Palestine Polytechnic University

Deanship of Graduate Studies and Scientific Research

Master of Intelligent Systems - Robotics Track

Reinforcement Learning-Based Human-Machine Co-Adaptation via Policy Gradient

Submitted By

Elias M Maharmeh

Thesis submitted in partial fulfillment of requirements of the degree of Master of Intelligent Systems

February 19, 2023

The undersigned hereby certify that they have read, examined and recommended to the Deanship of Graduate Studies and Scientific Research at Palestine Polytechnic University the approval of a thesis entitled:

Reinforcement Learning-Based Human-Machine Co-Adaptation via Policy Gradient Submitted By

Elias M Maharmeh

In partial fulfillment of requirements of the degree of Master of Intelligent Systems.

Graduate Advisor Committee:

Prof.Dr. Karim Tahboub
(Supervisor), Palestine Polytechnic University.
Signature: ..... Date: .....

Dr. Ahmad Qudaimat
(Internal committee member), Palestine Polytechnic University.
Signature: ..... Date: .....

Dr. Mamoun Nawahdah
(External committee member), Palestine Polytechnic University.
Signature: ..... Date: .....

Thesis Approved by:

Name: .....

Dean of Graduate Studies and Scientific Research
Palestine Polytechnic University

Signature: .....

Date: .....

Acknowledgment

I would like to express my deepest gratitude to my advisor, Prof. Dr. Karim Tahboub, for his guidance, support, and encouragement throughout the entire process of my master's study. His expertise and valuable insights have been instrumental in shaping my research and helping me achieve my goals.

I would also like to extend my sincere appreciation to the committee members for their time and effort in reviewing my thesis and providing valuable feedback and suggestions.

I am also forever grateful to my parents for their unwavering love and support. Their belief in me and my abilities has been a constant source of inspiration throughout my academic journey.

I would also like to thank my family, especially my wife and daughter, for their love, support, and understanding. They have been my rock, my home, and my safe haven. My child has been a source of constant joy and inspiration, and I am so grateful to have them in my life.

I am also immensely grateful to my friends, who have been with me through thick and thin. Their unwavering support, encouragement, and companionship have been invaluable, and I could not have done it without them.

Finally, I would like to express my gratitude to everyone who has supported me in any way during the course of my master's study. Your support, encouragement, and belief in me have been instrumental in my success.

Thank you all from the bottom of my heart.

Elias M Maharmeh
February 19, 2023

Abstract

Human Machine Co-adaptation (HMCo) is a critical problem in the design of intelligent systems that interact with humans. This thesis proposes a general framework for solving HMCo problems using a reinforcement-based approach called the policy gradient algorithm. The thesis goal is to empower the machine with the ability to learn a policy or a strategy in order to co-adapt to human behaviors. The proposed approach is based on the assumption of rationality on the human side and involves learning a policy that co-adapts to dynamic environments and aids the human while performing a specific task. The effectiveness of the proposed approach is demonstrated through case studies, including both direct and indirect shared control, and some of the challenges and limitations that must be addressed in order to further advance the field are highlighted. These challenges include the sensitivity of the algorithm to hyperparameters, the issue of local minima, and the complexity of the optimization process. The impact of the human factor during the training process is also considered, as is the need to enhance sampling complexity in order to handle the limitations of real-world interaction.

This thesis makes several key contributions to the fields of HMCo and intelligent systems design. First, it provides a general framework for solving HMCo problems that is based on policy gradient methods and is applicable to a wide range of environments and tasks. Second, it demonstrates and tests the feasibility and effectiveness of the proposed approach through case studies involving both direct and indirect shared control. Third, it identifies key challenges and limitations that must be addressed in order to further advance the field, such as the sensitivity of the algorithm to hyperparameters and the complexity of the optimization process.

Contents

Acknowledgment . . . III
Abstract . . . V
Contents . . . VII
List of Figures . . . IX
List of Abbreviations . . . XI
1 Introduction . . . 1
1.1 Mapping: Static vs. Dynamic . . . 2
1.2 HMCo components . . . 3
1.3 Motivation . . . 4
1.4 Problem Statement . . . 4
2 Literature review . . . 5
3 Reinforcement Learning . . . 7
3.1 Discounted Markov Decision Process . . . 7
3.1.1 Formal Definition of Discounted Markov Decision Process (MDP) 9
3.1.2 State Value Functions . . . 11
3.1.3 State-Action Value function . . . 11
3.2 Reinforcement Learning (RL) Algorithms . . . 12
3.2.1 Model-Based RL Algorithms . . . 12
3.2.2 Model-Free RL Algorithms . . . 12
3.3 Policy Gradient Method . . . 12
3.3.1 Policy Gradient Theorem . . . 13
3.3.2 REINFORCE Algorithm . . . 15
4 Methodology: Coadaptation Problem Within the Reinforcement Learning Framework . . . 19
4.1 Human Operator . . . 20
4.2 Machine model . . . 23
4.3 HMCo problem formulation . . . 24
4.3.1 MDP for HMCo . . . 25
4.3.2 Trajectory for HMCo . . . 26
4.4 Improved Policy Gradient (Policy Gradient (PG)) Algorithm . . . 27
4.4.1 Exploration - Exploitation Dilemma . . . 30
4.4.2 Time Complexity . . . 31

5 Results: Case Studies . . . 33
5.1 Indirect Shared Control Scenario . . . 34
5.1.1 The Environment Mathematical Model . . . 34
5.1.2 PG algorithm implementation and results . . . 36
5.1.3 Results . . . 37
5.2 Direct Shared Control . . . 41
5.2.1 The Environment Mathematical Model . . . 41
5.2.2 Algorithm Implementation . . . 44
5.2.3 Results . . . 45
5.3 Machine - Machine Co-adaptation . . . 48
5.3.1 Algorithm Implementation . . . 49
5.3.2 Results . . . 49
6 General Discussion . . . 53
6.1 The Importance of the Proposed Case Studies . . . 53
6.2 The Rationality Assumption . . . 54
6.3 Practical Aspects of the Proposed Algorithm . . . 55
6.3.1 Selection of Hyper-parameters . . . 55
6.3.2 Experience, Co-adaptation and Policy Iteration Loops . . . 57
6.3.3 Scalability, Complexity and Memory . . . 57
6.4 Limitations . . . 58
7 Conclusions . . . 61
7.1 Summary of main results and findings . . . 61
7.2 Implications and Contributions . . . 61
Bibliography . . . 63

List of Figures

Figure 1.1: Evolution of Operator Generations, [2] 1
Figure 1.2: Evolution of HMI, [5] 2
Figure 1.3: Static mapping Human Machine Interaction (HMI) scheme . . . 2
Figure 1.4: Dynamic Mapping HMI scheme 3
Figure 3.1: An agent interacts with an environment in an MDP 9
Figure 4.1: A segment of the human intention process to generate an action 22
Figure 4.2: The general HMCo framework 25
Figure 5.1: The problem of indirect shared control 36
Figure 5.2: The trajectory of the parameter θ through the co-adaptation process, where the $\times$-line represents the true twist value $\beta_{twist} = 50$ . . . 37
Figure 5.3: The expected long-term reward for indirect-shared control 38
Figure 5.4: The policy convergence, for fixed initial position. Where the $\times$ indicates the true twist value $\beta_{twist} = 10$ degree 39
Figure 5.5: The policy convergence, for random initial position.Where the $\times$ indicates the true twist value $\beta_{twist} = 10$ degree 40
Figure 5.6: The policy gradient of the co-adaptation loop 40
Figure 5.7: The SeaSaw shared control task 41
Figure 5.8: The problem of direct shared control 44
Figure 5.9: The expected long-term reward for direct-shared control, where the covariance $\Sigma_\times = 3 > \Sigma_\star = 1$ 45
Figure 5.10: The system damping ratio ζ, for high covariance $\Sigma_\times$ 46
Figure 5.11: The system damping ratio ζ, for low covariance $\Sigma_\star$ 46
Figure 5.12: The machine policy gradient for k_d 47
Figure 5.13: The time response for θ, by applying the obtained policy parameter k_{dm} 47
Figure 5.14: The problem of direct shared control (the two machines scenario) 48
Figure 5.15: The expected long-term reward for direct-shared control 49
Figure 5.16: The system damping ratio ζ, for unequal covariances 50
Figure 5.17: The policy gradient for k_p ($\times$-curve), and the policy gradient for k_d ($\star$-curve). Where $\Sigma_d > \Sigma_p$ 51
Figure 5.18: The policy gradient for k_{pm1} ($\times$-curve), and the policy gradient for k_{dm2} ($\star$-curve). Where $\Sigma_p > \Sigma_d$ 52

Figure 5.19: The time response for θ, by applying the obtained policies' parameters k_{pm1} and k_{dm2} 52
Figure 6.1: The policy convergence for random initial state. Where the $\times$ indicates the true value $\beta_{twist} = 120$ degree 60
Figure 6.2: The policy convergence for random initial state. Where the $\times$ indicates the true value $\beta_{twist} = 50$ degree 60

List of Abbreviations

HMI Human Machine Interaction
HMCo Human Machine Co-adaptation
ML Machine Learning
LQR Linear Quadratic Regulator
RL Reinforcement Learning
GD Gradient Descent
SGD Stochastic Gradient Descent
MDP Markov Decision Process
PGT Policy Gradient Theorem
PG Policy Gradient
SDM Sequential Decision Making
AI Artificial Intelligence
SGA Stochastic Gradient Ascent
POMDP Partial Observable MDP

1 Introduction

HMI is the process of communication and interaction. An interface medium or device is used to facilitate the interaction [1]. According to [2], a generational evolution may be used to describe the history of interactions between humans and various industrial technologies and equipment. where a generation of human operators is connected to each industrial revolution. There were four industrial revolutions underway at the time this thesis was written. Each industrial revolution is referred to as "industry X.0"; for instance, "industry 1.0" refers to the first industrial revolution, "industry 2.0" to the second, and so on. Additionally, each industry's human operator is referred to as "operator X.0" [3]. The generational evolution of an operator may be divided into the following categories, (see figure 1.1).

1. Operator 1.0: carries out manual and dexterous tasks
2. Operator 2.0: carries out aided tasks, such as a CNC machine operator
3. Operator 3.0: Works collaboratively with robots and other machines according to a robot cooperation scheme
4. Operator 4.0: uses a machine to assist in accomplishing a task, with the goal of enhancing human capacity to perform a certain activity

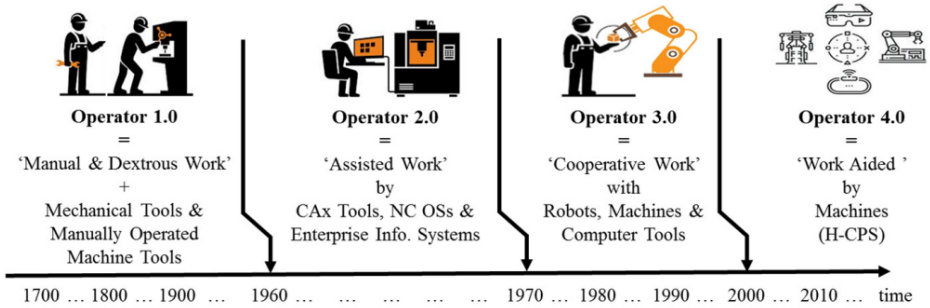

Figure 1.1: Evolution of Operator Generations, [2]

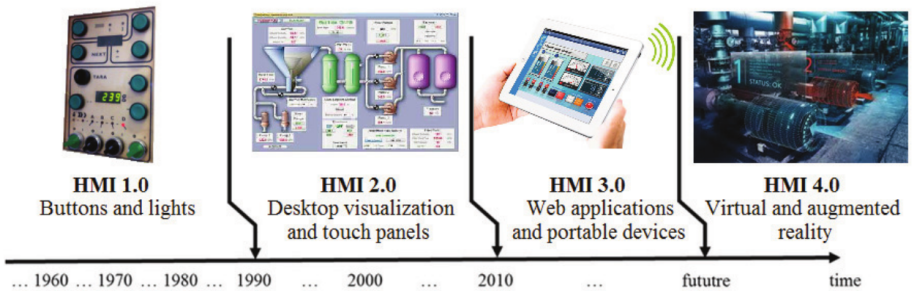

Figure 1.2: Evolution of HMI, [5]

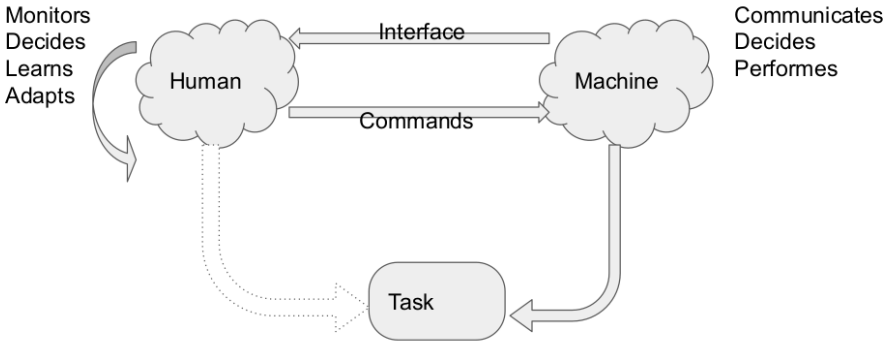

Figure 1.3: Static mapping HMI scheme

1.1 Mapping: Static vs. Dynamic

Classically, a human must use an interface to communicate with a machine, understand how it operates, and then adapt to the machine's interface, (see figure 1.2). The machine's response to human behavior is time-invariant during this cycle of human adaptation to the machine interface [4], [5]. This is known as a "static mapping" [6], which is the mapping between human commands and machine behavior where the burden of learning is on the human side. HMI is mostly used in static mapping scenarios from Industry 1.0 through Industry 4.0. A human, a machine, and a task are required for such HMI, (see figure 1.3).

Therefore, the machine's reaction to human behavior must be dynamic, as shown in figure 1.4, in order to shift the burden of learning and redistribute it between the machine and the human. In [6], this is referred to as "dynamic mapping". As a result, it is necessary for both humans and machines to interact and learn at the same time, just like humans do. That is, the machine needed to be able to interact, learn, and adapt just like a human.

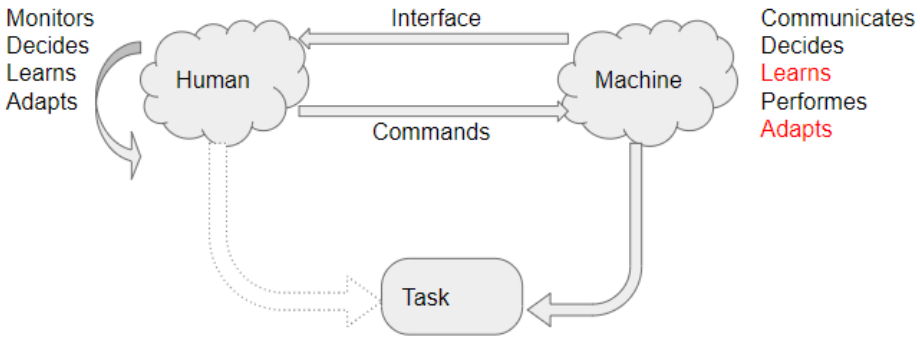

Figure 1.4: Dynamic Mapping HMI scheme

1.2 HMCo components

In this thesis, a generic framework for handling dynamic mapping is introduced. Within this framework, interaction, co-learning, and co-adaptation between the human and the machine occur simultaneously. This framework, known as HMCo, requires the following elements:

1. A human
2. A machine
3. A common task
4. Capability for co-learning (both the human and the machine learn simultaneously)
5. Capability for co-adaptation (both humans and machines adapt simultaneously)

The following define some key concepts used throughout this thesis:

- The Machine or the agent: the terms agent and machine are used interchangeably in this thesis to describe something that acts [7]. If the agent is human, it will be identified as a human-agent
- Co-learning: is the process through which many agents acquire new skills while working together to complete a task [8]
- Co-adaptation: is the process of adapting both the machine's and the operator's behavior while co-learning throughout the interaction

1.3 Motivation

Industry 4.0 is centered on autonomous systems and makes use of modern technologies like cloud computing, augmented reality, virtual reality, and AI. However, the COVID-19 pandemic made the significance of the human role very clear [9], [10]. This necessitates a new industrial revolution [11]–[13], wherein, according to [14]–[16], the objective behind introducing Industry 5.0 is to re-involve the human within the industry and re-create a human-centered paradigm while interacting with machines. The primary motivation for this thesis is to raise the level of human-machine interaction to that of human-human interaction. This will increase the use of the machine's capabilities in executing a specified shared task. Because machines are becoming more intelligent, the way they interact with humans and respond to their behavior must evolve. To respond to human behaviors or actions, the machine must be able to interpret them. This necessitates interaction at a higher level, similar to how humans interact, rather than being limited, for example, to a basic push-button situation (see figure 1.2).

Machines that have been programmed or trained to accomplish a given task will exhibit the same behavior or take the same actions regardless of the state of the human interacting with them. Because of the limitations of static mapping [4], [5], [17], this is acknowledged as one of the primary shortcomings of traditional HMI systems. The HMCo framework seeks to provide the machine with dynamic mapping capabilities while executing a certain common task. As a result, machine behavior is no longer time-invariant in a particular context. It is crucial to note that the human being can naturally accomplish dynamic mapping while learning and adapting within the targeted HMCo framework.

1.4 Problem Statement

First and foremost, the objective of this research is to formulate the HMCo problem as two learning and adapting agents, the human and the machine. The human and machine interact to achieve a predefined goal that is known to both the human and machine. To accomplish co-adaptation, the problem formulation presents five key parts: the human, the machine, the task, the capability to co-learn, and the capability to co-adapt. Second, an Artificial Intelligence (AI)-Machine Learning (ML) algorithm will be employed to resolve the HMCo problem.

2 Literature review

Many applications and principles of traditional HMI, which were described in chapter 1, are provided in [1], [3], [18]–[25]. There, the HMI issue is defined as a person learning and adapting while interacting with a machine. None of the following articles use the HMCo concept established in chapter 1. They are confined to the first three components, where a person learns and adapts while interacting with a machine, but lacks the machine's learning and adapting components. Some authors used the terminology adaptive-HMI. The paper's ultimate goal is for humans and machines to reach or converge on a collaborative behavior to solve a particular task. Some of the algorithms utilized in the articles, on the other hand, have an implicit machine learning and adaptation aspect. Throughout this chapter, all articles will be compared and assessed using the HMCo concept given in chapter 1

The following criteria are used to conduct the review:

1. HMCo's concept or definition
2. Applications or case studies
3. Methodology
4. Results

Ehrlish and Cheng [26] investigated the situation of a human and a robot playing an object-guessing game. The human must estimate which object the robot chose by observing the robot's gaze. Li et al. [27] investigated an instance of a human interacting with an arm manipulator to move an item. Although Ehrlish and Cheng and Li et al. employ the concept of co-adaptation in their work, they only incorporate the human, the machine, and the task. They both employ a Bayesian technique to solve the presented case. The Bayesian method necessitated the development of an explicit model for both humans and machines. According to the measure used, the proposed Bayesian technique produced satisfactory results in both instances. However, due to the nature of Bayesian methods, an explicit model for both humans and machines is required, which might not be available and must be learned from data. This constrains the performance accuracy to the accuracy level of the used models.

Flipse [28] explored the scenario of a self-driving car that interacts with humans via the car's different sensors, such as the touch sensor on the steering wheel and the gas pedal. The objective has been to drive safely along the street. Mohebbi [29] explored the application of rehabilitation robots. Robotic prostheses and exoskeletons were shown. The goal is to help human movements. For the two situations of self-driving cars and rehabilitation robots, the suggested method requires the machine to construct a model for the human and a model for the machine. The human and the machine are co-adapting to each other's behavior based on these acquired models. The co-adaptation problem is solved by employing the Linear Quadratic Regulator (LQR) controller. The framework presented by these two publications [28], [29], includes the five components of the HMCo problem.

The interaction between a human and a manipulator to complete a task was taken into consideration by Shuhei et al.[30], and Peternel et al.[31].They used supervised learning, where each loop of interaction is referred to as an "episode" and is given a good or bad label. A probability density function was created in [30] for each action the robot takes. In [31], they use weighted regression for robot actions, in contrast. The human role in the two scenarios of [30], [31] is to teach the machine while also learning from and adapting to its behavior. Learning from these examples or episodes leads to co-adaptation, which depends on the model capacity utilized in supervised learning approaches.

Z. Danziger et al. [32]–[35], investigated the issue of mapping from a higher space of glove joints, known as CyberGlove, to control a two-degree-of-freedom arm-manipulator on the screen in order to hit a predetermined target. In order to solve the problem, Z. Danziger et al. suggested two ML methods using Moore-Penrose (MP) pseudoinverse and Least Mean Squared (LMS) gradient descent. The interaction occurs in batches of episodes, and for each batch, the mapping from the higher space to the lower space is updated. The five elements of HMCo are included in the problem setup. Two sets of people assisted in conducting the study. The people in the control group are individuals who are operating the arm-manipulator without the use of ML algorithms. Additionally, the LMS group uses the ML algorithm to control the arm manipulator. After extensive training and episodes, the LMS group demonstrated improved performance and came close to attaining the level of human performance.

It is plainly obvious from this review of the literature that neither a formal definition of the problem nor a methodology for addressing it have been put forth. By focusing on the key five components of the HMCo problem, this thesis aims to provide a novel problem structure and definition. It also applies a decision-making framework to the problem's solution, which will be covered in the following chapters.

3 Reinforcement Learning

Creating agents that interact with their surroundings in order to discover the optimal course of action is one of the main objectives of AI [36]. The learning process is determined by the type of data used to learn from [7]. ML is a set of algorithms aiming to mimic human intelligence in learning from various types of data [37], [38].

