Learning to touch objects through stage-wise deep reinforcement learning

François de La Bourdonnaye, Céline Teulière, Jochen Triesch, Thierry Chateau

Learning to touch objects through stage-wise deep reinforcement learning

François de La Bourdonnaye[1], Céline Teulière[1], Jochen Triesch[2], and Thierry Chateau[1]

***Abstract*— Learning complex behaviors through reinforcement learning is particularly challenging when reward is only available upon successful completion of the full behavior. In manipulation robotics, so-called shaping rewards are often used to overcome this problem. However, these usually require human engineering or (partial) world models describing, e.g., the kinematics of the robot or high-level modules for perception. Here we propose an alternative method to learn an object palm-touching task through a weakly-supervised and stage-wise learning of simpler tasks. First, the robot learns to fixate the object with its cameras. Second, the robot learns eye-hand coordination by learning to fixate its end effector. Third, using the previously acquired skills an informative shaping reward can be computed which facilitates efficient learning of the object palm-touching task. We demonstrate in simulation that learning the full task with this shaping reward is comparable to learning with an informative supervised reward.**

I. Introduction

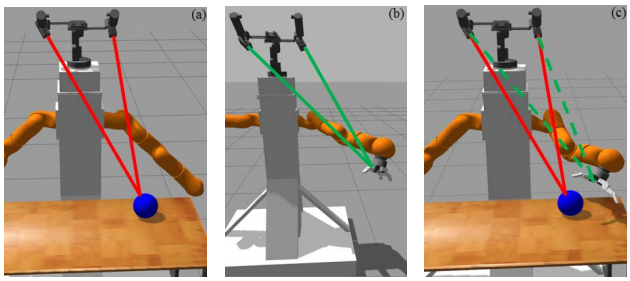

Fig. 1. Palm-touching learning process: (a) Object fixation, (b) End-effector fixation and hand-eye coordination, (c) Palm-touching

In recent years, deep reinforcement learning (RL) algorithms have allowed the learning of complex behaviors directly mapping pixels to actions without using hand-crafted features [1], [2]. However, learning can be prohibitively slow when reward is only available when the full task has been completed successfully. This problem of sparse rewards is frequently addressed by specifying more informative reward functions that rely on additional forms of supervision. Indeed, informative shaping rewards have long been applied in manipulation robotics [3], [4], because they help to discriminate values of close states. For instance, [3], [4], [5], [6] and [7] use a distance measure between the current pose and a target pose to design an informative reward in manipulation tasks such as block stacking, reaching and door pushing or pulling. These rewards require knowledge of robot kinematics and target pose. Similarly, for tasks such as placing wooden rings or screwing bottle caps onto bottles [2] and [8] use an informative reward based on the distance between actual positions of end-effector or target object and their target positions. Tracking these points requires knowledge of the kinematics or non-trivial vision modules. For the same kind of task, [1] designs a reward based on a distance between visual features produced by an autoencoder and target features. This requires to place the robot in a target position and extract the target visual features each time the target location changes.

In contrast to these previous works, the aim of this paper is a robot that learns an object palm-touching task with only weak human intervention or supervision i.e. without kinematic models, hand-crafted features, or calibration parameters. More precisely, the task consists of reaching with the end-effector palm an object put on a table. This behavior can be seen as a natural predecessor to grasping. To learn this task with little supervision, we propose a stage-wise learning strategy inspired by infant development [9], [10], [11]. The central idea is that an informative reward is constructed from knowledge that has been acquired during the learning of simpler predecessor tasks. Specifically, our robot first learns to fixate an object or its own end-effector using deep RL. Based on the knowledge acquired during these tasks an informative reward function is constructed that allows the robot to efficiently learn the object palm-touching task. Our results demonstrate that learning this task based on the learned informative reward signal works about as well as learning with a fully supervised reward, while naively learning from only a sparse reward signal fails completely in the same setting.

II. Related work

Our approach resembles developmental robotics methods [9], [12], [13], [14] which learn to reach using a hand-eye coordination function and object fixation. However, they are usually paired with supervision for the object and/or end-effector fixations [12], [13], [14] or computation of action primitives [9].

Our work is also related to methods which use sparse-only rewards in the sense that our sparse term fully defines

*This work is sponsored by the French government research program "Investissements d'avenir" through the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01), by the European Union through the program Regional competitiveness and employment (ERDF Auvergne region), and by the Auvergne region. J.Triesch acknowledges support from the Quandt foundation and the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 713010 (GOAL-Robots Goal-based Open-ended Autonomous Learning Robots).

