COACH: Learning continuous actions from COrrective Advice Communicated by Humans

Carlos Celemin
University of Chile
Advanced Mining Technology Center &Dept. of Elect. Eng.
Santiago, Chile
carlos.celemin@ing.uchile.cl

Javier Ruiz-del-Solar
University of Chile
Advanced Mining Technology Center &Dept. of Elect. Eng.
Santiago, Chile
jruizd@ing.uchile.cl

***Abstract*— COACH (COrrective Advice Communicated by Humans), a new interactive learning framework that allows non-expert humans to shape a policy through corrective advice, using a binary signal in the action domain of the agent, is proposed. One of the main innovative features of COACH is a mechanism for adaptively adjusting the amount of human feedback that a given action receives, taking into consideration past feedback. The performance of COACH is compared with the one of TAMER (Teaching an Agent Manually via Evaluative Reinforcement), ACTAMER (Actor-Critic TAMER), and an autonomous agent trained using SARSA(λ) in two reinforcement learning problems. COACH outperforms all other learning frameworks in the reported experiments. In addition, results show that COACH is able to transfer successfully human knowledge to agents with continuous actions, being a complementary approach to TAMER, which is appropriate for teaching in discrete action domains.**

***Keywords*— Robot learning, interactive learning, human teachers, human feedback in action domains.**

I. INTRODUCTION

The use of computational/machine learning techniques such as RL (Reinforcement Learning) allows robots, and agents in general, to address complex decision-making tasks. However, one of the main limitations of the use of learning approaches in real-world problems is the large number of learning trials required to learn complex behaviors. This can make non-viable the use of many learning approaches in problems such as autonomous driving, x-copter flight control, or soccer robotics, where the implementation of learning trials with real robots, in the real world, may have a high cost. This drawback can be addressed by using human feedback during learning, i.e., the learning process can be assisted by a human teacher who supervises the agent-environment interaction.

In this context, the main goal of this article is to propose COACH (COrrective Advice Communicated by Humans), a new learning framework that uses human feedback. COACH is inspired on the *shaping* approach [17], which allows interactively training an agent through signals of positive and negative reinforcement. But, as in the *Advice Operators* paradigm [7], the human feedback indicates the agent how the action has to be modified (increased or decreased). Thus, in COACH the human feedback is given in the action domains as in the Advice Operators paradigm [7], but the problem of using an off-line and data-driven based supervised learning process is solved managing the human feedback and the interactive update of the policy (the states to actions mapping) in a similar way as TAMER does [17].

The proposed learning framework is validated and compared with TAMER, ACTAMER [18], and a classical SARSA(λ) method, in two problems: (i) the very well-known *Mountain Car* problem [20], and (ii) ball dribbling with humanoid robots [19]. COACH outperforms all other learning frameworks in all described experiments.

The paper is organized as follows. In Section II some related work is presented. In Section III the COACH learning framework is described. In Section IV an experimental validation of the framework is presented. Finally, in Section V some conclusions of this work are given.

II. RELATED WORK

There are two main schemes for using human feedback in order to modify the policy of a learning agent: human feedback in the actions domain and human feedback in the evaluative domain.

A. Human feedback in actions domains

Learning from Demonstration (LfD) is a learning paradigm in which a teacher provides demonstrations of the desired policy, and the agent reproduces the demonstrated behavior. The actions that the learner associates to the states could be classified in low-level actions for motion control, basic high-level actions (often called action primitives) and complex behavioral actions for high-level control [1]. *Imitation Learning* is a kind of LfD in which the demonstrations are executed in a platform that is different to the one of the learning agent (e.g., a robot). Since some of the most frequent providers of demonstrations are humans, the challenges in this case consist of solving the problems of what and how to imitate [2]. There exists other paradigms such as *Apprenticeship Learning*, in which it is obtained a reward function from the set of demonstrations, then that derived function is used in a RL process [3].

