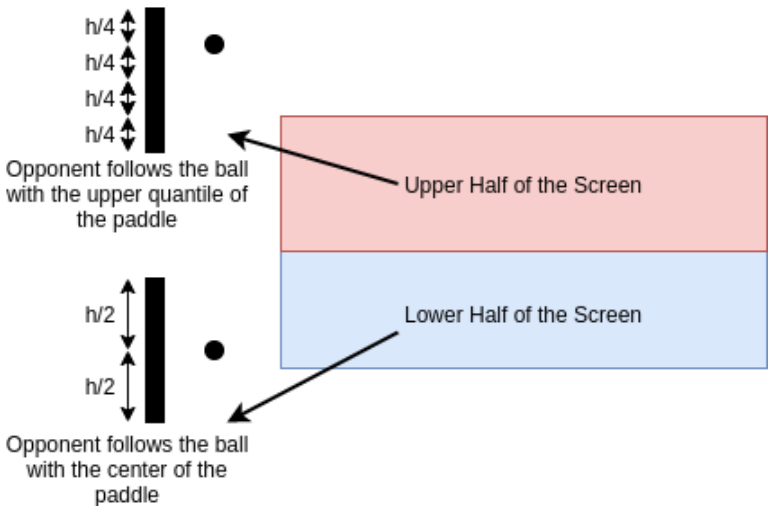

Figure 4: The ball following strategy of the opponent is depended on its position. Lower positions aim for symmetric ball reflection; whereas upper aims to achieve an asymmetric ball reflection.

in which the action space is discrete. PPO is adopted due to its flexibility and the wide range of tasks it can perform well. Since PPO is a local gradient method, proposed policies depend on stochastic exploration and initial network parameter settings. To obtain the general trend in the policies it finds, learning trials are repeated ten times for each agent, and the results are reported as means and standard deviations around the means.

4.3 State and Action Definitions

For the deep reinforcement learner, the fully informative game state (s) is defined as $s = [Opponent_y, Ball_x, Ball_y, Ball_{sin}, Ball_{cos}, Paddle_y]$ where the individual components denote the vertical (y) position of the opponent, x and y positions of the ball, sinus and cosine of the ball direction angle α, and the y position of the paddle the agent is controlling, respectively. Since the range of the actual values of these variables is not the same, they are normalized to be in between 0 and 1. The action space consists three actions defined as Up, $Stay$, and $Down$. As the action names imply, the latter two move the paddle of the agent by $\pm\Delta$ pixels, whereas $Stay$ does not change the paddle position. In the experiments reported in this thesis, Δ was taken as 12 pixels.

4.4 Reward Function

A reward function is designed such that the agent receives +1 when it wins and -1 when it loses. Winning is described as the ball passing beyond the opponent's paddle, whereas losing is defined as either the ball passing beyond the paddle of the RL agent or the frame limit being surpassed. During the intermediate game steps, the agent receives a 0 reward if it does not move. Otherwise, it receives a -10^{-4} reward. This may be considered as an effort penalty which models a human fatigue or an hint for the agent to win as quickly as possible.

4.5 Observational Noise Mechanism

The sensory information that humans receive while playing a game is not perfect. There are visual and motor-related delays and disruptions for perceiving the state and applying an action. For example, if an object is moving too fast, it is not possible for the visual system to perfectly determine its position. An experimental condition is created to mimic this sensorimotor noise where noise is introduced to the state-space representation. In this condition, the noisy state information (s') is given to the agent according to $s' \sim N(s, \sigma)$ where s corresponds to the normalized state. In the experiments reported in this thesis, the noise variance is taken as $\sigma^2 = 0.015 I_{6x6}$.

4.6 EPD in Pong

There are various strategies for solving the game of Pong. The first and the most basic one is to follow the ball at all times using only the y-position of the ball and the controlled paddle. This strategy is relatively simple but ultimately insufficient for cases where the ball moves faster than the paddle in the y-axis. This also results in a high cumulative action cost since the agent will never take a *Stay* action, thus resulting in a non-optimal policy according to the provided reward function. Another solution would be to follow the ball only when it is coming towards the paddle, which would require additional information, the ball's

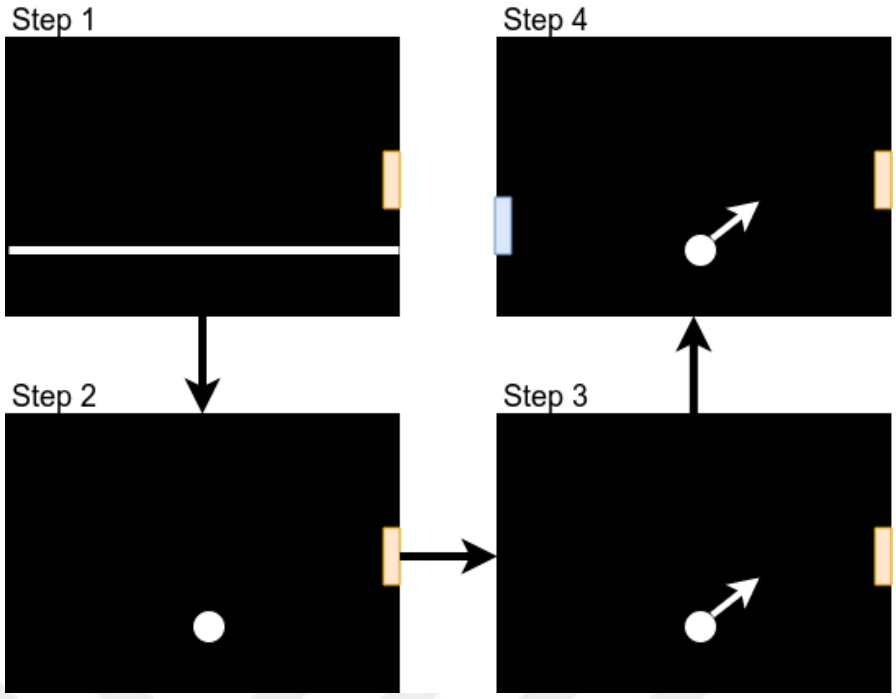

Figure 5: The proposed evolution of perceptual development in our Pong environment. State-space enhances over time for "better" observations. At the first step, the agent only receives the position of its paddle and the y position of the ball. Later, on Step 2, the X position is also revealed. On Step 3, the agent starts to receive the direction info of the ball, and finally, on Step 4, the opponent's position is revealed.

velocity along the x-axis. Due to the increase in the complexity of information, even though the goal stays the same, an increase in the observation capacity makes the task of finding a good policy harder for a human/agent with no prior knowledge about the environment. A trade-off naturally emerges in this situation. When the perceptual information on observation space increases, the policy found is usually closer to the theoretical optimal due to the higher information intake. In other words, more information is needed to extract a better policy. On the other hand, this process is more costly since the search space increases, especially when starting from a random point in the policy space (which is spanned by the weight space of the NN in the case of DRL). (e.g., [32]).

To make searching for an optimal policy in a larger state space less costly, an approach of EPD can be used where the capacity of the perceptual information, therefore the amount of actually observable data in the state, is progressively

increased. In the experiments presented, the unfolding of additional information gradually is undertaken by following a predetermined schedule for ease of analysis (see Table 2). This can generally be based on specific performance measures over the development progression. The information that is made available to the agent in stages is shown in Table 1 and illustrated in Figure 5. Even though the state size does not change (it is always six, in this case), some parts of the state are initially fixed at the median (i.e., 0.5) of the normalized values in the state vector. This approach does not conform the classical MDP setup so it cannot be solved with traditional RL approaches. However, since DRL employs powerful neural networks which can show generalization capabilities, this problem may be tackled. Indeed, with this work we aim to explore the extent this can be achieved.

Steps	Observation Space
1	$[Ball_y, 0.5, 0.5, 0.5, 0.5, Paddle_y]$
2	$[Ball_y, Ball_x, 0.5, 0.5, 0.5, Paddle_y]$
3	$[Ball_y, Ball_x, Ball_{sin}, Ball_{cos}, 0.5, Paddle_y]$
4	$[Ball_y, Ball_x, Ball_{sin}, Ball_{cos}, Opponent_y, Paddle_y]$

Table 1: Description of how the agent receives the state information at each EPD step

4.7 Results

This section presents the experiments conducted alongside their results to assess the effectiveness of EPD concept in RL. We created four experimental setups. (a) Basic (AC), (b) With EPD (PDAC) where the observation space is enriched over time, (c) Noisy Observation (NAC) where the observation the agent receives is noisy, and (d) with Noisy Observation and EPD (NPDAC) where the observation space is noisy, and the observation space is enriched over time. Table 2 shows the timeline of EPD applied. The ≈Episodic Training% corresponds to the amount of training conducted in terms of the episode count. For example, the total training consists of 15000 episodes, and at the end of step 1, which accounts for 350000 frames played, the agents are approximately at episode 1850. The timeline and emulation steps are selected by analyzing multiple training sessions' reward and

progress peaks.

First of all, an overall analysis of the results corresponding to the experimental cases are provided, then comparisons of (a) vs. (b) and (c) vs. (d) are performed to investigate the effect of the EPD process in different environmental setups. The training was carried out in all experimental conditions with 15000 episodes (i.e., gameplays) and repeated ten times. After every 100 episodes, a 50 episode test sequence was initiated to extract the current policy and success of the agents along with learning. The repetitions were applied to account for the stochasticity in the DRL adopted, which is common to all policy gradient-based RL methods.

Steps	Frame Interval	≈Episodic Training%
1	0-350k	12.3%
2	350k-700k	18.8%
3	700k-1.5m	22.7%
4	1.5m-Terminal	100%

Table 2: The timetable of our EPD process. The state representations corresponding to these steps can observed on Table 1.

4.7.1 Training Analysis

4.7.1.1 Action Selecting Frequency

The first evaluation step is to validate whether the agents successfully learned to solve the task. For this purpose, the policy of these agents is observed. Since an Action Cost mechanism is implemented, which punishes the agent every time it takes a non-*Stay* action, the agent must use *Up* and *Down* actions as little as possible to reach an at least sub-optimal policy. In Figure 6, the average frequency of each action is provided.

When Figure 6 is checked, it is observed that all of the agents learned that the *Stay* action is most beneficial for them since all of them preferred this action roughly 80% of the time.

