UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FíSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

A FRAMEWORK FOR LEARNING CONTINUOUS ACTIONS FROM CORRECTIVE ADVICE

TESIS PARA OPTAR AL GRADO DE
DOCTOR EN INGENIERÍA ELÉCTRICA

CARLOS EDUARDO CELEMIN PAEZ

PROFESOR GUÍA:
DR. JAVIER RUIZ DEL SOLAR

MIEMBROS DE LA COMISIÓN:
DR. EDUARDO MORALES MANZANARES
DR. ROBERT BABUSKA
DR. MARCOS ORCHARD CONCHA

SANTIAGO DE CHILE
2018

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE DOCTOR EN INGENIERÍA ELÉCTRICA
POR: CARLOS EDUARDO CELEMIN PAEZ
FECHA: 2018
PROF. GUÍA: DR. JAVIER RUIZ DEL SOLAR

A FRAMEWORK FOR LEARNING CONTINUOUS ACTIONS FROM CORRECTIVE ADVICE

Esta tesis presenta un método que permite que usuarios no expertos enseñen agentes a ejecutar tareas complejas durante tiempo de ejecución, con el principal propósito de acelerar la convergencia del aprendizaje y mejorar el desempeño final de las políticas aprendidas. En este sentido, se propone *COrrective Advice Communicated by Humans* (**COACH**), un framework interactivo para entrenar políticas con vagas correcciones respecto a las acciones ejecutadas, las cuales son cambios relativos de la magnitud de las acciones que están siendo ejecutadas. Así, los usuarios sugieren correcciones como: incrementar la fuerza, reducir la velocidad, ir más hacia la izquierda, etc.

Inicialmente, se propone un esquema de aprendizaje que permite a humanos enseñar políticas de acciones continuas por medio de correcciones correctivas, para problemas de acciones de una dimensión. Se incluye en el framework de aprendizaje un módulo que representa las intenciones del profesor, el cual se basa en la historia pasada de las correcciones. Luego, el framework se extiende a problemas de acciones de más de una dimensión, incluso para casos en los que las correcciones del usuario no están en el mismo espacio de la política.

Adicionalmente, el **COACH** propuesto es combinado con aprendizaje reforzado Policy Search con el fin de obtener la ventajas de ambas fuentes de información (correcciones humanas y funciones de recompensa) en el proceso de aprendizaje. Se proponen dos enfoques híbridos que combinan los dos enfoques, uno secuencial y uno simultáneo. Los resultados muestran que estos esquemas se benefician de las ventajas de cada uno de sus componentes, es decir se obtiene i) rápido progreso al principio del proceso de aprendizaje, y ii) aprendizaje robusto a errores humanos, junto con optimalidad local.

Además, este enfoque híbrido es extendido para entrenar primitivas de movimiento. Así, las ventajas previamente mencionadas son extendidas para aprender también políticas representadas como Dynamic Movement Primitives (DMP) y Probabilistic Movement Primitives (ProMP), las cuales son convenientes para aprender trayectorias.

El uso del enfoque propuesto es validado en muchos problemas tanto simulados como reales, con variadas características, recorriendo problemas de equilibrio, navegación con robots bípedos en el contexto del fútbol robótico, y también habilidades motoras con brazos robóticos en tareas como escritura de símbolos y el conocido juego "emboque". Los resultados muestran que el conocimiento de los usuarios no expertos puede apalancar procesos de aprendizaje de máquina, guiando hacia desempeños más altos con respecto a otros enfoques de aprendizaje de máquina interactivo y de aprendizaje reforzado, e incluso superando las capacidades de usuarios aprendiendo a tele-operar los agentes. Adicionalmente, los métodos presentados obtienen convergencias las cuales varían desde 3 hasta más de 40 veces más rápido que otras técnicas, dependiendo del problema.

Abstract

This thesis introduce methods that allow non-expert users to train agents to execute complex tasks during execution time, considering to accelerate the learning convergence and to improve the performance of the learned policies as main purpose. In this regard, it is proposed *COrrective Advice Communicated by Humans* (**COACH**), an interactive framework for training policies with vague and coarse corrections of the executed actions, which are relative changes of the actions magnitudes that are being executed. So the users advise corrections of the actions during execution time of the task, for instance the teacher advises corrections like, increase the force, decrease the velocity, go more to the left, etc.

Initially, it is proposed a learning scheme that allows humans to teach continuous action policies with corrective advice for one-dimensional action problems. A module that represents the teacher's intentions based in past history is included in the learning framework. Then, the framework is extended to multi-dimensional action problems, even for cases wherein the corrections of the user are not in the same domain that the policy.

Furthermore, the proposed **COACH** is combined with Reinforcement Learning Policy Search for obtaining the advantages of both sources of information (human corrections and encoded rewards) in the learning process. Two hybrid learning schemes that combine the two approaches are proposed, one sequential, and a second simultaneous. Validation experiments show that these learning schemes benefit from the advantages of each of its parts, that obtains: i) fast improvement in the first stage of the learning porcess, and ii) learning robust to human mistakes, and local optimality, both based in the cost function.

Additionally, the hybrid approach of human corrections and Policy Search is extended for training movement primitives. So the advantages of the previously metioned hybrid scheme for Markov Decision Processes is extended to policy representations like the Dynamic Movement Primitives (DMP) and the Probabilistic Movement Primitives (ProMP), which are convenient for learning trayectories.

The use of the proposed framework is validated in several simulated and real problems with varying characteristics, ranging from balancing tasks, navigation with biped robots in the context of robot soccer, and also motor skills with robot arms in tasks like writing and the game *ball in a cup*. Results show that the knowledge of non-expert users can leverage learning processes, heading to higher performances with respect to other Interactive Machine Learning, Reinforcement Learning approaches, and even outperforming the capabilities of users learning to teleoperate the agents. Additionally, the introduced methods obtain convergences which range from 3 to more than 40 times faster than other techniques, depending on the problem.

A cada una de las personas que me encontré en este camino por aquellos rincones del mundo que visité, gracias por compartir desde unas horas hasta algunos años de su vida durante esta etapa. De algunos aprendí sobre robots, pero de todos aprendí sobre la vida.

Agradecimientos

En primer lugar quiero agradecer a mi familia, mis papás Maribel y Armando junto con mis hermanos mando y milo que siempre me han apoyado en todos mis proyectos, sin importar la distancia que a veces nos separa, siempre siento de cerca todo su afecto y apoyo, siempre los tengo en mi mente y mi corazón. Especial dedicación a mis abuelos Pedro y Eduardo que partieron en estos años pero dejaron su legado en la familia.

Agradezco a mi profesor guía Javier Ruiz del Solar por aconsejarme en estos años, por confiar y darme la independencia que tuve para el desarrollo de este doctorado. A los profesores que me recibieron en las diferentes estancias: Eduardo Morales y Enrique Sucar en el INAOE, a Manuela Veloso en CMU y a Jens Kober en TU Delft. Especial agradecimiento a Guilherme Maeda, quien me recibió en TU Darmstadt compartiendo su experiencia y consejos conmigo.

En la Universidad, también le agradezco a mis compañeros de laboratorio y a los miembros del equipo UChile Robotics Team por la ayuda, por las experiencias compartidas, hago un reconocimiento a su inagotable e inspirado trabajo, además de la búsqueda permanente de respuestas a preguntas como "¿Qué robot?".

A la Sociedad de Poetas Expatriados, les agradezco por compartir tantos momentos, por ser el soporte más cercano, por todo el tiempo que compartimos y por la grandísima amistad que hemos creado. Todo fue más fácil y divertido gracias a su presencia. A Kathe, por su apoyo y paciencia en los momentos más complejos.

Especial agradecimiento a la familia Jaramillo Montoya: a Francisco, sus padres y hermanas quienes desde Temuco me abrieron las puertas de su hogar y me hicieron sentir parte de ellos, siempre inyectándome mucha energía. A Sebastian gracias por estar siempre de forma incondicional y por las sopas con merquén. A Pavlinka por la alegría, la perspectiva y la música.

Gracias también a mis amigos de toda la vida que me apoyan desde lejos, a Carlos y a Juan. A Fernando que lo reencontré más cerquita. A mis Compadres de la tierrita, Camilo y Jesús, con quienes continuamos compartiendo la música, "eche".

También agradezco el tiempo y esfuerzo para revisar esta tesis a los profesores miembros de esta comisión Eduardo Morales, Robert Babuska y Marcos Orchard.

Finalmente, gracias a la beca CONICYT-PCHA/Doctorado Nacional/2015-21151488, el proyecto FONDECYT 1161500, y al Advaced Mining Technology Center AMTC, Universidad de Chile, que financiaron esta tesis y a Chile por recibirme y adoptarme durante este tiempo.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Definition of the Problem

The ongoing developments of the new industry revolution are demanding specific characteristics of the applied technology, which will shape the advance during the coming years. Among some other principles, the cyber physical systems industry is characterized by the *Flexibility* in the development, maintenance, and the operation of automated systems, and *Integration of customers* for obtaining products able to be personalized to specific and individual needs [42]. These features shrink distances between system designers and end-users —operators of factory tools are also considered end-users— and make more necessary to have techniques that allow non-experts to design or modify the behavior of the systems.

The design of decision making systems for robots has an important role in most of the automation processes, especially in fields like manufacturing, agriculture, food, mining, aerospace, etc. along with upcoming household applications. So machine learning techniques can give to end-users capabilities for customizing and adapting robots to new situations or applications. Techniques that include human knowledge to leverage the process of learning agents have been developed in the last decades, Learning from Demonstration (LfD), Programming by Demonstration (PbD), Imitation Learning (IL), Interactive Machine Learning (IML), among others are the key words for the set of methods that include human teachers completely or partially in the learning loop. Through this thesis we refer to all this complete set of learning methods in general with the acronym IML.

These solutions are especially given for applications wherein the policy's definition or update requirement is detected by a final user, who likely is not a skilled programmer able to modify either the policy's code, or the goal definition for an autonomous RL learning process. In this case, the training time could be decreased with the inclusion of humans into the agent-environment interaction, for transferring the knowledge that is possessed or discovered during interaction.

The most widely known IML approaches are based on LfD. In this paradigm the agent receives information about how to do the tasks from several demonstration sources. The advantage of this approach is the fact that the information given by the user is communicated in a natural and intuitive way, but as long as the teacher is able to perform the demonstration, it means, he/she has to be an expert at executing the task.

This required expertise is the main disadvantage of LfD. Therefore it is limited to skilled users who could teach to solve a problem at a specific context; otherwise, non-experts would provide low quality demonstrations that require additional tuning.

On the other hand, there is a kind of techniques called Learning from critique; where the human teacher provides feedback with a signal of the desirability of the actions executed by the agent. The teacher supplies a measure of approval/disapproval. Most of these types of frameworks are based on RL algorithms in which the reward function is replaced by the human reinforcement.

The advantage of this kind of learning is that it does not need expert teachers (i.e. experts performing the task), since a non-expert teacher could evaluate whether the executed actions are appropriate or not. When the user considers the action inappropriate, the agent is forced by punishments to explore other actions until a better action is performed according to the teacher's criteria. However, evaluative feedback is not as intuitive or natural as feedback in the actions domain, people prefer "to tell the robot what to do" [94]. It means they prefer to provide feedback in the actions domain rather than evaluate the executed actions.

As it was mentioned, IML has two main branches depending on the type of information provided by the human teacher in the learning loop. Each of both branches has advantages that would be desirable to have in one scheme, i.e. it is desirable to have an algorithm that allows **non-experts users** to correct the agent while it is interacting with its environment, but based on a **feedback in the actions domain**, not like in an evaluative scheme.

Therefore, taking the previous idea into account, the proposed problem to approach is the fact that in an *LfD* scheme it could be eliminated the requirement of perfect demonstrations from expert users of the task execution to teach, but yet to obtain high performance policies of expert users level or better.

To solve this problem it is necessary to approach other intrinsic sub-problems of these approaches:

- The time delays that exist from the instant that the user observes the action execution until he/she communicates to the agent a feedback signal.
- The human teachers have different observability from the one of the agents (it could be more or less information available from the environment depending on the problem's nature).
- It is not easy for humans to provide feedback to high dimensional state-actions spaces, especially when there is no direct mapping between the space observed by the human (effector space) and the actuator space where the feedback has to take influence.
- The occasional mistakes human perform at advising learning agents that might harm the policy that is being learned.

With the solution of this list of problems, Interactive Machine Learning algorithms could have more adaptation capabilities to the users. Then the teacher does not have to adapt to the learning frameworks, instead machines would adapt to human users.

This thesis aims to propose a framework that eliminates or at least decreases the aforementioned problems, specifically oriented to most of the challenges of skill learning, which are focused on low level continuous actions problems. The main feature that is addressed in the proposal is about using the feedback as advice of corrections or modifications of the executed actions rather than actual demonstrations, so the user does not necessarily have to be qualified to operate the agent in order to perform the task. In other words, while the teacher observes the task execution by the current policy, he/she is advising how it has to be gradually modified.

1.2 Objectives

1.2.1 General Objective

To design, implement and validate with robotic tasks, a learning from corrections based framework, which models the human feedback and incorporates it into the learning process, in order to facilitate and improve the knowledge transference process between a human teacher and a machine learner.

1.2.2 Specific Objectives

- To propose, implement, and validate a framework that allows robots to learn tasks in continuous actions domains of one dimension, based on human feedback signals in the actions space. All in an on-line and interactive fashion that contribute to the reduction of the learning time and the improvement of the final performances regarding other learning approaches.
- To design a module that models the past of the feedback signals provided by a human teacher, for computing the correction value used in the current update, and to include this module in the proposed framework.
- To extend the learning framework to multi-dimensional action problems, even to tasks with non-obvious correspondence between the state-action space of the human teacher and the agent.
- To implement and validate the proposed framework, over real robotic platforms that perform complex tasks.

1.3 Hypotheses

- An on-line IML strategy for agents with continuous actions based on the feedback of human advice of corrections over the actions space, will contribute with reduction of the learning time, and also to improve the agent's final performance. This is with respect to the performance achieved by reinforcement based IML or autonomous agent. This type of learning process will allow learning from non-expert human teachers, even when they are not skilled to control the agent manually.
- The use of past information about the human feedback, during the learning process, will allow having user adaptive algorithms that facilitate the knowledge transference between the human teachers to the machine learners. The human decision processes take into account the current states like MDPs, but also the past history. Therefore, it is necessary to consider the use of more information than what was used by an MDP, when human teachers participate in the learning loop.
- The generalization of the proposed learning strategy based on corrective advice, for tasks with multiple actions, will attain improvements of the learning time and the final performances in more complex applications. A module that solves the correspondence issues between the human corrections and the agent's actions, can support the policy updating process for those problems where the human advice is not in the same domain or space that the actual policy.

1.4 Contribution

This thesis contributes to the fields of Machine Learning applied to robotics. The methods proposed belong specifically to the area of IML, wherein humans are in the learning loop.

Currently, most of the methods used for policy learning with robots are widely covered by RL, and Learning from Demonstration. These approaches possess limitations in time and availability of expert demonstrators, respectively, that make unfeasible to obtain successful learning processes for several applications with real physical systems.

In this regard, this thesis presents COrrective Advice Communicated by Humans (COACH), a framework which allows humans to teach agents with corrective feedback during policy execution in the context of continuous action Markov Decision Processes (MDPs). This method benefits from the user's rough insights about how the task may be corrected. The insights are part of the prior knowledge of the user or discovered along the agent's learning process. From the user's point of view, it is easier to have the rough insights about how to correct a policy than to gather high quality task execution demonstrations. Therefore, the proposed learning method can be applied to a wider range of applications than conventional LfD techniques, since the human teachers who advise the corrections do not need to be experts at the task execution. Additionally, the corrections based on the teacher's knowledge are more efficient than RL exploration techniques. Humans may conclude from the environment interaction, that some actions are not necessary to explore in certain regions of the state space, then, the learning process can be accelerated.

COACH updates the policy model immediately after each piece of advice is received from the teacher during policy execution, this is based on stochastic gradient descent. Moreover, the algorithm models the teacher's intention with the prediction of the corrections for adapting the size of the correction to be performed. Also, COACH uses a Credit Assigner module borrowed from TAMER [47, 56], which deals with the delay of the human response, in order to match the corrections with the actions the teacher expects to modify.

Since cost functions sometimes are not completely understood by human teachers, we introduce a method that combines COACH with Reinforcement Learning in order to benefit from the encoded reward functions that represent the objective of the task. This hybrid method allows to reach optimal policies faster than classical RL methods. The combined method is intended for Policy Search RL methods, which learn directly in the policy domain without estimating the value function, similarly to the original COACH.

Several skills can be modelled with trajectories encoded with movement primitives like Dynamic Movement Primitives (DMP) [41], primitives based on Gaussian Mixture Models [44], or Probabilistic Movement Primitives (ProMP) [79]. So for these policy representations, all the developments of COACH intended for learning policies in the context of MDPs are extended to learn trajectories. Both approaches of COACH with pure human feedback and the hybrid of COACH and Policy Search are fixed to train motor primitives, with the same advantages of learning faster and reaching higher performances than the obtained with conventional LfD and RL strategies.

All the developed algorithms are validated in simulations and with real systems, ranging from very simple toy problems like the "Mountain-Cart" to complex skills with real robots. The type of the explored tasks varies widely:

- **Navigation** as in the case of *Ball-Dribbling with biped robots* in the context of robot football soccer (at RoboCup competition) in simulation and with the real Nao robot.
- **Balancing** agents as the simulated *Cart-Pole*, and *Bike balancing.* on the other hand the methods were validated with real physical systems like the *Inverted swing-up pendulum* and an *Inverted wedge.*
- **Motor skills with robot arms** in a simulated task of reaching movements through via-points, *learning to write symbols* in simulation and with a real UR5 arm, and finally the *ball in a cup* task with a real lightweight Kuka arm.

In general, the results of the experiments outperform important state of the art approaches in terms of learning time and the performance of the learned policies, which validate the scope and the advantages of the methods presented in this thesis.

1.5 Outline

The contributions of this thesis are presented in the chapters 3-5, wherein the original framework COACH is presented and validated, along with its variations.

- **Chapter 2** presents a brief introduction to the context of robots learning. A short overview of RL and IML is discussed, specifically, the methods more related and that have inspired this work.
- **Chapter 3** introduces the interactive learning method based on corrective advice COACH. The learning strategy based on stochastic gradient descent and its modules is explained in detail and validated with several MDP problems. The performance of COACH is compared to other interactive learning, autonomous RL methods, and also with the learning curves of human operators that execute the tasks.
- **Chapter 4** discusses the combination of COACH with Reinforcement Learning (Policy Search), in which the learning agent uses the reward function for evaluating and updating the policy after some roll-outs, whereas the human corrections are used as "guidance" for the exploration in the actions domain. The approach is validated with simulated and real systems.
- **Chapter 5** presents the application of the methods proposed in Chapters 3 and 4 to problems of motor skills learning, which are controlled with movement primitives. Both pure COACH and the hybrid COACH with Policy Search are described for this type of policy representation. Additionally, this chapter discusses considerations of COACH for learning policies defined in the end effector domain, and in the actuator domain. This considerations can be applied in the context of COACH with MDPs (cases of Chapters 3 and 4).
- **Chapter 6** gathers the main conclusions obtained with the contributions and the experiments described in this thesis. Additionally, a short discussion of the inmediate future work is presented.

Chapter 2

State of the Art

In nature, learning capabilities give to animals benefits that are desirable in artificial systems. In biological systems there are two main benefits of learning: first, to adapt to different and new circumstances of the world; second, to reduce the amount of genetic information to be inherited. These benefits could be available to robots when Machine Learning is applied [21]. In the robotic case, the first aforementioned benefit is about the changes of the robot's environment, the second one, is about specification of the programming (genetic material).

The use of Machine Learning (ML) for solving problems in the context of robotics has been in classification and pattern recognition domains, but also for challenges in domains of control and decision making. The two main branches of ML developed for decision making problems with robots (the topic of interest of this thesis) are: Autonomous Learning, and IML. For the first case, in [21] is reported that autonomous learning is mostly based on Genetic Programming (GP) [14] and Reinforcement Learning (RL) [95], but RL has been most widely and successfully explored since long time ago [30]. The second branch is IML [27]. Both perspectives learn a policy which is a function that maps observations (states) of the robot's environment onto the actions space, and then the action is executed by the robot.

2.1 Reinforcement Learning

RL is an approach of autonomous learning wherein the robot has to explore the solution space, for deriving a policy from its experience. In this scenario the robot has a set of available actions, and depending on the states, the policy has to compute the associated action to accomplish the task. The agent or robot observes its actions influence over the environment, and then an assessment of the current policy is given based on the Reward Function, as shown in Figure 2.1. This function is one of the outstanding features of RL; it is the element that the user designs for formalizing what is the objective to the agent, i.e. the signal which determines the purpose or goal of the task to be performed [95].

The objective of RL algorithms is to maximize the cumulative reward obtained by the agent. The reward function, rather than policy is the most succinct, robust, and transferable

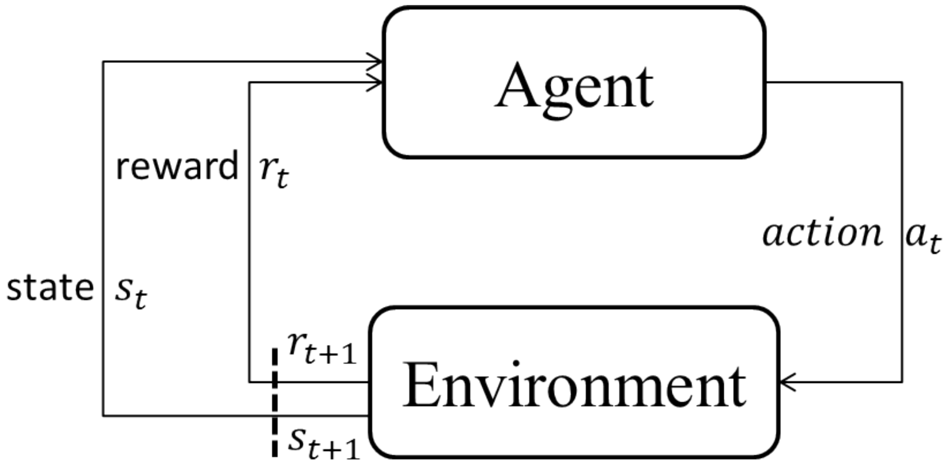


Figure 2.1: Reinforcement Learning scheme

definition of the task [76]. For complex domains, designers face the problem of defining a good reward or in general an objective function, to let the agent efficiently attain an intended goal [33].

RL has been successfully applied in several domains: games like a simple simulation of soccer with two players in a 4x5 grid [66], the game backgammon [100], Chess [15, 105], the Mountain-Car problem [95], the learning to drive a bicycle problem [84], the Cart-pole problem [95], packet routing in communication networks [18], and so many other applications.

There have been reported less cases of RL with real robots. Some of them are: In [86] an armed robot learn to juggle a "devil sticking" task. Also, robots controlling arms learning to solve problems like *ball-in-a-cup* and *ball-paddling* have been presented in [58], and the task of *hitting in table tennis*, and movements for tasks of throwing darts and throwing balls in [60]. Walking engines have been learned with RL algorithms by four legged [61] and biped robots [113] among others, usually those experiments are developed in the context of robotic soccer.

There are RL applications in the robotic soccer context to other problems like penalty scoring [40], ball-dribbling [64], and some other of behavior referenced in [85]. Another very challenging problem approached with RL is the helicopter flight [13, 45]. The aforementioned cases are just few of the examples available in literature, more applications and considerations of RL in robotics could be found in [43, 57].

The use of machine learning techniques such as RL allows robots and agents in general, to address complex decision-making tasks. However, one of the main limitations of the use of learning approaches in real-world problems is the large number of learning trials required to learn complex behaviors (many trials are required for the Reward Function tuning and for the learning process by itself).

This can make non-viable the use of many learning approaches in problems where the implementation of learning trials with real robots, in the real world, may have a high cost. This drawback can be addressed by using human feedback during learning, i.e., the learning

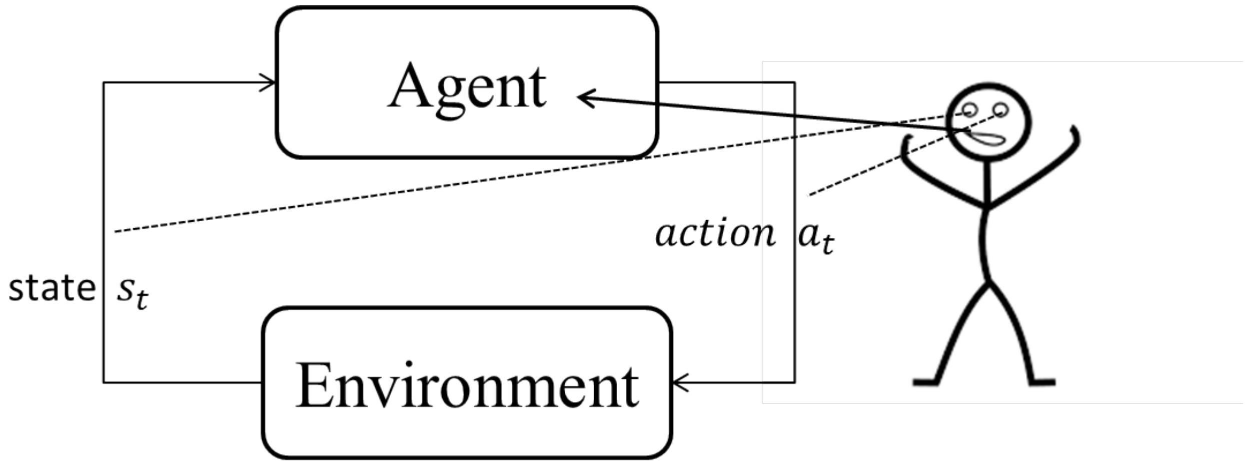


Figure 2.2: Interactive Machine Learning scheme

process can be assisted by a human teacher who supervises the agent-environment interaction. The participation of a human in the learning loop is given in the second perspective of Machine Learning with robots presented below.

2.2 Interactive Machine Learning

IML is another perspective of machine learning in which a human teacher takes part in the agent's learning process, for supervising the policy's improvement. Most of the general ideas about IML have been summarized in [27, 9, 16, 17, 31, 4]. Some developments have been applied for learning classifier systems [35, 108, 5, 77], but in this thesis the focus is in methods of IML for learning decision making systems with continuous actions (regression).

There have been proposed several methods of IML regarding the type of skills to learn, the type of users' feedback, the type of Human-Machine Interfaces (HMI) used, etc. There are two main schemes for using human feedback in order to modify the policy of a learning agent: human feedback in the actions domain and human feedback in the evaluative domain.

2.2.1 Human feedback in actions domains

Learning from Demonstration (LfD) is a learning paradigm in which a teacher provides demonstrations of the desired policy, and the agent reproduces the demonstrated behavior (Figure 2.3). The actions that the learner associates to the states could be classified in low-level actions for motion control, basic high-level actions (often called action primitives) and complex behavioral actions for high-level control [9]. There are some works which explores different applications of *LfD* in robotic domains as soccer [3, 36], driving applications [28], humanoids motion [109, 19], navigation [88, 111], grasping and manipulation [96, 65], and many other problems. *Imitation Learning* is a type of *LfD* in which the demonstrations

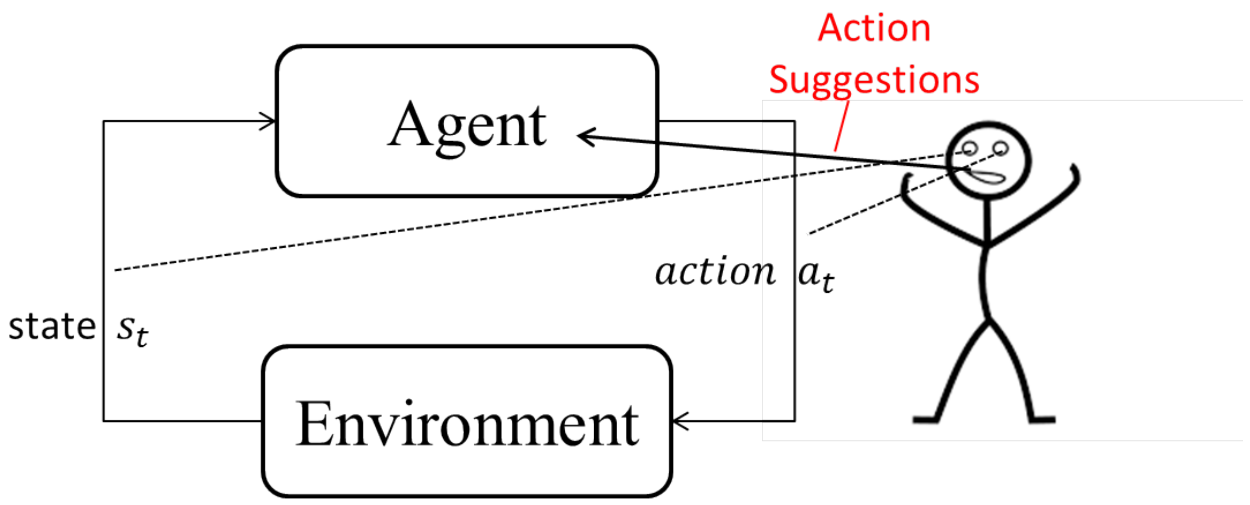


Figure 2.3: Learning from Demonstration scheme

are executed in a platform that is different to the one of the learning agent (e.g., a robot). Since some of the most frequent providers of demonstrations are humans, the challenges in this case consist of solving the problems of what and how to imitate [20]. For example, in [78] it is proposed a method that learns high level representations of the tasks from the demonstrations.

There exists other paradigms which use *LfD* at important stages, for example Active Learning queries for demonstrations for unknown instances [24]. Human-Agent Transfer (HAT) is a learning methodology that combines *LfD*, transfer Learning, and RL. It first learns from human teleoperation, and then uses transfer learning to use the knowledge in a RL process that improves the policy derived from demonstrations. The proposal was validated with the keepaway problem used for proof of concept [98]. Other case is a bridge between *LfD* and RL called *Apprenticeship Learning* [1], at its first stage a human begins controlling the agent; then in a second stage, using Inverse Reinforcement Learning (IRL) [76, 112] and the collected data, it is derived a reward function which represent the target of the task, according to the demonstrations given by the trainer; finally the agent continues improving the policy at a RL framework, using the reward function deduced with IRL in the preceding stage.

There are algorithms of *LfD* which are focused in learning from corrective demonstrations, instead of deriving a policy out of the demonstrations, keeping hand-coded algorithms as the primary source of the action policy, and using the demonstration data only to make exceptions as needed [29, 71, 72, 69].

In continuous actions domains, *Advice operators* have been proposed for correcting a current policy, the trainer chooses a set of observation and action pairs after its execution with an advice operator associated, it could be increasing (decreasing) the magnitude of the respective action, the data modified is attached to the demonstrations dataset for a policy re-derivation [6, 8, 7].

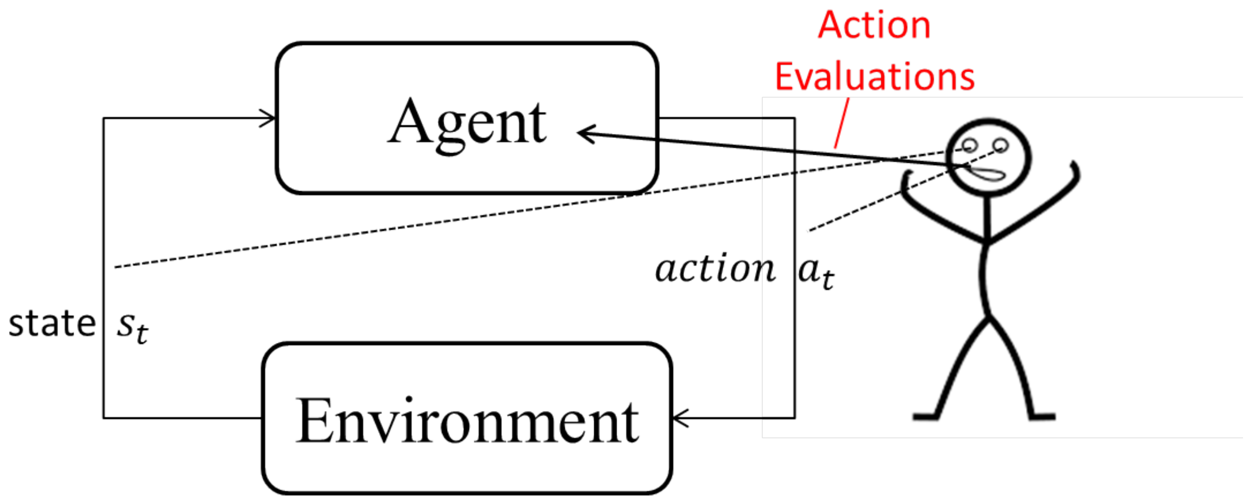


Figure 2.4: IML with evaluative feedback scheme

In most of the described approaches it is required accurate demonstrations and expert users as trainers. Methods based on the *Advice Operator* paradigm do not need an expert trainer, but have the drawback that always deducing the policy from a dataset in a batch learning scheme. The dataset is increased with the set of state-action pairs selected for the correction in every advising phase, increasing the computational burden of the policy re-derivation. Another drawback is that the policy re-derivation, is done off-line, producing that the effect of the advice correction only can be seen after the new policy is obtained. For instance, in [70] Corrective demonstrations and *Advice operators* were sequentially applied for improving the walk stability of a Nao humanoid robot.

2.2.2 Human feedback in evaluative domains

Under this paradigm, non-expert users can evolve decision making systems, even on-line, by delivering their feedback interactively as an evaluative (approval or disapproval) signal in a RL framework. In this paradigm the reward is partially or completely given by the human [73, 99, 63, 93, 81, 110]. In [102] the Interactive Reinforcement Learning enables a human user to provide positive and negative rewards in real time in response to robot actions, and to advise anticipatory guidance input that constrains action selection choice and guides the learner towards the desired behavior.

Since a human reward may have a different meaning with respect to an encoded MDP (Markov Decision Process) reward function, which is the basic reinforcement used in the conventional RL approaches, a series of works have analyzed how to model the human reinforcement [104, 103]. An important consideration taken into account for learning agents with human rewards in contrast to MDP rewards is that "MDP reward is informationally poor yet flawless and human reinforcement is rich yet flawed" [49]. The *shaping* approach allows interactively training an agent through signals of positive and negative reinforcement [48].