There are algorithms of LfD which are focused in learning from corrective demonstrations, instead of deriving a policy out of the demonstrations, keeping hand-coded algorithms as the primary source of the action policy, and using the

978-1-4673-7509-2/15/$31.00 ©2015 IEEE

demonstration data only to make exceptions as needed [4][5][6].

In most of the described approaches it is required accurate demonstrations and expert users as trainers. Methods based on the *Advice Operator* paradigm do not need an expert trainer, but have the drawback that always deducing the policy from a dataset, which is increased with the set of state-action pairs selected for the correction in every advising phase, increasing the computational burden of the policy re-derivation. Another drawback is that the policy re-derivation, is done off-line, producing that the effect of the advice correction only can be seen after the new policy is obtained.

B. Human feedback in evaluative domains

Under this paradigm, non-expert users can evolve decision making systems, even on-line, by delivering their feedback interactively as an evaluative (approval or disapproval) signal in a RL framework. In this paradigm the reward is partially or completely given by the human [8][9][10][11][12][13].

Since a human reward may have a different meaning than an encoded MDP (Markov Decision Process) reward function, a series of works have analyzed how to model the human feedback [14][15]. The *shaping* approach allows interactively training an agent through signals of positive and negative reinforcement [17]. One of the seminal works based on *shaping* is the TAMER framework (Training an Agent Manually via Evaluative Reinforcement) [16], which address how to use delayed human rewards in RL problems with discrete action domains. Later on, ACTAMER an Actor-Critic approach based on TAMER, addresses RL problems with continuous action domains and using the same kind of feedback [18].

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

III. The COACH Learning Framework

The COACH learning framework uses human binary feedback as a correction in the action domain, in order to update the current policy for the state wherein that action was executed. The trainer has to provide its feedback immediately after the execution of the action to be learnt.

COACH lets the trainer to shape the policy of an agent through occasional feedback. The method updates a policy model based on a supervised learning strategy supported by three main modules: *Human Feedback Modeling*, which characterizes the sequence of human advice, and determines how much feedback must be added to the executed action; *Supervised Learner*, which updates the parameters of the policy model taking into account the human feedback, the executed action and the obtained state, and *Credit Assigner,* which handles the time delay of human feedback.

A. Simple Framework

Figure 1 shows the general structure of the interactive learning framework, which incorporates a *Supervised Learner* for updating the policy model that supports the action selection. In the framework a human trainer observes the agent-environment interaction and provides feedback for teaching the agent.

In the COACH case, 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 difference regarding 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 an advice h. 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 value, respectively). The state $\vec{s}$, the executed action $P(\vec{s})$, and the human feedback h, are taken by the *Supervised Learner* module for updating the parameters of P (weights vector $\vec{w}$). Then, in the next time step, P has a new parameters set. When the trainer does not provide any feedback signal, h is taken as zero. The trainer is only allowed to give a binary correction, because COACH works under the assumption that a person cannot estimate the exact value of an appropriate correction, human just provide a trend of the modification (e.g. more/less force, velocity, energy, etc.). A gradient descent scheme needs a prediction error of the P model (difference between desired action and executed action), but the exact value of the desired action is not available due to the stated assumption. In order to solve this problem, the prediction error e is set as a constant for the whole learning process, and its sign is given by the human feedback signal.

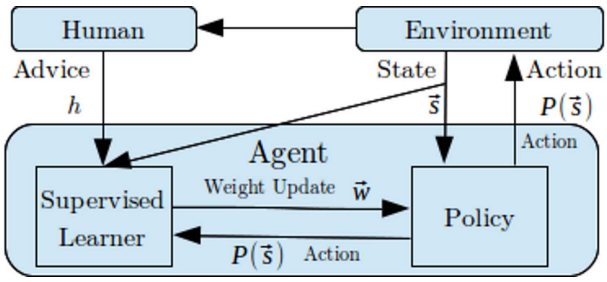

Fig. 1. General Structure of the Learning from Corrective Advice of a human trainer (modified after [16]).

