Interactive Learning of Temporal Features for Control

Rodrigo Pérez-Dattari[1], Carlos Celemin[1], Giovanni Franzese[1], Javier Ruiz-del-Solar[2] and Jens Kober[1]

I. Introduction

The ongoing industry revolution is demanding more flexible products, including robots in household environments or medium scale factories. Such robots should be able to adapt to new conditions and environments, and to be programmed with ease. As an example, let us suppose that there are robot manipulators working in an industrial production line that need to perform a new task. If these robots were hard coded, it could take days to adapt them to the new settings, which would stop the production of the factory. Easily programmable robots by non-expert humans would speed up this process considerably.

In this regard, we present a framework in which robots are capable to quickly learn new control policies and state representations, by using occasional corrective human feedback. To achieve this, we focus on interactively learning these policies from non-expert humans that act as teachers.

We present a Neural Network (NN) architecture, along with an Interactive Imitation Learning (IIL) method, which efficiently learns spatiotemporal features and policies from raw high dimensional observations (raw pixels from an image), for tasks in environments not fully temporally observable.

We denominate IIL as a branch of Imitation Learning (IL) where human teachers provide different kinds of feedback to the robots, like new demonstrations triggered by robot queries [1], corrections [2], preferences [3], reinforcements [4], etc. Most IL methods work under the assumption of learning from perfect demonstrations; therefore, they fail when teachers only have partial insights in the task execution. Non-expert teachers could be considered all the users who are neither Machine Learning (ML)/control experts, nor skilled to fully show the desired behavior of the policy.

Interactive approaches like COACH [5], and some Interactive Reinforcement Learning (IRL) approaches [4], [6], are intended for non-expert teachers, but are not completely deployable for end-users. Sequential decision-making learning methods (IL, IIL, IRL, etc.) rely on good state representations, which make the shaping of the policy landscape simple, and provide good generalization properties. However, this requirement brings the need of experts on feature engineering to pre-process the states properly, before running the learning algorithms.

[1]R. Pérez-Dattari, C. Celemin, G. Franzese and J. Kober are with the Department of Cognitive Robotics, Delft University of Technology, Delft, The Netherlands e-mail: (r.j.perezdattari, c.e.celeminpaez, g.franzese, j.kober)@tudelft.nl.

[2]J. Ruiz-Del-Solar is with the Department of Electrical Eng. and the Advanced Mining Technology Center, Universidad de Chile, Chile e-mail: jruizd@ing.uchile.cl.

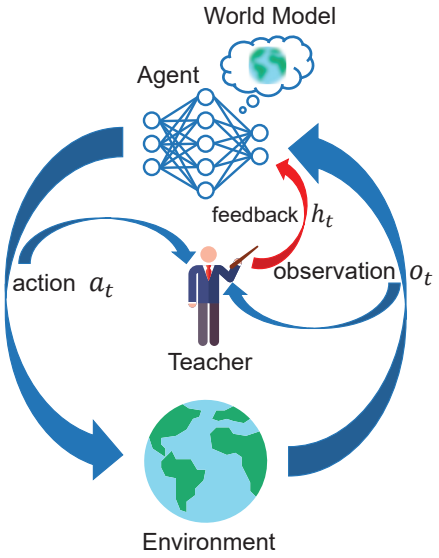


Fig. 1: Interactively shaping policies with agents that model the world.

The inclusion of Deep Learning (DL) in IL (given its popularity gained in the field of Reinforcement Learning (RL) [7]), allows to skip pre-processing modules for the input of the policies, since some architectures of NNs endow the agents with intrinsic feature extraction capabilities. This has been exhaustively tested in end-to-end settings [7]. In this regard, DL allows non-expert humans to shape policies based only on their feedback.

Nevertheless, in real-world problems, we commonly face tasks wherein the observations do not explain the complete state of the agent due to the lack of temporal information (e.g. rates of change), or because the agent-environment interaction is non-Markovian (e.g. dealing with occlusions). For these cases, it is necessary to provide memory to the learning policy. Recurrent Neural Networks (RNNs) can learn to model dependencies on the past, and map them to the current outputs. This recurrency has been used in RL and IL mostly using Long Short-Term Memory (LSTM) networks [8].

Therefore, LSTMs are included in our NN architecture to learn temporal features, which contain relevant information from the past. However, DL algorithms require large amounts of data, so as a way to tackle this shortcoming, State Representation Learning (SRL) has been used to learn features more efficiently [9], [10]. Considering that real robots and human users have time limitations, as an SRL strategy, a model of the world is learned to obtain state representations that make the policy convergence possible within feasible training time

intervals (see Fig. 1).

The combination of SRL and the teacher's feedback is a powerful strategy to efficiently learn temporal features from raw observations in non-Markovian environments.

The experiments presented in this paper show the impact of the proposed architecture in terms of data efficiency and policy final performance within the Deep COACH (D-COACH) IIL framework [11]. Additionally, the experimental procedure shows that the proposed architecture could be even used with other IL methods, such as Data Aggregation (DAgger) [12]. The code used in this paper can be found at: https://github.com/rperezdattari/Interactive-Learning-of-Temporal-Features-for-Control.

The paper is organized as follows: background on approaches used within our proposed method, and the related work are presented in Section II. Section III describes the proposed NN architecture along with the learning method. Experiments and results are given in Section IV, and finally the conclusions are drawn in Section V.

II. Background and Related Work

Our method combines elements from SRL, IL and memory in NN models to build a framework that enables non-expert teachers to interactively shape policies in tasks with non-Markovian environments. These elements are introduced hereunder.

A. Dealing with non-Markovian Environments

There are different reasons why a process could be partially observable. One of them is when the state describes time-dependent phenomena, but the observation only contains partial information of it. For instance, velocities cannot be estimated from camera images unless observations from different time steps are combined. Other examples of time-dependent phenomena are temporary occlusions or corrupted communication systems between the sensors and the agent.

For these environments, the temporal information needs to be implicitly obtained within the policy model. There are two well-known approaches for adding memory to agents in sequential decision-making problems when using NNs as function approximators:

1) **Observation stacking policies [13]**: stacking the last N observations $(o_t, o_{t-1}, ...o_N)$, and using this stack as the input of the network.
2) **Recurrent policies [14]:** including RNN layers in the policy architecture.

One of the main issues of observation stacking is that the memory of these models is determined by the number of stacked observations. The overhead increase rapidly for larger sequences in high-dimensional observation problems.