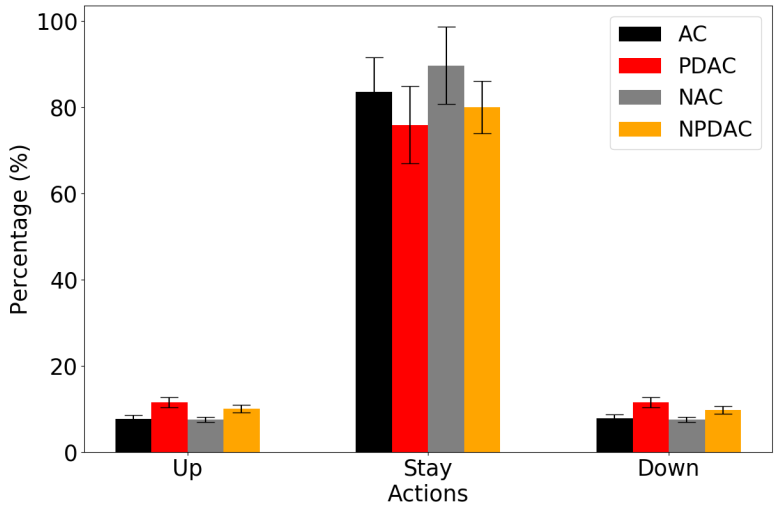

Figure 6: Mean action analysis for each agent after 10 repetitions. The error lines indicate the Standard Mean Error.

4.7.1.2 Accumulated Rewards

The aforementioned result does not validate that the agents extracted a feasible policy since it might also conclude that the agents just tend to stay overall and lose all the episodes. To take this analysis further, peak rewards (obtained within the 50-test games) must be checked to understand if the agents won at least a single game. To this end, the learning results are obtained and given on the right graph of Figure 7 as average maximum rewards obtained in the test cases.

The maximum reward an agent can take in a test sequence is 50 since there are 50 test runs, and a win gives the only possible positive reward of +1. The average rewards provided on the right graph in Figure 7 indicate that all agents were able to extract a suitable sub-optimal policy for the given task. It is also important to note that, due to the structure of the environment, winning every test episode, therefore reaching a reward of $50 - \epsilon$ ($\epsilon \ll 1$ being the total action cost) is not possible since there are cases where the learner cannot hit the ball.

4.7.1.3 Progress of RL Agents

Since the experiment limits are based on episode count and action cost is present, which also intuitively punishes longer episodes, another critical data that must be

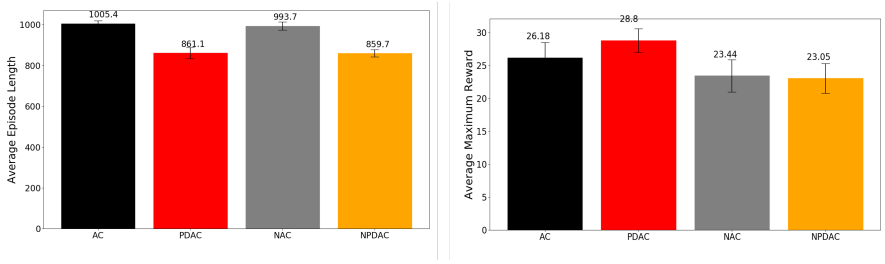

Figure 7: Mean frame counts per episode (right) and mean maximum rewards (left) over 10 trials are shown for each agent. The error lines indicate the Standard Error of the Mean. The agents with EPD (PDAC and NPDAC) exhibit shorter length game plays on the average. The average maximum rewards suggest that the PDAC agent performs better than the AC agent.

analyzed for the experiments is the average episode lengths for each agent. Figure 7 provides the average frame count (the time it took for single gameplay) for each agent and their maximum rewards throughout the training process. It is seen that the agents with EPD spent fewer steps for each episode when compared to the ones without it. Also, Table 3 indicates that this is a pattern that can be observed throughout training. This is further analyzed in the following subsections 4.7.2 and 4.7.3.

Agent	0%-10%	10%-20%	40%-100%
AC	156.08 ± 7.46	1211.03 ± 45.37	1190.2 ± 11.43
PDAC	125.62 ± 5.86	975.57 ± 65.85	1119.8 ± 17.08
NAC	145.71 ± 11.98	1123.19 ± 30.84	1179.1 ± 15.99
NPDAC	123.04 ± 4.88	972.40 ± 53.00	1147.5 ± 14.94

Table 3: Mean frame count per episode between certain periods of training and their corresponding Standard Mean Errors

4.7.2 Effect of EPD in Noise-free Setting

The first part of our experiment is to compare our proposed approach in a more usual RL scenario. For achieving this, the agents AC and PDAC, where one is trained according to the traditional RL training paradigm and the other with our proposed approach, are compared. Test results across frame counts (i.e. game length) can be found in Figure 8. When Figure 8 is analyzed, it is possible to make several deductions from the rewards and frame counts in the graph. Since

the episode count is fixed for eliminating training bias, the length along the x-axis for the left on Figure 8 indicates that the total frame count of the PDAC agent is shorter than the AC agent. This indicates that the average episode length is shorter for the PDAC agent, which can also be confirmed from the right graph on Figure 8, which displays average game lengths across training. Shorter episode lengths can also mean higher rewards due to the Action Cost. When the right graph on Figure 7 is analyzed, it is possible to see that the average maximum reward out of 10 repetitions for PDAC is higher than AC. This shows that overall, PDAC learned a better policy than AC. Finally, as can be observed from the rewards on Figure 8, even though the PDAC agent was learning worse at the beginning, it was able to reach a key milestone (winning 75% of the time, which is equivalent to reaching a cumulative reward of ≈ 25) faster than the AC agent.

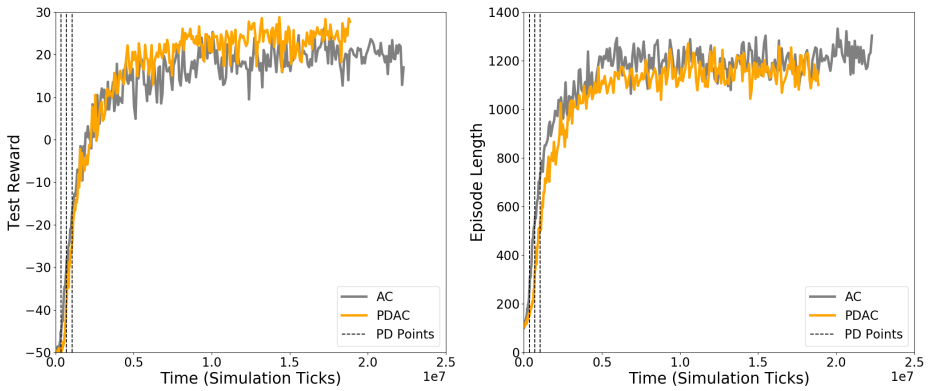

Figure 8: The average total reward obtained from test runs v.s. number of training steps(left) and the average game length from the test runs v.s. number of training steps(right) for AC and PDAC agent.

Another interesting observation can be made for this scenario by looking at the action percentages at Figure 6. It is observed that, in terms of percentage, the PDAC agent took fewer *Stay* actions than the AC agent. By combining this information with the aforementioned detail about episode lengths, it is possible to state that the PDAC agent was able to extract a policy more complex in terms of taking less *Stay* actions for terminating the episode faster, thus achieving a higher cumulative reward.

4.7.3 Effect of EPD in Noisy Setting

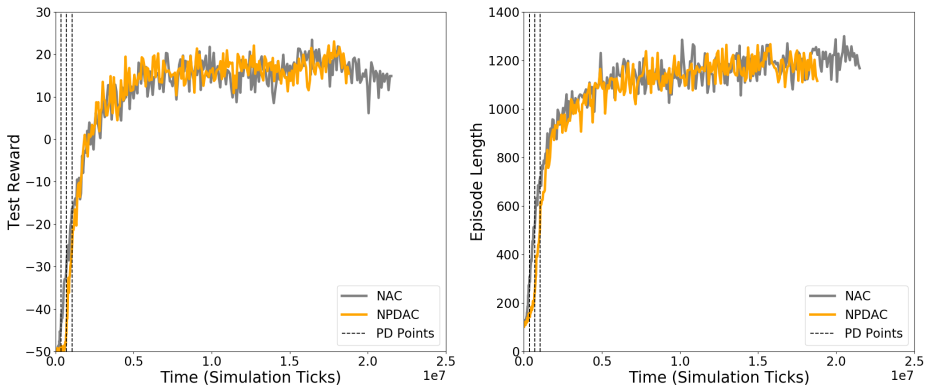

Figure 9: The average total reward obtained from test runs v.s. number of training steps(left) and the average game length from the test runs v.s. number of training steps(right) for NAC and NPDAC agent.

In this section, we investigate the effectiveness of EPD in a noisy setting by analyzing the agents NAC and NPDAC. The former is trained using the traditional RL training paradigm, and the latter is trained using our proposed approach. On this setup, the state of the environment is given to the agents after an additional noise process, explained in Section 4.5. In Figure 9, the details about these experiments can be observed. When the length of the reward graph of both agents is observed, patterns similar to the ones on the previous experiments (subsection 4.7.2) can be seen. The graph of NPDAC, which is the agent trained using EPD, is shorter, indicating the games finish quicker. This can be observed by re-visiting the left graph on Figure 7. Unlike the previous experiments, there seems to be no observable difference in total reward between the NAC and the NPDAC agent for this scenario. This is also confirmed by the right graph on Figure 7 and this table also indicates NAC agent outperformed the NPDAC agent with a small margin, therefore indicating that in a noisy setup, the EPD approach did not improve the agent in terms of capturing the positive terminal reward, which indicates winning the game, and NPDAC converged to a slightly worse policy than NAC. Even though the extracted policy is worse, the NPDAC agent reached certain milestones faster than NAC, showing that agents who employ EPD can learn faster.

4.7.4 Discussion

The results presented show that our EPD approach improved the overall policy of the agent in the noise-free setting. Moreover, the EPD applied agents showed a favorable learning regime, which resulted in agents winning faster and more than the RL agents trained with regular RL.

However, when noise is introduced to the observation mechanism, the EPD agent performed slightly worse than the regular agent in terms of the terminal reward. Thus, it won fewer episodes during the test than the agent trained using the traditional RL paradigm. However, the EPD agent again learned to finish an episode faster than its regular counterpart.

Both graphs on Figure 8 suggest that, in the noise-free condition, PDAC agent first lost quickly, then learned quickly, and finally reached a policy to win faster. In general, the agents that went through EPD (PDAC and NPDAC) seem first to maintain shorter episode counts indicating they lost quickly. While, the agents that did not go through EPD (NAC and AC) started to solve the task leaving PDAC and NPDAC behind in terms of reward and game length; but, PDAC and NPDAC caught on rapidly and equated their cumulative reward, and finally, PDAC agent converged to a better policy than AC while NPDAC managed to converge to a policy slightly worse than NAC.

