Learning Object Manipulation with Dexterous Hand-Arm Systems from Human Demonstration

Philipp Ruppel, Jianwei Zhang
{ruppel, zhang}@informatik.uni-hamburg.de

***Abstract*— We present a novel learning and control framework that combines artificial neural networks with online trajectory optimization to learn dexterous manipulation skills from human demonstration and to transfer the learned behaviors to real robots. Humans can perform the demonstrations with their own hands and with real objects. An instrumented glove is used to record motions and tactile data. Our system learns neural control policies that generalize to modified object poses directly from limited amounts of demonstration data. Outputs from the neural policy network are combined at runtime with kinematic and dynamic safety and feasibility constraints as well as a learned regularizer to obtain commands for a real robot through online trajectory optimization. We test our approach on multiple tasks and robots.**

I. Introduction

Humanoid robot hands offer unique opportunities for robotic manipulation by being ideally suited to handle a vast number of objects and tools that were originally designed for human hands and by potentially allowing for simple and intuitive teaching of robots through human demonstration without requiring explicit task-specific programming of motions or task goals. As an additional benefit, humanoid robot hands are modeled after a system that has proven to be extremely versatile and effective for millennia. However, despite many recent advances in robotics and AI, control and learning for dexterous manipulation with humanoid robot hands in the real world remains a significant challenge.

To make teaching simple and convenient, we want to allow humans to demonstrate tasks using their own hands with real objects, instead of having to use teleoperation, kinaesthetic teaching, or additional task-specific programming or annotation. While kinaesthetic teaching would provide robot poses directly and teleoperation could rely on human feedback to correct for errors and to ensure safety, our system has to produce accurate and safe robot commands autonomously. To apply current reinforcement learning techniques to robotic manipulation, human programmers usually have to implement task-specific rewards and simulation environments. Dynamic motion primitives can require explicit data annotation and task-specific programming to adapt and initiate different motion primitives. We do not want to burden human teachers with having to perform these additional steps and instead want our system to learn the required task information directly from demonstration data.

Authors are with the Department of Informatics, University of Hamburg.
This work was partially supported by the German Research Foundation (DFG) and the National Science Foundation of China (NSFC) in project Crossmodal Learning, TRR-169, www.crossmodal-learning.org

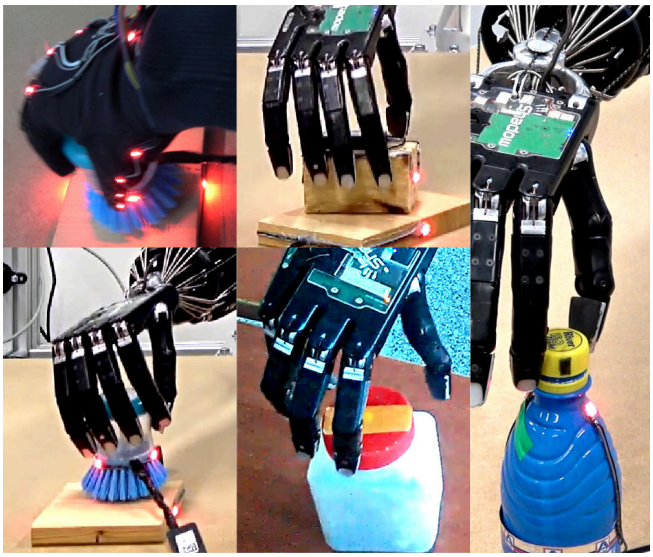


Fig. 1: We learn manipulation tasks from human demonstration (top-left) and execute the learned behaviors on real robots (bottom-left, middle, right).

We first record human demonstrations using an instrumented glove and extract motion trajectories as well as tactile information. To efficiently learn stable control policies that generalize to previously unseen situations, we introduce a set of trajectory-based training and data augmentation methods. We present and compare three different network architectures: a feed-forward policy, a deep recurrent structure that implicitly learns hidden state information, and a model-based approach which first learns neural models and then trains a multimodal policy network to also consider tactile sensations and state variables. At runtime, an online trajectory optimizer uses abstract and hardware-independent commands from the neural network as well as a learned regularizer to generate joint-space commands for a particular robot under kinematic and dynamic safety and feasibility constraints. Optimizing trajectories over multiple time steps allows our controller to avoid kinematic constraints and collisions while it is still possible to do so without violating dynamic limits or further deviating from the motion goals. We also use trajectory optimization for stable hand tracking. See figure 2 for an overview of our system. We test and demonstrate our methods on four different manipulation tasks and on three different robots.

II. Related Work

Inverse kinematics can find joint angles for a robot arm to reach a Cartesian goal pose, inverse dynamics computes joint velocities or torques from Cartesian goal velocities or forces [1] [2]. It can be desirable to optimize trajectories over multiple time steps simultaneously to fulfill kinematic as well as dynamic objectives and constraints. This can be accomplished using stochastic [3] or gradient-based [4] methods. Schulman et al. [5] generate collision-free trajectories using penalty terms. Mordatch et al. [6] use a special contact model to generate trajectories for simulated manipulation problems. Trajectory optimization for robots with many degrees of freedom is typically performed offline due to high computation time. However, for certain general classes of constrained optimization problems, it has been shown that interior-point methods can find solutions efficiently [7] [8] [9] [10].