[1]F. de La Bourdonnaye, C.Teulière and T. Chateau are with the university of Clermont Auvergne, the Pascal Institute CNRS, UMR6602, Aubière, France

[2]J.Triesch is with the Frankfurt Institute for Advanced Studies, Frankfurt am Main, Germany

the task. Though hardly using any supervision, sparse-only rewards generally require some strategies to increase the probability of getting a sparse reward. This can be done by increasing the number of agents learning the task [15] or by simplifying the action space using hand-crafted pre-processing [16]. Besides, if the agent has access to one of its goal states, a mechanism of learning from easy missions [17] can be implemented to learn very precise robotic manipulation tasks [18]. We propose to increase the touching probabilities by building a weakly-supervised reward shaping term from learned skills without using action space simplification, several agents or the indication of a goal state.

III. Methods

A. Background

An RL algorithm is usually based on Markov decision processes $< S, A, R, T >$ where S is the set of states, A the set of actions, T the transition model ($T : S \times A \rightarrow S$) and R the reward function ($R : S \times A \times S \rightarrow \mathbb{R}$).

The goal of the agent is to maximize the sum of discounted future rewards: $J = \sum_{k=0}^{\infty} r_k \gamma^k$, where $\gamma \in [0, 1]$ is a discount factor and r_k the reward value at step k.

To optimize the criterion, we train a deterministic policy $\pi : S \rightarrow A$. In the paper, we also estimate the state-action value function:

$$Q_\pi(s, a) = E_\pi \left[\sum_{k=0}^{\infty} r_k \gamma^k \middle| s, a \right], (s, a) \in S \times A. \quad (1)$$

Both the policy and the Q function are approximated by a neural network. Specifically, we use the DDPG algorithm [19] which combines the deterministic policy gradient algorithm [20] and the DQN [21]. DDPG is an "actor-critic" algorithm updating the critic Q_ϕ with parameters ϕ and the deterministic policy π_θ with parameters θ as follows. At each time-step, we choose N_b transitions from a large memory buffer using a uniform distribution:

$< s_i, a_i, r_i, s_i^{'} >_{i \in \{1,...,N_\text{b}\}} \in S \times A \times \mathbb{R} \times S$.

The targets of the Q_ϕ neural network are computed using a TD(0) update (with a learning rate equal to 1):

$$\forall i \in \{1, ..., N_\text{b}\}, y_i = r_i + \gamma Q_{\phi'} \left(s_i^{'}, \pi_{\theta'}(s_i^{'}) \right). \quad (2)$$

$\phi^{'}$ and $\theta^{'}$ are the parameters of the target networks updated using a rate parameter τ (t denotes a time-step):

$$\phi_{t+1}^{'} = \tau \phi_t + (1 - \tau)\phi_t^{'}, \theta_{t+1}^{'} = \tau \theta_t + (1 - \tau)\theta_t^{'}, \quad (3)$$

The Q_ϕ network updates its weights by minimizing the squared error $\frac{1}{2N_\text{b}} \sum_{i=1}^{N_\text{b}} \left(y_i - Q_\phi(s_i, a_i) \right)^2$.

Using the Q_ϕ network and the fact that the policy is deterministic, the following policy gradient is derived:

$$\frac{\partial Q_\phi}{\partial \theta} \simeq \frac{1}{N_b} \sum_{i=1}^{N_b} \frac{\partial Q_\phi \left(s_i, \pi_\theta(s_i) \right)}{\partial a} \frac{\partial \pi_\theta(s_i)}{\partial \theta}. \quad (4)$$

This update makes the policy select the actions that maximize the Q function at the batch states. In addition to this algorithm, we use the inverting gradient procedure of [22] to bound the actions. This method downscales the gradient when the action computed by the policy approaches its limit. When it exceeds its limit, the gradient is inverted. This mechanism prevents the actions from becoming too large.

B. Overview

We describe here the stage-wise learning process (see Figures 1 and 2 for a schematic view). The task consists of touching the object with the palm. In the following, we use the notations:

- $I = (I^\text{left}, I^\text{right})$ represents the images from the left and right cameras.
- $q = (q^\text{camera}, q^\text{robot})$ represents the 3 camera joint angles (one common tilt angle and two independent pan angles) and 7 joint arm joint angles.