In contrast, RNNs have the ability to model information for an arbitrarily long amount of time [15]. Also, they do not add input-related overheads, because when these models are evaluated, they only use the last observation. Therefore, RNNs have lower computational cost than observation stacking. Given the more practical usage of recurrent models and their capability of representing arbitrarily long sequences, in this work we use RNN-based policies (with LSTM layers) in the proposed NN architecture.

Nevertheless, the use of LSTMs has a critical disadvantage, since its training is more complex and requires more data, something very problematic when considering human teachers and real systems. We will now introduce SRL, which helps to accelerate the LSTM converge.

B. State Representation Learning

In most of the problems faced in robotics, the state s_t, which fully describes the situation of the environment at time step t, is not fully accessible from the robot's observation o_t. As mentioned before, in several problems the observations lack temporal information required in the state description. Evenmore, these observations tend to be raw sensor measurements that can be high-dimensional, highly redundant and ambiguous. A portion of this data may even be irrelevant.

As a consequence, to successfully solve these problems a policy needs to 1) find temporal correlations between several consecutive observations, and 2) extract relevant features from observations that are hard to interpret. However, finding relations between these large data structures with the underlying phenomena of the environment, while learning controllers, can be extremely inefficient. Therefore, efficiently building controllers on top of raw observations requires to learn informative low-dimensional state representations [16]. The objective of SRL is to obtain an observer capable of generating such representations.

A compact representation of a state is considered to be suitable for control if the resulting state representation:

- is Markovian,
- has good generalization to unseen states, and
- is defined in low dimensional space (considerably lower than the actual observation dimensionality) [9].

Along with the control objective function (e.g. reward function, imitation cost function), other objective functions can be used for SRL [10], namely:

- observation reconstruction,
- forward model or next observation prediction,
- inverse model,
- reward function, or
- value function.

C. Interactive Learning methods

This subsection introduces briefly two approaches for interactively learning from human teachers while agents are executing the task.

1) Data Aggregation: (HG-)DAgger

DAgger [12] is an IIL algorithm that aims to collect data with online sampling. To achieve this, trajectories are generated by combining the agent's policy π_θ and the expert's policy. The observations o_t and the demonstrator's corresponding actions

Algorithm 1 (HG-)DAgger

Require: demonstrations database $\mathcal{D}$ with initial demonstrations, policy update frequency b
for t = 1,2,... **do**
  if mod(t, b) is 0 **then**
    update π_θ from $\mathcal{D}$
  observe state o_t
  select action from agent or expert
  execute action
  feedback provide label a_t^* for o_t, if necessary
  aggregate (o_t, a_t^*) to $\mathcal{D}$

a_t^* are paired and added to a database $\mathcal{D}$, which is used for training the policy's parameters θ iteratively in a supervised learning manner, in order to asymptotically approach the expert's policy. At the beginning of the learning process, the demonstrator has all the influence over the trajectory made by the agent; then, the probability of following the demonstrator's actions decays exponentially.

For working in real-world systems, with humans as demonstrators, a variation of DAgger, Human-Gated DAgger (HG-DAgger) [2], was introduced. In this approach, the demonstrator is not expected to give labels over every action of the agent, but only in places where s/he considers that the agent's policy needs improvement. Only these labels are aggregated to the database and used for updating the policy. Additionally, every time feedback is given by the human, the policy will follow the provided action. As a safety measure, in HG-DAgger the uncertainty of the policy over the observation space is estimated; this element is omitted in this work. Algorithm 1 shows the general structure of DAgger and HG-DAgger.

2) Deep COACH

In this framework [11], humans shape policies giving occasional corrective feedback over the actions executed by the agents. If an agent takes an action that the human considers to be erroneous, then s/he would indicate with a binary signal h_t, the direction in which the action should be modified.

This feedback is used to generate an error signal for updating the policy parameters θ. It is done in a supervised learning manner with the cost function J using the mean squared error and stochastic gradient descent. Hence, the update rule is:

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta J(\theta). \tag{1}$$

The feedback given by the human only indicates the sign of the policy error. Its magnitude is supposed to be unknown, since the algorithm works under the assumption that the user is non-expert; therefore, s/he does not know the magnitude of the proper action. Instead, the error magnitude is defined as the hyperparameter e, that must be defined before starting the learning process. Thus, the policy $error_t$ is defined by $h_t \cdot e$.

To compute a gradient in the parameter space of the policy, the error needs to be a function of θ. This is achieved by observing that:

Algorithm 2 Deep COACH

Require: error magnitude e, buffer update interval b
Init: $\mathcal{B} = []$ *# initialize memory buffer*
for t = 1,2,... **do**
  observe state o_t
  execute action $a_t = \pi_\theta(o_t)$
  feedback human corrective advice h_t
  if h_t is not **0 then**
    $error_t = h_t \cdot e$
    $a_{target(t)} = a_t + error_t$
    update π using SGD with pair $(o_t, a_t^{\text{target}})$
    update π using SGD with a mini-batch sampled from $\mathcal{B}$
    append $(o_t, a_t^{\text{target}})$ to $\mathcal{B}$
  if mod(t, b) is 0 **then**
    update π_θ using SGD with a mini-batch sampled from $\mathcal{B}$

$$error_t(\theta) = a_t^{\text{target}} - \pi_\theta(o_t) \tag{2}$$

where a_t^{target} is the incremental objective generated by the feedback of the human $a_t^{\text{target}} = a_t + error_t$ and a_t is the current output of the policy π_θ. From Equations (1), (2), and the derivative of the mean squared error, we can get the **COACH update step**:

$$\theta \leftarrow \theta + \alpha \cdot error_t \cdot \nabla_\theta \pi_\theta. \tag{3}$$

To be more data efficient and to avoid locally over-fitting to the most recent corrections, Deep COACH has a memory buffer that stores the tuple $(o_t, a_t^{\text{target}})$ and replays this information during learning. Additionally, when working in problems with high-dimensional observations, an autoencoding cost is incorporated in Deep COACH as an observation reconstruction SRL strategy. In the Deep COACH pseudo-code (Algorithm 2) this SRL step is omitted. Deep COACH learns everything from scratch in only one interactive phase, unlike other deep interactive RL approaches [4], [6], which split the learning process into two sequential learning phases. First, recording samples of the environment for training a dimensionality reduction model (e.g. an autoencoder); secondly, using that model for the input of the policy network during the actual interactive learning process.

III. Learning temporal features based on interactive teaching and world modelling

In this section, the SRL NN architecture is described along with the interactive algorithm for policy shaping.

