Within Reach? Learning to touch objects without prior models

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

Within Reach? Learning to touch objects without prior models

François de La Bourdonnaye[§], Céline Teulière[†], Thierry Chateau[†], and Jochen Triesch[§]
[§]Frankfurt Institute for Advanced Studies, Frankfurt am Main, Germany
[†] Université Clermont Auvergne, CNRS, Institut Pascal, Clermont-Ferrand, France

***Abstract*—Human infants learn to manipulate objects in a largely autonomous fashion, starting without precise models of their bodies' kinematics and dynamics. Replicating such learning abilities in robots would make them more flexible and robust and is considered a grand challenge of Developmental Robotics. In this paper, we propose a developmental method that allows a robot to learn to touch an object, while also learning to predict if the object is within reach or not. Importantly, our method does not rely on any forward or inverse kinematics models. Instead it uses a stage-wise learning approach combining deep reinforcement learning and a form of self-supervised learning. In this approach, complex skills such as touching an object or predicting if it is within reach are learned on top of more basic skills such as object fixation and eye-hand-coordination.**

***Index Terms*—Sensori-motor learning, reinforcement learning, object reachability**

I. Introduction

For any manipulation robot, estimating the reachability of objects is a very important ability. Traditionally, the problem has been approached by using inverse kinematics, i.e. a mapping from 3D world coordinates to robot joint angles that would bring the robot's gripper to the desired 3D location. For example, [1] represents the reachable workspace of a humanoid robot as a set of 3D Cartesian points. Their method samples random 3D positions and simply checks if inverse kinematics provide a solution. Goal babbling [2] can be considered as a way to learn reachable 3D points while also learning inverse kinematics, provided that the forward kinematics are known. Finally, [3] estimates the workspace reachability of a mobile manipulator using a probabilistic motion planner. Importantly, all of these methods assume knowledge of the forward and/or inverse kinematics, which is undesirable in the context of Developmental Robotics. In contrast, [4] can dispense with it, because the reachability estimation is based directly on the robot's gaze. Specifically, the pan, tilt, and vergence angles of the binocular vision system implicitly encode the 3D position of an object. Thus, they can be used as a surrogate of a 3D Cartesian point, which may be more biologically plausible [5]. Here, we take inspiration from this idea and combine it with our own prior work on the learning of an object touching skill using deep reinforcement learning [6].

Our learning approach comprises three stages during which successively more sophisticated skills are acquired. Importantly, the entire learning process does not require any forward or inverse kinematics models, but relies exclusively on (deep) reinforcement learning and a form of self-supervised learning. In this sense, the complex behavior is essentially learned *from scratch*. The three learning stages are as follows. First, the robot learns to fixate an object to localise it. Second, it jointly learns to fixate on its own end-effector while learning a hand-eye coordination mapping. In the third stage, the robot uses the skills acquired in the two previous stages to jointly learn to reach for and touch objects and predict if an object is within reach or not. The eye-hand-coordination mapping learned during the second stage essentially substitutes for the forward kinematics during the final stage.

The fixation tasks as well as the reaching behaviours are learned using deep reinforcement learning. The hand-eye coordination and the reachability prediction are learned using a form of self-supervised learning which uses supervised learning with self-generated data. The final reachability prediction outputs a probability of the target object being within reach given the joint angles of the robot's cameras when fixating the target object.

II. Methods

A. Background

1) Reinforcement learning: Reinforcement learning (RL) is a class of algorithms used to solve sequential decision making problems through learning. Most RL algorithms are 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 ($R : S \times A \rightarrow \mathbb{R}$).

The agent learns from interaction with the environment and gathers transitions $< \boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s'} >$. $\boldsymbol{s}$ represents a state and $\boldsymbol{a}$ the action performed at state $\boldsymbol{s}$. After the execution of $\boldsymbol{a}$, the agent receives a reward r and reaches a new state $\boldsymbol{s'}$.