Supervised Learning, Unsupervised Learning, and Reinforcement Learning (or RL) are the three basic types of machine learning algorithms [39]. Finding a hypothesis or probability density function that best explains and fits the provided data is the aim of both supervised and unsupervised learning [39]–[41]. For supervised and unsupervised learning, respectively, the data takes the form of $\{(x^i, y^i)\}_{i=1}^{i=n}$ and $\{(x^i)\}_{i=1}^{i=n}$, where x^i is referred to as a feature vector and y^i is the response. Using optimization methods like Gradient Descent (GD) or Stochastic Gradient Descent (SGD), the parameters of the hypothesis are determined during the training process [41].

The RL paradigm of ML algorithms includes an agent interacting with its environment or surroundings to fulfill a predefined task [42]. The resulting data is presented as a vector that includes the reward as a scalar quantity, the agent's current state, and the action the agent performed. During an agent's interaction with the environment, data is generated online. In section 3.1, it will be discussed how to model the RL problem using the MDP framework. The objective of RL is to identify an optimal policy that directs the behavior of the agent to perform the desired task [7], [41], [42]. In section 3.2, these algorithms are covered.

3.1 Discounted Markov Decision Process

MDP is used to represent Sequential Decision Making (SDM) problems in which the agent has to undertake a sequence of actions in order to complete a task. An MDP consists of a set of environment states, a set of agent actions, the dynamics of the environment, and a reward function [43]. The state of an MDP is an abstract concept that comprises information about the environment, such as the agent's position and velocity within it. The state must be sufficient or Markvoian. Given the current state, the future state is independent of the past states, or history. This is known as the

Markov propriety [44]. Given the current state and action, the dynamics of an MDP describe the probability of ending up in a specific state. The reward function provides feedback to the agent on how well the action was accomplished. An MDP solution is one that maximizes the expected total reward the agent receives while interacting with the environment to accomplish a specific task. Consider the 1-D discrete state space model shown below:

$$x_k = x_{k-1} + u_k \tag{3.1}$$

where x_k and u_k represent the state and control at time step k. The system described in Eq.(3.1) is a first order Markovian system in which the next state is determined solely by the current state and not by the entire history. The dynamics of this MDP are deterministic, which means that the probability of ending up in the state $x_k = x$ after taking action $u_k = u$ is one. The system dynamics equation given by Eq.(3.1) is obtained by modeling the physical system, for example, using Eular-Lagrangian.

Due to non-linearity, approximation assumptions, ignoring elasticity, and other factors, such modeling technique is valid for complex systems only within a small region of the system's state and action. As a result, the system dynamics may not accurately reflect the system's real behavior. Furthermore, the system's sensors and actuators introduce noise into the system's state and actions. A stochastic MDP is used to deal with such modeling challenges. A stochastic MDP is one in which the environment's states, agent actions, environment dynamics, and reward are all stochastic. This is accomplished by assuming that all unmodeled effects, sensor noise, and actuator noise are generated by a specific probability density function.

Assuming a Gaussian random noise for the system in Eq. (3.1), the system dynamics become:

$$x_k = x_{k-1} + u_k + w_k \tag{3.2}$$

Where $w_k \sim \mathcal{N}(\mu.\sigma)$ is a Gaussian random variable with a mean of μ and a variance of σ. The system dynamics could alternatively be expressed as a probability density function, as seen below:

$$p(x_k|x_{k-1}, u_k) = \mathcal{N}(\mu_k = \mu_{k-1} + u_k, \sigma_k = \sigma) \tag{3.3}$$

Where μ_{k-1} denotes the mean of the random variable x_{k-1}.

3.1.1 Formal Definition of Discounted MDP

A discounted MDP is generally defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ [43], [45],in which:

- $\mathcal{S} \in R^n$ is a measurable state space
- $\mathcal{A} \in R^m$ is the action space
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(R^n)$ is the MDP dynamics
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \Delta(R^1)$ is the immediate reward distribution
- $\gamma \in [0, 1)$ is the discount factor

Where R^1,R^n denote the space of one- and n-dimensional real numbers, respectively, and $\Delta\mathcal{S}$ and $\Delta\mathbb{R}$ denote the space of all probability distributions over R^n and R^1, respectively.

In this chapter, a random variable is denoted by capital letters, but a random variable's instantiation is denoted by small letters. According to the MDP's general definition, the reward is drawn from a probability density function that is dependent on the action and current state. The reward is denoted by the capital letter R throughout this work and is assumed to be a deterministic function of the current state and action.

The interaction process between an agent and an environment according to the MDP framework is shown in figure 3.1. The agent observes the environment's state, $s = s_{t_k} \in \mathcal{S}$ at the k^{th}-time step, $t = t_k$. Afterward, the agent performs an action $a_{t_k} \in \mathcal{A}$ in accordance with its policy function, $\pi(a_{t_k}|s_{t_k}) : \mathcal{S} \to \mathcal{A}$. As a result, at time $t = t_{k+1}$, the environment's state evolves into a new state, $s_{t_{k+1}}$, and the agent immediately receives a reward from the environment, $R_{t_{k+1}}$. Until the agent reaches a terminal state or the desired state, the process is repeated [42]. This interaction loop generates a trajectory, which is a vector of random variables, with the formula $\tau^\pi = (s_{t_k}, a_{t_k}, s_{t_{k+1}}, R_{t_{k+1}}, a_{t_{k+1}}, s_{t_{k+2}}, R_{t_{k+2}}, \dots.)$.

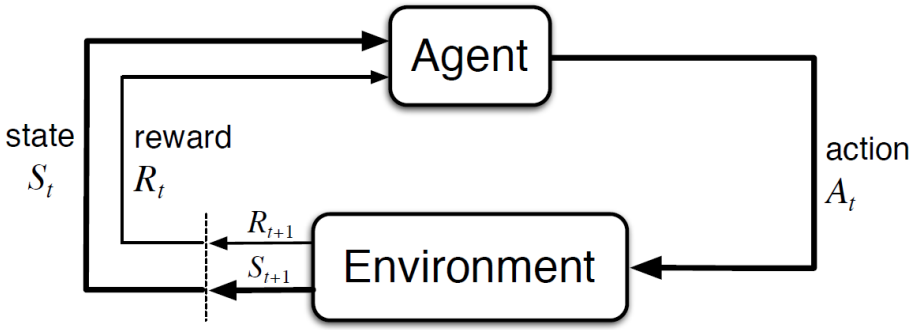

Figure 3.1: An agent interacts with an environment in an MDP

A trajectory, τ^{π}, is produced by each interaction loop. As a measure of how effective a policy is, $\pi(.)$, the agent uses the trajectory τ^{π} that is produced. The resulting trajectory consists of a sequence of states, actions, and rewards, with the rewards being feedback on how well the trajectory was executed. The trajectory is better the more rewards the agent obtains. The entire reward the agent will receive by following the policy, $\pi(.)$ is calculated using the notion of return, G^{π}.

$$G^{\pi}_{t_k} \coloneqq \sum_{n=t_k}^{\infty} \gamma^n R_n \tag{3.4}$$

Where $G^{\pi}_{t_k}$ is the return of beginning from a certain state at time t_k, following the policy $\pi(.)$.

The discount factor γ is used to alleviate the summation divergence in Eq.(3.4). Furthermore, the discount factor offers the power to affect the agent's behavior. For example, if $\gamma = 0$, the agent just attempts to maximize the immediate reward while disregarding future rewards. In this situation, the agent's behavior is deemed myopic. When $\gamma \approx 1$[1], the agent prioritizes not just the immediate reward, but also the future rewards; in this situation, the agent is deemed farsighted.

The return expression, provided by Eq.(3.4), is a random variable. Because a random variable cannot be directly optimized, the expected value of this random variable is maximized [7], [42]. The expected value of the return, $G^{\pi}_{t_k}$ is calculated using the probability distribution of the trajectory $p(\tau^{\pi})$ that leads to the generation of $G^{\pi}_{t_k}$. The agent's purpose is to maximize the expected return, which is provided by:

$$J(\pi) \coloneqq \mathbf{E}\left[G^{\pi}|\pi\right] \tag{3.5}$$

$$= \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi\right] \tag{3.6}$$

Where $J(\pi)$ is the expected long-term discounted reward that the agent seeks to maximize.

The following operation is used to find the optimal policy:

$$\pi^{\star} \in \arg\max_{\pi \in \Pi} J(\pi). \tag{3.7}$$

[1]For $\gamma = 1$, let the reward $R_n \leq R_{max}$ be bounded, then the return $G^{\pi} = \lim_{\gamma \to 1} \sum_{n=t_k}^{\infty} \gamma^n R_{max} \leq R_{max} \lim_{\gamma \to 1} \sum_{n=t_k}^{\infty} \gamma^n = \lim_{\gamma \to 1} \frac{R_{max}}{1-\gamma} = \infty$ diverges, and the condition of contraction mapping fails

Where the policy, $\pi(a|s)$, is defined as the mapping from each state $s \in \mathcal{S}$ to a probability distribution function over the action space $\mathcal{A}$.

In general, two approaches are employed to compute $J(\pi)$:

1. State-Value Function
2. State-Action Value Function

3.1.2 State Value Functions

The value of a state is the total expected return that an agent will receive by following a policy $\pi(.)$ starting from a certain state s. By assigning a value $v^{\pi}(s)$ to each state s, the agent can differentiate between being in a good state and a bad state by comparing the values of the states. The greater the state s value $v^{\pi}(s)$, the more advantageous it is for the agent to be in this state. As a result, the agent will choose the sequence of actions that leads to states with greater values. The state-value function is defined as the mapping of each state s in an MDP's state space $\mathcal{S}$ into a real value, $v^{\pi}(.) : \mathcal{S} \rightarrow \mathbb{R}, \forall s \in \mathcal{S}$. The state-value function is expressed as follows:

$$v^{\pi}(s) \coloneqq \mathbf{E}\left[G_{t_k} | S_{t_k} = s, \pi\right]. \tag{3.8}$$

3.1.3 State-Action Value function

State-action functions, as opposed to state-value functions, assign real values to each of the state-action pairs (s, a). The action that results in the higher future state value is thus directly accessible to the agent. Each possible state-action pair, $(s, a) \in \mathcal{S} \times \mathcal{A}$, of an MDP is mapped by the state-action value function into a real value, $q^{\pi}(.,.) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}, \forall s \in \mathcal{S}, a \in \mathcal{A}$. The following gives the expression for the state-action value function:

$$q^{\pi}(s, a) \coloneqq \mathbf{E}[G_{t_k} | S_{t_k} = s, A_{t_k} = a, \pi] \tag{3.9}$$

3.2 RL Algorithms

Finding an optimal strategy or policy π^* is necessary to solve an MDP. The set of algorithms known as RL algorithms is used to determine an optimal policy for an agent. There are two categories of these algorithms [7], [42] :

A) Model-based algorithms

B) Model-free algorithms

3.2.1 Model-Based RL Algorithms

Model-based algorithms are a class of algorithms that need to know the MDP dynamics beforehand. The dynamics of the MDP could be predicted ahead, for instance, by modeling the dynamics using a physics law. If the system is too complicated to model, the dynamics are estimated or learned by allowing the agent to interact with the environment and learn its dynamics. In both situations, the dynamics is used to calculate the state-value or state-action value, as shown in Eqs.(3.8) and (3.9)

3.2.2 Model-Free RL Algorithms

Rather than estimating the dynamics of the environment and then using it to construct the value functions, the value functions are directly estimated by interacting with the environment. One of the most successful model-free RL algorithms is Q-learning [46]. The algorithm's core involves iteratively updating the stat-action value function until it converges for all state and action pairs. The update rule for the Q-learning algorithm is as follows:

$$q_{tk}(s,a) \leftarrow q_{t_{k-1}} + \alpha(R_t - q_{t_{k-1}}) \tag{3.10}$$

Where $(R_t - q_{t_{k-1}})$ denotes the step size and α the learning rate..

3.3 Policy Gradient Method

The agent's memory stores state-values, state-action values (described in sections 3.1.2 and 3.1.3), and the agent's policy. These values are kept in tables for discrete MDPs. The curse of dimensionality refers to the fact that for continuous MDPs, the amount of memory needed grows exponentially with the cardinality of the state and action

spaces [47], [48]. Additionally, the quantization of continuous MDP yields a policy that depends on the number of quantization bins and is asymptotically optimal [49]. The amount of memory needed to store the discretized state space, action space, and policy increases as the number of quantization bins increases. To find a function that represents the tables of the state-values, state-action-values, and policy, function approximation techniques are utilized [42], [50], [51]. The ability of these functions to generalize for unexplored regions inside the environment, however, is restricted to the function capacity and the definition of the MDP Markovian state [42], [50].

Policy-based approaches [7], [42], [52] are presented as a result of all the aforementioned problems with value-based methodologies. In a policy-based method, the agent's policy is directly optimized. As a result, the policy is explicitly optimized rather than being derived from the state value function or state-action value function. The general framework for a policy-based approach [7], [42]:

1. Assume that the initial policy, $\pi_{\theta_0}(a_k|s_k)$, has a certain probability distribution function and is parameterized by a vector $\theta \in \mathbb{R}^n$
2. Generate a set of N-trajectories $\{\tau_i^\pi\}_{i=1}^{i=N}$ using the policy $\pi_{\theta_0}(a_k|s_k)$
3. Evaluate the N-trajectory return.

$$\hat{J}(\pi) \coloneqq \frac{1}{N}\sum_{i=1}^{N} G_i^\pi \tag{3.11}$$

4. Update the parameter vector $\theta \in \mathbb{R}^n$ to maximize Eq.(3.5), where the optimal θ is given by:

$$\theta^* \in \underset{\theta\in\mathbb{R}^n}{\arg\max}\, J(\theta) \tag{3.12}$$

   Where $\hat{J}(\pi) \to J(\pi)$ for consistent estimator.

3.3.1 Policy Gradient Theorem

Using the estimator $\hat{J}(\pi)$, one can estimate the objective function. The gradient of the objective function, Eq. (3.5), is determined in order to update the parameter vector $\theta \in \mathbb{R}^n$. However, there isn't a closed-form mathematical expression for $\hat{J}(\pi)$ that can be used to calculate its gradient. By computing the Policy Gradient (PG), Policy Gradient Theorem (PGT) provides a mathematical shortcut for determining the gradient of $\hat{J}(\pi)$.

Theorem 3.3.1 (Policy Gradient Theorem [52]–[56]). *Given an MDP, as in section 3.1, and a stochastic policy π, PGT provides a formula for determining the gradient of the objective function $J(\pi)$ with respect to the parameter vector $\theta \in \mathbb{R}^n$.*

Given :

$$J(\theta) = \mathbf{E}[G^{\pi}] = \int_{\mathbb{T}} p_{\theta}(\tau)\, g(\tau) d\tau \tag{3.13}$$

Where $p_\theta(\tau)$ denotes the trajectory density function, $g(\tau)$ is an instantiation of the return random variable G^π, and $\mathbb{T}$ denotes the space of all possible trajectories generated by the agent's policy.

Then :

$$\nabla_{\theta} J(\theta) = \mathbf{E}[\nabla_{\theta} \log p_{\theta}(\tau)\, g(\tau)] = \mathbf{E}[\sum_{k=0}^{N} \nabla_{\theta} \log \pi_{\theta}(a_{t_k} | s_{t_k})\, g(\tau)] \tag{3.14}$$

Where $\log \pi_\theta(a_{t_k} | s_{t_k})$, the only term that depends on the parameter θ, is the policy log-likelihood.

Proof. There are numerous sources for the proof of PGT [52]–[56], where the trajectory sequence needs to be defined as a vector of random variables, $\tau \sim p_\theta(\tau) = p(\tau|\theta)$, that depends on the parameter vector $\theta \in \mathbb{R}^n$. The definition of each trajectory's return is given in Eq.(3.4). In addition, the log likelihood ratio trick is employed to manipulate the trajectory probability distribution $p_\theta(\tau)$, leading to:

$$\begin{aligned} J(\pi) &\coloneqq \mathbf{E}[G^{\pi}|\pi] \\ &= \int_{\mathbb{T}} p_{\theta}(\tau)\, g(\tau) d\tau \end{aligned} \tag{3.15}$$

Given that the initial state distribution is $p(s_0)$, the trajectory density function is given by the following expression:

$$p_{\theta}(\tau) = p(s_0) \prod_{k=0}^{N} p(\mathbf{s}_{t_{k+1}} | s_{t_k}, a_{t_k})\, \pi_{\theta}(a_{t_k} | \mathbf{s}_{t_k}) \tag{3.16}$$

Using the log likelihood ratio trick[2], take the gradient of Eq. (4.4.1) with respect to the parameter vector θ:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_{\mathbb{T}} \nabla_\theta p_\theta(\tau)\, g(\tau) d\tau \\ &= \int_{\mathbb{T}} \underbrace{p_\theta(\tau)}_{\text{Probability Distribution}} \underbrace{\nabla_\theta \log p_\theta(\tau)\, g(\tau)}_{\text{Samples of } p_\theta(\tau)} d\tau \\ &= \mathbf{E}[\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta(a_{t_k} | s_{t_k})\, g(\tau)] \end{aligned} \tag{3.17}$$

□

The primary outcome of PGT is that it is not necessary to compute MDP dynamics like the one in Eq. (3.3) in order to calculate the expected long-term reward $J(\theta)$ gradient. This is a significant finding in the context of HMCo where it is assumed that agents have no previous knowledge of the dynamics of the environment. This is what is referred to as a Model-Free RL; (see section 3.2). Moreover, it is worth emphasize on the fact that the policy to learn in case of PG algorithm is a stochastic policy.

3.3.2 REINFORCE Algorithm

A stochastic gradient estimator, which is a consistent estimator, is introduced by PGT. Ronald J Williams [57], is the first to introduce the REINFORCE method (see Algorithm 1). The REINFORCE algorithm updates the parameter vector $\theta \in \mathbb{R}^n$ using Stochastic Gradient Ascent (SGA) [31], [42], [52], [53], [55], [56], [58] and the following update rule:

$$\theta^{new} \leftarrow \theta^{old} + \alpha \nabla_\theta J(\theta) \tag{3.18}$$

Algorithm 1 REINFORCE

Initialize θ arbitrarily **for** *each episode* **do**
  Generate an episode $s_0, a_0, R_0, s_1, a_1, R_1, \ldots, s_{H-1}, a_{H-1}, R_{H-1}$ using $\pi_\theta(a|s)$
  $\nabla_\theta J(\theta) = \sum_{i=1}^{H} \nabla_\theta p_\theta(\tau) G_i \;\; \theta \leftarrow \theta + \alpha \nabla J(\theta)$

Despite the fact that the REINFORCE approach produces a consistent estimate of the parameter vector θ [54], this estimate has a high variance. A base-line variable

[2] $\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)$

[52]–[55], [58] is used in order to lower the estimate's variance while keeping the estimator consistent. The idea behind the base-line is to subtract a constant value from the stochastic gradient samples, $\mathbf{E}\left[\nabla_\theta \log p_\theta(\tau)\left(g(\tau)-b\right)\right]$, which keeps the estimator consistent and lowers the estimation variance:

$$\mathbf{E}\left[\nabla_\theta \log p_\theta(\tau)\, b\right] = b\mathbf{E}\left[\nabla_\theta \log p_\theta(\tau)\right] = b\nabla_\theta 1 = 0 \tag{3.19}$$

The update rule for the gradient, given by Eq. (3.17), will have the following form once the base-line value is introduced:

$$\nabla_\theta J(\theta) = \mathbf{E}[\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)\left(g(\tau)-b\right)] \tag{3.20}$$

Because the purpose of introducing the base-line is to reduce the variance of the gradient estimate in Eq. (3.20), the following minimization problem is solved:

$$b_{minimizer} = \min_b Var(\nabla_\theta J(\theta)) \tag{3.21}$$

Where the gradient variance, $Var(\nabla_\theta J(\theta))$, is provided by:

$$\begin{aligned} Var(\nabla_\theta J(\theta)) &= \left(\mathbf{E}\left[\nabla_\theta J(\theta)^2\right] - \mathbf{E}\left[\nabla_\theta J(\theta)\right]^2\right) \\ &= \left(\mathbf{E}\left[\left(\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)\left(g(\tau)-b\right)\right)^2\right] - \mathbf{E}\left[\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)\left(g(\tau)-b\right)\right]^2\right) \end{aligned} \tag{3.22}$$

Where the second term in Eq. (3.22) equals 0, see Eq. (3.19), the variance minimization problem becomes:

$$b_{minimizer} = \min_b \mathbf{E}\left[\left(\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)\left(g(\tau)-b\right)\right)^2\right] \tag{3.23}$$

Taking the gradient of Eq. (3.23) with respect to the base-line variable, then calculate

the zeros of the resulting derivative:

$$\begin{aligned}
&\nabla_b \mathbf{E}\left[\left(\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)\left(g(\tau)-b\right)\right)^2\right] \\
&= -2\mathbf{E}\left[\left(\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)^2\left(g(\tau)-b\right)\right)\right] \\
&= \mathbf{E}\left[\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)^2 g(\tau)\right] - b\mathbf{E}\left[\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)^2\right] \\
&= 0
\end{aligned} \tag{3.24}$$

As a result of solving Eq. (3.24) for b, the base-line value is as follows:

$$b_{minimizer} = \frac{\mathbf{E}\left[\left(\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)\right)^2 g(\tau)\right]}{\mathbf{E}\left[\left(\sum_{k=0}^{N} \nabla_\theta \log \pi_\theta\left(a_{t_k}|s_{t_k}\right)\right)^2\right]} \tag{3.25}$$

The expected return, weighted by the square of the PG, produces the base-line $b_{minimizer}$. As a result, the baseline introduces a method to measure how well our agent performs in comparison to his past performance. Furthermore, this is analogous to centering the incoming data in the form of reward $g(\tau)$ on the average reward value of the agent's performance $b_{minimizer}$. Due to the base-line being subtracted from the trajectory total reward, $(g(\tau) - b)$, this will reduce the reward oscillation during the learning process. Algorithm 2 provides the modified REINFORCE algorithm using the baseline value.