Although positive learning effects of the proposed EPD was only clearly seen in noise-free observation condition, we believe that the idea of EPD is a novel and useful add-on to reinforcement learning. A natural follow-up work would be to explore ways to make the EPD robust to noise and expand its applicability to different RL methods. The type of RL method and the condition in which the EPD would yield better learning is a difficult question to answer since, with this approach, one attempts to improve RL while violating the basic RL assumptions. For example, we were not able to stabilize learning when we used a Deep-Q Network [15] with the proposed perceptual development. We believe this is because DQN initializes an Experience Replay mechanism which forces the agent to learn

from an experience buffer, and thus even with additions including intelligent selection mechanisms from the buffer[48] or [49], the random selection from the buffer conflicts with the gradual information revealing mechanism of our system.

CHAPTER V

INFERRING EFFORT-SAFETY TRADE OFF VIA IRL

This chapter covers the details of the research entitled Inferring Effort-Safety Trade Off via IRL. This research aims to propose an IRL framework in order to tackle the problem of inferring a reward function parameter, given a COM trajectory in a real-life task. The chapter first describes the environments and methods applied for the two proposed IRL methods. Later, the results are provided and compared to discover the superior approach in the given scope, and finally, conclusion and discussions are driven based on the results.

5.1 Environment Setup

For the simulations, the human body is modeled as a three Degrees-of-freedom (3 DOF) dynamic system using the PyDy library [50]. The model is attached to the floor through an actuated joint corresponding to the metatarsophalangeal joint. The following joints in the kinematic chain are the knee and hip joints. The simulated experiments start with the agent in a squat pose as illustrated in Figure 10. The simulation is adjusted so that at $t = 0$, the COM of the dynamic system is aligned with the foot connected to the ground.

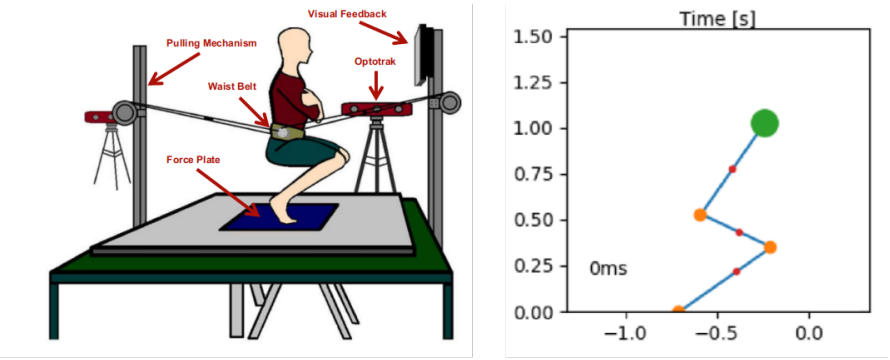

Figure 10: Left: Initial setup of human experiments, Right: Initial setup of the 3-DOF dynamic system

The agent's task is to turn the pose of the dynamic system into a standing

pose by providing "correct" torque values to each joint at each time step. The agent is given a limited amount of time and additional constraints concerning the joint and height range allowed to capture the human experiments and kinematic constraints. The agent is subjected to postural perturbation by applying a force proportional to the vertical velocity of the center of mass of the agent (COM). The state and action of the system is defined as $s = [x_1, x_2, x_3, \dot{x_1}, \dot{x_2}, \dot{x_3}]$ and $u = [u_1, u_2, u_3]$, where x_n and $\dot{x_n}$ represent the angle and the angular velocity of the n$^{\text{th}}$ joint; and u_n denotes the torque applied to the corresponding joint. The external perturbation is modeled as the following:

$$F = -30 * m * \dot{y} \tag{12}$$

where F, m, $\dot{y}$ represents the force of perturbation, mass of the dynamic body and velocity of the dynamic body in y-axis.

5.2 Encoding Effort-Safety Trade-Off in a Reward Function

Since the RL agent will be responsible for generating the synthetic COM trajectories for the squat-to-stand task, a suitable reward function must be designed. The reward function should guide the agent so that the COM of the dynamic body reaches the maximum possible height with close to zero final velocity. Within the environment, this constraint is modeled as COM reaching the 99% of the maximum height with a COM velocity $\leq 0.1m/s$ in the simulation experiments. On the other hand, if the joint limits are exceeded, the height of the head of the agent drops below 50% of the agents' height, or the allowed time is elapsed without any terminal condition reached, the episode terminates with a fail condition. Otherwise, the agent receives a step reward according to the Ecological Cost(EC) function, which is defined as:

$$EC_\omega(s, u) = \omega\|u\| - (1 - \omega)COM_x \tag{13}$$

where ω, s, u, COM_x denote searched trade-off parameter, state, action and

the X position of the agents' COM respectively. The EC function represents a trade-off between the torque used and the calculated safety. According to the given ω value, the function either aims to minimize the total torque usage or maximize the safety. Safety is defined as the tendency to lean forward for the described setup, so an increase in COM_x generates an increase in safety, hence minimizing the EC. It should also be noted that since the environment is formed with a dynamically valid setup, if the COM is forward in excessive amounts, the dynamic body will fall, so the RL agent is also expected to adapt to this dynamic and learn to lean towards in a constructed way.

In order to fit this EC function in to a viable RL scenario, the following cumulative reward function is composed:

$$r_\omega(s,u) = \begin{cases} 1000 & \text{if success} \\ -100 - 400\hat{h} & \text{if failed} \\ \hat{h} - EC_\omega(s,u) & \text{otherwise} \end{cases} \tag{14}$$

where s, u, t, h $\hat{h}$, EC denote state, action, time, current height divided by total height, $h-1$, and ecological cost function respectively.

5.3 Reinforcement Learning Approach

Our solution approach for the aforementioned squat-to-stand task involves solving a reinforcement learning scenario in each case to reach the expected result. This RL solver is utilized as the inner solver of the proposed IRL approach. It aims to output feasible policies for the provided ω values, hence, trajectory data for the IRL pipeline. Implementation details and constraints for the RL solver can be found in the following section.

For the RL solver, we have used Proximal Policy Optimization(PPO), which is an on-policy RL method that works well with continuous state and continuous action-based problems [18]. PPO is explained in detail on Section 2.3.2. To initialize the RL solver, first, the effort-safety trade-off parameter ω must be specified

to instantiate a reward function upon which it can search for an optimal policy.

5.4 Experiment Repetition

Since PPO is a policy gradient-based approach, it works as a stochastic system that converges to close-to-optimal policies with varying degrees of fidelity at each learning session. By repeating the experiments multiple times (20 in our experiments), it is possible to reduce the variance of the result. This will be explained further in Section 5.5.

5.5 Template Trajectory Generation

Algorithm 4 Template Trajectory Generation

```
weights = [0.1, ..., 1.0]
for ω ∈ weights do
    trajectories = []
    for trial = 1 to 20 do
        Instantiate an RL solver according to ω
        Extract top 10 policies from the RL training according reward
        Extract a mean trajectory representing this trial according to the extracted
        top policies
        Add this trajectory to the trajectories list
    end for
    Form a mean trajectory for ω value according to the trajectories list
end for
```

The outputted trajectories from Algorithm 4 is later subjected to the following normalization and smoothing steps:

- *Normalized between 0 and 1*
- *Rotated until the first point is at (0,0) and the final point is at (0,1)*
- *Fit the data with a Cubic Spline*

These post-processing techniques were also used during our IRL setup, mentioned in Section 5.7 and 5.8 as well as our RL analysis.

5.6 Analyzing the Generated Trajectories

The main goal of this research is to propose an IRL pipeline to infer the parameters ω of a reward function $r_\omega(s, u)$ with a known structure. To achieve this, the RL results must be validated to ensure that the envisioned IRL mechanism can be built around the solutions found. Trajectory area is determined as the quantifying metric for this validation and is defined as the area enclosed between the vertical axis and the COM trajectory. When the safety-effort trade-off parameter ω is 0, the system must disregard the importance of torque usage and only maximizes safety, which will increase the area of the trajectory. Likewise, in the case of $\omega = 1$, the system is expected to minimize torque usage and disregard the effect of safety, which will decrease the trajectory area. This was also validated in human experiments since most of the failed attempts were caused by not leaning forward enough.

5.7 Reward Parameter Estimation via Template Matching

A simple IRL mechanism can be built via template matching over a lookup table formed by a set of ω and corresponding simulated COM trajectories. Given a demonstrated trajectory D, one calculates the similarity of D w.r.t the trajectories corresponding to each ω value and either choose the most similar trajectory and return its ω value or make a similarity weighted average of all the ω values to estimate the D parameter that explains the system best. The weighted approach introduces flexibility to the system since if the labeling were a "winner takes all" approach, the labels would be discrete values between 0 and 1, with a precision of 0.1. For the scope of this study, a weighted approach is preferred and named IRL-TM [51].

5.7.1 Trajectory Similarity

To calculate the similarity between 2 trajectories, the following metric has been used:

$$\sum_{i=1}^{N} |\bar{X}_i - X_i|, \tag{15}$$

where $N, \bar{X}, X$ represents the number of points in a trajectory, first trajectory X values, and second trajectory X values, respectively. Since the trajectories are fitted with cubic splines, where the x-axis is the dependant variable, and the y-axis is the independent variable, this metric corresponds to the area of the trajectory.

5.7.2 Model Selection

IRL-TM approach inputs a trajectory and compares the trajectory with each of the template trajectories for $\omega = [0.0, 0.1, 0.2...1.0]$. Afterward, the outputted similarities are inputted to a parametric softmax function which depends on T, and then the result ω is obtained using these weights. The T value of the softmax function directly affects the performance of the IRL-TM, so a basic model selection mechanism is required for this parameter. For this study, the range of T values are selected from $1/2^k, k = [0, 1, 2, 3...8]$.

5.8 Reward Parameter Estimation via Neural Networks and Sample Population

An alternative to the tabular approach previously described in Section 5.7 is to directly use the COM data to train a neural network for predicting the reward function parameter ω. In this approach, the trajectories are fed directly to a DNN, whereas in the IRL-TM approach, the trajectories were labeled based on their area. The specifications of the DNN initialized for the scope of this research is as follows:

- Input Layer(COM Trajectory):
  - Size: 60
- Hidden Layers
  - Count: 5

- Size: 128
- Activation: LeakyRELU [52]

- Output Layer(Predicted ω Value):
  - Size: 1

This approach is called IRL via Data Augmented Neural Networks (IRL-DANN). Example representation for this approach can be found in Figure 11.

