2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561937

End-to-end grasping policies for human-in-the-loop robots via deep reinforcement learning*

Mohammadreza Sharif[1], Deniz Erdogmus[1], Christopher Amato[2], and Taskin Padir[1]

***Abstract*— State-of-the-art human-in-the-loop robot grasping is hugely suffered by Electromyography (EMG) inference robustness issues. As a workaround, researchers have been looking into integrating EMG with other signals, often in an *ad hoc* manner. In this paper, we are presenting a method for end-to-end training of a policy for human-in-the-loop robot grasping on *real* reaching trajectories. For this purpose we use Reinforcement Learning (RL) and Imitation Learning (IL) in DEXTRON (DEXTerity enviRONment), a stochastic simulation environment with real human trajectories that are augmented and selected using a Monte Carlo (MC) simulation method. We also offer a success model which once trained on the expert policy data and the RL policy roll-out transitions, can provide transparency to how the deep policy works and when it is probably going to fail.**

I. Introduction

Robot prosthetic hands provide a solution to bring back part of the lost ability of people with upper limb amputation by offering a way to interact with the environment through actuated fingers. Traditionally, there have been several methods to control a robot prosthetic hand, such as Electromyographic (EMG) control, Electroneurographic (ENG) control, and body-powered control methods. Although EMG control arguably provides the finest control by tele-operating the robot actuators, it is not the prevalent method of choice in the market. This lack of popularity of the EMG control method is mainly attributed to its lack of robustness, which is explained by real-life versus lab variations such as electrode shift [1], skin-electrode impedance, muscle fatigue [2], and stump posture change [3]. This lack of robustness often causes unpredictability which can lead to passive usage of the robot hand or its total abandonment [4]. Some works have tackled the robustness issue via increasing robot autonomy by using other informative signals along with EMG. Dosen et al. [5] et al. use RGB-D camera to detect grasp type. Gigli et al. [6] use Gaze information as a guide for detecting human intent and grasp type. In [7], human hand trajectories are utilized to infer time-to-arrive which can be used to gradually shape the hand. Zhuang et al. [8] use tactile sensors to grasp the object whenever the hand contacts the object. A low-level controller then maintains the force to avoid object slippage. During reach-to-grasp motion, humans change the shape of their hand gradually to conform to the object contours [9]. While EMG control provides more intuitive control than the other control methods like body-powered control, due to delayed [10] and noisy inference, the overall motion kinematics seems less smooth and reach/grasp seem to be less coordinated with respect to healthy humans [11]. This compromises grasp intuitiveness and the overall time for grasping [11]. EMG control method also needs active user attention and excessive energy consumption [12] which may not be desirable for lasting and repetitive activities. In this work, we do not use EMG signals. Rather, we assume knowledge of the human intent by knowing the task *a priori*.

*This work was supported by NSF under the award number 1544895, 1928654, 1935337 and 1944453.

[1]Electrical and Computer Engineering Department, Northeastern University, Boston, MA 02115, USA {sharif.mo, deniz, tpadir}@northeastern.edu
[2]Khoury College of Computer Sciences, Northeastern University, Boston, MA 02115, USA c.amato@northeastern.edu

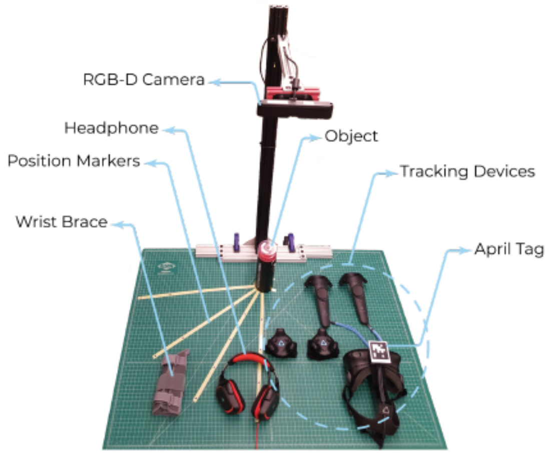


Fig. 1: The experimental setup for collecting real human trajectories for DEXTRON.

RL is a wide-spread method to address robotic grasping and manipulation problems [13], [14]. RL has also been used for robot prosthetic hands control. Pilarski et al. [15] use a continuous actor-critic RL method to learn finger actuation commands from EMG signal inputs. IL is a method that can be used to help RL find optimal solutions in shorter time [16], [17]. Vasan et al. [18] collect EMG signals from the intact hand as the demonstrator and learn a policy using RL for the robot prosthetic hand at the other side. Ficuciello et al. [19] leverage IL to learn a policy for grasping objects in the synergy space and then refine the learned policy using RL. RL and IL (RLIL) were used in [20] to learn an end-to-end policy for robot prosthetic hands where human hand transport was modeled by minimum-jerk trajectories.

