Machine Learning for Robotic Manipulation

Quan Vuong
UCSD
qvuong@ucsd.edu

ABSTRACT
The past decade has witnessed the tremendous successes of machine learning techniques in the supervised learning paradigm, where there is a clear demarcation between training and testing. In the supervised learning paradigm, learning is inherently passive, seeking to distill human-provided supervision in large-scale datasets into high capacity models. Following these successes, machine learning researchers have looked beyond this paradigm and became interested in tasks that are more dynamic. To them, robotics serve as an excellent test-bed, for the challenges of robotics break many of the assumptions that made supervised learning successful. Out of the many different areas within robotics, robotic manipulation has become a favorite area for researchers to demonstrate new algorithms because of the vast numbers of possible applications and its highly dynamical and complex nature. This document surveys recent robotics conferences and identifies the major trends with which machine learning techniques have been applied to the challenges of robotic manipulation.

1 INTRODUCTION
Robotic manipulation has long fascinated the research community [40], not only because it is an integrated science that requires inter-disciplinary knowledge, but also because a slew of primitive manipulation skills, such as throwing, using tools, and cooking, is so effortless for human to perform, yet so challenging to reproduce in machines.

With the successes of deep learning in computer vision, machine learning practitioners have looked to robotics as the next frontiers of challenges. At the same time, roboticists have adopted machine learning machinery into their traditional workflows. Due to the need to perform sustained mechanical work on the world, robotics problems break the independent and identical distribution assumption of the supervised learning paradigm. These problems have motivated new approaches, both from machine learning and classical model-based perspectives. Looking at published material from major robotics conferences in the last year (CORL 2019-2020, ICRA 2020, RSS 2020, IROS 2020), this paper summarizes the recent trends in which machine learning has been applied to robotic manipulation.

We begin by reviewing the basic mathematical formalism and tools in section 2. The subsequent sections identify distinct threads in which machine learning has been brought to bear on the challenges of robotic manipulations, as summarized below:

- section 3: Different forms of structure, such as graphs or 3D prior, can made learning more efficient and allow the integration of machine learning and traditional robotics methods.
- section 4: Learning can improve model-based methods when the ground truth physical models are hard to obtain or change during robot execution.
- section 5 and section 6: New methods to collect datasets for robotics tasks and new benchmarks.
- section 7: Learning methods allow the use of high-dimensional, otherwise hard to write algorithms for, tactile signal.

Due to the inter-disciplinary nature of robotics research, there is not a commonly used standard set of notations. We thus refrain from describing the algorithms presented in recent papers with notations. Please note that while we only discuss each paper in one section, the successful applications of machine learning methods to robotics problems often require successful execution in many orthogonal dimensions, i.e. both data collection and algorithm design. The inclusion of one paper in a particular section does not mean it does not have new insights to offer in other aspects.

2 BACKGROUND
In this section, we review the mathematical formalism and technical tools often found in robotics papers that use machine learning. We review both the machine learning tools and the more classical model-based methods. We divide the background material into the following sub-sections:

- subsection 2.1: Reinforcement Learning.
- subsection 2.2: Kinematics.
- subsection 2.3: Dynamics.
- subsection 2.4: Control.
2.1 Reinforcement Learning

Reinforcement Learning is an abstraction of the real world that allows for optimization of long-term behavior. As a formalism, its main benefit comes from its generality, in which the robot designers specify the tasks the robots need to perform as the maximization of an expected sum of rewards, without making additional assumptions or having access to domain knowledge, such as the physical environments that the robots might be operating in. On the flip side, such tabula rasa approach means that the robots might require many interactions with their environment to accomplish the tasks. The application of Reinforcement Learning to robotics problems therefore often require substantial background engineering to maximize the amount of interactions the robots have with the environment.

Typically, we represent an environment that the robot operates in and the task we want the robot to perform as a Markov Decision Process $M = (S, A, T, R, H)$, with state space $S$, action space $A$, transition function $T$, initial state distribution $T_0$, reward function $R$, and horizon $H$. At each discrete timestep $t$, the robot is in a state $s_t$, picks an action $a_t$, arrives at $s'_{t} \sim T(\cdot|s_t, a_t)$, and receives a reward $R(s_t, a_t, s'_{t})$. A policy is a function which maps a state to a distribution over the action space, from which we sample an action for the robot to take, $a_t \sim \pi(\cdot|s_t)$. The definitions of the state space, action space, transition function, initial state distribution and reward function are problem-specific. However, the transition function is Markovian, wherein its output only depends on the current state $s_t$ and action $a_t$ and does not depend on the previous states and actions before the current time $t$. The main difference between Reinforcement Learning and classical robotic methods are the assumption of the lack of knowledge about both the transition function $T$ and the reward function $R$, where no knowledge of these functions are assumed and exploited in the design of algorithms.

Given a policy $\pi$ and a Markov Decision Process $M$, we can first sample an initial state from the initial state distribution $s_0 \sim T_0$, use the policy to pick the first action $a_0$, which incurs a reward $r_0 = R(s_0, a_0, s_1)$ and transitions the Markov Decision Process to the next state $s_1 = T(s_0, a_0)$. If we repeat the procedure for $H$ times, we obtain a trajectory:

$$\tau_M = (s_0, a_0, s_1, a_1, \ldots, s_{H-1}, a_{H-1}, s_H)$$

The performance measure of policy $\pi$ is the expected sum of rewards:

$$J_M(\pi) = \mathbb{E}_{\tau_M \sim \pi} \left[ \sum_{t=0}^{H-1} R(s_t, a_t, s'_{t}) \right]$$

where the expectation is over the stochasticity present in the transition function, the reward function and the policy.

In Deep Reinforcement Learning, we parameterize the policy $\pi$ with a deep neural network. Many Deep Reinforcement Learning algorithms have been proposed and are now routinely used in robotic papers published at major robotics conferences. In addition to the policy, another major object often found in Reinforcement Learning algorithms is the state-value function $Q_\pi(s', a')$ which approximates the value of taking action $a'$ at state $s'$ and thereby picking action by sampling from policy $\pi$ until the end of the trajectory.

That is:

$$Q_\pi(s', a') \approx \mathbb{E}_{\tau_M \sim 1} \left[ \sum_{t'=t}^{H-1} R(s_t, a_t, s'_{t}) \right].$$

Together with collaborators, I have contributed five Deep Reinforcement Learning algorithms as publications at ICLR 2019 [66] (proposing a new algorithm based on convex optimization theory), neuRIPS 2019 spotlight [10] (improving sample efficiency with optimistic exploration), ICML 2020 [68] (understanding state-of-the-art method), neuRIPS 2020 [30] (extending Reinforcement Learning formalism to the multi-task offline setting) and neuRIPS 2020 spotlight [79] (Safe Reinforcement Learning algorithm).

Reinforcement Learning algorithms are usually presented absent of any assumptions about the environment in which the algorithms are deployed. In practice, when Reinforcement Learning algorithms are applied to robotics problem, some amount of domain expertise are assumed, such as an understanding of the mechanical properties of the robots or a low-level controller to convert desired torque to electrical signal at the motor-level.