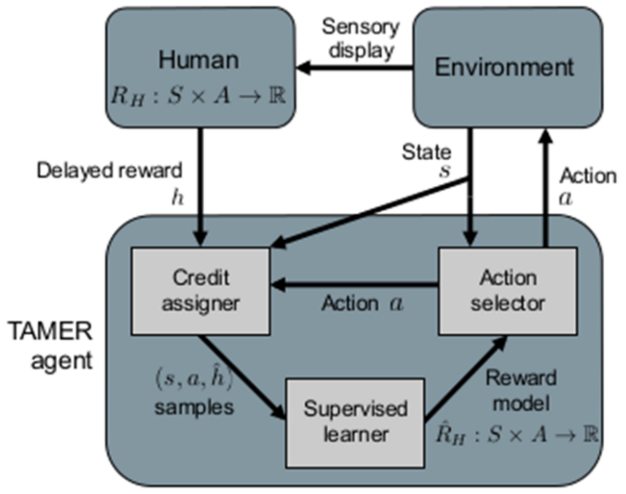


Figure 2.5: The TAMER framework (taken from [54])

One of the seminal works based on *shaping* is the TAMER framework (Training an Agent Manually via Evaluative Reinforcement) [47, 56], which addresses how to use delayed human rewards in RL problems with discrete action domains. There are also, some works which present the combination of human rewards and MDP reward functions applying transfer learning strategies; both rewards were combined first sequentially [49], and then in a simultaneous scheme [51].

The authors of TAMER studied the use of the discount factor used with the rewards in RL [50, 46, 52]. They firstly concluded that high discount (low discount factor) performs better for human reward functions used as MDP reward. However they presented a successful case of learning with discount from human reward [53]. In [55] it is presented the first implementation of TAMER algorithm on a real robot.

One important module of TAMER is the "Credit Assigner" intended for problems of "high" frequency regarding the human response capabilities. The credit assignment is "the problem of assigning *credit* or *blame* for overall outcomes to each of the internal decisions made by a learning machine and which contributed to those outcomes" [37]. In this case the module is for solving a temporal credit assignment problem, because a human trainer is not able to assess the effect of each action at each time step, this produce a delay between the action execution and the human response. The Credit Assigner proposed in TAMER, approaches this problem by associating the feedback not only to the last state-action pair, but to a past window of pairs. Each pair is weighted with the corresponding probability that characterizes the human delay [48]. In Figure 2.5 is a conceptual diagram for the TAMER framewok.

The probability density function of the human response could be extracted from the analysis presented in [38] with quantitative methods applied in psychological studies. Nevertheless, in [89] the parameters of the PDF were calculated and resulted similar to the ones presented with TAMER.

Later on, other authors proposed ACTAMER, an Actor-Critic approach based on TAMER, which addresses RL problems with continuous action domains and using the same kind of feedback [106, 107].

COACH, the here-proposed framework is based on TAMER, but it applies the same kind of human feedback used in the *Advice Operators* paradigm. One of the main features of COACH is a mechanism for adaptively adjusting the amount of human feedback that a given action receives, taking into consideration past feedback.

2.3 The Correspondence Problem

This is one of the most important difficulties in *LfD*. In general this problem is about trying to imitate an observed behavior with a candidate behavior [75]. When there is a problem of correspondence, the objective is not to reproduce the same actions of the observed behavior, because this problem arises from the question of what to imitate: actions, states, events, goals, sequences of sub-goals [74], or because in some cases a direct mapping does not exist between the teacher and learner due to differences in sensing ability, body structure or mechanics [27].

The solution of this problem is in the definition of the mapping between the teacher's and the learner's state-action spaces, for allowing the knowledge transference from one to the other [9]. One action of the demonstrator could be a sequence of actions for the imitator or vice versa. An example of *LfD* with the correspondence issue is the case reported in [70], wherein a biped walking engine is improved; the position of effector is obtained using forward kinematics and the joint's positions, the correction is given in the 3D effector space, then "the resulting feet positions are converted back into a vector of joint command angles using inverse kinematics".

Due to the proposal of this thesis will be based on feedback on the actions domain; it is possible to have correspondence issues in problems that could be approached with the proposal. Therefore the framework to be developed needs to include a module for solving this problem when it would be required.

Chapter 3

An Interactive Framework for Learning Continuous Actions Policies based on Corrective Feedback

The main goal of this chapter is to present COACH (COrrective Advice Communicated by Humans), a new learning framework that allows non-expert humans to advise an agent while it interacts with the environment in continuous action problems. The human feedback is given in the action domain as binary corrective signals (increase/decrease the current action magnitude), and COACH is able to adjust the amount of correction that a given action receives adaptively, taking state-dependent past feedback into consideration. COACH also manages the credit assignment problem that normally arises when actions in continuous time receive delayed corrections. The proposed framework is characterized and validated extensively using four well-known learning problems. The experimental analysis includes comparisons with other interactive learning frameworks, with classical reinforcement learning approaches, and with human teleoperators trying to solve the same learning problems by themselves. In all the reported experiments COACH outperforms the other methods in terms of learning speed and final performance. It is of interest to add that COACH has been applied successfully for addressing a complex real-world learning problem: the dribbling of the ball by humanoid soccer players.

3.1 Introduction

One of the main limitations of the use of Autonomous Learning approaches in real-world problems is the large number of learning trials or roll-outs required to learn complex behaviors. This can make the use of many learning approaches non-viable in problems such as autonomous driving, x-copter flight control, or soccer robotics, in which the execution of learning trials with real robots, in the real-world may have a high cost due to physical limitations. Machine Learning strategies can approach this drawback by leveraging the training processes with the use of human feedback while the agent is learning, i.e., the learning process is assisted by a human teacher in the loop who supervises the agent-environment interaction.

The main goal of this chapter is to present the general scheme and some variations of COACH (COrrective Advice Communicated by Humans), a learning framework for continuous action problems, that uses human corrective feedback. The COACH's structure is based on the shaping approach [48], which allows training an agent incrementally and interactively through positive and negative reinforcement signals. Nevertheless, in COACH the human feedback is provided in the actions domain, as it is in the Advice Operators paradigm [7], indicating to the agent how the magnitude of the action has to be modified (increased or decreased). But the problem of using an off-line and batch supervised learning process is solved by managing the human feedback and the interactive update of the policy (the states to actions mapping) in a similar way as it is done with TAMER [48], a well-known shaping approach.

The proposed interactive learning framework is characterized and validated extensively using four well-known learning problems: (i) *Mountain Car* [95], (ii) *Cart-Pole* [95], (iii) *Ball Dribbling with Humanoid Robots* [64] in the context of robot soccer, and (iv) *Learning to Balance on a Bicycle* [84]. For the first three problems the performances of agents trained using COACH are compared with the performances achieved by agents trained using TAMER [48], ACTAMER [107], and SARSA(λ). In the case of the *Mountain Car problem*, the performance of COACH is compared with the one of standard and interactive learning frameworks using a keyboard interface and a Gesture Recognition interface, in order to analyze how the learning process depends on the interface used. In the *Cart-Pole* problem, the dependence on the dynamics of the environment of the interactive learning frameworks is analyzed. In order to achieve this, the effect of using two different speeds of the simulated environment is analyzed. The *ball-dribbling* problem is used, first, to compare the different learning frameworks, and second, to show how complex behaviors learned using COACH can be applied successfully in the real-world by real robots. Through the reported experiments, the learning processes of autonomous and interactive agents are analyzed and compared with respect to the progress of pure human teleoperators trying to control the same systems. This comparison is carried out using the *Mountain Car, Cart-Pole*, and *Learning to Balance on a Bicycle* problems.

3.2 COACH: COrrective Advice Communicated by Humans

3.2.1 General Aspects

COACH lets the trainer shape the policy of an agent through occasional feedback. The method updates incrementally a policy model based on a supervised learning strategy supported by four main modules: *Policy Supervised Learner*, which updates the policy model taking the human feedback, the executed action, and the associated state into account; *Human Feedback Modeling*, which characterizes the sequence of human advice and determines how much feedback must be added to the executed action; *Human Feedback Supervised Learner* which updates the parameters of the human feedback model; and *Credit Assigner*, which handles the time delay of human feedback. In this chapter the policies are modeled with linear models of basis functions, and with Takagi Sugeno Fuzzy Systems. However, the principles of COACH are extensible to other approximation functions.

Policy Supervised Learner

In this Framework, when the Policy module observes the state vector $\mathbf{s}$, it executes a continuous action $P(\mathbf{s})$ according to the policy model $P : S \rightarrow \mathbb{R}$. (This is a difference in regard to the TAMER modeling, which bases the policy on a human trainer's reinforcement function from the state-action space $H_{TAMER} : S \times A \rightarrow \mathbb{R}$.) Then, the human trainer observes the effect of the action in the environment, and gives advice h, if he/she considers it is necessary to correct it with a relative change of the action magnitude.

The signal h is the binary feedback (1 or -1), which states how the current executed action has to be modified for that state $\mathbf{s}$ (increase or decrease its magnitude, respectively). The state $\mathbf{s}$, the executed action $P(\mathbf{s})$, and the human feedback h are taken by the *Policy Supervised Learner* module for updating the parameters of P (weight vectors or data instances if the model is actually non-parametric). Then, in the next time step, P has a new set of parameters. When the trainer does not provide any feedback signal, h is taken as zero. The rule for updating the policy parameters has to be based on an incremental scheme depending on the type of approximation used. In this work, Stochastic Gradient Descent (SGD) is used for adapting the models' parameters.

The trainer is allowed to give only a binary correction, because COACH works under the assumption that: a person cannot estimate the exact magnitude of an appropriate correction; the human teacher provides only a trend of the modification (e.g. more/less force, velocity, energy, etc.). Supervised learning schemes need a prediction error of the P model (the difference between the desired action and the executed action), but the exact magnitude of the desired action is not available due to the stated assumption. In order to solve this problem, the prediction error has magnitude e that is set as a constant for the whole learning process, and sign given by the human feedback signal h, thus, $error = \mathrm{e} \cdot h$. The learning rate is taken as an external parameter that can be constant or adapted by another module as shown in coming subsections.

Basic Learning Framework based only in the Policy Supervised Learner: In cases where it is not required a fine-tuning for the policy or where the human response delay is very low regarding the operation period, e.g. cases where the frequency is lower than 1 Hz approximately, a basic framework (see Figure 3.1), which does not use the *Human Feedback Modeling* and *Credit Assigner* modules, is defined. This basic framework is described first, in order to facilitate the description of the more general one.

Algorithm 3.1 states how the simplest version of COACH works. First, the error's magnitude e and the learning rate α are constants defined for all the learning process (lines 1-2). The loop between lines 3-13 occurs once per time step. The agent observes the new state $\mathbf{s}$ (line 4), and it computes the action a given by the current policy model $P(\mathbf{s})$ (line 5). Afterwards, $P(\mathbf{s})$ is executed (line 6). After action execution, the feedback signal h of the

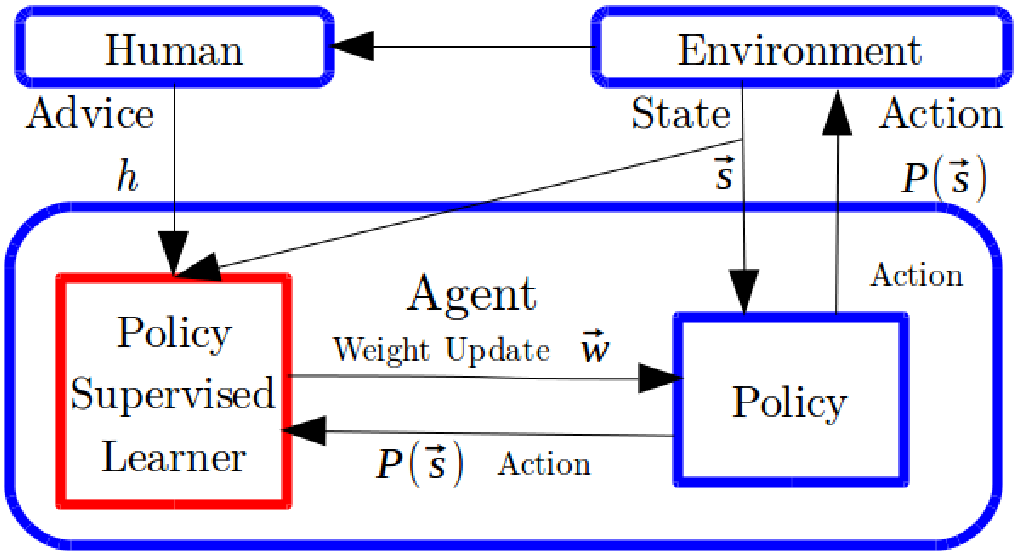


Figure 3.1: Basic structure of COACH

human trainer is read (line 8). If the teacher does not advise any correction, h is set to zero, otherwise it takes the binary value. The prediction error of P, whose magnitude and sign are given by e and h, respectively, is computed (line 10). Then, the model $P(\mathbf{s})$ is updated (line 11), where $updatePolicyModel(\alpha, error, \mathbf{s})$ is a function that reads the meta-parameters of the learning model, the input vector $\mathbf{s}$, and the prediction error associated to the input vector, and updates the model using SGD.

Human Feedback Modelling and Human Feedback Supervised Learner

The trainer intentions, observed in the binary feedback signal, can be considered a source of information that provides not only the sign (direction) of the corrections, but also their magnitude. Hence, in the COACH framework a model of the human feedback $H : S \to \mathbb{R}$ is built, which characterizes the human feedback signal over each region of the state space. The same type of approximator of the policy model $P(\mathbf{s})$ is used for representing the human feedback model $H(\mathbf{s})$. Therefore, two Supervised Learner modules are required in the framework, one for P and one for H (see the block diagram shown in Figure 3.3) at Section 3.2.2). The teacher's intentions captured by the human model are used for computing an adaptive learning rate for the Policy Supervised Learner.

In the proposed modeling, sequences of feedback signals with a constant sign over a specific state $\mathbf{s}_a$ would mean that the trainer is suggesting a large change in the magnitude of the associated action $P(\mathbf{s}_a)$. On the other hand, alternating the sign in the human feedback would mean that the trainer is trying to provide a finer change around a set point. Thus, using the information of H for computing an adaptive learning rate is appropriate for avoiding the dilemma of setting either a too large or too small magnitude of the learning rate for the policy model. As in general learning systems, the size of a learning rate brings different

Algorithm 3.1: Basic Structure of COACH (simple framework).

```
e ← constant   // error magnitude
α ← constant  // learning parameters
while true do
  s ← getStateVec()
  a = P(s)
  TakeAction(P(s))
  wait for next time step
  h ← gethumanCorrectiveAdvice()
  if h=! 0
    error ← h · e
    updatePolicyModel(α, error, s)
  end if
end while
```

advantages. In the COACH framework a large value of the learning rate allows the trainers to carry out large corrections, while a small value lets them perform fine adjustments.

Both P and H models map the same state space, and are based on the same kind of function approximator. Also, their respective *supervised learners* use the human feedback signal for updating the parameters. However, in the H model, assuming the constant error stated for the policy updating is not needed, because in this case the prediction error is known; it is the difference between the desired value h and the prediction $H(\mathbf{s})$.

The adaptive learning rate of $P(\mathbf{s})$ is computed as:

$$\alpha(\mathbf{s}) = |H(\boldsymbol{s})| + bias \tag{3.1}$$

where $bias$ is the default value of the learning rate.

The magnitude of $|H(\boldsymbol{s})|$ is close to 1 when most of the last human feedback signals for a specific state $\mathbf{s}_a$ have the same value (either only 1, or only -1). Contrarily, alternating values of the feedback signal decrease the magnitude of $|H(\boldsymbol{s})|$. Hence, $\alpha(\mathbf{s})$ is set to a large value when feedback signals of constant value are received, and $\alpha(\mathbf{s})$ is set to a small value when feedback signals of alternating value are received.

For demonstrative purposes, Figure 3.2 shows an example that compares the effect of giving human corrections over time, for a specific state $\mathbf{s}_a$, with constant, or adaptive, learning rates. The figure shows the value of the action when the agent visits only $\mathbf{s}_a$ and the human-feedback modifies the associated action. It can be observed that by using an adaptive learning rate, the trainer increased the magnitude of the action faster than when using a constant value; it takes 4 time steps to reach a magnitude of 6, instead of the 7 time steps required when using a constant learning rate. Afterwards, the trainer decided to modify the action amplitude to a negative action very close to zero. The adaptive learning rate allows doing it faster and finer than in the case of using the constant learning rate, using 3 time steps less

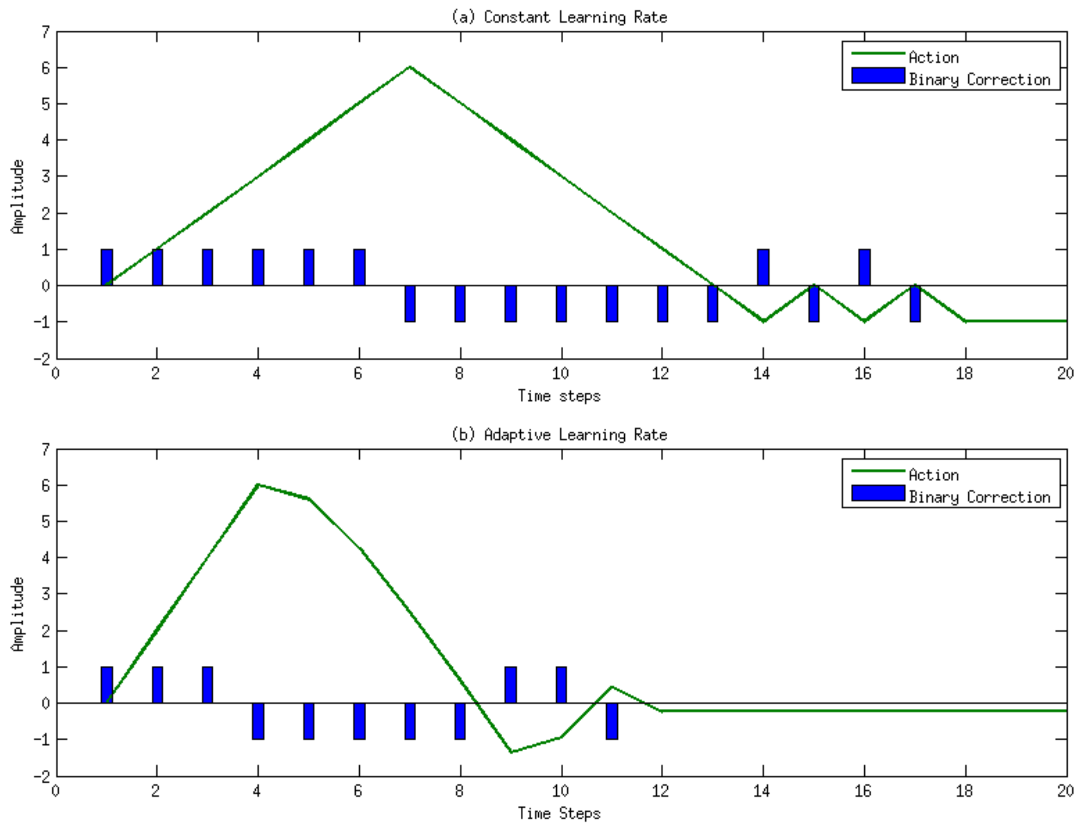


Figure 3.2: Human Feedback Progress at a Specific State $\mathbf{s}_a$, and its impact over the respective action, using: a constant learning rate (a), and an adaptive learning rate (b)

than the case of a constant learning rate.

Credit Assigner

The corrective advice has to be given after the agent executes each action. But in decision-making problems of high frequency, a human trainer is not able to assess the effect of each action and to give advice at each time step; there is a delay between the action execution and the human response that can be tackled as a temporal credit assignment problem. The *Credit Assigner* proposed in TAMER approaches this problem by associating the feedback not only with the last state-action pair, but with past state-action pairs. Each pair is weighted with the corresponding probability that characterizes the human delay, which is called the credit c_t.

The credit assignment proposed in TAMER is specified for linear models of basis functions. In those cases, the computation is simple; however, the idea is general and works for any approximation function.

The credit assigner uses a model of the human response that represents the probability

c_t that a human reaction is associated to an event which has taken place t time steps ago. COACH takes the correction advised at the current time step and associates it with the subset of n past state vectors, which are on the window time frame that supports the probability density function of the delay of the human's response model pdf_{delay}. The credit c_t is computed in (3.2) as the integral of pdf_{delay} from $t_{\text{i}-1}$ to t_{i} ; c_t represents the probability that the human signal given is intended to advise the state-action pair that had taken place t time steps ago. Since older states that have near zero probability are discarded, the sum of all the weights c_t is not exactly 1, but close to it, as is expressed in (3.3).

$$c_t = \int_{t_{\text{i}-1}}^{t_{\text{i}}} pdf_{\text{delay}}(x)\mathrm{d}x \tag{3.2}$$

$$\sum_{\text{i}=1}^{n} c_t \approx 1 \tag{3.3}$$

Thus, each time the human teacher advises a correction, the policy has to be updated n times; for each of the n past state-action vector pairs $[\mathbf{s}_t \quad P(\mathbf{s}_t)]$ the corresponding $error_t$ used for the $t-th$ update is the error weighted with its associated credit c_t:

$$\mathrm{e}rror_t = h \cdot \mathrm{e} \cdot c_t \tag{3.4}$$

The state-action $[\mathbf{s}_n \quad P(\mathbf{s}_n)]$ is the oldest state in the credit assigner's buffer which might be updated with the trainer's advice; c_n is usually very low. The probability density functions used by the credit assigner in this work are the same as those reported in the TAMER descriptions.

3.2.2 General Framework for Problems of high frequency and complexity

For more complex scenarios wherein the frequency of the environment is high considering the time response of humans, and also, wherein the dilemma of setting small/large learning rate for fine/wide corrections is present, the COACH framework requires the *Human Modeling* and *Credit Assigner* modules. The complete structure of COACH with all the modules is depicted in Figure 3.3. The H model is used for supporting the *Policy Supervised Learner* module, which updates the P model. The *Credit Assigner* module takes the states vector and computes credit assignments based on the history of past states.

Since the assumptions and principles of COACH can be applied to several kinds of approximation models, taking into account special considerations according to each specific case, the complete framework is presented here based on policies approximated with linear models of Radial Basis Functions (RBF), for one-dimensional action problems. The main reasons for this selection are: (i) this kind of function approximation is one of the most widely used

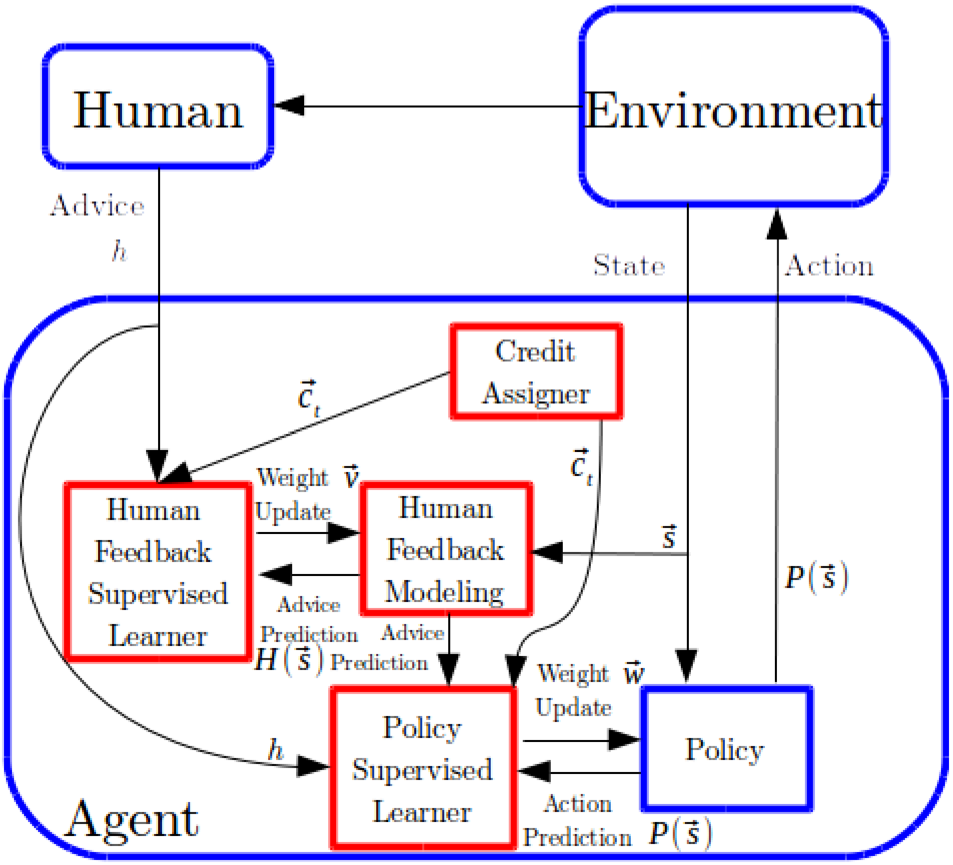


Figure 3.3: General structure of COACH, using an Adaptive Learning Rate for the Policy Update and the Credit Assigner

in RL [95, 22, 57], and (ii) COACH is mainly inspired by TAMER, which was originally proposed based on RBF linear models [48, 47].

Thus, when using RBF functions, the expression for $P(\mathbf{s})$ is the inner product between the weight vector $\mathbf{w}$ and the features vector $\mathbf{f}$ generated with Gaussian Kernels that map from the state space described by the state vector $\mathbf{s}$ onto the features space:

$$P(\mathbf{s}) = \mathbf{w}^\top \cdot \mathbf{f} \tag{3.5}$$

The $\mathbf{w}$ vector is updated using a gradient descent approach for minimizing the squared error as:

$$\Delta w_l = \alpha(\mathbf{s}) \cdot error \cdot \frac{\partial P(\mathbf{s})}{\partial w_l} = \alpha(\mathbf{s}) \cdot error \cdot f_l \tag{3.6}$$

with $error$ the product between h and e, and l the weight's/feature's index. The model of the human feedback H is built using the same features vector $\mathbf{f}$ of P, and a weight vector $\mathbf{v}$ as:

$$H(\mathbf{s}) = \mathbf{v}^\top \cdot \mathbf{f} \tag{3.7}$$

Both P and H models map the same state space, and are based on the same kind of function approximator. Also, their respective supervised learners use the human feedback signal for updating the parameters. However, the H model is updated using the prediction error based on the difference between h and $H(\mathbf{s})$. Therefore, using a gradient descent approach, the weights associated with the H model are updated as:

$$\Delta v_l = \beta \cdot (h - H(\mathbf{s})) \cdot \frac{\partial H(\mathbf{s})}{\partial v_l} = \beta \cdot (h - \mathbf{v}^\top \cdot \mathbf{f}) \cdot f_l \tag{3.8}$$

with β the learning rate and l, the index of the weights and features. Linear models of basis functions allow to have a simple update of the model when COACH is using the *credit assigner* module, instead of updating the model n times for each c_t, the linearity of the model allows accumulating the sums of the weighed feature vectors of the n past state-action pairs into the vector $\mathbf{f}^{cred}$; then the update step is computed just once using that $\mathbf{f}^{cred}$ vector. The credit assigner module presented for COACH is implemented exactly as it is for the TAMER's one. Hence this module computes the new features vector $\mathbf{f}^{cred}$ for replacing the original vector $\mathbf{f}$ in the H and P update process (equations (3.5)-(3.8)). $\mathbf{f}^{cred}$ is the sum of the n past feature vectors $\mathbf{f}_t$ that are weighted with the credit c_t:

$$\mathbf{f}^{cred} = \sum_{t=1}^{n} c_t \cdot \mathbf{f}_t \tag{3.9}$$

Algorithm 3.2 presents the COACH variant for training policies modeled with linear models of basis functions. First, the error magnitude e used for the policy model, and the learning rate β used for updating the human model are defined (lines 1-2). Then, the credit c_t weights are computed by the function $assignCredit(t)$. This yields the pdf_{delay} selected for modeling the human time response and computes (3.2).

The loop between lines 6-23 occurs once per time step. The agent observes the new state $\mathbf{s}$ (line 7), it maps the states into the features space of the basis functions (line 8), and it computes the policy model $P(\mathbf{s})$ (line 9). Afterwards, $P(\mathbf{s})$ is executed (line 10).

After the action execution, the feedback signal h of the human trainer is read (line 12). If the teacher does not advise any correction, no further updating computation is carried out. Otherwise, the models are updated with the received correction (lines 13-22). First, the credit assigner computes the vector f^{cred} taking the n past feature vectors $\mathbf{f}_t$ (lines 14-16). The human feedback model H is updated in lines 17-18, the prediction of the human advice $H(\mathbf{s})$ is executed (line 17), then it is used for updating the model (line 18). Afterwards, the policy's adaptive learning rate is calculated (line 19), and the previously mentioned constant error assumption is computed (line 20). Finally, the policy model is updated in a similar way as that used for updating the human model (line 21).

3.2.3 COACH with Takagi Sugeno Fuzzy Systems

In this section COACH is extended to learn the parameters of a TS-FS (Takagi-Sugeno Fuzzy System) approximation [97, 12]. A TS-FS has a similar structure to that of a linear model of RBF features, but in this case the feature vector is composed of the normalized activation of each rule (r_j, $1 \leq j \leq J$; $J = numberofrules$) of the fuzzy rule base:

$$f_j = \frac{r_j}{\sum_{\mathrm{i}=1}^{J} r_{\mathrm{i}}} \tag{3.10}$$

In this case, each feature or rule is associated with a function $u_j(\mathbf{s})$ instead of a simple weight as in the RBF linear model. The function $u_j(\mathbf{s})$ represents a direct mapping from the input space S to the output (in this case the action), and it is usually a linear combination of the input variables:

$$u_j(\mathbf{s}) = \sum_{k=1}^{K} a_{jk} s_k = \mathbf{a}_j^{\top} \cdot \mathbf{s} \tag{3.11}$$

Then, the complete TS-FS output is given by:

Algorithm 3.2: COACH with RBF linear models.

```
1:  $e \leftarrow constant$  // error magnitude of the policy model
2:  $\beta \leftarrow constant$  // learning rate of the human model
3:  for $1 \leq t \leq n$
4:    $c_t \leftarrow assignCredit(t)$
5:  end for
6:  while true do
7:    $\mathbf{s} \leftarrow getStateVec()$
8:    $\mathbf{f} \leftarrow getFeatures(\mathbf{s})$
9:    $P(\mathbf{s}) \leftarrow \mathbf{w}^T \cdot \mathbf{f}$
10:   $TakeAction(P(\mathbf{s}))$
11:   wait for next time step
12:   $h \leftarrow gethumanCorrectiveAdvice()$
13:   if $h =! 0$
14:     for $1 \leq t \leq n$
15:       $\mathbf{f}^{cred} = \mathbf{f}^{cred} + (c_t \cdot \mathbf{f}_t)$
16:     end for
17:     $H(\mathbf{s}) = \mathbf{v}^T \cdot \mathbf{f}^{cred}$
18:     $\Delta v_l = \beta \cdot \big(h - H(\mathbf{s})\big) \cdot f_l^{cred}; l = 1,..,N_{feat}$
        $v_l = v_l + \Delta v_l$
19:     $\alpha(\mathbf{s}) = | H(\mathbf{s}) | + bias = | \mathbf{v}^T \cdot \mathbf{f}^{cred} | + bias$
20:     $error \leftarrow h \cdot e$
21:     $\Delta w_l = \alpha(\mathbf{s}) \cdot error \cdot f_l^{cred}; l = 1,..,N_{feat}$
        $w_l = w_l + \Delta w_l$
22:   end if
23: end while
```

$$\begin{aligned} P(\mathbf{s}) &= \sum_J u_j(\mathbf{s}) f_j = \mathbf{u(s)}^\top \cdot \mathbf{f} \\ &= \sum_J (\sum_K a_{jk} s_k) f_j \end{aligned} \tag{3.12}$$

For updating the weights matrix **a** of a TS-FS, COACH uses the same stochastic gradient descent strategy which is applied when it learns the parameters of RBF linear models as:

$$\begin{aligned} P(\mathbf{s}) &= \sum_K (s_k \sum_J a_{jk} f_j) \\ &= \sum_K (s_k g_k(\mathbf{s})) \end{aligned} \tag{3.13}$$

$P(\mathbf{s})$ can be seen as a linear combination of the input variables (states), whereas weights g_k are input-dependent parameters. Then the gradient of the action with respect to a particular weight a_{jk} is as:

$$\frac{\partial P(\mathbf{s})}{\partial a_{jk}} = \frac{\partial P(\mathbf{s})}{\partial g_k(\mathbf{s})} \frac{\partial g_k(\mathbf{s})}{\partial a_{jk}} = s_k \cdot f_j \tag{3.14}$$

Then, COACH updates the weights as:

$$\Delta a_{jk} = \alpha(\mathbf{s}) \cdot error \cdot \frac{\partial P(\mathbf{s})}{\partial a_{jk}} = \alpha(\mathbf{s}) \cdot error \cdot s_k \cdot f_j \tag{3.15}$$

Thus, the COACH algorithm using the TS-FS model is similar to the one using RBF functions. The basic difference is in the way of computing and updating the models according to equations (3.10)-(3.15). The most important changes introduced are:

- The statement of the constant α_{max} at the beginning, which is the bound of the adaptive learning rate of $P(\mathbf{s})$. This is because the appropriate range of the learning rate would be different from [0, 1] depending on the problem. This term multiplies the term of line 19 in Algorithm 3.2.
- The function $getFeatures(\mathbf{s})$ maps the state space to the feature space using (3.10) (line 8), but the rule activation r_j is not necessarily obtained from Gaussian kernels in the same way as in the RBF linear model case; rather, it could be computed by any kind of fuzzy model.
- The use of Equations and terms in Equations (3.11)-(3.13) that have to be replaced in line 9, due the TS-FS structure.

The policy weights update given by Equation (3.14) in line 21. In this case the weights are in a matrix instead of a vector, and the updating term includes the factor s_k according to (3.14) and (3.15).