Reinforcement learning adjusts a policy to maximize a reward function through trial and error. For practical problems with sparse rewards, the required numbers of trials can be prohibitively large. It is often possible to accelerate reinforcement learning by programming smooth task-specific shaped reward functions. Marcin et al. use reinforcement learning in simulated environments and a special reward function to find finger motions for rotating a cube [11]. Akkaya et al. [12] and Li et al. [13] also learn motions for rotating the top-facing side of a Rubik's Cube and combine the learned behaviors with a traditional Rubik's Cube solver. Rajeswaran et al. teleoperate a virtual robot hand in a simulated environment. They use reinforcement learning to train a policy network to imitate the demonstrated motions using the same simulated robot hand and to maximize additional task-specific reward functions [14]. Abbeel et al. learn cost functions from human demonstration for controlling a car in a simplified driving simulator [15]. If we would use current reinforcement learning techniques for our work, human teachers would not only have to perform demonstrations, but would also have to prepare task-specific simulation environments with accurate object models.

Ijspeert et al. [16] capture human motions and represent the recorded joint-space trajectories using basis functions and an attractor term. The attractor acts as a low-pass filter and can be tuned for smooth or accurate control. Users have to choose between repetitive and non-repetitive basis functions, and for repetitive motions, annotate phases. Repetitive motions are executed indefinitely and have to be stopped by the user or by a program. For a single goal position, the entire motion primitive can be shifted by a fixed offset. Paraschos et al. [17] learn probability distributions from sets of joint-space trajectories to shape the transitions between motion primitives. Automatically adapting to multiple object poses, achieving rotational invariance, combining repetitive and non-repetitive motions, generating accurate as well as smooth and feasile motions, integrating additional sensor modalities, etc., would require further extensions. It is not always clear how these features could be integrated without task-specific programming or annotation.

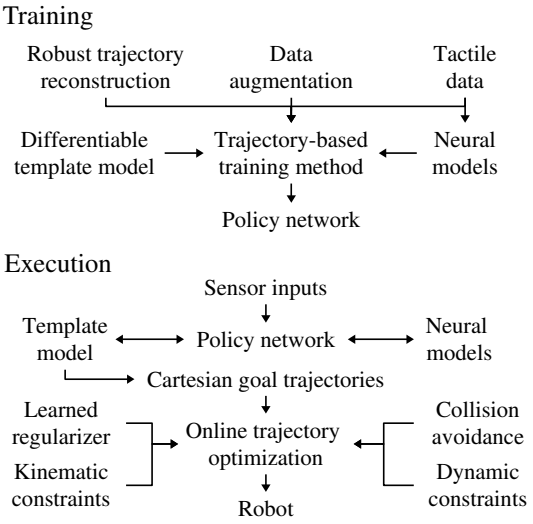


Fig. 2: Overview of our training methods (a) and of our system during execution (b).

Several teleoperation systems have been developed for controlling humanoid robot hands. These methods can rely on the human operator to ensure safety and to compensate for position offsets. Recent successful approaches mainly use relative objectives between fingertips [18] or treat the hand and the arm separately [19] [20]. For autonomous execution, our controller has to not only reproduce relative motions, but it also has to achieve accurate absolute positioning and enforce safety and feasibility constraints.

Convolutional neural networks [21] [22] use special connection patterns and weight sharing to achieve translation invariance. Weight sharing can also be used to process unordered point sets [23].

III. Data Acquisition and Trajectory Reconstruction

During each demonstration, we record finger and object motions as well as tactile sensations. Finger motions and tactile information are captured using an instrumented glove. We construct tactile sensors from conductive fabric and pressure-sensitive piezo-resistive materials. One sensor is attached to each fingertip. Human motions are recorded using LEDs on the fingertips and hand joints, and the line cameras of a Phasespace Impulse X2 system.

To achieve stable hand tracking without manual data cleanup, we introduce a trajectory-based reconstruction method. Each line sensor is modeled as a one-dimensional pinhole camera with position P_C, orientation matrix R_C, and polynomial distortion terms d_i. We optimize 3D marker positions p_j to minimize a robust loss l_j [24] with reprojection error e_j for observation o_j. During calibration, we also optimize camera parameters.

$$q_j = R_C^{-1} \cdot (p_j - P_C) \tag{1}$$

$$s_j = \frac{q_{j,1}}{q_{j,3}} \qquad e_j = s_j + \sum_{i=1} d_i s_j^{2i} - o_j \tag{2}$$

$$l_j = \begin{cases} \frac{1}{2}e_j^2 & \text{if } \|e_j\| < \delta \\ \delta(\|e_j\| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \quad (3)$$

To fill in data gaps caused by marker occlusions and to exploit observations from other time frames as additional evidence for the correct 3D marker positions, we add a dynamic regularization term $d_{i,t}$ with regularization weights w_f, w_g between marker positions $p_{i,t}$.

$$d_{i,t} = |p_{i,t} - p_{i,t+dt}|^2 w_f + |p_{i,t-dt} - 2p_{i,t} + p_{i,t+dt}|^2 w_g \quad (4)$$

To exploit the kinematic structure of the hand, we add an additional link regularizer $l_{i,j,t}$ with weight w_l between directly connected hand markers $m_{i,t}, m_{j,t}$. We use soft objective terms instead of hard constraints since the glove can slightly move and stretch.

$$l_{i,j,t} = |m_{i,t} - m_{j,t} - m_{i,t+dt} + m_{j,t+dt}|^2 w_l \quad (5)$$

To accelerate convergence, we solve a sequence of equations with exponentially increasing temporal resolution. Each previous solution is used as an initial guess for the next subdivision step.