In this paper, we are proposing learning an end-to-end policy for below-elbow upper limb robot prosthetic hands based on *real* human hand reaching trajectories. Knowing the human intent *a priori*, we learn a policy that not only gradually molds intuitively to the object shape, but also maintains the required grasp force on the object without relying on EMG. Unlike [20] where simulated hand trajectories were used, we create a stochastic environment with *real* hand reaching trajectories, called DEXTRON (DEXTerity enviRONment), from data we collect from several human subjects. Real trajectories impose several challenges which make the RL and IL techniques used in [20] not readily available. By not relying on the EMG signals, our method can also relax some of the other EMG control side-effects like continuous attention requirement, excessive energy consumption, and fatigue.

Contrary to the classic robot manipulation problem where the robot controls both the end-effector transportation and grasp, in the below-elbow robot prosthesis grasping problem the robot can merely control the fingers (and the wrist in some robots). Accordingly, since human and the robot are collaborating to reach a shared goal [21], e.g. grasping an object, this problem is categorized as human-robot collaboration (HRC), regardless of whether a shared control strategy is used for controlling the wrist and fingers or not. In HRC, both human and the robot can adapt to the other side's policy, i.e. human-robot mutual adaptation [22]. In this paper, we propose a predictive model, called a success model, which predicts the final outcome of the robot in grasping the object. This model, not only provides a way for more transparency for robot decisions through post hoc explanation [23], but also helps human get adapted to the robot quicker by getting more insight and thus trust into the robot control strategy [24], [25]. This model is inspired by [26] in which a similar model is used to predict the final expected outcome for grasping if an action is executed at a given state.

The contributions of this paper are: 1) development of an extension to the RL and IL method proposed in [20] to learn end-to-end grasping policy for robot prosthetic hands with real transport trajectories based on Monte Carlo simulation, 2) development of a stochastic simulation environment, DEXTRON, for robot prosthetic hands in Google DeepMind dm_control [27] which can be used by researchers as a stochastic environment with delayed sparse rewards in order to improve results of the current study or solve new problems regarding robot prosthetic hands, and 3) introduction of a success model to obtain insights into the trained policy and to compare it with the demonstration policy qualitatively. The source code of this paper is available on GitHub [1] under BSD open source license.

II. Methods

A. Problem definition and assumptions

We have a robot prosthetic hand for below-elbow amputation. The wrist is assumed to be fixed. The intent of the human is to grasp the only object in the simulation environment - a bottle always located at the world frame origin. We assume that the robot knows the human intent *a priori*. We want to learn an end-to-end policy which maps the environment states to the finger actuators' velocity commands. Hand transport trajectories are based on real hand transport trajectories collected from human subjects. A 5 degree of freedom robot prosthetic hand with 5 degrees of actuation is modeled in a simulation environment in agreement with common robot prosthetic hands in the market. We assume that all of our velocity-controlled fingers are coupled together, i.e. they all receive the same command, and our action space is $\mathcal{A} = [-1, 1]$, where an action greater than zero indicates a closing command. We further assume we have access to the raw system states $\mathcal{S} \subset \mathbb{R}^{48}$, which includes all joint and link positions and quaternions as well as their time derivatives. As opposed to our previous study [20], we had to change our definition for the reward function in order to accommodate inter-subject variations fairly. After the first timestep the object or the robot hand are above a threshold, a timer starts counting down for 20 timesteps. Then, for every timestep that the object is above the threshold, the agent is rewarded $r = +1$. With this reward function, the maximum sum of rewards an agent can receive in the environment is $+20$, and that happens whenever the agent grasps the object successfully and the object does not slide in the robot hand, showing the force has been maintained on the object. If the robot initially grasps the object successfully but later the object slides in the hand, the sum of rewards will be less than $+20$, and when the agent fails to grasp the object altogether, the sum of rewards would be 0.

For differentiation between the simulated trajectory environment (introduced in [20]) and our new real trajectory environment (introduced in this paper), we call them the simulated-trajectory environment and DEXTRON, respectively. Also, *environment* will refer to DEXTRON unless explicitly indicated otherwise.

B. DEXTRON instantiation

We define a trajectory, λ, as a sequence that indicates pose of the human subject's hand over time, i.e. $\lambda = \{(\mathbf{x}_i, \mathbf{q}_i) | i = 1, ..., T\}$, where $\mathbf{x}_i$ is the 3d position of a frame attached to the subject's hand at time instance i, $\mathbf{q}_i$ is the frame's quaternion at time instance i with respect to the global frame, and T is the total number of timesteps of the trajectory. We define $\delta = [\delta_x, \delta_y, 0]^T$ as the trajectory offset, and we have $\lambda + \delta \triangleq \{(\mathbf{x}_i + \delta, \mathbf{q}_i) | i = 1, ..., T\}$. We define Λ as the set of all collected *real* human hand trajectories. In order to further enrich diversity within the set of trajectories in Λ, we also randomize the duration of each trajectory by scaling its duration to $t_{\text{new}} = t_{\text{old}} \times t_s + t_n$, where t_{new} is the new duration of the trajectory, t_{old} is the old duration of the trajectory, t_s is a constant scaling factor, and t_n is the duration offset. We also resample each new trajectory to obtain an evenly sampled trajectory with a sampling interval of $0.02s$. We use piecewise B-spline interpolation for resampling $\mathbf{x}_i$ and Squad method [28] for resampling $\mathbf{q}_i$.