3.2.4 COACH Extension to Multi-Dimensional Action Problems

The generalization of COACH to multi-dimensional action domain problems, i.e. problems in which the teacher could advise corrections over different action spaces, is simple; principally the approximation model needs to be set to multiple outputs. From the point of view of the human teachers, the only modification is that now, in addition to giving binary feedback, they need either to select the action's dimension in which this feedback is given, or to provide a vector of binary signals, in order to communicate the action dimensions simultaneously.

The exact way in which the teacher provides the feedback to the agent depends on the type of Human Computer Interface (HCI) being used. In some cases, the feedback is given in only one dimension at a time, but in some other cases the binary feedback on some different dimensions can be given simultaneously, for instance by pressing different keys of a keyboard at the same time with both hands, or using a joystick, or a Wii Remote interface, etc. For the error assumption, the magnitude e would be a vector with the same dimensionality of the action vector, using different scales for the magnitude of the error in each of the action domains.

3.3 Experimental Validation and Analysis

COACH is characterized and validated in four learning problems: *Mountain Car, Cart-Pole, Ball Dribbling with Humanoid Robots* in the context of robot soccer, and *Learning to Balance on a Bicycle.* This validation includes a comparison in terms of performance with two other interactive frameworks, TAMER, and ACTAMER, with autonomous RL, discrete and continuous SARSA(λ), and with human teleoperators.

Between 10 and 20 people participated in the validation experiments with interactive agents, performing as teachers. The participants of the experiment were people from 20 to 39 years old, half of them students of electrical engineering, the rest had various occupations. At the beginning they watched a video with agents performing the tasks correctly and the learning procedure for each agent; and received the instructions of what to do, i.e. what kind of feedback they had to provide regarding each framework. For each problem to be solved, the users interacted with each learning framework in two stages: practice and teaching. In the practice stage, they interacted with each learning framework in two training runs per problem. The learning results were not recorded. In the teaching stage, they trained the agent twice per problem, and the results were recorded.

The reward functions applied by the RL agents were the cost functions used to compare the performances of all the agents. For all the experiments and types of learning methods,

Table 3.1: Learning Parameters for the experiments.

Algorithm	**Parameters**
Mountain-Car	
SARSA	$\lambda = 0.9, \quad \alpha_s = 0.15, \quad \varepsilon = 0.01 \quad \gamma = 0.98$
COACH	$\mathrm{e} = 0.5, \quad \beta = 0.35$
Cart-Pole	
SARSA	$\lambda = 0.95, \quad \alpha_s = 0.1, \quad \varepsilon = 0.02 \quad \gamma = 0.99$
COACH	$\mathrm{e} = 0.5, \quad \beta = 0.35$
Ball Dribbling 1-D	
SARSA	$\lambda = 0.85, \quad \alpha_s = 0.2, \quad \varepsilon = 0.05 \quad \gamma = 0.997$
COACH	$\mathrm{e} = 30, \quad \beta = 0.35$
Ball Dribbling 3-D	
COACH	$\mathrm{e}_x = 30, \quad \mathrm{e}_y = 20, \quad \mathrm{e}_\theta = 20, \quad \beta = 0.35$
Bike Balance	
SARSA	$\lambda = 0.95, \quad \alpha_s = 0.5, \quad \varepsilon = 0.01 \quad \gamma = 0.99$
COACH	$\mathrm{e}_T = 0.5, \quad \mathrm{e}_\mathrm{d} = 0.8, \quad \beta = 0.35$

the policies were initialized with actions $a = 0$ for the whole state space, making the agent to learn from scratch.

The hyper-parameters of the learning algorithms were obtained using hill-climbing during preliminary experiments for all the problems, except for bike balancing. In that case the parameters were taken from the original work and code reference [84, 83]. For SARSA(λ) the parameters are: the decay factor of the eligibility traces (λ), the learning rate (α_s), the exploration probability (ε), and the discount factor (γ); for COACH are: the error magnitude (e), and the learning rate of the Human Model (β). In table 3.1 are listed the hyper-parameters for all the experiments. The same learning rates of SARSA(λ) were used for TAMER and ACTAMER, since using different magnitudes obtain different human reinforcement functions, but they still map to the same actions. The error magnitude (e) can be tuned in the same order of magnitude of the action range.

The RL learning curves were taken as the baseline for comparison. The learning results of the RL agents are similar and even better than those reported by some of the related works.

After the validations with human users, finally, an ablation study is presented in order to evaluate the contribution of the *Human Feedback Modeling* module.

3.3.1 Mountain Car Problem

In this classical toy problem, a simulated car must get to the top of a hill [95]. The car starts between two steep hills and must go back and forth to gain enough momentum to reach the goal. In order to increase the complexity of the learning task, the continuous 2-dimensional state space was divided uniformly using 100 Gaussian features, a number that is higher than the ones used commonly in the studies reported in the literature. The reward function used

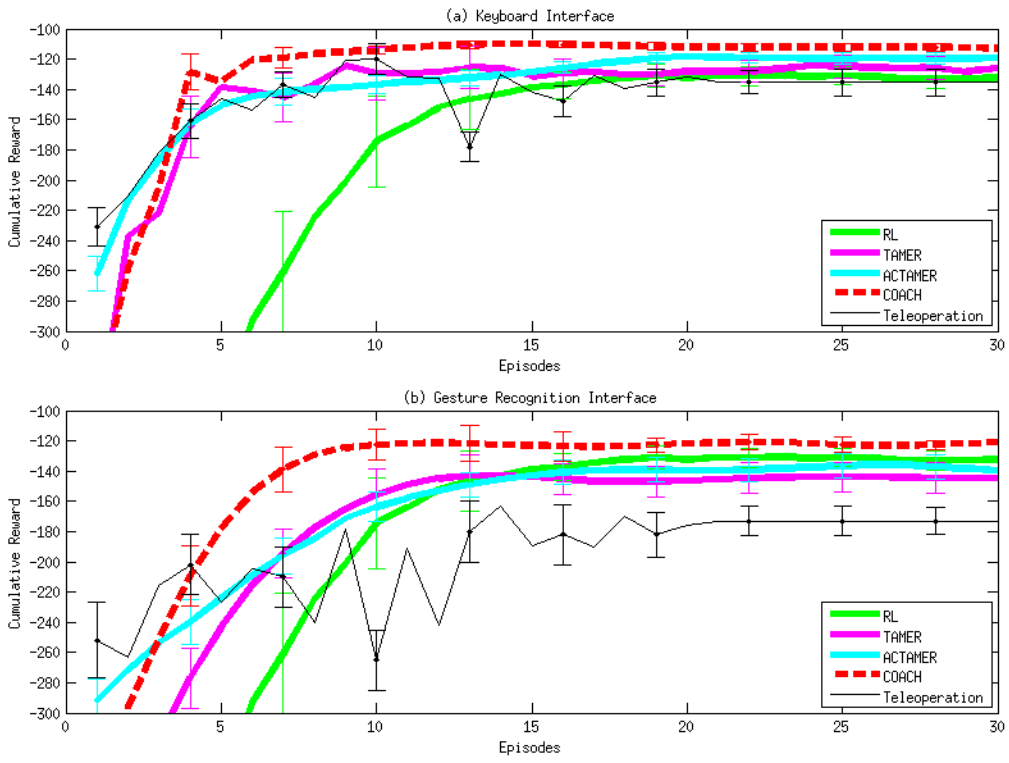


Figure 3.4: Average Cumulative Reward for the Mountain Car Problem. Results using (a) the Keyboard Interface, (b) the Hand Gesture Recognition Interface

was the one reported in [95], which punishes all time steps until the car reaches its goal, arriving at the top of the mountain. All the learning frameworks were tested under the same conditions, and the experiments with the interactive agents were executed using a keyboard interface, and then using a Hand-Gesture Recognition (HGR) interface presented in [26].

The learning curves obtained -average cumulative reward- are shown in Figure 3.4. In the displayed results it is possible to see that, in both kind of experiments, COACH agents outperform all the other agents in terms of convergence speed and performance. In the first case, when using the keyboard interface, all the agents trained with interactive frameworks clearly outperformed the autonomous learner -SARSA- in convergence time and in final performance. The differences between the interactive agents are relatively small. COACH has the fastest learning in the first episodes, and by the third, it outperforms the other agents; then in the next episodes, its improvement is small. At the beginning it is possible to see that the human teleoperation had the best performance. This "good" initial performance could be related to the intuition and previous knowledge of the operators about "what to do for trying to get the car out of the valley". However, the improvement rate of the users' learning is the lowest.

As previously stated, the interactive frameworks have higher performances with the keyboard interface than with the HGR interface. The main reason for this is that although the hand-gesture interface is a more natural way of communicating for human-machine interaction, the keyboard interface was easier for most of the users because it allowed them to alternate the signals given to the agent at a higher speed. This is a problem of low complexity, however the frequencies required for changing the actions magnitudes are high for the human teachers, so the kind of interface used to communicate to the agent the teachers' intentions play an important role, that is observed in the interfaces comparison. Additionally, the small rate of classification error of the gesture recognition system incorporates error in the learning process, whereas the keyboard is free of noise and only would communicate wrong feedback due to human mistakes.

3.3.2 Cart Pole Problem

This well-known problem in RL literature is an episodic task with the goal of keeping a pole balanced on top of a cart. The actions are the forces applied on the cart; the state space has four dimensions defined by the position and velocity of the cart, and the angle and angular velocity of the pole [95]. An episode is finished (a failure occurs) when the pole falls to a given angle regarding the vertical axis, or if the cart exceeds the bounds of the scene. In our modeling, the continuous, 4-dimensional state space is approximated and divided uniformly using 256 Gaussian RBF features as in [107].

All the learning frameworks were tested under the same conditions and with a similar experimental process to that of the mountain-car problem, except that, in this case, three experiment variants were carried out: first, using the keyboard interface as before; second, using the keyboard interface but with a simulation speed five times faster (more complex dynamics for users); and third, using the HGR interface with no acceleration in the simulation.

The learning curves obtained in this problem, trained with both the keyboard and the hand-gesture interfaces, are shown in Figure 3.5. The experiments with interactive agents were finished at episode 150 with a maximum of 5,000 allowed time steps, considered here as the optimum performance. As it is shown in the learning curves, COACH outperform the other autonomous and interactive agents since the beginning with wide difference, regardless the used interface. In the experiments with the keyboard interface and regular frequency of the environment, TAMER and ACTAMER achieved the lowest performances, although these algorithms achieved faster learning than SARSA in the early episodes (by episode 20). The autonomous SARSA agent converged with higher final performance than the ones achieved by TAMER and ACTAMER by episode 500, and in the first 150 episodes, it still had the second best learning. COACH outperformed the other algorithms from the first episode, and it achieved a performance four times higher than that of SARSA; in addition, COACH achieved the fastest convergence among all the agents, since the users only needed about 25 episodes for teaching the policies. The users' learning curve had the third best performance; at the beginning it had the best performance until episode 8, while COACH achieved better operation. Finally, the human teleoperation was outperformed by SARSA in episode 35.

When the frequency was five times faster with the keyboard interface, the learning curves of all the interactive agents were lower with respect to the first experiment, but the biggest difference was in the case of teleoperation, in which the users did not increase their performance through the episodes, i.e. they did not learn to operate the agent. However, in the same scenario, with the same inappropriate conditions for the teleoperation (using the same interface and environment characteristics), the same participants were able to teach the

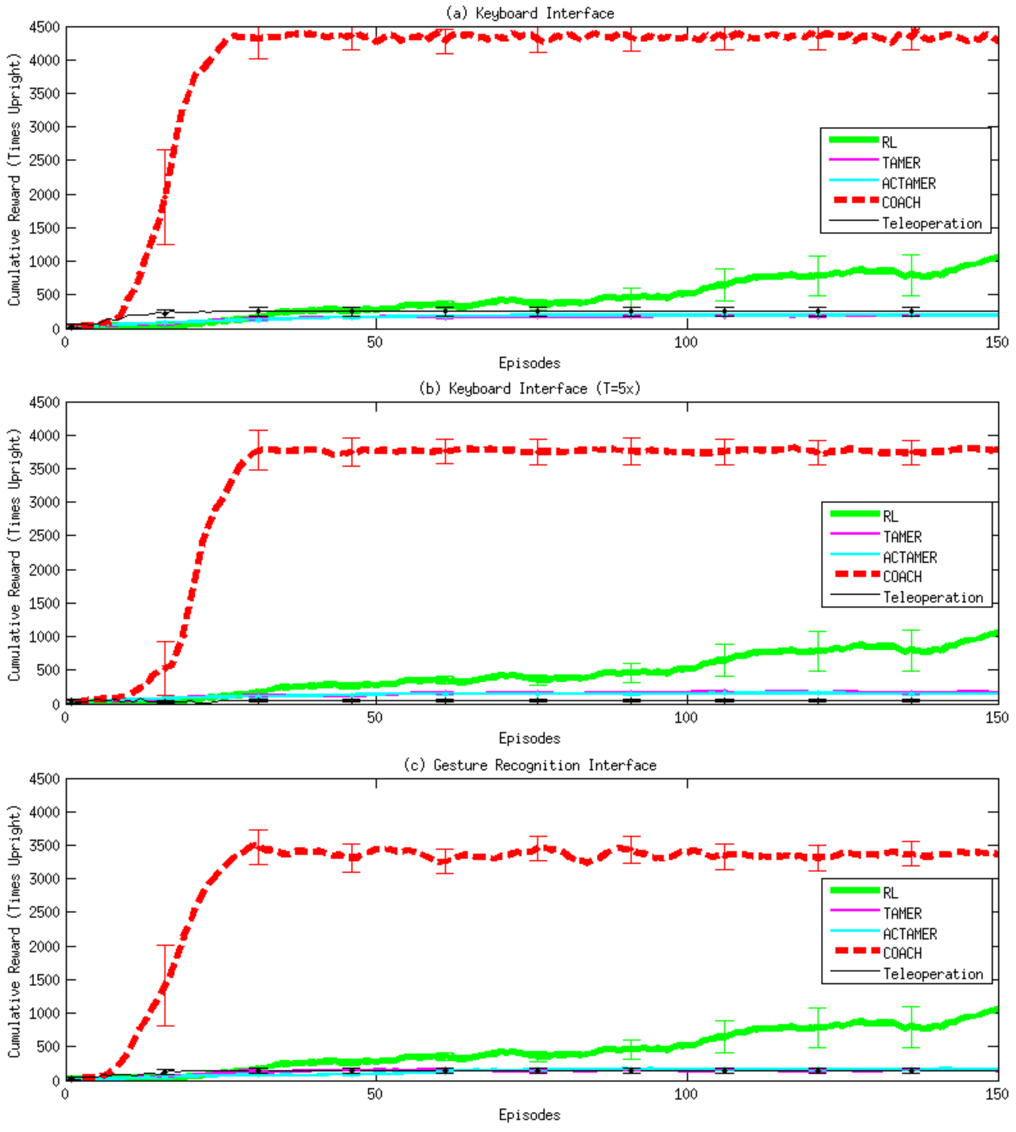


Figure 3.5: Average Cumulative Reward for the Cart-Pole Problem. Results using (a) the Keyboard Interface, (b) the Keyboard Interface with environment 5 times faster, (c) the Hand Gesture Recognition Interface

agent to execute the task using COACH. COACH obtained the best performance from the beginning, and it converged by episode 30.

In the experiments based on the use of the HGR interface, all the interactive agents' results presented similar trends but lower performances than the results obtained with the keyboard interface and regular frequency. In this case, the performance of the users was the best at the beginning, but by episode 6, COACH outperformed all the agents and converged by episode 28. During the whole learning process, SARSA had a slow and constant rate of improvement, and by approximately episode 26, SARSA outperformed the TAMER-type and the teleoperation learning curves, obtaining the second best performance. At the end, the third, fourth, and fifth best performances were respectively for ACTAMER, the users' teleoperation, and TAMER, however they were all very similar.

These results are similar to those obtained with mountain-car; COACH shows the best learning curves for all the experiments and presents more robustness to the occasional flawed feedback. In this balancing task, the corrective advice used by teachers with COACH obtained a wider relative difference of performance with the other agents, regarding the mountain-car problem. This could be because in the previous problem a fine tuning is less necessary since big magnitude of the actions make the car to move faster. However in this problem it is required more both wide changes of the actions, and also slight tuning when the pole is close to the equilibrium point.

3.3.3 Ball-Dribbling with Humanoid Robots

In the context of the RoboCup competition, *ball-dribbling* is a task in which a robot has to walk toward a target as fast as possible while keeping the ball in its possession. Keeping the ball in possession means that the robot keeps the ball close to its own walking feet. In this task the robot observes the environment (the positions of both the ball and the target) and decides the translational and rotational speeds, v_x, v_y and v_θ, respectively. This problem has been tackled using hybrid control strategies; in [64] a hybrid scheme is proposed, in which most of the time v_x, v_y and v_θ are controlled using a TS-FS controller trained off-line for aligning the robot to the ball and the target. But when the robot is already aligned, it switches the computation of v_x to a controller trained with autonomous RL for pushing the ball.

In this section, two experimental procedures for validating COACH with this task are presented. First, the simple hybrid scheme of [64] that learns v_x is trained with different learning agents. Second, a complete TS-FS controller is trained from scratch using COACH, and compared with the results of the first approach. The first experiment is a simple episodic learning task that is also intended to compare the interactive agents' evolution as in the previous problems; the second experiment is intended to evaluate two methodologies for training the dribbling engine based on COACH and to compare their results in real game scenarios.

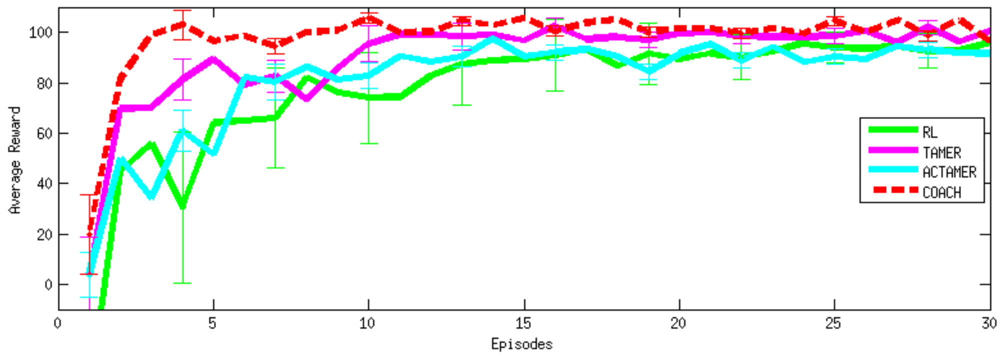


Figure 3.6: Average Cumulative Reward for the 1-D Ball Dribbling Problem

1-D Ball-Dribbling

The hybrid approach proposed in [64] splits the dribbling task into two simpler tasks: alignment to the ball and target, and ball-pushing. The ball-pushing is trained as an episodic task with RL. For the RL of the ball-pushing task, an episode is completed when the robot goes across the complete soccer field with the ball. For dribbling the ball in a straight line the robot estimates the distance to the ball ρ, and decides its forward (axis x) speed v_x. This problem, therefore, has a very small state space but a high level of uncertainty, due to the fact that the motion of the feet is not observed by the decision-making system of the robot.

A modification of the original reward function proposed in [64] is introduced here; it incorporates a parameter that defines a security robot-ball distance ρ_{max} that must not be exceeded:

$$r = \begin{cases} 100 + v_x & \rho \leq \rho_{max}, \\ -100 - (\rho - \rho_{max}) & \rho > \rho_{max} \end{cases} \tag{3.16}$$

This reward function redefines the ball-dribbling task: the goal is to walk as fast as possible, but without exceeding the robot-ball distance ρ_{max}. The robot speed v_x is set between 0 and 100 mm/s. For the algorithms with discretization of the action space (SARSA and TAMER), ten different magnitudes of speed were defined; the state space was divided uniformly into thirty features between 0 and 3 meters, and the ρ_{max} parameter was set to 300 mm.

SARSA(λ) agents along with TAMER, ACTAMER, and COACH agents are trained for being compared as in previous problems using similar setups. The objective function used for evaluating this problem and comparing the agents is the average reward function. In this first experiment COACH is based on RBF model approximation, since that model was used in previous work for this approach to 1D Ball-Dribbling.

The results obtained are shown in Figure 3.6. It can be observed that the three interactive learning algorithms converged faster than the non-interactive one (SARSA), and that COACH achieved the fastest convergence (in three episodes) and the highest average performance. The second best convergence was obtained by TAMER, which reached the final performance in 12 episodes. The COACH's learning curve is the most stable through the evolution of the policy, and presents the smallest changes in the performance from one episode to the next.

3-D Ball-Dribbling

In this case, a simpler approach, that uses only a TS-FS controller for aligning and pushing the ball, is presented. The TS-FS structure is the same as the one proposed in [64] for the subtask of alignment to the ball and target, but in this case its training is done fully online using COACH. In this approach, the teachers advise the three independent action dimensions: the forward speed v_x, the sidewards speed v_y, and the rotation speed v_θ.

The results of the previous three studied problems have shown that COACH is more convenient for these continuous action tasks than the other interactive and autonomous agents, then, this new experiment is only intended to validate the COACH implementation for TS-FS with multi-dimensional actions. Here the training is not set as an episodic task as it was in the previous case. Instead, it is modeled as a continuous task, in which the dribbling target is the opponent's goal. For this problem the teacher advises corrections to the executed actions, but also has the capability of moving the ball inside the field, in order to obtain states that the teacher needs for evaluating and/or training. In this training scheme, teachers interact with the agent and the environment (moving the ball) until they think that the final desired performance was achieved.

When trying to compare this dribbling approach to the 1D approach presented in the previous section, the learning curves cannot be compared directly since (i) there is no direct comparison between learning curves of episodic and continuous tasks, and (ii) the 1D solution requires an exhaustive tuning stage based on an evolutionary algorithm before the ball-pushing task learning.

Only the final performances of both approaches (1D and 3D ball-dribbling) are compared in three scenarios where the initial position of the ball on the field is varied (see Figure 3.7). The performance indices used in the dribbling evaluation are:

- The dribbling time t_{d}, measured from the initial position until the robot is at the target.
- The percentage of cumulated time of faults $\%t_{faults}$, which is the time when the robot is not keeping possession of the ball (in this case the ball-robot distance is greater than 500mm) in relation to the t_{d}.
- The percentage of increased walking distance $\%d_{\mathrm{i}}$. This index measures how efficient the path the robot walks is while it is dribbling the ball:

$$\%d_{\mathrm{i}} = (\frac{d_r}{d_{rb} + d_{bt}} - 1) \times 100\% \tag{3.17}$$

where d_r is the complete distance walked by the robot until it reaches the target, d_{rb} is

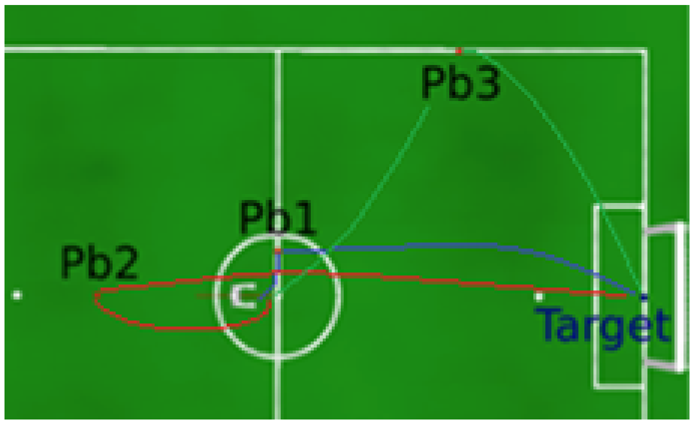


Figure 3.7: Soccer Field with the Robot's Initial Position (Pbi), and the Initial Positions of the Ball for the Evaluation Scenarios: Scenario 1 (blue), Scenario 2 (red), Scenario 3 (green)

the initial distance between the robot and the ball, and d_{bt} is the initial distance between the ball and the target.

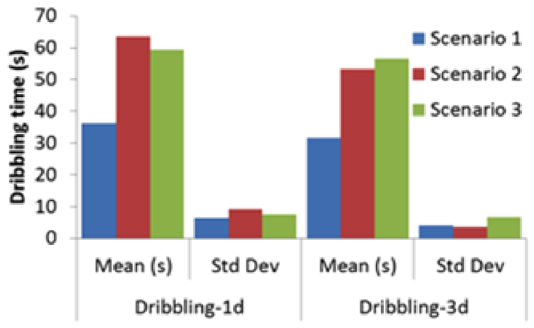
In this problem the interface is a game controller[1] system used for sharing the state of the game with the robots in real competitions, e.g. the game is either in state "set", or "play", or "finished", etc. This interface can be controlled with the mouse or keyboard. Before giving advice, the teacher has to select the action dimension to be corrected (v_x, v_y or v_θ), this is a limitation of the interface, however it does not have much impact because the dynamics of the problem are slow. Three keys are intended for selecting the action dimension, and two to advise the increase or decrease signals.

In Figure 3.8 it is possible to see that in all three scenarios, the variant of COACH algorithm with multi-dimensional action domains and TS-FS models obtains the best performance. Figure 3.8(a) shows a slight time reduction for the 3D model; nevertheless, that index does not lead to inferring hard conclusions by itself (e.g. if the robot always walks very fast, it would finish the scenario faster, but with low ball-possession).

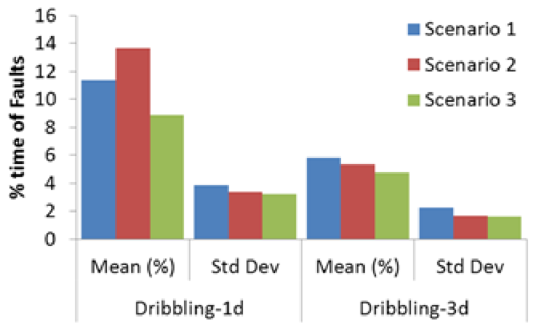
On the other hand, Figure 3.8(b) and Figure 3.8(c) show a greater reduction (around 50%) of the percentage of accumulated time of faults, and the percentage of increased walking distance when the 3D model is used, for the three evaluated scenarios. This indicates that the policies trained allowed dribbling the ball in a more controlled way when the 3D modeling was used, with more ball possession and fewer oscillations over the path between the initial ball position and the target. The reduction of the three indices permits concluding that this simpler approach for training the dribbling engine, based on COACH and TS-FS, obtains policies that dribble with more accuracy, greater speed, and with higher ball possession than the hybrid approach.

The average training time of the new approach was 24.73 minutes, which is much faster than the training period of the old strategy which takes more than two days and is done only in offline training of the TS-FS model using genetic algorithms.

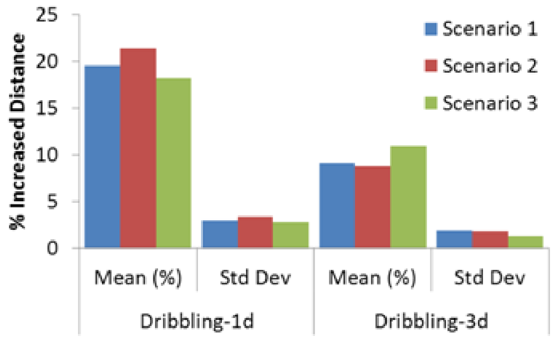
[1]http://spl.robocup.org/downloads/



(a)



(b)



(c)

Figure 3.8: Statistics for each Scenario: (a) ball-dribbling time, (b) percentage of cumulated time of faults, (c) percentage of increased walking distance

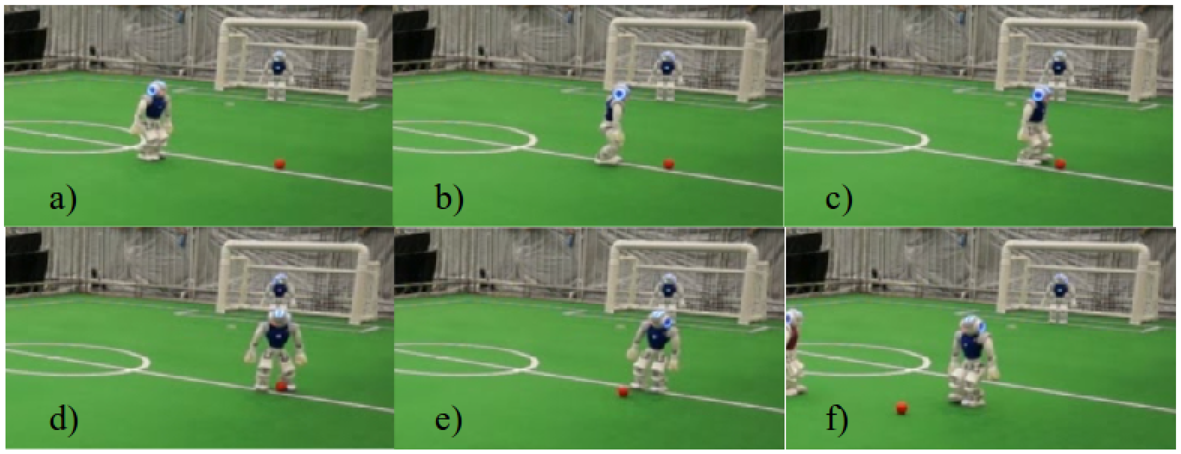


Figure 3.9: Sequence of Ball-Dribbling with humanoid robots in a game at RoboCup

The resulting ball-dribbling controller obtained by the extended COACH algorithm was used by our UChileRT soccer robotics team (http://uchilert.amtc.cl/) in the RoboCup 2015 and 2016 world competitions, allowing the team to achieve fourth place in the SPL (Standard Platform League), where humanoid robots are used . It is worth clarifying that when playing soccer, the dribbling engine has a variant target, computed at a high-level behavior level, in order to avoid obstacle collisions. In Figure 3.9 there are some illustrations of sequences of the ball dribbling in the RoboCup soccer competitions mentioned above.

This complete problem with real robots allows us to validate the COACH variant for TS-FS as policy models in applications with more than one action dimension. The quantitative indices obtained in laboratory along with the competition performances show that the proposed framework can be used to learn complex problems which are tested in competitive instances.

3.3.4 Learning to balance on a bike

This task, proposed in [84], is about learning to balance a simulated bike while it is ridden with constant speed. The agent observes the angle of the handle bar θ, its angular speed $\dot{\theta}$, the angle ω the bicycle is tilted from the vertical, its speed $\dot{\omega}$, and its acceleration $\ddot{\omega}$ (see the variables in Figure 3.10). The actions decided by the agent are: the torque applied to the handle bar T, and the displacement of the center of mass d, perpendicular to the plane of the bicycle. The action T is bounded in the range [-2N, 2N], whereas the action d is in the range [-2cm, 2cm]. Action d has an added uniform random noise between [-2cm, 2cm]. This is an episodic task with initial state vector $[\theta, \dot{\theta}, \omega, \dot{\omega}, \ddot{\omega}] = [0, 0, 0, 0, 0]$. It is considered a terminal state when the bicycle falls, defined by a threshold on angle ω.

For this problem, the experiments are carried out with the keyboard interface. Since in the first three problems presented in this chapter, the comparisons demonstrated that COACH outperforms the TAMER type agents with a large difference, in this problem, the experiments are focused on comparing the learning curve of the users teleoperatiing vs the COACH based learning agents. The SARSA agents are kept as the reference point of comparison,

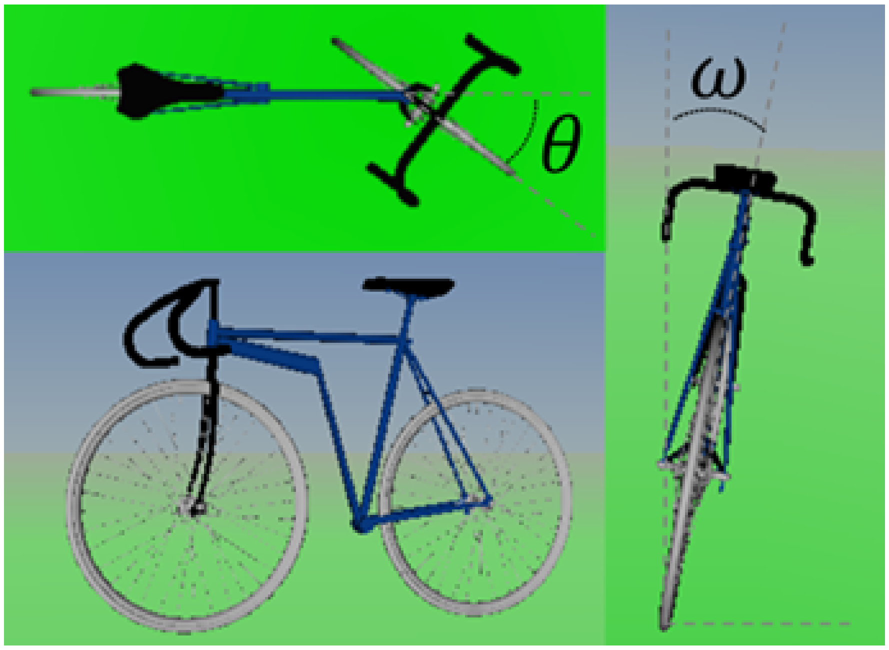


Figure 3.10: 3-D Simulated Bicycle Environment and the Angles Observed by the Agent

since this problem has already been approached and reported in the literature with that agent. Thus, the first controller corresponds to an autonomous RL agent implemented using tabular SARSA(λ) as proposed in [83]; the second one corresponds to an autonomous RL agent implemented using continuous SARSA(λ); the third controller corresponds to an agent trained interactively using COACH; and the last controller corresponds to the case of a human user teleoperating the bike. In this problem COACH learns a model of linear RBF features.

Following [83], the autonomous RL agents use the following parameters: $T \in$ [-2N,0N,2N] and $d \in$ [-2cm,0cm,2cm], which result in 9 possible action pairs. For all the agents the state space is mapped to a features space of 8,575 dimensions (states θ, ω, and $\dot{\omega}$ are divided into seven features for each dimension; and $\dot{\theta}$, and $\ddot{\omega}$ are divided into five features for each dimension). Thus, COACH learns a model of the same number of parameters (a weight per feature), whereas the autonomous RL agents use 77,175 parameters because there are 8,575 parameters for each of the nine actions.

In order to compile statistically significant results, 50 runs of 200 episodes were executed by the autonomous RL agents. In the COACH case, 12 subjects participated as teachers. These subjects performed two runs each with a variable number of training episodes, since each subject stopped providing feedback when he/she considered the best performance had been achieved. Afterwards, the agent continued performing the task without human feedback, then the policy parameters remained constant until episode number 200 was reached.