The goal of an RL agent is to adapt its behaviour to maximise the amount of future rewards. In this paper, we consider 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

This work was funded 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 Horizon 2020 Research and Innovation Programme under grant agreement no. 713010 (GOAL-Robots Goal-based Open-ended Autonomous Learning Robots) and through the program Regional competitiveness and employment (ERDF Auvergne region), and by the Auvergne region. J. Triesch acknowledges support from the Quandt foundation.

k. To optimise this criterion, we train a deterministic policy $\pi : S \rightarrow A$ jointly with the state-action value function Q in an actor-critic set-up:

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

2) Deep reinforcement learning: RL suffers from the curse of dimensionality. The common approach for dealing with this problem is to use function approximation, in particular using neural networks. This approach has become very popular with the arrival of GPUs and the wide-spread application of deep learning. In our work, we use the DDPG algorithm [7], which can solve RL problems with a high-dimensional state space and a continuous action space. This algorithm combines the deterministic policy gradient algorithm [8] and the deep Q network [9].

DDPG is an "actor-critic" algorithm updating the critic $Q_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\phi}$ and the deterministic policy $\pi_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$ as follows. At each time-step, we sample (uniformly) a mini-batch of N_b transitions from a large memory buffer T_buf of size N_trans:

$< \boldsymbol{s_i}, \boldsymbol{a_i}, r_i, \boldsymbol{s'_i} >_{i \in \{1,...,N_\mathrm{b}\}} \in S \times A \times \mathbb{R} \times S$.

The targets of the $Q_{\boldsymbol{\phi}}$ neural network are computed using a TD(0) update with a learning rate equal to 1:

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

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

$$\boldsymbol{\phi'_{t+1}} = \tau \boldsymbol{\phi_t} + (1 - \tau) \boldsymbol{\phi'_t} , \quad \boldsymbol{\theta'_{t+1}} = \tau \boldsymbol{\theta_t} + (1 - \tau) \boldsymbol{\theta'_t} , \quad (3)$$

The $Q_{\boldsymbol{\phi}}$ network updates its weights by minimizing the squared error $\frac{1}{2N_\mathrm{b}} \sum_{i=1}^{N_\mathrm{b}} \left(y_i - Q_{\boldsymbol{\phi}}(\boldsymbol{s_i}, \boldsymbol{a_i}) \right)^2$. Using target networks greatly contributes to the learning stability of the neural networks and using a memory buffer helps to satisfy the constraint of i.i.d samples for learning with neural networks.

Using the $Q_{\boldsymbol{\phi}}$ network and the fact that the policy is deterministic, the following policy gradient is derived:

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

This update makes the policy select the actions that maximise the Q function at the batch states. In addition to this algorithm, we use the inverting gradient procedure of [10] to bound the actions.

B. Task definition

The robot's goal is to learn to reach for and touch an object on a table in front of the robot and to predict if a successful reach is possible. We use a 7 DOF arm with a pair of cameras (see Fig. 1). Successful reaching means that the robot touches the object with the "palm" of its end-effector. For reachability prediction the robot maps its gaze angles onto a prediction whether the object is within reach or not. Learning occurs in a stage-wise approach, where the robot first learns simpler skills

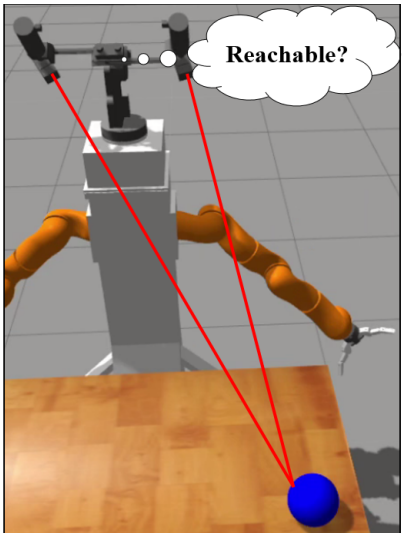

Fig. 1. Illustration of the reachability prediction skill. In our experiments, the robot's left arm (from the robot's view) is used.