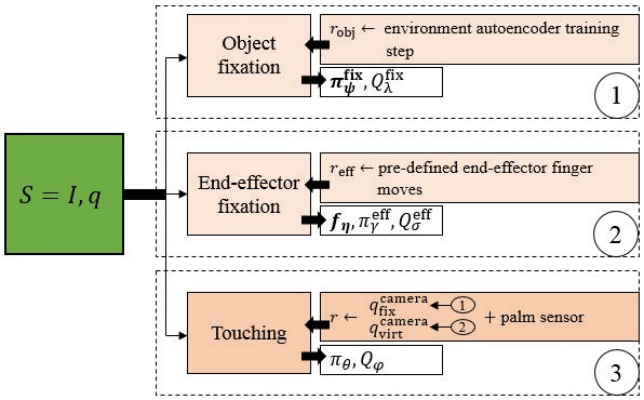

Fig. 2. Overall scheme of the touching task learning procedure

Our stage-wise learning framework involves three successive tasks:

First, the robot learns from raw pixels to fixate the object with a two-camera system. For this, we use [23] to learn to fixate an object with weak supervision. At the end of the fixation, the camera system coordinates $q_\text{fix}^\text{camera}$ implicitly encode the object position in 3D space.

Second, the robot learns a hand-eye coordination function f_η which maps robot joint coordinates to virtual camera coordinates:

$$q_\text{virt}^\text{camera} = f_\eta(q^\text{robot}). \quad (5)$$

These virtual camera coordinates correspond to the camera coordinates which would make the camera system look at the end-effector. Finally, a reward signal using $q_\text{fix}^\text{camera}$ and $q_\text{virt}^\text{camera}$ to make the end-effector close to the object is computed and combined with a sparse reward, indicating if there is palm-touching or not. In the following, we describe each of the three steps.

C. Learning binocular object fixations

Fixating an object means bringing the object at the center of I_left and I_right by moving the cameras. To learn it, we build on our prior work [23].

The task is learned with the DDPG algorithm [19] using I and q_{camera} as states, and Δq_{camera} as actions. The reward function is the sum of left and right camera components: $r_{\text{obj}} = r_{\text{obj}}^{\text{left}} + r_{\text{obj}}^{\text{right}}$. For each camera c = left or right, the reward function is an affine decreasing function of the distance between the image center P_{i} and the estimated object position P_{o}^{c} (computed below):

$$r_{\text{obj}}^{\text{c}} = 2\frac{\frac{1}{2}d_{\max} - ||P_{\text{i}} - P_{\text{o}}^{\text{c}}||_2}{d_{\max}} \in [-1, 1], \tag{6}$$

with $d_{\max}$ being the maximal distance between the image center and the object pixellic position.

An episodic set-up is used. For each episode, a random object is put at a random location above the table. The episode ends when a given number of transitions has been reached.

The object detection mechanism involves a convolutional autoencoder training step in which images of the environment without object are encoded. Then, when an object appears in the environment, it is detected as an anomaly and localized in the images I^{left} and I^{right}. More precisely, we apply a kernel density estimator on the autoencoder reconstruction error images respectively $|I^{\text{left}} - \hat{I}^{\text{left}}|$ and $|I^{\text{right}} - \hat{I}^{\text{right}}|$. After that, the estimated object pixellic positions, respectively $P_{\text{o}}^{\text{left}}$and $P_{\text{o}}^{\text{right}}$ are at the maximal probabilities as described in [23]. This object detection principle only requires an autoencoder pre-training step without object and the assumption that there is an object in the scene subsequently.

D. Learning a hand-eye coordination function f_η

To build f_η, we need to have a database D of input-output pairs $(q^{\text{robot}}, q^{\text{camera}})$ where q^{camera} makes the camera look at the end-effector. To produce such samples, we learn how to fixate the end-effector the same way [23] learns to fixate an object. We use the DDPG algorithm and a reward requiring weak supervision. The Markov Decision Process is the same as for the object fixation with the exception of the reward function. The latter involves the end-effector detection P_{e}^{c} (computed below) instead of P_{o}^{c}:

$$r_{\text{eff}}^{\text{c}} = 2\frac{\frac{1}{2}d_{\max} - ||P_{\text{i}} - P_{\text{e}}^{\text{c}}||_2}{d_{\max}} \in [-1, 1]. \tag{7}$$

The set-up is also episodic. For each episode, random robot joint coordinates are generated using uniform distribution with fixed limits. They are empirically set to provide a large variety of reachable arm configurations.