To instantiate DEXTRON, we sample the environment settings as $\lambda \sim \mathcal{U}(\Lambda)$, $\delta_x \sim \mathcal{U}(\Delta_x)$, $\delta_y \sim \mathcal{U}(\Delta_y)$, and $t_n \sim \mathcal{N}(\mu_t, \sigma_t)$, where $\mathcal{U}(A)$ indicates a uniform distribution over all elements of set A. We call the tuple $(\lambda, \delta_x, \delta_y, t_n)$

[1]https://github.com/sharif1093/dextron

the environment settings. The sets and other environment parameters are provided in Table I.

C. Creating the set of expert policies

In our previous study [20], we used the following two-phase p-controller as the expert policy, which roughly mimicked the human aperture control behavior [29]:

$$\pi_e(h, \hat{d}) = \begin{cases} K(h_{\text{open}} - h) & \hat{d} \geq \hat{d}_c \\ K(h_{\text{closed}} - h) & \hat{d} < \hat{d}_c \end{cases} \tag{1}$$

where h is the hand closure, $\hat{d}$ is the normalized relative hand-object distance, K is the controller gain, $\hat{d}_c$ is the critical normalized relative hand-object distance, and h_{open} and h_{closed} are the reference values for the controller in the first and second phases, respectively. For the simulated-trajectory environment, there existed a single tuple of the policy parameters $(K, \hat{d}_c)$ which resulted in the maximum reward for almost all samples of the environment parameters. However, in DEXTRON, a single controller parameter tuple results in successful grasps in a few samples of the environment.

We define the set of the above expert policies as:

$$\Pi_E = \Big\{\pi_e(h, \hat{d}) \big| \\ K \in [K_{min}, K_{max}] \;\&\; \hat{d}_c \in [\hat{d}_{c,min}, \hat{d}_{c,max}]\Big\} \tag{2}$$

where K_{min}, K_{max}, $\hat{d}_{c,min}$, and $\hat{d}_{c,max}$ are constants given in Table I. We run a Monte Carlo simulation to find a working expert policy $\pi_e \sim \mathcal{U}(\Pi_E)$ for every sample from the environment. For this, both the environment settings and the policy parameters are sampled each time, the policy is rolled out in the simulation environment. If the sum of rewards for that rollout is greater than 1, we store the environment and the policy sampled parameters tuple, $\Big((\lambda, \delta_x, \delta_y, t_n), (K, \hat{d}_c)\Big)$, in G_s, a database of *successful* policy-environment parameters. It is worth noting that for some environment settings, there exist no policies $\pi_e \in \Pi_E$ that result in successful grasps. This is due to two possible reasons. First, Π_E class of policies may not express a policy that works for some environment settings, i.e. the robot maneuvers required for timely grasping of the object without collision is not represented in Π_E. Second, some reaching trajectories collide with the object unconditionally, no matter what policy is being used, due to relative hand/object pose.

D. Learning End-to-end Policy

Here we provide our main method for training the end-to-end environment and some baselines.

1) RLIL: Like our previous study [20], we use Soft-Actor Critic (SAC) [30] method for the RL. As detailed in [20], our problem is an instance of a Keylock MDP with hard-to-explore sparse and delayed rewards. SAC is a maximum entropy reinforcement learning method which offers good exploration for finding the optimal solution. However, using RL alone may result in very high variance and sensitivity to random seeds. Accordingly, we use a method to guide the explorations, hence our use of IL besides RL. Using IL, RL finds rewarding states which otherwise could never be found due to the large state space dimensions. Since SAC is an off-policy method and off-policy methods do not care about the policy used to generate the transitions in a replay buffer, we can simply store the expert policy transitions in the same buffer as the agent. For the details of the method one may refer to [20]. Also, the architecture of value, action-value, and policy networks are the same as [20]. The hyper-parameters are given in Table II. We provide the results of a no-IL case (pure RL) along with other results as a baseline. We call the policy trained using RLIL or RL alone $a = \pi_{RL}(s)$ for future reference.

TABLE I: Controller and environment parameters

Parameter	**Value**
Environment	
Offset X (Δ_x)	$[-0.06, 0.06]$
Offset Y (Δ_y)	$[-0.06, 0.06]$
Time scale (t_s)	2.5
Time noise mean (μ_t)	0.5
Time noise variance (σ_t^2)	0.8^2
Controller	
Controller gain $[K_{min}, K_{max}]$	$[1, 1.5]$
Controller threshold $[\hat{d}_{c,min}, \hat{d}_{c,max}]$	$[0.01, 0.90]$
Hand open target (h_{open})	0
Hand closed target (h_{closed})	0.8

2) Behavior Cloning: As a baseline, we provide the results of Behavior Cloning (BC) which is training a policy $a = \pi_{BC}(s)$ in a supervised manner on a dataset including transition tuples (s, a) from rolling out the expert policies and environment settings sampled from G_s. The policy $\pi_(BC)$ is represented as a Neural Network having the same structure as the policy network of the RL. A scheduled learning rate with Adam optimizer was used for training.