A. Network Architecture for Extracting Temporal Features

For approaching problems that lack temporal information in the observations, the most common solution is to model the policy with RNNs as discussed in Section II-A; therefore, we propose to shape policies that are built on top of RNNs, with

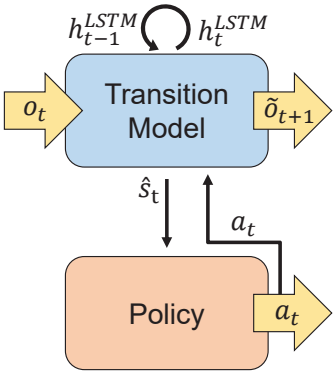


Fig. 2: Transition model and policy general structure.

occasional human feedback. In this work, we are using the terms world model and transition model interchangeably.

IIL methods can take advantage of SRL for training with other objective functions by 1) making use of all the experience collected in every time step, and 2) boosting the process of finding compact Markovian embeddings. We propose to have a neural architecture separated into two parts: 1) transition model, and 2) policy. The transition model is in charge of learning the dynamics of the environment in a supervised manner using samples collected by the agent. The policy part is shaped only using corrective feedback. Figure 2 shows a diagram of this architecture.

Learning to predict the next observation o_{t+1} forces a Markovian state representation. This has been successfully applied in RL [17]. RNNs can encode information from past observations in their hidden state h_t^{LSTM}. Thus, the objective of the first part of the neural network is to learn $\mathcal{M}(o_t, a_t, h_{t-1}^{\text{LSTM}}) = \widetilde{o}_{t+1}$, which, as a consequence, learns to embed past observations in h_t^{LSTM}. Additionally, when the observations are high-dimensional (raw images), the agents also need to learn to compress spatial information. To achieve this, a common approach is to compress this information in the latent space of an autoencoder.

For the first part of the architecture, we propose to use the combination of an autoencoder with an LSTM to compute the transition function model, i.e., predicting the next high-dimensional observation. A detailed diagram of this architecture can be seen in Figure 3.

In the second part of the architecture, the policy takes as input, a representation of the state $\hat{s}_t$, that is generated inside the transition model network. This representation is obtained at the output of a fully-connected layer (FC3), that combines the information of h_{t-1}^{LSTM} with the encoder compression of the current observation $e(o_t)$. This is achieved by adding a skipping connection between the output of the encoder and the output of the LSTM.

B. Interactive Algorithm for Policy and World-Model Learning

In Algorithm 3, the pseudo-code of the state representation learning strategy is presented. The hidden state of the LSTM is denoted as h^{LSTM}, and the human corrective feedback as h. In every time step, a buffer $\mathcal{D}$ stores the samples of the transitions with sequences of length τ (line 5). The agent executes an action based on its last observation and the current hidden state of the LSTM (line 6). This hidden state is updated using its previous value and the most recent observation and action (line 7). Line 8 captures the occasional feedback of the teacher, which could be a relative correction when using Deep COACH, or the corrective demonstration when using HG-DAgger. Also, depending on the learning algorithm, the policy is updated in different ways (line 9) .

$\mathcal{D}$ replays past transitions of the environment in order to update the transition function model (line 11). This is done following the *bootstrapped random updates* [14] strategy. This model is updated every d time steps.

Algorithm 3 Online Temporal Feature Learning

1: **Require:** Policy update algorithm π^{update}, training sequence length τ, model update rate d
2: **Init:** $\mathcal{D} = []$
3: **for** t = 1,2,... **do**
4: **observe** observation o_t
5: **append** $(o_{t-1}, ..., o_{t-\tau}, a_{t-1}, ..., a_{t-\tau}, o_t)$ to $\mathcal{D}$
6: **execute** action $a_t = \pi_\theta(o_t, h_{t-1}^{\text{LSTM}})$
7: **compute** h_t^{LSTM} from $\mathcal{M}(o_t, a_t, h_{t-1}^{\text{LSTM}})$
8: **feedback** human feedback h_t
9: **call** $\pi^{\text{update}}(o_t,\ a_t,\ h_t)$
10: **if** mod(t, d) is 0 **then**
11: **update** $\mathcal{M}$ using SGD with mini-batches of sequences sampled from $\mathcal{D}$

IV. Experiments and Results

In this section, experiments for validating the proposed neural network architecture and the interactive training algorithm are presented. In order to obtain a thorough evaluation, different experiments are carried out to compare and measure the performance (return i.e. sum of rewards) of the proposed components. Initially, the network architecture based on SRL is evaluated in an ablation study, aiming to quantify the data efficiency improvement added by its different components. Then, using the proposed architecture, D-COACH is compared with (HG-)DAgger using simulated tasks and simulated teachers (oracles). The third set of experiments is carried out with human teachers in simulated environments, again comparing different learning methods. Finally, a fourth set of validation experiments is carried out in real systems with human teachers. Most of the results are presented in this paper; however, some of them are in the supplementary material, along with more detailed information on the experiments

Two real and three simulated environments with different complexity levels were used, all of them using raw images as observations. The simulated environments are Mountain-Car, Swing-Up Pendulum, and Car Racing, whose implementations are taken from OpenAI Gym [18]. These simulations provide rendered image frames as observations of the environment. These frames visually describe the position of the system but

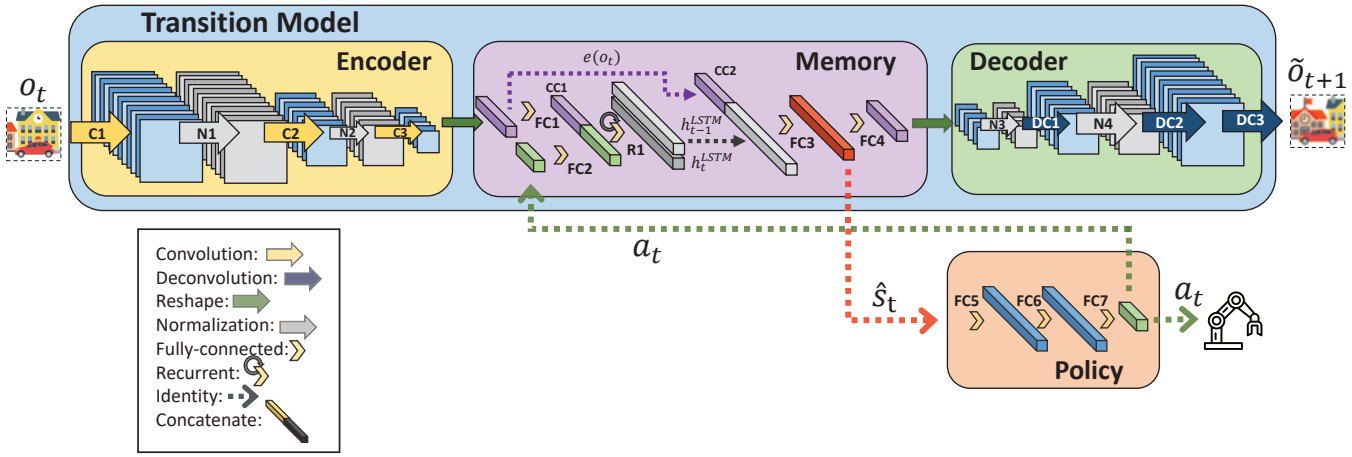


Fig. 3: Proposed neural network architecture. Convolutional and recurrent (LSTM) layers are included in the transition model in order to learn spatiotemporal state representations. The estimated state $\hat{s}_t$ is used as input to the policy, which is a fully-connected NN.