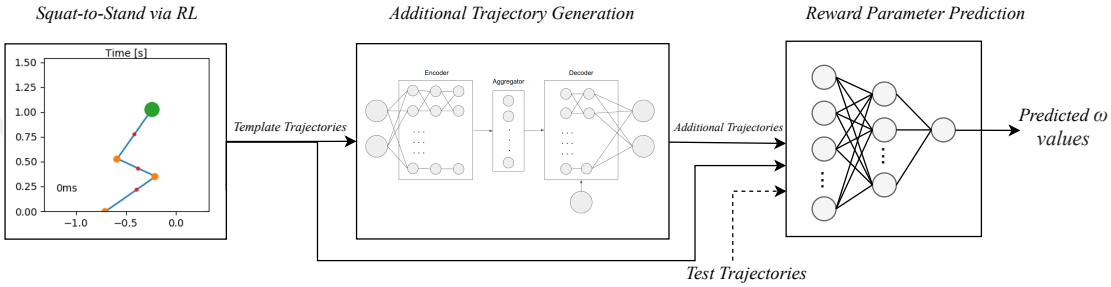

Figure 11: Pipeline representation for the IRL-DANN setup. Detailed description can be found on Section 5.8(E refers to Encoder; D refers to Decoder network within CNMP architecture; see text)

To increase the generalization capability of the DNN, in the same vein as the data augmentation techniques often adopted in machine learning [53, 54], additional trajectories are generated via a trajectory generation architecture described in Section 5.8.1.

5.8.1 Data Generation with CNMP

Conditional Neural Movement Primitives (CNMP) [55] is a framework based on Conditional Neural Processes (CNP) [56] and aims to tackle the problem of learning from demonstration [57]. CNMPs aim to generate trajectory distributions based on specified conditions. They can be conditioned for both single and multiple time steps. During the training of CNMPs, for each iteration, a random trajectory is selected using a uniform distribution. Afterwards, from the selected trajectory, n observation tuples $(t_i; p(t_i))$ are generated, where $p(t_i)$ is the vector of sensorimotor data at time t_i, and single or multiple target query $(t_t; p(t_t))$ are

sampled uniformly. n is a meta-parameter that decides the maximum number of samples selected from the chosen trajectory.

For the scope of this research, the sensorimotor data corresponds only to the horizontal position of the COM of the agent. To augment and populate the original COM data from the already conducted trajectory generation process, For each ω, a different network was trained.

After the selection of a random trajectory, in turn, $n \in \{1, 2, \ldots, 5\}$ random data points from the demonstration is selected as context points. Later, three target points are selected:

- a random target point
- COM location at the beginning(t=0)
- COM location at the end(t=len(T))

This is conducted to ensure that the learned trajectory generation stays faithful to the provided trajectory at the start and the end but differs from the original trajectory in between. Given the context mentioned above, the neural network predicts the mean and variance at each target time. Afterward, the network returns the log-likelihood of the ground truth targets, as the loss to be used, to update the model parameters. During the experiments, loss minimization is done using the Adam optimizer [58].

After training the network on the demonstrated COM trajectories, three condition points in the form of (t, COM_x) were given to the network, in which t is set to the beginning, middle, and the end of COM trajectory length. COM_x denotes the corresponding COM position at time t is set to 0 at the beginning and the end of the movement, and for the middle point, the COM_x position is randomly selected between $[COM_{min}, COM_{max}]$ in which COM_{min} and COM_{max} are the minimum and maximum of the COM position of the demonstrated trajectories at the given t respectively. Given these three condition points, the CNMP network

generates the new trajectory satisfying these conditions. Following this approach, for each ω value, 500 new trajectories are generated.

5.8.2 Training of IRL-DANN

To train IRL-DANN, we first split the original data coming from the RL results into 90% train and 10% validation sets. Later, to augment the data with the trajectories obtained from the CNMP trials for each ω, we add $K \in \{0, 50, 100, 150..., 500\}$ different trajectories to the training set. To get a robust assessment of the ideal number of CNMP generated trajectories to be used for augmentation, in other words, the K values which yield the highest performance, we repeat the previous data augmentation, training, and testing procedure 50 times (see Algorithm 5).

Algorithm 5 IRL Experiment Pipeline

```
ω = 0.0, 0.1, ..., 1.0
N = # of Repetitions
for K in 0, 50, 100, 150..., 500 do
  for trial = 1 to N do
    Split the data as train and test
    For each ω value, select K Number of trajectories from CNMP
    Add k trajectories to the training set
    Train the Neural Network and Compute MSEs, Predictions
  end for
  Calculate Mean Predictions, Mean and STD of MSE Values for ω
end for
```

5.9 Results

In this section, first, the COM trajectories obtained from the RL simulations are validated in terms of the effect of the provided ω value. Afterward, outputs obtained from IRL-TM and IRL-DANN approaches are presented and compared. Finally, calculated ω values for each human subject are presented.

5.9.1 Learned Effort-Safety Trade Off

The proposed reward function, which is used to train the RL agents, is implemented to act as a trade-off between torque usage, i.e., effort, and safety, according to the provided ω parameter. The safety is assumed to be directly proportional

with the distance between $x = 0$ and COM_x. In a typical squat-to-stand scenario, the safety is expected to decrease when the rigid body moves forward, but due to the backward perturbation mechanism, the safety is assumed to be directly proportional with the distance between $x = 0$ and COM_x. Thus, when $\omega = 0$, in other words, the reward function expects the system to maximize safety, the generated COM trajectories must be *fatter* compared to the trajectories generated when $\omega = 1$. To verify that the reward function and the RL system can discover different policies based on the logic described above, we obtained COM trajectories for each $\omega \in \{0.0, 0.1, ..., 1.0\}$ (See Section 5.5 for more details). The extracted mean trajectories for each ω can be found in Figure 12.

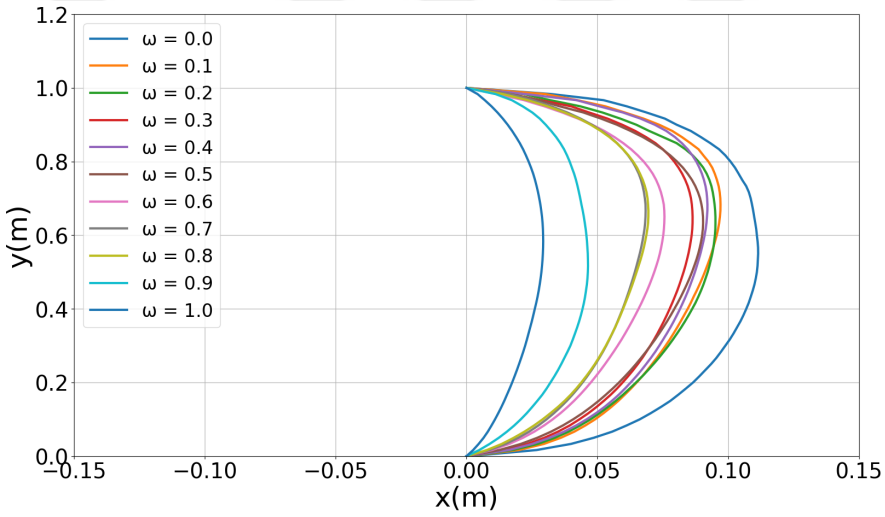

Figure 12: Normalized mean trajectories for each ω value obtained from RL trials

When the mean COM trajectories are inspected (see Figure 12), it can be seen that with few exceptions, they followed the expected pattern that fatness increased as the ω parameter decreased. Note that some order switches occurred between near ω values, which may be expected due to the inherent stochasticity in policy gradient RL methods. Instead of looking at trajectories, one can associate two numbers with a given stand-up trajectory, namely trajectory area and total torque usage, so that the variability in the solutions found can be visualized.

Figure 13 represent the total area and total torque used associated with the policies formed for each $\omega \in \{0.0, 0.1, ..., 1.0\}$. Here, it is possible to observe that

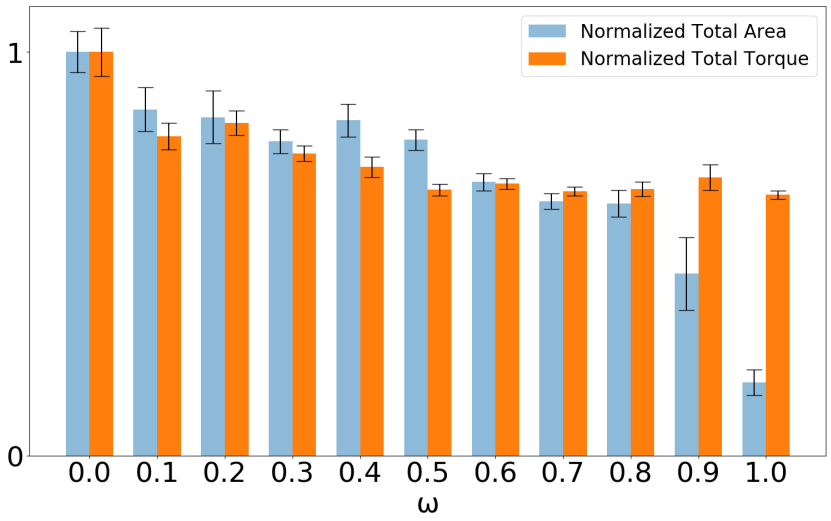

Figure 13: Normalized Area and Torque usage for each mean ω trajectory. For each ω, the means are calculated using top 10 trajectories in terms of rewards in each trial. The error bar stands for the Standard Mean Error(SME) of the means. Total torque stands for the sum of torque used in all joints across a single trial.

the total torque usage decreased when ω increased. This is an expected behavior since high ω values discourage high torques due to $-\omega||u||$ term in the ecological cost function (Equation 13). Furthermore, when the value of ω decreased, the area of the COM trajectory increased due to the expected leaning forward motion. These results suggest that the policies discovered using our RL setup are consistent and confirm our reward design logic, thus forming a valid base for an IRL system to estimate the reward function associated with a given COM trajectory.

5.9.2 Effort-Safety Trade-off via IRL-TM

The IRL-TM approach is used to form a baseline method of straightforward IRL in a table lookup fashion. This approach benefits from the fact that it can directly estimate ω values without an additional cost for the IRL schema. In other words, it can produce an output with a very low computational complexity while somewhat remaining local since it solely depends on the area of the trajectory. To generate the IRL-TM results, for each ω, test trajectories are generated from the synthetic date, excluding the ones to form the IRL-TM approach. This way, the train and test set of the IRL-TM approach is split since no baseline trajectory is used for generating the template trajectories on the IRL-TM system.

During model selection for the IRL-TM approach, the MSE value for each T is calculated. T value indicates the temperature parameter for the softmax function. It allows to increase/decrease the precision of the selection since the selections are based on weighted averaging of the output of the softmax function.