In this experimental procedure, we define an episode as successful and finished at time step number 1,000. The objective in the learning process is to maximize the time the bicycle is balanced. The same subjects that acted as teachers of the COACH agents also performed the direct teleoperation of the bike; six of them teleoperated the bike before teaching the COACH

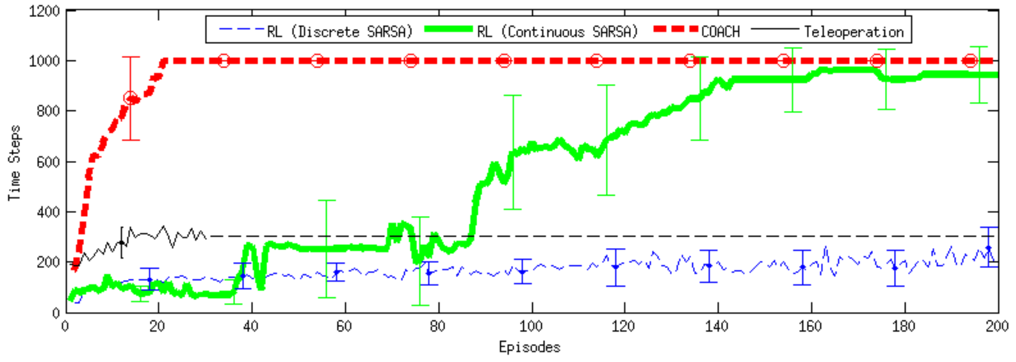


Figure 3.11: Average Learning Curves of the "Learning to Balance on a Bicycle" Problem

agents, and the other six interacted with COACH first. For teleoperating the bike, they first used several consecutive episodes for learning how to do this task. They decided how many episodes were enough. After this training phase, they were evaluated. A keyboard was used for the teleoperation and the COACH learning process. In this last case, the subjects were allowed to advise corrections simultaneously to each action dimension using different keys.

Figure 3.11 shows the learning curve of the three agents, and, additionally, the learning curve of the users who teleoperated the bicycle directly. It can be observed that COACH achieved the highest performance and the fastest convergence through the learning process; it converged to the successful performance at episode number 21. All the COACH agents were able to balance the bike for more than 1,000 time steps. On the other hand, the tabular SARSA(λ) agents had the worst performance; at the end of the learning process they were able to balance the bicycle with an average of 220 time steps. The continuous SARSA(λ) agents performed better than their tabular counterparts, with a convergence, on average, of 941 time steps at episode 140. The continuous SARSA(λ) agents did not attain the successful average performance of 1,000 time steps, as COACH did. However, 94% of the runs achieved the required performance. It can also be observed that COACH not only learns faster than the autonomous agents, but also obtains better policies than those learned by its teachers when they were operating the bicycle directly. This allows us to say that COACH offers a more efficient process of learning than independent approaches of either pure machine learning or pure human learning.

COACH as an interactive machine learning approach that was demonstrated to be more efficient than a conventional LfD scheme, which would require the bicycle teleoperation learning curve of the human teachers as a first step, then, a second step derives a policy from the best demonstrations given by humans, but as Figure 3.11 shows, the derived policy would be sub-optimal with very low performance because of the teachers' low outcomes. As observed with the Cart-pole problem, a learning agent of this balancing problem can benefit of the tuning with the corrective advice, which modifies the policy based only on the vague intuition of the teacher about the trend of the correction.

Some examples of COACH performances can be watched in:

https://youtu.be/T3NMRA0JPX8

3.3.5 Ablation Study

In section 3.2 it is stated that COACH may work with or without the *Human Feedback Modeling* module, nevertheless, in previous subsections all the tests were carried out with the complete framework. In this section, experiments intended to quantify the contribution of the model H are presented, but using a simulated human teacher based on a trained policy P_T in the environment of the Cart-pole problem. The simulated teacher is used in order to reduce the uncertainty introduced by human users due to external factors. This is considered since it is expected small differences in the performance when changing the *Human Feedback Modeling* module.

The simulated teacher uses P_T to provide corrective feedback to the agent in order to teach and replicate the trained policy P_T. The interaction between the simulated teacher and the learning agent differs from regular supervised learning because in this case, the error assumption proposed in COACH is used. Then, the advice of binary corrections is computed as:

$$h = sign(P_T(\mathbf{s}) - P(\mathbf{s})) \tag{3.18}$$

The feedback signal h is given randomly during the 50% of the time steps at the beginning. This frequency is diminished through the time.

The convergence of COACH is evaluated with (i) the complete framework, (ii) the framework without the H model, and (iii) the framework using average of the signals h for computing the prediction H. The third case is a simplification of the original *Human Feedback Modeling* module, in which the prediction H is not state dependent as in the proposed original module.

The learning curves in Figure 3.12 show that COACH without the prediction H takes 20% longer to reach the 95% of the final performance, whereas the COACH averaging the corrections takes twice to achieve it. Figure 3.13 shows the difference of reward obtained in each episode, relative to the original framework. That difference is normalized with the maximum possible reward that can be obtained in the task (5,000). In general the difference is negative, especially with the COACH averaging h.

The COACH agent without adaptive learning rate is faster than the original during the first three episodes in which large changes of the actions are carried out. But, when fine-tuning is required for achieving the stability of the pole, the original COACH speeds up the convergence. The COACH that uses an average of h for predicting the correction is the slowest because it has an adaptive learning rate that is not state dependent. Taking into account the previous corrections without the states where they have been advised may lead to small learning rates in states considered to have a large action modification, or to compute a large learning rate when a fine adjustment is considered.

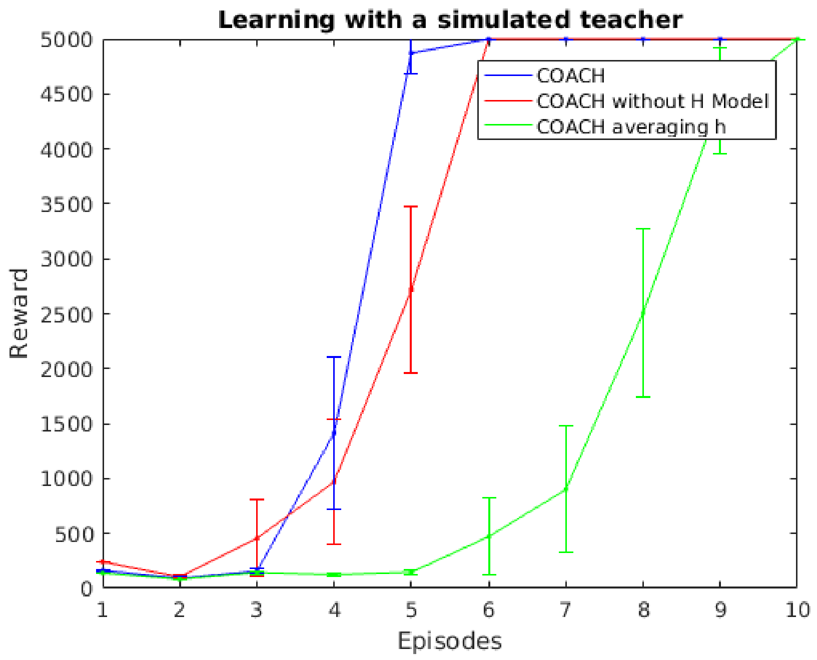


Figure 3.12: Ablation study. Convergence curves of COACH showing the influence of the Human Feedback Modeling module for learning the Cart Pole problem.

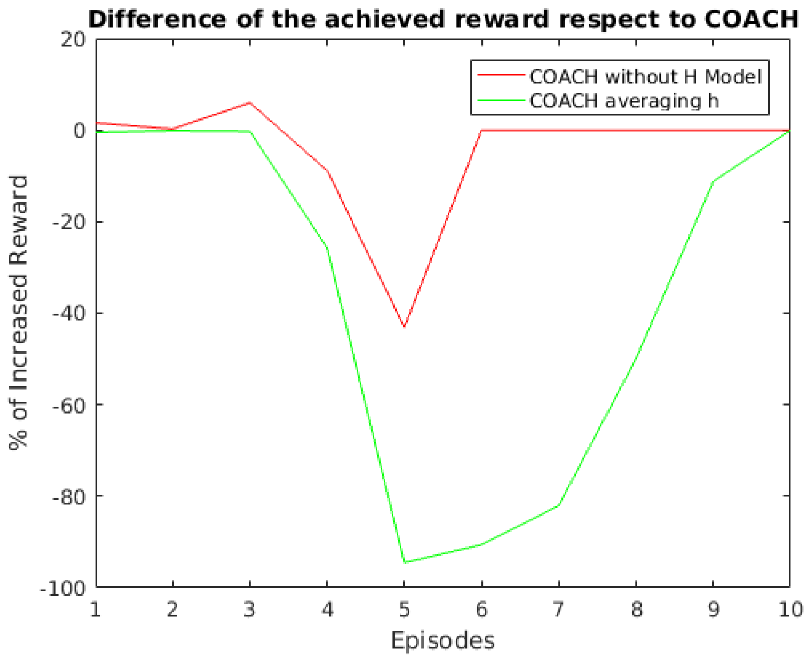


Figure 3.13: Ablation study. Percentage of increased reward regarding the original COACH for learning the Cart Pole problem

3.4 Discussion

The use of Interactive Machine Learning strategies is well known in applications where the users are experts on the task, who then have insights into how the decision-making system has to behave. There are several schemes that are appropriate for learning different levels of tasks and actions, but most of them operate on the assumption that demonstrations or feedback provided by teachers are of good quality. Some studies that deal with the fact that information coming from humans could be flawed, improve the derived policies using a complementary learning strategy such as RL.

The COACH framework is proposed to be used for learning tasks in which the users are not necessarily experts, but who can infer where the actions of the agents have to be modified in order to improve the executed policy by observing the agent-environment interaction. These inferred trends are applied as advice for corrections over the policy by COACH, and, depending on the problem, their effects over the updated policy can be observed immediately. However, COACH is limited for tasks wherein the actions' effects are not intuitive to the teacher, because they would not be able to suggest corrections.

In this chapter and during the complete thesis, it is proposed that COACH work with binary feedback, however, the algorithm can also work with continuous values of correction without any problem. In this work we keep on working only with binary corrections, since it reduces the variables and analysis for the experiments. The use of continuous corrections would give more freedom to the human teachers, although with the disadvantage of requiring interfaces more complex than a keyboard (e.g. joysticks, or motion capture), and probably demanding some time for the user to get used to the sensitivity of the interface.

The experiments performed in this chapter led to some interesting observations and conclusions about the properties of interactive learning of policies for continuous tasks, specifically when the human feedback is vague, as is the case with TAMER algorithms and COACH. TAMER is an algorithm that has a wider range of applications because it can be used to solve tasks of either discrete or continuous actions (discretizing the actions). However, the results of the experiments which involved the mountain-car, the cart-pole, and the one-dimensional ball-dribbling problems showed that it is simpler for the users to train an agent by providing corrections over the actions domain than it is in the evaluative domain. COACH allows users to shape the policies towards their beliefs of better decisions in a more controlled way than TAMER, but still using a vague signal of correction. This is the reason behind the higher improvement rate with COACH at the beginning of the learning processes with regard to the evaluative feedback based agents.

The human feedback model is a powerful element that obtains more implicit information from the pieces of advice sequences; this extracted information is applied to set a wide or subtle updating depending on the previous history. This module empowers the quick progression of the policies at the beginning of the learning process, but also the smooth tuning at the end, which is more stable than the behavior obtained by agents trained with other frameworks that present more oscillation during the whole evolution. The performed ablation study showed that this module contributes to speed up the convergence around 20% in the evaluated case. Also, that study demonstrated that is necessary the state-dependent computation of the prediction H, in order to use correctly the information of the history of the corrections advised by the teacher.

Some specific conclusions can be drawn from the comparison of interactive learning agents, autonomous agents, and human teleoperation of the same processes. It is clear that for most of the cases the interactive agents learned better than the autonomous ones, at least during the first episodes, which is important for applications that use physical platforms that are constrained to a few learning episodes. Moreover, in most of the cases the interactive agents outperformed the pure human learning; only some cases in the cart-pole problem were exceptions. This conclusion becomes even stronger if the users' learning is compared only with COACH. In this case, it is possible to say that users learn more slowly and worse than the agents they train, with large differences in learning time and quality of the final policies. The analysis of the learning curves show that COACH agents learn better policies in less time than with approaches of pure autonomous or human learning. It means that in cases of non-expert teachers, COACH would be better and more efficient than classic LfD schemes that require the slower and poorer human learning processes, followed by the demonstration gathering, and the policy derivation stages. This interesting relation between learner and teacher is also seen in sports, wherein the coach or trainer gives advice to sports players who are actually better than their advisors.

The experiments carried out with the mountain-car and the cart-pole problems showed that COACH is a strategy of interactive learning that is more robust to mistaken feedback due to noisy and occasional flawed corrections, which are always present in humans. When the interfaces and environments were modified in order to increase the possibility of flawed feedback, the learning curves of COACH were the most stable and had the smallest decrease with respect to the original experiment.

In the experiments with the ball dribbling by humanoid robots problem, COACH also outperforms the learning capacities of the autonomous and the TAMER agents. Secondly, COACH showed good results for learning a task with more than one action dimension, and COACH principles can be successfully applied to different model approximations, like those in this case, using a Takagi-Sugeno Fuzzy System. The third remark is a comparison of two strategies for developing this decision-making system: using a hybrid solution with interactive and evolutionary learning, and using a simple scheme that uses only COACH. The hybrid case has a smaller state space for the part solved based on COACH, and it is easier for the users to train the interactive stage when compared to the second case that used only COACH, and that includes all the state variables in the search space, and represents a more complex scenario for the interaction. However, results show that despite the fact that the second strategy was more complex for users during training, at the end, its use was more efficient and simple in terms of time, computational burden, and the human effort involved in the complete process.

Moreover, based on the results obtained, COACH was used for training the dribbling engine used by the UChileRT team in the soccer competitions during RoboCup 2015 and 2016. In 2015, COACH was used to train an agent from scratch. Since then, this strategy has been used just for tuning the dribbling engine anytime that is required, i.e. when changes of the environment dynamics occur, for example, a change in the carpet of the playing field, or a change of the ball (e.g. a new ball with different dynamics was used in RoboCup 2016). Moreover, a tuning is required anytime our game strategy is adjusted, for instance, when we decide to modify the security robot-ball distance definition, according to external criteria.

With the presented method and the obtained results of this chapter, the first and second objectives of this thesis were attained. The proposed learning framework based on corrective advice for one-dimensional action problems demonstrated to be faster and to obtain better policies than other interactive methods and RL methods. Additionally, the ablation study in the last part, demonstrated that the use of past information captured with the Human Feedback Modeling module has a positive impact in the learning process. The third objective which is about applying the learning framework to multi-dimensional action problems, was just partially approached in this chapter, since the problems with more than one action (Ball dribbling and bike balancing) do not have problems of correspondence. That objective is fully covered in Chapter 5.

To continue this work, we are interested in exploring and evaluating how interactive learning is beneficial when more sources of feedback are used in the learning process. For instance, reinforcement signals coming from humans, or encoded MDP reward functions are sources of information that would leverage the learning progression; the more challenging case would be to use all those sources of feedback simultaneously in the same learning framework. In the following chapters, the use of encoded rewards is explored combining Policy Search and COACH.

Chapter 4

Learning in MDPs with Human Corrective Feedback and Reinforcement Learning

Reinforcement Learning agents can be supported by users' feedback in approaches that include human teachers in the learning loop, in order to guide the learning process. In this chapter we propose two hybrid strategies of Policy Search Reinforcement Learning and Interactive Machine Learning that benefit from both sources of information, the cost function and the human corrective feedback, for accelerating the convergence and improving the final performance of the learning process. Experiments with simulated and real systems of balancing tasks and a 3 DoF robot arm validate the advantages of learning using the proposed learning strategies: (i) they speed up the convergence of the learning process between 3 and 30 times, saving considerable time during the agent adaptation, and (ii) they allow including non-expert feedback because they have low sensibility to erroneous human pieces of advice.

4.1 Introduction

An important issue of RL methods is the relative long training time of systems/controllers to be used in complex/dynamic environments, which can be a limitation for their application in robots interacting in the real-world. This shortcoming can be addressed with the support of human users who participate/collaborate in the learning process. Thus, LfD and RL can be sequentially combined for learning skills autonomously from an initial policy that could be obtained by human teachers' demonstrations [11, 60].

There are some works that combine RL with human reinforcements [104, 99, 51]. This combination takes advantage of the user's knowledge for speeding up the learning process while keeping the convergence properties of RL algorithms.

Thus, combining RL with interactive machine learning strategies might be a synergic relationship, in which the learning processes are sped up because RL benefits from the human knowledge of the task, but also provides more stability and robustness to the interactive framework that is susceptible to the inherent occasional erroneous feedback associated to human teachers (human are nor perfect and prone to fail in repetitive tasks).

In this context, we postulate that the use of RL with human feedback will allow addressing important requirements of robotics applications. Since Policy Search RL seems to be more appropriate for facing the challenges of robot RL than value based RL [32], this thesis proposes the use of learning methods based on Policy Search techniques that additionally make use of available human knowledge for reducing the learning time, which is one of the main constraints of classical robot learning applications. Corrective feedback advised by human teachers is used in the introduced approach instead of human reinforcements like in the aforementioned hybrid learning systems. In the proposed approach, human knowledge is provided to the PS learning agents with corrective advice using COACH.

4.2 Episode based Exploration and Evaluation Policy Search Background

PS is a branch of RL where parametrized policies are learnt directly in the parameter space, based on the cost given by the reward function. Thus, PS methods do not learn a value function as most of RL algorithms do. For this reason, PS has advantages respect to value function based RL in robotic applications, because computing the value function requires data from the complete state-action space. Additionally, the use of parametrized policies reduces the search space, which is important in cases of physical limitation as usually is the case of learning with robots [32].

Moreover, in robotic applications PS is a better choice regarding value-based methods due to the properties of scalability and stability of the convergence [57], because a small change of the policy may lead to a big change of the value function, that can again produce a big modification of the policy. The previous instability can be convenient for finding global optima after long time of learning in simulated environments, but with real robots it is desired a smooth and fast convergence.

The general structure of a PS method is presented in Algorithm 4.1, which includes three main steps: exploration, evaluation, and updating. The exploration step creates samples of the current policy for executing each roll-out or episode. This process can be step-based or episode-based, depending on whether the exploration elements are changing through the time steps or are constant during each episode, respectively. In the evaluation step, the quality of the executed roll-outs is examined, and the strategies can also be step-based or episode-based, depending on whether the assessment is over every single action or over the parameter vector. The update step uses the evaluation of the roll-outs to compute the new parameters of the policy. This update can be based on policy gradients, expectation-maximization, information theoretic, or stochastic optimization approaches. During the last years, several PS algorithms have been proposed and evaluated using different strategies in each of the three steps. Nevertheless, the most suitable method to be used depends on the specific application being addressed. In this work we use episode-based methods with stochastic optimization

Algorithm 4.1: Model Free Policy Search.

```
1:  Repeat
2:      Explore: Execute M roll-outs using policy $\pi_k$
3:      Evaluate: Obtain outcomes of trajectories or actions
4:      Update: Compute $\pi_{k+1}$ given the roll-outs and evaluations
5:  until Policy converges $\pi_{k+1} \approx \pi_k$
```

Algorithm 4.2: Policy Search with episode-based Policy evaluation.

```
1:  repeat
2:      Explore: sample parameter vector
                 $\theta^{[m]} \sim \pi_k(\theta), \ \ m = 1 \dots M$
3:      Evaluate: cost of each roll-out $R^{[m]} = \sum_{t=0}^{T} r_t^{[m]}$
4:      Update: Compute new policy parameters using
                 $\{\theta^{[m]}, R^{[m]}\}$
5:  until Policy converges $\pi_{k+1} \approx \pi_k$
```

update strategies that can be considered black-box optimizers, due to the good combination of simplicity and good learning convergence [91, 92, 39]. Its simplicity makes easier the combination of PS with other learning approaches. In Algorithm 4.2, the episode-based model free PS scheme used in this work is presented. In the exploration step, a set of noisy samples of the current policy parameter vector is generated. In the evaluation step, a global cost R of each *m-th* roll-out corresponding to the *m-th* sample of the policy parameter vector is measured. In the third step, the policy is updated using the exploration samples and their respective evaluations. In the next paragraphs three episode-based black-box methods used for policy improvement are described:

4.2.1 Cross-Entropy Method

The Cross-Entropy Method (CEM) for policy search was proposed in [68], and more recently has been used by [23, 90] for learning problems of discrete and continuous actions. Using this approach, the method creates M samples of the current policy for the exploration step, according to a normal distribution as:

$$\theta^{[m]} \sim \mathcal{N}(\theta_k, \Sigma_k), m = 1 \dots M \tag{4.1}$$

where θ_k is the policy vector at the *k-th* iteration and Σ_k the covariance matrix. The cost function should be minimized, and to return a scalar that represents the performance index. The update step is executed first by sorting the samples in ascending order with respect to the cost function $R^{[m]}$. Then, the normal distribution is updated as:

$$\theta_{k+1} = \sum_{m=1}^{M_\mathrm{e}} \frac{1}{M_\mathrm{e}} \theta^{[m]} \tag{4.2}$$

$$\Sigma_{k+1} = \sum_{m=1}^{M_\mathrm{e}} \frac{1}{M_\mathrm{e}} (\theta^{[m]} - \theta_{k+1})(\theta^{[m]} - \theta_{k+1})^T \tag{4.3}$$

taking into account only the first M_e "elite" samples with the highest performances, i.e. $M_\mathrm{e} < M$.

4.2.2 Episode-Based PI^2

The Path Integral Policy Improvement (PI^2) is a PS method formulated from the principles of stochastic optimal control, and is based on probability weighted average for updating the policy parameters, specially for learning in trajectory control problems [101]. In [90] PI^2 with covariance matrix adaptation (PI^2-CMA) is presented, which combines the standard PI^2 with a computation borrowed from the CEM to automatically adapt the exploration term. An episode-based version of PI^2 that can be classified as Black-Box optimization called PI^{BB} is presented in [91], obtaining interesting results with respect to the original version and other variants.

This method executes the exploration step like the CEM in (4.1), however, the covariance Σ is not updated by the method itself, but rather the size of this exploration term has to be externally controlled. The evaluation for obtaining $R^{[m]}$ has the same restrictions as for CEM, but before the update, the evaluation set is normalized between 0 (the best roll-out) and 1 (the worst). Then, the normalized costs are mapped to a probability distribution $P^{[m]}$ as:

$$P^{[m]} = \frac{\mathrm{e}^{-\frac{1}{\lambda} R^{[m]}}}{\sum_{n=1}^{M} \mathrm{e}^{-\frac{1}{\lambda} R^{[n]}}} \tag{4.4}$$

With λ the temperature parameter. Finally, the update step is carried out with the probability-weighted average using the parameter vector of every roll-out and its respective probability $P^{[m]}$:

$$\theta_{k+1} = \sum_{m=1}^{M} P^{[m]} \theta^{[m]}. \tag{4.5}$$

4.2.3 Episode-Based PI^2 with Covariance Matrix Adaptation

In the present work, we propose to extend the PI^2-CMA presented in [90] to the PI^{BB}. This extension is intended to incorporate an automatic adaptation of the exploration term in PI^{BB}, with a similar strategy as the one used in (4.3) for CEM. This alternative strategy can

Table 4.1: PS comparison

Algorithm	**Covariance Adaptation**	**Exploration**	**Evaluation**	**Probability for weighted averaging**
CEM	✓	episode	trajectory cost	uniform with the elite samples
PI2-CMA	✓	time-step	cost to go	normalized exponentiation
PIBB	✗	episode	trajectory cost	normalized exponentiation
PIBB-CMA	✓	episode	trajectory cost	normalized exponentiation
PI2	✗	time-step	cost to go	normalized exponentiation
COACH+PS	✓	time-step/ episode	trajectory cost	normalized exponentiation

be also seen as a combination of PI^{BB} and CEM. This extension uses the same algorithm employed by PI^{BB}, but, it updates Σ according to

$$\Sigma_{k+1} = \sum_{m=1}^{M} P^{[m]}(\theta^{[m]} - \theta_{k+1})(\theta^{[m]} - \theta_{k+1})^T \tag{4.6}$$

instead of (4.3), based on the probability computed using (4.4). We refer to this approach as PI^{BB}-CMA.

The three presented PS algorithms along with the original PI^2 and the PI^2-CMA have slight differences that are summarized in Table 4.1. The "Covariance Adaptation” field indicates whether the algorithms have covariance adaptation for the exploration distribution or not; the field “Exploration” shows the strategy for disturbing the parameters vector for exploration which can be episode based, or time-step based; the column of “Evaluation” shows the kind of evaluation that the algorithm has, for episode based evaluations it is used a “trajectory cost” which is a scalar cost index, whereas for time step based evaluations the “cost to go” is computed for every time step; all the listed algorithms update the policy with probability weighted averaging, so the corresponding field indicates the technique the algorithm uses to compute the probability for updating the policy, for instance, CEM takes the best M_e roll-outs with associated uniform probability $1/M_e$ used in (4.2) and (4.3), the rest algorithms use normalized exponentiation of the cost computed with (4.4).

Additionally, the proposed methods of this work are included in this table, but the details are presented in the next section.

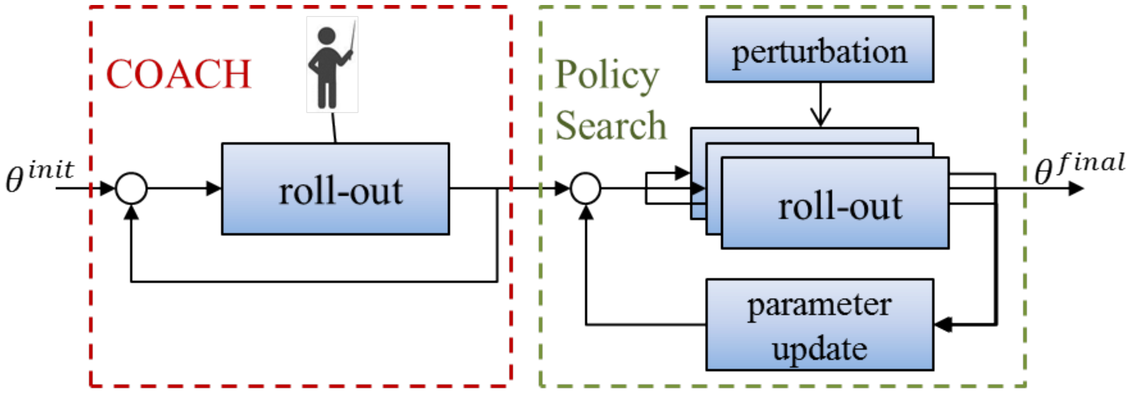


Figure 4.1: Learning Sequentially with COACH+PS.

4.3 Policy Search Guided with Human Corrective Feedback

Policy Search RL algorithms, as any autonomous RL algorithm, take only the feedback given by the cost or reward function and along with the exploration strategy, the search of the solution is performed. In this work it is proposed to combine episode-based PS with human guidance, where the human teacher is able to correct the policy every time step, whereas the PS only updates the policy model after each iteration of M trials, based on the global performance measurement of every roll-out. This combination of feedback sources is presented in two different approaches, where the learning agent is updated by the users with the COACH framework and the PS algorithm in either a sequential or a simultaneous fashion as explained below.

4.3.1 Learning Sequentially with COACH and Policy Search

In this scheme called SeqCOACH+PS, we propose to have two independent learning phases, as depicted in Figure 4.1. At the beginning, the human teacher interacts with COACH providing corrective feedback to the agent during the task execution. The learning could be done from scratch or the initial policy parameters can be set from a policy that needs to be refined. The interactive learning process is executed during several episodes until the user considers that he/she cannot improve the quality of the policy further. Thereafter, the resulting policy is taken as the initial set of parameters for the normal PS process. It has the objective of performing a refinement of the learned parameters in order to achieve an optimal point while locally exploiting the parameters space. The learning process finishes with the same criteria of convergence used when learning only with PS.

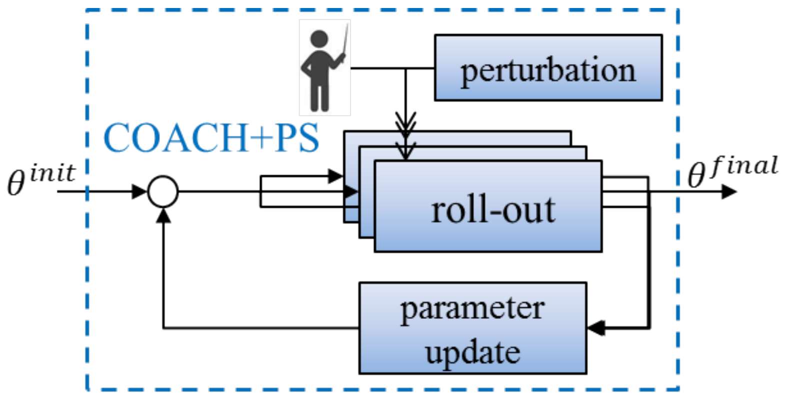


Figure 4.2: Learning Simultaneously with COACH+PS.

4.3.2 Learning Simultaneously with COACH and Policy Search

This second approach named SimuCOACH+PS uses simultaneously both sources of information, human feedback and parameter update of PS, for training an agent as shown in Figure 4.2. Since the learning progress with COACH is completely based on the human teacher criteria, the convergence is prone to be unstable when the users provide mistaken feedback. SeqCOACH+PS also suffers this sensitivity to human errors in its first stage of human advice.

We propose to have the PS algorithm at a supervisory level of COACH, in which the cost function determines whether the policy trained by the human teacher is evolving properly or not. Previously we stated that this work is based on episode-based PS, however the proposed simultaneous PS and COACH learning strategy can be seen as a PS algorithm in which the evaluation is episodic, but the exploration is step-based and completely given by the corrections of the human teacher, wherein the changes that COACH obtains over the policy vector are taken as exploration noise by the PS algorithm.

For this hybrid learning scheme, during each roll-out, a regular COACH process of interactive training is carried out. From the PS point of view, the evaluation and update stages are kept exactly the same as defined by episode-based PS in Algorithm 4.2 at lines 3 and 4 with equations (4.2)-(4.5), depending on the used algorithm.

The exploration stage is significantly different from the strategy given by (4.1), because the samples are only created from the distribution $\mathcal{N}(\theta, \Sigma)$ when it is detected that the user is not providing corrective advice anymore. However, during roll-outs advised via COACH, the users advise incrementally the agent, so little by little the policy is improved with the knowledge the users have about the previous policy. Then, creating independent samples from the current policy distribution for every roll-out can make that some samples obtain behaviors completely contrary to what has been obtained by the human teacher during the immediately previous executed roll-out. This situation tends to confuse the users, they can interpret this situation as “the agent is not interacting properly or is rejecting the results of the advice after each episode”. This situation diminishes the user interaction with the agent. Therefore, the system would not benefit from the human knowledge.

Algorithm 4.3: Policy Search with Simultaneous Human Guidance.

```
1:  repeat
2:      Explore: first roll-out with the current policy $\pi_k$: $\theta^{[1]} = \theta_k$
3:          for $m = 1 \dots M$
4:              $[\theta'^{[m]}] \leftarrow RunRollOut_{COACH}(\theta^{[m]})$
5:              if  $\theta'^{[m]} \neq \theta^{[m]}$
6:                  $\theta^{[m+1]} = \theta'^{[m]}$
7:              else
8                   $\theta^{[m+1]} \sim N(\theta_k, \Sigma_k)$
9:              end
10:             $\theta^{[m]} = \theta'^{[m]}$
11:         end
12:     Evaluate: cost of each roll-out $R^{[m]} = \sum_{t=0}^{T} r_t^{[m]}$
13:     Update: Compute new policy parameters using
                    $\{\theta^{[m]}, R^{[m]}\}$
14: until Policy converges $\pi_{k+1} \approx \pi_k$
```

To avoid this problem, in this hybrid learning strategy, the resulting policy of a roll-out is set as the initial policy for the next one. There are two different parameter vectors for each roll-out, the vector $\theta^{[m]}$ which is the initial parameter set at the $m-th$ roll-out advised with COACH, and the vector $\theta'^{[m]}$ that is the parameters set returned at the end of the roll-out. The difference of these two vectors is associated to the corrections advised by the teacher.

This sequential exploration allows the teachers to have insights of the operation of the policy, and notions of how to keep improving it through the next roll-out executions. After M trials, the PS updates the policy. In cases where the teacher is continuously improving the task execution, the parameters computed for θ_{k+1} have to be similar to the ones of the most recent task executions. According to the cost function, when the human feedback is harming the policy, the update stage should result in a set of parameters similar to the ones that controlled the agent, before the erroneous human feedback.

Algorithm 4.3 shows that for every $k-th$ iteration, the first roll-out is executed with the actual parameters vector of the current policy (line 2). The execution of every roll-out is given by $RunRollOut_{COACH}(\theta^{[m]})$ that has the policy vector as input argument, and returns the corrected policy (line 4). When the initial vector is the same returned at the end of the trial, it means that the teacher did not provide advice, consequently, the exploration can be carried out by the PS algorithm with the regular strategy (line 8). Regardless whether the roll-out is advised or not, the policy vector $\theta'^{[m]}$ returned after each execution is set in the group of samples (line 10) for the update stage.

4.4 Experiments and Results

The proposed hybrid learning approaches are tested using simulated and real problems. The proposed sequential and simultaneous approaches are validated and compared to standard Policy Search methods. First, we present an experiment where a previously trained policy is used to simulate a human teacher that interacts with the learning methods in order to train an agent to perform a task. This experiment with simulated teachers is carried out for evaluating the convergence of the learning systems in a setup with controlled human factors, i.e. the experiment only evaluates the capacity of the algorithms to track the intention of the teacher for approximating the policy.

Second, validation experiments with actual human teachers are carried out for learning balancing tasks in simulated and real environments. Finally, we compare the performance of users tele-operating the systems with the controllers learned interactively. This third comparative analysis is intended to show the performances that non-expert users can reach when they are teachers.