2.2 Robot Kinematics

Commonly used manipulators in research setting consist of a set of rigid links connected together as an acyclic graph by a set of joints. We attach motors to the joints so that we can generate motion of the links to perform tasks. The motion that each joint affords typically corresponds to subset of the special Euclidean group $SE(3)$, such as translation for prismatic joint or rotation for revolute joint. The kinematics of a manipulator describes the relationship between the motion of the joints of the manipulator and the resulting motion of the rigid links.

Figure 1 illustrates the schematic of a simple manipulator with 4 joints and 5 links. Associated with each joint is its current joint angle $q_i$. The vector of all joint angles $\theta = [\theta_1, \theta_2, \theta_3, \theta_4]$ represent the generalized coordinates of the manipulator. These are called “generalized” because even though they do not specify the positions of all points on the manipulator, knowing their values allow us to compute the position of any point because of the rigid body assumptions of the links. There is a stationary coordinate frame $S$, commonly referred to as the base frame. At the end of the manipulator is another frame $T$, commonly referred to as the tool frame. As the joint value changes, the configuration of frame $T$ with respect to frame $S$ changes. The forward kinematics problem determines the configuration of frame $T$ with respect to frame $S$ given the values of the joint angles. The forward kinematics problem has an elegant closed form solution called the product of exponentials formula.

Let $g_{st}(\theta)$ represents the configuration of frame $T$ with respect to frame $S$ for a particular value of the joint angles $\theta$. For each joint as indexed by $i$, construct a unit twist $\xi_i$, corresponding to the axis of the screw motion of the $i$-th joint with all remaining joint angles held fixed at 0. The product of exponentials formula for the forward kinematics problem for a manipulator with $n$ joint is as followed:

$$g_{st}(\theta) = e^{\xi_1 \theta_1} e^{\xi_2 \theta_2} \ldots e^{\xi_n \theta_n} g_{st}(0)$$

The inverse problem to the forward kinematics problem is the inverse kinematics problem: given a desired configuration of the frame $T$ with respect to the frame $S$, we wish to find the joint angles which achieve that configuration. The inverse kinematics problem
is substantially harder than the forward kinematics problem and can have zero, one or many solutions depending on the number of degrees of freedom of the manipulator. There are two general solution families to the inverse kinematics problem. For simple problems, we can decompose the problem into easier sub-problems by hand, such as solving for a rotation such that a point is rotated to another point given a fixed rotation axis. There are also mature software libraries to solve the inverse kinematics problem, such as IKFast from OpenRave.

Taking the time derivative of the configuration of $T$ with respect to $S$, we obtained $\dot{f}_{st}^s \in \mathbb{R}^{6 \times \text{dim}}$, referred to as the spatial manipulator Jacobian. The spatial manipulator Jacobian is of central importance in classical model-based robotics methods. It relates the velocity of the joint angles $\dot{\theta}$ and the velocity of the frame $T$ with respect to frame $S$, as seen from the $S$ frame, $\dot{V}_{st}^s$.

$$\dot{V}_{st}^s = f_{st}^s(\dot{\theta})\dot{\theta}$$

The transpose of the spatial manipulator Jacobian also allows us to relate a spatial wrench $F_s$ applied at the frame $T$ and the corresponding joint torques:

$$\tau = (\dot{f}_{st}^s)^T F_s$$

There are also body-frame equivalent of the base-frame spatial manipulator Jacobian, which relates the joint velocity to end-effector velocity and joint torque to end-effector wrench as seen from the body frame.

### 2.3 Robot Equation of Motion

While kinematics allows us to relate different quantities at a particular point in time, robotics is sequential in nature. The study of dynamics and more specifically the robot equation of motion allow us to relate different properties of the robots as they evolve in time.

The robot equation of motion, which relates the joint positions (angles) $\theta$, velocities $\dot{\theta}$, accelerations $\ddot{\theta}$ and the torques at the joints can be derived from Lagrange’s equations:

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \dot{Y}_i \quad i = 1, \ldots, m$$

where $q_i$ can be thought of as the angle of joint $i$, $L(q, \dot{q}) = T(q, \dot{q}) - V(q)$ is the Lagrangian (kinetic minus potential energy of the system), $m$ is the number of joints and $Y_i$ is the external force acting on joint $i$. For an open-chain manipulator, we can expand Lagrange’s equation to take into account the mass distribution of the rigid links and arrive at the following manipulator equation:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = \tau$$

where $M$ is the mass matrix, $C$ is the Coriolis matrix, $N$ models gravity terms and other external forces which act at the joints and $\tau$ indicates the torque exerted by the actuator at the joints. $M$ is always symmetric and positive definite. $M - 2C$ is a skew-symmetric matrix. $M, C, N$ are often obtained by careful experimental modeling of the robots.

### 2.4 Control

Given an understanding of how the robot evolves in time, we are now interested in commanding torques at the joints such that the robot performs useful mechanical work.

To control the manipulator, we can specify a desired sequence of joint angles $\theta_d(t)$ indexed by time that the robot’s joints should closely follow. A popular choice of control input is torque input because it allows for the use of a dynamical model of the robot in the design of the control law [36]. Let:

$$h(\theta, \dot{\theta}) = C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta})$$

Let the controller’s model of the dynamics be:

$$\tau = \dot{M}(\theta)\dot{\theta} + \dot{h}(\theta, \dot{\theta})$$

where the model is perfect if $\dot{M}(\theta) = M(\theta)$ and $\dot{h}(\theta, \dot{\theta}) = h(\theta, \dot{\theta})$.

Given a twice-differentiable desired sequence of joint angles $\theta_d(t)$, we can write the feed-forward torque controller as follows:

$$\tau(t) = \dot{M}(\theta_d(t))\ddot{\theta}(t) + \dot{h}(\theta_d(t), \dot{\theta}(t))$$

If the controller’s model of the dynamics is perfect and there is no initial error in the joint angles, then we expect the feed-forward torque controller to command the joint torques such that the joint angles follow the desired sequence exactly.

However, the feed-forward controller will not perform well if the model of the dynamics is inaccurate or if there are initial errors in the joint angles. Define the joint error as:

$$\theta_e(t) = \theta_d(t) - \theta(t)$$

We can use the error as a signal to change the commanded torque such that the robot tracks the desired sequence of joint angles better over time, leading to the following popular computed torque controller:

$$\tau = \dot{M}(\theta)\ddot{\theta}_d + K_p\theta_e + K_i \int \theta_e(t) dt + K_d \dot{\theta}_d + \dot{h}(\theta, \dot{\theta})$$

where $K_p$, $K_i$, and $K_d$ are positive-definite matrices and can be chosen to achieve good transient error response. This controller is also known as the feed-forward plus feedback linearizing controller or the inverse dynamics controller. This controller uses the planned acceleration $\ddot{\theta}_d$ to generate non-zero torque command even when the error in joint angles is zero. It is called feedback linearizing because the error signal from $\theta$ and $\dot{\theta}$ are used to generate a linear error dynamics that ensures exponential decay of the joint error. The term $\dot{h}(\theta, \dot{\theta})$ cancels the dynamics that depends non-linearly on the state of the system $\theta, \dot{\theta}$. The mass matrix $\dot{M}(\theta)$ converts the desired joint accelerations into commanded joint torques. The commanded joint torques are sent to low-level controller at the joint to convert
we want the robots to accomplish. We have used additional structure and insights to make learning more successful in robotic manipulation. We organize the material as follows:

- subsection 3.1: Graph inductive bias.
- subsection 3.2: Hierarchical structure.
- subsection 3.3: 3D prior.
- subsection 3.4: Different forms of action parameterization.
- subsection 3.5: Integration of language.