not its velocity, which is necessary to control the system. The experiments on the real physical systems consist of a Swing-Up Pendulum and a setup for picking oranges on a conveyor belt with a 3 degrees of freedom (DoF) robot arm.

The metrics used for the comparisons are the achieved final policy performance, and the speed of convergence, which is very relevant when dealing with both, real systems and human teachers. A video showing most of these experiments can be found at: youtu.be/4kWGfNdm21A.

A. Ablation study

In this ablation study, the architecture of the network is the independent variable of the study. Three independent comparisons were carried out using DAgger, HG-DAgger and D-COACH. The training sessions were run using a simulated teacher to avoid any influence of human factors.

Three different architectures were tested for learning the policy from an oracle. The structure of the networks is introduced below:

1) **Full network:** Proposed architecture.
2) **Memoryless state representation learning (M-less SRL):** Similar to the full network, but without using recurrence between the encoder and decoder. The autoencoder is trained using the reconstruction error of the observation.
3) **Direct policy learning (DPL):** Same architecture as in the full network, but without using SRL, i.e., not training the transition model. The encoding, recurrent layers and policy are trained only using the cost of the policy.

The ablation study is done on a modified version of the Car Racing environment. Normally, this environment provides an upper view of a car in a racing track. In this case, we occluded the bottom half of this observation, such that the agent is not able to precisely know its position in the track. This position can be estimated if past observations are taken into account. As a consequence, this is an appropriate setting for making a comparison of different neural network architectures. Table I shows the different performances obtained by the learning algorithms when modifying the structure of the network. These results show a normalized averaged return over 10 repetitions for each experiment, in which 5 evaluations were carried for each one of these repetitions.

TABLE I: Performance (return) comparison of different learning methods in the Car Racing problem. Returns were normalized with respect to the best performance (DAgger Full).

	FULL	M-less SRL	DPL
D-COACH	0.97	0.76	0.68
DAgger	1.00	0.87	0.96
HG-DAgger	0.89	0.69	0.90

As expected, DAgger with the Full architecture obtained the best performance, and given that it receives new samples every time step, it was robust against the changes in the architecture, even when it did not have memory. On the other hand, D-COACH was very sensitive to the changes in the architecture, especially with the DPL architecture. This shows how the Full model is able to enhance the performance of the agents in problems where temporal information is required. It even makes Deep COACH perform almost as well as DAgger, despite that the former does not require constant and perfect teacher feedback. Finally, HG-DAgger was more robust than D-COACH in the DPL case, but its performance with the full model was not as good.

B. Simulated tasks with simulated teachers

In the second set of experiments, a comparison between the algorithms DAgger, HG-DAgger, and Deep COACH was carried out using the proposed Full network architecture. To keep the experiments free from human factor effects, the teaching process was, once again, performed with simulated teachers. The methods were tested in the simulated problems Mountain Car (in the supplementary material), and Swing-Up Pendulum. A mean of the return obtained over 20 repetitions

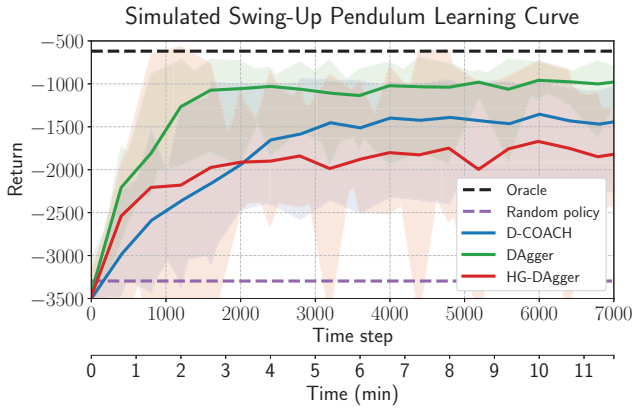


Fig. 4: D-COACH and (HG-)DAgger comparison in the Swing-Up Pendulum problem using a simulated teacher.

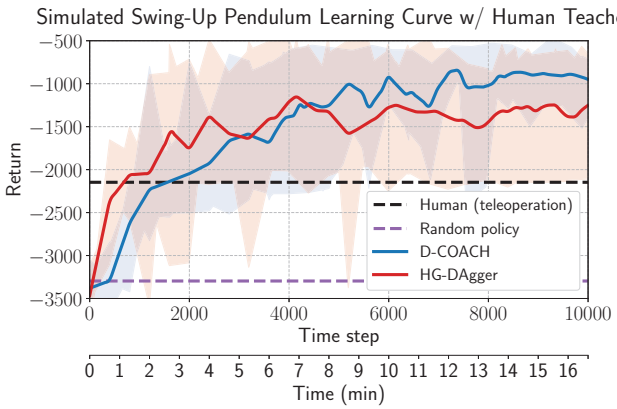


Fig. 5: Simulated Swing-Up Pendulum learning curve with human teachers

is presented for these experiments, along with the maximum and minimum values of these distributions.

Swing-Up Pendulum

In the case of the Swing-Up Pendulum, the results are very different for both DAgger agents (see Fig. 4). Both have a higher rate of improvement than Deep COACH during the first minutes, when the policy is learning the swinging behavior. Since the swinging part requires large actions, the improvement with Deep COACH is slower. However, once the policy is able to swing the pendulum up, the second part of the task is to keep the balance in the upright position, which requires fine actions. It is at this point when learning becomes easier for the Deep COACH agent, which obtains a constant and faster improvement than the HG-DAgger agent, even reaching a higher performance. In Fig. 4, the expected performance upper bound is showed with a black dashed line, which is the return obtained by the simulated teacher. The purple dashed line shows the performance of a random policy, which is the expected lower bound.