Algorithm 2 Policy Gradient Reinforcement Learning

Result: θ

initialization: Parameter vector, Policy **while** θ *not converged* **do**

  while g_{RF} *not converged* **do**

    Generate a Trajectory $S_{0:H}, A_{0:H}, R_{0:H}$ using Policy $\pi_\theta(s,a) = P[a|s,\theta]$

    $b_{minimizer} = \frac{\mathbf{E}\left[\left(\sum_{i=0}^{H} \nabla_\theta \log \pi_\theta(\mathbf{s}_i|\mathbf{a}_i)\right)^2 g(\tau)\right]}{\mathbf{E}\left[\left(\sum_{i=0}^{H} \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\right)^2\right]}$

    $\mathbf{g}_{\mathsf{RF}} = \mathbf{E}\left[\left(\sum_{i=0}^{H} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_i|\mathbf{s}_i\right)\right)\left(g(\tau) - b_{minimizer}\right)\right]$

    $\nabla_\theta J|_{\theta=\theta_j} = g_{RF}$

  end

  $\theta_{j+1} = \theta_j + \alpha_j \nabla_\theta J|_{\theta=\theta_j}$,

end

4 Methodology: Coadaptation Problem Within the Reinforcement Learning Framework

In chapter 1, the HMCo problem is presented. Two co-adapting, co-learning agents interacting to complete a shared task are the focus of the HMCo problem. The interaction or sharing control over the task can be direct or indirect. In the instance of direct interaction to accomplish the predetermined task, the human and the machine are directly interacting through the task itself and affecting the environment that both of them are part of. On the other side, in an indirect interaction, a human uses a machine to execute a task. This implies that the machine itself is the agent that needs to adapt its behavior to the human behavior and accomplish the task. Furthermore, the human and the machine have explicit knowledge of the desired task.

In chapter 3, RL techniques were presented with the goal of determining an optimal policy for an adaptive agent interacting with its environment (see section 3.2). In this thesis, a predetermined task needs to be fulfilled by a machine that is adaptive while interacting, cooperating, helping, and sharing control with a human. Machine co-adaptation is abstracted by an optimal policy that the machine will learn in order to do the task as designed. The PG-algorithm, which was covered in the chapter before, is used to implement this. The PG algorithm was chosen since neither the dynamics of the environment nor an explicit model of humans are required. The task must be fulfilled along with the assumption that the human behaves rationally when interacting with the machine.

One of this thesis' main merits is that the HMCo problem is formulated on the assumption that neither the machine nor the human has any prior knowledge of the dynamics of the environment. In addition, neither the human nor the machine has any internal representations for the models of the other. As a result, neither the machine nor the human can directly anticipate the other's behavior using a model of the other. This chapter builds on chapter 1 and chapter 3 to establish a general mathematical framework for the HMCo problem.

4.1 Human Operator

Many neuroscientists adopt the RL framework described in chapter 3, to explain how humans adapt to their environment [59]–[63]. One type of monoamine neurotransmitter produced by the hypothalamus is dopamine. Dopamine is also responsible for transmitting messages from the brain to the rest of the body [64]–[66].Dopamine is a component of the reward system in the human brain and is what triggers the Basal Ganglia's "reward prediction error" signal. Which is utilized to generate human behavior, decisions, or a sequence of decisions [67], [68]. The reward system is intended to either reward humans for engaging in positive acts or behaviors or, on the other hand, punish humans for engaging in negative behavior. According to [62], dopamine neurotransmitters are released in proportion to the state value or scenario value. As previously stated, the RL framework provides the best explanation and understanding of the function and mechanism of the development of human behavior.

As shown in figure 3.1, the RL framework formulates the learning problem by utilizing interaction information. If a human is the agent, she or he chooses an action based on its values by utilizing the state-value function or the state-action value function, which are represented by $V(s)$, and $q(s, a)$, respectively. Therefore, for a certain context or state s, the human chooses the action that is most likely to result in a larger expected long-term reward or accumulated reward. The human could be unaware of how to act at the beginning of an interaction for a specific context or task.However, as soon as the human begins interacting to achieve the task, she/he attempts to incorporate the information from the interaction to improve her/his behavior and encodes this information as a state-value function $V(s)$ or a state-action value function $q(s, a)$. As the interaction proceeds, the human learns how to behave properly in a given situation or state. The adopted RL paradigm and the dopamine neurotransmitter theory suggest that humans eventually reach a point where any new situation has no impact on their knowledge or behavior. The dopamine keeps firing and producing an electrical signal to and from the central nervous system as the human learns more and more about the appropriate behavior to perform in each situation. The mathematical idea of "reward prediction error" is used to express this electrical signal. The following equation captures the entire concept of knowledge accumulation:

$$q_{k+1} = q_k + \alpha(r_{k+1} - q_k) \tag{4.1}$$

$$= q_k + \alpha\delta_{k+1} \tag{4.2}$$

Where δ_{k+1} is the mathematical representation of reward prediction error.

The reward prediction error δ_k drives the human learning process. If $\delta_k \neq 0$, the human is in a new situation from which she or he can learn. The state value, or the state-action value, is updated as a result of the reward prediction error coded by dopamine. As a result, the human now has new knowledge about how to behave in such a situation. δ_k might be positive, negative, or zero. Positive and negative values reinforce or punish the related behavior, respectively. Because dopamine is not released or produced in the case of $\delta_k = 0$, the human is neither reinforced nor punished. This signifies that the human is in a situation that she or he has been in before and is aware of the correct form to behave. Consequently, Eq.(4.2) is not updated, as stated by the adopted RL framework.

According to [69], the intention of a human can be inferred from their behaviors or actions. The assumption in this thesis is that the human is rational to choose a sequence of actions that would accomplish the task, i.e., to maximize the objective function provided by Eq. (3.5). For the machine to recognize the human intention, the author of [69] suggests a Dynamic Bayesian Approach. Mathematically, the human intention is expressed as a discrete random variable $.I_n \in \mathcal{I}$, where $\mathcal{I}$ is the collection of human intentions. The work given in [70] draws some inspiration from the way mammals improve their action probability to maximize their reward by updating the human action probabilities using Bayesian methods and incorporating information, such as the intention. As a result, the theory from [70] is used to describe human rationality and how humans update the likelihood that one choice of action is preferable to another in an effort to maximize expected reward. Assuming that the reward prediction error is zero, or $\delta = 0$, then the maximum expected return, q, that an agent can accumulate is obtained. That is, the agent, in this example the human, already knows what action/s to take in the current situation, because the situation adds no information to the human's knowledge. As was previously stated, human behavior is related to the reward prediction error, which is to reinforce or punish human specific behavior or a sequence of actions. For an example of how to represent a human's intention over time using the structure provided in [69] (see figure 4.1). The current state, current action, and current intention are represented by the random variables S_t, A_t and I_{nt}, respectively. The reward error prediction, δ, and the intention random variable, I_n, are both considered to be continuous in this thesis. To clear up any misunderstandings, this thesis is not intended to infer human intention. Additionally, it is assumed in this thesis that the intention is to achieve the goal, where the goal is decided by the reward function. A proper sequence of actions will be followed in accordance with the task goal, given the human intention.

The process a human undergoes to optimize their odds of choosing the optimal action given their intention to accomplish the task successfully is illustrated by the following

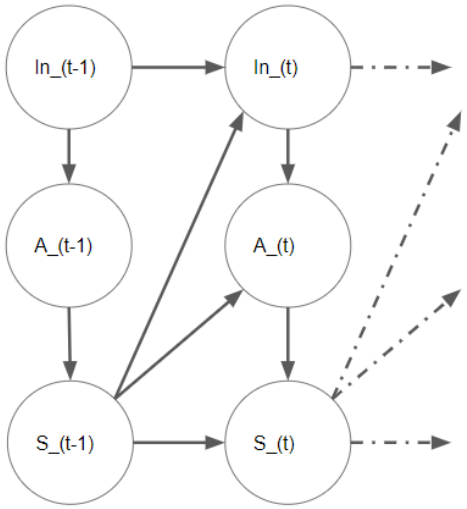

Figure 4.1: A segment of the human intention process to generate an action

steps:

1. In order to complete a predefined task, the human agent chooses an action $a_h \sim p(a_h|s)$ while interacting with the environment. The distribution $a_h \sim p(a_h|s)$, depicts how people choose their actions depending on their training and habits. It also includes the uncertainty that arises from several causes, such as stress, mood, and others.

2. The human receives a specific reward as a result of the action a_h decision, which mathematically corresponds to the reward prediction error δ, which is also a random variable.

3. Then, what follows naturally is that the human becomes adapted to this action, a_h, in a certain context for a specific task goal, which corresponds mathematically to $\delta = 0$. It is therefore hypothesized in this thesis that the intention to complete the task is directly proportional to the variable delta, $I_n = \delta$. To maximize expected accumulated rewards in converged rational behavior, $I_n = \delta = 0$.

Consequently, it makes sense to express the probability of human action given a certain situation and intention as the following:

$$p(a|s, I_n = \delta) = \frac{p(I_n = \delta|s, a)p(a|s)}{p(In = \delta)} \tag{4.3}$$

The fact that δ provides information about the reinforce/punish direction while updating the value function $q(s, a)$ makes the assumption that $I_n = \delta$ plausible. In addition, as was already indicated, the human maximizes the choice of the rational/optimal

action when $\delta = 0$, which indicates that the human is able to choose the optimal action.

The reward prediction error, $\delta \neq 0$, indicates that the human should indeed try different actions in order to optimize the reward or return. If the $\delta = 0$, on the other hand, the human is sufficiently confident in his or her understanding of the current circumstance and is able to choose the action that maximizes the expected return. Furthermore, it is understandable to believe that the state and the action have no effect on the human intention if the human intention is to maximize the expected reward by accomplishing the goal, which is indicated by the mathematical expression $p(In = \delta = 0|s, a) = p(In = \delta = 0)$.

This results in the probability distribution of the human action being as follows:

$$p(a|s, In = 0) = p(a|s) \tag{4.4}$$

The previous expression implies that humans acquire the capacity to choose actions that maximize reward. According to the prior expression, a human can choose to take actions that will maximize their reward. Eq.(4.4) is only applicable at $I_n = \delta = 0$ since the situation and human actions do not influence the human intention to satisfy other desires.

This section discussed how humans behave under the assumption that their intention is to maximize reward or minimize reward error prediction. This seamlessly adheres to the RL structure that is described in chapter 3. The human is therefore represented as an RL agent that minimizes reward error prediction while choosing behaviors to maximize reward.

4.2 Machine model

Both the ability to learn and the ability to be adaptive are presumed capabilities of the machine. In accordance with the RL structure, these two abilities are attained. As a result, the machine is an RL agent with an internal policy, and the agent's purpose is to learn an optimal policy. As a result, the machine develops the ability to simultaneously co-adapt to human behavior during interaction. which led to efficient task completion by the two agents.

The task goal, reward function, and human rationality are all made known to the machine. In the scenario of indirect interaction, the following provides the machine

policy:

$$a_m \sim p(a_m|goal, state) \tag{4.5}$$

Whereas the probability distribution function, $p(.)$, could be any admissible probability distribution function, such as a Gaussian distribution[1]. Eq. (4.5) parameters may change when there is direct interaction.

The probability distribution and its parametrization for the machine policy contain the machine's co-adaptation and co-learning capabilities. The parameters of the probability distribution function are learned via the interaction between the human and the machine while performing the task. Human actions lead to machine actions. Learning the mapping from human actions to machine actions is the objective of indirect shared control. In contrast, the objective of direct shared control is to directly learn the actions. In both situations, the aim of the interaction is for the machine to develop the capacity for co-adaptation to human behavior.

4.3 HMCo problem formulation

Two co-adapting agents interacting to complete a predetermined task is known as the HMCo problem; for more information, (see chapter 1). Previous sections defined the terms "human," "machine," "co-adapting," and "co-learning." The environment in which the interaction occurs, as well as the mathematical formulation of the problem, are addressed. Since the HMCo problem is modeled as involving two co-adaptive agents, the Two-Agent Markov game [71] is a potential framework to model the HMCo. The problem could be modified to suit the MDP concept introduced in section 3.1.1 because the human in this study is introduced as being rational and the machine agent has to learn an optimal policy to co-adapt to the human's behavior.

In the HMCo-MDP, A_h stands for the human action space, whereas A_m stands for the machine action space. The machine actions, which are conditioned on the task, are assumed to be the mapped actions into the human action space in order to fulfill the task. The machine action parametrization shown below is an example of indirect interaction.

$$A_m = R(\theta, A_h) \tag{4.6}$$

Where $R(.)$ takes different meanings depending on the scenario as discussed in the sequel.

[1] It is critical to emphasize that the aim is not a random variable, and hence one cannot condition on it. The convention is abused to emphasize the fact that the machine has explicit knowledge of the task while performing its actions

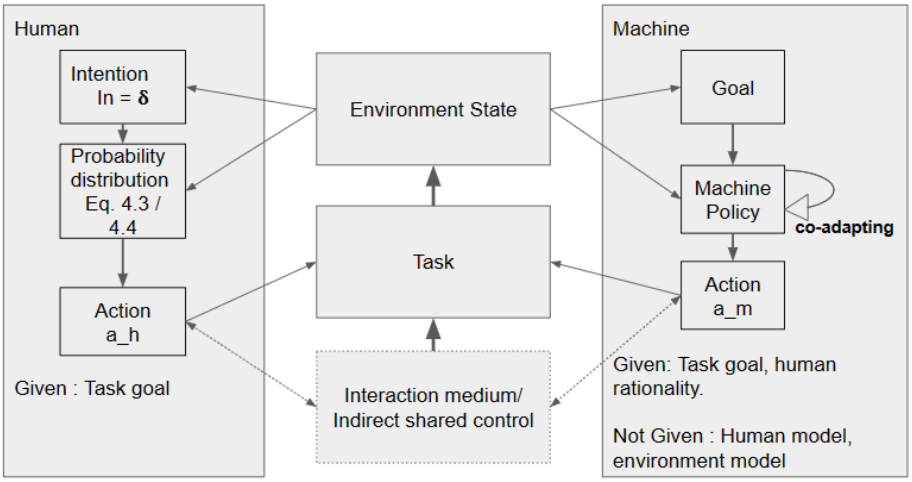

Figure 4.2: The general HMCo framework

Given the task goal, the machine goal is to learn an optimal policy $\pi(a_m|state)$, which is accomplished by learning the optimal mapping $R(.)$ that optimizes the machine policy. In the case of indirect shared control, for example, it is assumed that the human manipulates the task via an interface device, such as a joystick, which is considered to be co-adaptive. The machine will then decide which joystick parameter is optimum for accomplishing the task. The machine's objective in direct shared control is to execute actions that, when combined with human actions, are most efficient at achieving the task goal. Consider the game SeaSaw, where a human and a machine work together to establish equilibrium without oscillation. In order to achieve the goal, the machine should then generate actions that best match the human actions. Take into account that the definition of the mapping, $R(.,.)$ varies based on the task and the state of the environment. Figure 4.2 demonstrates the overall HMCo problem.

4.3.1 MDP for HMCo

HMCo-MDP is defined as a two-agent Markov game with the following components:

1. The environment state space, $\mathcal{S} \in R^n$, contains all possible configurations of all agents acting in the environment
2. The action space, $\mathcal{A}$, is subdivided into two parts, $\mathcal{A}_h \in R^{h_a}$ and $\mathcal{A}_m \in R^{m_a}$, which represent human and machine actions, respectively
3. A set of observations, $\mathcal{S}_h \in R^{h_s}$ and $\mathcal{S}_m \in R^{m_s}$ for the human and machine, respectively

4. The probability distribution for transitions[2], $\mathcal{P} : \mathcal{S} \times \mathcal{A}_h \times \mathcal{A}_m \to \Delta(R^n)$
5. The reward function of each agent, $\mathcal{R}_h : \mathcal{S} \times \mathcal{A}_h \times \mathcal{A}_m \to \Delta(R^1)$ and $\mathcal{R}_m : \mathcal{S} \times \mathcal{A}_h \times \mathcal{A}_m \to \Delta(R^1)$

Where R^1,R^n, R^{h_a}, R^{m_a}, R^{h_s} and R^{m_s} denote the $one-$, n-dimensional, h_a-dimensional, m_a-dimensional, h_s-dimensional and m_s-dimensional real number spaces, respectively. The spaces of all probability distributions over R^n and R^1 are denoted by $\Delta(\mathcal{S})$ and $\Delta(R^1)$, respectively.

As illustrated in section 1.2, the common goal is one of the HMCo components. This component requires that the reward definitions of both agents be the same, i.e. $\mathcal{R}_h$ and $\mathcal{R}_m$ are the same. The dynamics of the environment are also unknown to the two agents, and each agent alters and interacts with the environment while opting to undertake a specific action. The human rationality and the PG algorithm enable the machine to co-adapt to human behaviors by leveraging data from the interaction in the form of a trajectory τ.

4.3.2 Trajectory for HMCo

Consider the trajectory, $\tau^{h,m}$, formed by the interaction of the two agents in the HMCo framework, starting with $s_0 \sim p(s_0)$:

$$\tau^{h,m} = (s_0, a_0^h, s_1^h, a_0^m, s_1^m, ...) \tag{4.7}$$

Where :

- The pair (a_i^h, s_i^h) represents the human action and the state that results from that action as perceived by the human
- The pair (a_i^m, s_i^m) represents the machine action and the state that results from that action as perceived by the machine

It should be noted that the human and machine states may be identical. The trajectory definition in Eq.(4.7), is a typical definition for the multi-agent MDP [72]–[74]. However, due to human rationality and MDP's abstraction power, the trajectory is reformulated using the MDP definition in section 3.1.1 to represent the two-agent Markovian environment, as follows:

$$\tau^m = (s_0, a_0^m, r_1^m, s_1, a_1^m, r_2^m, s_2, ...) \tag{4.8}$$

[2]Remember that all agents acting in the environment are presumed to be unaware of the environment's dynamics

Where :

- s_i is the agent's perception of the environment, which captures the transition information caused by human action $a_i^h \rightarrow s_i^h$
- a_i^m represents the i^{th} machine action
- r_i^m is the reward given to the machine for executing the action a_i^m within the state s_i

For the trajectory τ^m, human rationality is regarded as being encoded inside the state observed by the machine. Consider this as adding more features to the state vector, which improves generalization capacity while learning the machine policy [42], [51]. The trajectory, Eq.(4.8), represents the interaction information, (s, a, r), and the machine agent should adapt its policy by utilizing this knowledge.

4.4 Improved Policy Gradient (PG) Algorithm

The improved PG algorithm, Algorithm 3, is a variation of the policy gradient algorithm,(Algorithm 2, section 3.3.2), that incorporates a co-adaptation loop to allow the machine to adapt to the human behavior during interaction. The algorithm consists of three main loops: a policy iteration loop, an interaction loop, and a policy co-adaptation loop.

The policy iteration loop is used to optimize the policy function, which maps the state of the environment to an action. This loop is similar to the one used in the original policy gradient algorithm, where the parameters of the policy function are updated to maximize the expected reward. The interaction loop is used to collect data from the interactions between the human and the machine. This data is used to update the machine's understanding of the human's behavior.

The policy co-adaptation loop is used to adjust the machine's policy function to better align with the human's behavior. The machine uses the data collected during the interaction loop to update its policy function, which allows it to co-adapt to the human's behavior. Overall, the algorithm allows for a dynamic and adaptive interaction between the human and the machine, where the machine is able to adapt its behavior to better align with the human's behavior. The algorithm is an extension of the original policy gradient algorithm, which allows for a more robust and efficient interaction between the human and the machine.

Algorithm 3 Policy Gradient Algorithm

Initialization $\theta_0 \in R^H, s_0, M_{max}, N_{max}, K$
input : $\alpha, \gamma, \Sigma, \theta \in R^H$
output : $\theta^* \in R^H$

while $(\theta \neq converged)$ **do**
  $\vec{u}^{\theta}_{human}$ = `HumanAction ()`
  $N = 1$
  while $(g_{RF} \neq converged)$ **do**
    for *k = 1* **to** *K* **do**
      $a_{machine} \sim \pi_\theta(a|s_{k-1})$
      $s_k, r_k, gll_k, done$ = `OneStep` (s_{k-1}, a)
      $R = r_k + \gamma * R$
      $\nabla gll = \nabla gll + gll_k$
      if $done$ **then** do
        Generate a new target at random (for indirect interaction)
        Generate a new starting state at random (for direct interaction)
        $N = N_{max}$
        break
      end
    end
    $N = N + 1$
    for *h = 1* **to** *H* **do**
      $b^h_N = (N-1) * b^h_N + \nabla gll(h)^2 * R$
      $b^h_D = (N-1) * b^h_D + \nabla gll(h)^2$
      $b^h = b^h_N / b^h_D$
      $g^h_{RF} = ((N-1) * g^h_{RF} + \nabla gll(h)^2 (R - b^h))/N$
    end
    if g_{RF} *is converged* $\vee$ $N > N_{max}$ **then** do
      break
    end
  end
  if g_{RF} *is converged* **then** do
    $\theta = \theta + (\alpha_{t_j} + \lambda_t \Lambda^2) g_{RF}$
    $M = M + 1$
  end
  if θ *is converged* $\vee$ $M > M_{max}$ **then** do
    break
  end
end

The improved PG algorithm, Algorithm 3, is explained for each line in the following.

