Journal of Intelligent & Robotic Systems (2019) 95:77–97
https://doi.org/10.1007/s10846-018-0839-z

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

Carlos Celemin[1] · **Javier Ruiz-del-Solar**[1]

Received: 21 July 2017 / Accepted: 21 February 2018 / Published online: 12 May 2018

Abstract

The main goal of this article 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.

Keywords Learning from demonstration · Interactive machine learning · Human feedback · Human teachers · Decision making systems

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 article 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 [1], 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 [2], 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 [1], a well-known shaping approach.

The proposed interactive learning framework is characterized and validated extensively using four well-known learning problems: (i) *Mountain Car* [3], (ii) *Cart-Pole* [3],

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10846-018-0839-z) contains supplementary material, which is available to authorized users.

✉ Carlos Celemin
carlos.celemin@ing.uchile.cl

Javier Ruiz-del-Solar
jruizd@ing.uchile.cl

[1] Advanced Mining Technology Center & Department of Electrical Engineering, Universidad de Chile, Av. Tupper 2007, Santiago, Chile

(iii) *Ball Dribbling with Humanoid Robots* [4] in the context of robot soccer, and (iv) *Learning to Balance on a Bicycle* [5]. For the first three problems the performances of agents trained using COACH are compared with the performances achieved by agents trained using TAMER [1], ACTAMER [6], 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.

In previous work COACH was introduced [7, 8], and a preliminary validation was presented. In this extended and improved version, a general COACH algorithm which does not depend on the approximation functions used, and that can be extended to multidimensional action problems, is presented. In addition, more extensive experiments that allow a deeper analysis of the COACH characteristics were carried out and are reported here.

The paper is organized as follows: In Section 2, some related work on Interactive Machine Learning is presented. In Section 3, the general COACH learning framework for one-dimensional action problems is described. Then, two variants of the framework based on different function approximations, namely, linear models of Radial Basis Functions and Takagi-Sugeno Fuzzy Systems, are presented. In addition, a generalization of COACH for multi-dimensional action problems is also given. In Section 4, the experimental validation of the framework is presented. Finally, in Section 5 some conclusions of this work are drawn.

2 Related Work

Interactive Machine Learning (IML) is a machine learning paradigm in which a human teacher takes part in the agent's learning process in order to supervise the improvement of the policy. Most of the general ideas about IML have been summarized in [9–14]. Some developments have been applied for learning classifier systems [15–18], but in this work the focus is on methods of IML for learning decision-making systems with continuous actions (regressions).

Several methods of IML have been proposed regarding the type of skills to learn, the type of user feedback, the type of Human-Machine Interfaces (HMI) used, etc. In Fig. 1 the basic interaction is shown between the human teacher who observes, and the agent that uses the action to modify the environment, which is modeled with the state vector. There are two main schemes for using human feedback in order to modify the policy of a learning agent: human feedback in the action domain, and human feedback in the evaluative domain.

2.1 Human Feedback in Action Domains

Learning from Demonstration (*LfD*) is a wide learning paradigm in which a human user demonstrates the execution of the task, whereas the agent learns and reproduces the demonstrated behavior. There are some works which explore different applications of *LfD* in robotic domains such as soccer [19, 20], driving applications [21], humanoid motion [22, 23], navigation [24, 25], grasping and manipulation [26, 27], and many other problems.

There are algorithms of LfD which are focused on learning from corrective demonstrations, which, instead of deriving a policy from the demonstrations, keep hand-coded algorithms as the primary source of the action policy, and use the demonstration data only to make exceptions as needed [28–31].

In continuous action domains, *Advice Operator Policy Improvement* has been proposed for correcting policies. After the task execution, the trainer chooses a subset of the whole state and action pairs dataset that she/he considers should be corrected; then, an advice operator is associated with this selected dataset. The operator could be increasing (decreasing) the magnitude of the respective actions; the data modified is attached to the demonstration dataset for a policy rederivation [2, 32, 33].

In most of the LfD approaches, accurate demonstrations and expert users as trainers are required. Methods based on the *Advice Operator* paradigm do not need an expert trainer at all, but have the drawback of always deducing the policy from a dataset in a batch learning scheme. The dataset is increased with the set of state-action pairs selected

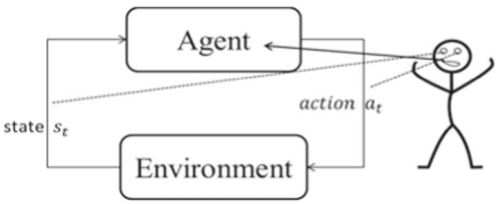

Fig. 1 Interactive machine learning scheme

for the correction in every advising phase, increasing the computational burden of the policy rederivation. Another drawback is that the effect of the corrective advice cannot be seen immediately, because the policy rederivation is computed off-line, therefore, further corrections might be harmful or contradictory, regarding earlier corrections which have not been taken for updating the policy. In [34], the walking stability of a Nao humanoid robot was improved, applying sequentially corrective demonstrations and *Advice Operators*.

2.2 Human Feedback in Evaluative Domains

Human Teachers can train agents providing evaluations in two main different ways, selecting preferences or providing reinforcements. With *human preferences,* the user communicates to the learner the preferences of the teacher by selecting the prefered policy execution among a set of executions that the agent demonstrates [35, 36], it obtained interesting results in toy problems. This approach has also been applied to learn complex simulated tasks with deep neural networks [37], and applied to manipulation tasks with real robots in [38].

The other possibility of teaching with evaluative feedback is with approaches in which the teacher evaluates with rewards or punishments the actions executed by the learner. In this branch of learning methods, non-expert users can evolve decision-making systems, even online, by interactively evaluating the executed actions with approvement or disapprovement signals in an RL fashion. In this paradigm the reward is partially or completely given by the human [39–45].

The *shaping* approach allows training an agent interactively through teacher's reinforcement [1], but it incorporates the human feedback taking into account especial considerations different from the way RL uses the MDP reward function.

The TAMER framework (Training an Agent Manually via Evaluative Reinforcement) [46, 47] addresses the problem of how to use delayed human rewards in RL problems with discrete action domains. In order to achieve this, it proposes the use of *credit assignment*. 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" [48]. 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, which produces 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 with 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 [1].

More recently, other authors proposed ACTAMER, an Actor-Critic approach based on TAMER, which addresses RL problems with continuous action domains [6, 49].

COACH, the framework proposed here, is based on the structure of TAMER for updating the model, but it receives from the teacher the same type of corrective feedback used in the *Advice Operators* paradigm, since in general human teachers prefer to provide feedback that communicate to the agent "what to do" instead of "how good is an action", i.e. they prefer the feedback in the actions domains [14, 50, 51].

One of the main features of COACH is a mechanism for adjusting the amount of human feedback that a given action receives adaptively, taking past feedback into consideration.

3 COACH: Corrective Advice Communicated by Humans

The COACH learning framework allows human teachers to train policies while the agents are interacting with the environment of continuous actions. It uses human binary feedback as a correction in the action domain, a principle inspired by the *advice operators'* paradigm. The correction is a relative change of the magnitude of the executed action, and it is given in order to update the current policy for the state in which that action was executed. The human trainers have to provide their feedback immediately after the execution of the action intended to be modified.

3.1 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 work 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.

3.1.1 Policy Supervised Learner

In this Framework, when the *Policy* module observes the state vector $\vec{s}$, it executes a continuous action $P(\vec{s})$ according to the policy model $P : S \rightarrow \mathbb{R}$. (This is a dif-

ference 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 $\vec{s}$ (increase or decrease its magnitude, respectively). The state $\vec{s}$, the executed action $P(\vec{s})$, 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, $error = e \cdot h$. The learning rate is taken as an external parameter that can be constant or adapted by another module as shown in coming subsections.