C. Simulated tasks with human teachers

The previous experiments give insights into how the policy architectures and/or the learning methods perform when imitating an oracle. Most IL methods are intended for learning from any source of expert demonstrations. It does not have to be a human necessarily; it can be any type of agent. However, the scope of this work is on learning from non-expert human teachers, who are complex to model and simulate. Therefore, conclusions have to be based on results that also include validation with real users.

Experiments with the Mountain-Car (in the supplementary material), and the Swing-Up Pendulum were run with 8 human teachers. In this case, the classical DAgger approach is not used, since, as discussed in Section II-C, it is not specifically designed for human users. Instead, HG-DAgger is validated.

Swing-Up Pendulum

This task is relatively simple from a control theory point of view. Nevertheless, it is quite challenging for humans to tele-operate the pendulum, due to its fast dynamics. Indeed, the participants were not able to successfully tele-operate the agent; therefore, unlike the Mountain-Car task, we could consider the participants as non-experts on the task.

Fig. 5 shows the results of this experiment, which are similar to the ones presented in Fig. 4. At the beginning, Deep COACH has a slower improvement when learning to swing up; however, it learns faster than HG-DAgger when the policy needs to learn the accurate task of balancing the pendulum. For the users, it is more intuitive and easier to improve the balancing with the relative corrections of Deep COACH than with the perfect corrective demonstrations of HG-DAgger, as they do not need to know the right action, rather just the direction of the correction. Unlike the performance of the simulated teacher depicted in Fig. 4, this plot shows the performance of the best human teacher tele-operating the pendulum with the same interface used for the teaching process. It can be seen that using both agents allowed to obtain policies that outperform the non-expert human teachers.

All the policies trained with Deep COACH were able to balance the pendulum, whereas with HG-DAgger the success rate was the half. Additionally, after the experiment, the participants were queried about what learning strategy they preferred. Seven out of eight expressed preference for Deep COACH.

D. Validation on physical systems with human teachers

The previous experiments performed comparison studies of the NN architectures and the learning methods under controlled conditions in simulated environments. In this section, Deep COACH is validated with human teachers and real systems in two different tasks: 1) a real Swing-Up Pendulum, and 2) a fruits classifier robot arm.

The real Swing-Up pendulum is a very complex system

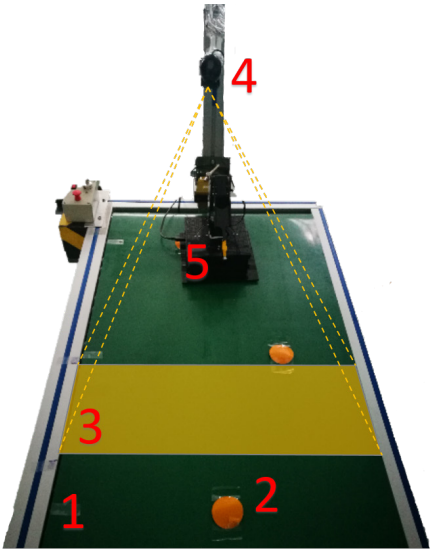


Fig. 6: Orange selector experimental set-up. 1) conveyor belt, 2) "orange" samples, 3) frame observed by the camera, 4) RGB camera, and 5) 3 DoF robot arm.

for a human to tele-operate. Its dynamics are faster than the simulated one of OpenAI Gym used in the previous experiments. The supplementary material provides more details of this environment along with the learning curve of the agents trained by the participants of this validation experiment. Those results, along with the video, show that non-expert teachers can manage to teach good policies.

Orange selector with a robot arm

This set-up consists of a conveyor belt transporting "pears" and "oranges", a 3 DoF robot arm located over the belt, and an RGB camera with a top view of the belt. The image of the camera does not capture the robot arm. The robot has to select oranges with the end effector, but avoid pears. The robot does not have any tool like a gripper or vacuum gripper to pick up the oranges. Therefore, in this context, we consider a successful selection of an orange when the end effector intersects the object. The performance of the learning policy is measured using two indices: 1) rate of oranges successfully selected, and 2) rate of pears successfully rejected.

The observations obtained by the camera are from a different region of the conveyor belt than where the robot is acting. Therefore observations cannot be used for compensating the robot position in the current time step, rather they are meaningful for future decisions. In other words, the current action must be based on past observations. Indeed, the delay between the observations and its influence on the actions is around 1.5 seconds. This delay is given by the difference between the time when the object gets out from the camera range and the time it reaches the robot's operating range. This is why this task requires to learn temporal features for the policy.

The problem is solved by splitting it into two sub-tasks which are trained separately:

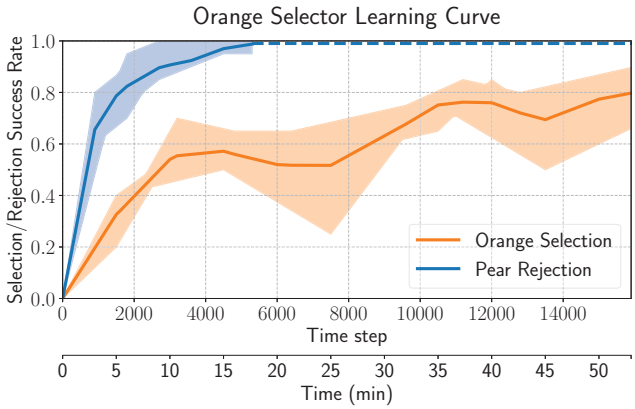


Fig. 7: Orange selection/pear rejection learning curve.

1) **Orange selection:** The robot must intercept the orange coordinate with the end effector, right when it passes below the robot.
2) **Pear rejection:** The robot must classify between oranges and pears, so when a pear is approaching under the robot, the end effector should be lifted far from the belt plane, otherwise it should get close.

These two sub-tasks can be trained sequentially. The orange selection is trained initially, with a procedure in which there are some oranges being transported by the belt with fixed positions, while some others are placed randomly. This in order to avoid over-fitting of the policy to specific sequences.

When the robot is able to track the oranges in its reach, the pear rejection learning starts. For that, pears are placed randomly throughout the sequences of oranges, and the human teacher advises corrections on the robot movement in order to make the end effector move away from the pears when they are in the operation region of the robot.

Fig. 7 depicts the average learning curves for this task after 5 runs of the teaching process. It is possible to see that the pear rejection sub-task is learned within 20 minutes with 100% success, while the orange selection is a harder sub-task that only reaches around 80% success after 50 minutes. Effectively, combining the two sub-tasks, the performance of the learned policies is given only by the success of the orange selection, since the pear rejection was perfectly attained in all the runs executed for this experiment.