In this section, we discuss the plethora of ways in which researchers have used additional structure and insights to make learning more successful in robotic manipulation. We organize the material as followed:

- subsection 3.1: Graph inductive bias.
- subsection 3.2: Hierarchical structure.
- subsection 3.3: 3D prior.
- subsection 3.4: Different forms of action parameterization.
- subsection 3.5: Integration of language.

Figure 3: Conjugate Task Graph system from [25]. The Demo Interpreter generates the Conjugate Task Graph (CTG) from the demonstration video. The Graph Completion Network completes the CTG by filling in the missing edges. The model executes the task graph by associating the current visual observation with a node in the graph with the Node Localizer and choose action to take with the Edge Classifier.

Figure 4: An example of the hierarchical graph video representation from [58]. Given the changes in the relative spatial arrangement of visual entities in the human demonstration from Step 1 to 2 (first two images), the robot detects matching visual entities in its workspace and moves to achieve similar changes in the entities’ relative spatial arrangement (last two images).

3 ADDING EXPLICIT STRUCTURE TO AID LEARNING

In this section, we discuss the plethora of ways in which researchers have used additional structure and insights to make learning more successful in robotic manipulation. We organize the material as followed:

- subsection 3.1: Graph inductive bias.
- subsection 3.2: Hierarchical structure.
- subsection 3.3: 3D prior.
- subsection 3.4: Different forms of action parameterization.
- subsection 3.5: Integration of language.

Figure 3: Conjugate Task Graph system from [25]. The Demo Interpreter generates the Conjugate Task Graph (CTG) from the demonstration video. The Graph Completion Network completes the CTG by filling in the missing edges. The model executes the task graph by associating the current visual observation with a node in the graph with the Node Localizer and choose action to take with the Edge Classifier.

To generate the desired joint sequence $\theta_d(t)$ to achieve a specific task, such as opening a door, we can use a high-level motion planner. Given a model of the robot’s environment and the desired configuration of the environment, the planner computes a sequence of joint angles such that by following the sequence, the robot accomplishes the desired change in environment’s configuration. For robotic manipulation tasks, it is often more convenient to specify the motion of the robot to achieve a particular task as a sequence of configurations of the end-effectors. There are two potential choices to convert the desired sequence of configurations of the end-effectors into joint torques: either use inverse kinematics to convert the desired sequence of end-effector configurations into desired sequence of joint angles, or implement the equivalent of the controller laws above natively in end-effector space, rather than joint space.

The controllers above implement motion control, where we specify the desired motion of the robot manipulator in joint space. In addition to specifying the motion of the robot, we can also specify the forces with which the robot end-effectors should exert on its environment. There are also hybrid force-motion controllers that allow the robot to exert motion in some axis while exerting force in others. Another popular class of controller is impedance controllers, which can be thought of as ensuring that the robot responses as a virtual spring when it comes into contact with the environment, thus providing a degree of flexibility and safety. Typically, the choice of a good controller depends on the specific tasks we want the robots to accomplish.

3.1 Graph Inductive Bias

Graphical data structures allow algorithm designers to specify relationships and hierarchy between abstract concepts in intuitive ways. It is thus of no surprise that they have found applications in object manipulation tasks, where the algorithms have to reason about the spatial and temporal relationships of objects. [25, 31, 58] use the hierarchical and relational structure of graphs to improve the generalization capability of neural networks.

Task graph is a common representation of high-level execution plans in robotics, where the states are nodes in the graph and the edges represent actions that transitions the environment from one state to another. [25] consider the problem of visual imitation, where the states are images. They thus assume their action space is a discrete, finite set and represent a task by its conjugate task graph (CTG), where the actions are the nodes and the states are the edges. Their assumption of a small and discrete action space allows them to keep the size of the CTG relatively small, which would not have been easily doable if the nodes in the task graph represent visual states instead. Their algorithm consists of two phases: CTG generation from a demonstration video and CTG execution (Figure 3).

To generate the CTG, the model first predicts the action sequence executed in the video, which forms the node in the CTG. The nodes are connected based on their order of execution in the predicted action sequence. The model then completes the CTG by predicting missing possible transitions between nodes in the CTG. Given the generated CTG, the model executes the corresponding task by localizing which nodes in the CTG correspond to the current visual observation. Given the localized node, it then predicts which node (action) connected to the localized node in the CTG should be executed next. In terms of dataset, the method proposed in [25] requires videos of the same task being accomplished with different action sequences. The action sequences should also be temporally aligned with the frames in the videos. These two sources of supervision provides the necessary signal for the CTG completion step, which predicts possible transitions between actions not seen in a demonstration video.

Instead of representing the policy as factorized modules that are trained separately as in [25], in visual Reinforcement Learning, the policy is commonly parameterized by convolutional networks followed by fully connected layers and trained end-to-end. While such parameterization suffices to achieve high single-task performance, [31] claims that it impedes generalization in multi-object manipulation tasks. Instead, they represent their policies by attention-based
Table 1: An overview of how recent works design hierarchy or modularized components.

| Papers | High-level component | Low-level component |
|--------|----------------------|---------------------|
| [59]   | Sampling-based planner | Learned sampler to sample sub-goal (desired object pose) for the sampling-based planner to plan with |
| [14]   | Model Predictive Controller (MPC) planner planning in the space of sub-goal | MPC planner planning low-level motion to achieve sub-goal |
| [26]   | Learned representation adapted from sim2real | Learned controller input: the learned representation output: desired robot velocity |
| [9]    | Sequences of parameterized skills, such as go-to pose | Parameters of the skills |
| [29, 38, 63], [32, 57, 60, 70] | Reinforcement Learning controller output: parameters of low-level controllers or their composition or sequence | Manually-defined or learned controllers. For example, trajectory tracking controller or Position, force or torque controllers or Imitation Learning controllers |
| [53]   | Riemannian motion policies flow controller | Imitation Learning controller |
| [46]   | Manually specified task-parameters. For example: where to pour | Learned parameters indicating how the tasks should be done. For example: how long to pour for |

The graph neural network (GNN), which has been shown to generalize better than convolutional networks for structured data [81], [31] trains the GNN-parameterized policies to perform block-stacking tasks through an increasingly more challenging curriculum where the numbers of block increase over time. The trained policy can generalize without further training to block-stacking tasks where the number of blocks that needed to be stacked are not seen during training. The use of GNN also allows some degree of interpretation into what the policy has learnt. Each node in the GNN represents a block in the environment. By examining the attention weights between the nodes, the author claims that the policy pays more attention to the sub-set of blocks relevant to solving a sub-problem, such as stacking one particular block onto another. They thus hypothesize that the policy has learnt to decompose the complex multiple-block stacking tasks into simpler sub-problem of stacking one block at a time.