During learning, training pairs $(q^{\text{robot}}, q^{\text{camera}})$ are added to D when the reward is above a fixed threshold. When the number of samples in D is higher than the batch size N_{bf}, we train f_η on random batches of D as soon as a new sample is added to D.

We describe here how we detect the end-effector in the image. Unlike [23] which uses an autoencoder to detect the object as an anomaly, the end-effector image position is computed using the difference in the image before and after pre-defined end-effector finger moves. This end-effector detection method inspired by [24] only requires to specify finger moves.

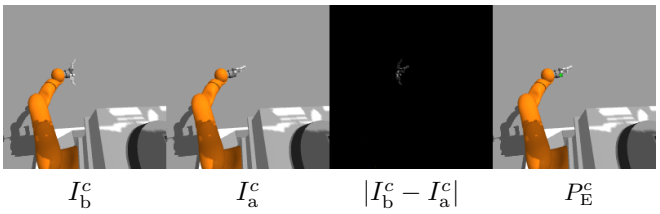

Fig. 3. End-effector computation scheme

Figure 3 presents the different steps of the end-effector detection:

- The image before the end-effector I_{b}^{c} is saved.
- The end-effector fingers make pre-defined moves and the image I_{a}^{c} is saved
- The difference of images is calculated and the end-effector position P_{E}^{c} is computed using a kernel density estimator the same way P_{O}^{c} is calculated from the autoencoder reconstruction error image.

E. Learning to touch

The goal of the robot is to touch with the end-effector palm the object above the table from a large set of initial arm positions.

It is defined by a sparse reward term r_{sparse} which indicates if there is palm-touching or not:

$$r_{\text{sparse}} = \begin{cases} 1, & \text{if palm-touching,} \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

As the object is kept at the same position, the state space S is composed of the arm and camera coordinates, q. Indeed, for this task, we do not need to add images in the state space since the right arm movements can be decided from the robot joint angles only. The actions consist of the variations of the robot joint angles: $a = \Delta q^{\text{robot}}$ which are seven real-valued scalars.

The object binocular fixation task and the hand-eye coordination function are used to compute the touching reward function.

After the execution of an object fixation step (using the object fixation policy π_ψ), we get the fixation camera angles $q_{\text{fix}}^{\text{camera}}$ which localize the object. After that, using equation 5 at each time-step, the hand-eye coordination function f_η gives us $q_{\text{virt}}^{\text{camera}}$ which localizes the end-effector. Then, a reward shaping term r_{shaping} can be computed:

$$r_{\text{shaping}} = c_{\text{cam}}||q_{\text{fix}}^{\text{camera}} - q_{\text{virt}}^{\text{camera}}||_2, \tag{9}$$

with $c_{\text{cam}} < 0$. r_{shaping} represents an informative term which depends on the distance between the virtual camera coordinates and the camera coordinates which make the camera system fixate the object. It can be seen as an open-loop term which constrains the 3 translational degrees of freedom of the end-effector. It takes negative values to ensure quick trajectories are learned. Furthermore, the slope c_{cam} is

chosen to ensure shaping rewards are small compared with the non-zero sparse reward. Finally, the reward function is the sum of the two terms:

$$r_{\text{cam}} = r_{\text{shaping}} + r_{\text{sparse}}. \tag{10}$$

IV. Experiments

In this section, we describe the experimental protocol. The experimental objective is two-fold. First, we want to evaluate performances of our weakly-supervised reward in a palm-touching task. Second, we wish to compare our reward with an informative reward requiring external information.

A. Experiment environment

The experiments are carried out using the Gazebo simulator jointly with the ROS [25] middleware. A two-camera system is mounted on a bi-arm robotic platform (see Figure 1). Only one arm is used for the experiments and a Barrett hand is attached to this arm. A table from the Gazebo object database is placed below the cameras and in front of the bi-arm platform. Note that this table is not present when we learn the hand-eye coordination function. Our method does not depend on a specific object model because the robot learned to look at any object in [23]. In our experiments, we use a blue ball.

B. Different reward functions

In this paragraph, we present the reward functions which will be used in our experiments:

- $r_{\text{cam}} = r_{\text{shaping}} + r_{\text{sparse}}$.
  This is the proposed reward function (described in Section III-E).