IV. Network Architectures

After recording human demonstrations and reconstructing 3D trajectories, the obtained data is used to train a neural policy. We substitute objects and the robot with simplified but differentiable Cartesian template models. This frees the network from having to learn hardware-specific details about a particular robot, and it allows us to focus our machine learning efforts on task information. It also allows us to train our policies directly using efficient gradient-based optimization. Both the objects and the hand are represented as point sets. Hand points $p_{t,j}$ can be controlled by the network through velocity commands $v_{t,j}$.

$$\frac{dp_{t,j}}{dt} = v_{t,j} \quad (6)$$

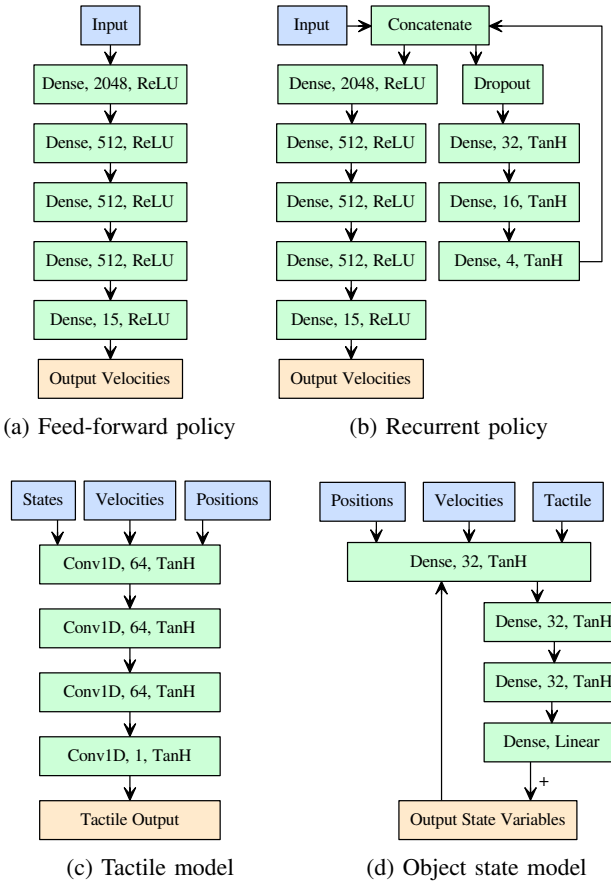
We provide relative position vectors of the hand and object points, velocities v_t, and if available additional state variables s_t and tactile measurements h_t as inputs I_t. Relative position vectors are obtained by subtracting the arithmetic mean of the N hand points $p_{t,j}$.

$$I_t = (p_t - \sum_{j=0}^{N} \frac{p_{t,j}}{N},\ v_t,\ s_t,\ h_t) \quad (7)$$

As in convolutional neural networks, translational invariance is ensured by the network architecture and rotational invariance is achieved through data augmentation.

A. Feed-Forward Policy Network

For simple tasks such as reach-to-grasp, which neither require memory nor tactile perception, a simple unimodal feed-forward policy should be sufficient. Our feed-forward network shown in figure 3a consists of 5 densely connected layers with 2048 input neurons, 512 neurons in each hidden layer, and 15 output neurons. We use ReLU activation for input and hidden layers and linear activation for the output layer. Each output neuron controls the velocity of a hand point along one Cartesian dimension.



(a) Feed-forward policy (b) Recurrent policy

(c) Tactile model (d) Object state model

Fig. 3: Neural policy and model networks.

B. Recurrent Policy Network

We enable the network to remember previous actions and observations by adding recurrent connections as shown in figure 3b. Inputs are shared with a recurrent branch, which consists of three densely connected layers. The outputs of the recurrent branch are concatenated to the inputs of the recurrent and of the feed-forward branch. We use significantly smaller layer sizes for the recurrent branch, with 32 TanH input units, 16 TanH hidden neurons, and four linear output neurons, and we apply 10% dropout at the inputs.

C. Neural Object Models

We extend our template models to also simulate tactile sensations and object state variables. To keep the training process simple for human teachers, we learn these models directly from demonstration data.

1) Tactile Object Model: Our tactile models map relative fingertip positions, fingertip velocities and object state variables to simulated tactile readings. We represent the tactile models using PointNet-inspired [23] fully convolutional neural networks with 1D convolutions over tactile sensor indices and a filter width of 1. The architecture consists of 4 layers with 64 convolutional TanH units in the input and hidden layers and one linear convolutional unit in the output layer.

2) Object State Model: During human demonstration, we measure and record additional object state variables. At runtime, the state variables are predicted. Our neural object state model takes relative Cartesian fingertip positions and velocities, current values for the state variables and tactile information as inputs. The outputs of the object state model are added to the current values of the state variables. The network consists of 4 densely connected layers with 32 TanH units in each input and hidden layer and one linear output unit for each state variable.