3) Testing Simulated-Trajectory Environment Policy in DEXTRON: As a comparison, we also trained a policy in the simulated-trajectory environment, and tested it in DEXTRON. The policy was trained using the same settings as described in [20].

E. Success Model

In this section, we propose a model that can add to the transparency of the trained RL policy, π_{RL}, by evaluating its selected actions based on actions which are normally selected by the expert policies in G_s in similar similar states. As mentioned in previous sections, we collected a database, G_s, of specialized Markovian policies for each environment instance. Existence of a specialized *Markovian* policy for each sample of a stochastic Environment does not guarantee the existence of a generic Markovian policy that performs as good as the specialized policies, which makes reasoning about the baseline performance of this problem hard. This can happen, for instance, due to partial observability in our stochastic environment with our current system states. Based on this fact, it may not be fair to compare the performance

TABLE II: Hyperparameter selection

Hyperparameter	Value
SAC	
Learning rate	3e-4
Agent replay buffer size	1e6
Demo replay buffer size	1e6
Mini-batch size	32
Polyak coefficient	0.01
Epoch size	1000 frames
Discount factor (γ)	0.99
Warm start	10000 frames
Network hidden size	256
Success Model & Behavior Cloning	
Learning rate	1e-3
Learning rate decay	0.5 every 100 epochs
Optimizer	Adam
Batch size (Success Model)	1024
Batch size (Behavior Cloning)	512

of our policy obtained by RLIL with the specialized expert policies performance, but we can still expect to see valuable information by comparing them collectively with our policy at each timestep. For this purpose, we propose a predictive network $\Omega(s, a)$ which is essentially a classifier that predicts the chance of final terminal success in state s if we take action a. By success, we mean obtaining a sum of rewards of 20 in the episode, i.e. $\sum r = 20$. This model is inspired by Levine et al. [26] in which they proposed a predictive model to take actions at each timestep that are expected to give the best final score for grasping.

We train the success model, $\Omega(s, a)$, in a supervised manner. We create a dataset, Υ, of tuples, (s, a, o), which include the state, action, and final outcome of rolling out a policy in the environment. The final outcome, o, is defined as 1 if we get a sum of rewards of 20 and 0 otherwise. The dataset, Υ, is a union of two smaller datasets, Υ_{EX} and Υ_{RL}, where Υ_{EX} includes transitions from rolling out sampled policy $\pi_e \sim \mathcal{U}(\Pi_E)$ in a sampled environment, and Υ_{RL} includes transitions from trained RL policy roll-outs in an environment setting sampled from G_s. The Υ_{EX} dataset would normally be highly imbalanced leaning towards failure cases, with a ratio of failure cases to success cases of about $50 : 1$. We use oversampling technique to overcome this imbalance, by sampling the policy (and the environment settings) half-of-the-times from G_s, and half-of-the-times from Π_E (and the general set of environment parameters). This ensures a balance between success and failure cases in the dataset. The purpose of including Υ_{RL} in the training dataset is to improve the predictive feature of $\Omega(s, a)$ over π_{RL}. This ensures $\Omega(s, a)$ can explain the behavior of the RL agent with respect to the collective expert policies while providing better predictions on the RL policy itself.

Here, $\Omega(s, a)$ is implemented as a multi-layer perceptron (MLP). We use 3 layers of a fully connected network with ReLU activation functions. The last layer output is of dimension 2 for two classes, success and failure, which can be later fed into a SoftMax for normalization. Cross Entropy loss function is used to calculate the difference between labels and network predictions. The network is then trained using back-propagation and Adams optimization method [31]. During training, 20% of the data points were separated as test data and 80% were set aside for training.

III. Experiments and Simulations

A. Experimental Setup

In order to collect real human hand trajectory data, we set up an experiment with a top-mount Primesense Carmine 1.09 RGB-D camera, an HTC Vive Tracker (2018) with HTC Vive Base Station 1.0[2], a headphone, and a wrist brace (Fig. 1). The HTC Vive Tracker is reported to have sub-millimeter precision and an average accuracy of about 2 mm in normal working conditions [32]. No further filtering was done in addition to the builtin proprietary processing of the tracking data. Subjects wore the headphone, the wrist brace, and an HTC Vive Tracker on their left hand (Fig. 2). The HTC Vive Headset was left on the workspace during the experiment. The workspace was marked with 16 markers evenly distributed on a 4×4 polar grid centered at the object's center in order to specify the start position of the reaching trajectories. Each subject had to perform 4 trials where each trial consisted of 16 reaching actions: start reaching from a marker after hearing a beep sound, grasp the object, lift the object up to a specified threshold, place the object where it was, retract the hand to the position of the next marker, and wait for the next beep sound. The HTC Vive Tracker recorded the pose of the left hand of the subject. The RGB-D camera recorded the point cloud and the RGB image of the experiment. The coordinate frames of the HTC Vive Tracker and the RGB-D camera were registered on to each other by using an April Tag [33] attached to the HTC Vive Headset as a connection intermediate frame known in both coordinate frames. ROS was used as the back-end system. Data were recorded in ROS's Bag file format.