Compared to [25], [58] proposes a different method to solve visual imitation. They propose hierarchical graph video representation. In this graph, the nodes represent visual entities detected in the scenes, such as keypoints on objects, and the edges represent their relative 3D spatial arrangement. Given a demonstration video, they first construct the corresponding hierarchical graph representation. They then compare the relative arrangement of the visual entities in the scene between the demonstration video and the imitator’s workspace to construct a reward function. They use this reward function for trajectory optimization to generate actions to be executed on the imitating robot (Figure 4). To detect keypoints on objects, match keypoints across the demonstration-imitator view and across time in a self-supervised manner, they perform multi-view self-supervised point feature learning, which is explained in more details in subsection 3.3. [58] demonstrates better generalization compared to method which maps the input images directly into a high-dimensional embedding space and use distance in the embedding space as the reward function. However, it is unclear how to extend their methods to more complex scenes where objects can not be represented as keypoints, such as in water pouring tasks.

A common limitation among methods which integrate graphs into learning is the data requirement. The dataset must provide relational signal to allow the learning process to capture the hierarchical and relational structure in the problem. For example, the method in [25] performs worse than baselines when there are less than 100 training tasks. For each task, the method also requires demonstrations that accomplish the tasks with different action sequences, which is crucial to learn the action-to-action transition and train the Conjugate Task Graph completion network. Similarly, the use of attention-based graph neural network to parameterize the policy in [31] only demonstrates benefit over simple multi-layer perceptron parameterization when the policies are trained with the curriculum, which might not be realizable in practice on real robots. These two methods are also not demonstrated on real robots.

3.2 Hierarchical structure

Previous applications of machine learning to robotic manipulation [21, 27, 47] take a black-box approach where the controller is a deep neural network, trained in an end-to-end manner, with no explicit hierarchical structure. A major trend in recent robotic manipulation research is the explicit integration of hierarchy into algorithm design, which allows for the integration of machine learning and classical model-based techniques. Table 1 provides an overview of how recent works design the hierarchical structures. In this section, I discuss three representative works, where the high and
They choose the goal state such that getting from the current state to the goal state does not require executing many actions. The method tackles this challenge through hierarchy: a high-level goal selection module and a low-level goal-conditioned imitation controller. Figure 5 provides a high-level overview of the different components in the method. The high-level goal selection module selects goal states for the low-level controller to move towards. The high-level goal selection module consists of 2 components: a conditional Variational Auto-Encoder (cVAE) which proposes goal states and a value function that models the expected return of the goal states. Given the current state, the cVAE proposes multiple future goal states. The value function evaluates them and sets the state with the highest expected return as the goal state for the low-level controller. The paper trains the cVAE using the evidence lower bound to capture the distributions of states within the demonstration, thereby modeling the multi-modal aspects of the demonstrations. They also train the value function using [16] to tackle the value function divergence issues often seen when Reinforcement Learning algorithms are trained from purely existing data without further interactions with the environment. They train the low-level controller using standard supervised Behavior Cloning loss to reach a goal state given the current state. They choose the goal state such that getting from the current state to the goal state does not require executing many actions. The paper argues that the multi-modal aspects of the demonstration only manifests over temporally extended sequences of state-action and is not present in shorter state-action sequences. The low-level controller thus does not need to tackle the multi-modality issue.

Instead of learning both the high and low level components, [32] only learns the high-level controller while the low-level controllers are designed using model-based methods. [32] tackles in-hand manipulation of objects. In-hand manipulation is the task of changing the pose of an object while ensuring the object and the manipulator maintain continuous contact. The manipulator considered consist of three fingers on a planar surface, assuming hard contact model and precision grasp. The object is an planar bar. Each contact configuration of the fingers with the object allow the fingers to move the object to a limited range of poses, but not all possible poses of the object. Thus, to move an object from an initial pose to a desired pose, the fingers might have to change the contact configuration using either the Sliding or Flipping motion primitives.

Instead of learning both the high and low level components, [32] only learns the high-level controller while the low-level controllers are designed using model-based methods. [32] tackles in-hand manipulation of objects. In-hand manipulation is the task of changing the pose of an object while ensuring the object and the manipulator maintain continuous contact. The manipulator considered consist of three fingers on a planar surface, assuming hard contact model and precision grasp. The object is an planar bar. Each contact configuration of the fingers with the object allow the fingers to move the object to a limited range of poses, but not all possible poses of the object. Thus, to move an object from an initial pose to a desired pose, the fingers might have to change the contact configuration using either the Sliding or Flipping motion primitives.

Instead of learning both the high and low level components, [32] only learns the high-level controller while the low-level controllers are designed using model-based methods. [32] tackles in-hand manipulation of objects. In-hand manipulation is the task of changing the pose of an object while ensuring the object and the manipulator maintain continuous contact. The manipulator considered consist of three fingers on a planar surface, assuming hard contact model and precision grasp. The object is an planar bar. Each contact configuration of the fingers with the object allow the fingers to move the object to a limited range of poses, but not all possible poses of the object. Thus, to move an object from an initial pose to a desired pose, the fingers might have to change the contact configuration using either the Sliding or Flipping motion primitives.

Instead of learning both the high and low level components, [32] only learns the high-level controller while the low-level controllers are designed using model-based methods. [32] tackles in-hand manipulation of objects. In-hand manipulation is the task of changing the pose of an object while ensuring the object and the manipulator maintain continuous contact. The manipulator considered consist of three fingers on a planar surface, assuming hard contact model and precision grasp. The object is an planar bar. Each contact configuration of the fingers with the object allow the fingers to move the object to a limited range of poses, but not all possible poses of the object. Thus, to move an object from an initial pose to a desired pose, the fingers might have to change the contact configuration using either the Sliding or Flipping motion primitives.

Instead of learning both the high and low level components, [32] only learns the high-level controller while the low-level controllers are designed using model-based methods. [32] tackles in-hand manipulation of objects. In-hand manipulation is the task of changing the pose of an object while ensuring the object and the manipulator maintain continuous contact. The manipulator considered consist of three fingers on a planar surface, assuming hard contact model and precision grasp. The object is an planar bar. Each contact configuration of the fingers with the object allow the fingers to move the object to a limited range of poses, but not all possible poses of the object. Thus, to move an object from an initial pose to a desired pose, the fingers might have to change the contact configuration using either the Sliding or Flipping motion primitives.
Different training strategies to improve the learned visual descriptors can be applied. [15] proposes a framework to learn to detect object keypoints without human annotation based on geometric prior and demonstrates applications of such keypoint-based object representation for manipulation tasks (Figure 8). To obtain a visual descriptor of an object keypoint, they first find pixel-wise correspondence between multiple views of the object. Given a static object, they move a RGBD camera mounted on a robot wrist to capture views of the object from different angles and perform dense 3D reconstruction. Two pixels from two images corresponding to two different views are a match if they correspond to the same vertex in the 3D reconstruction. Given the pixel-wise correspondence, they train a network to output per-pixel descriptor to represent the scene. Depending on the data sampling scheme during training, they demonstrate that the keypoints and their descriptors can generalize across different instances of the same object class, across different classes of objects, and are also invariant to viewpoint and object configuration and deformation. Using the learned descriptor, they demonstrate the ability to grasp an object at pre-defined locations while only requiring weak human annotation. To do so, a human can annotate a picture of an object by clicking on the object to indicate where the robot should grasp. When the robot observes the object from a different view, the learned network finds a pixel location in the current view that has similar descriptor value as the pixel annotated by the human. The robot proceeds to grasp the object at the found pixel location in the current view, using geometric method to generate grasp proposals. They also demonstrate the benefit of the keypoint representation in a constrained manipulation setting, where the manipulation does not have 6 DoF in free space [39], [65] further demonstrates how to obtain such semantic keypoints with only access to RGB views and without requiring accurate depth sensing. Using neural network to learn pixel-wise descriptor has also seen successes when deployed to a mobile manipulation system [2], which allows human operator to teach robot primitive skills via a Virtual Reality interface. The robot chooses which skills to perform based on how similar the learned visual description of the current scene is to the stored visual descriptors associated with the taught skills.