V. Trajectory-Based Training

We train our policy networks by simulating trajectories over multiple time steps and propagating gradients back in time. During each simulation step, the policy network P is called with inputs from a previous time step S_t. The velocity outputs of the policy network P are used together with our differentiable template model M and learned models L to compute new values S_{t+dt} for the state variables.

$$S_{t+dt} = \begin{pmatrix} M(S_t, P(S_t)) \\ L(S_t, P(S_t)) \end{pmatrix} \tag{8}$$

State variables $S_{t,m}$ are reset before the first simulation step t_0 and at randomly selected time frames with demonstration data $D_{t,m}$ and random perturbations. The reset probability is computed using a constant base r, a random exponent x and a uniformly distributed random number generator r_t. The random perturbations are composed of a normally distributed random vector $P_{t,m}$ for each point m and a scalar random exponent f for each trajectory segment. We introduce exponential terms to randomly scale the perturbations across multiple orders of magnitude to avoid having to manually fine-tune augmentation parameters. Rotational invariance is achieved through additional online data augmentation, multiplying each demonstration trajectory with a random rotation matrix R.

$$S_{t,m} = \begin{cases} R\ D_{t,m} + b^f\ P_{t,m} & \text{if } (t = t_0) \vee (r_t < b^x) \\ S'_{t-1,m} & \text{otherwise.} \end{cases} \tag{9}$$

We compute a loss value from simulated and demonstrated states over the entire simulated trajectory, propagate the gradients back in time until reaching the start of the trajectory or one of the random resets, and update the network weights for all contributing time steps. The loss function computes a weighted error over different modalities including Cartesian positions and velocities, and if available, tactile information and object state variables. The network weights are optimized via a batch gradient descent method [25]. A relatively large batch size between 128 and 512 should be used to obtain meaningful gradients despite strong randomization and to allow for efficient parallelization.

VI. Time Discretization

Even when using a feed-forward policy, our trajectory-based training method leads to a recurrent structure. During our experiments, we found that it is usually sufficient to train with relatively large time steps and that doing so reduces training time. However, at runtime, we want to use smaller time steps to allow for fast reaction to sensor input and to achieve smooth as well as accurate control. Therefore, we want to reformulate our networks as differential equations and use numerical integration with different step sizes for training and execution. Our continuous-time network N' computes network outputs o_t and the time derivatives of the network activations from current activations A and additional inputs I.

$$(\frac{dA}{dt}, o_t) = N'(A, I) \tag{10}$$

A practical obstacle to using this approach is that in current high-performance software libraries for implementing artificial neural networks, the network N effectively performs numerical integration with a fixed step size s.

$$(A_{t+s}, o_{t1}) = N(A_t, I_t) \tag{11}$$

For an explicit Euler step, finite differences could directly recover the exact gradients within numerical precision. In practice, the activations may be updated incrementally and we obtain a gradient approximation.

$$\lim_{s \to 0} \frac{dA}{dt} = \frac{N(A_t, I_t)_0 - A_t}{s} \tag{12}$$

The gradients can now be integrated with modified step sizes.

VII. Robot Vision

To allow the robot to manipulate unmodified objects, we train a fully convolutional neural network to detect virtual keypoints. We use pre-trained Mobilenet [22] layers up to the fifth separable convolutional block to compute feature embeddings and then add two 32-channel 1x1 convolutional hidden layers and a 1x1 linear convolutional output layer with one channel for each marker ID. The output is resampled to the size of the original input image using bicubic interpolation and maxima in the marker channels are interpreted as virtual marker detections.

Camera poses are calibrated using structure from motion. We attach multiple Aruco [26] tags to the forearm of the robot and automatically move the arm into randomly generated poses while recording marker detections. Since the surface of the robot and the markers is curved, we use corner-based subpixel refinement. To calibrate the cameras, we simultaneously optimize camera parameters and the 3D positions of the marker corners relative to the forearm link to minimize the reprojection error of each corner.

VIII. Online Trajectory Optimization

We translate Cartesian commands from the neural policy network into hardware-specific joint angles through kinodynamic online trajectory optimization. We first simulate Cartesian trajectories using the most recent measurements, the policy network, and our template and neural models. The resulting Cartesian trajectories are converted into sets of timestamped position goals, which are combined with

additional goals and constraints to optimize robot trajectories. Each optimization step is initialized with a timeshifted version of a previous trajectory.

For each trajectory update, we solve a non-linear optimization problem through sequential quadratic programming using a primal-dual interior-point method. The optimization problem is defined by instances of different goal classes. Each goal can specify quadratic objectives, equality constraints, inequality constraints, and box constraints. Inequality constraints are automatically converted into box constraints and equality constraints by inserting slack variables. We finally solve a sequence of unconstrained linear equations with objective gradients J_X, equality and inequality constraint gradients J_E and J_I, exponentially adjusted logarithmic barrier gradients B_X, B_S, and right-hand-side vectors r_x, r_e, r_i for the joint variables X, Lagrange multipliers L_E, L_I and slack variables S_B.

$$\begin{bmatrix} J_X^T J_X + B_X I & J_E & J_I & 0 \\ J_E & 0 & 0 & 0 \\ J_I & 0 & 0 & -I \\ 0 & 0 & -I & B_S I \end{bmatrix} \begin{bmatrix} X \\ L_E \\ L_I \\ S_B \end{bmatrix} = \begin{bmatrix} r_x \\ r_e \\ r_i \\ 0 \end{bmatrix} \quad (13)$$