- **Initialization, Input and Output** : These lines contain the hyperparameter settings for the algorithm, including the maximum number of trajectories N_{max}, the maximum number of policy iterations M_{max} and the maximum number of interaction steps K. As well as the initial conditions for the parameter vector θ, and the initial state s_0.

- **Policy iteration loop**(Line-1 - Line-34): This is the primary outer loop, and the machine executes a new policy for each step within it when the policy parameter vector is updated

  - **Human Action**(Line-2): This function is in capable of generating human rational action. However, in the case of two machines interacting together, this would be one of the machines' actions, as the following chapters will illustrate

  - **The co-adaptation loop** (Line-4 - Line-22) : The co-adaptation of the machine to human actions is accomplished via this inner loop. This is the fundamental improvement over the basic PG algorithm, Algorithm 2. The machine produces interaction trajectories τ^m within this loop in order to co-adapt to the human-executed action.These trajectories or experiences are generated until the policy gradient converges. The convergence of the policy gradient denotes the machine's optimal co-adaptation to the given human action. To prevent an infinite loop, a fixed maximum number of trajectories, N_{max}, is introduced

    * **Experience/Interaction loop**(Line-5 - Line-15): This loop generates the experience or interaction trajectory τ^m

    * **Policy gradient and Baseline loop**(Line-17 - Line-22): For each trajectory τ^m created by the **Experience loop**, this loop updates the policy gradient and the baseline

  - **Co-adaptation convergence**(Line-23 - Line-30): If the convergence of the policy gradient is true, the policy parameter is changed. If it is false, a new loop of co-adaptation is initiated using the old policy, and the parameter vector, θ, is not updated

- **Policy convergence**(Line-31 - Line-34): In two cases, the co-adaptation loop terminates. The first case, if the policy is converged, is $\theta \approx \theta_{optimal}$. The second case occurs when a predetermined number of policy gradient updates is surpassed, which is M_{max}

In comparison to the update rule introduced in Eq. (3.18), Line-31 contains a new update rule. This way of updating is known as the Levenberg-Marquardt method [75]. To the best of our knowledge, this update rule has never been used in the PG algorithm.

$$\theta = \theta + (\alpha_{t_j} + \lambda_t \Lambda^2) g_{RF} \tag{4.9}$$

Where:

- α_{t_j} is the learning rate that is varying with time , in this thesis $\alpha_{t_j} = 1/t$
- λ_t in this thesis it is called the exploration rate in gradient space

Levenberg-Marquardt method takes into account the curvature of the objective function, which helps to prevent overshooting or oscillations in the search for the optimal solution. A common choice for Λ is identity in cases where it is hard to compute the Hessian of the objective function or the objective function has no closed form [75].

The Levenberg-Marquardt method uses λ to balance the contribution of the gradient term and the curvature term. When λ is large, the algorithm behaves more like the Gauss-Newton method, which is sensitive to the curvature of the objective function. When λ is small, the algorithm behaves more like the gradient descent method, which is sensitive to the gradient of the objective function. This improves the convergence rate and the stability of the update process.

4.4.1 Exploration - Exploitation Dilemma

The choice of whether to explore or exploit is fundamental in the field of RL. Exploration is the process of an agent continuously testing, often at random, every action that might be taken in order to consider all possible consequences. Whereas exploitation implies that the agent continues with the action that results in the highest possible reward for a given situation [42], [52], [53], [56]. During the exploration phase, the agent learns more about the environment it interacts with [42]. In other words, the agent creates data through exploration that will be used for learning. The agent will exploit this data to further learn about the consequences of each action. Better learning often results from more data. The agent's ability to utilize (exploit) this newly created data in order to generate or produce a better decision is called exploitation .Through exploitation, the agent may select the action that, according to a given metric, such as the Q-value function, appears to have a better chance of achieving the task goal. Within the HMCo framework, the exploration rate encodes the agent's

ability to co-learn, while the exploitation of the data generated while co-learning encodes the agent's ability to co-adapt.A probability distribution function represents the agent's policy in the case of PG. As a result, the exploration-exploitation concept is implicitly used in the PG algorithm [31], [53], [58].The policy of the machine is assumed to be a Gaussian probability distribution function, and the exploration of the machine is controlled by the Gaussian covariance matrix. The following is the general form of a Gaussian policy:

$$a \sim \pi(.|s) = \mathcal{N}(\mu_{a|s}, \Sigma_{a|s}) = \frac{1}{(2\pi)^{m/2}|\Sigma_{a|s}|} \exp \frac{-(a-\mu_{a|s})^T \Sigma_{a|s}^{-1}(a-\mu_{a|s})}{2} \tag{4.10}$$

Where:

- $a \in R^m$ is an action sampled from the policy $\pi(.|s)$
- $s \in R^n$ is the MDP's state
- $\mu_{a|s} \in R^m$ is the mean value for the action a given the state s
- $\Sigma_{a|s} \in R^{m \times m}$ is the covariance matrix for the action a given the state s

A Gaussian probability distribution function's covariance matrix, $\Sigma_{a|s}$, governs the machine's ability to balance exploration and exploitation. Every time the policy is updated, the machine exploration rate decreases. This enables exploration at a lower rate while also enabling the machine to exploit (make use of) the new policy to accumulate more rewards. The exploration rate eventually falls to a predetermined minimum value, which is typically greater than zero and aids in escaping local minima during the co-learning and co-adaptation processes.

4.4.2 Time Complexity

The time complexity function, $T(.)$, for Algorithm 3 is derived in this section. There are four nested loops in Algorithm 3:

- **Policy iteration loop**: This is the outer loop, and it is run up to M_{max} times. This loop contains the **Co-adaptation loop**
- **Co-adaptation loop**: This loop is executed at most N_{max} times and contains the two loops listed below:
  - **Experience/Interaction loop**: The agent generates a trajectory with a maximum length of K, therefore this loop is run a maximum of K times.

- **Policy gradient and Baseline loop**: The maximum number of times that the policy gradient and baseline updates are performed out is H, where H is the dimension of the parameter vector θ.

To determine the time complexity order of this algorithm, the total number of operations in the algorithm have to be counted. The following is a general overview of the time complexity calculation rules:

- Let $T_1(n) = \mathcal{O}(f(n))$ and $T_2(n) = \mathcal{O}(g(n))$, then :
  - $T_1(n) + T_2(n) = \mathcal{O}(max(f(n), g(n)))$
  - $T_1(n) * T_2(n) = \mathcal{O}(f(n) * g(n))$

Using these rules and tracing the loops in Algorithm 3, it is evident that this algorithm's time complexity is provided by:

$$T_{PGA} = \mathcal{O}(M \cdot N \cdot max(K, H)) \tag{4.11}$$

The interpretation of the $max(\mathcal{O}(K), \mathcal{O}(H))$ term to the right is interesting. For each iteration to estimate the gradient, the agent must create a trajectory with time complexity of order $\mathcal{O}(K)$ and loop through all of the parameter variables in the parameter vector $\theta \in \mathcal{R}^H$.Therefore, the number of variables in the parameter vector θ has no impact on the time complexity of this algorithm, at least in theory, if the machine agent is able to accomplish $K > H$ steps before ending its trajectory.

Additionally, vectorization might be used to implement lines 20-25, which would accelerate the execution of this process even more than $\mathcal{O}(H)$. Intuitively, the size of the parameter vector θ, H, has no impact on the algorithm time complexity if the machine agent spends more time generating data, or trajectories, in order to co-adapt to the human behavior for the same task goal. The interaction time steps dominate the algorithm time complexity as a result. By order of T_{PGA}, the local minimum of this algorithm is guaranteed to be reached [31], [42], [52], [53], [55], [56], [58]. However, increasing the dimension of the parameter vector increases the complexity and non-linearity of the objective function, resulting in more local minimums in which the agent may get stuck. However, due to the nature of the stochastic policy within the PG framework, the agent might overcome some of these local minimums (see section 3.3.1).

5 Results: Case Studies

The proposed technique and corresponding algorithm are implemented in two main scenarios: indirect shared control and direct shared control. In the indirect shared control scenario, a human interacts with a joystick to track and reach a target on the screen. The target's position changes each time the human reaches it. The joystick deliberately mismaps human actions. In other words, when the joystick is moved in the direction d_1, it generates d_2. In the direct shared control scenario, the human and machine interact by playing the SeeSaw game. Reaching the SeeSaw's equilibrium point without oscillation is the objective of this task. In both scenarios, it is desired that the machine chooses actions to co-adapt to human behavior continuously. In these two scenarios, a machine and a human interact together to accomplish a common goal. Co-adaptation is accomplished in the HMCo framework by two agents who can co-learn and co-adapt simultaneously. In chapter 4, it is demonstrated how RL approaches, in which an adaptive agent is assumed to interact with the environment, can take into account co-learning and co-adapting abilities (see section 4.4.1). Humans are regarded as rational in the sense that they are able to assess every situation and decide what is best to do. The RL agent achieves this level of performance by interacting with the environment and minimizing the reward-prediction error δ_k. This leads to the introduction of a third scenario, in which two learning agents are collaborating to complete the task. This scenario is similar to the SeeSaw case, but instead of a human, a machine is used. It is assumed in the first two scenarios and for simulation purposes that the human has already learned and reached optimality. To demonstrate the effect of co-learning and co-adaptability, the third scenario is chosen in which two agents (two machines) play the SeaSaw and both of them explicitly perform RL. The two machines therefore have to interact in order to co-adapt to one another's actions in order to achieve the same goal, which governs the interaction.

The HMCo framework with the PG algorithm is introduced and discussed in the following sections for each of these three scenarios. It is worth mentioning that in RL community the expected long-term expected reward is the only value used to assess and measure the performance of a given algorithm [42]. The discussion of other quantities like the policy parameter convergence and the policy gradient is just to give a context for the algorithm's performance.

5.1 Indirect Shared Control Scenario

This scenario is based on [6]. The case contains all of the HMCo components. In order to follow a target on the screen, the human interacts with a joystick. The target moves to a new position randomly once the human reach it. The cursor on the screen does not move as the human intends, but rather moves in a twisted direction due to a wrong mapping between the received movement (the command) and the implemented movement (the cursor on the screen). The machine should fix this problem by learning and adaptation.

5.1.1 The Environment Mathematical Model

To be able to simulate this scenario, a mathematical model for the environment is needed. This model is not used for learning (through PG algorithm) since learning is model-free, as explained previously.

- **The Human Model**: As it is assumed that humans will behave in an optimal way, the human generates the action u_{human}, which, if correctly translated by the joystick, leads to the target. The human action is given by:

$$u_{human} = |u_{human}| \angle \gamma_{target} = (u^x_{human}, u^y_{human})^T \in R^2 \tag{5.1}$$

  where $|u_{human}|$[1] denotes the magnitude of the applied speed by the human, γ_{target} is the angle from the cursor position on the screen to the target position on the screen, u^x_{human} is the horizontal component of the applied human velocity, and u^y_{human} is the vertical component of the applied human velocity. The dimension of the action space in this case is $m = 2$.

- **The Joystick Model**: The joystick is represented as a twisted mapping from human action, u_{human}, to joystick action, $u_{joustick}$. The joystick action is then given by:

$$u_{joystick} = R(\beta_{twist}) u_{human} \in R^2 \tag{5.2}$$

  where $R(.) \in R^{2\times 2}$ is a rotation matrix[2] representing the joystick's twist and β_{twist} is the twisting angle. It is assumed that $|u_{joystick}| = |u_{human}| = 1$.

[1] The symbol $|.|$ denotes the 1-norm

[2] A rotation matrix is a square matrix for which $RR^T = I$

- **The Machine Policy**: The machine observes the result of the mismapped human action, $u_{joystick}$. The machine's policy then chooses the action $u_{machine}$, which is given by:

$$u_{machine} \sim \pi_\theta(.|u_{joystick}) = \mathcal{N}(R(\theta)u_{joystick}, \Sigma) \tag{5.3}$$

  where the machine's policy is given by a Gaussian probability distribution function around $u_{machine}$, $R(.)$ is a rotation matrix, the angle θ is the policy parameter, and Σ is the policy covariance or exploration rate.

- **The Dynamics or the Transition Model**: The transition model for this scenario's environment is given by:

$$p_{t+1} = p_t + \Delta t u_{machine} \in R^2 \tag{5.4}$$

  where $p_t in^2$ represents the t^{th} cursor position and Δt represents the sampling time between two consecutive positions

- **The Reward Function**: In order to quantify how well the machine responds to a specific human action, a feedback signal is required. This feedback signal is the immediate reward signal r_t. The proposed reward function is given by:

$$r_t = 1 + \frac{|d_{t-1}| - |d_t|}{|p_t - p_{t-1}|} \tag{5.5}$$

  where :

  - d_t is the distance between the cursor t^{th} position on the screen and the target current position[3]
  - p_t is the t^{th} cursor position
  - If $|p_t - p_{t-1}| = 0$, the immediate reward $r_t = 0$

  For a perfect motion toward the target and a perfect motion away from the target, the suggested reward function gives $+2$ and 0, respectively.

The goal is that the machine learns a policy for selecting actions to co-adapt to human behavior, that is indirectly observed through $u_{joystick}$, using the PG algorithm, with the human-machine interaction being as follows (see figure 5.1):

- Do until co-adaptation achieved:

[3]The target position changes randomly once it is reached

1. The human rationally generates what she/he believes to be the best or optimal action,u_{human}, by moving the joystick in accordance with the target on the screen
2. The joystick mismaps human actions,$u_{joystick}$, so the action displayed on the screen,$u_{machine}$, is not the intended human action
3. The machine policy receives a scalar reward signal that indicates how well it responded to the human action
4. The machine co-adapts to human actions, given the twisted joystick, using the PG algorithm based on the reward signal
5. The machine policy corrects (compensates) gradually for the joystick twisting

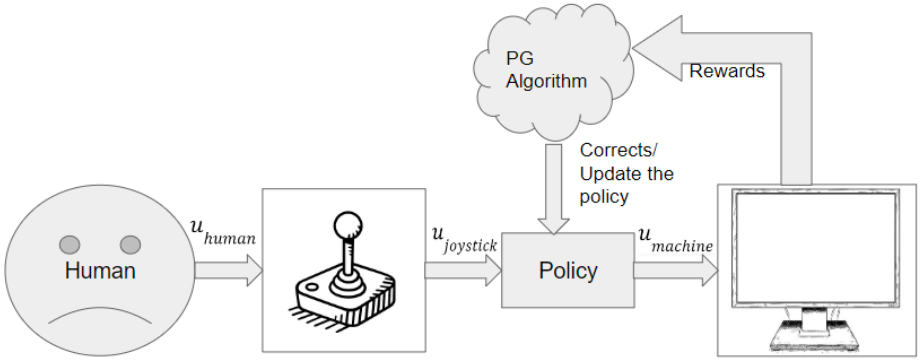

Figure 5.1: The problem of indirect shared control

5.1.2 PG algorithm implementation and results

The PG learning algorithm is demonstrated in Algorithm 3.Under the following hyper-parameter settings, the algorithm is run:

1. The screen size is 70x70 units
2. The learning rate is $\alpha = 0.9$
3. The discount factor is $\gamma = 0.85$
4. The exploration rate is $\Sigma = 0.1$
5. The sampling time is $\Delta t = 1/60$

Different joystick twisting angles, or β_{twist}, are examined in the sections that follow. According to [42], the performance of the RL agent is assessed using the long-term expected reward, or cumulative reward, of independent runs. To obtain the results, 100-independent runs are performed. Each run comes to an end when the agent's policy converges. For example, figure 5.2 shows the trajectory for 10-independent runs, where each run converges to a local optimal policy. The average over these runs is computed to measure the performance of the machine for a given β_{twist}. The simulation is run on a laptop with a Core i5-3230M processor running at 2.6 GHz, four CPUs, and 8 GB of RAM.

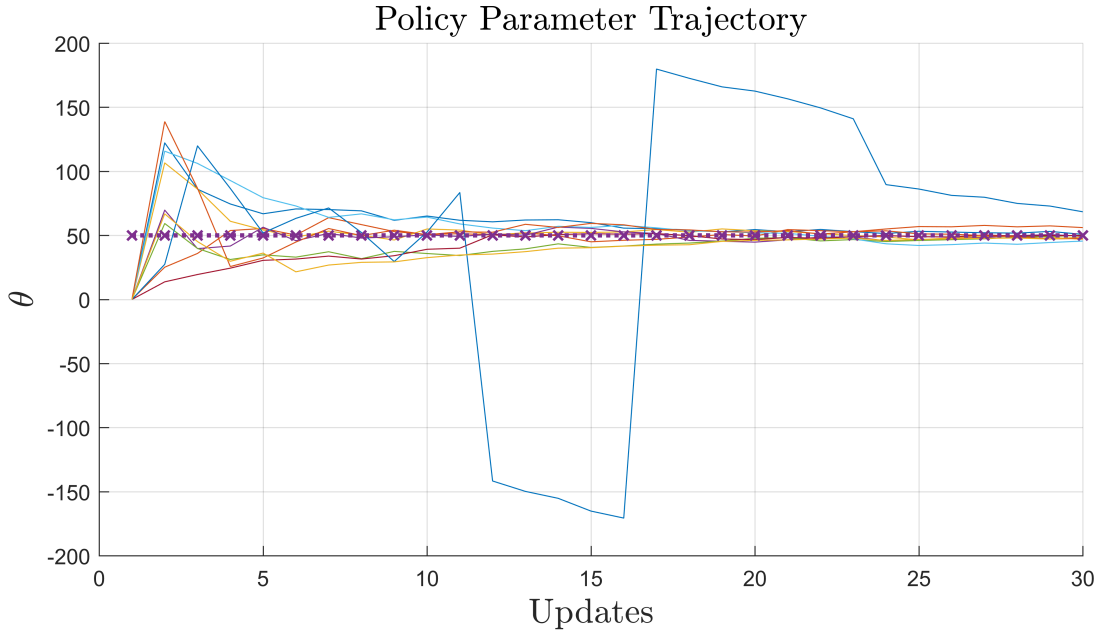

Figure 5.2: The trajectory of the parameter θ through the co-adaptation process, where the $\times$-line represents the true twist value $\beta_{twist} = 50$

5.1.3 Results

For this scenario of indirect shared control, the results for two cases (fixed initial position and random initial position) were obtained. The trajectory definition, Eq.(3.16),includes the initial position distribution p_0. If the initial position is assumed to be fixed, there is no randomness associated with it, and p_0 is changed to 1. If the initial position is determined at random, it is chosen randomly for any location within the screen's boundary. According to [42], this scenario is known as "Exploring-Starts", in which the agent, or the machine in this case, uses the randomness of the initial position to explore the state space. The expected long-term reward for the two cases is shown in figure 5.3. In this case, the β_{twist} is 10 degrees. The agent accumulates rewards

faster for a fixed initial position (the $\star$-curve) than for a random initial position (the $\times$-curve).This is due to the influence of the random initial position, which results in a random starting position for each trajectory and increases the agents' generalization to the entire state space along with the policy exploration rate Σ that is reduced gradually through the interaction (see section 4.4.1). The newly visited (explored) region within the screen state space also allows the agent to accumulate more rewards toward the end of the interaction process as depicted in figure 5.3 during the very last generated trajectories.