The MSE represents the difference in terms of area for provided trajectories. The average MSE values can be found on Figure 14.

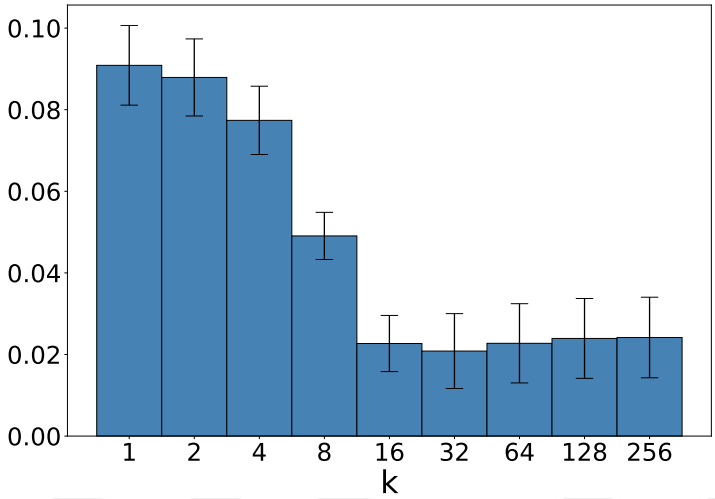

Figure 14: Average MSE Value between test trajectories and template trajectories for each K parameter, which controls the temperature in softmax computation with the relation T = $1/k$. The bars represent the mean of MSE values while the error lines represent the STD.

When this figure is inspected, it is visible that k values above 16, which stands for $T = 0.0625$, had no significant differences, but the lowest MSE occurred when the temperature was 0.03125, $k = 32$. This means that when the temperature is 0.03125, the IRL-TM approach performed the best in terms of accuracy of labeling trajectories with ω.

After the best performing k in terms of MSE is calculated, the next step is to present and analyze the final results of the IRL-TM approach, which can be found in Figure 15.

Figure 15 shows that the IRL-TM approach was successful in inferring the correct values for most of the test trajectories. Also, it is visible that IRL-TM performed better higher ω values(see Figure 19). This is reasonable since, due to

the proposed reward function, in lower ω values, the area of the trajectory is less important to maximize the cumulative reward.

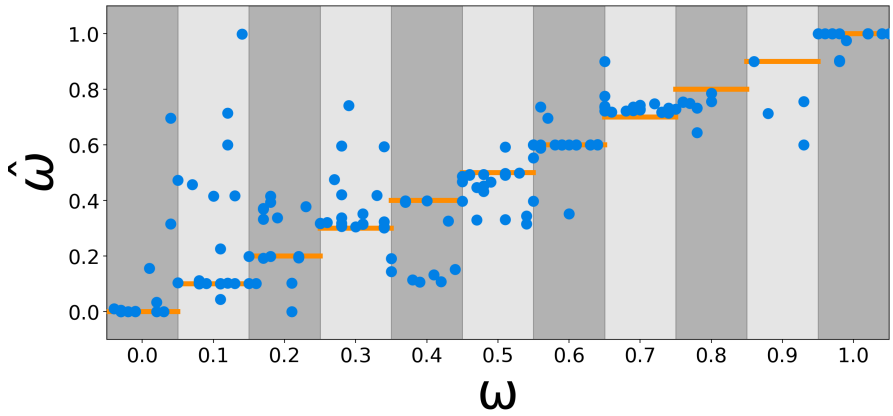

Figure 15: ω predictions for each test trajectory using the IRL-TM approach. The orange lines represent the ground truth while the blue dots represent the predicted ω value when the ground truth of the input trajectory is the orange value.

5.9.3 Effort-Safety Trade-off via IRL-DANN

Since IRL-TM is a local approach which solely depends on the area of the trajectory, a more sophisticated approach can be derived using NNs. Since our RL-based approach for template trajectory generation is costly and Neural Networks are data-driven, to form a feasible enough dataset for a NN, the initial RL dataset is populated using CNMPs. In this section, first, the new data from the CNMP outputs are analyzed and compared with the original RL data; later, the performance of the IRL-DANN approach itself and the effect of additional data on IRL-DANN is presented.

5.9.3.1 Validity of the Trajectories Generated by CNMP

As described earlier, to improve the reward parameter estimation accuracy, additional trajectories are generated from CNMP to augment the NN training data. To successfully apply this augmentation to the training data, it must be validated that the original data and the data generated from the CNMP are similar. The CNMP is trained using the original trajectories obtained from the RL trials, so the

output of this system is also expected to be similar to the original trials. To validate this, the mean and standard deviation of the CNMP outputs are computed and overlaid on top of the original trajectories. This inspection can be found in Figure 16. When the comparison of trajectories is checked, it is visible that although the input trajectories and mean CNMP trajectories have differences, they look similar overall.

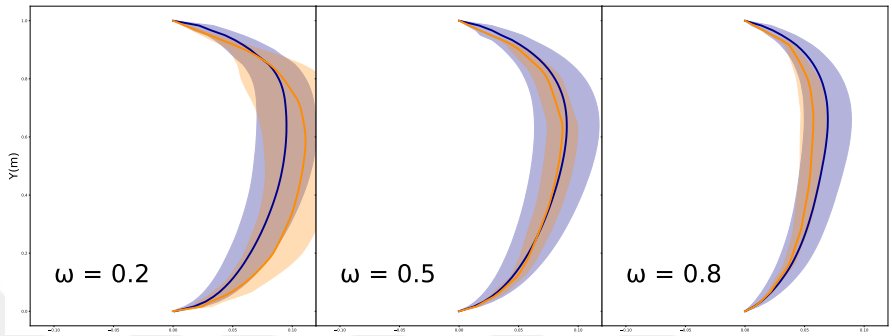

Figure 16: Example CNMP output trajectories together with their corresponding trajectories inputted to CNMP. The blue lines represent the CNMP inputs while the orange lines represent the mean of the CNMP outputs for each ω value. The shadow around the trajectories represent the standard deviation.

5.9.3.2 Addition of CNMP Trajectories

Since the generation of template trajectories is computationally expensive, data augmentation with CNMP aims to correctly and accurately repopulate and augment the dataset to be more suitable with NN-based systems. Since Neural Networks are data-driven, the described augmentation with CNMP approach makes the original RL dataset more suitable for a DNN based IRL approach, i.e., IRL-DANN. To validate the effectiveness of the augmentation approach, an iterative addition step is implemented. This step consists of, for each ω value, randomly taking M trajectories from the additional CNMP outputs, adding them to the train set, and calculating the MSE value to investigate the effectiveness of the data augmentation. This setup is repeated 50 times for each M to reduce the stochasticity caused by the random selection. M value is ranged between 0 and 500. So, for example, when M is 100, 1100 trajectories are added to the train set.

The effect of M on the MSE values, in other words, the performance of the system, can on Figure 17.

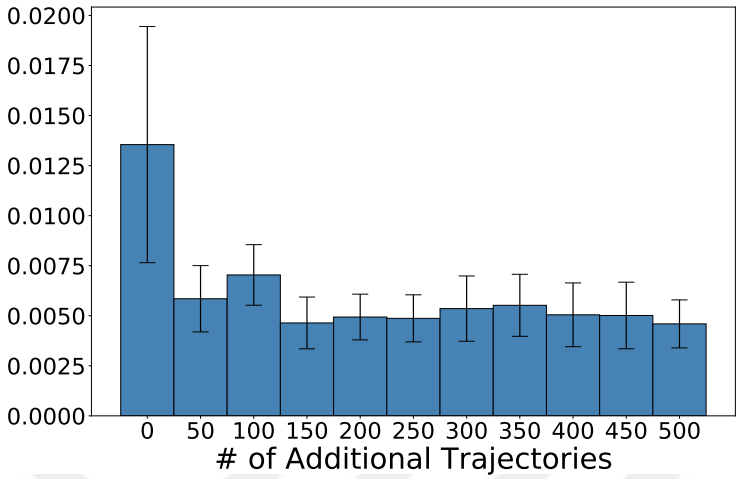

Figure 17: The effect of data augmentation in terms of the accuracy of the system. The bars represent the mean of MSE values while the error lines represent the STD. The number of added trajectories are for each *omega* so since we have 11 different ω values, 550 trajectories are added to the entire data set when number of additional trajectories is 50.

Figure 17 clearly shows that the addition of augmented trajectories significantly improved the performance of the neural network both in terms of the test MSE loss and the stochasticity of loss since the trials with augmented trajectories also reported less standard deviation.

5.9.3.3 Predicting the Effort-Safety Trade-Off

The visualizations of the ω predictions made by the IRL-DANN approach can be found on Figure 18.

When Figure 18 is checked, it is visible that the IRL-DANN approach successfully predicted the correct ω parameters for most of the inputted trajectories since the prediction points are mostly close to ground truths.

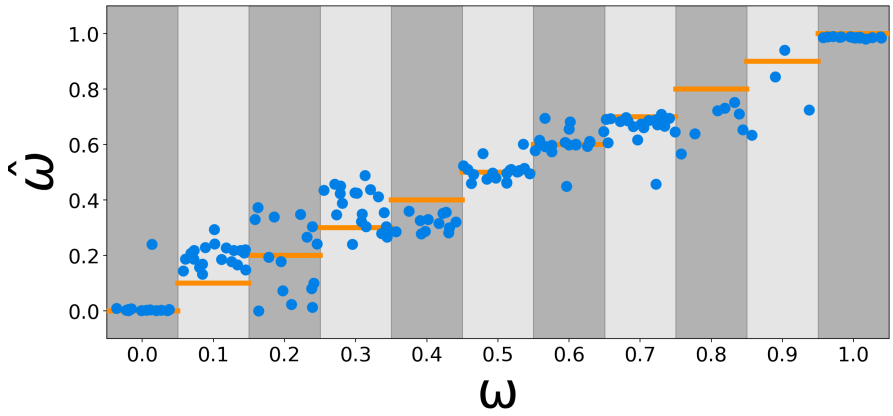

Figure 18: ω predictions for each test trajectory using the IRL-DANN approach. The orange lines represent the ground truth while the blue dots represent the predicted ω value when the ground truth of the input trajectory is the orange value.

5.9.4 IRL-TM vs IRL-DANN

Both IRL approaches aimed to tackle the same problem, extracting the ω value for given trajectories. The same experiment setup is used for both, and the results were calculated according to the same format. For both approaches, the performance metric is determined as the MSE value between the labels and the predicted values. When the MSE values for the IRL-TM and IRL-DANN approach are investigated (which can be found on Figure 14 and Figure 17 respectively), it can be observed that the IRL-DANN approach yielded lower MSE results when compared to the IRL-TM approach. The lowest MSE of the IRL-TM approach is approximately five times larger than the lowest MSE of the IRL-DANN. Also, Figure 17 shows that applying data augmentation improved the performance of the IRL-DANN approach.