3.1.2 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 Fig. 2), 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 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 $\vec{s}$ (line 4), and it computes the action a given by the current policy model $P(\vec{s})$ (line 5). Afterwards, $P(\vec{s})$ is executed (line 6). After action

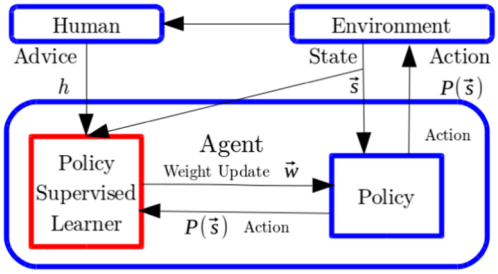

Fig. 2 Basic structure of COACH

execution, the feedback signal h of the 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 $P(\vec{s})$ is updated (line 11), where $updatePolicyModel(\alpha, error, \vec{s})$ is a function that reads the meta-parameters of the learning model, the input vector $\vec{s}$, and the prediction error associated to the input vector, and updates the model using SGD.

Algorithm 1 Basic structure of COACH (simple framework)

```
1:  e ← constant // error magnitude
2:  α ← constant // learning parameters
3:  while true do
4:      s ← getStateVec()
5:      a = P(s)
6:      TakeAction(P(s))
7:      wait for next time step
8:      h ← gethumanCorrectiveAdvice()
9:      if h = ! 0 then
10:         error ← h · e
11:         updatePolicyModel(α, error, s)
12:     end if
13: end while
```

3.1.3 Human Feedback Modeling & 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 $H : S \rightarrow \mathbb{R}$ is built, which characterizes the human feedback signal over each region of the state space. The same type of approximator of the policy model $P(\vec{s})$ is used for representing the human feedback model $H(\vec{s})$. Therefore, two *Supervised Learner* modules are required in the framework, one for P and one for H (see the block diagram shown in Fig. 4 at Section 3.2). 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 ($\overrightarrow{s_a}$) would mean that the trainer is suggesting a large change in the magnitude of the associated action $P(\overrightarrow{s_a})$. 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 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 s*upervised 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 $H(\vec{s})$.

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

$$\alpha(\vec{s}) = |H(\vec{s})| + bias \tag{1}$$

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

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

For demonstrative purposes, Fig. 3 shows an example that compares the effect of giving human corrections over time, for a specific state $\overrightarrow{s_a}$, with constant, or adaptive, learning rates. The figure shows the value of the action when the agent visits only $\overrightarrow{s_a}$ 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 than the case of a constant learning rate.

3.1.4 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 Eq. 2 as the integral of pdf_{delay} from t_{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 Eq. 3.

$$c_t = \int_{t_{i-1}}^{t_i} pdf_{delay}(x)\, dx \tag{2}$$

$$\sum_{i=1}^{n} c_t \approx 1 \tag{3}$$

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 $[\overrightarrow{s_t}\, P(\overrightarrow{s_t})]$ the corresponding $error_t$ used for the *t-th* update is the *error* weighted with its associated credit c_t:

$$error_t = h \cdot e \cdot c_t \tag{4}$$

The state-action $[\overrightarrow{s_n}\, P(\overrightarrow{s_n})]$ 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 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

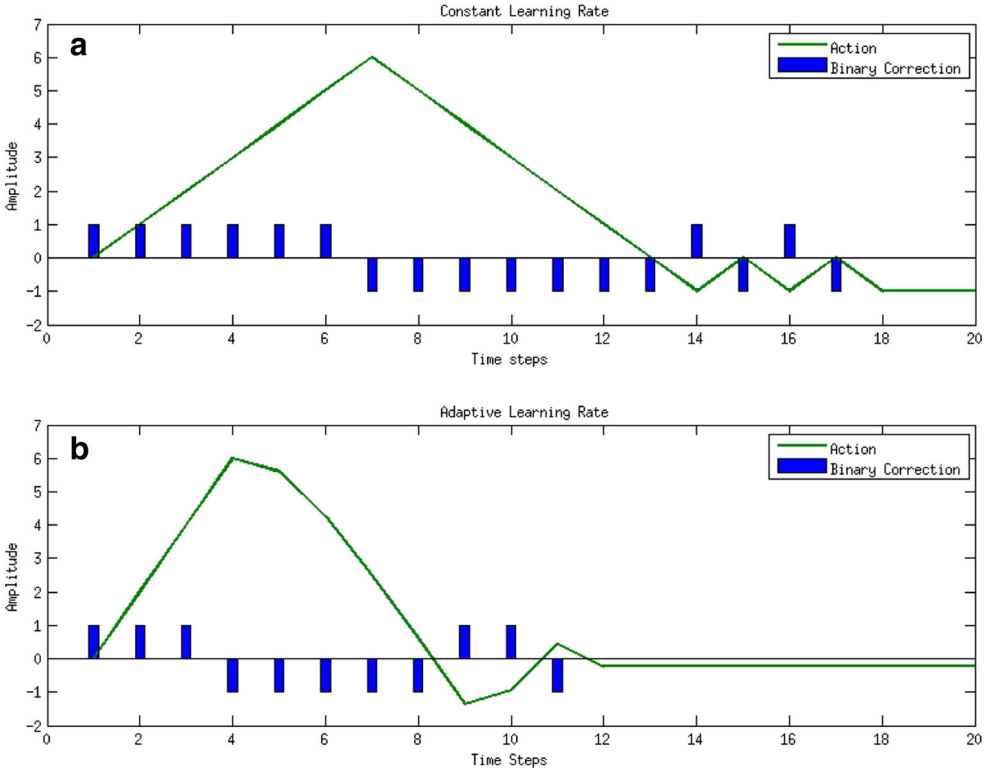

Fig. 3 Human feedback progress at a specific state $\overrightarrow{s_a}$, and its impact over the respective action, using: a constant learning rate (**a**), and an adaptive learning rate (**b**)

of humans, and also, wherein the dilemma of setting small/big learning rate for fine/wide corrections is present, the COACH framework requires the modules of *Human Modeling* and *Credit Assigner*. The complete structure of COACH with all the modules is depicted in Fig. 4. 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 in RL [3, 52, 53], and (ii) COACH is mainly inspired by TAMER, which was originally proposed based on RBF linear models [1, 46].

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

$$P(\vec{s}) = \overrightarrow{w}^T \cdot \vec{f} \tag{5}$$

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

$$\Delta w_l = \alpha(\vec{s}) \cdot error \cdot \frac{\partial P(\vec{s})}{\partial w_l} = \alpha(\vec{s}) \cdot error \cdot f_l \tag{6}$$

with *error* the product between h and e, and l the weight's/feature's index.