For all the experiments of this section the meta-parameters for the PS operation along with the corresponding with COACH are fixed. The amount of roll-outs for each PS iteration was set to $M = 10$; in the case of CEM the amount of elite samples is $M_{\mathrm{e}} = 5$. For COACH, the learning rate for the Human Feedback modeling was set $\beta = 0.3$, whereas the error magnitude e was fine tuned for every problem with an initial magnitude that is half of the action range, any change of this magnitude has proportional impact on the action updating, i.e. it is intuitive to tune for the user.

4.4.1 Learning with a Simulated Human Teacher

A first evaluation of the proposed methods is presented in this section, wherein a pre-trained policy is used for emulating a human teacher that advises the learning agent. The need to implement non-real human teachers is motivated by two reasons: (i) to evaluate robustness of the learning methods to mistaken human corrections, unfortunately it is not easy to quantify percentage of mistakes with real human teachers; (ii) to evaluate the learning strategies for correcting the policies itself in a transparent setup without influence of human factors, because humans not only perform mistakes, but also their attention and effort vary according to their motivation, mood and other factors (e.g., an unmotivated user who never advises corrections, never does mistakes, but also does not transfer any knowledge).

This experiment is carried out using the well known "cart-pole" RL problem [95], and our implementation is based in this environment[1]. The objective of the task is to learn the forces applied to a cart in order to keep the attached pole balanced. The observed states are the position and velocity of the cart along with the angle of the pole and its velocity respectively $s = [p, \dot{p}, \vartheta, \dot{\vartheta}]$. Usually the goal is to keep the pole balanced, but in this case the complexity of the task is slightly increased by the cost function to minimize $C(T) = \sum_{t=0}^{T} |p_t| - 1$, which requires the pole to be balanced with the cart in the center of the scenario in $p = 0$. The

[1]https://github.com/david78k/pendulum/tree/master/matlab/SARSACartPole

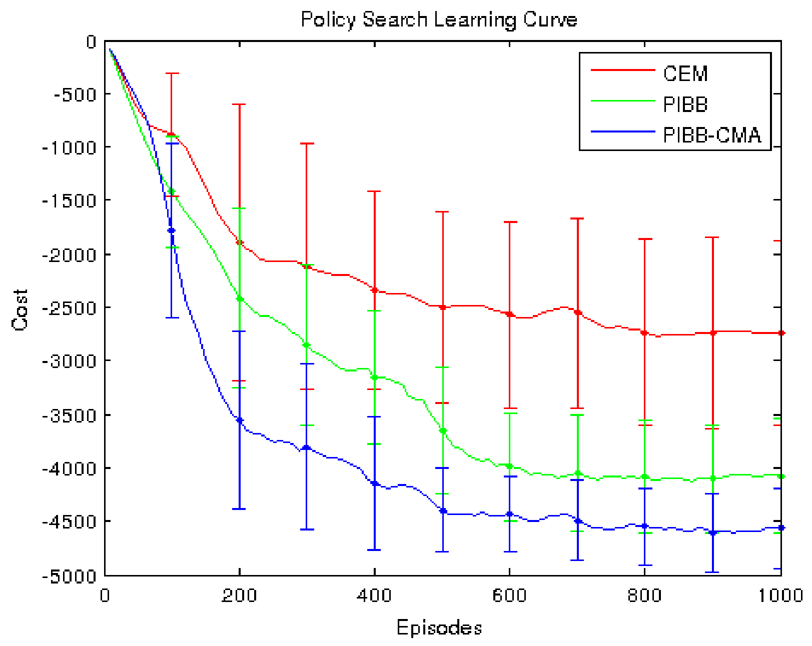


Figure 4.3: Learning curves of PS agents with the cart-pole task.

episodes start with the cart in the center of the scenario and the pole is in upright position; the episode finishes if the pole falls over, the cart reaches the boundaries of the scenario, or after 5,000 time steps. The policy is parameterized by splitting each state variable using 4 radial basis functions (RBF), for a total of 256 RBF features in the vector $f(s)$. For these experiments, 50 runs of each algorithm with constant parameters were carried out.

Policy Search Comparison

First, we compare the PS algorithms for learning to solve the cart-pole problem. The task is approached with agents performing the CEM, PI^{BB}, and PI^{BB}-CMA strategies. Each algorithm was run 50 times, and the statistical results are presented in Figure 4.3. The learning curves show that PI^{BB}-CMA agents obtain better final performances than the other two PS approaches. Both PI^{BB}-like agents have slightly faster convergence than CEM, and reach considerable lower cost functions. Additionally, the covariance adaptation component of PI^{BB}-CMA improved 10% of the cost obtained with the original PI^{BB}. Therefore, this faster algorithm is used in the experiments of the hybrid approaches proposed in this work.

Hybrid Agents with Simulated Teacher Comparison

The emulated human teacher is a block added to the learning loop, that contains the pre-trained policy $P_{pt}(s)$ from Section 4.4.1. The objective of this block is to advise corrections like humans do to the learning policy $P(s)$, in order to converge to a similar performance of $P_{pt}(s)$. The corrections h are the same kind of vague binary signals that users provide to a COACH agent, e.g., increase or decrease the executed action. Since the human teachers

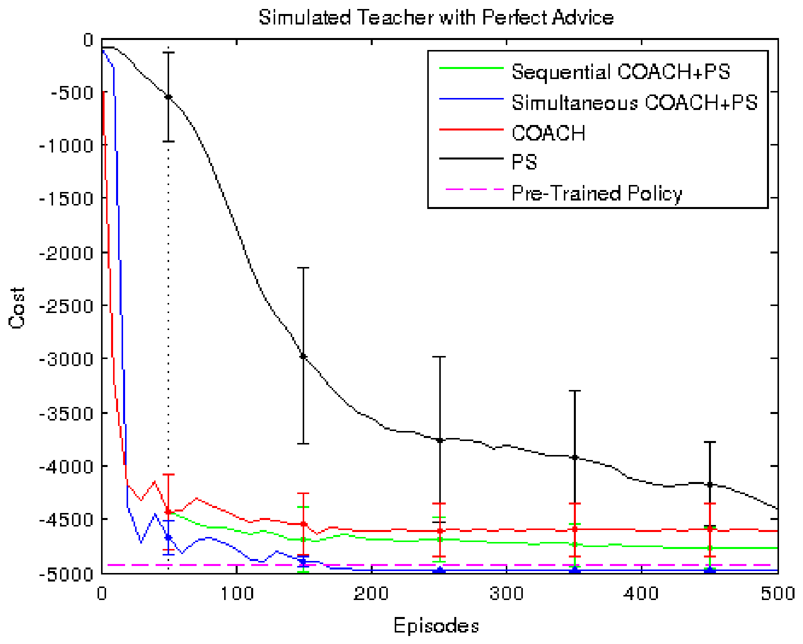


Figure 4.4: Learning curves of hybrid agents with simulated human teachers without mistakes.

do not correct in every time step of the executing policy, in this block the frequency of the advice is controlled with a probability ε. Therefore, the correction is computed every time step as:

$$h = \begin{cases} \operatorname{sign}(P_{pt}(s) - P(s)) & \text{with probability } \varepsilon \\ 0 & \text{with probability } 1 - \varepsilon \end{cases} \tag{4.7}$$

where h would be the "human corrective advice" of line 12 in Algorithm 3.2. The probability ε is diminished after every roll-out with a decay factor of 0.95.

In these experiments, we also consider the fact that in general mistakes are always inherent to human feedback or demonstrations. Therefore, for the simulated teacher, wrong corrective advice is provided to the agent with a probability η as

$$h = \begin{cases} -h & \text{with probability } \eta \\ h & \text{with probability } 1 - \eta \end{cases} \tag{4.8}$$

The probability η was varied for running learning processes with 0%, 20%, and 40% of mistakes. In the case of the SeqCOACH+PS, the human feedback is given to the COACH agent only during the first 50 roll-outs, this is indicated with a dotted line in the plots. Then, the PS exploration continues the learning process. Additionally, the hybrid approaches are compared to the performance of the pure COACH agent, the PI^{BB}-CMA approach previously presented in Figure 4.3, and the performance of the pre-trained agent used for simulating the human teacher using (4.7).

The experiments show that for this task, approaches that use the "human feedback" learn faster than the pure Policy Search agent. COACH-only agents have the fastest learning in the very first episodes, however they converge to lower performances than the hybrid

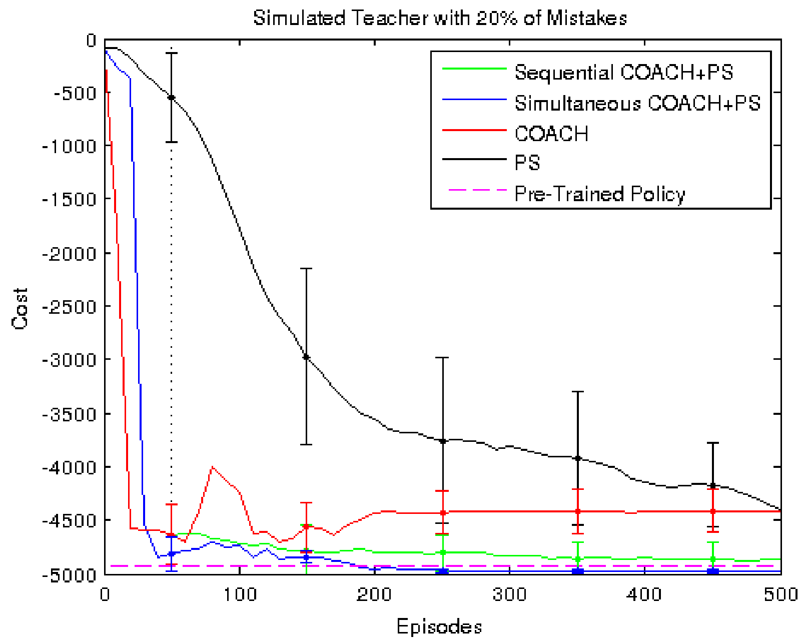


Figure 4.5: Learning curves of hybrid agents with simulated human teachers with 20% of mistaken advice.

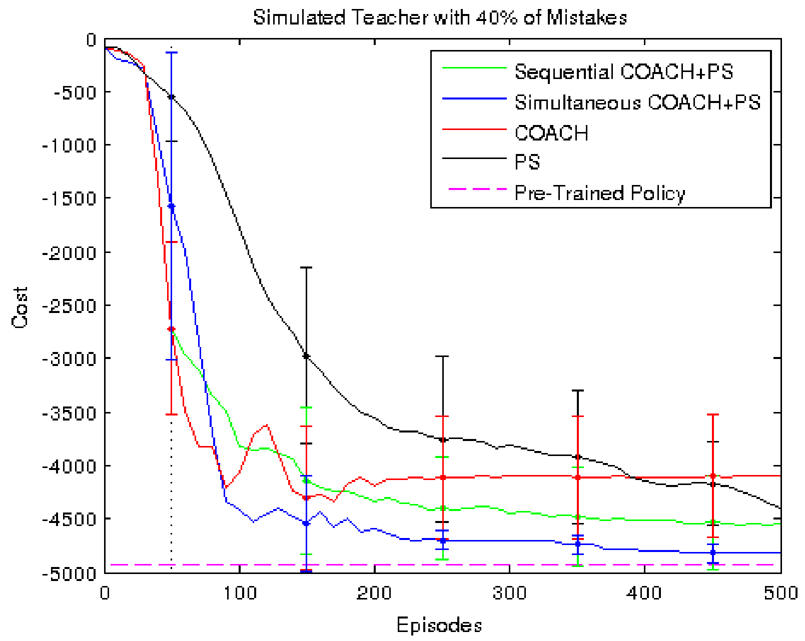


Figure 4.6: Learning curves of hybrid agents with simulated human teachers with 40% of mistaken advice.

agents. Moreover, COACH performance is the most sensible to mistaken feedback, since its convergence decreases more than the other agents when the probability of mistakes η is higher. The previous observation is expected because the human advice is the unique source of information for COACH. Hence, the policy performance only depends on the quality of the feedback. This is one of the motivations for combining COACH with RL.

SeqCOACH+PS shows the most stable convergence, since from the 50th episode on, there is no human advice that might harm the already good policies obtained with COACH. The results of the experiments with 0% and 20% of mistakes (Figure 4.4 and Figure 4.5), show that when the stage of learning with COACH stops and the PS phase starts, the learning curve turns into the least steep but the most monotonic. This happens as PS can be seen as a fine tuning of the policy obtained by the COACH stage, which quickly learns good policies. In the case of 40% of mistakes in Figure 4.6, the PS stage drastically improves the performance, the learning curve looks like the pure PS convergence, but displaced to the left by 300 trials.

The SimuCOACH+PS approach obtains the best final performances of the policies compared to the other agents, it even outperforms the pre-trained policy used for simulating the teacher when there is 0 and 20% of erroneous feedback. The simultaneous combination of interactive learning and PS makes the learning slower in the very first episodes compared to COACH. However, it outperforms the COACH's performances after 30, 40, and 90 episodes approximately in the cases of "human feedback" with 0%, 20%, and 40% of incorrect advice respectively. This improvement with respect to SeqCOACH+PS is also obtained because in this simultaneous case, the covariance matrix is computed since the very beginning of the training, then it can be seen as a dimensionality reduction, because when the learning is left only to the RL agent, the exploration could be very small in some of the parameters.

These experiments show that hybrid approaches are more robust to noisy corrections, and faster than pure autonomous or pure interactive learning approaches.

4.4.2 Learning with Real Human Feedback

A more detailed validation of the proposed methods is carried out with experiments involving real human teachers interacting with simulated and real problems, in which the corrective feedback is provided to the agent with a keyboard. Again, the cart-pole problem is approached along with a real inverted pendulum swing-up [2]. In these experiments five participants interacted with the agents as teachers. They advised SimuCOACH+PS agents during the episodes they considered appropriate. The users also interacted with the COACH agent during 30 episodes, and the obtained policies were used for the initial policy in the second stage of the SeqCOACH+PS. The learning curves are also compared to those obtained by pure PS agents. The video[2] shows the interactive learning and execution of the agents.

[2]https://youtu.be/VIJiK7Rhe4o

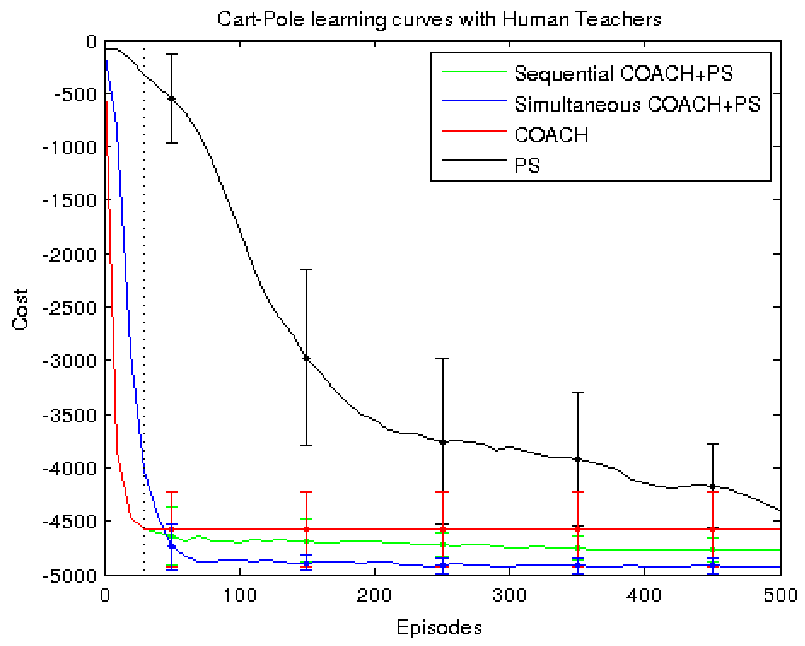


Figure 4.7: Learning curves of the experiments for the cart-pole problem with real human teachers.

Cart-Pole

In this validation, the same environment used in the previous experiments is used for learning to execute the task with support of real humans. Results in Figure 4.7 show that all the agents that use human advice obtain five times faster convergence than pure PS, and that the hybrid agents obtain better policies than pure COACH or pure PS. The SimuCOACH+PS obtains the best performances, but in the very first episodes is slower than COACH, and consequently slower than the sequential scheme. For this problem it is possible to see that the convergence is more monotonic than the experiments with simulated human teachers, although lower final performances are obtained. This can be due to the capacity of real human teachers that are adapting to advise the current policy, leading to different final policies with similar performance, whereas, the simulated teachers only try to teach to imitate the pre-trained policy.

Swing-Up Pendulum

The second experiment is carried out using an under-actuated inverted pendulum swing up, which is a weight attached to a DC motor depicted in Figure4.8. The observed states are the angle and its velocity $s = [\vartheta, \dot{\vartheta}]$, while the action is the voltage u applied to the motor, which is in the range [-2, 2]V. As the motor does not have enough force to rotate the pendulum up directly, the first problem to solve is to learn to swing the weight back and forth for reaching the upper position ($\vartheta = \pi$). Then, the second problem is to learn to keep the pendulum balanced in the unstable equilibrium. In this case, each state variable is split into 20 RBF basis functions (20x20) for a total of 400 features that compound the vector

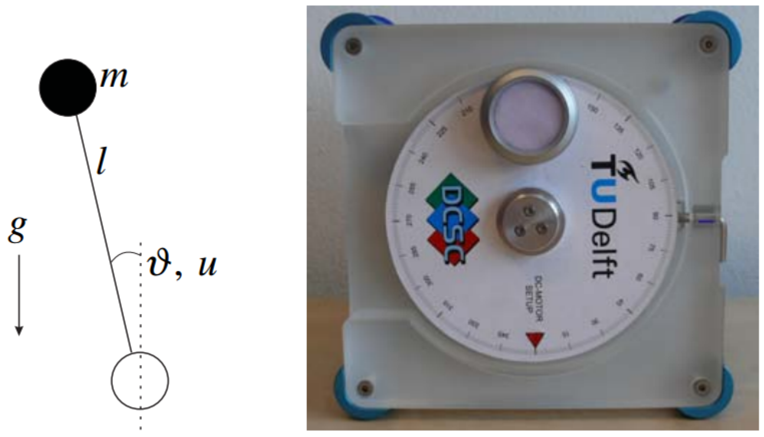


Figure 4.8: Inverted Pendulum Swing-Up setup (taken from [2]).

f. The episodes are finished after 500 time steps (10 s), and the cost function is given by $C(T) = \sum_{t=0}^{T} -(\vartheta_t/\pi)^4$.

For this problem COACH uses an Inverse Kinematics (IK) model to map the advice from the effector space to the actuator space. The user provides corrections like “move the pendulum more towards right, left, up, or down” with the arrows of a keyboard, and this module based on the current state, uses the IK for mapping the advice into “apply more or less voltage to the motor”.

The results obtained with the real system are very similar to the simulations (both in Figure 4.9). In this new set of experiments we again observe that all interactive approaches are faster than the pure PS. However, the performances obtained with only COACH are outperformed by PS after several episodes. The PS agent takes seven times more episodes to reach the performance reached by the users after 20 trials advised with COACH. The hybrid schemes have the best cost indices; the sequential approach successfully employs PS to fine-tune the initial policy obtained via COACH. The simultaneous scheme also has slower convergence than pure COACH in the first episodes, but keeps improving until reaching the highest performances.

SimuCOACH+PS is slower than SeqCOACH+PS at the beginning, because in the simultaneous scheme, the probability weighted average computed by the PS can be compared to a lowpass filter, that avoids drastic changes in the parameters that might obtain considerable positive or negative impact on the performance. Nevertheless, during the episodes previous to the convergence, the occasional human feedback (not present in SeqCOACH+PS at that moment) seems to be more efficient than the random exploration given by the normal distribution of (4.1).

3DoF Arm Inverse Kinematics

This third evaluated problem is about learning the inverse kinematics model for a real 3 DoF robot arm (Figure 4.10). The model has to map the input request of a 3D coordinate

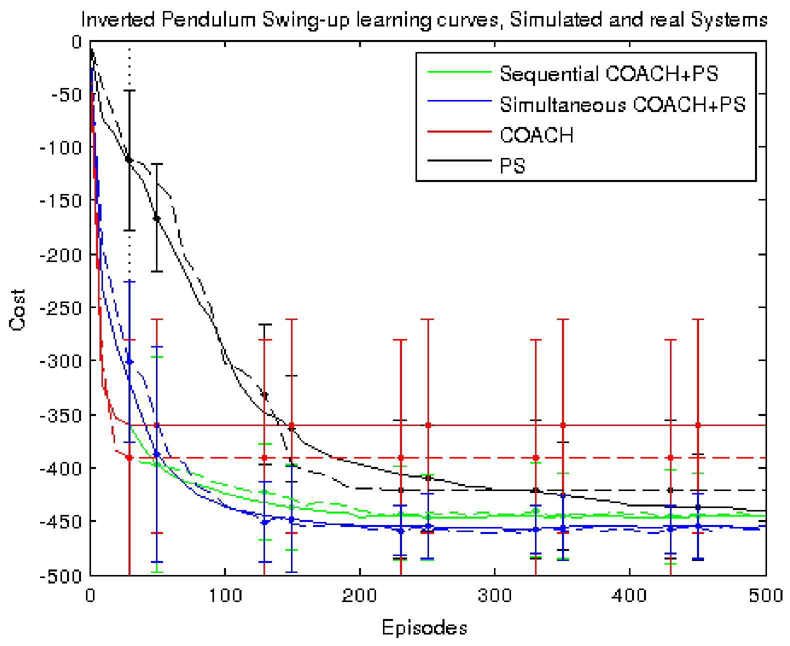


Figure 4.9: Learning curves of the experiments for the inverted pendulum swing-up problem with the simulated system (normal lines) and the real system (dashed).

position into the space of the angles of the three servos that compose the robot arm, which is the output. The cost function used in this problem is the Euclidean distance between the points requested to the model and the actual arm's end effector position, which is given in centimeters.

In this application, when the human teachers interact with the learning robot, they observe the position of the end effector with respect to the target point, then they provide corrective advice in the joints space for decreasing the distance. This robot does not have an operation mode for kinesthetic teaching, so this corrective advice is the only way to obtain human feedback in the action domain. For the experiments of learning with COACH and SeqCOACH+PS, the human teachers only interact with the robot during the first five episodes, whereas with SimuCOACH+PS the users continue advising when considered necessary.

The results obtained from the learning processes in Figure 4.11 show similar trends than the previous experiments. In this case, the PS algorithm, which is a local search methods, converges to local minima. With the interactive agents, the obtained costs are considerable lower than the reached with only PS. Again SimuCOACH+PS showed to be slowest at the first episodes, however it outperforms the performance of pure COACH after 40 roll-outs. For this problem the sequential hybrid algorithm converge to a slightly lower cost than the simultaneous counterpart, however both reduce 40% of the error obtained with only COACH, thus the average error of the hybrid agents is 7mm, while in the best run the policy model converged to an error of 3mm.

In this experiment, the human knowledge leveraged the convergence of the policy, the visual perception of the human teacher can help in the very first part of the learning process, nevertheless, that can bring on inaccuracies, especially for sensing depth. However, the

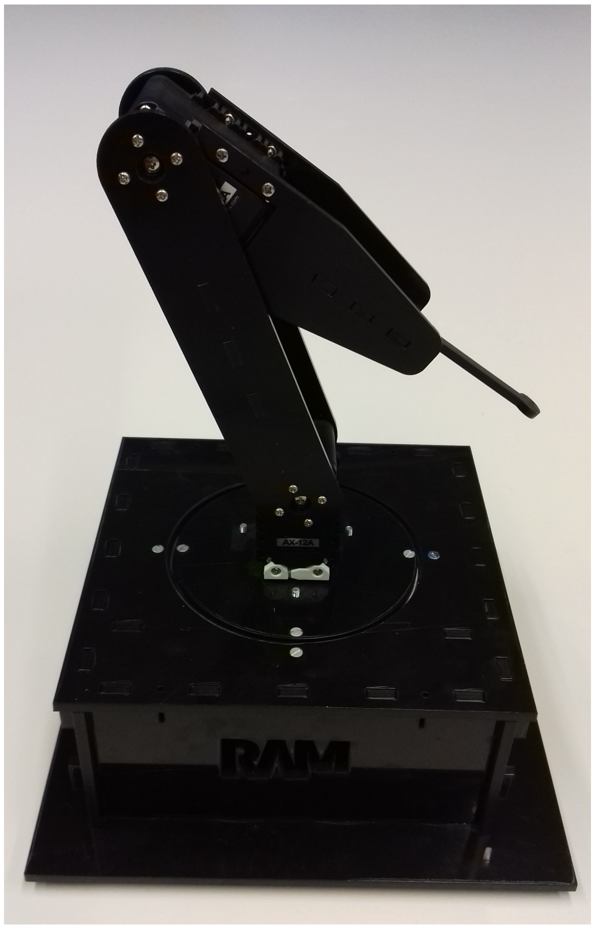


Figure 4.10: Robot Arm used for learning the IK model.

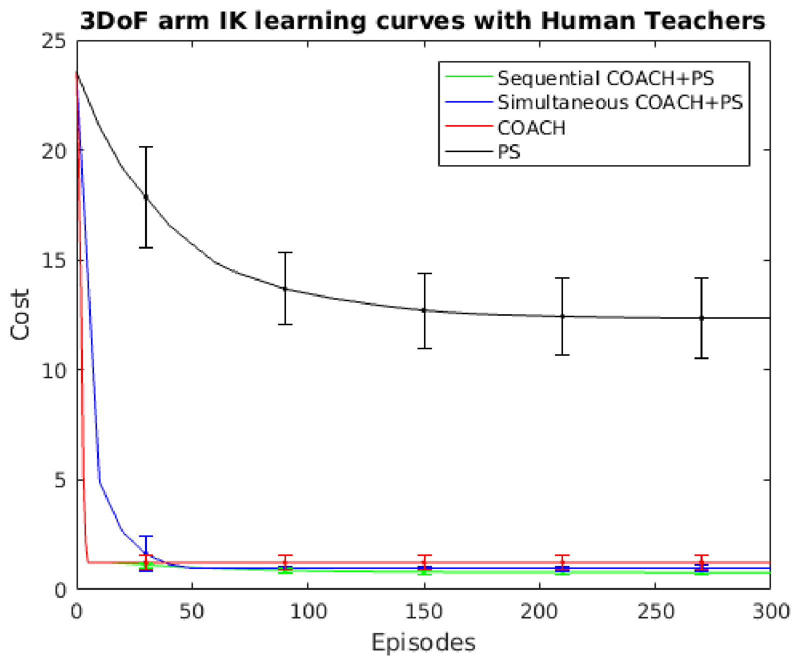


Figure 4.11: Learning curves of the experiments for the inverse kinematics for a 3 DoF robot arm. Average Cost in cm.

Table 4.2: Comparison of users tele-operating and teaching

Task	**Performances**	
	Tele-operation	**Interactive Learning**
Cart-pole	-249.13	-4576.43
Simulated Pendulum swing-up	-81.79	-360.52
Real Pendulum swing-up	-68.44	-390.55
Simulated Inverted Wedge	0.2276	0.07024
Real Inverted Wedge	0.2718	0.0838

RL component of the algorithms, that evaluates based on the cost function supports the refinement of the model for attaining a better accuracy.

4.4.3 Comparison between tele-operation and learning

In this set of experiments, the abilities of the users for teaching the agents to perform the tasks are compared to the capacities of the users for actually executing the tasks with tele-operation. The participants interacted with the system several trials for learning to execute the tasks. Their best execution is compared to the performance obtained using COACH. Additionally, an inverted wedge is approached as fourth case study; for this balancing task the cost function is the average angle per episode, where the equilibrium point is with zero degrees. Execution of this task is shown in the video.

Table 4.2 shows the results comparing the users' tele-operation and teaching; since the cost functions have to be minimized, the lower the index the better the performance. For all the cases explored in this chapter, the users did not learn to tele-operate the systems successfully, as reflected in Table 4.2. They kept the equilibrium only for a few seconds for all the problems except for the pendulum swing up, in which the pendulum was never balanced. It is interesting to observe that the users can obtain agents which can perform tasks that the human teachers cannot demonstrate, i.e., the interactive learning approaches based on advise of corrections let non-expert users in the task domain teach policies of good quality from vague binary pieces of advice. The numeric results in Table I show the big difference between the two options of interaction of users with the systems (operating and teaching). This highlights the advantages of sharing the work with intelligent systems that can learn from the users who are not able to provide good demonstrations. These outstanding features of the hybrid approaches are desirable in environments where users frequently need to adapt the agent to new conditions or tasks.

4.5 Discussion

This chapter has presented and validated approaches for policy search supported with human feedback. Two schemes of combining PS with COACH were presented: a sequential scheme and a scheme that learns simultaneously from human and autonomous feedback.

The experiments with balancing tasks and the Inverse kinematics model showed that the hybrid algorithms can benefit from the advantages of both kinds of learning strategies, where the corrections provided by human teachers result in fast learning to a high but suboptimal performance, whereas PS can optimize policies based on cost functions that are not very explicit or intuitive to the users' understanding, or simply when the human perception becomes limited to support the learning process. Therefore, the addition of human support to PS speeds up the convergence between 3 to 30 times according to the results obtained. From the point of view of interactive machine learning, these hybrid strategies provide more robustness to the convergence, since the sensitivity to noisy or mistaken corrections is diminished. Moreover the quality of the policies is improved with the cost based corrections of PS which perform fine tuning of the policies taught by the users.

The proposed hybrid schemes showed to be better choices than pure COACH or PS frameworks in applications that need fast learning. The SeqCOACH+PS is a simple scheme easy to implement and completely agnostic of the type of PS used; it facilitates the learning especially in the first trials. The SimuCOACH+PS scheme showed to be slower than the sequential one at the very beginning. However, it benefits from the occasional corrections given by the teachers, which guide the exploration to the highest achieved performances.

The comparison of all the learning approaches, and even the performance of the users tele-operating the agents, shows that the proposed strategies can have high impact with the cyber physical systems of the coming industry developments, that require to reduce the workload of factory operators and also to ease the adaptability of the products for the final users who can interact for modifying the operation of technological products.

This Chapter satisfies the proposed fourth objective. The validations of COACH in three real systems allowed us to demonstrate that the learning method satisfactorily works also in physical systems. In the next Chapter more applications with simulated and real robot arms are presented. That chapter presents also hybrid learning schemes of COACH and RL, but with the focus on learning trajectories. Additionally, the COACH method is generalized for problems of multiple action dimensions with problems of mismatch between the human teacher feedback space and the actions domain, that is related to the third objective of this thesis.

Chapter 5

Policy Search with Human Corrective Feedback for Motor Primitives Learning

In this chapter, we propose the use of human corrective advice in the actions domain for learning motor trajectories. Additionally, we combine this human feedback with reward functions in a Policy Search learning scheme. The use of both sources of information speeds up the learning process, since the intuitive knowledge of the human teacher can be easily transferred to the agent, while the Policy Search with the cost/reward function take over for supervising the process and reducing the influence of occasional wrong human corrections. This interactive approach has been validated for learning movement primitives with simulated arms with several DoFs in reaching via-points movements, and also using real robots in tasks like "writing characters" and the game ball in a cup. Compared to a standard Reinforcement Learning without human advice, the results show that the proposed method not only convergences to higher rewards when learning movement primitives, but also the learning is sped up by a factors of 4 to 40 times depending on the task.

5.1 Introduction

Robot motor skill learning has been subject of research for several years. Machine Learning has been used for obtaining trajectory representations, since several applications face the necessity of encoding sequences of points into a policy model that can be used to execute a motor skill. Some Motor Primitives models have been used to represent the movements required for tasks like 'drumming'[82]; 'T-ball batting' [80]; 'ball in a cup' [59]; 'pancake flipping' [62]; 'ball throwing', 'dart throwing', 'robot table tennis' [60]; and 'golf swing' [67] among others.

The mentioned tasks have been solved using policies based on models that present some generalization capabilities such as the Dynamic Movement Primitives (DMP) [41], primitives based on Gaussian Mixture Models [44], or Probabilistic Movement Primitives (ProMP) [79]. These models have interesting properties that can be convenient in many applications.

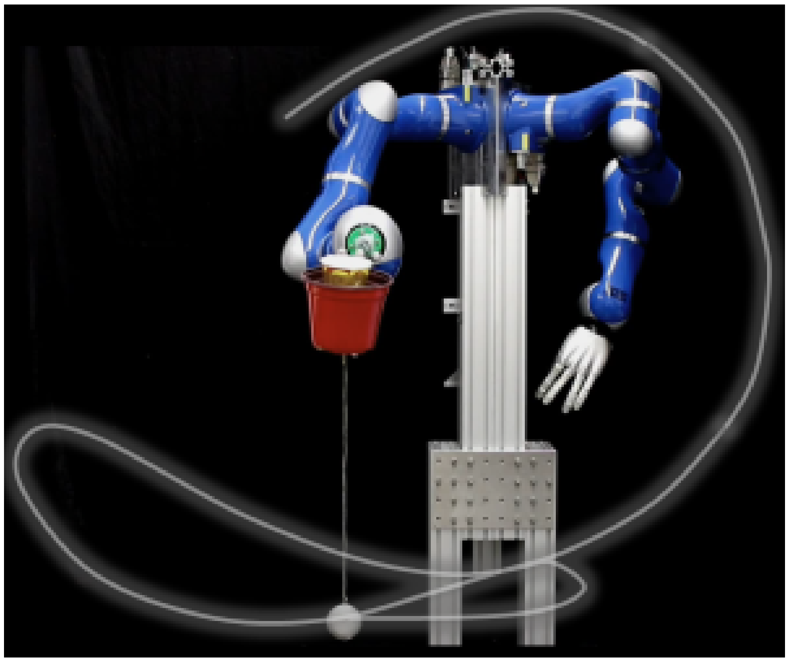


Figure 5.1: Ball-in-a-cup task execution of a policy learned using the Interactive Policy Search method proposed.

Movement primitives can be learned through demonstrations, and/or by self-improvement using Reinforcement Learning.

In robotics, RL [57] has been used to learn and improve movement primitives, often initially acquired from demonstrations. RL-PS methods have shown particularly suitable for learning with real robots [32], attaining higher rewards with respect to the initial (demonstrated) policy. However, the optimization process requires a large number of trials, which is usually impractical or expensive when using real systems. As it was said in the previous chapter, PS strongly relies on good demonstrations, since it is a local search method [32], but for certain tasks the human may not be able to provide useful demonstrations, particularly when the dynamics of the task or limitations of the robot are unknown. For example, how many swings are required to achieve a ball-in-a-cup task given a heavy ball and a robot with limited accelerations that cannot toss the ball high enough in one shot. This lack of intuition suggests some form of interactive learning process where human knowledge is added/transferred as the robot optimizes the policy.