To conduct that the predictions of IRL-TM and IRL-DANN are different, we can perform an unequal variance t-test, commonly referred to as the Welch's t-test [59]. It is a two sided test for the null hypothesis that two independent samples have identical expected values. In other words, the test conducts whether two set of samples are drawn approximately from the same distribution. If the calculated p value is less than 0.05, it is considered that the null hypothesis is rejected and

the distributions are not equal. When the Welch's t-test is conducted on the MSE values for each of the approaches for the test set, the p value is calculated as 0.006, which shows that the predictions taken from both of the approaches does not follow the same distribution.

For comparing the performance of these two approaches in detail, the mean MSE values for each ω can be compared. This comparison can be found in Figure 19.

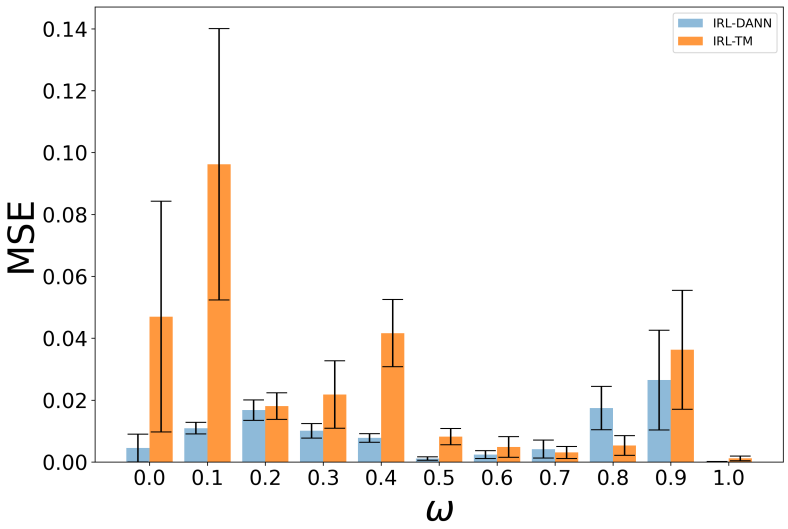

Figure 19: MSE values for IRL-TM and IRL-DANN for each ω value. The error bars indicate the standard error of the mean. Blue bars represent the IRL-DANN approach while the orange bars represent the IRL-TM approach.

Figure 19 shows that not only IRL-DANN outperformed IRL-TM in most of the ω values, it also produced better results with less variance.

All of these comparions mentioned above indicates that IRL-DANN outperformed IRL-TM for the the given task.

5.9.5 Effort-Safety Trade-Off in Human Data

To show the applicability of our IRL system to real experiment data, we analyzed a set of human COM trajectories obtained during squat-to-stand movements performed under perturbation. As described in Section 5.9.4, the IRL-DANN approach has been selected for the task of evaluating human trajectories due to its

performance on the synthetic data. The extracted ω values for each subject can be found on Table 4. Since the true effort-safety trade-off of human subjects is not known, it is not possible to quantitatively assess the correctness of our estimations, however the synthetic COM trajectories obtained with the estimated trade-off parameters suggests that the developed IRL based estimation pipeline is sound.

Table 4: Extracted ω value for each human subject according to the IRL-DANN approach

Subject	ω Prediction	Subject	ω Prediction
1	0.85 ± 0.08	11	0.99 ± 0.01
2	0.62 ± 0.09	12	1.0 ± 0.0
3	0.87 ± 0.02	13	0.59 ± 0.07
4	0.86 ± 0.03	14	0.8 ± 0.05
5	0.86 ± 0.03	15	0.31 ± 0.04
6	0.31 ± 0.11	16	0.72 ± 0.04
7	0.36 ± 0.27	17	0.6 ± 0.03
8	0.64 ± 0.05	18	0.78 ± 0.03
9	0.69 ± 0.18	19	0.79 ± 0.03
10	0.83 ± 0.02	20	0.95 ± 0.02

5.10 Discussion

This chapter aimed to provide valid IRL mechanisms to report the correct ω values for the provided trajectories, where *omega* indicates a trade-off parameter in the reward function to decide whether the subject tries to minimize effort-usage or maximize safety. For extracting ω values, two different IRL approaches have been implemented. The first one(IRL-TM) was a more naive method, directly using the data coming from the trajectory generation pipeline, while the second one(IRL-DANN) used both the initial set and the augmented set outputted from CNMP. Also, IRL-TM relied solely on the area constraint while the IRL-DANN method let the neural network decide what the base constraint was, hence having the upper hand towards generalization. Since the actual ω values of human trials can not be determined, it is not possible to state that one approach was better than the other in the human domain, but the second approach outperformed the first one in all

of the validation trajectories drawn during the RL template trajectory generation step.

We used CNMP to augment our data to increase the size and diversity of our data, which improved our estimation accuracy. We believe that this data augmentation was advantageous in our work as it helped the network be more versatile and adapt to minor alterations to our existing training data.

The analyzed Human ω results show that most human subjects favored minimizing the torque usage instead of maximizing safety. Torque minimization is a mechanism used by the human body in various tasks, including a regular standing up task, but the definition of safety in this setup is different from usual standing up tasks. That is why, due to the addition of external perturbation, this result is somewhat excepted.

The developed IRL method consists of an inner RL loop, which implements a PG-RL-based algorithm called PPO. This allowed our proposed approach to work on continuous state/continuous action environments. Another alternative PR-RL-based approach we have built was based on DDPG, but the results were not as precise as the results coming from PPO. The main disadvantage of using PPO was that it does not guarantee an optimal solution but a close-to-optimal solution. Also, due to the stochasticity of PPO, the policy formed was different after each training. To resolve this problem, we have developed a pipeline that repeats the training process multiple times and outputs a single template trajectory based on the discovered policies.

It should also be noted that to compute the ω values in an unbiased way, a separate model for each human must be trained since all subjects have different body dynamics. Also, the dynamic model we have utilized for these experiments had 3 DoF while the actual system contained 4.

CHAPTER VI

CONCLUSION AND FUTURE WORK

Reinforcement Learning and Inverse Reinforcement Learning are theories concerned with how the learning and decision mechanism of the brain works. In Computer Science, these approaches have been used to tackle various different tasks. Even though they are very powerful, both of these approaches have crucial limitations. In Reinforcement Learning, even the most efficient and successful model needs extensive training to solve some of the tasks that a human can learn with a single trial. In Inverse Reinforcement Learning, although PG based methods can be applied to non-trivial imitation learning problems (e.g [39]), reward function extraction for real life tasks is scarce. One reason for this is, the amount of IRL methods that can be applicable to continuous action and state space problems are very few.(e.g. [45]). Since both of these approaches are biologically inspired, it is a question whether additional biological factors can be facilitated to boost their performance and efficiency.

In this thesis, by using Reinforcement Learning and Inverse Reinforcement Learning, we have proposed novel frameworks to explore how humans learn and how the cognitive decision-making mechanisms of the brain work.

In Chapter 4, inspired from developmental learning, we explored whether RL can benefit from a suitable perceptual development regime and found encouraging results towards an affirmative answer. In particular, we proposed a state representation that changes in stages revealing gradually increasing information about the environment. To show the plausibility of our proposal, we developed a modified Pong game environment. In it we implemented four different types of agents: two with increasing perceptual information and two without in order to assess whether changing state space representation during learning can be exploited for

better performance. The results indicate that a EPD scheme can improve the performance of RL in policy gradient based RL. In particular, we found out that our EPD approach cuts the learning time and usually increases the success of the agent in terms of total cumulative reward for environments without noise.

The future work will address the testing of our proposal on a wide range of environments and testing novel perceptual development mechanisms to obtain principles on how EPD should be managed for better and more robust learning performance, in particular, in the case of observations with noise. On a larger scale, a challenge for future research is to reduce the gap between biological learning and current machine learning algorithms.

In Chapter 5 a PG-RL based IRL method is developed that can be used for analyzing behavioral data for uncovering optimality criteria leading to those data. In particular, the method is utilized for inferring the cost function associated with a given COM trajectory of human (or human model) obtained during a perturbed squat-to-stand movement.

Two different approaches, IRL-TM and IRL-DANN have been developed and within the constraints of the synthetic data, they have been compared in terms of performance. The stochasticity of the RL technique used and the complexity of the task have been resolved by wrapping proposed approaches to an IRL pipeline, which includes averaging a *better* trajectory from the whole. The IRL-DANN approach was found to be better on the synthetic data, hence the human trial data has also been evaluated using this approach. As a result, in terms with the described constraints of the system, most of the human subjects were found to be favoring the minimization of torque usage, rather than maximization of safety.

As future work, we plan to develop a system which can directly propose a non-linear reward function, rather than proposing the reward function parameter. Even though certain systems exists[60, 61, 39, 62], their reliability on real life complex tasks are yet to be observed.

Another future work is to combine the two tasks provided in this research.

If the proposed IRL scenario can be adapted to the EPD environment, one can extract a developmental reward function or direct behavior to better understand and mimic the learning process of an infant. This combined approach can also be used to provide a baseline for behavioral scientists to better understand uncommon behavioral patterns in various fields including daily tasks or game-play [63, 64].