The policy model P can be implemented using any function approximator. COACH uses a linear model of Gaussian features as TAMER and continuous SARSA (in general the Radial Basis Functions (RBF) features are the most used for function approximation in RL [20][21]).

Algorithm 1 states how the simplest version of COACH works. First, the error magnitude e and the learning rate α for the weights update are defined (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 basis functions / features of the policy model $P(\vec{s})$ (line 5). Thus, the agent takes $\vec{s}$ from the state space and maps it into the feature-space for obtaining the features vector $\vec{f}$. $P(\vec{s})$ is obtained as the inner product between $\vec{f}$ and the weight's vector $\vec{w}$ ($P(\vec{s}) = \vec{w}^T \cdot \vec{f}$) (line 6). Afterwards, $P(\vec{s})$ is executed (line 7).

After action execution, the human trainer sends a feedback signal h (line 9), and a new set of parameters $\vec{w}$ is computed

(lines 10-12). In order to compute this, first the prediction *error* of *P*, whose magnitude/sign is given by *e*/*h* is computed (line 11). Then, the weights are updated using the gradient of $P(\vec{s})$ (line 12):

$$w_l = w_l + \left(\alpha \times error \times \frac{\partial P(\vec{s})}{\partial w_l}\right) = w_l + \alpha \times error \times f_l \quad (1)$$

Algorithm 1: General Structure of the Learning from Corrective Advice of a human trainer (simple framework).

```
1:  e ← constant  // error magnitude
2:  α ← constant  // learning rate
3:  while true do
4:    s⃗ ← getStateVec()
5:    f⃗ ← getFeatures(s⃗)
6:    P(s⃗) ← w⃗^T · f⃗
7:    TakeAction( P(s⃗) )
8:    wait for next time step
9:    h ← gethumanCorrectiveAdvice()
10:   if h=! 0
11:     error ← h · e
12:     w_l = w_l + α × error × f_l; l = 1,..,N_feat
13:   end if
14: end while
```

B. Complete Framework

The simple framework described in the former section is enhanced by including two additional functionalities that allow obtaining an algorithm more suitable for human-agent interaction. The first functionality is the modeling of the human feedback that allows estimating a variable error rate of the supervised learning process. The second functionality allows managing the time delay of the feedback signal respect to the evaluated action.

Human Feedback Modeling

The trainer intentions, observed in the binary feedback signal, can be considered a source of information that not only provides the sign (direction) of the corrections, but also its 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. As in the case of the policy model *P*, a linear parameterization of RBF features is used for representing the human feedback model *H*. Therefore, two *Supervised Learner* modules are required in the framework, one for *P* and one for *H* (Figure 3).

In the proposed modeling, sequences of feedback signals with constant sign over a specific state $(\overrightarrow{s_a})$, would mean the trainer suggest a large change of the magnitude of the associated action $P(\overrightarrow{s_a})$. On the other hand, sequences of alternating values of 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 large or a small magnitude of *e* in Algorithm 1. In the complete COACH framework a large value of the *learning* rate allows the trainer to carry out large corrections, while a small value lets him/her to perform fine adjustments.

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:

$$H(\vec{s}) = \vec{v}^T \cdot \vec{f} \quad (2)$$

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 of h and $H(\vec{s})$. Therefore, using the same gradient approach described for the policy model, the weights associated to the *H* model are updated as:

$$v_l = v_l + \beta \times \left(h - H(\vec{s})\right) \times \partial H(\vec{s})/\partial v_l = v_l + \beta \times \left(h - \vec{v}^T \cdot \vec{f}\right) \times f_l \quad (3)$$

with *l* the weight's/feature's index.

Then, the adaptive learning rate (of the policy model learning process) is computed as:

$$\alpha(\vec{s}) = |H(\vec{s})| + bias = \left|\vec{v}^T \cdot \vec{f}\right| + bias \quad (4)$$

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). On the contrary, 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 smaller value when feedback signals of alternating value are received.