B. Data Collection and Processing

In total, trajectory data from 9 male subjects were recorded. Due to different hand sizes of the subjects, each trajectory file was visually inspected and manually offset so the pose of the human's hand in the depth sensor coordinate frame approximately matched the pose of the simulated robot hand model based on the Tracker's information. Exact matching was not pursued since this was compensated later in the Monte Carlo simulation by randomizing the trajectory offsets, δ_x and δ_y. Each Bag file was annotated manually for the important time instances, onset of the reaching trajectory, θ_1, the instance of hand actually reaching the object just before the grasp, θ_2, object reaching its maximum height in the lifting phase, θ_3, the releasing moment, θ_4, and hand arriving at the rest position again, θ_5. An automated script then segmented each bag file into individual reaching trajectories. Each extracted trajectory included from θ_1 to θ_3 instances. Totally, $9 \times 16 \times 4 = 576$ individual trajectories were obtained. The whole data collection was before the COVID-19 pandemic hence no health protocols were required.

[2]https://www.vive.com/

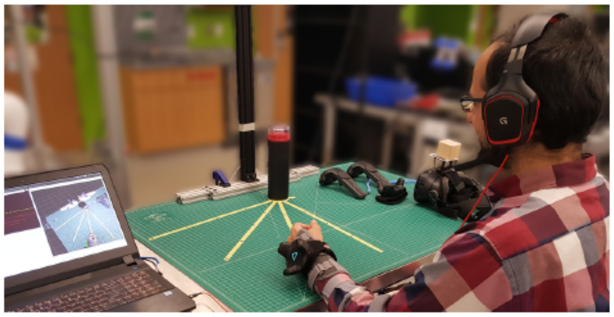


Fig. 2: The experiment setup with a subject.

C. Simulations

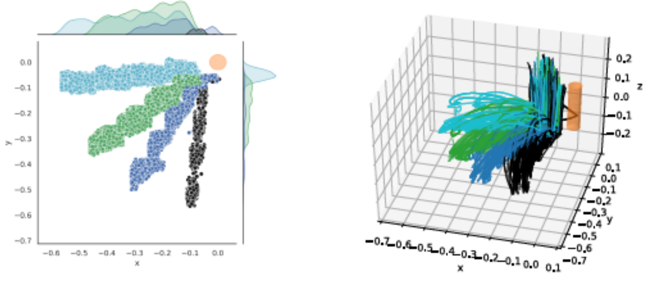
We parallelized the execution of the Monte Carlo simulation using 6 nodes on a computer cluster, with each node running 25 simultaneous threads at the same time. This took about 12 hours for each node to test 3,068,600 episodes in total from which 71798 episodes resulted in a sum of reward greater than 1 (2.3% rate of success).

We used Digideep [34] for its implementation of the SAC method. We utilized PyTorch [35] for the Neural Networks and MuJoCo [36] for physics modeling. The environment of MuJoCo HAPTIX [37] was used as a base for our simulation environments. DeepMind's dm_control was adopted as a Python wrapper for MuJoCo. The model used for the robot hand was built based on the Ottobock BeBionic [38] robot prosthetic hand in Solidworks, then exported to MuJoCo. Training our RL models took about 25 hours on a single NVIDIA P100 GPU. Generating the datasets for training the success model took between 2 to 6 hours using 20 threads simultaneously. The Υ dataset included 30M transitions which were from running 180K episodes and it required 18 GB for storage. The ratio of success labels to total number of transitions was 54%. The dataset for training the BC policy included 10M transitions from running 45K episodes.

IV. Results

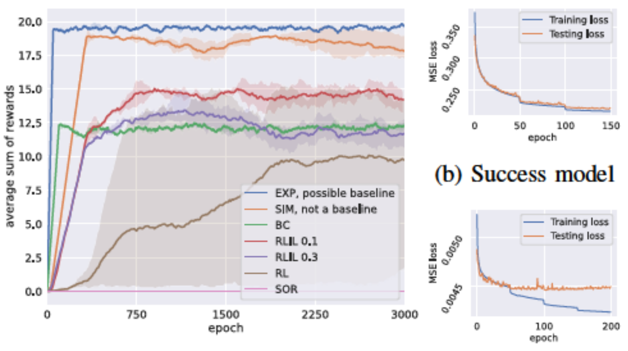
By using the Monte Carlo simulation, we could obtain the distribution over the randomized controller and environment parameters. Fig. 3 shows the distribution of the trajectories in 3d (before MC) and starting points of successful trajectories (after MC). It is worth noting that the region indicated with black trajectories and dots included the least successful grasps, probably due to the narrow angle of hand approach with respect to the object, which results in a collision most of the time.