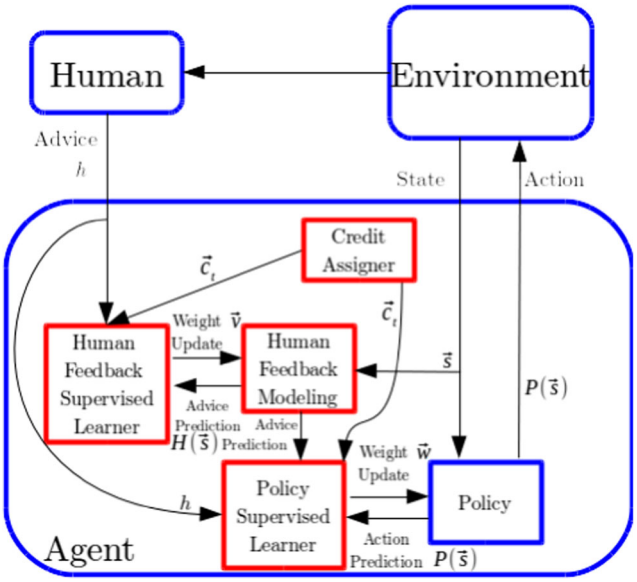

Fig. 4 General structure of COACH, using an adaptive learning rate for the policy update and the credit assigner

The model of the human feedback H is built using the same features vector $\vec{f}$ of P, and a weight vector $\vec{v}$ as:

$$H(\vec{s}) = \vec{v}^T \cdot \vec{f} \tag{7}$$

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 $H(\vec{s})$. Therefore, using a gradient descent approach, the weights associated with the H model are updated as:

$$\Delta v_l = \beta \cdot (h - H(\vec{s})) \cdot \frac{\partial H(\vec{s})}{\partial v_l} = \beta \cdot \left(h - \vec{v}^T \cdot \vec{f}\right) \cdot f_l \tag{8}$$

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 f^{cred}; then the update step is computed just once using that 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 $\overrightarrow{f^{cred}}$ for replacing the original vector $\vec{f}$ in the H and P update process (5)–(8). $\overrightarrow{f^{cred}}$ is the sum of the n past feature vectors $\overrightarrow{f_t}$ that are weighted with the credit c_t:

$$\overrightarrow{f^{cred}} = \sum_{t=1}^{n} c_t \cdot \overrightarrow{f_t} \tag{9}$$

Algorithm 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 (2).

The loop between lines 6–23 occurs once per time step. The agent observes the new state $\vec{s}$ (line 7), it maps the states into the features space of the basis functions (line 8) and it computes the policy model $P(\vec{s})$ (line 9). Afterwards, $P(\vec{s})$ 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, with the received correction, the models are updated (lines 13–22). First, the credit assigner computes the vector $\overrightarrow{f^{cred}}$ taking the n past feature vectors $\overrightarrow{f_t}$ (lines 14–16). The human feedback model H is updated in lines 17–18, the prediction of the human advice $H(\vec{s})$ 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).

Algorithm 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** $1 \leq t \leq n$ **do**
4: $\quad c_t \leftarrow assignCredit(t)$
5: **end for**
6: **while** *true* **do**
7: $\quad \vec{s} \leftarrow getStateVec()$
8: $\quad \vec{f} \leftarrow getFeatures(\vec{s})$
9: $\quad P(\vec{s}) \leftarrow \vec{w}^T \cdot \vec{f}$
10: $\quad TakeAction(P(\vec{s}))$
11: $\quad$ wait for next time step
12: $\quad h \leftarrow gethumanCorrectiveAdvice()$
13: $\quad$ **if** $h = ! 0$ **then**
14: $\quad\quad$ **for** $1 \leq t \leq n$ **do**
15: $\quad\quad\quad \overrightarrow{f^{cred}} = \overrightarrow{f^{cred}} + \left(c_t \cdot \overrightarrow{f_t}\right)$
16: $\quad\quad$ **end for**
17: $\quad\quad H(\vec{s}) = \vec{v}^T \cdot \overrightarrow{f^{cred}}$
18: $\quad\quad \Delta v_l = \beta \cdot (h - H(\vec{s})) \cdot f_l^{cred}; l = 1, .., N_{feat}$
$v_l = v_l + \Delta v_l$
19: $\quad\quad \alpha(\vec{s}) = \mid H(\vec{s}) \mid + bias = \mid \vec{v}^T \cdot \overrightarrow{f^{cred}} \mid + bias$
20: $\quad\quad error \leftarrow h \cdot e$
21: $\quad\quad \Delta w_l = \alpha(\vec{s}) \cdot error \cdot f_l^{cred}; l = 1, .., N_{feat}$
$w_l = w_l + \Delta w_l$
22: $\quad$ **end if**
23: **end while**

3.3 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 [54, 55].

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, $1 \leq j \leq J$; $J = number\ of\ rules$) of the fuzzy rule base:

$$f_j = \frac{r_j}{\sum_J r_j} \tag{10}$$

In this case, each feature or rule is associated with a function $u_j(\vec{s})$ instead of a simple weight as in the RBF linear model. The function $u_j(\vec{s})$ 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:

$$u_j(\vec{s}) = \sum_K a_{jk} s_k = \vec{a_j}^T \vec{s} \tag{11}$$

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

$$\begin{aligned} P(\vec{s}) &= \sum_J u_j(\vec{s}) f_j = \overrightarrow{u(\vec{s})}^T \cdot \vec{f} \\ &= \sum_J \left(\sum_K a_{jk} s_k \right) f_j \end{aligned} \tag{12}$$

For updating the weights matrix $\vec{a}$ 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:

$$\begin{aligned} P(\vec{s}) &= \sum_K s_k \left(\sum_J a_{jk} f_j \right) \\ &= \sum_K s_k g_k(\vec{s}) \end{aligned} \tag{13}$$

$P(\vec{s})$ 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:

$$\frac{\partial P(\vec{s})}{\partial a_{jk}} = \frac{\partial P(\vec{s})}{\partial g_k(\vec{s})} \frac{\partial g_k(\vec{s})}{\partial a_{jk}} = s_k \cdot f_j \tag{14}$$

Then, COACH updates the weights as:

$$\Delta a_{jk} = \alpha \, error \frac{\partial P(\vec{s})}{\partial a_{jk}} = \alpha \cdot error \cdot s_k \cdot f_j \tag{15}$$

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 Eqs. 10–15. 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 $P(\vec{s})$. 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 2.
- The function *getFeatures*($\vec{s}$) maps the state space to the feature space using Eq. 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 Eqs. 11–13 that have to be replaced in line 9, due the TS-FS structure.

The policy weights update given by Eq. 14 in line 21. In this case the weights are a matrix instead of a vector, and the updating term includes the factor s_k according to Eqs. 14 and 15.

3.4 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.

4 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 other half with 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, 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 [5, 56]. 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 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 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.

4.1 Mountain-Car Problem

In this classical toy problem, a simulated car must get to the top of a hill [3]. The car starts between two steep hills and must go back and forth to gain enough momentum to reach the goal. In this work, 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 was the one reported in [3], 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, but in this problem, the experiments with the interactive agents were executed using a keyboard interface, and then using a Hand-Gesture Recognition (HGR) interface, which is described in the Appendix.

Table 1 Learning parameters for the experiments

Algorithm	Parameters
Mountain-car	
SARSA	$\lambda = 0.9,\ \alpha_S = 0.15,\ \epsilon = 0.01,\ \gamma = 0.98$
COACH	$e = 0.5,\ \beta = 0.35$
Cart-pole	
SARSA	$\lambda = 0.95,\ \alpha_S = 0.1,\ \epsilon = 0.02,\ \gamma = 0.99$
COACH	$e = 5,\ \beta = 0.35$
Ball dribbling 1-d	
SARSA	$\lambda = 0.85,\ \alpha_S = 0.2,\ \epsilon = 0.05,\ \gamma = 0.997$
COACH	$e = 30,\ \beta = 0.35$
Ball dribbling 3-d	
COACH	$e_x = 30, e_y = 20, e_\theta = 20,\ \beta = 0.35$
Bike balance	
SARSA	$\lambda = 0.95,\ \alpha_S = 0.5,\ \epsilon = 0.01,\ \gamma = 0.99$
COACH	$e_T = 0.5, e_d = 0.8,\ \beta = 0.35$

The learning curves obtained -average cumulative reward- are shown in Fig. 5. 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 velocity 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 achievement. 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.