Cartesian trajectories are translated into quadratic position goals. For each template model point $p_{i,t}$ with time t and point index i, we assign a corresponding reference point r_i relative to a link pose $L_{i,t}$ and minimize the squared distance $d_{i,t}$ between both point positions.

$$d_{i,t} = \|p_{i,t} - L_{i,t}\, r_i\|^2 \quad (14)$$

For each joint position variable $q_{j,t}$ with time t, step size dt and joint index j, we specify upper and lower joint position limits u_j, l_j, a fixed trust region c relative to the last candidate solution $r_{j,t}$, as well as maximum joint velocities v_j and maximum joint accelerations a_j.

$$max(r_{j,t} - c, l_j) < q_{j,t} < min(u_j, r_{j,t} + c) \quad (15)$$

$$-v_j < \frac{q_{j,t+dt} - q_{j,t}}{dt} < v_j \quad (16)$$

$$-a_j < \frac{q_{j,t-dt} + q_{j,t+dt} - 2q_{j,t}}{2dt} < a_j \quad (17)$$

To prevent jumps during trajectory replacement, we constrain the first two keyframes of each new trajectory to match the corresponding two keyframes of the previous trajectory. Mechanical couplings between finger joints on underactuated hands are modeled as additional equality constraints.

For collision avoidance, we construct a convex polyhedral approximation of the workspace in Cartesian space and approximate the shape of each link by a convex hull around a set of spheres. Since the workspace approximation is convex, constraining only the spheres is sufficient to prevent collisions with the entire link bodies. We insert pairwise linear constraints between boundary planes and link spheres. Each boundary plane is represented by a normal n_k and a distance d_k. Each sphere has a center c_l relative to a link pose P_l and a radius r_l.

$$P_l c_l \cdot n_k < d_k - r_l \quad (18)$$

TABLE I: Robot experiments for different tasks, robots, networks architectures, demonstration counts (D.) and trajectory optimization windows (Traj.). For each experiment, we test whether the task is performed successfully during multiple consecutive trials (Succ.) and for different object poses (Inv.).

Task	Robot	Network	D.	Traj.	Succ.	Inv.
Pick Place	C5 UR10e	Feed-Fwd.	10	10	Yes	Yes
Wiping	C5 UR10e	Feed-Fwd.	5	10	Yes	Yes
C. Bottle	C5 LBR4+	Feed-Fwd.	1	10	No	n/a
C. Bottle	C5 LBR4+	Recurrent	1	10	Yes	Yes
C. Bottle	C6 UR10	Model-B.	1	10	Yes	Yes
B. Bottle	C5 UR10e	Feed-Fwd.	1	3, 4	No	n/a
B. Bottle	C5 UR10e	Feed-Fwd.	1	5..10	Yes	Yes

If multiple solutions can be found which fulfill the objective function almost equally well without violating any of the constraints, we want to prefer natural hand poses that would also be preferred by a human. We therefore introduce a learned regularizer.

$$r_i = \left\| \frac{v_i - m_i}{s_i} \right\|^2 \quad (19)$$

From an existing hand pose dataset [27] [28], we compute averages and standard deviations for the joint angles and construct a multivariate Gaussian distribution. For each Gaussian with mean m_i, standard deviation s_i, and corresponding joint variable v_i, we add a quadratic regularization term r_i.

IX. Experiments

We test our methods on three different manipulation problems: a pick-place and a wiping task, opening a chemical bottle with a wide lid, and opening a beverage bottle with a small lid. The experiments are performed with real objects and robots. We use a UR10e arm with a Shadow C5 hand, a KUKA LBR 4+ arm with a Shadow C5 hand, and a UR10 arm with a Shadow C6 hand. An overview of our robot experiments is given in table I.

A. Pick-and-Place Task

The robot has to grasp an elongated box-shaped object and place it onto a rectangular plate. Both objects are equipped with LEDs as tracking markers. We collect a total of 10 human demonstrations. Before each demonstration, both items are moved into different positions and orientations. During the demonstrations, a human grasps the box and places it onto the plate. We use the recorded trajectories to train our feed-forward network. As training data, we use the positions of two markers on each object, one marker on each fingertip, and one marker on each knuckle and at the base of the thumb. At runtime, we use observed marker positions as inputs and pass outputs from the network to our trajectory optimizer. The resulting motions are executed on a UR10e arm with a Shadow C5 hand. The robot is able to successfully perform the task even if the box, the plate, and the hand are placed in previously unseen poses. Grasp poses are adapted if the box is rotated. The lengths of the

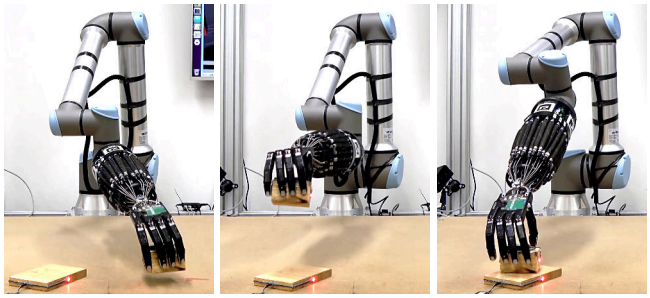


Fig. 4: UR10e arm with Shadow C5 hand while performing a pick-and-place task.