In this chapter, we propose a method for learning motor skills with real robots, which makes feasible the convergence in few episodes. This extends the ideas of the methods presented in the previous chapter, but to be applied in trajectory learning for movement primitives, combining PS algorithms with human corrections, in order to leverage the exploration provided by policy search with the knowledge of a human teacher. The method here proposed guides the exploration of a PS algorithm with the human teacher's current knowledge of the task. We assume this process is dynamic in the sense that the teacher knowledge also improves as he/she observes the effects of his/her corrections through the interaction of the robot with the environment and its respective outcomes.

The experiments presented in this chapter show that the proposed method can be used by users to shape detailed trajectories with vague action corrections. Significant reductions of the number of trials required to learn reaching movements with simulated arms are presented. Actually the introduced method is tens of times faster than conventional PS when the arm has 50 DoF. The method is also tested in the real world problem ball-in-a-cup task (Figure 5.1). In this case, successful policies can be obtained 4 times faster respecting the traditional PS approach.

5.2 Related Work

Interactive corrections can be applied in the context of movement primitives. In this case, corrections are used to update the parameters that shape the characteristics of the robot movement. In [25] a feeding assistance robot is pre-programmed with a ProMP [79] for feeding disabled people. Then, a framework is proposed to allow caregivers to personalize the original trajectory to the preferences of the disabled person. In this framework the caregiver physically adapts the ProMP execution through kinesthetic feedback. The new executed path is recorded to create a new ProMP. In [10] a system is proposed that allows the modification of a primitive during execution with tactile feedback. If the user provides a correction with an effector displacement with respect to the original trajectory, the displacement is applied from there on to the rest of the path, then all the data-points are recorded for re-deriving the policy after the execution.

Kinesthetic teaching is used for incremental refinement of trajectories of context-dependent policies [34] represented with ProMPs, wherein the user may modify the trajectory execution for performing a correction. Then the data-points of the recorded trajectory are applied to update the probability distribution of the ProMP. That method was tested in reaching tasks with a robot arm.

However, there are a number of cases where none of the previous approaches can actually be applied. Here we describe a few scenarios: (a) There is no expert available to provide high-quality demonstrations that lead to a policy with acceptable performance (*e.g.* a person with disability without a caregiver, who needs to fix a policy for a new task or environment); (b) the final user does not have an available interface to provide new demonstrations such as intuitive tele-operation interfaces, wearable sensors, or motion capture systems; (c) the robot is not back-driveable or its dimensions are not suited for direct human physical interaction required by kinesthetic teaching; and (d) the task involves fast robot movements, making kinesthetic corrections impractical and/or unsafe for the teacher.

In these cases, wherein detailed feedback cannot be provided to the learner, approaches like A-OPI or COACH that are based on simpler signals of correction are better suited. Since there is no possibility of detailed corrections, these algorithms are suitable because in this case the human-agent interface does not need tactile or force sensors, and can be limited to simple, sparse, occasional and vague commands given with keyboards, voice commands or gestures.

The closest work to our approach is [87], where the problem caused by the absence of expert demonstrations was partially overcome by combining a PS algorithm with the interactive definitions of via-points for adapting DMPs. In that work, the teacher could stop the trajectory execution and move (physically or remotely) a robot to a desired position at that specific time step. This via-point correction was then used to update the distribution used for the PS exploration. The method was validated in simulations of writing letters and object insertion with robot arm. The approach in [87], however, only addressed the kinematics of the tasks, focusing on the shape of the trajectories. This interactive PS strategy is not suitable for tasks where the dynamics are dominant (e.g. throwing and catching an object) since stopping the task to adapt a specific via-point is physically impractical and the instant of correction is not evident. Also, modifying a via-point to satisfy certain kinematic configurations invariably affects the acceleration and thus the outcomes of a dynamical task.

In this thesis, we leverage on COACH [26] as the mechanism for interactive corrections as it allows human feedback to be introduced *during* robot execution. However, COACH was originally designed for interactive optimization of policies in a Markov Decision Process (MDP) setting. Therefore, we provide a new formulation of COACH for time-dependent parametrized policies that makes it compatible with the improvement of movement primitives.

5.3 Corrective Advice for shaping movement primitives

This section proposes modifications of the COACH algorithm for training parameterized movement primitives. The original COACH algorithm was proposed on a MDP setting [26] where human feedback is used to improve a policy that evolves under Markovian assumptions in a state-space $\boldsymbol{s}$. Here, the foundations for using COACH in the context of movement primitives is to allow for human feedback on a policy whose evolution is (indirectly) governed by time. We focus on movement primitives such as DMPs and ProMPs whose evolutions are defined by a phase variable z_t, but other representations are also possible [44]. Essentially, the representation changes from the state variable $\boldsymbol{s}$ to the phase variable z_t. In DMPs, the policy is represented by a dynamical system comprised of a linear spring-damper (5.1) attached to a goal attractor g.

$$f_t = \alpha(\beta(g - x_t) - \dot{x}_t). \tag{5.1}$$

This system is modified by an arbitrary non-linear term $\mathbf{g}(z_t)^\top \boldsymbol{w}$ for shaping complex trajectories such that its acceleration is:

$$\frac{1}{\tau}\ddot{x}_t = f_t + \mathbf{g}(z_t)^\top \boldsymbol{w}, \tag{5.2}$$

where $\boldsymbol{w}$ is the parameters vector (or the weight vector), and $\mathbf{g}(z_t)$ is the basis function vector.

In the case of ProMPs, the policy model is a probability distribution in the parameter space, represented with a mean and a co-variance matrix which are obtained from a set of demonstrations. During execution, this model is conditioned on an observation for computing a trajectory associated to that specific context. Using Bayesian inference, a set of parameters $\boldsymbol{w}$ of a linear model of basis functions is obtained and the trajectory is computed with (5.3)

$$x_t = \mathbf{g}(z_t)^\top \boldsymbol{w}. \tag{5.3}$$

COACH can be used to optimize the policy by updating the parameter vector $\boldsymbol{w}$ via human advice. The update uses stochastic gradient descent with the error assumption in Section 3.2.1, and the derivative

$$\frac{\partial x_t}{\partial \boldsymbol{w}_l} = [\mathbf{g}_t]_l, \tag{5.4}$$

where the right-hand side of the equation represents the l-th basis function at time step t. Since this work is focused on correcting single trajectories, in the cases of ProMPs, the corrected vector $\boldsymbol{w}$ can be used for updating the global mean and co-variance matrices as in [34].

The user advises local corrections during the motor skill execution. This advice propagates over the next time steps as changes of the weights are smoothed ahead by the shape of the basis functions (usually radial-basis functions), so the effect of the correction can be appreciated immediately.

While COACH can be used to advise corrections in both task or joint space of the robot, advising corrections at the joint level is not intuitive for robots with multiple degrees-of-freedom. Thus, hereinafter, derivations will address the case where corrections are made on the task space of the robot. In other words, we will assume that the human intention is to correct the robot's end-effector movement as opposed to its joint movements. Under this assumption, since policies can be represented either at the task or joint levels two realizations of the algorithm must be addressed. If the policy is represented at the task level in the Cartesian space, an Inverse Kinematics (IK) function must be used as a last step to map the learned policy in the joint space of the robot. Conversely, if the policy is represented in the joint space, the IK function must be used on the human feedback to translate the human correction into the corresponding space. Both cases will be described in detail.

5.3.1 Case of policies defined in the Cartesian space

For policies that represent the end-effector trajectory, the learning scheme is simpler because according to our assumption human corrections are used to modify the policy directly. However, this mode of operation is constrained to the robot at hand, because an operation mode that request commands in the end effector space and maps it to the actions in the joints level is required, e.g. the IK with the position request in order to obtain the respective vector of joint angles. In Figure 5.2 it is depicted the scheme for this case of policies. In the left hand side the stage in which the demonstrations are gathered and the initial policy is obtained; in the right hand side is the stage of learning with COACH wherein the human teacher advises the executed action. The *Update* block computes the COACH assumptions and modifies the parameters vector of the policy; the blocks surrounded by the red dashed line work during

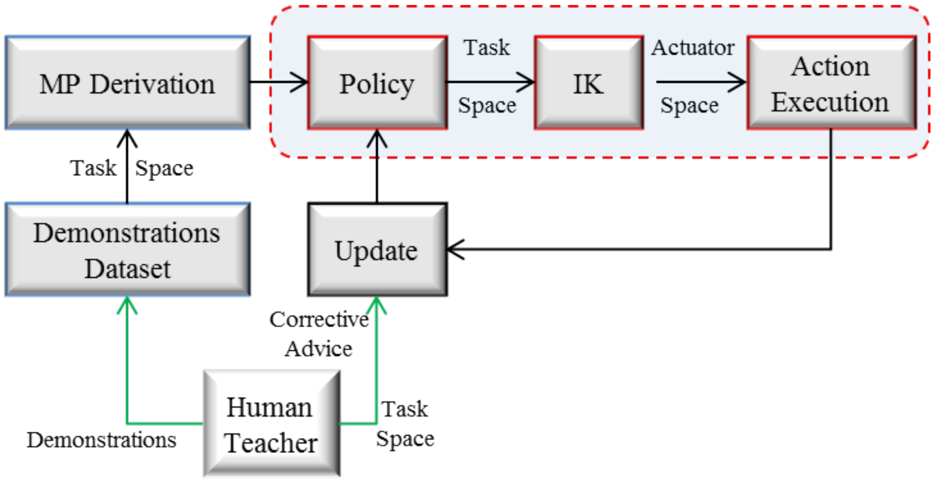


Figure 5.2: Learning Scheme for Movement Primitives in the Cartesian space.

and after the learning process for executing the current policy. Between the policy computation and the action execution there is a block for mapping the action computed by the policy onto the actuators space.

In Algorithm 5.1 is defined a version of COACH for trajectories represented with a linear combination of basis functions. It is possible to see that at the beginning some variables are stated, first the initial positions of the joints $\boldsymbol{q}_t$ to be used by the Inverse Kinematic model (line 1). Then the magnitude for the error assumption e in Section 3.2.1 and the learning rate β are defined (lines 2-3). First, the weights c_t that represent the probability function of the credit assigner are computed (lines 4-6). The loop between lines 7 and 24 is executed once per time step. The function $getBasisFunctions(z_t)$ maps the phase variable to the features vector $\mathbf{g}_t$ (line 8); $\boldsymbol{X}_t$, the point of the trajectory at time step t in the effector domain is computed by the linear model (line 9); the Inverse Kinematic (IK) is computed to obtain the joint requests $\boldsymbol{q}_t$ (line 10). The action is executed by the robot (line 11) and if a human advice h is received before the next time step (line 13), the condition in lines 15-22 is executed. If the human teacher provides a correction, the features vector that includes the weighted sum of the past features vectors $\boldsymbol{\Phi}^{cred}$ is computed (lines 15-17); afterwards, the Human Feedback prediction $H(z_t)$ is computed (line 18), and its parameters υ updated with the SGD rule (line 19); the adaptive learning rate $\alpha(z_t)$ is obtained (line 20). The error assumption from Section 3.2.1 is computed (line 21), named $\boldsymbol{error}_X$ where the subscript X means that this error is defined in the Cartesian space. Then the policy model is updated in a similar way as the Human model but using the $\boldsymbol{error}_X$ assumption (line 22).

The function $IK(\boldsymbol{X}_t, \boldsymbol{q}_t)$ takes the end effector pose $\boldsymbol{X}_t$ and the current joints vector $\boldsymbol{q}_t$ in order to find a solution $\boldsymbol{q}_t$ that is close to the current input. For instance, if the IK solver is iterative, the current $\boldsymbol{q}_t$ is used as the initial value for the iterations, otherwise, with a closed IK representation the current $\boldsymbol{q}_t$ used as input is taken for choosing among several solutions, based on a similarity criterion.

A simplified version of the Algorithm 5.1 can be implemented for simple and slow problems. This version disconnects the Credit Assigner module and the Human Feedback modelling module. In this regard, the implementation ignores lines 4-6, and lines 15-20. Then $\alpha(z_t)$ is

Algorithm 5.1: COACH for training a MP in the Cartesian space.

```
1:  $q_t \leftarrow q_i$
2:  $e \leftarrow constant$
3:  $\beta \leftarrow constant$
4:  for $1 \leq t \leq n$
5:    $c_t \leftarrow assignCredit(t)$
6:  end for
7:  for all $z_t$ do
8:    $g_t \leftarrow getBasisFunctions(z_t)$
9:    $X_t \leftarrow g_t^T w$
10:   $q_t \leftarrow IK(X_t, q_t)$
11:   $takeAction(q_t)$
12:   wait for next time step
13:   $h \leftarrow getHumanCorrectiveAdvice()$
14:   if $h =! 0$
15:     for $1 \leq t \leq n$
16:       $\Phi^{cred} \leftarrow \Phi^{cred} + (c_t \cdot g_t)$
17:     end for
18:     $H(z_t) \leftarrow {\Phi^{cred}}^T v$
19:     $\Delta v \leftarrow \beta \cdot (h - H(z_t)) \cdot \Phi^{cred}$
        $v \leftarrow v + \Delta v$
20:     $\alpha(z_t) \leftarrow | H(z_t) | + bias = | {\Phi^{cred}}^T v | + bias$
21:     $error_X \leftarrow h \cdot e$
22:     $\Delta w \leftarrow \alpha(z_t) \cdot error_X \cdot \Phi^{cred}$
        $w \leftarrow w + \Delta w$
23:   end if
24: end for
```

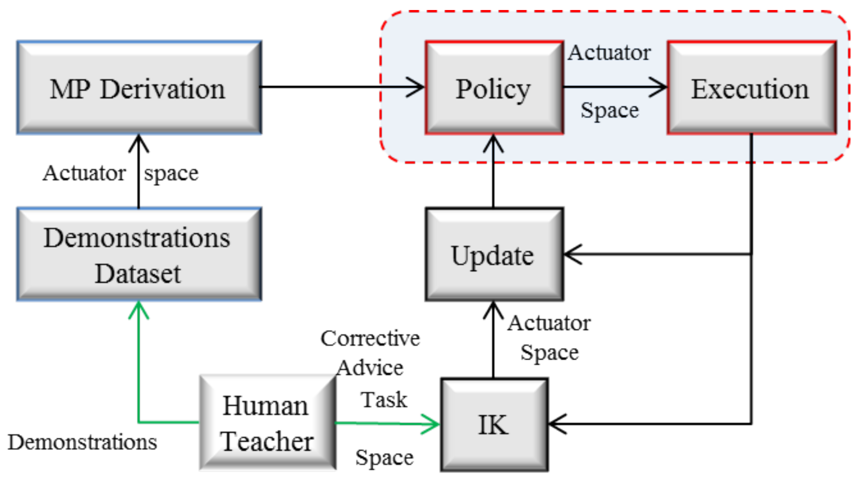


Figure 5.3: Learning Scheme for Movement Primitives in the joint space.

set constant, and the vector $\mathbf{\Phi}^{cred}$ is replaced by $\mathbf{g}_t$ in line 22.

5.3.2 Case of policies defined in the joint space

Policies that map directly the actions in the joint space are simpler to execute as they are already given as actuator commands. However, during learning, more steps are required for solving the correspondence problem between the human advice and the actuators space. Figure 5.3 shows the scheme for learning this kind of policies, in which the user advises the correction in the task space, an inverse kinematics block propagates this correction to the joints space, and the updating block modifies the weights $\boldsymbol{w}$ based on the modules of COACH. During execution of the policy (area surrounded by the red dashed line), the inverse model is not needed because is used only during the time steps advised by the teacher in the learning process.

The COACH version for this learning scheme is in Algorithm 5.2, where the kinematic models are only used when human advice is given (policy updating in lines 13-25). The policy computes directly the action in the joints space $\boldsymbol{q}_t$ (line 8). In contrast to the case in Section 5.3.1, here the human model H (which is always in the task domain) and the policy model are in different spaces. Therefore, the rule for computing the error in the policy update of Section 3.2.1 is redefined. The adaptive learning rate of the policy from the original COACH is now included in the error as below

$$\boldsymbol{error}_X = h \cdot \mathrm{e} \cdot \alpha(z_t). \tag{5.5}$$

When a correction is suggested by the teacher, the $\boldsymbol{error}_X$ assumption in the Cartesian space is computed (line 20). The current effector position is computed with forward kinematics and added to the $\boldsymbol{error}_X$ in order to obtain the "desired" effector position (line 21), which is used to obtain the "desired" joint vector (line 22). The difference between the "desired" joint vector and the one computed by the current policy is considered the propagation of the error assumption from the task space to the actuator space, called $\boldsymbol{error}_q$ (line 23). Finally the SGD is computed for updating the weights $\boldsymbol{w}$ (line 24).

Notice that in the update (lines 12-25), the vector $\boldsymbol{q}_t$ is not computed with the current basis functions vector $\mathbf{g}_t$, but with the basis functions given by the credit assigner, i.e. the expected action according to the used human delay probability distribution.

For disconnecting the Credit Assigner and Human Feedback Modelling modules, the implementation should eliminate lines 3-5, 13-16, and 18-19. Additionally, in line 17 the vector $\boldsymbol{\Phi}^{cred}$ is replaced by $\mathbf{g}_t$, the term $\alpha(z_t)$ should be eliminated from line 20, and set a constant α to multiply the Δw in line 24.

5.4 Corrective Advice and Policy Search for shaping movement primitives

The learning methods presented in the previous section are fully based on human corrections. This can be useful in some simple problems, but, in some others, the inherent drawbacks of interactive learning related to the human mistakes and capabilities may influence more negatively the convergence. Similarly, to the previous chapter, this section presents a core contribution of this chapter—a synergistic combination of Policy Search (PS) with human corrections for learning movement primitives—in the sense that PS is used to reduce the impact of erroneous human feedback, while correct human feedback is used to speed up the learning process of a PS algorithm.

The here proposed method is an extension of the simultaneous PS and COACH presented in the previous chapter. So, according to the cost function, the PS algorithm decreases or "filters out" the influence of improper corrections of the trajectories given by the teacher. In essence, our proposed method enables human guidance based on COACH to influence and bias the exploration of a PS algorithm in the form of exploration noise. Figure 4.2 depicts the proposed scheme. An initial parameter vector $\boldsymbol{w}^{init}$ is disturbed either with the original exploration strategy defined by the PS algorithm or with human guidance through the roll-outs execution. The parameter update is carried out according to the particular PS implementation. These iterations continues until convergence, resulting in a final policy $\boldsymbol{w}^{final}$. As a result, this combination enables for joint skill learning, where the robot can start with a blank policy, and human exploration is added whenever the human judges his/her knowledge on the task can benefit the robot learning.

Thus, during the roll-outs selected to be advised by the user, any type of COACH like Algorithms 5.1 or 5.2 is run, but taking into account that the vector $\boldsymbol{w}$ is loaded into a $\boldsymbol{w}_t$ vector every time step in order to have all the changes in memory, since the evaluation stage (Algorithm 4.1, line 3) can be step-based depending on the PS algorithm used as baseline [32].

Algorithm 5.3 is a high level description of the proposed strategy for complementing the PS with human advice during the exploration stage. Similar to Algorithm 4.1, the Explore,

Algorithm 5.2: COACH for training a MP in the joint space.

```
1:  $e \leftarrow constant$
2:  $\beta \leftarrow constant$
3:  for $1 \leq t \leq n$
4:    $c_t \leftarrow assignCredit(t)$
5:  end for
6:  for all $z_t$ do
7:    $g_t \leftarrow getBasisFunctions(z_t)$
8:    $q_t \leftarrow g_t^T w$
9:    $takeAction(q_t)$
10:   wait for next time step
11:   $h \leftarrow getHumanCorrectiveAdvice()$
12:   if $h =! 0$
13:     for $1 \leq t \leq n$
14:       $\Phi^{cred} \leftarrow \Phi^{cred} + (c_t \cdot g_t)$
15:     end for
16:     $H(z_t) \leftarrow {\Phi^{cred}}^T v$
17:     $q_t \leftarrow {\Phi^{cred}}^T w$
18:     $\Delta v \leftarrow \beta \cdot \big(h - H(z_t)\big) \cdot \Phi^{cred}$
        $v \leftarrow v + \Delta v$
19:     $\alpha(z_t) \leftarrow \mid H(z_t) \mid + bias = \mid {\Phi^{cred}}^T v \mid + bias$
20:     $error_X \leftarrow h \cdot e \cdot \alpha(z_t)$
21:     $\widetilde{X_t} \leftarrow FK(q_t) + error_X$
22:     $\widetilde{q_t} \leftarrow IK(\widetilde{X_t}, q_t)$
23:     $error_q \leftarrow \widetilde{q_t} - q_t$
24:     $\Delta w \leftarrow error_q \cdot \Phi^{cred}$
        $w \leftarrow w + \Delta w$
25:   end if
26: end for
```

Algorithm 5.3: Policy Search with Simultaneous Human Guidance.

```
1:  repeat
2:      Explore: first roll-out with the current policy $\pi_k$: $w_1 = w^{[k]}$
3:          for $m = 1 \dots M$
4:              if (HumanGuidance==True)
5:                  $[w_m]_t \leftarrow RunRollOut_{COACH}([w_m]_1)$
6:                  $[w_{m+1}]_1 = [w_m]_T$
7:              else
8                   $[w_m]_t \leftarrow PS\_exploration(w^{[k]})$
9:                  $RunRollOut([w_m]_t)$
10:             end
11:         end
12:     Evaluate: cost of each roll-out $[R_m]_t = \phi_T + \sum_{t=0}^{T} r_t^{[m]}$
13:     Update: Compute new policy parameters using
                    $w^{[k+1]} \leftarrow Update([w_m]_t, [R_m]_t)$
14: until Policy converges $\pi_{k+1} \approx \pi_k$
```

Evaluate, and Update stages are run every *k-th* iteration, but with an important difference in the exploration process. There are 2 exploration modes: the original exploration strategy of the PS method, and the exploration based on COACH. The user chooses one of them through the flag *HumanGuidance* (line 4). S/he would choose the COACH based exploration when considers that is necessary and possible to advise the agent, then the roll-out using COACH is run (line 5), otherwise, the teacher allows the random exploration of the original PS (lines 8-9).

The flag *HumanGuidance* can be switched differently depending on the algorithm implementation. For instance, the human-machine interface can query it before every roll-out execution. Particularly, for the implementations of this work, this flag is set *False* by default and switched on when the user advises a correction; if during a roll-out the user does not give corrective feedback, the flag is set *False* for the following roll-out.

During each iteration of the PS algorithm, M roll-outs of T time steps are carried out; the vector $\boldsymbol{w}^{[1]}$ is the same $\boldsymbol{w}^{[init]}$ in Figure 4.2. In $[\boldsymbol{w}_m]_t$ is contained the parameters vector $\boldsymbol{w}$ at time step t of the *m-th* roll-out. For the roll-outs in which the user is giving corrective feedback (*HumanGuidance*==True), the initial parameters vector used for the *m+1-th* roll-out (line 6) is the last one resulting from the immediate previous roll-out, i.e. $[\boldsymbol{w}_m]_T$. This is in order to keep the same policy that is being incrementally advised by the teacher. The rest of the algorithm follows the regular PS scheme.

5.5 Experiments and Results

The use of human corrective feedback is validated first for shaping trajectories without the use of PS. Thereafter, more complex experiments are presented to compare the proposed Interactive PS with respect to conventional PS.

5.5.1 Learning movement primitives with corrective advice

Experiments were carried out exclusively for evaluating the use of COACH for training movement primitives. As a proof-of-concept, we proposed the problem of teaching a robot how to write letters, in simulation and using a real robot. The objective in this experiment is to evaluate improvements that can be made on the shape of a trajectory (encoded as a movement primitive) via corrective advice. This improvement is quantified against the original set of points that compose a letter, used as ground truth. Two different approaches were evaluated: policy refinement, and policy reuse. In the first case, the goal is to improve the shape of a given letter. The latter case addresses an application of transfer learning, where the goal is to reuse one of the existing policies, and reshape it via corrective advice to fit a new desired symbol.

The procedure consisted of an initial stage, where a user interface was used to visually indicate on a screen a reference letter to be drawn. The user then provided a set of demonstrations of trajectories for each indicated letter. A RGB camera captured the user's pen movement and recorded the whole path into a dataset, which was used to train an initial Cartesian policy, parametrized as Equation (5.3). Figure 5.4 shows the screen of the interface for recording demonstrations. The top figure shows one instance of human demonstration. The bottom figure shows the reference symbol (in blue) overlapped with one of the provided demonstrations (in red). In the attached video[1] is shown the interface while is recording demonstrations.

In a second stage, the user attempted to refine the policy resulting from the first stage. Two different interactive approaches were used for correcting the policy: (i) corrections with more demonstrations and (ii) corrections with COACH. To quantify the performance of both strategies, the learned symbols are compared with the ground truth symbols using euclidean distance after alignment with Dynamic Time Warping.

The experiments were carried out both in simulation, with a 3-DoF robot arm, and with a real UR5 robot with 6-DoFs. Figure 5.5 shows the simulated case where the robotic arm draws the learned symbol (in red), while the teacher provides corrective feedback to correct the trajectory towards the ground truth reference (in blue). In each experiment, five participants demonstrated and corrected the robot primitive. Each user took a first session of practice to become habituated to the interaction with the recording and corrective interfaces.

[1]https://www.dropbox.com/s/rrv1muavrqgtgbb/demos.mpeg?dl=0

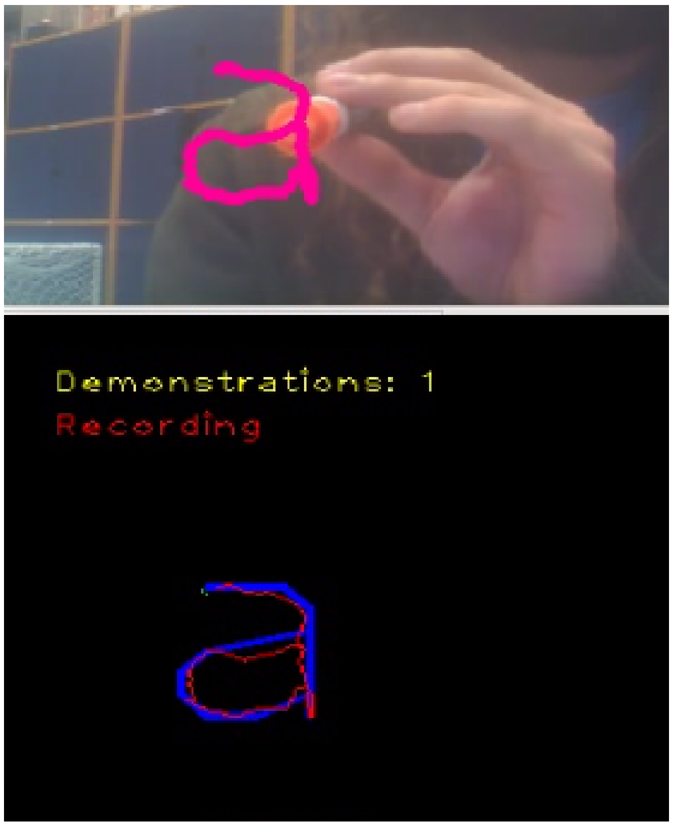


Figure 5.4: Demonstrations recording.

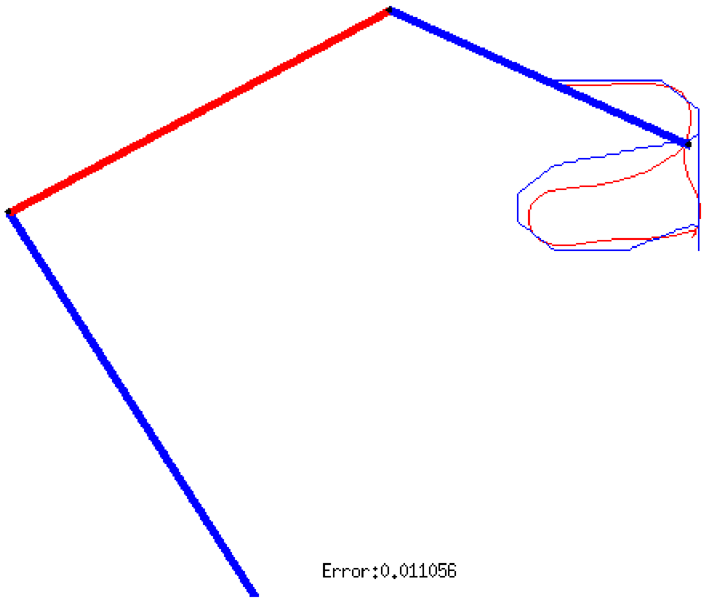


Figure 5.5: Policy execution and correction with the simulated 3-DoF arm. Human feedback was used to make the robot draw as close as possible to the reference letter (in blue). The initial demonstration is shown in red.

Experiments of Policy Refinement

In this experiment, a set of six symbols was learned (letters: a,c,I,m,p,s). The objective was to improve and refine the trajectories learned from demonstrations. For each symbol, three policies derived out from the first set of demonstration are compared.

-**Primitives resulting from the original demonstrations:** For each letter, a policy was learned using five demonstrated trajectories.

-**Primitives resulting from corrections with COACH:** During five sequential executions, the users observed the policy execution resulting from the five demonstrations and simultaneously interacted to provide corrections using COACH. The corrections were relative to the original policy in the Cartesian space. The users used a keyboard with two keys for correcting along the x axis, and two keys for correcting along the y axis. The users advised corrections to make the end-effector to pass closer to the reference symbol. Since the policy is learned in the Cartesian space, the COACH version applied to this problem is the one presented in Algorithm 5.1.

-**Primitives resulting from corrections with more demonstrations:** The users observed the initial policy from the 5 demonstrations, and provided five additional demonstrations for improvement.

Results of Learning with a Simulated Arm

The average learning curves for all the symbols are plotted on Figure 5.6. In the cases of learning only with demonstrations, the figure plots the final error of the resulting policies from the datasets. Since the symbols used as reference are sets of points without physical dimensions, the demonstrations recorded in the pixels space are normalized, so the error measurements do not have units.

It is possible to see that correcting the trajectory with COACH, shows that providing corrective feedback during 5 repetitions of the trajectory, the error is decreased by 79.83%. On the other hand, the strategy of correcting with more demonstrations only obtained around 40% error reduction with respect to the achieved by the primitives learned with the first demonstrations dataset. The aforementioned reductions mean that with the same amount of trials (five new demonstrations vs. five episodes of correction with COACH), with corrective feedback, a human teacher can achieve almost twice the error reduction with respect to the strategy of recording more demonstrations.

Moreover, from the learning curves it is possible to highlight, that with only one episode of corrective advice with COACH, the human teacher can attain better improvement than the reached with the new set of five demonstrations. After the first episode of advising corrections, the average error of the trajectories is decreased 73%. In the first episode is wherein most of the improvement is achieved. This observation is not only taken from the learning curves, but also from the appearance of the learned letters. For instance, in Figure 5.7 is printed the progress of improving a trajectory with corrective advice, where most of

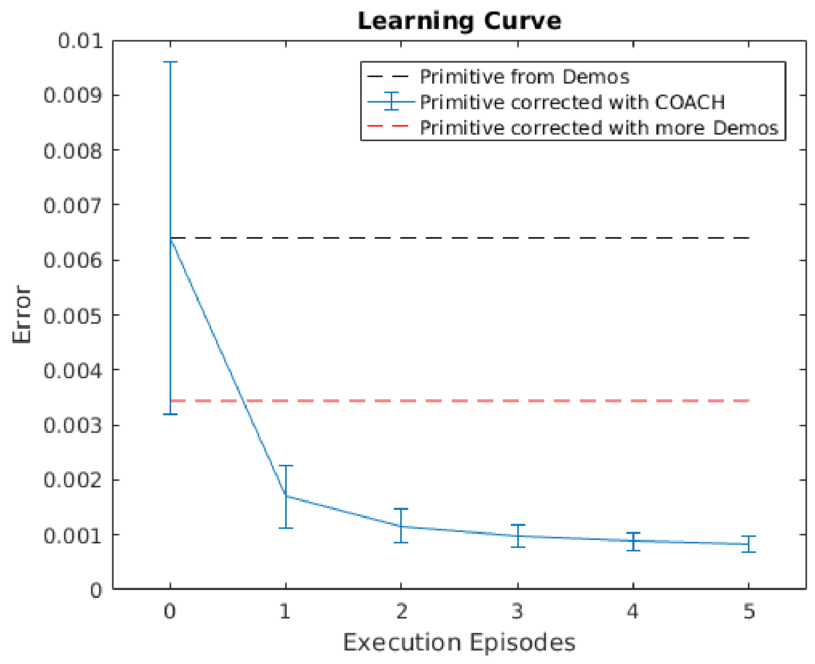


Figure 5.6: Average error during learning of the symbols.

the change obtained from the corrections is done in the first episode.

Table 5.1 shows specifically for each of the six explored letters, the final errors obtained after correcting the policies with COACH and with more demonstrations. Basically, the trends of the average results are kept, only an anomaly is highlighted: In the case of the letter "p", after the process of correction by recording more demonstrations, the error was increased 2.41%. A correcting session with the simulated arm is shown in the video[2].