When using the HGR interface, COACH kept the highest and fastest learning curve; the agents trained with COACH converged at episode 9, but the difference between COACH and the rest of the interactive agents is greater in this experiment; the teleoperation learning curve is especially lower with the HGR interface. This fact could be related to the well-known problem of LfD associated with noisy and occasionally flawed human feedback. With the hand gesture communication there are two sources of mistaken feedback: the first is the previously mentioned time constraint of the users trying to alternate hand gestures that could produce a mismatch with the feedback signals. This problem is different from the human response delay in deciding on an advisement or evaluation that is managed by the Credit Assigner. The second source of wrong information is the classifier of the HGR interface (see details in the Appendix), which does not have a perfect recognition rate. Then, taking this into account, it is possible to say that COACH is more robust to erroneous advice. COACH allows fixing a policy affected by erroneous feedback more easily than with the other interactive frameworks. TAMER type algorithms have an intermediate sensitivity and the teleoperation fashion is the most sensitive to this issue. Actually, the users'

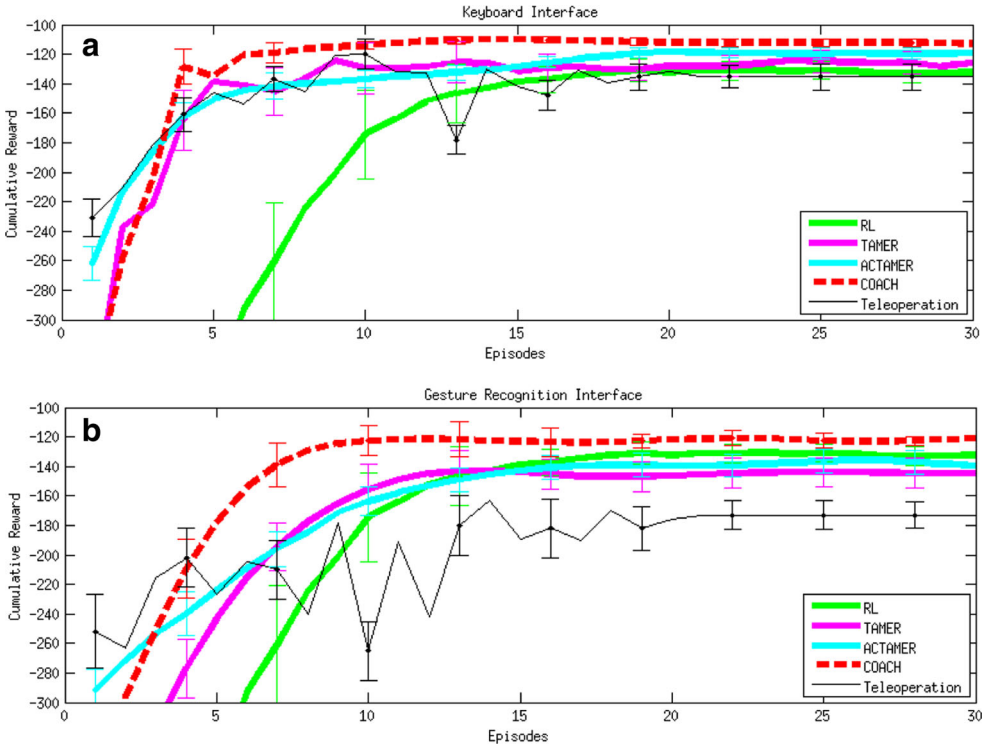

Fig. 5 Average cumulative reward for the mountain car problem. Results using **a** the Keyboard Interface, **b** the Hand Gesture Recognition Interface

performance makes a more considerable relative difference with respect to the other agents.

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.

4.2 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 [3]. 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 [6].

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 Fig. 6. The experiments with interactive agents were finished at episode 150 with a maximum of 5000 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.

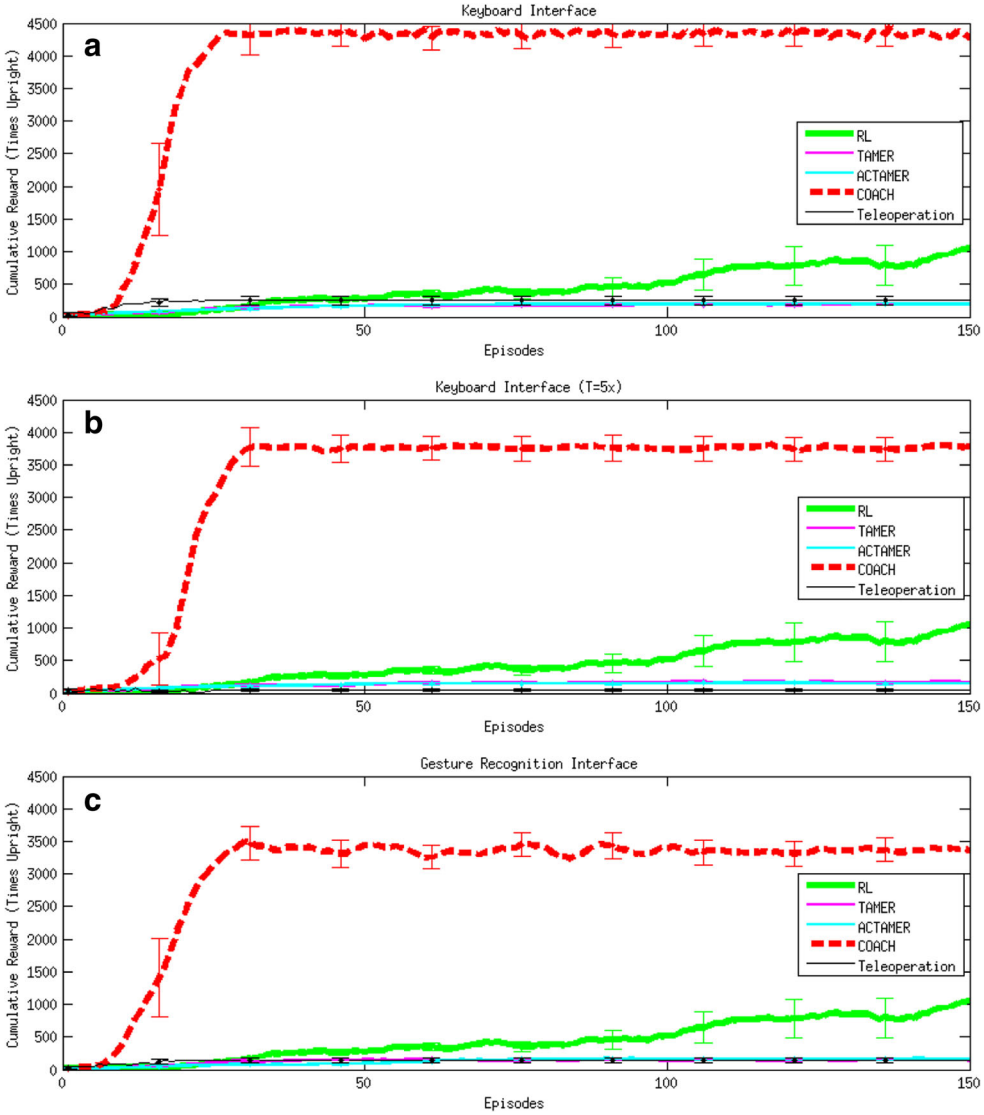

Fig. 6 Average cumulative reward for the cart-pole problem. Results using **a** the Keyboard Interface, **b** the Keyboard Interface with environment 5 times faster, **c** the Hand Gesture Recognition Interface

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 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.