- $r_{\text{sparsePen}} = \begin{cases} 1, & \text{if success,} \\ -0.0125, & \text{otherwise.} \end{cases}$
  This rewards the robot only when the palm touches the object. Besides, it penalizes each unsuccessful movement to encourage the robot to quickly touch the object.

- $r_{\text{cart}} = c_{\text{cart}}||X - X_{\text{target}}||_2 + r_{\text{sparse}}$, with $c_{\text{cart}} < 0$. To build this reward, we give a 3-dimensional end-effector target pose X_{target} for the shaping part and we add a sparse reward. This reward is the closest to the proposed r_{cam} but its shaping term requires forward kinematics and 3D object pose information. Finally, the slope c_{cart} is chosen to make the shaping term take about the same values as r_{shaping}.

We choose not to compare our reward function with one which would constrain both the end-effector position and orientation. Indeed, for such a reward function, a success would be to touch with the palm from a specific orientation and position. In our case, a success can be to touch with the palm from any position. The tasks are then too different to be compared in terms of touching improvement.

C. Experimental protocol

We describe how we compare the reward functions for the touching task. The protocol contains a training and a test phase.

1) Training phase:

For training, we use the DDPG algorithm [19] and the previously defined reward functions. Learning happens on N_{tot} fixed-length episodes of size N_{eps}. Each episode has an initial arm position and an object position. The object position is kept fixed. For the initial arm position, we use two different settings. The first ($P1$) one uses a single initial arm position for learning as in [7]. This position is made well oriented for a palm-touching purpose. This means that the robot has to move its end-effector mainly in translation to touch the object. The second ($P2$) uses random robot joint angles generated from uniform distributions. The goal of this setting is to check if we can learn to touch the object with our reward signal including initial positions with a badly oriented end-effector. In such a position, the robot has to substantially modify its end-effector orientation to touch the object. Furthermore, it is known that learning from different initial states allows to learn policies that are more robust to perturbations [26].

We use constant zero-mean Gaussian noises ϵ_{trans} and ϵ_{rot} for the exploration. The variances of the noise are higher for joints which strongly influence the end-effector orientation (ϵ_{rot}). Indeed, more exploration is required for wrist angles because "touching" orientations are only rewarded by the sparse reward.

To accelerate learning, we handle the times when the robot reaches "unsuccessful" contact areas where its movement is blocked. Indeed, in the training phase, the robot often touches the object without a palm contact though it is close to being successful. To make exploration easier in these areas, we record contact-less positions through a short-sized circular buffer. In case of contact, we pick the oldest position in the buffer, compute the action to go to this position, and apply it. This allows to more correctly discriminate actions in the contact areas.

Through the training experiments, we wish to compare our reward requiring weak supervision with other ones. Consequently, for all the reward signals, in order to monitor the learning progress, we specifically plot the touching frequency $f_{\text{t}}^{\text{type}}$ over the episodes, type referring to a specific reward function. We average three experiments per setting with the exception of the sparse reward setting. The latter is done once.

2) Test phase:

The goal of test experiments is to evaluate final reaching performances. We test the learned policies without any exploration noise on N_{tot} random episodes. At the end, we compute the touching frequency $f_{\text{test}}^{\text{type}}$. Note that we do not apply the trick used in the training phase to deal with blocked situations. Instead, when the robot is blocked, it just computes its next action with the policy. Like in the training phase, the results are averaged over three experiments per setting.

D. Implementation details

For all the neural network algorithms, we use the caffe library [27]. A GPU (nvidia GeForce GTX Titan X) is used for the experiments.

We use the same neural network structures as in [23] for the end-effector and object fixation tasks. The hyperparameter values are also the same with the exception of the number of iterations: 200000. The hand-eye coordination function is a neural network with 2 fully connected hidden layers of 10 and 5 neurons and a batch size N_{bf} of 32 is used to learn it. For the episode initialization of the end-effector fixation task, the seven arm joint angle distribution amplitudes are 11, 46, 69, 92, 92, 86 and 0° if we consider the ascending order in the kinematic chain i.e. from the base link to the end-effector.

For the touching task, the Q network has 3 fully connected layers with 250, 200 and 1 neural units. The policy network involves 3 fully connected layers with 200, 150 and 7 neural units. The weights are updated using the Adam [28] solver. Table I and II provide values for the parameters used in the experiments. For the Q update, the discount factor γ is equal to 0.99. For the episode initialization of the $P2$ problem, the distribution limits are the same as in the hand-eye coordination learning with the exception of the last joint (end-effector spin angle) for which the amplitude is 11°.