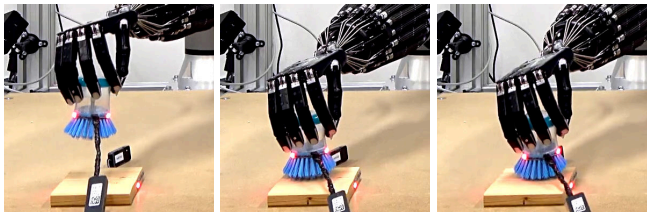


Fig. 5: UR10e with Shadow C5 hand during a wiping task.

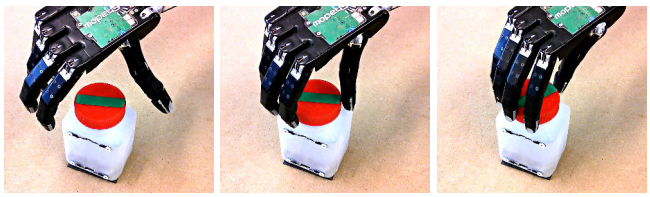


Fig. 6: Turning the lid of a chemical bottle (feed-forward network, LBR4+ arm, C5 hand).

trajectories are adjusted if the object positions are changed. Figure 4 shows the robot during execution.

B. Wiping Task

We record five demonstrations of a wiping task that requires grasping a brush, moving to a target object, and performing oscillating cleaning motions. As for the pick-and-place experiment, we use the feed-forward architecture and a UR10e arm with a Shadow C5 hand. At runtime, the robot approaches and grasps the brush, lifts it, places it onto the target object, and performs periodic cleaning motions, with the bristles of the brush wiping across the surface. The task can be performed successfully for previously unseen hand, brush and target poses. Figure 5 shows the robot during the wiping task.

C. Opening a Chemical Bottle

The robot has to turn the lid of a chemical bottle until it has been loosened, grasp the lid, lift it, and place it next to the bottle. We record a single demonstration with tactile readings and Cartesian motion trajectories for the fingertips and bottle position.

1) Feed-Forward Policy: We train our feed-forward architecture with the recorded trajectories and use a KUKA

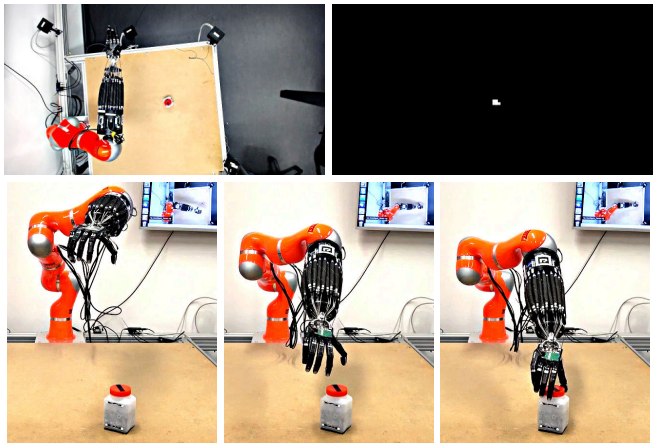


Fig. 7: Opening the chemical bottle using our recurrent policy network (bottom), image from an overhead camera (top-left), output of our vision network (top-right).

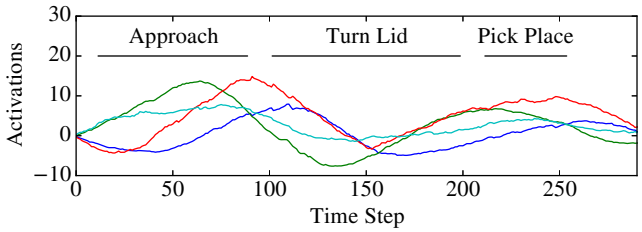


Fig. 8: Recurrent neural activations while opening the chemical bottle, with approximate sub-task annotations.

LBR 4+ with a Shadow C5 hand for execution. For a first test, we assume a fixed bottle pose. If the bottle is carefully placed in the correct position, the robot performs a correct approach motion, and the finger motions turn the lid (see figure 6). Since the network does not possess memory and can neither use recurrent models nor tactile information, it is not able to determine when the lid can be lifted off and continues to perform turning motions indefinitely.

2) Vision Network: We use our vision network described in section VII to automatically determine the bottle position without needing LED markers. After performing SfM-based calibration, our marker-less tracking method delivers results that are accurate enough for approaching the bottle and turning the lid. The object can still be detected and manipulated if it is placed in different positions and orientations on the table.

3) Recurrent Policy Network: We use the same data as before to train our recurrent policy network described in section IV-B for the bottle opening task. While the feed-forward network keeps performing turning motions indefinitely and fails to remove the lid, our recurrent network stops rotating the lid at an appropriate time. It then grasps the lid, lifts it, performs a sideways motion, lowers the hand, and places the lid next to the bottle. We execute the policy on an LBR 4+ arm with a C5 hand. Figure 8 shows the neural activations in the output layer of the recurrent column over time. If we dampen the connections between the last layer in the recurrent column and the concatenation layer,

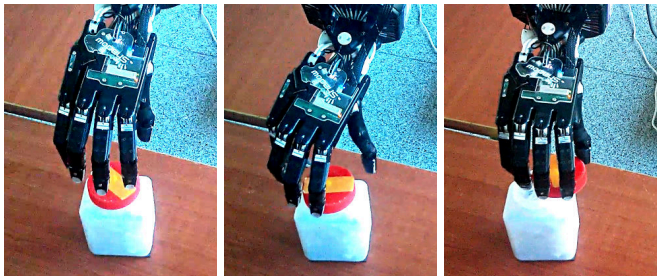


Fig. 9: Opening a chemical bottle and removing the lid (model-based learning, UR10, C6 hand).