4.3 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 [4] 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 paper, two experimental procedures for validating COACH with this task are presented. First, the simple hybrid scheme of [4] 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 over real game scenarios.

1D Ball-Dribbling The hybrid approach proposed in [4] 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 [4] is introduced here; it incorporates a parameter that defines a security robot-ball distance ρ_{max} that must not be exceeded:

$$r = \begin{cases} 100 + v_x, & \rho \leq \rho_{max} \\ -100 - (\rho - \rho_{max}) + v_{x'}, & \rho > \rho_{max} \end{cases} \quad (16)$$

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 Fig. 7. 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.

3D 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 [4] 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 velocity v_x, the sideward velocity v_y, and the rotation velocity 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

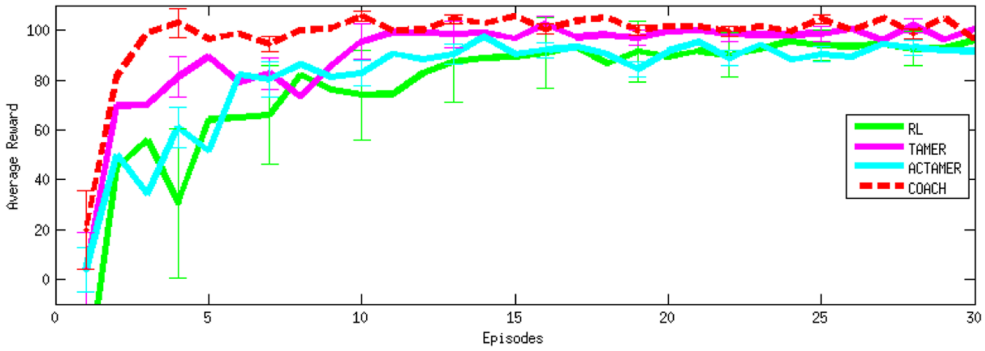

Fig. 7 Average cumulative reward for the 1-D ball dribbling problem

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 Fig. 8). 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 $\%d_i$. This index measures how efficient the path the robot walks is while it is dribbling the ball:

$$\%d_i = \left(\frac{d_r}{(d_{rb} + d_{bt})} - 1\right) \times 100\% \tag{17}$$

where d_r is the complete distance walked by the robot until it reaches the target, d_{rb} is 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 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). Three keys are intended for selecting the action dimension, and two to advise the increase or decrease signals.

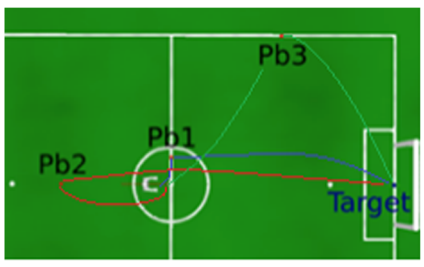

Fig. 8 Soccer field with the robot's initial position (Pbi), and the initial positions of the ball for the evaluation scenarios: Scenario 1 (blue), Scenario 2 (red), Scenario 3 (green)

In Fig. 9 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 9a 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, Fig. 9b and c show a greater reduction (around 50%) of the percentage of cumulated time of faults, and the percentage of increased walking distance when the 3D model is used, for the three scenarios evaluated. 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.

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 competitions, allowing the team to achieve fourth place for these last two participations in the SPL (Standard Platform League), where humanoid robots are used.[1] It is worth clarifying that when playing soccer, the

[1] See results in http://www.informatik.uni-bremen.de/spl/bin/view/Website/Results2015 and http://www.informatik.uni-bremen.de/spl/bin/view/Website/Results2016

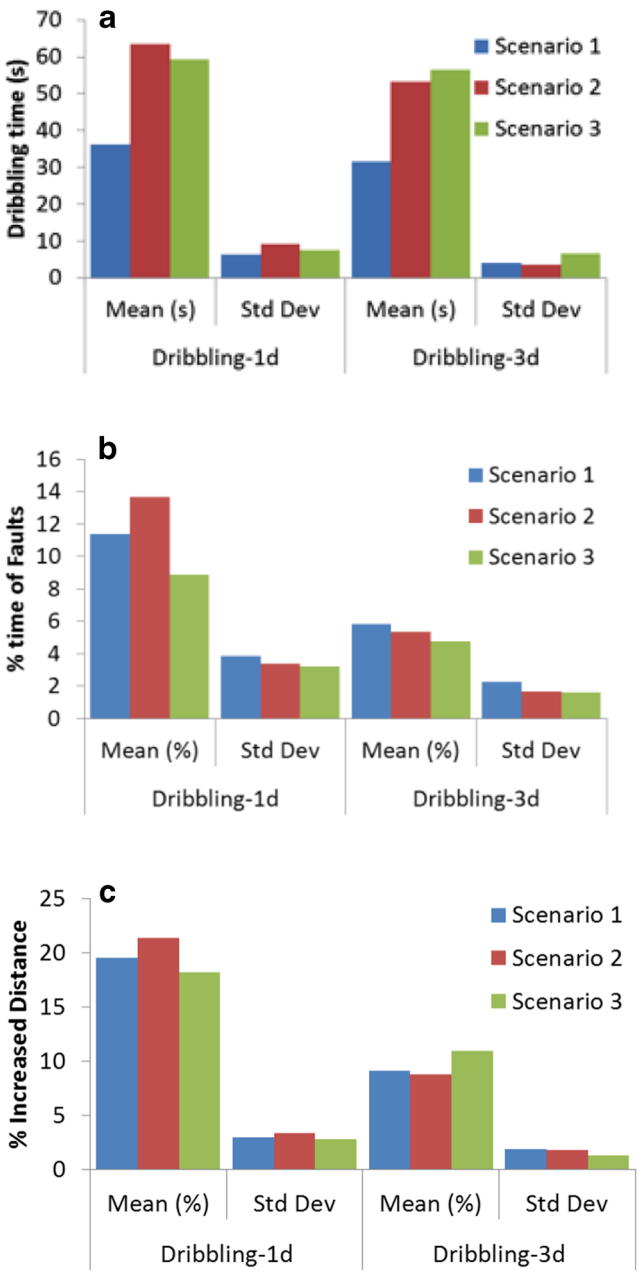

Fig. 9 Statistics for each scenario: **a** ball-dribbling time, **b** percentage of cumulated time of faults, **c** percentage of increased walking distance

dribbling engine has a variant target, computed at a high-level behavior level, in order to avoid obstacle collisions. In Fig. 10 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.

4.4 Learning to Balance on a Bicycle

This task, proposed in [5], is about learning to balance a simulated bike while it is ridden with constant velocity. The agent observes the angle of the handle bar θ, its angular velocity $\dot{\theta}$, the angle ω the bicycle is tilted from the vertical, its velocity $\dot{\omega}$, and its acceleration $\ddot{\omega}$ (see the definition of variables in Fig. 11). 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 $[-2\text{N}, 2\text{N}]$, whereas the action d is in the range $[-2\text{cm}, 2\text{cm}]$. Action d has an added uniform random noise between $[-2\text{cm}, 2\text{cm}]$. 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 only the keyboard interface. Since in the first three problems presented in this work the comparisons demonstrated that COACH outperforms the TAMER type agents with a large difference, in this problem, the experiments are focused on comparing the teleoperation curve vs the COACH learning. The SARSA agents are kept as the reference point of comparison, 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 [56]; 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 [56], 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 8575 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 8575 parameters for each of the nine actions.