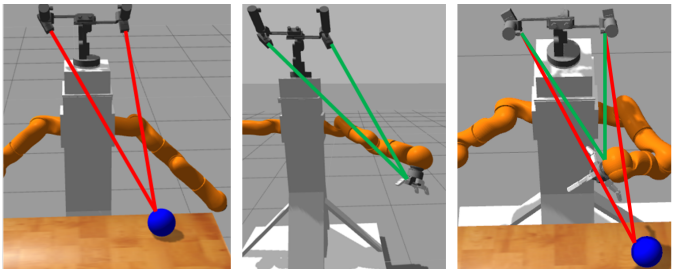

Fig. 2. Overall scheme of the stage-wise learning procedure

of object fixation and hand-eye coordination before learning the reaching skill (see Fig. 2 for a schematic view).

We use the following notations:

- $\mathbf{I} = (\boldsymbol{I}^{\mathbf{left}}, \boldsymbol{I}^{\mathbf{right}})$ represents the images from the left and right cameras.
- $\mathbf{q} = (\boldsymbol{q}^{\mathbf{camera}}, \boldsymbol{q}^{\mathbf{robot}})$ represents the 3 camera joint angles (one common tilt angle and two independent pan angles) and 7 robot arm joint angles.
- $\boldsymbol{c}_\mathbf{b}$ represents the touch sensation of the robot. It is an 8-element binary vector representing 8 areas of the robot's fingers. They correspond to the proximal, medial, and distal areas of the three fingers, with the exception of the proximal area of one finger which is linked to the palm. When in contact, the corresponding entry in $\boldsymbol{c}_\mathbf{b}$ becomes 1 and 0 otherwise.

The reachability prediction takes the form of a function mapping camera joint angles $\boldsymbol{q}^{\mathbf{camera}}$ to a probability $P \in [0, 1]$ that the robot will be able to touch the object. This mapping is approximated by a feed-forward neural network trained with example pairs $(\boldsymbol{q}^{\mathbf{camera}}, P_\mathrm{T})$ collected while the robot learns to reach. More precisely, after each episode, we

record the pair $(\boldsymbol{q}^{\mathbf{camera}}, P_{\mathrm{T}})$, where P_{T} is defined as:

$$P_{\mathrm{T}} = \begin{cases} 1, & \text{if reach was successful,} \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Our stage-wise learning framework involves three successive tasks that are described in the following.

C. Learning object fixation

First, the robot learns from raw pixels to fixate on the object with its two cameras [11]. This consists of moving the cameras such that the object comes to the image centers of both cameras. This task is learned using the DDPG algorithm with a weakly supervised reward function. The learned object fixation policy is noted $\pi_{\boldsymbol{\psi}}^{\mathrm{fix}}$. The reward signal is computed using an anomaly object localisation technique and requires neither calibration parameters nor hand-crafted visual modules (see [11] for details). At the end of an object fixation, the camera joint angles $\boldsymbol{q}_{\mathbf{fix}}^{\mathbf{camera}}$ implicitly encode the object position in 3D space. This object localisation technique has been proven efficient to learn to touch an object [6].

D. Learning hand-eye coordination

In the second step, the robot learns a hand-eye coordination function $f_{\boldsymbol{\eta}}$, which maps arm joint angles to "virtual" camera joint angles:

$$\boldsymbol{q}_{\mathbf{virt}}^{\mathbf{camera}} = f_{\boldsymbol{\eta}}(\boldsymbol{q}^{\mathbf{robot}}). \tag{6}$$

These "virtual" camera joint angles correspond to the ones which would make the camera system fixate on the end-effector. The function $f_{\boldsymbol{\eta}}$ is approximated by a feed-forward neural network and the input-output pairs $(\boldsymbol{q}^{\mathbf{robot}}, \boldsymbol{q}_{\mathbf{virt}}^{\mathbf{camera}})$ are generated via the parallel learning of an end-effector fixation task. The latter is learned from raw pixels using the DDPG algorithm and a reward function requiring little prior knowledge. The reward signal is computed based on the localisation of the end-effector in the left and right images using a pre-defined motion of an end-effector finger (see [6] for details).