PG algorithm,(see section 4.4, Algorithm 3), is referred to to highlight the results and provide further context. Within the **experience loop**, trajectories are generated that demonstrate the machine's experience, to learn from, through interaction with the human. As a result, the machine would receive more rewards by enabling this interaction to take place across the entire state space, or all potential starting positions, as opposed to a fixed starting position. As a result, compared to a fixed initial position, the **policy gradient loop** generates a direction that increases reward accumulation. Finally, the **co-adaptation loop** updates the policy and create a new machine's policy that is more adapting to human behavior than the old policy. The obtained

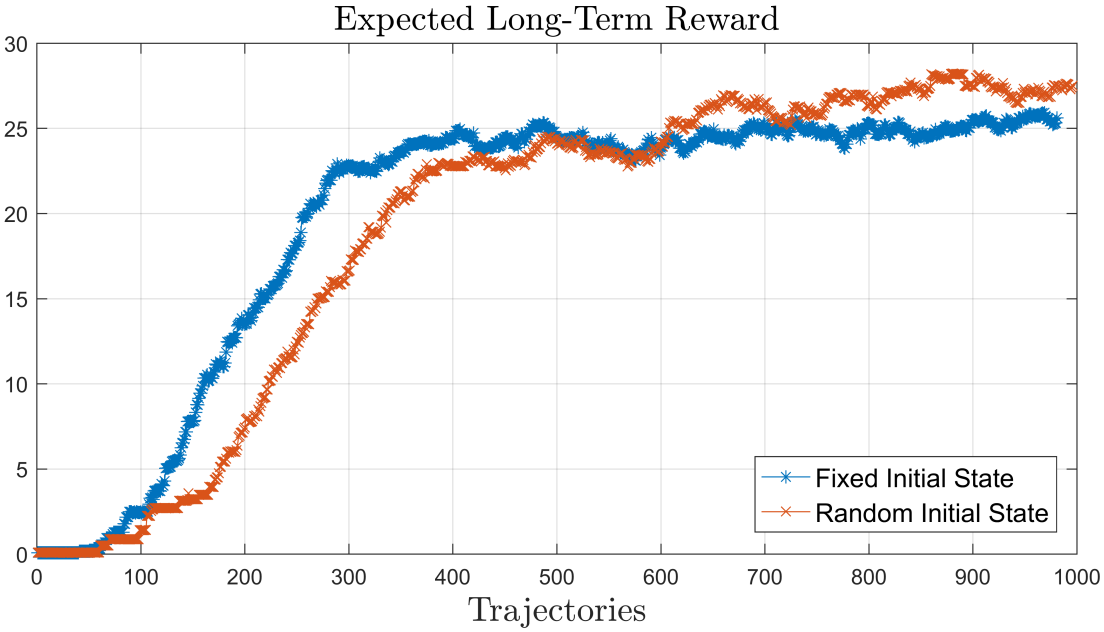

Figure 5.3: The expected long-term reward for indirect-shared control

parameter θ, by the **co-adaptation loop**, will be applied by the new policy through the rotation matrix $R(.)$ in order to account/compensate for the twist angle β_{twist}.The following equation represents how compensation occurred by combining Eq.(5.2) and Eq.(5.3):

$$\mu_{u_{machine}} = R(\theta + \beta_{twist})u_{human} \tag{5.6}$$

Where $\mu_{u_{machine}}$ is the mean value for the machine action[4]. Therefore, in order to account for the twisted joystick, the machine policy will now apply a rotation matrix $R(-\theta)$, as stated in Eq.(5.6). This produces the identity matrix, indicating that the applied human action is correctly mapped by the machine policy in order to track the target on the screen. The machine co-adaptation to the human behaviour

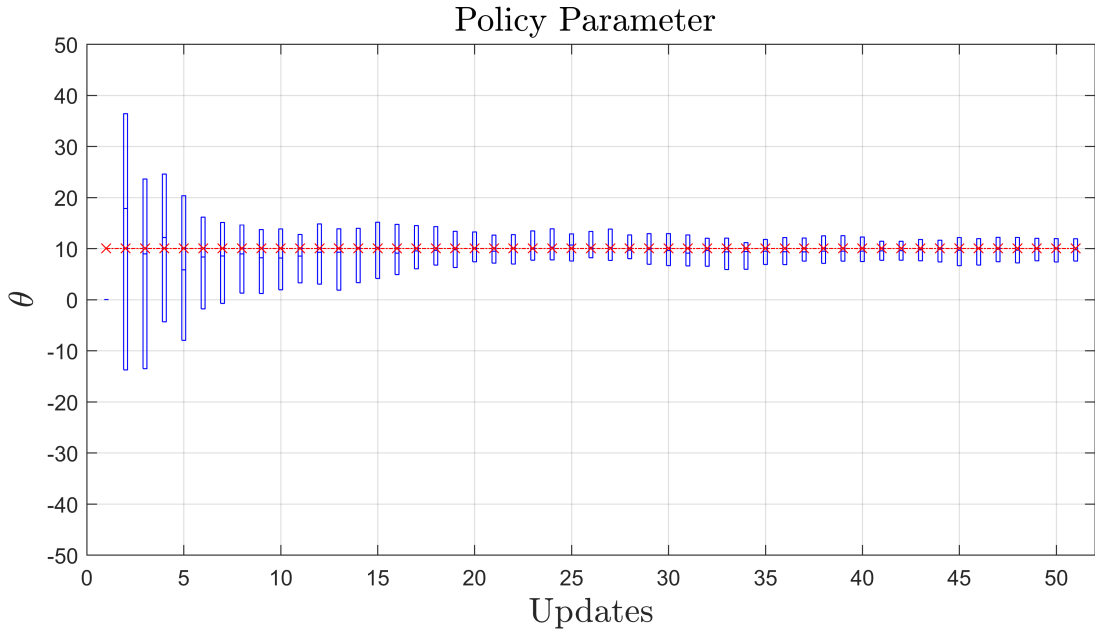

Figure 5.4: The policy convergence, for fixed initial position. Where the $\times$ indicates the true twist value $\beta_{twist} = 10$ degree

is depicted in figures 5.4 and 5.5. The convergence of the policy is assessed by the convergence of the parameter θ to the correct joystick twisting that is β_{twist}. The box-plot in Figures 5.4 and 5.5 show the average of 100-independant runs, where each run results in a sequence for θ values through the adaptation process. The box-plot shows the mean value, the horizontal line that splits the box, of the parameter θ and the covariance, the vertical length of the box, across all runs. The dots along the vertical length of the boxes represent outliers for the parameter θ within that update phase. The agent's capacity for learning, that is, exploiting the generated trajectories in order to learn the policy (policy parameter), is significantly influenced by the randomness introduced by the machine's policy and the initial position distribution. The baseline $b_{minimizer}$, (introduced in section 3.3.2, Eq.(3.21) and Eq.(3.25)), reduces the covariance of the estimated parameter θ to its minimum value. Thus, the randomness aids in fast learning and better generalization, while the covariance of the learned policy parameter is minimized.

Figure 5.6 shows the policy gradient estimate for the 100-independant runs. The

[4]Keep in mind that the actual β_{twist} is unknown for both the machine and the algorithm

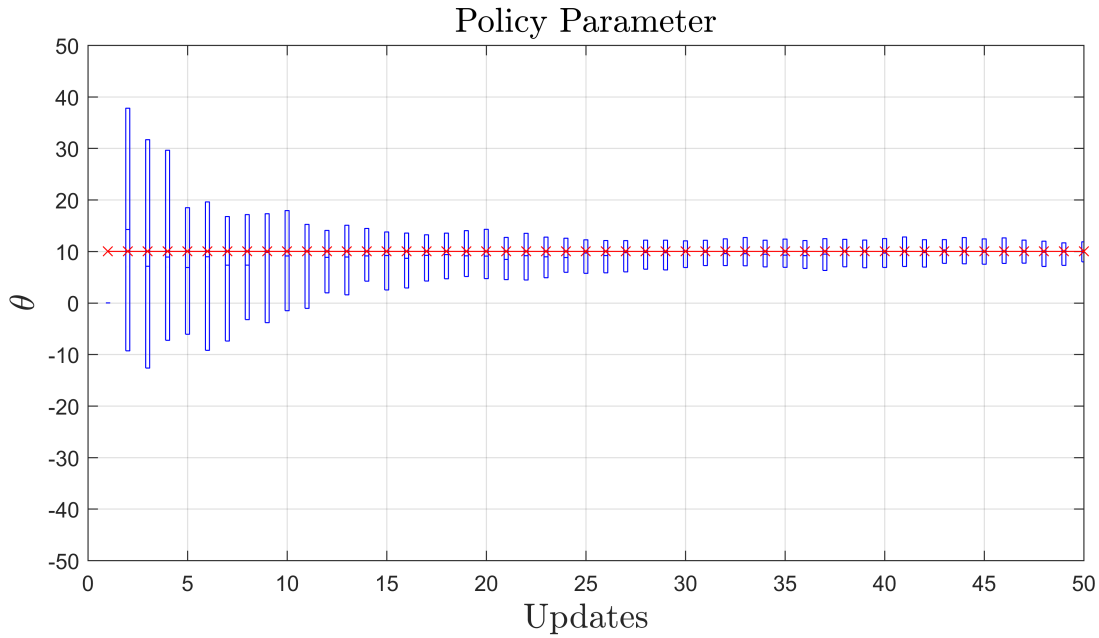

Figure 5.5: The policy convergence, for random initial position.Where the $\times$ indicates the true twist value $\beta_{twist} = 10$ degree

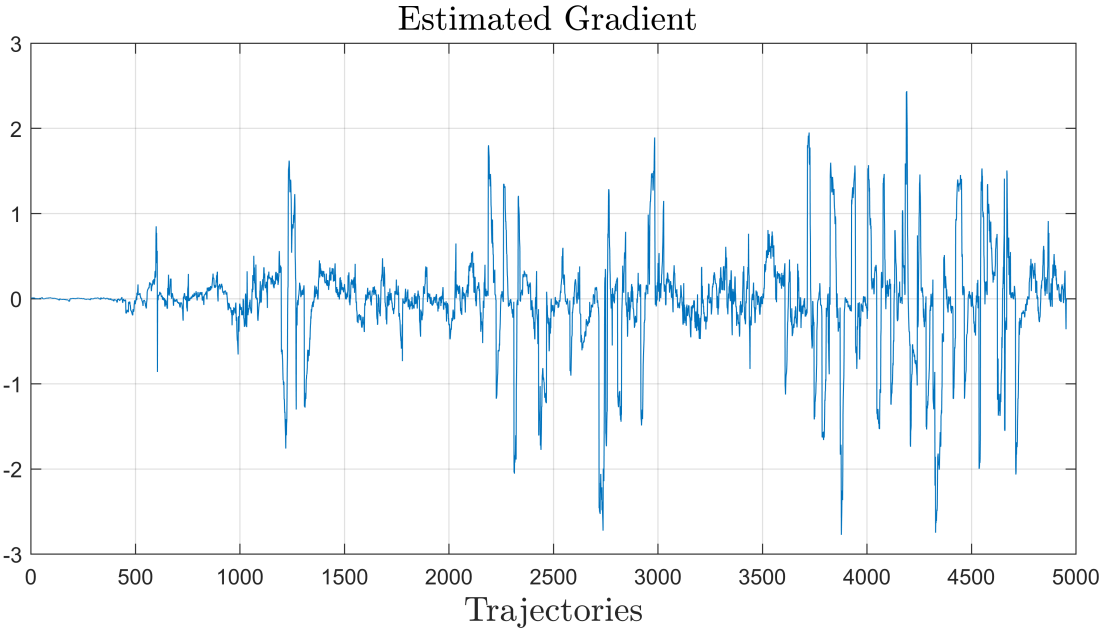

Figure 5.6: The policy gradient of the co-adaptation loop

co-adaptation, that is the update of the parameter θ, occurs once the policy gradient converges locally for a given set of generated trajectories that is generated by the **experience loop**, (see section 4.4, Algorithm 3).

5.2 Direct Shared Control

In this scenario, there is direct human-machine interaction required to complete the task. Without any oscillation, the SeaSaw stick should be balanced. The system must be critically damped, which means that the damping ratio ζ is equal to 1. In this scenario, which is depicted in figure 5.7,both the human and the machine go through a sequence of actions to balance the stick without oscillation. In order to co-adapt to human behavior, the machine must directly learn the actions.

5.2.1 The Environment Mathematical Model

As previously stated in section 5.1.1, the models are required only for simulation purposes and do not used by PG algorithm.

- **The Human Model**: In this scenario, the following assumptions are used for the human model (for further illustration see the **Transition Model)**):
  - The human continues to perform the same policy, while interacting with the machine
  - Human actions can be thought of as adding stiffness or damping to the system. However, the machine performs a damping action when the human performs a stiffness action, and vice versa
- **The Machine Model** or **The Policy**: The machine policy is represented by a probability distribution function as illustrated in the next step for Transition model or the Dynamics.

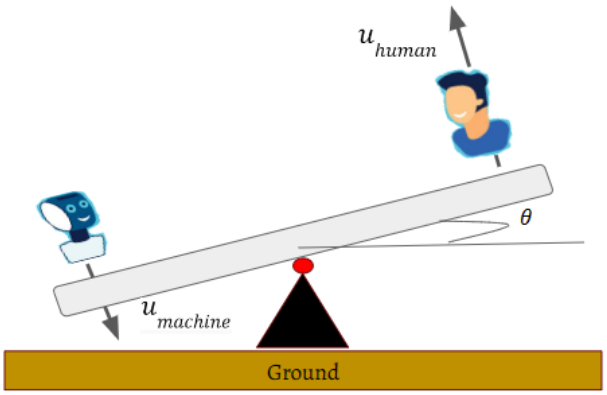

Figure 5.7: The SeaSaw shared control task

- **The Dynamics** or the **Transition Model**: According to figure 5.7, the human applies its action, $u_{human} \in R^1$, and the machine applies its actions, $u_{machine} \in R^1$, thus the total applied action $u_{total} \in R^1$, or in this case the total applied torque is:

$$u_{total} = u_{human} + u_{macine} \tag{5.7}$$

In order to balance the SeaSaw, the angular angle[5] θ and the angular velocity $\dot{\theta}$ are desired to be equal to zeros. That is, when $(\theta_d, \dot{\theta}_d) = (0, 0)$, θ_d and $\dot{\theta}$ are the desired angular angle and angular velocity, respectively. The dynamics that governs this interaction is given by the following differential equation under the assumption of no friction and no damping in the SeaSaw's joint (see figure 5.7):

$$J\ddot{\theta} = u_{total} \tag{5.8}$$

where J is the SeaSaw inertia, and u_{total} is given as:

$$u_{total} = (k_{p_h} + k_{p_m})(\theta_d - \theta) + (k_{d_h} + k_{d_m})(\dot{\theta}_d - \dot{\theta})$$

Under the assumption that $(\theta_d, \dot{\theta}_d) = (0, 0)$, the following is obtained:

$$u_{total} = -(k_{p_h} + k_{p_m})\theta - (k_{d_h} + k_{d_m})\dot{\theta} \tag{5.9}$$

where k_{d_h} and k_{p_h} are the human policy parameters for damping and stiffness effect, respectively, and k_{d_m} and k_{p_m} are the machine policy parameters for damping and stiffness effect, respectively. From this, the human and the machine policies are represented as follows:

  - The human policy is:

$$u_{human} = -k_{p_h}\theta - k_{d_h}\dot{\theta} \tag{5.10}$$

  - The machine policy is represented by a probability distribution function as follows:

$$u_{machine} \sim \pi(.|\theta, \dot{\theta}) = \mathcal{N}(-k_{p_m}\theta - k_{d_m}\dot{\theta}, \Sigma) \tag{5.11}$$

    where Σ denotes the machine's policy covariance.

When the human only performs stiffness action, $k_{d_h} = 0$, whereas when the human performs damping action, $k_{p_h} = 0$.The same is true for the machine side, in that both humans and machines are responsible for one type of action. The

[5] Do not get confused, θ is the angular angle and not the policy parameter

following differential equation, with the assumption that $\dot{\theta}_d = 0$, is obtained by substituting Eq.(5.9) in Eq.(5.8):

$$J\ddot{\theta} + (k_{d_h} + k_{d_m})\dot{\theta} + (k_{p_h} + k_{p_m})\theta = (k_{p_h} + k_{p_m})\theta_d \tag{5.12}$$

By taking the Laplace transform of Eq.(5.12), and finding the transfer function, the following is obtained[6]:

$$\frac{\theta(s)}{\theta_d(s)} = \frac{\frac{k_{p_h}+k_{p_m}}{J}}{s^2 + \frac{k_{d_h}+k_{d_m}}{J}s + \frac{k_{p_h}+k_{p_m}}{J}}$$

$$\frac{\theta(s)}{\theta_d(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{5.13}$$

where ζ denotes the system damping and ω_n denotes the system natural frequency.

- **The Reward Function**: The following reward function is proposed to fulfill the task goal:

$$r_t = -(1 - \zeta_t)^2 - e_t^2 \tag{5.14}$$

where ζ_t is the current system damping and $e_t = \theta - \theta_d$ is the current error between the current and desired orientation.

The machine learns a policy for selecting actions, $u_{machine} = k_{d_m}\dot{\theta}$, to co-adapt to human behavior, $u_{human} = k_{p_h}\theta$, using the PG algorithm, with the human-machine interaction being as follows (see figure 5.8):

- Do until co-adaptation achieved:
  1. In this scenario, it is assumed that the human will take the same policy u_{human},i.e the human's policy parameters are fixed
  2. The machine observes the environment state, $\theta, \dot{\theta}$, and generates an action $u_{machine}$
  3. The next state, θ, is used to calculate e_t, is generated by the environment dynamics (see Eq.(5.12) and Eq.(5.13))
  4. For the applied machine and human actions, the current system damping ζ_t is calculated
  5. The machine policy receives a scalar reward signal, r_t, indicating how well it responds to human action in terms of e_t and ζ_t

[6] s here denotes the Laplacian operator

6. Based on the reward signal, the machine co-adapts to human actions using the PG algorithm
7. The machine's policy improves as it receives the reward feedback signal

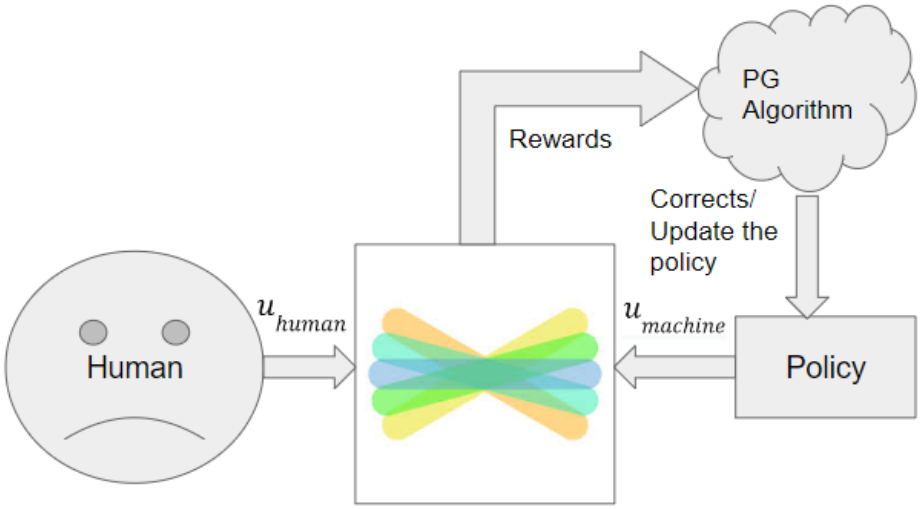

Figure 5.8: The problem of direct shared control

5.2.2 Algorithm Implementation

The PG learning algorithm is demonstrated in Algorithm 3.Under the following hyper-parameter settings, the algorithm is run:

1. The learning rate is $\alpha = 0.9$
2. The discount factor is $\gamma = 0.95$
3. The exploration range is $\Sigma_{d,p} = [1, 5]$
4. The sampling time is $\Delta t = 1/60$
5. The value for the human and the machine actions are conditioned to be in the following interval, $k_{ph} = 10, k_{dm} \in [1, 250]$
6. The SeaSaw's inertia is $J = 1$

To obtain the results, 100-independent runs are performed. Each run comes to an end when the agent's policy converges.

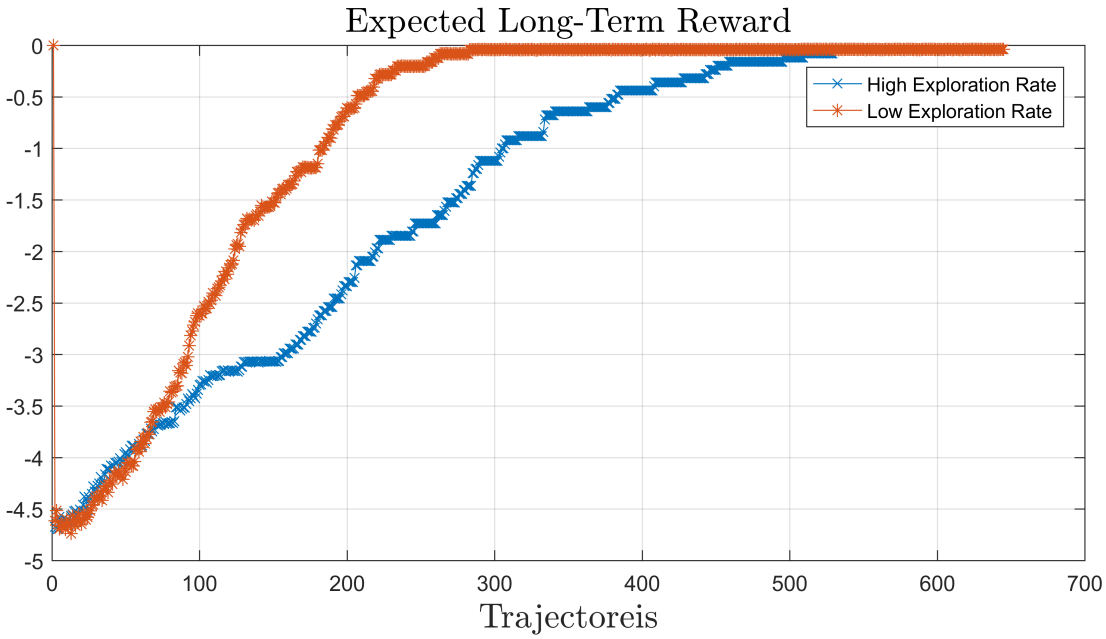

Figure 5.9: The expected long-term reward for direct-shared control, where the covariance $\Sigma_{\times} = 3 > \Sigma_{\star} = 1$

5.2.3 Results

Figure 5.9 shows the total reward for 100-independent runs for achieving the $\zeta = 1$ goal.Figure 5.9 depicts the results for two different exploration rates (different policy covariances of $\Sigma_{\star}$ and $\Sigma_{\times}$) and a random initial orientating. The $\times$-curve shows the result of higher exploration, while the $\star$-curve shows the result of a lower exploration rate. The cumulative reward curve in these two cases is smoother compared to the cumulative reward in the case of indirect shared control (see figure 5.3), where the dynamics is an algebraic equation (see Eq.(5.4)). The environment model in this scenario is a double integrator, Eq.(5.8), which plays the role of a filter that filters out some of the environment's randomness due to the machine's policy and random initial starting position. The ζ values through the co-adaptation process are shown in figures 5.10 and 5.11, where on average the desired no oscillation behavior is satisfied.