Finally, the weights associated to the *P* model are now updated using $\alpha(\vec{s})$ as:

$$w_l = w_l + \alpha(\vec{s}) \times error \times \frac{\partial P(\vec{s})}{\partial w_l} = w_l + \alpha(\vec{s}) \times error \times f_l \quad (5)$$

with *l* the weight's/feature's index.

For demonstrative purposes, Figure 2 shows an example that compares the effect of giving human corrections over time, for a specific state $\overrightarrow{s_a}$, when adaptive and constant learning rates are used. The Figure shows the value of the action when the agent visit $\overrightarrow{s_a}$ and the human-feedback modifies the associated action. It can be observed, that by using an adaptive learning rate, the trainer increased the action until a value of 6 faster than in the case of using a constant value; it takes 4 instead of 7 time steps (Figure 2 (b)). 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 constant case.

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 at each time step, this produce a delay between the action execution and the human response. The *Credit Assigner* proposed in TAMER, approaches this problem by associating the feedback not only to the last state-action pair, but to a past window of pairs. Each pair is weighted with the corresponding probability that characterizes the human delay. COACH uses

the TAMER's credit assigner. Implementation details of this module can be found in [17], and in the text below.

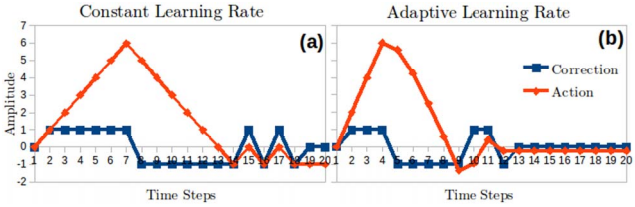

Fig. 2. Human Feedback progress at a specific state $\overrightarrow{s_a}$, and its impact over the respective action, using: a constant learning rate (a), and an adaptive learning rate (b).

Complete Algorithm

The complete structure of COACH is depicted in Figure 3. The new *Human Feedback Modeling*, *Human Feedback Supervised Learner* and *Credit Assigner* modules are shown. 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. The *Supervised Leaners* modules do not read directly the state vector $\vec{s}$, but instead take the features vector provided by the *Credit Assigner* module. This features vector is the average weighted sum of the past features.

Algorithm 2: Learning from Corrective Advice of a human trainer, using an adaptive learning rate for the policy update (complete framework).

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

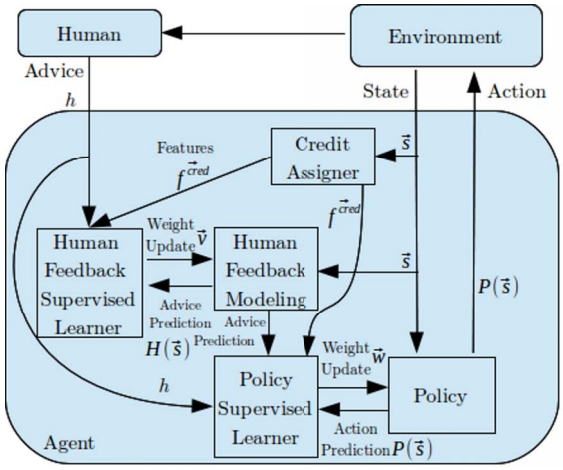

Fig. 3. Structure of the Learning from Corrective Advice of a human trainer, using an adaptive learning rate for the policy update and Credit Assigner.

Algorithm 2 shows the complete version of COACH. This improved version is similar to the simple version (Algorithm 1), except for the credit assignment returning the vector $\overrightarrow{f^{cred}}$ used in following steps (lines 10-13), the update of the human feedback model *H* (lines 15-16), the computation of the variable learning rate (line 17), and the new update of the policy model *P* (line 18-19).

For the credit assignment, the index *t* is varied according with the amount of time steps *T* that compound the history window (line 10); c_t obtains the credit associated with the t^{th} prior time step (line 11), which depends on the probability density function used for modeling the human delay. $\vec{f_t}$ is the features vector computed in line 5, *t* time steps ago, which is weighted by the respective c_t and cumulated in $\overrightarrow{f^{cred}}$ (line 12).