A common approach to apply learning to robotics problems is to train a dynamics model and then use it for planning. An advantage of this approach is that the networks can take raw visual observation as input. We can also train the networks with robot interaction data, which can be easier to acquire than human annotation. Previous approaches have mostly modeled only the visible surfaces of objects, such as mapping RGB image to RGB image by predicting optical flow. [73] argues that such approaches perform poorly in cluttered environments, where objects frequently go out of view. They thus argue for explicit 3D structure in the representation of the learnt dynamics. They aim to achieve object permanence, amodal completeness and consistent object tracking.
Their method thus only requires 10 view. Figure 9 provides a simple example to illustrate the benefit of Randomization. Allowing the Reinforcement Learning agents to take actions in the low-dimensional action space increases the probability the agents will perform actions that have positive rewards, thereby improving learning efficiency compared to agents operating in torque space. [55] also tackles deformable object manipulation. However, they train their policy to imitate a state-based expert controller in simulation. They also design a low-dimensional pick-and-pull action space, which makes it easier for the policy to imitate the expert controller. The main limitations of these works are that the tasks considered are fairly simple, such as flattening or straightening objects (Figure 10). That being said, these tasks are still challenging due to the lack of a canonical state representation and the complex dynamics of highly deformable objects like cloth.

When turning the continuous action space into a low-dimensional discrete one, it is also possible to simplify the problem further by assuming the decision making problem is one-step, rather than multi-step. For example, [19] tackles the challenge of picking up unseen food items using a fork in an assistive feeding scenario. They design simple food picking action primitives consisting of three different pitch and two different roll angles of the fork when it approaches the food items. As such, they can apply contextual bandit algorithms with well-known regret bound to ensure convergence to a successful food picking strategy. Figure 11 illustrates the three different fork pitch angles. A similar simplification can be found in [12], which tackles the task of learning to grasp challenging objects. They assume an object has a small number of stable poses and that their perception system can differentiate among these poses. For each stable pose of the object, they sample a small number of potentially successful grasp proposal from DexNet [37] and let these proposals form the discrete action space associated with the stable pose. They thus maintain as many instantiations of bandit algorithms as there are stable poses and attempts to continuously grasp an object until a good grasp can be found. These methods benefit from the one-step nature of the simplified decision making problem due to having a much smaller action space to explore.

Another method to improve the exploration capability of a Reinforcement Learning agent is to augment the torque-based action space with a motion planner [74]. They train a Reinforcement Learning agent to predict the changes in joint values. If the predicted changes are small, they use low-level torque controller to achieve the desired changes. This allows the robot to perform manipulation actions requiring precision, such as those involving contacts. If the

---

**Figure 10:** Policy execution for deformable object manipulation from [55]. (top row) The policy learns to flatten the cloth in simulation by imitating an expert (bottom rows) Execution on a physical da Vinci surgical robot. The action space consists of a pick location and a pull direction in the 2D plane.

**Figure 11:** Three different fork pitch angles from [19]. When faced with an unknown food item, the contextual bandit algorithm explores the three different fork pitch angles (TA: tilted-angle, VS: vertical, TV: tilted-vertical) and two different roll angles (not shown) to find successful strategies to pick up a food item with the fork.
Machine Learning for Robotic Manipulation

Figure 12: Examples of human-annotated task-oriented grasps from [41]. For each object, the authors provide RGB images, point cloud and stable grasps which were chosen by human annotators to satisfy a particular task, such as pick up or handover.

Figure 13: Failed grasp proposal from [41]. Since their method does not take into account post-grasp object-gripper interaction, the configuration of the grasped object would not allow the manipulator to perform the task in this case. The task is “saute”, which would require the pot bottom to be roughly parallel to the ground. However, the top-down grasp instead flips the pot such that the pot’s bottom is perpendicular to the ground instead.

Changes are large, they use a motion planner to find joint trajectory for the robot to arrive at the final desired joint position.

A less active, though interesting research direction, is to design the action space from a control perspective. [5] shows that instead of letting the neural network predicts desired torque directly, having it outputs the desired joint positions and convert the desired joint positions into torques using a variable gain PD controller leads to better performance in tasks involving hybrid motion-force control.

3.5 Integration of language

The advances of deep learning in natural language understanding have also motivated the integration of these techniques into robotics. This has enabled the use of language-based semantics to improve the generalization of grasping systems [41] and specifies tasks to the robots in a more intuitive manner [45, 56].

[41] proposes a new dataset and algorithm to generate task-oriented grasp. The dataset contains RGB images and pointclouds of 191 different objects from 75 different object categories. There are also 56 different task labels, such as pick up and handover. For each object, human annotators label which tasks a stable grasp for the object will allow the manipulator to perform out of the 56 different task labels. Figure 12 provides examples of the objects in the dataset and their corresponding task-oriented grasp labels. To predict the task-oriented grasp for an object, they encode the object categories and the tasks they afford in a knowledge graph. The feature for each node in the graph is the word embedding of the corresponding word at the node. They then perform message passing to predict the task-oriented grasp and train the network end-to-end. They demonstrate that the incorporation of the semantic knowledge of the object category and the tasks in terms of their word embedding and expressing relationships between the words in a knowledge graph lead to better generalization to unseen objects. A limitation of their approach is that they do not consider the post-grasp object-gripper interaction, which can lead to false positive in the generated task-oriented grasp. Figure 13 provides an example for such a false positive.

We can also use language to specify tasks that the robot should perform [56]. Given an image of the current scene and a language instruction, the controller should generate a robot trajectory that accomplishes the task specified in the instruction. The colored arrow indicates the different language instructions and the corresponding motion trajectories.

Figure 14: Examples of language instructions from [56]. Given an image of the current scene and a language instruction, the controller should generate a robot trajectory that accomplishes the task specified in the instruction. The colored arrow indicates the different language instructions and the corresponding motion trajectories.

Figure 14: Examples of language instructions from [56]. Given an image of the current scene and a language instruction, the controller should generate a robot trajectory that accomplishes the task specified in the instruction. The colored arrow indicates the different language instructions and the corresponding motion trajectories.
task with. Their method can generalize to unseen object classes and unknown nouns in the language commands.

4 USING LEARNING TO IMPROVE MODEL-BASED METHODS

While the methods presented in the previous sections predominantly use additional structure to improve the learning efficiency, researchers have also explored using learning techniques to improve the performance of more classical model-based methods.