Parameters	ϵ_{Trans}	ϵ_{Rot}	N_{eps}	N_{b}	c_{cam}	c_{cart}
Values	0.01	0.04	100	32	$-\frac{1}{30}$	$-\frac{1}{40}$

TABLE I
PARAMETER VALUES

Phase	P_1	P_2
Training	4000	40000
Test	300	1000

TABLE II
N_{tot} VALUES

V. Results and Discussion

We present here the results in the two previously defined settings which are used in the experiments.

A. $P1$

Figure 4 shows the touching rates. First, we observe that $f_t^{\text{sparsePen}}$ is zero almost everywhere. In the experiment, it touches one time and is never able to further improve. This is because transitions with high TD values are not trained enough through DDPG. Second, we have similarly good performance of over 90% for the proposed reward r_{cam} and the reward function r_{cart}, which requires forward kinematics and 3D object pose information. For this task, this shows that a shaping term which requires forward kinematics and precise 3D target information can be replaced without loss of performance by a weakly-supervised shaping reward resulting from learning simpler predecessor tasks. The reason is that the objective of the task is defined by the terminal sparse reward. The shaping term is used only to get closer to good touching postures. When the terminal rewards are back-propagated through the Q function to all the trajectory states, the shaping term becomes small compared to the Q values and therefore less important.

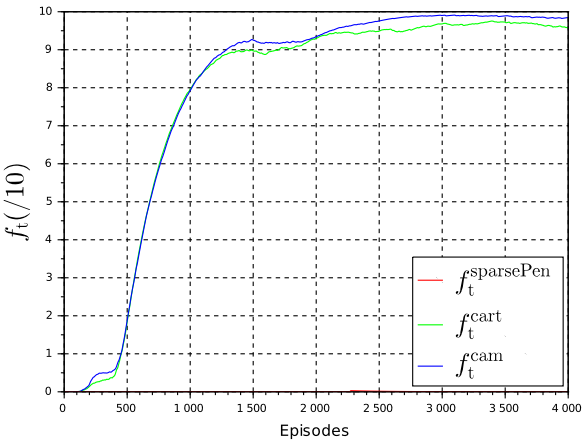

Fig. 4. Learning curves for the $P1$ problem

The results from policy exploitation show that we reach the object with 100 % of successes for both policies for 300 test experiments. This shows that the residual errors at the end of learning were only due to the exploration noise applied to the policy.

B. $P2$

Figure 5 shows the results of training experiments for the $P2$ problem. We still observe that, with our setting, sparse rewards do not allow to efficiently learn a touching policy. Furthermore, we observe again that our weakly-supervised reward and its supervised counterpart give similar performances. It means that with very different initial positions including those with difficult end-effector orientations, the robot can learn to correctly orientate its end-effector given a long-delayed sparse reward. In the test phase, we

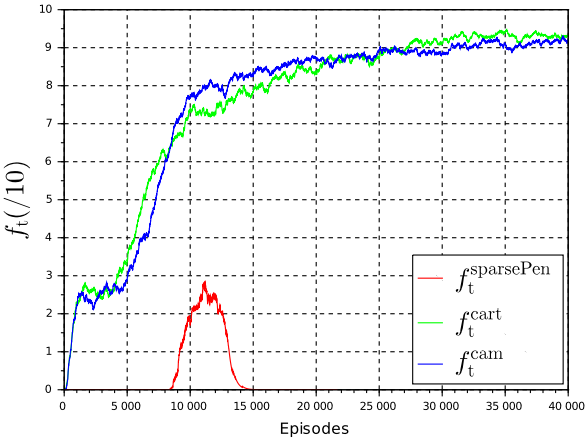

Fig. 5. Learning curves for the $P2$ problem

observe 94,9 % and 95.4 % of success for the policy learned with, respectively, r_{cam} and r_{cart}, which represent good and

comparable results. This shows that our reward signal with very little supervision allows to learn a complex touching task from different initial positions.