IV. Experiments and Results

The performance of the COACH framework is validated and compared with the ones of TAMER [16], ACTAMER [18], and an autonomous agent trained using SARSA(λ). Two learning problems are used for the comparisons: the Mountain Car [20] and ball dribbling with Humanoid robots [19]. In the first problem, 15 subjects interacted with each interactive learning framework (COACH, TAMER and ACTAMER), while in the second problem 10 subjects trained an agent using each framework.

The participants of the experiment were people from 20 to 39 years old, half of them students of electrical engineering, the other half with varying occupations. At the beginning they watched a video with agents performing correctly the tasks and the learning procedure for each agent; and received the instructions of what to do, 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 two times per problem, and results were recorded. In the case of the SARSA(λ)

method, 50 training runs were executed.

Although, only the SARSA(λ) agent uses an environmental reward function during learning, the cumulative environmental reward and the average environmental reward per episode were computed and used as performance metrics, respectively. In all described experiments, each training run consists of one hundred episodes.

A. Mountain Car

In this classical toy problem a simulated car must get to the top of a hill [20]. The car starts between two steep hills and must go forth and back 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 is divided uniformly using 100 Gaussian features, an amount that is higher than the ones used commonly in the literature. All learning frameworks were tested under the same conditions.

The obtained learning curves -average cumulative reward and the respective confidence intervals- are shown in Figure 4. It can be observed how the interactive frameworks outperform the autonomous learner –SARSA- clearly, both in convergence time and in final performance. The COACH framework outperforms all other methods. It is the fastest method -it converges in two episodes- highlighting its increasing rate at the very first episodes, as the highest among all the approaches, and is the one with the highest final cumulative reward. The second best method is ACTAMER, which reaches a high performance closer to the final in four episodes (since that episode the improvement rate decreases), but its final performance is achieved in the sixteenth episode. TAMER obtains a slightly worse performance than ACTAMER.

B. Ball Dribbling with Humanoid robots

In the context of humanoid robotics soccer, ball dribbling is a relevant navigation problem in which a robot walks to a target as fast as possible by keeping the ball possession. Keeping the ball possession means having the ball close to the robot's feet while walking (description of this behavior in [19]).

In a recent work this problem has been modeled as two simpler tasks: ball pushing, and alignment to the ball and target. The first task reduces the problem to dribbling on a straight line (one dimension), and it is tackled using autonomous RL [19]. A learning 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 robot-ball distance ρ (the only one state), and decides its walking speed (the action). Therefore this problem has a very small continuous state-action space but with high level of uncertainty, due to the fact that the feet motion is not observed by this decision-making system of the robot, it means that for the same state-action pair describing the environment at different times, the outcomes or transitions could be significantly different, the main outstanding property of the problem's complexity.

This work proposes a modification of the reward function used in [19], consisting on incorporating a parameter that defines a security robot-ball distance ρ_{max} that must not be exceeded:

$$r = \begin{cases} 100 + v_x, & \rho \leq \rho_{max} \\ -100 - (\rho - \rho_{max}) + v_x, & \rho > \rho_{max} \end{cases} \quad (6)$$

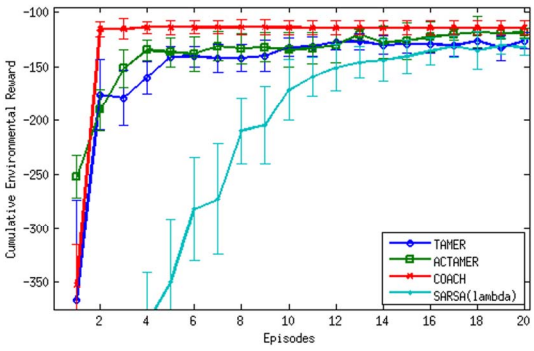

Fig. 4. Average cumulative reward for the Mountain Car problem.