[2]https://www.dropbox.com/s/f60xv60d0kmgvch/corrections.mpeg?dl=0

Table 5.1: Average error for each symbol trained. The error is multiplied by 10^{-2}

Symbol	**Demos**	**Corrections (COACH)**		**Corrections (more Demos)**	
	Error	Error	% Decreased error	Error	% Decreased error
a	1.322	0.115	91.27	0.638	51.75
c	0.709	0.085	88.05	0.338	52.29
I	0.125	0.033	73.50	0.041	66.93
m	0.878	0.088	89.93	0.458	47.85
p	0.182	0.089	51.05	0.187	-2.41
s	0.432	0.064	85.17	0.300	30.48

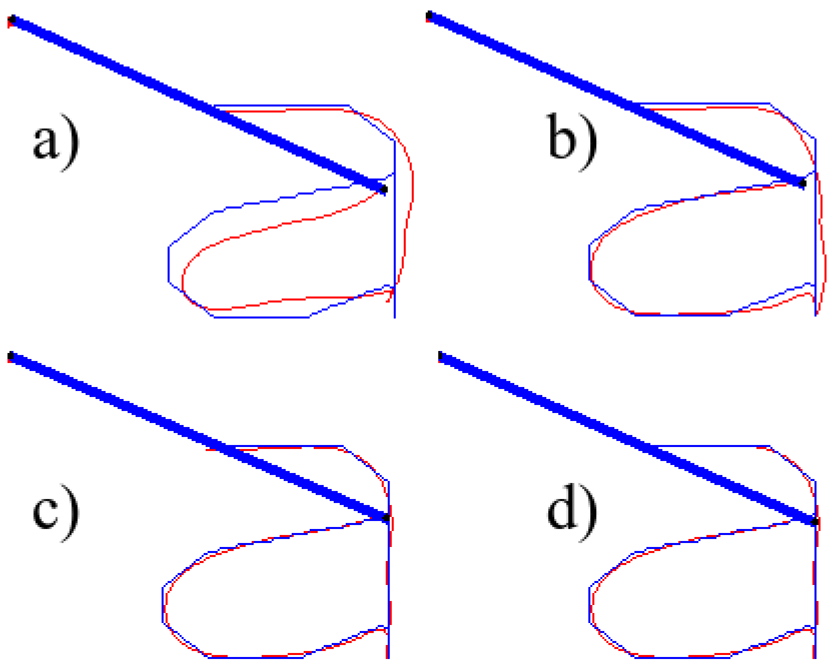


Figure 5.7: Trajectory correction progress: Learned Primitive (red), symbol of reference (blue). a) Policy without corrections, b) after 1 episode of corrections, c) after 5 episodes, d) after 8 episodes.

Results of Learning with a Real Arm

The previous experiments were replicated using a real UR5 arm. The same symbols and comparisons are used in this case. However different participants from the previous experiment played the teacher role. The points composing the drawn trajectories are obtained from the robot's odometry, and compared to the reference symbol for the error calculation.

Examples of corrected symbols are shown in Figure 5.8, where the initial policy obtained from demonstrations is drawn with white color, while in red color is the final trajectory after 5 episodes of correction. In Figure 5.9 are shown the average curves of learning and correcting with demonstrations in contrast with the learning curve of correcting with COACH. The error reduction obtained using more demonstrations is in average 30.7% and around 84.4% when using COACH. In this case, the error reduction with the first episode of correction is 53.8%, which again is higher than the one achieved with five new demonstrations.

The final results per symbol are listed in Table 5.2. These average errors have similar trends regarding the results with the simulated arm. The lowest error reduction is with the symbol "p", that is still higher than the average of the error reductions resulting from the strategy of correcting with more demonstrations. In general, these results are consistent with the previous ones. This lets to conclude that the use of corrective advice to shape trajectories is a good strategy for learning agents from human teachers, especially in situations wherein the combination of user expertise and quality of the user interface does not obtain the best conditions for recording high performance demonstrations. The results show that detailed trajectories can be shaped easily only using vague binary corrective feedback. A correcting

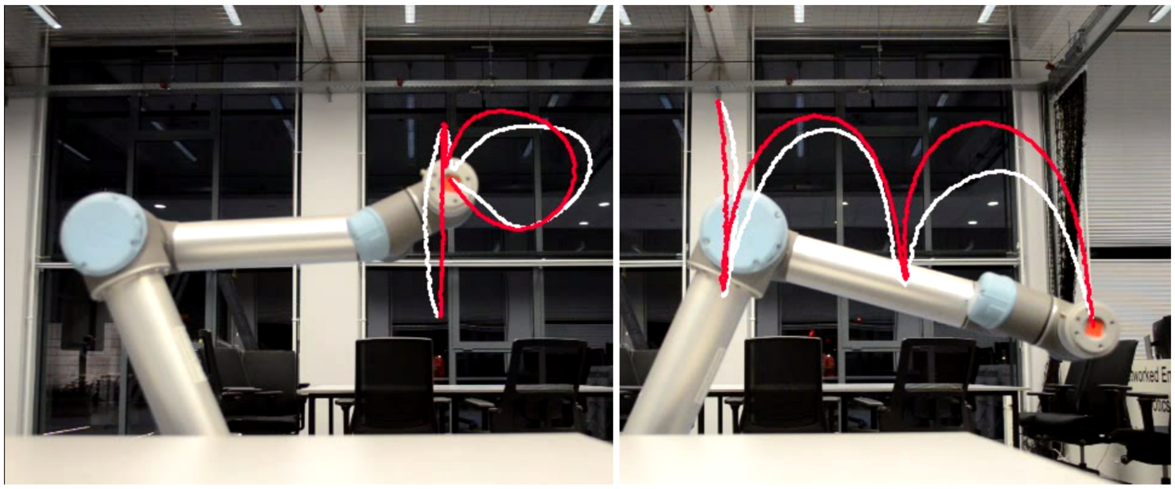


Figure 5.8: Examples of learning the letters "p" and "m" with the real robot. Policy derived from demonstrations (white), and policy trained with corrective advice (red).

Table 5.2: Average error for each symbol trained using the real robot. The error is multiplied by 10^{-2}

Symbol	**Demos**	**Corrections (COACH)**		**Corrections (more Demos)**	
	Error	Error	% Decreased error	Error	% Decreased error
a	1.483	0.158	89.35	1.151	22.45
c	0.842	0.115	86.34	0.401	52.38
I	0.284	0.096	66.20	0.182	36.62
m	1.287	0.127	90.13	0.967	24.86
p	0.174	0.109	37.36	0.159	8.62
s	1.206	0.216	82.09	0.796	34.00

session with the real UR5 arm is shown in the video[3].

Experiments of Transfer Learning for Policy Reuse with a Simulated Arm

In policy reuse, the user can provide corrections to a primitive whenever a task is changed, or when the task has to be performed in a new environment. There can be cases in which recording demonstrations can be complicated due to different reasons, e.g. 1) the absence of an expert user in the task, who is able to provide high-quality demonstrations that lead to a policy with acceptable performance; 2) when the final user does not have an available interface to provide new demonstrations like wearable sensors or complex vision systems, 3) when Kinesthetic teaching is not possible, since it is constrained to robots with physical dimensions

[3]https://www.dropbox.com/s/nwnzrkdoic5eurn/ur5shorttitle.mp4?dl=0

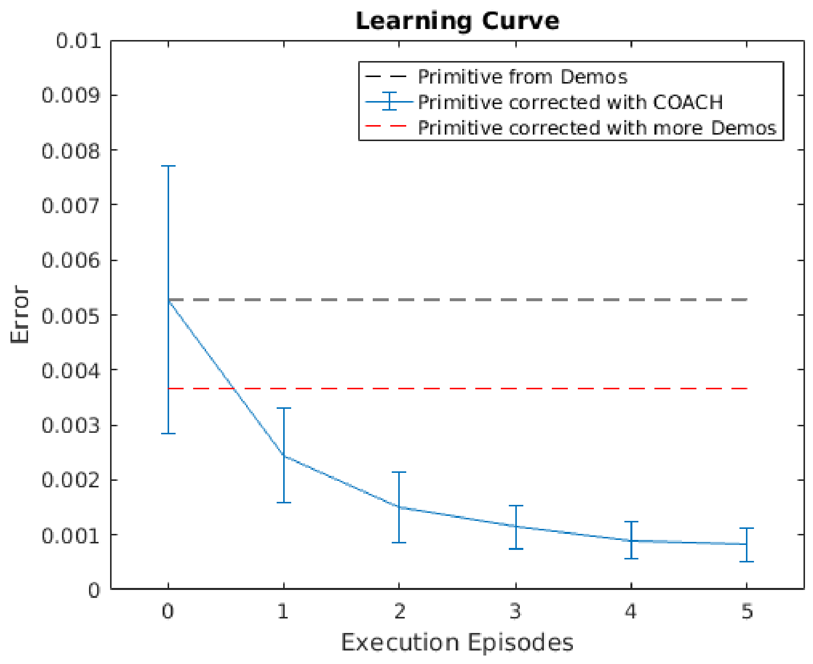


Figure 5.9: Average error during learning to write with a real arm.

that a human teacher can handle. For some of those cases the knowledge already represented by the primitive can be reused, then the user only needs to execute local modifications for the points of the trajectory that need to be fixed for the new conditions. This previous discussion motivates the evaluation approach of policy reuse that is presented after the results of policy refinement with a simulated robotic arm.

Here, users had to correct and improve a trajectory to reduce the error between the path printed by the simulated arm and the reference symbol taken as ground truth. In contrast to the previous experiments of policy refinement, in this case, the initial policy corresponds to a symbol that is different from the desired one, resulting in larger initial errors.

Two symbols were explored for evaluating COACH for policy reuse. First a primitive for the letter "z" was used as initial policy for the task of drawing a "2". The second was a "V" used for drawing an "U".

In Figure 5.10, it is possible to see in the left hand side, the reference letter (blue) used for recording the demonstrations for the initial policy derivation, and subsequently the corrective process with COACH that obtained the final policy (red). In the right hand side is shown the same final policy for the symbol "z" (red), which is used as the initial policy for learning the symbol "2", and its baseline (blue).

In this experiment the user has to provide the corrective feedback during 10 episodes of the path execution. In Figure 5.11 it is possible to see the evolution of the error through the episodes of correction during policy execution. Before the correction of the trajectories, the average error was 0.1717. Unlike the policy refinement experiments, in this case at the first episode, the users advised large changes of the policy that decreased about 90% of the initial error. By the fourth episode, the corrections obtained 99% of error reduction.

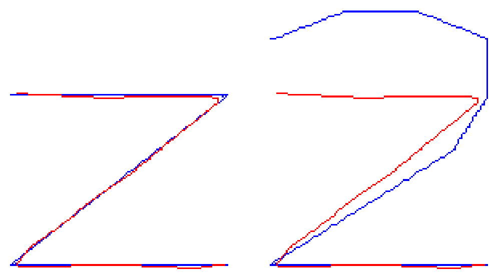


Figure 5.10: Initial trajectory for the transfer learning process: from "z" to "2".

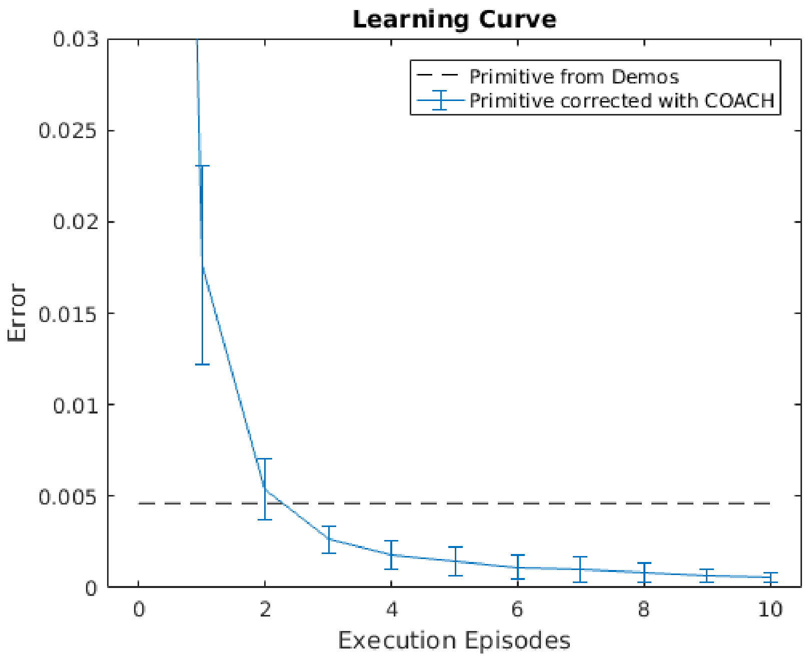


Figure 5.11: Policy reuse, average error for learning symbols.

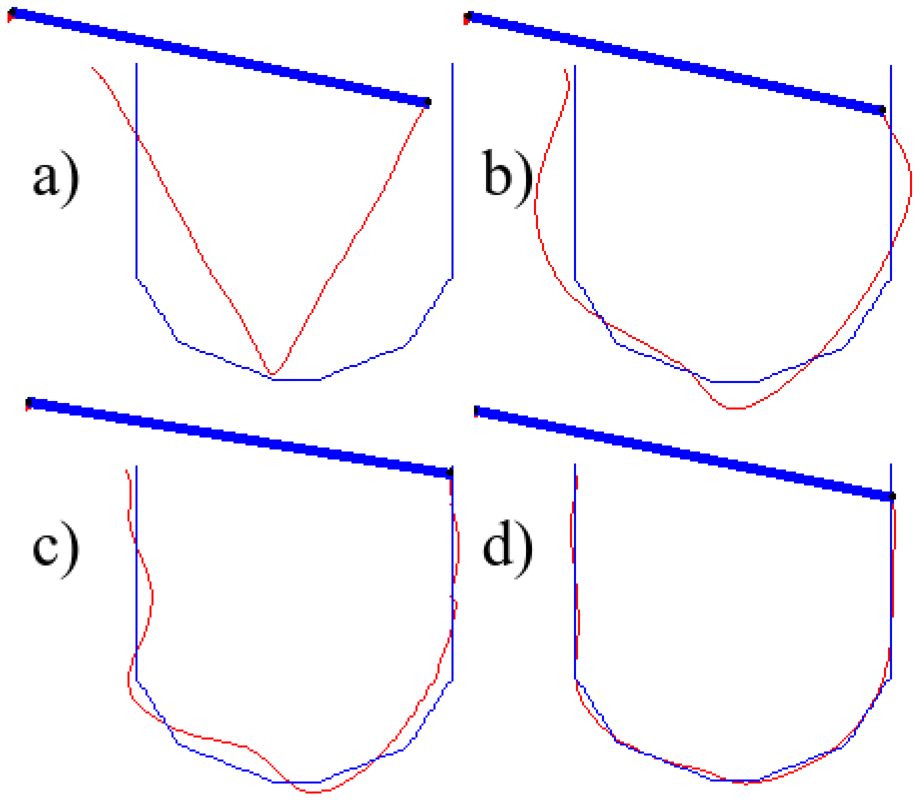


Figure 5.12: Trajectory correction progress for policy reuse: Learned Primitive (red), symbol of reference (blue). a) Policy without corrections, b) after 1 episode of corrections, c) after 5 episodes, d) after 10 episodes.

Moreover, it is possible to observe that by the third episode of corrections, the error was reduced to a level lower than the obtained by a policy derived out from datasets of five demonstrations; by the fifth episode, the percentage of error reduction was around 68%, also with respect to the policy obtained from demonstrations, which is a similar error obtained in the experiments of policy refinement. In Figure 5.12 is depicted the progress of the shape drawn for the symbol "U", where the initial policy is a "V".

5.5.2 Learning with Simultaneous Corrective Advice and Policy Search

So far, trajectories were optimized purely on human feedback with COACH. In this section, we validate the combination of PS with the human-guidance-based exploration using COACH in well-known problems in simulation and with a real robot.

In simulation, experiments were carried out using an arm with varying degrees-of-freedom in a reaching via-point task. In a second set of experiments, the "ball in the cup" task was learned using a real robot. In both tasks, we compared the proposed interactive PS strategy with a standard PS algorithm in terms of the convergence rate and final performance of the policies. Although the proposed hybrid method can be implemented with different PS algorithms, in this experimental procedure both the standard and the hybrid PS are based on PI^2 [101].

Learning Multi-DoF Via-Point Movement Tasks

The first set of experiments for the validation of the hybrid approach is carried out by replicating the experiments intended to compare PS algorithms in [101], and that has been repeated in [90, 92]. The experiment consisted of learning robot arm reaching movements (similar to human reaching movements) with a total duration of 0.5 seconds. The task has the condition of reaching a specific via-point at $t = 0.3s$, which is an approximation to hitting movements, because they require time-space synchronization.

The learning task is evaluated in four different cases: first, with a one-dimensional moving point (1 DoF); the next three cases are with planar arms of 2, 10, and 50 degrees-of-freedom (DoF). The policies are represented with DMPs, that compute the actions in the joint space for the multi-DoF tasks. The experiments were executed first with the original PS algorithm PI^2, and followed by our hybrid approach combining PS and the COACH variation for policies defined in the joint space (Algorithm 5.2). For every explored case, 20 runs of 500 roll-outs were executed for each of the algorithms. The obtained results are averaged and presented with their standard deviation.

1 DoF Via-Point Task: In this task the initial position of the movement is $y_{t0} = 0$, and the DMP has the goal atractor $g = 1$ in order to finish the movement with $y_{t500ms} \approx 1$. The cost function is $r_t = 0$ for all time steps except in $t = 300ms$, as shown in Equation (5.6), where G is the via-point set to $G = 0.25$.

$$r_{300ms} = 10^8 (G - y_{t300ms})^2 \tag{5.6}$$

When the user participates in the learning process, s/he observes the movement execution and advises the binary correction with a keyboard, similarly to the interaction in the experiments of learning to write letters, but only using 2 keys.

In Figure 5.13 the evolution of the cost function through the roll-outs execution is shown. The human feedback supporting the PS improvement makes a significant difference regarding the original PS algorithm. The convergence time is reduced one order of magnitude, the interactive PS method is about 83% faster than the conventional PS. Moreover, it is possible to see that the variance of the cost function is decreased with the human guidance.

Multi-DoF Via-Point Tasks: In these cases of simulated planar arms, the initial position is $a = 0$ for all the joint angles, it sets a robot pose that is a straight line parallel to the horizontal axis. The goal attractor x_{goal} makes the movement to finish in a semicircle configuration, as shown in Figure 5.14, where the end effector of a 10 DoF arm touches the y axis. The end effector is moving in the 2-D space, and has to pass through the via-point $G = (0.5, 0.5)$.

Figure 5.14 a) shows the trajectory of the arm with the initial policy of the learning

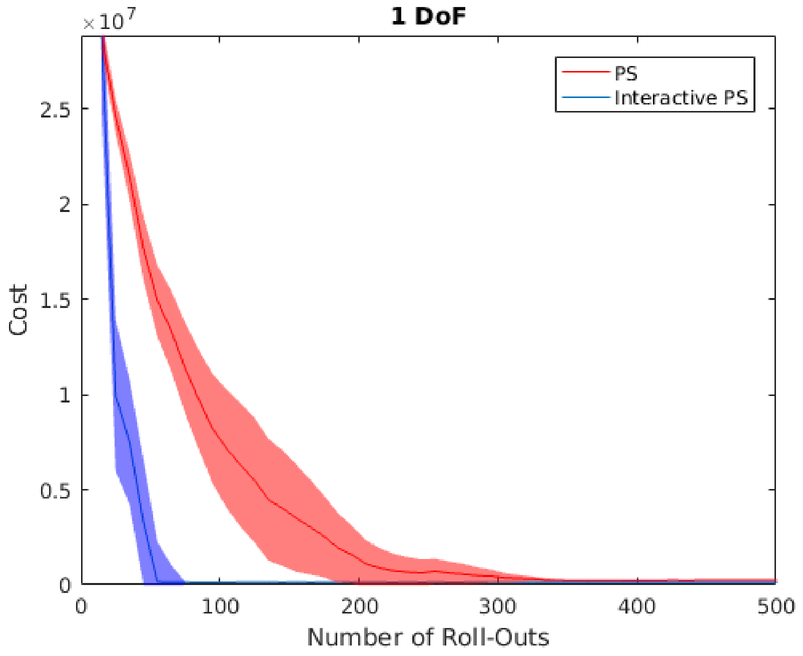


Figure 5.13: Learning Curve of the 1 DoF via-point movement task.

process. In part b) the end effector is already passing through the via point.

The objective of the cost function is to reduce the distance between the end effector and the via-point at $t = 300ms$. Additionally, it also tries to reduce joints accelerations, giving more priority to the joints that are closer to the “shoulder" of the arm, with D the number of DoF of the robotic arm, as shown in Equation (5.7).

$$\begin{aligned} r_t = 10^8 \delta(t - 300ms) \cdot ((x_t - 0.5)^2 + (y_t - 0.5)^2) \\ + \frac{\sum_{\mathrm{d}=1}^{D} (D + 1 - \mathrm{d})(\ddot{a}_{\mathrm{d},t})^2}{\sum_{\mathrm{d}=1}^{D} (D + 1 - \mathrm{d})} \end{aligned} \tag{5.7}$$

During the interactive learning process, the user advises the corrections with binary corrections in both axes, as it was done for correcting the letters in the previous subsection. In previous works these experiments have been carried out for learning policies in the joint space domain, nevertheless, in this thesis we approach the problem both in the Cartesian and the joint domain.

The learning curves in Figure 5.15-5.17 show the improvement achieved when PS considers the human corrective feedback. The general trend shown in the curves is that the hybrid agents converge faster than standard PS. In the cases of 10 and 50 DoF, the standard PS learns faster in the Cartesian domain than in the joint space due to the smaller search space. In contrast, the interactive PS learns faster when learning policies in the joint level with respect to policies represented in the end effector domain.

Since the original problem is only explored with policies in the joint space, and also because

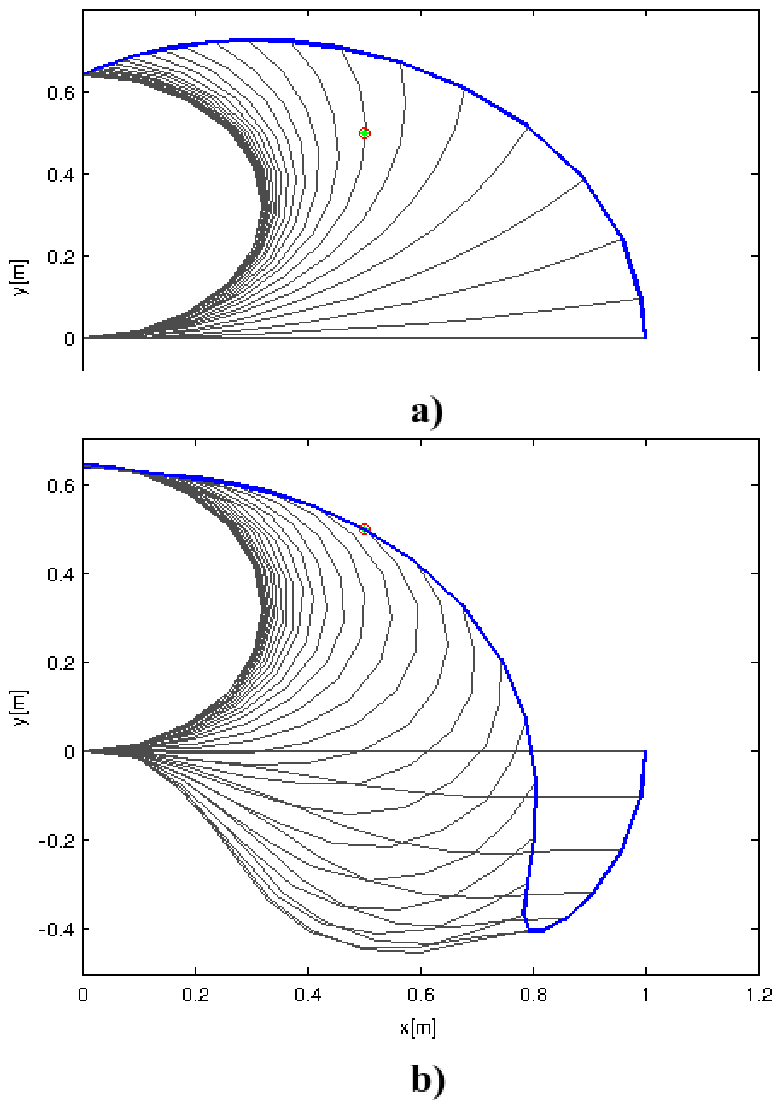


Figure 5.14: "Stroboscopic" visualization of the 10 DoF planar robot arm movement, a) simply towards the goal, b) through the via-point (green/red dot).

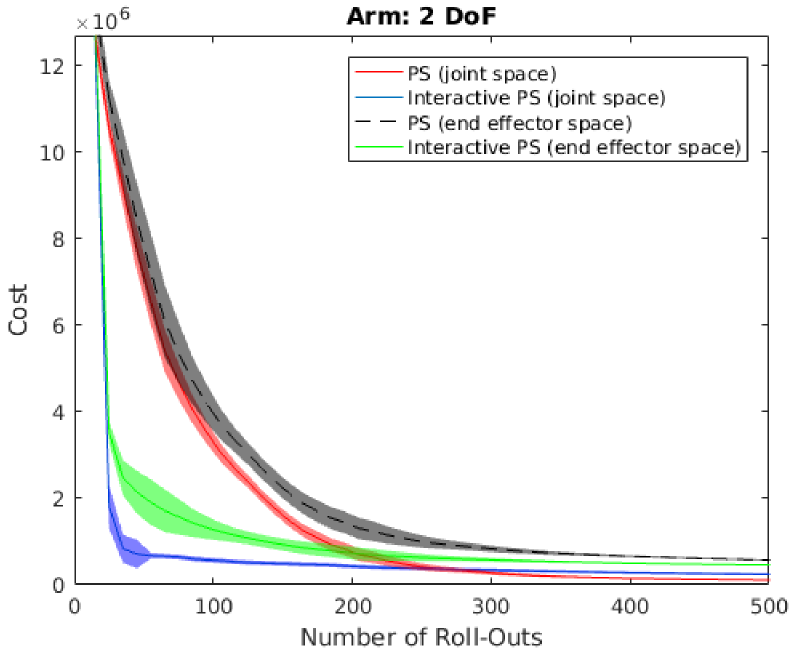


Figure 5.15: Learning Curve of the 2 DoF via-point movement task. Average and ± 1 standard deviation of 20 runs.

the best obtained policies are in that domain, below, the rest of the analysis is only focused in the comparison between standard and interactive PS for learning policies in the joint space.

For the task with the 2 DoF arm the interactive PS decreases 95% of the initial cost within the first 50 roll-outs, and keeps a slight rate of improvement, reaching 97.9% by the 500 trials. In contrast, the conventional PS attains the 95% of cost reduction approximately after 210 trials, i.e., it is 4 times slower than the interactive PS. However, the conventional PS keeps the error reduction, and outperforms the performance of the interactive PS after 280 episodes, reaching a total reduction of 99.2% with 500 episodes.

In the experiments with the 10 DoF arm, results are similar, but with a bigger difference between the cost of both algorithms. The 95% of reduction is obtained after 30 trials with the interactive PS, whereas the conventional PS is 11 times slower for achieving that performance and by the 500 roll-outs reaches the curve of the interactive PS.

For the last case with 50 DoF, again the difference is increased a lot, as 500 roll-outs are not enough for the PS agent to converge in this problem. Then, at the end of the learning process, the PS agent only decreased 86.9% of the initial cost. On the other hand, the interactive PS achieved the 95% of reduction by approximately 25 episodes and converged completely after the 60-th.

In the previously presented results, the convergence of PS is affected when the number of DoF is incremented, due to the curse of dimensionality. But the convergences of the proposed interactive PS show a counterintuitive trend after the increment of the DoF. Indeed, the fastest convergence obtained with the human feedback is in the case of the arm with 50 DoF. The reason behind this effect has to do with the correspondence problem between the

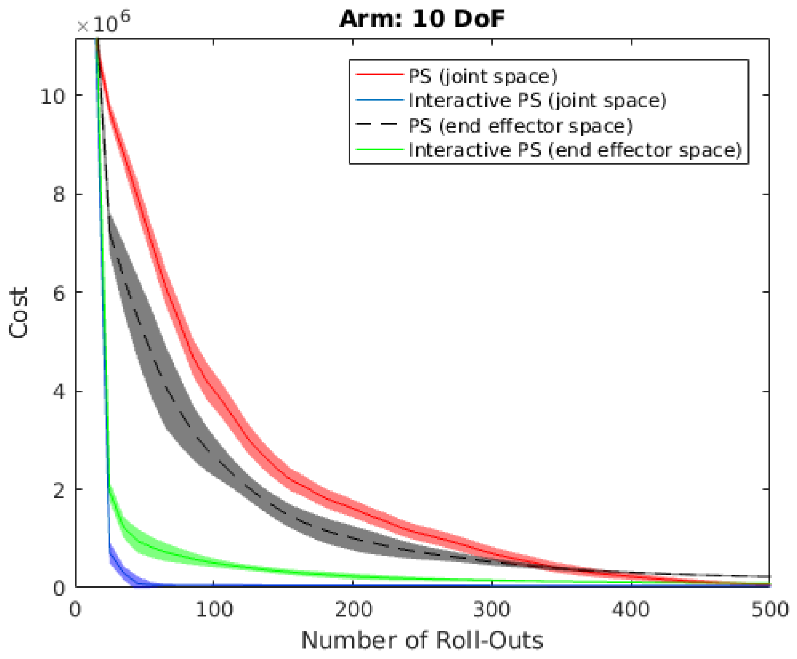


Figure 5.16: Learning Curve of the 10 DoF via-point movement task. Average and $\pm$ 1 standard deviation of 20 runs.

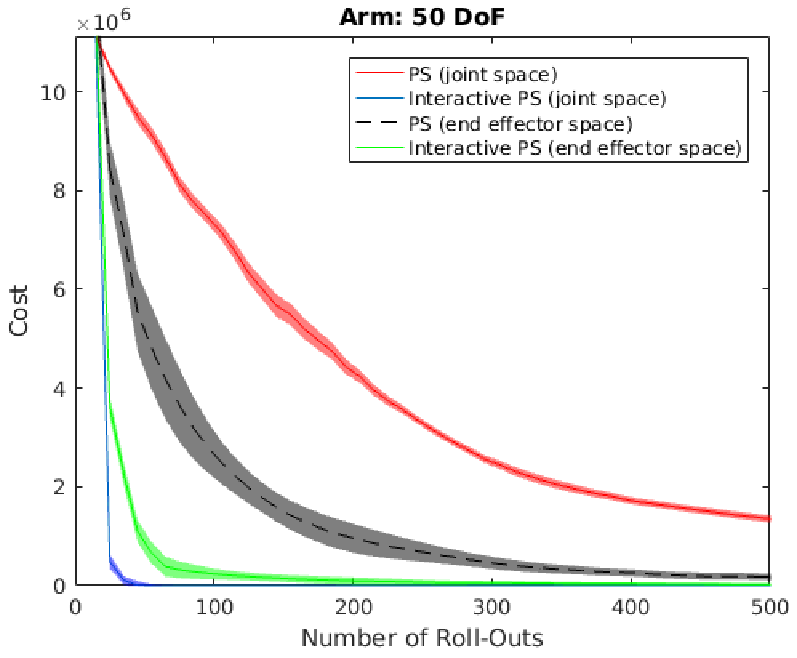


Figure 5.17: Learning Curve of the 50 DoF via-point movement task. Average and $\pm$ 1 standard deviation of 20 runs.

corrective advice in the effector domain, that is mapped to the joints space, where the policy is defined. When the human teacher advises a correction to the 2 DoF arm, in several cases the solution found by the IK could be a joint configuration very different from the previous one, or simply it cannot find a proper solution that matches with the end effector position correction.

These problems cause policy updates that do not match with the user intention, therefore the teacher's correction may harm the policy from time to time. Nevertheless, when the task has more DoF this problem diminishes. The more redundant is the arm is, the easier it is to shape the end effector trajectory with corrective advice. Although for the case of the 2 DoF arm the corrective advice does not work perfectly, still the PS benefits strongly from the human guidance and reduces the convergence time by 76% when compared to the conventional PS. .

Ball in a Cup

The *ball in a cup* is a challenging children's game that requires accurate skills in a relatively fast movement. The game uses a toy composed of a ball attached to a cup with a string. The cup is held with the hand of the player, or attached to the end effector of the robot in this case. Initially, the ball is hanging steady below the cup, and the arm has to move the cup fast enough to launch the ball high in the air to catch it during the landing.

A reward function that represents the task objective would be one that punishes a failed trial, and rewards when the ball falls into the cup. However, such function is not informative for efficient (or even feasible) learning in a reinforcement learning setting. This problem was approached in [59] with a more complex function of the form

$$r_t = \begin{cases} exp(-\alpha(x_c - x_b)^2 - \alpha(y_c - y_b)^2) & \text{if } t = t_c, \\ 0 & \text{otherwise,} \end{cases} \tag{5.8}$$

using the PS algorithm PoWER on a real robotic arm Barrett WAM$^{\text{TM}}$.

In the reward function above, the ball and cup positions are $[x_b, y_b, z_b]$ and $[x_c, y_c, z_c]$, respectively. The time $t = t_c$ is the moment when the ball passes the rim of the cup with downward direction, for all the other time steps $t \neq t_c$ the function is $r_t = 0$. This reward function was used by the PS method to improve an initial policy obtained from a human demonstration.

In this work, we validate the proposed interactive PS method with this problem using a 7 DoF KUKA lightweight arm, and an OptiTrack system which tracks the position of the ball and the cup, for computing the reward function. Nevertheless, there are two important differences with respect to the work of Kober et al. [59]. First, here the policy computes the trajectory of the end-effector instead of computing actions in the joint space; secondly, we consider not only to improve policies learned from human demonstrations, but also to learn the policies from scratch, therefore the reward function of Equation (5.8) is complemented in order to make it more informative.

When the arm tosses the ball with a height lower than the cup, Equation (5.8) is completely uninformative as it always results in a zero. In order to lead the policy to a behavior wherein (5.8) is applicable, we propose to complement this function with a term that rewards the height obtained in those cases.

Then, when the ball does not reach the height of the cup, the term (5.9) is applied:

$$r_{t_h} = \frac{z_b - z_c}{l_s + l_c} \tag{5.9}$$

With l_c and l_s denoting the length of the cup and the string respectively, and $t = t_h$ is the moment when the ball reaches the maximum height. The sum of l_c and l_s is the distance between the ball and the rim of the cup when the ball is hanging motionless, like in Figure 5.18.

With this extension, the task "ball in a cup" can be considered a composition of the sub-task "swinging the ball" with the objective of tossing the ball higher than the cup, followed by the sub-task "catching the ball" that aims to move the arm for intercepting the ball with the cup. For the computations of the learning algorithms, the reward function is transformed into a cost function multiplying it by -1.

Different from previous works that use DMPs for this problem, we opted to represent the policy with the ProMP form of (5.3), like in the experiments of Section 5.5.1, since the convergence to goal attractors is not a necessary property for this task. Again PI^2 is the base PS algorithm for these experiments.

In the experimental procedure, for the learning processes, cups of two different sizes are used in order to change the difficulty of the task. In Figure 5.18 the robot arm is attached with a stick to the big cup (approximately twice the diameter of the ball), which is also containing the small cup (diameter approximately 1.3 times the ball's diameter). During the learning processes, the human teacher is sitting in front of the robot with a perspective similar to Figure 5.18.