V. Conclusion

This paper has introduced and validated a SRL strategy for learning policies interactively from human teachers in environments not fully temporally observable. Results show that when meaningful spatiotemporal features are extracted, it is possible to teach complex end-to-end policies to agents using just occasional, relative, and binary corrective signals. Even more, these policies can be learned from teachers who are not skilled to execute the task.

The evaluations with the Data Aggregation approaches and Deep COACH depict the potential of this kind of architecture

to work on different IIL methods. Especially in methods based on occasional feedback, which are intended to reduce the human workload.

The comparative results between HG-DAgger and Deep COACH with non-expert teachers showed that with the former, the policy will remain biased with mistaken samples even if the teacher makes sure of not providing more wrong corrections (given that it works with the assumption of expert demonstrations); hence, it makes harder to refine the policy. On the other hand, Deep COACH proved to be more robust to mistaken corrections given by humans, since all the non-expert users were able to teach tasks that they were not able to demonstrate.

The previous mentioned shortcoming of DAgger algorithms open possibilities for future works, which are intended to study how to deal with databases with mistaken examples. Another field of study is the one of data-efficient movement generation in animation [19], which combined with our method, would make it possible to learn (non-)periodic movements using spatiotemporal features and IIL. Challenges such as the generation of smooth, precise, and stylistic movements (i.e. dealing with high-frequency details [20]) could be also addressed.

ACKNOWLEDGMENT

This research has been funded by the Netherlands Organization for Scientific Research (NWO) project FlexCRAFT, grant number P17-01, by the ERC Stg TERI, project reference #804907, by FONDECYT project [1201170], and by CONICYT project [AFB180004].

REFERENCES

[1] S. Chernova and M. Veloso, "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, pp. 1–25, 2009.

[2] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. Kochenderfer, "Hg-dagger: Interactive imitation learning with human experts," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8077–8083.

[3] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, 2017, pp. 4299–4307.

[4] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The TAMER framework," in *Fifth International Conference on Knowledge Capture*. ACM, 2009, pp. 9–16.

[5] C. Celemin and J. Ruiz-del Solar, "An interactive framework for learning continuous actions policies based on corrective feedback," *Journal of Intelligent & Robotic Systems*, pp. 1–21, 2018.

[6] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor, and M. L. Littman, "Interactive learning from policy-dependent human feedback," in *34th International Conference on Machine Learning - Volume 70*. JMLR. org, 2017, pp. 2285–2294.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] W. Böhmer, J. T. Springenberg, J. Boedecker, M. Riedmiller, and K. Obermayer, "Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations," *KI-Künstliche Intelligenz*, vol. 29, no. 4, pp. 353–362, 2015.

[10] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Networks*, vol. 108, pp. 379–392, 2018.

[11] R. Pérez-Dattari, C. Celemin, J. Ruiz-del Solar, and J. Kober, "Continuous control for high-dimensional state spaces: An interactive learning approach," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7611–7617.

[12] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 627–635.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[14] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *2015 AAAI Fall Symposium Series*, 2015.

[15] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[16] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, 2018, pp. 2450–2462.

[17] A. Zhang, H. Satija, and J. Pineau, "Decoupling dynamics and reward for transfer learning," *arXiv preprint arXiv:1804.10689*, 2018.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016.

[19] I. Mason, S. Starke, H. Zhang, H. Bilen, and T. Komura, "Few-shot learning of homogeneous human locomotion styles," in *Computer Graphics Forum*, vol. 37, no. 7, 2018, pp. 143–153.

[20] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–11, 2018.

Supplementary Material: Interactive Learning of Temporal Features for Control

Rodrigo Pérez-Dattari[1], Carlos Celemin[1], Giovanni Franzese[1], Javier Ruiz-del-Solar[2] and Jens Kober[1]

The scope of this Supplementary Material is to provide some other details of the experiments, that are not present in the main document [1]. Next section is making a brief recap of the method. The additional information is divided in three main parts:

- **Ablation study:** Some more details are added about the environment of the Racing Car.
- **Simulated tasks with simulated/human teachers:** where some specific details are added on the experimental procedure along with some results.
- **Validation on real robot tasks with human teachers:** where experiments and results on a real inverted pendulum are described. Some more details about the architecture and the set-up of the orange selector and pear rejection are also introduced.

I. Recap of the Method

The role of the human as a teacher is to observe the decision-making strategy of the agent, and give corrections over the agent's actions when necessary. The policy is updated with these corrections. To do this in an efficient way, a model of the world is learned simultaneously with the policy (see Fig. 1). Learning this model forces to pre-processes the high-dimensional observations to create a state representation of the environment. This representation of the state contains spatiotemporal information that the agent can use to make decisions.

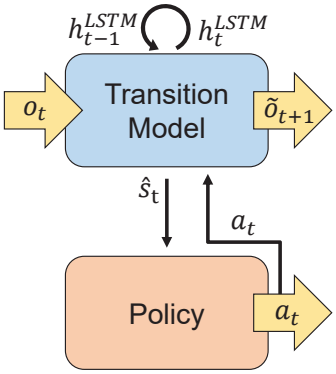


Fig. 1: Transition model and policy general structure.

[1]R. Pérez-Dattari, C. Celemin, G. Franzese and J. Kober are with the Department of Cognitive Robotics, Delft University of Technology, Delft, The Netherlands e-mail: (r.j.perezdattari, c.e.celeminpaez, g.franzese, j.kober)@tudelft.nl.

[2]J. Ruiz-Del-Solar is with the Electrical Engineering Department and the AMTC, University of Chile, Santiago, Chile e-mail: jruizd@ing.uchile.cl.

II. Additional Information of Experiments and Results

The metrics used for the comparisons are the final policy's performance achieved and the speed of convergence, which is very relevant when dealing with real systems and human teachers. In every learning curve, a mean of the return obtained over 20 repetitions for every experiment is presented, along with the maximum and minimum values of these distributions. The performance was measured in the simulated environments with the objective functions included in OpenAI Gym. For the real pendulum, a similar function to the one implemented in OpenAI Gym was used, while for the task with the robot arm, the success rate was used.

For the experiments with simulated teachers, high-performance policies trained by expert teachers were used as oracles for providing the corrections. These policies have performances in the level of the state of the art (performance plotted in the figures). In Deep COACH, the corrections h are the sign of the relative change advised by the teacher. Therefore, to compute these binary corrections, the simulated teacher computes $h = \text{sign}(a_{teacher} - a_{agent})$ for each of the dimensions composing the action vector, wherein $a_{teacher}$ is the action computed by the policy of the teacher and a_{agent} is the action computed by the learning policy, as it has been implemented in [2], [3]. The frequency of the corrections given by the simulated human teacher are controlled with a probability that is reducing with the time.