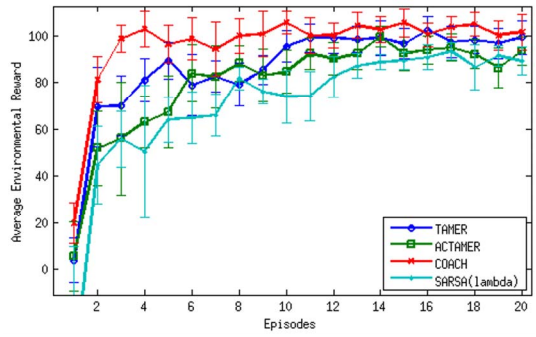

Fig. 5. Average cumulative reward for the Ball Dribbling problem.

This reward function re defines the ball-dribbling task: the goal is to walk as fast as possible, without exceeding the distance ρ_{max}.

The robot speed v_x is 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 in thirty features between 0 and 3 meters, the ρ_{max} parameter was set to 300 mm, then the state-action space consists of 300 features.

The fitness function used for evaluating this problem is also the cumulative reward function, which depends on the action $v_x$and the state ρ. The obtained results are shown in Figure 5. As it can be observed in this figure, the results obtained in this experiment are similar to the one obtained in the Mountain Car problem: the three interactive learning algorithms converged faster than the non-interactive one (SARSA), and COACH achieves the fastest convergence (in three episodes), and the highest average performance. Unlike in the Mountain Car problem, in this problem the second best convergence was obtained by TAMER, which in this problem outperforms ACTAMER and SARSA. The COACH's learning curve is the most stable through the evolution of the policy and presents the smallest sudden decreases of the performance from an

episode to the next.

Table I summarizes the obtained results. The policies shaped with COACH were always more efficient than the others. However, the main advantage of COACH in both problems was the time reduction of the training procedure; a reduction of approximately 73%/64% was obtained in the mountain-car/ball-dribbling problem, when compared with the fastest counterpart (TAMER).

Table I. Algorithms comparison. Note than in the mountain-car/ball-dribbling problem performance is defined negative/positive. EC: Number of episodes for convergence. AP: Average of final performance.

	Mountain Car		**Ball Dribbling**	
	EC	**AP**	**EC**	**AP**
TAMER	11	-125,05	11	96,24
ACTAMER	16	-118.62	11	93,76
COACH	2	-114,18	3	104,82
SARSA	17	-132,29	21	91,63

V. CONCLUSIONS

Interactive learning algorithms have the ability to allow the transfer of knowledge from humans to machines in problems where the human perception could be enough for evaluating a policy. COACH maintains the powerful property of TAMER that considers the behavior of the human feedback and lets to a trainer with no expertise, to shape a policy with a binary signal. COACH sets the feedback in the domain of the actions space as in *LfD* approaches, which is more natural and intuitive than the evaluative feedback domain. Due to the above and the obtained results, it is stated that COACH is an easy-to-use framework for teaching an agent that have to perform continuous actions.

The learning curves of the statistical results show that COACH's convergence is the most stable one. This may be due to: first, there is less uncertainty about the updated policy and its performance of COACH than in evaluative approaches, because for evaluative case, when actions are punished, the modification of the policy is not controlled or guided by the user; second, the inconsistent feedback that an user sometimes could provide, is easier and simpler for fixing with COACH, due to the awareness of the feedback's impact on the policy.

With the reported experiments we have validated the hypothesis that interactive learning frameworks for shaping the policies of learning agents may be the solution in applications where a human is able to observe the agent-environment interaction and to advice the agent.

COACH is a complementary approach to TAMER, while TAMER is appropriate for teaching in discrete domains, COACH is more suitable for continuous domain applications.

ACKNOWLEDGMENT

This work was funded by FONDECYT Project 1130153.

REFERENCES

[1] Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. Robotics and autonomous systems, 57(5), 469-483.