The mean values for the machine policy parameter k_{dm} for the high exploration case and the low exploration case were found to be 16 and 3.8 respectively. Figure 5.12 illustrates the **co-adaptation loop behaviour** where the changes (peaks) in the gradient of the lower exploration curve ($\star$-curve) occurs earlier compared to the case of a higher exploration rate ($\times$-curve). Once the gradient converges (the right side of the figure 5.12), the agent reward accumulation starts to settle in and there is no more reward to obtain, thus the policy also converges. The baseline $b_{minimizer}$, (introduced

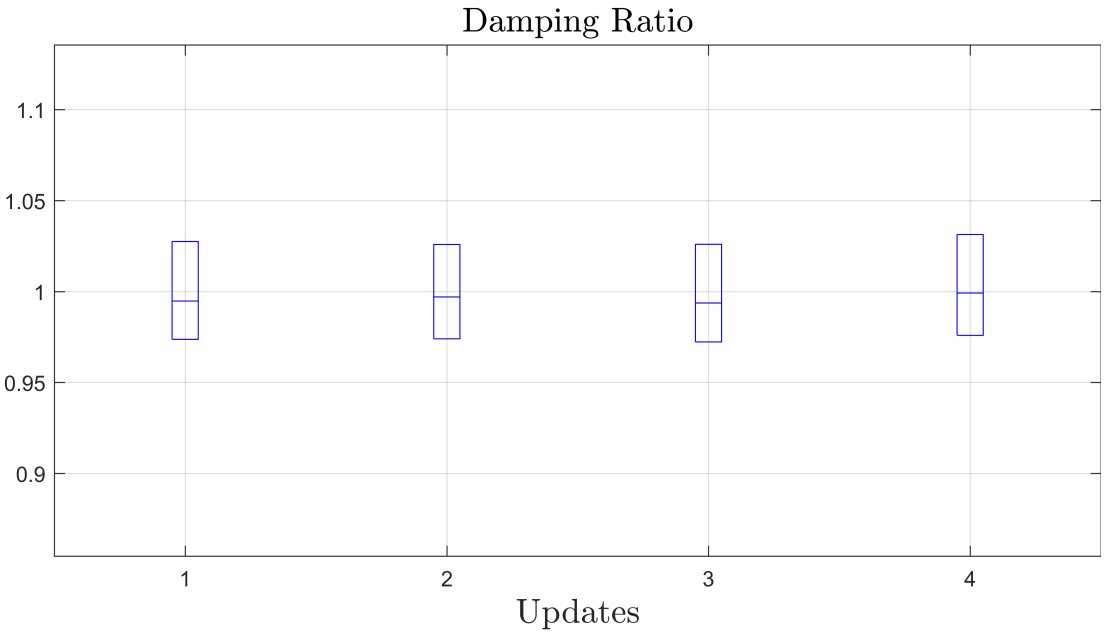

Figure 5.10: The system damping ratio ζ, for high covariance $\Sigma_{\times}$

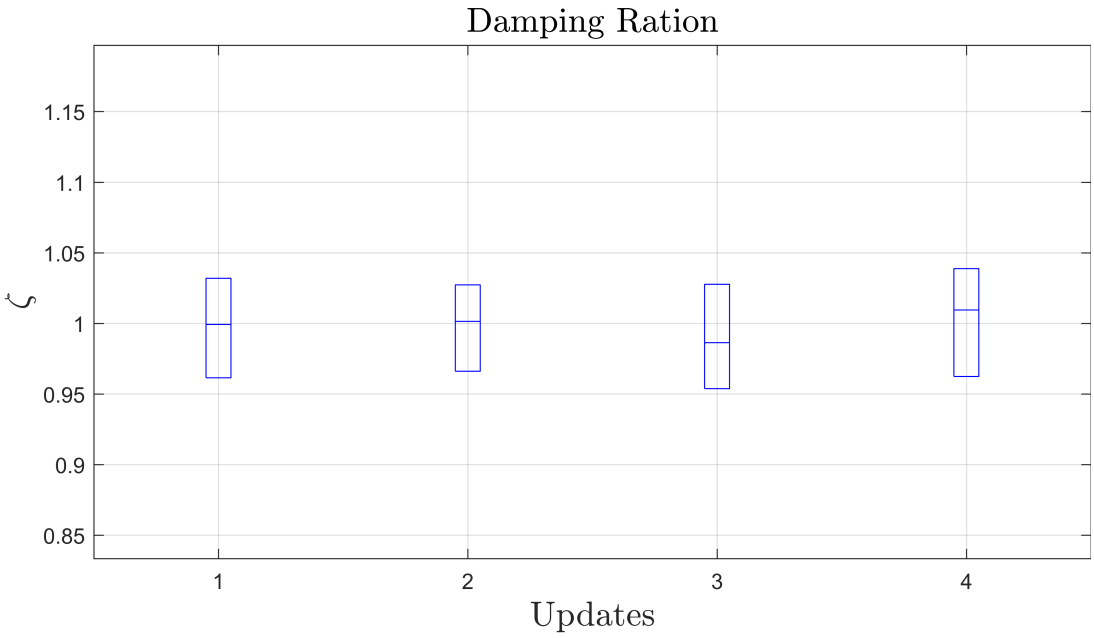

Figure 5.11: The system damping ratio ζ, for low covariance $\Sigma_{\star}$

in section 3.3.2, Eq.(3.21) and Eq.(3.25)), reduces the covariance of the estimated parameter k_{d_m} to its minimum value. Thus, the randomness aids in fast learning and better generalization, while the covariance of the learned policy parameter is minimized and bounded. The time response for the seesaw orientation position after applying the obtained policy parameter for the two exploration scenarios is shown in Figure 5.13. Applying the policy parameter found for the low exploration example

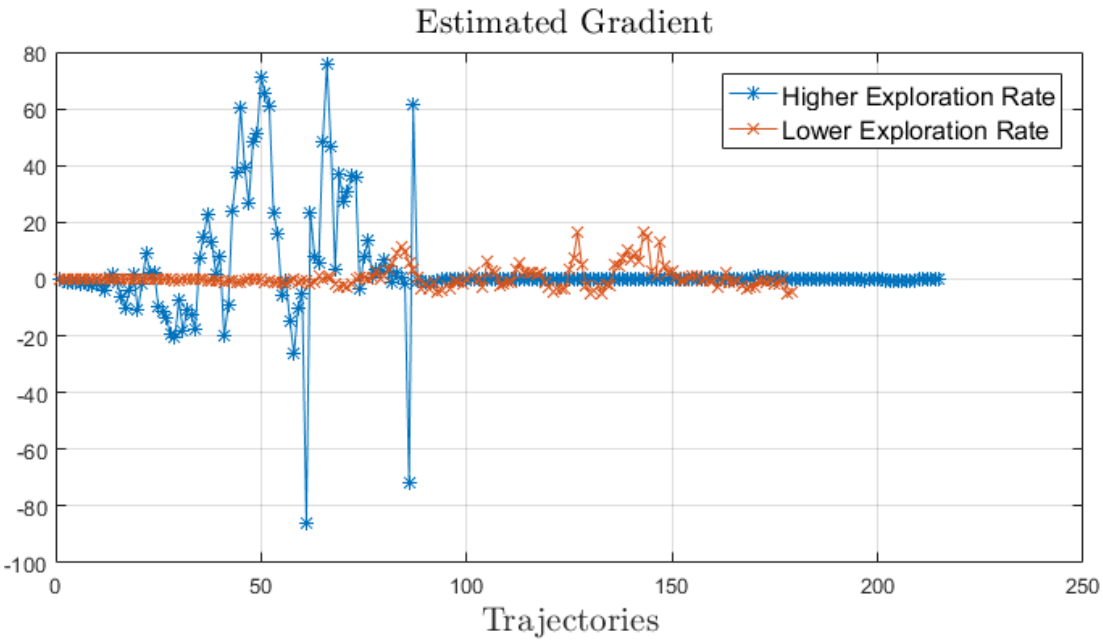

Figure 5.12: The machine policy gradient for k_d

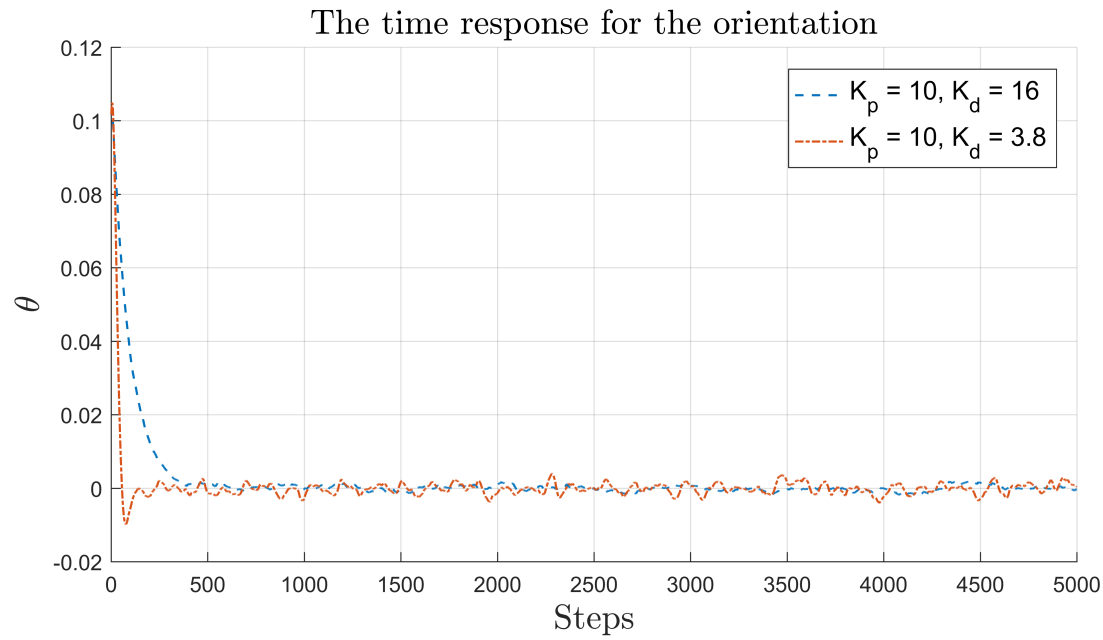

Figure 5.13: The time response for θ, by applying the obtained policy parameter k_{dm}

leads to the response displayed in Figure 5.13, which oscillates before achieving equilibrium. In contrast, the response doesn't oscillate and approaches equilibrium in the case of a high exploration rate.

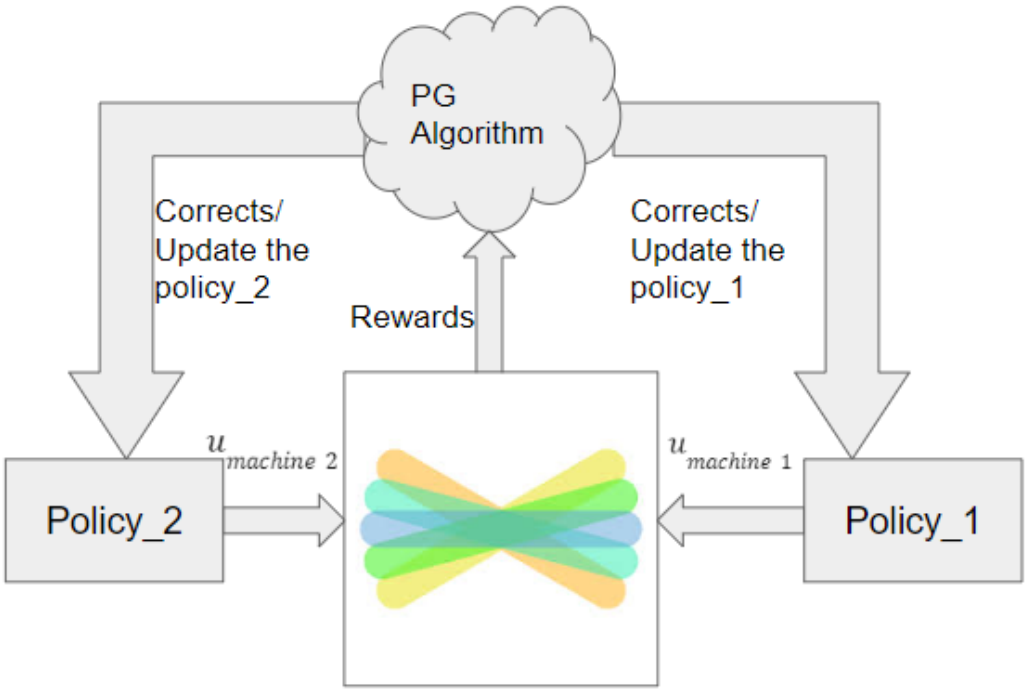

Figure 5.14: The problem of direct shared control (the two machines scenario)

5.3 Machine - Machine Co-adaptation

In this scenario, the two agents are merely machines. In the previous case, the human is assumed to have already learned his optimal policy. This case's motivation is to examine what happens if two agents co-learn and co-adapt simultaneously. This case belongs to the direct shared control, where Eq.(5.9) contains two machines actions instead of one for the human and the other for the machine as follows.

$$u_{total} = u_{machine_1} + u_{machine_2} \in R^1 \tag{5.15}$$

where:

$$u_{machine_1} \sim \pi(.|\theta) = \mathcal{N}(-k_{p_{m_1}}\theta, \Sigma_1) \tag{5.16}$$

$$u_{machine_2} \sim \pi(.|\dot{\theta}) = \mathcal{N}(-k_{d_{m2}}\dot{\theta}, \Sigma_2) \tag{5.17}$$

Each machine is responsible for just performing one type of action. Σ_1 and Σ_2 represent the covariance (exploration rates) for the first and second machines, respectively.The direct shared control structure is used in this situation as well.

5.3.1 Algorithm Implementation

The PG learning algorithm is demonstrated in Algorithm 3.Under the following hyper-parameter settings, the algorithm is run:

1. The learning rate is $\alpha = 0.9$
2. The discount factor is $\gamma = 0.95$
3. The exploration range is $\Sigma_{d,p} = [1, 5]$
4. The sampling time is $\Delta t = 1/60$
5. The value for the human and the machine actions are conditioned to be in the following interval, $k_{pm1}, k_{dm2} \in [1, 250]$
6. The SeaSaw's inertia is $J = 1$

To obtain the results, 100-independent runs are performed. Each run comes to an end when the the machines' polices converge.

5.3.2 Results

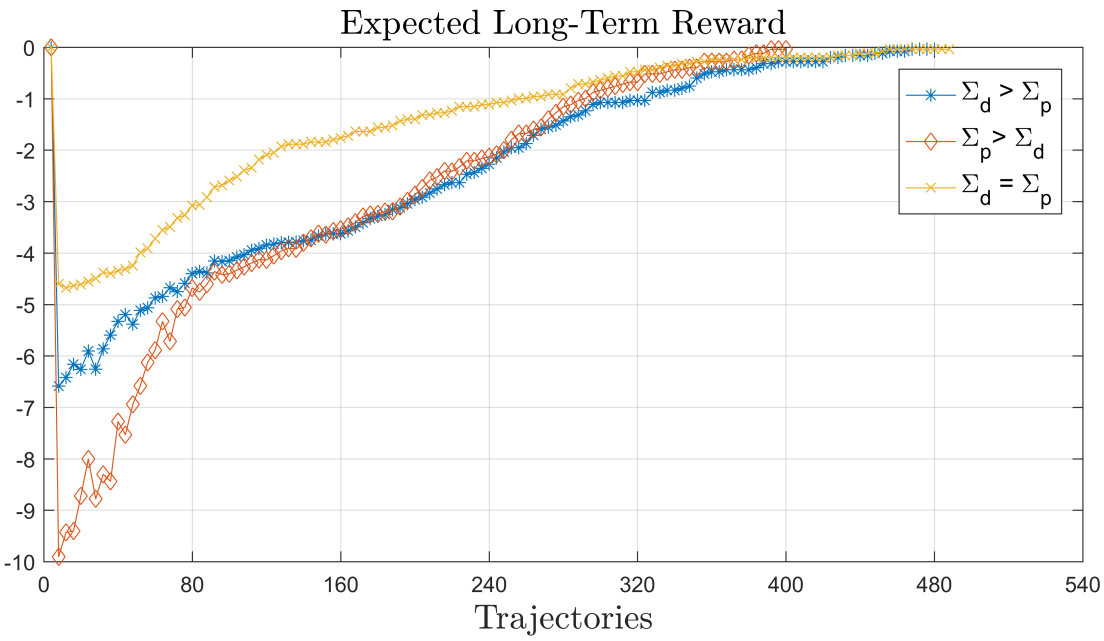

Figure 5.15: The expected long-term reward for direct-shared control

The expected cumulative reward is shown in figure 5.15. Both machines aim to maximize a shared, common, reward function (see section 4.3.1), in order to achieve

the task goal that is $\zeta = 1$ and $e_t = 0$. When both machines explore at the same rate ($\times$-curve, $\Sigma_d = \Sigma_p = 1$), convergence occurs faster than when they explore at different rates ($\star$-curve and $\blacklozenge$-curve). When the covariances differ, that is, when the exploration differs, the accumulating reward behavior changes.

The convergence occurs faster when the machine that adds damping to the system explores more ($\star$-curve, $\Sigma_d = 1 < \Sigma_p = 2$) than when the machine that adds stiffness explores more ($\blacklozenge$-curve, $\Sigma_p = 2 > \Sigma_d = 1$). This is due to the fact that increasing the system's damping will increase the damping ratio and reduce oscillation. To achieve the desired outcome of $\zeta = 1$, more interaction steps are required when the stiffness term is present. This is due to the increase in oscillation and the speed of the system's response. The ζ values through the co-adaptation process are shown in

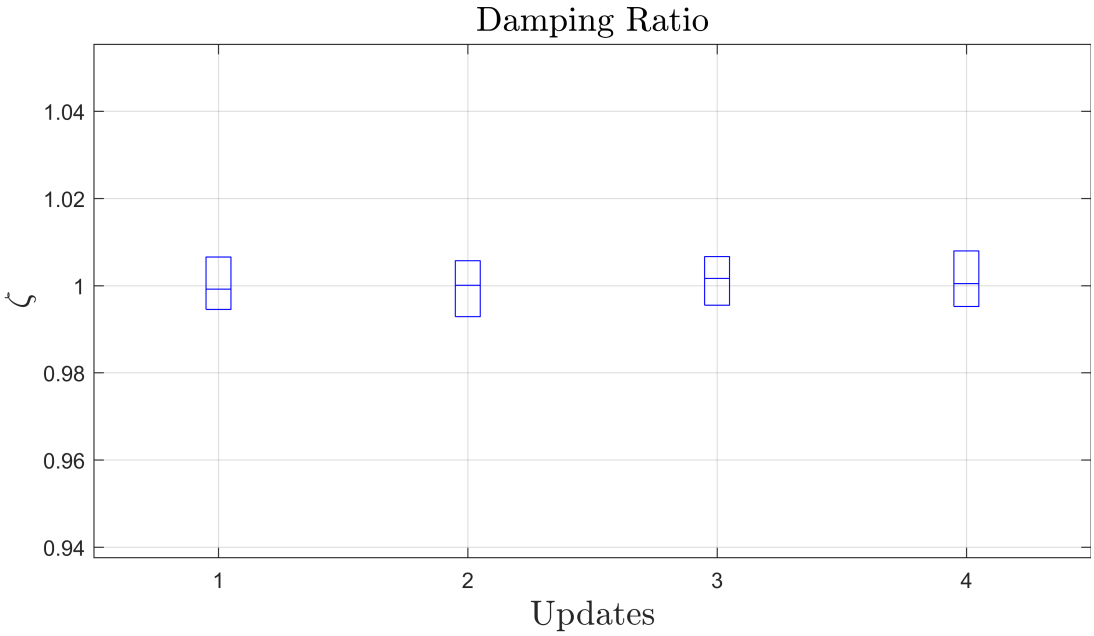

Figure 5.16: The system damping ratio ζ, for unequal covariances

figure 5.16, where on average the desired no oscillation behavior is satisfied.

The mean values for the first machine policy parameter, k_{pm_1}, were found to be 16 in the case of $\Sigma_d > \Sigma_p$, and 25 in the case of $\Sigma_p > \Sigma_d$. For the second machine policy parameter, k_{dm_2}, the mean values were found to be 12 in the case of $\Sigma_d > \Sigma_p$, and 58 in the case of $\Sigma_p > \Sigma_d$.