[1] modifies the parameters of a physical simulator such that the simulation better matches a target physical model. There are two main insights offered in their paper. First, estimating the residual between the current simulation parameters and the target simulation parameters is more effective than predicting the target simulation parameters directly. They argue this is because small differences are more frequently observed in the training dataset. Second, they generate their training data using only simulation. However, their neural network trained with such synthetic data can generalize to real-world scenarios. The training and inference procedures are illustrated in Figure 15. They demonstrated their method in estimating the coefficient of restitution of a bouncing ball. The observation in this case is the position of the ball. The main limitation of this work is the need to carefully measure the observation from the target model, which might be challenging to obtain for more complex object interactions. They also require the arrangement of the objects in simulation to roughly match those in the target model, to impose constraints on the estimation problem and allow estimating the residuals with a small number of data coming from the target model.

In addition to improving the model, we can also use learning to improve the controller. To improve Feedback Linearization control, [7] performs online learning to generate torque compensation to account for unmodeled dynamics. As discussed in subsection 2.4,

given a perfect model of the robot, we can cancel out the nonlinear dynamics in the computed torque control law, leading to a stable linear error dynamics. The model of the robot is often computed offline before robot deployment. However, during execution, the model might change, for example, when the robot picks up a box. In such case, the large change in the structural property of the robot inhibits the ability of the computed torque control law to accurately track the desired robot motion. [7] interprets the difference in the desired and achieved motion as if Feedback Linearization was accurate, but the robot was driven by an unknown acceleration reference. They thus estimate the unknown acceleration reference with the Controllability Gramian, which allows for acceleration estimation without resorting to numerical differentiation. The estimated acceleration forms a dataset with which they train a Gaussian Process Regression model to generate the torque compensation for the unmodeled dynamics. They emphasize that their main difference compared to previous work is that they do not use torque measurement, which is often noisier than joint position and velocity measurement. It would be interesting to compare their approach against a dynamic gain PID controller in terms of the effectiveness in tracking desired motion in the presence of large unmodeled dynamics.

5 DATA COLLECTION METHODS

Learning methods require a high amount of interactions with their environment to exhibit interesting behaviors. In this section, we discuss how successful applications of learning methods have instrumented either the environment [44, 77, 78] or the robots [23, 52, 61, 75] to automate the data collection process and increase the usefulness of the collected data.

We split the section into autonomous data collection (subsection 5.1) and human-assisted data collection (subsection 5.2).

In addition to the works discussed below, [22, 35] considers how to better make use of structure present in existing datasets.

5.1 Autonomous data collection

Environment reset is a critical mechanism that allows robots to autonomously collect data. To enable environment reset, we need

![Figure 15: Training and inference procedure of TuneNet [1].](image)

![Figure 16: Environment reset mechanism from [78].](image)
Figure 17: Environment reset mechanism from [44]. The 24-DoF anthropomorphic hand learns how to rotate balls on its palm without dropping them. Note that the hand faces upwards, therefore making use of gravity to ensure the balls are likely to stay inside the palm. If the hand drops the ball during the course of learning, the white robot picks up the ball and places them inside the palm of the hand. This allows the hand to continuously learn without human intervention.

A recent successful demonstration of learning for manipulation is [78]. They propose a system that allows robots to throw arbitrary objects into bins. This is a challenging task because the ballistic trajectories and the final locations of the thrown objects are a complex function of the throwing forces, gripper-object contact locations and object’s inertial and material properties. The environment reset mechanism in [78] allows their robot to repeatedly try to throw objects into randomly chosen bins. Even if the thrown objects do not end up in the correct bins, it would arrive at the neighboring bins, which allow for effective environment reset. Figure 16 illustrates the environment set-up that allows for reset.

Learning has also been successfully applied to tasks requiring repeatedly making and breaking contacts, which are hard to model and estimate accurately [44]. The paper teaches a robot with 24-DoF anthropomorphic hand to perform complex tasks, such as moving a pencil to write or rotating baoding balls in-hand. Their method consists of a learned predictive model and a trajectory optimization method to generate actions. To enable the hand to continuously learn how to rotate the balls without dropping them, they have another robot arm which picks up the ball and places them in the palm of the multi-finger hand. Figure 17 illustrates this mechanism.

Other notable successes include [77], which learns to perform the tasks of assembly. This involves picking up objects and placing them into a specific location inside a container with tight clearance. They have two main mechanisms to reset the environment. They restrict their methods to planar setting, which allows for the design of simple motion primitives that can pick and place object. More importantly, they generate the training data by disassembly, that is, taking apart already assembled parts. This was helpful because disassembly is much easier compared to assembly for the objects and tasks they consider. Performing disassembly therefore allows them to back-play the trajectories generated from the disassembly process and treat them as successful assembly trajectories.

5.2 Human-assisted data collection

Providing low-friction methods for human to perform demonstration is another interesting line of research. [61, 75] use simple, cheap and easily controlled grippers to collect demonstration for grasping. There are two main benefits. First, they can theoretically collect demonstrations on realistic objects and in more unconstrained scenarios compared to allowing the robots to collect data autonomously. When controlled by human, the chances that the robots will damage itself or its environments are significantly reduced. Since learning methods have not quite yet generalize outside the training distribution, the ability to collect data in more unconstrained setting means that the learned policies might perform better in real-world deployment scenarios. Second, the control interface is intuitive and the grippers are cheap, thereby allowing them to scale data collection. Figure 18 illustrates the gripper design from [61].

Figure 18: Gripper design from [61]. They use simple gripper to collect grasping demonstration in more realistic scenarios compared to prior works.

Figure 19: Hardware set-up from [61]. They use four depth cameras to track human poses and the values of the finger joints. They perform kinematic retargeting to control a 23-DoF Allegro hand.

Clever use of easily controlled mechanism can also generate data for object segmentation [4]. The main limitation of these methods is the need to design task-specific environment reset mechanism. This limits their applicability and increases the amount of background engineering required to teach robot new skills. Allowing the robots to autonomously collect data in simulations and transferring the learned policies to real world scenarios are also widely used [11, 24, 26, 42, 49, 50, 62, 69]. These works highlight the benefits of having accurate approximate physical models, which allows for simulation.
dataset is still collected in a constrained lab setting, which means equal in size to ImageNet, it has two main limitations. First, the dataset consists of more than 160 simple grasping primitives with added Gaussian noise. Even though the controller for the robots to interact with the objects, they use sequences of 7 trajectories with video and action images. Researchers have therefore applied learning to perceive tactile signal, which is also high-dimensional and hard to model accurately. Tactile signal provides direct sensory access to information that would otherwise be non-trivial to obtain through vision sensing alone, such as the magnitude of forces applied on objects or the objects’ material properties. Tactile information can also supplement visual information during occlusions of the object, which frequently occur in robotic manipulation.

On the other end of the end-effector spectrum, [23] proposes a system for human to tele-operate complex multi-finger hands using only vision sensors. Figure 19 illustrates their hardware setup. Because their system controls a hand with many degrees of freedom, they require significantly more complex hardware system and algorithms compared to the simple gripper line of works. They use four depth cameras and a model-based tracker [54] to track human hand pose and the value of the human finger joints. They then perform kinematic retargeting to map the detected human hand poses and joint angles to the poses and angles of the robot hand. Through careful tuning, they demonstrate impressive task completion such as retrieving a tea bag from a drawer.

Recent interesting works on human-in-the-loop learning also include [18, 48].

6 NEW DATASETS AND BENCHMARKS

It would be a travesty to discussing learning without discussing new datasets and benchmarks.