Most of the validation experiments are carried out with the big cup. Standard PS is compared to learning with human feedback in the interactive PS approach, along with the pure COACH method, executing ten runs for each approach. Since the learned policy computes actions in the end effector domain, the two interactive methods are based on the complete COACH for training motor primitives in the Cartesian space (Algorithm 5.1). A keyboard is used for advising the corrections by the user.

The first set of experiments starts with a policy derived from a kinesthetic demonstration and using the big cup. The learning curves presented in Figure 5.19 show the big difference between learning with PS and the interactive methods. PS achieves policies that catch the ball with the cup around the 60th trial, and keep improving until convergence after 90 trials. In contrast, with COACH and the interactive PS the task is achieved after 10 and 15 episodes respectively. With COACH the improvement stops very soon, because when the human teacher observes a successful policy, s/he reduces the effort for enhancing it, due to

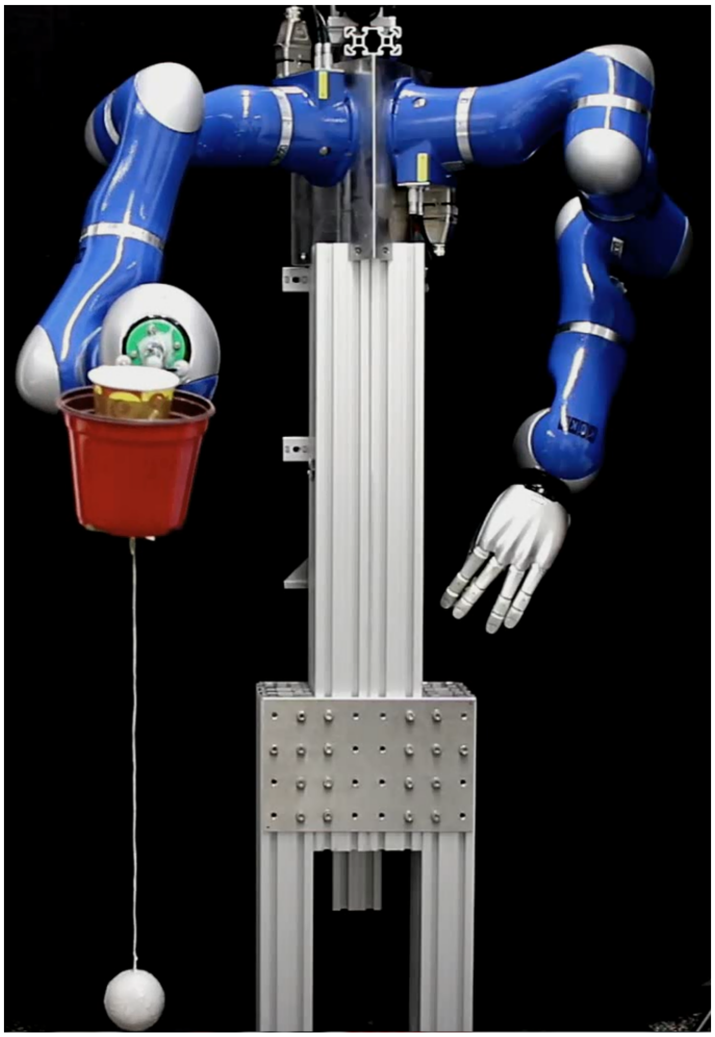


Figure 5.18: Robot set-up for learning the "ball in a cup" task.

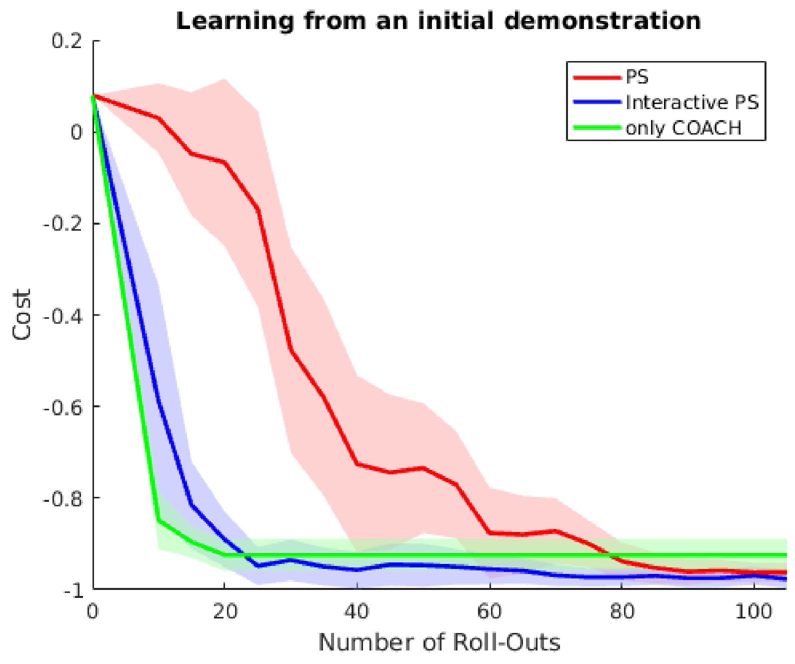


Figure 5.19: Convergence curves for learning "ball in a cup" with an initial demonstration.

the fact that it is not very evident and/or necessary. For human teachers is hard to infer corrections when sub-optimal policies are close to the optimal.

On the other hand, at the beginning, the interactive PS is slightly slower than pure COACH due to the influence of some of the first roll-outs in the update process. However, with the hybrid method, the improvement continues until the 70^{th} episode reaching the best average performance. This improvement results from both sources of feedback, especially from the reward function.

The learning curve of the PS reaches and outperforms the cost obtained with COACH within 75-80 trials, i.e., it needs 4 times more trials. However, with 100 trials, the PS does not attain the performance obtained by the interactive PS.

For a second set of experiments, a more challenging scenario is used for testing the learning algorithms, wherein a previous demonstration is not given by the human teacher (learning from scratch). Therefore, a policy that does not request any movement is set at the beginning of the learning process.

Since the arm is not able to toss the ball to the necessary height in one shot, the robot needs to learn the “swinging the ball" sub-task, so that it oscillates like a pendulum and obtains enough momentum. In that first part of the learning process, the term (5.9) of the reward function plays an important role. When the policy evolves maximizing (5.9), it continues learning to catch the ball with the cup using (5.8).

The results of this experiments are shown in Figure 5.20. It depicts that these interactive methods based on vague corrective advice are more robust to the initial policy than the standard PS. Both learning curves of the corrective advice based methods are basically the

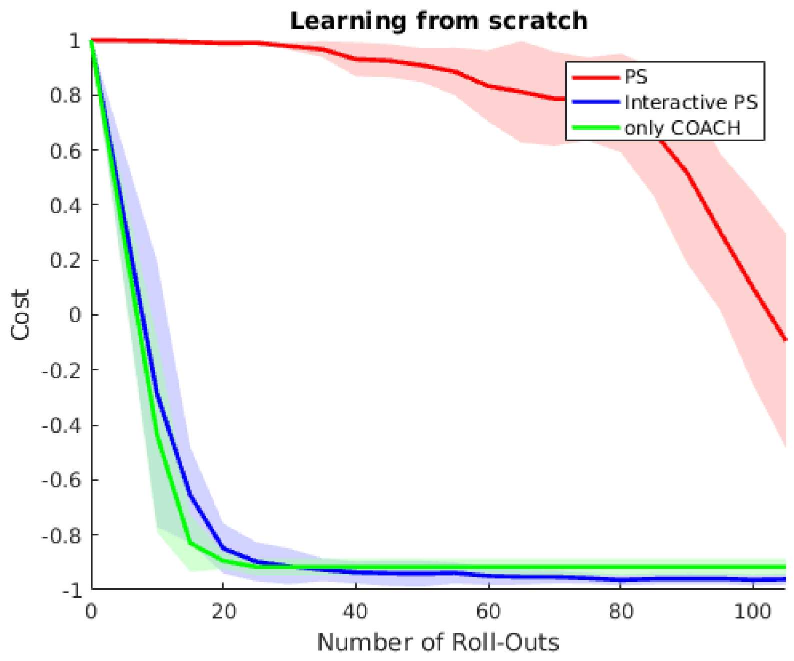


Figure 5.20: Convergence curves for learning "ball in a cup" from scratch.

same of the experiments of learning with an initial demonstration, but delayed approximately 5 trials due to the episodes intended to learn to swing the ball.

The convergence of the PS is very sensitive to the initial policy. In this scenario PS takes about 170 episodes to attain successful policies. The very beginning of the learning process is very slow because the random movements tend to diminish the effect of the previous ones, even considering that the algorithm implementations of this work use state dependent exploration as in [59, 101] for avoiding high-frequency actions.

Finally, the interactive PS keeps optimizing the policy when the improvement is not evident for the human teachers, outperforming the outcomes obtained with only COACH after 30 trials.

The set-up with the small cup is used to compare the interactive algorithms while learning from scratch. In Figure 5.21 the means of the learning curves based on human feedback presented in Figure 5.20 are taken as a reference to be compared with the experiments of learning using the small cup.

Since the cost function is the same, which basically takes into account the distance of the ball to the center of the cup, the size of the cup would not affect the cost evolution with an algorithm that is only based on the computed reward for updating the policy, like the PS. But in the cases of the methods with human feedback, it is possible to see that the cost is decreased faster in the set-up with the small cup.

The previous counter-intuitive observation is due to problems with the human perception, because for a teacher is hard to know whether the ball is falling into the cup exactly through its center or not. The users may not be able to estimate depth accurately, so when teachers

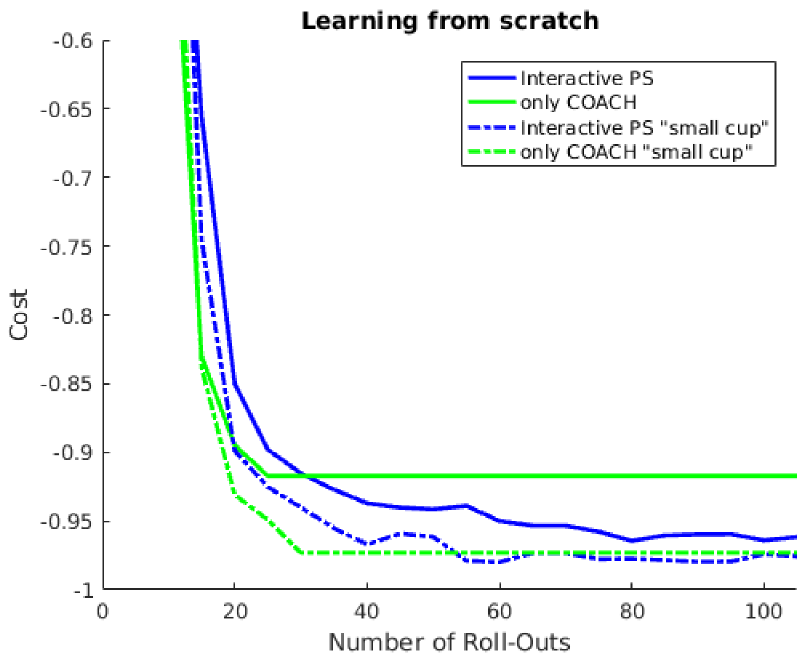


Figure 5.21: Comparison of Learning curves for the "ball in a cup" task, in the scenario of learning from scratch using the big and small cups.

are in front of the robot, it is harder to make the ball to pass through the x and y center of the big cup. With the smaller cup, the rim of the cup itself is a visual aid. Then, when the ball hits the rim, the user infers the correction based on that visual information. Therefore, when the policy makes the ball to fall into the cup without touching the rim, it is already crossing its center or very close to it.

With the small cup, the users could track more the progress of the policy, so in Figure 5.21 it is shown that users were engaged with the learning process during more trials, e.g., when learning with only COACH and the big cup, the policy improvement stopped at rollout 25, whereas with the small cup they kept correcting during 5 more rollouts.

In the previous experiments, the last part of the improvement with the interactive PS is mostly based on the reward function. However, this last experiment with the small cup shows that in cases wherein the human perception is enough to obtain insights about how good is a policy when is close to the optimum (rarely), with only COACH is possible to achieve performances like the obtained with the interactive PS. In the link[4] a video shows the learning process of the task with the proposed learning method.

5.6 Discussion

In this chapter, we have proposed the use of human corrective feedback within the frame of PS methods for learning movement primitives. First, the application of pure corrective advice for

[4]https://www.dropbox.com/s/tdu6f73zkuoq4a4/ball_in_a_cup.mp4?dl=0

adapting parametrized trajectories during time execution was presented as a simple extension of the framework COACH. Secondly, this extension was integrated in the exploration stage of standard PS algorithms in order to have together both sources of improvement: 1) Random exploration, and 2) Human corrections.

Schemes based on pure human corrective advice showed that this kind of relative corrections with vague binary signals provide human teachers with the capability to modify trajectories while a robot executes it. The exercise of writing symbols showed that users can obtain very good shapes for the symbols with corrective feedback. The users obtained better policies using corrective advice than the ones obtained solely from demonstrations, which means that the application of corrective advice renders it less necessary to have users with high level of expertise in the task and using human-robot interfaces.

The validation of the proposed interactive PS showed outstanding results in two well known benchmark problems with simulated and real robots. The learning curves showed that the proposed method speeds up the convergence of PS from 4 to more than 40 times. The human feedback is extremely powerful to accelerate the learning process at the beginning, whereas the cost function has an important influence for performing a fine tuning when suboptimal policies have a good performance, but the users' perception is not good to determine how can be obtained more improvement through corrections, e.g., when the policy already accomplishes the task but still the energy used can be reduced.

The validation of the proposed hybrid learning scheme showed that it is possible to learn complex skills such as the ones required to solve the "ball in a cup" task, without previous demonstrations. This method allows to start learning processes from (scratch) initial static policies, that incrementally receive the user's corrective advice. Little by little, the human teachers guide the robot to policies that satisfy their understanding about the fulfillment of the task. Also, the results show it is possible to learn this skill based only on human corrections.

Moreover, the proposed strategy to cope with the correspondence problem — matching between the human feedback given in the task domain, and the policy in the joint space — has shown that the method scales to high dimensionality problems, actually, in the problems of multi-DoF planar arms, the best results obtained are with the arm of highest amount of DoF.

The use of the inverse kinematic models solves the problem of matching the difference between the human teachers with the domain of the actions of the agent. In this Chapter, two schemes for learning policies with corresponding issues were proposed and validated, fulfilling the third objective of this thesis. Additionally, this generalization of COACH was tested with real robots, that continues achieving the fourth specific objective.

Chapter 6

Conclusions and Future Work

In this thesis a learning method for training agents during policy execution based on human corrections has been presented. Additionally, the learning approach was combined with Policy Search Reinforcement Learning methods in order to obtain more robust learning agents that benefit from both human knowledge and feedback of the environment.

The carried out experiments let to obtain interesting results which show how the learning approaches based on vague, binary (binary in each action dimension), coarse corrections are convenient for running faster learning convergences. These relative corrections of the executed action to be modified allow to transfer easily the insights of the users to the learning agents. COACH showed to be useful with problems in which the users are not able to demonstrate the task execution, this makes wider the spectrum of applications wherein non-expert users can assist the design of controllers or decision making systems, with respect to other LfD methods.

The use of the original COACH showed to be more robust than Interactive Reinforcement Learning methods (TAMER-like algorithms) to the occasional mistaken feedback, which is inherent to humans. The corrective feedback in the actions domain allows to clearly understand what is the effect of the advice on the policy, then, mistakes during teaching can be revised by the user with more ease.

Additionally, the hybrid learning method that combines COACH and Policy Search decreases the workload of the user, especially at the end of the learning process when the reward function may have more influence. The shared duties between the human corrections and the reward function makes more flexible the learning method in terms of human teacher's capabilities. Since wrong corrections are evaluated by the reward/cost function and weighted with low importance, the RL part of the method "discard" the wrong human feedback and keeps the progress only with the "good" corrections.

The applicability of COACH, and in general the corrective feedback was evaluated under different conditions, with simulated and real agents, using different human-machine interfaces, tested by several users, and for several kinds of tasks like navigation, balancing, and motor skills with robot arms performing fast movements of 1-2 seconds duration. The results led to conclude that COACH have a strong impact in the learning convergence, especially during the first episodes, which is very appropriate for applications with real robots. The acceleration of the convergence with respect to other learning agents was between 3 to more than 30 times in some of the explored tasks.

During the development of this research, the approached problems with COACH, and in most of the cases with other different learning methods for comparison purposes were:

- Mountain-Car
- Cart-Pole
- Learning to Balance on a bike
- Ball Dribbling with Biped robots 1D
- Ball Dribbling with Biped robots 3D: Nao Robot (validated with real robots in adversarial environments during RoboCup competitions)
- Simulated Pendulum swing-up
- Real Pendulum Swing-up
- Cart-Pole extended (to place the cart in the center of the scene while balancing the pole)
- Inverse Kinematics model with a real 3 DoF robotic arm
- Inverted wedge (real system)
- Writing Symbols with a simulated 3 DoF arm
- Writing Symbols with a real 6 DoF arm: UR5
- Reaching movements through via-points with simulated arms of 1, 2, 10, 50 DoF
- Ball in a cup with a real 7 DoF arm: Kuka lightweight arm

Moreover, the use of COACH was validated with good results in two additional problems that are not reported in this thesis.

First, in the context of robot soccer, corrective feedback was used for training a decision making system that computes the desired direction of the ball for the robot to dribble. This system works in a higher level than the dribbling engine presented in chapter 3, so the direction computed is used as input of the dribbling engine in order to compute the walking velocities requests. This system observes the positions of the ball, the opponents, the opponents' goal, and the field dimensions, in order to make a decision that avoid the opponents while pushing the ball closer to the goal, but decreasing the risk of pushing the ball put of the field. This problem along with the dribbling system presented before, was approached with COACH in order to solve the problem and use the final system in the competition, rather than using it for COACH evaluation and comparison purposes.

The second problem was learning to throw a ball to different distances with a 6 DoF Katana arm of Neuronics. In this application, COACH worked easily to obtain a policy that throws a ball to a small cup with accuracy of a couple of centimetres.

In the future work, it is considered the use of human corrective feedback for training policies without predefined state representations, so the human corrections not only would shape the magnitude of actions, but indirectly, this feedback also defines and improves the way states are represented. In this regard, the recent developments on learning with deep neural networks can obtain good advantages. An extension of COACH for training policies represented with deep neural networks would shrink the distance between end-users and systems' designers, the features engineering necessary before all the learning processes would be eliminated. Research in this direction may reduce the prior knowledge or assumptions that are necessary in the methods proposed in this thesis, e.g., the design of basis functions.

Also, it is considered to develop algorithms based on COACH, which actively learn policies while requesting corrections in state space regions with high uncertainty of the actions to execute. In this regard, models that measure the uncertainty associated to the decision would be necessary for the policy model, like Gaussian processes, or Gaussian Mixtures Models. Also, the uncertainty can be used for adapting the exploration in the case of hybrid learning agents that learn from reward functions.

Another extension of COACH would be for integrating of interactive learning with shared control. A human operator that shares the control of an automated system may give commands (as normal shared control), but the system can learn from those commands and after some executions the load on the user would be reduced, the system would predict the human contribution in the actions controller.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Sander Adam, Lucian Busoniu, and Robert Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):201–212, 2012.

[3] Ricardo Aler, Oscar Garcia, and José María Valls. Correcting and improving imitation models of humans for robosoccer agents. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2402–2409. IEEE, 2005.

[4] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4):105–120, 2014.

[5] Saleema Amershi, James Fogarty, and Daniel Weld. Regroup: Interactive machine learning for on-demand group creation in social networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 21–30. ACM, 2012.

[6] Brenna D Argall. *Learning mobile robot motion control from demonstration and corrective feedback*. PhD thesis, Carnegie Mellon University, 2009.

[7] Brenna D Argall, Brett Browning, and Manuela Veloso. Learning robot motion control with demonstration and advice-operators. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 399–404. IEEE, 2008.

[8] Brenna D Argall, Brett Browning, and Manuela M Veloso. Teacher feedback to scaffold and refine demonstrated motion primitives on a mobile robot. *Robotics and Autonomous Systems*, 59(3):243–255, 2011.

[9] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[10] Brenna D Argall, Eric L Sauser, and Aude G Billard. *Tactile guidance for policy adaptation*, volume 2. Now Publishers Inc, 2011.

[11] Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20, 1997.

[12] Robert Babuska. Fuzzy and neural control. disc course lecture notes. *Delft University of Technology. Delft, the Netherlands*, 2001.

[13] J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1615–1620. IEEE, 2001.

[14] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming: an introduction*, volume 1. Morgan Kaufmann San Francisco, 1998.

[15] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Reinforcement learning and chess. In *Machines that learn to play games*, pages 91–116. Nova Science Publishers, Inc., 2001.

[16] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

[17] Erik A Billing and Thomas Hellström. A formalism for learning from demonstration. *Paladyn, Journal of Behavioral Robotics*, 1(1):1–13, 2010.

[18] Justin A Boyan and Michael L Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in neural information processing systems*, pages 671–678, 1994.

[19] Cynthia Breazeal, Matt Berlin, Andrew Brooks, Jesse Gray, and Andrea L Thomaz. Using perspective taking to learn from ambiguous demonstrations. *Robotics and autonomous systems*, 54(5):385–393, 2006.

[20] Cynthia Breazeal and Brian Scassellati. Robots that imitate humans. *Trends in cognitive sciences*, 6(11):481–487, 2002.

[21] Rodney A Brooks and Maja J Mataric. Real robots, real learning problems. *Robot learning*, pages 193–213, 1993.

[22] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[23] Lucian Busoniu, Damien Ernst, Bart De Schutter, and Robert Babuska. Cross-entropy optimization of control policies with adaptive basis functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(1):196–209, 2011.

[24] Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 17–24. ACM, 2012.

[25] Gerard Canal, Guillem Alenyà, and Carme Torras. Personalization framework for adaptive robotic feeding assistance. In *International Conference on Social Robotics*, pages 22–31. Springer, 2016.

[26] Carlos Celemin and Javier Ruiz-del Solar. Interactive learning of continuous actions from corrective advice communicated by humans. In *Robot Soccer World Cup*, pages 16–27. Springer, 2015.

[27] Sonia Chernova and Andrea L Thomaz. Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(3):1–121, 2014.

[28] Sonia Chernova and Manuela Veloso. Multi-thresholded approach to demonstration selection for interactive robot learning. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pages 225–232. ACM, 2008.

[29] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):1, 2009.

[30] Jonathan H Connell and Sridhar Mahadevan. Rapid task learning for real robots. *Robot Learning*, pages 105–139, 1993.

[31] Heriberto Cuayáhuitl, Martijn van Otterlo, Nina Dethlefs, and Lutz Frommberger. Machine learning for interactive systems and robots: a brief introduction. In *Proceedings of the 2nd Workshop on Machine Learning for Interactive Systems: Bridging the Gap Between Perception, Action and Communication*, pages 19–28. ACM, 2013.

[32] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[33] Kenji Doya. Designing the reward system: Computational and biological principles. In *FOCI*, page 645, 2007.

[34] Marco Ewerton, Guilherme Maeda, Gerrit Kollegger, Josef Wiemeyer, and Jan Peters. Incremental imitation learning of context-dependent motor skills. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 351–358. IEEE, 2016.

[35] Jerry Alan Fails and Dan R Olsen Jr. Interactive machine learning. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 39–45. ACM, 2003.

[36] Daniel H Grollman and Odest Chadwicke Jenkins. Learning robot soccer skills from demonstration. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 276–281. IEEE, 2007.

[37] Simon S Haykin. *Neural networks: a comprehensive foundation.* Tsinghua University Press, 2001.

[38] Andrew Heathcote, Stephen J Popiel, and DJ Mewhort. Analysis of response time

distributions: An example using the stroop task. *Psychological Bulletin*, 109(2):340, 1991.

[39] Verena Heidrich-Meisner and Christian Igel. Evolution strategies for direct policy search. In *PPSN*, pages 428–437. Springer, 2008.

[40] Todd Hester, Michael Quinlan, and Peter Stone. Generalized model learning for reinforcement learning on a humanoid robot. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2369–2374. IEEE, 2010.

[41] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1398–1403. IEEE, 2002.

[42] Nasser Jazdi. Cyber physical systems in the context of industry 4.0. In *Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on*, pages 1–4. IEEE, 2014.

[43] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[44] S Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[45] H Jin Kim, Michael I Jordan, Shankar Sastry, and Andrew Y Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*, pages 799–806, 2004.

[46] W Bradley Knox, Cynthia Breazeal, and Peter Stone. Learning from feedback on actions past and intended. In *In Proceedings of 7th ACM/IEEE International Conference on Human-Robot Interaction, Late-Breaking Reports Session (HRI 2012)*, 2012.

[47] W Bradley Knox and Peter Stone. Tamer: Training an agent manually via evaluative reinforcement. In *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*, pages 292–297. IEEE, 2008.

[48] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM, 2009.

[49] W Bradley Knox and Peter Stone. Combining manual feedback with subsequent mdp reward signals for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 5–12. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[50] W Bradley Knox and Peter Stone. Reinforcement learning from human reward: Discounting in episodic tasks. In *RO-MAN, 2012 IEEE*, pages 878–885. IEEE, 2012.

[51] W Bradley Knox and Peter Stone. Reinforcement learning from simultaneous human and mdp reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 475–482. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[52] W Bradley Knox and Peter Stone. Learning non-myopically from human-generated reward. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, pages 191–202. ACM, 2013.

[53] W Bradley Knox and Peter Stone. Framing reinforcement learning from human reward: Reward positivity, temporal discounting, episodicity, and performance. *Artificial Intelligence*, 225:24–50, 2015.

[54] W Bradley Knox, Peter Stone, and Cynthia Breazeal. Teaching agents with human feedback: a demonstration of the tamer framework. In *Proceedings of the companion publication of the 2013 international conference on Intelligent user interfaces companion*, pages 65–66. ACM, 2013.

[55] W Bradley Knox, Peter Stone, and Cynthia Breazeal. Training a robot via human feedback: A case study. In *International Conference on Social Robotics*, pages 460–470. Springer, 2013.

[56] William Bradley Knox. Learning from human-generated reward. 2012.

[57] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[58] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2112–2118. IEEE, 2009.

[59] Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

[60] Jens Kober, Andreas Wilhelm, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4):361–379, 2012.

[61] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2619–2624. IEEE, 2004.

[62] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3232–3237. IEEE, 2010.

[63] Adrián León, Eduardo Morales, Leopoldo Altamirano, and Jaime Ruiz. Teaching a robot to perform task through imitation and on-line feedback. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 549–556, 2011.

[64] Leonardo Leottau, Carlos Celemin, and Javier Ruiz-del Solar. Ball dribbling for humanoid biped robots: a reinforcement learning and fuzzy control approach. In *Robot Soccer World Cup*, pages 549–561. Springer, 2014.

[65] Yun Lin, Shaogang Ren, Matthew Clevenger, and Yu Sun. Learning grasping force from demonstration. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1526–1531. IEEE, 2012.

[66] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pages 157–163, 1994.

[67] Guilherme Maeda, Marco Ewerton, Dorothea Koert, and Jan Peters. Acquiring and generalizing the embodiment mapping from human observations to robot skills. *IEEE Robotics and Automation Letters*, 1(2):784–791, 2016.

[68] Shie Mannor, Reuven Y Rubinstein, and Yohai Gat. The cross entropy method for fast policy search. In *ICML*, pages 512–519, 2003.

[69] Cetin Mericli. *Multi-Resolution Model Plus Correction Paradigm for Task and Skill Refinement on Autonomous Robots*. PhD thesis, Citeseer, 2011.

[70] Çetin Meriçli and Manuela Veloso. Improving biped walk stability using real-time corrective human feedback. In *Robot Soccer World Cup*, pages 194–205. Springer, 2010.

[71] Cetin Meriçli, Manuela Veloso, and H Levent Akin. Complementary humanoid behavior shaping using corrective demonstration. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 334–339. IEEE, 2010.

[72] Çetin Meriçli, Manuela M Veloso, and H Levent Akin. Task refinement for autonomous robots using complementary corrective human feedback. *International Journal of Advanced Robotic Systems*, 2011.

[73] Noriaki Mitsunaga, Christian Smith, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. Adapting robot behavior for human–robot interaction. *IEEE Transactions on Robotics*, 24(4):911–916, 2008.

[74] Chrystopher L Nehaniv and Kerstin Dautenhahn. Of hummingbirds and helicopters: An algebraic framework for interdisciplinary studies of imitation and its applications. In *Interdisciplinary approaches to robot learning*, pages 136–161. World Scientific, 2000.

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

[76] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[77] Hung Ngo, Matthew Luciw, Jawas Nagi, Alexander Forster, Jürgen Schmidhuber, and Ngo Anh Vien. Efficient interactive multiclass learning from binary feedback. *ACM*

Transactions on Interactive Intelligent Systems (TiiS), 4(3):12, 2014.

[78] Monica N Nicolescu and Maja J Mataric. Learning and interacting in human-robot domains. *IEEE Transactions on Systems, man, and Cybernetics-part A: Systems and Humans*, 31(5):419–430, 2001.

[79] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems*, pages 2616–2624, 2013.

[80] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[81] Patrick M Pilarski, Michael R Dawson, Thomas Degris, Farbod Fahimi, Jason P Carey, and Richard S Sutton. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, pages 1–7. IEEE, 2011.

[82] Dimitris Pongas, Aude Billard, and Stefan Schaal. Rapid synchronization and accurate phase-locking of rhythmic motor primitives. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2911–2916. IEEE, 2005.

[83] M Rahat. Matlab implementation of controlling a bicycle using reinforcement learning. 2010.

[84] Jette Randlov and Preben Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471, 1998.

[85] Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

[86] Stefan Schaal and Christopher G Atkeson. Robot juggling: implementation of memory-based learning. *IEEE Control Systems*, 14(1):57–71, 1994.

[87] Yannick Schroecker, Heni Ben Amor, and Andrea Thomaz. Directing policy search with interactively taught via-points. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1052–1059. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[88] David Silver, J Andrew Bagnell, and Anthony Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29(12):1565–1592, 2010.

[89] Mohan Sridharan. Augmented reinforcement learning for interaction with non-expert humans in agent domains. In *Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on*, volume 1, pages 424–429. IEEE, 2011.

[90] Freek Stulp and Olivier Sigaud. Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*, 2012.

[91] Freek Stulp and Olivier Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. 2012.

[92] Freek Stulp and Olivier Sigaud. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61, 2013.

[93] Halit Bener Suay and Sonia Chernova. Effect of human guidance and state space size on interactive reinforcement learning. In *RO-MAN, 2011 IEEE*, pages 1–6. IEEE, 2011.

[94] Halit Bener Suay, Russell Toris, and Sonia Chernova. A practical comparison of three robot learning from demonstration algorithm. *International Journal of Social Robotics*, 4(4):319–330, 2012.

[95] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[96] John D Sweeney and Rod Grupen. A model of shared grasp affordances from demonstration. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 27–35. IEEE, 2007.

[97] Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. In *Readings in Fuzzy Sets for Intelligent Systems*, pages 387–403. Elsevier, 1993.

[98] Matthew E Taylor and Sonia Chernova. Integrating human demonstration and reinforcement learning: Initial results in human-agent transfer. In *Proceedings of the Agents Learning Interactively with Human Teachers AAMAS workshop*, page 23, 2010.

[99] Ana C Tenorio-Gonzalez, Eduardo F Morales, and Luis Villaseñor-Pineda. Dynamic reward shaping: training a robot by voice. In *Ibero-American Conference on Artificial Intelligence*, pages 483–492. Springer, 2010.

[100] Gerald Tesauro. Practical issues in temporal difference learning. In *Advances in neural information processing systems*, pages 259–266, 1992.

[101] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11(Nov):3137–3181, 2010.

[102] AL Thomaz and C Breazeal. Adding guidance to interactive reinforcement learning. In *Proceedings of the Twentieth Conference on Artificial Intelligence (AAAI)*, 2006.

[103] Andrea L Thomaz and Cynthia Breazeal. Asymmetric interpretations of positive and negative human feedback for a social learning agent. In *RO-MAN 2007-The 16th IEEE International Symposium on Robot and Human Interactive Communication*, pages 720–725. IEEE, 2007.

[104] Andrea L Thomaz, Guy Hoffman, and Cynthia Breazeal. Reinforcement learning with human teachers: Understanding how people want to teach robots. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 352–357. IEEE, 2006.

[105] Sebastian Thrun. Learning to play the game of chess. In *Advances in neural information processing systems*, pages 1069–1076, 1995.

[106] Ngo Anh Vien and Wolfgang Ertel. Reinforcement learning combined with human feedback in continuous state and action spaces. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–6. IEEE, 2012.

[107] Ngo Anh Vien, Wolfgang Ertel, and Tae Choong Chung. Learning via human feedback in continuous state and action spaces. *Applied intelligence*, 39(2):267–278, 2013.

[108] Malcolm Ware, Eibe Frank, Geoffrey Holmes, Mark Hall, and Ian H Witten. Interactive machine learning: letting users build classifiers. *International Journal of Human-Computer Studies*, 55(3):281–292, 2001.

[109] Astrid Weiss, Judith Igelsbock, Sylvain Calinon, Aude Billard, and Manfred Tscheligi. Teaching a humanoid: A user study on learning by demonstration with hoap-3. In *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, pages 147–152. IEEE, 2009.

[110] Paul M Yanik, Joe Manganelli, Jessica Merino, Anthony L Threatt, Johnell O Brooks, Keith Evan Green, and Ian D Walker. A gesture learning interface for simulated robot path shaping with a human teacher. *IEEE Transactions on Human-Machine Systems*, 44(1):41–54, 2014.

[111] Chung-Che Yu and Chieh-Chih Wang. Interactive learning from demonstration with a multilevel mechanism for collision-free navigation in dynamic environments. In *Technologies and Applications of Artificial Intelligence (TAAI), 2013 Conference on*, pages 240–245. IEEE, 2013.

[112] Shao Zhifei and Er Meng Joo. A review of inverse reinforcement learning theory and recent advances. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[113] Changjiu Zhou. Neuro-fuzzy gait synthesis with reinforcement learning for a biped walking robot. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 4(4):238–250, 2000.