For various exploration rates for each machine, the **co-adaptation loop** behavior is depicted in figures 5.17 and 5.18. In comparison to the machine that stiffens the system, the policy gradient for the machine that dampens the system has experienced higher peaks or changes for the two exploration rates. In other words, even though

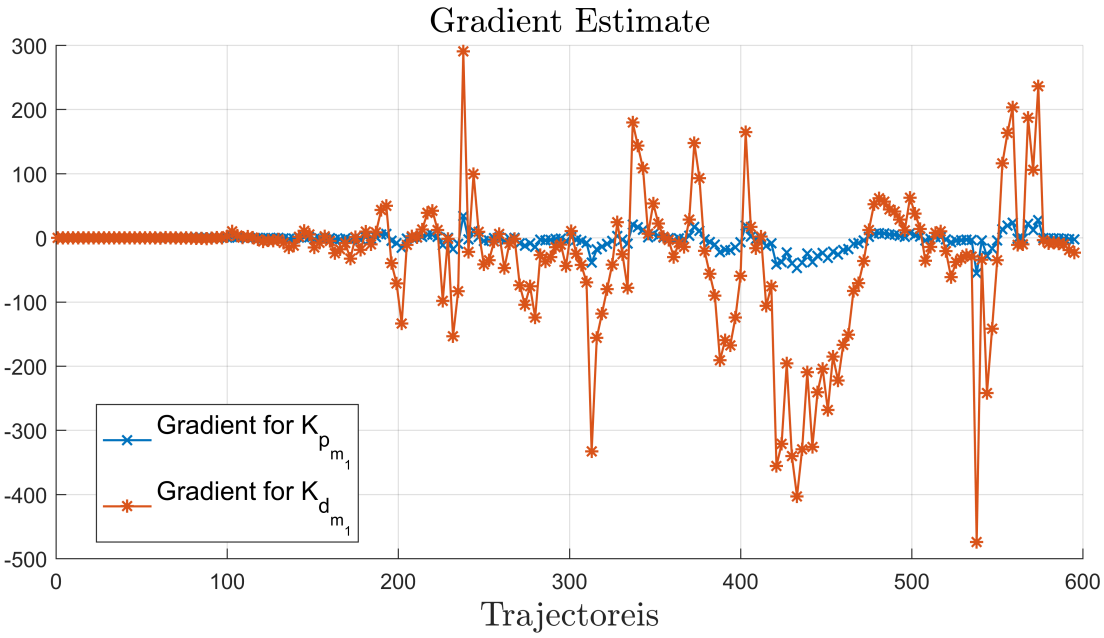

Figure 5.17: The policy gradient for k_p ($\times$-curve), and the policy gradient for k_d ($\star$-curve). Where $\Sigma_d > \Sigma_p$

the exploration rate for damping machines is lower than the exploration rate for stiffening machines, the PG algorithm tends to explore more within damping actions (dampening machine). This implies that the goal of the proposed reward function and the PG algorithm is to select the action that results in faster and less oscillatory behavior. As previously discussed, increasing damping stabilizes the system, helps it reach equilibrium faster, and dampens system overshoot by simultaneously lowering $(\zeta_t - 1)$ and e_t. The baseline $b_{minimizer}$, (introduced in section 3.3.2, Eq.(3.21) and Eq.(3.25)), reduces the covariance of the estimated parameter k_{d_m} to its minimum value. Thus, the randomness aids in fast learning and better generalization, while the covariance of the learned policy parameter is minimized and bounded. Figure 5.19 shows the time response for the seesaw orientation position after applying the obtained policies' parameters. It can be seen that higher exploration rates for the stiffness parameter compared to the damping parameter result in a higher k_{dm2} value than lower exploration rates for the damping parameter. This is due to the fact that, through interaction, in order to reduce the effect of a higher stiffness factor, k_{pm1}, a higher value for k_{dm2} is required to reduce its effect and satisfy the required task reward function, which is $\zeta_t - 1 \to 0$ and $e_t \to 0$. Whereas in the case of a higher exploration rate for the damping parameter, a higher stiffness parameter is needed to reduce the effects of high damping and reach the intended task goal. It is worth pointing out that the difference in the values for the two scenarios is due to the fact that there is no direct access to the applied action by each machine through

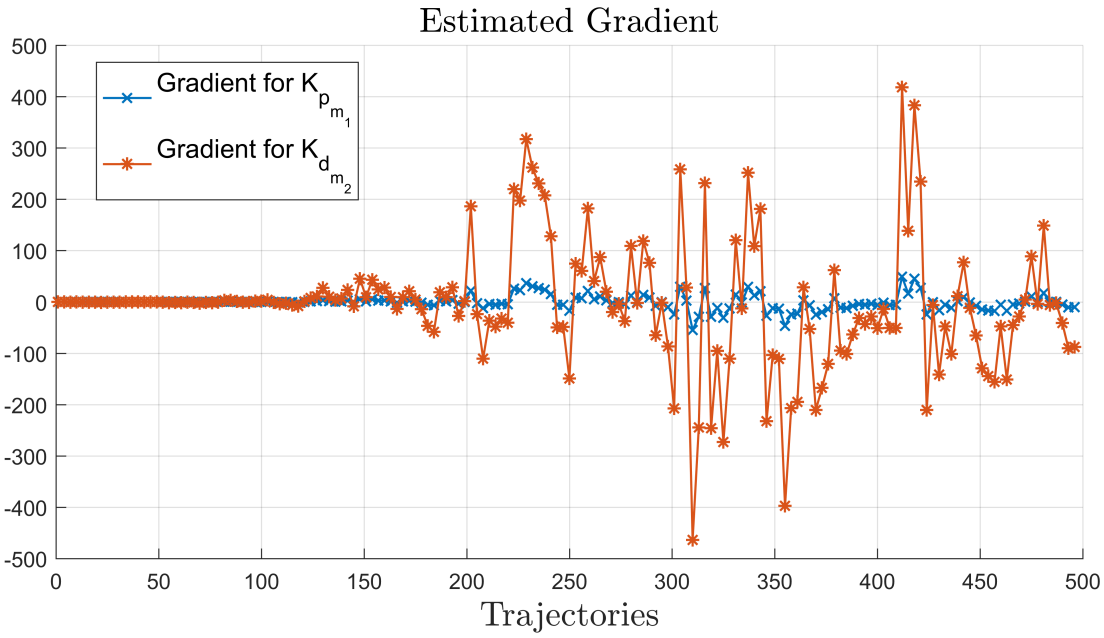

Figure 5.18: The policy gradient for k_{pm1} ($\times$-curve), and the policy gradient for k_{dm2} ($\star$-curve). Where $\Sigma_p > \Sigma_d$

interaction, since each machine notices the effect of the other machine's action. Moreover, the exploration rates, whether for damping or stiffness, drive the generation of differing trajectories of interaction. This results in different policy gradients that drive the update for the policy parameter.

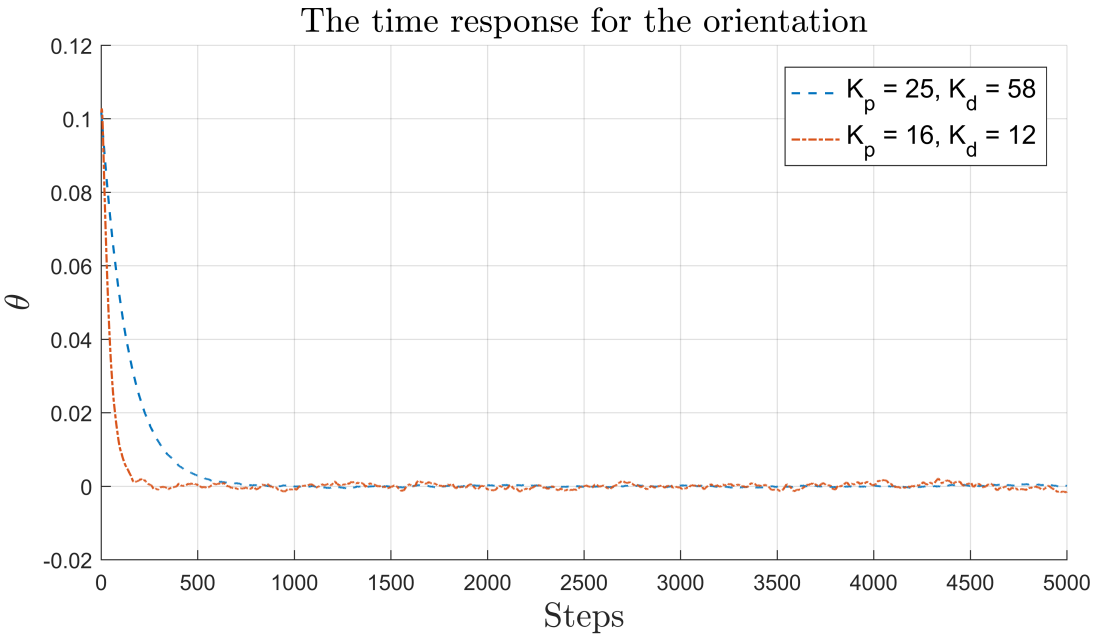

Figure 5.19: The time response for θ, by applying the obtained policies' parameters k_{pm1} and k_{dm2}

6 General Discussion

The main focus of this chapter is to examine the practical implementation of the proposed algorithm for co-adapting a machine's policy to a human's behavior in the context of the HMCo problem. The proposed scenarios presented in this thesis are highly relevant to real-life applications, particularly in robotics and teleoperation, where coordination between human operators and machines is crucial. In this chapter, the rationality assumption, which is a fundamental aspect of the proposed algorithm, will be discussed, as will the implications of not considering this assumption. Additionally, this chapter will cover the limitations and computational complexity of the algorithm and the importance of selecting the appropriate hyperparameters for the policy gradient algorithm to achieve optimal performance.

6.1 The Importance of the Proposed Case Studies

Firstly, the double integrator system represents the SeaSaw example, a model that is frequently used in robotics and other control systems. It is a simple and mathematically tractable model that can be used to represent a wide range of systems, such as robotic arms, drones, and other mobile robots [76]. Furthermore, the double integrator system is a well-established benchmark for control algorithms [76]–[78], making it a suitable case study for evaluating the performance of HMCo algorithms.

Secondly, the joystick example is a practical application of HMCo in the field of teleoperation [79]. Teleoperation is a method of controlling a remote machine or robot using a human operator, and it is commonly used in applications such as remote surgery, nuclear power plant maintenance, and space exploration [80]–[82]. The human operator uses a joystick to control the remote machine, but the joystick may be uncertain or have other issues, such as time delay, that make it difficult for the operator to control the machine. In such cases, using a policy gradient algorithm to co-adapt the machine's policy to the human's behavior can improve the performance and usability of the teleoperation system. This is important in applications where human-machine coordination is crucial, such as in remote surgery, where precise control is required to avoid damaging healthy tissue [83].

Overall, both the double integrator system and the joystick example are important case studies in the field of human-machine co-adaptation. The double integrator system allows researchers to study the fundamental principles of control theory in a simplified setting, while the joystick example is a practical application of HMCo in the field of teleoperation. Both case studies have important implications for a wide range of real-world applications, including robotics, control theory, and teleoperation. Furthermore, both cases can be used as a base for further research in the field.

6.2 The Rationality Assumption

In this thesis, one of the key assumptions made is that humans are rational in their actions, meaning they always pick the right action to achieve a given task. However, in many real-world scenarios, the human may not have the knowledge or expertise to act the right way and may learn through interaction with the machine. In this case, the actions may not be aligned with the task goal, leading to increased complexity for the machine to co-adapt to their behavior. This can lead to an increase in complexity for the machine to co-adapt to the human's behavior.

It is important to note that in the context of HMCo, the human and the machine do not have prior knowledge of each other's behavior, making the interaction model-free. This means that the machine cannot predict the human's actions, and the human cannot predict the machine's. In this context, the assumption of rationality is the basic minimum requirement to establish a framework for the machine to co-adapt to the human's behavior. The rationality assumption allows the machine to evaluate the human's actions in terms of how they contribute to achieving the task's goal, and the machine can evaluate its own actions based on the reward it receives. Additionally, the rationality assumption allows the machine to learn about the human's intentions and co-adapt accordingly without the need for a detailed model of the human's behavior. It is worth noting that assuming rationality does not mean that humans are always optimal or that their actions are always the best; it means that humans are trying to achieve the task's goal, and the machine can co-adapt to this behavior.

Without this assumption, the problem of HMCo becomes underconstrained, and it is unclear what the machine should co-adapt to. Therefore, the problem becomes more complex. This is because, as outlined in Eq. (4.4), the human's intention is no longer directly associated with achieving the goal, i.e. $\delta \neq 0$. To handle this, the problem can be reformulated as a Partial Observable MDP (POMDP), in which the human's intention is included as a part of the POMDP's state [84]–[88]. This requires inferring and incrementally updating the human's intention using Eq. (4.3), which

increases the computational complexity of the problem and affects the convergence characteristics in terms of hyperparameters.

This solution is compatible with the policy gradient algorithm, which can also be applied in the case of POMDP [89], [90], and the problem remains model-free as no explicit model for the human is required except for the human intention model that is inferred incrementally through interaction.

6.3 Practical Aspects of the Proposed Algorithm

This section discusses some of the practical implementation aspects of the proposed algorithm. First of all, the hyper-parameter selection is discussed, followed by the algorithm's performance regarding the various loops defined in section 4.4, and finally, the algorithm's scalability, complexity, and memory are discussed.

For the two cases of indirect shared control and direct shared control, the human action is associated with a small uniform randomness that accounts for the randomness caused by the environment that affects the human action; that is, even if the human generates a rational action, the action's implementation in real life may be associated with some randomness due to the associated used equipment during the experiment set up.

6.3.1 Selection of Hyper-parameters

Hyperparameter selection is an important step in the policy gradient algorithm, as it can significantly impact the performance of the algorithm. The learning rate controls the rate at which the algorithm updates the policy parameters [41]. A higher learning rate will result in more rapid updates but may also lead to instability and overshooting the optimal solution [91]. A lower learning rate will result in more stable updates but may be slower to converge. The exploration rate controls the degree of exploration versus exploitation in the algorithm. A higher exploration rate will result in the algorithm exploring more of the state space, while a lower exploration rate will result in the algorithm exploiting the current policy more [42], [55], [56]. The discount factor γ controls the trade-off between short-term and long-term rewards in the algorithm. A higher discount factor will result in the algorithm placing more emphasis on long-term rewards, while a lower discount factor will place more emphasis on short-term rewards [45]. Finally, the initialization of the policy can also have a significant impact on the performance of the algorithm. A poor initialization can lead to the algorithm

getting stuck in a suboptimal solution, while a good initialization can result in faster convergence to the optimal solution.

The following summarizes the selection of each parameter and its effect on other values.

- Learning rate α: The learning rate controls the step size within the gradient-space of the objective function that is the expected long-term reward. The learning rate is decreased gradually with each update of the parameter (see Eq.(4.9)). This decrease in learning rate aids in exploring more within the gradient-space at the start of updates and decreases as the update progresses. Small values tend to give a very slow (and divergent behavior in some cases), whereas starting from higher values like 0.9 or 1 results in a more stable behavior for the algorithm. Finally, the update method of the Eq.(4.9) reduces simulation run-time. This is due to the fact that it makes use of the curvature of the objective function via the parameter Λ. A function's second derivative is frequently chosen to be Λ. This improves gradient space exploration to produce a more stable update that is robust to noise in gradient estimation, has numerical round-off, and is non-stationary for the MDP environment probability space. This reduces the learning time for each run by an average of 3.20 versus the update rule in Eq.(3.18)

- The discount factor $\gamma \in [0, 1)$: The discount factor controls policy behavior; that is, low values result in a greedy (shortsighted policy), whereas higher values result in a farsighted policy that considers the effect of delayed rewards. A compromising value for γ is selected. It is worth noting that the discount factor γ actually controls the number of actions or steps to be performed through the **experience loop**. That is as the number of steps/action increase the factor $\gamma^k \to 0$, thus the weighted immediate reward $\gamma^k r_k \to 0$

- The machine's policy covariance: The machine's policy covariances for all cases are also selected by trial and error. Where higher values for the covariances lead to the algorithm diverging, that is because the generated trajectories via **experience loop** are too noisy, and thus the same for the action log-likelihood. This leads to the divergence of the estimate of the gradient via the **co-adaptation loop**

- The machine's policy initialization: The same arguments for the policy covariance hold in this situation

6.3.2 Experience, Co-adaptation and Policy Iteration Loops

The **experience loop** is performed at most for K-times, where the **co-adaptation loop** is performed at most N_{max}-times. Thus, each update for the policy parameter requires a time of order $\mathcal{O}(KN_{max})$. The policy gradient is, according to Eq.(3.3.1) and Eq.(3.3.1), the average of the policy log-likelihood for each applied action in each position. Thus, once the policy gradient is used to update the policy, the direction in which the update takes place is the direction that enables the machine's policy to generate actions toward a higher cumulative reward. Moreover, as the machine becomes co-adapted to human behavior (that is, the policy parameter converges) and the policy gradient converges locally, the **policy iteration loop** terminates and results in the optimal policy,(see figures 5.6, 5.12, 5.17 and 5.18, for examples) toward the very last episodes.

6.3.3 Scalability, Complexity and Memory

In section 3.3, the advantages of PG algorithm is discussed. It is shown that optimizing the policy directly, rather than using state-value and state-action value based methods, solves the curse of dimensionality problem due to the continuous state space and action space. Thus, by using a stochastic parameterized policy, the dimension of the problem's state and action space is not an issue. Thus, the algorithm can be scaled up to large, high-dimensional problems. However, the parameterization space for the policy becomes crucial. The parameterization could be a simple linear or non-linear function or a complex neural network. However, as discussed in section 4.4.2, the effect of the parameter space dimension is negligible as the number of interaction steps or actions, K, exceeds the dimension of the parametrization, H. The previous discussion means that the algorithm's time complexity depends on the parameterization and the number of steps or actions through interaction that are affected by the discount factor γ. Thus, the algorithm can be scaled up by carefully selecting the parameterization, hyper-parameter, and convergence guarantee (see section 4.4.2, last paragraph). In terms of memory complexity, the algorithm only stores the policy parameterization update trajectory, the reward trajectory generated by the **experience loop** for different independent runs, and, if necessary, the gradient and baseline values.Thus, a linear data structure such as arrays with the appropriate dimensions is used. In the case of indirect shared control, for a t-independent run and a 1-D policy parameterization, a $t \times K$ and a $t \times M$ are used to store the trajectory of accumulated rewards, where M is the number of policy parameter updates.

6.4 Limitations

This section discusses the limitations of the proposed algorithm, in terms of the policy parameter range and initial values, where the algorithm performs on average as intended. It is worth pointing out here that, like all SGA algorithms (section 3.3.2), the algorithm behavior depends on the initial conditions. In cases of direct shared control, the algorithm's behavior is highly dependent on the initial values for the variables k_p and k_d. That is, if the initial values of human and/or machine action are large, the algorithm may become stuck in a local maxima, a point at which $\zeta \neq 1$ or the algorithm diverges. Thus, even if the initial value is near the higher value of the interval, $[1, 250]$, the algorithm might not satisfy the intended reward function or the task goal. For example, in the case of indirect shared control through the joystick, the algorithm stuck in a local minima for values larger than 120-degrees (see figures 6.1 and 6.2). According to [90] stucking in such local minima is reduced due to the fact that PG algorithm offers a framework for learning a stochastic policy (see section 3.3.1). That is due to the incomplete information about the interaction between the two agents, i.e., each agent has no knowledge about the action of the other. The learned local optimal stochastic policy helps overcome this due to the covariance introduced through performing the learned policy [89], [90]. While this thesis made significant progress in addressing the problem of HMCo, it is important to acknowledge that it is not without limitations. One limitation of the proposed approach is that it is based on the assumption of a stationary Markovian environment in which the state transitions and rewards are independent of the past history of the system. This assumption may not hold in practice, especially in real-world systems where there may be hidden dependencies or non-stationarity. To address this limitation, future work may need to consider more sophisticated models of the environment and the HMI that can capture these dependencies and non-stationarity.

Another limitation of the proposed approach is the sensitivity of the policy gradient algorithm to the choice of hyperparameters, such as the learning rate and the discount factor. While the algorithm is able to achieve good results by carefully selecting these parameters, this process can be time-consuming and might not always yield optimal solutions. To address this limitation, future work may need to consider more robust or adaptive methods for selecting these parameters or may need to develop more advanced variants of the policy gradient algorithm that are less sensitive to these choices.

Another challenge that is identified in this thesis is the human factor in training such an algorithm. In order to train the algorithm in real life, a human must interact with the machine, which can be time-consuming and potentially boring

or confusing for the human. To address this challenge, future work may need to consider ways to enhance the sampling complexity of the algorithm, such as using more advanced models of humans or incorporating methods for active learning or "human-in-the-loop" optimization.

Finally, it is observed that the policy gradient algorithm has a time complexity that increases linearly with the dimensionality of the policy parameterization vectors and that this complexity can lead to an increase in the number of local minima. To address this limitation, future work may need to consider techniques for reducing the dimensionality of the policy parameterization vectors or for regularizing the optimization process in order to reduce the number of local minima. Alternatively, researchers may need to explore alternative optimization algorithms that are less sensitive to these issues, such as evolutionary algorithms.