VI. Conclusion

For a complex palm-touching task, we have designed a stage-wise learning procedure which does not require any kinematic models, hand-crafted features, calibration parameters and needs minimal human supervision. Inspired by developmental robotics, our approach involves three steps. First, a binocular vision system learns to fixate an object. This implicitly encodes the position of the object in the 3D camera angle space, while requiring only an autoencoder pre-training step. Second, by learning to fixate the end-effector from its finger moves, the robot learns a hand-eye coordination function which maps robot joint angles to camera angles. In a nutshell, from a given arm configuration, this function allows to localize the end-effector in the 3D camera angle space. Third, we compute the reward signal for the palm-touching task by summing an informative shaping reward computed with the previous steps and a sparse terminal reward indicating the true objective of the task. This shaping term encourages the end-effector to be close to the object in the 3D camera angle space.

The experiments show that our reward signal allows to learn complex touching policies, notably from various initial positions. In addition, policies learned with our reward and with a supervised counterpart reward give similar performances. This shows that for this complex task, our reward based on weak supervision can replace a reward based on forward kinematics and object tracking.

One advantage of our framework is that it is suitable for other types of manipulation tasks with different terminal rewards such as object grasping or pushing. Indeed, our shaping term is used to bring the robot closer to terminal rewards. Thus, switching to another task just requires to change the terminal sparse reward.

Even though our reward computation framework is based on weak supervision, it still has a limitation regarding autonomy. Indeed, even if the object fixation method hardly requires external information, it cannot cope with environment changes in an online way. If the environment varies, the autoencoder has to be trained again. In future work, we wish to implement an object detection method which would adapt to environment variations. Finally, we wish to learn the same task in a real world setting.

References

[1] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *ICRA*, 2016.

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end Training of Deep Visuomotor Policies," *J. Mach. Learn. Res.*, 2016.

[3] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," in *IROS*, 2017.

[4] Y. Chebotar, K. Hausman, M. Zhang, G. S. Sukhatme, S. Schaal, and S. Levine, "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning." in *ICML*, 2017.

[5] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning." in *Robotics: Science and Systems*, 2011.

[6] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates." in *ICRA*, 2017.

[7] Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara, "Deep dynamic policy programming for robot control with raw images." in *IROS*, 2017.

[8] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search." in *ICRA*, 2015.

[9] H. Hoffmann, W. Schenck, and R. Möller, "Learning visuomotor transformations for gaze-control and grasping." *Biological Cybernetics*, 2005.

[10] D. Carey, R. Coleman, and S. Della Salla, "Magnetic Misreaching," *Cortex*, 2018.

[11] K. Fischer, "A Theory of Cognitive Development: The Control and Construction of Hierarchies of Skills," *Psychological Review*, 1980.

[12] F. Nori, L. Natale, G. Sandini, and G. Metta, "Autonomous learning of 3D reaching in a humanoid robot." in *IROS*, 2007.

[13] J. Law, P. Shaw, M. Lee, and M. Sheldon, "From Saccades to Grasping: A Model of Coordinated Reaching Through Simulated Development on a Humanoid Robot," *IEEE Trans. on Autonomous Mental Development*, 2014.

[14] E. Chinellato, M. Antonelli, B. J. Grzyb, and A. P. del Pobil, "Implicit Sensorimotor Mapping of the Peripersonal Space by Gazing and Reaching," *IEEE Trans. on Autonomous Mental Development*, 2011.

[15] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *Int. J. of Rob. Res.*, 2017.

[16] A. Boularias, J. A. D. Bagnell, and A. T. Stentz, "Learning to Manipulate Unknown Objects in Clutter by Reinforcement," in *AAAI*, 2015.

[17] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning." *Machine Learning*, 1996.

[18] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse Curriculum Generation for Reinforcement Learning," in *CoRL*, 2017.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR*, 2016.

[20] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *ICML*, Beijing, China, 2014.

[21] V. Mnih, K. Kavukcuoglu , D. Silver, A. Rusu Andrei, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski , S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg , and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[22] M. J. Hausknecht and P. Stone, "Deep Reinforcement Learning in Parameterized Action Space." in *ICLR*, 2016.

[23] F. de La Bourdonnaye, C. Teulière, T. Chateau, and J. Triesch, "Learning of binocular fixations using anomaly detection with deep reinforcement learning." in *IJCNN*, 2017.

[24] G. Metta and P. Fitzpatrick, "Early Integration of Vision and Manipulation," *Adaptive Behavior special issue on Epigenetic Robotics*, 2003.

[25] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[26] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade, "Towards Generalization and Simplicity in Continuous Control," in *Advances in NIPS*, 2017.

[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *ACM*, 2014.

[28] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2015.