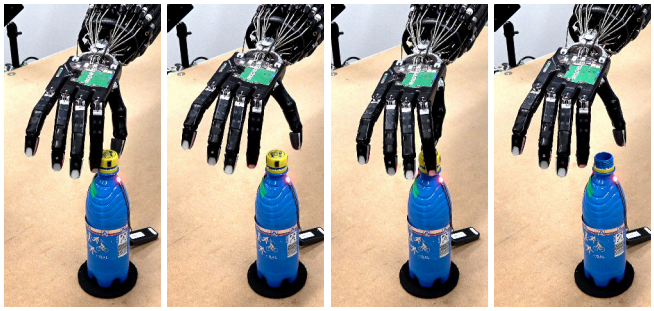


Fig. 10: Turning and removing the lid of a beverage bottle (feed-forward network, UR10e, C5 hand).

the transition from the lid-rotation phase to the pick-place phase is delayed. The overall behavior of the network and the speed of the finger motions remain the same. See figure 7 for photos of the robot during execution as well as an input image and output activations of the vision network.

4) Crossmodal Model-Based Learning: We use tactile data collected during demonstration of the bottle opening task to train a tactile object model as described in section IV-C.1. We also train a recurrent object state model as described in section IV-C.2 with lid orientation as a state variable. Using our model networks, we then train a feed-forward policy network as described in sections IV-A and V. During execution, we use predicted object state information from the object state model and a mixture of predicted and measured tactile readings. A 50-50 combination leads to stable yet responsive behavior. We test the policy on a UR10 arm with a Shadow C6 hand. Each fingertip is equipped with a tactile pressure sensor. If a human touches multiple robot fingertips, the robot hand opens, and after removing the externally induced stimulus, the robot hand closes again until the fingers touch the lid. Our crossmodal model-based architecture was able to perform the bottle opening task successfully in 10 out of 10 trials.

D. Opening a Beverage Bottle

The feed-forward architecture is trained to open a beverage bottle with a smaller lid. We use a single demonstration with trajectories of 21 hand markers at the fingertips and joints, and two markers on the bottle. At runtime, the bottle markers are located using the tracking system, and the generated motions are executed on a UR10e arm with a Shadow C5 hand. See figure 10 for different states during execution. The robot is able to successfully turn the lid. After the lid has been screwed off, it falls onto the table. While the chemical bottle requires a recurrent structure to initiate a final pick-and-place phase, the beverage bottle task can be considered successfully solved by the simpler feed-forward architecture. If we set the window size of the trajectory optimizer to 3, the fingers push the bottle instead of turning the lid. With a trajectory length of 5 or above, the lid is turned successfully.

TABLE II: Average tracking errors for different trajectory lengths while following Cartesian goal trajectories generated by our recurrent network for opening the chemical bottle.

Trajectory Length	3	4	5	7	10
MSE	0.0014	0.0005	0.0003	0.0003	0.0002

E. Trajectory Optimization

Table II shows mean squared tracking errors for different trajectory lengths while opening the chemical bottle. The first two time frames are constrained to match a previous trajectory to allow for smooth trajectory replacement. For each time step, the non-linear problem is solved to convergence. Optimizing only a single new robot pose or very short trajectories leads to high tracking errors. The errors quickly decrease if the trajectory length is increased.

F. Training Time

The neural networks are trained using Tensorflow [29] on an NVIDIA GTX 1080. While the feed-forward network and the model-based approach can learn successful manipulation policies in about 30 minutes, the recurrent network requires approximately three hours of training.

X. Implementation

The components of our system are implemented as ROS [30] nodes and libraries. For neural networks, we use tensorflow [29], Python [31], and Keras [32]. The trajectory optimizer, calibration tools, and the trajectory reconstruction method are implemented in C++ using Eigen [33] for linear algebra. Robot models and states are exchanged as MoveIt [34] objects. For execution, we used roscontrol [35], FRI [36], ur_modern_driver [37], ur_robot_driver, the etherCAT interface of the C6 hand, and a custom driver for the C5 hand.

XI. Conclusion and Future Work

We introduced a novel learning and control framework that allows human teachers to train humanoid robotic manipulators by demonstrating tasks using their own hands with real objects. We successfully tested our approach on multiple tasks and robots.

Three neural network architectures were presented. A feed-forward policy network was able to successfully learn a pick-place, a cleaning, and a bottle-opening task. A different bottle-opening task could not be finished by the feed-forward network. Our recurrent networks completed

the bottle opening task by learning to automatically transition from a periodic turning motion to a final pick-and-place motion. Our trajectory-based training and data augmentation methods allow the system to learn stable neural policies that can automatically adapt to modified object poses from limited amounts of data. As demonstrated by the pick-place and the wiping task, our system can not only produce approach motions but also learn to automatically generate trajectories between objects.