The sum of rewards for several cases are demonstrated in Fig. 4a. The expert policies (stored in G_s) are shown as "EXP" in the plot. Note that although only $r = 20$ cases were included in the G_s database, the result of replaying those policies in the environment sometimes did not give us a sum of reward of 20. We could not find the source for this discrepancy, but we guess it might be related to some stochasticity in the MuJoCo environment which causes slightly different results based on different environment random seeds. The reproduced results of [20] using the new reward function are



(a) Start locations after MC. (b) Original trajectories before MC.

Fig. 3: Sample successful trajectories of the MC simulation. Object is located at $x = 0$ and $y = 0$.



(a) Baseline comparison (b) Success model (c) Behavior cloning

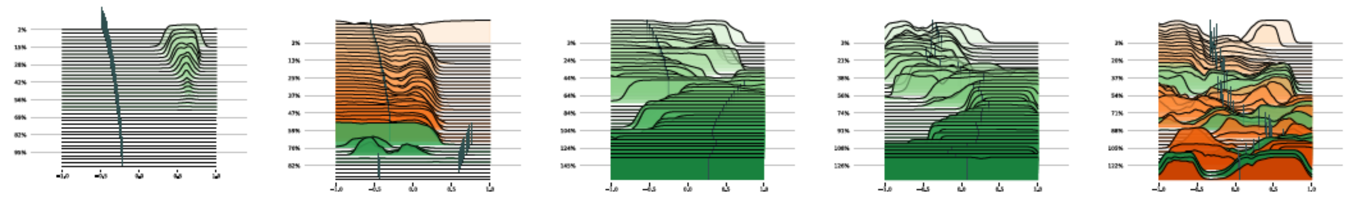
Fig. 4: Learning curves

shown as "SIM". This is obviously not a baseline since it is trained on a different environment (simulated-trajectory environment). The BC sum of reward is shown under "BC". Also, the learning curve of the BC method is shown in Fig. 4c. The effect of Demo-Use Ratio (DUR), i.e. the ratio of demo transitions sampled for training as defined in [20], is shown by "RLIL 0.1" for $DUR = 0.1$ and "RLIL 0.3" for $DUR = 0.3$. The pure RL case is shown as "RL". Rolling out the policy of "SIM" in DEXTRON is shown as "SOR". The results are shown for 3 different random seeds. Every epoch includes 1000 frames.

The success model inference plots are shown in Fig. 5. At each state s, we calculate the value of the function for a discretized set of actions $a \in \{-1, -0.96, ..., 0.96, 1\}$, and draw the success curve for that time instance. The graph shows the rate of final success we expect in state s if we perform action a. Five cases are displayed in Fig. 5.

V. Discussion

In this work, we offered training an end-to-end policy for human-in-the-loop robot grasping via RL. In a previous work [20], we provided an end-to-end policy for a simulated-trajectory environment, while in this work a real-trajectory environment (i.e. DEXTRON) was considered. From the "SOR" plot in Fig.4a, it is noticed that the policy is sensitive to the trajectories it sees during training: the policy trained



(a) Disappearing hope window due to continued wrong choice.

(b) Early closing command caused failure.

(c) Correct prediction for expert policy.

(d) Correct prediction for RL policy.

(e) A case in which the prediction model fails.

Fig. 5: Applying the success model on five sample trajectories. The vertical axis shows the normalized timestep of the trajectory with respect to its time-of-reach. The horizontal axis shows the action. The short lines show the action that was actually chosen by the policy. The 🟩 color shows a true prediction of the final outcome. The 🟥 color shows misclassifications.

in the simulated-trajectory environment does not transfer to DEXTRON successfully.

"RLIL 0.1" provides the best result of all other methods plotted in Fig.4a, where it has the highest average sum of rewards and reasonable variance (w.r.t. "RL" method). The "BC" performed moderately as its policy is only trained on expert data and has no chance of interaction with the environment. Contrary to [20] where pure RL could not learn anything, "RL" learns useful policies in DEXTRON, probably due to stochasticity and higher variance in the environment. Here, we could reach a maximum score of $15/20$ from among all of our different tested methods, i.e. a success rate of 75% in grasping the objects. This is certainly not desirable as an end product; however, it provides a good starting point and baseline for future extensions of this work.