APPENDIX A

ADDITIONAL MATERIAL FOR EPD

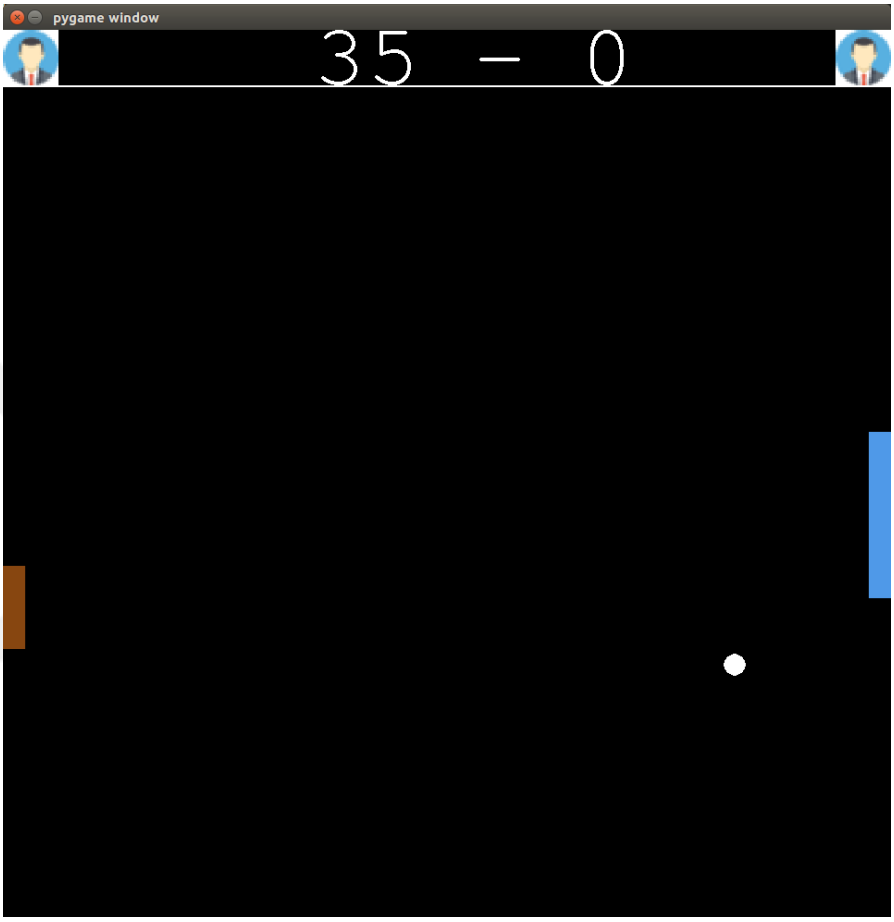

Figure 20: The Environment Initialized for the EPD Research. The left paddle represents the opponent while the right paddle is being controlled by the RL agent.

Network Architecture Details for PPO:

- Actor
  - Fully Connected Layer: 7((STATE+ADVANTAGE+OLD PREDICTION)x128
  - Fully Connected Layer: 128x128
  - Fully Connected Layer: 128x3

- Critic
  - Fully Connected Layer: 3x128
  - Fully Connected Layer: 128x128
  - Fully Connected Layer: 128x1

APPENDIX B

ADDITIONAL MATERIAL FOR THE PROPOSED IRL PIPELINE

Network Architecture Details for PPO:

- Actor
  - Fully Connected Layer: 14(STATE+ADVANTAGE+OLD PREDICTION)x256
  - Fully Connected Layer: 256x256
  - Fully Connected Layer: 256x3
- Critic
  - Fully Connected Layer: 10x256
  - Fully Connected Layer: 256x256
  - Fully Connected Layer: 256x1

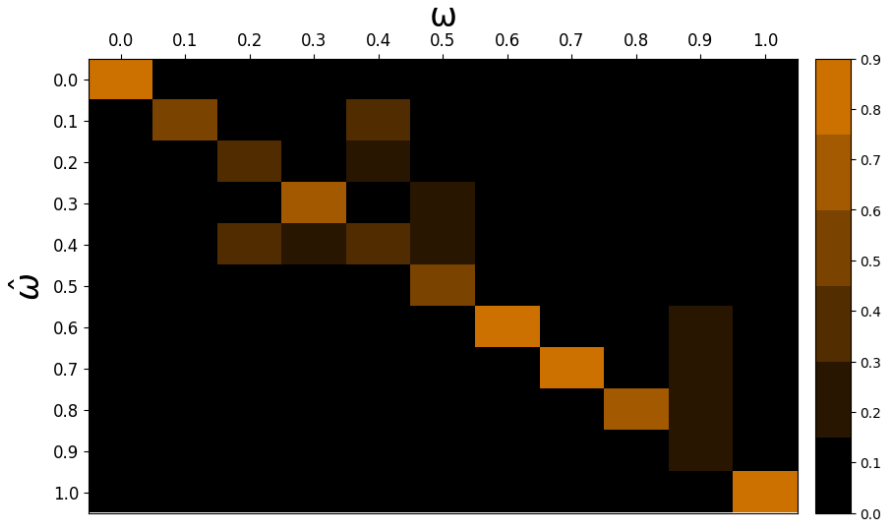

Figure 21: The heatmap distribution of predictions for the IRL-TM approach(found in Figure 15. The heatmap shows that the proposed approach was successful in predicting the ω values for the test set since the heatmap forms a diagonal in most of the cases.

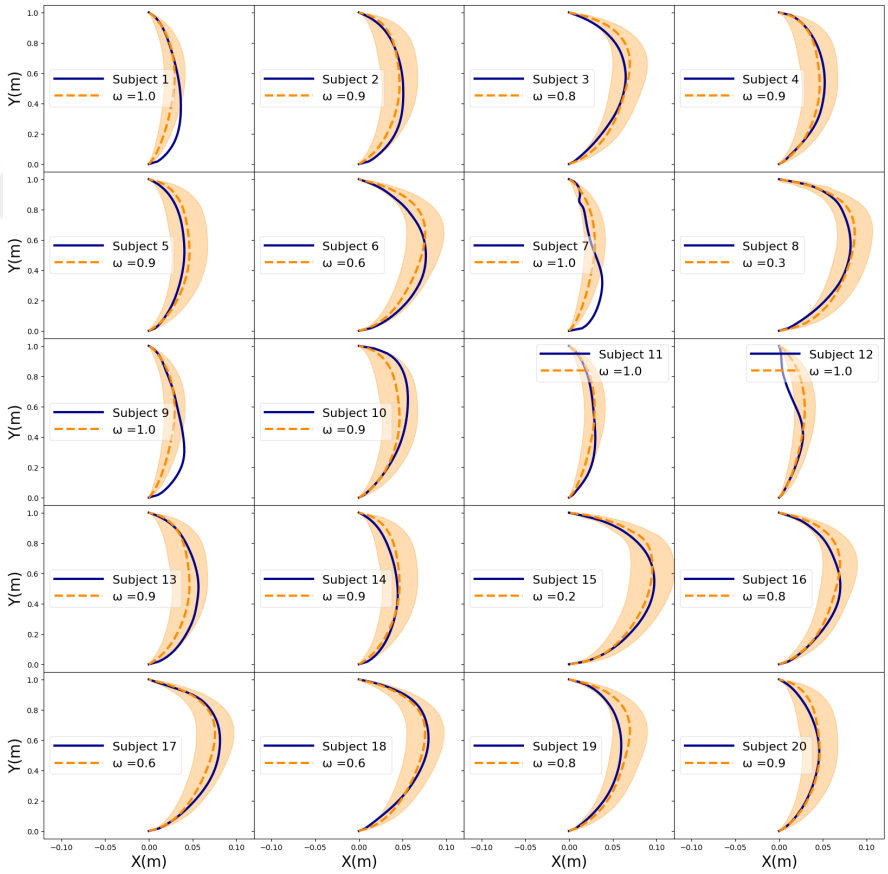

Figure 22: The predicted ω trajectory for each human subject based on IRL-TM approach. The trajectories are selected by the rounding of predictions. Blue lines represent the human subject while the orange dashed lines represent the predicted template trajectory and its' variance. This figure shows that the our system was able to predict suitable ω values for most of the subjects.

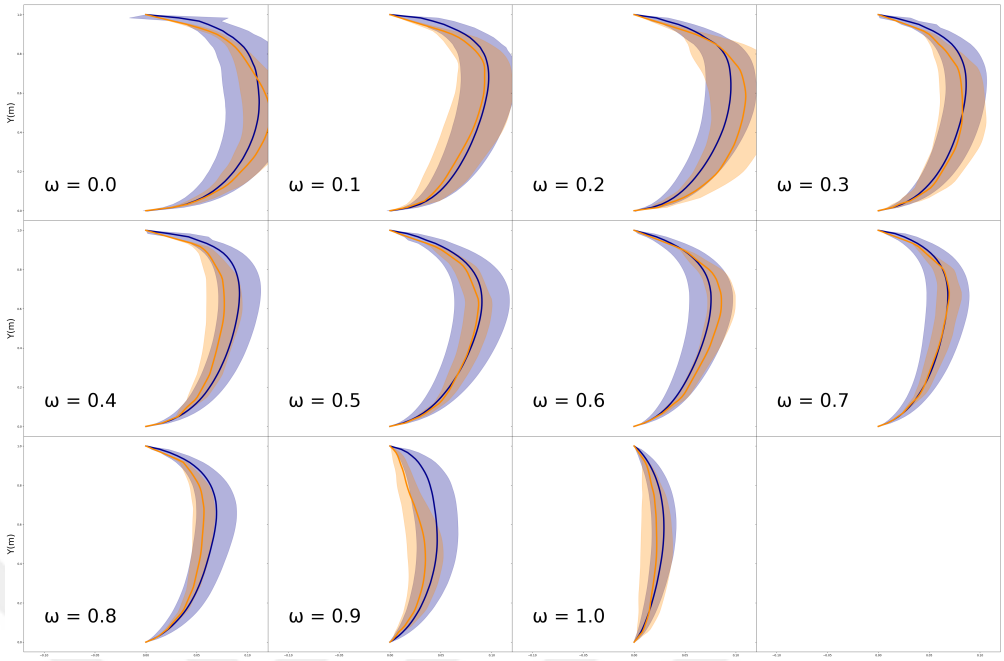

Figure 23: Comparing the mean of trajectories generated by RL simulations and the ones generated by CNMP. The mean of the CNMP and template trajectories are very close for most ω values which indicates that augmented data generation system worked as expected.

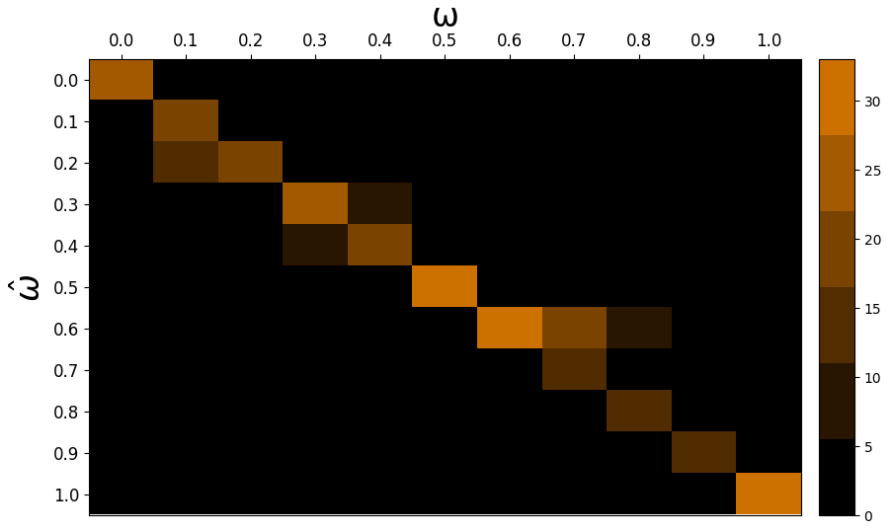

Figure 24: The heatmap distribution of predictions for the IRL-DANN approach(found in Figure 15. The heatmap shows that the proposed approach was successful in predicting the ω values for the test set since the heatmap forms a diagonal in most of the cases.