[33] proposes a simulated benchmark for deformable object manipulation (Figure 20). This paper fills an important missing gap in that previous simulated benchmark either provides direct access to low-dimensional state of the environment or only involves manipulating rigid bodies. The paper demonstrates the sub-optimality of current Reinforcement Learning algorithms when the input are images and the agent needs to manipulate objects with high intrinsic dimension such as cloth. They also demonstrate the realism of the simulated dynamics by comparing the result of cloth manipulation in simulation against the result of manipulating a real piece of cloth.

[13] provides a large dataset of real world robot interactions. It consists of more than 160,000 trajectories with video and action sequences of 7 robots interacting with hundreds of objects. To script the controller for the robots to interact with the objects, they use simple grasping primitives with added Gaussian noise. Even though the dataset consists of more than 15 million frames, which is roughly equal in size to ImageNet, it has two main limitations. First, the dataset is still collected in a constrained lab setting, which means any models trained from the dataset are unlikely to generalize immediately to more practical settings. Second, the tasks considered are still quite simple because the scripted policies to generate data only consist of grasping primitives. I believe that methods which allow for easier data collection in unconstrained environments, as discussed in subsection 5.2, will be more impactful in the future.

7 LEARNING FOR TACTILE SENSING

A major advantage of learning systems is their ability to perceive and learn useful representation for high-dimensional sensory input, as evident by their successes in tasks where the input consists of images. Researchers have therefore applied learning to perceive tactile signal, which is also high-dimensional and hard to model accurately. Tactile signal provides direct sensory access to information that would otherwise be non-trivial to obtain through vision sensing alone, such as the magnitude of forces applied on objects or the objects’ material properties. Tactile information can also supplement visual information during occlusions of the object, which frequently occur in robotic manipulation.

In [67], the robot learns a low-dimensional embedding of the physical features of a held object, such as frictional properties, from dense tactile feedback. They use the vision-based GelSight sensor [76], which provides high resolution information about the contact surface between the object and the finger. They also add markers along the sensing surface to obtain the tangential displacement of objects. Figure 21 provides an illustration of the tactile signal in the form of a vector field. They design two action primitives, tilting and shaking, to obtain tactile signal which can disambiguate the properties of the objects, including friction, center of mass, mass, and moment of inertia. They learn a low-dimensional embedding which fuses the data generated from the two actions. The embedding can be disentangled such that each embedding dimension corresponds to one physically meaningful property. Given the embedding, they learn a forward dynamics model which allows them to optimize for actions that swing a held object to a chosen angle to within 1° error. While swing-up is not a particular practical robotics task, the paper demonstrates the benefits of the tactile sensor and the
ability to interpret the resulting sensory information with learning methods.

In addition to inferring the mechanical properties of object, tactile signal can also be used to infer object’s pose. Figure 22 illustrates the system from [3] to estimate the object poses of an object from a single interaction between the tactile sensor and the object. They match the estimated contact shape against a pre-determined set of contact shapes to determine the most likely object pose that generates the estimated contact shape. To ensure efficiency, they assume access to the object 3D model and develop object-specific model. They argue these are realistic assumptions in industrial setting.

Other recent works which use learning to process high-dimensional tactile data include [17] (using high frequency tactile information to achieve stable in-grasp manipulation for low-cost multi-fingered hands), [80] (predicting contact events such as slipping from tactile information to improve robustness of closed-loop grasping) and [71] (proposing a novel curriculum of action motion magnitude which makes learning more tractable).

8 CONCLUSION

The applications of machine learning to robotic manipulation have enabled new capabilities and opened up interesting research venues. That being said, the successes of machine learning in robotic manipulation tasks have in some ways seemed more limited than the successes in computer vision or natural language understanding. We can venture guesses why this is the case. First, robotic manipulation research is significantly more open-ended and the community does not have a set of mutually agreed upon “north star” problems. For this reason and the need to run experiments on physical robots, researchers often do not compare their algorithms against previous works. The lack of reproducibility and a common set of challenges mean researchers often demonstrate their algorithms in simple tasks, such as block pushing and stacking. Thus, despite an outpouring of research, it is unclear whether the community as a whole is making progress on the challenges of robotic manipulation.

Second, it is challenging to collect realistic large-scale datasets to train robots with. While the internet and low-cost crowd-source platform have enabled the collections of massive dataset that fuels the advances in computer vision and natural language tasks, such mechanism has yet to exist for robotic manipulation. Most datasets used in robotics paper are collected from constrained lab setting. The models trained from these datasets will often not generalize to realistic use cases. Third, robotics is just really hard, especially for manipulation tasks where the robot needs to act in complex ways to manipulate previously unseen objects. Despite commendable efforts by researchers, we still do not have in-depth physical understanding of the many aspects of the problems. For example, the study of friction, a central aspect of object manipulation, is still an active area of research [51]. Even if learning algorithm does not use physical insights directly in their inference step, knowing the relevant physical model-based knowledge will allow researchers to develop better learning methods and debug them faster when they do not work.