The success model provides a post-hoc explanation about the operation of the RL policy network. This is significant since it provides a way to compare the quality of the RL policy with the set of expert policies collectively. The results of applying the success models on 5 sampled policy roll-outs is demonstrated in Fig. 5. The success model has successfully predicted a failure outcome in Fig. 5a in all timesteps; where a success window (the bumps in the graph) existed in the early stages of the trajectory, but that diminishes as the action is constantly chosen far from that window. In Fig.5b, the final failure outcome is not attributed to the policy's actions in the early stages of the trajectory, rather, it is because of a too early closing command at about 70% of the trajectory which results in a collision with the object and diminishes all chances of recovery quite fast. In Fig. 5c and Fig. 5d the success model results are visualized for two successful outcomes from the expert and the trained RL policy (from "RLIL 0.1"). We see how appropriate choices of actions by policies' have resulted in a successful final outcome. It is important to note though, like any model, the success model can fail to provide a useful explanation (Fig. 5e).

By providing predictions into the future, the success model may also be used in real time to give feedback to the user about the outcomes of being at a specific state. While the success model acts like an action-value function, it is still different in that it is trained on failure cases as well. So, the success model can be thought to provide failure-/ success-awareness for the policy.

As our best method reaches a maximum score in about 750 epochs (Fig.4a), which means 750k frames or 4 hours of experiment, it is feasible to learn the policy directly in real life. One may also use sim2real methods [39], [40] to transfer the policy to real life. The current dataset and environment, considers only the reaching part of the trajectory which limits its use in real life unless we create mechanisms to detect onset of the reaching action. Also, the current approach does not include any releasing. In future works, we plan to extend the current approach to learning end-to-end policies for more complex tasks which may contain several grasps/releases. Including a variety of objects, learning from images to relax the need for raw system states, and learning shared control strategies to use EMG signals along with the system states are a few other suggestions for future works. The last improvement can relax our assumption about knowing human intent *a priori* by providing a shared control framework.

VI. Conclusion

In this work we offer the methodology to train end-to-end policies for human-in-the-loop robot grasping on real reaching trajectories. The fact that human moves the robot makes this problem naturally fall into the Human-Robot Collaboration category. We trained our policy using real human trajectories which were augmented using Monte Carlo simulation. The fact that our method does not rely on EMG makes it suitable for lengthy and repetitive tasks; however, the fact that it has pretty small information about the human intent makes it less readily available for generic applications. We hope that a hybrid method integrating our method with EMG/ENG inference can take advantage of both methods while reducing robustness issues of the EMG-based control methods. Also, by providing DEXTRON, we hope to attract the attention of the RL and HRC communities to the human-in-the-loop robot grasping problem.

Acknowledgment

We would like to thank Eoin Daly for his help with data processing for the real trajectories.

REFERENCES

[1] S. Muceli, N. Jiang, and D. Farina, “Extracting signals robust to electrode number and shift for online simultaneous and proportional myoelectric control by factorization algorithms,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 22, no. 3, pp. 623–633, May 2014.

[2] M. Hakonen, H. Piitulainen, and A. Visala, “Current state of digital signal processing in myoelectric interfaces and related applications,” *Biomed. Signal Process. Control*, vol. 18, pp. 334–359, Apr. 2015.

[3] H.-J. Hwang, J. M. Hahne, and K.-R. Müller, “Real-time robustness evaluation of regression based myoelectric control against arm position change and donning/doffing,” *PLoS One*, vol. 12, no. 11, p. e0186318, Nov. 2017.

[4] A. Chadwell, L. Kenney, S. Thies, A. Galpin, and J. Head, “The reality of myoelectric prostheses: Understanding what makes these devices difficult for some users to control,” *Front. Neurorobot.*, vol. 10, p. 7, Aug. 2016.

[5] S. Došen, C. Cipriani, M. Kostić, M. Controzzi, M. C. Carrozza, and D. B. Popović, “Cognitive vision system for control of dexterous prosthetic hands: Experimental evaluation,” *J. Neuroeng. Rehabil.*, vol. 7, no. 1, p. 42, Aug. 2010.

[6] A. Gigli, A. Gijsberts, V. Gregori, M. Cognolato, M. Atzori, and B. Caputo, “Visual cues to improve myoelectric control of upper limb prostheses,” Aug. 2017.

[7] M. Sharif, D. Erdogmus, and T. Padir, “Particle filters vs hidden markov models for prosthetic robot hand grasp selection,” *International Journal of Robotic Computing*, vol. 1, no. 2, p. 25, Jul. 2019.

[8] K. Z. Zhuang, N. Sommer, V. Mendez, S. Aryan, E. Formento, E. D’Anna, F. Artoni, F. Petrini, G. Granata, G. Cannaviello, W. Raffoul, A. Billard, and S. Micera, “Shared human–robot proportional control of a dexterous myoelectric prosthesis,” *Nature Machine Intelligence*, vol. 1, no. 9, pp. 400–411, Sep. 2019.

[9] M. Santello and J. F. Soechting, “Gradual molding of the hand to object contours,” *J. Neurophysiol.*, vol. 79, no. 3, pp. 1307–1320, Mar. 1998.