For the experiments in which actual human teachers participated correcting policies, the corrections were provided through a keyboard. For each action dimension, two keys were designated, so the user could advise increase or decrease, in each of the axis.

A. Neural Network Architecture Details

Fig. 3 of the main document [1] shows the structure of the proposed NN architecture. The hyperparameters of each of these layers are shown in Tables 1 and 2.

B. Additional details about the environment in the Ablation study

All the results obtained in this study are shown in the main document [1]. The comparisons were carried out only with the Car Racing problem of OpenAI Gym. In this environment, the agent has three action dimensions which are: steering, acceleration, and brake.

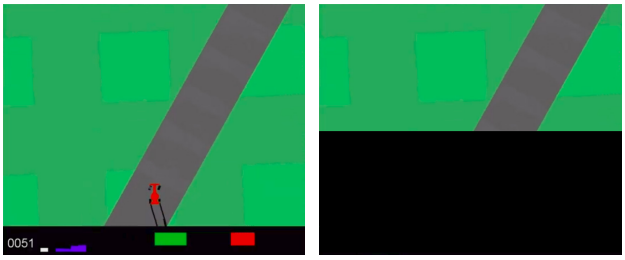
Layer	Activation	Filters	Filter size	Stride
C1	ReLU	16	3×3	2
C2	ReLU	8	3×3	2
C3	sigmoid	4	3×3	2
DC1	ReLU	8	3×3	2
DC2	ReLU	16	3×3	2
DC3	sigmoid	1	3×3	2

TABLE I: Hyperparameters of convolutional and deconvolutional layers.

Layer	Activation	N° neurons
FC1	tanh	256
FC2	tanh	256
FC3	tanh	1000
FC4	tanh	256
FC5	ReLU	1000
FC6	ReLU	1000
FC7	tanh	Task action dimension
R1	LSTM activations	h^{LSTM} = 150

TABLE II: Hyperparameters of fully connected and recurrent layers.

As mentioned in the main document [1], the bottom half of the frame is occluded in order to force the agent to take decisions based in past observations. An example of the occluded frame is shown in Fig. 2, the current position of the car in the road is not observed by the learning policy; however, the entire frame is observed by the policy used as simulated teacher. Therefore, the corrections are based on appropriate actions with respect to the real state of the environment.



(a) Original frame (b) Occluded frame

Fig. 2: Original frame of the Car Racing environment, on the left. Occluded frame used as observation in the network, on the right.

C. Simulated tasks with simulated teachers

The Mountain Car and the Swing-Up Pendulum environments originally provide, at each time step, their low-dimensional explicit state. These are the position and velocity of the car in the x axis, and the angle and angular velocity of the pendulum, respectively. In order to obtain a high-dimensional observation (raw image) we have modified the source code, such that the environment returns an array with the RGB frame that is rendered.

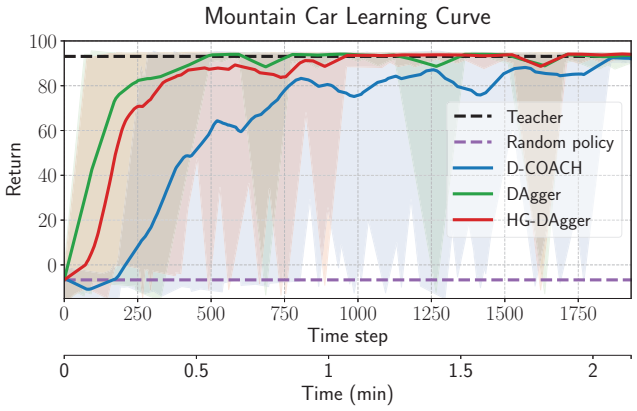


Fig. 3: D-COACH and DAgger comparison in the Mountain Car problem using a simulated teacher.

Mountain Car

The action of the agent is the force applied in order to move the Car. It is known that the optimal solution for this task is a bang-bang controller (using the extreme actions -1 or 1). Therefore, the correct actions given by the oracle are very different from the initial policy in the whole state space. This makes it easier to perform abrupt changes when updating the policy for DAgger-like agents than for Deep COACH, because the latter performs smaller steps based on incremental and relative corrections (corrections are in the direction of the oracle's action, but not directly the actual action).

As it can be seen in Fig. 3, as expected, the learning convergence is faster for DAgger, followed by HG-DAgger, whereas Deep COACH is the slowest. DAgger converges faster than its modified version, HG-DAgger, because the former trains the policy with corrections provided by the oracle during every time step, which is most of the times not feasible when teaching with human users. All agents reach the oracle's performance at the end of the learning process.

D. Simulated tasks with human teachers

The validation was executed with 8 participants between 20 and 30 years. In the experiments, the users observed the desired performance of the agent, they received instructions on how to interact with each kind of agent, and they had the chance to practice with the learning agent before recording results, in order to get used to the role of teacher. The users interact with the learning agent using a keyboard. Users correct the policies until they consider they cannot improve them anymore, or until a maximum duration of the training session of 500 seconds (8.33 minutes). The episodes of the tasks had a duration of 20 seconds, which for Mountain Car means the maximum duration, if the goal is not reached, whereas for the pendulum this is constant. The duration of the time steps is 0.05 s.

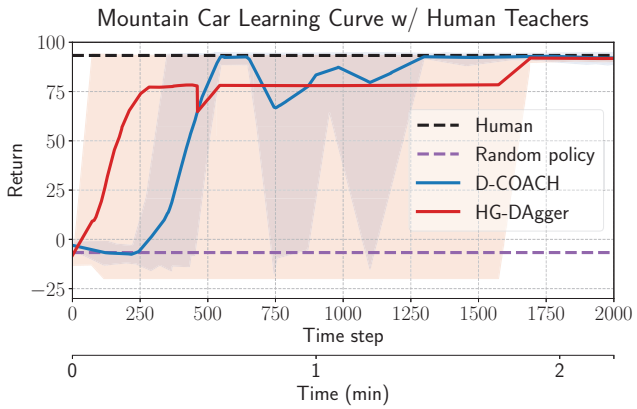


Fig. 4: Mountain Car learning curve with human teachers

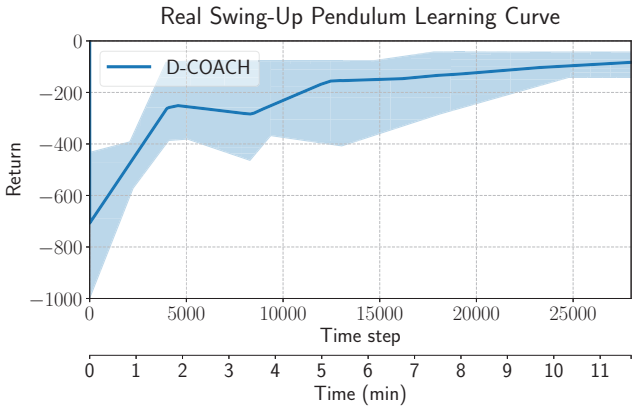


Fig. 5: Real Swing-Up Pendulum learning curve.