E. Learning reaching skills

1) Learning to reach: In the final learning step, the object fixation policy $\pi_{\boldsymbol{\psi}}^{\mathrm{fix}}$ and the hand-eye coordination mapping $f_{\boldsymbol{\eta}}$ are utilised to learn reaching and reachability prediction. The reaching behaviour is learned using the DDPG algorithm. The state space S is composed of the arm and camera joint angles $\boldsymbol{q}$ as well as eight binary tactile sensors $\boldsymbol{c}_{\mathbf{b}}$ attached to the fingers of the hand. Images are not required here because we use a single object and consider that the camera joint angles give sufficient information about the 3D object position. However, they would be necessary if objects with different shapes were used in the experiments. The actions are changes of the robot joint angles: $\boldsymbol{a} = \boldsymbol{\Delta q}^{\mathbf{robot}}$, which are seven real-valued scalars. The reward signal for the reaching task is the sum of three components:

$$r = r_{\mathrm{touch}} + r_{\mathrm{contact}} + r_{\mathrm{shaping}}. \tag{7}$$

r_{touch} defines the objective of the task:

$$r_{\mathrm{touch}} = \begin{cases} 1, & \text{if success,} \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where success means that the robot has reached the object with its hand's palm, ending the episode.

r_{contact} penalises contacts between the end-effector and the table. This term is negative when contact occurs and prevents the robot from exploring too much areas in which one of the fingers (but not the palm) touches the table:

$$r_{\mathrm{contact}} = \begin{cases} -0.01, & \text{if contact,} \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

The constant contact value has been empirically found and its absolute value must be small compared to the sparse reward value. If it is too high, the arm is repelled by the table and this slows down learning.

r_{shaping} is an additional shaping reward which favors actions that bring the robot end-effector closer to the point the robot is fixating with its cameras:

$$r_{\mathrm{shaping}} = \begin{cases} 0, & \text{if success,} \\ -\frac{1}{30}||\boldsymbol{q}_{\mathbf{fix}}^{\mathbf{camera}} - \boldsymbol{q}_{\mathbf{virt}}^{\mathbf{camera}}||_2, & \text{otherwise.} \end{cases} \tag{10}$$

Here, $\boldsymbol{q}_{\mathbf{fix}}^{\mathbf{camera}}$ is the vector of camera joint angles which makes the camera system fixate on the object. To obtain it, the object fixation policy $\pi_{\boldsymbol{\psi}}^{\mathrm{fix}}$ is used. $\boldsymbol{q}_{\mathbf{virt}}^{\mathbf{camera}}$ is computed using (6) and represents the camera joint angles which make the camera system fixate on the end-effector. This requires instantaneous robot joint angles $\boldsymbol{q}^{\mathbf{robot}}$ and the learned hand-eye coordination mapping $f_{\boldsymbol{\eta}}$. As for r_{contact}, the shaping term should be small with respect to the sparse reward value. The coefficient of $-\frac{1}{30}$ has been found empirically.

The sum of these three reward terms has been proven efficient to learn the object reaching task for various initial conditions and requires little prior knowledge [6].

The learning of the reaching policy uses an episodic set-up. For each episode, the robot uses $\pi_{\boldsymbol{\psi}}^{\mathrm{fix}}$ to fixate on the object and the arm joint angles are set to their initial values $\boldsymbol{q}_{\mathbf{0}}^{\mathbf{robot}}$. The robot learns to reach during N_{tot} episodes of at most N_{max} iterations. An episode ends if the robot successfully reaches the object or after N_{max} iterations. For exploration, we use the Ornstein-Uhlenbeck process. A noise term $\epsilon_j(t)$ is added to the motor command $\Delta q^{\mathrm{robot}}$ in every time-step and is computed as follows:

$$\epsilon_j(t) = \theta_j \mu_j + (1 - \theta_j)\epsilon_j(t-1) + \xi_j(t). \tag{11}$$

θ_j is called the mean reversion speed, μ_j is the equilibrium value, $\epsilon_j(t-1)$ is the noise computed at time-step $t-1$ and $\xi_j(t)$ is a centered Gaussian noise of standard deviation σ_j. This exploration strategy is particularly helpful for discovering behaviours that imply a succession of actions which have the same direction.