[10] L. H. Smith, L. J. Hargrove, B. A. Lock, and T. A. Kuiken, “Determining the optimal window length for pattern recognition-based myoelectric control: balancing the competing effects of classification error and controller delay,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 19, no. 2, pp. 186–192, Apr. 2011.

[11] H. Bouwsema, C. K. van der Sluis, and R. M. Bongers, “Movement characteristics of upper extremity prostheses during basic goal-directed tasks,” *Clin. Biomech.*, vol. 25, no. 6, pp. 523–529, Jul. 2010.

[12] M. Merad, É. de Montalivet, A. Touillet, N. Martinet, A. Roby-Brami, and N. Jarrassé, “Can we achieve intuitive prosthetic elbow control based on healthy upper limb motor strategies?” *Front. Neurorobot.*, vol. 12, p. 1, Feb. 2018.

[13] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine, “SOLAR: Deep structured representations for Model-Based reinforcement learning,” Aug. 2018.

[14] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous Off-Policy updates,” *Robotics and Automation*, Oct. 2016.

[15] P. M. Pilarski, M. R. Dawson, T. Degris, F. Fahimi, J. P. Carey, and R. S. Sutton, “Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning,” *IEEE Int. Conf. Rehabil. Robot.*, vol. 2011, p. 5975338, 2011.

[16] S. Ross, G. J. Gordon, and J. Andrew Bagnell, “A reduction of imitation learning and structured prediction to No-Regret online learning,” Nov. 2010.

[17] Y. Gao, Huazhe, Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, “Reinforcement learning from imperfect demonstrations,” Feb. 2018.

[18] G. Vasan and P. M. Pilarski, “Learning from demonstration: Teaching a myoelectric prosthesis with an intact limb via reinforcement learning,” *IEEE Int. Conf. Rehabil. Robot.*, vol. 2017, pp. 1457–1464, Jul. 2017.

[19] F. Ficuciello, A. Migliozzi, G. Laudante, P. Falco, and B. Siciliano, “Vision-based grasp learning of an anthropomorphic hand-arm system in a synergy-based control framework,” *Science Robotics*, vol. 4, no. 26, p. eaao4900, Jan. 2019.

[20] M. Sharif, D. Erdogmus, C. Amato, and T. Padir, “Towards End-to-End control of a robot prosthetic hand via reinforcement learning,” in *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*, Nov. 2020, pp. 641–647.

[21] B. J. Grosz and S. Kraus, “Collaborative plans for complex group action,” *Artif. Intell.*, vol. 86, no. 2, pp. 269–357, Oct. 1996.

[22] S. Nikolaidis, Y. X. Zhu, D. Hsu, and S. Srinivasa, “Human-Robot mutual adaptation in shared autonomy,” in *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction - HRI ’17*. New York, New York, USA: ACM Press, Jan. 2017, pp. 294–302.

[23] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, “Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,” *Inf. Fusion*, vol. 58, pp. 82–115, Jun. 2020.

[24] N. Amirshirzad, A. Kumru, and E. Oztop, “Human adaptation to Human–Robot shared control,” *IEEE Transactions on Human-Machine Systems*, vol. 49, no. 2, pp. 126–136, Apr. 2019.

[25] M. Lewis, K. Sycara, and P. Walker, “The role of trust in Human-Robot interaction,” in *Foundations of Trusted Autonomy*, H. A. Abbass, J. Scholz, and D. J. Reid, Eds. Cham: Springer International Publishing, 2018, pp. 135–159.

[26] 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,” *The Int’l Journal of Robotics Research*, p. 027836491771031, Mar. 2016.

[27] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller, “DeepMind control suite,” Jan. 2018.

[28] E. B. Dam, M. Koch, and M. Lillholm, *Quaternions, interpolation and animation*. Citeseer, 1998, vol. 2.

[29] B. Hoff and M. A. Arbib, “Models of trajectory formation and temporal interaction of reach and grasp,” *J. Mot. Behav.*, vol. 25, no. 3, pp. 175–192, Sep. 1993.

[30] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft Actor-Critic: Off-Policy maximum entropy deep reinforcement learning with a stochastic actor,” Jan. 2018.

[31] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” Dec. 2014.

[32] M. Borges, A. Symington, B. Coltin, T. Smith, and R. Ventura, “HTC vive: Analysis and accuracy improvement,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 2610–2615.

[33] E. Olson, “AprilTag: A robust and flexible visual fiducial system,” in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 3400–3407.

[34] M. Sharif, “Digideep: A DeepRL pipeline for developers,” 2019.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An imperative style, High-Performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[36] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 5026–5033.

[37] V. Kumar and E. Todorov, “MuJoCo HAPTIX: A virtual reality system for hand manipulation,” in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Nov. 2015, pp. 657–663.

[38] C. Medynski and B. Rattray, “Bebionic prosthetic design.” dukespace.lib.duke.edu, 2011.

[39] F. Sadeghi and S. Levine, “CAD2RL: Real Single-Image flight without a single real image,” Nov. 2016.

[40] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” Mar. 2017.