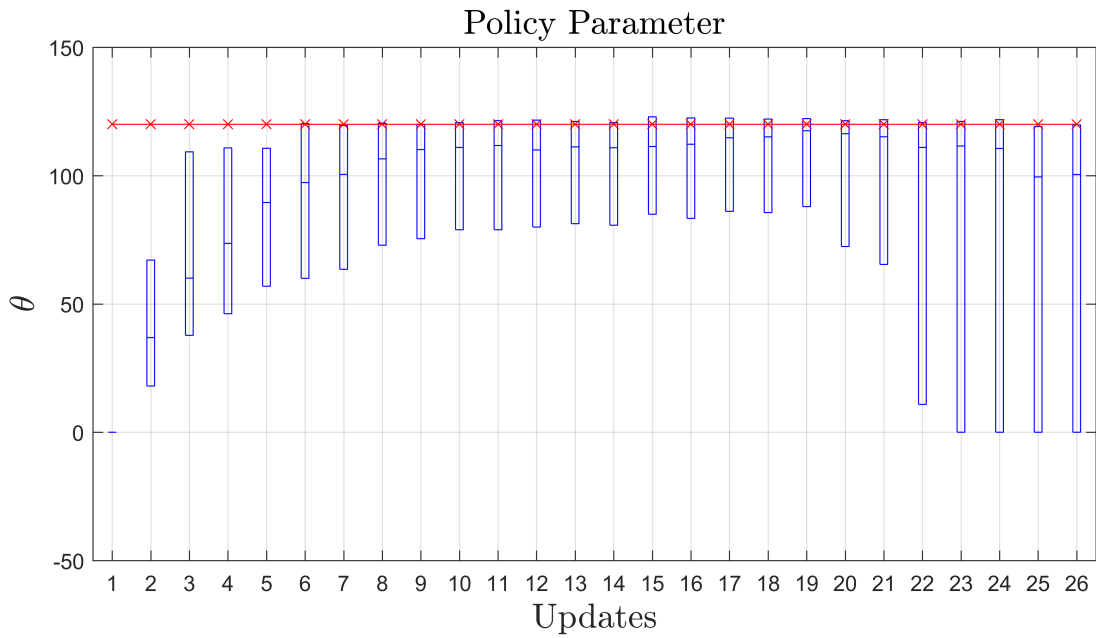

Figure 6.1: The policy convergence for random initial state. Where the $\times$ indicates the true value $\beta_{twist} = 120$ degree

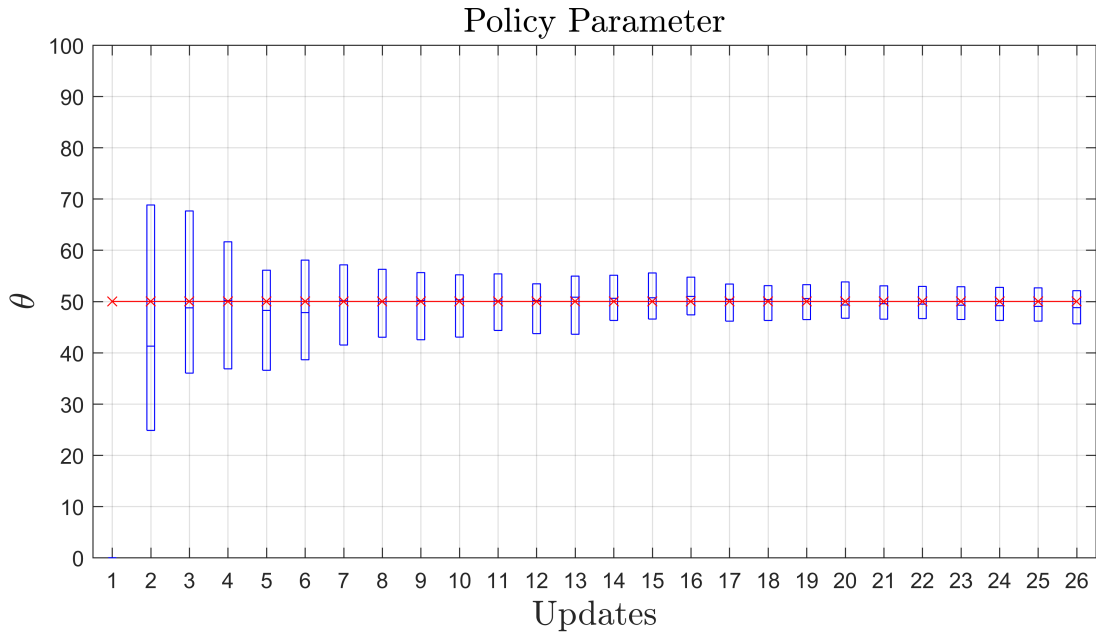

Figure 6.2: The policy convergence for random initial state. Where the $\times$ indicates the true value $\beta_{twist} = 50$ degree

7 Conclusions

The main objective of this thesis is to propose a general framework for solving the problem of Human-Machine Co-adaptation (HMCo) and to propose a solution to this problem using policy gradient algorithm. To achieve this goal, a variety of scenarios involving both direct and indirect shared control, including a joystick example and a seesaw example, are considered. In all cases, the goal is common and known to all agents, and the human is assumed to be rational with no direct communication between the human and the machine or the two machines.

Through this thesis, the aim is to develop a policy gradient algorithm that is able to converge to a solution. The solution depends on the reward function that encodes the task goal. Moreover, this thesis addresses the challenges and limitations that are inherent in HMCo problems.

7.1 Summary of main results and findings

The results show that the policy gradient algorithm is able to converge to a solution in all of the scenarios that are considered, including the joystick and seesaw examples. However, it is also observed that the algorithm is sensitive to the initial conditions, especially in cases of direct interaction, the learning rate in both cases of interaction, and it suffers from the issue of local minima due to the nature of the policy gradient algorithm. These findings suggest that while policy gradient algorithms can be an effective tool for HMCo, they may require careful tuning and selection of hyperparameters in order to achieve optimal performance.

7.2 Implications and Contributions

This thesis has several important implications for the design and operation of human-machine systems in dynamic environments, and makes several contributions to the field of human-machine co-adaptation. By proposing a general framework for solving HMCo problems using the policy gradient algorithm, and demonstrating the feasibility and effectiveness of this approach, it is shown that it can be used to adapt to changing

environments and improve the performance of human-machine systems. Additionally, by identifying and addressing some of the challenges and limitations of using policy gradients for HMCo, this thesis has the potential to inspire new approaches to other problems in the field of control and optimization and to contribute to the broader goal of developing intelligent, adaptive systems that can interact effectively with humans.

Some specific implications and contributions of this thesis include:

- Providing a general framework for solving HMCo problems using the policy gradient algorithm: By proposing a general framework for using policy gradient algorithm to solve HMCo problems, it is demonstrated that this approach is a feasible and effective way to address these types of problems
- Identifying the challenges and limitations of using the policy gradient algorithm for HMCo: By analyzing the performance of the policy gradient algorithm in different scenarios and under different conditions, several challenges and limitations are identified. These must be addressed in order to further advance the use of policy gradients for HMCo. These include the sensitivity of the algorithm to hyperparameters, the issue of local minima, and the complexity of the optimization process
- Suggesting directions for future research: This thesis has also identified several directions for future research that could help to address the challenges and limitations of using the policy gradient algorithm for HMCo. These include developing more advanced models of the environment and the human-machine interaction, exploring more robust or adaptive methods for selecting hyperparameters, and developing techniques for reducing the dimensionality of the policy parameterization vectors or for regularizing the optimization process. By addressing these challenges and limitations, the policy gradient algorithm can be used to further advance the HMCo and improve the performance of human-machine systems in dynamic environments

Bibliography

[1] Christian Krupitzer, Sebastian Müller, Veronika Lesch, et al. *A Survey on Human Machine Interaction in Industry 4.0*. 2020. DOI: 10.48550/ARXIV.2002.01025. URL: https://arxiv.org/abs/2002.01025.

[2] David Romero, Peter Bernus, Ovidiu Noran, et al. "The Operator 4.0: Human Cyber-Physical Systems Adaptive Automation Towards Human-Automation Symbiosis Work Systems". In: Sept. 2016. ISBN: 978-3-319-51132-0. DOI: 10.1007/978-3-319-51133-7_80.

[3] Mario di nardo, D. Forino, and Teresa Murino. "The evolution of man–machine interaction: the role of human in Industry 4.0 paradigm". In: *Production Manufacturing Research* 8 (Jan. 2020), pp. 20–34. DOI: 10.1080/21693277.2020.1737592.

[4] Andrey Koptelov. *Human-machine interfaces in the Industry 4.0 Era*. URL: https://www.itransition.com/blog/human-machine-interfaces.

[5] Peter Papcun, Erik Kajáti, and Jiři Koziorek. "Human machine interface in concept of industry 4.0". In: *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*. IEEE. 2018, pp. 289–296.

[6] Karim A Tahboub. "Human-Machine Coadaptation Based on Reinforcement Learning with Policy Gradients". In: *2019 8th International Conference on Systems and Control (ICSC)*. IEEE. 2019, pp. 247–251.

[7] S Russell and P Norvig. "Artificial intelligence: A modern approach, global edition 4th". In: *Foundations* 19 (2021), p. 23.

[8] Yoav Shoham and Moshe Tennenholtz. *Co-learning and the evolution of social activity*. Tech. rep. STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1994.

[9] Xun Xu, Yuqian Lu, Birgit Vogel-Heuser, et al. "Industry 4.0 and Industry 5.0—Inception, conception and perception". In: *Journal of Manufacturing Systems* 61 (2021), pp. 530–535. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2021.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0278612521002119.

[10] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, et al. "Industry 5.0: Potential Applications in COVID-19". In: *Journal of Industrial Integration and Management* 05.04 (2020), pp. 507–530. DOI: 10.1142/S2424862220500220. eprint: https://doi.org/10.1142/S2424862220500220. URL: https://doi.org/10.1142/S2424862220500220.

[11] Julian Müller. "Enabling Technologies for Industry 5.0". In: *European Commission* (2020), pp. 8–10.

[12] Gates believes that after Modern. "Post COVID-19 Industrial Revolution 5.0. The dawn of Cobot, Chipbot and Curbot". In: *Editorial Board* (2020), p. 122.

[13] S Jack Hu. "Evolving paradigms of manufacturing: From mass production to mass customization and personalization". In: *Procedia Cirp* 7 (2013), pp. 3–8.

[14] Marina Crnjac Zizic, Marko Mladineo, Nikola Gjeldum, et al. "From Industry 4.0 towards Industry 5.0: A Review and Analysis of Paradigm Shift for the People, Organization and Technology". In: *Energies* 15.14 (2022), p. 5221.

[15] Aditya Akundi, Daniel Euresti, Sergio Luna, et al. "State of Industry 5.0—Analysis and Identification of Current Research Trends". In: *Applied System Innovation* 5.1 (2022), p. 27.

[16] Sebastian Saniuk, Sandra Grabowska, and Martin Straka. "Identification of Social and Economic Expectations: Contextual Reasons for the Transformation Process of Industry 4.0 into the Industry 5.0 Concept". In: *Sustainability* 14.3 (2022), p. 1391.

[17] Rachelle Meijer. *Three challenges for adaptive human-machine interaction - NLR*. Oct. 2020. URL: https://www.nlr.org/nlr-blog/three-challenges-for-adaptive-human-machine-interaction/.

[18] Michael Wooldridge. "Intelligent Agents: The Key Concepts". In: *Multi-Agent Systems and Applications II*. Ed. by Vladimír Mařík, Olga Štěpánková, Hana Krautwurmová, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 3–43. ISBN: 978-3-540-45982-8.

[19] Abir Gallala, Atal Anil Kumar, Bassem Hichri, et al. "Digital Twin for Human–Robot Interactions by Means of Industry 4.0 Enabling Technologies". In: *Sensors* 22.13 (2022), p. 4950.

[20] Paolo Gallina, Nicola Bellotto, and Massimiliano Di Luca. "Progressive co-adaptation in human-machine interaction". In: *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*. Vol. 2. IEEE. 2015, pp. 362–368.

[21] Jean-Michel Hoc. "From human–machine interaction to human–machine cooperation". In: *Ergonomics* 43.7 (2000), pp. 833–843.

[22] Christian Krupitzer, Sebastian Müller, Veronika Lesch, et al. "A survey on human machine interaction in industry 4.0". In: *arXiv preprint arXiv:2002.01025* (2020).

[23] Guy A Boy. "A human-centered design approach". In: *The Handbook of Human-Machine Interaction*. CRC Press, 2017, pp. 1–20.

[24] Gunnar Johannsen. "Human-machine interaction". In: *Control Systems, Robotics and Automation* 21 (2009), pp. 132–62.

[25] Michael A Goodrich and Alan C Schultz. *Human-robot interaction: a survey*. Now Publishers Inc, 2008.

[26] Stefan Ehrlich and Gordon Cheng. "A computational model of human decision making and learning for assessment of co-adaptation in neuro-adaptive human-robot interaction". In: July 2019. DOI: 10.1109/SMC.2019.8913872.

[27] Yanan Li, Aran Sena, Ziwei Wang, et al. "A review on interaction control for contact robots through intent detection". In: *Progress in Biomedical Engineering* 4.3 (July 2022), p. 032004. DOI: 10.1088/2516-1091/ac8193. URL: https://doi.org/10.1088/2516-1091/ac8193.

[28] Lorenzo Flipse. "Guiding Co-Adaptation in Physically Interacting Human-Robot Teams". In: (2021).

[29] Abolfazl Mohebbi. "Human-robot interaction in rehabilitation and assistance: a review". In: *Current Robotics Reports* 1.3 (2020), pp. 131–144.

[30] Shuhei Ikemoto, Heni Ben Amor, Takashi Minato, et al. "Physical human-robot interaction: Mutual learning and adaptation". In: *IEEE robotics & automation magazine* 19.4 (2012), pp. 24–35.

[31] Luka Peternel, Nikos Tsagarakis, Darwin Caldwell, et al. "Robot adaptation to human physical fatigue in human–robot co-manipulation". In: *Autonomous Robots* 42.5 (2018), pp. 1011–1021.

[32] Ferdinando A Mussa-Ivaldi, Maura Casadio, Zachary C Danziger, et al. "Sensory motor remapping of space in human–machine interfaces". In: *Progress in brain research* 191 (2011), pp. 45–64.

[33] FA Mussa-Ivaldi and Z Danziger. "The remapping of space in motor learning and human–machine interfaces". In: *Journal of Physiology-Paris* 103.3-5 (2009), pp. 263–275.

[34] Zachary Danziger, Alon Fishbach, and Ferdinando A Mussa-Ivaldi. "Learning algorithms for human–machine interfaces". In: *IEEE Transactions on Biomedical Engineering* 56.5 (2009), pp. 1502–1511.

[35] Zachary Danziger, Alon Fishbach, and Ferdinando A Mussa-Ivaldi. "Adapting human-machine interfaces to user performance". In: *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2008, pp. 4486–4490.

[36] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, et al. "A brief survey of deep reinforcement learning". In: *arXiv preprint arXiv:1708.05866* (2017).

[37] Issam El Naqa and Martin J Murphy. "What is machine learning?" In: *machine learning in radiation oncology*. Springer, 2015, pp. 3–11.

[38] Jaime G Carbonell, Ryszard S Michalski, and Tom M Mitchell. "An overview of machine learning". In: *Machine learning* (1983), pp. 3–23.

[39] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[40] Stephen Marsland. *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2011.

[41] Tom M Mitchell and Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.

[42] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[43] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[44] Alexey B Piunovskiy. *Examples in Markov decision processes*. Vol. 2. World Scientific, 2013.

[45] Alekh Agarwal, Nan Jiang, Sham M Kakade, et al. "Reinforcement learning: Theory and algorithms". In: *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep* (2019).

[46] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292.

[47] Kai Zhu and Tao Zhang. "Deep reinforcement learning based mobile robot navigation: A review". In: *Tsinghua Science and Technology* 26.5 (2021), pp. 674–691.

[48] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[49] Ali Devran Kara and Serdar Yüksel. "Near Optimality of Finite Memory Feedback Policies in Partially Observed Markov Decision Processes." In: *J. Mach. Learn. Res.* 23 (2022), pp. 11–1.

[50] Thomas Gabel and Martin Riedmiller. "Cbr for state value function approximation in reinforcement learning". In: *International Conference on Case-Based Reasoning*. Springer. 2005, pp. 206–221.

[51] Dimitri P Bertsekas and John N Tsitsiklis. "Neuro-dynamic programming: an overview". In: *Proceedings of 1995 34th IEEE conference on decision and control*. Vol. 1. IEEE. 1995, pp. 560–564.

[52] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural networks* 21.4 (2008), pp. 682–697.

[53] J. Peters. "Policy gradient methods". In: *Scholarpedia* 5.11 (2010). revision #137199, p. 3698. DOI: 10.4249/scholarpedia.3698.

[54] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. "A survey on policy search for robotics". In: *Foundations and trends in Robotics* 2.1-2 (2013), pp. 388–403.

[55] Jens Kober and Jan Peters. "Policy search for motor primitives in robotics". In: *Machine learning* 84.1 (2011), pp. 171–203.

[56] Richard S Sutton, David McAllester, Satinder Singh, et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12 (1999).

[57] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3 (1992), pp. 229–256.

[58] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural Networks* 21.4 (2008). Robotics and Neuroscience, pp. 682–697. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2008.02.003. URL: https://www.sciencedirect.com/science/article/pii/S0893608008000701.

[59] Daeyeol Lee, Hyojung Seo, and Min Whan Jung. "Neural basis of reinforcement learning and decision making". In: *Annual review of neuroscience* 35 (2012), p. 287.

[60] Yael Niv. "Reinforcement learning in the brain". In: *Journal of Mathematical Psychology* 53.3 (2009), pp. 139–154.

[61] Kenji Doya. "Reinforcement learning: Computational theory and biological mechanisms". In: *HFSP journal* 1.1 (2007), p. 30.

[62] Wolfram Schultz, Peter Dayan, and P Read Montague. "A neural substrate of prediction and reward". In: *Science* 275.5306 (1997), pp. 1593–1599.

[63] Hannah M Bayer and Paul W Glimcher. "Midbrain dopamine neurons encode a quantitative reward prediction error signal". In: *Neuron* 47.1 (2005), pp. 129–141.

[64] Hsing-Chen Tsai, Feng Zhang, Antoine Adamantidis, et al. "Phasic firing in dopaminergic neurons is sufficient for behavioral conditioning". In: *Science* 324.5930 (2009), pp. 1080–1084.

[65] Elizabeth E Steinberg, Ronald Keiflin, Josiah R Boivin, et al. "A causal link between prediction errors, dopamine neurons and learning". In: *Nature neuroscience* 16.7 (2013), pp. 966–973.

[66] James C Houk, Joel L Davis, and David G Beiser. "Reward-related signals carried by dopamine neurons". In: (1994).

[67] Henry H Yin and Barbara J Knowlton. "The role of the basal ganglia in habit formation". In: *Nature Reviews Neuroscience* 7.6 (2006), pp. 464–476.

[68] James C Houk, Joel L Davis, and David G Beiser. *Models of information processing in the basal ganglia*. MIT press, 1995.

[69] Karim A Tahboub. "Intelligent human-machine interaction based on dynamic bayesian networks probabilistic intention recognition". In: *Journal of Intelligent and Robotic Systems* 45.1 (2006), pp. 31–52.

[70] Rafal Bogacz. "Dopamine role in learning and action inference". In: *Elife* 9 (2020).

[71] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. "Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems". In: *The Knowledge Engineering Review* 27.1 (2012), pp. 1–31.

[72] Dimitri Bertsekas. "Multiagent reinforcement learning: Rollout and policy iteration". In: *IEEE/CAA Journal of Automatica Sinica* 8.2 (2021), pp. 249–272.

[73] Dimitri Bertsekas. "Multiagent value iteration algorithms in dynamic programming and reinforcement learning". In: *Results in Control and Optimization* 1 (2020), p. 100003.

[74] Sushmita Bhattacharya, Siva Kailas, Sahil Badyal, et al. "Multiagent rollout and policy iteration for POMDP with application to multi-robot repair problems". In: *arXiv preprint arXiv:2011.04222* (2020).

[75] Henri P Gavin. "The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems". In: *Department of Civil and Environmental Engineering, Duke University* 19 (2019).

[76] Petar V Kokotovic. "The joy of feedback: nonlinear and adaptive". In: *IEEE Control Systems Magazine* 12.3 (1992), pp. 7–17.

[77] Petros A Ioannou and Jing Sun. *Robust adaptive control*. Courier Corporation, 2012.

[78] Zhiyong Sun, Brian DO Anderson, Mohammad Deghat, et al. "Rigid formation control of double-integrator systems". In: *International Journal of Control* 90.7 (2017), pp. 1403–1419.

[79] Chao-Wei Lin, Mun-Hooi Khong, and Yen-Chen Liu. "Experiments on human-in-the-loop coordination for multirobot system with task abstraction". In: *IEEE Transactions on Automation Science and Engineering* 12.3 (2015), pp. 981–989.

[80] Jonathan Kofman, Xianghai Wu, Timothy J Luu, et al. "Teleoperation of a robot manipulator using a vision-based human-robot interface". In: *IEEE transactions on industrial electronics* 52.5 (2005), pp. 1206–1219.

[81] David B Van de Merwe, Leendert Van Maanen, Frank B Ter Haar, et al. "Human-robot interaction during virtual reality mediated teleoperation: How environment information affects spatial task performance and operator situation awareness". In: *Virtual, Augmented and Mixed Reality. Applications and Case Studies: 11th International Conference, VAMR 2019, Held as Part of the 21st HCI International Conference, HCII 2019, Orlando, FL, USA, July 26–31, 2019, Proceedings, Part II 21*. Springer. 2019, pp. 163–177.

[82] Thomas B Sheridan. "Human–robot interaction: status and challenges". In: *Human factors* 58.4 (2016), pp. 525–532.

[83] Riccardo Muradore and Paolo Fiorini. "A review of bilateral teleoperation algorithms". In: *Acta Polytechnica Hungarica* 13.1 (2016), pp. 191–208.

[84] Chris L Baker and Joshua B Tenenbaum. "Modeling human plan recognition using Bayesian theory of mind". In: *Plan, activity, and intent recognition: Theory and practice* 7 (2014), pp. 177–204.

[85] Chris L Baker, Julian Jara-Ettinger, Rebecca Saxe, et al. "Rational quantitative attribution of beliefs, desires and percepts in human mentalizing". In: *Nature Human Behaviour* 1.4 (2017), pp. 1–10.

[86] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. "Action understanding as inverse planning". In: *Cognition* 113.3 (2009), pp. 329–349.

[87] Nikolos Gurney, Stacy Marsella, Volkan Ustun, et al. "Operationalizing theories of theory of mind: A survey". In: *Computational Theory of Mind for Human-Machine Teams: First International Symposium, ToM for Teams 2021, Virtual Event, November 4–6, 2021, Revised Selected Papers*. Springer. 2023, pp. 3–20.

[88] Antti Oulasvirta, Jussi PP Jokinen, and Andrew Howes. "Computational Rationality as a Theory of Interaction". In: *CHI Conference on Human Factors in Computing Systems*. 2022, pp. 1–14.

[89] Andrew Y Ng. *Shaping and policy search in reinforcement learning*. University of California, Berkeley, 2003.

[90] Andrew Ng. "Machine learning lectures". In: *https://cs229tanfordėdu/notes2022fall/main*$_{n}ote$ (2022).

[91] Jon Doyle. *Artificial intelligence and rational self-government*. Carnegie-Mellon University. Department of Computer Science, 1988.