Mountain Car

This task is very simple, since users understand properly how to tele-operate the agent, therefore, we could state that for this task, the teachers are always experts.

Results in Fig. 4 are similar to the ones observed in Fig. 3 in Section II-C, wherein at the beginning HG-DAgger improves way faster. However, in this case HG-DAgger gets stuck and is outperformed by Deep COACH after 20 seconds. This happens as, despite the fact that the teachers are considered experts, they could sometimes provide mistaken or ambiguous corrections. These inconsistencies remain permanently in the database, so it is hard for the user to fix the generated error.

E. Validation on physical systems with human teachers

1) Real Swing-Up Pendulum

This system is similar to the one used previously from OpenAI Gym, although in this set-up the dynamics are even much faster, and the actions are voltages instead of torques. In this environment a camera is set in front of the pendulum to obtain observations similar to the simulated environment, as it is shown in Fig. 6, wherein the camera is in the bottom, aligned with the pendulum. The observation was down-sampled to 32 $\times$ 32 pixels images, while for all the other tasks the down-sampling was to 64 $\times$ 64 pixels. An example of the actual observation of the camera is in Fig. 7, where it is possible to see the input of the NN (down sampled image), along with the prediction of the observation in $t + 1$, with a model that has been trained with the proposed architecture. In this example, it is interesting to observe that the pendulum was rotating clockwise, therefore the prediction shows the weight of the pendulum in a lower position.

Fig. 5 shows the learning curve of 10 runs. In average, the users manage to teach a policy able to balance after 29 episodes, and they keep on improving the policy during the subsequent episodes. Preliminary tests showed that the sampling time needs to be reduced to 0.025 s to be able to control the system.

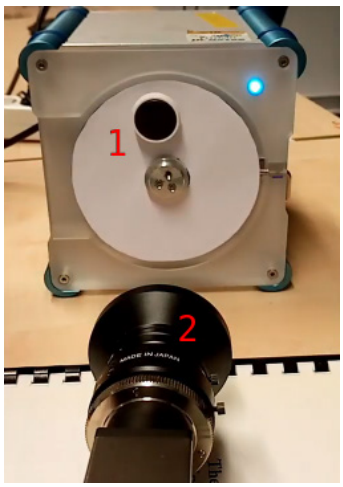
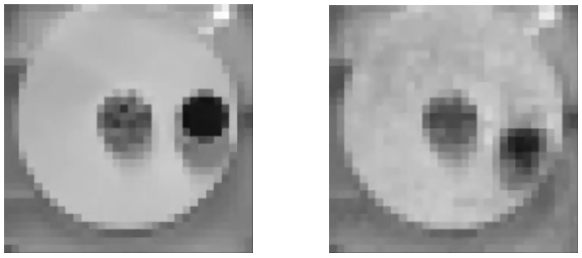


Fig. 6: Real Swing-Up Pendulum Setup. 1) Pendulum; 2) RGB camera.

2) Orange selector with a robot arm

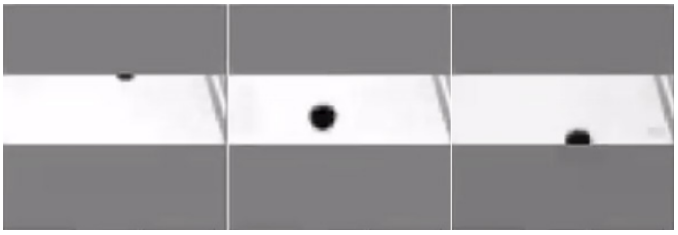
As mentioned in the main document [1], this problem is composed by two sub-tasks, that are sequentially trained. In the experiments, at the beginning when learning the **Orange selection**, the teachers corrected the policy only in order to make the robot to move the end-effector over the horizontal plane. Therefore, the transition model along with a controller in charge of intercepting the oranges with the end-effector are learned in the first stage. Then, in order to learn the second sub-task (**Pear rejection**) composing the problem, the already learned transition model is reused in the training process of a second controller that either moves the end-effector close to the belt or far when a pear is detected. These two controllers work in parallel since their actions are over different joints.

In the associated video it is shown examples of the actual images that go to the input of the network, along with the prediction of the next observation. However due to the velocity of the video it is not easy to see that the predicted oranges are slightly shifted. In Fig. 8 are shown examples of three different oranges crossing the field of view of the camera.

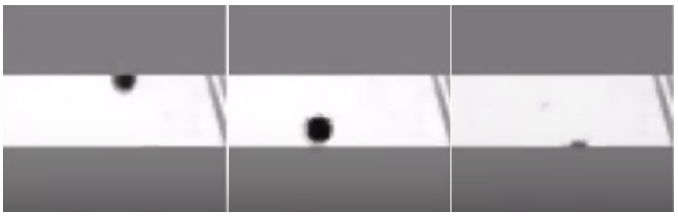


(a) Network input (b) Observation prediction

Fig. 7: Downsampled observation of the camera, which is the input of the Network, on the left. Next observation prediction, on the right.



(a) Network input



(b) Observation prediction

Fig. 8: Examples of images in the input of the Network, on the top. Observation predicted for the next time step by the learned transition model, on the bottom.

The examples are input-output pairs of the transition model. The oranges moved by the belt are observed by the camera while crossing from the top to the bottom of the field of view. It could be seen that the predicted oranges are in a lower position with respect to the position observed in the input, i.e. the model learns to predict the movement of the belt.

REFERENCES

[1] R. Pérez-Dattari, C. Celemin, G. Franzese, J. Ruiz-del Solar, and J. Kober, "Interactive learning of temporal features for control," *Robotics and Automation Magazine (RAM)*, 2020.

[2] R. Pérez-Dattari, C. Celemin, J. Ruiz-del Solar, and J. Kober, "Continuous control for high-dimensional state spaces: An interactive learning approach," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7611–7617.

[3] C. Celemin and J. Ruiz-del Solar, "An interactive framework for learning continuous actions policies based on corrective feedback," *Journal of Intelligent & Robotic Systems*, pp. 1–21, 2018.