To accelerate learning, we use a strategy to avoid situations where the robot's movement is blocked because of unwanted contact of the fingers with the table or the object. When such

a contact occurs, the robot takes a backward action to go to a previous contact-free position.

2) Learning reachability: The reachability mapping $Re_{\boldsymbol{\alpha}}$ is learned using supervised learning. The input-output pairs are collected during the learning of reaching movements. Specifically, the data $(\boldsymbol{q}^{\mathbf{camera}}, P_{\mathrm{T}})$ are added to a database when the reaching policy starts to perform well (from N_{start} episodes) and reachability learning starts when this database contains a sufficient number of elements N_{upd}.

The database of the reachability mapping $\mathcal{D}$ is a circular buffer of size $N_{\mathcal{D}}$. This architecture is chosen because it allows to forget episodes in which the reaching policy is not performing well. If, e.g., the robot does not reach a reachable object position, the algorithm should be able to forget this corrupted datum. Algorithm 1 summarises the whole joint learning procedure.

III. Experiments

The experiments are carried out in a simulated environment using the Gazebo simulator jointly with the ROS middleware.

A. Implementation details

For all the neural network algorithms, we use the caffe library [12]. A GPU (nvidia GeForce GTX Titan X) is used for the experiments. For the reaching 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 hand-eye coordination function is a neural network with 2 fully connected hidden layers of 10 and 5 neurons. The neural network representing the reachability prediction has 2 fully connected hidden layers of 5 units. The neural network structures approximating the policy and the Q function for the fixation tasks are described in [11]. The Adam solver [13] is used to update the weights. Parameter values from Algorithm 1 are listed in Table I, II and III.

Parameters	γ	$N_{\max}$	N_{b}	N_{trans}	N_{tot}
Values	0.99	100	256	60 000	40 000

TABLE I
Parameter values (1/2)

Parameters	N_{upd}	$N_{\mathcal{D}}$	N_{start}
Values	200	20 000	20 000

TABLE II
Parameter values (2/2)

Parameters	θ_j	μ_j	$\sigma_1, \sigma_2, \sigma_3, \sigma_4$	$\sigma_5, \sigma_6, \sigma_7$
Values	0.8	0	0.01	0.04

TABLE III
Ornstein-Uhlenbeck process parameters

B. Ground-truth reachability estimate

To evaluate the learned reachability network, we need to estimate the ground-truth workspace reachability. For this estimation, we build on the knowledge of the robot structure. First, we empirically know that the left arm can reach an object if it is put at the bottom left of the table (from the robot's point of view). Second, we know from the geometry that the furthest points that the robot can reach follow two circular arcs of different radii. Indeed, in Fig. 3, we can see the ways to reach the furthest points. In the first configuration, the arm is outstretched and the second revolute joint can make the arm move while keeping the arm outstretched. The second configuration corresponds to the position where the robot cannot move around its second revolute joint, because it would be hitting its structure. Thus, the arm can reach the furthest points only by moving around its fourth revolute joint and forcing the end-effector to keep a given orientation.

Using these facts, the workspace limits are some borders of

Algorithm 1 Joint learning of reaching and reachability

Parameters:
N_{trans}: size of circular transition buffer
γ: discount factor
Γ: parameters of the Ornstein-Uhlenbeck process
N_{tot}: total number of episodes
$N_{\max}$: maximum number of iterations per episode
N_{b}: number of transitions per batch
N_{upd}: number of samples in the reachability database required to start the learning of reachability
$N_{\mathcal{D}}$: size of the reachability database
N_{start}: number of episodes required to start the learning of reachability

Inputs:
$\boldsymbol{q}_{\mathbf{0}}^{\mathbf{robot}}$: initial arm joint angles
$\pi_{\psi}^{\mathrm{fix}}$: fixation policy
$f_{\boldsymbol{\eta}}$: hand-eye coordination mapping

Outputs: $Q_{\boldsymbol{\phi}}$, $\pi_{\boldsymbol{\theta}}$, $Re_{\boldsymbol{\alpha}}$