REFERENCES

[1] Adam Allevato, Elaine Schaertl Short, Mitch Pryor, and Andrea L. Thomaz. 2020. TuneNet: One-Shot Residual Tuning for System Identification and Sim-to-Real Robot Task Transfer. arXiv:1907.11200 [cs.RO]
[2] M. Bajracharya, J. Borders, D. Helmick, T. Kollar, M. Laskey, J. Lechtert, J. Ma, U. Nagarajan, A. Ochtai, J. Petersen, K. Shankar, K. Stone, and Y. Takaoka. 2020. A Mobile Manipulation System for One-Shot Teaching of Complex Tasks in Homes. In 2020 IEEE International Conference on Robotics and Automation (ICRA). 11039–11045. https://doi.org/10.1109/ICRA40945.2020.9196677
[3] Maria Bassa, Eric Valls, Bryan Lim, Theo Seshoousou, and Alberto Rodriguez. [n.d.] Tactile Localization From The First Touch.
[4] Wout Boerdjik, Martin Sundermeyer, Maximilian Durner, and Rudolph Triebel. 2020. Self-Supervised Object-in-Gripper Segmentation from Robotic Motions. arXiv:2002.04487 [cs.CV]
[5] Miroslav Bogdanovic, Majid Khadiv, and Ludovic Righetti. 2020. Learning Variable Impedance Control for Contact Sensitive Tasks. IEEE Robotics and Automation Letters (Early Access) (July 2020).
[6] Michel Breyer, Jen Jen Chung, Lionel Ott, Siegwart Roland, and Nieto Juan. 2020. Volumetric Grasping Network: Real-time 6 DOF Grasp Detection in Clutter. In Conference on Robot Learning.
[7] M. Capotondi, G. Turisci, C. Gaz, V. Modugno, G. Oriolo, and A. De Luca. 2020. An Online Learning Procedure for Feedback Linearization Control without Torque Measurements. In Proceedings of the Conference on Robot Learning (Proceedings of Machine Learning Research). Leslie Pack Kaelbling, Danica Kragic, and Komei Sugizuka (Eds.), Vol. 10. PMLR, 1359–1368. http://proceedings.mlr.press/v100/capotondi20a.html
[8] Chung-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan D. Ratliff. 2018. RPIMatow: A Computational Graph for Automatic Motion Policy Generation. CoRR abs/1811.07049 (2018). arXiv:1811.07049 http://arxiv.org/abs/1811.07049
[9] Rohan Chittnis, Shubham Tulsiani, Saurabh Gupta, and Abhinav Gupta. 2019. Efficient Bimanual Manipulation Using Learned Task Schemata. CoRR abs/1909.13874 (2019). arXiv:1909.13874 http://arxiv.org/abs/1909.13874
[10] Kamil Ciossek, Quan Vuong, Robert Löfflin, and Katja Hofmann. 2019. Better Exploration with Optimistic Actor-Critic. arXiv:1910.12807 [stat.ML]
[11] Michael Danielczuk, Anelia Angelova, Vincent Vanhoucke, and Ken Goldberg. 2020. X-Ray: Mechanical Search for an Occluded Object by Minimizing Support of Learned Occupancy Distributions. arXiv:2004.09039 [cs.RO]
[12] Michael Danielczuk, Ashwin Balakrishna, Daniel S. Brown, Shivin Devgon, and Ken Goldberg. 2020. Exploratory Grasping: Asymptotically Optimal Algorithms for Grasping Challenging Polyhedral Objects. arXiv:2011.05632 [cs.RO]
[13] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. 2020. RoboNet: Large-Scale Multi-Robot Learning. arXiv:1910.11215 [cs.RO]
[14] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. 2019. Dynamical Learning with Cascaded Variational Inference for Multi-Step Manipulation. CoRR abs/1910.13395 (2019). arXiv:1910.13395 http://arxiv.org/abs/1910.13395
[15] Peter R. Florence, Lucas Manuelli, and Buss Tedrake. 2018. Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation. CoRR abs/1806.08756 (2018). arXiv:1806.08756 http://arxiv.org/abs/1806.08756
[16] Scott Fujimoto, David Meger, and Doina Precup. 2018. Off-Policy Deep Reinforcement Learning without Exploration. CoRR abs/1812.02900 (2018). arXiv:1812.02900 http://arxiv.org/abs/1812.02900
[61] Shuran Song, Andy Zeng, Johnny Lee, and Thomas Funkhouser. 2020. Grasping in the Wild: Learning 6DoF Closed-Loop Grasping from Low-Cost Demonstrations. arXiv:1912.04344 [cs.CV]

[62] S. Stević, S. Christen, and O. Hilliges. 2020. Learning to Assemble: Estimating 6D Poses for Robotic Object-Object Manipulation. IEEE Robotics and Automation Letters 5, 2 (2020), 1159–1166. https://doi.org/10.1109/LRA.2020.2967325

[63] Robin A. M. Strudel, Alexander Parshevich, Igor Kalevatykh, Ivan Laptev, Josef Sivic, and Cordelia Schmid. 2019. Combining learned skills and reinforcement learning for robotic manipulations. arXiv:1908.00722 (2019). arXiv:1908.00722 http://arxiv.org/abs/1908.00722

[64] Hsiao-Yu Fish Tung, Zhou Xian, Mihir Prabhudesai, Shamit Lal, and Katerina Fragkiadaki. 2020. 3D-OES: Viewpoint-Invariant Object-Factorized Environment Simulators. arXiv:2011.06464 [cs.RO]

[65] Mel Vecerik, Jean-Baptiste Regli, Oleg Sushkov, David Barker, Rugile Pevcevičiute, Thomas Rothörl, Christopher Schuster, Raia Hadsell, Lourdes Agapito, and Jonathan Scholz. 2020. S3K: Self-Supervised Semantic Keypoints for Robotic Manipulation via Multi-View Consistency. arXiv:2009.14711 [cs.RO]

[66] Quan Ho Vuong, Yiming Zhang, and Keith W. Ross. 2018. Supervised Policy Update. CoRR abs/1805.11706 (2018). arXiv:1805.11706 http://arxiv.org/abs/1805.11706

[67] Chen Wang, Shaoxiong Wang, Branden Romero, Filipe Veiga, and Edward H. Adelson. [n.d.]. SwingBot: Learning Physical Features from In-Hand Tactile Exploration for Dynamic Swing-up Manipulation.

[68] Che Wang, Yanqiu Wu, Quan Vuong, and Keith W. Ross. 2019. Towards Simplicity in Deep Reinforcement Learning: Streamlined Off-Policy Learning. CoRR abs/1910.02208 (2019). arXiv:1910.02208 http://arxiv.org/abs/1910.02208

[69] Matthew Wilson and Tucker Hermans. 2020. Learning to Manipulate Object Collections Using Grounded State Representations. arXiv:1909.07876 [cs.RO]

[70] Florian Winnhofer, Sebastian Philipp Schmitt, von Georg Wichert, and Wolfram Burgard. 2020. Controlling Contact-Rich Manipulation Under Partial Observability. RSS 2020 (2020).

[71] Bohan Wu, Iretiayo Akinola, Jacob Varley, and Peter Allen. 2019. MAT: Multi-Fingered Adaptive Tactile Grasping via Deep Reinforcement Learning. arXiv:1909.04787 [cs.RO]

[72] Yilin Wu, Wilson Yan, Thanard Kurutch, Lerrel Pinto, and Pieter Abbeel. 2019. Learning to Manipulate Deformable Objects without Demonstrations. CoRR abs/1910.13439 (2019). arXiv:1910.13439 http://arxiv.org/abs/1910.13439

[73] Zhenjia Xu, Zhanpeng He, Jiajun Wu, and Shuran Song. 2020. Learning 3D Dynamic Scene Representations for Robot Manipulation. arXiv:2011.01968 [cs.RO]

[74] Jun Yamada, Youngwoon Lee, Gautam Salhotra, Karl Pertsch, Max Pflieger, Gaurav S. Sukhatme, Joseph J. Lim, and Peter Englert. 2020. Motion Planner Augmented Reinforcement Learning for Robot Manipulation in Obstructed Environments. arXiv:2010.11940 [cs.RO]

[75] Sarah Young, Dhraj Gandhi, Shubham Tulsiani, Abhinav Gupta, Pieter Abbeel, and Lerrel Pinto. 2020. Visual Imitation Made Easy. arXiv:2008.04899 [cs.RO]

[76] Wenzhen Yuan, Siyuan Dong, and Edward H. Adelson. [n.d.]. GelSight: High-Resolution Robot Tactile Sensors for Estimating Geometry and Force.

[77] Kevin Zakka, Andy Zeng, Johnny Lee, and Shuran Song. 2020. Form2Fit: Learning Shape Priors for Generalizable Assembly from Disassembly.

[78] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. 2020. TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. arXiv:1903.11239 [cs.RO]

[79] Yiming Zhang, Quan Vuong, and Keith W. Ross. 2020. First Order Constrained Optimization in Policy Space. arXiv:2002.06506 [cs.LG]

[80] Yazhan Zhang, Weihao Yuan, Zicheng Kan, and Michael Yu Wang. 2019. Towards Learning to Detect and Predict Contact Events on Vision-based Tactile Sensors.

[81] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. CoRR abs/1812.08434 (2018). arXiv:1812.08434 http://arxiv.org/abs/1812.08434

[82] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. CoRR abs/1812.08434 (2018). arXiv:1812.08434 http://arxiv.org/abs/1812.08434