We found that it is possible to learn local object models which are sufficiently accurate for model-based policy optimization directly from demonstration data. In contrast to previous work based on reinforcement learning, our method does not require the user to program task-specific reward functions or simulation environments. By substituting the robot with simplified but differentiable template models, we were able to use efficient gradient-based training, and we could focus our machine learning efforts on task information.

Our trajectory optimizer is fast enough for online control of hand-arm systems with many degrees of freedom. If only a single robot state is optimized at a time, as in inverse kinematics, tracking errors increase and the robot consistently fails during a bottle opening task. We also use trajectory optimization to achieve stable hand tracking. At runtime, unmodified objects can be manipulated via learned keypoints. To prefer natural hand poses, we introduced a learned regularizer.

While our policy networks already accept point lists as input, we are currently using only small numbers of points from the motion tracking system or from neural keypoint detectors. In future work, we want to use point clouds from depth cameras or raw color images. Tactile perception on our instrumented gloves could be improved with high-resolution matrix sensors and we would like to further investigate methods for using tactile information. It would also be interesting to test our system on a larger number of tasks. We plan to further improve our software and to develop it into a set of public open-source packages.

References

[1] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," in *Proc. IEEE RAS Humanoids Conference*, Seoul, Korea, Nov. 2015.

[2] R. Smits, "KDL: Kinematics and Dynamics Library." [Online]. Available: http://www.orocos.org/kdl

[3] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE International Conference on Robotics and Automation*, 2011, pp. 4569–4574.

[4] M. Zucker *et al.*, "CHOMP: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, pp. 1164–1193, Aug. 2013.

[5] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, pp. 1251–1270, Aug. 2014.

[6] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Proc. Eurographics conference on Computer Animation*, July 2012, pp. 137–144.

[7] R. Frisch, "The multiplex method for linear programming," *The Indian Journal of Statistics*, pp. 329–362, Sept. 1957.

[8] D. F. Shanno, "Who invented the interior-point method?" *Documenta Mathematica, Extra Volume: Optimization Stories*, 2012.

[9] A. V. Fiacco and G. P. McCormick, *Nonlinear programming: Sequential unconstrained minimization techniques*. Society for Industrial and Applied Mathematics, Jan. 1968.

[10] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, p. 373–395, Dec. 1984.

[11] OpenAI *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, Aug. 2018.

[12] ——, "Solving rubik's cube with a robot hand," Oct. 2019.

[13] T. Li *et al.*, "Learning to solve a rubik's cube with a dexterous hand," in *Proc. IEEE International Conference on Robotics and Biomimetics*, Dec. 2019.

[14] A. Rajeswaran*, V. Kumar*, *et al.*, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Proc. Robotics: Science and Systems (RSS)*, June 2018.

[15] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004*, Sept. 2004.

[16] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Proc. Advances in Neural Information Processing Systems*, Jan. 2002, pp. 1523–1530.

[17] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Proc. Advances in Neural Information Processing Systems*, Jan. 2013.

[18] A. Handa *et al.*, "Dexpilot: Vision based teleoperation of dexterous robotic hand-arm system," in *IEEE International Conference on Robotics and Automation*, 2020.

[19] S. Li *et al.*, "Vision-based teleoperation of shadow dexterous hand using end-to-end deep neural network," in *Proc. IEEE International Conference on Robotics and Automation*, 2019.

[20] ——, "A mobile robot hand-arm teleoperation system by vision and IMU," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, in press.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278 – 2324, Dec. 1998.

[22] A. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv*, Apr. 2017.

[23] R. Charles, H. Su, M. Kaichun, and L. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, July 2017, pp. 77–85.

[24] P. J. Huber, "Robust estimation of a location parameter," *Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, Mar. 1964.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. International Conference for Learning Representations*, Dec. 2014.

[26] F. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and Vision Computing*, vol. 76, June 2018.

[27] A. Bernardino, M. Henriques, N. Hendrich, and J. Zhang, "Precision grasp synergies for dexterous robotic hands," in *Proc. IEEE International Conference on Robotics and Biomimetics*, Dec. 2013, pp. 62–67.

[28] N. Hendrich and A. Bernardino, "Affordance-based grasp planning for anthropomorphic hands from human demonstration," in *Proc. ROBOT2013: First Iberian Robotics Conference*, 2014, pp. 687–701.

[29] M. Abadi, A. Agarwal, P. Barham, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: http://tensorflow.org/

[30] M. Quigley *et al.*, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[31] G. van Rossum, "Python tutorial," Centrum voor Wiskunde en Informatica (CWI), Amsterdam, Tech. Rep. CS-R9526, May 1995.

[32] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: https://keras.io

[33] G. Guennebaud *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[34] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *Journal of Software Engineering for Robotics*, Apr. 2014.

[35] S. Chitta *et al.*, "ros_control: A generic and simple control framework for ROS," *The Journal of Open Source Software*, Dec. 2017.

[36] G. Schreiber, A. Stemmer, and R. Bischoff, "The fast research interface for the kuka lightweight robot," in *IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications (ICRA 2010)*, May 2010.

[37] T. Andersen, *Optimizing the Universal Robots ROS driver*. Technical University of Denmark, Department of Electrical Engineering, 2015.