In order to compile significant results statistically, 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

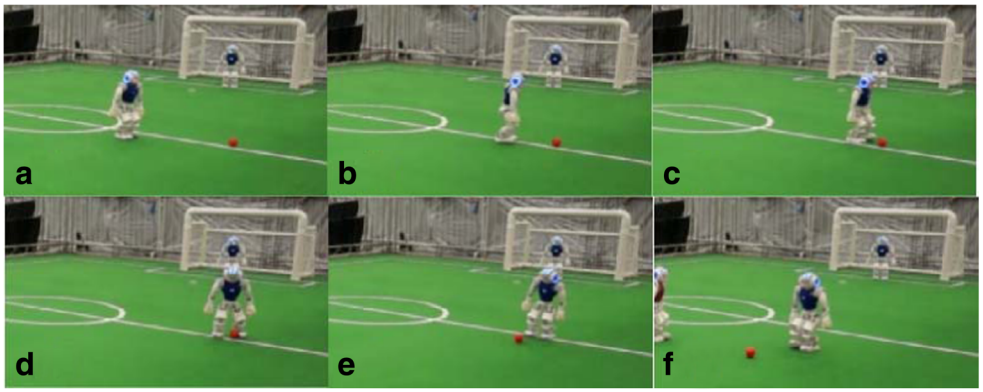

Fig. 10 Sequence of ball-dribbling with humanoid robots in a game at RoboCup

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 1000. 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 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 12 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 1000 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 1000 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.

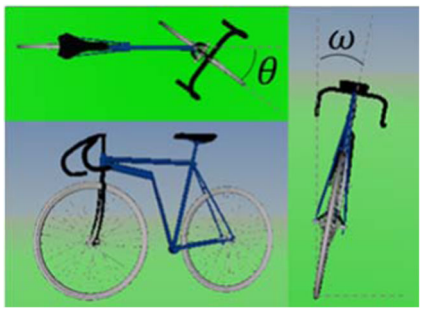

Fig. 11 3-D simulated bicycle environment and the angles observed by the agent

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 Fig. 12 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.

4.5 Ablation Study

In Section 3 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.

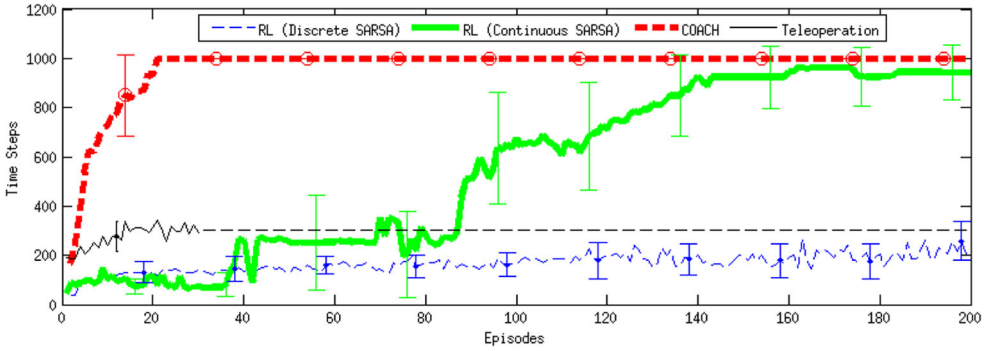

Fig. 12 Average learning curves of the "learning to balance on a bicycle" problem

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:

$$h = sign(P_T(s) - P(s)) \tag{18}$$

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 Fig. 13 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 14 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 (5000). 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.

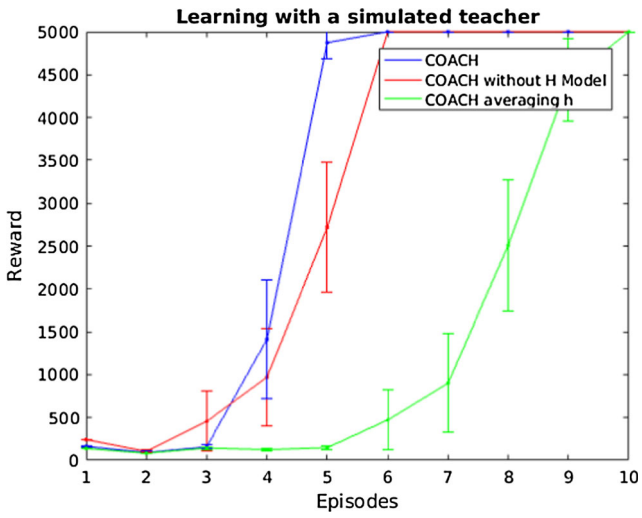

Fig. 13 Ablation study. Convergence curves of COACH showing the influence of the *Human Feedback Modeling* module for learning the cart pole problem

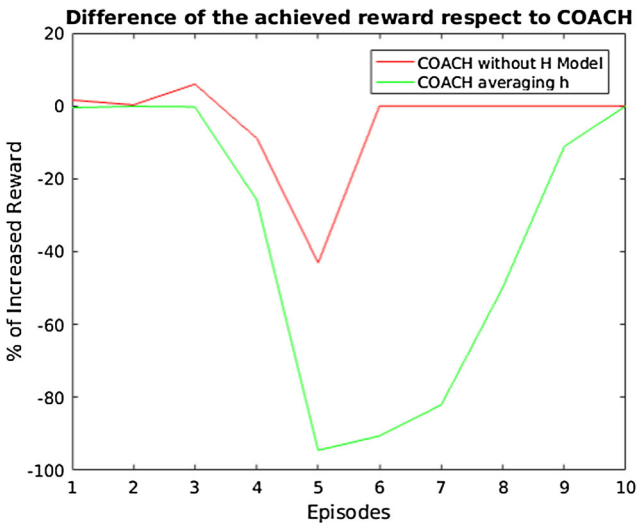

Fig. 14 Ablation study. Percentage of increased reward regarding the original COACH for learning the cart pole problem

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.

5 Conclusions

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.

The experiments performed in this work 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 or 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 toward 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.

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.

The experiments with the ball dribbling by humanoid robots problem: First, in this problem COACH also outperforms the learning capacities of the autonomous and the TAMER agents. Second, 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.

As part of our future 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.

Acknowledgements This work was partially funded by FONDECYT project 1161500 and CONICYT-PCHA/Doctorado Nacional/2015-21151488.

Appendix

Given that human feedback is a key component of the proposed learning framework, a new Hand-Gesture Recognition (HGR) interface that allows providing feedback to the agent is proposed. The interface allows detecting 5 gestures: positive correction, negative correction, a neutral gesture used when users do not need to provide feedback, a reward, and a punishment (see gestures in Fig. 15.)

In order for the proposed system to be robust to variations in illumination, colors, and non-uniform backgrounds, it uses: (i) Gaussian Mixture Models (GMM) and based Background Subtraction (BS) to detect regions of interest (ROI), i.e. hand candidates, (ii) Kalman filtering for tracking the hand candidates, (iii) Local Binary Patterns (LBP) as features for characterizing the ROIs, and (iv) SVM classifiers for the final detection of the hand-gestures. The block diagram is shown in Fig. 16. The main functionalities are described in the following paragraphs:

- *Detection of Regions of Interest* (ROI): Movement blobs are first detected using background subtraction. Then, adjacent blobs are merged and filtered using morphological filters, and the largest blob is selected as a hand candidate and fed to the tracking system.