Steps:
Initialise $\mathcal{D}, T_{\mathrm{buf}}$, $Q_{\boldsymbol{\phi}}$, $\pi_{\boldsymbol{\theta}}$ and $Re_{\boldsymbol{\alpha}}$
$n_{\mathrm{eps}} \leftarrow 0$
while $n_{\mathrm{eps}} < N_{\mathrm{tot}}$ **do**
  $t \leftarrow 0$ and $cond \leftarrow$ False
  Reset arm joint angles and randomly place the object
  Fixate on the object using $\pi_{\psi}^{\mathrm{fix}}$
  Go to the position $\boldsymbol{q}_{\mathbf{0}}^{\mathbf{robot}}$
  while !$cond$ **do**
    if blocked **then**
      Apply a backward action $\boldsymbol{a_t}$
    else
      Update $\boldsymbol{\epsilon}^{\Gamma}(t)$ using equation (11)
      Apply $\boldsymbol{a_t} \leftarrow \pi_{\boldsymbol{\theta}}(\boldsymbol{s_t}) + \boldsymbol{\epsilon}^{\Gamma}(t)$
    Compute the reward signal r_t and sense $\boldsymbol{s_{t+1}}$
    Add $< \boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{r_t}, \boldsymbol{s_{t+1}} >$ to T_{buf}
    Pick N_{b} random transitions from T_{buf}
    Update $Q_{\boldsymbol{\phi}}$ and $\pi_{\boldsymbol{\theta}}$ using DDPG
    $cond \leftarrow (Success)$ or $(t == N_{\max} - 1)$
    $t \leftarrow t + 1$
  if $n_{\mathrm{eps}} \geq N_{\mathrm{start}}$ **then**
    Append $(\boldsymbol{q}^{\mathbf{camera}}, Success)$ to $\mathcal{D}$
    if $(size(\mathcal{D}) \geq N_{\mathrm{upd}})$ **then**
      Update Re_{α}
  $n_{\mathrm{eps}} \leftarrow n_{\mathrm{eps}} + 1$

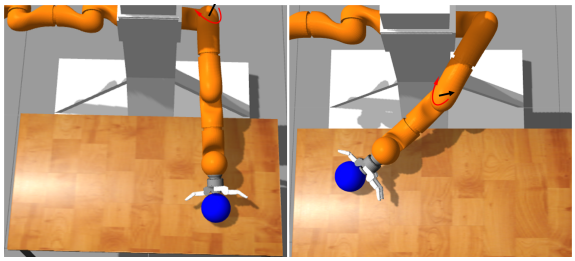

Fig. 3. Top views of the left arm illustrating the limits of reachability. The left figure illustrates construction of the first circular arc by rotating the shoulder joint. The right figure illustrates the construction of the second circular arc by rotating the elbow joint.

the table and the mentioned circular arcs. Then, we fit the two circular arcs on measured points using least mean squares.

C. Training evaluation

Our experiments start by the analysis of the training data, which has two objectives. First, we want to evaluate how accurate the learned reachability mapping is compared to the estimated ground-truth reachability. For this, we plot the reachability prediction error, which is the difference between the prediction of the learned mapping and the ground-truth estimate prediction, as a function of episode number. The prediction of the learned mapping is obtained by thresholding the output of the reachability prediction network at $1/2$. Second, we wish to evaluate how often the robot matches the ground-truth prediction (through the learned reaching policy). To this end, we compare the reaching success with the ground-truth object reachability. We call this measure the reaching performance, because we measure whether the policy reaches reachable positions. For each of these measures, we provide 95 percent confidence intervals $[\mu - \frac{1.96\sigma}{\sqrt{N_{\text{run}}}}, \mu + \frac{1.96\sigma}{\sqrt{N_{\text{run}}}}]$. N_{run} (equal to 3) is the number of run experiments, μ and σ are the average and the standard deviation of the measure at a given episode.

In Figure 4, we notice first that the reaching performance converges to a decent performance of around 95%. This shows that the reaching learning framework allows to learn to reach in most areas where the object is reachable. Besides, we notice that the reachability prediction error falls below 10% within the 40,000 episodes of training.