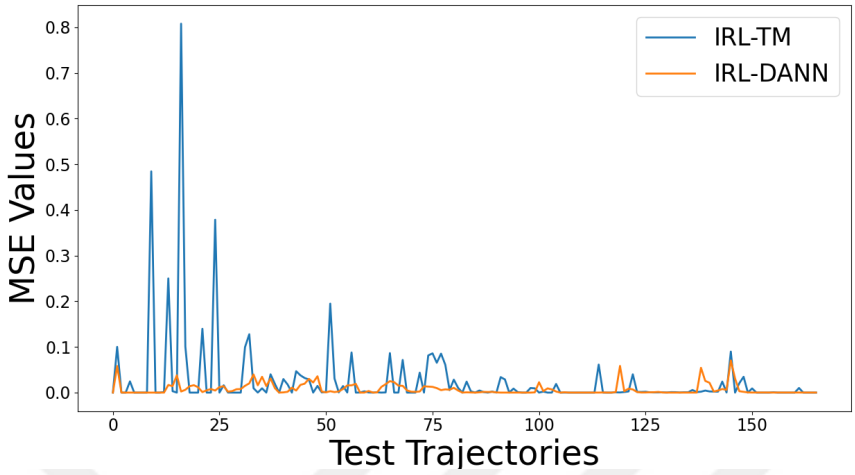

Figure 25: The distribution of MSE values for IRL-TM and IRL-DANN. Based on the Welsh's t-test, the p-value between these two distributions is 0.006 which rejects the null hypothesis and indicates that these samples are drawn from different distributions.

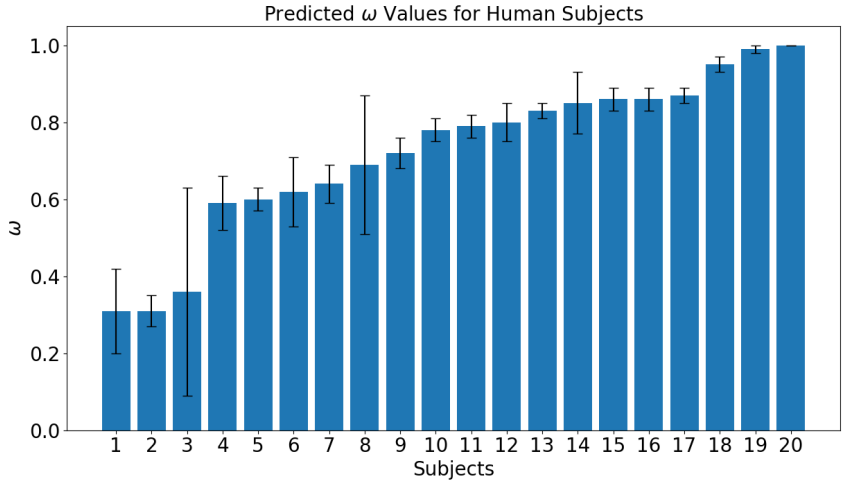

Figure 26: The visual representation of predicted ω values for each human subject based on the IRL-DANN approach. This figure shows that based on the predictions made by IRL-DANN, most of the subjects preferred to optimize effort instead of safety.

Bibliography

[1] J. J. C. Smart, "The Mind/Brain Identity Theory," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab, Stanford University, Spring 2017 ed., 2017.

[2] D. A. Sousa, *How the brain learns / David A. Sousa.* Hawker Brownlow Education Heatherton, Vic, 3rd ed. ed., 2006.

[3] I. o. M. U. F. o. N. a. N. S. Disorders, *Grand Challenge: Nature Versus Nurture: How Does the Interplay of Biology and Experience Shape Our Brains and Make Us Who We Are?* National Academies Press (US), 2008.

[4] R. S. Sutton, *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, 1984.

[5] 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, Dec 1943.

[6] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[7] A.-L. Cauchy, "Analyse mathématique.-méthodc générale pour la résolution des systèmes d'équations simultanées," 1847.

[8] H. E. Robbins, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 2007.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book, 2018.

[10] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[11] W. Beranek, "Ronald a. howard "dynamic programming and markov processes,"," *Technometrics*, vol. 3, pp. 120–121, 04 2012.

[12] C. J. C. H. Watkins, *Learning from Delayed Rewards.* PhD thesis, 1989.

[13] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," tech. rep., 1994.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *ArXiv*, vol. abs/1312.5602, 2013.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb 2015.

[16] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[17] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, 09 2015.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms.," *CoRR*, vol. abs/1707.06347, 2017.

[19] R. E. Kalman, "When Is a Linear Control System Optimal?," *Journal of Basic Engineering*, vol. 86, pp. 51–60, 03 1964.

[20] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Society for Industrial and Applied Mathematics, 1994.

[21] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, (San Francisco, CA, USA), p. 663–670, Morgan Kaufmann Publishers Inc., 2000.

[22] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," *Proceedings of the twenty-first international conference on Machine learning*, 2004.

[23] D. Wood, J. S. Bruner, and G. Ross, "The role of tutoring in problem solving*," *Journal of Child Psychology and Psychiatry*, vol. 17, no. 2, pp. 89–100, 1976.

[24] M. Arbib, E. Oztop, and P. Zukow-Goldring, "Language and the mirror system: A perception/action based approach to communicative development," *Cognition, Brain, Behaviour*, 2005.

[25] B. Reiser, "Why scaffolding should sometimes make tasks more difficult for learners," 01 2002.

[26] E. Uchibe, M. Asada, and K. Hosoda, "Environmental complexity control for vision-based learning mobile robot," *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 3, pp. 1865–1870 vol.3, 1998.

[27] J. Elman, "Learning and development in neural networks: the importance of starting small," *Cognition*, vol. 48, pp. 71–99, 08 1993.

[28] Y. Nagai, M. Asada, and K. Hosoda, "Learning for joint attention helped by functional development," *Advanced Robotics*, vol. 20, no. 10, pp. 1165–1181, 2006.

[29] M. Zimmer, P. Viappiani, and P. Weng, "Teacher-student framework: a reinforcement learning approach," 2014.

[30] A. Fachantidis, M. Taylor, and I. Vlahavas, "Learning to teach reinforcement learning agents," *Machine Learning and Knowledge Extraction*, vol. 1, 07 2017.

[31] A. Çelikyilmaz, L. Deng, L. Li, and C. Wang, "Scaffolding networks: Incremental learning and teaching through questioning," 2017.

[32] H. B. Suay and S. Chernova, "Effect of human guidance and state space size on interactive reinforcement learning," in *2011 RO-MAN*, pp. 1–6, 2011.

[33] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (New York, NY, USA), p. 41–48, Association for Computing Machinery, 2009.

[34] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher-student curriculum learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, 07 2017.

[35] A. Rusu, N. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 06 2016.

[36] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, pp. 275–293, Aug 2014.

[37] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 4565–4573, Curran Associates, Inc., 2016.

[38] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 1071–1079, Curran Associates, Inc., 2014.

[39] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *ICML*, 2016.

[40] N. Amirshirzad, O. Kaya, and E. Oztop, "Synergistic human-robot shared control via human goal estimation," in *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, pp. 691–695, 2016.

[41] N. Amirshirzad, A. Kumru, and E. Oztop, "Human adaptation to human–robot shared control," *IEEE Transactions on Human-Machine Systems*, vol. 49, no. 2, pp. 126–136, 2019.

[42] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI*, 2008.

[43] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," 2015.

[44] E. Uchibe, "Model-free deep inverse reinforcement learning by logistic regression," *Neural Processing Letters*, vol. 47, pp. 891–905, Jun 2018.

[45] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," in *ICML '12: Proceedings of the 29th International Conference on Machine Learning*, 2012.

[46] K. Dvijotham and E. Todorov, "Inverse optimal control with linearly-solvable mdps," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (Madison, WI, USA), p. 335–342, Omnipress, 2010.

[47] S. Yamaguchi, H. Naoki, M. Ikeda, Y. Tsukada, S. Nakano, I. Mori, and S. Ishii, "Identification of animal behavioral strategies by inverse reinforcement learning," *PLoS Computational Biology*, vol. 14, 2018.

[48] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 11 2016.

[49] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. H. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *NIPS*, 2017.

[50] G. Gede, D. Peterson, A. Nanjangud, J. Moore, and M. Hubbard, "Constrained multibody dynamics with python: From symbolic equation generation to publication," in *ASME*, 2013.

[51] E. Arditi, T. Kunavar, E. Ugur, J. Babic, and E. Oztop, "Inferring cost functions using reward parameter search and policy gradient reinforcement learning," in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1–6, 2021.

[52] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[53] Z. Wu, S. Wang, Y. Qian, and K. Yu, "Data augmentation using variational autoencoder for embedding based speaker verification," pp. 1163–1167, 09 2019.

[54] K. Babaei, Z. Chen, and T. Maul, "Data augmentation by autoencoders for unsupervised anomaly detection," 2019.

[55] M. Y. Seker, M. Imre, J. Piater, and E. Ugur, "Conditional neural movement primitives," in *Proceedings of Robotics: Science and Systems*, (Freiburgim-Breisgau, Germany), June 2019.

[56] M. Garnelo, D. Rosenbaum, C. J. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. M. A. Eslami, "Conditional neural processes," 2018.

[57] 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.

[58] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[59] G. D. Ruxton, "The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test," *Behavioral Ecology*, vol. 17, pp. 688–690, 05 2006.

[60] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," 2015.

[61] A. Bighashdel, P. Meletis, P. Jancura, and G. Dubbelman, "Deep adaptive multi-intention inverse reinforcement learning," 2021.

[62] Y. Xu, J. Xie, T. Zhao, C. Baker, Y. Zhao, and Y. N. Wu, "Energy-based continuous inverse optimal control," 2019.

[63] P. Lanillos, D. Oliva, A. Philippsen, Y. Yamashita, Y. Nagai, and G. Cheng, "A review on neural network models of schizophrenia and autism spectrum disorder," *Neural Networks*, vol. 122, pp. 338–363, 2020.

[64] A. Philippsen and Y. Nagai, "Understanding the cognitive mechanisms underlying autistic behavior: a recurrent neural network study," in *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 84–90, 2018.