  In parallel, a second process applies BS to color edges: First, a binary edge image is computed, and then color information is incorporated into the edges. Afterwards, BS and area filtering is applied in the edge's domain. Finally, the output of the area-filtering module is intersected with the color edges in the block "&". In order to manage occlusions properly (see Fig. 16b) the block "&" deletes the blobs associated with the occluded edges, which are labeled by BS as regions with movement (Fig. 15 left); since those edges are not present in the original image. The output is a blob with the detected moving, color edges (Fig. 17 right).
- *Tracking:* The parameters of the bounding box of the largest blob taken as a hand candidate by the prior module are used as observations by a Kalman filter, which estimates the final hand candidates, based on the fusion of the current ROI information with the prior ones. Afterwards, the image computed in the block "&" of the previous module is intersected with the Kalman-filtered bounding box. Examples of the resulting images are shown in Fig. 15.
- *Features Extraction and Classification:* The image window given by the *Tracking* module is analyzed in order to classify the captured gesture. Histograms of LBP features are computed inside the image window. Since this window is a binary image, LBP are used as discretized measurements of the gradient. Then, the histograms of the LBP features are similar to Histograms of Gradient (HOG). This feature vector feeds five SVM classifiers, one trained for each gesture, where the gestures are detected.

The dataset used for training the SVM was built using images generated by the tracking module. Altogether, 1654 images of the five hand-gestures were recorded, 60% of them used for training, and 40% for validation. The classification error is 9.05%, which is considered appropriate to be used as an interface for the learning problems described in Section 4.

Fig. 15 Examples of recognized hand gestures

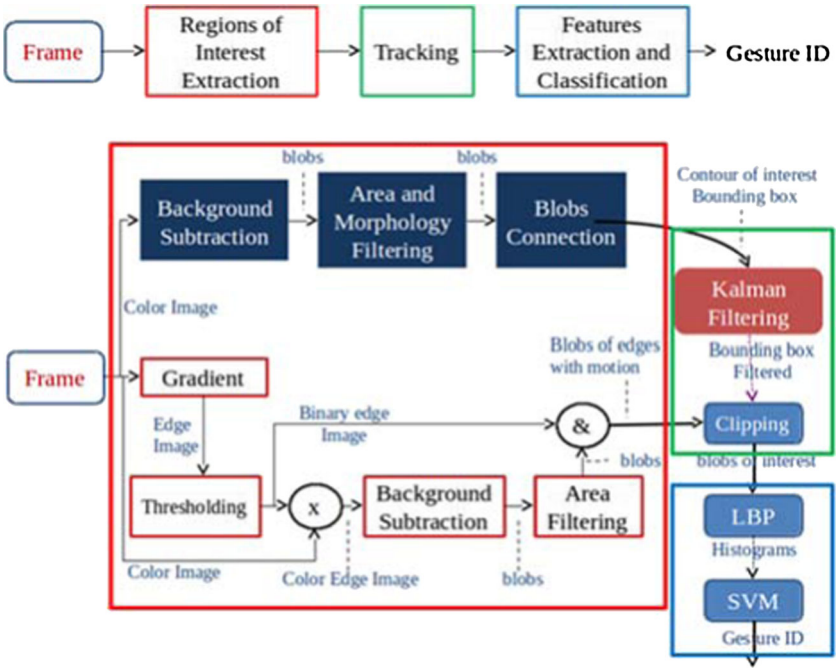

Fig. 16 Hand gesture recognition system. **a** General scheme, **b** Detailed scheme

Fig. 17 Hands occluding edges in the edges domain (left), results of the intersection "&" module (right)

References

1. Knox, W.B., Stone, P.: Interactively shaping agents via human reinforcement: the TAMER framework. In: The Fifth International Conference on Knowledge Capture (2009)
2. Argall, B.D., Browning, B., Veloso, M.: Learning robot motion control with demonstration and advice-operators. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (2008)
3. Sutton, R.S., Barto, A.G.: Reinforcement Learning: an Introduction, vol. 1, no. 1. MIT Press, Cambridge (1998)
4. Leottau, L., Celemin, C., Ruiz-del-Solar, J.: Ball dribbling for humanoid biped robots: a reinforcement learning and fuzzy control approach. In: Robocup 2014: Robot World Cup XVIII, pp. 549–561. Springer (2015)
5. Randløv, J., Alstrøm, P.: Learning to drive a bicycle using reinforcement learning and shaping. In: ICML, vol. 98, pp. 463–471 (1998)
6. Vien, N.A., Ertel, W., Chung, T.C.: Learning via human feedback in continuous state and action spaces. Appl. Intell. **39**(2), 267–278 (2013)
7. Celemin, C., Ruiz-del-Solar, J.: Interactive learning of continuous actions from corrective advice communicated by humans. In: Robocup 2015: Robot World Cup XIX (2015)
8. Celemin, C., Ruiz-del-Solar, J.: COACH: learning continuous actions from corrective advice communicated by humans. In: 2015 International Conference on Advanced Robotics (ICAR), pp. 581–586 (2015)
9. Chernova, S., Thomaz, A.L.: Robot learning from human teachers. Synth. Lect. Artif. Intell. Mach. Learn. **8**(3), 1–121 (2014)
10. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Rob. Auton. Syst. **57**(5), 469–483 (2009)
11. Billard, A., Calinon, S., Dillmann, R., Schaal, S.: Robot programming by demonstration. In: Springer handbook of robotics, pp. 1371–1394. Springer (2008)
12. Billing, E.A., Hellström, T.: A formalism for learning from demonstration. Paladyn J. Behav. Robot. **1**(1), 1–13 (2010)
13. Cuayáhuitl, H., van Otterlo, M., Dethlefs, N., Frommberger, L.: 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, pp. 19–28, ACM (2013)
14. Amershi, S., Cakmak, M., Knox, W.B., Kulesza, T.: Power to the people: the role of humans in interactive machine learning. AI Mag. **35**(4), 105–120 (2014)
15. Fails, J.A., Olsen, D.R. Jr.: Interactive machine learning. In: Proceedings of the 8th International Conference on Intelligent User Interfaces, pp. 39–45 (2003)
16. Ware, M., Frank, E., Holmes, G., Hall, M., Witten, I.H.: Interactive machine learning: letting users build classifiers. Int. J. Hum. Comput. Stud. **55**(3), 281–292 (2001)
17. Amershi, S., Fogarty, J., Weld, D.: Regroup: interactive machine learning for on-demand group creation in social networks. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 21–30 (2012)
18. Ngo, H., Luciw, M., Nagi, J., Forster, A., Schmidhuber, J., Vien, N.A.: Efficient interactive multiclass learning from binary feedback. ACM Trans. Interact. Intell. Syst. **4**(3), 1–25 (2014)
19. Aler, R., Garcia, O., Valls, J.M.: Correcting and improving imitation models of humans for robosoccer agents. In: The 2005 IEEE Congress on Evolutionary Computation, 2005, vol. 3, pp. 2402–2409 (2005)
20. Grollman, D.H., Jenkins, O.C.: Learning robot soccer skills from demonstration. In: IEEE 6th International Conference on Development and Learning, 2007. ICDL 2007, pp. 276–281 (2007)
21. Chernova, S., Veloso, M.: Multi-thresholded approach to demonstration selection for interactive robot learning. In: 2008 3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI), pp. 225–232 (2008)
22. Weiss, A., Igelsböck, J., Calinon, S., Billard, A., Tscheligi, M.: Teaching a humanoid: a user study on learning by demonstration with hoap-3. In: The 18th IEEE International Symposium on Robot and Human Interactive Communication, 2009. RO-MAN 2009, pp. 147–152 (2009)
23. Breazeal, C., Berlin, M., Brooks, A., Gray, J., Thomaz, A.L.: Using perspective taking to learn from ambiguous demonstrations. Rob. Auton. Syst. **54**(5), 385–393 (2006)
24. Silver, D., Bagnell, J.A., Stentz, A.: Learning from demonstration for autonomous navigation in complex unstructured terrain. Int. J. Rob. Res. **29**(12), 1565–1592 (2010)
25. Yu, C.-C., Wang, C.-C.: Interactive learning from demonstration with a multilevel mechanism for collision-free navigation in dynamic environments. In: 2013 Conference on Technologies and Applications of Artificial Intelligence (TAAI), pp. 240–245 (2013)
26. Sweeney, J.D., Grupen, R.: A model of shared grasp affordances from demonstration. In: 2007 7th IEEE-RAS International Conference on Humanoid Robots, pp. 27–35 (2007)
27. Lin, Y., Ren, S., Clevenger, M., Sun, Y.: Learning grasping force from demonstration. In: 2012 IEEE International Conference on Robotics and Automation (ICRA), pp. 1526–1531 (2012)
28. Chernova, S.: Interactive policy learning through con?dence-based autonomy (2009).pdf. J. Artif. Intell. Res. **34**, 1–25 (2009)
29. Meriçli, C., Veloso, M., Akin, H.: Complementary humanoid behavior shaping using corrective demonstration. In: 2010 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp. 334–339 (2010)
30. Meriçli, Ç., Veloso, M., Akin, H.: Task refinement for autonomous robots using complementary corrective human feedback. Int. J. Adv. Robot. Syst. **8**(2), 68–79 (2011)
31. Mericli, C.: Multi-Resolution Model Plus Correction Paradigm for Task and Skill Refinement on Autonomous Robots, Citeseer p. 135 (2011)
32. Argall, B.D.: Learning mobile robot motion control from demonstration and corrective feedback. Thesis (2009)
33. Argall, B.D., Browning, B., Veloso, M.M.: Teacher feedback to scaffold and refine demonstrated motion primitives on a mobile robot. Rob. Auton. Syst. **59**(3–4), 243–255 (2011)
34. Meriçli, Ç., Veloso, M.: Improving biped walk stability using real-time corrective human feedback. In: Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6556 LNAI, pp. 194–205 (2011)
35. Akrour, R., Schoenauer, M., Sebag, M.: Preference-based policy learning. In: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), Vol. 6911 LNAI, No. PART 1, pp. 12–27 (2011)
36. Akrour, R., Schoenauer, M., Souplet, J.-C., Sebag, M.: Programming by feedback. In: Proceedings of the 31St International Conference on Machine Learning, vol. 32, pp. 1503–1511 (2014)
37. Christiano, P.F., Leike, J., Brown, T., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences. In: Advances in Neural Information Processing Systems, pp. 4302–4310 (2017)