D. Resulting reachability

After analysing the training data, we wish to evaluate the learned reachability network. For a regular grid of object positions on the table, the object fixation policy π_{ψ}^{fix} is applied. Then, the resulting camera joint angles $\boldsymbol{q}_{\mathbf{fix}}^{\mathbf{camera}}$ feed the reachability prediction network. Figure 5 represents a heat map of the predicted reaching probabilities. The white curve represents the border of the estimated ground-truth reachability.

We observe that the overall results are very good. The border of the learned reachability prediction mostly matches the one of the estimated ground-truth reachability. The small errors close to the border may occur because of object localisation

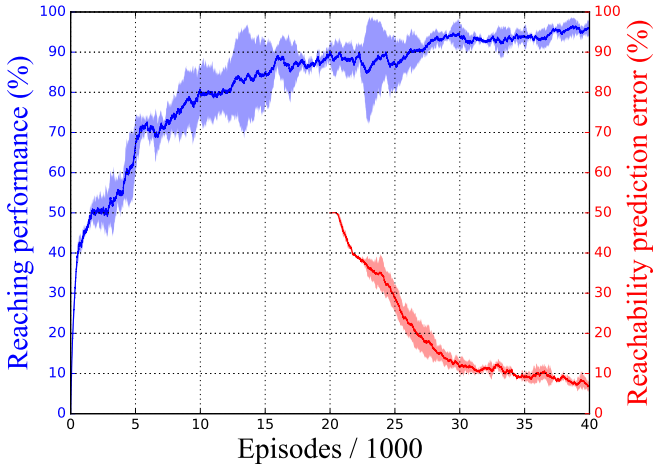

Fig. 4. Reaching performance and reachability prediction error as a function of episode number. Lines are averages from three independent experiments and dashed regions show 95 percent confidence intervals.

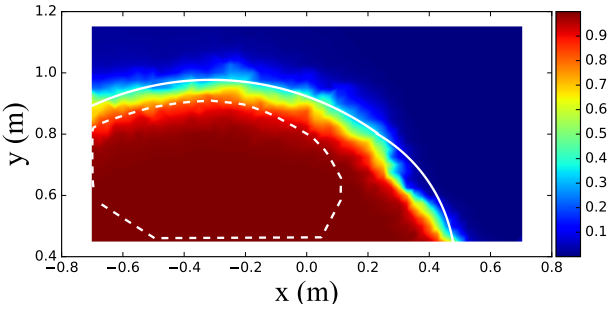

Fig. 5. Prediction of reachability across the table. The solid white curve shows the estimated ground truth reachability limit. The dashed white curve shows the convex hull of the projections of the end-effector positions used during training of the hand-eye coordination onto the table. The robot base is located at the origin of the coordinate system.

errors implied by the object fixation policy. Furthermore, we observe that the robot can still efficiently learn both to reach and the reachability in most of the reachable and unvisited (during hand-eye coordination training) area. In Figure 5, the latter is localised outside the space surrounded by a dashed curve and inside the one surrounded by the full curve. However, it has difficulties with a specific area. To explain this phenomenon, the most probable hypothesis is that the hand-eye coordination outputs aberrant values for some arm joint angles required to reach these object positions. Then, this slows down the learning of reaching and reachability. We believe that these object positions could be mastered if we make experiments run longer or if the hand-eye coordination is trained on a larger domain.

IV. Discussion

Building robots that can autonomously learn to interact with their environment, manipulate objects, and communicate with people remains a grand challenge for Developmental Robotics.

Here, we have proposed a learning approach through which a robot can learn to reach for and touch objects while also learning if an object is within reach or not. Importantly and in contrast to prior work, our approach works completely without the use of any forward or inverse kinematics models. Our approach also does not require any calibration parameters, visual markers, or hand-tuned vision modules. Instead, it relies on a combination of deep reinforcement learning and a form of self-supervised learning, where input-output training samples are generated through the robot's own behavior. Thus, our learning is more *autonomous*, as it requires less prior knowledge or structures than previous approaches.