[2] Breazeal, C.; and Scassellati, B. 2002. Robots that imitate humans. Trends in cognitive sciences, 6(11), 481-487.

[3] Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning. ACM.

[4] Chernova, S.; and Veloso, M. 2009. Interactive policy learning through confidence-based autonomy. Journal of Artificial Intelligence Research, 34(1), 1.

[5] Meriçli, C., Veloso, M., and Akin, H. L. 2010. Complementary humanoid behavior shaping using corrective demonstration. In 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids), 334-339. IEEE.

[6] Meriçli, Ç.; Veloso, M.; and Akin, H. L. 2011. Task refinement for autonomous robots using complementary corrective human feedback. International Journal of Advanced Robotic Systems, 8(2), 68.

[7] Argall, B. D., Browning, B., and Veloso, M. 2008. Learning robot motion control with demonstration and advice-operators. In IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2008, 399-404. IEEE.

[8] Mitsunaga, N.; Smith, C.; Kanda, T.; Ishiguro, H.; and Hagita, N. 2008. Adapting Robot Behavior for Human--Robot Interaction. In IEEE Transactions on Robotics, 24(4), 911-916. IEEE.

[9] Tenorio-Gonzalez, A. C., Morales, E. F., and Villaseñor-Pineda, L. 2010. Dynamic reward shaping: training a robot by voice. In Advances in Artificial Intelligence–IBERAMIA, 483-492. Springer Berlin Heidelberg.

[10] León, A., Morales, E. F., Altamirano, L., and Ruiz, J. R. 2011. Teaching a robot to perform task through imitation and on-line feedback. In Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, 549-556. Springer Berlin Heidelberg.

[11] Suay, H. B., and Chernova, S. 2011. Effect of human guidance and state space size on interactive reinforcement learning. In RO-MAN 2011 IEEE, 1-6. IEEE.

[12] Pilarski, P. M., Dawson, M. R., Degris, T., Fahimi, F., Carey, J. P., and Sutton, R. S. 2011. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In IEEE International Conference on Rehabilitation Robotics (ICORR), 1-7. IEEE.

[13] Yanik, P. M., Manganelli, J., Merino, J., Threatt, A. L., Brooks, J. O., Green, K. E., and Walker, I. D. 2014. A Gesture Learning Interface for Simulated Robot Path Shaping With a Human Teacher. In IEEE Transactions on Human-Machine Systems, 41-54. IEEE.

[14] Thomaz, A. L., Hoffman, G., and Breazeal, C. 2006. Reinforcement learning with human teachers: Understanding how people want to teach robots. In The 15th IEEE International Symposium on Robot and Human Interactive Communication, ROMAN 2006, 352-357. IEEE.

[15] Thomaz, A. L., and Breazeal, C. 2007. Asymmetric interpretations of positive and negative human feedback for a social learning agent. In The 16th IEEE International Symposium on Robot and Human interactive Communication, RO-MAN 2007, 720-725. IEEE.

[16] Knox, W. B., and Stone, P. 2008. TAMER: Training an agent manually via evaluative reinforcement. In 7th IEEE International Conference on Development and Learning, ICDL 2008, 292-297. IEEE.

[17] Knox, W. B., and Stone, P. 2009. Interactively shaping agents via human reinforcement: The TAMER framework. In Proceedings of the fifth international conference on Knowledge capture 9-16. ACM.

[18] Vien, N. A.; Ertel, W.; and Chung, T. C. 2013. Learning via human feedback in continuous state and action spaces. Applied intelligence, 39(2), 267-278.

[19] Leottau, L., Celemin, C., and Ruiz-del-Solar, J. 2014. Ball Dribbling for Humanoid Biped Robots: A Reinforcement Learning and Fuzzy Control Approach. In RoboCup 2014: Robot World Cup XVIII.

[20] Sutton, R.S., and Barto, A.G. 1998. Reinforcement Learning: an introduction. MIT press, Cambridge.

[21] Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D. 2010. Reinforcement learning and dynamic programming using function approximators. CRC press.