38. Jain, A., Wojcik, B., Joachims, T., Saxena, A.: Learning trajectory preferences for manipulators via iterative improvement. In: Advances in neural information processing systems, pp. 575–583 (2013)
39. Mitsunaga, N., Smith, C., Kanda, T.: Adapting robot behavior for human – robot interaction. IEEE Trans. Robot. **24**(4), 911–916 (2008)
40. Tenorio-Gonzalez, A.C., Morales, E.F., Villaseñor-Pineda, L.: Dynamic reward shaping: training a robot by voice. In: Advances in Artificial Intelligence–IBERAMIA 2010, No. 214262, pp. 483–492. Springer (2010)
41. León, A., Morales, E.F., Altamirano, L., Ruiz, J.R.: Teaching a robot to perform task through imitation and on-line feedback. In: Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, pp. 549–556 (2011)
42. Suay, H., Chernova, S.: Effect of human guidance and state space size on interactive reinforcement learning. In: RO-MAN, 2011 IEEE, pp. 1–6 (2011)
43. Pilarski, P.M., Dawson, M.R., Degris, T., Fahimi, F., Carey, J.P., Sutton, R.S.: Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In: IEEE International Conference on Rehabilitation Robotics, vol. 2011, p. 5975338 (2011)
44. Yanik, P.M., Manganelli, J., Merino, J., Threatt, A.L., Brooks, J.O., Green, K.E., Walker, I.D.: A gesture learning interface for simulated robot path shaping with a human teacher. IEEE Trans. Human-Machine Syst. **44**(1), 41–54 (2014)
45. Najar, A., Sigaud, O., Chetouani, M.: Training a robot with evaluative feedback and unlabeled guidance signals. In: IEEE International Symposium on Robot and Human Interactive Communication (ROMAN), pp. 261–266 (2016)
46. Knox, W.B., Stone, P.: TAMER: training an agent manually via evaluative reinforcement. In: 2008 7th IEEE International Conference on Development and Learning, pp. 292–297 (2008)
47. Knox, W.B.: Learning from human-generated reward. In: PhD Dissertation, The University of Texas at Austin (2012)
48. Haykin, S.: Neural networks: a comprehensive foundation. Knowl. Eng. Rev. **13**, 4 (1999)
49. Vien, N.A., Ertel, W.: Reinforcement learning combined with human feedback in continuous state and action spaces. In: 2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL), pp. 1–6 (2012)
50. Thomaz, A., Hoffman, G., Breazeal, C.: Reinforcement learning with human teachers: understanding how people want to teach robots. In: Proceedings - IEEE International Workshop on Robot and Human Interactive Communication, pp. 352–357 (2006)
51. Toris, R., Suay, H.B., Chernova, S.: A practical comparison of three robot learning from demonstration algorithms. In: 2012 7th ACM/IEEE International Conference on Human-Robot Interact. (HRI), pp. 261–262 (2012)
52. Busoniu, L., Babuska, R., De Schutter, B., Ernst, D.: Reinforcement Learning and Dynamic Programming using Function Approximators, vol. 39. CRC Press (2010)
53. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: a survey. Int. J. Rob. Res. **32**, 1238–1274 (2013)
54. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. IEEE Trans. Syst. Man Cybern. **1**, 116–132 (1985)
55. Babuska, R.: Fuzzy and Neural Control. Disc Course Lecture Notes. Delft University Technology, Delft, Netherlands (2001)
56. Rahat, A.A.M.: Matlab implementation of controlling a bicycle using reinforcement learning. https://bitbucket.org/arahat/matlab-implementation-of-controlling-a-bicycle-using (2010)

Carlos Celemin is a Ph.D. candidate at University of Chile in the Department of Electrical Engineering, member of the Advanced Mining Technology Center, and the UChile Robotics Team that yearly participates in the RoboCup competition. He received the best paper award in the International RoboCup Symposium 2015. His research interests are in machine learning, robotics, human-machine interaction, and decision making systems. He has been a visiting research student in the CORAL Lab at Carnegie Mellon University, the Delft Center for Systems and Control (DCSC) at Delft University of Technology, and the Intelligent Autonomous Systems Group (IAS) at Technical University of Darmstadt.

Javier Ruiz-del-Solar received his diploma in Electrical Engineering and the MS degree in Electronic Engineering from the Technical University Federico Santa Maria (Valparaíso, Chl) in 1991 and 1992, respectively, and the Doctor-Engineer degree from the Technical University of Berlin (Ger) in 1997. In 1998 he joined the Electrical Engineering Department of the Universidad de Chile (Santiago, Chl) as Assistant Professor. In 2001 he became Director of the Robotics Laboratory, in 2005 Associate Professor and in 2011 Full Professor. His research interests include mobile robotics, human-robot interaction, and face analysis. Dr. Ruiz-del-Solar is recipient of the IEEE RAB Achievement Award 2003, RoboCup Engineering Challenge Award 2004,

RoboCup @Home Innovation Award 2007, and RoboCup@Home Innovation Award 2008. Since 2006 he has been a Senior Member of the IEEE, and since 2008 a Distinguished Lecturer of the IEEE Robotics and Automation Society. He is currently Director of the Advanced Mining Technology Center at the Universidad de Chile.