Several studies used also fixation and/or hand-eye coordination to learn reaching behaviours [14], [15]. However, they do not learn an explicit model of reachability. In contrast, [4] jointly learned reaching and reachability, but their approach differs from ours in some crucial aspects. First, in our method the binocular fixation and the reaching control are learned using reinforcement learning whereas in their case the binocular fixation task is pre-wired and the reaching task is achieved through the inversion of a learned hand-eye coordination mapping. Second, unlike their approach, we do not assume any visual marker or hand-crafted segmentation to detect the pixellic end-effector and object localisations. Third, both our method and theirs learn the reachability as a function mapping the robot's gaze to the reachability prediction. However, we differ in the ways the targets (for supervised learning) are generated. [4] detects if an object has been reached from the distance between the end-effector and the object in terms of camera joint angles. Then, the targets are computed based on this distance if the object has not been reached and on the optimality of the resulting arm configuration otherwise. In our strategy, we assume that the robot has reached the object when the palm of the end-effector touches the object. Then, we generate a binary target according to the reaching success. This means that the orientation control of the end-effector is involved in our learning of reaching and implies a more complex reaching behaviour. The experiments in simulation show that our framework is efficient to learn both reaching and reachability.

Despite the benefits of our method, the learning procedure still has a number of limitations. First, in our approach we strictly enforce a specific ordered sequence in which the different skills are learned. This makes the approach somewhat inflexible. If, e.g., the hand-eye coordination mapping needs to be re-learned because of some damage to the robot or problem with the joint encoders, a human has to decide this. A system that can autonomously recalibrate "on the job" would be superior and more biologically plausible, see, e.g., [16]. In the future, it will also be interesting to investigate to what extent such a stage-wise learning could emerge autonomously as a result of a curiosity mechanism [17].

A second shortcoming is that once the robot fixates the target object, the subsequent reaching does not make use of any visual information anymore, but only relies on the learned eye-hand-coordination model. Thus the proposed approach would not be suitable for reaching towards moving objects. Note that using images for reaching is not incompatible with our framework since they can be added to the state space.

Third, our approach has been proven efficient only in a simulated environment and may need substantial efforts to be validated using a physical robot.

Despite these limitations, an advantage of our framework is its generality. While we have focused on learning to touch an object with the palm of the hand, it seems likely that a whole repertoire of skills could be learned in the future by providing different reward functions during additional learning phases.

REFERENCES

[1] Y. Guan and K. Yokoi, "Reachable Space Generation of A Humanoid Robot Using The Monte Carlo Method," in *IROS*, 2006.

[2] M. Rolf, "Goal babbling with unknown ranges: A direction-sampling approach," in *ICDL*, 2013.

[3] J. Yang, P. Dymond, and M. Jenkin, "Exploiting hierarchical probabilistic motion planning for robot reachable workspace estimation," in *Informatics in Control Automation and Robotics*, 2011.

[4] L. Jamone, M. Brandao, L. Natale, K. Hashimoto, G. Sandini, and A. Takanishi, "Autonomous online generation of a motor representation of the workspace for intelligent whole-body reaching," *Robotics and Autonomous Systems*, 2014.

[5] W. P. Medendorp, H. C. Goltz, T. Vilis, and J. D. Crawford, "Gaze-centered updating of visual space in human parietal cortex," *Journal of Neuroscience*, 2003.

[6] F. de La Bourdonnaye, C. Teulière, J. Triesch, and T. Chateau, "Stage-Wise Learning of Reaching Using Little Prior Knowledge," *Frontiers in Robotics and AI*, 2018.

[7] 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.

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

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. 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.

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

[11] 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.

[12] 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.

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

[14] A. Ghadirzadeh, A. Maki, and M. Björkman, "A sensorimotor approach for self-learning of hand-eye coordination," *IROS*, 2015.

[15] E. Chinellato, M. Antonelli, B. J. Grzyb, and A. P. del Pobil, "Implicit sensorimotor mapping of the peripersonal space by gazing and reaching," *IEEE Transactions on Autonomous Mental Development*, 2011.

[16] L. Lonini, S. Forestier, C. Teulière, Y. Zhao, B. E. Shi, and J. Triesch, "Robust active binocular vision through intrinsically motivated learning," *Frontiers in neurorobotics*, 2013.

[17] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE transactions on evolutionary computation*, 2007.